Table of Contents

Welcome to the **NeoFPS** documentation

NeoFPS is a first person shooter asset and toolkit for the Unity game engine. Its goal is to enable you to create an FPS that matches your vision without restrictions. Designed to be flexible and extensible, NeoFPS can be the perfect starting point for your game.

Installation

For instructions on getting set up with NeoFPS, see the section titled NeoFPS Installation.

Learning NeoFPS

More comprehensive documentation with more frequent updates can be found on the NeoFPS documentation website.

For information on using NeoFPS in the Unity editor, see the Manual.

For details on writing code for NeoFPS, see the Scripting Reference.

You can find a selection of helpful tutorials here.

Support

NeoFPS is a complex asset and as such there are bound to be occasional problems. If something seems broken or counter-intuitive, or if you have a suggestion that would make NeoFPS better for developers then please get in touch.

You can access the support page on the website at NeoFPS Support.

Alternatively, you can email support@neofps.com.

Further sources of information

- The NeoFPS Website - tutorials, roadmap and support
- NeoFPS Unity Forum Thread - coming soon

Working With NeoFPS

Overview

NeoFPS is a framework for building PC and console based first person games.

Since it is impossible to cover all of the first person game types in a single asset, instead NeoFPS concentrates on the core mechanics of FPS games. Namely: movement, camera, interaction and shooting. It also provides implementations for features that Unity lacks out of the box, but which are especially important in first person games such as bindable controls, surface based effects, runtime settings and separating physics for shooting and movement.

NeoFPS was designed with the following goals:

- To help raise the quality bar of first person games created with the Unity game engine.
- To make it possible for a developer to achieve their vision of a first person game without placing constraints on how the game plays.
- To make the more complex mechanics of first person shooters accessible with a minimum of coding required.
- To allow developers to choose which features to use and which to replace without having to modify the code.

Installation

NeoFPS is a complex asset that requires a number of custom settings and has dependencies on other packages. Please see the Installation Instructions for a guide to getting NeoFPS up and running.

Getting Started

If this is your first time using NeoFPS, take a look at the Getting Started section for details on where to start. The Common Tasks section runs through some of the early tasks that are frequently raised in the NeoFPS Discord server and can point you in the right direction for the steps and documentation required.

Generated Constants

To meet the goals of enabling developers to achieve their vision and create complex mechanics without rewriting code, a flexible way of referencing objects and states is required. It would be easy to implement somthing using strings for keys, but the performance and memory impact of this can be a problem if the system is used in a lot of places or many times a frame. NeoFPS gets around this by providing a simple interface for creating constants similar to Unity's layers. These constants are turned into code that makes the constants easy to use in scripts as well as using them for properties in the inspector.

For more information see Generated Constants.

Layers and Tags

NeoFPS makes heavy use of Unity layers to enable complex features and to separate out scene elements for better performance.

For more information see Layers and Tags.

Game Settings

NeoFPS exposes a number of runtime settings to the player through text based settings files.

For more information see Game Settings.

See Also

NeoFPS Installation

Getting Started

Generated Constants

Layers and Tags

Game Settings

NeoFPS Installation

Overview

On first adding NeoFPS to a project, once the assets have been imported, the NeoFPS hub should appear, guiding you through the installation and providing useful ingormation on getting started. This document covers any extra steps and gives details of what is happening in case you want more manual control of the installation process.

Unity Settings

NeoFPS requires various project settings to be applied in order to function correctly. This includes custom layers, custom input axes, and an optimised layer collision matrix. NeoFPS has an automated system for applying Unity settings built into the NeoFPS Hub. It uses version numbers to track when updates to NeoFPS require new settings to be applied and will open the hub on the **Unity Settings** page if they are not up to date. You can also find the hub in the toolbar via *Tools/NeoFPS/NeoFPS Hub*.
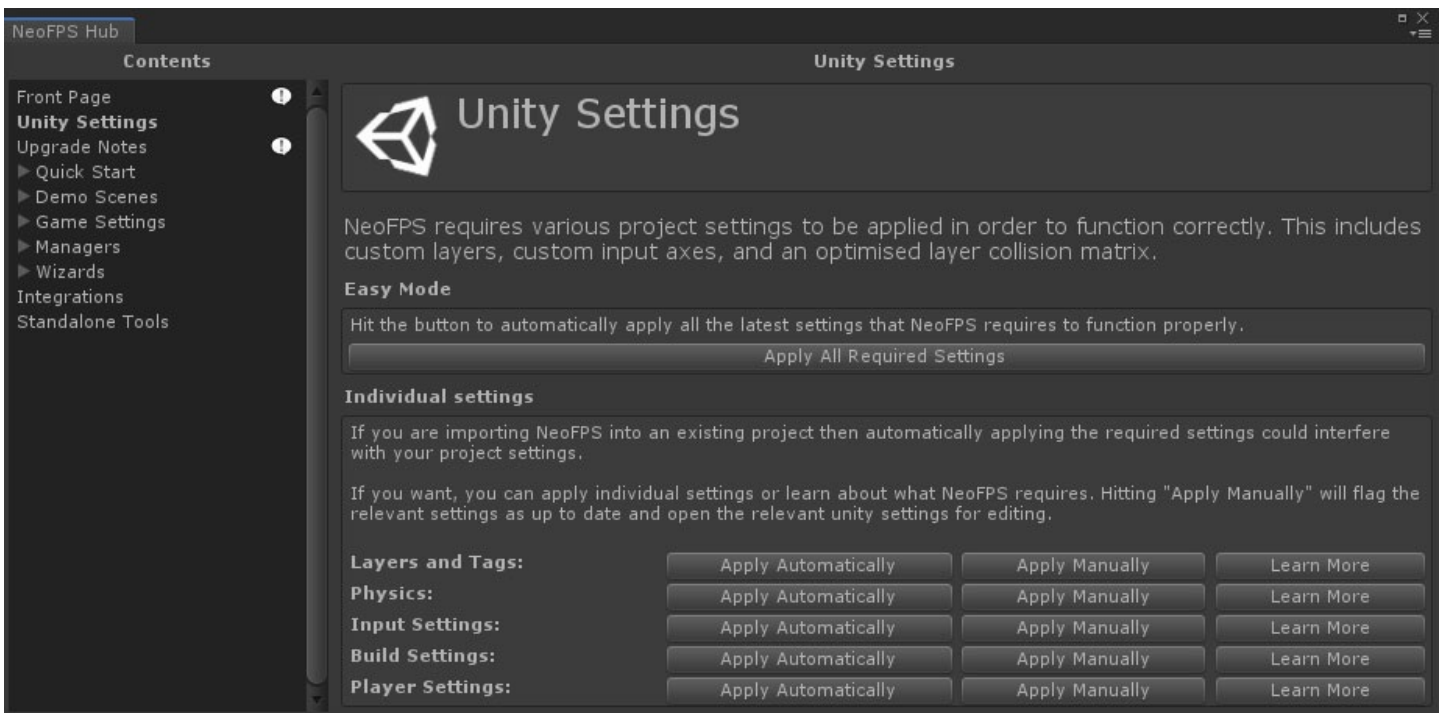
You can apply all required settings automatically here. If you are installing NeoFPS in a fresh project then this is the recommended approach. If you would prefer to apply the correct settings manually then you can do so using the **Individual Settings** section. Clicking the **Apply Manually** button for a section will open the relevant Unity settings editor and flag the settings as up to date in the NeoFPS hub. The following is a breakdown of what is required for each section:

SETTINGS	DETAILS
Layers and Tags	NeoFPS makes extensive use of custom layers. For more information, see the Layers and Tags documentation.
Physics	NeoFPS uses the physics layer collision matrix to define interactions between the custom layers. For more information see the **Layer Collision Matrix** section of the Layers and Tags documentation. This must be changed after the layers and tags have been set up.
Input	To enable consistent controller mapping across platforms, NeoFPS has some fairly complex input settings. You can find a full list in the Input Settings documentation.
Player Settings	NeoFPS uses the **Linear** color space (this will be set to "Gamma" by default). Without this change, the sample scenes will look very washed out and bright.

SETTINGS	DETAILS
Build Settings	See the *Sample Scenes* section below.

Sample Scenes

Sample scenes that demonstrate NeoFPS' features can be found in the project folder: *NeoFPS/Samples/SinglePlayer/Scenes*

The **MainMenu** scene is intended as a jumping off point for the individual demos and needs to be set as the first scene in the build settings (index 0). This means it will be the start-up scene for standalone builds. The **Loading** scene is a loading screen that is displayed when loading demo scenes. By default, this must be the second scene in the build settings (index 1), though this can be changed via the Neo Scene Manager in the hub.

The **Unity Settings** page applies the above settings automatically as part of the **Build Settings** section.

Post Processing

NeoFPS makes use of Unity's Post Processing Stack V2 package. On first adding NeoFPS to your project this should be installed automatically. If you want to check manually that the package is installed then you can access the package manager via the toolbar at: *Window/Package Manager*. Once opened, click the *All* tab at the top of the package list on the left. Scroll down and select *Post Processing*. At the bottom right of the description panel is an install button unless the package is already installed.

Due to a bug in the way the **PostProcessLayer** component is serialized, this has been removed from all cameras and replaced with a script which attaches and sets up the component at runtime. For more information see the PostProcessLayerFix behaviour reference.

See Also

Layers and Tags

Input Settings

Getting Started

NeoFPS Hub

On first adding NeoFPS to your project, once the assets have been imported, you should see the NeoFPS hub window appear:
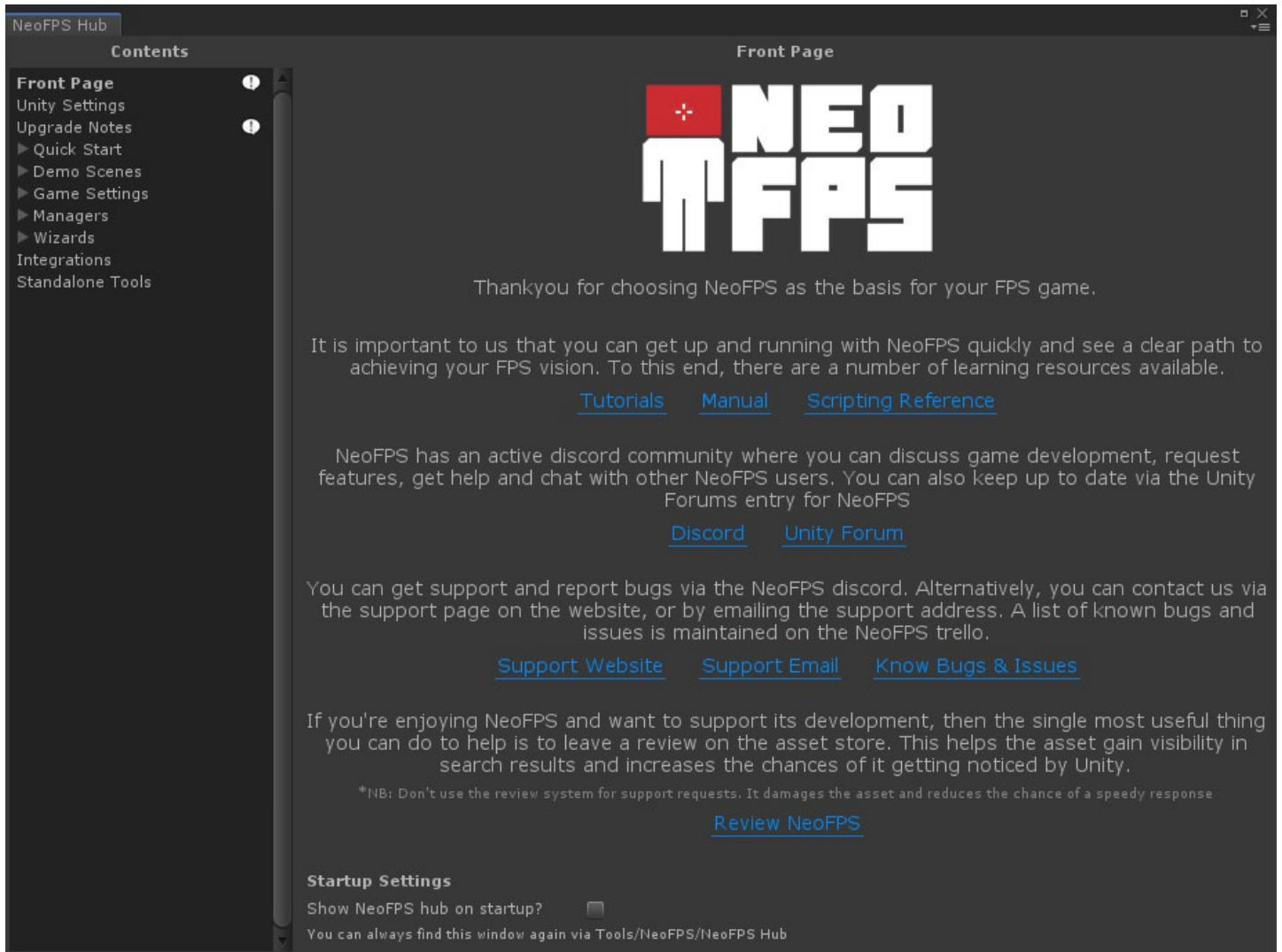
The hub will notify you of any changes to the required settings that must be applied (for more information see NeoFPS Installation). It also provides a quick way of accessing the sample scenes along with helpful links for getting the most from NeoFPS;

The hub pop-up will only be shown automatically this first time. If you want to see it again then it can be found in the toolbar via *Tools/NeoFPS/NeoFPS Hub*. Alternatively you can check the **Show hub on startup** checkbox at the bottom of the front page, and it will be shown each time the project is loaded.

Down the left-hand side of the hub is a navigation pane that lets you choose the page to view. The hub is split into the following sections:

- The **Front Page** contains useful links to aid with learning NeoFPS, getting support and engaging with the NeoFPS community.
- **Unity Settings** is used to apply the Unity settings that NeoFPS requires.
- **Upgrade Notes** tracks the installed version of NeoFPS and describes the steps required to upgrade to the latest update (both in scripts and components)
- **Quick Start** contains a selection of readme sub-pages which aid in getting up and running in NeoFPS, link to the relevant docs, and highlight useful assets and folders.
- **Demo Scenes** lists the available demo scenes and provides a quick way to find and load them. Once loaded, the **Scene Info** sub-page displays a readme for the scene with links to relevant docs and scene or project items. You can also find an

object in each demo scene called **Readme** which contains this information.

- **Game Settings** organises the player facing game options, allowing you to change default / starting values or delete your local settings files. These are the options the player can access through the in-game menus such as volume settings, mouse sensitivity, etc.
- **Managers** organises the manager settings for the various NeoFPS systems.
- **Wizards** provides a series of tools for quickly creating and setting up items in NeoFPS. For more information, see NeoFPS Wizards.
- **Integations** lists the available integrations for NeoFPS and provides links to the asset store and GitHub repositories. For the latest integrations, see the integrations page on the website.
- **Standalone Tools** lists the non-hub based tools available in NeoFPS such as the motion graph editor, the motion debugger, and the save file inspector.

Project Structure

All NeoFPS assets are contained in the **NeoFPS** folder. Inside this there are 4 subfolders:

- **Constants** contains all the Generated Constants used by NeoFPS and its samples.
- **Core** contains all the code and assets required by NeoFPS.
- **Resources** contains the managers and settings assets that need to be loaded at runtime.
- **Samples** contains a number of sample assets and scenes that demonstrate the different features of NeoFPS and provide a reference for implementing your own contents and mechanics. For more information, see Samples.

Using NeoFPS In Your Game

The **Scene Setup Quick-Start** guide available in the hub details the required scene components and links to the relevant prefabs in the project.

The quickest way to start prototyping in NeoFPS is to use the template scene as a starting point. You can find the template scene at the following location:

Assets\NeoFPS\Samples\SinglePlayer\Scenes\FeatureDemos\Template\FeatureDemo_Template.unity

You can either duplicate this scene as a starting point, or alternatively, if you already have a scene that you want to add then you can use Unity's multi-scene editing features to open the template scene additively in order to copy the relevant objects across. With your starting scene selected, open the template scene additively by right clicking on the scene asset and selecting the **Open Scene Additive** option. Both scenes will now be shown in the hierarchy and you can copy objects from the template scene and paste them in your own scene. Once you are done, remember to close the template scene by clicking on its options dropdown in the hierarchy and selecting **Remove Scene**.

The important objects in the demo scene are:

- **SimpleSpawnerAndGameMode** is a prefab game setup that handles player spawning and death (explained below). You can also find this prefab at*Assets\NeoFPS\Samples\SinglePlayer\Prefabs\SimpleSpawnerAndGameMode.prefab*
- **GlobalProfile** is a post-processing profile. You can use this or create your own.
- **SceneSaveInfo** is required in any scene that you want to support the NeoFPS save game system. This is not a prefab as its settings are unique to each scene. You will also need to add the scene to the build settings.
- **ScenePoolHandler** is used by the pooling system to manage pooled objects. This will be created automatically if not found, but including one in your scenes leads to a smoother experience.
- **HudAndMenuCanvas** contains the ingame menu and player HUD
- **HudAndMenuEventSystem** contains a Unity UI event system that handles input in the HudAndMenuCanvas object above.

In NeoFPS, the player character is not usually placed directly in a scene. Instead it is spawned at runtime from a spawn point and attached to a player object. This system allows for features such as swapping characters and respawning while persisting player information. The **SimpleSpawnerAndGameMode** prefab listed above contains the relevant spawner, along with a minimal game mode called **FpsSoloGameMinimal**. This component references the character object and player object to spawn. For more information, see Game Modes and FPS Characters.

The default demo character can be found at *Assets\NeoFPS\Samples\SinglePlayer\Prefabs\NeoFpsSoloPlayerCharacter.prefab*. You can also find a spawnerless character called *PrototypeSpawnerlessCharacter.prefab* at the same location. This can be placed directly in a scene instead of the SimpleSpawnerAndGameMode object, but it is limited in how it can handle player death (it defaults to reloading the scene).

To add environmental features to your scene, see Layers and Tags for details on what layers to use for environmental geometry, and Surfaces to add surface audio and impact effects.

For information on how to set up weapons for your game, see Weapons.

To add interaction to your game such as switches, doors and puzzles, see Interaction.

To add your own characters or customise the sample character, see FPS Characters.

NeoFPS is also designed to be extended to suit your vision. Each of the features is implemented in such a way that it can be replaced or expanded. For more information see Extending NeoFPS.

See Also

NeoFPS Installation

Samples

Common Tasks

Overview

The following are tasks that are frequently raised by new users on the NeoFPS Discord server. More will be added over time based on user feedback.

How To Interact With Objects In A Scene

NeoFPS lets you walk up to objects in a scene and use them or pick them up with the interact button using the Interactive Objects system.

There is an interactive object creation wizard available in the NeoFPS Hub that can create new interactive objects and set them up with render geometry, physics, sound effects and animations. Once you have an object set up, you would use the events on the interactive object behaviour to trigger your own scripts or components. Alternatively you can create a script that inherits the InteractiveObject class and extend its `Interact()` method.

One example of an interactive object is the scene switcher buttons in the persistence demo. The use the interactive object events to trigger a NeoFpsSceneSwitcher component when pressed, changing the scene.

How To Remove Weapons From A Player Character And Add Your Own

The player character's loadout is specified in the inventory component attached to the root of the prefab. There are 3 inventory types implemented in the demos: FpsInventoryQuickSwitch, FpsInventoryStacked and FpsInventorySwappable. Each of these have a **Starting Items** list in the inspector. You can add items to this as required. The **Backup Item** is the inventory item that is used when all others have been dropped, or when the selected item is holstered. By default, this is the demo hands, but it can be switched to any inventory item you like.

You can also create inventory loadout assets in the project hierarchy using right-click: *Create/NeoFPS/Inventory/Loadout*. These are assigned to the game mode component in your scene (for example the FpsSoloGameMinimal component). When the game mode spawns a character, it will assign it that loadout instead of its default starting loadout. The backup item (hands) will remain the same. See FpsInventoryLoadout for more details.

How To Add New Item Keys To The Inventory

Inventory items are each identified using a unique key. These keys are stored in the inventory database which is divided into tables. When adding your own inventory item keys, the first thing you should do is create a new FpsInventoryDbTable for your project by right-clicking in your project folder and using *Create/NeoFPS/Inventory/Database Table*. You then need to add the table to the inventory database by opening the database in the inspector (you can find it in the NeoFPS Hub managers section, or in the *NeoFPS/Resources* folder) and dragging your new table asset into the **Add Table Asset** field at the top. This means that there is no risk of the keys you add being overwritten when updating the asset (though you might need to re-add your table to the database).

Once you have an inventory database table for your project, there are a number of ways you can add a key. Firstly, you can select the table asset and use the controls at the bottom of the inspector to add a new key. Secondly, you can select the inventory database itself in the hub or resources folder. Clicking on your project table in the database table list will show its contents below. Lastly, the most convenient approach is to add the key at the moment you need it. Whenever a component has an inventory key property that yu can set in the inspector, it will pop up the inventory browser when clicked. At the bottom of this is a section to add a new key. You must select your project's table, insert a name for the item, and click create. This will add the key to the selected inventory database table, and assign it to the property that opened the browser.

Renaming a key in the asset database will not break connections. Any items that use that key will use the new name automatically.

For more information see the Inventory section.

How To Make Player Weapons, Items And Health Persist Across Play

Sessions

NeoFPS has a full save game system built in which can also be used to save and load persistent data to memory between scene switches.

The save system revolves around the NeoSerializedGameObject component which must be added to each GameObject with components that need saving (including its transform). You can tell the NeoSerializedGameObject which child objects, components and properties to save as required. You can also add overrides for different save modes such as **persistence**. You would usually save less information for persistence data. For example, you don't want to save the position of the character or the state of the selected weapon's animations. You do want to save information such as their health though. By default, all components and child objects will be saved.

If you have your own scripts or components that need saving then you can do it one of two ways: Firstly, you can modify your script to inherit from the `INeoSerializableComponent` interface and implement its `ReadProperties` and `WriteProperties` methods. If you don't have access to modify the scripts then you can write formatters that read/write them to the save system's binary format. For more information see Serializing Data.

How To Make Your Own Stat System Affect Movement And Combat

The NeoFPS motion graph is the movement system that drives the player characters. Each movement state can be set to use motion data such as movement speed, jump height or directional multipliers. This motion data can be overriden by assigning an object to the motion controller that implements the `IMotionGraphDataOverride` interface. The motion graph has a data override built in that just lets you assign different values in an asset, but you can also use MonoBehaviours that change the values at runtime. This allows you to base speed on character level or attributes, or add buffs and debuffs.

For stat based health effects, you would need to write a behaviour that implements the `IHealthManager` interface, or inherit from one of the existing health managers and add your extra stat based behaviour.

To modify the way firearms work based on stats, you can create your own ammo effect module scripts that deal damage based on the shooter's stats, or inherit from the existing `BulletAmmoEffect` to do the same thing.

For more information on extending NeoFPS, see the Extending NeoFPS section of the docs (online only).

How To Control Where The Player Spawns

In NeoFPS the player character is spawned in to the scene using spawn points. Spawning is controlled via the game mode along with what action to perform on the character's death. The demo scenes use a prefab called the *SimpleSpawnerAndGameMode* that has a game mode and a spawn point built in.

There's a few ways that you can control where the player character spawns. By default, spawn points register with the game mode and spawn system on Awake. If multiple spawn points are registered, you can choose if they will be used in the order they were registered or at random. You can also add an OrderedSpawnPointGroup to enforce the registration order.

To track progress through a level, you can enable and disable spawn points as you reach checkpoints. Combining this with the save game system, you can add autosaves when reaching the checkpoint. If the spawn points each have a NeoSerializedGameObject attached then their active state will be saved, meaning that the character will spawn at the correct spawm points when the game is re-loaded.

If you want a much simpler system for the sake of prototyping, then you can also create a spawnerless character that can be placed directly in the scene, though this is a much less flexible setup. For more information see spawning.

See Also

NeoFPS Wizards

Samples

NeoFPS Wizards

Overview

NeoFPS comes with a number of item creation wizards to simplify the setup of common objects such as characters and weapons. These can be found in the **Wizards** section of the NeoFPS Hub, which is available through Unity's menus at *Tools/NeoFPS/NeoFPS Hub*. Each wizard (with the exception of the script creation wizard) allows you to create an object and/or save a template for use when creating your next weapon.

Available Wizards

The following wizards are available to use now:

Modular Firearm Wizard

The modular firearm wizard runs through the setup of a modular firearm, from assigning a view model to picking the relevant firearm modules, and even adding advanced features such as weapon overheat, bullet penetration and sprint animations. The wizard does not set up an animator controller for the weapon, but if you have one available then you can attach it and hook up the parameters through the wizard.

Player Character Wizard

The player character wizard allows you to build a playable character prefab. This includes setting up health, inventory and more advanced features such as stamina. The end result can be added to a spawner, and spawned at runtime immediately after setup.

Melee Weapon Wizard

The melee weapon wizard runs you through the creation of a basic melee weapon prefab for use in NeoFPS. This includes setting up the weapon itself, how it appears in the inventory, and more advanced settings such as sprint animations. The wizard can also create a new animator controller for you based on the weapon's settings, allowing you to pick specific animation clips to use. Alternatively, you can hook up an existing animator controller so the correct parameters are used.

Thrown Weapon wizard

The melee weapon wizard runs you through the creation of a basic thrown weapon prefab for use in NeoFPS. This includes setting up the weapon itself, how it appears in the inventory, and more advanced settings such as sprint animations. The wizard can also create a new animator controller for you based on the weapon's settings, allowing you to pick specific animation clips to use. Alternatively, you can hook up an existing animator controller so the correct parameters are used.

Pickup Wizards

The pickup wizard is used to create pickups / powerups that the player character can use. The available pickup types are:

- Wieldable items such as melee or thrown weapons.
- Modular firearm drops (these track the ammo in the weapon when it is dropped).
- Inventory item pickups.
- Multi-item inventory pickups.
- Health packs.
- Shield boosters.

Interactive Object Wizard

The interactive object wizard allows you to add NeoFPS' interaction system to objects to allow the player character to use them. This includes assigning geometry and physics, highlighting when looked at, as well as some preset interactions such as playing animations or audio. You can then use the events on the created prefabs' InteractiveObject component to trigger your own code or components.

Script Creation Wizard

The script creation wizard provides a number of template scripts for various NeoFPS features that you can use as a starting point. This includes motion graph elements, firearm modules, save system formatters, and input behaviours. To use the wizard:

- Select the script type from the dropdown.
- Give it a name and a namespace.
- Fill in any script specific properties that appear.
- Select output folders for the generated scripts. If these are not set, then the new scripts will be created in the root of the *Assets* folder, while any editor scripts will be created in *Assets/Editor*.

Coming Soon

The following wizards are coming soon:

- **Firearm AnimatorController Wizard** will take a modular firearm prefab, and create and set up an AnimatorController based on the modules it has attached.

See Also

Getting Started

Game Modes

Overview

Game modes in NeoFPS are intended to control the flow of a game. This means creating a player object, spawning and respawning characters and attaching them to the player. They are derived from the `FpsGameMode` base behaviour which is an abstract class and therefore cannot be placed in the scene without a specific implementation.

In this version of NeoFPS the only implementation available is called FpsSoloGameMinimal. This implementation simply spawns both a player and character from prefabs on start. On a character's death, the FpsSoloGameMinimal can be set to either spawn a new character, reload the scene, return to the main menu, or load the last valid save game (for more information on save games, see Save Games. The minimal implementation is designed for testing and development of prototypes and as a template for implementing a custom game mode for your own game.

The game mode also handles persisting data such as player health and inventory between scenes, also using the save game system. This can be toggled on or off.

Game modes will be expanded in future versions of NeoFPS to provide a simple starting point when developing common types of FPS games.


See Also

Working With NeoFPS

Save Games

Generated Constants

Overview

To meet the goals of enabling developers to achieve their vision and create complex mechanics without rewriting code, a flexible way of referencing objects and states is required. It would be easy to implement something using strings for keys, but the performance and memory impact of this can be a problem if the system is used in a lot of places or many times a frame. NeoFPS gets around this by providing a simple interface for creating constants similar to Unity's layers. These constants are turned into code that makes them easy to use in scripts as well as using them for properties in the inspector.
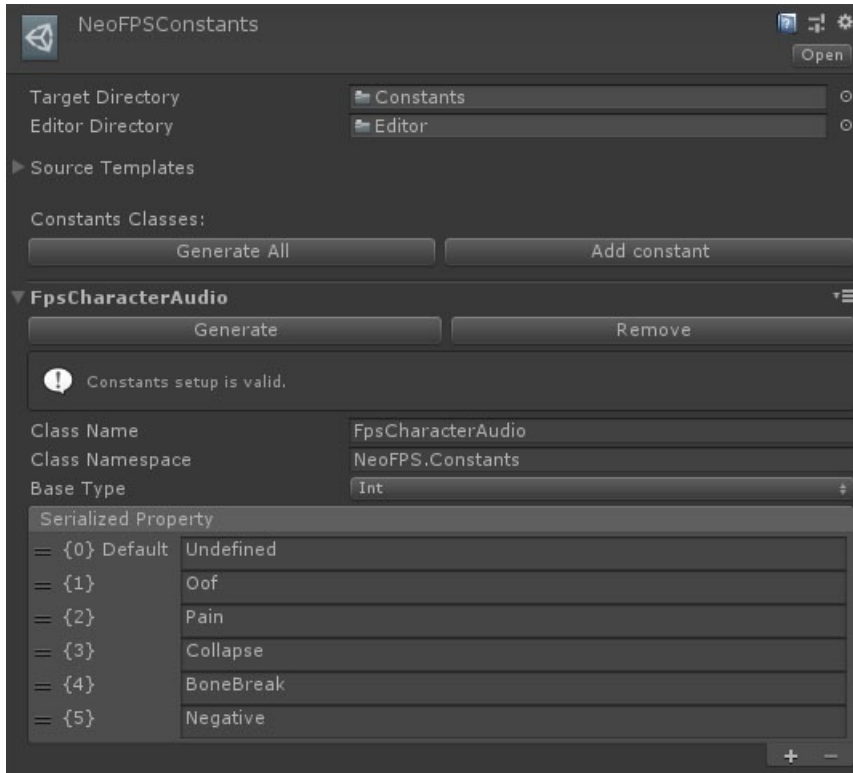
The ConstantsSettings scriptable object specifies constants to generate for your game. You can have as many of these settings files as you like and add new constants as required. If you do plan to add your own constants then it is best to create a new constants settings file in case the one that comes with NeoFPS is modified in a future version. It is also important to consider whether it is best to use a constant or a string. If the situation involves a set of keys or IDs that are referenced frequently, but will rarely be changed then generated constants are a good fit. If there is a high chance that new keys will be needed or the keys will be changed regularly, then it might be better to use strings for the ID and a dictionary to store items.

Generated Constants Limitations

The biggest limitation with generated constants is that they are serialized by index. This means that reorganising the values will not be reflected in the inspector.

As an example, let's say you have a **Vehicle** constant with the following values:

1. Bike
2. Car
3. Truck

You also have a monobehaviour that stores a vehicle as a serialized field and in the inspector it is set to **Car**. Later on you decide to add a number of new values as follows:

1. Bike
2. Motorbike
3. Car

4. Van
5. Truck

Looking at the previous monobehaviour in the inspector, it would now be set to **Motorbike**. This is because the monobehaviour only actually references the number 2, not the word "Car". You can work around this by making sure to keep the order the same and only add new values to the end.

If you remove values instead of adding, then any serialized properties that reference values beyond the new limit will be reset to the first constant value. It is best practice to reserve the first value (0) as a default with a name that reflects this.

Generated Constants In Scripts

You can use constants in scripts as though you were working with an enum. For example, the `FpsInputAxis` constant can be used like so:

```
FpsInputAxis axis = FpsInputAxis.MouseX;
```

Constants can also be implicitly cast as in the following examples:

```
FpsInputAxis axis = FpsInputAxis.MouseX;
var element = inputAxisArray[axis];
```

and:

```
FpsInputAxis axis = 2;
```

This makes them very efficient as keys to store items. If you were using strings, you would need to store and reference objects from a dictionary as follows:

```
var dictionary = new Dictionary<string, MyObject>();

//...

MyObject result = dictionary[myStringKey];
```

This is a relatively expensive operation as the string needs to be hashed, and then the dictionary searched for the key value pair that the key references. If the key is not found then this code would throw an exception so you would also want to add extra error checking for safety. With constants the object can be stored and referenced in an array as follows.

```
var array = new MyObject[MyConstant.count];

//...

MyObject result = array[myConstantKey];
```

This is a much faster operation as you are simply accessing an array by index. By allocating an array that is the size of the constant (count is a preset property in the constants templates included with NeoFPS) you are guaranteed that accessing via a constant key will never be out of bounds. Preallocating like this is a good speed optimisation, but if the constant has a lot of values, and the array is sparsely populated then it would be wasteful to implement in this way. If so then you could either use the string option, or a dictionary with constant keys to save the string hashing step.

How They Are Generated

constants are generated by taking a template script and replacing sections with values in the settings file. The following keys are defined which will be replaced:

KEY	DESCRIPTION
%NAME%	The generated constants container name.
%NAMESPACE%	The namespace for the generated constants container.
%TYPE%	The underlying type for the value (for example **int**, **ushort**, **byte**).
%VALUES%	The values to be added. These will be written as a number of const values of the specified type.
%VALUE_NAMES%	An array of strings to be populated with the value names. This is used for the inspector value dropdown among other things.
%COUNT%	The number of values written.

The generation process requires 2 templates. One for the output constant, and one for an editor drawer that draws the constant in the inspector as a dropdown.

See Also

ConstantsSettings

Layers and Tags

Overview

NeoFPS has a number of systems that require objects to be filtered using layers. The following is the layer set-up required by NeoFPS, along with an overview of some useful features for working with layers in code.

NeoFPS Layers
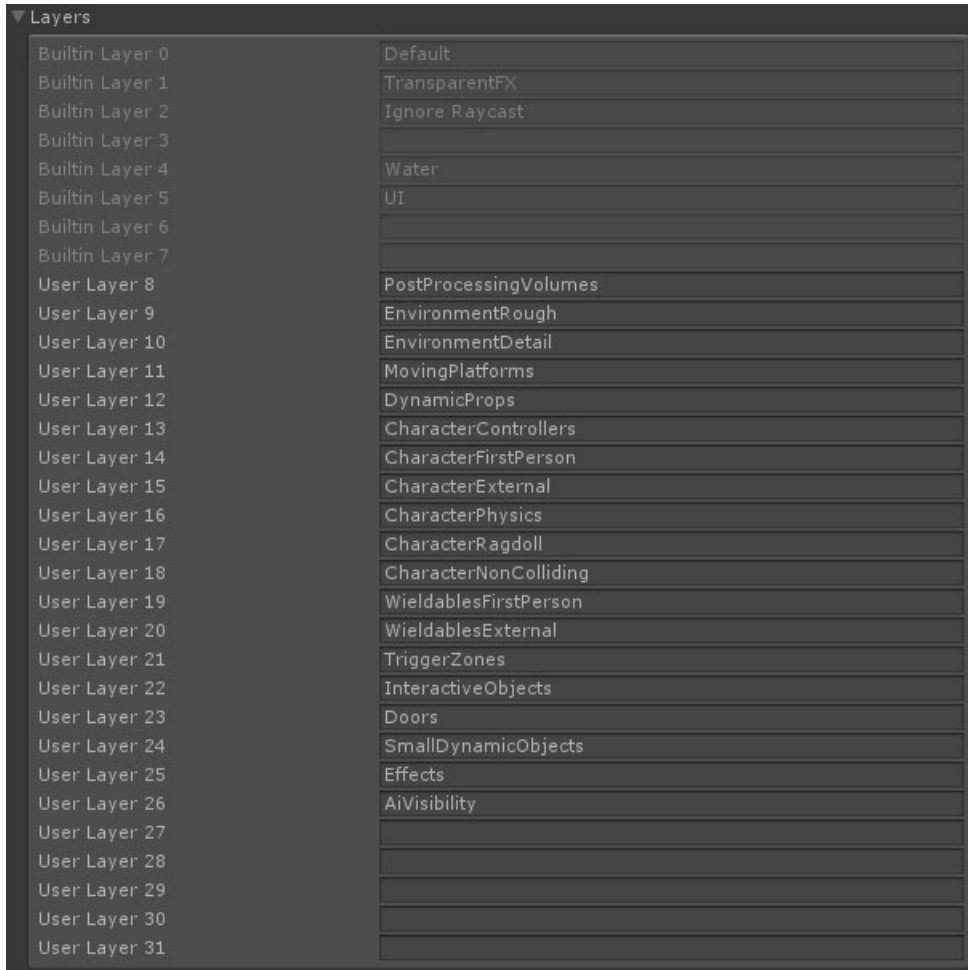
NAME	DESCRIPTION
PostProcessingVolumes	Trigger volumes used by the Unity post processing system to define override volumes.
EnvironmentRough	Low detail mesh and primitive colliders used for character motion and traversal.
EnvironmentDetail	High detail mesh and primitive colliders used for weapon impacts.
MovingPlatforms	Low detail mesh and primitive colliders used for character motion on moving platforms.
DynamicProps	Dynamic rigidbody props. Used for larger objects that could affect characters such as barrels.
CharacterControllers	Used for character controller root objects.
CharacterFirstPerson	Character geometry and objects visible from the first person view.
CharacterExternal	Character geometry and objects visible from the external views.

NAME	DESCRIPTION
CharacterPhysics	Character body colliders, used for things like bullet impact detection.
CharacterPhysics	Character body colliders used for ragdolls, so set up to collide against the ground but not detail physics.
CharacterNonColliding	Used for objects that are tested against using Physics casts, but not the Unity collision system.
WieldablesFirstPerson	First person geometry and colliders for weapons and other wieldable objects.
WieldablesExternal	Geometry and colliders for weapons and other wieldable objects when seen from an external view.
TriggerZones	Trigger volumes that act on character triggers.
InteractiveObjects	Low detail trigger volumes used for detecting interactive objects.
DoorPhysics	Colliders for door objects.
SmallDynamicObjects	Small rigidbody objects that should not noticably affect characters. A character can push them around, but they can't really push a character.
Effects	A layer used for debris and particle effects.
AiVisibility	Low detail colliders used by AI for visibility checks.

NeoFPS Tags

NAME	DESCRIPTION
AI	Used to tag AI characters.

Layer Collision Matrix
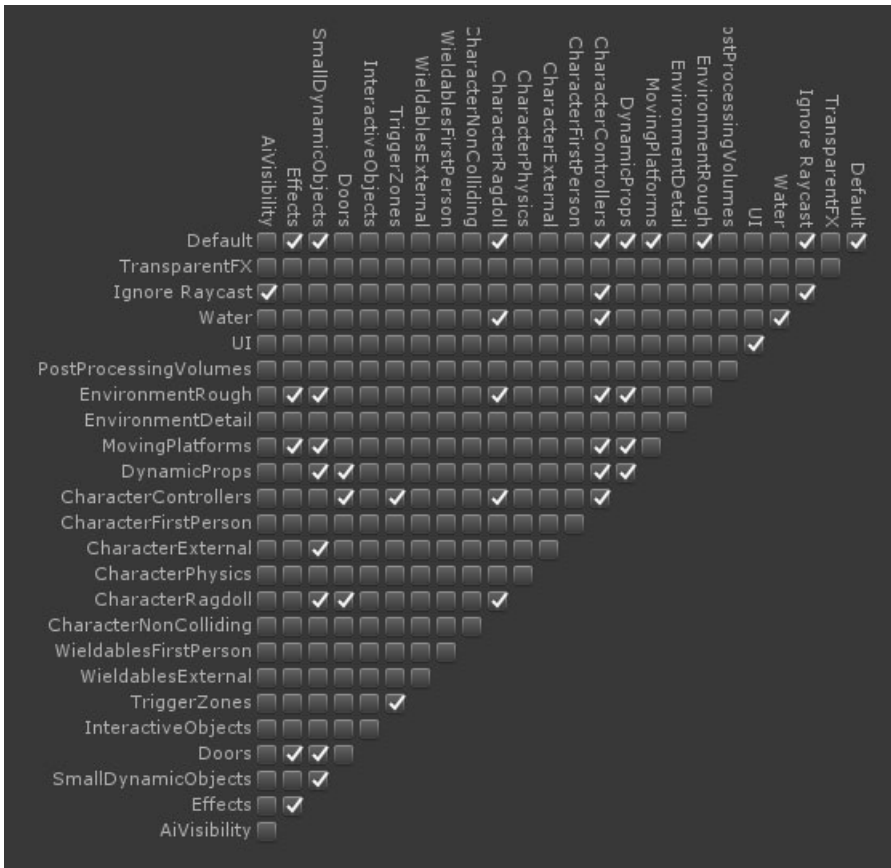
The layer collision matrix is used to define which layers can interact with each other in the Unity physics system. Keeping a minimum of layers from colliding with each other is important to maintaining performance as well as for keeping interactions between different systems clean.
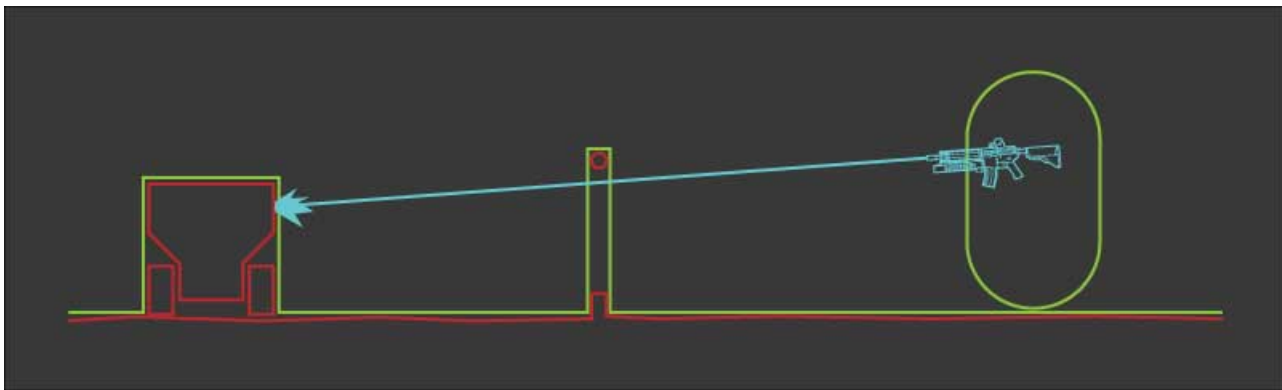
In the above example image, the green objects are character blockers on the CharacterRough and EnvironmentRough layers that are used for moving characters around the scene. The red objects are bullet blockers on the EnvironmentDetail layer that are used for bullet hits.

In code you can use the `PhysicsFilter` type to define filters when performing physics operations. This also has a number of preset constants for aiding in writing code.

`PhysicsFilter.LayerIndex` contains the layer indices matching the layers as specified in the layers and tags settings.

`PhysicsFilter.LayerFilter` contains a number of preset physics filters matching the layers as specified in the layers and tags settings.

`PhysicsFilter.Masks` contains a number of filters for commonly used groups of layers as follows:

NAME	DESCRIPTION
BulletBlockers	Layers that bullet raycasts can hit.
CharacterBlockers	Layers that characters traverse. Includes larger props and platforms.
DynamicCharacterBlockers	Layers that characters traverse. Excludes static environment physics.
Interactable	Colliders attached to interactive objects, along with environment colliders that can block them from view.
SpawnBlockers	Any dynamic objects that can block spawn points.
ShowDecals	Objects that can accept decals.
AiVisibilityCheck	AI visibility markers and environmental colliders that can block them from view.

See Also

Unity Tags and Layers

Unity Layer Based Collision

Game Settings

Overview

NeoFPS exposes various game settings to players through text based settings files. This provides much more flexibility for players than the existing system of build settings that Unity hides for runtime builds.

The following settings files are created by a NeoFPS project when it is run:

NAME	FILENAME	DESCRIPTION
Audio	Audio.settings	Contains settings for volumes (spatial effect, music and global.
Graphics	Graphics.settings	Contains various graphics and quality settings such as resolution, vsync and antialiasing.
Gameplay	Gameplay.settings	Contains miscellaneous gameplay settings such as crosshair colour.
Input	Input.settings	Contains settings for mouse and keyboard input including sensitivity, smoothing and acceleration.
Gamepad	Gamepad.settings	Contains settings for gamepad input such as profile preset and analog sensitivity.
Key Bindings	KeyBindings.settings	Contains individual key bindings for the different inputs.

If any of these files are found when the game starts, then the game settings will be based off these. The individual settings are mapped to the following scriptable objects:

NAME	BEHAVIOUR
Audio	FpsAudioSettings
Graphics	FpsGraphicsSettings
Gameplay	FpsGameplaySettings
Input	FpsInputSettings
Gamepad	FpsGamepadSettings
Key Bindings	FpsKeyBindings

If no settings file is found for one of the above, then the settings will default to those in the scriptable object asset located in the *NeoFPS/Resources* folder. These assets can be customised to set the default settings. Modifying the settings inn game and via the ".settings" files will not modify the assets.

If any of the settings are changed at runtime then the ".settings" file is saved again and an event is fired. This event allows objects to interact to changes of settings. For example, if the crosshair colour is changed in the gameplay settings is changed then the HUD crosshair can immediately change to reflect this.

These settings behaviours and files are intended as a basis for your own projects. Add any new settings as required and use these files as an example though be aware that these files will change in future versions of NeoFPS to reflect new features such as HDRP and post-processing settings potentially being added to the graphics settings.

See Also

NeoFPS Shaders

Overview

NeoFPS contains a number of shaders that have been created for the built-in (standard) render pipeline. It does not support the SRPs such as HDRP or URP out of the box.

The NeoFPS shaders were created using Amplify Shader Editor. If you own this asset then you should be able to load the shaders up in ASE and convert most of them to your target SRP by changing the shader type in the output node and re-attaching the relevant connections.

Available Shaders

All of the NeoFPS shaders are currently located in the folder: *Assets\NeoFPS\Samples\Shared\Effects\Shaders*

They include the following:

- **NeoFPS_DissolveTrail_...** are used for hitscan bullet trail. They start as a solid tracer line and then dissipate. The edge blend shaders fade off the tracer texture at the edges of the trail's line renderer.
- **NeoFPS_DistortionTrail** is another bullet trail which provides a shockwave style distortion.
- **NeoFPS_FireballSheet...** can be used for particle effects and pull flipbook style greyscale fireballs from the individual channels of a texture and use a gradient to add colour.
- **NeoFPS_FirstPersonWeapon_Stencil...** are used for the first person weapon models that you need to be stencilled out when looking through stencil scopes (such as the scope mounting geometry).
- **NeoFPS_HeatHaze** is an animated distortion which is used to add a rippling heat haze effect.
- **NeoFPS_PowerupGlow** overlays a colourful animated shimmer over a standard shader to highlight objects in the scene.
- **NeoFPS_Shockwavce_Standard** provides a noisy distortion effect around the edges of a shockwave. The shockwave strength is influenced by vertex colour, making it useful for particle effects (start white and fade to black over lifetime).
- **NeoFPS_Water** is a simple water surface shader used for the swimming demo scenes.
- **NeoFPS_WaterCaustics** is a caustics volume shader that can be used to add a caustics lighting effect when underwater. *This shader is only compatible with deferred rendering.*
- **Glow/NeoFPS_Glow...** shaders add a heat glow to first person weapons when overheating. The glow can be masked by position ranges, distance from a point (both object space) or using an alpha mask.
- **Highlight/NeoFPS_InteractiveHighlight** adds a shimmer effect to a standard shader to help highlight interactive objects in the scene.
- **Optics/NeoFPS_HoloSightReticule** is used for stencil based reticules that can only be seen through the glass of a holographic or red-dot sight. It also adds brightness and colour controls through the material.
- **Optics/NeoFPS_HoloSightStencil** is used to define the area you can see the reticule through. It is not otherwise visible.
- **Optics/NeoFPS_HoloSightStencilGlass** is used to define the area you can see the reticule through. It also uses a standard transparent base.
- **Optics/NeoFPS_LaserPointerBeam** defines an animated noisy like with a stronger central line for a laser look.
- **Optics/NeoFPS_RedDotReticule** is similar to the holosight reticule, but without the need for a reticule mask texture.
- **Optics/NeoFPS_RenderTextureScope** displays a render texture, adding a parallax scope ring and fading to an opaque, reflective glass based on how far off the scope object axis the camera is.
- **Optics/NeoFPS_StencilScopeInner...** are used for the parallax effect when looking through stencil scopes. Objects using this shader will only be visible through the lens of the scope.
- **Optics/NeoFPS_StencilScopeLens** defines the stencil that the scope inner surfaces can be seen through, and which the **NeoFPS_FirstPersonWeapon_Stencil...** cannot.

NB: The stencil scope shaders are currently not working properly with deferred rendering. A solution is being investigated

Converting To HDRP / URP

There is no official support for HDRP/URP in NeoFPS, but it should be possible to convert the shaders to your target SRP using

[Amplify Shader Editor][asset-amplify].

You can find more information on using Amplify Shader Editor with the SRPs via the documentation, and in their Youtube video: Getting Started with ASE - SRP Pitfalls.

For the NeoFPS shaders, you will need to pay special attention to the **SubShader / Pass** section of the output node properties, making sure to recreate the **Depth**, **Blend Mode** and **Stencil**:

You can swap the shader type by using the **Shader Type** property at the top of the Common Properties. For example, to switch an Unlit shader to HDRP, you will need to swap the shader type to **HD/Unlit**. This will automatically update the depth and blend mode settings.

Due to the way that the SRPs handle stencils and grabbing screen pixels, the stencil based shaders such as the holographic sights and stencil scopes will not convert directly. You will need to write custom pass scripts to achieve this. How these work can vary between HDRP and URP, and between different versions as well. The same goes for distortion shaders such as the shockwave and heat haze shaders.

There is a future task for adding SRP compatibility to NeoFPS that aims to address these limitations and provide SRP support out of the box.

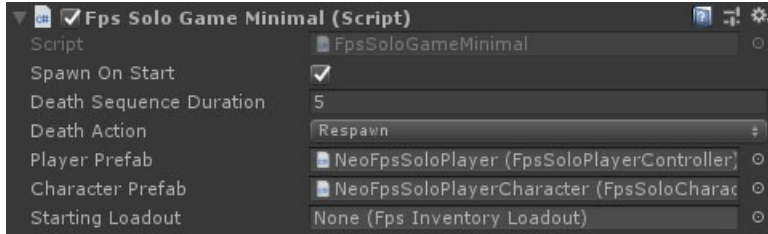# See Also

Modular Firearms

[Amplify Shader Editor][asset-amplify]

FpsSoloGameMinimal MonoBehaviour

Overview

The minimal game setup is intended for testing and development purposes. It simply spawns the specifed player and character, and can be set to perform various actions on the player character's death.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Spawn On Start	Boolean	Should the game mode automatically spawn a player character immediately on start.
Death Sequence Duration	Float	The time in seconds between the player character dying and respawning.
Death Action	Dropdown	What to do if the player character is killed. Available options are: • **Respawn** will spawn a new player character at the next available spawn point • **ReloadScene** will reload the scene from fresh, resetting everything • **MainMenu** will unload the scene and take the player back to the main menu • **ContinueFromSave** will load the last save (based on the SaveGameManager settings) if available and reload the scene if not.
Player Prefab	FpsSoloPlayerController	The player prefab to spawn if no player exists (players are persistant).
Character Prefab	FpsSoloCharacter	The character prefab to spawn.
Starting Loadout	FpsInventoryLoadout	An **optional** inventory loadout for the character on spawn (this will replace their starting items).

See Also

FpsSoloPlayerController

FpsSoloCharacter

FpsInventoryLoadout

ConstantsSettings ScriptableObject

Overview

The ConstantsSettings asset is used to generate and maintain arbitrary constants for a game. The generated output script contains a wrapped constant that acts in code like an enum, but can intrisically convert to its base type. A drawer is also generated that shows a dropdown with the constant values when the generated constant is serialized in the inspector. These constants have a number of uses such as object IDs or keys, without the overhead of using strings and dictionaries.

See Generated Constants for more information.

Inspector
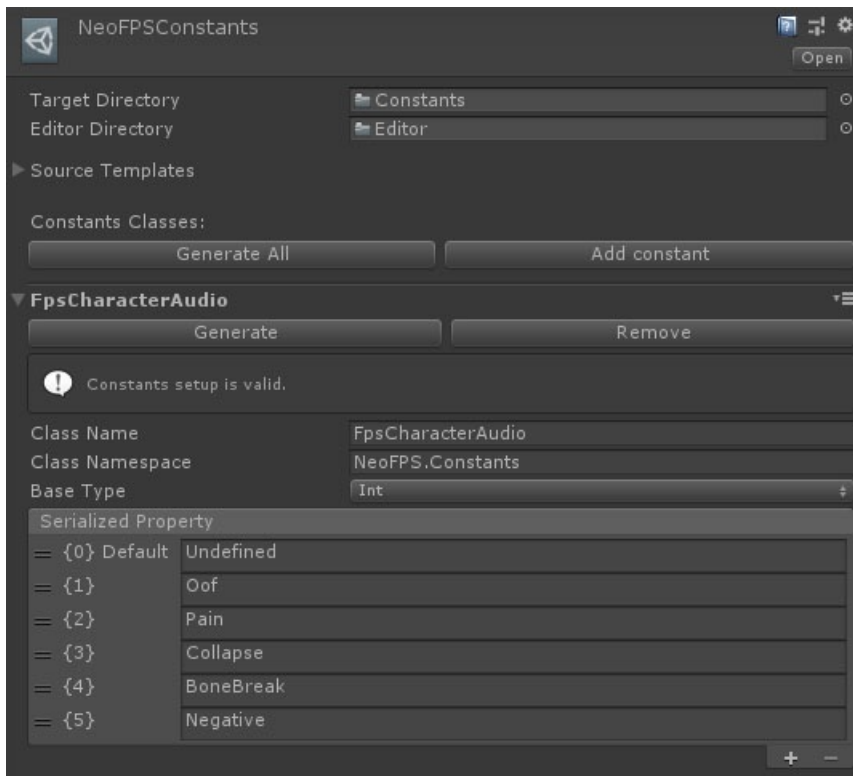
Properties

Output

NAME	TYPE	DESCRIPTION
TargetDirectory	Folder	Where the generated constant script should be output.
EditorDirectory	Folder	Where the generated constant editor script should be output.

Source Templates

NAME	TYPE	DESCRIPTION
ByteConstant	TextFile	The text file to use when generating byte constants.*
ByteDrawer	TextFile	The text file to use when generating a byte constant editor script.*
IntConstant	TextFile	The text file to use when generating integer constants.*

NAME	TYPE	DESCRIPTION
IntDrawer	TextFile	The text file to use when generating an integer constant editor script.*
UIntConstant	TextFile	The text file to use when generating unsigned integer constants.*
UIntDrawer	TextFile	The text file to use when generating an unsigned integer constant editor script.*
ShortConstant	TextFile	The text file to use when generating short constants.*
ShortDrawer	TextFile	The text file to use when generating a short constant editor script.*
UShortConstant	TextFile	The text file to use when generating unsigned short constants.*
UShortDrawer	TextFile	The text file to use when generating an unsigned short constant editor script.*

* For more information on how the template is turned into a script see Generated Constants.

Controls

The **Generate All** button will output a script file and a drawer script file to the specified directories for each constant, overwriting any existing files with the same name.

The **Add Constant** button will add a new constant settings entry to this scriptable object

Constant settings

Each constant has a number of common controls:

The **Generate** button will output a script file and a drawer script file to the specified directories, overwriting any existing files with the same name.

The **Remove** button will remove this constant from the scriptable object. Any existing generated files will not be touched.

Each constant also has the following properties:

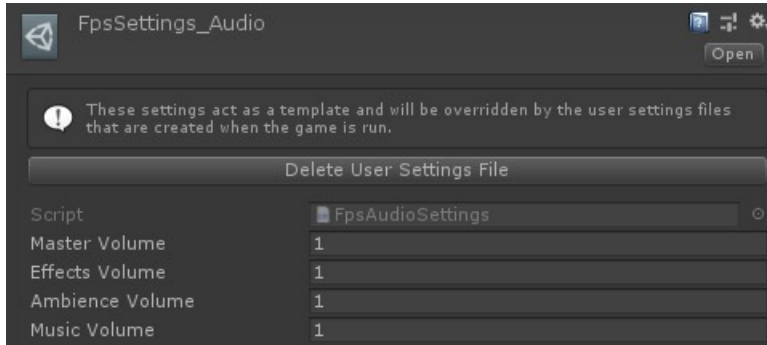| NAME	TYPE	DESCRIPTION
Class Name	String	The name for the output constant. This will also be the output script file name, while the output drawer script will be named Drawer.
Class Namespace	String	The namespace for the output scripts.
Base Type	Dropdown	This value specifies which source templates should be used to generate the constant.
Constant Values	String Array	A sequential array of constant value names. These must be valid names, and not duplicated. Use the **+** and **-** buttons to add or remove values, or reorder by dragging the handle on the left of the array entry.

See Also

FpsAudioSettings ScriptableObject

Overview

The FpsAudioSettings asset specifies the default audio settings and loads / saves them to a *.settings* file on disk for player editing.

The **Delete User Settings** button is used to delete the settings file that was generated by running the game. If the settings file exists then it will override the settings specified in this asset, so this can be useful if you have changed or added default settings.

Inspector

Properties

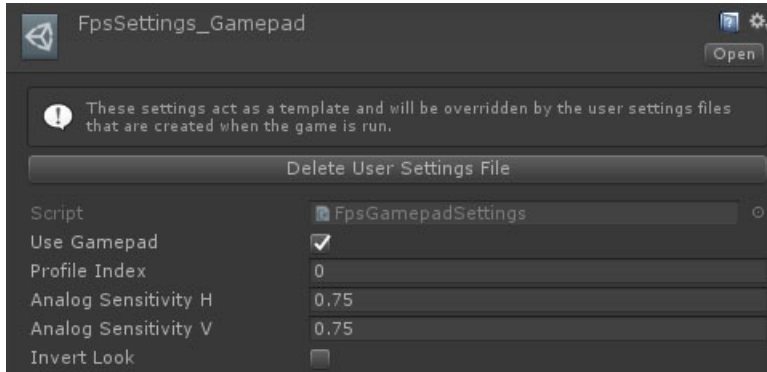| NAME	TYPE	DESCRIPTION
Master Volume	Float	The overall game volume.
Effects Volume	Float	The volume for in game effects.
Ambience Volume	Float	The volume for ambience effects.
Music Volume	Float	The volume for music.

See Also

Audio Systems

FpsGamepadSettings ScriptableObject

Overview

The FpsGamepadSettings asset specifies the default gamepad settings and loads / saves them to a *.settings* file on disk for player editing.

The **Delete User Settings** button is used to delete the settings file that was generated by running the game. If the settings file exists then it will override the settings specified in this asset, so this can be useful if you have changed or added default settings.

Inspector

Properties

| NAME	TYPE	DESCRIPTION
Use Gamepad	Boolean	Should gamepad input be registered.
Profile Index	Int	The gamepad profile to use.
Analog Sensitivity H	Float	The horizontal gamepad aim sensitivity.
Analog Sensitivity V	Float	The vertical gamepad aim sensitivity.
Invert Look	Boolean	Invert the gamepad vertical aim.

See Also

FpsGameplaySettings ScriptableObject

Overview

The FpsGameplaySettings asset specifies the default gameplay settings and loads / saves them to a *.settings* file on disk for player editing.
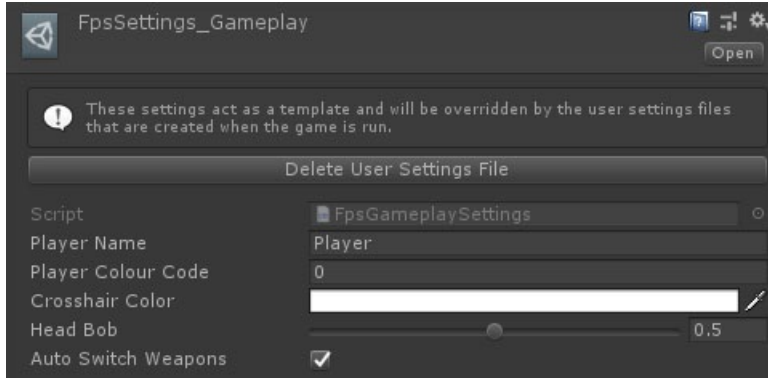
Inspector

The **Delete User Settings** button is used to delete the settings file that was generated by running the game. If the settings file exists then it will override the settings specified in this asset, so this can be useful if you have changed or added default settings.

Properties

| NAME	TYPE	DESCRIPTION
Player Name	String	The default player name.
Player Colour Code	Int	The default player colour.
Crosshair Colour	Color	The default crosshair colour.
Head Bob	Float	The ratio of head vs item bob. Head bob looks more natural when close to scenery, but some people can find it uncomfortable.
Auto Switch Weapons	Boolean	Should the player character automatically switch to a better weapon when picking it up.

See Also

FpsGraphicsSettings ScriptableObject

Overview

The FpsGraphicsSettings asset specifies the default graphics settings and loads / saves them to a *settings* file on disk for player editing.
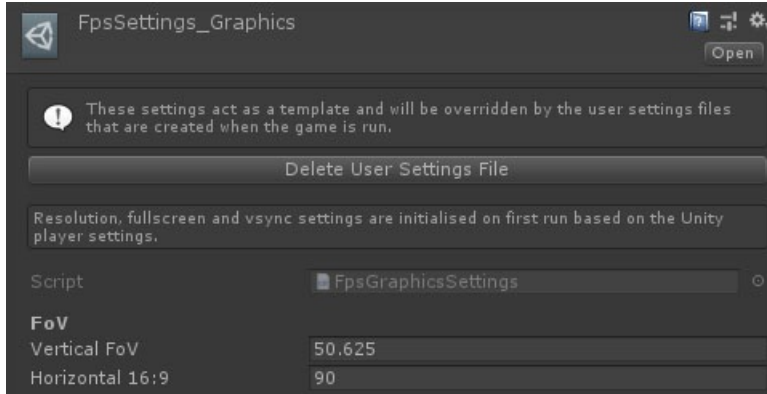
Inspector

The **Delete User Settings** button is used to delete the settings file that was generated by running the game. If the settings file exists then it will override the settings specified in this asset, so this can be useful if you have changed or added default settings.

Properties

| NAME	TYPE	DESCRIPTION
Vertical FoV	Float	The field of view vertically of the player camera. This is how field of view is saved.
Horizontal 16:9	Float	The horizontaly field of view on a 16:9 monitor. This is actually calculated from the vertical FoV, and when it is changed, the vertical FoV is modified.

See Also

Unity Graphics

Unity Quality Settings

FpsInputSettings ScriptableObject

Overview

The FpsInputSettings asset specifies the default input settings and loads / saves them to a *.settings* file on disk for player editing.
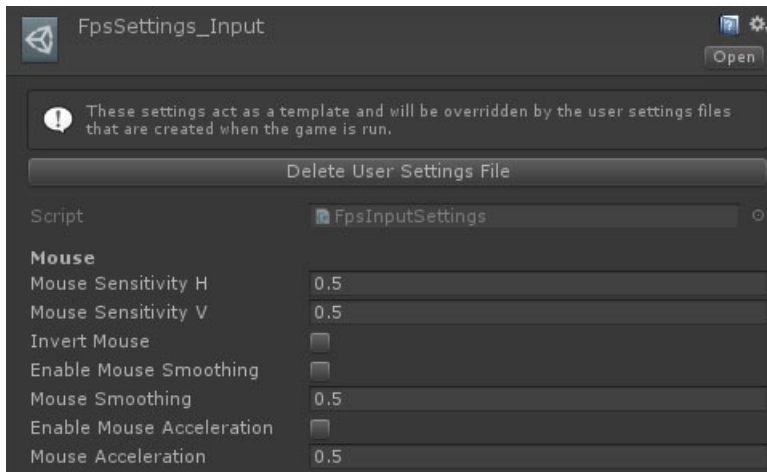
Inspector

The **Delete User Settings** button is used to delete the settings file that was generated by running the game. If the settings file exists then it will override the settings specified in this asset, so this can be useful if you have changed or added default settings.

Properties

| NAME	TYPE	DESCRIPTION
Mouse Sensitivity H	Float	The horizontal mouse look sensitivity.
Mouse Sensitivity V	Float	The vertical mouse look sensitivity.
Invert Mouse	Boolean	Invert the mouse vertical aim.
Enable Mouse Smoothing	Boolean	Mouse smoothing takes a weighted average of the mouse movement over time for a smoother effect.
Mouse Smoothing	Float	The amount of mouse smoothing to add.
Enable Mouse Acceleration	Boolean	Mouse acceleration amplifies faster mouse movements.
Mouse Acceleration	Float	The amount of mouse acceleration to add.

See Also

NeoFPS Input System

FpsKeyBindings ScriptableObject

Overview

The FpsKeyBindings scriptable object specifies the default input key bindings settings and loads / saves them to a *.settings* file on disk for player editing.
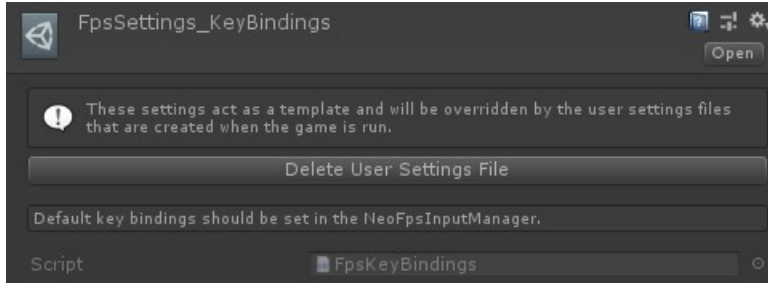
Inspector

The **Delete User Settings** button is used to delete the settings file that was generated by running the game. If the settings file exists then it will override the settings specified in this asset, so this can be useful if you have changed or added default settings.

Properties

No properties are exposed in the inspector. The default key bindings are specified in the NeoFpsInputManager instead.
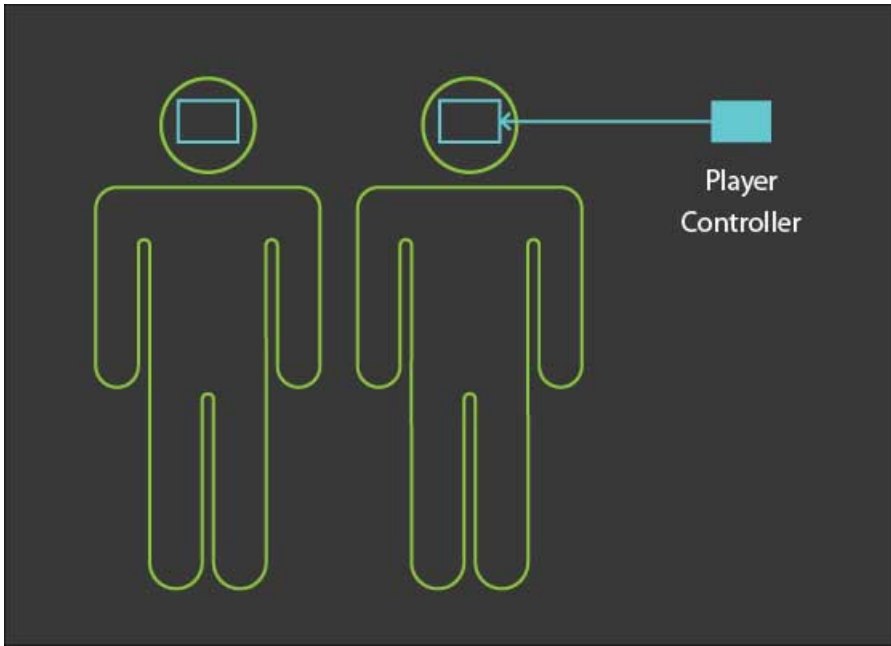

See Also

Unity Input Settings

FPS Characters

Overview

In NeoFPS, a first person character is split into two parts: the character and a controller. If a character dies then the controller persists and can be attached to a new character. Controllers can also be persisted between scenes if the game design requires.
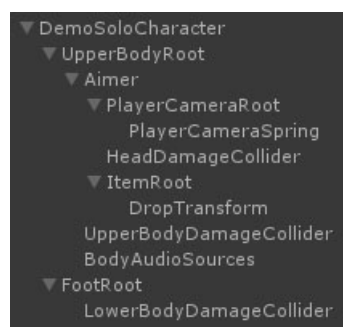

Characters

An FPS character ties together all the systems of NeoFPS into a single unit. Characters implement the ICharacter interface, which means you are free to define your own implementations of the character without being fixed to a specific base class.

A typical character hierarchy is as follows:

The root object is the most complex, containing the majority of components for the character:

- The ICharacter component
- Capsule Collider
- Rigidbody
- NeoCharacterController
- MotionController
- AimController
- Inventory
- Interaction handler
- Character audio handler
- Input behaviours

The player camera root is another object with a number of components:

- First person camera
- Additive transform handler and effects

There are also a number of damage handlers in the hierarchy for detecing incoming damage. These can be very simple, or more complex such as hit boxes on an animated character skeleton.

Controller

The FPS controller is persistent and can be attached to one character at any time. This system helps decouple the characters' mechanical implementation from their behaviour. In a multiplayer game there would be multiple player controllers or there could be a mix of player and AI controllers if bots were required. It also helps data such as stats persist between character deaths.

See Also

FpsSoloCharacter

FpsSoloPlayerController

Spawning

Overview

The NeoFPS player character should not be added to your scenes directly. Instead is uses a spawner system that allows you to place a number of spawn points around your map. This allows for greater flexibility in game mechanics such as:

- Spawn at one of a number of random locations
- Respawn on death without resetting the scenes
- Activate and deactivate spawn points as you move through a level so that respawning drops you closer to where you died
- Add a UI menu to choose a character on start
- Use multiple spawn points in case your preferred spawn is blocked

To make getting started easy, the **SimpleSpawnerAndGameMode** prefab contains a spawn point and a simple game mode that handles player spawning on start, as well as respawn on death. You can find this prefab at: *Assets\NeoFPS\Samples\SinglePlayer\Prefabs\SimpleSpawnerAndGameMode.prefab*

Spawn Points
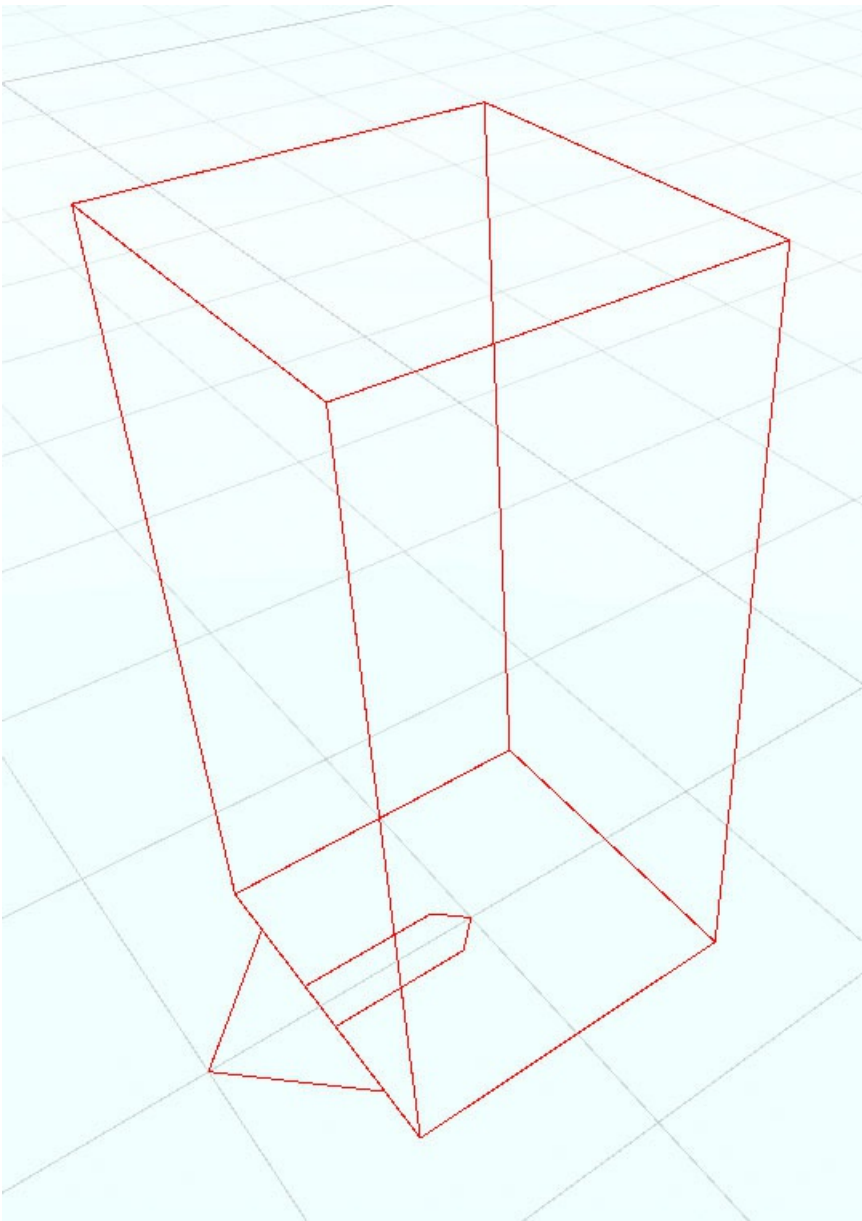
Spawn points are objects that are placed in the scene and positioned as required. By default, they will register themselves with the spawn manager as soon as the object is enabled, and unregister themselves as soon as the object is disabled.

You can also use an OrderedSpawnPointGroup behaviour to enforce a specific order to the registration of your spawn points. This

is useful when using the **Round-Robin** spawn mode which iterates through the spawn points, spawning in order, or the **First Valid** spawn mode which will iterated through the points in order and spawn at the first one that isn't blocked by an overlapping physics object. To control which is the active spawn mode, you can add a SpawnManager behaviour to an object in your scene.

Prototype Character

The NeoFPS samples also include a spawnerless player character that can be placed directly in your scenes. If the character dies then the scene is reloaded. This character is intended for rapid prototyping, and not for final production games.

You can find the prototype character prefab at the following location:
Assets\NeoFPS\Samples\SinglePlayer\Prefabs\PrototypeSpawnerlessCharacter.prefab

---

See Also

FPS Characters

Game Modes

Stamina

Overview

The stamina system in NeoFPS allows you to model various fatigue effects on your player character. Fatigue can be used to alter movement speed, to drive weapons' procedural animation (breathing), and to set thresholds where exhaustion sets in.

Movement & Stamina

The stamina system implementation provided can act as a motion graph data override. This means that the motion data used to specify movement speed will be overridden by values that the stamina system specifies. You can use animation curves on the StaminaSystem behaviour to blend between difference speeds as the character gets tired.

Animation

The current stamina level is used to specify a breathing rate and strength for the character. NeoFPS uses a number of procedural animation effects for animating weapons, including the BreathingEffect. This applies position and rotation animation with each breath, making aiming harder as the character gets tired.

Modifying Stamina

There are a number of example ways to modify the character's stamina provided out of the box. The DrainStamina and ModifyStamina motion graph behaviours allow you to modify the stamina based on the current state of the Motion Graph. The FirearmAimFatigue monobehaviour is attached to modular firearms and applies a stamina drain whilst aiming down sights.

The stamina system also has a setting to recharge stamina over time. An example stamina implementation would be to give both the character's walking state and the firearm aiming a stamina drains that are each smaller than the stamina system's refresh rate, but are larger when combined. This would mean that the player could move or aim without getting tired, but doing both together would cause them to slowly fatigue.

You can add your own systems to modify the character's stamina via a simple API. You can directly alter the stamina and max stamina values on the system by using the following methods:

```
void IStaminaSystem.SetStamina(float amount, bool normalised = false);
void IStaminaSystem.IncrementStamina(float amount, bool isFactor = false);
void IStaminaSystem.DecrementStamina(float amount, bool isFactor = false);
```

The `normalised` and `isFactor` parameters are used to specify if the value you are applying is directly to the stamina value (false), or as a factor of stamina divided by max stamina (true).

You can also add stamina drain delegates to the system that allow you to modify the stamina over time based on any number of outside influences. You do this using the following methods:

```
void IStaminaSystem.AddStaminaDrain(StaminaDrainDelegate drain);
void IStaminaSystem.RemoveStaminaDrain(StaminaDrainDelegate drain);
```

Where the StaminaDrainDelegate is defined as:

```
public delegate float StaminaDrainDelegate(IStaminaSystem system, float modifiedStamina);
```

Here the parameters are the stamina system itself, and the current stamina (this might not match the stamina value on the system as this stamina drain delegate might be one of many, and the altered stamina is passed to each delegate in sequence before it is applied to the system). The delegate returns the amount to reduce the stamina by this frame.

See Also

The Motion Graph

Modular Firearms

Additive Transforms And Effects

FpsSoloPlayerController MonoBehaviour

Overview

The FpsPrototypePlayerController is a version of the player character controller which can be added to the character object directly to bypass the spawning system.

If the player is killed then the scene will be reloaded.

This controller is intended purely for prototyping and testing. If more complex behaviour is required such as respawning or loading from saves, then use the spawn system as in the demo scenes.

Inspector

Properties

The FpsSoloPlayerController has no properties exposed in the inspector.
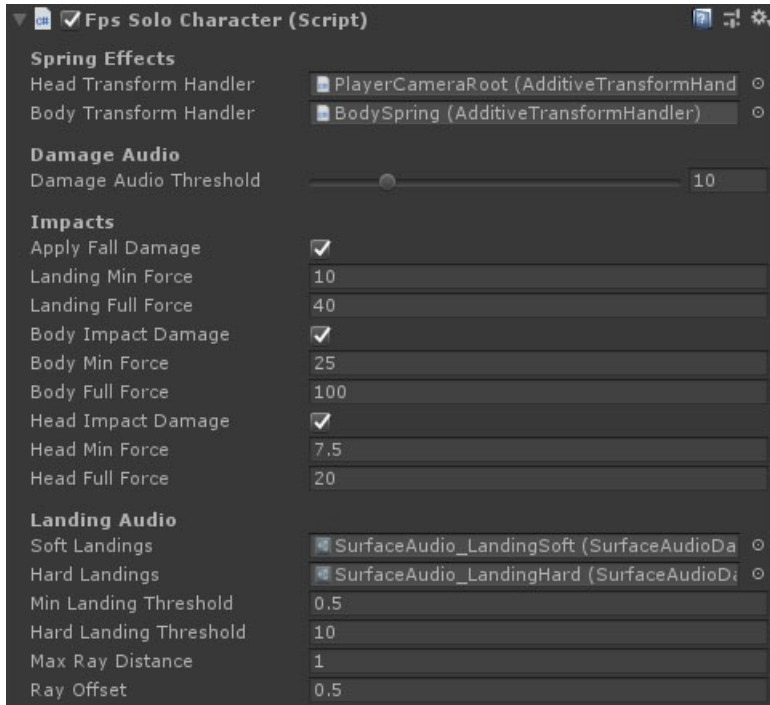

See Also

FPS Characters

FpsSoloCharacter MonoBehaviour

Overview

FpsSoloCharacter is the base character behaviour for single player games. Handles impact and landing damage and audio, and specifies the transform handlers for reacting events.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Head Transform Handler	AdditiveTransformHandler	The additive transform handler attached to the head hierarchy of this character (used for things like weapon recoil and impacts). The spring effects here affec the camera but **not** the carried items.
Body Transform Handler	AdditiveTransformHandler	The additive transform handler attached to the body hierarchy of this character (used for things like weapon recoil and impacts). The spring effects here affec the camera **and** the carried items.
Damage Audio Threshold	Float	The amount of damage to take in a single hit before playing a character damage audio clip.
Apply Fall Damage	Boolean	Should the character be subject to damage from landing impacts (impacts where the character capsule is hit in the bottom hemisphere).
Landing Min Force	Float	The minimum landing impact magnitude before any damage is applied.
Landing Full Force	Float	The landing impact magnitude where a full 100 damage will be applied.
Body Impact Damage	Boolean	Should the character be subject to damage from body impacts (impacts where the character capsule is hit in the central cylinder).

| NAME	TYPE	DESCRIPTION
Body Min Force	Float	The minimum body impact magnitude before any damage is applied.
Body Full Force	Float	The body impact magnitude where a full 100 damage will be applied.
Head Impact Damage	Boolean	Should the character be subject to damage from head impacts (impacts where the character capsule is hit in the top hemisphere).
Head Min Force	Float	The minimum head impact magnitude before any damage is applied.
Head Full Force	Float	The head impact magnitude where a full 100 damage will be applied.
Soft Landings	SurfaceAudioData	Surface audio library used to trigger the correct sound when the character lands below the "hard landing" threshold.
Hard Landings	SurfaceAudioData	Surface audio library used to trigger the correct sound when the character makes a heavy landing.
Min Landing Threshold	Float	The magnitude of the landing force below which no landing sound will be played.
Hard Landing Threshold	Float	The magnitude of the landing force above which to play a hard landing sound.
Max Ray Distance	Float	The maximum downward ray length for a ground test.
Ray Offset	Float	The vertical offset from the absolute bottom of the character to start the ground test raycast.

See Also

AdditiveTransformHandler

SurfaceAudioData

FpsSoloPlayerController MonoBehaviour

Overview

The FpsSoloPlayerController is the base player character controller for single player games.

Inspector
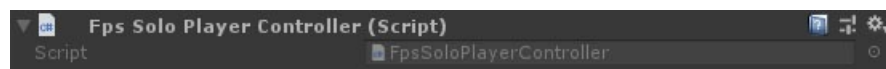
Properties

The FpsSoloPlayerController has no properties exposed in the inspector.
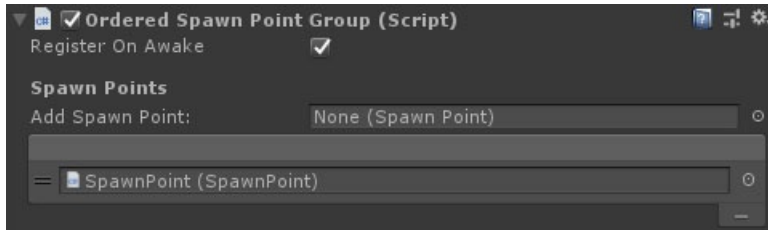
See Also

FPS Characters

OrderedSpawnPointGroup MonoBehaviour

Overview

The OrderedSpawnPointGroup behaviour is used to enforce a spawn order for multiple spawn points. The order that Unity calls Start() on multiple objects can be unpredictable, so this behaviour allows you to add the spawn points to a list and rearrange the order. This works best with the SpawnManager set to use the "Round Robin" spawn mode.

Inspector

Properties

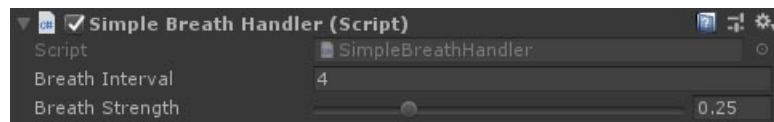| NAME	TYPE	DESCRIPTION
Register On Awake	Dropdown	Should the spawn points be registered with the manager as soon as this object is awoken.
Spawn Points	SpawnPoint Array	The spawn points to register in order.

See Also

SpawnManager

SpawnPoint

SimpleBreathHandler MonoBehaviour

Overview

The SimpleBreathHandler is attached to a character to give them a constant breathing rate. This is used to drive the BreathingEffect procedural animation on weapons. You can alternatively use a StaminaSystem behaviour for a more complex breathing system that adapts to character fatigue.

Inspector

Properties

| NAME	TYPE	DESCRIPTION
Breath Interval	Float	The time in seconds between breaths.
Breath Strength	Float	The strength of the character's breathing (0 = non-existant, 1 = heaving/panting).

See Also

BreathingEffect

StaminaSystem

SoloPlayerCharacterEventWatcher MonoBehaviour

Overview

The SoloPlayerCharacterEventWatcher behvaiour attaches to an event that is fired when the player character changes. It then passes that change on to its subscribers. An example use is in the player HUD, where it is used to bind various HUD elements such as health and inventory to the player character.
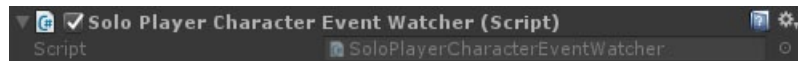
Inspector

Properties

The SoloPlayerCharacterEventWatcher behaviour has no properties exposed in the inspector.

See Also

AdditiveTransformHandler

SurfaceAudioData

SpawnManager MonoBehaviour

Overview

The spawn manager is used to spawn characters. Spawn points register with the manager on start and deregister when disabled or destroyed. You do not need this behaviour in your scene, but adding it will allow you to change the spawn behaviour by setting the behaviour properties.

Inspector

Properties

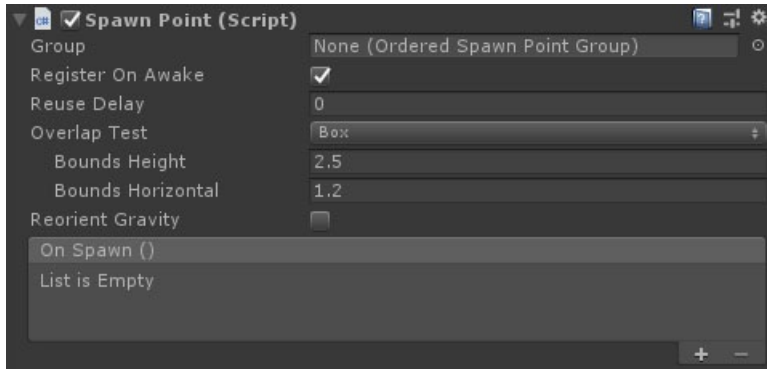| NAME	TYPE	DESCRIPTION
Spawn Mode	Dropdown	How the next spawn point is chosen: **RoundRobin** picks each spawn point in sequence. **FirstValid** starting at the first registered spawwn point and iterating until a valid one is found **Random** picked at random until a valid point is found.

See Also

[SpawnPoint][1]

SpawnPoint MonoBehaviour

Overview

The SpawnPoint behaviour specifies where characters are spawned. It will register with the SpawnManager when active and enabled, and deregister when deactivated or destroyed. It is recommended to have more than one in the scene in case one or more is blocked.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Register On Awake	Boolean	Should the spawn point be registered with the SpawnManager immediately on awake? This option will be disabled if the spawn point is part of a OrderedSpawnPointGroup.
Reuse Delay	Float	How long before the spawn point can be used again.
Overlap Test	Dropdown	The collider volume type for checking if the spawn point is clear or overlapped by another object. Options are **Box**, **Capsule**, **None**.
Bounds Height	Float	The vertical height of the bounding volume for overlap checks.
Bounds Horizontal	Float	The horizontal dimension of the bounding volume for overlap checks.
Reorient Gravity	Boolean	Should the character's gravity be reoriented to match the spawn point. If the spawn is tilted on one side, this will make the character's down direction equal to the spawn point's.
On Spawn	Event	A UnityEvent fired when a character is spawned at this point. Allows for simple triggering of spawn audio and visual effects.

See Also

SpawnManager

OrderedSpawnPointGroup

StaminaSystem MonoBehaviour

Overview

The StaminaSystem behaviour is attached to a character and used to model various fatigue and exertion effects.

Stamina regenerates at a preset rate and can be drained or modified via a simple API. NeoFPS includes examples such as the FirearmAimFatigue behaviour which drains stamina while aiming down sights, or the DrainStaminaBehaviour and ModifyStaminaBehaviour motion graph behaviours which can be used to drain stamina over time in a specific motion state such as sprinting, or on entering a state such as jumping.

Stamina can be used to drive the BreathingEffect procedural animation on weapons. You can specify the breathing rate and strength based on fatigue using animation curves for each.

Stamina can also be used to control movement speed, slowing the character as they get tired. There is an optional exhaustion setting which can be tied to the motion graph to prevent the character from sprinting once they reach the exhaustion threshold, and re-enable sprinting once stamina has recovered to a certain point.

Inspector

Properties

Stamina

NAME	TYPE	DESCRIPTION
Stamina	Float	The current stamina of the character. This acts as the starting stamina and changes at runtime.
Max Stamina	Float	The maximum stamina of the character.
Stamina Refresh Rate	Float	The rate that stamina increases over time when no drains are applied.

Movement Speed

NAME	TYPE	DESCRIPTION
Affect Movement Speed	Boolean	Should the stamina system modify movement speed based on current stamina. The other settings will be hidden if this is false.
Min Walk Multiplier	Float	A multiplier applied to the walking speed at minimum stamina.
Min Sprint Multiplier	Float	A multiplier applied to the sprinting speed at minimum stamina.
Min Crouch Multiplier	Float	A multiplier applied to the crouching speed at minimum stamina.
Move Speed Curve	Unity AnimationCurve	A curve that defines the character speed based on stamina. The X axis is the normalised stamina (stamina / max), while the Y axis is the min to max lerp value (0 = min, 1 = max).
Walk Speed Data	String	The name of the motion data property on the motion graph that defines walk speed.
Aim Walk Speed Data	String	The name of the motion data property on the motion graph that defines walk speed when aiming.
Sprint Speed Data	String	The name of the motion data property on the motion graph that defines sprint speed.
Aim Sprint Speed Data	String	The name of the motion data property on the motion graph that defines sprint speed when aiming.
Crouch Speed Data	String	The name of the motion data property on the motion graph that defines crouch movement speed.
Aim Crouch Speed Data	String	The name of the motion data property on the motion graph that defines crouch movement speed when aiming.

Breathing

NAME	TYPE	DESCRIPTION
Breathe Slow Interval	Float	The time in seconds between breaths (when breathing slow).

NAME	TYPE	DESCRIPTION
Breathe Fast Interval	Float	The time in seconds between breaths (when breathing fast).
Breathing Rate Curve	Unity AnimationCurve	A curve that defines the breathing rate based on stamina. The X-axis is the normalised stamina (stamina / max stamina), and the Y-Axis is a lerp between slow and fast breathing rate (0 = slow, 1 = fast).
Breathing Strength Curve	Unity AnimationCurve	A curve that defines the breathing strength based on stamina. The X-axis is the normalised stamina (stamina / max stamina), and the Y-Axis is the strength of the character's breathing (0 = non-existant, 1 = heaving/panting).

Exhaustion

NAME	TYPE	DESCRIPTION
Use Exhaustion	Bool	Should the character suffer an exhaustion effect on hitting a specific stamina threshold. The other properties will be hidden if this is false.
Exhaustion Threshold	Float	The stamina level below which the character will become exhausted.
Recover Threshold	Float	The character will stop being exhausted once their stamina has recovered above this value.
Exhausted Motion Parameter	String	The name of the switch motion graph parameter that the graph uses as a condition for preventing sprinting.
Sprint Motion Parameter	String	The name of the switch motion graph parameter that the character input handler sets to tell the motion graph to start sprinting.
On Exhausted	UnityEvent	An event which is fired when the character hits the exhaustion threshold.
On Recovered	UnityEvent	An event which is fired when the character recovers from exhaustion.

See Also

BreathingEffect

Motion Graph Parameters And Data

Modular Firearms

Unity AnimationCurve

First Person Camera

Overview

The first person camera in NeoFPS is a companion component for the Unity camera and adds a number of useful features for first person shooters.

Field of View

The NeoFPS camera adds additional functionality for controlling the field of view.

Firstly, it allows a horizontal field of view to be specified instead of the default vertical field of view. This is considered a standard in first person shooters, and a number of players will have their preferred FoV settings that they apply in and FPS games they play.

Secondly, it allows the field of view to be modified and animated using multipliers for zoom effects such as aiming down sights. You can modify and reset the FoV in your scripts using the following methods:

```
public void FirstPersonCamera.SetFov (float targetFovMult, float aimTime);

public void FirstPersonCamera.ResetFov (float aimTime);
```

The `SetFov` method takes an FoV multiplier that multiplies based on the standard field of view settings, along with an aim time that controls the duration of the animation from the existing FoV to the new FoV. The `ResetFov` method takes the same aim time parameter which controls how long it takes to return to the standard field of view settings.

Camera Offset

The camera offset is a position and rotation offset that can be applied and removed to move and animate the camera within the hierarchy. An example use is the modular firearms' HeadMoveAimer which actually moves the head down to the weapon instead of the weapon up to the camera. This allows a lean angle to be added and for the camera rotation to be modified making for an interesting aim effect. You can modify and reset the offset in scripts using the following methods:

```
public void FirstPersonCamera.SetOffset (Vector3 posOffset, Quaternion rotOffset, float aimTime);

public void FirstPersonCamera.ResetOffset (float aimTime);
```

These methods take offset parameters, along with an aim time parameter that controls how long it takes to reach the offset and return.

Events

The first person camera also contains the following event:

```
public static event FirstPersonCamera.UnityAction<FirstPersonCamera> onCurrentCameraChanged;
```

Subscribing to this event will notify when the camera changes (when it is being looked through or not). This can be used for things such as enabling a scene or spectator camera if the player dies.

See Also

FirstPersonCamera Behaviour

additive-Effects

Unity Camera

Aim Controllers

Overview

Aim controllers take input and control a character's look direction before additive effects are applied.

Turn Rate

The aimer turn rate can be altered using the `turnRateMultiplier` property. This is used when setting the first person camera field of view, for example when aiming down sights.

Setting Yaw And Pitch

You can modify the aim controller's rotation from scripts using the following methods:

```
void AddYaw (float rotation);

void AddPitch (float rotation);

void AddRotation (float y, float p);
```

Aim Constraints

The aim controller can be constrained to a specific range of rotations on either the vertical or horizontal axes or both. This is used in situations such as climbing ladders or using a mounted turret, where it would be unnatural for the character to be able to turn a full 360 degrees.

If the aim controller rotation is outside of the constraints when they are set, then it uses a damped rotation to smoothly turn into the constraints.

Since the fps character is able to change their up direction, the yaw constraint must use a direction vector for the constraints center. This means that as the character tilts, they can still constrain to the direction. Camera pitch is relative to the character.

In order to set the aim controller constraints from a script you can use the following methods on the aim controller component:

```
void SetYawConstraints(Vector3 forward, float range);

void SetPitchConstraints(float minimum, float maximum);

void ResetYawConstraints();

void ResetPitchConstraints();
```
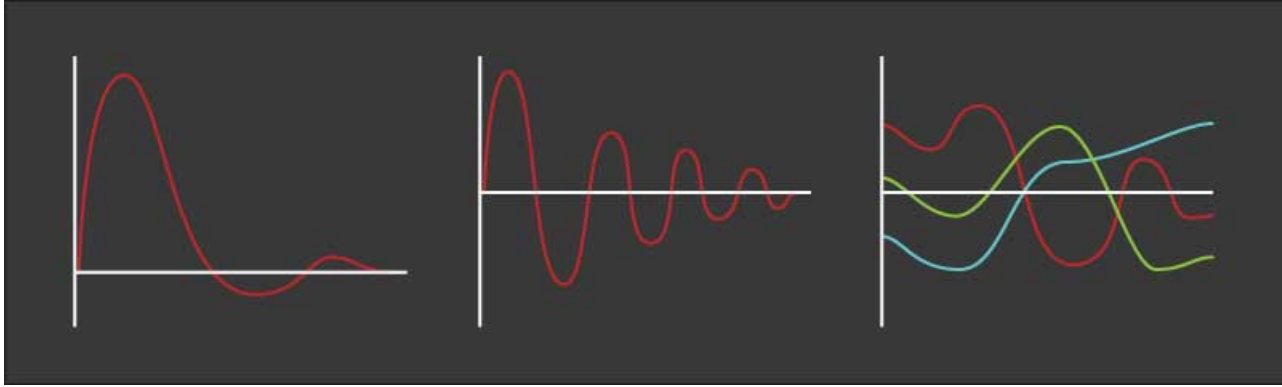
See Also

FirstPersonCamera Behaviour

additive-Effects

Unity Camera

Additive Transforms and Effects

Overview

NeoFPS uses a system called additive transforms to manage camera and object transform effects such as recoil and shake. In order to make use of additive transforms, the AdditiveTransformHandler is added to an object and then various additive effects are attached to it. It is very simple to write custom additive effects, but a number of useful effects are included with NeoFPS including:

NAME	DESCRIPTION	USE
AdditiveKicker	A simple knock and spring return. Triggered by the CharacterEventKickTrigger and ImpactHandlerKickTrigger.	Any
AdditiveJiggle	A simple rotation around the z-axis with a bouncy spring return.	Any
BodyLean	Lean to either side	Body
BodyTilt	Tilt the body in an arbitrary direction	Body
BreathingEffect	Applies a breathing motion to the selected weapon.	Weapon
CameraShake	Apply continuous and one-off shakes to the camera.	Camera
CharacterMovementSway	A position and rotation offset applied to the weapon based on character velocity.	Weapon
CharacterRecoilEffect	Applies recoil spring animation to the character. Driven by a BetterSpringRecoilHandler firearm module.	Body
HeadBob	**(Deprecated)** Position bob using curves	Camera
HeadBobV2	An enhanced head bob with more control over animation curves, aim compensation, and in-game strength settings.	Body
HeadDuck	A simple lowering of the head used for charged jumps	Camera
FirearmRecoilEffect	Applies recoil spring animation to a modular firearm. Driven by a BetterSpringRecoilHandler firearm module.	Weapon
OverShoulder	Rotate the head to look backwards, while leaving the weapon and body pointing forwards.	Camera
PeekVertical	Peek over or under obstacles by moving the head up or down.	Camera

NAME	DESCRIPTION	USE
PositionBob	Position bob using curves. Syncs and blends bob between head and weapon based on player settings by sharing bob settings with a PositionBobData scriptable object.	Camera And Weapon
RotationBob	Rotation bob using curves.	Weapon
TransformMatcher	Matches the offset of one transform relative to another. Used to blend keyframe animation in with procedural effects.	Camera
WeaponAimAmplifier	Multiplies the movement of the camera back onto the held weapon to add weight and momentum.	Weapon
WeaponBob	**(Deprecated)** A bob effect for weapons, similar to the head bob but with rotation too.	Weapon
WeaponMomentumSway	Similar to the WeaponAimAmplifier, but changes the position instead of the rotation.	Weapon

The AdditiveTransformHandler will check each of the effects every tick (the update frequency can be set in the component properties) and then layer them on top of each other in order to create a combined offset. Translation and rotation are handled separately, meaning that the rotation offset of an effect will not alter the position offset of another effect further along the chain. This makes the outcome more predictable when a number of effects are applied at once.

You can also specify a pivot point for the resulting transform if desired.

Footstep driven animation such as head or weapon bob is driven through a step tracking system attached to the character's motion controller. This ensures that any animations like this sync up, are only active while in the correct movement state, and are correctly driven by the character's velocity. The character's stride length (and therefore bob speed while moving) is controlled by adding a TrackSteps motion graph behaviour to the relevant states or sub-graphs in your character's motion graph.

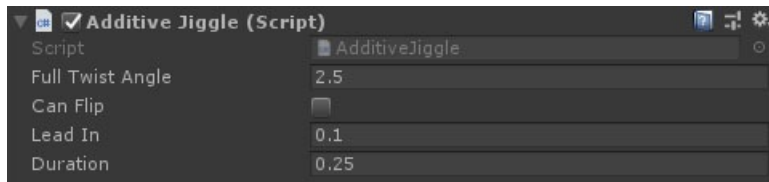## See Also

AdditiveTransformHandler

FirstPersonCamera

AdditiveJiggle MonoBehaviour

Overview

The additive jiggle is used to knock the object's rotation (roll) and then spring back.

Inspector

Properties

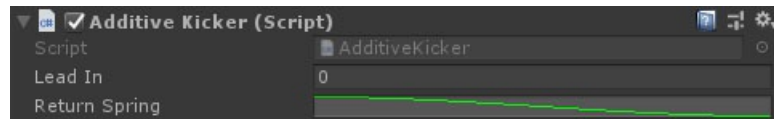| NAME	TYPE	DESCRIPTION
Full Twist Angle	Float	The angle (either side of the axis) of a full strength jiggle.
Can Flip	Boolean	If true, the jiggle could rotate in either direction.
Lead In	Float	The time taken to ease into the jiggle.
Duration	Float	The time taken for the jiggle spring to ease out.

See Also

Additive Transforms

AdditiveKicker

AdditiveKicker MonoBehaviour

Overview

The additive kicker is used to knock the camera or object and then spring back. Pair it with CharacterImpactKicker and KickerImpactHandler behaviours in order to consume character impact events.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Lead In	Float	The time taken to ease into the kick.
Return Spring	AnimationCurve	The return spring is an animation curve that dictates how the kicker returns from the kick angle/position (1 on the y-axis) to its original state (0 on the y axis).

See Also

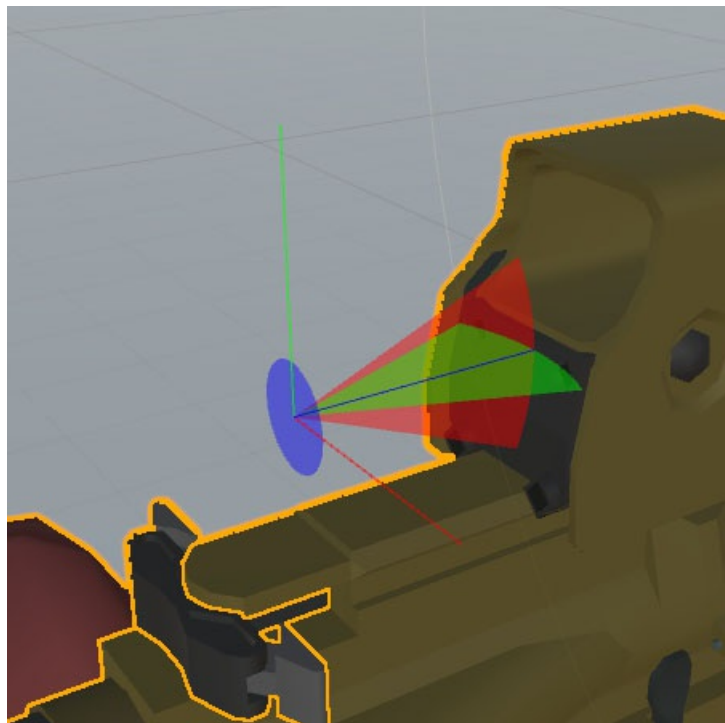Additive Transforms

CharacterImpactKicker

KickerImpactHandler

Unity AnimationCurve

AdditiveTransformHandler MonoBehaviour
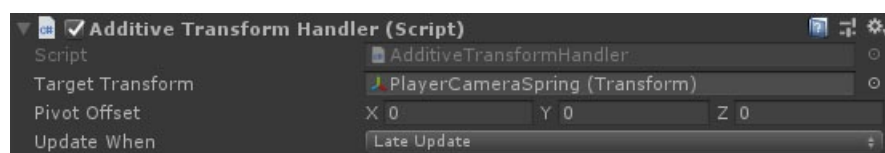
Overview

The AdditiveTransformHandler behaviour sums together all the additive transform effects applied to it and uses them to transform an object such as the camera.

In The Scene View

With a gameobject selected that uses an AdditiveTransformHandler component, the guide handle will be visible in the scene view. This represents the point the target transform will rotate around.

Inspector

Properties

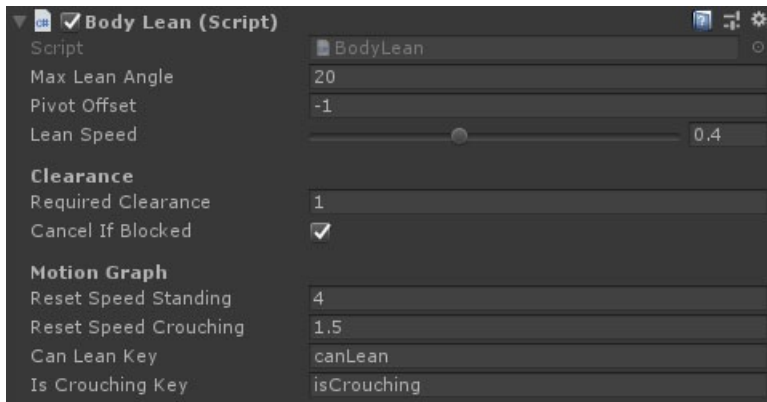| NAME	TYPE	DESCRIPTION
Target Transform	Transform	The transform that the handler affects.
Pivot Offset	Vector3	The offset from the transform origin for the pivot point to rotate around.
Update When	Dropdown	When should the additive transform effects be calculated and applied. Options are: **Update**, **LateUpdate**, **FixedUpdate**, **FixedAndLerp**, **FixedAndLateLerp** (The "Lerp" options calculate in fixed update and then interpolate between results during Update/LateUpdate for smooth results).

See Also

Additive Transforms

WeaponBob

Unity AnimationCurve

BodyLean MonoBehaviour

Overview

The BodyLean behaviour is used to lean the character body to either side.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Max Lean Angle	Float	The angle to lean to either side when lean power is set to 1 or -1.
Pivot Offset	Float	The downward offset for the pivot point from the transform origin.
Lean Speed	Float	The speed of transitioning from 1 lean value to another. 1 is nearly instant.
Weapon Tilt	Float	How much of the lean rotation is reflected in the weapon.
Head Counter Tilt	Float	A counter rotation of the head compared to the weapon.
Required Clearance	Float	The distance to check from the center line for clearance space to lean. The lean amount will be capped based on the result.
Cancel If Blocked	Boolean	If there is no clearance space then return the lean amount to 0 until manually re-applied.
Reset Speed Standing	Float	The maximum speed the character can travel before the lean is cancelled. (0 = no max speed).
Reset Speed Crouching	Float	The maximum speed the character can travel before the lean is cancelled. (0 = no max speed).
Can Lean Key	String	The key to a motion graph switch parameter that dictates if the character can lean or not.
Is Crouching Key	String	The key to a motion graph switch parameter that dictates if the character is crouching or standing.

See Also
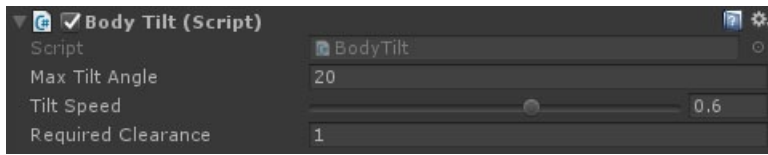
Additive Transforms

Motion Graph Parameters And Data

HeadBob MonoBehaviour

Overview

The BodyTilt behaviour is used to lean the character body based on an input vector.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Max Tilt Angle	Float	The maximum angle the character can tilt. Angles above this will be capped.
Tilt Speed	Float	The speed of transitioning from 1 tilt value to another. 1 is nearly instant.
Required Clearance	Float	The distance to check from the center line for clearance space to lean. The lean amount will be capped based on the result.

See Also

[Additive Transforms](#)

[Motion Graph Parameters And Data](#)

BoxShakeZone MonoBehaviour

Overview

The BoxShakeZone behaviour defines a 3D box area with a constant shake value used by the CameraShake additive transform effect.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Strength	Float	The strength of the shake.
Size	Vector	The dimensions of the box along each axis (centered on position).
Falloff Distance	Float	The distance outside of the box that the strength falls off to zero.

See Also
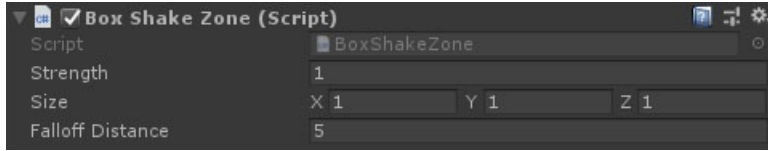
Additive Transforms

CameraShake

BoxShakeZone2D MonoBehaviour

Overview

The BoxShakeZone2D behaviour defines a 2D box area (on the x-y plane) with a constant shake value used by the CameraShake additive transform effect.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Strength	Float	The strength of the shake.
Size	Vector	The dimensions of the box along each axis (centered on position).
Falloff Distance	Float	The distance outside of the box that the strength falls off to zero.

See Also
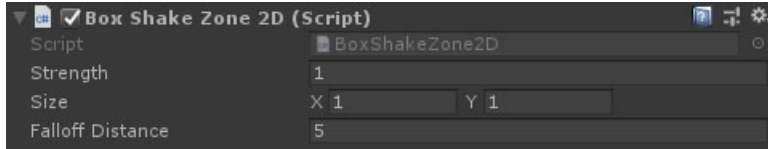
Additive Transforms

CameraShake

BreathingEffect MonoBehaviour

Overview

The BreathingEffect behaviour applies a procedural breathing animation based on a character's breath handler.

Example breath handlers include the SimpleBreathHandler which has a constant breathing rate, or the StaminaSystem which modifies the breathing rate and strength based on fatigue.

Inspector
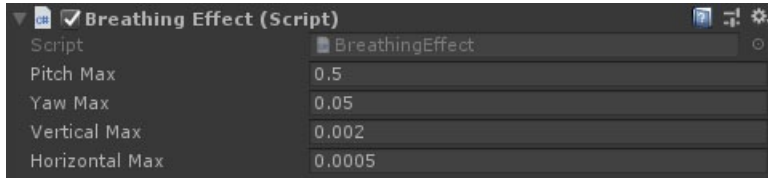
Properties

NAME	TYPE	DESCRIPTION
Pitch Max	Float	The maximum pitch rotation at breathing strength 1.
Yaw Max	Float	The maximum yaw rotation at breathing strength 1.
Vertical Max	Float	The maximum vertical position offset at breathing strength 1.
Horizontal Max	Float	The maximum horizontal position offset at breathing strength 1.

See Also

Additive Transforms

SimpleBreathHandler

StaminaSystem

CameraShake MonoBehaviour

Overview

The CameraShake behaviour is used to add constant and one-shot shake effects to the first person camera.

Constant shake effects can be applied using shake zones such as the BoxShakeZone, BoxShakeZone2D, SphereShakeZone and CircleShakeZone. You can also set the global shake value directly with scripts using the following code:
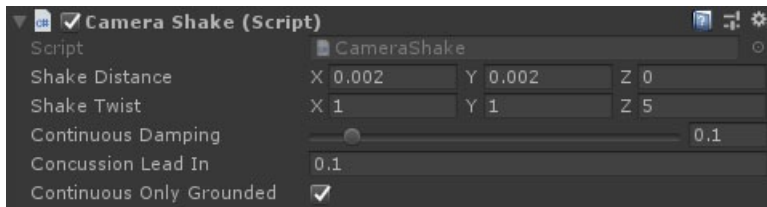
```
ShakeHandler.globalShake = shakeAmount;
```

Where `shakeAmount` is a float value between 0 (no shake) and 1 (maximum shake).

You can apply a one-shot shake using the following method:

```
public static void ShakeHandler.Shake(Vector3 position, float innerRadius, float falloffDistance, float
strength, float duration, bool requiresGrounding = false)
```

Inspector

Properties

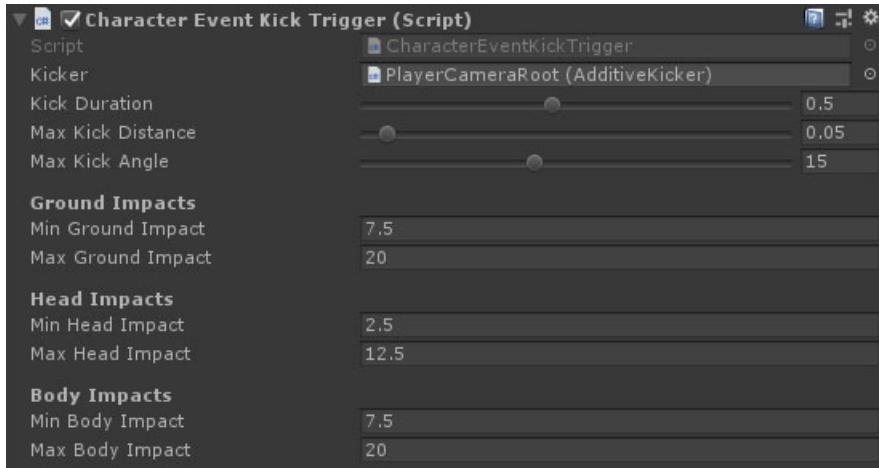| NAME	TYPE	DESCRIPTION
Shake Distance	Vector	The distance the camera can move either side of the origin on each axis at a shake strength of 1.
Shake Twist	Vector	The max rotation (in each direction) for a shake strength of 1.
Continuous Damping	Float	Damping smooths the blend between different continuous shake strengths.
Concussion Lead In	Float	The time it takes for the shake to go from 0 to 1.
Continuous Only Grounded	Boolean	Should continuous shake only be applied while the character this is attached to is grounded (if there is a character).

See Also

Additive Transforms

CharacterEventKickTrigger MonoBehaviour

Overview

The CharacterEventKickTrigger consumes character impact events such as landings and bullet hits, and uses them to drive an additive kicker effect.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Kicker	AdditiveKicker	The additive kicker used to react to the force.
Kick Duration	Float	The time taken to recover from the kick.
Max Kick Distance	Float	The downward position kick distance at max impulse.
Max Kick Angle	Float	The forward kick angle at max impulse.
Min Ground Impact	Float	A ground impact impulse with magnitude lower than this will be ignored.
Max Ground Impact	Float	The ground impact impulse magnitude that gives the maximum kick.
Min Head Impact	Float	A head impact impulse with magnitude lower than this will be ignored.
Max Head Impact	Float	The head impact impulse magnitude that gives the maximum kick.
Min Body Impact	Float	A body impact impulse with magnitude lower than this will be ignored.
Max Body Impact	Float	The body impact impulse magnitude that gives the maximum kick.

See Also

Additive Transforms

AdditiveKicker

CharacterRecoilEffect MonoBehaviour

Overview

The CharacterRecoilEffect behaviour is a part of the modular firearm recoil system and applies the character (body and head) based component of the recoil animation. This is paired with the FirearmRecoilEffect behaviour to apply the firearm component of the animation, and controlled by the BetterSpringRecoil firearm module.

Inspector

Properties

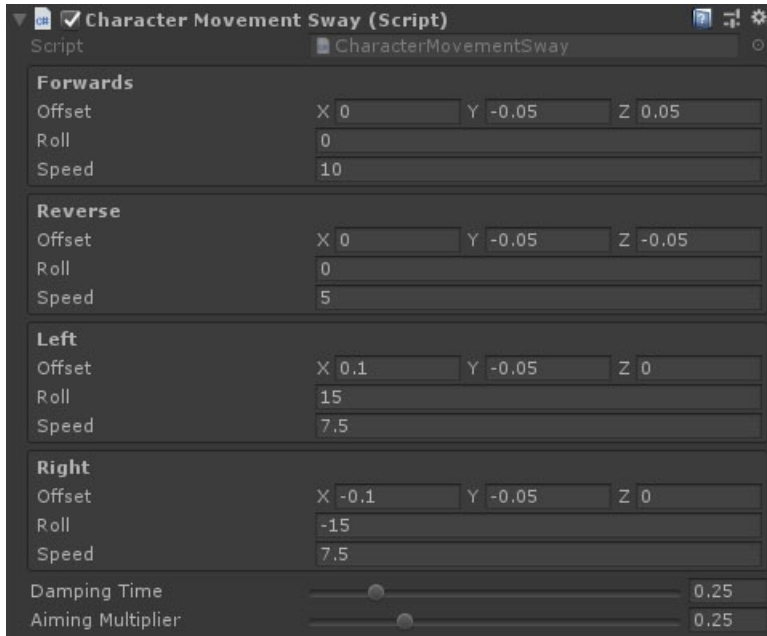| NAME	TYPE	DESCRIPTION

See Also

Additive Transforms

FirearmRecoilEffect

BetterSpringRecoil

CharacterMovementSway MonoBehaviour

Overview

The CharacterMovementSway behaviour adds an offset to the held weapon based on the character's velocity. This can help exaggerate movement and make the weapon feel more dynamic.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Damping Time	Float	Approximately the time it will take to reach the target sway. A smaller value will reach the target faster.
Aiming Multiplier	Float	A multiplier applied to the offset when aiming down sights. This helps make the weapon more controlled

There are also 4 sway limit sections - one for each direction. When moving diagonally, the sway will blend between the closest 2. Each of these section has the following properties:

NAME	TYPE	DESCRIPTION
Offset	Vector	The position offset to apply to the weapon when at or above the target speed for this direction.
Roll	Float	The rotation around the weapon's forward axis when at or above the target speed for this direction.
Speed	Float	The speed in this direction at or above which the full offset and roll will be applied.

See Also

Additive Transforms

CircleShakeZone MonoBehaviour

Overview

The CircleShakeZone behaviour defines a 2D circular area (on the x-y plane) with a constant shake value used by the CameraShake additive transform effect.

Inspector
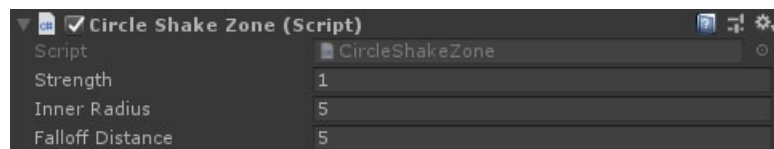
Properties

NAME	TYPE	DESCRIPTION
Strength	Float	The strength of the shake.
Inner Radius	Float	The inner radius of the circle. Inside this, the strength is constant.
Falloff Distance	Float	The distance outside of the box that the strength falls off to zero.

See Also

Additive Transforms

CameraShake

CutsceneCamera MonoBehaviour

Overview

The CutsceneCamera behaviour is used to switch camera from a first person character to an externally controlled camera. It handles disabling the player character's input, and hiding the HUD. Once the camera is disabled again, control is restored and the HUD is re-enabled.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Can Skip	Boolean	Can the cutscene be skipped by holding the use button.
Skip Hold	Float	The amount of time the use button must be held to skip the cutscene.
On Skip	UnityEvent	An event fired when the cutscene is skipped.

See Also

Additive Transforms

FirearmRecoilEffect MonoBehaviour

Overview

The FirearmRecoilEffect behaviour is a part of the modular firearm recoil system and applies the character (body and head) based component of the recoil animation. This is paired with the CharacterRecoilEffect behaviour to apply the body and head components of the animation, and controlled by the BetterSpringRecoil firearm module.

Inspector
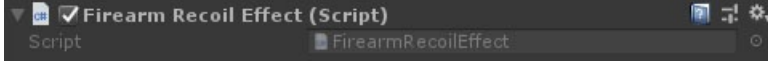
Properties

NAME	TYPE	DESCRIPTION

See Also

Additive Transforms

CharacterRecoilEffect

BetterSpringRecoil

FirstPersonCamera MonoBehaviour

Overview

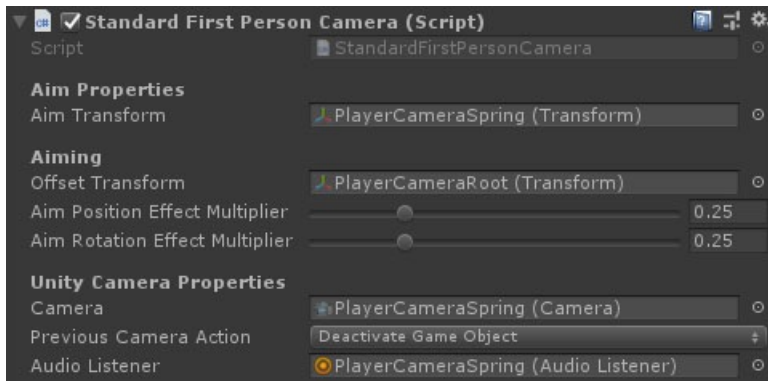The FirstPersonCamera MonoBehaviour is used with the standard Unity Camera to add useful features.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Camera	Camera	The main camera for the first person view.
Aim Transform	Transform	The transform to use for accurate shooting. If you add extra spring effects to the camera that don't affect the gun, you might want to set this to something higher up the hierarchy.
Previous Camera Action	Dropdown	What to do with the main camera in the scene? Use this to prevent wasted render cycles and multiple listeners. Options: **DeactivateGameObject**, **DisableComponent**, **DestroyGameObject**, **Ignore**.
Default FoV	Float	The default vertical field of view. This will also change the horizontal fov below.
Horizontal 16:9	Float	The default horizontal field of view for a 16:9 screen. This will also change the vertical fov above.
Offset Transform	Transform	The offset from standard upright position for moving the head. Used for aiming down sights, etc.
Aim Position Effect Multiplier	Float	The multiplier applied to additive spring effects while aiming.
Aim Rotation Effect Multiplier	Float	The multiplier applied to additive spring effects while aiming.

See Also
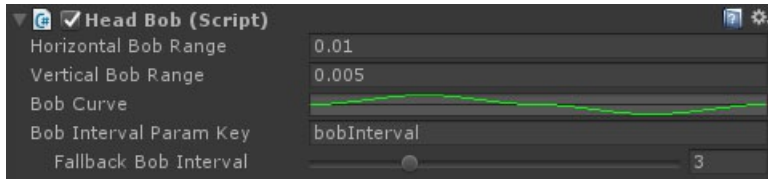
Additive Effects

Unity Camera

HeadBob MonoBehaviour (Deprecated)

Overview

The HeadBob behaviour is used to move the camera in sync with character movement. This behaviour has been deprecated and replaced with the PositionBob and RotationBob systems which sync weapon and head bobbing, and allow player settings to reduce head movement.

The head bob can tie into the moion graph and subscribe to a float parameter to allow the motion graph elements to control when the bob occurs and the bob rate. Without this, the bob occurs whenever the character is grounded, and the bob rate is based on movement speed.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Horizontal Bob Range	Float	The maximum position offset along the x-axis in either direction.
Vertical Bob Range	Float	The maximum position offset along the y-axis in either direction.
Bob Curve	AnimationCurve	The curve over one step cycle for the weapon bob.
Bob Interval Param Key	FloatParameter	The name of a float parameter on the character motion graph that sets the bob interval distance.
(Fallback) Bob Interval	Float	The distance travelled for one full bob cycle. If the parameter key above is set then this value will only be used if the parameter can not be found.

See Also

Additive Transforms

WeaponBob

Unity AnimationCurve
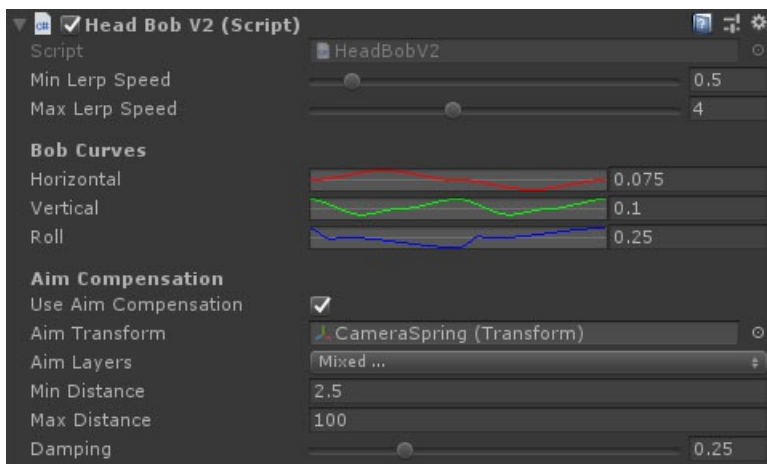
HeadBobV2 MonoBehaviour (Deprecated)

Overview

The HeadBobV2 behaviour is a new spring effect which uses the character step tracking system to sync complex head bob to the character. Alongside simple position bob it adds the following features:

- In-game strength settings via the gameplay options or quick-options popup
- Individual animation curves for horizontal offset, vertical offset and roll
- Aim compensation which adds a counter rotation to keep the crosshair fixed on objects as the head moves

Despite the name, the HeadBobV2 component is intended to be applied to the upper body spring so that the weapon moves with it.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Min Lerp Speed	Float	At or below this speed the bob will be scaled to zero.
Max Lerp Speed	Float	At or above this speed the bob will have its full effect.
Horizontal	AnimationCurve + Float	The maximum position offset along the horizontal axis in either direction. The horizontal axis of the animation curve goes from -1 to 1 where -1 is left foot, 0 is right foot, and 1 is back to left foot. The Y axis is a multiplier for the float value next to the animation curve.
Vertical	AnimationCurve	The maximum position offset along the vertical axis in either direction. The horizontal axis of the animation curve goes from -1 to 1 where -1 is left foot, 0 is right foot, and 1 is back to left foot. The Y axis is a multiplier for the float value next to the animation curve.
Roll	AnimationCurve	The maximum angle to roll the camera in either direction. The horizontal axis of the animation curve goes from -1 to 1 where -1 is left foot, 0 is right foot, and 1 is back to left foot. The Y axis is a multiplier for the float value next to the animation curve.
Use Aim Compensation	Boolean	Aim compensation involves rotating the camera so that the crosshair stays fixed on the same point in space.
Aim Transform	Transform	The transform to use for camera/crosshair casting.

NAME	TYPE	DESCRIPTION
Aim Layers	LayerMask	The layers to check against for aim depth.
Min Distance	Float	The minimum distance to compensate against. At very close distances, the bob will have to rotate larger angles to keep the crosshair fixed on target.
Max Distance	Float	The maximum distance to compensate against.
Damping	Float	A damping against the crosshair target. Prevents objects crossing the camera at close ranges from causing sudden shifts.

See Also

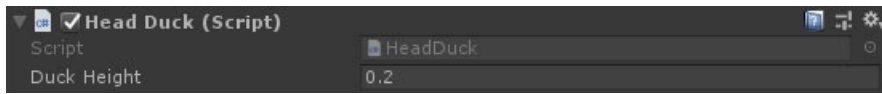Additive Transforms

Unity AnimationCurve

HeadDuck MonoBehaviour

Overview

The HeadDuck effect lowers the head. It is used in charged jumps.

Inspector

Properties

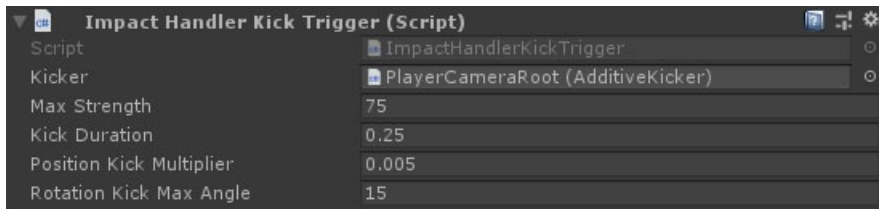| NAME	TYPE	DESCRIPTION
Duck Height	Float	The distance to duck the head downwards.

See Also

Additive Transforms

ImpactHandlerKickTrigger MonoBehaviour

Overview

The ImpactHandlerKickTrigger behaviour is used to drive the AdditiveKicker.

Inspector

Properties

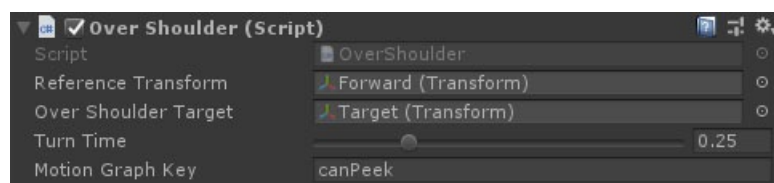| NAME	TYPE	DESCRIPTION
Kicker	AdditiveKicker	The additive kicker used to react to the force.
Max Strength	Float	The strength cap. Above this strength, the kick will not increase.
Kick Duration	Float	The time taken to recover from the kick.
Position Kick Multiplier	Float	The camera moves along the direction of the force by strength * position multiplier.
Rotation Kick Max Angle	Float	The rotation kick angle at full strength.

See Also

Additive Transforms

AdditiveKicker

OverShoulder MonoBehaviour

Overview

The OverShoulder behaviour allows you look over your shoulder, keeping your weapon pointing in its original direction.

Inspector

Properties

NAME	TYPE	DESCRIPTION
ReferenceTransform	Transform	The transform to use as the un altered aim direction.
OverShoulderTarget	Transform	The transform to look down (forwards) when looking over shoulder.
TurnTime	Float	The time taken to turn.
MotionGraphKey	String	The key to a motion graph switch parameter that dictates if the character can peek or not

See Also

Additive Transforms

Motion Graph Parameters And Data

PeekVertical MonoBehaviour

Overview

The PeekVertical behaviour allows you to peek over or under obstacles by moving the head up/down.

Inspector
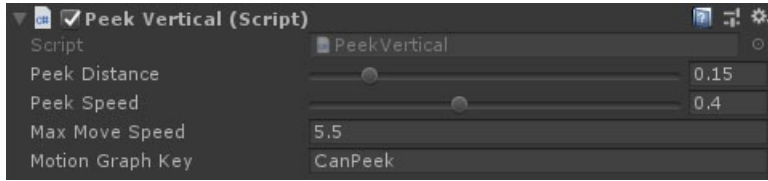
Properties

NAME	TYPE	DESCRIPTION
Peek Distance	Float	The distance up or down to move the camera when peeking up or down.
Peek Speed	Float	The speed the character can change lean amount.
Max Move Speed	Float	The maximum speed the character can travel before the peek is cancelled. (0 = no max speed).
Motion Graph Key	String	The key to a motion graph switch parameter that dictates if the character can peek or not.

See Also

Additive Transforms

Motion Graph Parameters And Data

PositionBob MonoBehaviour

Overview

The PositionBob behaviour is used to apply a bob effect to the player character's head and weapon. The bob syncs steps with the RotationBob effect and the various sprint handlers. It can also blend the bob effect between the head and weapon object to keep a consistent effect while reducing head movement for people that find it causes them motion sickness. This is controlled via the head bob setting in the FpsGameplaySettings.

Inspector
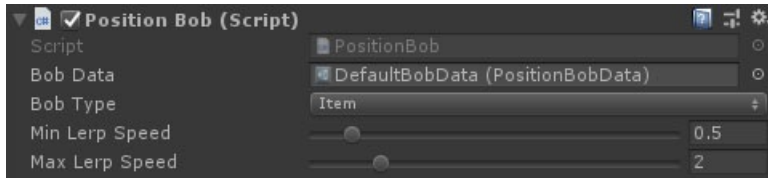
Properties

NAME	TYPE	DESCRIPTION
Bob Data	PositionBobData	The bob animation data, shared between the head and the item/weapon.
Bob Type	Dropdown	Is this bob being applied to the head or the item (allows the effect to blend between the 2 with similar results based on game settings).
Min Lerp Speed	Float	At or below this speed the bob will be scaled to zero.
Max Lerp Speed	Float	At or above this speed the bob will have its full effect.

See Also

Additive Transforms

RotationBob

Gameplay Settings

PostProcessLayerFix MonoBehaviour

Overview

The PostProcessLayerFix behaviour adds the **PostProcessLayer** component to any cameras on start at runtime.

The Unity Post Processing Stack V2 has a number of bugs relating to how it serializes and deserializes references its shared resources. This makes it very difficult to use in an asset because on first import it can lead to bugs ranging from post-processing effects not being applied, to errors being spammed to the console. The fixes for this involve manually editing every camera object, and so this behaviour was added in an attempt to shift the requirement for this away from the end user.

Inspector
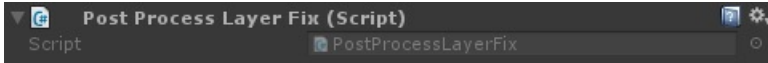
Properties

The PostProcessLayerFix behaviour has no properties exposed in the inspector.

See Also

Post Processing Stack V2

PostProcessLayerSettings

RotationBob MonoBehaviour

Overview

The RotationBob behaviour is used to apply a bob effect to the player character's weapon. The bob syncs steps with the PositionBob effect and the various sprint handlers. Rotation bob should be used sparingly and not applied to the character body or head as it can be very disorientating.

Inspector
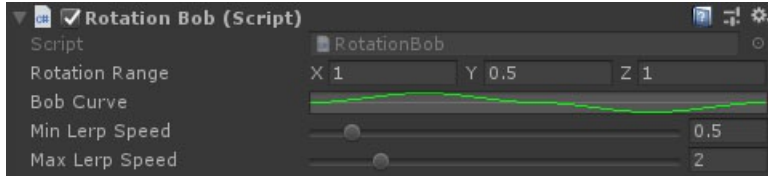
Properties

NAME	TYPE	DESCRIPTION
Rotation Range	Vector	The maximum rotation on each axis at the peak of the bob. Negative values essentially offset the timing for that access to the other foot.
Bob Curve	[AnimationCurve][unity-animationcurve]	The curve over one step cycle for the bob effect.
Min Lerp Speed	Float	At or below this speed the bob will be scaled to zero.
Max Lerp Speed	Float	At or above this speed the bob will have its full effect.

See Also

Additive Transforms

PositionBob

SphereShakeZone MonoBehaviour

Overview

The SphereShakeZone behaviour defines a spherical area with a constant shake value used by the CameraShake additive transform effect.

Inspector
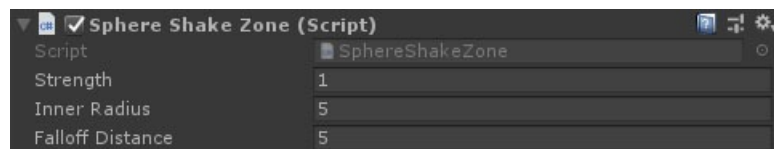
Properties

NAME	TYPE	DESCRIPTION
Strength	Float	The strength of the shake.
Inner Radius	Float	The inner radius of the sphere. Inside this, the strength is constant.
Falloff Distance	Float	The distance outside of the box that the strength falls off to zero.

See Also

Additive Transforms

CameraShake

TransformMatcher MonoBehaviour

Overview

The TransformMatcher behaviour is an additive effect that matches the position and rotation of one transform relative to another. It is useful for layering keyframed animation in with the procedural animation of the other additive effects.

Inspector
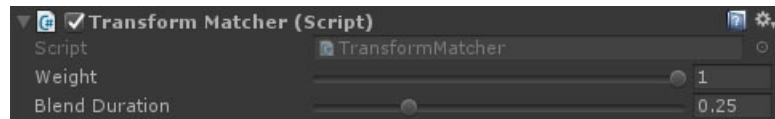
Properties

NAME	TYPE	DESCRIPTION
Weight	Float	The strength of the effect. 1 matches the movement absolutely, while 0 is no movement.
Blend Duration	Float	The time it takes to blend in or out of the movement when the transforms are changed.

See Also

Additive Transforms

FirearmTransformMatchSetter

WeaponAimAmplifier MonoBehaviour

Overview

The WeaponAimAmplifier uses the camera rotation to twist and turn the held weapon, adding the illusion of weight and momentum.

Inspector

Properties

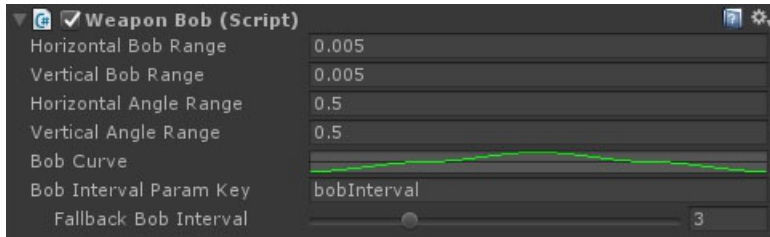| NAME	TYPE	DESCRIPTION
Horizontal Multiplier	Float	The multiplier for the resulting weapon rotation side to side.
Vertical Multiplier	Float	The multiplier for the resulting weapon rotation up and down.
Sensitivity	Float	How sensitive the sway is to camera rotation. Higher sensitivity means the sway approaches its peak with slower rotations
Damping Time	Float	Approximately the time it will take to reach the target rotation. A smaller value will reach the target faster.

See Also

Additive Transforms

WeaponBob MonoBehaviour (Deprecated)

Overview

The WeaponBob behaviour is used to move the gun or held item in sync with character movement. This behaviour has been deprecated and replaced with the PositionBob and RotationBob systems which sync weapon and head bobbing, and allow player settings to reduce head movement.

The weapon bob can tie into the moion graph and subscribe to a float parameter to allow the motion graph elements to control when the bob occurs and the bob rate. Without this, the bob occurs whenever the character is grounded, and the bob rate is based on movement speed.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Horizontal Bob Range	Float	The maximum position offset along the x-axis in either direction.
Vertical Bob Range	Float	The maximum position offset along the y-axis in either direction.
Horizontal Angle Range	Float	The maximum rotation offset around the x-axis in either direction.
Vertical Angle Range	Float	The maximum rotation offset around the y-axis in either direction.
Bob Curve	AnimationCurve	The curve over one step cycle for the weapon bob.
(Fallback) Bob Interval	Float	The distance travelled for one full bob cycle. If the parameter key above is set then this value will only be used if the parameter can not be found.
(Fallback) Bob Interval	Float	The distance travelled for one full bob cycle.

See Also

Additive Transforms

HeadBob

Unity AnimationCureve

WeaponMomentumSway MonoBehaviour

Overview

The WeaponMomentumSway behaviour is an alternative to the WeaponAimAmplifier behaviour and moves instead of rotates the weapon. This makes it more suitable to weapons that are set to use the gun transform for the shooting direction as the weapon will stilll be pointing forwards as it moves.

Inspector

Properties

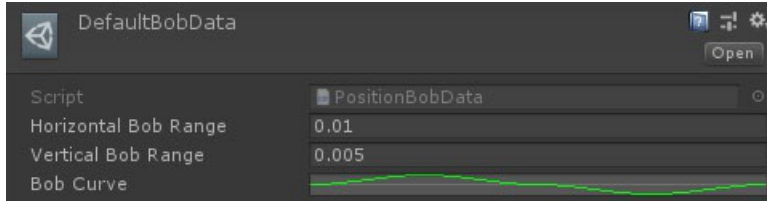| NAME	TYPE	DESCRIPTION
Horizontal Multiplier	Float	The multiplier for the position offset side to side.
Vertical Multiplier	Float	The multiplier for the position offset up and down.
Sensitivity	Float	How sensitive the sway is to camera rotation. Higher sensitivity means the sway approaches its peak with slower rotations
Damping Time	Float	Approximately the time it will take to reach the target rotation. A smaller value will reach the target faster.

See Also

[Additive Transforms](#)

PositionBobData ScriptableObject

Overview

The PositionBobData scriptable object stores bob information to be shared between multiple PositionBob behaviours. This allows all the bob effects on a character to sync up, and for the player to blend between item and head bob if htey find head bob uncomfortable.

Inspector

Properties

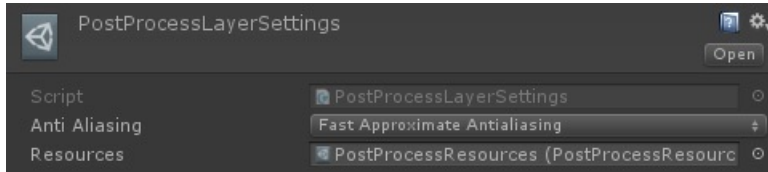| NAME	TYPE	DESCRIPTION
Horizontal Bob Range	Float	The maximum position offset along the x-axis in either direction.
Vertical Bob Range	Float	The maximum position offset along the y-axis in either direction.
Bob Curve	AnimationCurve	The curve over one step cycle for the bob effect.

See Also

Additive Transforms

PositionBob

Unity AnimationCurve

PostProcessLayerSettings ScriptableObject

Overview

The PostProcessLayerSettings asset is used to store common post processing settings that are applied to any cameras on start.

Inspector

Properties

| NAME	TYPE	DESCRIPTION
Anti Aliasing	Dropdown	The anti-aliasing option to use when adding post processing to a camera.
Resources	PostProcessResources	The post processing resources to apply to the camera. **Note:** on first import this property will be null. The resources are created when the post processing package is imported and not accessible via the inspector. When you first access play mode, this component will be retrieved and applied, and the property will persist outside of play mode.

See Also

Post Processing Stack V2

PostProcessLayerFix

The Motion Graph

Overview
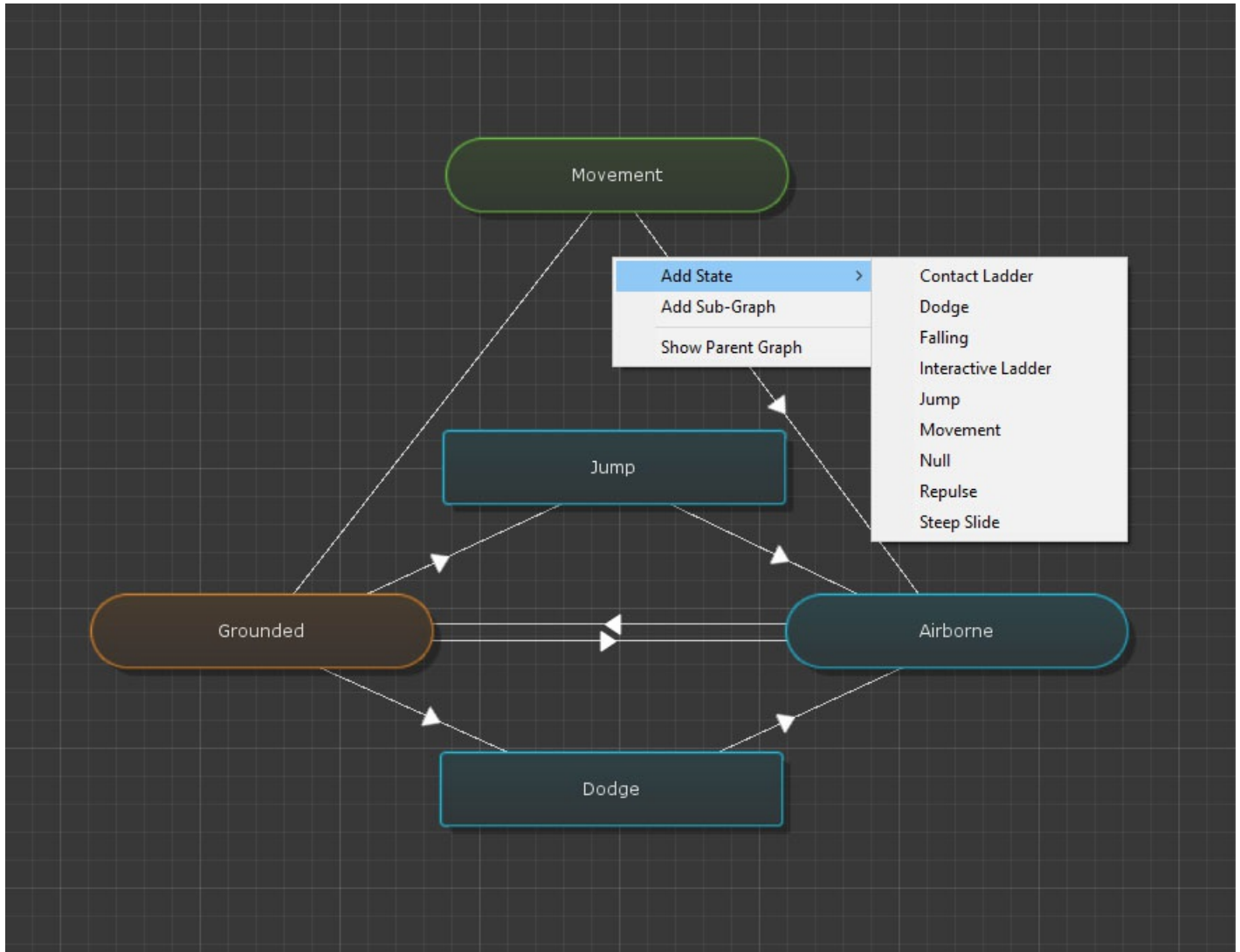
The motion graph is the core movement system of NeoFPS. It provides a powerful but flexible tool for designers to control how the FPS character moves, and how the character's movement state affects other Unity systems such as audio and animation. The motion graph means designers aren't tied to a specific style of movement, and minimises the need for complex custom code when trying to achieve a specific FPS vision (though it is also designed to be highly extensible).

The motion graph is a state machine that tracks and manages the movement state of the FPS character. It is built up of States, Connections, Sub-Graphs, Behaviours, Parameters and Motion Data.

The NeoFPS movement system uses a MotionController behaviour attached to the FPS character, which drives a NeoCharacterController. The MotionController has parameters to select the desired motion graph ScriptableObject, along with a [MotionControllerData ScriptableObject][7] that the graph queries for parameters such as movement speed and strafe multipliers. By separating the motion controller from the graph and data, NeoFPS allows for a huge amount of flexibility in creating separate characters with unique movement styles and traits in a single game. This can be very useful if the game design involves character classes or RPG style stats.

Graph Elements

States

States are the building blocks of the motion graph. Each state is a style of movement and has complete control of the character's MotionController.

The states base their movement on the motion graph Parameters and Motion Data as well as a number of properties from the MotionController and NeoCharacterController such as ground contact and input.

For more information, see Motion Graph States.

Connections
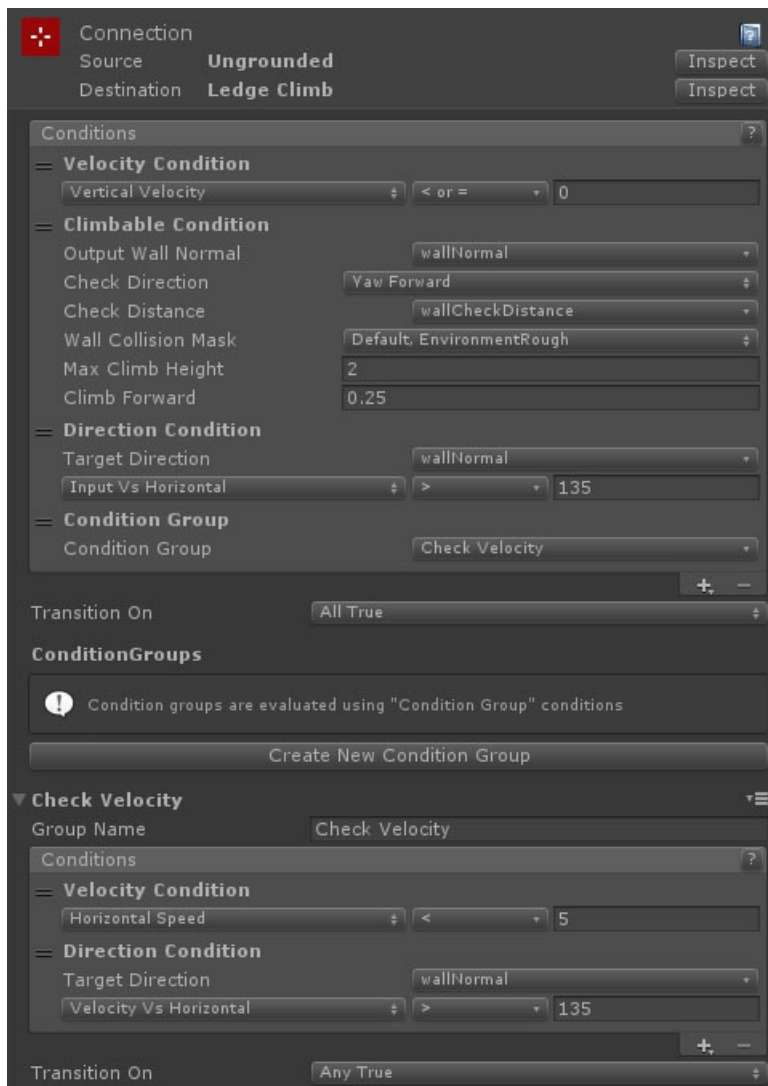
Connections handle the relationship between movement states. Each update tick, all of the outbound connections of the current state are checked in sequence. If an outbound connection is found to be valid, then that state in turn is checked until the final state in the chain is reached. That final state becomes the new "current" state, and is then updated.

In order for a connection to be valid, the conditions attached to it must return true (all or any depending on the setting).

Connections are evaluated for parent sub-graphs as well to allow for grouping logic. The connection checks are handled in order from the highest level sub-graph containing the current state, down the sub-graph hierarchy, and lastly the current state. An example of how his can be useful is switching between grounded and airborne movement. You can group all of the grounded movement states (running, walking, crouching, etc) and connections together in their own sub-graph, and group all of the

airborne states and connections together in another sub-graph. Connecting these 2 sub-graphs using a ground contact check condition means that these connections are much higher priority than the connections inside the respective sub-graphs. It doesn't matter what connections exist and conditions are met for grounded movement if the character is no longer touching the ground. Without this ordering and grouping of nodes into sub-graphs, every grounded movement state would require a connection to the airborne states.
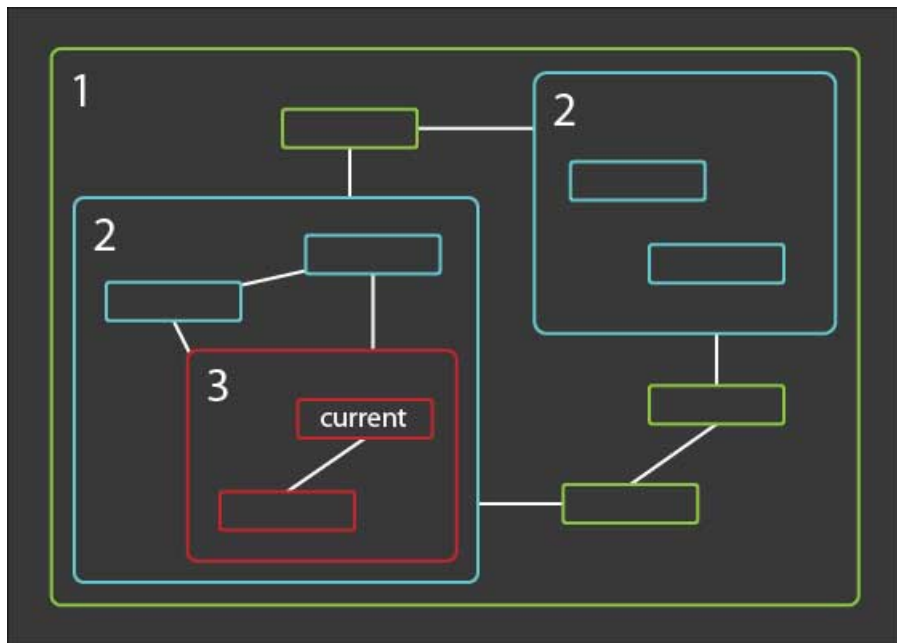
Conditions
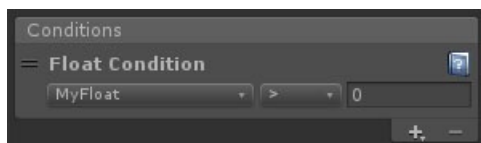
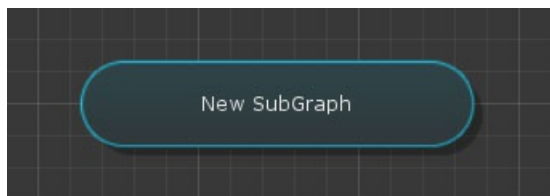
Conditions are a one off test that returns either true or false. The tests can be based on the graph state, graph parameters, or external systems such as physics and character health.

For more information, see Motion Graph Conditions.

Sub-Graphs

Sub-Graphs allow states to be logically grouped together into smaller graphs. They have inbound and outbound connections just like a state and are tested before the current state in order from the graph root down to the current sub-graph and finally the current state. This enables complex logic to designed through a simple interface. For example, most characters will have an **Airborne** and a **Grounded** sub-graph. Inside the **Grounded** sub-graph will be various states handling movement such as walking, sprinting and sliding, as well as the connections between them. If the character loses ground contact then it doesn't matter which of those states the character is in. The sub-graph connections will handle the connection into the *Airborne* sub-graph and then select the relevant airborne state.

Each sub-graph has a **default** state. On entering a sub-graph, all of the connections to child states are evaluated in sequence. If none of them are valid then the default state automatically becomes the current state.

Parameters And Data

Motion graph parameters are similar to the parameters inside the Unity AnimatorController. Any number of parameters can be added to the motion graph and then referenced by key or hash. You can also store the parameter reference for quicker access.

Certain parameters such as triggers and switches can be blocked, either by graph behaviours, or from outside the graph.

The motion graph also allows for an event parameter. This can be subscribed to from outside the graph and triggered by graph behaviours, allowing the graph to interact with components outside of the motion graph system.

Motion data behaves similarly but cannot be changed from outside the graph, except for using an override asset. Motion data are referenced in graph states and behaviors, and are completely optional.

For more information, see Motion Graph Parameters And Data.

Behaviours
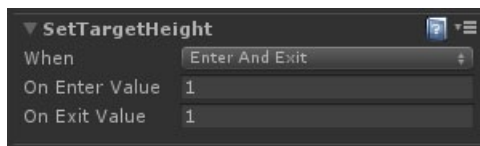
Motion graph behaviours add extra functionality to the graph. They can process logic on entering or exiting a state or sub-graph and/or when a state is updated. Certain behaviours can only be attached to states, and others can only be attached to sub-graphs, while most can be attached to both.

Motion graph behaviours allow for simple logic such as resetting or modifying parameters. They can also be complex systems on their own, such as the motion graph behaviour based footsteps.

For more information, see Motion Graph Behaviours.

See Also

Motion Graph Editor

Motion Graph States

Motion Graph Conditions

Motion Graph Parameters And Data

Motion Graph Behaviours

MotionController Behaviour

Unity AnimatorController

Ladders

Moving Platforms

Extending The Motion Graph

NeoCharacterController

Overview

The NeoCharacterController is a replacement for the Unity Character Controller which adds additional functionality such as:

- Moving platforms
- Interpolation between fixed framerate physics movement
- Arbitrary gravity and up-vector
- Height change, including jump-crouching
- Built-in pushing of rigidbody objects
- Reaction to impacts from rigidbody objects
- Physical interaction with other character controllers
- Curve driven slope speed modifiers
- Ledge friction property to allow sliding off ledges when overhanging

The NeoCharacterController is a kinematic character controller that uses a collide and slide technique to ensure smooth movement and collisions, while preventing fast movement penetrating through thin objects.

For more information on the properties that are exposed in the inspector see the NeoCharacterController Behaviour.

The other classes within NeoFPS refer to the NeoCharacterController using the `INeoCharacterController` interface. This means that the character controller can be replaced with another implementation, as long as that implements the correct interface.

Friction

The **Slope Friction** property dictates how much of the downward movement is redirected down the slope instead of cancelled out. At a friction level of 1, any vertical movement into the slope will be cancelled out completely. At a friction level of 0, the character will slide down the slope due to gravity.

The **Ledge Friction** property specifies what happens when the character is overhanging a ledge. The NeoCharacterController has basic awareness of ground contacts that lets it know when it is in contact with an edge as opposed to a flat surface. If the character centerpoint is overhanging a ledge, and the drop distance is large enough, then the character will slide off the ledge depending on this setting. It can be used to prevent the character from hovering off the edge of obstacles unrealistically. If this value is higher than the slope friction, then the slope friction will be usd instead. This prevents sliding down a slope and then sticking on the bottom edge.

Rigidbody and Character Interaction

The NeoCharacterController can push dynamic rigidbodies, as well as react to impacts from rigidbodies.

The **Low Rigidbody Push Mass** is used to set a mass that the character can easily push. Any rigidbody at or below this mass will have a proportionate force exerted on it that will achieve the same effect. Above this mass, that force will drop away to zero as the rigidbody approaches the **Max Rigidbody Push Mass**. The **Rigidbody Push** property is the push power and can be experimented with to get the correct effect.

Similarly, the NeoCharacterController can also push and be pushed by other NeoCharacterController objects.

Gravity and Up-Vector

The NeoCharacterController operates using its own gravity value instead of the Unity Physics gravity. This allows for more flexibility to achieve a specific game feel. First person shooters often use a higher gravity than normal for character movement as it feels more realistic and less floaty.

The NeoCharacterController can also have its up-vector changed. This allows for features such as walking around on relatively small planets, or localised gravity zones. All features such as step height and slopes work the same as gravity and the up-vector rotate.

By default, the up-vector will be adjusted to face the opposite direction to gravity whenever the character gravity is changed. If the gravity is set to zero then the up-vector can be freely changed to allow for unlimited zero-g movement. A range of motion graph states will be added to take advantage of this in a future update.

See Also

Moving Platforms

NeoCharacterController Behaviour

Unity Character Controller

The Motion Graph Editor

Overview

The motion graph editor is a set of tools for creating and editing NeoFPS motion graphs. It contains a viewport for navigating and visualising the graph, along with an inspector for changing the properties of the graph and its components.
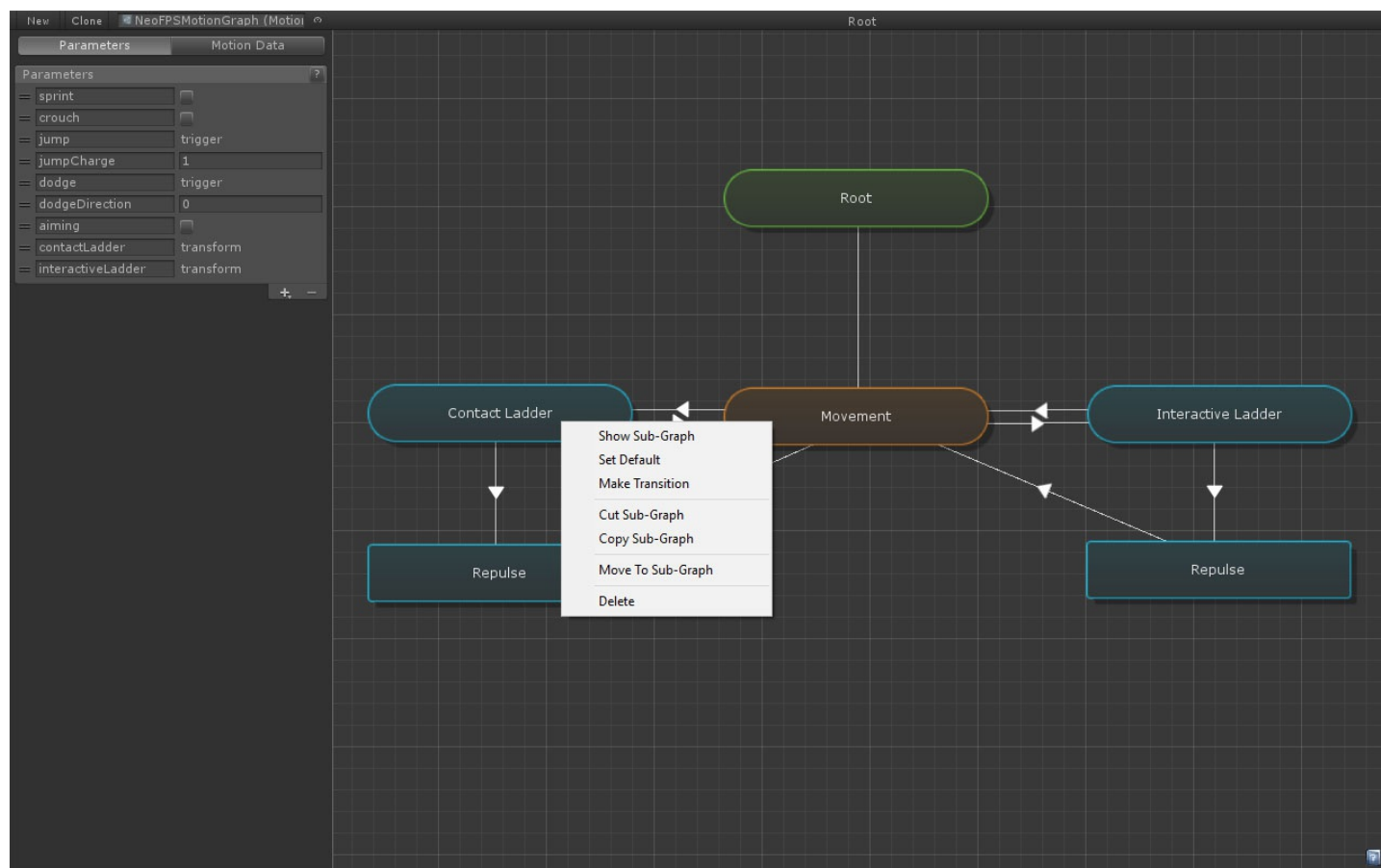
You can access the motion graph editor through the Unity menu: **Window/NeoFPS/Motion Graph Editor**. You can also access the editor from the **Show Motion Graph Editor** button that appears in the inspector for a motion graph asset.

You can create a new motion graph from the motion graph editor by clicking the **New** button in the top bar. The new asset will be placed in the project root folder. You can also right-click in the project and create a new motion graph through the create menu using the following command: **Create/NeoFPS/Motion Graph**. Selecting a motion graph asset with the motion graph editor open will update the editor to show the contents of the selected graph.

Motion Graph Editor UI
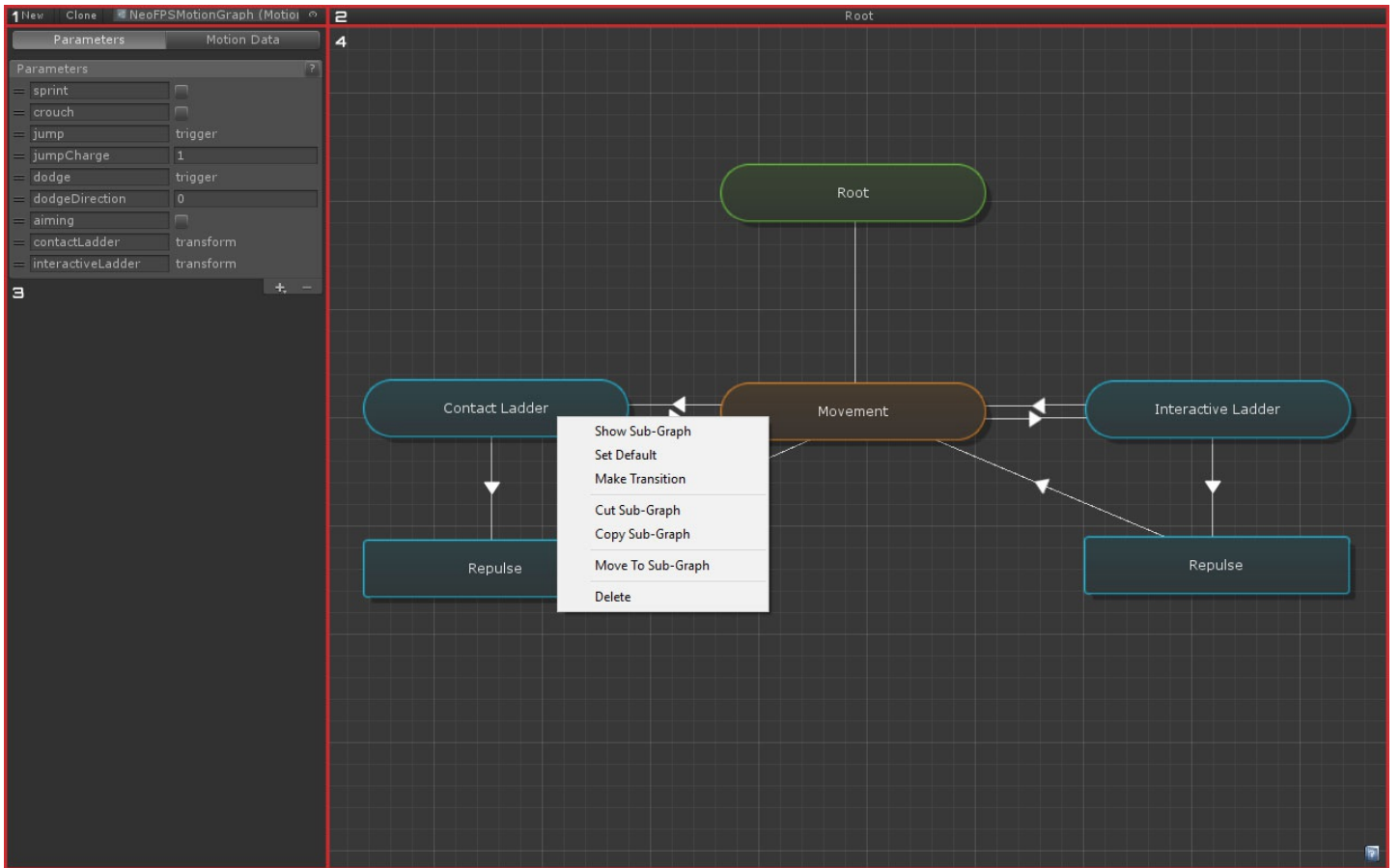
The following is a breakdown of what you will see when you open the editor:

1. The **asset** controls allow you to create and save graph assets as well as select the graph currently being edited.
2. The **breadcrumb** at the top shows the current sub-graph of the hierarchy that is currently visible in the viewport.
3. The **side bar** allows you to add and modify parameters and motion data on the graph.
4. The **viewport** shows the layout of the graph.

Motion Graph Viewport
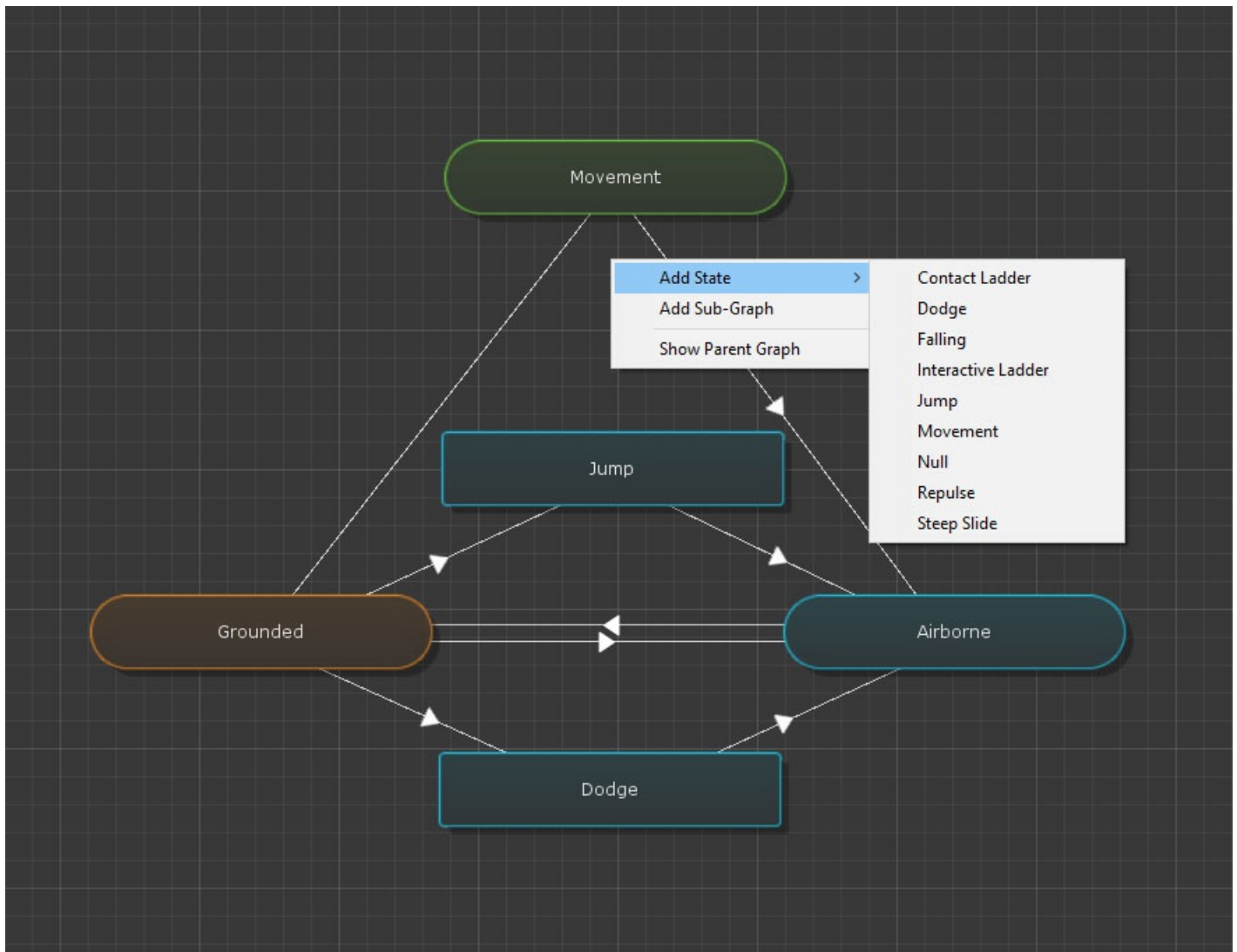
The motion graph viewport gives a visual representation of the graph, allowing you to navigate the connections, add elements and select and modify existing elements.

To move the viewport hold the **middle mouse button** and drag the cursor. Alternatively, you can hold the **Left Alt** key and click the **left mouse button** on an empty area of the viewport and then hold and drag the mouse button to move.

To open a context menu you can right click anywhere in the viewport. This menu will contain different entries depending on what you clicked and covers actions such as navigating the graph, cut/copy/paste/deleting elements and creating transitions between elements.

Inside the viewport, the visible elements are as follows:

- The rectangular boxes are motion graph states. These handle the movement of the character and the motion controller depending on the motion controller's current state. **Left-click** a state to edit its properties in the editor inspector panel.
- The angled boxes are sub-graphs. These contain their own graph layout and can be transitioned into and out of as if they were a state. **Left-click** a sub-graph to edit its properties in the inspector panel. **Double-click** a sub-graph to replace the viewport contents with that sub-graph's contents.
- The green angled box is the sub-graph that is currently being shown in the viewport. You can **double-click** this element or use the context menus to show its parent graph in the viewport.
- The lines with arrows between different elements are the motion graph connections. These define which states and sub-graphs connect to each other and what conditions must be met to transition between them. **Left-click** the arrow on a connection to edit its properties in the inspector panel.

You can also group select states and sub-graphs in order to perform group operations. To drag-select elements, **Left-click** in an empty area of the viewport and, holding the mouse button down, drag across the desired states and sub-graphs. Releasing the mouse button will select the states and sub-graphs within the area. Connections can not be group selected, but any group

operations will affect the connections into and out of the modified elements. You can also **Control-Left-Click** or **Command-Left-Click** on an element to add it to the group, or **Alt-Left-Click** to remove it from the group.

Any selected states or sub-graphs will be highlighted in the viewport.

Inspecting Motion Graph Elements

Selecting an element in the viewport will show it in the Unity editor inspector.

Connectables (States an Sub-graphs)
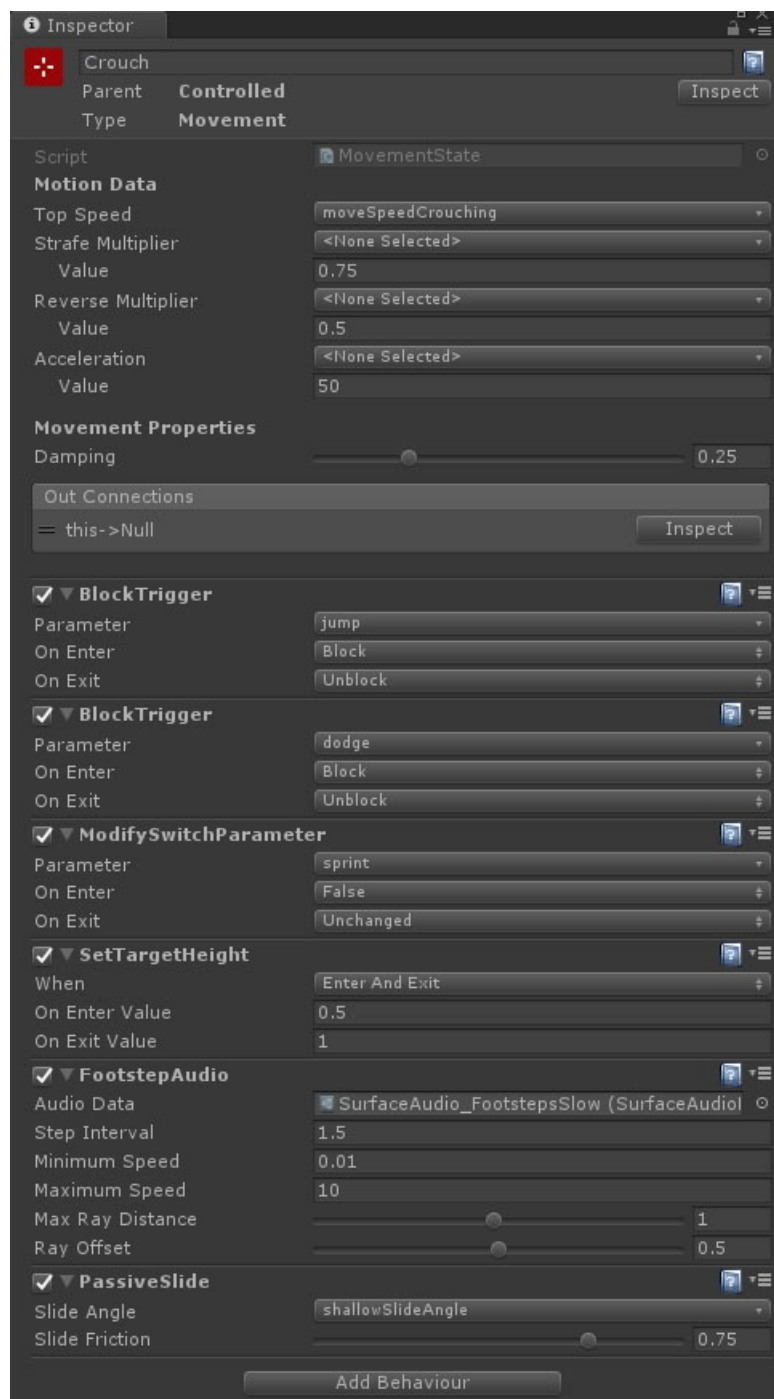
The connectable inspector is used for both states and sub-graph elements due to the similarities between them.

The header of the inspector allows you to change the element's name, see (and inspect) its parent sub-graph, and in the case of states, see its type.

Straight after the header are the connectable properties. States can have any number of properties, and in some cases might have complex editor logic such as you would expect from a MonoBehaviour or ScriptableObject.

After the connectable properties is an **Out Connections** array that shows all of the connections out of the inspected element. Each entry in the array shows the destination, along with an **Inspect** button. Clicking this will select the connection element and show its properties in the inspector for editing.

Both states and sub-graphs can also have a number of behaviours attached to them. These are listed after the out connections. Each behaviour has a foldout title bar that can be used to show or hide its properties. They also have a link to the relevant manual reference for that behaviour, and a dropdown context menu with options for changing order, copy/paste and removal. You can add behaviours to the state by clicking the **Add Behaviour** dropdown button and selecting a behaviour. For more information on behaviours, see Motion Graph Behaviours.

Connections
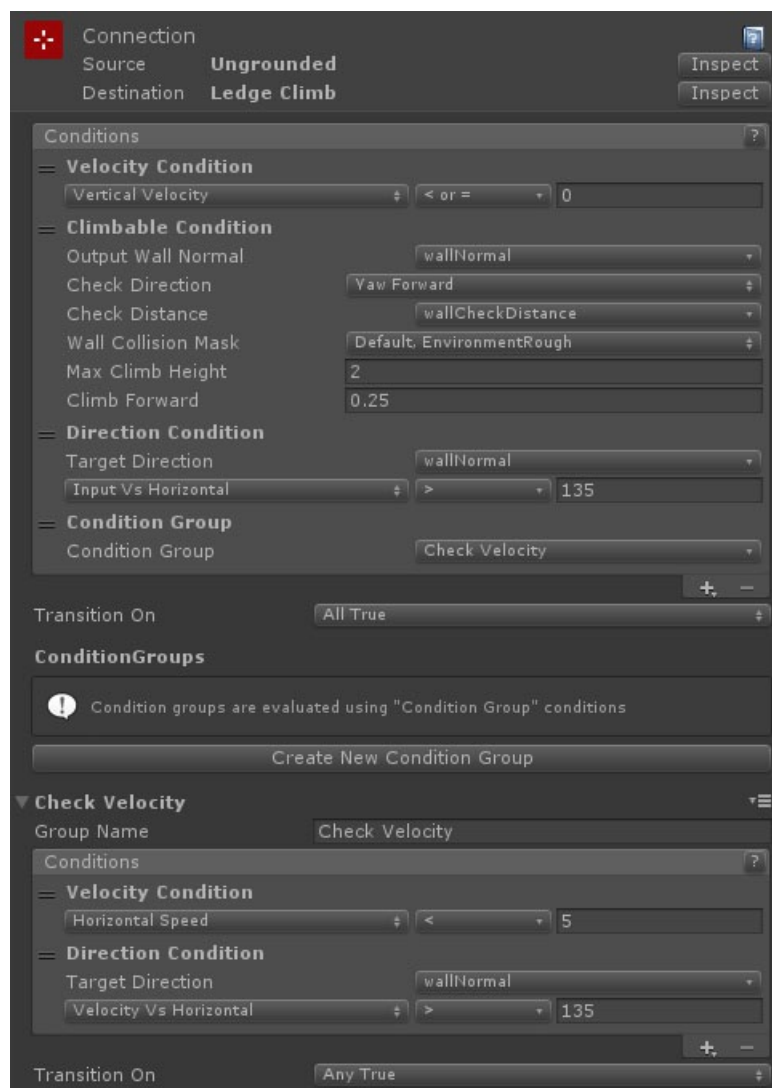
The header of the connection inspector shows the source and destination elements and allows you to quickly inspect either one.

The **Conditions** array shows all the conditions that decide if the connection is valid. For more information on conditions, see Motion Graph Conditions. You can add a new condition by clicking the **+** button at the bottom of the array and selecting the relevant condition from the dropdown.

The **Transition On** dropdown species if all of the conditions must be met for control to be transferred, or if any one of the conditions must be met.

Parameters and Data
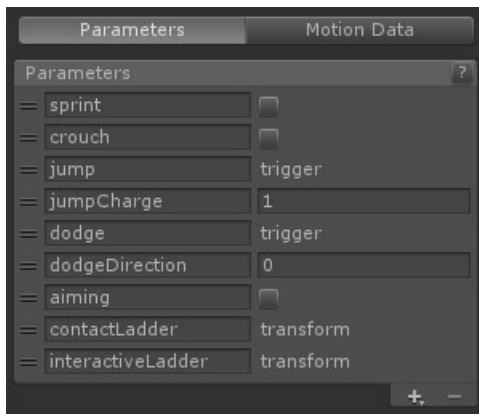
The side-bar of the editor has 2 tabs that show all of the parameters attached to the graph, and all of the motion data attached to the graph. Parameters and data can be referenced from any state, behaviour or condition in the graph. Scripts can also get a reference to any of the attached parameters from outside the graph, creating a flexible system for communication back and forth between the graph and other systems.

For more information, see Motion Graph Parameters And Data.

See Also

The Motion Graph

Motion Graph States

Motion Graph Conditions

Motion Graph Parameters And Data

Motion Graph Behaviours

Unity AnimatorController

Motion Graph States

Overview

States are the building blocks of the motion graph. Each state is a style of movement. It takes input and basic physics information from the MotionController and returns a move vector each tick.

Included States

Ground Movement States

NAME	DESCRIPTION
Dash	Surge in a direction based on yaw or movement direcion.
Movement	Standard ground based movements such as walking or running.
Ski	Maintain ground speed and allow steering based on speed (slower speed is tighter steering).
Steep Slide	Sliding on a steep slope, with some control over direction.

Airborne States

NAME	DESCRIPTION
Controlled Jetpack	Applies an upward force, with some air control.
Falling	Air control based movement with gravity.
Fly	No-clip style movement that can move in all axes.
Jetpack	Applies an upward force and maintains horizontal velocity.

Instantaneous States

NAME	DESCRIPTION
Boost Pad	Instantly sets velocity to match a vector parameter.
Dodge	As jump, but lower and further.
Impulse	Instantly sets velocity based on a vector with options for space and ground alignment.
Jump	An instant upward jump. Completes in one frame.
JumpDirection	A jump, but the direction is rotated from the vertical based on input.
JumpDirectionV2	A jump with a fixed vertical speed and a horizontal speed based on input.
Push Off	Applies velocity based on a vector parameter, with optional upward force.
Repulse	An instant impulse away from a transform in the scene.

Misc States

NAME	DESCRIPTION
Constant Move	Accelerates to a fixed velocity and maintains it.
Maintain Velocity	Simply returns the velocity from the previous frame.
Match Transform	Attaches the character (position with or without yaw and up-vector) to a transform in the scene.
Move To Point	Moves from the starting point to the position set in a vector parameter.
Null	An empty state used to branch to others.

Wall Movement States

NAME	DESCRIPTION
Mantle	Climb onto a ledge.
Vertical Wall Run	Run directly up a wall.
Wall Run	Run along a wall surface.

Swimming States

NAME	DESCRIPTION
Swim Smooth Surface	Swimming state that tries to stick to a water zone's surface. Smooth movement.
Swim Smooth Underwater	Swimming state that can move in all axes. Smooth movement.
Swim Stroke Surface	Swimming state that tries to stick to a water zone's surface. Moves in pulses.
Swim Stroke Underwater	Swimming state that can move in all axes. Moves in pulses.
Swim Submerge	Used to transition from surface to underwater states while taking the movement of the surface into account.
Wading	A variation of the movement state with speed dependent on the amount of a character's body below the water surface.

Ladder States

| NAME	DESCRIPTION
Contact Ladder	A basic climb state when touching a contact ladder.
Interactive Ladder	A climb state constrained to an interactive ladder.

In The Viewport

Context Menu Options

NAME	DESCRIPTION
Set Default	When entering a subgraph, the motion graph checks the connections out of the sub-graph to its child nodes in sequence. If none of the connections are valid then the default state or sub-graph will be chosen as a fallback.
Make Transition	Starts a connection out of this node in the viewport. Clicking on another node will form a connection from this node to the clicked node. Clicking an empty area of the viewport will cancel the connection.
Move To Sub-Graph	This will replace the state with a sub-graph with the same name, and then make this state a child of the new sub-graph.
Cut State	Removes the state from the graph and adds it to the clipboard for pasting later. Any inbound and outbound connections will be removed. Right-click on an empty area of viewport to paste.
Copy State	Copy the state to the clipboard. Right-click on an empty area of viewport to paste.
Delete	Remove the state from the graph entirely, along with any inbound and outbound connections and attached behaviours.

In The Inspector
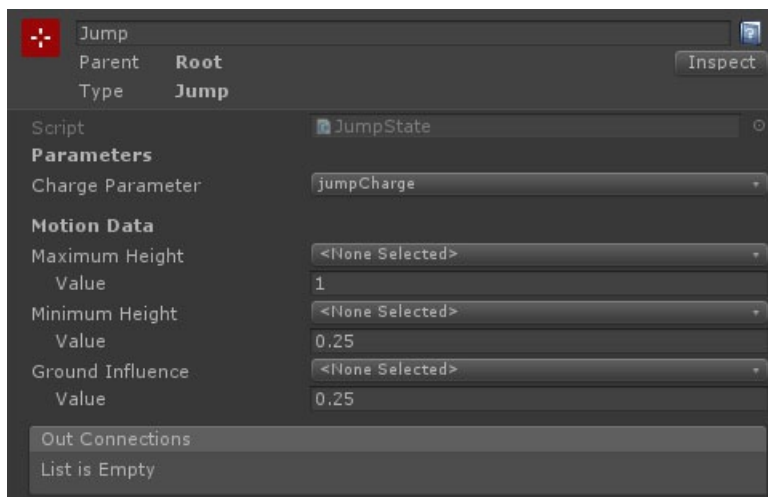
Each state will have its own properties visible before the **Out Connections** array. This array lists the connections from this state to other connectables, allowing you to jump to inspecting each of those targets. The **Add Behaviour** button allows you to add motion graph behaviours to the state which will be evaluated while the state is active.

See Also

[The Motion Graph](#)

[The Motion Graph Editor](#)

Motion Graph Behaviours

Overview

Motion graph behaviours add extra functionality to the graph. They can process logic on entering or exiting a state or sub-graph and/or when a state is updated. Certain behaviours can only be attached to states, and others can only be attached to sub-graphs, while most can be attached to both.

Motion graph behaviours allow for simple logic such as resetting or modifying parameters. They can also be complex systems on their own, such as the motion graph behaviour based footsteps.

Motion graph behaviours update at the same rate as the motion controller - in FixedUpdate. This should be kept in mind if creating new behaviours for your project.

Included Behaviours

Character Behaviours

Character behaviours act on components on the character object.

These include:

- BodyTiltBehaviour
- ConstrainCameraYawBehaviour
- DrainStaminaBehaviour
- LockInventorySelectionBehaviour
- ModifyStaminaBehaviour
- SetWieldableStanceBehaviour
- TrackStepsBehaviour
- UnlockInventorySelectionBehaviour

Graph Parameter Behaviours

Graph parameter behaviours act on motion graph parameters.

These include:

- ClampFloatParameterBehaviour
- ClampIntParameterBehaviour
- ModifyIntParameterBehaviour
- ModifyFloatParameterBehaviour
- ModifySwitchParameterBehaviour
- ModifyTriggerParameterBehaviour
- ModifyTransformParameterBehaviour
- ModifyVectorParameterBehaviour
- BlockSwitchParameterBehaviour
- BlockTriggerParameterBehaviour
- InvokeEventBehaviour
- TimeOpsBehaviour

Audio Behaviours

Audio behaviours trigger audio in the scene and interact with the various audio systems.

These include:

- FootstepAudioBehaviour

- SlidingAudioBehaviour
- LadderAudioBehaviour
- PlayCharacterAudioBehaviour
- PlayAudioClipBehaviour
- LoopingAudioBehaviour
- SurfaceAudioBehaviour
- SurfaceFootstepAudioBehaviour

Physics Behaviours

Physics behaviours modify the character physics.

These include:

- DisableColliderBehaviour
- PassiveSlideBehaviour
- SetTargetHeightBehaviour

Miscellaneous Behaviours

Miscellaneous behaviours don't fall into the above categories.

These include:

- SetTimeScaleBehaviour

In The Inspector
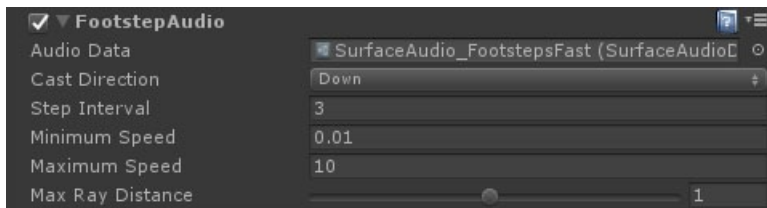
Editing behaviours is very straightforward. Each one will have its own unique properties, along with the following controls as highlighted in the above image:

1. A checkmark to enable or disable the behaviour. Disabled behaviours will not be evaluated until enabled.
2. The type of the behaviour. Click this to collapse or expand the behaviour and its properties.
3. A link to the manual reference for the behaviour.
4. A dropdown of actions for the behaviour including changing the order on the state or sub-graph, copy / pasting properties and removing.

See Also

Motion Graph Parameters And Data

Motion Graph Conditions

Overview

Conditions are simple true/false checks that are used to specify when the motion graph changes its active state. Conditions can be far reaching, checking against the graph, the controller, the wider character, or any external system. Connections can transition when a single condition is met or when all conditions in a connection are met.

Condition Groups

Condition groups are used to evaluate conditions together and allow for more complex rules than just "any" and "all".

For example, when swimming on the surface of a water zone, you might transition to the underwater state if you **either** press crouch **or** look down **and** press forwards. In this situation you would set the main connection to transition on **any** of: crouch is pressed or the *Move Down* condition group is true. The *Move Down* condition group will be set to be true if **all** of: the character is looking down and the player is pressing forward.
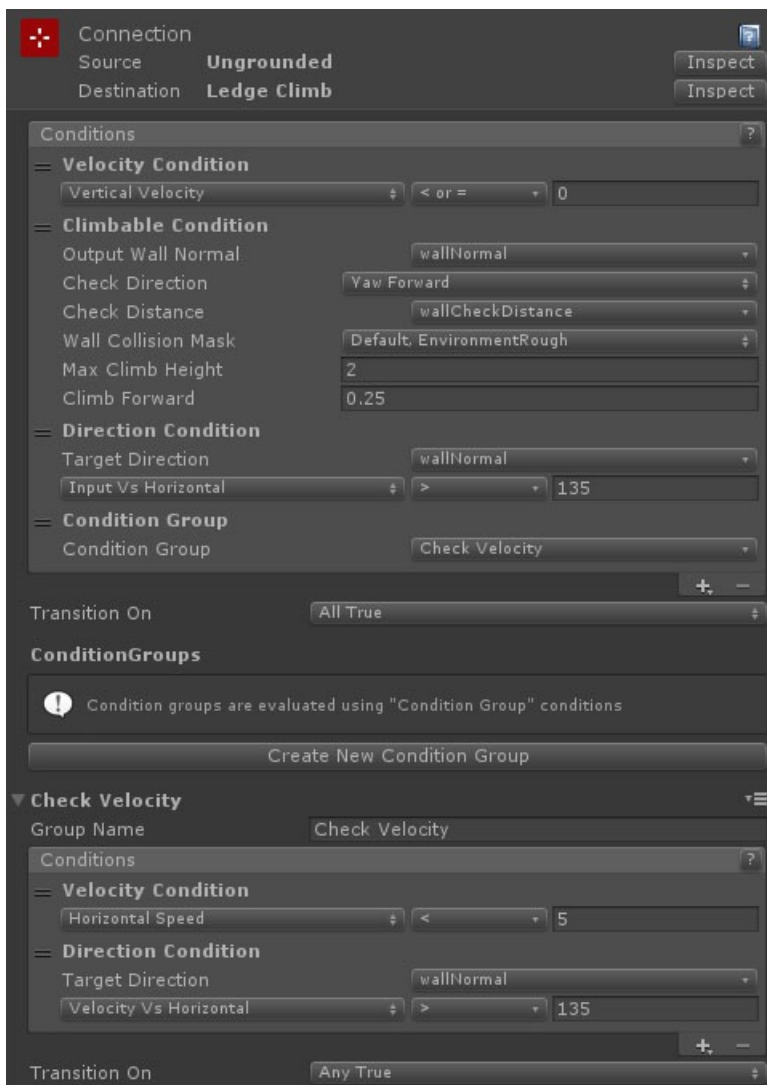
The above connection is used for checking if a character can climb a wall. It is set up with a condition group that checks if either the horizontal character speed is low or the direction of movement is directly into the wall (within a certain angle range).

The results of a condition group are recorded when evaluated (reset each frame). This prevents infinite loops where 2 condition groups reference each other.

Conditions

The following conditions are available:

Graph Conditions

These are conditions that check against the graph state.

NAME	DESCRIPTION
CompletedCondition	Checks if the state it is connected from has its **complete** flag set.
ConditionGroupCondition	Used to evaluate multiple conditions together (see above).
DebugCondition	Used to debug connections by specifying the value and outputting a message to the debug log.
ElapsedTime	Checks if the state it is connected from has been active for a set period.

Parameter Conditions

These are conditions that check against the [parameters][3] attached to the graph.

NAME	DESCRIPTION
CompareFloatsCondition	Compares two float parameters attached to the graph.
CompareIntsCondition	Compares two int parameters attached to the graph.
CompareSwitchesCondition	Compares two switch parameters attached to the graph.
CompareTime	Compares the current time with the time stored in a float parameter attached to the graph.
FloatCondition	Compares a float parameter attached to the graph to a specific value.
IntCondition	Compares an int parameter attached to the graph to a specific value.
SwitchCondition	Compares a switch parameter attached to the graph to a specific value.
TransformCondition	Compares a transform parameter attached to the graph to a specific value.
TriggerCondition	Compares if a trigger parameter attached to the graph has been triggered.
VectorCondition	Compares if a trigger parameter attached to the graph has been triggered.

Physics Conditions

These conditions use the physics system to check for collisions and contacts.

NAME	DESCRIPTION
CapsuleCast	Casts the character capsule to check for a collision.
CapsuleCast (Enhanced)	Casts the character capsule to check for a collision and can store the results in parameters.
CapsuleLookahead	Casts the character capsule based on its movement.

NAME	DESCRIPTION
CapsuleLookahead (Enhanced)	Casts the character capsule based on its movement and can store the results in parameters.
Climbable	Checks if a wall has an unobstructed climbable ledge within a certain height.
RayCast	Performs a ray cast ffrom a point on the character.
RayCast (Enhanced)	Performs a ray cast from a point on the character and can store the results in parameters.
RayLookahead	Performs a ray cast based on the movement of the character.
RayLookahead (Enhanced)	Performs a ray cast based on the movement of the character and can store the results in parameters.
SphereCast	Performs a sphere cast from a point on the character.
SphereCast (Enhanced)	Performs a sphere cast from a point on the character and can store the results in parameters.
SphereLookahead	Performs a sphere cast based on the movement of the character.
SphereLookahead (Enhanced)	Performs a sphere cast based on the movement of the character and can store the results in parameters.

Character Conditions

These conditions interface with the character that is controlled by the graph.

NAME	DESCRIPTION
AirTime	Checks how long the character has been ungrounded.
CharacterHeight	Checks the character capsule height or height multiplier (from standing).
CollisionFlags	Checks the collision flags generated on the last movement frame.
Direction	Checks the character's aim, input or movement direction.
GroundContact	Checks if the character is touching the ground.
GroundNormal	Tests against the normal vector of the ground contact.
GroundSurfaceNormal	Tests against the normal vector of the ground surface at the contact point. If the contact is an edge, this will be the normal of the top face connected to that edge.
HeightRestriction	Checks if the character can reach a specific height (eg trying to stand while crouched in an air vent.
InputVector	Checks against the input vector provided to the motion controller.
Pitch	Checks the character's aim pitch.
ScriptedComponent	Paired with a component attached to the character that implements the `IScriptedComponentCondition` interface.

NAME	DESCRIPTION
Velocity	Checks the character's velocity in various directions.
Water	Checks the character's position relative to a water zone.

See Also

The Motion Graph

Motion Graph Parameters And Data
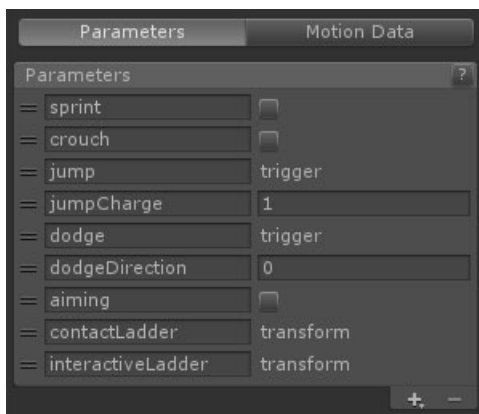
Motion Graph Parameters And Data

Overview

The Motion Graph allows you to add parameter and data entries to the graph.

Parameters are accessible from elements inside the graph and also externally via the get and set parameters outlined below. This enables various systems and your own scripts to feed data to the motion graph that it can use to drive and influence the character movement.

Motion data are used inside the graph, but cannot be changed directly from outside. These values are used for things like speeds that are considered more constant than parameters. You can, however, create override assets which can be attached to the graph at runtime and then

Parameters

Parameters are accessible through the motion graph editor by clicking the **Parameters** tab in the motion graph editor side-bar.

To add a new parameter click the **+** button on the list and select the type from the dropdown.

To remove a parameter select the relevant entry and click the **-** button on the list.

To edit a parameter use the fields in the relevant entry. The text field on the left is the desired name for the parameter. This should be unique to this parameter or only the first entry in the list with that name will be accessible. On the right side of the name is the default starting value. This is the value the parameter will have on initialisation and whenever it is reset.

The following parameter types are available:

NAME	DESCRIPTION
Integer	The integer parameter is a decimal value.
Float	The float parameter is a floating point value.
Switch	The switch parameter is a boolean value. It can be toggled and blocked.
Trigger	The trigger parameter is a one off trigger. It will be reset when it is consumed in the graph. It can also be blocked.
Transform	The transform parameter is a reference to a Transform component.
Vector	The vector parameter is a Vector3 value.
Event	The event parameter is a C# event that can be invoked from inside the graph and subscribed to from outside.

For parameters that can be blocked, they will always return their default value as long as there are blockers. Use `SwitchParameter.AddBlocker()` to add a blocker to a switch parameter, and `SwitchParameter.RemoveBlocker()` to remove a blocker. The parameter will be blocked when the number of blockers is not 0. Be careful to remove any blockers you add when no longer needed.


Parameters are referenced from within the graph using dropdowns. The user can choose to select **None**, **Create New** (another way to add a new parameter to the graph), or any of the relevant parameters attached to the graph.

Events

The event parameter is a useful parameter for interacting with external code from within a motion graph. They can be subscribed and unsubscribed by using the `EventParameter.AddListener (UnityAction listener)` and `EventParameter.RemoveListener (UnityAction listener)` methods.

You can invoke an event parameter from your own code using `EventParameter.Invoke ()` or using a InvokeEventBehaviour motion graph behaviour.

Motion Data
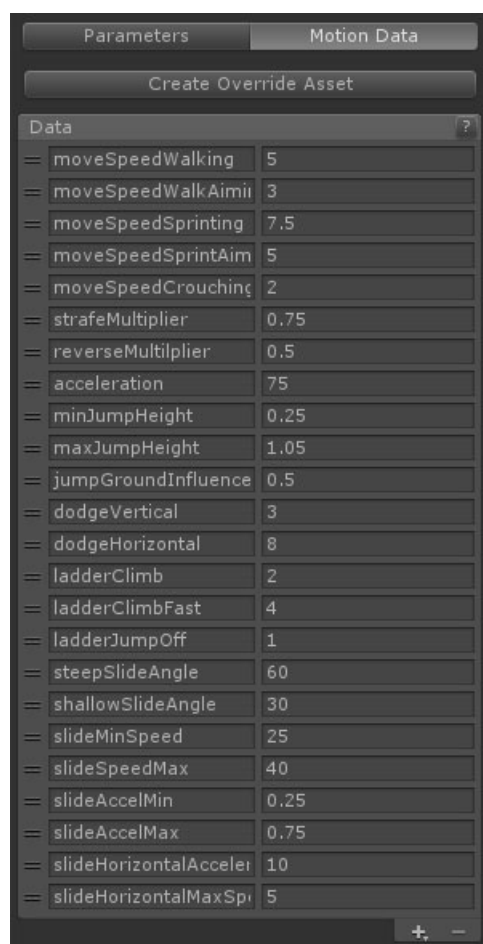
Motion data are accessible through the motion graph editor by clicking the **Motion Data** tab in the motion graph editor side-bar.

To add a new data entry click the **+** button on the list and select the type from the dropdown.

To remove a data entry select the relevant entry and click the **-** button on the list.

The **Create Override Asset** button will create a new MotionGraphDataOverrideAsset in the same folder as the edited motion graph and with the same name.

Various properties on graph states or behaviours can be assigned a motion data value using a dropdown as above. The user can choose to select **None**, **Create New** (another way to add a new motion data entry to the graph), or any of the relevant motion data entries attached to the graph. If the dropdown is set to **None** then a separate **Value** field will be visible below. In simple cases you can use this value directly, but setting the property to a motion data entry allows you to share a single data entry across multiple properties, and also to override from an asset later.

Accessing Parameters

Parameters are accessed from the motion graph root object. They can be set directly with the following methods:

```
int MotionGraphRoot.GetInt (string parameterName);
int MotionGraphRoot.GetInt (int hash);
void MotionGraphRoot.SetInt (string parameterName, int value);
void MotionGraphRoot.SetInt (int hash, int value);

float MotionGraphRoot.GetFloat (string parameterName);
float MotionGraphRoot.GetFloat (int hash);
void MotionGraphRoot.SetFloat (string parameterName, float value);
void MotionGraphRoot.SetFloat (int hash, float value);

bool MotionGraphRoot.GetSwitch (string parameterName);
bool MotionGraphRoot.GetSwitch (int hash);
void MotionGraphRoot.SetSwitch (string parameterName, bool value);
void MotionGraphRoot.SetSwitch (int hash, bool value);

Transform MotionGraphRoot.GetTransform (string parameterName);
Transform MotionGraphRoot.GetTransform (int hash);
void MotionGraphRoot.SetTransform (string parameterName, Transform value);
void MotionGraphRoot.SetTransform (int hash, Transform value);

void MotionGraphRoot.SetTrigger (string parameterName);
void MotionGraphRoot.SetTrigger (int hash);

void MotionGraphRoot.AddEventListener (string parameterName, UnityAction listener);
void MotionGraphRoot.AddEventListener (int hash, UnityAction listener);
void MotionGraphRoot.RemoveEventListener (string parameterName, UnityAction listener);
void MotionGraphRoot.RemoveEventListener (int hash, UnityAction listener);
```

Passing a string parameter for parameterName is more convenient, but performance can be increased by storing a hash of the string and using that instead. You can generate a hash using Unity's `Animator.StringToHash (string input)` method. This is similar to the way you would access a parameter in an animator controller graph.

If you plan to access a parameter multiple times then the most efficient way to do so is to store a reference to the parameter itself. You can do this using the following methods:

```
IntParameter MotionGraphRoot.GetIntParameter (string parameterName);
IntParameter MotionGraphRoot.GetIntParameter (int hash);

FloatParameter MotionGraphRoot.GetFloatParameter (string parameterName);
FloatParameter MotionGraphRoot.GetFloatParameter (int hash);

SwitchParameter MotionGraphRoot.GetSwitchParameter (string parameterName);
SwitchParameter MotionGraphRoot.GetSwitchParameter (int hash);

TransformParameter MotionGraphRoot.GetTransformParameter (string parameterName);
TransformParameter MotionGraphRoot.GetTransformParameter (int hash);

TriggerParameter MotionGraphRoot.GetTriggerParameter (string parameterName);
TriggerParameter MotionGraphRoot.GetTriggerParameter (int hash);

VectorParameter MotionGraphRoot.GetVectorParameter (string parameterName);
VectorParameter MotionGraphRoot.GetVectorParameter (int hash);

EventParameter MotionGraphRoot.GetEventParameter (string parameterName);
EventParameter MotionGraphRoot.GetEventParameter (int hash);
```

This is also the way that the included motion graph elements reference parameters. If you plan to expand on the motion graph with your own custom elements then there are a number of GUI helpers available that simplify adding custom parameter and data selectors to inspectors. For more information see Extending the Motion Graph.

See Also

The Motion Graph

The Motion Graph Editor

Extending the Motion Graph

Motion Graph Ladders

Overview

Ladders have been an ubiquitous part of the first person shooter genre, with a huge range of implementations. Many FPS ladders are hacky workarounds for what is a surprisingly complex problem, and a poor implementation can often stand out.

NeoFPS attempts to set a standard for first person shooter ladders. They come in 2 flavours: contact ladders and interactive ladders, but both share the same underlying system.

Contact Ladders

Contact ladders are climbed automatically as soon as the player character touches them. They often allow the player to climb sideways across the ladder, dropping off if they pass the edge. They also allow the player to turn a full 360 degrees while on the ladder without any look constraints.

Contact ladders are often used in fast paced shooters where flow is much more important than realism.

For more information see the ContactLadder reference.

Interactive Ladders

Interactive ladders require the player to look at and use the ladder and attach to it. Once attached, the character is constrained to moving up and down the ladder until they reach the ends, let go (by interacting again) or jump off.

Interactive ladders are more restrictive than contact ladders, keeping the character fixed to one axis of movement. They can also constrain the camera, blocking the camera from turning too far away from the ladder if at all. Because of these features they are often used in more realistic games or in games where pacing needs to be more controlled.

For more information see the InteractiveLadder reference.

Ladder Implementation

In NeoFPS ladders have 2 elements. The ladder surface itself, and a wrap around at the top of the ladder.
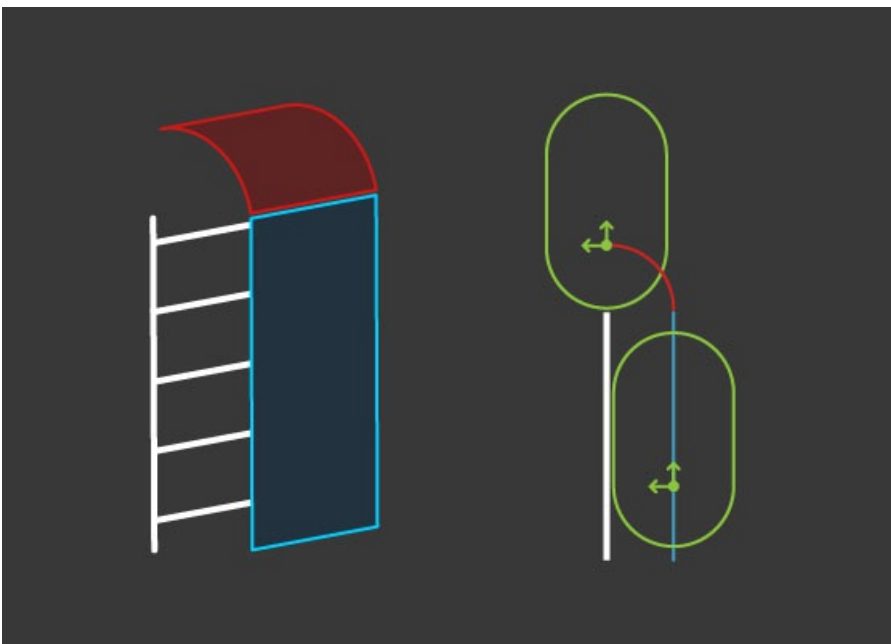
These sections are inflated from the ladder geometry by the character's radius. The system then moves a point at the bottom center of the character collider around in this "ladder space".
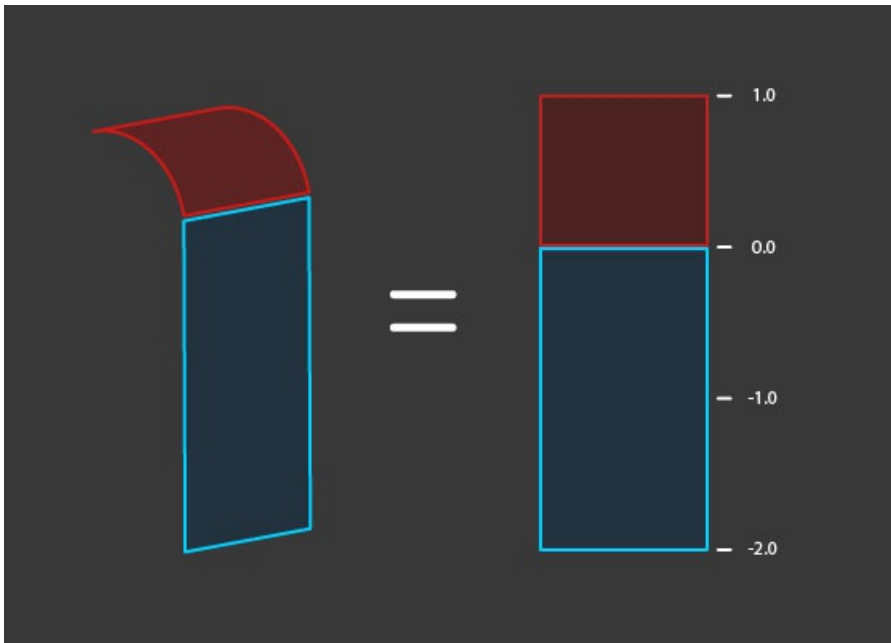
This system has 2 benefits: Firstly, it is easy to stick the character to the shape, making climbing consistent both part way up the ladder and also at the top of the ladder where some implementations can have difficulty registering contact. Secondly, the look direction can be judged in ladder space instead of world space.

By calculating look direction in ladder space, it means that a character at the top of a ladder and looking forwards is actually looking **up** the ladder. This makes changing climb direction based on aim much more intuitive than if the character had to look straight up in this situation. Another example is if the player approaches a ladder from the top. Walking forwards while looking at the edge will walk **down** the ladder. Looking away from the edge and slightly down will still be considered looking **up** the ladder in ladder space, so walking backwards will climb **down** the ladder as expected.
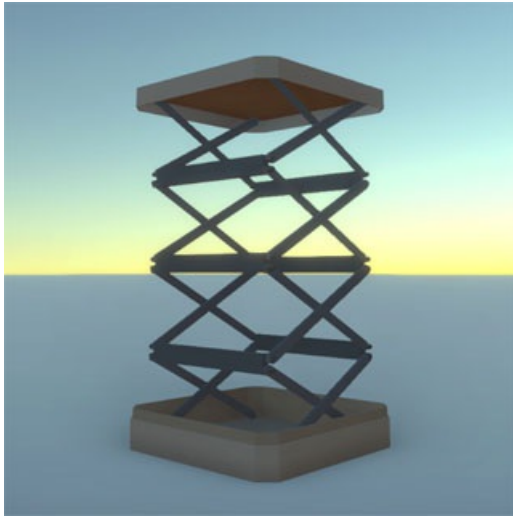
See Also

ContactLadder

InteractiveLadder

Unity CharacterController

Motion Graph Moving Platforms

Overview

A moving platform is a kinematic or dynamic rigidbody that the player can stand on. The player will move and rotate with the moving platform.

For a demonstration of the moving platforms, you can check out the sample scene.

How To Set Up A Moving Platform

Creating a moving platform is relatively simple. The key things that need setting up correctly are the rigidbody and the object layers.

1. Scene Heirarchy

▼ MovingPlatform
  Render

2. Moving Platform Object

🟦	☑ MovingPlatform	☐ Static ▼
	Tag Untagged	Layer MovingPlatforms

▼ ⚙ **Transform**   🔲 ⇄ ✿₊
Position   X 0   Y 0   Z 0
Rotation   X 0   Y 0   Z 0
Scale   X 1   Y 1   Z 1

▼ 🟩 ☑ **Box Collider**   🔲 ⇄ ✿₊
                        🔲 Edit Collider
Is Trigger   ☐
Material   None (Physic Material)   ◉
Center   X 0   Y 0   Z 0
Size   X 2   Y 0.5   Z 1

▼ ⚙ **Rigidbody**   🔲 ⇄ ✿₊
Mass   1
Drag   0
Angular Drag   0.05
Use Gravity   ☐
Is Kinematic   ☑
Interpolate   Interpolate   ▲
Collision Detection Discrete   ▲
▶ Constraints

▼ 📜 ☑ **Simple Moving Platform (Script)**   🔲 ⇄ ✿₊
Script   📄 SimpleMovingPlatform   ◉
Offset Position   X 0   Y 0   Z 0
Movement Duration 5
Pause Duration   5

3. Render Geometry

🟦	☑ Render	☐ Static ▼
	Tag Untagged	Layer EnvironmentDetail

▼ ⚙ **Transform**   🔲 ⇄ ✿₊
Position   X 0   Y 0   Z 0
Rotation   X 0   Y 0   Z 0
Scale   X 1   Y 1   Z 1

▼ 🟦 **Demo Facility_Prop_Moving Platform Horiz (Mesh Filter)** 🔲 ⇄ ✿₊
Mesh   🟦 DemoFacility_Prop_MovingPlatformHoriz   ◉

▼ 🟦 ☑ **Mesh Renderer**   🔲 ⇄ ✿₊
▼ Lighting
   Light Probes   Blend Probes   ▲
   Reflection Probes Simple   ▲
   Anchor Override   None (Transform)   ◉

   ❗ In Deferred Shading, all objects receive shadows and get per-pixel reflection probes.

   Cast Shadows   On   ▲
   Receive Shadows   ☑
   Motion Vectors   Per Object Motion   ▲
   Lightmap Static   ☐

   ❗ To enable generation of lightmaps for this Mesh Renderer, please enable the 'Lightmap Static' property.

▼ Materials
   Size   1
   Element 0   ⬤ MAT_DemoFacilityPalette   ◉
Dynamic Occluded   ☑

▼ 🟩 ☑ **Mesh Collider**   🔲 ⇄ ✿₊
Convex   ☐
   Is Trigger   ☐
Cooking Options   Mixed ...   ▲
Material   None (Physic Material)   ◉
Mesh   🟦 DemoFacility_Prop_MovingPlatformHoriz   ◉

⬤	MAT_DemoFacilityPalette	🔲 ⇄ ✿₊
	Shader Standard	▼

In the above image there are 2 objects. The first is the physical platform that the character interacts with. This uses one or more primitive colliders to define a rough shape and should be set to use the **Moving Platform** layer. It also requires an object that derives from the `IMovingPlatform` interface or `BaseMovingPlatform` abstract class. These communicate with the NeoCharacterController to smoothly move the character. Moving platform movement should always be handled through the **Rigidbody move and rotate methods**. NeoFPS relies on rigidbody interpolation to get smooth interpolation between the fixed update frames. You should use `Rigidbody.MovePosition()` and `Rigidbody.MoveRotation()` instead of setting the position and rotation directly. If you don't do this, then the platforms will appear to stutter when you ride them.

The second object is the render geometry. This should be set to use the **EnvironmentDetail** layer and the collider should match the render geometry as closely as possible.

When a character is in contact with the moving platform it will match any change of position or rotation of the platform each frame. Any character movement will be additive on top of this.

Example Moving Platform Behaviours

NeoFPS comes with the following moving platform types already implemented, though it is easy to add more with scripting:

NAME	DESCRIPTION
SimpleMovingPlatform	The SimpleMovingPlatform behaviour is used to create a platform that moves between 2 points at set intervals.
SimpleRotatingPlatform	The SimpleRotatingPlatform behaviour is used to create a platform that rotates at set intervals.
ConstantRotatingPlatform	The ConstantRotatingPlatform behaviour is used to create a platform that constantly turns at a set rate.
DrivenMovingPlatform	The DrivenMovingPlatform behaviour is added to a rigidbody that is driven by physics or a script and turns it into a moving platform. See the above note on rigidbody interpolation.
WaypointMovingPlatform	The WaypointMovingPlatform behaviour is used to create a platform that moves between a set of waypoints in sequence or directly.
ElevatorMovingPlatform	The ElevatorMovingPlatform behaviour is used to control the movement of an elevator cab. It is attached to the kinematic rigidbody that acts as the elevator cab or platform, and moves the character smoothly with it.

See Also

NeoCharacterController

The Motion Graph

The Motion Graph Editor

Motion Graph Swimming

Overview

Swimming in NeoFPS is broken down into 3 sets of states: surface, underwater, and wading. The motion graph has the flexibility to craft a range of swimming movement styles using any of those states, while controlling things like capsule height and camera effects using the various graph behaviours.

Water Zones

Water zones are used in the scene to define a volume that the character can swim in. The BasicWaterZone behaviour is placed on an object with a trigger box collider and will notify any character that enters it.

The water zone provides the motion graph with details of the water surface height and normal, as well as the flow velocity, at a specific point.

The BasicWaterZone simply returns the top face of the attached box collider, along with a constant flow vector. More complex water zone behaviours can be created by inheriting from the `IWaterZone` interface. This requires the following methods:

```
Vector3 FlowAtPosition(Vector3 position);
WaterSurfaceInfo SurfaceInfoAtPosition(Vector3 position);
```

Surface Swimming

The surface swimming states will try to keep the character's head at a certain height above the water surface. If the surface level rises too fast, or the character is already moving up/down too quickly, then the head can breach the water's surface. In this case, you can use the Water MotionGraphCondition to check the submerged depth of the character and transition into an underwater state.

The available surface swimming states are:

- SwimSmoothSurfaceState which moves smoothly along the water surface
- SwimStrokeSurfaceState which moves in pulses to simulate swimming strokes. The pulse frequency is based on movement direction.

Underwater Swimming

The underwater swimming states move in the direction of the character's aim (including pitch), while jump and crouch are used to move straight up and down.

The available underwater swimming states are:

- SwimSmoothUnderwaterState which moves smoothly
- SwimStrokeUnderwaterState which moves in pulses to simulate swimming strokes. The pulse frequency is based on movement direction, while jump/crouch driven movement is smooth.

Wading

The Wading state is a variation of the Movement state which is used for walking and swimming. It changes the target speed based on how much of the character is below the water surface.

Drowning

Drowning can be implemented simply by recording the time spend in underwater states in a float parameter on the graph and acting based on its value. The DrowningMotionGraphWatcher monobehaviour is attached to the character and applies damage once a parameter reaches a certain value and at set increments therafter.

Future Development

The current implementation of swimming in NeoFPS should be considered version 1. Version 2 will extend this with extra features and behaviour such as:

- Swimming hand animations
- Options to prevent shooting while underwater
- Audio effects
- Waves
- Better underwater visuals and post-processing
- Bouyancy and bouyant objects
- Better handling of multiple overlapping water zones

See Also

The Motion Graph

The Motion Graph Editor

Motion Debugger

Overview

The motion debugger is use to help diagnose unexpected and unwanted movement behaviour by the NeoFPS Motion Graph. It allows you to chart movement details in real time, pause, and focus in on a specific moment to inspect the input and output of the motion graph and NeoCharacterController

Accessing The Motion Debugger

The motion debugger can be found in the menus at *Tools/NeoFPS/Motion Debugger* or via the **Attach Debugger** button on the [MotionController][3] component attached to the character. If the character is in the scene and the game is playing then this button will attach the selected character to the debugger and start recording its movement data. Alternatively, you can select **Attach Automatically** at the top of the motion debugger, and it will attach to the local player character whenever it spawns.

Each time a new character is attached to the debugger, it resets the recorded data, so if the problem movement results in the death of the character then you will either need to pause the game before the character respawns, or disable the automatic attachment.

Output Graph

The graph will chart the value selected in the **Graph Readout** dropdown just above. Values are recorded regardless of whether

they are displayed, so changing this value does not reset the graph.

The vertical axes of the graph will resize dynamically based on the minimum and maximum value recorded in the time range. These min and max values are displayed to one side of the graph.

If the game is paused or play mode has ended, then you can click and drag on the graph to inspect a specific frame. The selected frame number along with the charted value at that point will be displayed alongside a vertical marker showing your position on the graph.

Scene View Ghost
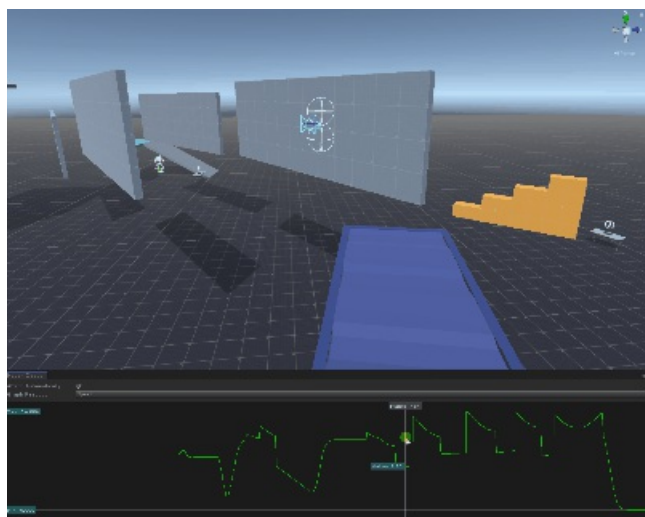
Scrubbing the graph while the game is paused will show a representation of the character at that frame in the scene view. You can use this to help find the exact point that the character did something unexpected, and then use the readout values to diagnose the problem. Due to the way that Unity draws object gizmos in the scene, this can only be done in pause mode and not after exiting play mode to editor mode, though the recorded values will persist until a new character is attached to the debugger.

Readout

Underneath the graph is a readout of the various variables for the current frame and previous frame. You can filter which values are visible using the **Show Details** dropdown at the top of the list.

Output

The outputs are the values the motion graph sent to the NeoCharacterController on the inspected frame. This includes the state, target move vector and whether gravity and ground snapping should be applied.

Inputs

These are the variables the NeoCharacterController provided to the motion graph to base its decisions from. Most of these are the result of the previous frame's movement.

Parameters

These are the values of the motion graph parameters at the point the output was sent to the NeoCharacterController. These can help diagnose which of the conditions were valid that frame, though bear in mind that these can be modifed by the conditions and behaviours on the graph, as well as external sources.

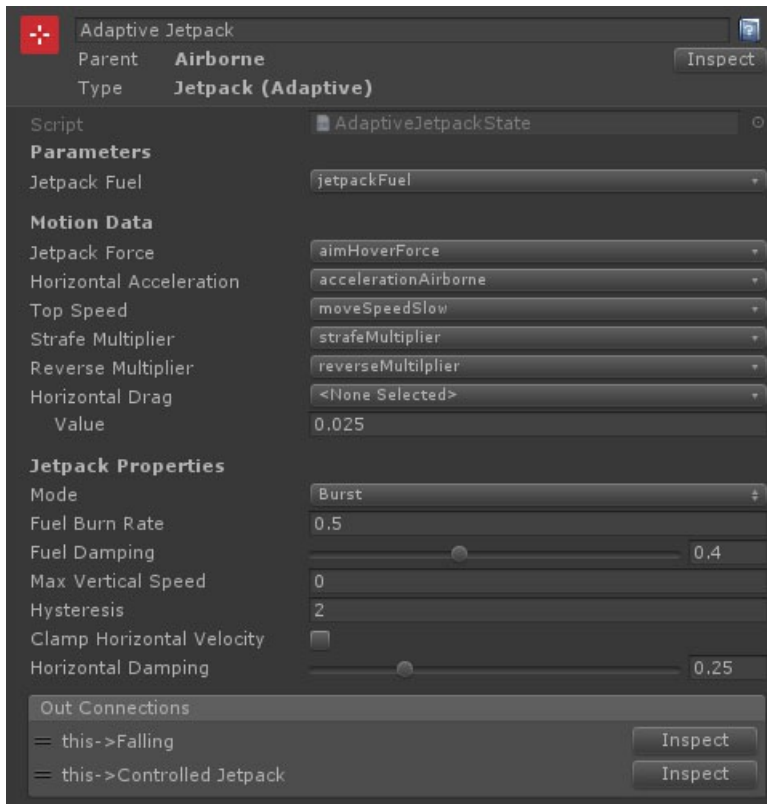## See Also

ContactLadder

InteractiveLadder

AdaptiveJetpack MotionGraphState

Overview

The AdaptiveJetpack state adds an upward force to the character while allowing some control over direction.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Jetpack Fuel	FloatParameter	An optional parameter for the jetpack fuel. Will be consumed based on output.
Jetpack Force	FloatData	An acceleration force (ignores mass) upwards for the jetpack
Horizontal Acceleration	Float Data	The input driven acceleration while falling. This can either be set as a value, or by referencing a float data entry.
Top Speed	Float Data	The top horizontal movement speed (for keyboard input or max analog input). This can either be set as a value, or by referencing a float data entry.
Strafe Multiplier	Float Data	The multiplier applied to the max movement speed when strafing. This can either be set as a value, or by referencing a float data entry.
Reverse Multiplier	Float Data	The multiplier applied to the max movement speed when moving in reverse. This can either be set as a value, or by referencing a float data entry.
Mode	Dropdown	How the jetpack should work. **Smooth** reduces power as approaching the max vertical speed. **Burst** sets a minimum jetpack burst duration and gap between bursts to create a bouncier hover.

NAME	TYPE	DESCRIPTION
m_MaxVerticalSpeed	Float	The maximum vertical speed, at which the jetpack stops pushing upwards. A speed of zero is hovering.
m_SpeedFalloff	Float	The speed below the max at which jetpack power (and fuel consumption) starts to fall off. The power will fade out exponentially the closer you get to the max speed. Only shown if the "Mode" dropdown is set to **Smooth**.
m_Hysteresis	Float	A speed differential where the jetpack will switch on/off. Max + half this = off. Max - half this = on. Only shown if the "Mode" dropdown is set to **Burst**.
m_FuelBurnRate	Float	The amount of fuel burned per second at full burn. Only shown if the "Jetpack Fuel" parameter is not null.
m_FuelDamping	Float	A damping amount for the fuel consumption to smooth it out. Set to zero for direct feedback. Only shown if the "Jetpack Fuel" parameter is not null.
m_MinFuelBurn	Float	The fuel burn rate when at the target speed, as opposed to accelerating towards it. Only shown if the "Jetpack Fuel" parameter is not null.
Clamp Speed	Boolean	Should the speed of the character decelerate to top speed.
Damping	Float	The amount of damping to apply when changing direction.

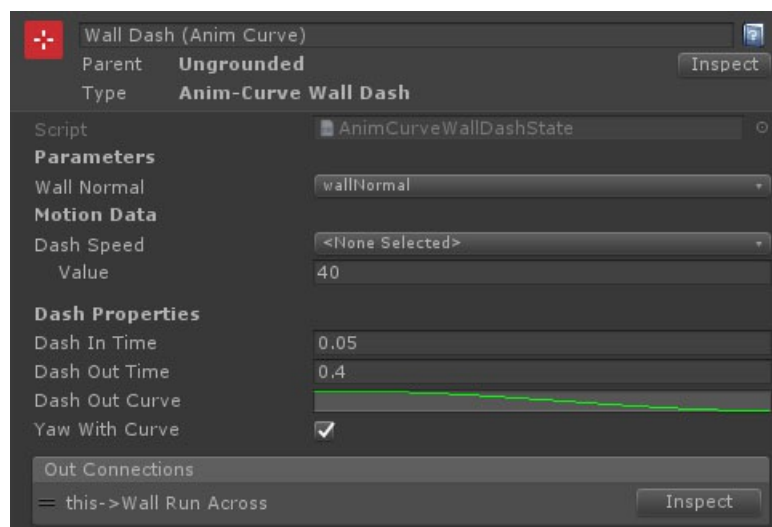## See Also

[The Motion Graph](#)

[The Motion Graph Editor](#)

[Motion Graph States](#)

AnimCurveDash MotionGraphState

Overview

The AnimCurveDash state is a simple movement state that layers a directional dash with speed based on an animation curve, on top of a basic movement system. This gives a lot of control over the feel of the dash, and also allows the player to have some control over the exit speed and dash distance.

Inspector
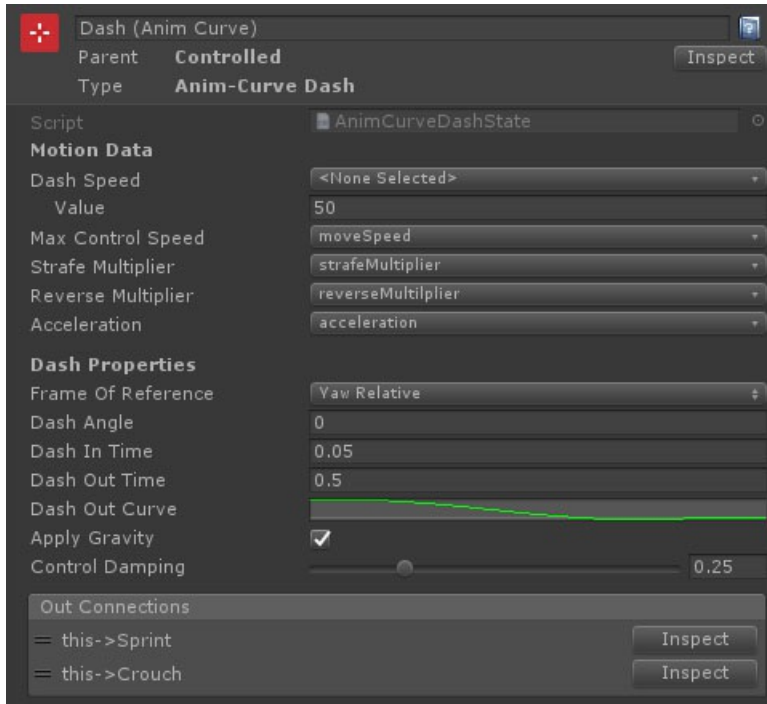
Properties

NAME	TYPE	DESCRIPTION
Dash Speed	FloatData	The target speed for the dash to reach. This will be layered on top of the control speed.
m_MaxControlSpeed	FloatData	The maximum speed the character can reach under motor control (driven by input). The dash velocity will be layered on top of this.
Strafe Multiplier	Float Data	The multiplier applied to the max movement speed when strafing. This can either be set as a value, or by referencing a float data entry.
Reverse Multiplier	Float Data	The multiplier applied to the max movement speed when moving in reverse. This can either be set as a value, or by referencing a float data entry.
Acceleration	Float Data	The input driven acceleration.
Dash Direction	Dropdown	The direction to base the dash off. This can be **Yaw Relative** or **Move Relative**.
Dash Angle	Float	The angle offset for the dash direction. For example, yaw relative and an angle of 90 will dash to the right. -90 will dash to the left.
Dash In Time	Float	The amount of time it takes to reach the dash speed. At this point, the animation curve kicks in to ease out of the dash. A Dash In Time of 0 is instant.

NAME	TYPE	DESCRIPTION
Dash Out Time	Float	The amount of time it takes for the animation curve kicks to ease out of the dash.
Dash Out Curve	Animation Curve	The ease out curve for the dash velocity. This should start at 1. Dipping below zero will mean the dash is moving backwards.
Apply Gravity	Boolean	Should the character fall with gravity during the dash.
Control Damping	Float	The amount of damping to apply to the controlled velocity when changing direction.

See Also

The Motion Graph

The Motion Graph Editor

Motion Graph States

AnimCurveWallDash MotionGraphState

Overview

The AnimCurveWallDash state is a simple movement state that layers a directional dash with speed based on an animation curve, on top of a basic movement system. This gives a lot of control over the feel of the dash, and also allows the player to have some control over the exit speed and dash distance.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Dash Speed	FloatData	The target speed for the dash to reach. This will be layered on top of the control speed.
Wall Normal	Vector Parameter	The wall normal parameter, as used by the wall run states.
Dash In Time	Float	The amount of time it takes to reach the dash speed. At this point, the animation curve kicks in to ease out of the dash. A Dash In Time of 0 is instant.
Dash Out Time	Float	The amount of time it takes for the animation curve kicks to ease out of the dash.
Dash Out Curve	[Animation Curve] [unity-animationcurve]	The ease out curve for the dash velocity. This should start at 1. Dipping below zero will mean the dash is moving backwards.
Yaw With Curve	Boolean	Should the yaw direction of the character be turned if the character must change directions to curve with the wall. It's easy to become disoriented with this disabled.

See Also

The Motion Graph

The Motion Graph Editor

Motion Graph States

BoostPad MotionGraphState

Overview

The BoostPad state is a simple one frame movement state which sets the character's velocity to the velocity vector scaled.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Boost Vector	VectorParameter	The movement velocity to apply.
Boost Mode	Dropdown	How the boost vector is applied to the character. Options are: **Absolute** sets the character velocity, **Additive** adds the boost to the character velocity, **MaintainPerpendicular** sets the velocity in the direction of the boost, but keeps any velocity along the plane perpendicular to the boost.
Multiplier	FloatData	A multiplier for the movement velocity.

See Also

The Motion Graph

The Motion Graph Editor

Motion Graph States

[MotionControllerData ScriptableObject][6]

ConstantMove MotionGraphState
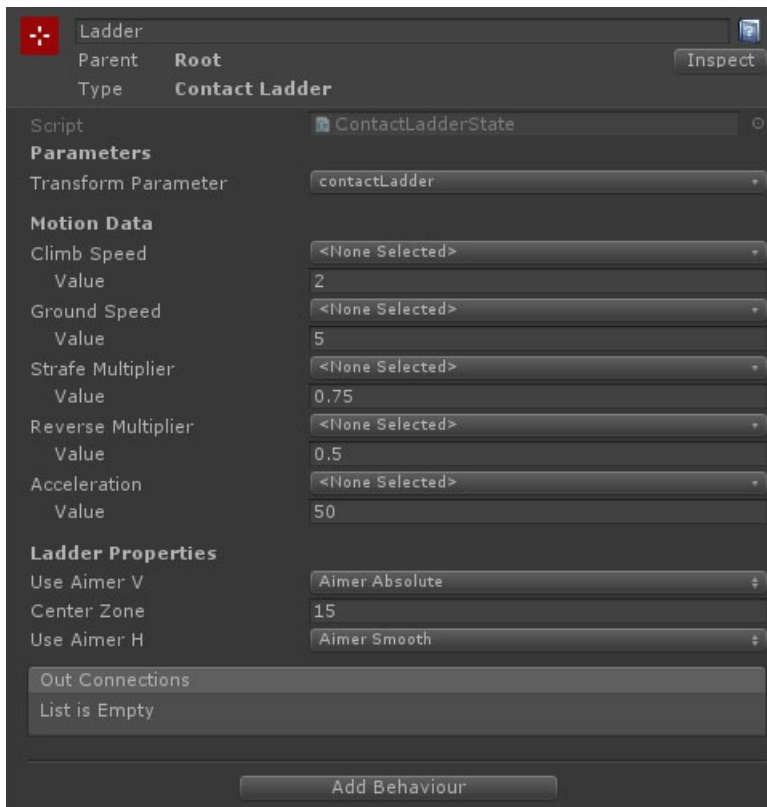
Overview

The ConstantMove state accelerates to a set velocity and then maintains it indefinitely.

Inspector
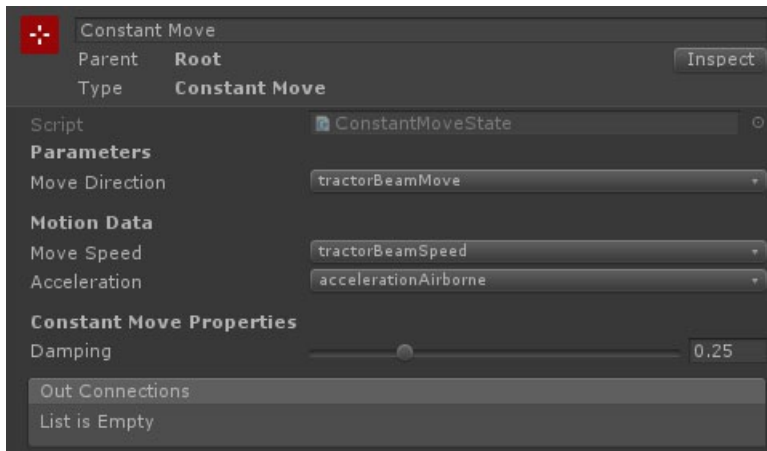
Properties

NAME	TYPE	DESCRIPTION
Move Direction	VectorParameter	A vector parameter used to define the direction in world space.
Move Speed	FloatData	The target movement speed. This can also be negative to move back along the direction vector.
Acceleration	FloatData	The maximum acceleration.
Damping	Float	The amount of damping to apply when changing direction or speed.

See Also

The Motion Graph

The Motion Graph Editor

Motion Graph States

[MotionControllerData ScriptableObject][6]

ContactLadder MotionGraphState

Overview

The contact ladder state handles climbing up and down physics ladders. As soon as the character walks into one of these ladders, the ladder TransformProperty will be set. You should set up an in-bound connection to this state with a TransformCondition that checks if the property is not null, and an outbound connection with another that checks if the property is null.

The character can exit the ladder by jumping or moving past the ladder limits (horizontal and vertical). Doing either of these will set the ladder TransformProperty to null and the state completed flag to true.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Transform Parameter	TransformParameter	The transform parameter on the graph that is used to attach to ladder transforms in the scene.
Climb Speed	Float Data	The maximum climb speed (based on input and aim). This can either be set as a value, or by referencing a float data entry.
Ground Speed	Float Data	The top movement speed with ground under foot (top and bottom of the ladder). This can either be set as a value, or by referencing a float data entry.
Strafe Multiplier	Float Data	The multiplier applied to the max movement speed when strafing. This can either be set as a value, or by referencing a float data entry.
Reverse Multiplier	Float Data	The multiplier applied to the max movement speed when moving in reverse. This can either be set as a value, or by referencing a float data entry.
Acceleration	Float Data	The maximum acceleration. This can either be set as a value, or by referencing a float data entry.

NAME	TYPE	DESCRIPTION
Use Aimer V	Dropdown	Should the direction of the camera aim influence the vertical move direction? • **IgnoreAimer** = No • **AimerAbsolute** = Direction of aim changes direction of move • **AimerSmooth** = Direction of aim changes direction and speed of move • **AimerHeading** = Direction and speed based on yaw only (factors in both input axes) • **AimerAllAxes** = Direction and speed based on yaw and pitch (factors in both input axes)
Center Zone	Float	For **AimerAbsolute**, the angle past the horizontal you need to aim to flip directions. For **AimerSmooth** or **AimerAllAxes**, the angle past the horizontal that reaches full speed.
Use Aimer H	Dropdown	Should the camera aim influence the horizontal move direction? • **IgnoreAimer** = No • **AimerAbsolute** = Direction of aim changes direction of move • **AimerSmooth** = Direction of aim changes direction and speed of move • **AimerHeading** = Direction and speed based on yaw only (factors in both input axes) • **AimerAllAxes** = Direction and speed based on yaw and pitch (factors in both input axes)

See Also

The Motion Graph

Motion Graph States

Motion Graph Parameters And Data

ControlledJetpack MotionGraphState

Overview

The ControlledJetpack state adds an upward force to the character while allowing some control over direction.

Inspector
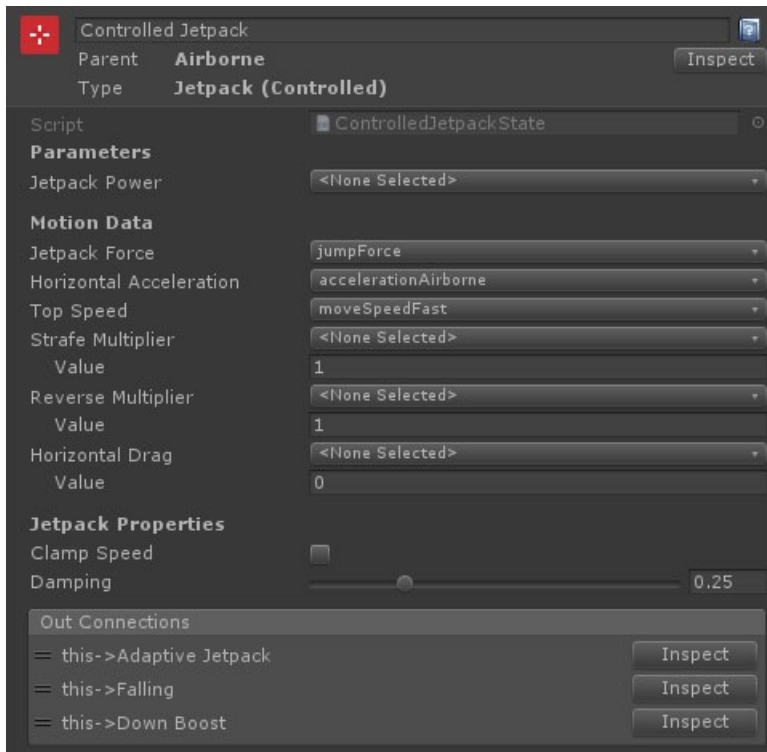
Properties

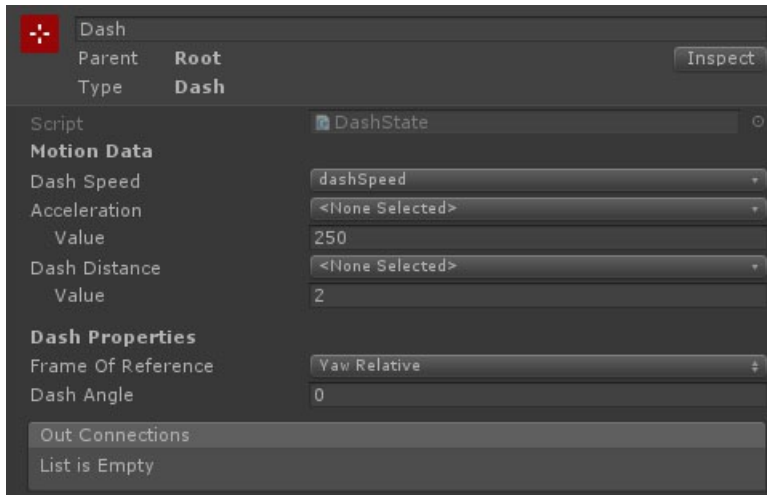| NAME	TYPE	DESCRIPTION
Jetpack Power	FloatParameter	An optional parameter that acts as a multiplier for the jetpack force (0.5 is half power, etc)..
Jetpack Force	FloatData	An acceleration force (ignores mass) upwards for the jetpack
Horizontal Acceleration	Float Data	The input driven acceleration while falling. This can either be set as a value, or by referencing a float data entry.
Top Speed	Float Data	The top horizontal movement speed (for keyboard input or max analog input). This can either be set as a value, or by referencing a float data entry.
Strafe Multiplier	Float Data	The multiplier applied to the max movement speed when strafing. This can either be set as a value, or by referencing a float data entry.
Reverse Multiplier	Float Data	The multiplier applied to the max movement speed when moving in reverse. This can either be set as a value, or by referencing a float data entry.
Clamp Speed	Boolean	Should the speed of the character decelerate to top speed.
Damping	Float	The amount of damping to apply when changing direction.

See Also

Dash MotionGraphState

Overview

The Dash state is a simple movement state that surges in a specific direction. The direction can be based off the character's yaw forward or the direction it is currently moving, along with an angle offset.

Once the dash has attempted to move a specific distance it will signal completion.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Dash Speed	FloatData	The target speed for the dash to reach.
Dash Acceleration	FloatData	The acceleration to reach the target dash speed.
Dash Distance	FloatData	The distance to dash before the state completes.
Dash Direction	Dropdown	The direction to base the dash off. This can be **Yaw Relative** or **Move Relative**.
Dash Angle	Float	The angle offset for the dash direction. For example, yaw relative and an angle of 90 will dash to the right. -90 will dash to the left.

See Also

The Motion Graph

The Motion Graph Editor

Motion Graph States

Dodge MotionGraphState

Overview

The dodge state adds a simple directional impulse to the character for a single tick and then sets its completed flag. You should add an out-bound connection from this state that uses the CompletedCondition, connecting to an airborne state.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Vertical Speed	Float Data	The vertical dodge speed. This can either be set as a value, or by referencing a float data entry.
Horizontal Speed	Float Data	The horizontal dodge speed. This can either be set as a value, or by referencing a float data entry.
Dodge Direction Parameter	IntParameter	The dodge direction is a simple compass heading relative to the character's forward direction. It is set in one of the graph properties and conforms to the following: 1. North 2. North East 3. East 4. South East 5. South 6. South West 7. West 8. North West

See Also

The Motion Graph

Motion Graph States

Motion Graph Parameters And Data

Falling MotionGraphState

Overview

The falling state behaves as the name suggests, but provides some level of control when in the air.

Inspector

Properties

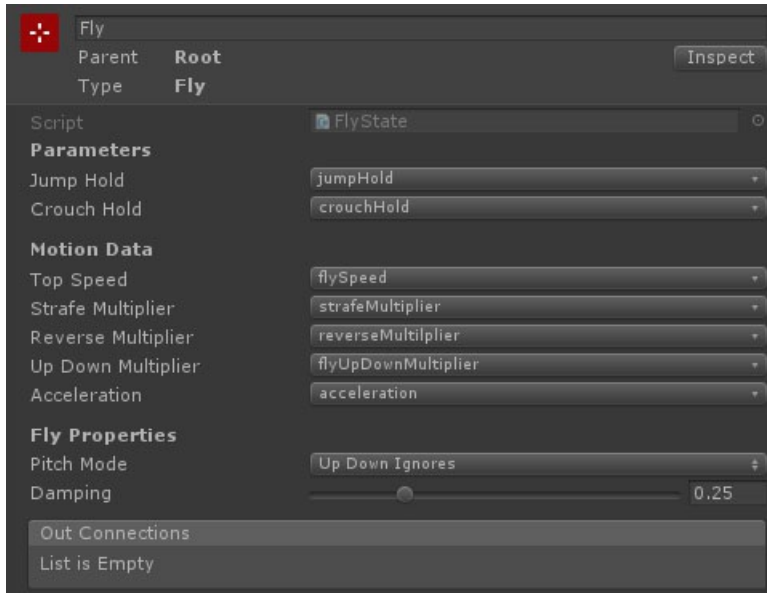| NAME	TYPE	DESCRIPTION
Horizontal Acceleration	Float Data	The input driven acceleration while falling. This can either be set as a value, or by referencing a float data entry.
Top Speed	Float Data	The top horizontal movement speed (for keyboard input or max analog input). This can either be set as a value, or by referencing a float data entry.
Strafe Multiplier	Float Data	The multiplier applied to the max movement speed when strafing. This can either be set as a value, or by referencing a float data entry.
Reverse Multiplier	Float Data	The multiplier applied to the max movement speed when moving in reverse. This can either be set as a value, or by referencing a float data entry.
Clamp Speed	Boolean	Should the speed of the character decelerate to top speed.
Damping	Float	The amount of damping to apply when changing direction.

See Also

The Motion Graph

Motion Graph States

Motion Graph Parameters And Data

Fly MotionGraphState

Overview

The Fly state is a simple noclip style flying style. The character moves based on their aim direction. Jump moves up, while crouch moves down.

Inspector

Properties

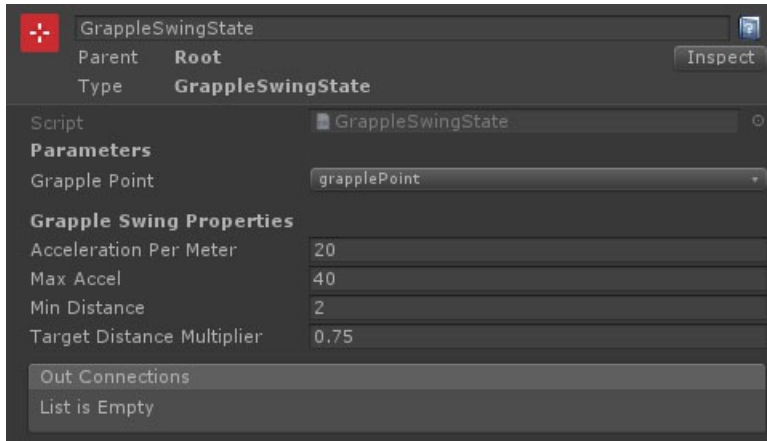| NAME	TYPE	DESCRIPTION
Jump Hold	SwitchParameter	The crouch hold parameter (used for flying up).
Crouch Hold	SwitchParameter	The crouch hold parameter (used for flying down).
Top Speed	FloatData	The top movement speed (for keyboard input or max analog input).
Strafe Multiplier	FloatData	The multiplier applied to the max movement speed when strafing.
Reverse Multiplier	FloatData	The multiplier applied to the max movement speed when moving backwards.
Up Down Multiplier	FloatData	The multiplier applied to the max movement speed when moving up/down.
Acceleration	FloatData	The maximum acceleration.
Pitch Mode	Dropdown	How does the camera pitch affect the movement. The available options are: • **Up Down Ignores** means jump/crouch move straight up or down. • **Affects All Axes** means that up/down are along the camera up axis. • **Ignore** only takes yaw into account for all movement
Damping	Float	The amount of damping to apply when changing direction or speed.

See Also

[The Motion Graph](#)

GrappleSwing MotionGraphState

Overview

The GrappleSwing state pulls the character towards a grapple point and swings as though tethered to it with a springy rope. One option for setting the grapple point is by using the GrappleToolModule with the wieldable tools system

Inspector

Properties

NAME	TYPE	DESCRIPTION
Grapple Point	VectorParameter	The point in space that the grapple is tethered to.
Target Distance Multiplier	Float	When entering the state, a target distance will be calculated based on multiplying the current distance to the grapple point by this multiplier.
Min Distance	Float	Below this distance from the grapple point, the character will actually be pushed away.
Acceleration Per Meter	Float	The acceleration towards the grapple point per meter distance above the target distance.
Max Accel	Float	The maximum acceleration towards the grapple point.

See Also

The Motion Graph

The Motion Graph Editor

Motion Graph States

GrappleToolModule

Impulse MotionGraphState

Overview

The Impulse state is a one frame state that adds or sets the character velocity. An example use is at the start of a crouch-slide to add an instantaneous speed boost.

Inspector
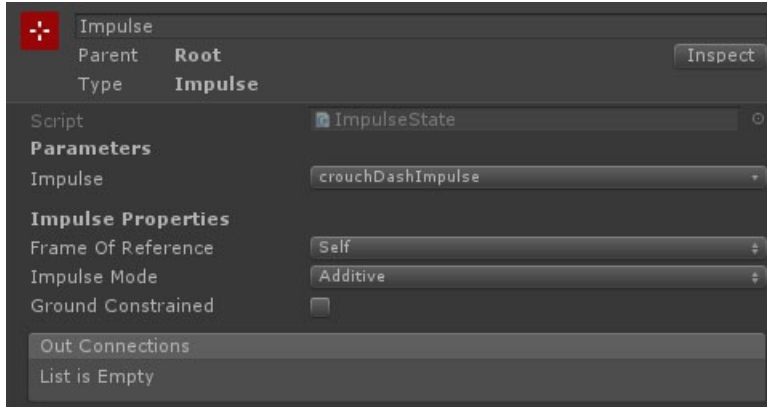
Properties

NAME	TYPE	DESCRIPTION
Impulse	VectorParameter	The velocity impulse to apply.
Frame Of Reference	Dropdown	The coordinate space to apply the impulse in. Options are **Self** and **World**.
Impulse Mode	Dropdown	How should the impulse be applied. **Additive** will add the impulse velocity to the original velocity. **Replace Velocity** will ignore the original velocity and use the impulse alone.
Ground Constrained	Boolean	If true, the impulse vector will be aligned onto the ground surface normal plane.

See Also

The Motion Graph

The Motion Graph Editor

Motion Graph States

InteractiveLadder MotionGraphState

Overview

The InteractiveLadder state handles climbing up and down interactive ladders. These are ladders are simple obstacles until interacted with, at which point the ladder TransformProperty will be set. You should set up an in-bound connection to this state with a TransformCondition that checks if the property is not null, and an outbound connection with another that checks if the property is null.

The character can exit the ladder by interacting with it again, jumping, or moving past the ladder limits. Doing any of these will set the ladder TransformProperty to null and the state completed flag to true.

The state has various properties for how the character climbs the ladder, along with camera constraints.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Transform Parameter	TransformParameter	The transform parameter on the graph that is used to attach to ladder transforms in the scene.
Climb Speed	Float Data	The maximum climb speed (based on input and aim). This can either be set as a value, or by referencing a float data entry.
Acceleration	Float Data	The acceleration when on the ladder or attching / dismounting. This can either be set as a value, or by referencing a float data entry.
Use Aimer V	Dropdown	Should the direction of the camera aim influence the vertical move direction? • **IgnoreAimer** = No • **AimerUpDown** = Only direction • **AimerSmooth** = Direction and speed
Center Zone	Float	For **AimerUpDown**, the angle past the horizontal you need to aim to flip directions. For **AimerSmooth**, the angle past the horizontal that reaches full speed.

NAME	TYPE	DESCRIPTION
Dismount Delay	Float	How long is the character blocked from stepping straight off the ladder using up/down? For example, if they attach to the top of the ladder then they should not immediately step off by pressing up.
Constrain Camera	Boolean	If true, constrain the camera's horizontal rotation to a maximum range from looking directly at the ladder.
Look Range	Float	The angle range (degrees) that the camera can turn from looking straight at the ladder (180 = 90 degrees to either side).

See Also

The Motion Graph

Motion Graph States

Motion Graph Parameters And Data

Jetpack MotionGraphState

Overview

The Jetpack state maintains horizontal velocity and adds an upward force.

Inspector
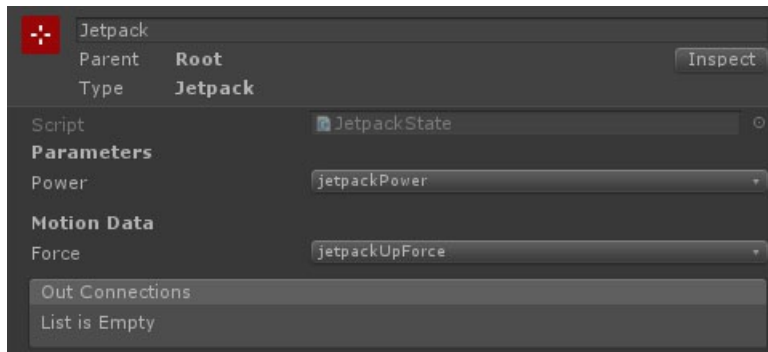
Properties

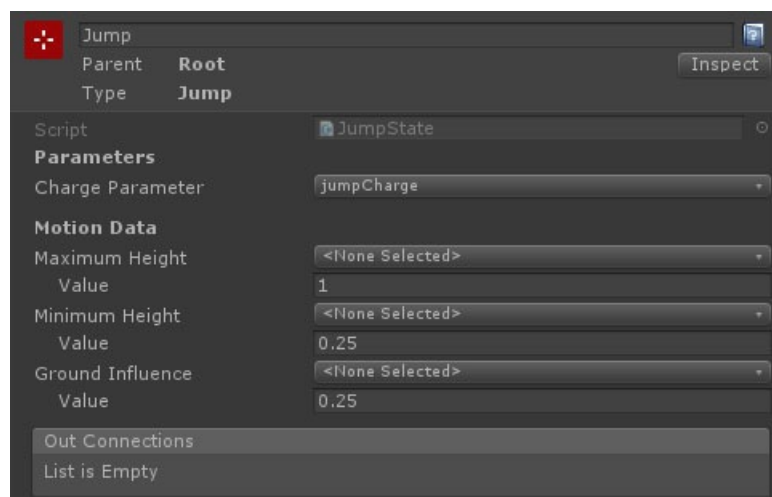| NAME	TYPE	DESCRIPTION
Power	FloatParameter	An optional parameter that acts as a multiplier for the jetpack force (0.5 is half power, etc).
Force	FloatData	An acceleration force (ignores mass) upwards for the jetpack

See Also

The Motion Graph

The Motion Graph Editor

Motion Graph States

Jump MotionGraphState

Overview

The jump state adds a simple upward impulse to the character for a single tick and then sets its completed flag. You should add an out-bound connection from this state that uses the CompletedCondition, connecting to an airborne state.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Charge Parameter	FloatParameter	The charge property is a float with value 0 to 1 that defines how high to jump. If no graph property is selected, the jump will always be performed with full strength.
Maximum Height	Float Data	The height the player will jump when fully charged. Note: Changes to gravity or physics multiplier after start will affect this. Also does not take step height into account. This can either be set as a value, or by referencing a float data entry.
Minimum Height	Float Data	The smallest height the player will jump (at the equivalent of a zero tap - actually unattainable). This can either be set as a value, or by referencing a float data entry.
Ground Influence	Float Data	The amount of influence the ground has over the direction of the jump. 0 = up, 1 = ground normal. This can either be set as a value, or by referencing a float data entry.

See Also

The Motion Graph

Motion Graph States

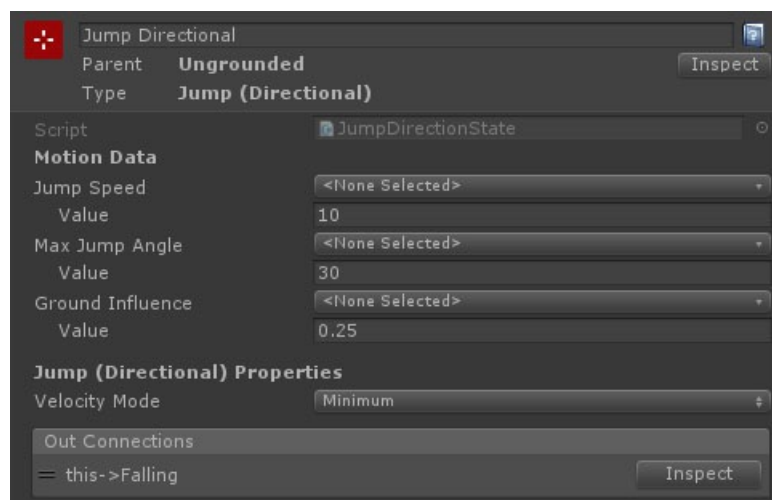Motion Graph Parameters And Data

JumpDirection MotionGraphState

Overview

The jump direction state adds a jump direction to the Jump state. You should add an out-bound connection from this state that uses the CompletedCondition, connecting to an airborne state. This state should be replaced with the JumpDirectionV2 motion graph state instead, as it gives more predictable results with analogue controllers.

Inspector

Properties

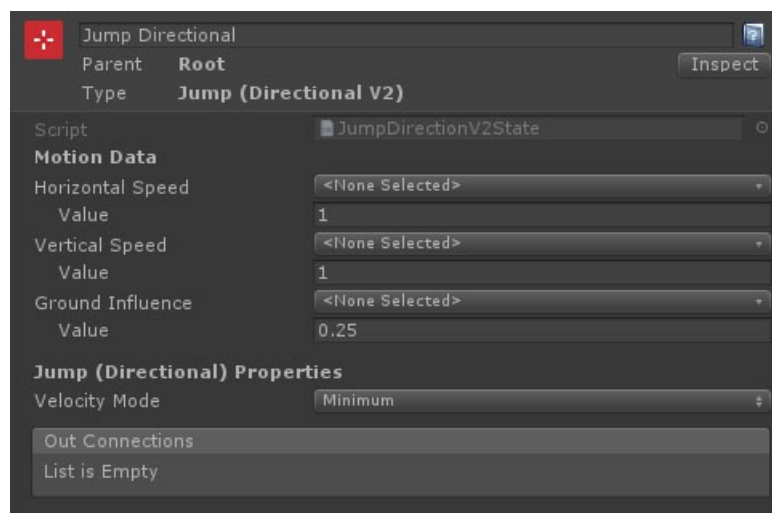| NAME	TYPE	DESCRIPTION
Jump Speed	Float Data	The instant speed of the jump.
Maximum Height	Float Data	The angle the jump direction will be tilted in the character input direction when input scale is 1.
Ground Influence	Float Data	The amount of influence the ground has over the direction of the jump. 0 = up, 1 = ground normal.
Velocity Mode	Dropdown	How should the velocity be applied. Available options are: • **Additive** will add the jump velocity to the original velocity. • **Absolute** will ignore the original velocity. • **Minimum** will boost the character velocity if it is less than the jump speed in the jump direction.

See Also

The Motion Graph

Motion Graph States

Motion Graph Parameters And Data

JumpDirectionV2 MotionGraphState

Overview

The JumpDirectionV2 state adds a jump direction to the Jump state. This version gives a more consistent result with analogue controllers than the original JumpDirection motion graph state and should be used in its place.

Inspector

Properties

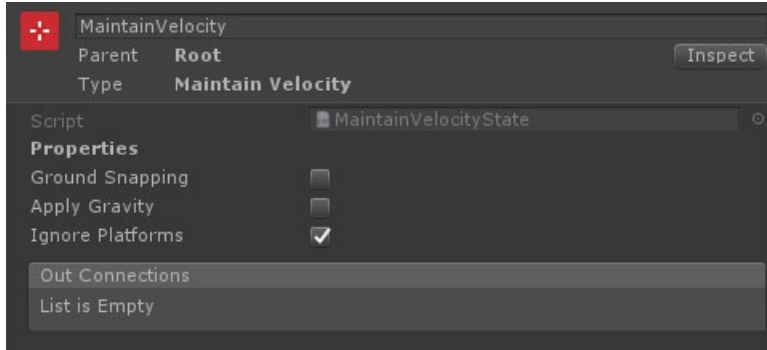| NAME	TYPE	DESCRIPTION
Horizontal Speed	Float Data	The horizontal speed of the jump.
Vertical Speed	Float Data	The upward speed of the jump.
Ground Influence	Float Data	The amount of influence the ground has over the direction of the jump. 0 = up, 1 = ground normal.
Velocity Mode	Dropdown	How should the velocity be applied. Available options are: • **Additive** will add the jump velocity to the original velocity. • **Absolute** will ignore the original velocity. • **Minimum** will boost the character velocity if it is less than the jump speed in the jump direction.

See Also

The Motion Graph

Motion Graph States

Motion Graph Parameters And Data

MaintainVelocity MotionGraphState

Overview

The MaintainVelocity state is a utility state that simply moves the character at the same velocity as last frame. This will be affected by collisions.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Ground Snapping	Boolean	Should ground snapping be applied.
Apply Gravity	Boolean	Should gravity force be applied.
Ignore Platforms	Boolean	Should the character inherit movement from platforms it's touching.

See Also

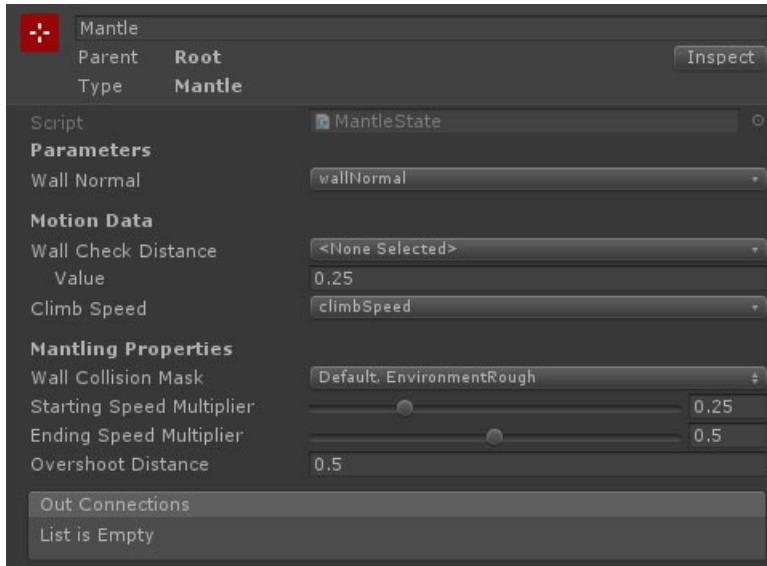[The Motion Graph](#)

[The Motion Graph Editor](#)

[Motion Graph States](#)

Mantle MotionGraphState

Overview

The Mantle state is used to climb a wall ledge onto its top surface. It is paired with the Climbable condition to test if a wall impact is climbable.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Wall Normal	VectorParameter	The normal of the wall to climb. This value will be read from **and** written to each frame.
Wall Check Distance	FloatData	The cast distance for the initial wall check. Using a motion data entry allows the value to be shared with the Climbable condition.
Climb Speed	FloatData	The movement speed while climbing the surface.
Wall Collision Mask	LayerMask	The collision mask to use when checking the wall normal.
Starting Speed Multiplier	Float	The climb speed multiplier (for the data value above) on entering the state.
Ending Speed Multiplier	Float	The climb speed multiplier (for the data value above) on completing the ledge mantle.
Overshoot Distance	Float	The distance to move past the edge onto flat ground before completing.

See Also

The Motion Graph

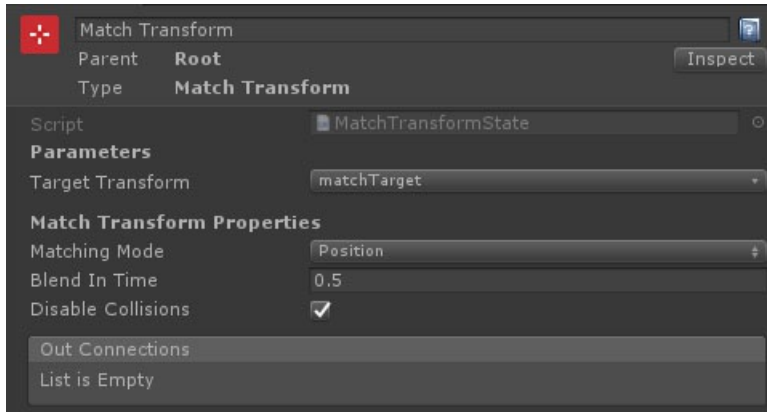The Motion Graph Editor

Motion Graph States

MatchTransform MotionGraphState

Overview

The MatchTransform state makes the character move as though it is attached to a transform in the scene. This allows you to sync the character to animated cut-scenes or create complex on-rails movement systems for certain situations.

Inspector

Properties

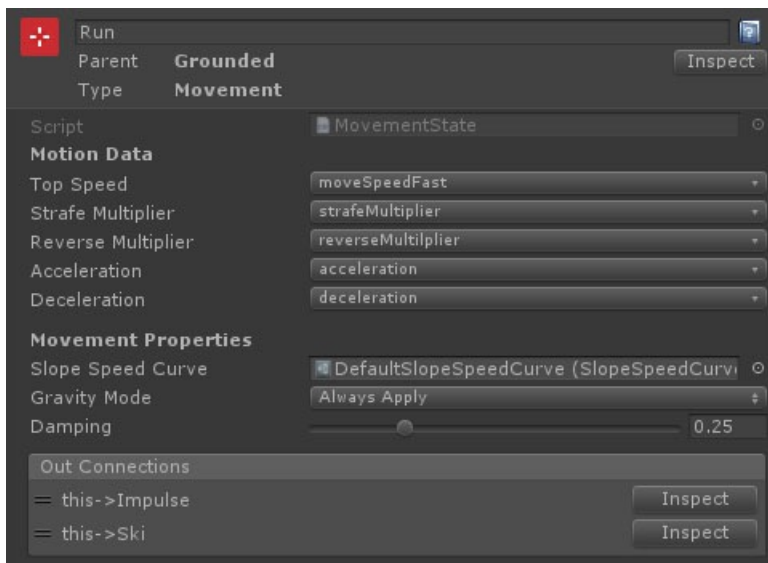| NAME	TYPE	DESCRIPTION
Target Transform	Transform Parameter	The transform to match to.
Matching Mode	Dropdown	What transform components to match. Options are: **Position** (match the position but ignore rotation), **PositionAndUp** (match the position and keep the character up-vector aligned but ignore yaw), **PositionAndDirection** (match position and yaw direction or heading, but maintain the character's up vector), and **All** (match position, up vector and heading).
Blend In Time	Float	The time taken to blend from current position and velocity to match the transform.
Disable Collisions	Boolean	Should collisions be disabled for the duration of the movement.

See Also

The Motion Graph

Motion Graph States

Motion Graph Parameters And Data

Movement MotionGraphState

Overview

The movement state handles basic ground based movement such as walking, sprinting and sneaking.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Top Speed	Float Data	The top movement speed (for keyboard input or max analog input). This can either be set as a value, or by referencing a float data entry.
Strafe Multiplier	Float Data	The multiplier applied to the max movement speed when strafing. This can either be set as a value, or by referencing a float data entry.
Reverse Multiplier	Float Data	The multiplier applied to the max movement speed when moving in reverse. This can either be set as a value, or by referencing a float data entry.
Acceleration	Float Data	The maximum acceleration. This can either be set as a value, or by referencing a float data entry.
Deceleration	Float Data	The maximum deceleration (when no input is applied). This can either be set as a value, or by referencing a float data entry.
Slope Speed Curve	SlopeSpeedCurve	an optional slope speed curve which controls the movement speed on sloping surfaces. Without this, the character controller will handle the speed change automatically.
Gravity Mode	Dropdown	When to apply gravity to the resulting move vector. Options are: **Always Apply**, **When Not Grounded** and **Never Apply**.
Damping	Float	The amount of damping to apply when changing direction.

See Also

The Motion Graph

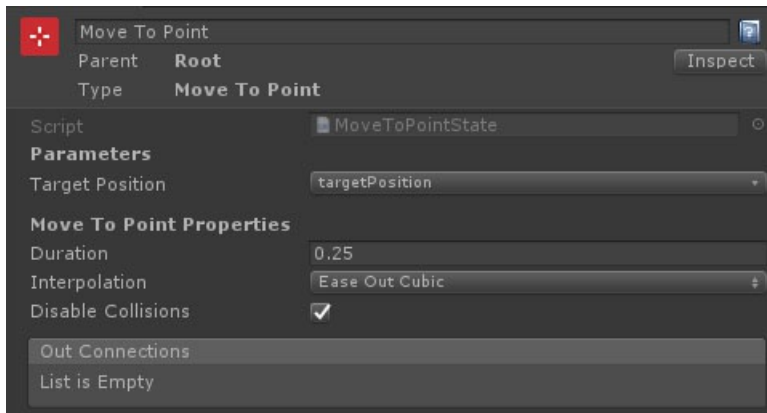Motion Graph States

Motion Graph Parameters And Data

SlopeSpeedCurve ScriptableObject

MoveToPoint MotionGraphState

Overview

The MoveToPoint state is used to move directly from the starting point to a target point with various interpolation options.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Target Position	Vector Parameter	The position to move to.
Duration	Float	The time required to reach the target.
Interpolation	Dropdown	The interpolation method from start to end. Options are: **Linear**, **EaseOutQuadratic**, **EaseOutCubic**, **Spring** and **Bounce**.
Disable Collisions	Boolean	Should collisions be disabled for the duration of the movement.

See Also

The Motion Graph

Motion Graph States

Motion Graph Parameters And Data

Null MotionGraphState

Overview

The null state is only used to branch to other states in the graph. The state always has the completed flag set, so it is best practice to set the last outbound connection from this state to one that uses the CompletedCondition. This would act as a default connection if none of the others are valid.

Inspector
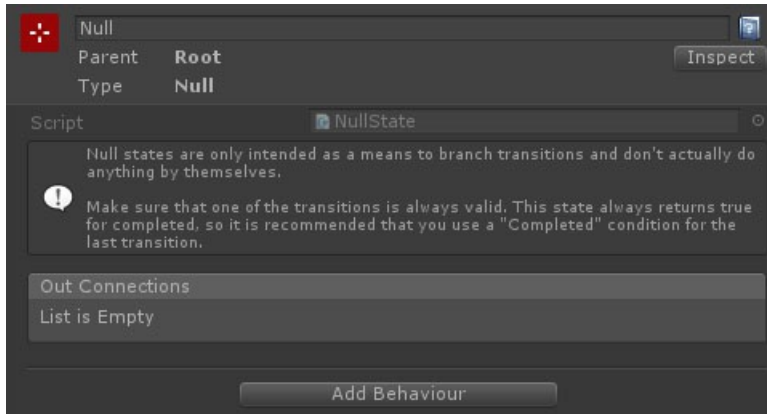
Properties

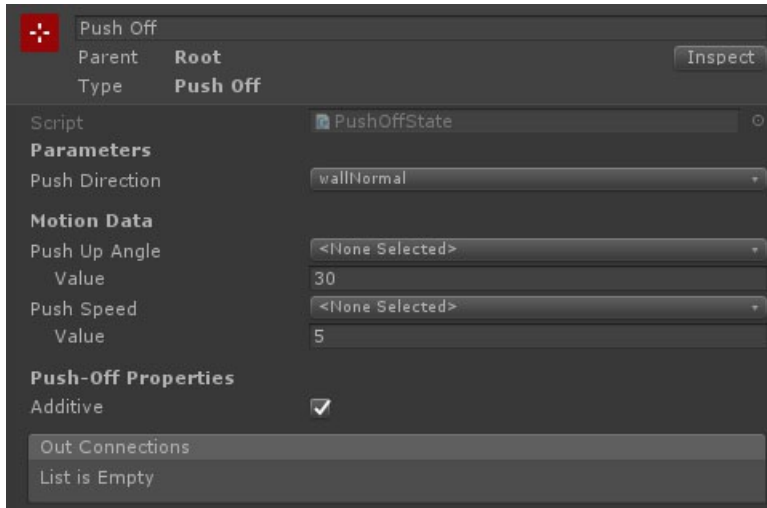| NAME	TYPE	DESCRIPTION
Name	String	The name of the state as visible in the graph viewport.

See Also

The Motion Graph

The Motion Graph Editor

Motion Graph States

PushOff MotionGraphState

Overview

The PushOff state is a one frame state which adds velocity based on a direction vector and can add an optional extra vertical boost.

Inspector

Properties

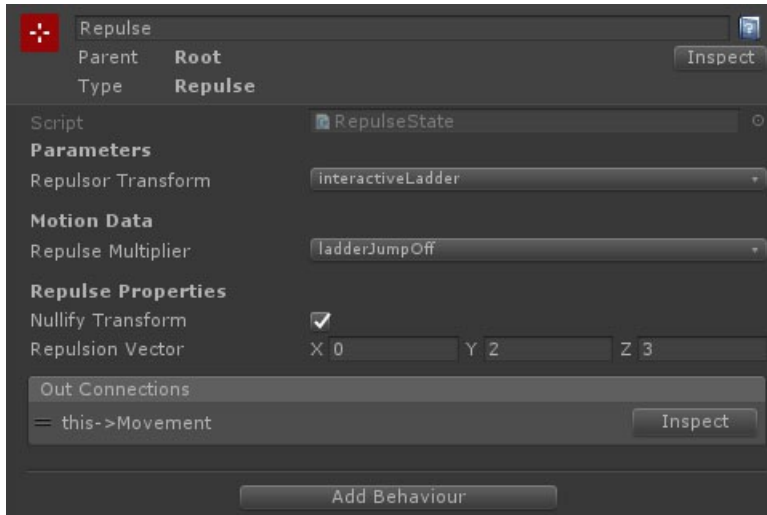| NAME	TYPE	DESCRIPTION
Push Direction	VectorParameter	The world direction to push in (you can fill this parameter using enhanced cast conditions).
Push Up Angle	FloatData	An additional upward rotation applied to the push direction. Resulting direction won't rotate past up/down.
Push Speed	FloatData	The speed to along the rotated push direction.
Additive	Boolean	Should the resulting velocity be added to the original character velocity or replace it.

See Also

The Motion Graph

The Motion Graph Editor

Motion Graph States

Repulse MotionGraphState

Overview

The repulse state adds a simple directional impulse away from the repulsor transform to the character for a single tick and then sets its completed flag. You should add an out-bound connection from this state that uses the CompletedCondition, connecting to an airborne state.

Inspector

Properties

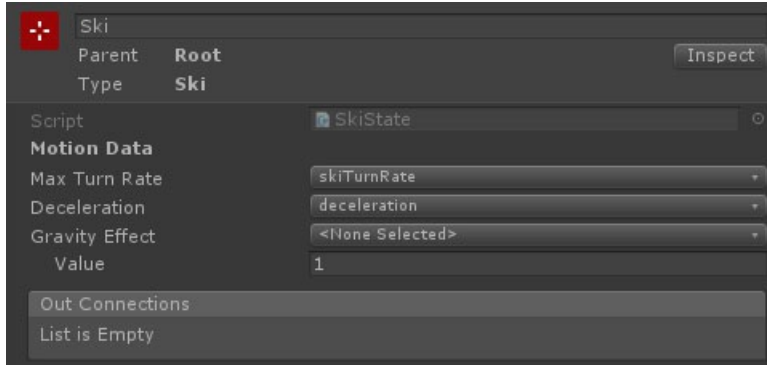| NAME	TYPE	DESCRIPTION
Repulsor Transform	TransformParameter	The transform parameter on the graph that is used to specify which transform to repulse from.
Nullify Transform	Boolean	Should the transform be nullified after use?
Repulsion Vector	Vector3	The velocity vector to apply to the character relative to the repulsor transform.
Repulse Multiplier	Float Data	A multiplier for the repulsion vector. This can either be set as a value, or by referencing a float data entry.

See Also

The Motion Graph

Motion Graph States

Motion Graph Parameters And Data

Ski MotionGraphState

Overview

The Ski state maintains forward velocity and allows the player to steer left and right. Steering turn rate is based on movement speed, with slower movement speed allowing turning in tighter circles.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Max Turn Rate	FloatData	The maximum turn rate in degrees per second.
Deceleration	FloatData	A constant deceleration (meters per second squared) while skiing. Once the character speed reaches zero, the state completes.
Gravity Effect	FloatData	A multiplier for gravity which affects speed on slopes.

See Also

The Motion Graph

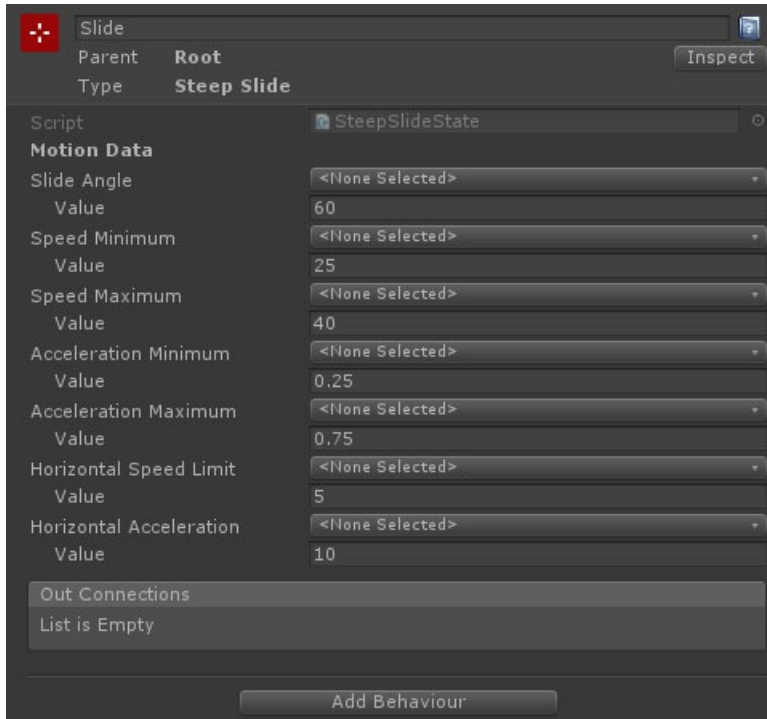The Motion Graph Editor

Motion Graph States

SteepSlide MotionGraphState

Overview

The steep slide state handles sliding down slopes that are too steep to walk on. The character has some degree of control while sliding, but only in the horizontal direction. They can't affect the speed of the slide down the slope.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Slide Angle	Float Data	The angle above which a character loses motor control and is in pure slide mode. This can either be set as a value, or by referencing a float data entry.
Speed Minimum	Float Data	The sliding speed the character will reach (downwards only) at the lowest slope angle for a full slide. This can either be set as a value, or by referencing a float data entry.
Speed Maximum	Float Data	The fastest possible sliding speed the character can reach (downwards only) during a near vertical slide. This can either be set as a value, or by referencing a float data entry.
Acceleration Minimum	Float Data	The down-slope acceleration multiplier applied to a character during a shallow slide. This can either be set as a value, or by referencing a float data entry.
Acceleration Maximum	Float Data	The down-slope acceleration multiplier applied to a character during a near vertical slide. This can either be set as a value, or by referencing a float data entry.
Horizontal Speed Limit	Float Data	The top speed the character can reach against the slide (side to side).This can either be set as a value, or by referencing a float data entry.
Horizontal Acceleration	Float Data	The across slope accleration when trying to redirect slide sideways (0 is instant).This can either be set as a value, or by referencing a float data entry.

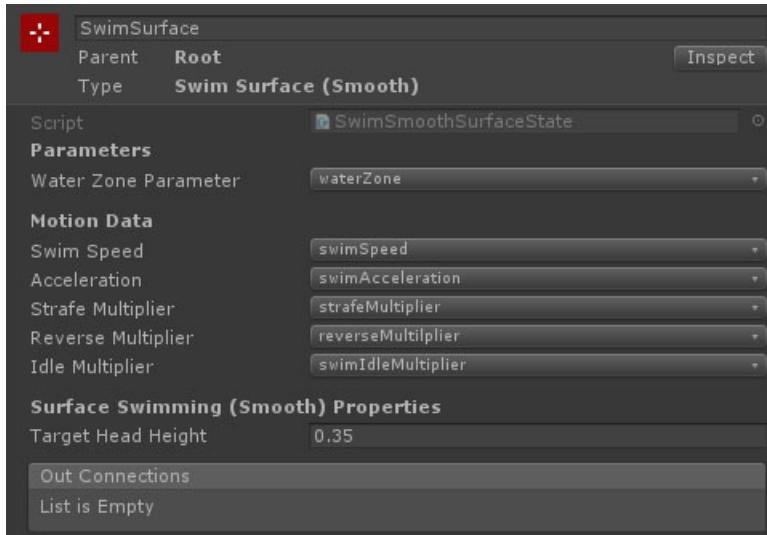## See Also

SwimSmoothSurfaceState MotionGraphState

Overview

The SwimSmoothSurfaceState state is used to move in all axes when underwater. It moves in strokes, pulling forward and the recovering.

Inspector

Properties

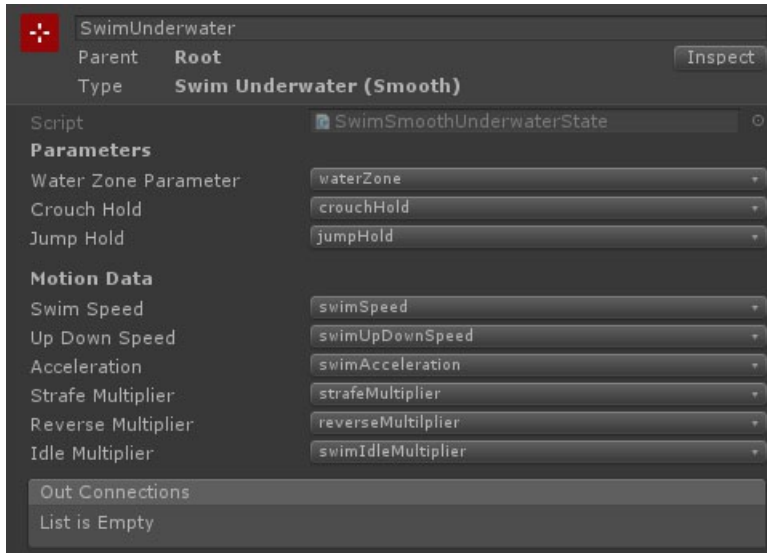| NAME	TYPE	DESCRIPTION
Water Zone	TransformParameter	The transform parameter which contains the transform of the water zone object.
Swim Speed	FloatData	The top movement speed (for keyboard input or max analog input).
Acceleration	FloatData	The maximum acceleration.
Strafe Multiplier	FloatData	The multiplier applied to the max movement speed and acceleration when strafing.
Reverse Multiplier	FloatData	The multiplier applied to the max movement speed and acceleration when moving backwards.
Idle Multiplier	FloatData	The multiplier applied to the acceleration when no input is detected.
Target Head Height	Float	A target for the distance the character capsule should be above the surface.

See Also

The Motion Graph

The Motion Graph Editor

Motion Graph States

SwimSmoothUnderwaterState MotionGraphState

Overview

The SwimSmoothUnderwaterState state is used to move in all axes when underwater. It moves in strokes, pulling forward and the recovering.

Inspector

Properties

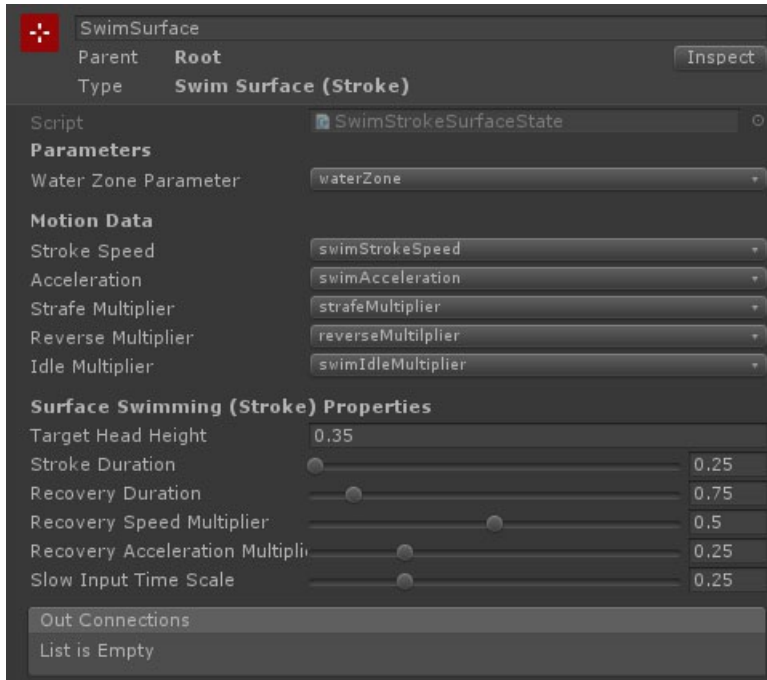| NAME	TYPE	DESCRIPTION
Water Zone	TransformParameter	The transform parameter which contains the transform of the water zone object.
Jump Hold	SwitchParameter	A switch parameter which tracks if the jump button is held (swim up).
Crouch Hold	SwitchParameter	A switch parameter which tracks if the crouch button is held (swim down).
Swim Speed	FloatData	The top movement speed (for keyboard input or max analog input).
Acceleration	FloatData	The maximum acceleration.
Strafe Multiplier	FloatData	The multiplier applied to the max movement speed and acceleration when strafing.
Reverse Multiplier	FloatData	The multiplier applied to the max movement speed and acceleration when moving backwards.
Idle Multiplier	FloatData	The multiplier applied to the acceleration when no input is detected.
Up Down Speed	FloatData	The maximum movement speed and acceleration due to up or down input.

See Also

The Motion Graph

The Motion Graph Editor

Motion Graph States

SwimStrokeSurfaceState MotionGraphState

Overview

The SwimStrokeSurfaceState state is used to move in all axes when underwater. It moves in strokes, pulling forward and the recovering.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Water Zone	TransformParameter	The transform parameter which contains the transform of the water zone object.
Stroke Speed	FloatData	The top movement speed (for keyboard input or max analog input).
Acceleration	FloatData	The maximum acceleration.
Strafe Multiplier	FloatData	The multiplier applied to the max movement speed and acceleration when strafing.
Reverse Multiplier	FloatData	The multiplier applied to the max movement speed and acceleration when moving backwards.
Idle Multiplier	FloatData	The multiplier applied to the acceleration when no input is detected.
Target Head Height	Float	A target for the distance the character capsule should be above the surface.
Stroke Duration	Float	The length of time a swimming stroke lasts (will be scaled by strafe / reverse multipliers).
Recovery Duration	Float	The length of time in between swimming strkes (will be scaled by strafe / reverse multipliers).
Recovery Speed Multiplier	Float	A multiplier applied to the speed in between strokes.

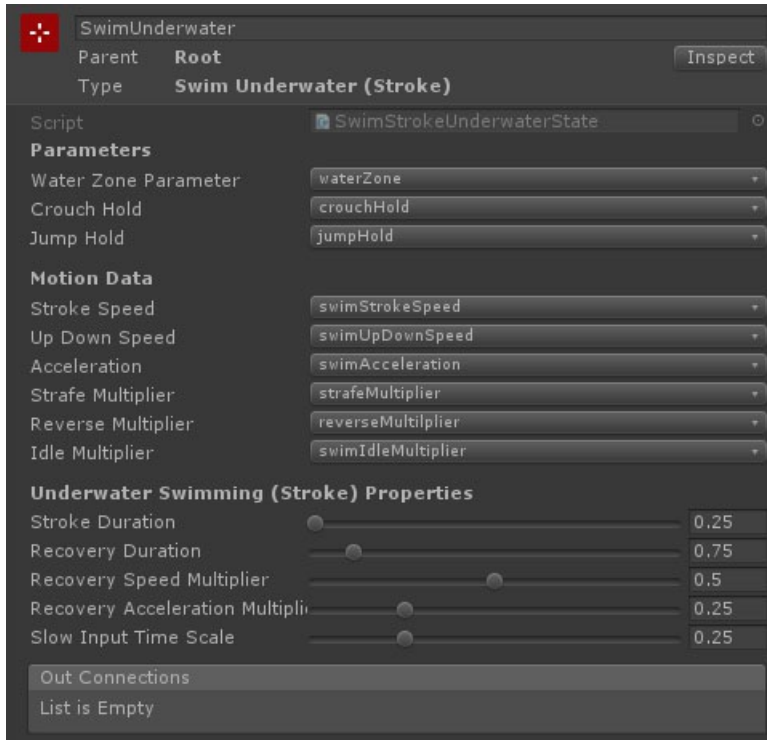| NAME	TYPE	DESCRIPTION
Recovery Acceleration Multiplier	Float	A multiplier applied to the acceleration (and deceleration) in between strokes.
Slow Input Time Scale	Float	At the minimum input amount, how much slower are strokes and recovery.

See Also

The Motion Graph

The Motion Graph Editor

Motion Graph States

SwimStrokeUnderwaterState MotionGraphState

Overview

The SwimStrokeUnderwaterState state is used to move in all axes when underwater. It moves in strokes, pulling forward and the recovering.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Water Zone	TransformParameter	The transform parameter which contains the transform of the water zone object.
Jump Hold	SwitchParameter	A switch parameter which tracks if the jump button is held (swim up).
Crouch Hold	SwitchParameter	A switch parameter which tracks if the crouch button is held (swim down).
Stroke Speed	FloatData	The top movement speed (for keyboard input or max analog input).
Acceleration	FloatData	The maximum acceleration.
Strafe Multiplier	FloatData	The multiplier applied to the max movement speed and acceleration when strafing.
Reverse Multiplier	FloatData	The multiplier applied to the max movement speed and acceleration when moving backwards.
Idle Multiplier	FloatData	The multiplier applied to the acceleration when no input is detected.
Up Down Speed	FloatData	The maximum movement speed and acceleration due to up or down input.
Stroke Duration	Float	The length of time a swimming stroke lasts (will be scaled by strafe / reverse multipliers).

NAME	TYPE	DESCRIPTION
Recovery Duration	Float	The length of time in between swimming strkes (will be scaled by strafe / reverse multipliers).
Recovery Speed Multiplier	Float	A multiplier applied to the speed in between strokes.
Recovery Acceleration Multiplier	Float	A multiplier applied to the acceleration (and deceleration) in between strokes.
Slow Input Time Scale	Float	At the minimum input amount, how much slower are strokes and recovery.

See Also

The Motion Graph

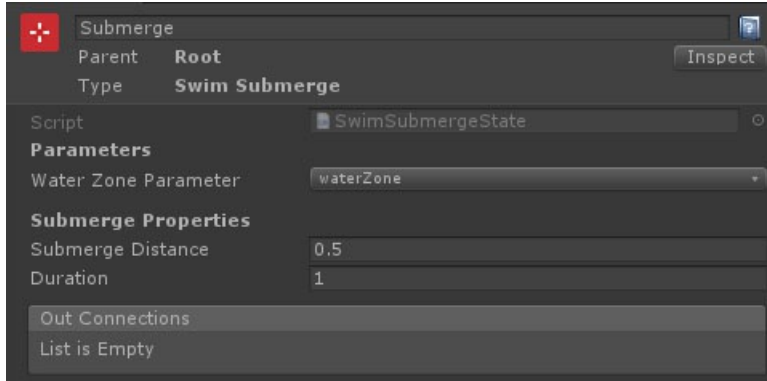The Motion Graph Editor

Motion Graph States

SwimSubmergeState MotionGraphState

Overview

The SwimSubmergeState state is used to transition from surface to underwater swimming while adapting to a moving water surface (eg waves).

Inspector

Properties

NAME	TYPE	DESCRIPTION
Water Zone Parameter	TransformParameter	The transform parameter which contains the transform of the water zone object.
Submerge Distance	Float	The distance below the surface of the water to submerge.
Duration	Float	The time to take while submerging (will be instant if already below submerge distance).

See Also

The Motion Graph

The Motion Graph Editor

Motion Graph States

VerticalWallRun MotionGraphState

Overview

The VerticalWallRun state is used to make a character run up a wall when they run and jump into it. The state completes once the upward speed reaches zero.

The VerticalWallRun state is often paired with the PushOff state, using the wall normal to jump away from the wall.

Inspector
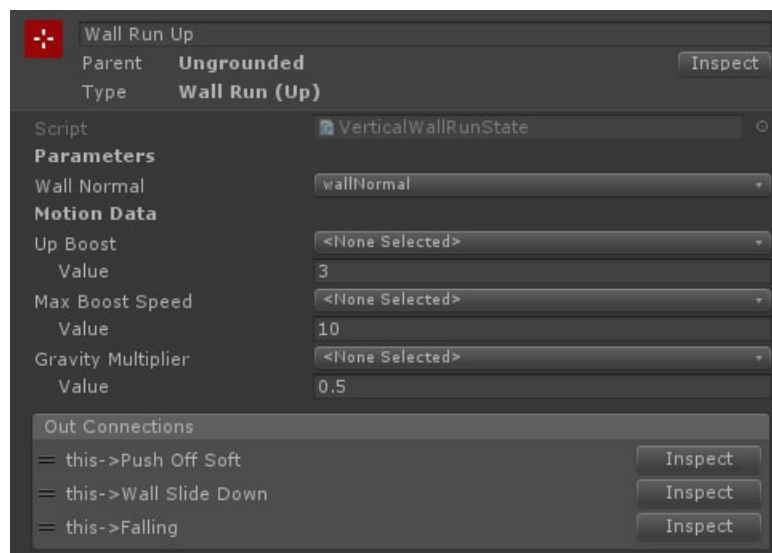
Properties

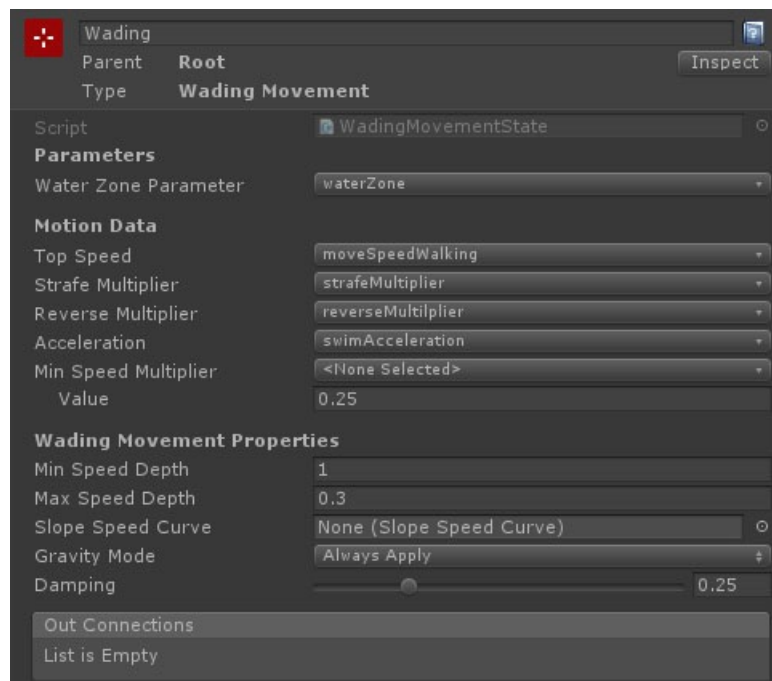| NAME	TYPE	DESCRIPTION
Up Boost	FloatData	An upward speed boost applied when entering the state.
Max Boost Speed	FloatData	The upward speed can not be boosted above this value (though it can start higher than this).
Gravity Multiplier	FloatData	A multiplier that is used to reduce the effects of gravity when running up the wall.

See Also

The Motion Graph

The Motion Graph Editor

Motion Graph States

Wading MotionGraphState

Overview

The Wading state is a variation of the Movement state which slows the character down based on how much of their capsule is below the water line.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Water Zone	TransformParameter	The transform parameter which contains the transform of the water zone object.
Name	FloatData	The multiplier for the standard movement speed when submerged to min speed depth.
Name	Float	The depth the character must be submerged to move at minimum speed.
Name	Float	The submersion depth of the character where their speed is not affected.

See Also

The Motion Graph

The Motion Graph Editor

Motion Graph States

WallRun MotionGraphState

Overview

The WallRun state sticks the character to a wall and runs along it. It maintains the horizontal velocity and reduces gravity so that the character slowly moves down the wall. The state completes when the wall angles away from its initial plane, but you can also use conditions to transition out for things like downwards velocity over a certain threshold or input away from the wall.

The WallRun state is often paired with the PushOff state, using the wall normal to jump away from the wall.

Inspector
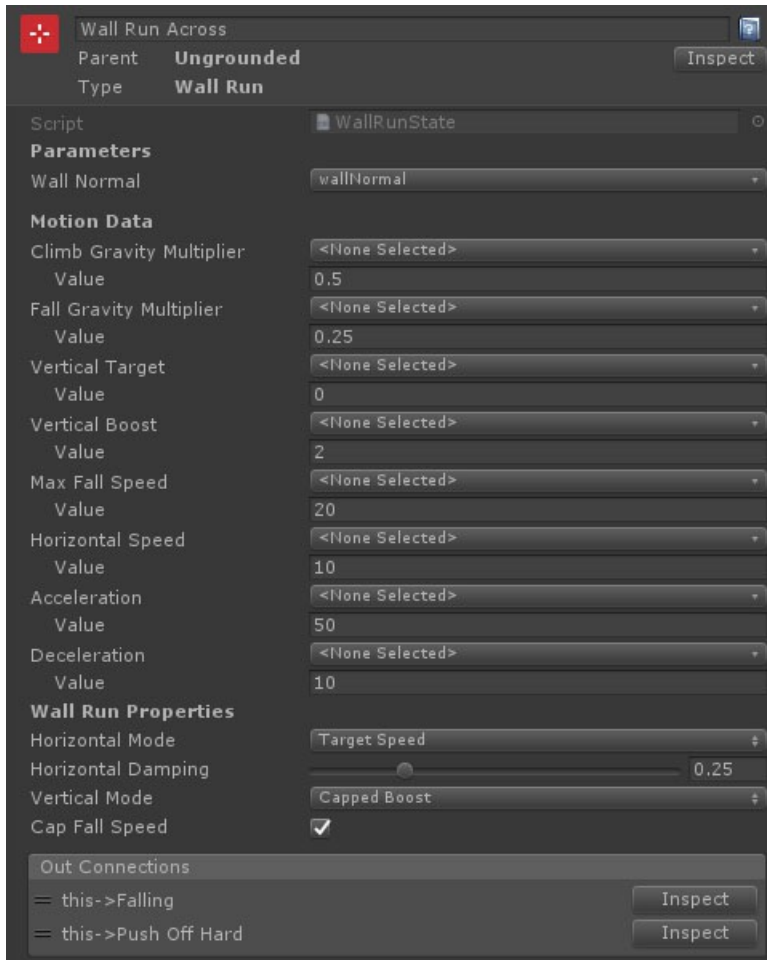
Properties

NAME	TYPE	DESCRIPTION
Wall Normal	VectorParameter	The vector parameter containing the wall normal. This will be read **and** written to each frame.
Climb Gravity Multiplier	FloatData	A multiplier applied to gravity acceleration when moving up the wall.
Fall Gravity Multiplier	FloatData	A multiplier applied to gravity acceleration when moving down the wall.
Vertical Target	FloatData	The target vertical speed. Visible if **Horizontal Mode** is set to **CappedBoost**, **Minimum** or **FixedSpeed**.

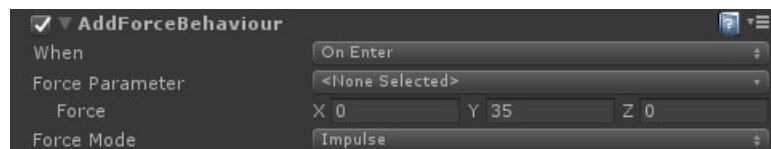| NAME	TYPE	DESCRIPTION
Vertical Boost	FloatData	An upward speed boost when first entering the state. Visible if **Horizontal Mode** is set to **Vertical Boost** or **CappedBoost**.
Max Fall Speed	FloatData	The maximum downward speed the character can reach while wall running. This property is only visible if **Cap Fall Speed** is set to **true**.
Horizontal Speed	FloatData	The target horizontal speed. Only visible if **Horizontal Mode** is set to **TargetSpeed** or **MinimumSpeed**.
Acceleration	FloatData	The acceleration up to the target speed. Only visible if **Horizontal Mode** is set to **TargetSpeed** or **MinimumSpeed**.
Deceleration	FloatData	The deceleration down to the target speed. Only visible if **Horizontal Mode** is set to **TargetSpeed**.
Horizontal Mode	Dropdown	How the horizontal wall run speed is calculated. **MaintainExisting** keeps the horizontal speed from the previous frame, **TargetSpeed** accelerates / decelerates to a set horizontal speed, **MinimumSpeed** accelerates up to the minimum speed if falling below it, but does not decelerate if faster.
Horizontal Damping	Float	The amount of damping to apply when changing direction or speed. Only visible if **Horizontal Mode** is set to **TargetSpeed** or **MinimumSpeed**.
Vertical Mode	Dropdown	How the vertical speed is calculated when entering the wall run. **VerticalBoost** adds an upward speed boost, **CappedBoost** adds an upward boost up to a maximum vertical speed, **Minimum** raises the vertical speed up to the minimum, **MaintainExisting** uses the vertical entry speed with no changes, **FixedSpeed** sets the vertical speed to a specific value
Cap Fall Speed	Boolean	Should the downwards fall speed be limited .

See Also

The Motion Graph

The Motion Graph Editor

Motion Graph States

AddForce MotionGraphBehaviour

Overview

The AddForce behaviour adds a force to the character at specific points. This can have a variety of uses, but one key one is as an alternative way to handle jumping that doesn't require interrupting other states like the jump motion state does.

Inspector

Properties

NAME	TYPE	DESCRIPTION
When	Dropdown	When should the force be added to the character. Options are **OnEnter**, **OnExit** and **WhenTriggered**.
Trigger Parameter	Trigger Parameter	A trigger that will be checked each frame, and the force applied if the trigger is set. This property is only visible if "When" is set to **WhenTriggered**.
Force Parameter	Vector Parameter	The force to apply to the character.
Force	Vector	The force to apply to the character. This property is shown if a "Force Parameter" has not been chosen.
Force Mode	ForceMode	How should the force be applied. Options are **Force** (takes mass and time into account), **Impulse** (takes mass into account), **VelocityChange** (mass and time are irrelevant), **Acceleration** (takes time into account).

See Also
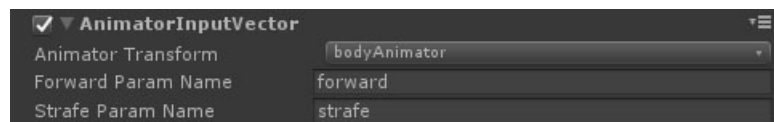
NeoCharacterController

Motion Graph Parameters And Data

AnimatorInputVector MotionGraphBehaviour

Overview

The AnimatorInputVector behaviour outputs the current move input of the character to animator controller parameters for use in blend trees.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Animator Transform	Transform Parameter	A parameter that points to the transform of the character's animator component.
Forward Param Name	String	The animator parameter name the forward input value should be written to.
Strafe Param Name	String	The animator parameter name the strafe input value should be written to. (positive = right).

See Also
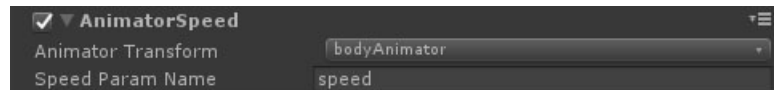
NeoCharacterController

Motion Graph Parameters And Data

AnimatorSpeed MotionGraphBehaviour

Overview

The AnimatorSpeed behaviour outputs the current move speed of the character to an animator controller parameter for use in blend trees or clip speeds.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Animator Transform	Transform Parameter	A parameter that points to the transform of the character's animator component.
Speed Param Name	String	The animator parameter name the speed value should be written to.

See Also

NeoCharacterController
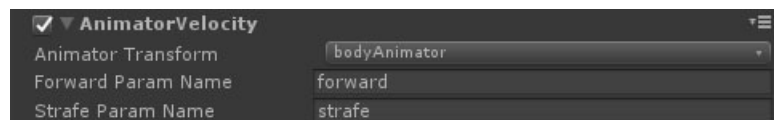
Motion Graph Parameters And Data

AnimatorVelocity MotionGraphBehaviour

Overview

The AnimatorVelocity behaviour outputs the current velocity of the character (in local space) to animator controller parameters for use in blend trees.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Animator Transform	Transform Parameter	A parameter that points to the transform of the character's animator component.
Forward Param Name	String	The animator parameter name the forward input value should be written to.
Strafe Param Name	String	The animator parameter name the strafe input value should be written to. (positive = right).

See Also
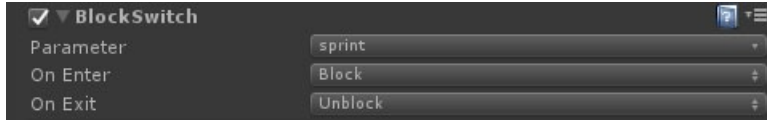
NeoCharacterController

Motion Graph Parameters And Data

BlockSwitchParameter MotionGraphBehaviour

Overview

The BlockSwitchParameter behaviour will block the specified switch parameter, meaning that it can only return false for the duration of the block.

Inspector

Properties

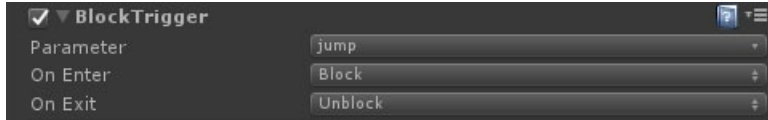| NAME	TYPE	DESCRIPTION
Parameter	SwitchParameter	The parameter to modify.
OnEnter	Dropdown	Whether to block or unblock the parameter on entering the state. Options are **Block**, **Unblock**, **Nothing**.
OnEnter	Dropdown	Whether to block or unblock the parameter on exiting the state. Options are **Block**, **Unblock**, **Nothing**.

See Also

Motion Graph Parameters And Data

BlockTriggerParameter MotionGraphBehaviour

Overview

The BlockTriggerParameter behaviour will block the specified trigger parameter, meaning that it cannot fire. This is useful for situations like preventing jump triggers while crouched.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Parameter	TriggerParameter	The propery to modify.
OnEnter	Dropdown	Whether to block or unblock the parameter on entering the state. Options are **Block**, **Unblock**, **Nothing**.
OnEnter	Dropdown	Whether to block or unblock the parameter on exiting the state. Options are **Block**, **Unblock**, **Nothing**.

See Also
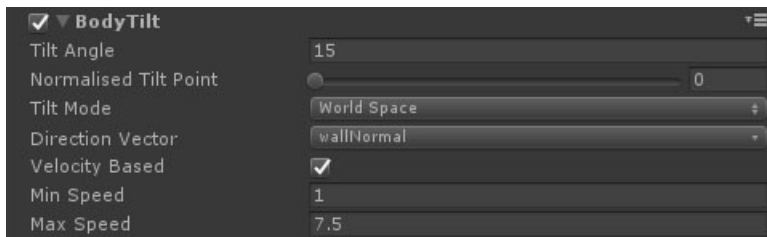
Motion Graph Parameters And Data

BodyTilt MotionGraphBehaviour

Overview

The BodyTilt behaviour tilts the character based on the specified tilt mode. Some example uses are:

- It is used in the parkour demo to tilt away from the wall by using the wall normal
- It is also used in the parkour demo to tilt back when crouch sliding by setting the mode to velocity, and the tilt angle to negative
- It is used to add a bob to the swimming in the swimming demo by setting the mode to velocity

Inspector

Properties

NAME	TYPE	DESCRIPTION
Tilt Angle	Float	The angle to tilt.
Normalised Tilt Point	Float	The point on the character to tilt from. 0 is the bottom, 1 is the top.
Tilt Mode	Dropdown	How the tilt direction is calculated: • **Character Relative** uses a vector parameter to specify a tilt vector relative to the character. • **World Space** does the same but in world space. • **Velocity** tilts in the direction of movement of the character. • **Velocity Lateral** tilts left or right based on the speed of the character in those directions. • **Input** tilts based on the player input and the direction the character is facing. • **Input Lateral** tilts left or right based on the player input.
Direction Vector	Vector Parameter	The direction to tilt in. This property is only shown if the tilt mode is set to **Character Relative** or **World Space**.
Velocity Based	Boolean	If set then the tilt angle will be based on the speed of the character.
Min Speed	Float	The speed below which the tilt angle will be 0.
Max Speed	Float	The speed above which the tilt will reach the full tilt angle specified above.

See Also

NeoCharacterController

Motion Graph Parameters And Data

CameraJiggleSpring MotionGraphBehaviour

Overview

The CameraJiggleSpring behaviour triggers the player character's camera jiggle spring additive effect with the supplied strength.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Switch Condition	SwitchParameter	An **optional** switch condition that defines if the jiggle should be triggered.
Trigger Condition	TriggerParameter	An **optional** trigger condition that defines if the jiggle should be triggered.
When	Dropdown	When should the camera jiggle spring be triggered.
Angle	Float	The strength of the jiggle effect (max angle is set in the Additive Jiggle component on the camera spring transform).
Angle	Float	Should the CW/CCW direction of the jiggle be chosen at random each time.

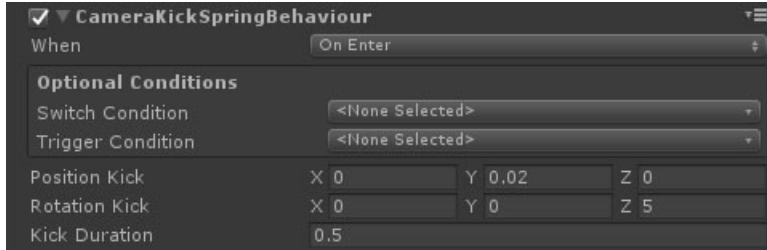## See Also

MotionController

Motion Graph Parameters And Data

Additive Transforms and Effects

CameraKickSpring MotionGraphBehaviour

Overview

The CameraKickSpring behaviour triggers the player character's camera kick spring additive effect with the supplied offset and rotation.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Switch Condition	SwitchParameter	An **optional** switch condition that defines if the kick should be triggered.
Trigger Condition	TriggerParameter	An **optional** trigger condition that defines if the kick should be triggered.
When	Dropdown	When should the camera kick spring be triggered.
Position Kick	Vector3	The position offset for the camera at the strongest point of the kick. Keep these values small (cm, not meters) or you risk clipping scenery or weapon geometry.
Rotation Kick	Vector3	The rotation offset for the camera at the strongest point of the kick. Positive X nods forwards. Positive Y turns right. Positive Z tilts counter-clockwise.
Kick Duration	Float	The amount of time the kick effect should last.

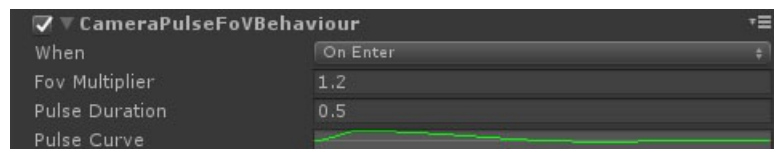## See Also

MotionController

Motion Graph Parameters And Data

Additive Transforms and Effects

CameraPulseFoV MotionGraphBehaviour

Overview

The CameraPulseFoV behaviour can apply a brief pulsed multiplier to the player camera's field of view. This is great for quick acceleration and impact effects.

Inspector

Properties

NAME	TYPE	DESCRIPTION
When	Dropdown	When should the camera FoV pulse be triggered.
FovMultiplier	Float	The FoV multiplier to apply to the camera when the animation curve Y-axis is at 1.
PulseDuration	Float	The duration in seconds for the pulse to last.
PulseCurve	Animation Curve	A curve for the strength of the pulse. X is normalised time. Y = 0 means the FoV is 1x (no effect), Y = 1 means the FoV is the target FoV multiplier.

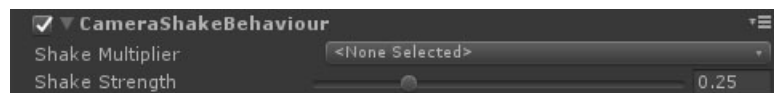## See Also

MotionController

Additive Transforms and Effects

CameraShake MotionGraphBehaviour

Overview

The CameraShake behaviour adds a constant shake to the camera whilst in the current state or sub-graph. You can also use the shake multiplier to fade in or out the effect over time.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Shake Multiplier	FloatParameter	An **optional** float parameter to multiply the shake value by. This allows for increasing shake while falling, etc.
Shake Strength	Float	The strength of the jiggle effect (max angle is set in the Additive Jiggle component on the camera spring transform).

See Also

MotionController

Motion Graph Parameters And Data

Additive Transforms and Effects

ClampFloat MotionGraphBehaviour

Overview

The ClampFloat behaviour can be used to clamp a float parameter within a specific range. The clamp is applied every tick.

Inspector

Properties

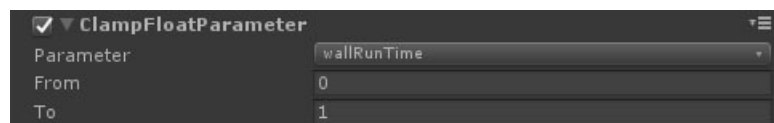| NAME	TYPE	DESCRIPTION
Parameter	FloatParameter	The parameter to modify.
From	Float	The minimum parameter value.
To	Float	The maximum parameter value.

See Also

Motion Graph Parameters And Data

ClampInt MotionGraphBehaviour

Overview

The ClampInt behaviour can be used to clamp a int parameter within a specific range. The clamp is applied every tick.

Inspector

Properties

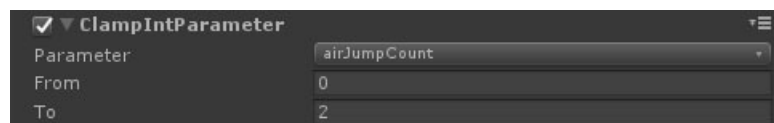| NAME	TYPE	DESCRIPTION
Parameter	IntParameter	The parameter to modify.
From	Float	The minimum parameter value.
To	Float	The maximum parameter value.

See Also

Motion Graph Parameters And Data

ConstrainCameraPitch MotionGraphBehaviour

Overview

The ConstrainCameraPitch behaviour is used to constrain the player character's camera to a specific pitch range direction while inside a motion state.

Inspector

Properties

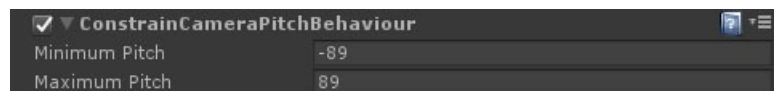| NAME	TYPE	DESCRIPTION
Minimum Pitch	Float	The minimum angle the camera can look down.
Maximum Pitch	Float	The maximum angle the camera can look up.

See Also

MotionController

ConstrainCameraYaw MotionGraphBehaviour

Overview

The ConstrainCameraYaw behaviour is used to constrain the player character's camera to a specific yaw direction while inside a motion state. An example is using the wall normal vector during a wall run to prevent the character turning into the wall (and to turn when running along curving walls).

Inspector
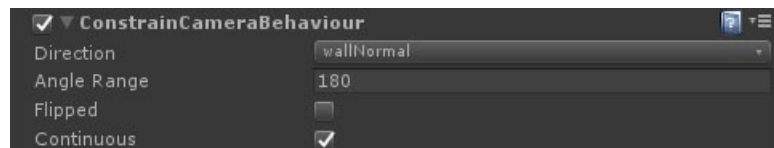
Properties

NAME	TYPE	DESCRIPTION
Direction	VectorParameter	The vector parameter to use as the constraint direction.
Angle Range	Float	The angle range to constrain to.
Flipped	Boolean	Flip the direction vector.
Continuous	Boolean	Should the constraints be updated each frame (if the vector parameter changes).

See Also

MotionController

Debug MotionGraphBehaviour

Overview

The Debug behaviour can be used to help understand when the motion graph enters or exits a specific state or sub-graph.

Inspector
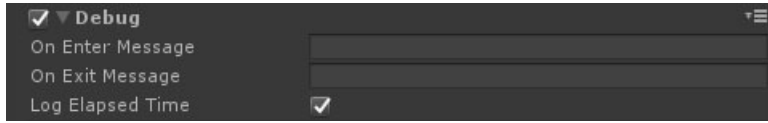
Properties

NAME	TYPE	DESCRIPTION
On Enter Message	String	A message to print to the console on entering the state or sub-graph.
On Exit Message	String	A message to print to the console on exiting the state or sub-graph.
Log Elapsed Time	Boolean	If set, prints the time spent in the state or sub-graph to the console on exiting it.

DisableCollider MotionGraphBehaviour

Overview

The DisableCollider behaviour will completely disable the character's CharacterController. This is useful in situations such as scripted or animated movements where environment collisions could interfere.

Inspector
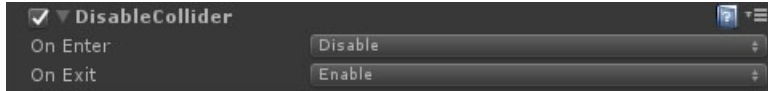
Properties

NAME	TYPE	DESCRIPTION
On Enter	Dropdown	What to do to the character collider on entering the state. Options are: **Enable**, **Disable**, **Nothing**.
On Exit	Dropdown	What to do to the character collider on exiting the state. Options are: **Enable**, **Disable**, **Nothing**.

See Also

Unity CharacterController

DrainStamina MotionGraphBehaviour

Overview

The DrainStamina behaviour applies a constant stamina drain to a StaminaSystem attached to the character.

Inspector
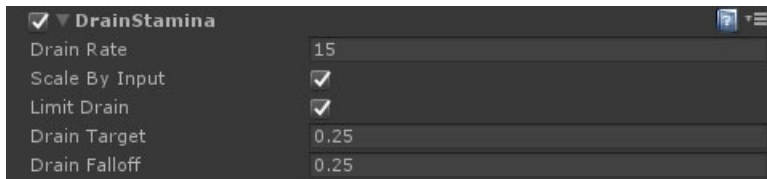
Properties

NAME	TYPE	DESCRIPTION
Drain Rate	Float	The rate to drain the stamina at (bear in mind the stamina system also refreshes at a certain rate too, so these can cancel out).
Scale By Input	Boolean	Should the controller's move input scale also scale the stamina drain.
Limit Drain	Boolean	Is there a lower limit that the behaviour will drain stamina to before the drain rate falls off. The following properties will be exposed if this is true.
Drain Target	Float	The minimum level that the behaviour can drain stamina to.
Drain Falloff	Float	The stamina drain falls away to 0 as it approaches the target level, starting at this falloff value above it.

See Also

StaminaSystem

FootstepAudio MotionGraphBehaviour

Overview

The FootstepAudio behaviour plays footsteps based on the character speed and the ground surface below them.

Inspector

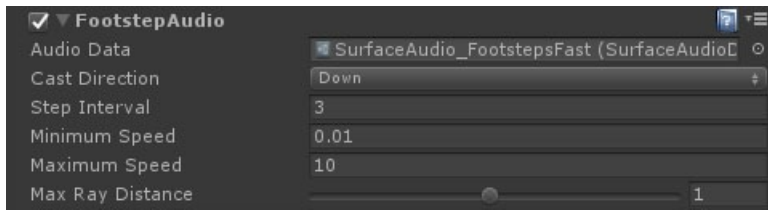## Properties

NAME	TYPE	DESCRIPTION
Audio Data	SurfaceAudioData	The surface audio library for the slide audio clips.
Cast Direction	Dropdown	The direction to perform the footstep surface check. Available options are: • **Down** casts downwards (based on the character up vector). • **LocalVector** casts based on the *Cast Vector* property in local space. • **WorldVector** casts based on the *Cast Vector* property in world space. • **WorldParameter** casts based on the vector value of the *Vector Parameter* property in world space. • **WorldParameterInverse** casts based on the flipped vector value of the *Vector Parameter* property in world space. • **LocalParameter** casts based on the vector value of the *Vector Parameter* property in local space. • **LocalParameterInverse** casts based on the flipped vector value of the *Vector Parameter* property in local space.
Cast Vector	Vector3	The direction to cast in for footstep surface checks.
Vector Parameter	VectorParameter	A vector parameter containing the direction vector to cast in for footstep surface checks.
Step Interval	Float	The interval between steps. Higher numbers mean the steps are further apart.
Minimum Speed	Float	The speed below which no footstep audio will be played.
Maximum Speed	Float	The maximum speed that the actual speed will be clamped to. Prevents rapid fire footsteps.
Max Ray Distance	Float	The downward raycast length for the ground surface test.
Ray Offset	Float	The vertical offset above the absolute bottom of the character collider to start the downward surface test.

See Also

SurfaceAudioData

ImpactDamage MotionGraphBehaviour

Overview

The ImpactDamage behaviour enables or disables character damage on impact when entering or exiting a state.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Fall Damage On Enter	Dropdown	Should fall damage be enabled, disabled or unchanged on entering the state.
Fall Damage On Exit	Dropdown	Should fall damage be enabled, disabled or unchanged on exiting the state.
Body Impact Damage On Enter	Dropdown	Should body impact damage be enabled, disabled or unchanged on entering the state.
Body Impact Damage On Exit	Dropdown	Should body impact damage be enabled, disabled or unchanged on exiting the state.
Head Impact Damage On Enter	Dropdown	Should head impact damage be enabled, disabled or unchanged on entering the state.
Head Impact Damage On Exit	Dropdown	Should head impact damage be enabled, disabled or unchanged on exiting the state.

See Also

Motion Graph Parameters And Data

Health and Damage

InvokeEvent MotionGraphBehaviour

Overview

The InvokeEvent behaviour will invoke the specified event on entering the state, on exiting or both.

Inspector
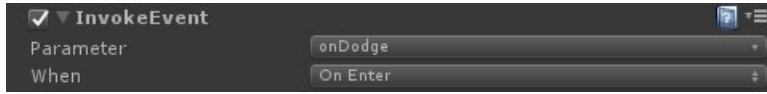
Properties

NAME	TYPE	DESCRIPTION
Parameter	EventParameter	The event parameter to invoke.
When	Dropdown	When should event be invoked. Options are **EnterAndExit**, **EnterOnly**, **ExitOnly**.

See Also

Motion Graph Parameters And Data

LadderAudio MotionGraphBehaviour

Overview

The LadderAudio behaviour is used to trigger audio cues when climbing a ladder. This works in a similar way to the footsteps system, but based on movement along the ladder's up axis.

Inspector
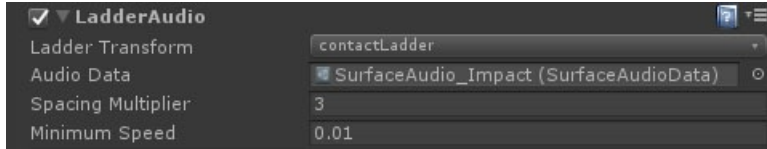
Properties

NAME	TYPE	DESCRIPTION
Audio Data	SurfaceAudioData	The surface audio library for the slide audio clips.
Ladder Transform	TransformProperty	The transform property holding the ladder transform.
Spacing Multiplier	Float	How many rungs apart to play a sound. This is based on the ladder **spacing** property.
Minimum Speed	Float	The speed below which no audio will be played.

See Also

Ladders

SurfaceAudioData

LockInventorySelection MotionGraphBehaviour

Overview

The LockInventorySelection behaviour will lock the character's inventory selection to the specified item until unlocked by an UnlockInventorySelection behaviour.
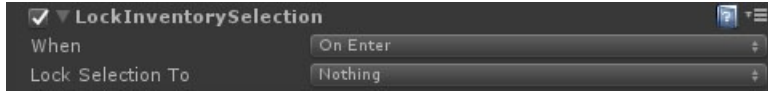
Inspector

Properties

NAME	TYPE	DESCRIPTION
When	Dropdown	When should the inventory selection be set. Options are: **OnEnter** and **OnExit**.
What	Dropdown	What to lock the inventory selection to. Options are: **Nothing**, **BackupItem** and **SlotIndex**
Slot Index	Float	The quick slot to lock the selection to.

See Also

Inventory

UnlockInventorySelection

LoopingAudio MotionGraphBehaviour

Overview

The LoopingAudio behaviour plays a looping audio clip from the specified source.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Clip	[AudioClip][unity-audioclip]	The looping audio clip to play.
Source	FpsCharacterAudioSource	The source ID to play from (generated constant).
Pitch	Float	The pitch of the loop.

See Also

Generated Constants

ModifyCharacterVelocity MotionGraphBehaviour

Overview

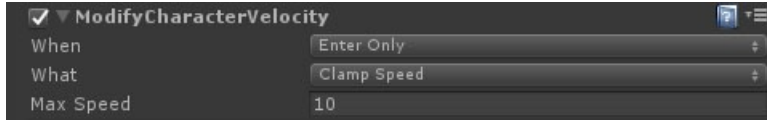The ModifyCharacterVelocity behaviour can change the character controller's velocity in various ways on entering or exiting the state.

Inspector

Properties

NAME	TYPE	DESCRIPTION
When	Dropdown	When should the velocity be modified. Options are **EnterAndExit**, **EnterOnly**, **ExitOnly**.
What	Dropdown	How should the velocity be modified. **SetLocal** sets the velocity relative to the character's current heading, **SetWorld** applies the new velocity in world space, **ClampSpeed** reduces the speed to the limit value if it is above it, and **Multiply** multiplies the current velocity by a value.
Local Velocity	Vector3	The target velocity of the character controller relative to its direction. Only visible if "What" is set to **SetLocal**.
World Velocity	Vector3	The target velocity of the character controller in world space. Only visible if "What" is set to **SetWorld**.
Max Speed	Float	The maximum speed the character can travel at. Only visible if "What" is set to **ClampSpeed**.
Multiplier	Float	A multiplier to apply to the character's velocity. Only visible if "What" is set to **Multiply**.

See Also

NeoCharacterController

Motion Graph Parameters And Data

ModifyFloatParameter MotionGraphBehaviour

Overview

The ModifyFloatParameter behaviour either sets or modifies the specified float parameter on entering the state, on exit or both.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Parameter	FloatParameter	The parameter to modify.
When	Dropdown	When should the parameter be modified. Options are **EnterAndExit**, **EnterOnly**, **ExitOnly**.
What	Dropdown	How should the parameter be modified. Options are **Set**, **Reset**, **Add**, **Subtract**, **Floor** (round down to the nearest whole number) and **Ceiling** (round up to the nearest whole number) .
Value	Float	The value to set to, add or subtract based on the **What** parameter.

See Also

Motion Graph Parameters And Data

ModifyIntParameter MotionGraphBehaviour

Overview

The ModifyIntParameter behaviour either sets or modifies the specified int parameter on entering the state, on exit or both.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Parameter	IntParameter	The parameter to modify.
When	Dropdown	When should the parameter be modified. Options are **EnterAndExit**, **EnterOnly**, **ExitOnly**.
What	Dropdown	How should the parameter be modified. Options are **Set**, **Reset**, **Add**, **Subtract**.
Value	Int	The value to set to, add or subtract based on the **What** parameter.

See Also

Motion Graph Parameters And Data

ModifyStamina MotionGraphBehaviour

Overview

The ModifyStamina behaviour performs an operation on the StaminaSystem attached to the character.

Inspector

Properties

NAME	TYPE	DESCRIPTION
When	Dropdown	When should stamina be modified. Options are: **OnEnter**, **OnExit** and **Both**.
What	Dropdown	What should the modification be. Options are: **Increment**, **Decrement**, **IncrementNormalised**, **DecrementNormalised**, **SetToValue**, **SetToValueNormalised**, **SetToMax**, **SetToZero**. The normalised options act on the stamina as a factor of max stamina.
Amount	Float	Value to use for modifying the stamina.

See Also

StaminaSystem

ModifySwitchParameter MotionGraphBehaviour

Overview

The ModifySwitchParameter behaviour sets the specified switch on entering the state, on exit or both.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Parameter	SwitchParameter	The parameter to set.
On Enter	Dropdown	How should the parameter be modified on entering the state. Options are **Unchanged**, **True**, **False**, **Toggle**, **Reset**, **Previous**.*
On Exit	Dropdown	How should the parameter be modified on exiting the state. Options are **Unchanged**, **True**, **False**, **Toggle**, **Reset**, **Previous**.*

* Previous is only valid on exit, and will set the switch to its state before entering.

See Also

Motion Graph Parameters And Data

ModifyTransformParameter MotionGraphBehaviour

Overview

The ModifyTransformParameter behaviour either sets or resets the specified transform parameter on entering the state, on exit or both.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Parameter	TransformParameter	The parameter to modify.
When	Dropdown	When should the parameter be modified. Options are **EnterAndExit**, **EnterOnly**, **ExitOnly**.
What	Dropdown	What is the action to do. **Nullify** clears the value and **Find** will set the value using GameObject.Find.

See Also

Motion Graph Parameters And Data

GameObject.Find

ModifyTriggerParameter MotionGraphBehaviour

Overview

The ModifyTriggerParameter behaviour either sets or resets the specified trigger on entering the state, on exit or both.

Inspector

Properties

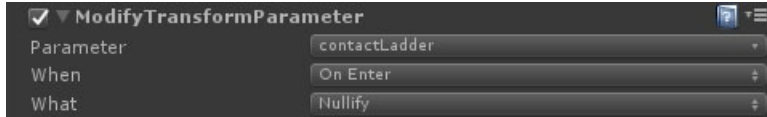| NAME	TYPE	DESCRIPTION
Parameter	TriggerParameter	The parameter to modify.
When	Dropdown	When should the parameter be modified. Options are **EnterAndExit**, **EnterOnly**, **ExitOnly**.
What	Dropdown	What is the action to do. Options are **Set** and **Reset**.

See Also

Motion Graph Parameters And Data

ModifyVectorParameter MotionGraphBehaviour

Overview

The ModifyVectorParameter behaviour either sets or modifies the specified vector parameter on entering the state, on exit or both.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Parameter	VectorParameter	The parameter to modify.
When	Dropdown	When should the parameter be modified. Options are **EnterAndExit**, **EnterOnly**, **ExitOnly**.
What	Dropdown	How should the parameter be modified. The available options are: • **Set** will set the vector to the specified value. • **Reset** resets the vector to its starting value. • **Add** adds the specified vector to the parameter. • **Subtract** subtracts the specified vector from the parameter. • **Multiply** multiplies the parameter by the specified multiplier. • **Normalize** normalizes the parameter vector to magnitude. 1

- **Flatten** flattens the parameter vector onto the horizontal plane of the character
- **ClampMagnitude** clamps the magnitude to a specified maximum length || Value | Vector3 | The value to set to, add or subtract based on the **What** property. || Multiplier | Float | A multiplier to apply to the vector if the **What** property is set to **Multiply**. || Clamp | Float | The magnitude to clamp the vector to if the **What** property is set to **ClampMagnitude**. |

See Also

Motion Graph Parameters And Data

PassiveSlide MotionGraphBehaviour

Overview

The PassiveSlide behaviour lowers ground friction when the character is on a steep enough slope, and is not attempting to move. If the motion controller receives move input then the NeoCharacterController slope friction will be set to its value on entering the state or sub-graph. If not, then the slope friction will be set to **Slide Friction** setting as long as this is lower than the friction already set and the angle is steep enough.

By default, the character must be stood on a slope steeper than 30 degrees in order to slide. If the **Slope Data Key** corresponds to a valid Slope MotionControllerDataEntry then the slide will be triggered when the angle is greater than its **Slope Slide Angle** property.

Inspector
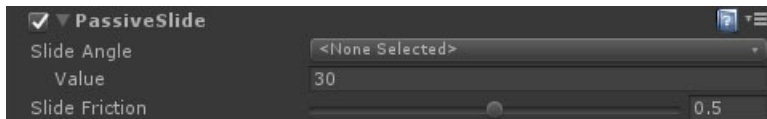
Properties

NAME	TYPE	DESCRIPTION
Slope Angle	Float Data	A slope angle that chooses when to start sliding. This can either be set as a value, or by referencing a float data entry.
Slide Friction	Float	The slope friction to apply when sliding.

See Also

NeoCharacterController

Motion Graph Parameters And Data

PlayAudioClip MotionGraphBehaviour

Overview

The PlayAudioClip behaviour plays an audio clip on entering the state, on exiting, or both.
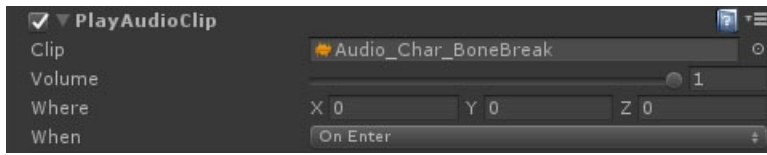
Inspector

Properties

NAME	TYPE	DESCRIPTION
Clip	AudioClip	The audio clip to play.
Volume	Float	The volume to play the clip at.
Where	Vector3	The offset from the character controller transform position to play the clip at.
When	Dropdown	When should the clip be played. Options are **EnterAndExit**, **EnterOnly**, **ExitOnly**.

See Also

Unity AudioClip

PlayCharacterAudio MotionGraphBehaviour

Overview

The PlayCharacterAudio behaviour will play an audio clip on entering the state, exiting the state, or both.

Inspector
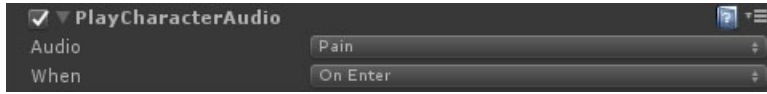
Properties

NAME	TYPE	DESCRIPTION
Audio	FpsCharacterAudio	The audio ID to play (generated constant).
When	Dropdown	When should the audio be played. Options are **EnterAndExit**, **EnterOnly**, **ExitOnly**.

See Also

Generated Constants

RecordVelocity MotionGraphBehaviour

Overview

The RecordVelocity behaviour outputs the current velocity of the character to a vector parameter on the graph for use in other conditions or states.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Parameter	Vector Parameter	The parameter to modify.
When	Dropdown	At what point should the velocity be recorded. Options are **OnEnter**, **OnExit** and **Always**. The latter records the velocity each frame while in the state or subgraph it's attached to.

See Also

NeoCharacterController

Motion Graph Parameters And Data

SetAnimatorBool MotionGraphBehaviour

Overview

The SetAnimatorBool behaviour writes to an animator controller parameter on entering or exiting the state or sub-graph.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Animator Transform	Transform Parameter	A parameter that points to the transform of the character's animator component.
Parameter Name	String	The name of the animator parameter to write to.
When	Dropdown	When should the parameter be modified.
On Enter Value	Boolean	The value to write to the parameter on entering the state / subgraph.
On Exit Value	Boolean	The value to write to the parameter on exiting the state / subgraph.

See Also

MotionController

Motion Graph Parameters And Data

SetAnimatorFloat MotionGraphBehaviour

Overview

The SetAnimatorFloat behaviour writes to an animator controller parameter on entering or exiting the state or sub-graph.

Inspector

Properties

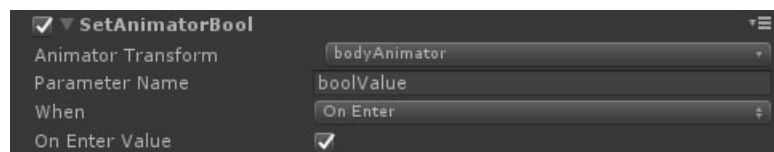| NAME	TYPE	DESCRIPTION
Animator Transform	Transform Parameter	A parameter that points to the transform of the character's animator component.
Parameter Name	String	The name of the animator parameter to write to.
When	Dropdown	When should the parameter be modified.
On Enter Value	Float	The value to write to the parameter on entering the state / subgraph.
On Exit Value	Float	The value to write to the parameter on exiting the state / subgraph.

See Also

MotionController

Motion Graph Parameters And Data

SetAnimatorInt MotionGraphBehaviour

Overview

The SetAnimatorInt behaviour writes to an animator controller parameter on entering or exiting the state or sub-graph.

Inspector

Properties

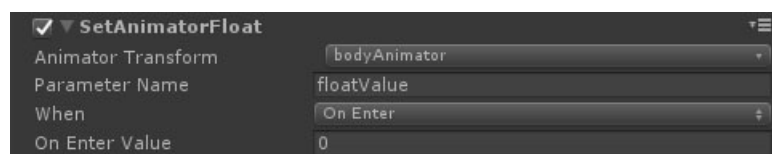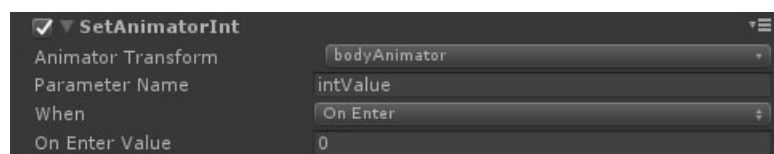| NAME	TYPE	DESCRIPTION
Animator Transform	Transform Parameter	A parameter that points to the transform of the character's animator component.
Parameter Name	String	The name of the animator parameter to write to.
When	Dropdown	When should the parameter be modified.
On Enter Value	Integer	The value to write to the parameter on entering the state / subgraph.
On Exit Value	Integer	The value to write to the parameter on exiting the state / subgraph.

See Also

MotionController

Motion Graph Parameters And Data

SetAnimatorBool MotionGraphBehaviour

Overview

The SetAnimatorTrigger behaviour writes to an animator controller parameter on entering or exiting the state or sub-graph.

Inspector
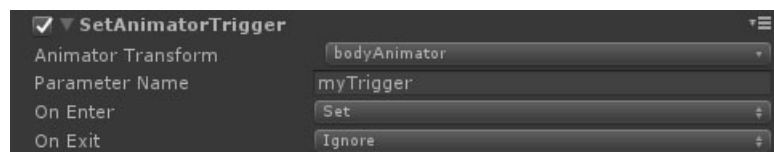
Properties

NAME	TYPE	DESCRIPTION
Animator Transform	Transform Parameter	A parameter that points to the transform of the character's animator component.
Parameter Name	String	The name of the animator parameter to write to.
On Enter	Dropdown	The action to perform on entering the state / subgraph.
On Exit	Dropdown	The action to perform on exiting the state / subgraph.

See Also

MotionController

Motion Graph Parameters And Data

SetSteering MotionGraphBehaviour

Overview

The NeoCharacterController that NeoFPS is based on has a steering system that can decouple the aim direction from the character's move direction. The SetSteering behaviour is used set a speed at which the body rotates to match the aim direction.

Inspector
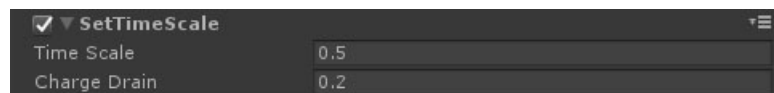
Properties

NAME	TYPE	DESCRIPTION
On Enter	Dropdown	What to do to the steering rate on entering the state.
On Exit	Dropdown	What to do to the steering rate on exiting the state.
Entry Value	Float	The value to set the steering rate to on entering the state.
Exit Value	Float	The value to set the steering rate to on exiting the state.

See Also

NeoCharacterController

MotionController

SetTargetHeight MotionGraphBehaviour

Overview

The SetTargetHeight behaviour signals the desired character height multiplier for the controller to lerp to.

Inspector

Properties

NAME	TYPE	DESCRIPTION
When	Dropdown	When should the target height be set. Options are **EnterAndExit**, **EnterOnly**, **ExitOnly**.
On Enter Value	Float	The character height multiplier (standing height) to set on entering this state (if *when* is set to **EnterandExit** or **EnterOnly**).
On Exit Value	Float	The character height multiplier (standing height) to set on exiting this state (if *when* is set to **EnterandExit** or **ExitOnly**).
Resize Duration	Float	The time taken to change heights.
From Point	Dropdown	Where is the character resized from. Available options are: • **Automatic** will scale from the bottom if grounded and the top if airborne and crouch jumping is enabled. • **Bottom** always resizes from the bottom of the character. • **Top** always resizes from the top of the character.

See Also

MotionController

SetTimeScale MotionGraphBehaviour

Overview

The SetTimeScale behaviour is used to trigger slow motion effects when entering a motion state.

Inspector
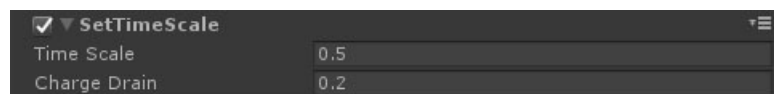
Properties

NAME	TYPE	DESCRIPTION
Time Scale	Float	The target timescale to set. Will be reset to 1 on exit.
Charge Drain	Float	The amount of charge drained per (real, unscaled) second.

See Also

MotionController

SetWieldableStance MotionGraphBehaviour

Overview

The SetWieldableStance behaviour is used to choose the pose that a wieldable inventory item such as a firearm or melee weapon is held in. This can be used to telegraph when the character is crouching or falling, for example.

Inspector
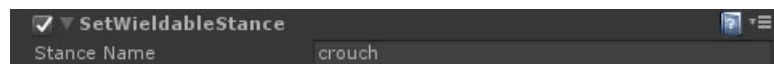
Properties

NAME	TYPE	DESCRIPTION
Stance Name	String	The name of the stance to use. The wieldable item needs a WieldableStanceManager component, with a stance that has this name.

See Also

Inventory

SlidingAudio MotionGraphBehaviour

Overview

The SlidingAudio behaviour plays a looping audio clip based on the ground contact surface, pitch shifted based on the character speed.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Audio Data	SurfaceAudioData	The surface audio library for the slide audio clips.
Surface Test Interval	Int	Every n-th frame, the behaviour will check what the ground surface is and switch the sliding audio if required.
Minimum Speed	Float	The speed below which the pitch will be at its minimum.
Maximum Speed	Float	The speed above which the pitch will be at its maximum.
Minimum Pitch	Float	The minimum pitch for the slide loop.
Maximum Pitch	Float	The maximum pitch for the slide loop.
Max Ray Distance	Float	The downward raycast length for the ground surface test.
Ray Offset	Float	The vertical offset above the absolute bottom of the character collider to start the downward surface test.

See Also

SurfaceAudioData

SurfaceAudio MotionGraphBehaviour
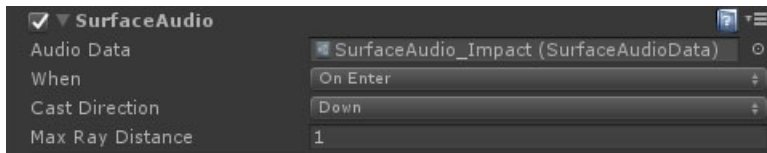
Overview

The SurfaceAudio behaviour plays an audio clip based on the current ground surface.

Inspector

Properties

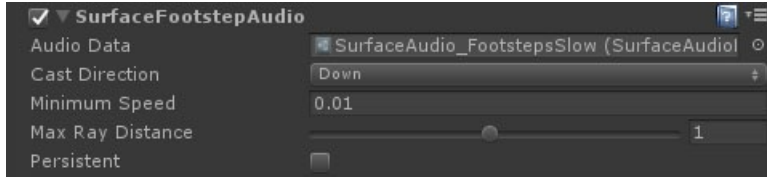| NAME	TYPE	DESCRIPTION
Audio Data	SurfaceAudioData	The surface audio library for the audio clips.
When	Dropdown	When should the audio be played. Options are **EnterAndExit**, **EnterOnly**, **ExitOnly**.
Cast Direction	Dropdown	The direction to perform the footstep surface check. Available options are: • **Down** casts downwards (based on the character up vector). • **LocalVector** casts based on the *Cast Vector* property in local space. • **WorldVector** casts based on the *Cast Vector* property in world space. • **WorldParameter** casts based on the vector value of the *Vector Parameter* property in world space. • **WorldParameterInverse** casts based on the flipped vector value of the *Vector Parameter* property in world space. • **LocalParameter** casts based on the vector value of the *Vector Parameter* property in local space. • **LocalParameterInverse** casts based on the flipped vector value of the *Vector Parameter* property in local space.
Cast Vector	Vector3	The direction to cast in for footstep surface checks.
Vector Parameter	VectorParameter	A vector parameter containing the direction vector to cast in for footstep surface checks.
Max Ray Distance	Float	The downward raycast length for the ground surface test.

See Also

Surfaces

Audio Systems

SurfaceFootstepAudio MotionGraphBehaviour

Overview

The SurfaceFootstepAudio behaviour interacts with the [SurfaceFootstepAudioSystem][3] monobehaviour on the root of the character to control how it processes footsteps.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Audio Data	SurfaceAudioData	The surface audio library for the slide audio clips.
Cast Direction	Dropdown	The direction to perform the footstep surface check. Available options are: • **Down** casts downwards (based on the character up vector). • **LocalVector** casts based on the *Cast Vector* property in local space. • **WorldVector** casts based on the *Cast Vector* property in world space. • **WorldParameter** casts based on the vector value of the *Vector Parameter* property in world space. • **WorldParameterInverse** casts based on the flipped vector value of the *Vector Parameter* property in world space. • **LocalParameter** casts based on the vector value of the *Vector Parameter* property in local space. • **LocalParameterInverse** casts based on the flipped vector value of the *Vector Parameter* property in local space.
Cast Vector	Vector3	The direction to cast in for footstep surface checks.
Transform Parameter	TransformParameter	A parameter containing the transform to use for the cast's space. If none is selected or the parameter value is null then the cast is performed in world space.
Vector Parameter	VectorParameter	A vector parameter containing the direction vector to cast in for footstep surface checks.
Minimum Speed	Float	The speed below which no footstep audio will be played.
Max Ray Distance	Float	The downward raycast length for the ground surface test.
Persistent	Boolean	If persistent is true, then exiting this state or sub-graph will keep the footstep settings until they are explicitly set from elsewhere.

See Also

[SurfaceFootstepAudioSystem][3]

SurfaceAudioData

TimeOps MotionGraphBehaviour

Overview

The TimeOps behaviour modifies a float parameter on the graph based on the Time. It can be used to record the time that specific events occur or to track the time spent in one or more states or sub-graphs.

Inspector

Properties

NAME	TYPE	DESCRIPTION
What	Dropdown	How should the parameter be modified. Available options are: • **Add Elapsed Time** adds elapsed time to the output parameter. • **Add Elapsed Time Scaled** adds elapsed time, multiplier by a scale factor to the output parameter. • **Record Entry Time** sets the output parameter to the current time on entering the state or sub-graph. • **Record Exit Time** sets the output parameter to the current time on exiting the state or sub-graph. • **Record Time** sets the output parameter to the current time every frame.
Output	FloatParameter	The parameter to modify.
Multiplier	Float	A multiplier to apply to the output if **What** is set to **Add Elapsed Time Scaled**.

See Also

Motion Graph Parameters And Data

TrackStepsBehaviour MotionGraphBehaviour

Overview

The TrackStepsBehaviour behaviour is used to provide a consistent step count for use by the various bob effects such as PositionBob and RotationBob. Stride length will be reset to the previous value on exiting the behaviour, so these behaviours can be nested.

Inspector
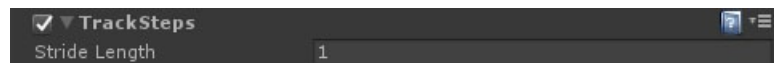
Properties

NAME	TYPE	DESCRIPTION
Stride Length	Float	The travel distance for one stride.

See Also

PositionBob

RotationBob

UnlockInventorySelection MotionGraphBehaviour

Overview

The UnlockInventorySelection behaviour will unlock the character's inventory selection after being locked by a LockInventorySelection behaviour.
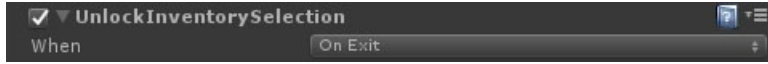
Inspector

Properties

NAME	TYPE	DESCRIPTION
When	Dropdown	When should the inventory selection be set. Options are: **OnEnter** and **OnExit**.

See Also

Inventory

LockInventorySelection

AirTime MotionGraphCondition

Overview

The AirTime condition checks against the airtime value of the character's NeoCharacterController.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Comparison	Dropdown	The comparison type between the airtime and value. Options are **EqualTo (=)**, **NotEqualTo (!=)**, **GreaterThan (>)**, **GreaterOrEqual (>=)**, **LessThan (<)**, **LessOrEqual (<=)**.
Value	Float	The value to check against.

See Also

NeoCharacterController

CapsuleCast MotionGraphCondition

Overview

The CapsuleCast condition performs a cast of the NeoCharacterController capsule in the specified direction.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Cast Vector	Vector3	The direction and distance to cast. The vector does not have to be normalised, as the magnitude will be the maximum distance.
Layer Mask	Layermask	The layers to check against.
Does Hit	Boolean	Is the codition true if the cast hits something or does not.

See Also

Layers And Tags

CapsuleLookahead MotionGraphCondition

Overview

The CapsuleLookahead condition performs a cast of the CharacterController capsule based on either its movement, the direction it's facing, or the input direction.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Lookahead	Dropdown	How the lookahead direction is calculated. The available options are: • **Velocity All Axes** uses the current velocity of the character. • **Velocity Horizontal Plane** uses the character velocity constrained to its horizontal plane. • **Velocity Vertical** uses the character velocity constrained to its up-vector. • **Direction All Axes** uses the direction the character is moving in, but checks a fixed distance ahead. • **Direction Horizontal Plane** uses the character movement direction constrained to its horizontal plane. • **Direction Vertical** uses the character movement direction constrained to its up-vector. • **Input Direction** uses the character input vector transformed into the character's local space.
Lookahead Time	Float	When using the **Velocity** lookahead types, the distance is based on velocity x time.
Lookahead Distance	Float	The cast distance when using the **Direction** or **Input** lookahead types.
Layer Mask	Layermask	The layers to check against.
Does Hit	Boolean	Is the codition true if the cast hits something or does not.

See Also

Layers And Tags

CharacterHeight MotionGraphCondition

Overview

The CharacterHeight condition checks the height of the character capsule, or its multiplier compared to its standing height.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Compare	Dropdown	What to check. Available options are **Multiplier** and **Actual Height**.
Comparison	Dropdown	The comparison type between the character height and value. Options are **EqualTo (=)**, **NotEqualTo (!=)**, **GreaterThan (>)**, **GreaterOrEqual (>=)**, **LessThan (<)**, **LessOrEqual (<=)**.
Value	Float	The value to check against.

See Also

NeoCharacterController

Climbable MotionGraphCondition

Overview

The Climbable condition performs is a complex condition which checks if a character can climb onto a ledge. It has a number of steps:

1. It capsule casts to find the wall surface to climb
2. It sphere casts up to check available head height
3. It casts forward into the wall plane at either the maximum climb height or the point the head cast hit in order to check that the space is clear to climb

The condition outputs the wall normal found in step 1 to a Vector Parameter which can then be used by the Mantle state to actually perform the climbing action. The mantle state performs its own checks each frame so you do not need to keep testing if a wall is climbable once the mantle state has control.

Since the climbable condition uses a number of complex checks, it is worth putting any cheaper conditions ahead of it in the transition in order to narrow down the situations where it will be checked.

Inspector
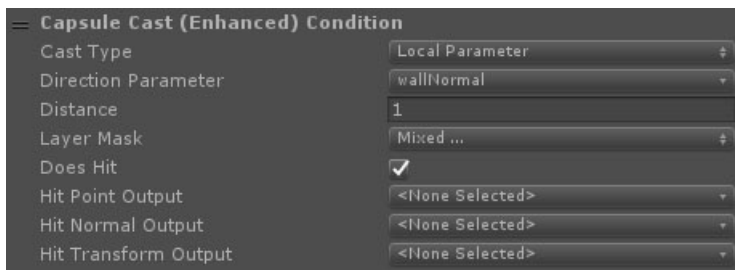
Properties

NAME	TYPE	DESCRIPTION
Output Wall Normal	Vector Parameter	The vector parameter used to store the wall normal.
Check Direction	Dropdown	The direction to perform the initial wall check in **Yaw Forward** checks straight ahead of the character. **Inverse Wall Normal** reads the value of the wall normal parameter above and flips it, casting back into the wall.
Check Distance	Float Data	The distance of the initial capsule cast to check wall contact. It is advised to set up a motion data entry and share it with the Mantle state to ensure they have the same value.
Wall Collision Mask	Layermask	The layers to check against.
Max Climb Height	Float	The maximum height the character can pull itself up to reach the top surface.
Climb Forward	Float	After reaching the top surface, this is the distance to move in past the edge before the state completes.
Output Climb Height	Float Parameter	An optional parameter to store the climb height.

See Also

Motion Graph Parameters And Data

Layers And Tags

CollisionFlags MotionGraphCondition

Overview

The CollisionFlags condition checks the NeoCharacterController collision flags that resulted from the last movement frame.

Inspector
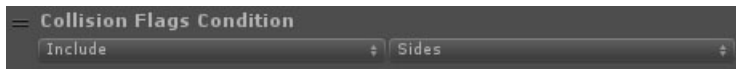
Properties

NAME	TYPE	DESCRIPTION
Comparison	Dropdown	Check if the NeoCharacterController collision flags **Include** or **Exclude** the "Compare To" flags.
Compare To	NeoCharacterControllerHit	The collision flags to check against.

Note

If you want to check against individual sides, then use the **Mask** values. Using the non-mask versions also sets the **Sides** flag which is shared between all of them.

See Also

NeoCharacterController

CompareFloats CapsuleCastCondition

Overview

The CompareFloats condition compares two float parameters.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Parameter A	FloatParameter	The left hand side parameter of the comparison
Comparison	Dropdown	The comparison between the two parameters for which the condition is true. Options are **Equal To (=)**, **Not Equal To (!=)**, **Greater Than (>)**, **Greater Or Equal To (>=)**, **Less Than (<)**, **Less Than Or Equal To (<=)**.
Parameter B	FloatParameter	The right hand side parameter of the comparison

See Also

Motion Graph Parameters And Data

CompareInts CapsuleCastCondition

Overview

The CompareInts condition compares two int parameters.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Parameter A	IntParameter	The left hand side parameter of the comparison
Comparison	Dropdown	The comparison between the two parameters for which the condition is true. Options are **Equal To (=)**, **Not Equal To (!=)**, **Greater Than (>)**, **Greater Or Equal To (>=)**, **Less Than (<)**, **Less Than Or Equal To (<=)**.
Parameter B	IntParameter	The right hand side parameter of the comparison

See Also
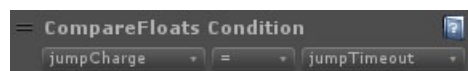
Motion Graph Parameters And Data

CompareSwitches CapsuleCastCondition

Overview

The CompareSwitches condition compares two switch parameters.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Parameter A	SwitchParameter	The left hand side parameter of the comparison
Comparison	Dropdown	The comparison between the two parameters for which the condition is true. Options are **Equal To (=)**, **Not Equal To (!=)**.
Parameter B	SwitchParameter	The right hand side parameter of the comparison

See Also
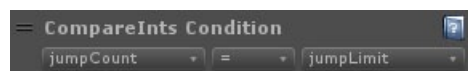
Motion Graph Parameters And Data

CompareTime MotionGraphCondition

Overview

The CompareTime condition is used in conjunction with the TimeOps motion graph behaviour to check the current time against the parameter values it outputs.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Time Parameter	FloatParameter	The float parameter with the time stored in it byyt TimeOps.
Comparison	Dropdown	The comparison type between the parameter and value. Options are **GreaterThan (>)**, **GreaterOrEqual (>=)**, **LessThan (<)**, **LessOrEqual (<=)**.
Value	Float	The value to check against.

See Also

Motion Graph Parameters And Data

TimeOps MotionGraphBehaviour

Completed MotionGraphCondition

Overview

The Completed condition checks the current state's completed flag. If the state has completed, the condition is true.

Inspector

Properties

The Completed condition has no properties exposed in the motion graph inspector.

See Also

ConditionGroup MotionGraphCondition

Overview

The ConditionGroup condition is used to group conditions together to allow for more complex rules than any and all. For example, when swimming on the surface of a water zone, you might transition to the underwater state if you **either** press crouch **or** look down **and** press forwards.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Condition Group	Dropdown	A dropdown that lists the available groups on the connection or allows you to create a new one.

Note

The result of a condition group is recorded when it is evaluated, so you will not be able to create infinite loops where group A checks group B which checks group A again.

See Also

Motion Graph Conditions

Debug MotionGraphCondition

Overview

The Debug condition is only used to check the float of a motion graph. Add it to connections that are not behaving as expected, before and after the problem condition with a message to help check when the condition is hit. Do not leave debug conditions in a complete motion graph.

Inspector
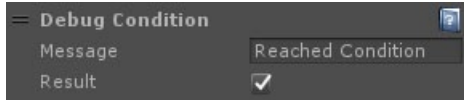
Properties

NAME	TYPE	DESCRIPTION
Message	String	A message to send to the debug console when this condition is checked.
Result	Boolean	The result the condition should give.

See Also

Direction MotionGraphCondition

Overview

The Direction condition checks the specified vector parameter against the character's direction

Inspector

Properties

NAME	TYPE	DESCRIPTION
Parameter	VectorParameter	The vector parameter to check against.
Compare	Dropdown	What direction to compare against the parameter. Available options are: • **YawVsVector** checks the character's yaw forward direction against the parameter. • **YawVsHorizontal** checks the character's yaw forward direction against the parameter aligned to the character's horizontal plane. • **AimVsVector** checks the character's aim direction against the parameter. • **VelocityVsVector** checks the character's move direction against the parameter. • **VelocityVsHorizontal** checks the character's movement direction against the horizontal aligned parameter. • **InputVsHorizontal** checks the character's input direction (aligned to the character) against the horizontal aligned parameter.
Comparison	Dropdown	The comparison type between the parameter and value. This can be less than **"<"** or greater than **">"**.
Angle	Float	The angle value to check against.

See Also

NeoCharacterController

Motion Graph Parameters And Data

ElapsedTime MotionGraphCondition

Overview

The Elapsed Time condition checks how long the motion graph has been in the current state.

Inspector

Properties

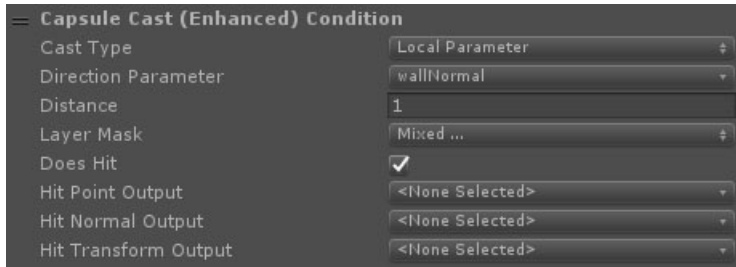| NAME	TYPE	DESCRIPTION
Timeout Value	Float	The time after which the condition will be valid. If the parameter is set, this will be ignored.
Timeout Property	FloatProperty	An optional parameter containg a value for the time after which the condition will be valid. Selecting a parameter will remove the value field until this parameter is removed again.

See Also

Motion Graph Parameters And Data

EnhancedCapsuleCast MotionGraphCondition

Overview

The EnhancedCapsuleCast condition performs a cast of the NeoCharacterController capsule in the specified direction and then outputs the results to motion graph parameters for use in other conditions and states.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Cast Type	Dropdown	What to use for the cast vector. Available options are: • **LocalVector** uses a preset vector relative to the character. Magnitude is distance. • **WorldVector** uses a preset vector in world space. Magnitude is distance. • **LocalParameter** uses a vector parameter to get the direction relative to the character and casts a set distance. • **LocalParameterInverse** uses a vector parameter to get the direction relative to the character (reversed) and casts a set distance. • **WorldParameter** uses a vector parameter to get the direction in world space and casts a set distance. • **WorldParameterInverse** uses a vector parameter to get the direction (reversed) in world space and casts a set distance.
Cast Vector	Vector3	The direction and distance to cast. The vector does not have to be normalised, as the magnitude will be the maximum distance. Visible if Cast Type is set to **World Vector** or **Local Vector**.
Direction Parameter	VectorParameter	The direction to cast in. Only visible with Cast Vector set to one of the **Parameter** settings.
Distance	Float	The distance to cast. Only visible with Cast Vector set to one of the **Parameter** settings.
Layer Mask	Layermask	The layers to check against.
Does Hit	Boolean	Is the codition true if the cast hits something or does not.
Hit Point Output	Vector Parameter	An optional graph parameter to output the capsule cast hit point to.
Hit Normal Output	Vector Parameter	An optional graph parameter to output the capsule cast hit normal to.
Hit Transform Output	Transform Parameter	An optional graph parameter to output the capsule cast hit transform to.

See Also

Motion Graph Parameters And Data

Layers And Tags

EnhancedCapsuleLookahead MotionGraphCondition

Overview

The EnhancedCapsuleLookahead condition performs a cast of the NeoCharacterController capsule based on either its movement, the direction it's facing, or the input direction. The cast results are output to motion graph parameters for later use.

Inspector

Properties

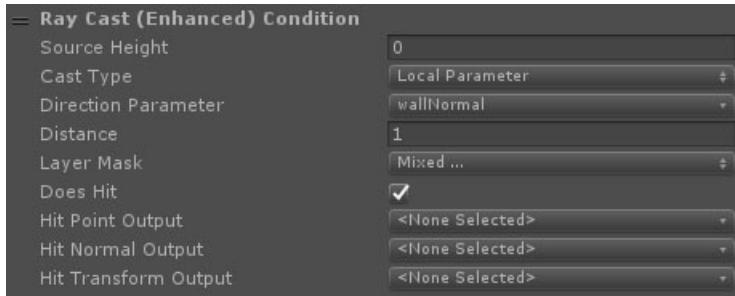| NAME	TYPE	DESCRIPTION
Lookahead	Dropdown	How the lookahead direction is calculated. The available options are: • **Velocity All Axes** uses the current velocity of the character. • **Velocity Horizontal Plane** uses the character velocity constrained to its horizontal plane. • **Velocity Vertical** uses the character velocity constrained to its up-vector. • **Direction All Axes** uses the direction the character is moving in, but checks a fixed distance ahead. • **Direction Horizontal Plane** uses the character movement direction constrained to its horizontal plane. • **Direction Vertical** uses the character movement direction constrained to its up-vector. • **Input Direction** uses the character input vector transformed into the character's local space.
Lookahead Time	Float	When using the **Velocity** lookahead types, the distance is based on velocity x time.
Lookahead Distance	Float	The cast distance when using the **Direction** or **Input** lookahead types.
Layer Mask	Layermask	The layers to check against.
Does Hit	Boolean	Is the codition true if the cast hits something or does not.
Hit Point Output	Vector Parameter	An optional graph parameter to output the capsule cast hit point to.
Hit Normal Output	Vector Parameter	An optional graph parameter to output the capsule cast hit normal to.
Hit Transform Output	Transform Parameter	An optional graph parameter to output the capsule cast hit transform to.

See Also

Motion Graph Parameters And Data

Layers And Tags

EnhancedRayCast MotionGraphCondition

Overview

The EnhancedRayCast condition performs a raycast from the specified point on the character. The cast results are output to motion graph parameters for later use.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Source Height	Float	The point on the character capsule to cast from. 0 is the base of the capsule centerline. 1 is the top of the capsule centerline.
Cast Type	Dropdown	What to use for the cast vector. Available options are: • **LocalVector** uses a preset vector relative to the character. Magnitude is distance. • **WorldVector** uses a preset vector in world space. Magnitude is distance. • **LocalParameter** uses a vector parameter to get the direction relative to the character and casts a set distance. • **LocalParameterInverse** uses a vector parameter to get the direction relative to the character (reversed) and casts a set distance. • **WorldParameter** uses a vector parameter to get the direction in world space and casts a set distance. • **WorldParameterInverse** uses a vector parameter to get the direction (reversed) in world space and casts a set distance.
Cast Vector	Vector3	The direction and distance to cast. The vector does not have to be normalised, as the magnitude will be the maximum distance. Visible if Cast Type is set to **World Vector** or **Local Vector**.
Direction Parameter	VectorParameter	The direction to cast in. Only visible with Cast Vector set to one of the **Parameter** settings.
Distance	Float	The distance to cast. Only visible with Cast Vector set to one of the **Parameter** settings.
Layer Mask	Layermask	The layers to check against.
Does Hit	Boolean	Is the codition true if the cast hits something or does not.
Hit Point Output	Vector Parameter	An optional graph parameter to output the capsule cast hit point to.
Hit Normal Output	Vector Parameter	An optional graph parameter to output the capsule cast hit normal to.

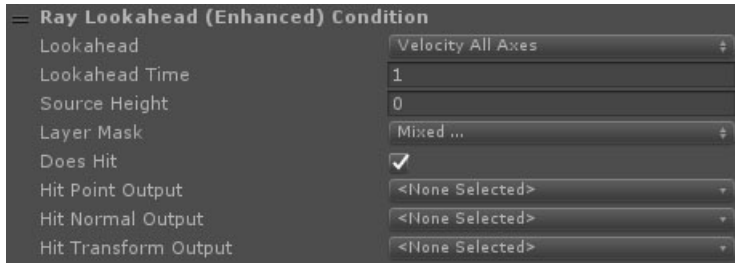| NAME	TYPE	DESCRIPTION
Hit Transform Output	Transform Parameter	An optional graph parameter to output the capsule cast hit transform to.

See Also

Motion Graph Parameters And Data

Layers And Tags

EnhancedRayLookahead MotionGraphCondition

Overview

The EnhancedRayLookahead condition performs a raycast from the specified point on the character and based on either its movement, the direction it's facing, or the input direction. The cast results are output to motion graph parameters for later use.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Lookahead	Dropdown	How the lookahead direction is calculated. The available options are: • **Velocity All Axes** uses the current velocity of the character. • **Velocity Horizontal Plane** uses the character velocity constrained to its horizontal plane. • **Velocity Vertical** uses the character velocity constrained to its up-vector. • **Direction All Axes** uses the direction the character is moving in, but checks a fixed distance ahead. • **Direction Horizontal Plane** uses the character movement direction constrained to its horizontal plane. • **Direction Vertical** uses the character movement direction constrained to its up-vector. • **Input Direction** uses the character input vector transformed into the character's local space.
Lookahead Time	Float	When using the **Velocity** lookahead types, the distance is based on velocity x time.
Lookahead Distance	Float	The cast distance when using the **Direction** or **Input** lookahead types.
Source Height	Float	The point on the character capsule to cast from. 0 is the base of the capsule centerline. 1 is the top of the capsule centerline.
Layer Mask	Layermask	The layers to check against.
Does Hit	Boolean	Is the codition true if the cast hits something or does not.
Hit Point Output	Vector Parameter	An optional graph parameter to output the capsule cast hit point to.
Hit Normal Output	Vector Parameter	An optional graph parameter to output the capsule cast hit normal to.
Hit Transform Output	Transform Parameter	An optional graph parameter to output the capsule cast hit transform to.

See Also

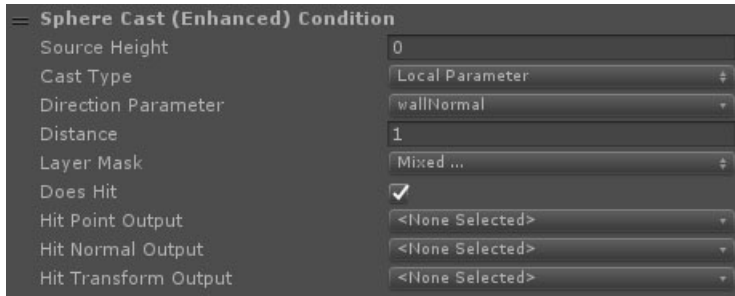Motion Graph Parameters And Data

Layers And Tags

EnhancedSphereCast MotionGraphCondition

Overview

The EnhancedSphereCast condition performs a spherecast from the specified point on the character. The cast results are output to motion graph parameters for later use.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Source Height	Float	The point on the character capsule to cast from. 0 is the base of the capsule centerline. 1 is the top of the capsule centerline.
Cast Type	Dropdown	What to use for the cast vector. Available options are: • **LocalVector** uses a preset vector relative to the character. Magnitude is distance. • **WorldVector** uses a preset vector in world space. Magnitude is distance. • **LocalParameter** uses a vector parameter to get the direction relative to the character and casts a set distance. • **LocalParameterInverse** uses a vector parameter to get the direction relative to the character (reversed) and casts a set distance. • **WorldParameter** uses a vector parameter to get the direction in world space and casts a set distance. • **WorldParameterInverse** uses a vector parameter to get the direction (reversed) in world space and casts a set distance.
Cast Vector	Vector3	The direction and distance to cast. The vector does not have to be normalised, as the magnitude will be the maximum distance. Visible if Cast Type is set to **World Vector** or **Local Vector**.
Direction Parameter	VectorParameter	The direction to cast in. Only visible with Cast Vector set to one of the **Parameter** settings.
Distance	Float	The distance to cast. Only visible with Cast Vector set to one of the **Parameter** settings.
Layer Mask	Layermask	The layers to check against.
Does Hit	Boolean	Is the codition true if the cast hits something or does not.
Hit Point Output	Vector Parameter	An optional graph parameter to output the capsule cast hit point to.
Hit Normal Output	Vector Parameter	An optional graph parameter to output the capsule cast hit normal to.

NAME	TYPE	DESCRIPTION
Hit Transform Output	Transform Parameter	An optional graph parameter to output the capsule cast hit transform to.

See Also

Motion Graph Parameters And Data

Layers And Tags

NAME	TYPE	DESCRIPTION
Hit Transform Output	Transform Parameter	An optional graph parameter to output the capsule cast hit transform to.

EnhancedSphereLookahead MotionGraphCondition

Overview

The EnhancedSphereLookahead condition performs a spherecast from the specified point on the character and based on either its movement, the direction it's facing, or the input direction. The cast results are output to motion graph parameters for later use.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Lookahead	Dropdown	How the lookahead direction is calculated. The available options are: • **Velocity All Axes** uses the current velocity of the character. • **Velocity Horizontal Plane** uses the character velocity constrained to its horizontal plane. • **Velocity Vertical** uses the character velocity constrained to its up-vector. • **Direction All Axes** uses the direction the character is moving in, but checks a fixed distance ahead. • **Direction Horizontal Plane** uses the character movement direction constrained to its horizontal plane. • **Direction Vertical** uses the character movement direction constrained to its up-vector. • **Input Direction** uses the character input vector transformed into the character's local space.
Lookahead Time	Float	When using the **Velocity** lookahead types, the distance is based on velocity x time.
Lookahead Distance	Float	The cast distance when using the **Direction** or **Input** lookahead types.
Source Height	Float	The point on the character capsule to cast from. 0 is the base of the capsule centerline. 1 is the top of the capsule centerline.
Layer Mask	Layermask	The layers to check against.
Does Hit	Boolean	Is the codition true if the cast hits something or does not.
Hit Point Output	Vector Parameter	An optional graph parameter to output the capsule cast hit point to.
Hit Normal Output	Vector Parameter	An optional graph parameter to output the capsule cast hit normal to.
Hit Transform Output	Transform Parameter	An optional graph parameter to output the capsule cast hit transform to.

See Also

Motion Graph Parameters And Data

Float MotionGraphCondition

Overview

The Float condition checks the specified parameter against a value.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Parameter	FloatParameter	The parameter to check.
Comparison	Dropdown	The comparison type between the parameter and value. Options are **EqualTo (=)**, **NotEqualTo (!=)**, **GreaterThan (>)**, **GreaterOrEqual (>=)**, **LessThan (<)**, **LessOrEqual (<=)**.
Value	Float	The value to check against.

See Also

Motion Graph Parameters And Data

GroundContact MotionGraphCondition

Overview

The GroundContact condition checks if the character is airborne or touching the ground.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Equals	Boolean	Is the condition true if the character is touching the ground or not.

See Also

MotionController

GroundNormal MotionGraphCondition

Overview

The GroundNormal condition is used to check against the angle of the character ground contact.

This condition differs from the GroundSurfaceNormal condition in that it is only aware of the point of contact and does not take the shape of the collision geometry into account. As a character walks off the edge of a flat surface, this angle will slope down, whereas the ground surface normal will stay pointing straight up until the character loses contact with the surface.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Comparison	Dropdown	The comparison type. Options are **GreaterThan ()**, **GreaterOrEqual ()**, **LessThan ()**, **LessOrEqual ()**, **EqualTo ()**.
Angle	Float	The ground slope angle in degrees from horizontal.

See Also

MotionController

GroundSurfaceNormal Condition

GroundSurfaceNormal MotionGraphCondition

Overview

The GroundSurfaceNormal condition is used to check against the slope of the ground surface the character is currently standing on. If the character is in contact with a flat surface then the resulting angle is the angle of the slope. If the character is in contact with an edge then the resulting angle is the slope of the top face the edge belongs to.

This condition differs from the GroundNormal condition which only takes into account the point of contact and not the shape of the collision geometry.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Comparison	Dropdown	The comparison type. Options are **GreaterThan ()**, **GreaterOrEqual ()**, **LessThan ()**, **LessOrEqual ()**, **EqualTo ()**.
Angle	Float	The ground slope angle in degrees from horizontal.

See Also

MotionController

GroundNormal Condition

HeightRestriction MotionGraphCondition

Overview

The HeightRestriction condition checks if the character height is restricted by a ceiling. This is useful for situations such as preventing the character moving to a sprint state while stuck in an air vent.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Target Height	Float	The target height multiplier for the character to check against.
Is Blocked	Boolean	Is the condition true if the character height is blocked or not blocked.

See Also

MotionController

InputVector MotionGraphCondition

Overview

The InputVector condition checks against the input vector provided by the motion controller.

Inspector

Properties

NAME	TYPE	DESCRIPTION
InputComponent	Dropdown	What component of the input to check. Available options are: • **Magnitude** 0 is no input, while 1 is one of the direction keys, or full tilt on an analog stick. • **InputY** the forward or back amount (1 is forward, -1 is back). • **InputX** the left or right amount (1 is right, -1 is left). • **AbsoluteY** the absolute forward or back amount. • **AbsoluteX** the absolute left or right amount.
Comparison	Dropdown	The comparison type between the actual input vector magnitude and the value specified. Options are greater than (**>**), less than (**<**).
Value	Float	The value to compare against.

See Also

Int MotionGraphCondition

Overview

The Int condition checks the specified parameter against a value.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Parameter	IntParameter	The parameter to check.
Comparison	Dropdown	The comparison type between the propery and value. Options are **EqualTo (=)**, **NotEqualTo (!=)**, **GreaterThan (>)**, **GreaterOrEqual (>=)**, **LessThan (<)**, **LessOrEqual (<=)**.
Value	Int	The value to check against.

See Also

Motion Graph Parameters And Data

Pitch MotionGraphCondition

Overview

The Pitch condition checks pitch (up/down rotation) of the character's aim controller.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Comparison	Dropdown	The comparison type between the parameter and value. Options are **EqualTo (=)**, **NotEqualTo (!=)**, **GreaterThan (>)**, **GreaterOrEqual (>=)**, **LessThan (<)**, **LessOrEqual (<=)**.
Angle	Float	The pitch angle value to check against. 90 is straight up and -90 is straight down.

See Also

Aim Controllers

RayCast MotionGraphCondition

Overview

The RayCast condition performs a cast in the specified direction and from the specified source.

Inspector
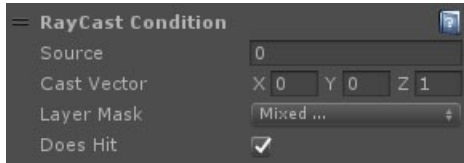
Properties

Properties

NAME	TYPE	DESCRIPTION
Normalise Height	Float	The point on the character capsule to cast from. 0 is the base of the capsule centerline. 1 is the top of the capsule centerline.
Cast Vector	Vector3	The direction and distance to cast relative to the character. The vector does not have to be normalised, as the magnitude will be the maximum distance.
Layer Mask	Layermask	The layers to check against.
Does Hit	Boolean	Is the condition true if the cast hits something or if it does not.

See Also

Layers And Tags
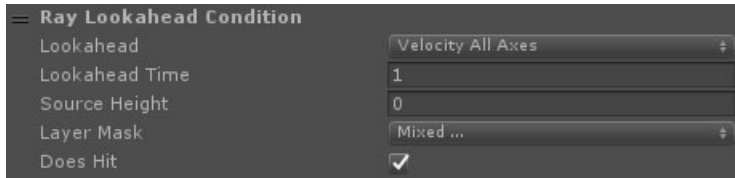
RayLookahead MotionGraphCondition

Overview

The RayLookahead condition performs a raycast from the specified point on the character and based on either its movement, the direction it's facing, or the input direction.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Lookahead	Dropdown	How the lookahead direction is calculated. The available options are: • **Velocity All Axes** uses the current velocity of the character. • **Velocity Horizontal Plane** uses the character velocity constrained to its horizontal plane. • **Velocity Vertical** uses the character velocity constrained to its up-vector. • **Direction All Axes** uses the direction the character is moving in, but checks a fixed distance ahead. • **Direction Horizontal Plane** uses the character movement direction constrained to its horizontal plane. • **Direction Vertical** uses the character movement direction constrained to its up-vector. • **Input Direction** uses the character input vector transformed into the character's local space.
Lookahead Time	Float	When using the **Velocity** lookahead types, the distance is based on velocity x time.
Lookahead Distance	Float	The cast distance when using the **Direction** or **Input** lookahead types.
Source Height	Float	The point on the character capsule to cast from. 0 is the base of the capsule centerline. 1 is the top of the capsule centerline.
Layer Mask	Layermask	The layers to check against.
Does Hit	Boolean	Is the codition true if the cast hits something or does not.

See Also

Motion Graph Parameters And Data

Layers And Tags

ScriptedComponent MotionGraphCondition

Overview

The ScriptedComponent condition attaches to a component on the controller's game object. Each time the condition is checked, the component is queried and the result returned. Valid components must implement the `IScriptedComponentCondition` interface.

Inspector
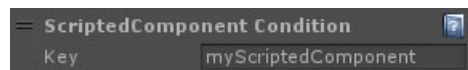
Properties

NAME	TYPE	DESCRIPTION
Key	String	The name of the specific component. This is used to distinguish between multiple components on the same game object.

See Also

SphereCast MotionGraphCondition

Overview

The SphereCast condition performs a cast in the specified direction and from the specified source.

Inspector
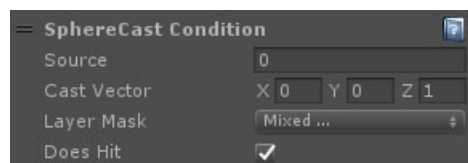
Properties

NAME	TYPE	DESCRIPTION
Normalise Height	Float	The point on the character capsule to cast from. 0 is the base of the capsule. 1 is the top of the capsule.
Cast Vector	Vector3	The direction and distance to cast relative to the character. The vector does not have to be normalised, as the magnitude will be the maximum distance.
Layer Mask	Layermask	The layers to check against.
Does Hit	Boolean	Is the condition true if the cast hits something or if it does not.

See Also

Layers And Tags

SphereLookahead MotionGraphCondition

Overview

The SphereLookahead condition performs a spherecast from the specified point on the character and based on either its movement, the direction it's facing, or the input direction.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Lookahead	Dropdown	How the lookahead direction is calculated. The available options are: • **Velocity All Axes** uses the current velocity of the character. • **Velocity Horizontal Plane** uses the character velocity constrained to its horizontal plane. • **Velocity Vertical** uses the character velocity constrained to its up-vector. • **Direction All Axes** uses the direction the character is moving in, but checks a fixed distance ahead. • **Direction Horizontal Plane** uses the character movement direction constrained to its horizontal plane. • **Direction Vertical** uses the character movement direction constrained to its up-vector. • **Input Direction** uses the character input vector transformed into the character's local space.
Lookahead Time	Float	When using the **Velocity** lookahead types, the distance is based on velocity x time.
Lookahead Distance	Float	The cast distance when using the **Direction** or **Input** lookahead types.
Source Height	Float	The point on the character capsule to cast from. 0 is the base of the capsule centerline. 1 is the top of the capsule centerline.
Layer Mask	Layermask	The layers to check against.
Does Hit	Boolean	Is the codition true if the cast hits something or does not.

See Also

Motion Graph Parameters And Data

Layers And Tags

Switch MotionGraphCondition

Overview

The Switch condition checks the value of the specified switch parameter.

Inspector
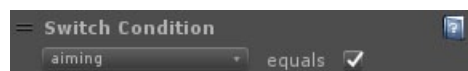
Properties

NAME	TYPE	DESCRIPTION
Parameter	SwitchParameter	The parameter to check.
Equals	Boolean	The value of the switch that the condition should be true for.

See Also

Motion Graph Parameters And Data

Transform MotionGraphCondition

Overview

The Transform condition checks if the specified transform parameter is null or not.

Inspector
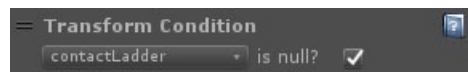
Properties

NAME	TYPE	DESCRIPTION
Parameter	TransformParameter	The parameter to check.
Is Null	Boolean	Is the condition true if the transform is null or not null.

See Also

Motion Graph Parameters And Data

Trigger MotionGraphCondition

Overview

The Trigger condition checks the specified trigger parameter and then resets it if triggered.

Inspector
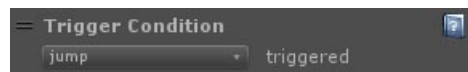
Properties

NAME	TYPE	DESCRIPTION
Parameter	TriggerParameter	The trigger to check.

See Also

Motion Graph Parameters And Data

Vector MotionGraphCondition

Overview

The Vector condition checks the specified parameter against a value.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Parameter	VectorParameter	The parameter to check.
Compare	Dropdown	What element of the vector parameter to check against. The available options are: • **Magnitude** checks the value against the magnitude of the vector. • **X** checks against the vector's x axis. • **Y** checks against the vector's y axis. • **Z** checks against the vector's z axis. • **Character Horizontal** checks against the magnitude of the vector after it has been projected onto the character's horizontal plane. • **Character Up** checks against the magnitude of the vector after it has been projected onto the character's up vector.
Comparison	Dropdown	The comparison type between the parameter and value. Options are **EqualTo (=)**, **NotEqualTo (!=)**, **GreaterThan (>)**, **GreaterOrEqual (>=)**, **LessThan (<)**, **LessOrEqual (<=)**.
Value	Float	The value to check against.

See Also

Motion Graph Parameters And Data

Velocity MotionGraphCondition

Overview

The Velocity condition checks the velocity output of the character's NeoCharacterController.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Compare	Dropdown	What element of the velocity to check against. The available options are: • **Character Speed** checks the magnitude of the velocity vector against the value. • **Horizontal Speed** checks the velocity after it has been projected on the character's horizontal plane. • **Vertical Velocity** checks the velocity after it has been projected on the character's up vector. • **Ground Speed** checks the velocity after it has been projected on the ground normal plane. • **Ground Surface Speed** checks the velocity after it has been projected on the ground surface normal plane. • **Yaw Velocity** checks the velocity of the character along its forward vector. • **Yaw Ground Velocity** checks the velocity of the character along its forward vector projected on the ground normal plane. • **Yaw Ground SurfaceVelocity** checks the velocity of the character along its forward vector projected on the ground surface normal plane.
Comparison	Dropdown	The comparison type between the parameter and value. Options are **EqualTo (=)**, **NotEqualTo (!=)**, **GreaterThan (>)**, **GreaterOrEqual (>=)**, **LessThan (<)**, **LessOrEqual (<=)**.
Value	Float	The value to check against.

See Also

NeoCharacterController

Water MotionGraphCondition

Overview

The Water condition checks the character's position relative to a water zone.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Water Zone	TransformParameter	The transform parameter with the transform of the water zone object. If this is null, the condition returns false.
Check	Dropdown	The comparison type between the character and the water zone. Available options are: • **Above Water Greater Than** checks if the height of the character above the water line against the value. • **Above Water Less Than** checks if the height of the character above the water line against the value. • **Below Water Greater Than** checks if the height of the character below the water line against the value. • **Below Water Less Than** checks if the height of the character below the water line against the value.
Value	Float	The value to check against.

See Also

Motion Graph Parameters And Data

BasicWaterZone MonoBehaviour

Overview

The BasicWaterZone behaviour is used to communicate water properties to the motion graph, including surface height and normal, and flow vectors.

The water zone uses a box collider to detect when the character enters or leaves the zone.

Inspector
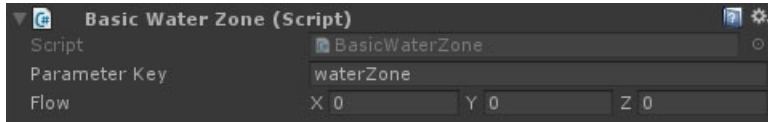
Properties

NAME	TYPE	DESCRIPTION
Parameter Key	string	The name of the transform parameter on the motion graph for any character that enters the water zone. This wil be set with the water zone's transform.
Flow	Vector3	A flow velocity (m/s) for the water zone.

See Also

Motion Graph Parameters And Data

CharacterEventKickTrigger MonoBehaviour

Overview

The CharacterImpactHandler consumes impact events such as bullet hits, and uses them to apply a force to the character's NeoCharacterController.

Inspector
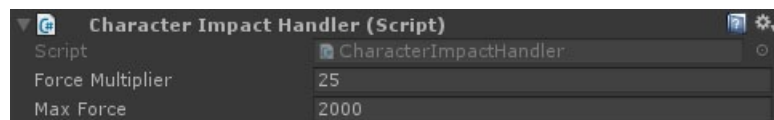
Properties

NAME	TYPE	DESCRIPTION
Force Multiplier	Float	A multiplier to apply to inbound forces for a more exagerrated effect.
Max Force	Float	The maximum force that can be applied to the character.

See Also

NeoCharacterController

ConstantRotatingPlatform MonoBehaviour

Overview

The ConstantRotatingPlatform behaviour is used to create a moving platform that constantly turns at a set rate.

Inspector
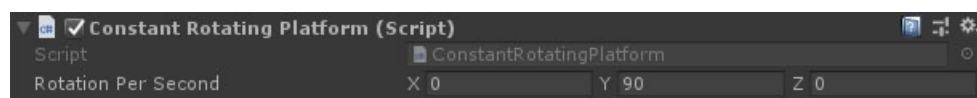
Properties

NAME	TYPE	DESCRIPTION
Rotation Per Second	Vector3	The rotation in degrees on each axis per second.

See Also

Moving Platforms

ContactLadder MonoBehaviour

Overview

The ContactLadder behaviour defines a ladder in the scene.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Trigger Zone	TriggerZone	The trigger area for detecting contact with the ladder.
Rough Collider	BoxCollider	The box collider for the ladder geometry.
Property Key	String	The motion graph parameter name to set the ladder to.
Top	Vector3	The top of the ladder surface relative to the transform position.
Spacing	Float	The spacing between rungs on the ladder.
Length	Float	The length of the ladder along the ladder transform down axis from the top offset.
Width	Float	The width of the ladder surface.

See Also

Ladders

InteractiveLadder

Motion Graph Parameters And Data

DrivenMovingPlatform MonoBehaviour

Overview

The DrivenMovingPlatform behaviour is added to a rigidbody that is driven by physics or a script and turns it into a moving platform. It tracks the position and rotation of the object in a way that the NeoCharacterController can use.

Inspector

Properties

The DrivenMovingPlatform has no properties exposed in the inspector;

See Also

Moving Platforms

NeoCharacterController

DrowningMotionGraphWatcher MonoBehaviour

Overview

The DrowningMotionGraphWatcher watches a float parameter on the character motion graph and if the float value exceeds the specified amount then it starts to apply damage to the player's health manager.

Inspector
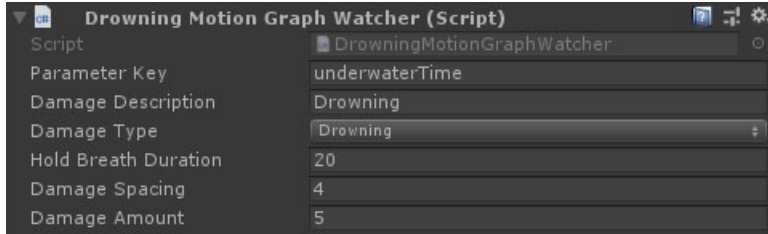
Properties

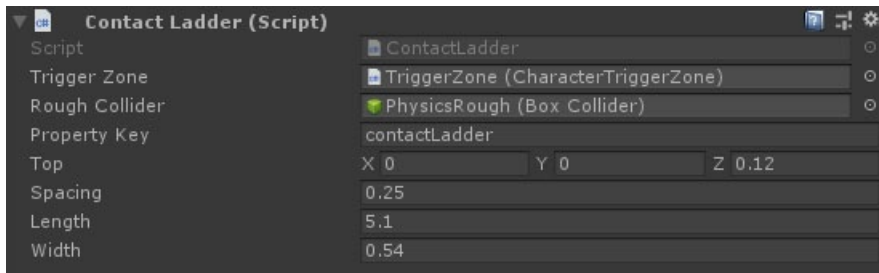| NAME	TYPE	DESCRIPTION
Parameter Key	String	The name of the float parameter on the motion graph that the watcher should track.
Damage Description	String	A descriptive name for the damage that can be used in any HUD logs.
Damage Type	Dropdown	The damage type the drowning should be classed as. Used for filtering the damage.
Hold Breath Duration	Float	The value the parameter needs to reach before damage is applied.
Damage Spacing	Float	Damage will be applied every time the parameter passes an increment of this amount above the drown duration.
Damage Amount	Float	The amount of damage to apply to the character.

See Also

Health and Damage

Motion Graph Parameters And Data

InteractiveLadder MonoBehaviour

Overview

The InteractiveLadder behaviour defines a ladder in the scene.

Inspector
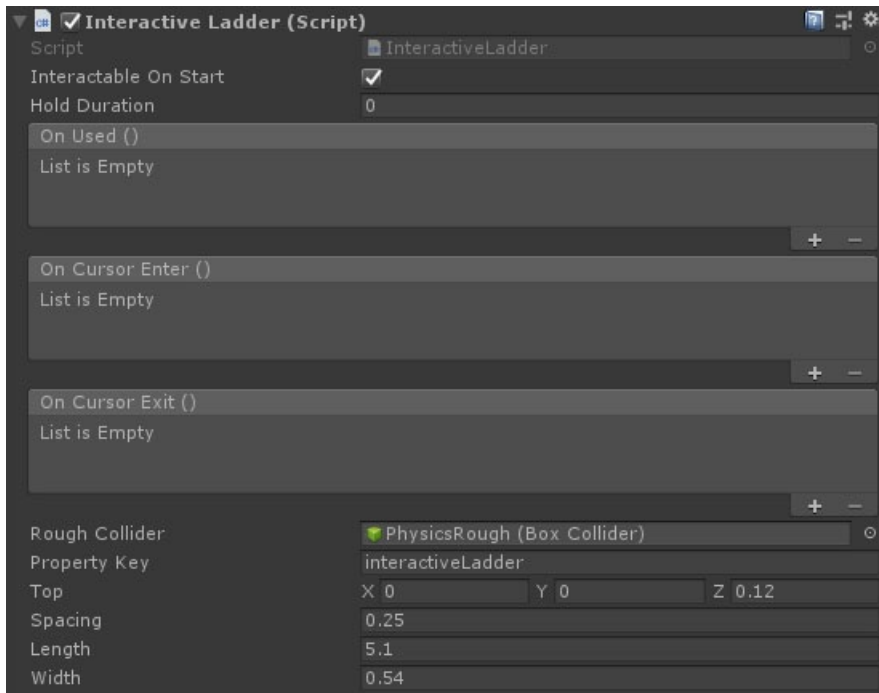
Properties

The InteractiveLadder inherits from the InteractiveObject. Check the reference for information on its properties.

NAME	TYPE	DESCRIPTION
Rough Collider	BoxCollider	The box collider for the ladder geometry.
Property Key	String	The motion graph parameter name to set the ladder to.
Top	Vector3	The top of the ladder surface relative to the transform position.
Spacing	Float	The spacing between rungs on the ladder.
Length	Float	The length of the ladder along the ladder transform down axis from the top offset.
Width	Float	The width of the ladder surface.

See Also

Ladders

Interaction

ContactLadder

Motion Graph Parameters And Data

JumpPad MonoBehaviour

Overview

The JumpPad behaviour detects when a character touches it and passes a move vector to its motion graph.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Parameter Name	String	The motion graph parameter name to set with the jump vector.
Boost Vector	Vector3	The velocity vector to give to the character.
Boost Relative To	Float	The space in which to apply the vector. If this is set to self, the vector will be relative to this object's transform.
Cooldown	Float	The cooldown time to prevent the jumppad registering multiple times.

See Also

Motion Graph Parameters And Data

MotionController Behaviour

Overview

The MotionController is the workhorse of the MotionGraph system. It handles the actual movement of the character, ticks the motion graph, and sends input and information to the motion graph that it needs to react to the player and environment.

Inspector
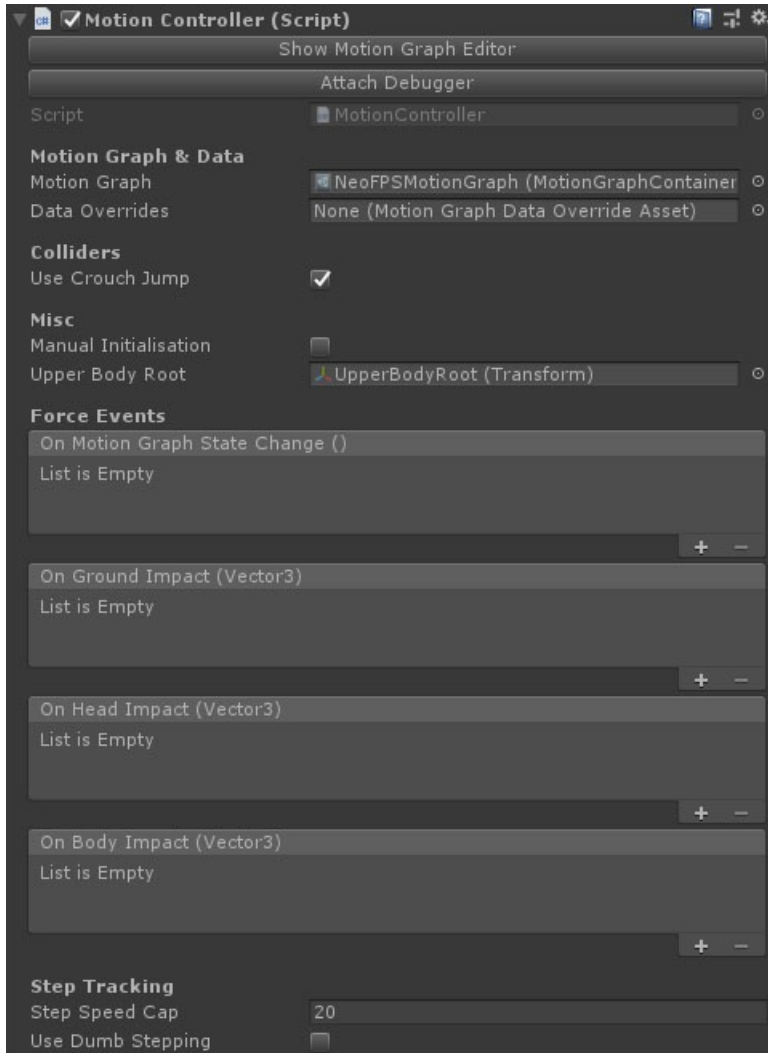
The **Show Motion Graph Editor** button will show the motion graph editor, with this controller's motion graph opened for editing. If the controller is attached to a character in the scene and the game in play mode, then the opened graph will update at runtime to show the current state.

The **Attach Debugger** button will open the motion debugger and attach the character to start recording its movement details.

Properties

Motion Graph & Data

NAME	TYPE	DESCRIPTION
Motion Graph	MotionGraph	The motion graph for the controller to use (a unique instance will be instantiated from this).
Override	MotionGraphDataOverrideAsset	The motion data override for the controller to use.

Colliders

NAME	TYPE	DESCRIPTION
Use Crouch Jump	Boolean	If this is enabled, then the collider will provide an offset that can be used to give extra height to a jump so it appears the legs are tucked up instead of the head ducked down.

Misc

NAME	TYPE	DESCRIPTION
Manual Initialisation	Boolean	Should the component be initialised manually or automatically in Awake and Start? Switch this on for things like networked players.
Upper Body Root	Transform	The root transform of the head hierarchy (used to determine heading).

Force Events

NAME	TYPE	DESCRIPTION
On Motion Graph State Change	UnityEvent	This event is called whenever the controller graph state changes (only includes the end state when traversing multiple states in the graph).
On Ground Impact	UnityEvent	This event is called when the controller first contacts the ground after being airborne. **Parameters** = `Vector3 impulse, float mass`
On Head Impact	UnityEvent	This event is called whenever the top of the controller capsule makes initial contact with a collider. **Parameters** = `Vector3 impulse, float mass`
On Body Impact	UnityEvent	This event is called whenever the sides of the controller capsule makes initial contact with a collider. **Parameters** = `Vector3 impulse, float mass`

STEP TRACKING

NAME	TYPE	DESCRIPTION
Step Speed Cap	Float	The maximum speed when calculating distance travelled for footsteps.
Use Dumb Stepping	Boolean	Switch this to true if you aren't tracking steps in the motion graph, and they will simply be counted at a default rate whenever the character is grounded.

See Also

The Motion Graph

Motion Controller Data

Unity CharacterController

Unity Execution Order of Event Functions

NeoCharacterController Behaviour

Overview

The NeoCharacterController handles movement of characters and their interaction with the physical environment.

Inspector

Properties

Collisions

NAME	TYPE	DESCRIPTION
Depenetration Mask	LayerMask	The physics layers that the character will depenetrate from. This cannot include anything outside of the collision matrix for the gameobject layer. Use this to filter out small dynamic props that should not influence the player.
Skin Width	Float	When performing the move loop, the capsule is shrunk by this amount. When testing for contacts it is grown by this amount.
Slope Limit	Float	The maximum distance above the ground to apply a "sticky" downforce on the frame after leaving the ground.
Slope Friction	Float	The friction of ground contacts when standiong on a slope. At 1, all downward velocity will be cancelled out. At 0, the character will slide down the slope.
Ledge Friction	Float	The friction of ground contacts when overhanging a ledge. At 1, the character will not slide off the ledge.
Wall Angle	Float	The angle (in degrees) from the vertical for a surface to be considered a wall.

NAME	TYPE	DESCRIPTION
Deflection Curve	AnimationCurve	A curve that defines the deflection drop off based on angle from normal. Y-axis is the deflection multiplier, X-axis is the normalised angle (0 = 0 degrees, 1 = 90 degrees)
Step Height	Float	The character will traverse any ledge up to their radius in height. If the step is equal to or below the step height then the character will not lose any horizontal speed when stepping up, and any vertical movement does not count to the character's velocity calculations.
Ground Snap Height	Float	The maximum distance above the ground to apply a "sticky" downforce on the frame after leaving the ground in certain conditions. This prevents leaving the ground when stepping onto down-slopes or off low steps.
Stick To Ground	Boolean	Should the character stick to the ground when walking down steep slopes or over the top of ramps.
Ground Hit Lookahead	Float	The distance to check ahead of a contact (based on contact normal) to see if it was a slope or a step. Set this higher if you are using physics with bevelled corners instead of primitives (naughty).

Rigidbody and Character Interaction

NAME	TYPE	DESCRIPTION
Push Rigidbodies	Boolean	Do not apply forces to non-kinematic rigidbodies if false.
Low Rigidbody Push Mass	Float	Any rigidbodies this mass or below will be pushed with the full push multiplier. Above this and it drops off to zero at max mass.
Max Rigidbody Push Mass	Float	Any rigidbodies above this mass will have zero force applied to them.
Rigidbody Push	Float	A multiplier for the push force at or below the minimum push mass. At normal gravity with no physics materials applied, a 1m box will be on the threshold of moving when this is set to 10. Higher will push the box up to the character's velocity with greater acceleration.
Pushed By Characters	Boolean	Can this character be pushed by other INeoCharacterControllers.
Push Characters	Boolean	Can this character push other INeoCharacterControllers.
Character Push	Float	A multiplier for the push force when pushing characters at or below this characters mass. Drops to 0 when approaching max push mass.

Moving Platforms

NAME	TYPE	DESCRIPTION
Inherit Platform Yaw	Boolean	Does the character inherit yaw changes from moving platforms.

NAME	TYPE	DESCRIPTION
Inherit Platform Velocity	Dropdown	What component of the platform velocity should be included in the character velocity. Options are as follows: • **None** - any movement of the platform is ignored when calculating the character velocity. This prevents exponential acceleration if the character tracks momentum. • **Full** - the movement of the platform is included in character velocity calculations. • **Horizontal Only** - only the horizontal movement of the platform is included in character velocity calculations. Vertical movement is ignored. • **Vertical Only** - only the vertical movement of the platform is included in character velocity calculations. Horizontal movement is ignored.

Gravity

NAME	TYPE	DESCRIPTION
Gravity	Vector3	The gravity vector (direction and acceleration) for the character.
OrientUpWithGravity	Boolean	If this is true, then adjusting the gravity direction will reorient the character so that down is in the direction of gravity, and up is opposed.
UpSmoothing	Float	The duration (in seconds) it takes to rotate the character up vector a whole 180 degrees.

Note

Due to a change in the way ground slope collisions are resolved, the slope speed curve settings have been removed and are now applied to the movement states in the motion graph directly.

See Also

NeoCharacterController

Unity CharacterController

SimpleMovingPlatform MonoBehaviour

Overview

The SimpleMovingPlatform behaviour is used to create a moving platform that moves between 2 points at set intervals.

Inspector

Properties

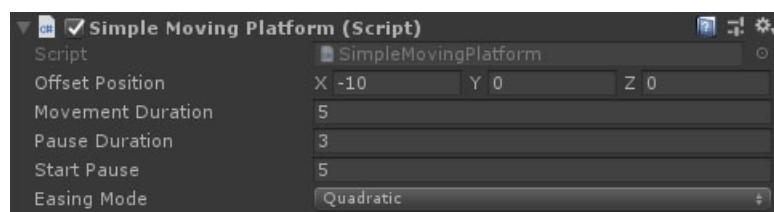| NAME	TYPE	DESCRIPTION
Offset Position	Vector3	The position to move to (offset from current position in world space).
Movement Duration	Float	The time it takes to move.
Pause Duration	Float	The pause before returning to original position or moving again.
Start Pause	Float	The delay before the first move.
Easing Mode	Dropdown	The easing mode for the movement. Options are: **Linear**, **Quadratic**, **Cubic**, **Quartic**.

See Also

Moving Platforms

SimpleRotatingPlatform MonoBehaviour

Overview

The SimpleRotatingPlatform behaviour is used to create a moving platform that rotates at set intervals.

Inspector

Properties

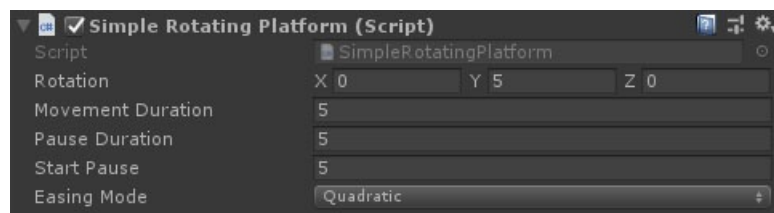| NAME	TYPE	DESCRIPTION
Rotation	Vector3	The total rotation for each rotation phase (relative to the starting rotation of the phase, in world space).
Movement Duration	Float	The time it takes to move.
Pause Duration	Float	The pause before returning to original position or moving again.
Start Pause	Float	The delay before the first move.
Easing Mode	Dropdown	The easing mode for the movement. Options are: **Linear**, **Quadratic**, **Cubic**, **Quartic**.

See Also
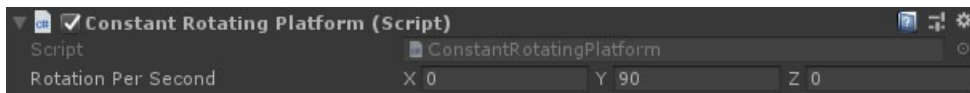
Moving Platforms

WaypointMovingPlatform MonoBehaviour

Overview

The WaypointMovingPlatform behaviour is used to create a moving platform that moves between a set of waypoints.

The waypoints include position and rotation and can be moved to in sequence or directly. The waypoints can also be set to loop round from last back to first, or to form a broken chain.

Movement can be triggered on startup so the platform loops through waypoints, or it can be triggered via scripts and UnityEvents. Using scripting and events you can set the platform to move to a specific waypoint, either via the intermediate waypoints or directly, or you can start and stop the platform looping through waypoints.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Starting Waypoint	Integer	The waypoint the platform starts at (will be repositioned on start).
Speed Curve	AnimationCurve	An animation curve to apply easing to movement between waypoints.
Delay	Float	The delay between waypoints when moving through a sequence. The platform will stop at a waypoint for this duration.
On Start	Dropdown	What to do on start. Options are as follows: • **Nothing** - the platform will not do anything until another script or component triggers it. • **Loop Forwards** - the platform will keep looping through all of the waypoints in order until stopped. If the waypoints are not circular, it will change directions when it reaches the last waypoint. • **Loop Backwards** - the platform will keep looping through all of the waypoints in reverse order until stopped. If the waypoints are not circular, it will change directions when it reaches the first waypoint.
Circular	Boolean	If the waypoints are circular then there is a direct route from the first to last waypoints without going through the others.

The inspector then shows a list of waypoints with the following properties:

NAME	TYPE	DESCRIPTION
Position	Vector3	The position of the platform at this waypoint.
Rotation	Vector3	The rotation of the platform at this waypoint.
Time To Waypoint X	Float	The journey time to reach the next platform in the sequence.
Use Current Transform	Button	Set the position and rotation to match the platform transform in the scene.
Move To Waypoint	Button	Instantly move the platform transform to the position and rotation of the waypoint in the scene.

NAME	TYPE	DESCRIPTION
Up	Button	Move this waypoint up in the sequence.
Down	Button	Move this waypoint down in the sequence.
Insert After	Button	This will insert a new waypoint immediately after this one.
Remove	Button	This will remove the waypoint from the list.

See Also

Moving Platforms

MotionGraph ScriptableObject

Overview

The MotionGraph scriptable object contains the motion graph layout and components. During runtime, a unique instance of the graph will be instantiated so that any modifications to the properties are unique to the character in question.

Inspector

Properties

The MotionGraph has no properties exposed in the inspector.

The **Show Motion Graph Editor** button will open the motion graph editor window, editing this graph.
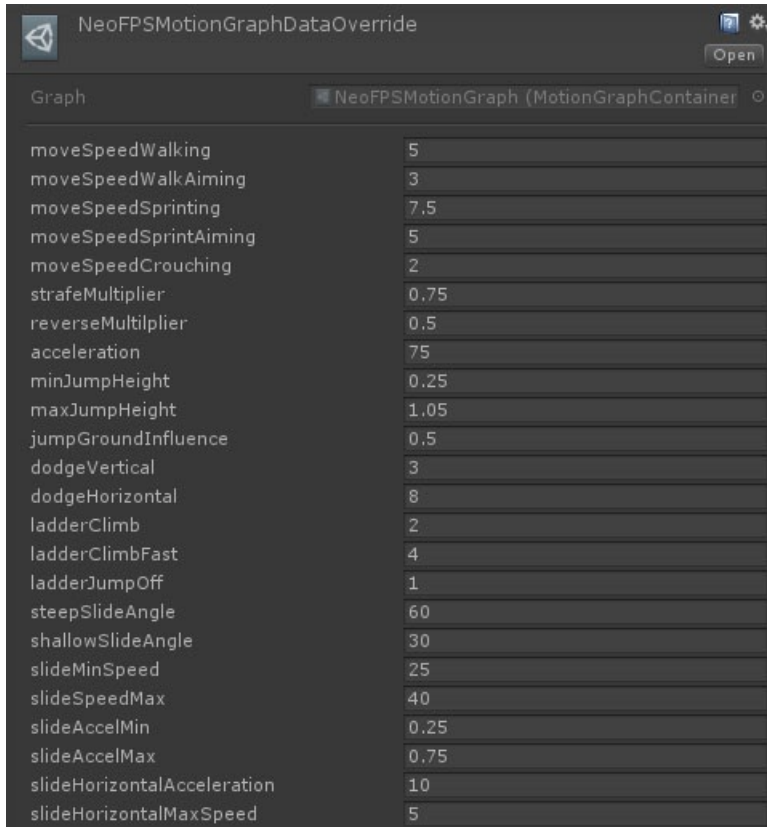
See Also

The Motion Graph

MotionGraphDataOverrideAsset ScriptableObject

Overview

The MotionGraphDataOverrideAsset scriptable object is created from the motion graph editor in the **Motion Data** section. It is permanently attached the the graph you were editing when creating the asset, and will allow you to override each of the data entries on the graph.

Inspector

Properties

The MotionGraphDataOverrideAsset displays an override entry for each of the motion data entries on the motion graph it is attached to. For more information see Motion Graph Parameters And Data .

See Also

The Motion Graph

Motion Graph Parameters And Data

SlopeSpeedCurve ScriptableObject

Overview

The SlopeSpeedCurve scriptable object is used by the grounded Movement state to allow more control over the character's speed on slopes. The X axis of the curve represents the slope, from slope limit downhill at -1 to completely flat at 0 and up to the character slope limit uphill at 1. The y value is the amount of horizontal speed that is maintained when deflected by the slope.

The Movement state aligns its target velocity to the ground surface, clamps the new move vector to the original movement speed and then multiplies by the result of the slope speed curve at the relevant angle. This means that the character can maintain their speed up to a set angle and then it drops off slowly as they approach the limit instead of hitting an invisible wall as the slope angle increases.

Inspector
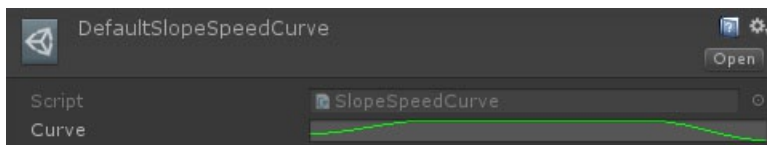
Properties

NAME	TYPE	DESCRIPTION
Curve	AnimationCurve	The desired speed curve as specified above. Ensure that the x-axis extends from -1 to 1 and the y-axis does not go below 0.

See Also

Movement MotionGraphState

NeoFPS Input System

Overview

NeoFPS is built from the ground up to facilitate developing first person shooters for PC and console.

In order for NeoFPS' input system to work it requires a number of custom project settings to be applied. For more information see the Input Settings page.

Unity's existing input system has a number of shortcomings when designing a control scheme. This has led Unity to develop a new input system which is currently in testing.

NeoFPS extends the current input system to address a number of its problems. It adds features such as:

- Runtime bindable keys
- Consistent gamepad profiles
- Mouse smoothing and accleration
- Input contexts (character, menu, etc).

Once Unity's new input system is in full release the NeoFPS system will be adapted to that. You can also completely replace the NeoFPS input system if you have an input solution that you prefer. Look at the existing input handlers to see which properties and methods they access and use these with your preferred system.

Input Contexts

In order to categorise inputs and control which inputs are active at any time, NeoFPS makes use of input contexts. These contexts are defined in the `FpsInputContext` generated constant through the [NeoFpsInputManager][6] asset. Each input handler defines its input context in script and its inputs will only be processed if its context is **None** or if no higher priority context is active. Context priority is simply based on the constant value, with higher numbers being higher priority. A context becomes active when one or more input handlers with that context are active. For example, if a UI element has a InputMenu handler attached to it, then when the UI element is active all character input will be blocked.

Keyboard

NeoFPS is preset with a full WASD control scheme, but any keys can be remapped from the in-game menu. The available buttons and their default key bindings can be specified in the [NeoFpsInputManager][6] asset.

Movement		—
Forward	W	
Backward	S	
Left	A	
Right	D	
Jump	Space	
Sprint (Hold)		
Sprint (Toggle)	LeftShift	
Crouch (Hold)		
Crouch (Toggle)	LeftControl	
Interaction		—
Use	E	
Pick Up	Z	
Combat		—
Primary Fire	Mouse0	
Secondary Fire	Mouse2	
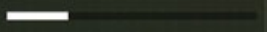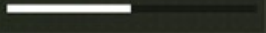
At the top of the input bindings menu is a drop down to reset to defaults. This also allows you to select a keyboard layout. Resetting to a different keyboard layout will map the keys so that the hand positions match a Qwerty keyboard. Available keyboard layouts currently include:

- Qwerty
- Azerty
- Qwertz
- Dvorak
- Colemak

To request more keyboard layouts, please contact support via the discord, or using support@neofps.com

Mouse

NeoFPS features detailed mouse support with inverse look, smoothing and acceleration.

Horizontal Mouse Sensitivity	— ▬▬	25	+
Vertical Mouse Sensitivity	— ▬▬	25	+
Invert Mouse	◁ No	▷	
Mouse Acceleration	◁ Enabled	▷	
Mouse Acceleration Amount	— ▬▬▬	50	+
Mouse Smoothing	◁ Enabled	▷	
Mouse Smoothing Amount	— ▬▬▬	50	+

Smoothing

Mouse smoothing is performed with a weighted average system. You can customise the number of frames to look back as well as the weighting to apply in the MouseAndGamepadAimController behaviour.

Acceleration

Acceleration can be customised via the MouseAndGamepadAimController behaviour, including setting limits and whether the acceleration is linear or quadratic.

Gamepads

Gamepad support in Unity can be quite complicated, with the same gamepad being mapped differently across platforms.

For more information see the Unity Wiki

NeoFPS maps all of the available axes and buttons of the gamepad and wraps them up into preset profiles. The player can then choose between these profiles from the in-game menus.

On console those profiles might be fairly minimal such as *standard* and *south paw*. On standalone builds there might be a number more options as more individual controllers are supported. You can set up your own gamepad profiles via the [NeoFpsInputManager][6].

See Also

Input Settings

Creating Custom Input Handlers

Input Settings

Overview

NeoFPS uses controller profiles to achieve consistent mapping for game controllers across multiple platforms. Since the button layouts provided to Unity by the various controller drivers are not consistent, NeoFPS requires every available axis to be mapped in the input settings and then builds the profiles in code using conditional compilation for the different platforms. Examples of the different controller mappings can be found on the Unify Community Wiki.

For more details of how Unity handles input out of the box, see the Unity Input Settings.

A new input system is in development at Unity and available to use through the package manager in the editor. Once this is in full release, a NeoFPS implementation will be added that makes use of it.

It should also be noted that you can use an alternative input system in NeoFPS by replacing the included input handlers with yoor own implementations.

Required Axes

The following are the axes that NeoFPS uses:

Movement

PROPERTIES
Name = Horizontal
Negative Button = left
Positive Button = right
Alt Negative Button = a
Alt Negative Button = d
Gravity = 3.0
Dead = 0.001
Sensitivity = 3.0
Snap = true
Invert = false
Type = Key or Mouse Button
PROPERTIES
Name = Horizontal
Gravity = 0.0
Dead = 0.3

PROPERTIES

Sensitivity = 1.0

Snap = false

Invert = false

Type = Joystick Axis

Axis = X Axis

Joy Num = Get Motion from all Joysticks

PROPERTIES

Name = Vertical

Negative Button = down

Positive Button = up

Alt Negative Button = s

Alt Negative Button = w

Gravity = 3.0

Dead = 0.001

Sensitivity = 3.0

Snap = true

Invert = false

Type = Key or Mouse Button

PROPERTIES

Name = Vertical

Gravity = 0.0

Dead = 0.3

Sensitivity = 1.0

Snap = false

Invert = true

PROPERTIES	
Type = Joystick Axis	
Axis = Y Axis	
Joy Num = Get Motion from all Joysticks	

Mouse Controls

PROPERTIES	
Name = Mouse X	
Gravity = 0.0	
Dead = 0.0	
Sensitivity = 0.1	
Snap = false	
Invert = false	
Type = Mouse Movement	
Axis = X Axis	
PROPERTIES	
Name = Mouse Y	
Gravity = 0.0	
Dead = 0.0	
Sensitivity = 0.1	
Snap = false	
Invert = false	
Type = Mouse Movement	
Axis = Y Axis	
PROPERTIES	
Name = Mouse ScrollWheel	
Gravity = 0.0	

Dead = 0.0

Sensitivity = 0.1

Snap = false

Invert = false

Type = Mouse Movement

Axis = 3rd Axis (Joysticks and Scrollwheel)

UI

Name = Submit

Positive Button = return

Alt Positive Button = enter

Gravity = 1000.0

Dead = 0.001

Sensitivity = 1000.0

Snap = false

Invert = false

Type = Key or Mouse Button

Name = Submit

Positive Button = joystick button 0

Alt Positive Button = joystick button 16

Gravity = 1000.0

Dead = 0.001

Sensitivity = 1000.0

Snap = false

PROPERTIES
Invert = false
Type = Key or Mouse Button

PROPERTIES
Name = Cancel
Positive Button = escape
Gravity = 1000.0
Dead = 0.001
Sensitivity = 1000.0
Snap = false
Invert = false
Type = Key or Mouse Button

Individual Axes

Axes 1 to 5 are mapped to generic axes with the names **Gamepad Axis 1** to **Gamepad Axis 5** and the following settings. The **Axis** property is set to the relevant entry (1-5).

PROPERTIES
Gravity = 0.0
Dead = 0.2
Sensitivity = 1.0
Snap = false
Type = Joystick Axis

Each axis is also mapped with and inverted version called **Gamepad AxisInv 1** to **Gamepad AxisInv 5**. As you would expect, the non-inverse axes have the **invert** property set to *false*, while the inverse axes have the property set to *true*.

Individual Buttons

Gamepad buttons are mapped with the names **Gamepad Button 0** to **Gamepad Button 19** and the following settings. The **Positive Button** property is set to *joystick button* , where is replaced with the corresponding button number.

PROPERTIES
Gravity = 1000.0

PROPERTIES	
Dead = 0.001	
Sensitivity = 1000.0	
Snap = true	
Invert = false	
Type = Key or Mouse Button	

Button Axes

On some platforms, some controller buttons are actually presented as axes. To handle these cases the buttons are mapped with the names **Gamepad AxisBtn 3** to **Gamepad AxisBtn 10** with the following settings. The **Axis** property is set to the relevant entry (3-10).

PROPERTIES	
Gravity = 1000.0	
Dead = 0.001	
Sensitivity = 1000.0	
Snap = true	
Invert = false	
Type = Joystick Axis	

There are also 2 inverse button axes called **Gamepad AxisBtnInv 6** and **Gamepad AxisBtnInv 7** where the **Invert** property is set to *true*.

See Also

NeoFPS Input System

Unity Input Settings

Unify Community Wiki - Gamepads

Creating Custom Input Handlers

Overview

NeoFPS includes a number of example input handlers:

- InputCharacterMotion
- InputFirearm
- InputGame
- InputMeleeWeapon
- InputThrownWeapon
- InputInventory

Handlers are attached to objects that require a unique control scheme. For example, each firearm has an input handler that is different from the handlers attached to melee weapons.

Handlers are context based and will not return any input when the context is not current.

FpsInputButton

NeoFPS uses a generated constant to define the available button/key inputs. This constant can be expanded based on your game requirements by changing the ConstantsSettings and regenerating the constant.

Take care that reordering entries in the constant will break those values in the inspector as they are stored based on index. Code will adapt to the changes in order, but inspector properties will need checking.

FpsInput Base Class

All input handlers derive from the `FpsInput` class. This controls context and exposes functions similar to Unity's `Input` class but uses the FpsInputButton constant instead of key codes and button IDs in order to separate the input from the key bindings.

The following is a simple example input handler:

```
public class InputExample : FpsInput
{
    public override FpsInputContext inputContext
    {
        get { return FpsInputContext.Character; }
    }

 void Update()
 {
  if (!hasFocus)
   return;

  if (GetButtonDown (FpsInputButton.PrimaryFire))
  {
   // React to the fire button
  }
 }
}
```

See Also

InputAbilityFirearm MonoBehaviour

Overview

The InputAbilityFirearm behaviour reads the **Ability** button and uses it to fire the ModularFirearm that it's attached to. These firearms can be placed in the character hierarchy outside of the inventory system to create things like shoulder mounter launchers.

Inspector
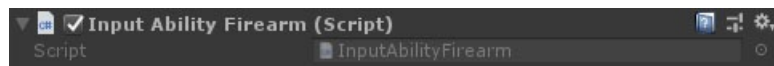
Properties

The InputAbilityFirearm behaviour has no properties exposed in the inspector.

See Also

NeoFPS Input

Unity Input

Modular Firearms

InputCharacterMotion MonoBehaviour

Overview

The InputCharacterMotion behaviour passes character input to the character motion controller.

Inspector
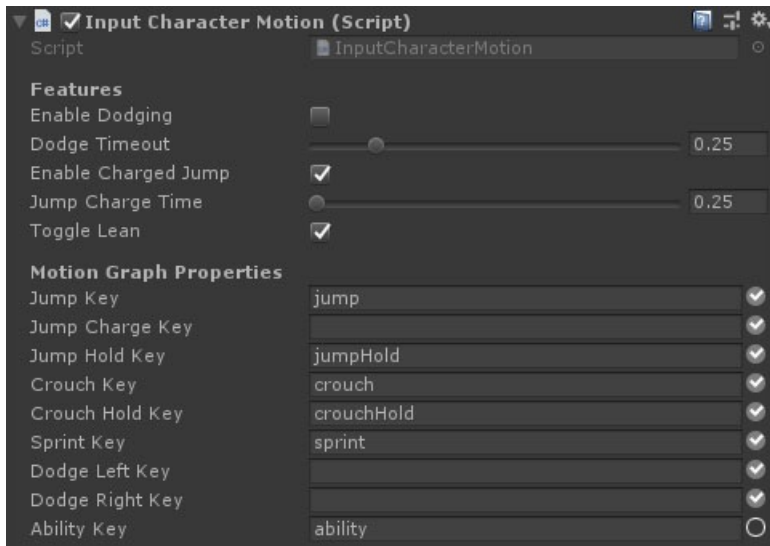
Properties

NAME	TYPE	DESCRIPTION
Enable Dodging	Boolean	Should double tapping a move direction dodge the character.
Dodge Timeout	Float	Multiple taps of a direction within this time range register as a double tap.
Enable Charged Jump	Boolean	Does holding the jump button charge up a jump or does the character dodge as soon as the button is pressed.
Jump Charge Time	Float	The time it takes to charge up a full power jump if charged jumps are enabled.
Toggle Lean	Boolean	Toggle leaning or hold to lean.
Jump Key	String	The key to the jump trigger parameter in the character motion graph.
Jump Charge Key	String	The optional key to the jump charge float parameter in the character motion graph. This is used for charging stronger jumps.
Jump Hold Key	String	The key to the jump hold switch parameter in the character motion graph. This is used for movement like swimming or flying where holding jump moves up.
Crouch Key	String	The key to the crouch switch parameter in the character motion graph.
Crouch Hold Key	String	The optional key to the crouch hold switch parameter in the character motion graph. This is used for movement like swimming or flying where holding jump moves down.

NAME	TYPE	DESCRIPTION
Sprint Key	String	The key to the sprint switch parameter in the character motion graph.
Sprint Hold Key	String	The key to the sprint hold switch parameter in the character motion graph.
Dodge Left Key	String	The optional key to the dodge left trigger parameter in the character motion graph.
Dodge Right Key	String	The optional key to the dodge right trigger parameter in the character motion graph.
Ability Key	String	The optional key to the "ability" trigger parameter in the character motion graph. This can be used for a variety of uses such as dashes, teleports, etc.

See Also

NeoFPS Input

Unity Input

The Motion Graph

InputCharacterSlowMo MonoBehaviour

Overview

The InputCharacterSlowMo behaviour reads the **Ability** button input and toggles slow-motion effects via the character's SlowMoSystem.

Inspector
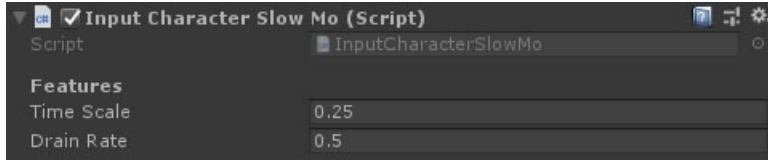
Properties

NAME	TYPE	DESCRIPTION
Time Scale	Float	The time-scale to use for ability based slow-mo.
Drain Rate	Float	The rate to drain slow-mo charge (time scale will return to normal when charge reaches zero).

See Also

NeoFPS Input

Unity Input

InputFirearm MonoBehaviour

Overview

The InputFirearm behaviour handles input for the modular firearms

Inspector
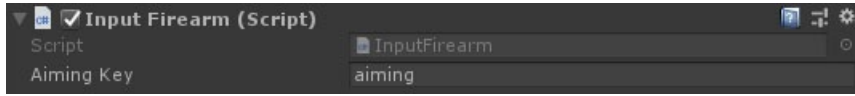
Properties

NAME	TYPE	DESCRIPTION
Aiming Key	String	The property key for the character motion graph (switch parammeter).

See Also

NeoFPS Input

Unity Input

Modular Firearms

InputGame MonoBehaviour

Overview

The InputGame behaviour is a placeholder input handler for game specific input (currently does nothing).

Inspector
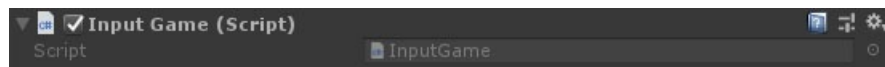
Properties

The InputGame behaviour has no properties exposed in the inspector.

See Also

NeoFPS Input

Unity Input

InputInventory MonoBehaviour

Overview

The InputInventory manages the character inventory.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Repeat Delay	Float	The delay between repeating input when holding the next or previous weapon buttons.
Scroll Delay	Float	The delay between repeating input when rolling the mouse scroll wheel.

See Also

NeoFPS Input

Unity Input

The Inventory

InputLockpick MonoBehaviour

Overview

The InputLockpick behaviour uses the NeoFPS input system to control a simple lock-picking mini-game.

Inspector
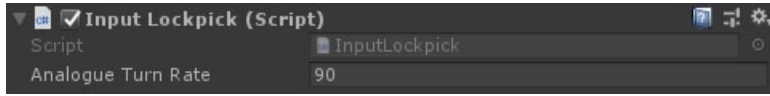
Properties

NAME	TYPE	DESCRIPTION
Analogue Turn Rate	Float	The maximum turn rate of the pick object in degrees per second.

See Also

NeoFPS Input

Unity Input

LockPickPopup3D Behaviour

InputMeleeWeapon MonoBehaviour

Overview

The InputMeleeWeapon behaviour sends input to a MeleeWeapon behaviour on the same object as this.

Inspector

Properties

The InputMeleeWeapon behaviour has no properties exposed in the inspector.

See Also

NeoFPS Input

Unity Input

InputMenu MonoBehaviour

Overview

The InputMenu behaviour is added to UI elements thst should block character input when visible. It adds an input context to the **FpsInputContext.Menu** context when enabled and removes it when disabled.

Inspector
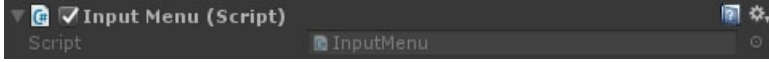
Properties

The InputMenu behaviour has no properties exposed in the inspector.

See Also

NeoFPS Input

InputThrownWeapon MonoBehaviour

Overview

The InputThrownWeapon behaviour sends input to a ThrownWeapon behaviour on the same game object.
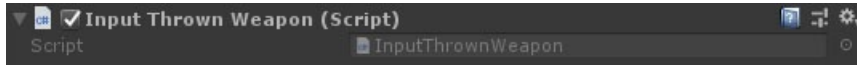
Inspector

Properties

The InputThrownWeapon behaviour has no properties exposed in the inspector.
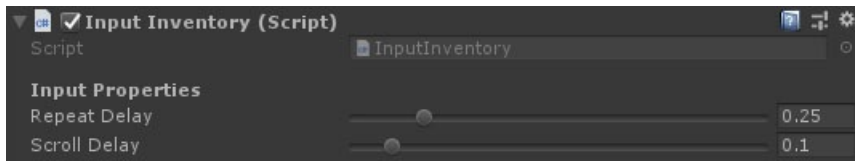

See Also

NeoFPS Input

Unity Input

InputWieldableTool MonoBehaviour

Overview

The InputWieldableTool behaviour sends input to a WieldableTool behaviour on the same game object.
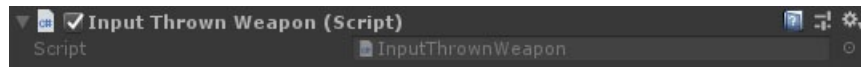
Inspector

Properties

The InputWieldableTool behaviour has no properties exposed in the inspector.

See Also

NeoFPS Input

Wieldable Tools

Unity Input

MouseAndGamepadAimController MonoBehaviour

Overview

The MouseAndGamepadAimController behaviour handles mouse and gamepad input for moving the camera, including smoothing and acceleration.

Inspector
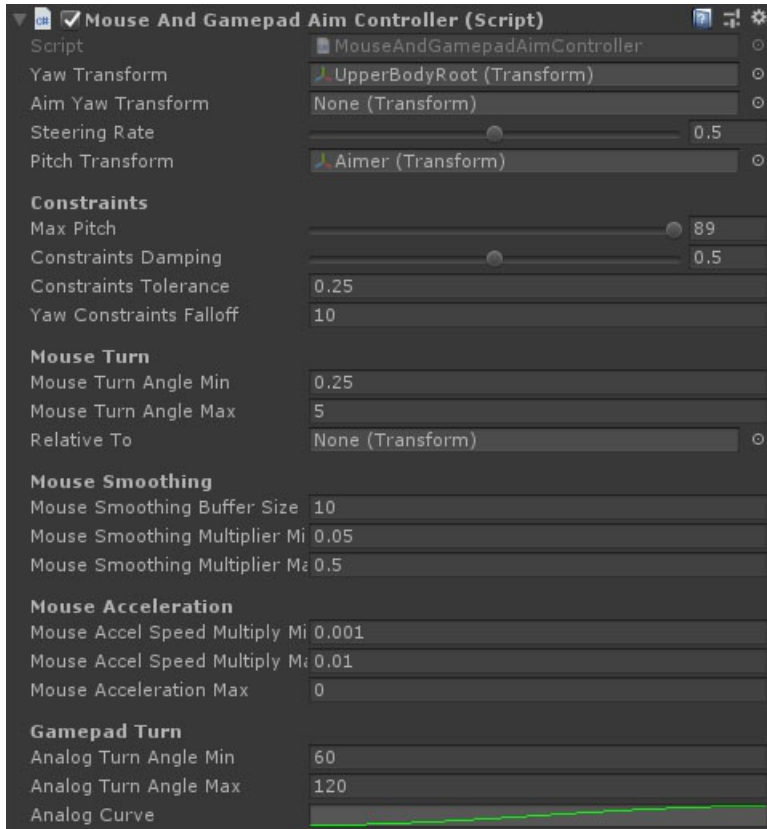
Properties

| NAME	TYPE	DESCRIPTION
Yaw Transform	Transform	The transform to yaw when aiming. This should be a parent of the pitch transform.
Aim Yaw Transform	Transform	This optional transform detaches the character direction from the aim direction.
Steering Rate	Float	The time taken to turn the character to the aim-yaw direction (if Aim Yaw Transform is set). 0 = call LerpYawToAim() manually, 1 = instant.
Pitch Transform	Transform	The transform to pitch when aiming. This should be a child of the yaw transform.
Max Pitch	Float	The maximum pitch from horizontal the aimer can rotate.
Constraints Damping	Float	The amount of damping applied when rotating the camera to match constraints.
Constraints Tolerance	Float	Once the angle outside constraints goes below this value, the camera will snap to the constraints. Larger values will have a visible effect.
YawConstraintsFalloff	Float	An angle range from the yaw constraint limits where the input falls off. This gives the effect of softer constraint limits instead of hitting an invisible wall.

NAME	TYPE	DESCRIPTION
Mouse Turn Angle Min	Float	Number of degrees for 1 unit of mouse movement if sensitivity is set to 0.
Mouse Turn Angle Max	Float	Number of degrees for 1 unit of mouse movement if sensitivity is set to 1.
Relative To	Transform	The transform to calculate the input relative to. If the character can tilt left or right then this transform is required to prevent tilt messing up the yaw calculations.
Mouse Smoothing Buffer Size	Float	The number of frames to store and use for the mouse smoothing history.
Mouse Smoothing Multiplier Min	Float	The weight multiplier for the previous frame when averaging if the smoothing is set to minimum.
Mouse Smoothing Multiplier Max	Float	The weight multiplier for the previous frame when averaging if the smoothing is set to maximum.
Mouse Accel Speed Multiply Min	Float	The base acceleration multiplier when acceleration is set to the minimum.
Mouse Accel Speed Multiply Max	Float	The base acceleration multiplier when acceleration is set to the maximum.
Mouse Acceleration Max	Float	The maximum multiplier acceleration can apply to the mouse input (0 means no maximum).
Mouse Acceleration Type	Dropdown	Does mouse speed affect the input linearly or based on the square of the speed. Options are **Linear**, **Quadratic**.
Analog Turn Angle Min	Float	Number of degrees per second for the gamepad analog at its limit, if sensitivity is set to 0.
Analog Turn Angle Max	Float	Number of degrees per second for the gamepad analog at its limit, if sensitivity is set to 1.
Analog Curve	AnimationCurve	The input curve for analog input. This can be used to define a deadzone, and damp smaller movements.

See Also

NeoFPS Input

Unity Input

NeoFpsInputManager ScriptableObject

Overview

The NeoFpsInputManager scriptable object handles the

Inspector

Properties

The NeoFpsInputManager inspector is split into multiple sections:

Input Buttons

The input buttons section defines the button or key style controls for the game, as well as their default key bindings.

NAME	TYPE	DESCRIPTION
Default Keyboard Layout	Dropdown	The keyboard layout that the buttons below are set up for (the system converts to qwerty and then on to the required layout when resetting to defaults. Available options are: **Qwerty**, **Azerty**, **Qwertz**, **Dvorak**, **Colemak**.

Each button has the following properties:

NAME	TYPE	DESCRIPTION
Name	String	The input button name, as it was referenced in code.
Display Name	String	The name as shown in the key binding game options.
Category	Dropdown	The category of the button, used to organise the buttons in the key binding game options.
Context	Dropdown	The context is used to define when key bindings can be shared by multiple input buttons. An example is weapons that can be aimed, vs weapons that have a secondary function.
Default Keys	Dropdown(s)	The primary and secondary key bindings for the input button. These will be overriden by the player's key binding settings.

Changing any of the buttons in the array or adding / removing constants requires the script to be regenerated. There are also a number of predefined buttos that will also be added for use in menus (None, Menu, Back and Cancel).

The **Generate FpsInputButton Constants** button will generate the button constants script. **Revert To Last Generated** will undo any changes to the buttons to the values when the script was last generated. **Create Snapshot** will store the current button setup, allowing you to revert back to this state using the **Revert To Snapshot** button, The snapshot will be reset when the script is generated.

Input Axes

The input axes are used to track analogue and variable axes in input handlers. The input axes are a generated constants script. Changing any of the constants in the array or adding / removing constants requires the script to be regenerated.

NAME	TYPE	DESCRIPTION
Mouse X Axis	String	The name of the mouse x axis in the Unity input settings.
Mouse Y Axis	String	The name of the mouse y axis in the Unity input settings.
Mouse Scroll Axis	String	The name of the mouse scroll wheel axis in the Unity input settings.
Constants	String Array	The names of the input axes.

Gamepad Profiles

Each gamepad profile under this section can be expanded to map its controls. The **Name** property is the profile name as shown in the game's gamepad options. The **Analogue Setup** property defines what each analogue stick on the gamepad controls. The rest of the properties are mappings for each of the gamepad buttons based on an XBox or PS4 style controller (2 analogues, triggers, d-pad and buttons). Gamepads are detected when connected, and the primary gamepad will be mapped to the profile for player character control. Each gamepad button can be mapped to multiple inputs. The **Add Button...** dropdown will show all of the valid inputs that you can apply. If the gamepad button already has an input then this will only show inputs that are able to clash with this one.

Input Contexts

Input contexts are used by input handlers to define which handler should process input and which should be ignored at any one time. Pushing a context to the stack will only allow input handlers with that context to process input. The input context is a generated constants script. Changing any of the constants in the array or adding / removing constants requires the script to be regenerated.

NAME	TYPE	DESCRIPTION
Constants	String Array	The names of the input contexts.

See Also

NeoFPS Input

Unity Input

Interaction With The World

Overview

Unity and NeoFPS provide a number of ways for characters to interact with the world.

Interactive Objects

NeoFPS also has its own system of interactive objects. These allow the player to use items in order to trigger actions.

Doors are a good example of interactive objects, and NeoFPS includes a number of examples to use as reference.

Trigger Zones

Trigger zones are a fundamental concept for level design and game design as a whole. For information on how to make use of them, please see the Unity documentation.

NeoFPS has a number of helper behaviours that simpllify using triggers to drive game mechanics.

The CharacterZoneTrigger and CharacterZoneTriggerPersistant fire code events when a character enters. These events are not tied to a specific implementation of the NeoFPS character and the only requirement is that they inherit from the `ICharacter` interface. Some scripting is required to use these behaviours.

The SoloCharacterZoneTrigger and SoloCharacterZoneTriggerPersistant fire Unity events when an FpsSoloCharacter enters. As such, they require less scripting knowledge to use, but they are constrained to the specific character implementation or characters that derive from this.

See Also

Interactive Objects

Doors and Locks

Unity Colliders

Doors and Locks

Overview

NeoFPS includes a number of example doors. Each of which can be opened and closed using either trigger zones or interactive objects.

Kinematic Hinge Doors

These are swing doors that use simple procedural animation to open or close. They use an optional double hinge system to allow for opening in both directions without overlapping into the frame.

Kinematic doors will push the player back and do not react to physics in the scene.

For more information see the KinematicHingeDoor reference.

Physics Hinge Doors

These are swing doors that use a Unity HingeJoint to drive the door's animation. These doors react physically to the environment and to player actions.

Physics hinge doors can only open in one direction and will automatically latch when closed.

For more information see the PhysicsHingeDoor reference.

Sliding Doors

Sliding doors involve one or more sliding components and are animated procedurally.

For more information see the SlidingDoor reference.

Elevators Doors
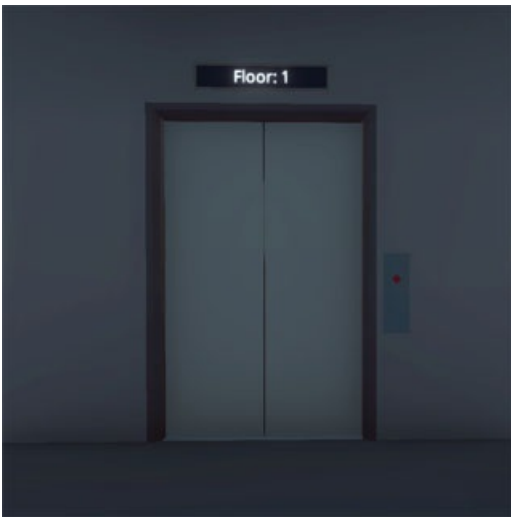
Elevators are a complex system of interactive objects, moving platforms and sliding doors. The elevator cab moves up and down between floors. Only the doors for the floor the cab is currently on can open and only when the cab has stopped moving. The cab inner door sections are shared with each of the outer doors, making them sync up as any of those are opened.

For more information see the ElevatorController reference.

Locked Doors
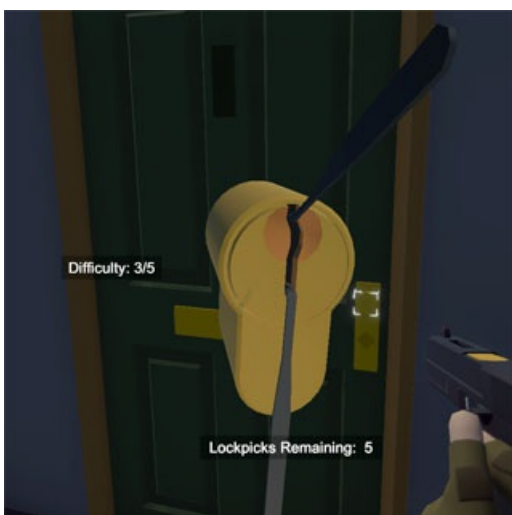
NeoFPS also features a system for locking and unlocking doors. The majority of locks rely on the keyring system. The KeyRing is an inventory object which tracks the key IDs that a character knows. This can include keycodes for entering into keypads, as well as keys for physical locks. Picking up a keyring will merge the contents with the one in the character inventory.

When you set up a locked door, you specify a key ID. If no ID is set, then the door must be unlocked via an event or the scripting API. As an example, events can be used to unlock a door on destroying an object. This is how the destructible padlock demo is set up in the doors demo scene. The API is used by the lockpicking mini-game to unlock the door if the player has lockpicks in their inventory but does not have the door's key.

The following components are available for working with locks and locked doors:

NAME	DESCRIPTION
KeyRing	The keyring stores multiple key IDs.
LockedDoorInteractiveObject	A version of the door component that lets you specify a lock ID.
LockedDoorTrigger	A trigger zone that is used to open an attached door, but only if the character entering has a key with the correct ID.
LockedTriggerZone	A trigger zone that will fire events if a character with the correct key enters.
KeypadInteractiveObject	An interactive object that is attached to a locked door, and displays a UI based KeypadPopup when interacted with. Entering the correct code will unlock the door.
KeypadPopup	The UI based keypad popup that is shown by the KeypadInteractiveObject.
PickableLockedDoorInteractiveObject	A locked door that displays a lockpicking mini-game when interacted with if the character does not have the correct key to unlock it.
LockpickPopup3D	One implementation of the lockpicking mini-game that is used to unlock a locked door.
LockpickPopupUI	A UI overlay for the LockpickPopup3D mini-game that displays remaining pick count and lock difficulty.

See Also

Interactive Objects

Moving Platforms

Interactive Objects

Overview

Interactive objects are objects in the scene that the player can approach and *use* to perform some action.

Objects can specify whether to react instantly or once the use button has been held for a set period of time.

As the player looks at an interactive object, events are fired that allow custom highlights and effects to be added. This can be utilised to show markers as in the sample assets, to change a shader setting, or to flag up an object with a HUD element.

Example interactive objects include doors, buttons and weapon pickups.

For more information see the CharacterInteractionHandler and InteractiveObject references.

Creating An Interactive Object
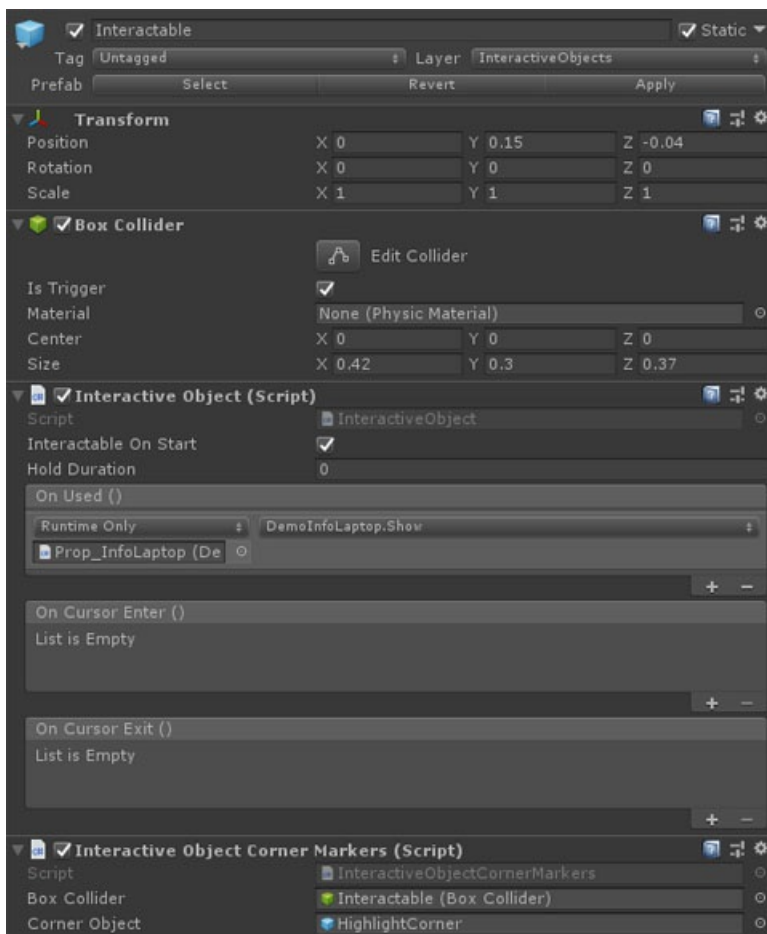
Interactive objects use events to call methods on attached components and perform some function.

To make an object interactive, first add a GameObject to its hierarchy and set its layer to **InteractiveObjects**. Next, add a primitive collider to the new object and set its **IsTrigger** property to true. With the trigger collider set up, add an InteractiveObject component to the object and set its hold duration (0 means the object will be used instantly when the player hits the use button, anything else means the player needs to hold the use button down for that duration). Finally, set the **OnUse** event on the component to point to a relevant method on one of the original object's components.

Alternatively, you can write a script that inherits from `InteractiveObject` and override the `Interact (ICharacter character)` method to add implement the required behaviour.

See Also

CharacterInteractionHandler

InteractiveObject

AnimatedDoorHandle MonoBehaviour

Overview

The AnimatedDoorHandle behaviour is attached to a handle object and makes it twist and release when triggered.

Inspector
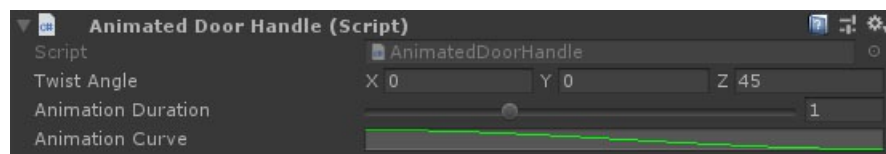
Properties

NAME	TYPE	DESCRIPTION
Twist Angle	Vector3	The desired euler angles when twisting the handle.
Twist Duration	Float	How long does the twist and release last.
Twist Curve	AnimationCurve	The animation curve for the twist.
Jiggle Angle	Vector3	The maximum euler angles when shaking the handle (if the door is locked).
Jiggle Duration	Float	How long does the locked door jiggle last.
Jiggle Curve	AnimationCurve	The animation curve for the locked door handle jiggle.

See Also

Doors

Unity AnimationCurve

CharacterInteractionHandler MonoBehaviour

Overview

The CharacterInteractionHandler behaviour is attached to a character and handles interaction with interactive objects in the scene.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Max Distance	Float	The maximum distance from the camera to trigger interactions.
Tick Rate	Int	How frequently does the handler cast forward to check for an object. Smaller numbers mean more responsive but more wasted calculations.
Layers	LayerMask	The layers that will be checked against when casting for valid interaction targets.
Error Audio	FpsCharacterAudio	The character audio clip to play for an invalid interaction.

See Also

Interactive Objects

FpsCharacterAudioHandler

CharacterTriggerZone MonoBehaviour

Overview

The CharacterTriggerZone behaviour fires script events when a character enters and exits the collider.
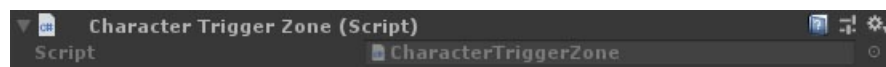
Inspector

Properties

The CharacterTriggerZone behaviour has no properties exposed in the inspector.

See Also

Unity BoxCollider

Unity SphereCollider

Unity CapsuleCollider

CharacterTriggerZonePersistant MonoBehaviour

Overview

The CharacterTriggerZonePersistant behaviour fires script events when characters enter and exit.
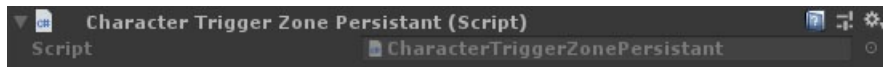
Inspector

Properties

The CharacterTriggerZonePersistant behaviour has no properties exposed in the inspector.

See Also

Layers And Tags

Unity BoxCollider

Unity SphereCollider

Unity CapsuleCollider

DoorInteractiveObject MonoBehaviour

Overview

The DoorInteractiveObject behaviour is an extension of the InteractiveObject used to trigger door open and close.

Inspector
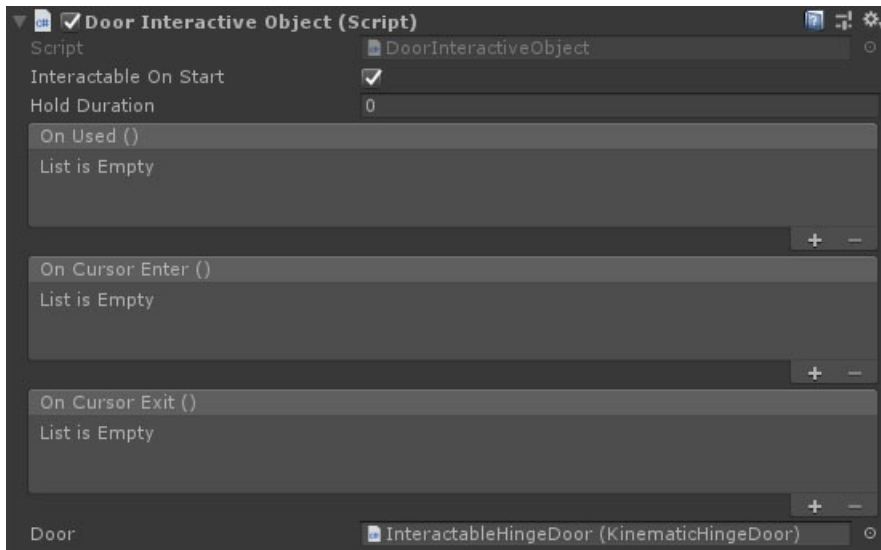
Properties

The DoorInteractiveObject inherits from the InteractiveObject. Check the reference for information on its properties.

NAME	TYPE	DESCRIPTION
Door	Door	The door to open (will accept any door that inherits from `DoorBase`).

See Also

InteractiveObject

DoorTrigger MonoBehaviour

Overview

The DoorTrigger behaviour is attached to an object with a trigger collider. It will open the specified door when a character enters, and close it when they leave.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Door	Door	The door to open (will accept any door that inherits from `DoorBase`).
Characters Only	Boolean	Should the door only open for characters, not any collider.

See Also

Doors

Unity BoxCollider

Unity SphereCollider

Unity CapsuleCollider

ElevatorController MonoBehaviour

Overview

The ElevatorController behaviour manages a single elevator.

An elevator also requires an ElevatorMovingPlatform to function. This acts as the cab or platform and actually moves between floors.

Each floor has a door that will be opened when the elevator cab arrives at that floor. If that door is a SlidingDoor behaviour, then the cab's door geometry can be added to every floor as a door section. This means that the cab's inner doors will open in sync with the relevant floor's outer doors.

The ElevatorController exposes the `PressFloorButton(int floorIndex)` function. This can be attached to InteractiveObject events for the exterior call buttons and the cab interior floor buttons. If the current floor button is pressed the doors will open for a set time. If another floor button is pressed the doors will close, the cab will move to the correct floor, and then the doors will open for a set time.

Inspector
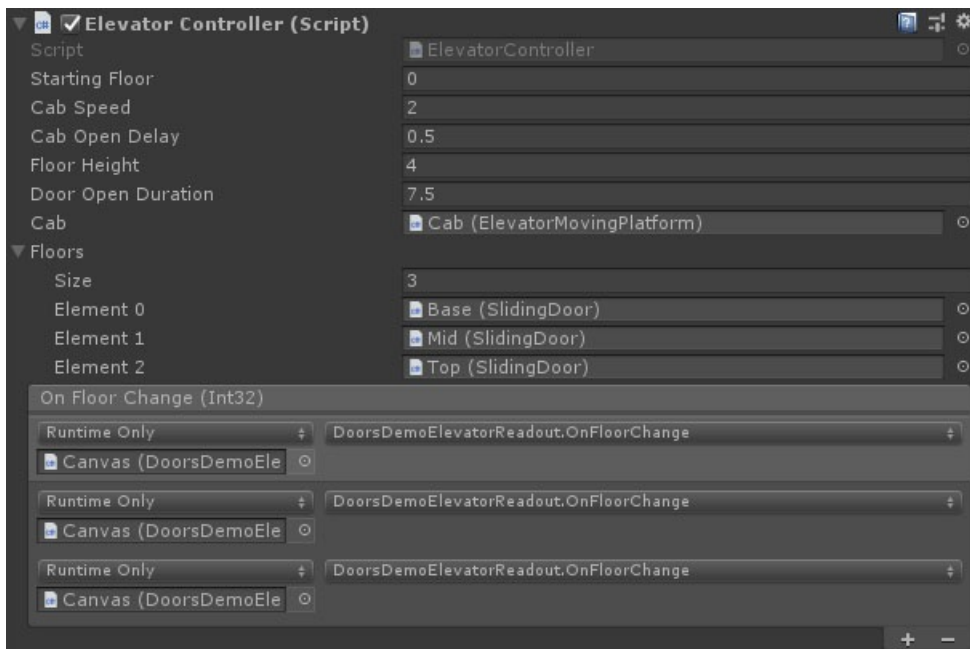
Properties

NAME	TYPE	DESCRIPTION
Starting Floor	Int	The floor the elevator cab starts on. It is best to move the cab to this position in the editor to prevent it jumping there instantly which can cause problems if there are dynamic objects in the cab.
Cab Speed	Float	The movement speed of the cab.
Cab Open Delay	Float	The dlay between reaching a floor and opening the doors.
Floor Height	Float	The distance between floors.

NAME	TYPE	DESCRIPTION
Door Open Duration	Float	The duration the elevator doors will remain open unless interrupted.
Cab	ElevatorMovingPlatform	The moving platform of the elevator cab.
Floors	Door Array	The doors for each floor (will accept any door that inherits from `DoorBase`).
On Floor Change	UnityEvent	An event that is invoked every time the cab switches floors. Useful for any elevator readout or chime.

See Also

Doors

ElevatorMovingPlatform

SlidingDoor

InteractiveObject

Unity Events

ElevatorMovingPlatform MonoBehaviour

Overview

The ElevatorMovingPlatform behaviour is a moving platform which works with the ElevatorController. It is attached to the kinematic rigidbody that acts as the elevator cab or platform, and moves the character smoothly with it.

Inspector
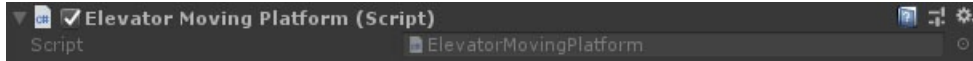
Properties

The ElevatorMovingPlatform behaviour has no properties exposed in the inspector.

See Also

ElevatorController

Moving Platforms

InteractiveObject MonoBehaviour

Overview

The InteractiveObject behaviour is used to represent any object in the world that the player can interact with via the **use** button.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Tooltip Name	String	The name of the item in the HUD tooltip.
Tooltip Action	String	A description of the action for use in the HUD tooltip, eg pick up.
Interactable On Start	Boolean	Can the object be interacted with immediately.
Hold Duration	Float	How long does the use button have to be held for interaction.
On Used	UnityEvent	An event that is triggered when the object is used.
On Cursor Enter	UnityEvent	An event that is triggered when the player looks directly at the object.
On Cursor Exit	UnityEvent	An event that is triggered when the player looks away from the object.

See Also

Interactive Objects

Unity Events

InteractiveObjectCornerMarkerrs MonoBehaviour

Overview

The InteractiveObjectCornerMarkerrs behaviour is used to highlight interactive objects when the player looks at them. It spawns markers on the bounding box corners of the object. These are then displayed when the object is highlighted and hidden when it is not.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Box Colliders	BoxCollider Array	The box colliders of the InteractiveObject to attach markers to.
Corner Object	GameObject	The prefab to use for the corner objects. 8 instances of this will be instantiated and placed at the corners of the box.

See Also

InteractiveObject

Unity BoxCollider

InteractiveObjectMaterialMarker MonoBehaviour

Overview

The InteractiveObjectMaterialMarker behaviour is used to highlight interactive objects when the player looks at them. It fades in a glowing pulse effect on the object as long as it has a material that uses the *NeoFPS/Standard/InteractiveHighlightMetallic* or *NeoFPS/Standard/InteractiveHighlightSpecular* shaders.

Inspector
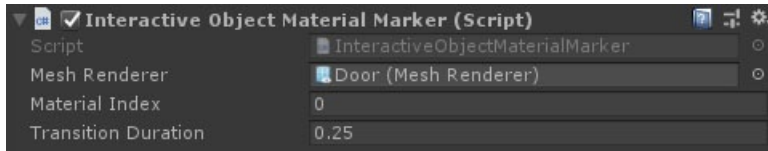
Properties

NAME	TYPE	DESCRIPTION
Mesh Renderer	MeshRenderer	The mesh renderer of the object (used to set the relevant material property blocks).
Material Index	Integer	The index of the highlight material on the mesh renderer.
Transition Duration	Float	A fade time between he highlighted state and the non-highlighted state.

See Also

InteractiveObject

[Unity BoxCollider][unity-boxcollider]

KeypadInteractiveObject MonoBehaviour

Overview

The KeypadInteractiveObject behaviour is an extension of the InteractiveObject that can be used alongside locked doors. On using the object it displays a keypad popup, and if the player enters the correct key code then the door will be unlocked.

Inspector

Properties

The DoorInteractiveObject inherits from the InteractiveObject. Check the reference for information on its properties.

NAME	TYPE	DESCRIPTION
Tooltip Name	String	The name of the item in the HUD tooltip.
Tooltip Action	String	A description of the action for use in the HUD tooltip, eg pick up.
Interactable On Start	Boolean	Can the object be interacted with immediately.
Hold Duration	Float	How long does the use button have to be held for interaction.
On Used	UnityEvent	An event that is triggered when the object is used.
On Cursor Enter	UnityEvent	An event that is triggered when the player looks directly at the object.
On Cursor Exit	UnityEvent	An event that is triggered when the player looks away from the object.

NAME	TYPE	DESCRIPTION
Door	Door	The door to open (will accept any door that inherits from `DoorBase`).
m_KeypadPopup	KeypadPopup	The keypad UI popup to show.
m_PassCode	Integer Array	The key code to unlock the door.
Lock Ids	String Array	A unique ID for this lock. IF the player has an equivalent key in their inventory key ring then the digits will be shown with the popup.
m_StartLocked	Boolean	Should the door be locked on start.

See Also

InteractiveObject

KeypadPopup Behaviour

KeyRing Behaviour

KeypadPopup MonoBehaviour

Overview

The KeypadPopup behaviour is a UI popup which can be displayed for locked doors and items. Events are fired for correct and incorrect passcodes. If the passcode is known, then it will be displayed alongside the keypad.

Inspector
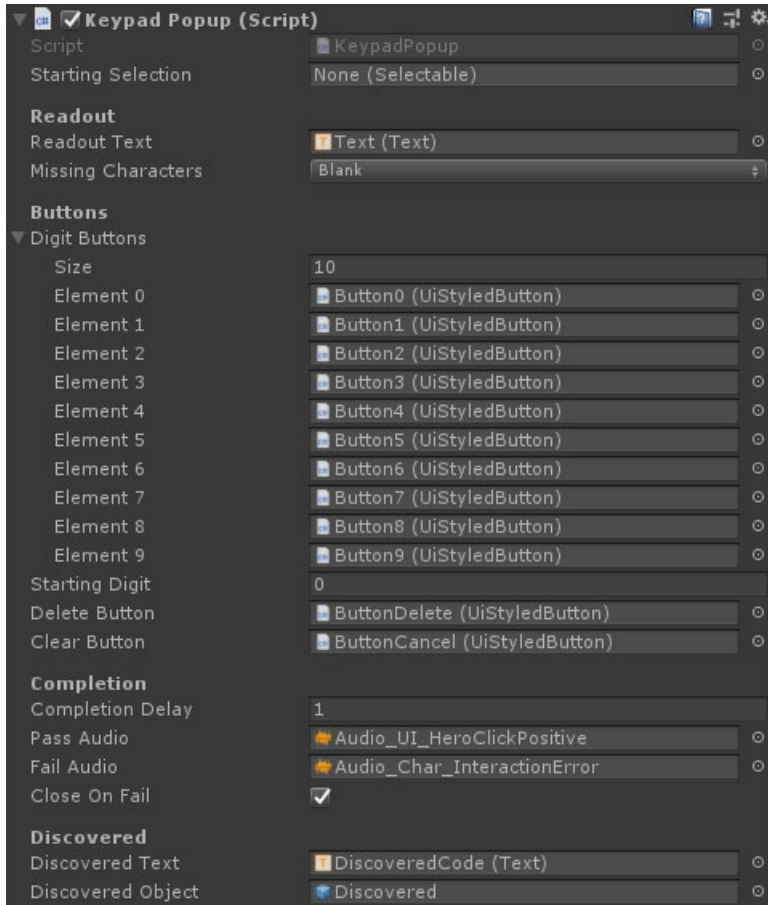
Properties

NAME	TYPE	DESCRIPTION
Starting Selection	Selectable	The initial UI element to select when the popup is shown.
Readout Text	Text	Output text when typing code.
Missing Characters	Dropdown	The character to use for missing digits in the code.
Digit Buttons	Button Array	The number buttons of the keypad in numberic order (eg 0,1,2,3,4...).
Starting Digit	Integer	The lowest numbered button.
Delete Button	Button	The delete button is used to delete the last input digit.
Clear Button	Button	The clear button clears all typed digits.
Completion Delay	Float	The time after the last digit is input before the popup closes.

NAME	TYPE	DESCRIPTION
Pass Audio	AudioClip	The audio clip to play if the correct code is input.
Fail Audio	AudioClip	The audio clip to play if an incorrect code is input.
Close On Fail	Boolean	Should the popup close after the wrong code is input or allow the player to enter another.
Discovered Text	Text	If the keycode is known it will be shown here.
Discovered Object	GameObject	If the keycode is known then this object will be activated.

See Also

Doors

Sample UI

KeyRing MonoBehaviour

Overview

The KeyRing behaviour is an inventory item which stores keys and keycodes. Picking up a keyring will add it to the character's inventory if they do not already have one, or merge the contents with the existing one if they do.

Inspector
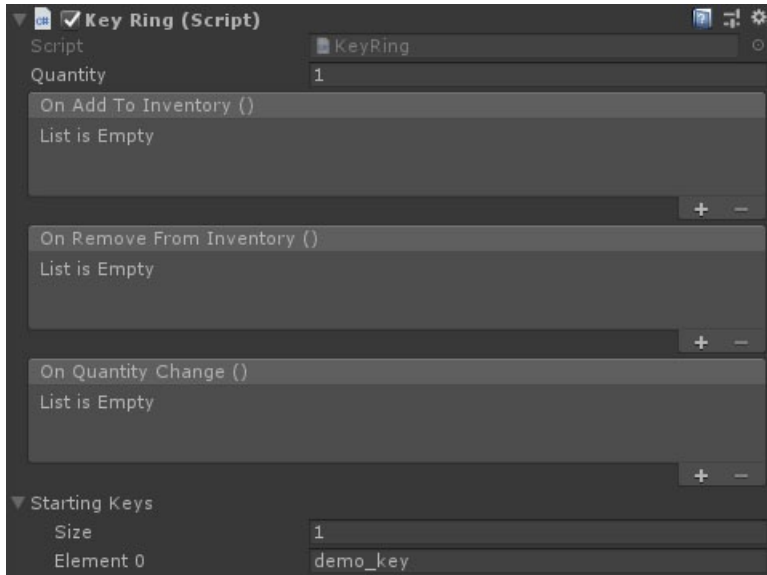
Properties

NAME	TYPE	DESCRIPTION
Starting Keys	String Array	A list of key IDs this keyring starts with.

See Also

Doors

Inventory

KinematicHingeDoor MonoBehaviour

Overview

The KinematicHingeDoor behaviour handles opening, closing and animating a basic hinge door. The door rotates around 2 points depending on which direction it opens to prevent the door object overlapping with its frame.

Inspector
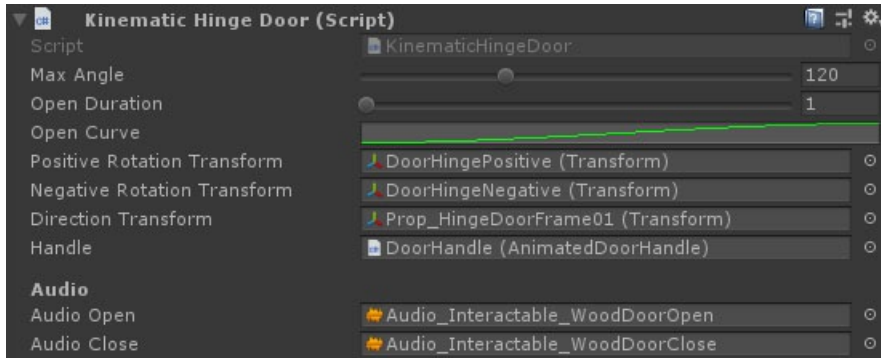
Properties

NAME	TYPE	DESCRIPTION
Max Angle	Float	The maximum open angle of the door.
Open Duration	Float	The time it takes to go from fully closed to fully open and vice versa.
Open Curve	AnimationCurve	The interpolation curve for opening and closing the door.
Positive Rotation Transform	Transform	The transform point for the positive rotation of the door (defaults to the transform for the object this behaviour is attached to).
Negative Rotation Transform	Transform	The transform point for the negative rotation of the door. This allows hinges at both edges of a door to prevent overlap. If this is null the door will only open on one side.
Direction Transform	Transform	A fixed transform used to check which side of the doorway a character is on (defaults to the transform for the object this behaviour is attached to).
Handle	AnimatedDoorHandle	An optional animated door handle. This will turn and release when the door is opened.
Audio Open	AudioClip	The audio to play when the door is unlatched and opened.
Audio Close	AudioClip	The audio to play when the door closes and latches.
Audio Locked	AudioClip	The audio to play when attempting to open the door while locked.
Audio Unlock	AudioClip	The audio to play when unlocking or locking the door.

See Also

Doors

AnimatedDoorHandle

LockedDoorInteractiveObject MonoBehaviour

Overview

The LockedDoorInteractiveObject behaviour is an extension of the InteractiveObject used to trigger door open and close. If the door is locked then the character must have a keyring inventory item with the correct keycode to unlock the door before it can be opened.

Inspector

Properties

The DoorInteractiveObject inherits from the InteractiveObject. Check the reference for information on its properties.

NAME	TYPE	DESCRIPTION
Tooltip Name	String	The name of the item in the HUD tooltip.
Tooltip Action	String	A description of the action for use in the HUD tooltip, eg pick up.
Interactable On Start	Boolean	Can the object be interacted with immediately.
Hold Duration	Float	How long does the use button have to be held for interaction.
On Used	UnityEvent	An event that is triggered when the object is used.
On Cursor Enter	UnityEvent	An event that is triggered when the player looks directly at the object.
On Cursor Exit	UnityEvent	An event that is triggered when the player looks away from the object.
Door	Door	The door to open (will accept any door that inherits from `DoorBase`).

NAME	TYPE	DESCRIPTION
Lock Ids	String Array	An array of IDs for this lock. The player must have an equivalent key in their inventory key ring to unlock. If this is empty then the door must be unlocked via events or the API. IDs can be unique to this lock, or shared between multiple for things like skeleton keys.
m_StartLocked	Boolean	Should the door be locked on start.
m_OpenOnUnlock	Boolean	Should the door be opened when it's unlocked.
m_TooltipLockedAction	String	The tooltip action to use when the door is locked. Use the open action toolrip for the other tooltip action.

See Also

InteractiveObject

KeyRing Behaviour

LockedDoorTrigger MonoBehaviour

Overview

The LockedDoorTrigger behaviour is attached to an object with a trigger collider. It will open the specified door when a character with the correct key code in their inventory key ring enters, and close it when they leave.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Lock Ids	String Array	An array of IDs for this lock. The player must have an equivalent key in their inventory [key ring][3] to unlock. If this is empty then the door must be unlocked via events or the API. IDs can be unique to this lock, or shared between multiple for things like skeleton keys.
On Trigger Enter	UnityEvent	The event that is fired when a character enters the trigger collider.
On Trigger Exit	UnityEvent	The event that is fired when a character exits the trigger collider.
Door	Door	The door to open (will accept any door that inherits from `DoorBase`).

See Also

Doors

KeyRing Behaviour

Unity BoxCollider

Unity SphereCollider

Unity CapsuleCollider

LockedTriggerZone MonoBehaviour

Overview

The LockedTriggerZone behaviour is a trigger zone that only activates if the character has a specific key code in their inventory key ring.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Lock Ids	String Array	An array of IDs for this lock. The player must have an equivalent key in their inventory key ring to activate the trigger zone. IDs can be unique to this lock, or shared between multiple for things like skeleton keys.
On Trigger Enter	UnityEvent	The event that is fired when a character enters the trigger collider.
On Trigger Exit	UnityEvent	The event that is fired when a character exits the trigger collider.

See Also

KeyRing Behaviour

Unity BoxCollider

Unity SphereCollider

Unity CapsuleCollider

LockPickPopup3D MonoBehaviour

Overview

The LockPickPopup3D behaviour is a specific lockpicking mini-game in the style of fallout or dying light. Your scene can contain multiple lock-picking minigames, all differentiated by their IDs, as long as their scripts inherit from the `LockPickPopup` base class.

Inspector
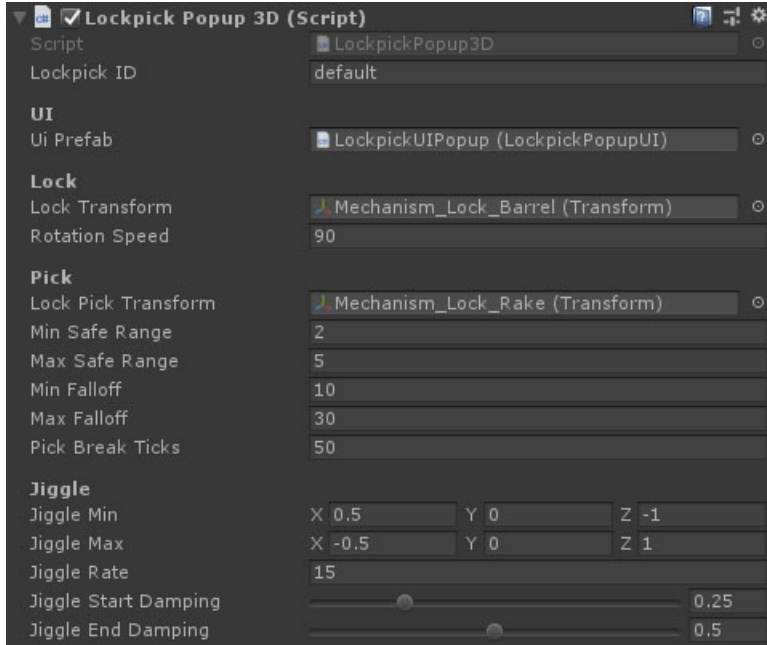
Properties

NAME	TYPE	DESCRIPTION
UI Prefab	LockpickPopupUI	The UI popup prefab to be drawn over the top of this minigame using the prefab popup container system.
Lock Transform	Transform	The transform of the lock barrel.
Rotation Speed	Float	The rotation speed of the lock barrel when tensioned (in degrees per second).
Lock Pick Transform	Transform	The transform of the lock pick object. Its pivot should be lined up with the hole of the lock.
Min Safe Range	Float	The smallest size the safe range for the pick can be (at highest difficulty).
Max Safe Range	Float	The maximum size the safe range for the pick can be (at lowest difficulty).
Min Falloff	Float	The smallest falloff outside the safe range, where the lock can rotate but will still snag (at highest difficulty).
Max Falloff	Float	The largest falloff outside the safe range, where the lock can rotate but will still snag (at lowest difficulty).
Pick Break Ticks	Integer	The number of fixed update ticks where the pick is catching before it will break.
Jiggle Min	Vector3	The minimum jiggle angle when the pick catches. It will bounce between this and max.

NAME	TYPE	DESCRIPTION
Jiggle Max	Vector3	The maximum jiggle angle when the pick catches. It will bounce between this and min.
Jiggle Rate	Float	The number of shakes per second when catching.
Jiggle Start Damping	Float	The amount of time it takes for the jiggle to fade in when the lock catches.
Jiggle End Damping	Float	The amount of time it takes for the jiggle to fade out once tension is released.

See Also

Doors

LockpickPopupUI Behaviour

LockPickPopupUI MonoBehaviour

Overview

The LockPickPopupUI behaviour is attached to an object with a trigger collider. It will open the specified door when a character enters, and close it when they leave.

Inspector
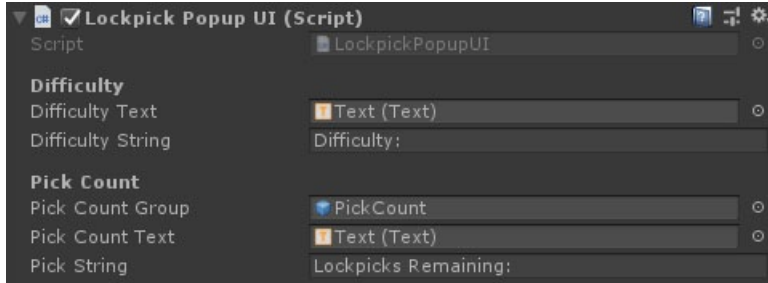
Properties

NAME	TYPE	DESCRIPTION
Difficulty Text	Text	The UI text element that will show the difficulty rating of the lock.
Difficulty String	String	The pick difficulty prefix.
Pick Count Group	GameObject	The parent object of the UI elements that show the pick count. If the lock does not use inventory picks, this object and its children will be hidden.
Pick Count Text	Text	The text readout for the remaining pick count.
Pick String	String	The pick count prefix string.

See Also

Doors

Sample UI

LockpickPopup3D Behaviour

PhysicsHingeDoor MonoBehaviour

Overview

The PhysicsHingeDoor behaviour handles latching and release of a physical hinge based door.

Inspector
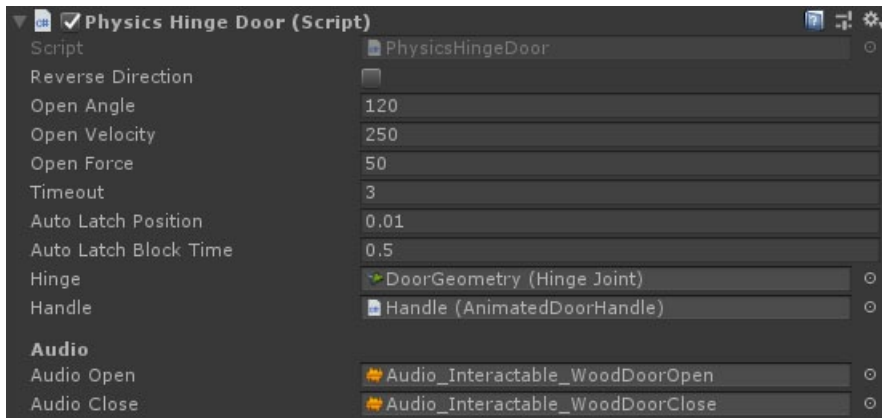
Properties

NAME	TYPE	DESCRIPTION
Reverse Direction	Boolean	Reverses the door opening direction.
Open Angle	Float	The open limit of the door. When opening, force will be applied until the door reaches this angle.
Open Velocity	Float	The target speed to move the door.
Open Force	Float	The force applied to the door when opening or closing.
Timeout	Float	A time limit for force to be applied. If the door is blocked, it would never reach full open or closed and this value prevents it trying forever.
Auto Latch Position	Float	The normalised position (0 to 1 translates to closed to full open angle) at which the door will latch when closing.
Auto Latch Block Time	Float	Prevent latching for a short perios when opened.
Hinge	HingeJoint	The hinge joint of the door.
Handle	AnimatedDoorHandle	An optional animated door handle. This will turn and release when the door is opened.
Audio Open	AudioClip	The audio to play when the door is unlatched and opened.
Audio Close	AudioClip	The audio to play when the door closes and latches.
Audio Locked	AudioClip	The audio to play when attempting to open the door while locked.
Audio Unlock	AudioClip	The audio to play when unlocking or locking the door.

See Also

Doors

AnimatedDoorHandle

Unity HingeJoint

PickableLockedDoorInteractiveObject MonoBehaviour

Overview

The PickableLockedDoorInteractiveObject behaviour is an extension of the InteractiveObject used to trigger door open and close. If the door is locked then there are 2 ways to unlock it. Firstly, if the character has a keyring invnentory item with the correct keycode, then they will unlock the door instantly. Secondly, if they do not have the correct keycode but do have lockpick items in their inventory, then using the door will show a lock-picking minigame popup.

Inspector

Properties

The DoorInteractiveObject inherits from the InteractiveObject. Check the reference for information on its properties.

NAME	TYPE	DESCRIPTION
Tooltip Name	String	The name of the item in the HUD tooltip.
Tooltip Action	String	A description of the action for use in the HUD tooltip, eg pick up.
Interactable On Start	Boolean	Can the object be interacted with immediately.
Hold Duration	Float	How long does the use button have to be held for interaction.
On Used	UnityEvent	An event that is triggered when the object is used.
On Cursor Enter	UnityEvent	An event that is triggered when the player looks directly at the object.
On Cursor Exit	UnityEvent	An event that is triggered when the player looks away from the object.

NAME	TYPE	DESCRIPTION
Door	Door	The door to open (will accept any door that inherits from `DoorBase`).
Lock Ids	String Array	An array of IDs for this lock. The player must have an equivalent key in their inventory key ring to unlock. If this is empty then the door must be unlocked via events or the API. IDs can be unique to this lock, or shared between multiple for things like skeleton keys.
Lockpick ID	String	The ID for the lockpick to use. This allows for multiple lockpick styles in a single scene.
Lockpick Difficulty	Float	The difficulty of this specific lock.
Requires Pick Item	Boolean	Does the character require a lockpick item in their inventory.
m_StartLocked	Boolean	Should the door be locked on start.
m_OpenOnUnlock	Boolean	Should the door be opened when it's unlocked.
m_TooltipLockedAction	String	The tooltip action to use when the door is locked. Use the open action toolrip for the other tooltip action.

See Also

Doors

InteractiveObject

KeyRing Behaviour

LockPickPopup3D Behaviour

SlidingDoor MonoBehaviour

Overview

The SlidingDoor behaviour is used to control and animate moving door sections.

Inspector
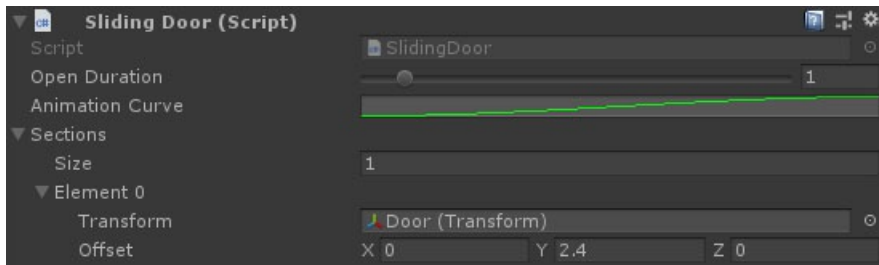
Properties

NAME	TYPE	DESCRIPTION
Open Duration	Float	The time it takes to go from fully closed to fully open and vice versa.
Animation Curve	AnimationCurve	The interpolation curve for animating the door sections.
Sections	Door Section Array	One or more door sections. All will move when the door opens and closes.
Audio Open	[AudioClip][unity-audioclip]	The audio to play when the door is unlatched and opened.
Audio Close	[AudioClip][unity-audioclip]	The audio to play when the door closes and latches.
Audio Locked	[AudioClip][unity-audioclip]	The audio to play when attempting to open the door while locked.
Audio Unlock	[AudioClip][unity-audioclip]	The audio to play when unlocking or locking the door.

Door Section

NAME	TYPE	DESCRIPTION
Transform	Transform	The door section transform.
Offset	Vector3	The offset from starting (closed) position when opened.

See Also

Doors

Unity AnimationCurve

SoloCharacterTriggerZone MonoBehaviour

Overview

The SoloCharacterTriggerZone behaviour fires script events when a character enters and exits the collider.

Inspector

Properties

NAME	TYPE	DESCRIPTION
On Trigger Enter	UnityEvent	The event that is fired when a character enters the trigger collider.
On Trigger Exit	UnityEvent	The event that is fired when a character exits the trigger collider.

See Also

Layers And Tags

Unity Events

Unity BoxCollider

Unity SphereCollider
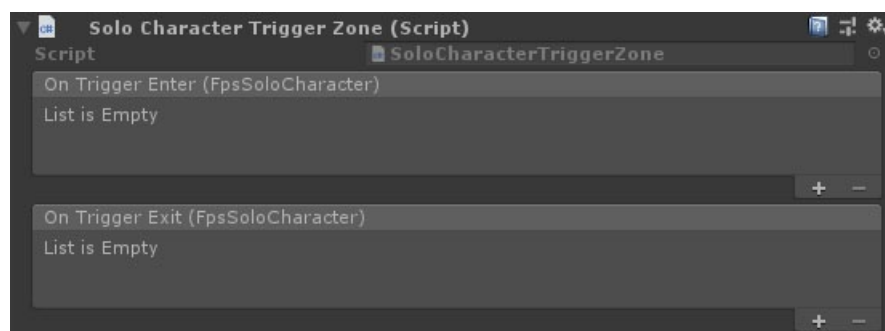
Unity CapsuleCollider

SoloCharacterTriggerZonePersistant MonoBehaviour

Overview

The SoloCharacterTriggerZonePersistant behaviour fires unity events when characters enter and exit.
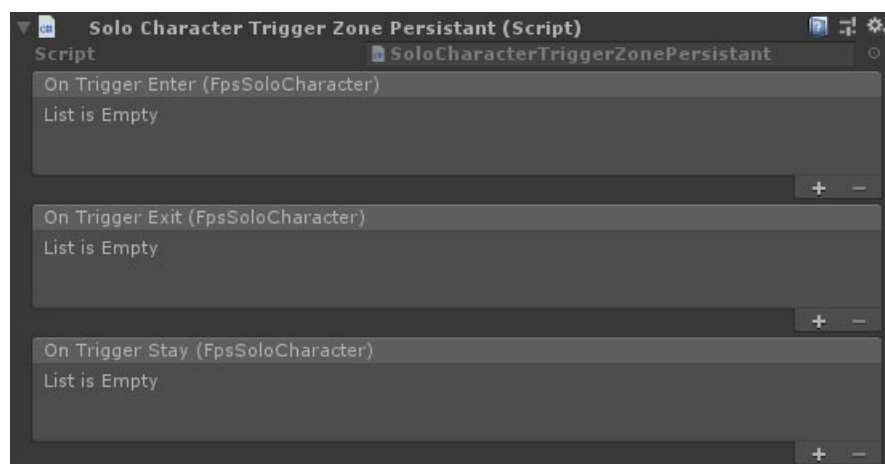
Inspector

Properties

NAME	TYPE	DESCRIPTION
On Trigger Enter	UnityEvent	The event that is fired when a character enters the trigger collider.
On Trigger Exit	UnityEvent	The event that is fired when a character exits the trigger collider.
On Trigger Stay	UnityEvent	The event that is fired each frame a character stays inside the trigger collider.

See Also

Layers And Tags

Unity Events

Unity BoxCollider

Unity SphereCollider

Unity CapsuleCollider

TriggerZoneColliderCounter MonoBehaviour

Overview

The TriggerZoneColliderCounter behaviour tracks and counts all the colliders that enter it with the specified layer mask.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Valid Layers	LayerMask	The valid layers for objects to track. Colliders on another layer will be ignored.

See Also

Layers And Tags

Unity BoxCollider

Unity SphereCollider

Unity CapsuleCollider

Audio Systems

Overview

NeoFPS has a number of audio systems for specific cases. Some are specific or important to first person shooter games, while others are features that are missing from Unity and have had simple versions implemented in order to provide a more complete game experience out of the box.

Systems

Footsteps

In first person games, more than any other genre, footsteps are an important part of the immersive experience. NeoFPS uses a number of tools that tie footsteps to the character motion states and give designers full control over how the character's feet interact with the world.

Character Audio

NeoFPS has a customisable system that allows code and events to trigger character specific audio clips. The clips are identified using generated constants, which allows developers to expand the availabe clips with ease.

Character audio is implemented through a combination of the FpsCharacterAudioHandler MonoBehaviour and the FpsCharacterAudioData ScriptableObject.

Surface Audio

NeoFPS has a simple in-development system for identifying surfaces (LINK). The SurfaceAudioData ScriptableObject is used in a number of places throughout NeoFPS to specify audio for things such as footsteps, bullet hits and slides.

Contact Audio

The ClipSetContactAudioHandler MonoBehaviour is a simple component that can be added to physics objects in order to play audio on collisions. The audio is picked at random from a set of valid clips. You can also use a SurfaceContactAudioHandler MonoBehaviour which uses the surface system to pick the correct audio to play.

See Also

Surfaces

Generated Constants

Footsteps

Overview

Footsteps in NeoFPS are handled using a combination of MonoBehaviours and motion graph behaviours. This allows a great deal of flexibility and control over when and how footsteps are triggered. For more information on the specific behaviours, follow the links below.

Footstep Implementation

Basic Footsteps

The motion graph SurfaceFootstepAudioSystem is component added to the character that is used for basic footsteps on various surfaces. It is controlled via the SurfaceFootstepAudioBehaviour motion graph behaviour. This has properties for adjusting the frequency of the steps, the surface audio library to use, and parameters such as the cast direction and distance for surface checks. For each step taken it will choose an audio clip at random from the surface audio library to play. The behaviour can be attached to multiple motion graph states with different properties, so the system can be used to trigger different audio clips based on the character's motion state such as running, sneaking, walking, etc. The component on the character also fires a C# event on steps that can be used to tie it in to other systems.

Sliding

The motion graph SlidingAudioBehaviour is used to play looping audio when the character is sliding on various surfaces. Properties are available to change the surface audio library, and to pitch shift the audio based on the speed of the character.

Lift-offs

The motion graph SurfaceAudioBehaviour can be used to trigger surface based audio clips in various situations, including jump lift-offs. It can be combined with the motion graph PlayCharacterAudioBehaviour to trigger character grunts and exertion noises when jumping.

Ladders

The motion graph LadderAudioBehaviour is used to play audio clips as the character ascends and descends a ladder. It has properties to control the frequency of the audio clips based on the climb speed, as well selecting the relevant surface audio library.

Landings

Landings are handled slightly differently, as they need to react to the force of the impact in order to add flavour such as bone crunches or grunts for very heavy landings. When the character lands, the motion controller fires impact events that can be handled by any behaviour. The demo character implements this feature and has properties for soft and hard landings.

See Also

The Motion Graph

Motion Graph Behaviours

Motion Graph FootstepAudioBehaviour

Motion Graph SlidingAudioBehaviour

Motion Graph SurfaceAudioBehaviour

Motion Graph PlayCharacterAudioBehaviour

Motion Graph LadderAudioBehaviour

SurfaceAudioData

FPS Characters

AnimationEventAudioPlayer MonoBehaviour

Overview

The AnimationEventAudioPlayer behaviour is attached to objects with an Animator component and handles any animation events sent.

Inspector
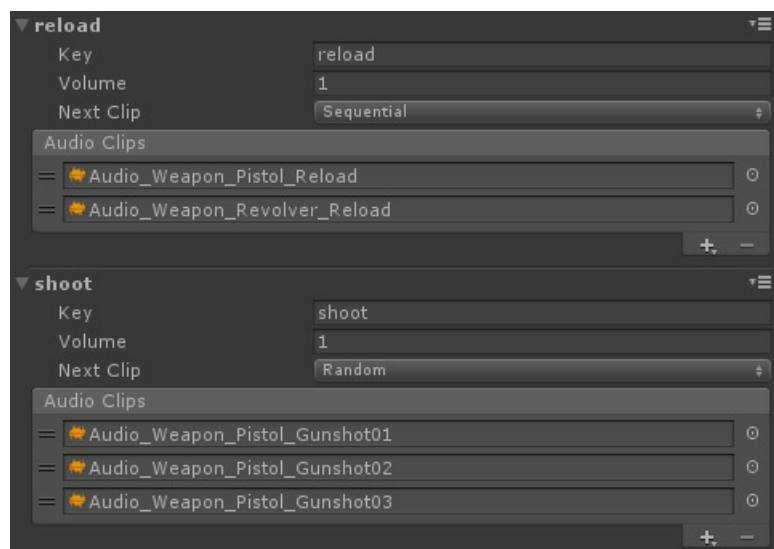
Properties

NAME	TYPE	DESCRIPTION
Audio Source	AudioSource	The audio source to play from.

The **Add New Set** button will add a new audio clip set to the component with the following properties:

NAME	TYPE	DESCRIPTION
Key	String	The name of the clip set, used as the parameter of the animation events
Next Clip	Dropdown	How the next clip should be selected. Available options are: **Sequential** loops through the clips in order while **Random** selects a clip at random.
Volume	Float	The volume to play the clip at.
Clips	AudioClip Array	The audio clips to choose from.

Triggering Audio Clips

The audio clips are triggered by adding animation events to the animation clips of the animated object. This is usually done through the import settings in the inspector.
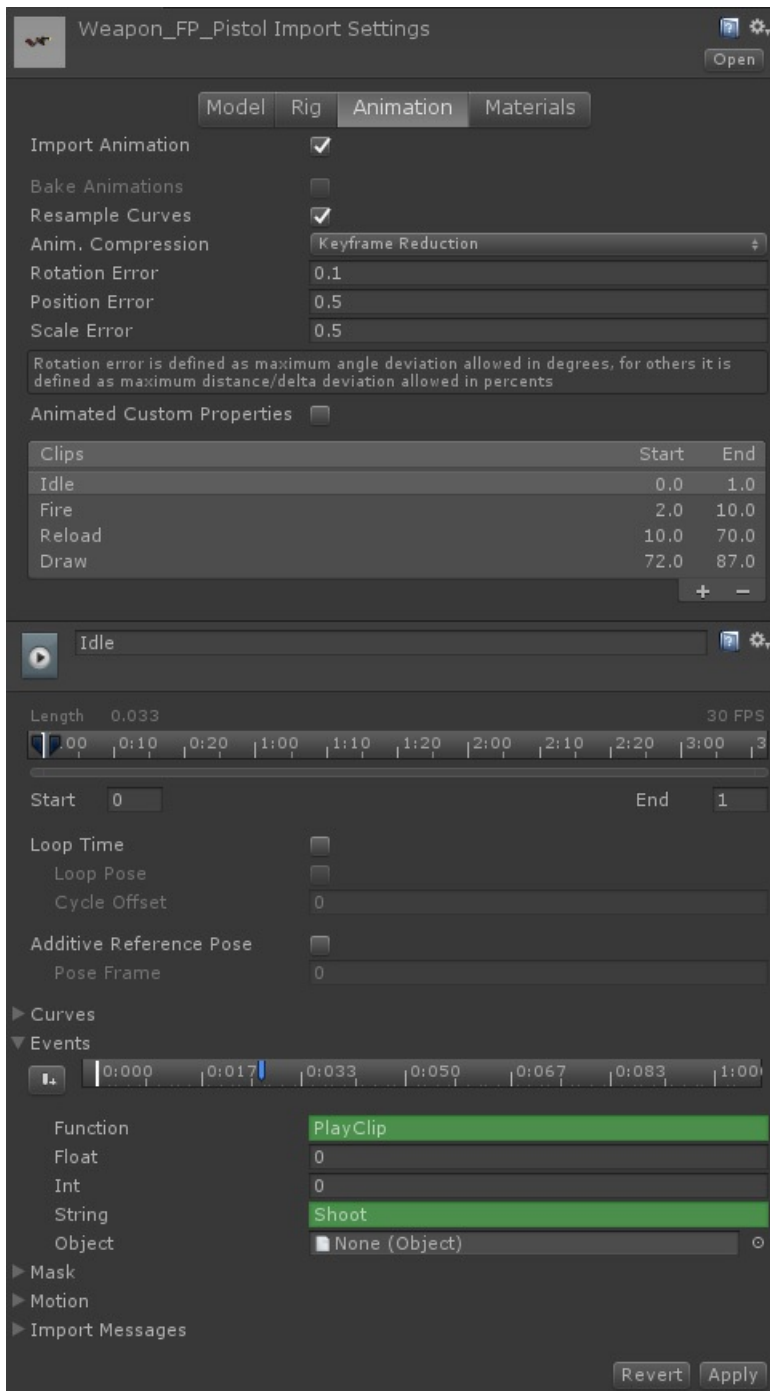
The above example has an event set up that plays an audio clip from the "Shoot" set. The important properties are the **Function** property which must be set to **Play Clip**, and the **String** property which should match the **Key** in the AnimationEventAudioPlayer.

See Also

Animation Events

AudioTimeScalPitchBend MonoBehaviour

Overview

The AudioTimeScalPitchBend behaviour alters the pitch of an audio source to match changes to time scale. Pitch also affects the duration of an audio clip, so halving the pitch will double the duration. By default, Unity does not slow audio to match the time scale, so this behaviour compensates for that.

Inspector
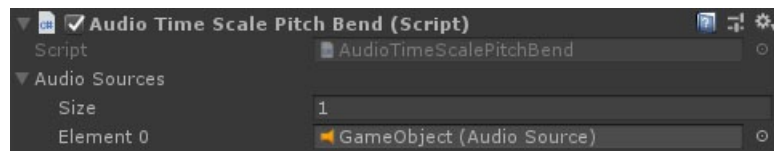
Properties

NAME	TYPE	DESCRIPTION
Audio Sources	AudioSource Array	The audio sources to modify based on time scale.

ClipSetContactAudioHandler MonoBehaviour

Overview

The ClipSetContactAudioHandler behaviour is attached to physics objects and triggers audio clips when they collide with other objects or the environment.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Min Impulse	Float	The minimum impulse between this object and the object it collides with for an impact noise to be played.
Min Delay	Float	The minimum time between impact sounds.
Clips	AudioClip Array	The audio clips to choose from on impact.

FpsCharacterAudioData ScriptableObject

Overview

The FpsCharacterAudioData scriptable object specifies audio to play for the different character audio keys.

Inspector
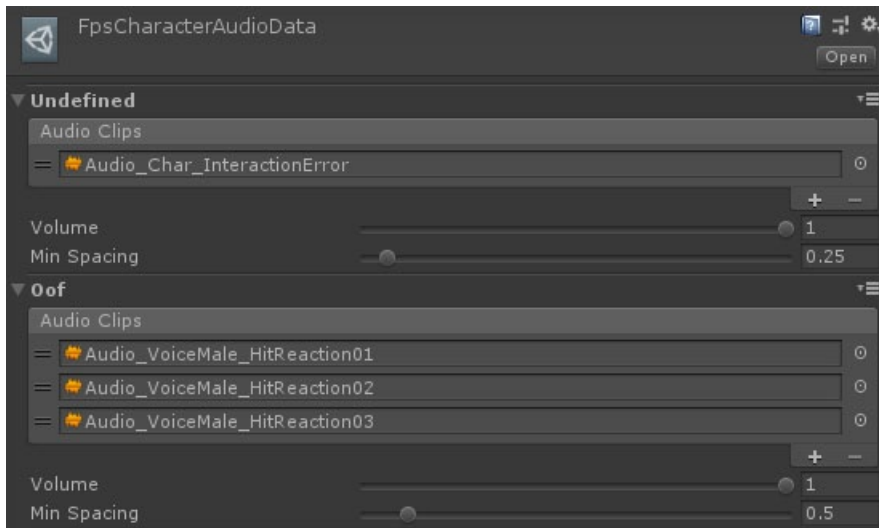
Properties

The data is divided into sections for each key in the FpsCharacterAudio generated constant. Each section contains the following properties:

NAME	TYPE	DESCRIPTION
Audio Clips	AudioClip Array	A selection of audio clips to pick from. Will be selected at random to prevent repetition.
Volume	Float	The volume to play the clip at.
Min Spacing	Float	New clips will be blocked from playing for this duration after a clip plays. Prevents rapid fire audio.

See Also

Generated Constants

Unity AudioClip

FpsCharacterAudioHandler MonoBehaviour

Overview

The FpsCharacterAudioHandler plays audio clips from a library of character audio files.

Inspector
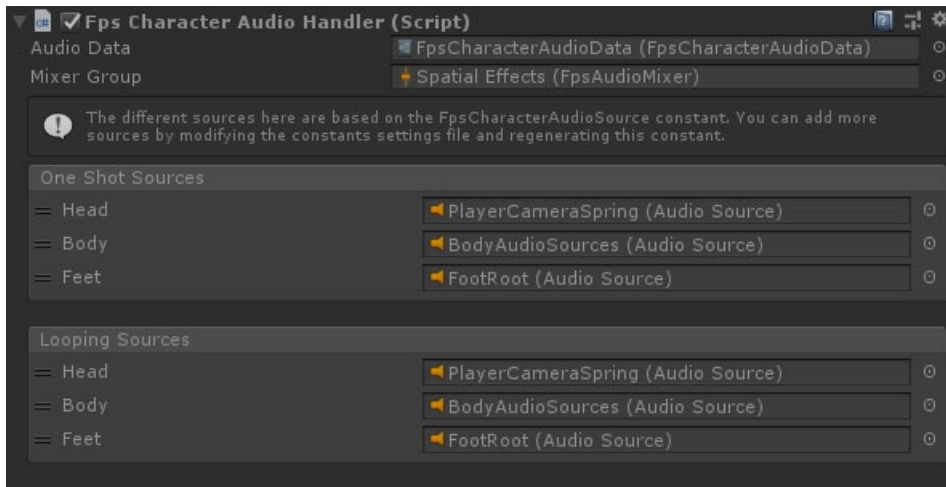
Properties

NAME	TYPE	DESCRIPTION
Audio Data	FpsCharacterAudioData	The character audio library to use.
Mixer Group	MixerGroup	The mixer group for character sound effects.
Sources	AudioSource Array	The sources to use for playing audio. One for each FpsCharacterAudioSource generated constant.

See Also

FpsCharacterAudioData

Generated Constants

NeoFpsAudioManager ScriptableObject

Overview

The NeoFpsAudioManager scriptable object specifies the audio mixer and outputs for audio in a NeoFPS project, and provides pooled audio sources for different effects.

Inspector
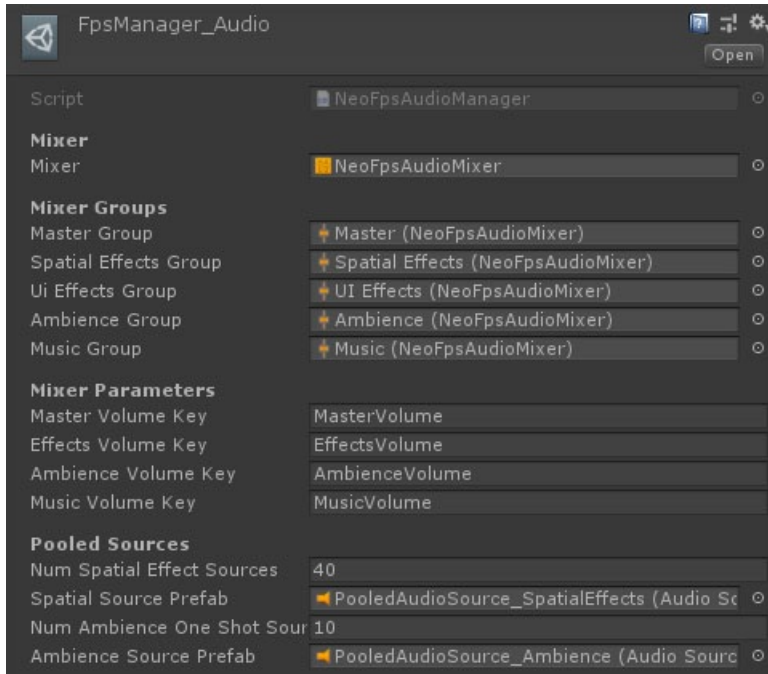
Properties

NAME	TYPE	DESCRIPTION
Mixer	[AudioMixer][unity-audiomixer]	The audio mixer for the project.
Master Group	[AudioMixerGroup][unity-audiomixer]	The audio mixer group that controls the overall volume.
Spatial Effects Group	[AudioMixerGroup][unity-audiomixer]	The audio mixer group used to control volumes and filters for spatial sound effects.
Ui Effects Group	[AudioMixerGroup][unity-audiomixer]	The audio mixer group used to control volumes and filters for UI sound effects.
Ambience Group	[AudioMixerGroup][unity-audiomixer]	The audio mixer group used to control volumes and filters for ambient sound effects and looping ambient audio.
Music Group	[AudioMixerGroup][unity-audiomixer]	The audio mixer group used to control volumes and filters for music.
Master Volume Key	String	The name of the master volume parameter on the audio mixer.
Effects Volume Key	String	The name of the volume parameter that controls sound effects volume.
Ambience Volume Key	String	The name of the volume parameter on the audio mixer for ambient loops and effects.

NAME	TYPE	DESCRIPTION
Music Volume Key	String	The name of the volume parameter on the audio mixer for the music audio.
Num Spatial Effect Sources	Integer	The number of pooled audio sources for spatial sound effects.
Spatial Source Prefab	[AudioSource][unity-audiosource]	An optional prefab for the spatial effects sources. If not provided then the objects will be created from scratch.
Num Ambience One Shot Sources	Integer	The number of pooled audio sources for ambient sound effects.
Ambience Source Prefab	[AudioSource][unity-audiosource]	An optional prefab for the ambience effects sources. If not provided then the objects will be created from scratch.

See Also

Surfaces

Generated Constants

[Unity AudioClip][unity-audioclip]

[AudioMixer][unity-audiomixer]: https://docs.unity3d.com/Manual/class-AudioMixer.html [unity-audioclip]: https://docs.unity3d.com/Manual/class-AudioClip.html [unity-audiosource]: https://docs.unity3d.com/Manual/class-AudioSource.html

SurfaceAudioData ScriptableObject

Overview

The SurfaceAudioData scriptable object specifies audio clips for each surface type. It is used to organise clip sets such as impact sounds, footsteps and slides.

Inspector
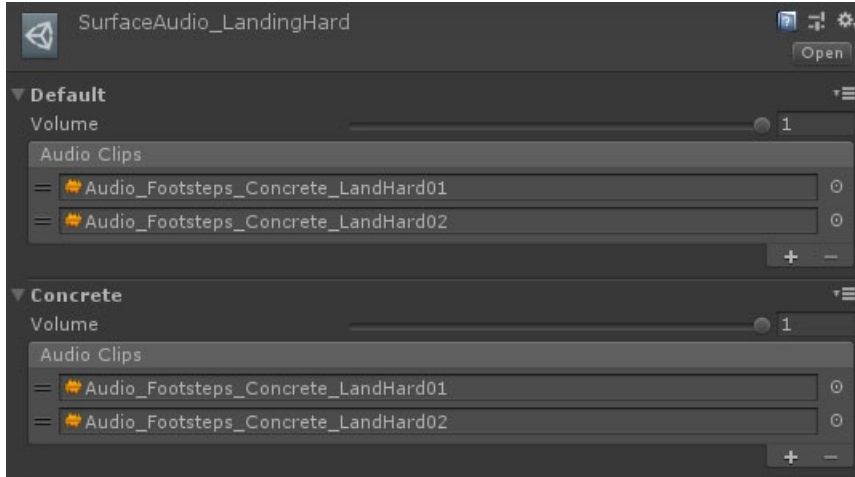
Properties

The SurfaceAudioData entries are grouped based on the Surface generated constant. Each surface has the following properties.

NAME	TYPE	DESCRIPTION
Volume	Float	The volume to play the clips at.
Clips	AudioClip Array	Audio clips (will be picked at random).

See Also

Surfaces

Generated Constants

Unity AudioClip

SurfaceContactAudioHandler MonoBehaviour

Overview

The SurfaceContactAudioHandler behaviour is attached to physics objects triggers sound effects when they collide with other objects or the environment. Playing the audio itself is handled by the SurfaceManager.

Inspector
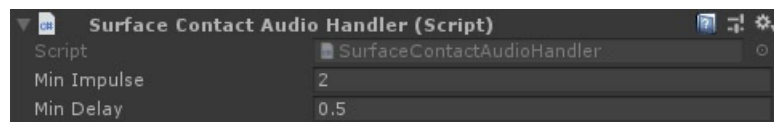
Properties

NAME	TYPE	DESCRIPTION
Min Impulse	Float	The minimum impulse between this object and the object it collides with for an impact noise to be played.
Min Delay	Float	The minimum time between impact sounds.

See Also

Surface Manager

Inventory

Overview

NeoFPS has its own inventory system that it uses for all of its weapons and items. The inventory is implemented as separate components to the itemas themselves so it should be easy to replace with an alternative system if desired.

The NeoFPS inventory system is implemented in 2 parts: inventory and quickslots.

Inventory
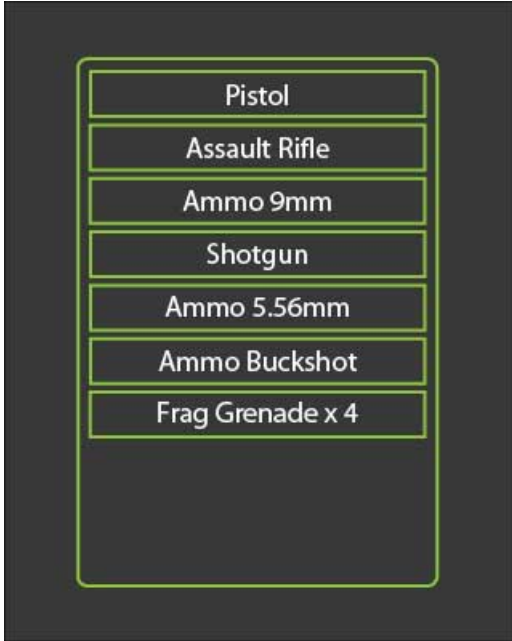
The inventory is a container of items. It manages item ownership and quantities and invokes events to react to changes.

The demo implementations of the inventory are not fixed. The inventory is referenced externally via the `IInventory` interface, meaning that it can be swapped out with another implementation and the system as a whole will adapt.

Quick Slots

The quick-slots system binds items to slots that can be mapped to inputs. In a simple, traditional FPS system that would mean each slot being mapped to a number key on the keyboard. It could also mean a slot for each direction on a game controller d-pad.

As with the inventory, the implementation is not fixed. It is referenced externally via the `IQuickSlots` interface and, like the inventory, the implementation can be swapped and the system will adapt.

NeoFPS comes with a number of example inventories that implement quick slots in different ways to model popular first person shooters. For more information see Inventory Examples.

Items And Quick-Slot Items

Items are objects that implement the `IInventoryItem` interface. The interface has properties for quantity and ownership, along with methods that are called when added or removed from an inventory.

Quick-Slot items implement the `IQuickSlotItem` interface. This has properties and methods for selection and for dropping.

Inventory Pickups

Inventory items pickups can be interactive, meaning you need to look at them and hit use, or contact based, meaning you need to walk over them.

The simplest way to create an inventory pickup is using the Pickup Wizard in the NeoFPS Hub. There are also specialised pickups for modular firearms that track ammo count, and for other wieldable items such as melee and thrown weapons. These can all be created from the wizard.

The Inventory Database
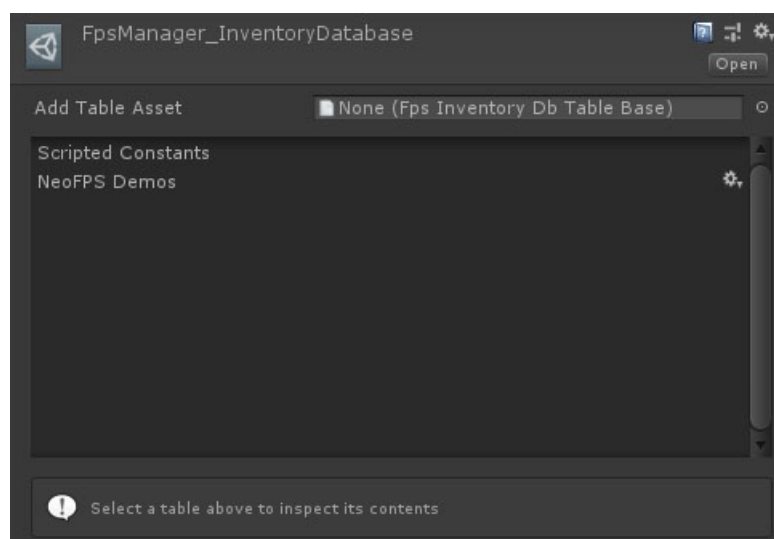
The inventory database is a system for managing the inventory keys used to identify each unique item. The database is made up of tables, which are used to gather inventory IDs together. The inventory system used to use GeneratedConstant based keys. The new database system has one fixed table that references the old **FpsInventoryKey** constants and uses them as keys. This allows you to use the constants to reference inventory item IDs directly in code - useful for items like the keyring, which is a unique, but constant item.

You can create an inventory database table for your project by right-clicking in the project browser and selecting *Create/NeoFPS/Inventory/Database Table*. This allows you to add your own keys without the risk of clashing with any changes to the NeoFPS demo inventory keys in future updates. Once you have created a table, add it to the database via the NeoFPS Hub: *Managers/Inventory Database*.

To assign an ID to an object, clicking the ID button will show the inventory database browser.

From here you can browse the different tables to select an ID. You can filter by name using the filter field at the top. You can also add new keys from the browser itself. In the NewEntry section, select the table you want to add to, input a name, and hit **Add New Entry** button. This will add the a new entry to the selected table, and apply its ID to the object you are editing.

Inventory Loadouts

Inventory loadouts are a simple list of inventory objects that should be added to a character's inventory.
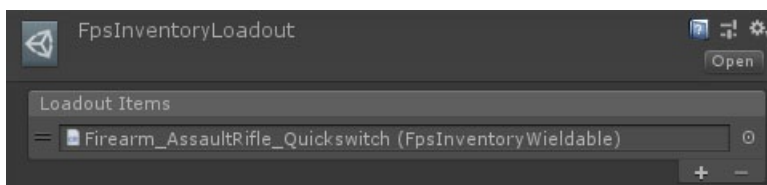
You can assign this to a game mode component, and it will apply the loadout on spawning a player character instead of using the character's starting items.

See Also

Inventory Examples

Inventory Examples

Overview

NeoFPS comes with a number of example inventories that are modelled after popular first person shooters.

Inventory Base

All of the example inventories inherit from a single base class `FpsInventoryBase`. This base class handles both the inventory storage and the quick-slots. It also provides inspector properties for initialising the inventory's starting contents and the priority order for the quick slots.

Items and Wieldables

The example inventory items inherit from a single base class `FpsInventoryItem`. The example inventory does not require that the items it stores use this behaviour, but all of the provided examples do.

The `FpsInventoryWieldable` behaviour is a subclass of `FpsInventoryItem` that also inherits `IQuickSlotItem`. It is intended as the base for all items that the character holds and uses such as weapons and consumable items (health packs, tools, scanners, etc).

Standard PC Inventory

The "standard" quick-switch inventory emulates old-school FPS games. Each weapon has a set slot corresponding to the keyboard keys 1 to 0. You can also cycle through the weapons using the mouse wheel and inputs for previous and next weapon. Lastly, the inventory also implements a quick-switch system so you can switch between your current and last weapon as in games such as Counter Strike.

Most of the NeoFPS demos use a character with the standard inventory. You can find weapon and item prefabs at the following locations:

- *Assets\NeoFPS\Samples\Shared\Prefabs\Weapons\QuickSwitchInventory* for the first person weapons and pickups/drops.
- *Assets\NeoFPS\Samples\Shared\Prefabs\Weapons\Ammo* for the ammo pickups

Stacked Inventory

The stacked inventory emulates games like half-life with larger, more structured inventories than the older FPS games could manage. Weapons and items are organised into stacks. Each stack holds a specific type of weapon such as melee, pistols, heavy, thrown. Selecting the same slot multiple times cycles through the weapons in that group. The mouse wheel and previous / next weapon inputs cycle through each of the weapons in a stack before mmoving onto the previous / next stack slots. The order of the weapons within a stack is fixed.

The slot index of an item added to a stacked inventory must take the stack into account. The maximum stack size is 10. Slot indices

0-9 fall in the first stack, 10-19 in the second, 20-29 in the third, and so on.

The stacked inventory requires a different HUD setup to the others due to how it organises the items.

This inventory is also well suited to console based FPS games where, for example, each direction on the D-Pad is a stack.

The relevant assets for the stacked inventory can be found at the following locations:

- *Assets\NeoFPS\Samples\SinglePlayer\Scenes\FeatureDemos\Inventory\FeatureDemo_InventoryStacked.unity* for a demo scene using the inventory
- *Assets\NeoFPS\Samples\SinglePlayer\Scenes\FeatureDemos\Inventory\Inventory_CharacterStackedInventory.prefab* for a player character that uses the stacked inventory
- *Assets\NeoFPS\Samples\Shared\Prefabs\Weapons\StackedInventory* for the first person weapons and pickups/drops.
- *Assets\NeoFPS\Samples\SinglePlayer\Scenes\FeatureDemos\Inventory\Inventory_HUDStacked.prefab* for a replacement inventory HUD that is set up for stackedd inventories

Swappable Inventory

The swappable inventory emulates FPS games that keep a constrained inventory such as 2 primary weapons, one melee and one throwable. This is a common inventory setup since more FPS games have started launching on console alongside or even ahead of PC.

The swappable inventory groups weapons and items into types similarly to the stacked inventory. A group can hold a fixed amount of weapons. If the group is full then the last selected weapon from that group is dropped when the character picks up another weapon of that type. In simple cases a group will only be able to hold a single weapon or item. A group can also be larger, and will encompass multiple quick slots. In this case, the order of weapons within the group is defined by the order they are dropped and picked up only.

The relevant assets for the swappable inventory can be found at the following locations:

- *Assets\NeoFPS\Samples\SinglePlayer\Scenes\FeatureDemos\Inventory\FeatureDemo_InventorySwappable.unity* for a demo scene using the inventory
- *Assets\NeoFPS\Samples\SinglePlayer\Scenes\FeatureDemos\Inventory\Inventory_CharacterSwappableInventory.prefab* for a player character that uses the swappable inventory
- *Assets\NeoFPS\Samples\Shared\Prefabs\Weapons\SwappableInventory* for the first person weapons and pickups/drops.

See Also

Inventory

FpsInventoryAmmo MonoBehaviour

Overview

The FpsInventoryAmmo behaviour is an ammo type for the modular firearms that is stored in a character's inventory. It allows the ammo to be shared between different weapons.

Inspector
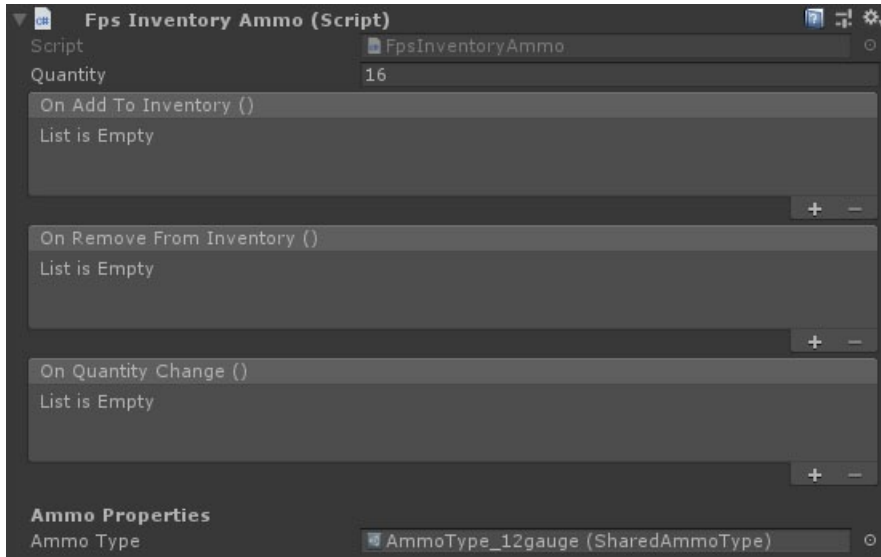
Properties

The FpsInventoryAmmo behaviour inherits the properties from FpsInventoryItem. It also adds the following:

NAME	TYPE	DESCRIPTION
Ammo Type	SharedAmmoType	The type of ammo.

See Also

FpsInventoryItem

Modular Firearms

SharedAmmoType

FpsInventoryItem MonoBehaviour

Overview

The FpsInventoryItem behaviour is an object that can be stored in a character inventory.

Inspector
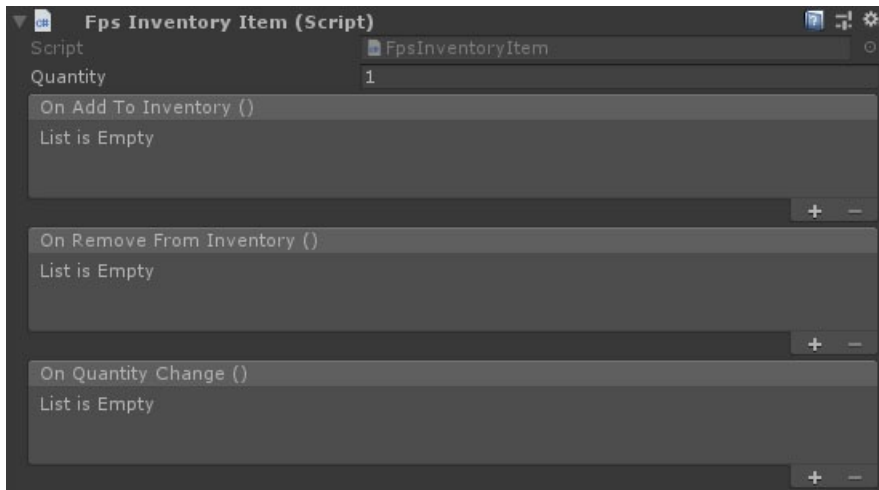
Properties

NAME	TYPE	DESCRIPTION
Quantity	Int	The quantity of items in the stack.
On Add To Inventory	UnityEvent	An event that is invoked when the object is first added to the character inventory.
On Remove From Inventory	UnityEvent	An event that is invoked when the object is completely removed from the character inventory.
On Quantity Change	UnityEvent	An event that is invoked when the quantity of objects in the stack changes.

See Also

Inventories

Unity Events

FpsInventoryQuickSwitch MonoBehaviour

Overview

The quick-switch inventory is a standard PC FPS inventory. Pressing the appropriate number button selects the item in that quick slot. Pressing the *switch weapons* button will switch to the previous selected item.

Inspector
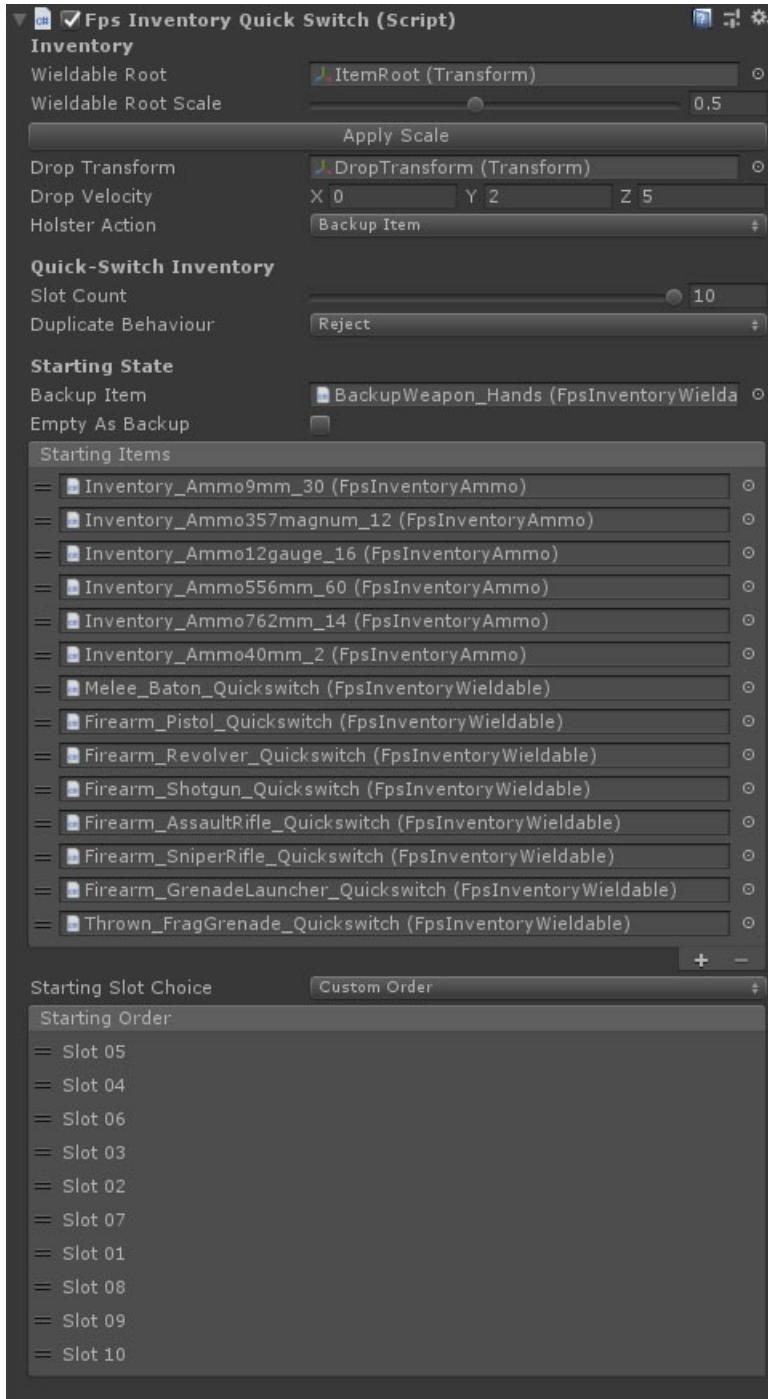
Properties

NAME	TYPE	DESCRIPTION
Wieldable Root	Transform	The transform to set as the parent of any objects added to the inventory.
Wieldable Root Scale	Float	A scale value for the wieldable root and any child items. Used to prevent weapons clipping into the scenery.

NAME	TYPE	DESCRIPTION
Drop Transform	Transform	A proxy transform used to set the drop position and rotation when a wieldable item is dropped.
Drop Velocity	Vector3	The velocity of any dropped items relative to the character forward direction.
Holster Action	Dropdown	What should be selected when you holster your weapon. Options are **Backup Item**, **Nothing**.
Slot Count	Int	The number of item quick slots.
Starting Slot Choice	Dropdown	The selection method for the starting slot. Options are **Ascending**, **Descending**, **Custom Order**.
Starting Order*	Int Array	This array specifies the selection order on start. The highest on the list that exists will be the starting selection.
Backup Item	FpsInventoryWieldable	An item to use if no wieldables are in the inventory. This could be empty hands or an infinite weapon such as a knife.
Empty As Backup	Boolean	If this is true, then selecting an empty slot will switch to the backup item.
Starting Items	FpsInventoryItem Array	A selection of inventory items to be added to the inventory on start.
Duplicate Behaviour	Dropdown	What to do when trying to add an item to the inventory that already exists. Options are **Reject**, **DestroyOld**, **DropOld**.

* This property is only visible if the starting slot choice is set to Custom Order.

See Also

Inventory Examples

FpsInventoryItem

FpsInventoryWieldable

FpsInventoryStacked MonoBehaviour

Overview

The stacked inventory groups items together into stacks. Selecting a stack multiple times cycles through the items in it. Similar to the inventory system in Half-Life.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Wieldable Root	Transform	The transform to set as the parent of any objects added to the inventory.
Wieldable Root Scale	Float	A scale value for the wieldable root and any child items. Used to prevent weapons clipping into the scenery.
Drop Transform	Transform	A proxy transform used to set the drop position and rotation when a wieldable item is dropped.
Drop Velocity	Vector3	The velocity of any dropped items relative to the character forward direction.
Holster Action	Dropdown	What should be selected when you holster your weapon. Options are **Backup Item**, **Nothing**.
Stack Count	Int	The number of available stacks.

NAME	TYPE	DESCRIPTION
Starting Slot Choice	Dropdown	The selection method for the starting slot. Options are **Ascending**, **Descending**, **Custom Order**.
Starting Order*	Int Array	This array specifies the selection order on start. The highest on the list that exists will be the starting selection.
Backup Item	FpsInventoryWieldable	An item to use if no wieldables are in the inventory. This could be empty hands or an infinite weapon such as a knife.
Empty As Backup	Boolean	*This setting does not currently work with the stacked inventory.*
Starting Items	FpsInventoryItem Array	A selection of inventory items to be added to the inventory on start.
Duplicate Behaviour	Dropdown	What to do when trying to add an item to the inventory that already exists. Options are **Reject**, **DestroyOld**, **DropOld**.

* This property is only visible if the starting slot choice is set to Custom Order.

See Also

Inventory Examples

FpsInventoryItem
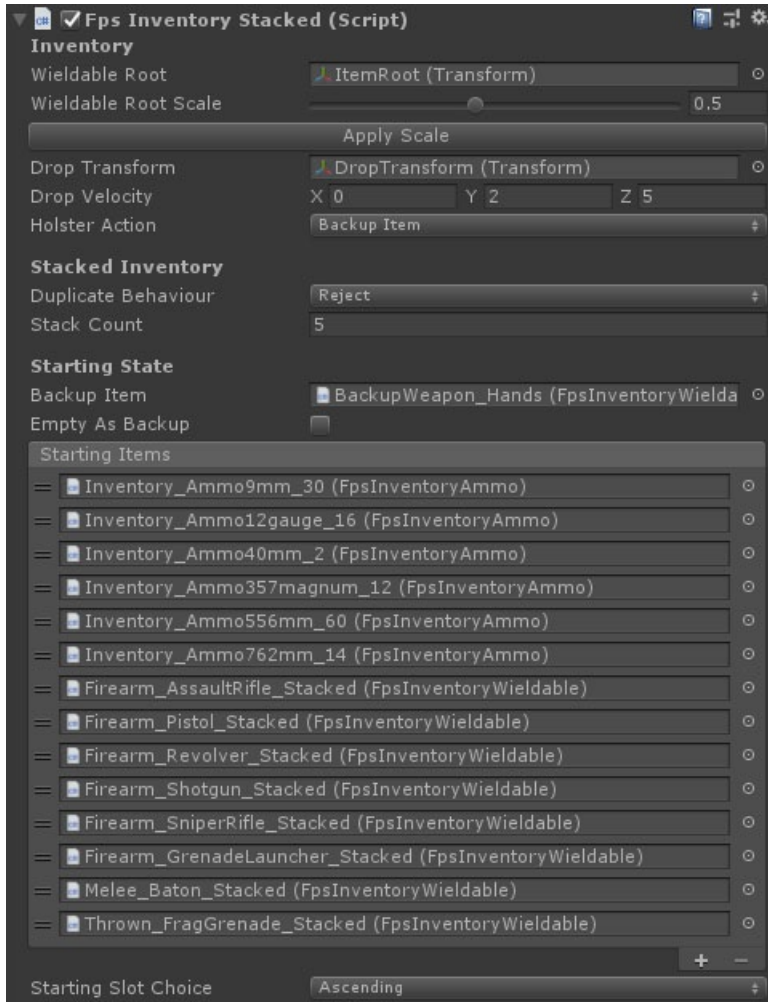
FpsInventoryWieldable

FpsInventorySwappable MonoBehaviour

Overview

The swappable inventory groups items into categories. If the quick slots for a category are all full then picking up a new item of that category will drop the current or last accessed item of that category.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Wieldable Root	Transform	The transform to set as the parent of any objects added to the inventory.
Wieldable Root Scale	Float	A scale value for the wieldable root and any child items. Used to prevent weapons clipping into the scenery.
Drop Transform	Transform	A proxy transform used to set the drop position and rotation when a wieldable item is dropped.
Drop Velocity	Vector3	The velocity of any dropped items relative to the character forward direction.
Swap Action	Dropdown	What to do when replacing an old item with a new one. Options are **Drop**, **Destroy**.
Holster Action	Dropdown	What should be selected when you holster your weapon. Options are **Backup Item**, **Nothing**.
Group Sizes	Int	The number of quick slots available for each category. The categories are defined in a generated constant called **FpsSwappableCategory**. Extending this constant will add new categories in the FpsInventorySwappable editor.

NAME	TYPE	DESCRIPTION
Starting Slot Choice	Dropdown	The selection method for the starting slot. Options are **Ascending**, **Descending**, **Custom Order**.
Starting Order*	Int Array	This array specifies the selection order on start. The highest on the list that exists will be the starting selection.
Backup Item	FpsInventoryWieldable	An item to use if no wieldables are in the inventory. This could be empty hands or an infinite weapon such as a knife.
Empty As Backup	Boolean	If this is true, then selecting an empty slot will switch to the backup item.
Starting Items	FpsInventoryItem Array	A selection of inventory items to be added to the inventory on start.
Duplicate Behaviour	Dropdown	What to do when trying to add an item to the inventory that already exists. Options are **Reject**, **DestroyOld**, **DropOld**, **Allow**. The last option will allow you to pick up multiple weapons of the same type.

* This property is only visible if the starting slot choice is set to Custom Order.

See Also

Inventory Examples

FpsInventoryItem

FpsInventoryWieldable
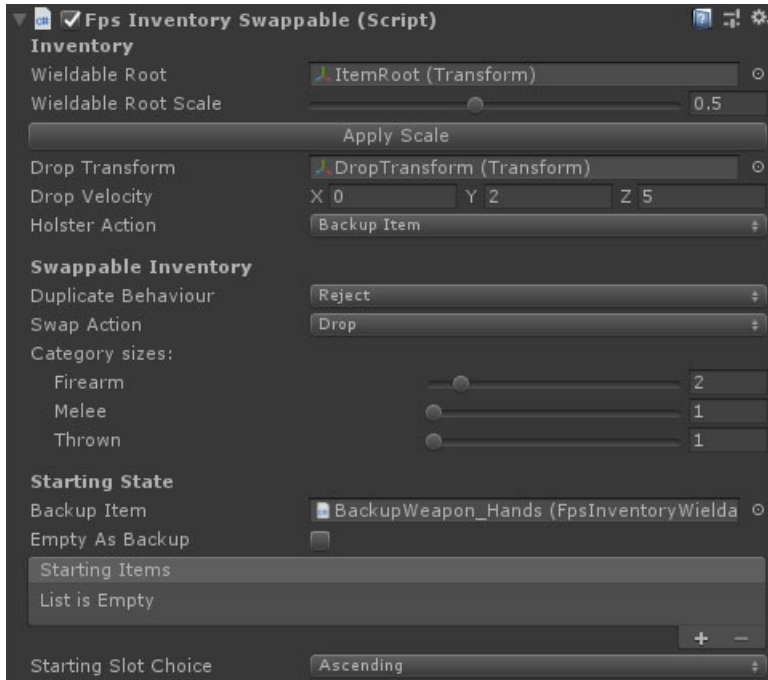
FpsInventoryWieldable MonoBehaviour

Overview

The FpsInventoryWieldable behaviour is an inventory item that can be held in the hands and used, such as a weapon.

Inspector

Properties

The FpsInventoryWieldable behaviour inherits from the FpsInventoryItem. Check the reference for information on its properties.

NAME	TYPE	DESCRIPTION
Inventory ID	ID Picker	The inventory item key. Clicking this button will open the inventory database item picker.
Display Image	Sprite	The image to use in the inventory HUD.
Quick Slot	Int	The quick slot the item should be placed in. If you are using a stacked inventory, remember that each stack is 10 slots (0-9 = stack 1, 10-19 = stack 2, etc).
Max Quantity	Int	The maximum quantity you can hold.
Deselect Action	Dropdown	What to do when the weapon is deselected. Available options are: **Disable Game Object**, **Disable Wieldable Component** and **Nothing**.
On Select	Unity Event	An event called when the wieldable is selected. Use this to enable components, etc.

NAME	TYPE	DESCRIPTION
On Deselect	Unity Event	An event called when the wieldable is deselected. Use this to disable components, etc.
Drop Object	FpsInventoryWieldableDrop	The prefab to spawn when the wieldable item is dropped.

See Also

Inventory Examples

Generated Constants

FpsInventoryItem

FpsInventoryWieldableDrop

FpsInventoryWieldableDrop MonoBehaviour

Overview

The FpsInventoryWieldableDrop behaviour is a pickup that is spawned when a character drops a wieldable item.

Inspector
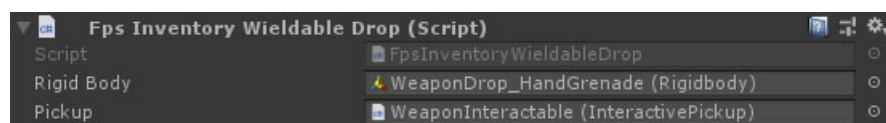
Properties

NAME	TYPE	DESCRIPTION
Rigidbody	Rigidbody	The object rigidbody for the drop. This will be thrown away from the character that drops it.
Pickup	InteractablePickup	The pickup for the item. This will be initialised with the correct quantity based on the dropper's inventory.

See Also

Inventory

Unity Rigidbody

FpsInventoryWieldable MonoBehaviour

Overview

The FpsInventoryWieldableSwappable behaviour is a variant of the FpsInventoryWieldable for the swappable inventory. It has a property for the wieldable category instead of a specific quick slot.

Inspector

Properties

The FpsInventoryWieldable behaviour inherits from the FpsInventoryItem. Check the reference for information on its properties.

NAME	TYPE	DESCRIPTION
Inventory ID	ID Picker	The inventory item key. Clicking this button will open the inventory database item picker.
Display Image	Sprite	The image to use in the inventory HUD.
Max Quantity	Int	The maximum quantity you can hold.
Deselect Action	Dropdown	What to do when the weapon is deselected. Available options are: **Disable Game Object**, **Disable Wieldable Component** and **Nothing**.
On Select	Unity Event	An event called when the wieldable is selected. Use this to enable components, etc.

NAME	TYPE	DESCRIPTION
On Deselect	Unity Event	An event called when the wieldable is deselected. Use this to disable components, etc.
Drop Object	FpsInventoryWieldableDrop	The prefab to spawn when the wieldable item is dropped.
Category	FpsSwappableCategory	The wieldable category.

See Also

Inventory Examples

Generated Constants

FpsInventoryItem

FpsInventoryWieldableDrop

InteractiveMultiPickup MonoBehaviour

Overview

The InteractiveMultiPickup behaviour is an object that will give a number of inventory items to the character that interacts with it.

Inspector

Properties

The InteractiveMultiPickup inherits from the InteractiveObject. Check the reference for information on its properties.

NAME	TYPE	DESCRIPTION
Items	FpsInventoryItem Array	The item prefabs to add to the character inventory.
Replenish	Boolean	Do the inventory items replenish. If the items are removed new items will be instantiated. If they are partially removed (to top up an existing item) the quantity will be reset afterwards.

See Also

InteractiveObject

FpsInventoryItem

InteractivePickup MonoBehaviour

Overview

The InteractivePickup behaviour is an object that must be interacted with to pick up and add an item to the character inventory.
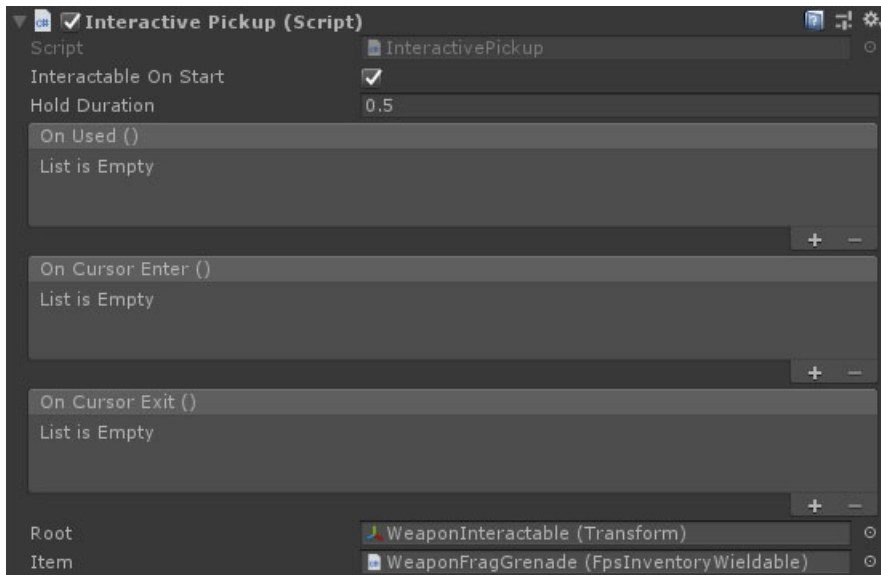
Inspector

Properties

The InteractiveMultiPickup inherits from the InteractiveObject. Check the reference for information on its properties.

NAME	TYPE	DESCRIPTION
Root	Transform	The root object (destroyed when the item is picked up).
Item	FpsInventoryItem	The item prefab to add to the character inventory.

See Also

InteractiveObject

FpsInventoryItem

InteractivePickupTrigger MonoBehaviour

Overview

The InteractivePickupTrigger behaviour is used to create interactive item pickups or powerups that must be used to consume.

Inspector
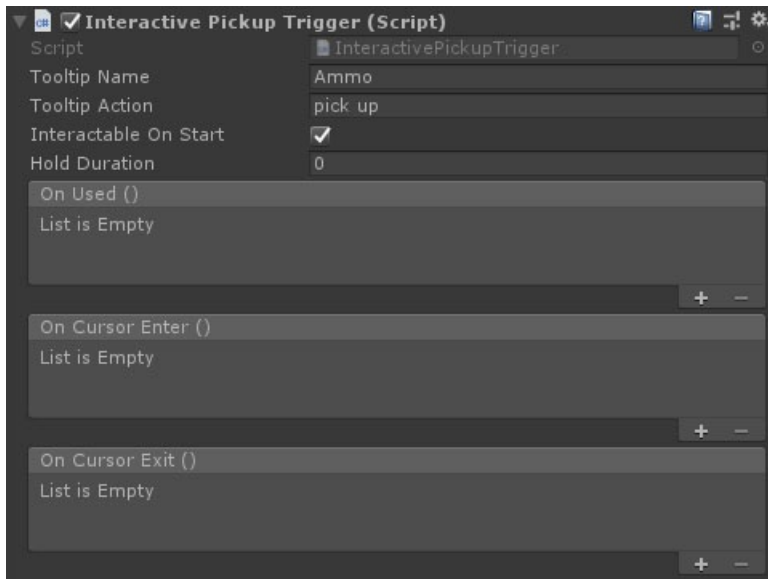
Properties

The InteractivePickupTrigger behaviour has no properties exposed in the inspector.

See Also

Inventory

InteractiveObject

InventoryItemPickup MonoBehaviour

Overview

The InventoryItemPickup behaviour is combined with a PickupTriggerZone to create a pickup that is added to the character inventory when walked over.

Inspector
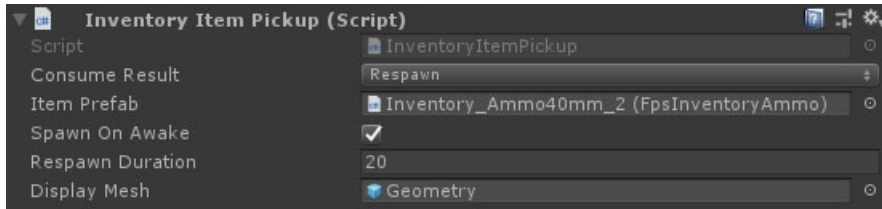
Properties

NAME	TYPE	DESCRIPTION
Consume Result	Dropdown	What to do to the pickup object once its item has been transferred to the character inventory. Options are **Destroy**, **Disable**, **Respawn**.
Item Prefab	FpsInventoryItem	The inventory item prefab to give to the character.
Spawn On Awake	Boolean	Should the pickup be spawned immediately, or triggered externally.
Respawn Duration	Float	How long to wait before respawning if the consume result is set to **Respawn**.
Display Mesh	GameObject	The display mesh of the pickup. This should not be the same game object as this, so that if this is disabled the pickup will still respawn if required.

See Also

InteractiveObject

FpsInventoryItem

PickupTriggerZone MonoBehaviour

Overview

The PickupTriggerZone behaviour is added to objects that also contain a pickup behaviour, and a collider set to act as a trigger. It captures trigger events from the collider and uses them to activate the pickup.

Inspector
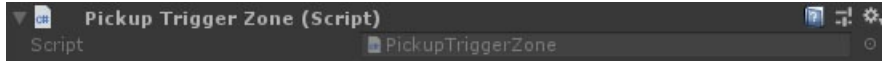
Properties

The PickupTriggerZone behaviour has no properties exposed in the inspector.

See Also

Inventory

Unity BoxCollider

Unity SphereCollider

Unity CapsuleCollider

FpsInventoryDbTable Scriptable Object

Overview

The FpsInventoryDbTable scriptable object contains a list of inventory items and manages their IDs. You can add new items to the inventory database from this object, or directly from the inventory item components themselves.

Inspector
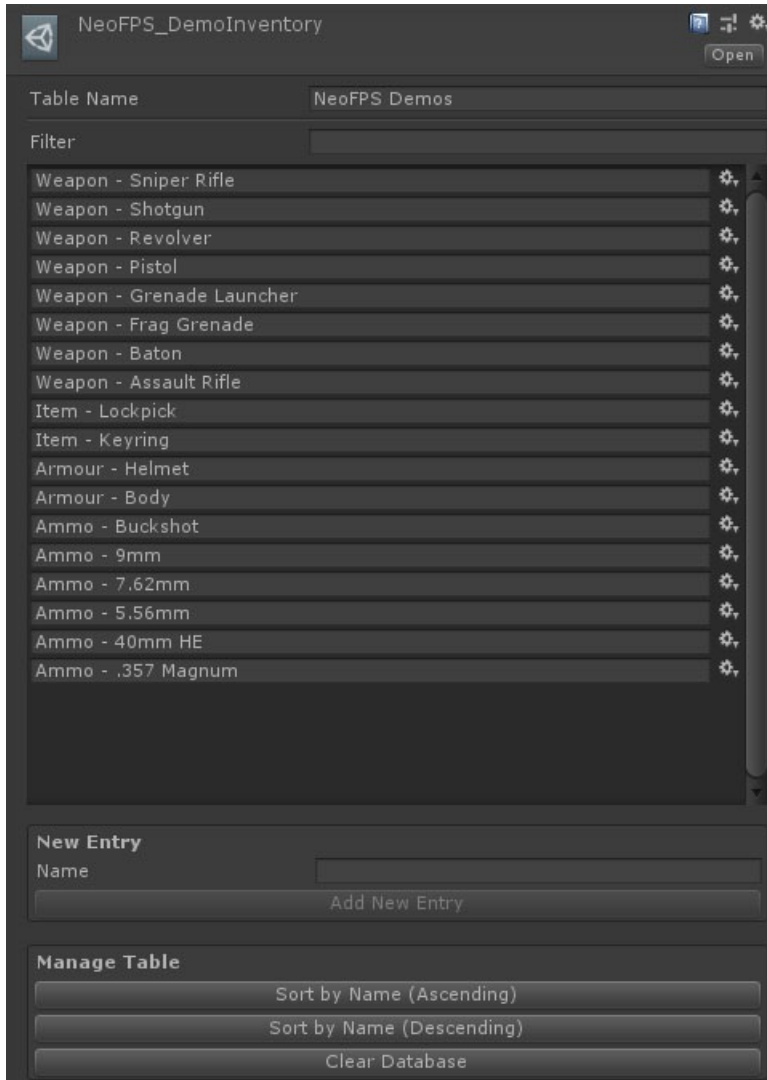
Properties

NAME	TYPE	DESCRIPTION
Table Name	String	A unique name for this table, to help differentiate them in the database and key picker.

The entries table shows each of the current entries in this table. You can click on the options button to the right of each entry for a list of context sensitive options, or you can click on the name to rename the item. Each item uses a unique ID code, so renaming items will not break connections.

You can sort the table using the **Sort By Name (Ascending)** and **Sort By Name (Descending)** buttons. You can also quickly clear all items with the **Clear All Items** button. Any objects that reference that item will show their ID as "Missing: (ID code)".

You can add new items by entering a name in the item name field and hitting the *Add Entry* button.

See Also

Inventory

NeoFpsInventoryDatabase Scriptable Object

FpsInventoryKeyDbTable Scriptable Object

Overview

The FpsInventoryKeyDbTable scriptable object is an inventory database table that takes its keys and names directly from the `FpsInventoryKey` constant. This allows the relevant keys to be accessed directly from code (useful for things like keyrings).

Inspector

The FpsInventoryKeyDbTable behaviour is only visible through the NeoFpsInventoryDatabase inspector.

Properties

The FpsInventoryKeyDbTable behaviour has no properties exposed in the inspector.

See Also

Inventory

NeoFpsInventoryDatabase Scriptable Object

FpsInventoryLoadout Scriptable Object

Overview

The FpsInventoryLoadout scriptable object is used to replace a character's starting inventory on spawn. These are assigned to the loadout property on the FpsSoloGameMinimal component.

Inspector
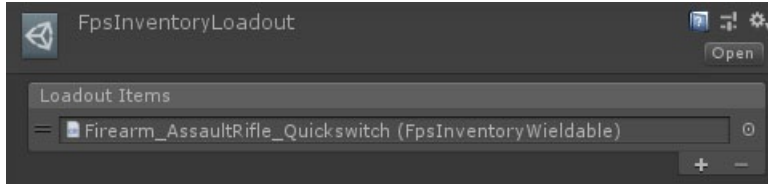
Properties

NAME	TYPE	DESCRIPTION
Items	FpsInventoryItem Array	A selection of inventory items to be added to the character inventory on start.

See Also

Inventory Examples

FpsInventoryItem

FpsInventoryWieldable

NeoFpsInventoryDatabase Scriptable Object

Overview

The NeoFpsInventoryDatabase scriptable object is a global inventory database that can be accessed from scripts at runtime and in editor code. It contains a list of database tables, allowing you to divide items up for projects and demos. The first table is always the FpsInventoryKeyDbTable. The other table type that is provided is the FpsInventoryDbTable, though more can be added by inheriting from the common base class.

Inspector
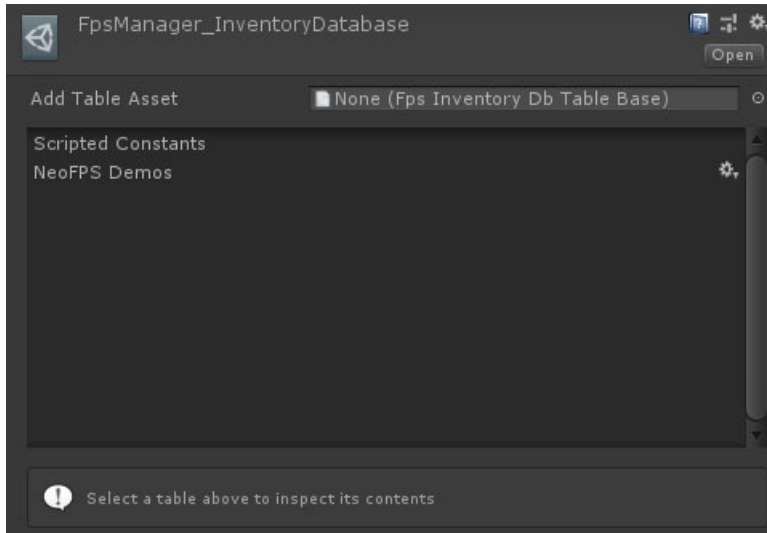
Properties

You can add new tables to the database by dragging and dropping it into the **Add Table Asset** field at the top of the database inspector.

Underneath this is a list of the current tables in use by the database. The options button at the side of each table will show a list of context options. Selecting a table will show its inspector below the list of tables.

See Also

Inventory

FpsInventoryKeyDbTable

Weapons

Overview

NeoFPS currently supports three types of weapons: melee, thrown and modular firearms.

Melee Weapons

Melee weapons are weapons such as clubs and swords that are held onto and swung at the enemy. For more information see Melee Weapons.

Thrown Weapons

Thrown weapons are items such as grenades and ninja stars that are stored in the character inventory and thrown by hand. For more information see Thrown Weapons.

Modular Firearms

Modular firearms are gun type weapons. They are assembled from a number of swappable modules that, together, model the behaviour of a firearm. For more information see Modular Firearms.

Explosions

The NeoFPS weapons can make use of a simple explosions system which deals damage and adds a repelling force to any objects in an explosion's blast radius. For more information see Explosions.

Wieldable Tools

Wieldable tools are items that are carried in the player character's hands and used with the primary or secondary fire buttons. They are assembled from a number of tool modules and actions. For more information see Wieldable Tools.

Demo Weapons

The following weapons are included in the samples in order to show off the different weapon features:

The baton is a simple demonstration of the melee weapons.
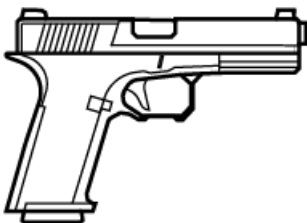
The pistol is a low caliber firearm with plenty of ammunition. It uses the basic features of the modular firearms.

The revolver is a high caliber, heavy hitting firearm. The trigger pull is quite heavy so firing can have a delay.
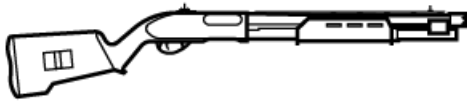
The shotgun is a pump action weapon with a conical spread. Shells are loaded into the weapon individually and the process can be aborted after the current round is inserted by pressing the fire button.

The assault rifle is a 5.56mm carbine with 3 fire modes: full auto, 3 round burst and semi-automatic.
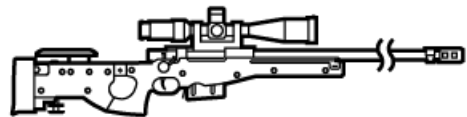
The sniper rifle is a heavy caliber bolt-action rifle with a long range scope.

The grenade launcher is a single-shot break open launcher that fires 40mm high explosive grenades.

The fragmentation grenade is a thrown weapon. Pull the pin and wait 5 seconds for an explosion.

More demo weapons will be added in future updates.

See Also

Melee Weapons

Modular Firearms

Overview
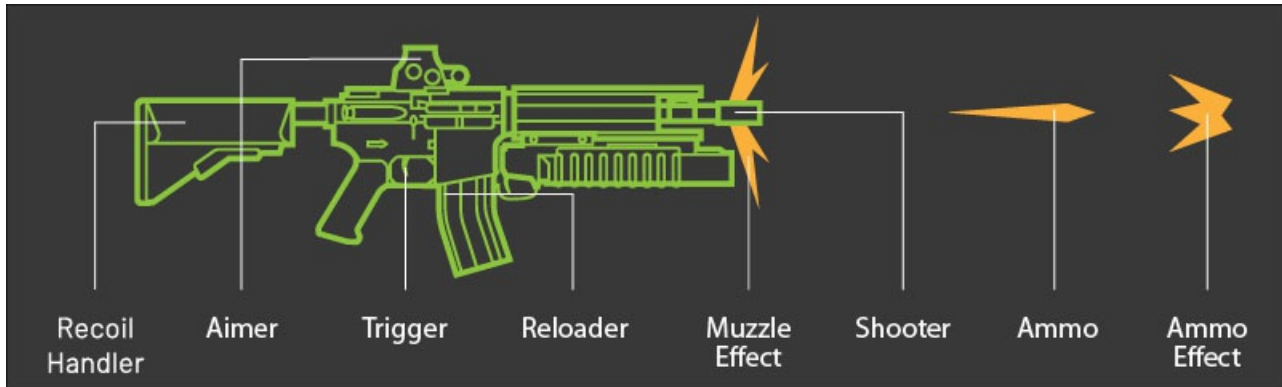
Gun style weapons in NeoFPS use a system of modules that work together with a central ModularFirearm behaviour to model the behaviour of a firearm. Multiple modules of each type can exist on a firearm at any time, but only one of each can be active. When another is enabled it automatically disables the previous. Modules can be attached to a child gameobject of the firearm, or on the same gameobject. This allows for weapon behaviour to be built on attachments that can be swapped in and out at runtime.

Creating a Modular Firearm

There are 2 ways to create a new modular firearm. The quickest and easiest is to use the Modular Firearm Wizard in the NeoFPS Hub. You can find out more about this on the NeoFPS Wizards page. The more flexible approach is to use the Modular Firearm component itself.

You can create a new modular firearm using the ModularFirearm behaviours setup controls in the inspector. Create a new GameObject in the scene and add the ModularFirearm behaviour to it. To start off with, the behaviour will be in quick-setup mode. This allows you to select a model to use, along with a few basic choices on input and inventory type. Once you have completed quick-setup you will see a "modules" section, which allows you to pick from the available firearm modules as listed below. Any errors in the setup of the modules will be shown in this section, and the module will be highlighted. For more information, see the ModularFirearm behaviour reference.

Modules

The modules that work together to model the firearm are as follows:

Shooters

Shooter modules are responsible for the actual gunshot. This could mean checking a raycast or spawning a projectile. The shooter is provided with a source position and direction, an accuracy value (0 to 1) and an ammo effect. NeoFPS comes with the following examples:

NAME	DESCRIPTION
HitscanShooter	The hitscan shooter uses a physics raycast to detect a hit.
BallisticShooter	The ballistic shooter spawns a BallisticProjectile object and passes the ammo effect to it to handle any impacts.
SimpleBallisticShooter	The simple ballistic shooter spawns a BallisticProjectile just like the ballistic shooter, but isn't affected by accuracy or the camera aim.
SpreadHitscanShooter	The spread shooter acts as a cone of hitscan shooters. It is used for weapons like shotguns.

NAME	DESCRIPTION
SpreadBallisticShooter	The ballistic shooter spawns multiple BallisticProjectile objects in a cone and passes the ammo effect to them to handle any impacts.
PatternHitscanShooter	The pattern hitscan shooter fires multiple hitscan shots in a preset pattern.
PatternBallisticShooter	The pattern ballistic shooter spawns multiple BallisticProjectile objects in a preset pattern.

The modular firearm shooters allow the choice of either hitscan or projectile versions. Hitscan shooters cast an instant ray from the gun barrel. This helps them feel very responsive, especially at close quarters. Projectile shooters spawn a projectile at the weapon's muzzle tip. These projectiles have full control of their movement and hit detection, and then pass the hit info back to the relevant ammo effect. Some example projectiles include the BallisticProjectile, BallisticProjectileWithSimpleDrag and BallisticProjectileWithParticles.

Triggers

Trigger modules handle the rate of fire of the firearm and react to input. NeoFPS comes with the following examples:

NAME	DESCRIPTION
AutomaticTrigger	The automatic trigger models a machinegun with a set rate of fire. It will keep firing as long as the trigger is held.
SemiAutoTrigger	The semi-auto trigger fires one shot for each press of the trigger. It can optionally repeat fire after a set time.
BurstFireTrigger	The burst fire trigger fires a set number of bullets at a set rate of fire. It has options to repeat after a set time and to allow / disallow cancelling the burst.
ChargedTrigger	The charged trigger must charge up before firing. If the trigger input is released the charge drops. Once the trigger has been charging for a set time the gun will fire.
QueuedTrigger	The queued trigger will queue up shots up to a maximum count while the trigger is held, and fire once it's released. You can optional cancel with the reload button.
TargetLockTrigger	The target lock trigger will lock onto any valid targets within the firearm's detection cone. Holding the trigger will build the lock. Releasing the trigger at full lock will fire. Releasing early or hitting reload will cancel the lock.
MultiTargetLockTrigger	The multi-target lock trigger is essentially a queued trigger that locks onto a unique target for each shot.

Ammo Effects

Ammo effect modules specify what happens when a shot actually connects with something. This includes the impact visuals and audio, dealing damage and adding impact forces. NeoFPS comes with the following examples:

NAME	DESCRIPTION
BulletAmmoEffect	The bullet ammo effect uses the Surfaces system to react to a hit.
AdvancedBulletAmmoEffect	The advance bullet ammo effect expands on the BulletAmmoEffect to add randomised damage and damage drop off over range.
PooledExplosionAmmoEffect	The pooled explosion ammo effect spawns an explosion at the impact point.
ParticleAmmoEffect	The particle ammo effect spawns a pooled ParticleImpactEffect particle system at the impact point.

NAME	DESCRIPTION
PenetratingHitscanAmmoEffect	Penetrating ammo effects allow shots to penetrate through surfaces up to a specific depth. This version performs a hitscan shot on the other side of the surface.
PenetratingProjectileAmmoEffect	Penetrating ammo effects allow shots to penetrate through surfaces up to a specific depth. This version spawns a new projectile on the other side of the surface.
RicochetHitscanAmmoEffect	Ricochet ammo effects allow shots to bounce off surfaces based on the hit angle. This version performs a hitscan shot from the point of impact.
RicochetProjectileAmmoEffect	Ricochet ammo effects allow shots to bounce off surfaces based on the hit angle. This version spawns a new projectile at the point of impact.
SurfaceBulletPhysicsAmmoEffect	The surface based bullet physics ammo effect gives detailed per-surface type control over a projectile bullet's penetration and ricochet behaviour. Surface info is specified in a SurfaceBulletPhysicsInfo asset.
TargetTrackingAmmoEffect	The target tracking ammo effect is a targeting system. The last object hit with this ammo effect will become a target for guided projectiles.

Ammo

The ammo module handles the ammo type of the weapon. This includes the total and current amount. NeoFPS comes with the following examples:

NAME	DESCRIPTION
SharedPoolAmmo	The shared ammo pool references an ammo inventory item. This allows ammo to be shared between multiple firearms.
CustomAmmo	Custom ammo is a self contained ammo component unique to the firearm it is attached to.
InfiniteAmmo	The infinite ammo module is actually both an ammo module and a reloader module. It completely removes the need to reload.
RechargingAmmo	The recharging ammo will build up its ammo pool up to a maximum over time.

Reloaders

Reloader modules act like the magazine attached to the firearm. It reloads using ammo from the attached ammo module. NeoFPS comes with the following examples:

NAME	DESCRIPTION
SimpleReloader	The simple reloader holds a set amount of ammo and reloads it all in one go.
ChamberedReloader	The chambered reloader uses a different delay and animation when reloading from empty vs with a round chambered.
IncrementalReloader	The incremental reloader is reloaded in chunks. It can be interrupted by firing and resumed later. It is used in the example shotgun to reload one shell at a time.
CustomRevolverReloader	This reloader is a variant of the simple reloader that displays the correct number of bullets in the revolver drum and swaps them for empties. It is used to demonstrate how to interact with animations to create complex effects.

NAME	DESCRIPTION
PassthroughReloader	The passthrough reloader does away with the magazine and fires bullets directly from the ammo module attached to the weapon.

You can also add the ReloaderCountdown behaviour to a firearm to play audio clips as the magazine count approaches zero. This can be useful for adding a warning sound that the magazine is almost empty, or for effects such as the iconic ping of a garand rifle.

Aimer

Aimer modules describe what happens when the player tries to aim down the weapon sights. A sniper rifle would hide the weapon geometry and show a UI scope. A pistol would raise the weapon to align its sights to the camera. NeoFPS comes with the following examples:

NAME	DESCRIPTION
InstantScopedAimer	Instantly hides the weapon geometry and shows a UI scope.
ScopedAimer	Raises the weapon to align with the camera over a set duration. Once the weapon is raised the its geometry is hidden and a UI scope is shown.
WeaponMoveAimer	Raises the weapon to align its sights with the camera.
HeadMoveAimer	Offsets the first person camera to align it to the weapon sights (allows tilt).
AnimOnlyAimer	Sets an animation bool parameter, but does not affect the camera or crosshair. It is mainly intended for using the modular firearm system with AI.

Recoil Handlers

Recoil handlers define how recoil affects the firearm object when it shoots. NeoFPS comes with the following examples:

NAME	DESCRIPTION
BetterSpringRecoilHandler	Uses the additive transform system to add procedural animation when the firearm recoils, along with modifying its accuracy. This new module is more intuitive and easier to control than the old SpringRecoilHandler module.
AccuracyOnlyRecoil Handler	Modifies the accuracy of the weapon with each shot, recovering over time.
SpringRecoilHandler	(Deprecated) Uses the additive transform system to add procedural animation when the firearm recoils, along with modifying its accuracy. This has been replaced by the BetterSpringRecoilHandler module.

Muzzle Effects

Muzzle effect modules display the muzzle flash of the weapon and plays gunshot audio. NeoFPS comes with the following examples:

NAME	DESCRIPTION
BasicGameObjectMuzzleEffect	Activates a game object for a brief period and then deactivates it again. Picks audio at random from an array of audio clips.

NAME	DESCRIPTION
SimpleParticleMuzzleEffect	Triggers a particle system to emit with each shot.
AdvancedParticleMuzzleEffect	Triggers one or more particle systems to emit with each shot. The particle systems are reparented to the character on start so they can persist between weapon switches, and to allow simulation in character space.
RandomObjectMuzzleEffect	Activates a game object chosen at random from a pool, and then deactivates it again after a brief delay. Picks audio at random from an array of audio clips.

Ejectors

Ejector modules optionally throw empty shells out of the weapon when it fires. NeoFPS comes with the following examples:

NAME	DESCRIPTION
StandardShellEject	Spawns a pooled shell object and throws it out from the weapon. Can be triggered instantly on firing, after a delay, or with an animation event.
ObjectSwapEjector	Swaps the specified object with a pooled physics object. Used to swap animated shell casings with physics based ones at a certain point in the weapon animation.
MultiObjectSwapEjector	As the object swap ejector, but can swap multiple items at once.
ParticleSystemShellEject	Triggers play on one or more particle systems whenever a shell is ejected.

Utilities

The modular firearms also come with various utilities to extend them with new features:

NAME	DESCRIPTION
AnimatedFirearmSprintHandler	Models firearm sprint behaviour using keyframed sprint animations on an Animator.
AttachedAmmoCounter	This is connected to a UI text element to display the firearm's magazine ammo on a world-space UI.
FirearmAimFatigue	Applies a stamina drain to the wielder's StaminaSystem when aiming down sights.
FirearmOverheat	Adds a barrel glow and haze effect on firing. You can also enable overheating, meaning that once the heat limit is reached, the firearm trigger will be blocked until it cools down enough. This can be paired with a HudFirearmOverheatBar to show the heat status in the player HUD.
FirearmTransformMatchSetter	Communicates with a TransformMatcher additive transform behaviour to sync the wielder's head animation to the firearm.
FirearmWieldableStanceManager	Used to specify procedural or keyframed weapon stances such as crouching or falling.
HolographicSight	Controls the colour and brightness of a holographic projection weapon optic.
LaserPointerAimerSwitch	Combined with a laser pointer, switches the aimer module while the laser is on.
ProceduralFirearmSprintHandler	Models firearm sprint behaviour using procedural animmation.
RecoilPushback	Add this to a firearm to apply a force directly away from the shot direction when firing.

NAME	DESCRIPTION
RenderTextureScope	Controls the camera, reticule and material for a render texture scope to add parallax and off-axis fade.

Animations

Animation of weapons is based in part on the ModularFirearm behaviour and also on the attached modules as well. As different modules behave in different ways they often need to be animated differently to match. For example, the semi-auto trigger uses an animator trigger property to tell the animator to play the trigger press and release animation. The full auto trigger uses a boolean property instead to tell the animator how long to hold the trigger, and the charged trigger uses a float property to tell the animator how much to blend the trigger pull / hammer cock animation.

Each animation key can be changed in the firearm or module's inspector and is set by default to match the example assets provided with NeoFPS.

In some cases, the animation can also define the timing for individual actions such as incrementing the magazine ammo count with the incremental reloader. NeoFPS comes with an example animation event handler called the FirearmAnimEventHandler which catches Unity animation events. It is very simple to implement your own animation event handlers and provides a great deal of flexibility beyond using timers, however in the example modules any potentially timed actions have an option to set a specific wait duration or to wait for an animation event before triggering.

Alongside keyframed animation, NeoFPS also allows you to add procedural animation to your weapons using Additive Transforms and Effects. These can be very simple, such as the spring effects driven by the recoil modules. They can also be more complicated, such as the procedural sprinting animation using the ProceduralFirearmSprintHandler. For larger scale movements, the main modular firearm behaviour has a pose system built in which allows for seamless transitions between poses. This is used by a number of modules such as the WeaponMoveAimer or the FirearmWieldableStanceManager.

Footstep driven animation such as sprinting or weapon bob is driven through a step tracking system attached to the character's motion controller. This ensures that any animations like this sync up, are only active while in the correct movement state, and are correctly driven by the character's velocity. The character's stride length (and therefore bob speed while moving) is controlled by adding a TrackSteps motion graph behaviour to the relevant states or sub-graphs in your character's motion graph.

*Note: If you have a weapon lower animation in your animator controller, then it is a good idea to transition to weapon raise at the end of it. The Unity animator can sometimes fail to transition if too many are triggered in very quick succession (eg tapping between 2 weapon slots), and this will prevent it from causing a problem.

Drops and Pickups

Firearms have their own "drop" system that allows a player or character to drop them from their hands. The firearms need their own system to account for ammo usage. When a firearm drop is spawned, the attached ammo is set to the amount in the firearm's magazine. In the sample assets the ammo can be picked up by walking over the weapon drop, while the firearm itself needs to be interacted with to pick back up. For more information, see ModularFirearmDrop and ModularFirearmAmmoPickup. The easiest way to create a firearm drop is using the PickupWizard in the NeoFPS hub.

Switching Modes

The NeoFPS firearms have a simple system for switching weapon modes. Mode switching is delegated to the ModularFirearmModeSwitcher behaviour, which should be attached to the GameObject with the ModularFirearm behaviour. This works by grouping firearm modules into weapon modes and then enabling them once the mode is switched. Since only one of each type of module can be enabled at any one time, the ModularFirearm behaviour will automatically disable the old modules when new ones are activated.

Some examples of things you can do with the firearm mode switcher include:

- Switching trigger groups to cycle between semi-auto, burst fire and full auto
- Switching ammo types in a grenade launcher
- Swapping aimer and trigger when switching a rifle from close combat to long range mode
- Switching muzzle effect (audio and visual) once a silencer is attached to the gun

Visual Effects

The modular firearms feature various options and modules for controlling their visual effects. The samples are set up with a low poly style that uses simple geometry and objects for muzzle effects and ejected shells. You can also use particle systems to create more realistic effects, and the asset ships with more realistic drop-in replacements for each of the demo firearms. These muzzle effect prefabs can be found at: *Assets\NeoFPS\Samples\Shared\Prefabs\Weapons\MuzzleFlashes*

To use the new muzzle effects, you will need to remove the Basic Game Object Muzzle Effect from the firearm prefab. Navigate through the weapon hierarchy until you find the **MuzzleFlash** game object. Delete this, and drag & drop the relevant realistic muzzle effect from the above folder into the same place in the hierarchy. Each of these realistic effects is ready set up with an Advanced Particle Muzzle Effect behaviour and positioned based on the main weapon mesh position.

Alongside the muzzle effects, NeoFPS also comes with a variety of hitscan bullet trail prefabs, and projectile prefabs. These can be found in the following folders: *Assets/NeoFPS/Samples/Shared/Prefabs/Weapons/Trails/* and *Assets/NeoFPS/Samples/Shared/Prefabs/Weapons/Projectiles/*

These are dropped onto the relevant property on the shooter module. For hitscan trails, you can set the size/radius and the duration of the trail (for example, the distortion bullet effect works best with a much larger size and longer duration than the additive trails).

Lastly, the NeoFPS samples include a number of shaders to achieve the above effects. These are currently only available for the standard pipeline and located at: *E:\Projects\neofps\unity\neofps_stable\Assets\NeoFPS\Samples\Shared\Effects\Shaders*

Each of the shaders has been created using the Amplify Shader Editor. If you own this asset, then you will be able to tweak them for your needs or use them as a starting point for your own shaders. Shaders that are intended for use with particle systems, such as the shockwave or fireball sheets make use of vertex colours to control intensity. A number of shaders also expose properties through material property blocks such as the glow amount on the glow shaders or highlight amount on the interactive object highlight shaders.

See Also

Hitscan vs Projectiles

One of the most important choices you can make when defining the feel of your guns is whether to use a hitscan or projectile based shooter.

Hitscan

Hitscan uses a raycast to detect what a shot hits instantaneously. The shooter module picks a direction for the shot based on the camera direction, the weapon direction, and the current accuracy of the firearm and draws a straight line from the muzzle tip to the first thing it hits.

The instantaneous nature of hitscan makes it well suited to fast paced, arcade style games where responsiveness is key. You don't need to lead your targets or compensate for bullet drop.

In NeoFPS, the visual aspects of hitscan shots are represented by the hitscan trail behaviours and a range of shaders. The hitscan trails available are:

- Line Renderer Hitscan Trail uses a Unity line renderer to draw a line from the muzzle tip to the hit point. The tracer size property on the shooter modules acts as a multiplier on the width set in the line renderer.
- Particle System Hitscan Trail emits a set number of particles per meter along the line from muzzle tip to hit point. |
- Line And Particle Hitscan Trail combines a line renderer and particle system in one. This is useful for effects like tracer using the line renderer, followed by smoke or dust from the particle system.
- Noisy Line Hitscan Trail uses a particle system with noise to drive the points in a line renderer. This creates an effect where the tracer starts as a straight line and then curls and spirals apart.

The following shaders are also available to create more interesting hitscan trail effects:

- The **NeoFPS_DissolveTrail** shaders start out solid and then dissolve away based on the tracer duration setting on the firearm's shooter module. There are additive and alpha shaded versions, along with edge blend versions which fade out the material at the outside edges of the trail.
- The **NeoFPS_DistortionTrail** adds a refraction style distortion effect similar to that scene in games like fear. This works especially well with a larger tracer size and longer duration. The shaders are available at *Assets\NeoFPS\Samples\Shared\Effects\Shaders*.

Projectiles

Projectile based shooters spawn a projectile at the firearm's muzzle tip with the specified velocity. A projectile travels over time until it hits something. They can also optionally be affected by gravity or even driven by complex movement logic. A large and slow projectile with gravity can be used to model grenade launchers and similar weapons, but you can also use much higher velocities to create standard bullets. For example, the muzzle velocity of a .45" pistol is in the region of 250m/s, a grenade launcher is more like 75m/s, while an assault rifle would be more like 750-900m/s. In games with smaller environments, you might want to keep your muzzle velocities smaller than real life to exagerrate the visuals and bullet drop.

A number of example projectiles can be found at *Assets\NeoFPS\Samples\Shared\Prefabs\Weapons\Projectiles*.

Projectiles are usually comprised of a root object with the projectile component attached, a mesh, which will be enabled after the projectile has travelled a set distance, and a trail renderer. Projectiles are implemented as a PooledObject and will be returned to the pool on impact. They can also have a NeoSerializedGameObject component attached, allowing the NeoFPS save system to save and load projectiles in flight.

Guided projectiles
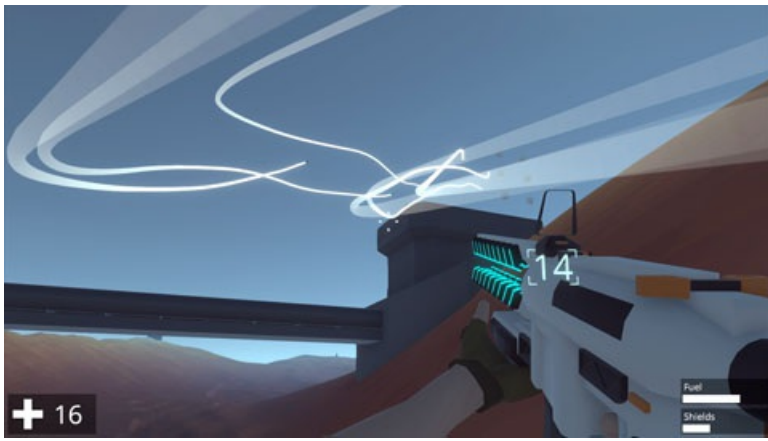
NeoFPS has a system that lets you build guided projectiles from a combination of a tracker component and a motor component. The tracker component is responsible for detecting and choosing a target, while the motor component is responsible for flying towards it. Currently there are 2 motor components available:

- The Simple Steering Motor simply turns towards the target each frame with a maximum steering rate that increases over time (this prevents the projectile from orbiting targets).
- The Drunk Missile Motor has a much more erratic flight path. At brief intervals it picks a random direction (tending towards the target) and turns towards that. Once the projectile gets close enough to the target it adopts a more standard steering system. The projectile can also boost its speed at random intervals as well. Together these give an effect similar to the missiles seen in anime.

There are a range of tracking options for guided projectiles in NeoFPS:

- The NearestObjectWithTagTracker component checks for colliders with a specific layer and tag in its vicinity. It will target the closest valid target.
- The PlayerTracker component homes in on the player character.
- The TargetingSystemTracker component defers its target selection to a separate targeting system. There are multiple targeting systems available, which are attached to the firearm. The ballistic shooter firearm modules will check for a targeting system on the firearm, as well as a TargetingSystemTracker on the projectile, and register the projectile with the targeting system if both are found. Targeting system options available are:
- LaserTargetingSystem which is attached to a WieldableLaserPointer. If the laser is on, the projectile will home in on the point the laser hits.
- RaycastTargetingSystem casts a ray from either the weapon or the camera, and passes the hit point to the projectiles. This can be a one-shot or continuous cast.
- TargetLockTrigger is a firearm trigger that checks for valid targets within a cone in front of the weapon. Holding the trigger will lock onto the target over time. Releasing the trigger before the lock completes will cancel the shot, while releasing the trigger after will fire the gun and pass the locked target to the projectile. The target is tracked continuously from that point until the projectile hits or is destroyed. You can cancel firing with the reload button.
- MultiTargetLockTrigger is a firearm trigger which locks onto multiple targets one by one as you hold the trigger. Releasing the trigger will fire as many shots as there were target locks in a burst. You can cancel firing with the reload button.
- TargetTrackingAmmoEffect allows you to tag a target with a shot, which the guided projectiles will then home in on. This can track moving targets, and will track the exact point on the target that the bullet hit. You can use this in conjunction with a the ModularFirearmModeSwitcher to allow you to switch weapon modes between tagging and missiles.

See Also

Modular Firearms

Scopes & Optics

Overview

NeoFPS provides a number of ways to implement different scopes and optics for your weapons. These include close range and long range optics in different styles.

Iron Sights

Iron sights are the simplest form of ADS optics. You simply need to make sure that your iron sights geometry is a sensible shape and layout, and then use one of the firearm aimer modules to align the weapon to the camera when aiming.

Holographic Sights and Red-Dots

Holographic and red dot sights project a reticule at a distance in front of the weapon that is only visible through the sight. This makes it easy to quickly acquire a target and adjust to the movement of the weapon during recoil.

You can increase or decrease the brightness of the reticule using the **Optics Brightness +/-** input buttons in game.

For more information see the HolographicSight Reference. The Demo Facility assault rifle uses a holographic sight by default.

Render Texture Scopes

Render texture based scopes are used in games such as Escape From Tarkov that emphasise realism. They provide a zoomed in view within the scope lens, without zooming in the rest of the frame. This means that the player can maintain awareness of their surroundings much easier than they can with a HUD or stencil based scope where their whole view is zoomed in.

The NeoFPS RT scopes also have a parallax effect when moving. This involves moving the reticule and a blurred scope ring as the scope goes off the player camera's axis. It can also rotate the scope camera slightly to compensate for this off axis viewing so that the scope image feels more dynamic.

When the render texture scope gets far enough from the camera axis it starts to fade out to an opaque, reflective material. Once fully opaque, the scope camera will stop rendering.

Scopes implemented in this way incur a performance penalty since they involve rendering the scene twice. This can be mitigated slightly by showing the scope smaller on the screen (and therefore using a smaller render texture), but you should be aware that the increased realism comes at a cost.

For more information see the RenderTextureScope Reference. The Demo Facility sniper rifle uses a render texture scope by default.

Stencil Based Scopes

Stencil based scopes use stencil shaders to define what parts of the weapon are visible through the lens of the scope. They provide no zoom themselves, so the entire player camera must be zoomed in. The stencil shaders are used to cut out the scope housing and mount behind the lens so that your view is unobstructed. They are also used in the opposite way to display the scope reticle and inner tube, providing a slight parallax effect, while cutting out any of the geometry that extends outside your view of the lens.

Stencil based scopes are much more performant than render texture scopes, with the trade off that the view around the scope is also zoomed in.

The Demo Facility sniper rifle has stencil based scopes disabled in its hierarchy.

Note: The stencil based scopes are currently not working with deferred rendering. A solution is being looked into.

HUD Scopes

HUD based scopes essentially hide all weapon geometry, and overlay a crosshair over the entire screen. This can be instant, or triggered at the end points of an aim-down-sights animation.

These scopes tend to have a much more arcadey / responsive feel than the other options, though it can be harder to make them look good.

You should also make sure that the shooter module on the firearm is set to use the camera aim when aiming so that the bullet

hits where the crosshair is pointing every time.

The sniper rifle in the feature demos uses a HUD based scope.

See Also

RenderTextureScope Reference

HolographicSight Reference

Firearm Attachments

Overview

The firearm attachment system is still in development and not yet ready for use, but the NeoFPS demo assets include a number of attachment models and prefabs. Besides the various optics attachments, there are also currently flashlights and laser pointers

Flashlights

The flashlights just use a spotlight object which is toggled on or off with the **Flashlight** input button. There is also an API for setting the flashlight's brightness.

For more information, see the WieldableFlashlight Reference. The Demo Facility pistol and shotgun both use a wieldable flashlight attachment by default.

Laser Pointers

The laser pointer also uses the flashlight system to toggle on or off using the **Flashlight** input button. It uses a line renderer with a custom shader for the beam and a generated billboard quad for the flare at the point of impact.

Using laser pointers does have some gameplay implications. By default, the NeoFPS firearms use an accuracy system for determining where shots hit, and also use the camera / HUD crosshair to determine the direction to fire. As accuracy decreases due to recoil, movement or other modifiers, the bullets spread out from the aim point. This can feel wrong when you have a laser pointer attached to the weapon which shows you where each of your bullets *should* hit. You can help with this by changing some of the default settings on the weapon:

- Set the shooter modules to **never** use the camera aim. This means that the weapon will always fire directly in front of it.
- Reduced the max spread / max aim offset to a very low number.

These 2 changes will mean that bullets should always hit where the laser is pointing, though it also means that your weapons are now incredibly accurate. You can mitigate this by emphasising the procedural animation systems and keyframed animation on the model. For example:

- Add a WeaponAimAmplifier additive transform effect to rotate the weapon as you move your view
- Use a BreathingEffect additive transform effect along with either a SimpleBreathHandler or StaminaSystem on the player character to add breathing motion.
- Add a CameraShake additive transform effect to the weapon itself with a subtle amount of twist / rotation shake (no position shake).
- Add a RotationBob additive transform effect to the weapon to layer in rotation as you move instead of just the default position bob.
- Use the **wander** settings on the weapon's recoil handler module to increase the "spread" of the weapon when firing rapidly.

You can use the **On Toggle On** and **On Toggle Off** events on the laser pointer component to enable/disable the above components, or to change their settings while the laser is switched on.

Another component that can be used to modify the feel of a weapon with a laser pointer is the LaserPointerAimerSwitch. This component works by detecting the active aimer module on the firearm at start, and then switching between that and another aimer module that you set as the laser pointer aimer. This means that you can switch to a canted pose that obstructs less of your view with the laser on, and switch back to optics when it's off.

For more information, see the WieldableLaserPointer Reference. The Demo Facility assault rifle uses a laser pointer by default.

You can check this out for an example of how to use the laser pointer events to modify the weapon's feel when toggling the laser on or off.

See Also

Modular Firearms

Scopes & Optics

Additive Transforms & Effects

Melee Weapons

Overview

Melee weapons in NeoFPS are a type of FpsInventoryWieldable. They inflict damage and impact force directly ahead when the animation is at the striking point.

The NeoFPS melee system will be expanded with more options in future updates.

Weapon Pickups

You can create a pickup object for your melee weapon using the Pickup Wizard in the NeoFPS Hub. This allows you to pick the weapon up in the scene, but also to drop it from your hands and pick it up again later.

Melee Animations
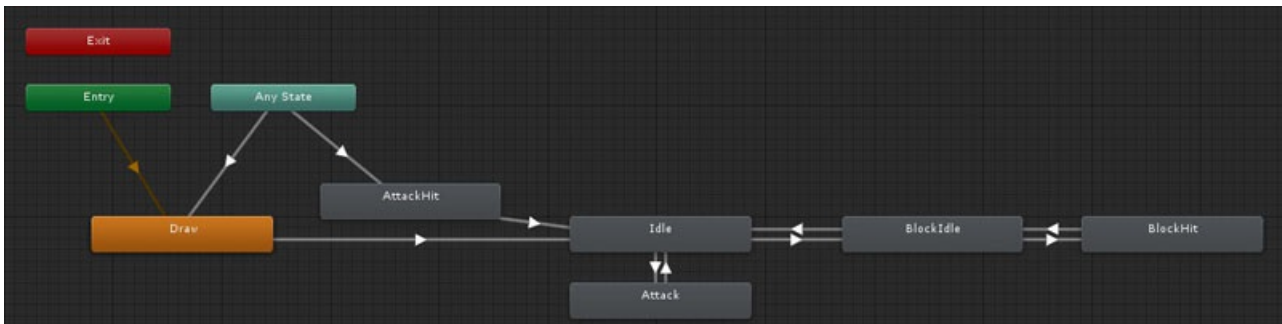
The melee weapon animations are relatively simple. See the sample assets for details on how the graph is set up.

The relevant animator controller properties are:

NAME	TYPE	DESCRIPTION
Draw	Trigger	Used when the weapon is drawn to raise it from off the screen. This is also the entry state of the controller.
Attack	Trigger	Signals the start of an attack.
Attack Hit	Trigger	Signals that the attack connected. This is used to trigger a bounce back reaction animation and will interrupt the attack animation that would otherwise follow through.
Block	Boolean	Used to specify when the weapon should be raised in a blocking state.
Block Hit	Trigger	A hit reaction knock while blocking.

The property names can be changed in the MeleeWeapon behaviour.

Alongside keyframed animation, NeoFPS also allows you to add procedural animation to your weapons using Additive Transforms and Effects. These can be very simple, such as the weapon and head bob effects. They can also be more complicated, such as the procedural sprinting animation using the ProceduralMeleeSprintHandler. For larger scale movements, the main melee weapon behaviour has a pose system built in which allows for seamless transitions between poses and is used by the sprinting system along with the MeleeWieldableStanceManager.

Adding Custom Melee Behaviour

If you have custom behaviour that you want to be applied when you use a melee weapon, the easiest way is to add a "melee hit extension" component. This is simply a Monobehaviour that implements the `IMeleeHitExtension` interface. This interface has a single `OnMeleeHit(RaycastHit hit)` method. Any extension components that are found on the melee weapon's root object on start will be triggered every time you hit an object with the weapon.

If you actually want to change the hit behaviour, you can also create your own melee weapon class that inherits from the `MeleeWeapon` class. You can override the virtual `OnMeleeHit(RaycastHit hit, Vector3 attackDirection)` method here to either add or replace the functionality.

For more complex behaviour, you can also use the `BaseMeleeWeapon` class as a base for your own. This removes all the hit and block functionality, and exposes the following methods

```
public abstract void PrimaryPress();
public abstract void PrimaryRelease();
public abstract void SecondaryPress();
public abstract void SecondaryRelease();
```

See Also

MeleeWeapon

Unity AnimatorController

Unity State Machine Behaviours

Thrown Weapons

Overview

Thrown weapons in NeoFPS are a type of FpsInventoryWieldable. They spawn a PooledObject at a specified point with a set velocity. They can be stacked so that the character can carry a number of them. Each throw the stack is reduced until all have been used, at which point the item is removed from the character inventory.

Thrown weapons have a strong throw and a weak thrown. They can also inherit velocity from the character as it moves in a similar style to games like Counter Strike.

The NeoFPS thrown weapons system will be expanded with more options in future updates.

Weapon Pickups

You can create a pickup object for your thrown weapon using the Pickup Wizard in the NeoFPS Hub. This allows you to pick the weapon up in the scene, but also to drop it from your hands and pick it up again later.

Thrown Weapon Animations
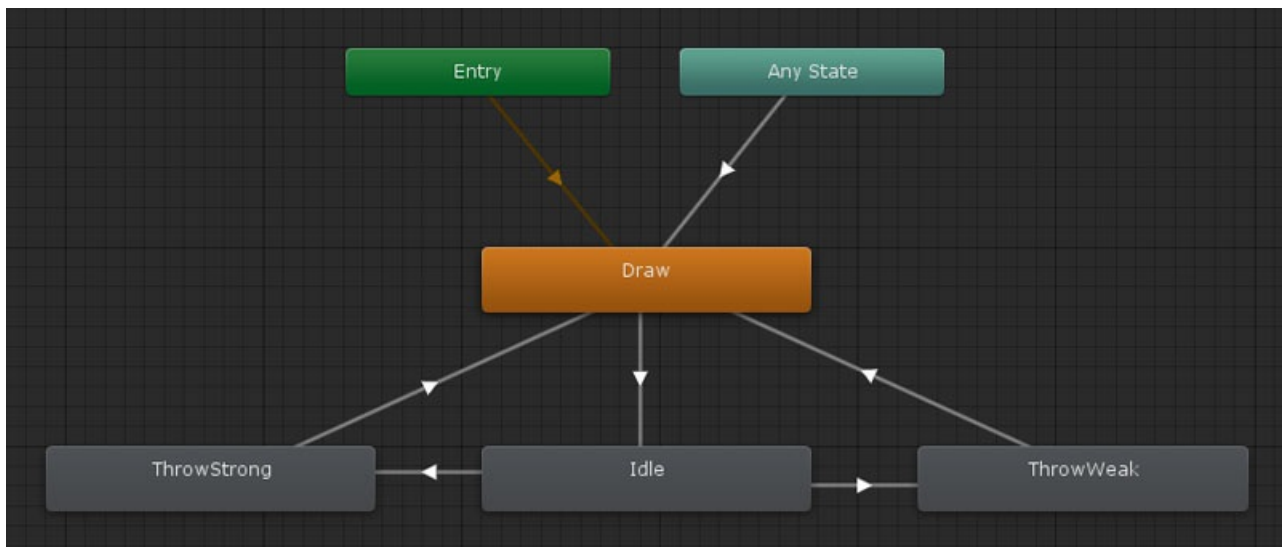
The thrown weapon animations are relatively simple. See the sample assets for details on how the graph is set up.

The relevant animator controller properties are:

NAME	TYPE	DESCRIPTION
Draw	Trigger	Used when the weapon is drawn to raise it from off the screen. This is also the entry state of the controller.
Throw Light	Trigger	Signals the start of a weak throw attack.
Throw Heavy	Trigger	Signals the start of a strong throw attack.

The property names can be changed in the ThrownWeapon behaviour.

Alongside keyframed animation, NeoFPS also allows you to add procedural animation to your weapons using Additive Transforms and Effects. These can be very simple, such as the weapon and head bob effects. They can also be more complicated, such as the procedural sprinting animation using the ProceduralThrownSprintHandler. For larger scale movements, the main thrown weapon behaviour has a pose system built in which allows for seamless transitions between poses and is used by the sprinting system along with the ThrownWieldableStanceManager.

See Also

ThrownWeapon

Unity AnimatorController

Unity State Machine Behaviours

Wieldable Tools

Overview

The wieldable tools feature is a new **early access** feature that allows you to create wieldable items to perform a range of actions that don't fit into the categories of firearm, melee or thrown weapon. Some examples are healing stim packs, shield boosters and grapple hooks.

Each wieldable tool has a primary and secondary fire that you assign actions and tool modules to through the inspector. Each of these actions and modules is a separate MonoBehaviour component that is added to the object, similarly to the modular firearm modules.

You can find a pair of example tool prefabs in the following folder: *NeoFPS/Samples/Shared/Prefabs/Tools/*

The flashlight tool can be toggled via the primary fire button or the flashlight button. The grapple tool requires that the character you use it with have a vector parameter on their motion graph that the tool can assign the grapple point to. The parkour demo motion graph is set up with this. Note that the demo tools use the quick-switch inventory type and have specific quick slot indices that might clash with existing demo weapons. You will need to remove those weapons from the inventory before adding these tools in order for them to work (in the demos these are the baton and pistol).

Note: The wieldable tools system does not currently come with any art assets to demo it, but art and animations work in the same way as for melee or thrown weapons. There is also no wizard currently set up to create wieldable tools through the NeoFPS Hub. Once enough feedback has been provided and the systems finalised, the wizard will be added.

Actions And The WieldableTool Component
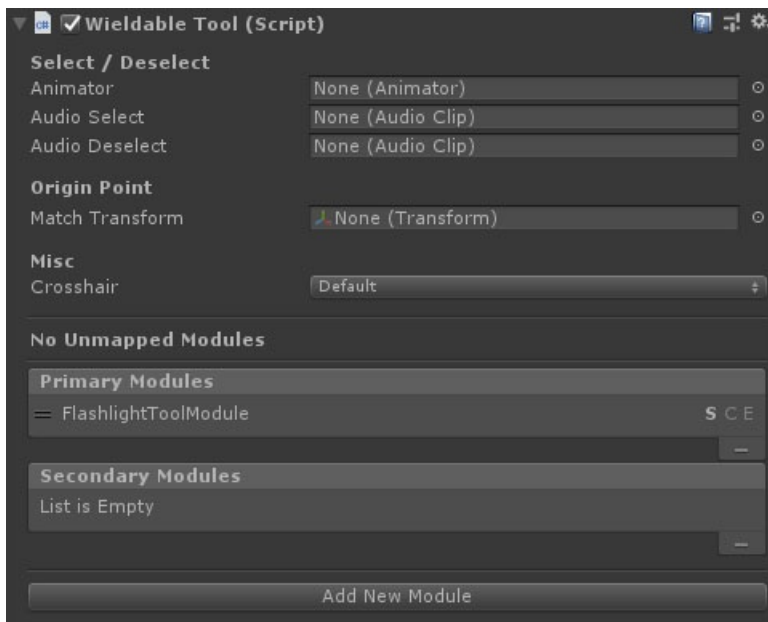
The WieldableTool component checks for new modules and actions on the object and displays them in the **Unmapped Modules** section. You can then choose if they should be assigned to the primary or secondary fire for the tool (or removed from the object).

Each module has three letters next to it: **S, C, E**. These represent when the actions fire: start, continuous and end. Start actions fire as soon as the relevant fire button is pressed. Continuous actions fire each fixed update frame as long as the fire button is held. End actions fire as soon as the fire button is released, or when the firing is interrupted. These end actions can often be set to only fire on "success", which is defined by the other actions. For example, the ChargeToolAction uses a continuous action to build up a charge. If the fire button is released before the charge reaches 100%, then the end actions are called with a success value of *false*. If the charge reaches 100%, then the primary/secondary fire is interrupted and the end actions are called with a success value of *true*. An example use for this would be recreating the shield boosters from Apex Legends. You would use a ChargeToolAction along with a ShieldBoosterToolAction set to apply the shield boost as an end action that requires success.

A number of modules or actions allow you to specify whether they fire on start or end (including success / fail), or continuously.

If a tool has no continuous actions then the end actions will fire immediately after the start actions.

The currently available wieldable tool modules are:

NAME	DESCRIPTION
AddInventoryItemToolAction	Adds an inventory item on start and/or end, or in increments as the relevant trigger is held.
AnimatorBoolToolAction	Sets a bool parameter on the tool's animator controller as long as the trigger is held.
AnimatorTriggerToolAction	Sets a trigger parameter on the tool's animator controller as a start and/or end action.
ChargeToolAction	Builds a charge as a continuous action, setting the end actions' success parameter to true if full charge is reached and false if not. Charging can alse be set up to play a looping sound with a pitch shift as charge builds, and play animations based on charge level.
BlinkToolModule	Used to create an Arkane style blink ability. The tool checks for a valid blink target and sends it to a motion graph parameter on releasing the trigger. Used with a MoveToPointState motion graph state.
ConsumeInventoryItemToolAction	Consumes an inventory item on start and/or end, or in increments as the relevant trigger is held.
FlashlightToolModule	A flashlight that can be toggled on or off.
GrappleToolModule	Casts a ray based on the wielder's aim, and uses it to set a motion graph parameter for the hit point. Releasing the trigger resets the motion graph parameter. When paired with the GrappleSwingState motion graph state, this allows for grappling and swinging around scenes.
HealToolAction	Can be set to heal a set amount instantly on start and/or end, or to heal in increments continuously as long as the trigger is held.
PlayAudioToolAction	Plays a specific sound on start and/or end.
ShieldBoosterToolAction	Restores shield steps on start and/or end.
UnityEventToolAction	Fires a unity event on start and/or end that you can use to call methods on other components.

Wieldable Tool Pickups

You can create a pickup object for your wieldable tool using the Pickup Wizard in the NeoFPS Hub. This allows you to pick the weapon up in the scene, but also to drop it from your hands and pick it up again later. This is done in the same way as for a melee or thrown weapon pickup (you can use the melee baton pickup template as a starting point), by selecting the "Wieldable Item Drop" pickup type.

Wieldable Tool Animations

Wieldable tool animations are generally dictated by the modules that are attached, though the raise and lower animations are specified on the wieldable tool component and behave the same as for all other wieldables (firearms, melee and thrown weapons).

The [AnimatorTriggerToolAction][tools-anim] can be used to add animations at the start and/or end actions for each fire mode by specifying trigger parameters in the tool's animator controller.

You can also add keyframed sprinting animations using the AnimatedToolSprintHandler

Alongside keyframed animation, NeoFPS also allows you to add procedural animation to your weapons using Additive Transforms and Effects. These can be very simple, such as the weapon and head bob effects. They can also be more complicated, such as the procedural sprinting animation using the ProceduralToolSprintHandler. For larger scale movements, the wieldable tool behaviour has a pose system built in which allows for seamless transitions between poses and is used by the sprinting system along with the WieldableToolStanceManager.

See Also

WieldableTool

Unity AnimatorController

Explosions

Overview

NeoFPS has a simple system for spawning explosions that deal damage and repel physics objects based on their distance from the explosion center.

The sample explosions are built from a series of particle effects, though you can turn any object into an explosion by adding a PooledExplosion behaviour.

An explosion can be assigned a source and a damage filter for complex damage effects. For more information, see Health and Damage.

To spawn an explosion from a script, first you need to grab an explosion from the pool manager:

```
var explosion = GetPooledObject<PooledExplosion> (prototype, position, rotation);
```

To apply damage, call the `Explode()` method on the pooled explosion:

```
explosion.Explode(damage, maxForce, sourceController, transformToIgnore);
```

There are a number of scripts already available to work with explosions, such as:

NAME	DESCRIPTION
PooledExplosionSpawner	Simply spawns an explosion on command. Connect it to events to control execution.
ExplosiveObject	Represents a destructible object that spawns an explosion when killed.
PooledExplosionAmmoEffect	A firearm module that spawns an explosion when shooting something.

Explosive Barrels

Explosive barrels can be implemented using the above ExplosiveObject behaviour. This allows you to set a health amount for the barrel, specify a PooledExplosion to spawn once health reaches zero, and specify explosion damage and forces. A Unity event is also provided which you can use for actions like spawning debris objects.

An example can be found in the Demo Facility sample scene, and is available as a prefab at *Assets\NeoFPS\Samples\Shared\Prefabs\Props\Prop_Barrel01.prefab*.

See Also

PooledExplosion

Health and Damage

Pooling

Unity Particle System Explosions

AccuracyOnlyRecoilHandler MonoBehaviour

Overview

The AccuracyOnlyRecoilHandler module modifies the accuracy of the weapon with each shot.

Inspector

Properties

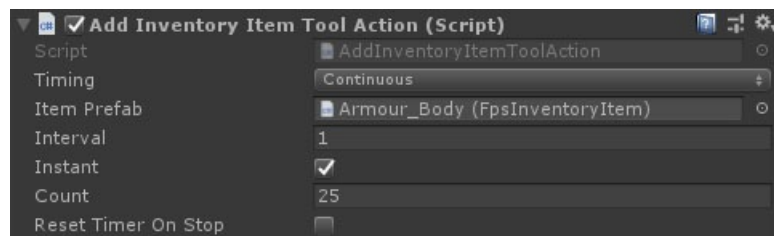| NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active recoil handler immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: **Component** means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. **GameObject** activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
On Recoil	UnityEvent	An event that fires every time the weapon recoils.
Hip Accuracy Kick	Float	The accuracy decrement per shot in hip fire mode (accuracy has a 0-1 range).
Hip Accuracy Recover	Float	The accuracy recovered per second in hip fire mode (accuracy has a 0-1 range).
Sighted Accuracy Kick	Float	The accuracy decrement per shot in sighted fire mode (accuracy has a 0-1 range).
Sighted Accuracy Recover	Float	The accuracy recovered per second in sighted fire mode (accuracy has a 0-1 range).

See Also

Modular Firearms

AddInventoryItemToolAction MonoBehaviour

Overview

The AddInventoryItemToolAction behaviour is used to add inventory items to the tool wielder's inventory when used. This is useful for things like repairing armour.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Timing	Dropdown	When should the item be consumed.
Item Prefab	FpsInventoryItem	An inventory item prefab to add to the character inventory if none is found.
Interval	Integer	The items should be added every nth fixed update frame in continuous mode.
Instant	Boolean	Should the item be added on the first frame of the continuous action, or should it wait for the first interval to elapse.
Count	Integer	The number of times to add the item.
Reset Timer On Stop	Boolean	Should the countdown between adding items be reset when you stop using the item.

See Also

Wieldable Tools
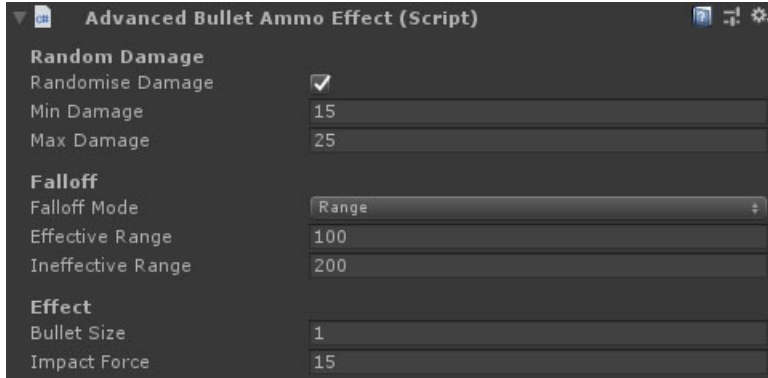
Inventory

AdvancedBulletAmmoEffect MonoBehaviour

Overview

The AdvancedBulletAmmoEffect module uses the surfaces system to show impact effects, and imparts damage and force to whatever it hits. It is similar to the [BulletAmmoEffect][4] but adds damage falloff over range and damage randomisation.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Damage Type	DamageType	The type of damage the weapon should do with this ammo.
Randomise Damage	Boolean	Should the damage value be randomised or not.
Damage	Float	The damage the bullet does before falloff is applied. This property is only visible if **Randomise Damage** is set to **False**.
Min Damage	Float	The minimum damage the bullet does before falloff is applied. This property is only visible if **Randomise Damage** is set to **True**.
Max Damage	Float	The maximum damage the bullet does before falloff is applied. This property is only visible if **Randomise Damage** is set to **True**.
Falloff Mode	Dropdown	How to apply damage falloff. **None** means damage will not fall off over distance or speed, **Range** means that damage will be reduced the further the target, **Speed** is used with projectiles to reduce damage as they slow down.
Effective Range	Float	The max range where the bullet does the full damage (no falloff applied). This property is only visible if **Falloff Mode** is set to **Range**.
Ineffective Range	Float	The range where the bullet does 0 damage. This property is only visible if **Falloff Mode** is set to **Range**.
Effective Speed	Float	The speed above which the bullet does full damage. This property is only visible if **Falloff Mode** is set to **Speed**.
Minimum Speed	Float	The speed below which the bullet does zero damage. This property is only visible if **Falloff Mode** is set to **Speed**.
Bullet Size	Float	The size of the bullet. Used to size decals.

NAME	TYPE	DESCRIPTION
Impact Force	Float	The force to be imparted onto the hit object before falloff is applied. Requires either a Rigidbody or an impact handler.

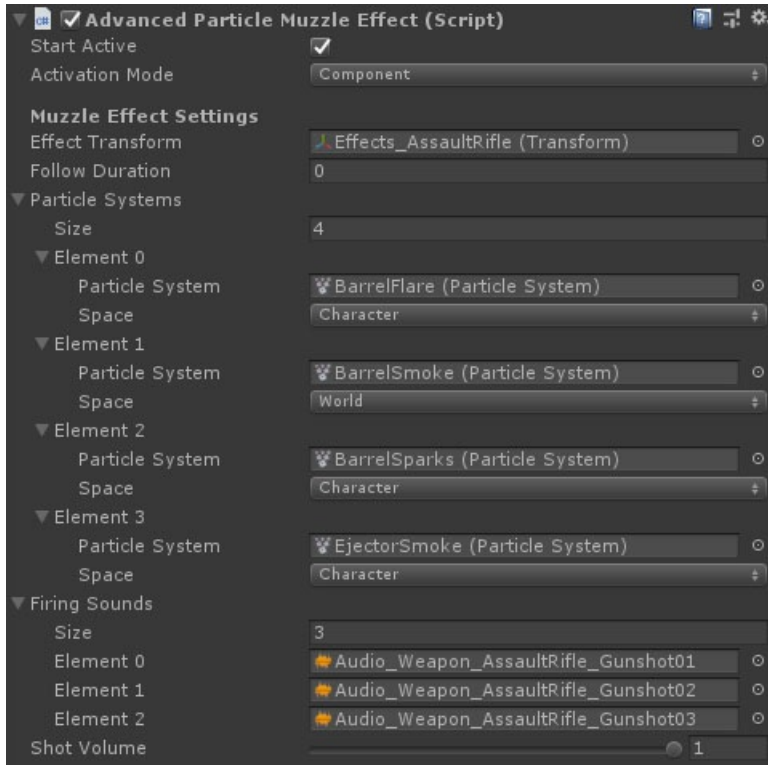# See Also

Modular Firearms

Surfaces

AdvancedParticleMuzzleEffect MonoBehaviour

Overview

The ParticleMuzzleEffect behaviour uses one or more particle systems which are played for each shot. On start, the systems are moved from the weapon to a specific transform in the character hierarchy so that the particles can persist when weapons are switched. This also allows you to simulate the particles in the character's coordinate space.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active muzzle effect immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: **Component** means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. **GameObject** activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Effect Transform	Transform	The root of the muzzle effect hierarchy.
Particle Systems	Particle System Info Array	The particle system to play.
Firing Sounds	AudioClip Array	The audio clips to use when firing. Chosen at random.

Particle System Info

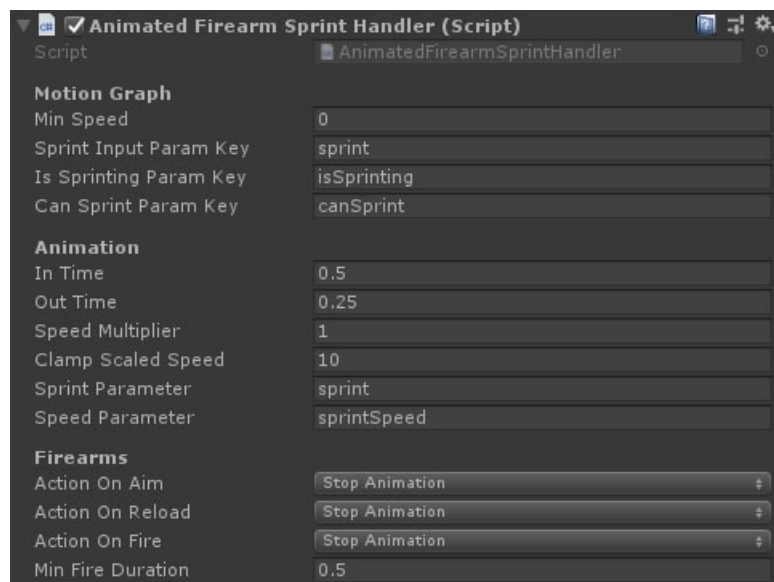| NAME	TYPE	DESCRIPTION
Particle System	ParticleSystem	The particle system to play.
Space	Dropdown	The simulation space for the particles. **World** means that emitted particles move freely in the world, **Weapon** means that particles move relative to the weapon (if it moves, they move with it), **Character** moves with the character (handles rapid character movement better than world space), **NoChange** leaves the simulation space as defined in the particle system

See Also

Modular Firearms

AnimatedFirearmSprintHandler MonoBehaviour

Overview

The AnimatedFirearmSprintHandler behaviour is attached to a modular firearm to model different sprinting mechanics. This version allows you to use keyframed sprint animations on your weapons by setting a bool parameter on an Animator component when the sprint animation should play, and a float parameter to tell the animator the speed. You can use this to set the clip playback speed in the AnimatorController, or to drive a blend between multiple animation clips based on speed.

Inspector

Properties

Motion Graph

NAME	TYPE	DESCRIPTION
Min Speed	Float	The minimum speed the character must be moving for the sprint animation to play.
Sprint Input Param Key	String	(Required) The switch parameter on the motion graph which is set by the input handler to tell the character when to sprint.
Is Sprinting Param Key	String	(Required) A switch parameter on the motion graph which the graph sets when the character is sprinting.
Can Sprint Param Key	String	(Optional) A switch parameter on the motion graph which tells the character if it can sprint or not.

Animation

NAME	TYPE	DESCRIPTION
In Time	Float	The time taken to blend into the sprint animation.
Out Time	Float	The time taken to blend out of the sprint animation to idle.
Unscaled Sprint Move Speed	Float	The movement speed that the sprint animations are synced to when playing at 1x speed..

NAME	TYPE	DESCRIPTION
Max Speed	Float	A maximum speed clamp for the character when used to calculate the animation speed multiplier.
Blend Zero Speed	Float	The speed below which the light sprint animation will be 100% used. Above this, the heavy animation is blended in.
Blend Full Speed	Float	The speed above which the he\vy sprint animation will be 100% used. Below this, the light animation is blended in.
Sprint Bool Parameter	String	A bool parameter on the Animator to signify when the weapon enters or exits sprint.
Speed Float Parameter	String	A float parameter on the Animator to set the playback speed of the sprint animation.
Blend Float Parameter	String	A float parameter on the Animator used to blend between the light and heavy sprint animations.

Firearms

NAME	TYPE	DESCRIPTION
Action On Aim	Dropdown	What to do when the firearm enters / exits ADS. **Stop Sprinting** completely stops the character from sprinting until they leave ADS. **Stop Animation** stops the firearm sprint animation without affecting the character movement.
Action On Reload	Dropdown	What to do when the firearm is reloaded. **Stop Sprinting** completely stops the character from sprinting until the reload animation completes. **Stop Animation** stops the firearm sprint animation without affecting the character movement.
Action On Fire	Dropdown	What to do when the firearm trigger is pulled while sprinting. **Stop Sprinting** completely stops the character from sprinting until the weapon stops firing. **Stop Animation** stops the firearm sprint animation without affecting the character movement (both of these have a slight delay before firing to allow the weapon to be aligned). **Cannot Fire** means that the firearm trigger is blocked until the character stops sprinting.
Min Fire Duration	Float	The minimum amount of time the firearm sprint animation will be paused or sprinting blocked when the trigger is pulled. Prevents rapid tapping of the trigger popping in and out of sprint.

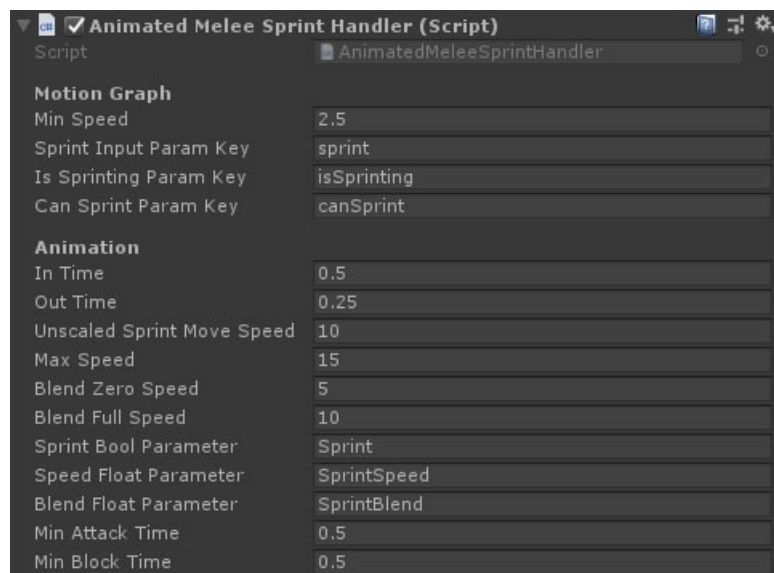# See Also

Modular Firearms

The Motion Graph

Unity Animator

Unity AnimatorController

AnimatedMeleeSprintHandler MonoBehaviour

Overview

The AnimatedMeleeSprintHandler behaviour is attached to a melee weapon to model different sprinting mechanics. This version allows you to use keyframed sprint animations on your weapons by setting a bool parameter on an Animator component when the sprint animation should play, and a float parameter to tell the animator the speed. You can use this to set the clip playback speed in the AnimatorController, or to drive a blend between multiple animation clips based on speed.

Inspector

Properties

Motion Graph

NAME	TYPE	DESCRIPTION
Min Speed	Float	The minimum speed the character must be moving for the sprint animation to play.
Sprint Input Param Key	String	(Required) The switch parameter on the motion graph which is set by the input handler to tell the character when to sprint.
Is Sprinting Param Key	String	(Required) A switch parameter on the motion graph which the graph sets when the character is sprinting.
Can Sprint Param Key	String	(Optional) A switch parameter on the motion graph which tells the character if it can sprint or not.

Animation

NAME	TYPE	DESCRIPTION
In Time	Float	The time taken to blend into the sprint animation.
Out Time	Float	The time taken to blend out of the sprint animation to idle.
Unscaled Sprint Move Speed	Float	The movement speed that the sprint animations are synced to when playing at 1x speed..

NAME	TYPE	DESCRIPTION
Max Speed	Float	A maximum speed clamp for the character when used to calculate the animation speed multiplier.
Blend Zero Speed	Float	The speed below which the light sprint animation will be 100% used. Above this, the heavy animation is blended in.
Blend Full Speed	Float	The speed above which the he\vy sprint animation will be 100% used. Below this, the light animation is blended in.
Sprint Bool Parameter	String	A bool parameter on the Animator to signify when the weapon enters or exits sprint.
Speed Float Parameter	String	A float parameter on the Animator to set the playback speed of the sprint animation.
Blend Float Parameter	String	A float parameter on the Animator used to blend between the light and heavy sprint animations.

Melee Weapon

NAME	TYPE	DESCRIPTION
Min Attack Time	Float	The minimum amount of time the sprint animation will be paused after an attack.
Min Block Time	Float	The minimum amount of time the sprint animation will be paused when blocking (prevents tapping block).

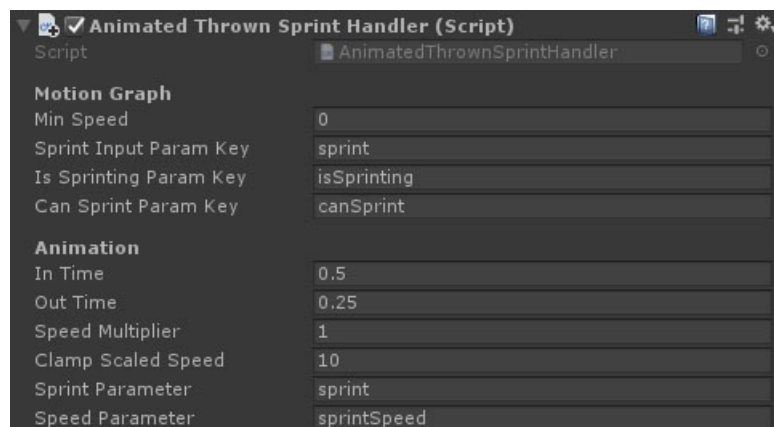## See Also

Melee Weapons

The Motion Graph

Unity Animator

Unity AnimatorController

AnimatedThrownSprintHandler MonoBehaviour

Overview

The AnimatedThrownSprintHandler behaviour is attached to a thrown weapon to model different sprinting mechanics. This version allows you to use keyframed sprint animations on your weapons by setting a bool parameter on an Animator component when the sprint animation should play, and a float parameter to tell the animator the speed. You can use this to set the clip playback speed in the AnimatorController, or to drive a blend between multiple animation clips based on speed.

Inspector

Properties

Motion Graph

NAME	TYPE	DESCRIPTION
Min Speed	Float	The minimum speed the character must be moving for the sprint animation to play.
Sprint Input Param Key	String	(Required) The switch parameter on the motion graph which is set by the input handler to tell the character when to sprint.
Is Sprinting Param Key	String	(Required) A switch parameter on the motion graph which the graph sets when the character is sprinting.
Can Sprint Param Key	String	(Optional) A switch parameter on the motion graph which tells the character if it can sprint or not.

Animation

NAME	TYPE	DESCRIPTION
In Time	Float	The time taken to blend into the sprint animation.
Out Time	Float	The time taken to blend out of the sprint animation to idle.
Unscaled Sprint Move Speed	Float	The movement speed that the sprint animations are synced to when playing at 1x speed..
Max Speed	Float	A maximum speed clamp for the character when used to calculate the animation speed multiplier.
Blend Zero Speed	Float	The speed below which the light sprint animation will be 100% used. Above this, the heavy animation is blended in.

NAME	TYPE	DESCRIPTION
Blend Full Speed	Float	The speed above which the he\vy sprint animation will be 100% used. Below this, the light animation is blended in.
Sprint Bool Parameter	String	A bool parameter on the Animator to signify when the weapon enters or exits sprint.
Speed Float Parameter	String	A float parameter on the Animator to set the playback speed of the sprint animation.
Blend Float Parameter	String	A float parameter on the Animator used to blend between the light and heavy sprint animations.

See Also

Thrown Weapons
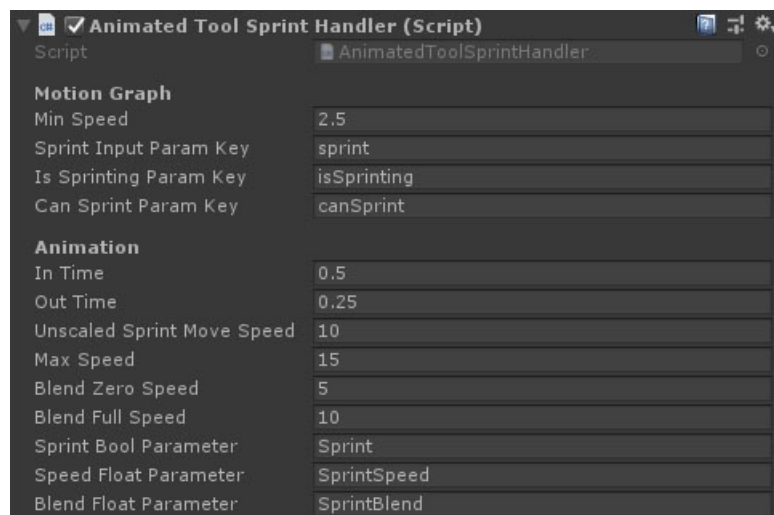
The Motion Graph

Unity Animator

Unity AnimatorController

AnimatedToolSprintHandler MonoBehaviour

Overview

The AnimatedToolSprintHandler behaviour is attached to a wieldable tool to model different sprinting mechanics. This version allows you to use keyframed sprint animations on your tools by setting a bool parameter on an Animator component when the sprint animation should play, and a float parameter to tell the animator the speed. You can use this to set the clip playback speed in the AnimatorController, or to drive a blend between multiple animation clips based on speed.

Inspector

Properties

Motion Graph

NAME	TYPE	DESCRIPTION
Min Speed	Float	The minimum speed the character must be moving for the sprint animation to play.
Sprint Input Param Key	String	(Required) The switch parameter on the motion graph which is set by the input handler to tell the character when to sprint.
Is Sprinting Param Key	String	(Required) A switch parameter on the motion graph which the graph sets when the character is sprinting.
Can Sprint Param Key	String	(Optional) A switch parameter on the motion graph which tells the character if it can sprint or not.

Animation

NAME	TYPE	DESCRIPTION
In Time	Float	The time taken to blend into the sprint animation.
Out Time	Float	The time taken to blend out of the sprint animation to idle.
Unscaled Sprint Move Speed	Float	The movement speed that the sprint animations are synced to when playing at 1x speed..
Max Speed	Float	A maximum speed clamp for the character when used to calculate the animation speed multiplier.

NAME	TYPE	DESCRIPTION
Blend Zero Speed	Float	The speed below which the light sprint animation will be 100% used. Above this, the heavy animation is blended in.
Blend Full Speed	Float	The speed above which the he\vy sprint animation will be 100% used. Below this, the light animation is blended in.
Sprint Bool Parameter	String	A bool parameter on the Animator to signify when the weapon enters or exits sprint.
Speed Float Parameter	String	A float parameter on the Animator to set the playback speed of the sprint animation.
Blend Float Parameter	String	A float parameter on the Animator used to blend between the light and heavy sprint animations.

See Also

Wieldable Tools
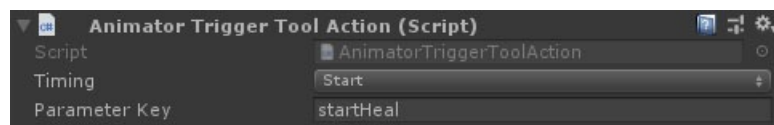
The Motion Graph

Unity Animator

Unity AnimatorController

AnimatorTriggerToolAction MonoBehaviour

Overview

The AnimatorTriggerToolAction sets a trigger parameter on the tool's Animator.

Inspector

Properties

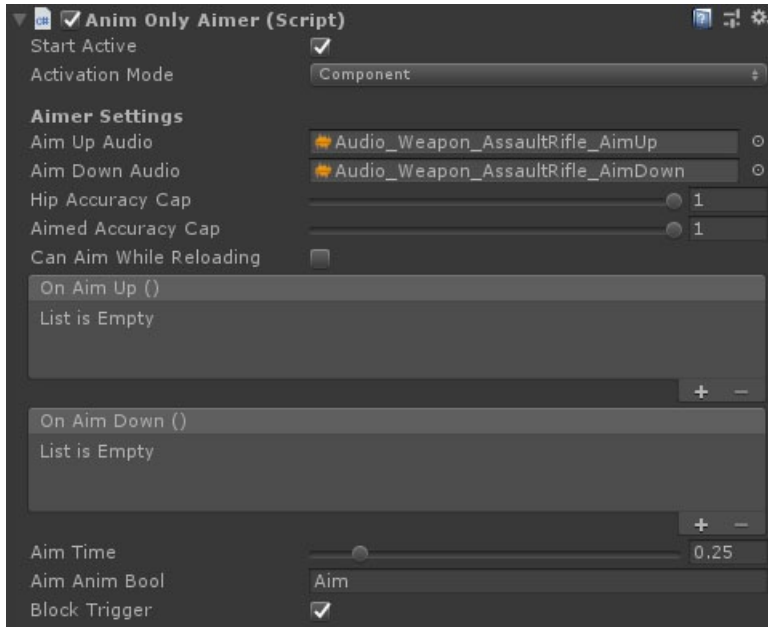| NAME	TYPE	DESCRIPTION
Timing	Dropdown	When should the trigger fire.
Parameter Key	String	The name of the animator trigger parameter to fire.

See Also

Wieldable Tools

Unity Animator

Unity AnimatorController

AnimOnlyAimer MonoBehaviour

Overview

The AnimOnlyAimer module triggers an animation, without moving the camera or weapon, or changing the crosshair. Its main use is when using the modular firearm system with AI.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active aimer immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: **Component** means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. **GameObject** activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Aim Up Audio	AudioClip	An audio clip to play when the weapon is raised.
Aim Down Audio	AudioClip	An audio clip to play when the weapon is lowered.
Hip Accuracy Cap	Float	The highest accuracy the firearm can achieve while not aiming down sights.
Aimed Accuracy Cap	Float	The highest accuracy the firearm can achieve while aiming down sights.
Can Aim While Reloading	Boolean	Should the weapon be lowered when reloading or can it stay aimed.

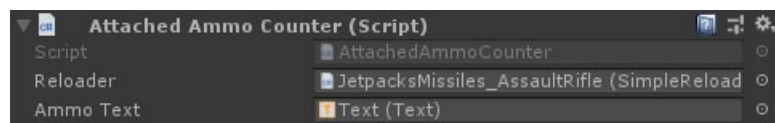| NAME	TYPE	DESCRIPTION
On Aim Up	UnityEvent	An event called when the weapon is fully raised.
On Aim Down	UnityEvent	An event called when the weapon is fully lowered.
Aim Time	Float	The time it takes to reach full aim, or return to zero aim.
Aim Anim Bool	String	The animator parameter key for a bool used to control aiming state in animations.
Block Trigger	Boolean	If true then the gun cannot fire while transitioning in and out of aim mode. This is used to prevent gunshots interrupting the animation. This property will only be shown if the Aim Anim Bool property is true.

See Also

Modular Firearms

First Person Camera

AttachedAmmoCounter MonoBehaviour

Overview

The AttachedAmmoCounter behaviour is added to a world space canvas and used to track the ammo in a gun's magazine.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Reloader	Float	The reloader module to track the current magazine from.
Ammo Text	[Text][unity-uitext]	The text output for the current magazine count.

See Also

Modular Firearms
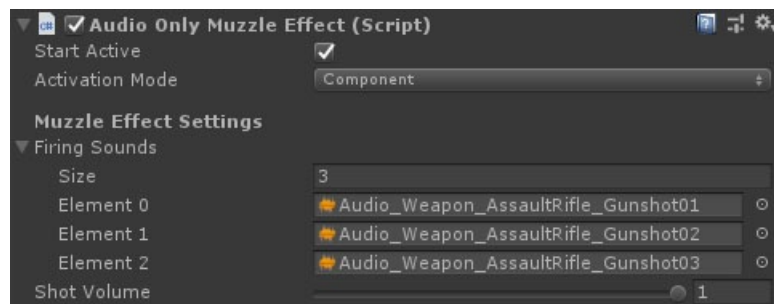
StaminaSystem

AudioOnlyMuzzleEffect MonoBehaviour

Overview

The AudioOnlyMuzzleEffect module plays a sound when the firearm is fired, but has no visual effect. This can be useful for weapons like crossbows.

Inspector

Properties

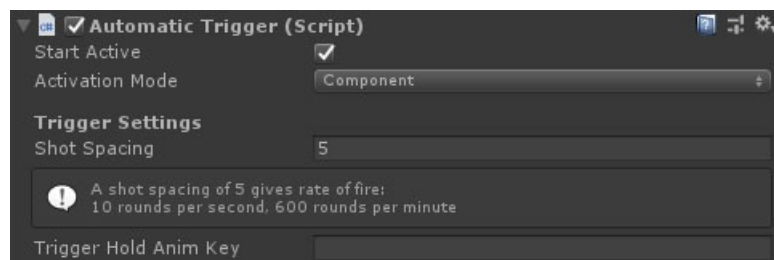| NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active muzzle effect immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: **Component** means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. **GameObject** activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Firing Sounds	AudioClip Array	The audio clips to use when firing. Chosen at random.

See Also

Modular Firearms

AutomaticTrigger MonoBehaviour

Overview

The AutomaticTrigger module fires continuously

Inspector

Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active trigger immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: **Component** means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. **GameObject** activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Shot Spacing	Int	How many fixed update frames between shots. The info box below this show the rate of fire based on the shot spacing.
Trigger Hold Anim Key	String	The AnimatorController bool property key to set while the trigger is pressed.

See Also

Modular Firearms

Unity Animator

Unity AnimatorController

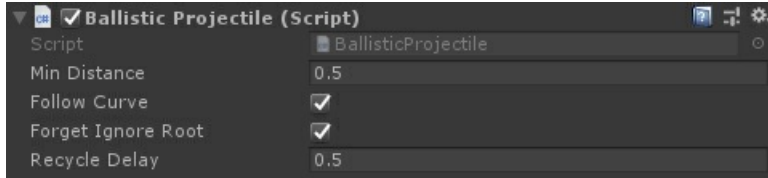# BallisticProjectile MonoBehaviour

Overview

The BallisticProjectile is a basic projectile that follows gravity. It will be given an ammo effect by the firearm that it uses when it impacts.

Inspector

Properties

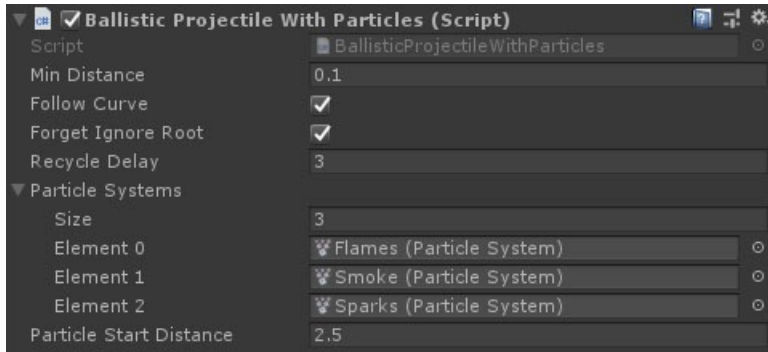| NAME	TYPE	DESCRIPTION
Min Distance	Float	The minimum distance before the projectile will appear.
Follow Curve	Boolean	Should the projectile rotate so it is always facing down the curve.
Forget Ignore Root	Boolean	Forget the character's "ignore root", meaning it can detonate on the character collider.
Recycle Delay	Float	The time after the bullet hits an object before it is returned to the pool (allows trail renderers to complete).

See Also

Modular Firearms

BallisticShooter

BallisticProjectileWithParticles MonoBehaviour

Overview

The BallisticProjectileWithParticles behaviour is a basic projectile that follows gravity. It will be given an ammo effect by the firearm that it uses when it impacts. This projectile uses a particle system to leave a bullet trail which is started a short distance after firing, and ended as soon as the projectile hits its target.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Min Distance	Float	The minimum distance before the projectile will appear.
Follow Curve	Boolean	Should the projectile rotate so it is always facing down the curve.
Forget Ignore Root	Boolean	Forget the character's "ignore root", meaning it can detonate on the character collider.
Recycle Delay	Float	The time after the bullet hits an object before it is returned to the pool (allows trail renderers to complete).
Particle Systems	Particle System Array	The particle systems to play.
Particle Start Distance	Float	The distance the projectile must travel before it starts emitting particles.

See Also

Modular Firearms
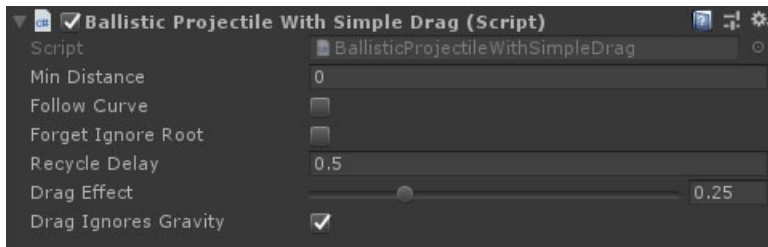
BallisticShooter

BallisticProjectileWithSimpleDrag MonoBehaviour

Overview

The BallisticProjectileWithSimpleDrag adds a simple air resistance / drag option to the BallisticProjectile.

Inspector

Properties

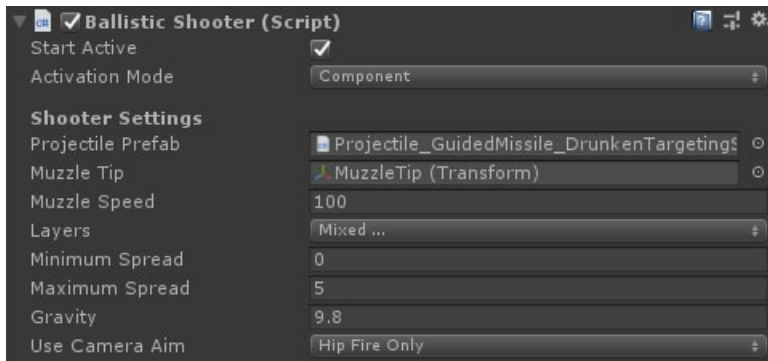| NAME	TYPE	DESCRIPTION
Min Distance	Float	The minimum distance before the projectile will appear.
Follow Curve	Boolean	Should the projectile rotate so it is always facing down the curve.
Forget Ignore Root	Boolean	Forget the character's "ignore root", meaning it can detonate on the character collider.
Recycle Delay	Float	The time after the bullet hits an object before it is returned to the pool (allows trail renderers to complete).
Drag Effect	Float	The strength of the drag on the projectile (uses a simplified multiplier).
Drag Ignores Gravity	Float	If true, then drag will only be applied to the vertical velocity while the projectile is climbing.

See Also

Modular Firearms

BallisticShooter

BallisticShooter MonoBehaviour

Overview

The BallisticShooter module spawns an BallisticProjectile.

Inspector

Properties

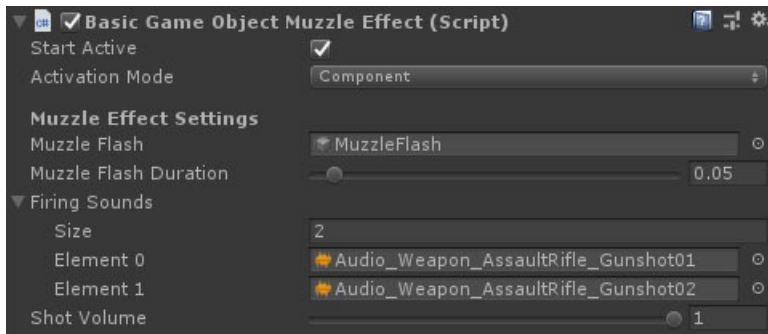| NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active shooter immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: **Component** means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. **GameObject** activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Projectile Prefab	PooledObject	The projectile to spawn.
Muzzle Tip	Transform	The position and direction the projectile is spawned.
Muzzle Speed	Float	The speed of the projectile.
Layers	LayerMask	The physics collision layers the shot can hit.
Minimum Spread	Float	The minimum accuracy spread (in degrees) of the weapon when accuracy is at 1.
Maximum Spread	Float	The maximum accuracy spread (in degrees) of the weapon when accuracy is at 0.
Gravity	Float	The gravity for the projectile.
Use Camera Aim	Dropdown	When set to use camera aim, the gun first casts from the FirstPersonCamera's aim transform, and then from the muzzle tip to that point to get more accurate firing. Options are: **HipFireOnly**, **HipAndAimDownSights**, **AimDownSightsOnly**, **Never**.

See Also

Modular Firearms

BasicGameObjectMuzzleEffect MonoBehaviour

Overview

The BasicGameObjectMuzzleEffect module simple enables a specified game object, and then disables it again after a brief wait.

Inspector

Properties

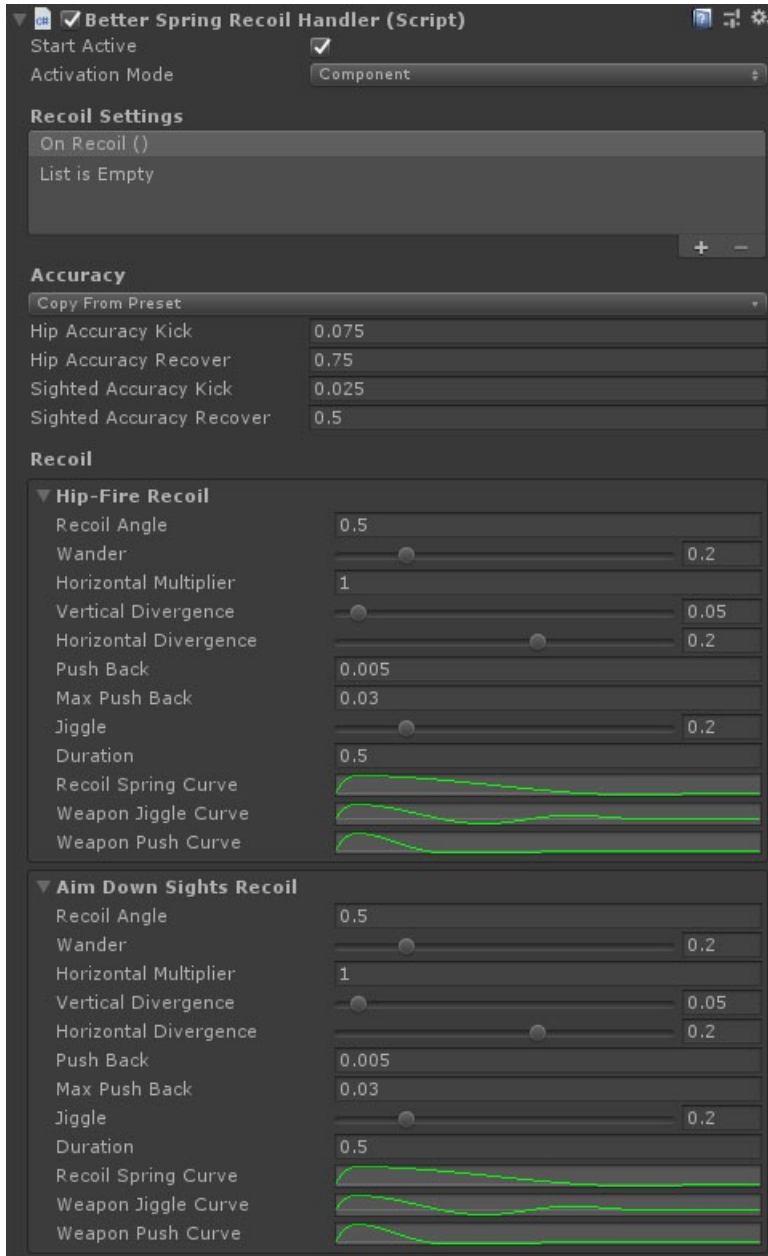| NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active muzzle effect immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: **Component** means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. **GameObject** activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Muzzle Flash	GameObject	The muzzle flash game object.
Muzzle Flash Duration	Float	The duration the flash should remain visible in seconds.
Firing Sounds	AudioClip Array	The audio clips to use when firing. Chosen at random.

See Also

[Modular Firearms](#)

BetterSpringRecoilHandler MonoBehaviour

Overview

The BetterSpringRecoilHandler module controls a FirearmRecoilEffect and a CharacterRecoilEffect behaviour to apply procedural recoil animation to the weapon and player character.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active recoil handler immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: **Component** means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. **GameObject** activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.

NAME	TYPE	DESCRIPTION
On Recoil	[UnityEvent] [unity-event]	An event that fires every time the weapon recoils.
Hip Accuracy Kick	Float	The accuracy decrement per shot in hip fire mode (accuracy has a 0-1 range).
Hip Accuracy Recover	Float	The accuracy recovered per second in hip fire mode (accuracy has a 0-1 range).
Sighted Accuracy Kick	Float	The accuracy decrement per shot in sighted fire mode (accuracy has a 0-1 range).
Sighted Accuracy Recover	Float	The accuracy recovered per second in sighted fire mode (accuracy has a 0-1 range).

Recoil Profiles

The BetterSpringRecoilHandler module has a recoil profile for aiming down sights and for hip fire. Each of these have the following properties:

NAME	TYPE	DESCRIPTION
RecoilAngle	Float	The angle to rotate the gun on recoil (before modifiers).
Wander	Float	The maximum amount of side to side movement during a recoil. At wander = 1, the gun can rotate anything up to 90 degrees from up. At 0, the gun will only rotate upwards.
HorizontalMultiplier	Float	A multiplier applied to any horizontal rotation of the recoil.
VerticalDivergence	Float	The split between head and weapon recoil. At 0, the head/body will rotate, and the firearm will move with it. At 1, the firearm will rotate and the head/body will remain still.
HorizontalDivergence	Float	The split between head and weapon recoil. At 0, the head/body will rotate, and the firearm will move with it. At 1, the firearm will rotate and the head/body will remain still.
PushBack	Float	The distance the firearm will be pushed backwards each shot.
MaxPushBack	Float	The maximum distance the firearm can be pushed back.
Jiggle	Float	The amount of jiggle (spring rotation around the z-axis) to be applied to the firearm.
Duration	Float	The time taken for the head and firearm to return to their pre-recoil state.
RecoilSpringCurve	AnimationCurve	The animation curve used to drive the amount of recoil rotation over the duration of the recoil effect.
WeaponJiggleCurve	AnimationCurve	The animation curve used to drive the firearm's jiggle rotation over the duration of the recoil effect.

NAME	TYPE	DESCRIPTION
WeaponPushCurve	AnimationCurve	The animation curve used to drive the firearm's push-back over the duration of the recoil effect.

See Also

Modular Firearms

FirearmRecoilEffect

CharacterRecoilEffect

Additive Transforms and Effects

BlinkToolModule MonoBehaviour

Overview

The BlinkToolModule behaviour is used to model an Arkane style blink (Dishonoured, Deathloop, Prey) as a tool action. Aim at an area of flat ground or a ledge, hold the fire button until the markers are shown. Release to blink to that location. This is intended to be used alongside the MoveToPointState motion graph state.

Inspector
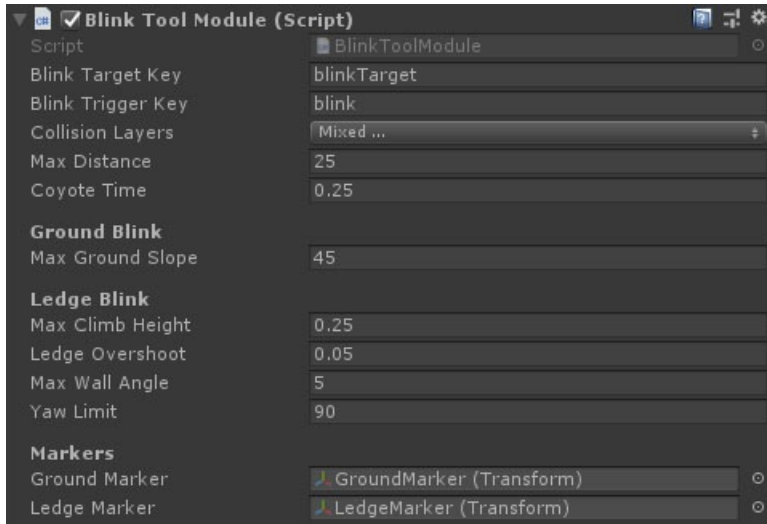
Properties

NAME	TYPE	DESCRIPTION
Blink Target Key	String	The key to a vector parameter in the wielder's motion graph that will be set with the target position.
Blink Trigger Key	String	The key to a trigger parameter in the wielder's motion graph that will be set when the blink is performed (trigger released on a valid target location).
Collision Layers	Layer Mask	The physics layers the tool can blink onto.
Max Distance	Float	The maximum distance that the character can blink.
Coyote Time	Float	The last valid blink target will be stored this long and used. Makes the target selection more forgiving.
Max Ground Slope	Float	The maximum slope that a ground plane can have for you to blink onto it.
Max Climb Height	Float	The maximum distance down a wall you can look and it will blink to the top ledge.
Ledge Overshoot	Float	If looking at the top of a wall, this is the distance past the edge to overshoot when blinking onto the ledge.

NAME	TYPE	DESCRIPTION
Max Wall Angle	Float	The maximum angle away from vertical that the blink tool can register as a wall with a ledge.
Yaw Limit	Float	An angle range centered on the wall normal that you can blink within. At 180 degrees you can blink to any ledge. At 0 degrees you would have to be looking perfectly flat on to the wall to blink to its ledge.
Ground Marker	Transform	An object in the tool's hierarchy to use as the ground position marker for the blink target (the object will be moved out of the tool's hierarchy).
Ledge Marker	Transform	An object in the tool's hierarchy to use as the ledge position marker for the blink target (the object will be moved out of the tool's hierarchy).

See Also

Wieldable Tools

Motion Graph Parameters And Data
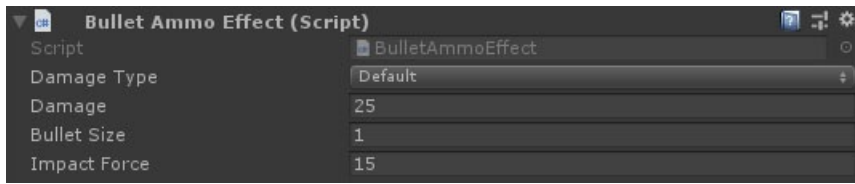
BulletAmmoEffect MonoBehaviour

Overview

The BulletAmmoEffect module uses the surfaces system to show impact effects, and imparts damage and force to whatever it hits.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Damage Type	DamageType	The type of damage the weapon should do with this ammo.
Damage	Float	The damage the bullet does.
Bullet Size	Float	The size of the bullet. Used to size decals.
Impact Force	Float	The force to be imparted onto the hit object. Requires either a Rigidbody or an impact handler.

See Also

Modular Firearms

Surfaces

BurstFireTrigger MonoBehaviour

Overview

The BurstFireTrigger module fires a set number of shots in rapid succession.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active trigger immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: **Component** means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. **GameObject** activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Burst Size	Int	The number of shots in a burst.
Burst Spacing	Int	Cooldown between shots in a burst (number of fixed update frames). The info box below this show the rate of fire based on the shot spacing.
Repeat Delay	Int	How many fixed update frames from last shot of a burst before starting another (0 = requires fresh trigger press).
Cancel On Release	Boolean	Is the burst cancelled when the trigger is released.

See Also

Modular Firearms
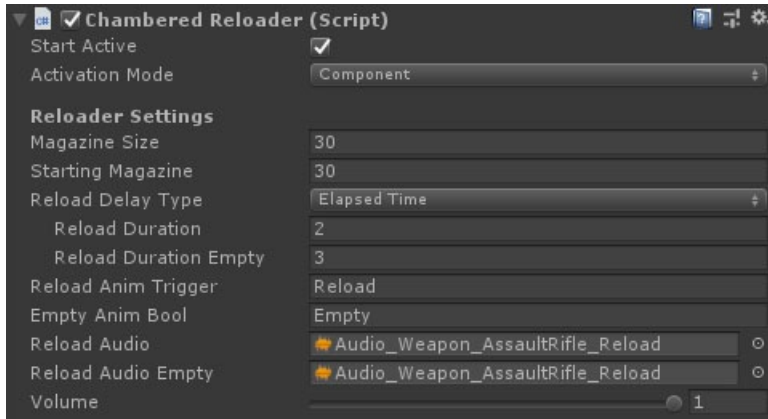
ChamberedReloader MonoBehaviour

Overview

The ChamberedReloader behaviour triggers a reload animation and waits until it completes before taking ammo from the firearm's ammo module and adding it to the magazine. The ChamberedReloader will set a bool parameter in the firearm's [AnimatorController][unity-animatorcontroller] so that it can change the animations it plays, and it can have a different wait duration and audio clip to play if reloading from an empty magazine compared to with a round in the chamber.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active reloader immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: **Component** means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. **GameObject** activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Magazine Size	Float	The number of rounds that can be fit in the magazine at once.
Starting Magazine	Float	The number of rounds in the magazine on initialisation.
Reload Delay Type	Dropdown	The delay type between starting and completing a reload. The options are **None**, **Elapsed Time**, **External Trigger**.
Reload Duration	Float	The time taken to reload.
Reload Duration Empty	Float	The time taken to reload if reloading from empty.
Reload Anim Trigger	String	The [AnimatorController][unity-animatorcontroller] trigger parameter key for the reload animation.

NAME	TYPE	DESCRIPTION
Empty Anim Bool	String	The key to an [AnimatorController][unity-animatorcontroller] bool parameter to set when the weapon is empty.
Reload Audio	AudioClip	The audio clip to play while reloading.
Reload Audio Empty	AudioClip	The audio clip to play if reloading from empty.

See Also

Modular Firearms
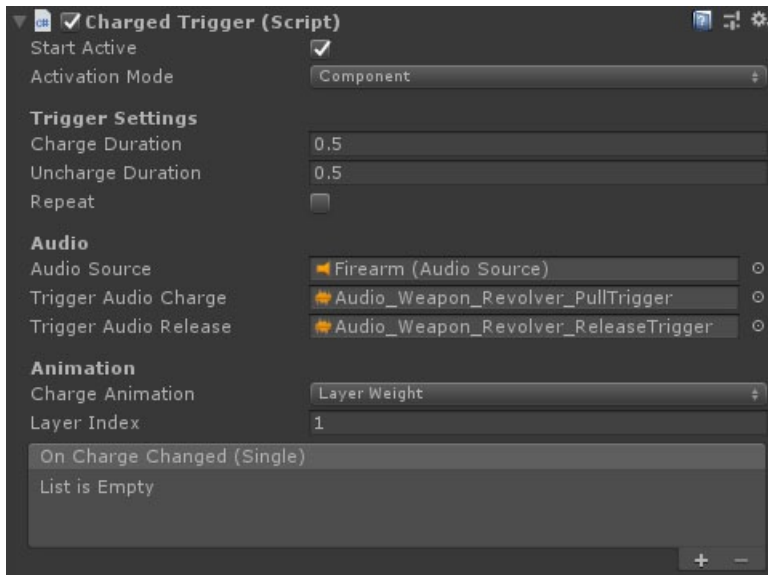
FirearmAnimEventsHandler

[Unity Animator][unity-animator]

[Unity AnimatorController][unity-animatorcontroller]

ChargedTrigger MonoBehaviour

Overview

The ChargedTrigger module charges up while the fire button is held down and fires once it hits full charge.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active trigger immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: **Component** means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. **GameObject** activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Charge Duration	Float	How long does it take to charge the trigger.
Uncharge Duration	Float	How long does it take to uncharge the trigger, assuming it hasn't gone off.
Repeat	Boolean	Once the shot is fired, start charging the next shot if this is true.
Repeat Delay	Float	The time between a shot firing and starting charging the next shot.
Audio Source	AudioSource	The source to play the audio from (needs its own as it must be interrupted and seeked).
Trigger Audio Charge	AudioClip	The audio clip to play on charge.
Trigger Audio Release	AudioClip	The audio clip to play on release.

NAME	TYPE	DESCRIPTION
Charge Animation	Dropdown	How should the charge be animated. **LayerWeight** blends in a layer in the weapon's animator as charge increases. **FloatParameter** sets a float parameter value on the animator. **EventsOnly** fires an event when the charge changes, but does not affect the attached animator.
Layer Index	Integer	The animator layer index to blend in for the charge effect. This option will only be visible if "Charge Animation" is set to **LayerWeight**.
Charge Anim Key	String	The float animator parameter key to set when the trigger charge changes. This option will only be visible if "Charge Animation" is set to **FloatParameter**.
On Charge Changed	UnityEvent	An event fired each frame the charge changes.

See Also

Modular Firearms

ChargeToolAction MonoBehaviour

Overview

The ChargeToolAction behaviour is used to add a charge up effect to a wieldable tool. Charging all the way to full will interrupt the tool and send the `success = true` signal to the end actions. Releasing the trigger early will send the `success = false` signal.

Inspector
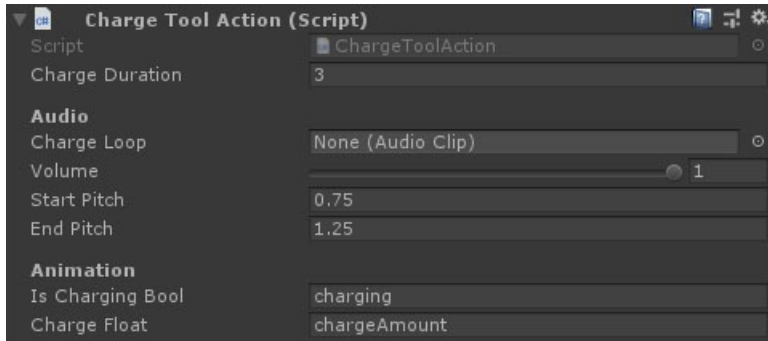
Properties

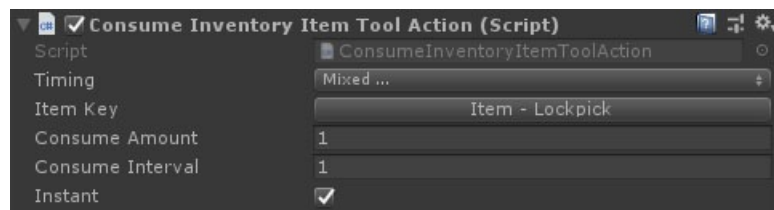| NAME	TYPE	DESCRIPTION
Charge Duration	Float	The time it takes to reach full charge.
Charge Loop	AudioClip	The audio to play while charging.
Volume	Float	The volume of the charging audio loop.
Start Pitch	Float	The pitch of the audio loop at the start of the charge.
End Pitch	Float	The pitch of the audio loop when charge hits 100%.
Is Charging Bool	String	A bool parameter in the tool's animator that should be set while charging.
Charge Float	String	A float parameter in the tool's animator used to show charge progress (0-1).

See Also

Wieldable Tools

Unity Animator

Unity AnimatorController

ConsumeInventoryItemToolAction MonoBehaviour

Overview

The ConsumeInventoryItemToolAction behaviour is used to add the effect of using an inventory item to a wieldable tool either on start or end, or continuously over time.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Timing	Dropdown	When should the item be consumed.
Item Key	ID Picker	The inventory ID of the object to consume. Clicking this button will open the inventory database item picker.
Consume Amount	Integer	How many of the item should be consumed when triggered or ticked.
Consume Interval	Integer	The items should be consumed every nth fixed update frame in continuous mode.
Instant	Boolean	Should the item be consumed on the first frame of the continuous action, or should it wait for the first interval to elapse.

See Also

Wieldable Tools

Inventory

ContactGrenadeThrownProjectile MonoBehaviour

Overview

The ContactGrenadeThrownProjectile behaviour is used to create grenades which explode the moment they hit a target.

Inspector
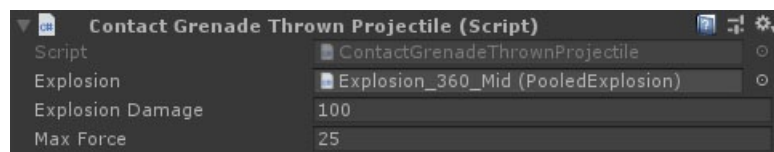
Properties

NAME	TYPE	DESCRIPTION
Explosion	PooledExplosion	The explosion object to spawn on contact with an object.
Damage	Float	The damage the explosion does at its center.
Max Force	Float	The maximum force to be imparted onto any objects in the explosion's radius. Requires either a [Rigidbody] [unity-rigidbody] or an impact handler and drops off to zero the further from the center the object is.

See Also

Explosions

Health and Damage

CustomAmmo MonoBehaviour

Overview

The CustomAmmo module is a unique ammo for the firearm it is attached to.

Inspector

Properties

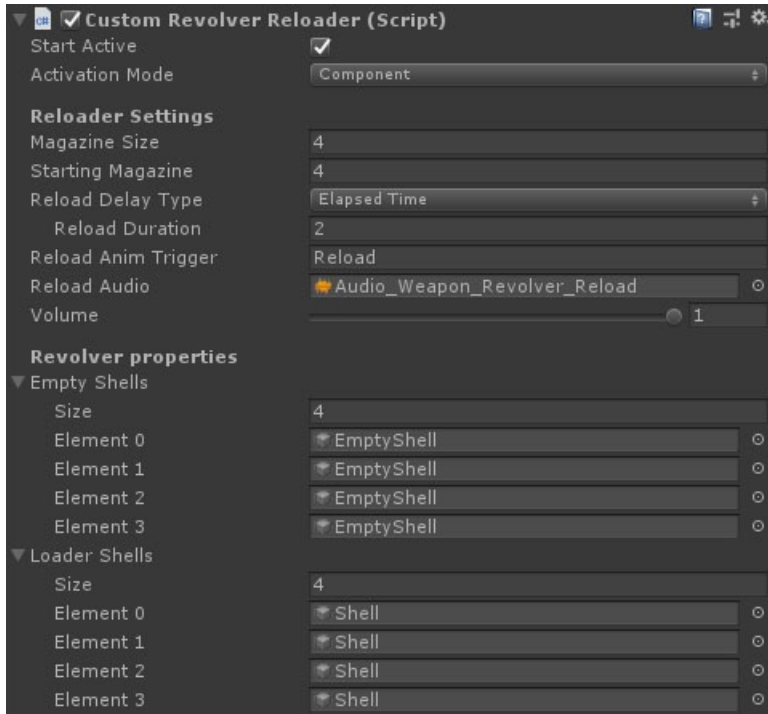| NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active ammo module immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: **Component** means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. **GameObject** activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Printable Name	String	The name to show on the HUD.
Effect	Dropdown	A dropdown which shows all the AmmoEffect modules on the GameObject. This is the effect the bullets will have on hitting a target.
Starting Ammo	Int	The amount of ammo the weapon starts with.
Max Ammo	Int	The maximum amount of ammo the weapon can carry.

See Also

Modular Firearms

CustomRevolverReloader MonoBehaviour

Overview

The CustomRevolverReloader module is adpated from the SimpleReloader. It adds functionality to make the spent and reloaded rounds visible and the visbile count correct based on how many bullets are loaded into the gun.

Inspector

Properties

The CustomRevolverReloader module inherits most of its properties from the SimpleReloader. It adds the following properties:

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active reloader immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: **Component** means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. **GameObject** activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Magazine Size	Float	The number of rounds that can be fit in the magazine at once.
Starting Magazine	Float	The number of rounds in the magazine on initialisation.
Empty Shells	GameObject Array	The empty shells in the revolver cylinder.
Loader Shells	GameObject Array	The unused shells in the revolver speed loader.

See Also

Modular Firearms
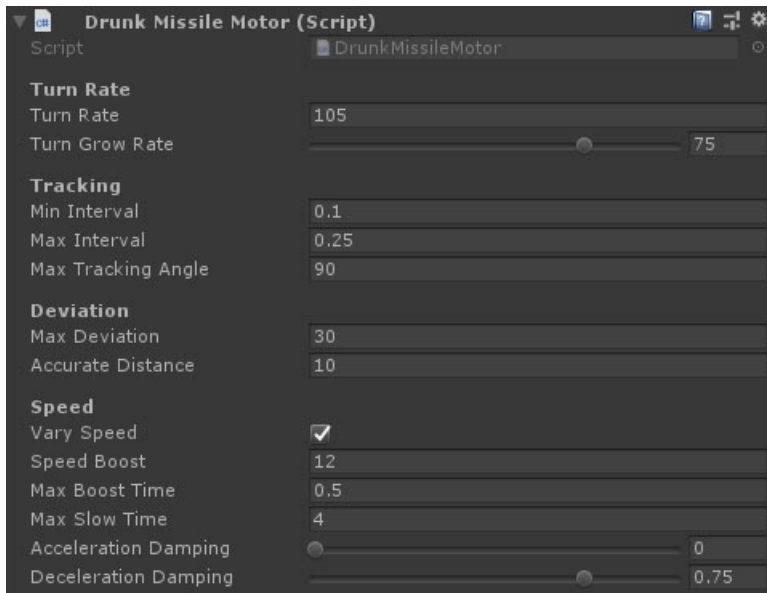
[Unity Animator][unity-animator]

[Unity AnimatorController][unity-animatorcontroller]

DrunkMissileMotor MonoBehaviour

Overview

The DrunkMissileMotor behaviour is added to a guided projectile to make it follow an erratic path towards its target.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Turn Rate	Float	The turn rate of the projectile (degrees per second).
Turn Grow Rate	Float	An increase to the turn rate based on elapsed time in flight (base turn rate + turn grow rate * elapsed time).
Min Interval	Float	The minimum amount of time between tracking ticks (when the projectile updates its target location).
Max Interval	Float	The maximum amount of time between tracking ticks (when the projectile updates its target location).
Max Tracking Angle	Float	The maximum angle off the forwards direction that the tracker can swivel to look at the target each tick.
Max Deviation	Float	The maximum deviation from the tracker angle that the projectile will steer to add the drunken effect.
Accurate Distance	Float	The distance from the target below which the projectile will switch to a more consistent tracking (reduces projectiles overshooting).
Vary Speed	Boolean	Should the projectile's speed vary with occasional boosts.
Speed Boost	Float	The target boost speed (added to the base speed of the projectile when fired).
Max Boost Time	Float	The maximum amount of time a boost will last.
Max Slow Time	Float	The maximum amount of time between boosts.

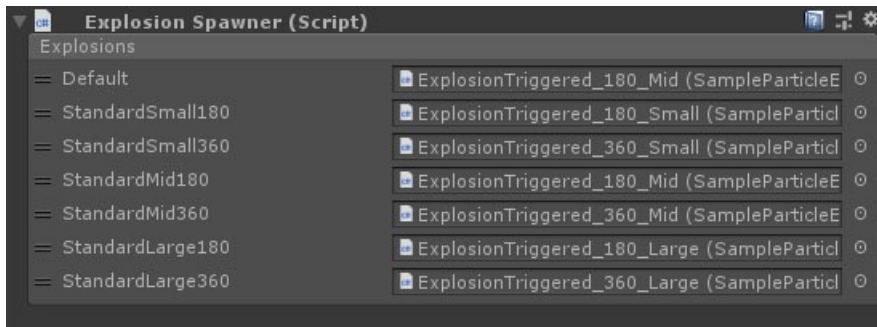| NAME	TYPE	DESCRIPTION
Acceleration Damping	Float	The damping when accelerating up to boost speed.
Deceleration Damping	Float	The damping when decelerating from boost back down to base speed.

See Also

Modular Firearms

Hitscan vs Projectiles

ExplosionSpawner MonoBehaviour

Overview

The explosion spawner will spawn an explosion at a point in space on request. This is a demo implementation.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Exploder	Explosion	The explosion to spawn. Must innherit from `BaseExplosionBehaviour`

See Also

Explosions

ExplosiveAmmoEffect MonoBehaviour

Overview

The ExplosiveAmmoEffect module spawns an explosion on impact, dealing damage and repelling objects.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Damage Type	DamageType	The type of damage the weapon should do with this ammo.
Explosion Type	ExplosionType	The id of the explosion to spawn.
Radius	Float	The radius of the explosion.
Damage	Float	The damage the explosion does at its center.
Max Force	Float	The max force to be imparted onto any objects in the explosion radius. The force falls off as distance from the center increases. Requires either a Rigidbody or an impact handler.

See Also

Modular Firearms

Explosions

ExplosiveObject MonoBehaviour

Overview

The ExplosiveObject behaviour is used to create objects like barrels that spawn a PooledExplosion when killed.

Inspector

Properties

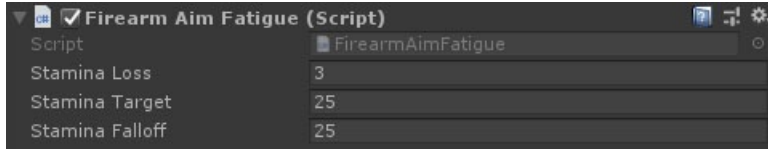| NAME	TYPE	DESCRIPTION
Explosion	PooledExplosion	The explosion object to spawn.
Offset	Vector3	An offset from the object's origin that the explosion will spawn.
Damage	Float	The damage does at the center of the explosion. Drops off to zero at the edge of the radius.
Max Force	Float	The maximum force to be imparted onto any objects in the explosion's radius. Requires either a [Rigidbody][unity-rigidbody] or an impact handler and drops off to zero the further from the center the object is.
Takes Damage	DamageType	The type of damage the object can take.
Health	Float	The amount of health the object has.
Destroy Gameobject	Dropdown	What to do to the object when it's killed. Available options are: **Destroy**, **Disable**, **Nothing**.
On Destroyed	UnityEvent	An event invoked when the item's health reaches zero

See Also

Explosions

Health and Damage

FirearmAimFatigue MonoBehaviour

Overview

The FirearmAimFatigue behaviour is used to drain a character's stamina while aiming down sights using the. This requires the character to have a StaminaSystem behaviour attached.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Stamina Loss	Float	The stamina loss per second when aiming down sights.
Stamina Target	Float	The stamina level to drain down to.
Stamina Falloff	Float	Stamina drain fades when approaching the target, starting at this falloff value above it.

See Also

Modular Firearms

StaminaSystem

FirearmAnimEventsHandler MonoBehaviour

Overview

The FirearmAnimEventsHandler behaviour is used to catch events fired from the Animator attached to this game object. This enables specific triggers to be synced with animation and then feed back to the firearm.

The currently captured events are:

```
public void WeaponRaised ()
```

```
public void FirearmReloadPartial (int count)
```

```
public void FirearmReloadComplete ()
```

```
public void FirearmEjectShell ()
```

Inspector

Properties

The FirearmAnimEventsHandler behaviour has no properties exposed in the inspector.

See Also

Modular Firearms

Unity Animator

Unity Animation Events

FirearmOverheat MonoBehaviour

Overview

The FirearmOverheat behaviour is used to model a simple heat accumulation in a gun as it shoots. It can also optionally cause the weapon to overheat and lock the trigger until it cools to a certain threshold.


The glow effect requires that one of the NeoFPS glow shaders be used. These are found under the **NeoFPS/Standard** shader path, and are named based on their PBR surface type and glow type: **GlowMetallic** or **GlowSpecular** and **Alpha Masked**, **Distance Masked** and **Position Masked**.

- **Alpha Masked** uses a greyscale glow mask. Glow will be shown wherever the texture is not black, with full strength where it is white.
- **Distance Masked** specifies a 3D point (in object space), and the glow fades out based on the distance from that point.
- **Position Masked** sets start and end points on the object space z-axis, and the glow fades in and out between these points.

The haze effect requires a mesh renderer with a material using the **NeoFPS/Standard/HeatHaze** shader. This shader uses a property block to set the haze strength as the heat increases, and the object is disabled when the heat is zero. A simple haze effect can be created by combining a quad 3D geometry object with a BillboardOrientation Monobehaviour to keep it oriented at the first person camera. An example prefab can be found at *NeoFPS/Samples/Shared/Effects/Prefabs/GunBarrelHazeBillboard*.

Inspector
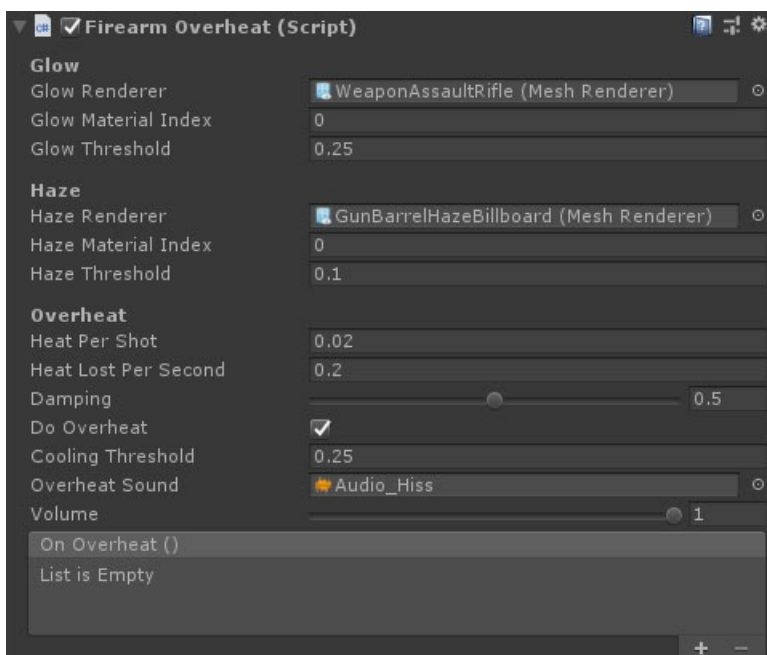
Properties

NAME	TYPE	DESCRIPTION
Glow Renderer	MeshRenderer	The mesh renderer of the glow material.
Glow Material Index	Integer	The index of the glow material in the mesh renderer.
Glow Threshold	Float	The heat level (0 to 1) before the glow starts to appear.
Haze Renderer	MeshRenderer	The mesh renderer of the glow material.
Haze Material Index	Integer	The index of the haze material in the mesh renderer.
Haze Threshold	Float	The heat level (0 to 1) before the haze starts to appear.
Heat Per Shot	Float	The amount of heat to add with each shot of the weapon. When this reaches 1, the gun must cool down before it can fire again.
Heat Lost Per Second	Float	The amount of heat that is dissipated per second. The weapon will never overheat if this is higher than the heat per shot multiplied by rate of fire (rounds per second).
Do Overheat	Float	If true, then once the weapon reaches max heat the weapon will overheat, blocking the trigger until it has cooled down to a set threshold.
Cooling Threshold	Float	Once overheated, the weapon must cool to this heat level before it can fire again.
Overheat Sound	AudioClip	The audio clip to play once max heat is hit and the trigger is blocked.
Volume	Float	The volume to play the overheat sound at.
On Overheat	Float	An event that is fired when the heat hits the max level.

See Also

Modular Firearms

BillboardOrientation Monobehaviour

FirearmTransformMatchSetter MonoBehaviour

Overview

The FirearmTransformMatchSetter behaviour is attached to a firearm object. When that firearm is added to a character and equipped it will set the transforms of the TransformMatcher additive effect attached to the character head. This can be used to sync the character head movement to animations attached to the head, while still allowing additive effects such as recoil and shake.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Target	Transform	The target transform to match.
Relative To	Transform	The transform to use when calculating the offset of the target transform.
Weight	Float	The strength of the effect. 1 matches the movement absolutely, while 0 is no movement.

See Also

Modular Firearms

TransformMatcher

FirearmWieldableStanceManager MonoBehaviour

Overview

The FirearmWieldableStanceManager behaviour is used to specify poses or stances for a modular firearm. The entire weapon will be moved to match the stance, and can optionally set an animator bool parameter too.

The stance will be temporarily exited when aiming and reloading.

Inspector

Properties

Use the **Add Stance** buttons to add a new stance to the manager.

Stances

Individual stances have the following properties:

NAME	TYPE	DESCRIPTION
Name	String	The name of the stance.
Animator Bool Key	String	An optional name of a bool parameter in the weapon's Animator.
Position	Vector	The position to move the weapon to in this stance.
Rotation	Vector	The rotation of the weapon in this stance.
In Position Blend	Dropdown	The easing method for blending between the source position and stance position on entering the stance. Options are: **Lerp, EaseIn, EaseOut, EaseInOut, SwingAcross, SwingUp, Spring, Bounce** and **Overshoot**.
In Rotation Blend	Dropdown	The easing method for blending between the source rotation and stance rotation on entering the stance. Options are: **Lerp, Slerp, EaseIn, EaseOut, EaseInOut, Spring, Bounce, Overshoot**.
In Time	Float	The time taken to enter the stance.

NAME	TYPE	DESCRIPTION
Out Position Blend	Dropdown	The easing method for blending between the stance position and idle position on exiting the stance. Options are: **Lerp, **EaseIn**, **EaseOut**, **EaseInOut**, **SwingAcross**, **SwingUp**, **Spring**, **Bounce** and **Overshoot**.
Out Rotation Blend	Dropdown	The easing method for blending between the stance rotation and Idle rotation on exiting the stance. Options are: **Lerp, **Slerp**, **EaseIn**, **EaseOut**, **EaseInOut**, **Spring**, **Bounce**, **Overshoot**.
Out Time	Float	The time taken to enter the stance.

See Also

Modular Firearms

FlashlightToolModule MonoBehaviour

Overview

The FlashlightToolModule behaviour is used to add a toggleable flashlight to wieldable tools.

Inspector
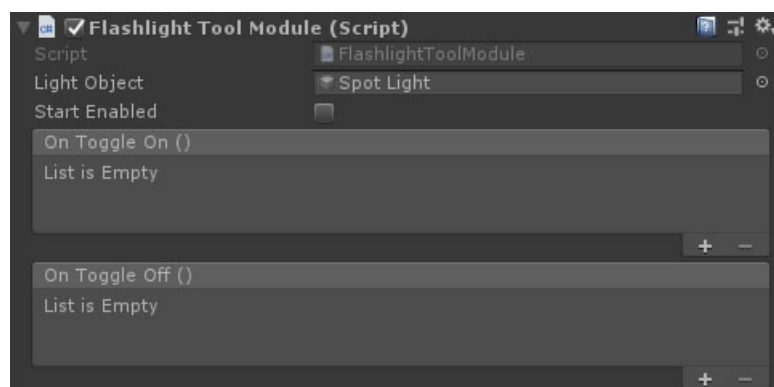
Properties

NAME	TYPE	DESCRIPTION
m_LightObject	GameObject	A child object with a light component attached.
m_StartEnabled	Boolean	Should the flashlight be enabled on start.

See Also

Wieldable Tools

GrappleToolModule MonoBehaviour

Overview

The GrappleToolModule behaviour is used to set a vector parameter on the wielding character's motion graph with an anchor point to grapple to. This can be combined with the GrappleSwingState motion graph state to add grapple mechanics to the game.

Inspector
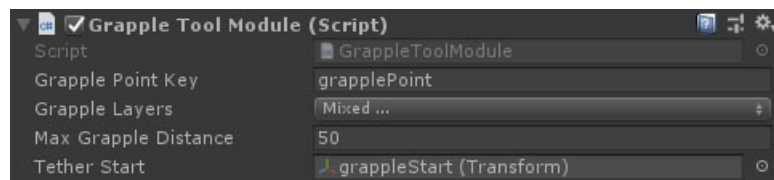
Properties

NAME	TYPE	DESCRIPTION
Grapple Point Key	String	The name of the vector parameter on the character's motion graph that you want to set the grapple point to.
Grapple Layers	LayerMask	The physics layers the tool can grapple onto.
Max Grapple Distance	Float	The maximum distance that a grapple can connect.
Tether Start	Transform	The transform to use as the start point of the tether line renderer.

See Also

Modular Firearms

The Motion Graph

GrenadeThrownProjectile MonoBehaviour

Overview

The GrenadeThrownProjectile behaviour is used to create grenades which explode after a set amount of time.

Inspector
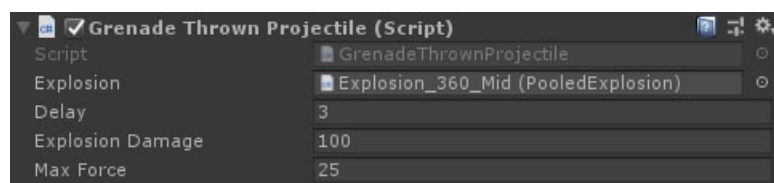
Properties

NAME	TYPE	DESCRIPTION
Explosion	PooledExplosion	The explosion object to spawn once the timer expires.
Delay	Float	The delay before exploding.
Damage	Float	The damage the explosion does at its center.
Max Force	Float	The maximum force to be imparted onto any objects in the explosion's radius. Requires either a [Rigidbody] [unity-rigidbody] or an impact handler and drops off to zero the further from the center the object is.

See Also

Explosions

Health and Damage

GuidedBallisticProjectile MonoBehaviour

Overview

The GuidedBallisticProjectile is a projectile that uses a tracker component and a motor component to control its flight path. The tracker component is used to define what the projectile is tracking, while the motor defines its flight characteristics.

Inspector
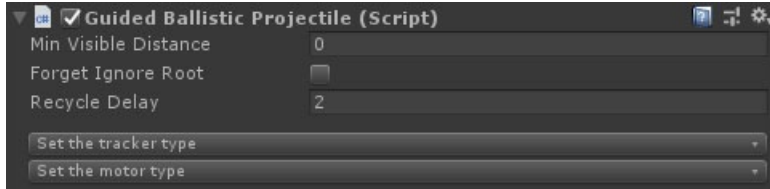
Properties

NAME	TYPE	DESCRIPTION
Min Visible Distance	Float	The minimum distance before the projectile will appear.
Forget Ignore Root	Boolean	Forget the character's "ignore root", meaning it can detonate on the character collider.
Recycle Delay	Float	The time after the bullet hits an object before it is returned to the pool (allows trail renderers to complete).

See Also

[Modular Firearms](#)

[BallisticShooter](#)

[Hitscan vs Projectiles](#)

HeadMoveAimer MonoBehaviour

Overview

The HeadMoveAimer module moves the character head to align the center of the camera with the weapon sights. This can actually tilt the camera, providing a different effect to the standard WeaponMoveAimer

In The Scene View

With a gameobject selected that uses a HeadMoveAimer component, the aimer guide handle will be visible in the scene view. This is used to represent the aim point of the aimer, with the blue arrow pointing forwards, and the green arrow pointing up.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active aimer immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: **Component** means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. **GameObject** activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Aim Up Audio	AudioClip	An audio clip to play when the weapon is raised.
Aim Down Audio	AudioClip	An audio clip to play when the weapon is lowered.
Hip Accuracy Cap	Float	The highest accuracy the firearm can achieve while not aiming down sights.
Aimed Accuracy Cap	Float	The highest accuracy the firearm can achieve while aiming down sights.
Can Aim While Reloading	Boolean	Should the weapon be lowered when reloading or can it stay aimed.
On Aim Up	UnityEvent	An event called when the weapon is fully raised.
On Aim Down	UnityEvent	An event called when the weapon is fully lowered.
Root Transform	Transform	The root transform of the hierarchy (should align with the camera).
Aim Offset	Transform	A target aim transform. The camera will be moved to align it to this transform when aiming down sights. Offsets are calculated on Awake, so moving this transform at after this point has no effect.
Aim Position Offset	Vector3	The offset for the camera transform to move to align the weapon sights with the camera. This property will only be visible if no **AimOffset** transform is set.
Aim Rotation Offset	Vector3	An euler angle offset for the camera to move to align the weapon sights with the camera. This property will only be visible if no **AimOffset** transform is set.
Fov Multiplier	Float	A multiplier for the camera FoV for aim zoom.
Input Multiplier	Float	A multiplier for the camera input when aiming down sights. By default, this value is locked to the FoV multiplier. Clicking the lock icon next to the 2 multipliers allows you to modify it separately for situations such as render texture scopes.
Aim Time	Float	The time it takes to reach full aim, or return to zero aim.

NAME	TYPE	DESCRIPTION
Position Spring Multiplier	Float	A multiplier for procedural spring position animation on the weapon. Used to reduce movement when the firearm is close to the camera.
Rotation Spring Multiplier	Float	A multiplier for procedural spring rotation animation on the weapon. Used to reduce movement when the firearm is close to the camera.
Aim Anim Bool	String	The animator parameter key for a bool used to control aiming state in animations.
Block Trigger	Boolean	If true then the gun cannot fire while transitioning in and out of aim mode. This is used to prevent gunshots interrupting the animation. This property will only be shown if the Aim Anim Bool property is true.
Crosshair Up	FpsCrosshair	The crosshair to show when aiming down sights.
Crosshair Down	FpsCrosshair	The crosshair to show when not aiming down sights.

See Also

Modular Firearms

First Person Camera

HealToolAction MonoBehaviour

Overview

The HealToolAction behaviour adds the ability to heal the wielder or another subject to a wieldable tool.

Inspector
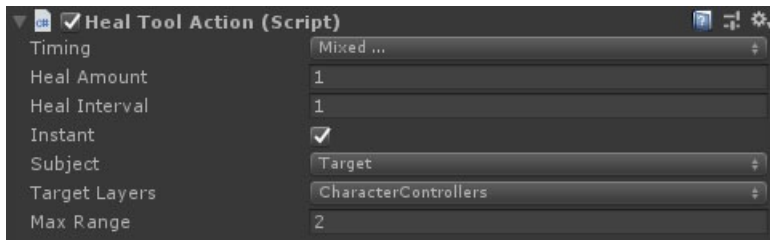
Properties

NAME	TYPE	DESCRIPTION
Timing	Dropdown	When should the tool apply the heal.
Heal Amount	Integer	How many points to heal the subject for.
Subject	Dropdown	Who/what to heal. Wielder means the character using the tool will heal themselves. Target means the tool will be used on the health manager in front of the user.
Heal Interval	Integer	The heal will be applied every nth fixed update tick for continuous heals.
Instant	Boolean	Should the heal be applied on the first frame of the continuous action, or should it wait for the first interval to elapse.
Target Layers	LayerMask	The physics layers that the tool should check against to get a valid heal subject.
Max Range	Float	The maximum distance in front of the character that the tool should cast to check for valid heal subjects.

See Also

Wieldable Tools

Health and Damage

HitscanShooter MonoBehaviour

Overview

The HitscanShooter module raycasts directly out from the muzzle of the gun based on the current accuracy.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active shooter immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: **Component** means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. **GameObject** activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Max Distance	Float	The maximum distance that the weapon will register a hit.
Muzzle Tip	Transform	The transform that the bullet actually fires from.
Layers	LayerMask	The physics collision layers the shot can hit.
Minimum Spread	Float	The minimum accuracy spread (in degrees) of the weapon.
Maximum Spread	Float	The maximum accuracy spread (in degrees) of the weapon
Use Camera Aim	Dropdown	When set to use camera aim, the gun first casts from the FirstPersonCamera's aim transform, and then from the muzzle tip to that point to get more accurate firing. Options are: **HipFireOnly**, **HipAndAimDownSights**, **AimDownSightsOnly**, **Never**.
Tracer Prototype	Pooled Object	The optional pooled tracer prototype to use (must implement the IPooledHitscanTrail interface).
Tracer Size	Float	The size (thickness/radius) of the tracer line.

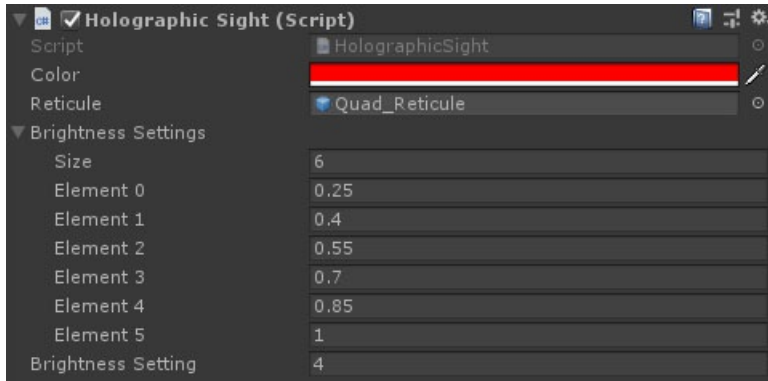| NAME	TYPE	DESCRIPTION
Tracer Duration	Float	How long should the tracer object stay visible.

See Also

[Modular Firearms](#)

HolographicSight MonoBehaviour

Overview

The InfiniteAmmo behaviour is used to define weapon optics that project a reticule in front of the weapon when viewed through the sight. This is achieved using stencil shaders, and the holgraphic sight behaviour's main use is to control the colour and the brightness of the reticule.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Color	Color	The base colour of the reticule.
Reticule	GameObject	The game object with the reticule mesh attached. This should be placed directly in front of the weapon at the desired distance.
Brightness Settings	Float Array	A series of brightness values that can be cycled through with the **Optics Brightness +/-** inputs.
Brightness Setting	int	The index of the starting brightness setting from the above array.

See Also

[Modular Firearms](#)

[Scopes & Optics](#)

IncrementalReloader MonoBehaviour

Overview

The IncrementalReloader module is a more complex reloader that increments the ammo count bit by bit. It will increment for each bullet required and can be interrupted. It is used by the demo shotgun.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active reloader immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: **Component** means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. **GameObject** activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Magazine Size	Float	The number of rounds that can be fit in the magazine at once.
Starting Magazine	Float	The number of rounds in the magazine on initialisation.
Rounds Per Increment	Integer	The number of rounds to load into the gun each increment.
Use External Triggers	Boolean	If true, requires an external trigger such as a script or animation event to increment and complete the reload. If false, uses a timer.
Reload Start Duration*	Float	The duration of the reload start animation.

NAME	TYPE	DESCRIPTION
Reload Increment Duration*	Float	The duration of a single increment of the reload.
Reload End Duration*	Float	The duration of the reload end.
Reload Anim Trigger	String	The AnimatorController trigger key for the reload animation.
Reload Anim Count Prop	String	The AnimatorController propery key for the reload count.
Reload Audio Start	AudioClip	The audio clip to play when the reload starts.
Reload Audio Increment	AudioClip	The audio clip to play during an increment of the reload.
Reload Audio End	AudioClip	The audio clip to play when the reload ends.

* This property is not visible unless the Use External Triggers property is set to False.

See Also

Modular Firearms

FirearmAnimEventsHandler

Unity Animator

Unity AnimatorController

InfiniteAmmo MonoBehaviour

Overview

The InfiniteAmmo module provides a limitless pool of ammo for reloader modules to reload from.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active reloader immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: **Component** means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. **GameObject** activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Printable Name	String	The name to show on the HUD.
Fixed Size	Integer	The ammo quantity available to any reloaders - must be >= to the magazine size.

See Also

[Modular Firearms](#)

InstantScopedAimer MonoBehaviour

Overview

The InstantScopedAimer behaviour instantly zooms the camera and switches to a HUD scope

Inspector

Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active aimer immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: **Component** means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. **GameObject** activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Aim Up Audio	AudioClip	An audio clip to play when the weapon is raised.
Aim Down Audio	AudioClip	An audio clip to play when the weapon is lowered.
Hip Accuracy Cap	Float	The highest accuracy the firearm can achieve while not aiming down sights.
Aimed Accuracy Cap	Float	The highest accuracy the firearm can achieve while aiming down sights.
Can Aim While Reloading	Boolean	Should the weapon be lowered when reloading or can it stay aimed.

NAME	TYPE	DESCRIPTION
On Aim Up	UnityEvent	An event called when the weapon is fully raised.
On Aim Down	UnityEvent	An event called when the weapon is fully lowered.
Hud Scope Key	String	The key for the specific scope to show.
Fov Multiplier	Float	A multiplier for the camera FoV for aim zoom.
Position Spring Multiplier	Float	A multiplier for procedural spring position animation on the weapon. Used to reduce movement when the firearm is close to the camera.
Rotation Spring Multiplier	Float	A multiplier for procedural spring rotation animation on the weapon. Used to reduce movement when the firearm is close to the camera.
Crosshair Up	FpsCrosshair	The crosshair to show when aiming down sights.
Crosshair Down	FpsCrosshair	The crosshair to show when not aiming down sights.

See Also

Modular Firearms

First Person Camera

HudScope

LaserPointerAimerSwitch MonoBehaviour

Overview

The LaserPointerAimerSwitch behaviour can be added to a laser pointer to give a simplified system for switching a weapon's aimer module when the laser is active. This can be used for holding weapons tilted and using the laser to aim.

Inspector
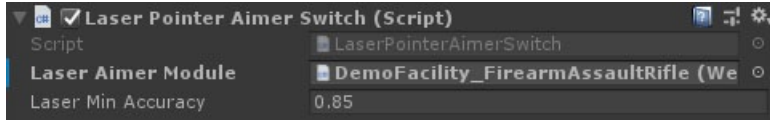
Properties

NAME	TYPE	DESCRIPTION
LaserAimerModule	Aimer	The aimer module associated with this laser pointer. This should be disabled on start.
LaserMinAccuracy	Float	The minimum accuracy of the firearm when the laser is switched on.

See Also

Modular Firearms

WieldableLaserPointer Behaviour

LaserTargetingSystem MonoBehaviour

Overview

The LaserTargetingSystem behaviour is paired with a WieldableLaserPointer to send the laser hit point to the registered guided projectiles as a target point. Switching the laser off or pointing it at the sky will clear the projectile's target. The guided projectile must use a TargetingSystemTracker component.

Inspector
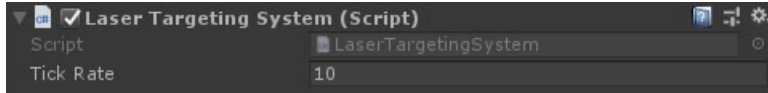
Properties

NAME	TYPE	DESCRIPTION
Tick Rate	Integer	The laser pointer hit location will be updated every nth fixed update frame.

See Also

Modular Firearms

TargetingSystemTracker Behaviour

WieldableLaserPointer Behaviour

Hitscan vs Projectiles

LineAndParticleHitscanTrail MonoBehaviour

Overview

The LineAndParticleHitscanTrail behaviour is used to display the trajectory of a hitscan shooter using both a line renderer and a particle system.

Inspector
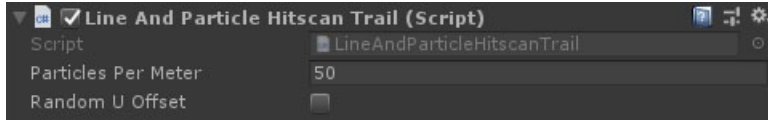
Properties

NAME	TYPE	DESCRIPTION
Particles Per Meter	Float	The number of particles per meter of trail. Used for the emit count to ensure a predictable density.
Random U Offset	Boolean	Randomise the line renderer texture's U mapping 0-1. Requires the material to have an **"_OffsetU"** property accessible via property block.

See Also

Modular Firearms

LineRendererHitscanTrail MonoBehaviour

Overview

The LineRendererHitscanTrail behaviour is used to display the trajectory of a hitscan shooter using a line renderer.

Inspector
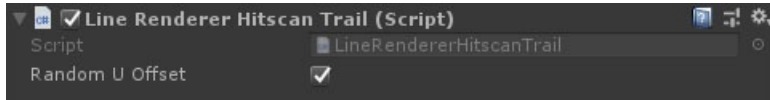
Properties

NAME	TYPE	DESCRIPTION
Random U Offset	Boolean	Randomise the texture's U mapping 0-1. Requires the material to have an **"_OffsetU"** property accessible via property block

See Also

Modular Firearms

MeleeWeapon MonoBehaviour

Overview

The MeleeWeapon behaviour is a hand held melee weapon like a club or knife.

Inspector
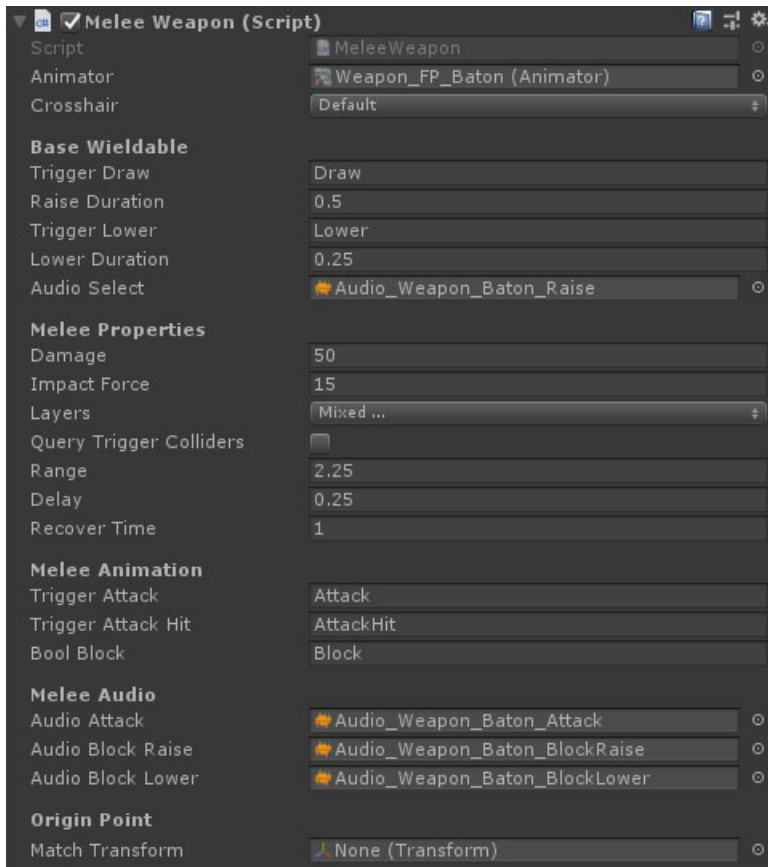
Properties

NAME	TYPE	DESCRIPTION
Animator	Animator	The animator component of the weapon.
Damage	Float	The damage the weapon does.
Impact Force	Float	The force to impart on the hit object. Requires either a Rigidbody or an impact handler on the hit object.
Layers	Float	The layers the attack will collide with.
Query Trigger Colliders	Boolean	Should the attack be tested against trigger colliders.
Range	Float	The range that the melee weapon can reach.
Delay	Float	The delay from starting the attack to checking for an impact. Should be synced with the striking point in the animation.
Recover Time	Float	The recovery time after a hit.

NAME	TYPE	DESCRIPTION
Crosshair	FpsCrosshair	The crosshair to show when the weapon is drawn.
Trigger Draw	String	The AnimatorController trigger key for the raise animation.
Raise Duration	Float	The time taken to raise the item on selection.
Trigger Lower	String	The AnimatorController trigger key for the weapon lower animation (blank = no animation).
Lower Duration	Float	The time taken to lower the item on deselection.
Trigger Attack	String	The AnimatorController trigger key for the attack animation.
Trigger Attack Hit	String	The AnimatorController trigger key for the attack hit animation.
Bool Block	String	The AnimatorController bool key for the block animation.
Audio Select	AudioClip	The audio clip when raising the weapon.
Audio Attack	AudioClip	The audio clip when attacking.
Audio Block Raise	AudioClip	The audio clip when bringing the weapon into block position.
Audio Block Lower	AudioClip	The audio clip when bringing the weapon out of block position.

Origin Point

The NeoFPS weapons assume that the camera is placed at the origin. For many assets or 3rd party weapons, the origin is at the character feet or hips and the camera is a child object of the weapon's hierarchy. To align the object to correctly work with NeoFPS you can drag the camera object of the weapon into the **Match Transform** field under the **Origin Point** heading. This will move everything below the spring object to match up.

See Also

Melee Weapons

Unity Animator

Unity AnimatorController

MeleeWieldableStanceManager MonoBehaviour

Overview

The MeleeWieldableStanceManager behaviour is used to specify poses or stances for a melee weapon. The entire weapon will be moved to match the stance, and can optionally set an animator bool parameter too.

The weapon will exit the stance temporarily while attacking or blocking.

Inspector
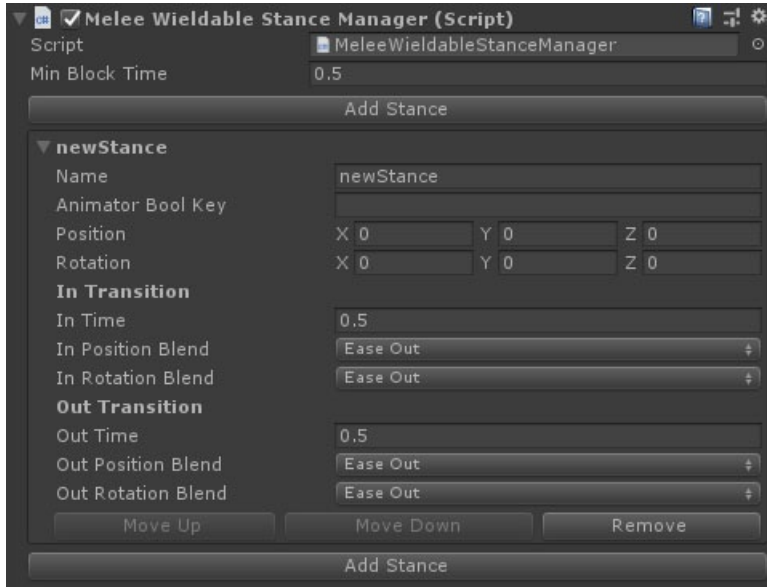
Properties

NAME	TYPE	DESCRIPTION
Min Attack Time	Float	The minimum amount of time the stance will be temporarily exited for an attack.
Min Block Time	Float	The minimum amount of time the stance will be temporarily exited when blocking (prevents tapping block).

Use the **Add Stance** buttons to add a new stance to the manager.

Stances

Individual stances have the following properties:

NAME	TYPE	DESCRIPTION
Name	String	The name of the stance.
Animator Bool Key	String	An optional name of a bool parameter in the weapon's Animator.
Position	Vector	The position to move the weapon to in this stance.
Rotation	Vector	The rotation of the weapon in this stance.
In Position Blend	Dropdown	The easing method for blending between the source position and stance position on entering the stance. Options are: **Lerp, **EaseIn**, **EaseOut**, **EaseInOut**, **SwingAcross**, **SwingUp**, **Spring**, **Bounce** and **Overshoot**.

NAME	TYPE	DESCRIPTION
In Rotation Blend	Dropdown	The easing method for blending between the source rotation and stance rotation on entering the stance. Options are: **Lerp, **Slerp**, **EaseIn**, **EaseOut**, **EaseInOut**, **Spring**, **Bounce**, **Overshoot**.
In Time	Float	The time taken to enter the stance.
Out Position Blend	Dropdown	The easing method for blending between the stance position and idle position on exiting the stance. Options are: **Lerp, **EaseIn**, **EaseOut**, **EaseInOut**, **SwingAcross**, **SwingUp**, **Spring**, **Bounce** and **Overshoot**.
Out Rotation Blend	Dropdown	The easing method for blending between the stance rotation and Idle rotation on exiting the stance. Options are: **Lerp, **Slerp**, **EaseIn**, **EaseOut**, **EaseInOut**, **Spring**, **Bounce**, **Overshoot**.
Out Time	Float	The time taken to enter the stance.

See Also

Melee Weapons

ModularAssaultRifle MonoBehaviour

Overview

The ModularAssaultRifle behaviour is an example of adding functionality to the Modular Firearm. Use the switch modes input button (defaults to middle mouse button) to cycle through the trigger modules attached to the weapon, acting like a fire select.

Inspector
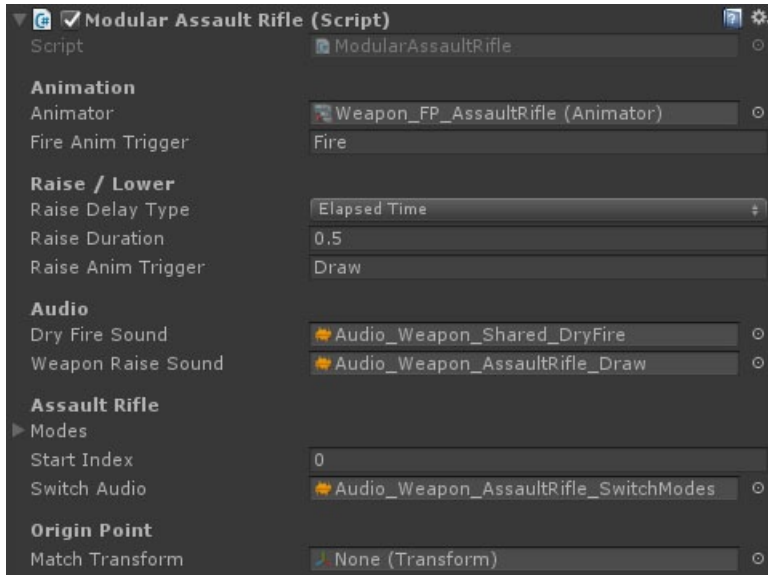
Properties

NAME	TYPE	DESCRIPTION
Modes	Fire Mode Array	The different fire modes as outlined below.
Start Index	Int	Which fire mode should be used on start.
Switch Audio	AudioClip	The audio clip to play when the weapon mode is switched.

Fire Mode

NAME	TYPE	DESCRIPTION
Name	String	The name of the fire mode for the HUD.
Trigger	Trigger Module	The trigger to activate.

See Also

Modular Firearms

ModularFirearm MonoBehaviour

Overview

The ModularFirearm behaviour manages all of the modules that make up a firearm.

Inspector
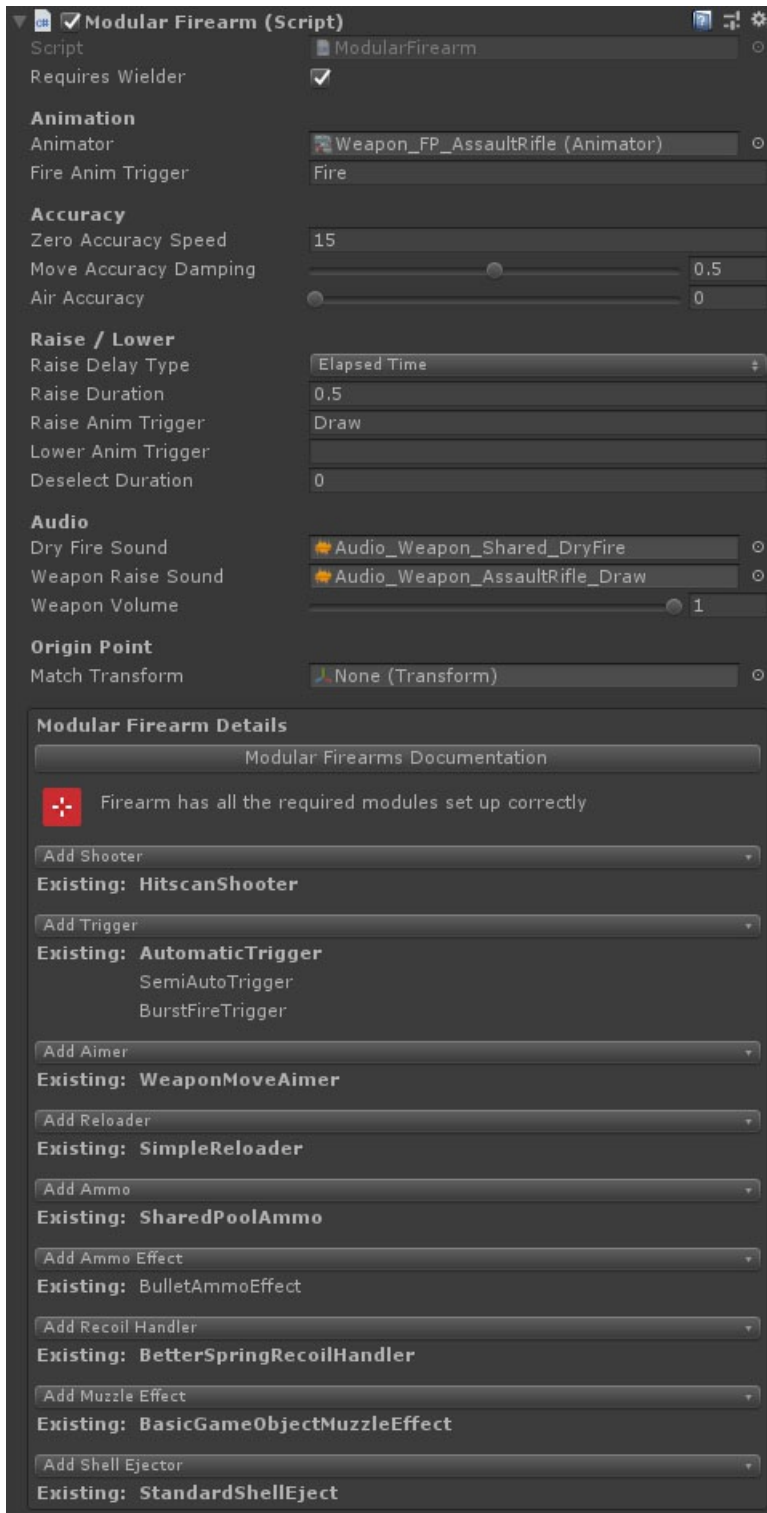
Properties

NAME	TYPE	DESCRIPTION
Animator	Animator	The weapon geometry animator (optional).

NAME	TYPE	DESCRIPTION
Fire Anim Trigger	String	The AnimatorController trigger key for the fire animation (blank = no animation).
Raise Delay Type	Dropdown	The delay type for raising the weapon (you can use FirearmAnimEventsHandler to tie delays to animation). Options are **None**, **Elapsed Time**, **External Trigger**.
Raise Duration	Float	The duration in seconds for raising the weapon.
Raise Anim Trigger	String	The AnimatorController trigger key for the weapon raise animation (blank = no animation).
Lower Anim Trigger	String	The AnimatorController trigger key for the weapon lower animation (blank = no animation).
Deselect Duration	Float	The time taken to lower the item on deselection.
Dry Fire Sound	AudioClip	The audio clip to play if attempting to fire while empty.
Weapon Raise Sound	AudioClip	The audio clip to play when the weapon is drawn.

Origin Point

The NeoFPS weapons assume that the camera is placed at the origin. For many assets or 3rd party weapons, the origin is at the character feet or hips and the camera is a child object of the weapon's hierarchy. To align the object to correctly work with NeoFPS you can drag the camera object of the weapon into the **Match Transform** field under the **Origin Point** heading. This will move everything below the spring object to match up.

Modular Firearm Details

The modular firearm details section assists in setting up a modular firearm. It displays the current state of the firearm along with the attached modules, and allows you to quickly add modules to the firearm.

Quick Setup

When the ModularFirearm behaviour is first added to an object, the details section will display options for quick setup. This is only available if the object is placed in a scene. If it is accessed as a prefab in the project hierarchy, then these options will not appear because of issues with modifying prefab object heirarchies via scripts.
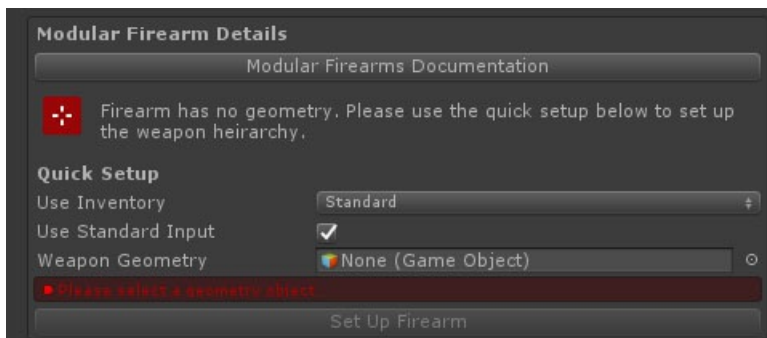

This is used to set up the firearm hierarchy and attach render geometry. It has the following properties:

NAME	TYPE	DESCRIPTION
Use Inventory	Dropdown	Which of the inventory implementations do you want the firearm to use. Choosing the **Custom** option does not add an inventory wieldable behaviour, meaning that you will need to add your own solution.
Use Standard Input	Boolean	Should the firearm use the standard NeoFPS input handler. Switch this off if you wan to add your own custom input handler.
Weapon Geometry	GameObject	This value is a prefab or model that you want to use to represent the first person weapon. If the object does not contain a mesh renderer or skinned mesh renderer in its hierarchy then it will cause an error. There is also a warning if the object has no animator in its hierarchy, though this is not compulsary.

With a valid weapon geometry object selected, the **Set Up Firearm** button will be available. Pressing this will create a spring object set up with procedural animation behaviours for handling things like recoil and weapon bob, and then will copy the weapon geometry under this.

Modules

If the firearm already has child objects then the details section will instead show the modules options:
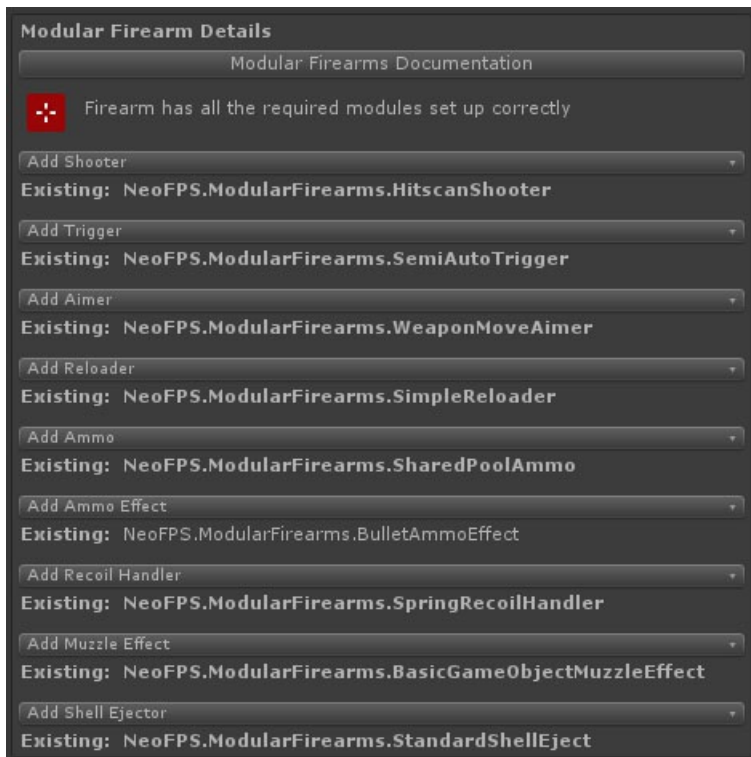
To add a firearm module click the relevant dropdown button and select the desired module from the menu. If a module is required for a firearm to operate then the info section at the top of the modular firearm details will display an error if that module is not attached. The firearm can also have multiple modules of each type, though only one should be set to start active.

Each of the attached modules can also report back any errors to the firearm they are attached to, and broken modules will be displayed as red in this list.

See Also

FirearmAnimEventsHandler

Additive Transform And Effects

Unity Animator

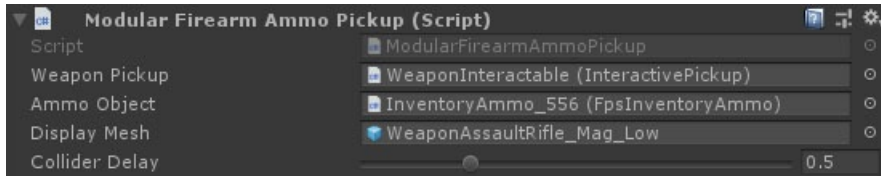# ModularFirearmAmmoPickup MonoBehaviour

Overview

The ModularFirearmAmmoPickup behaviour is used in tandem with the ModularFirearmDrop when a character drops the weapon they are holding. It takes the magazine ammo count from the weapon as it is dropped, so that no extra ammo is spawned or lost.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Weapon Pickup	InteractablePickup	The pickup attached to the weapon drop.
Ammo Object	FpsInventoryAmmo	The ammo inventory item.
Display Mesh	GameObject	The display mesh for the weapon magazine. Will be hidden when the ammo is collected.
Collider Delay	Float	The delay before the ammo collider is enabled. Prevents the character dropping the weapon collecting the ammo immediately.

See Also

Modular Firearms

ModularFirearmDrop

FpsInventoryAmmo

ModularFirearmDrop MonoBehaviour

Overview

The ModularFirearmDrop is spawned when a character drops their current firearm. It acts as an interactive pickup for other characters to pick it up, and manages the magazine ammo as a separate pickup.

Inspector
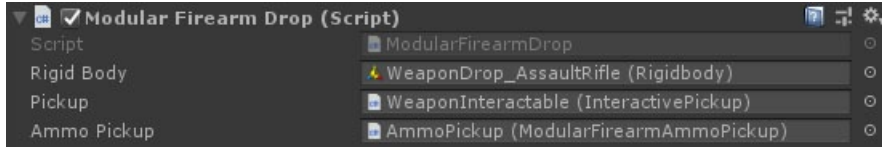
Properties

The ModularFirearmDrop inherits from the FpsInventoryWieldableDrop. Check the reference for information on its properties.

NAME	TYPE	DESCRIPTION
Ammo Pickkup	ModularFirearmAmmoPickup	The ammo pickup for this weapon's magazine.

See Also

Modular Firearms

ModularFirearmAmmoPickup

InteractivePickup

ModularFirearmModeSwitcher MonoBehaviour

Overview

The ModularFirearmModeSwitcher behaviour is attached to modular firearms to handle mode switch commands by enabling specific firearm modules or components.

Since only one of each module type can be enabled on a firearm at any one time, when the mode switcher enables one, the previous is automatically disabled. This does not apply to components which are not firearm modules.

Inspector
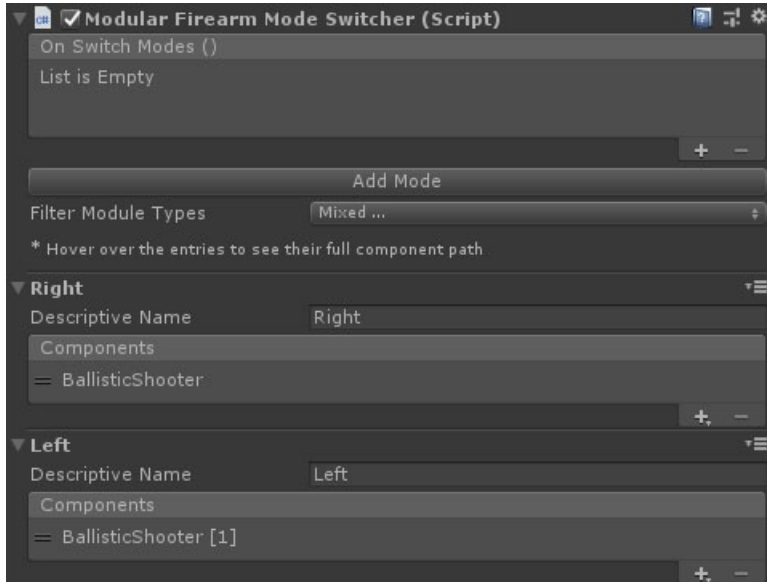
Properties

NAME	TYPE	DESCRIPTION
On Switch Modes	UnityEvent	An event fired whenever the firearm mode is switched.

The **Add Mode** button adds a new blank mode to the component. Each mode has the following properties:

NAME	TYPE	DESCRIPTION
Descriptive Name	String	The name displayed on the HUD for this mode. Leave this blank if you do not want the mode name displayed.
Components	Component Array	The components to enable for this mode (for firearm module components, this will disable the old component automatically).

See Also

Modular Firearms

MultiObjectSwapEjector MonoBehaviour

Overview

The MultiObjectSwapEjector behaviour replaces and disables multiple animated objects with pooled bullet casing objects.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active ejector immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: **Component** means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. **GameObject** activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Target Transforms	Transform Array	The transforms of the objects in the firearm heirachy to swap out.
Shell Prefab	PooledObject	The bullet casing prefab object to spawn.
Eject On Fire	Boolean	Should the ejector start the ejection process immediately when the gun is fired or wait until triggered by an animation event or other.
Swap Inactive	Boolean	If any of the target transforms are inactive, should they be swapped or ignored.
Delay	Float	The delay time between starting the ejection process and the objects being swapped

See Also

Modular Firearms

PooledObject

MultiTargetLockTrigger MonoBehaviour

Overview

The MultiTargetLockTrigger module charges up while the fire button is held down and fires once it hits full charge.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active trigger immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: **Component** means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. **GameObject** activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Charge Duration	Float	How long does it take to charge the trigger.
Uncharge Duration	Float	How long does it take to uncharge the trigger, assuming it hasn't gone off.
Repeat	Boolean	Once the shot is fired, start charging the next shot if this is true.
Repeat Delay	Float	The time between a shot firing and starting charging the next shot.
Audio Source	AudioSource	The source to play the audio from (needs its own as it must be interrupted and seeked).
Trigger Audio Charge	AudioClip	The audio clip to play on charge.
Trigger Audio Release	AudioClip	The audio clip to play on release.

See Also

Modular Firearms

NearestObjectWithTagTracker MonoBehaviour

Overview

The NearestObjectWithTagTracker behaviour is a guided projectile tracker that checks its vicinity for objects with the correct layer and tag, then tracks the closest one.

Inspector
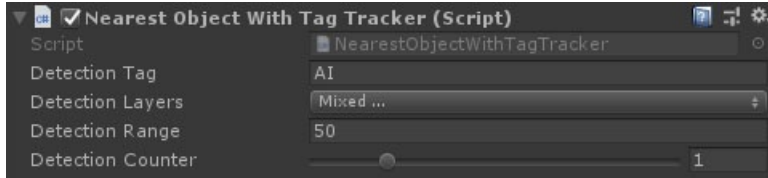
Properties

NAME	TYPE	DESCRIPTION
Detection Tag	String	The object tag to home in on.
Detection Layers	LayerMask	The layers to check for colliders on.
Detection Range	Float	The max distance for targets to home in on.
Detection Counter	Float	The time between searching for targets.

See Also

Modular Firearms

Hitscan vs Projectiles

NoisyLineHitscanTrail MonoBehaviour

Overview

The NoisyLineHitscanTrail behaviour is used to display the trajectory of a hitscan shooter using a line renderer. The points of the line are driven by a particle system which applies noise over time, making the line drift over time.

Inspector
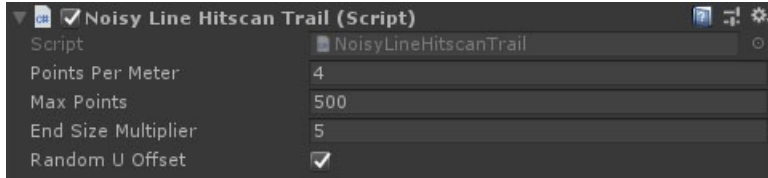
Properties

NAME	TYPE	DESCRIPTION
Points Per Meter	Float	The number of particles per meter of trail. Used for the emit count to ensure a predictable density.
Max Points	Integer	The maximum number of points for the line.
End Size Multiplier	Float	The line width once the trail dies.
Random U Offset	Float	Randomise the texture's U mapping 0-1. Requires the material to have an **"_OffsetU"** property accessible via property block

See Also

Modular Firearms

ObjectSwapEjector MonoBehaviour

Overview

The ObjectSwapEjector behaviour replaces and disables an animated object with a pooled bullet casing object.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active ejector immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: **Component** means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. **GameObject** activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Target Transform	Transform	The transform of an object in the firearm heirachy to swap out.
Shell Prefab	PooledObject	The bullet casing prefab object to spawn.
Eject On Fire	Boolean	Should the ejector start the ejection process immediately when the gun is fired or wait until triggered by an animation event or other.
Delay	Float	The delay time between starting the ejection process and the object being swapped

See Also

Modular Firearms

PooledObject

ParticleAmmoEffect MonoBehaviour

Overview

The ParticleAmmoEffect behaviour spawns a pooled particle system object on impact.

Inspector
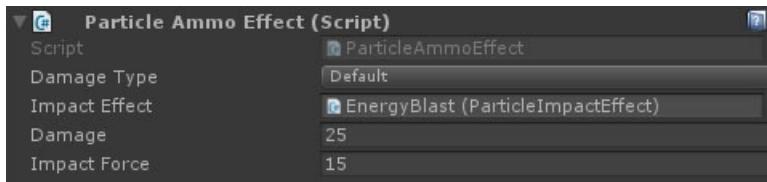
Properties

NAME	TYPE	DESCRIPTION
Impact Effect	ParticleImpactEffect	The object to spawn at the impact location.
Damage	Float	The damage the bullet does.
Impact Force	Float	The force to be imparted onto the hit object. Requires either a Rigidbody or an impact handler.

See Also

Modular Firearms

ParticleImpactEffect

Unity Rigidbody

ParticleSystemHitscanTrail MonoBehaviour

Overview

The ParticleSystemHitscanTrail behaviour uses a particle system to spawn particles along a hitscan bullet trajectory.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Particles Per Meter	Float	The number of particles per meter of trail. Used for the emit count to ensure a predictable density.

See Also

Modular Firearms

ParticleImpactEffect MonoBehaviour

Overview

The ParticleImpactEffect behaviour manages the lifecycle of a pooled particle object that is spawned on a bullet hit by the ParticleAmmoEffect behaviour.

Inspector
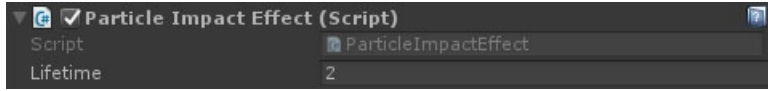
Properties

NAME	TYPE	DESCRIPTION
Lifetime	Float	Duration the object should remain active before being returned to the pool.

See Also

Modular Firearms

ParticleAmmoEffect

Unity Rigidbody

ParticleSystemShellEject MonoBehaviour

Overview

The ParticleSystemShellEject module triggers one or more particle systems on each shot.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active ejector immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: **Component** means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. **GameObject** activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Particle Systems	ParticleSystem Array	The particle systems to play when a shell is ejected.
Delay Type	Dropdown	The delay type between firing and ejecting a shell. Options are **None**, **Elapsed Time**, **External Trigger**.
Delay*	Float	The delay time between firing and ejecting a shell.

* This property will only be visible if the delay type is set to Elapsed Time

---

See Also

Modular Firearms

PooledObject

ParticleToRigidbodyShellEject MonoBehaviour

Overview

The ParticleToRigidbodyShellEject module triggers one or more particle systems on each shot.

Inspector

Properties

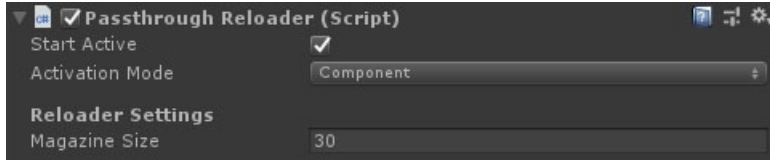| NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active ejector immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: **Component** means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. **GameObject** activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Particle System	ParticleSystem	The particle system to play when a shell is ejected.
Rigidbody Prefab	PooledObject	The pooled rigidbody to swap the particles for once out of view.
Max Particles	Integer	The maximum number of particles that can be visible.
Delay Type	Dropdown	The delay type between firing and ejecting a shell. Options are **None**, **Elapsed Time**, **External Trigger**.
Delay	Float	The delay time between firing and ejecting a shell. This property will only be visible if the delay type is set to **Elapsed Time**

See Also

Modular Firearms

PooledObject

PassthroughReloader MonoBehaviour

Overview

The PassthroughReloader module has no magazine size, but pulls ammo directly from the firearm's ammo module with every shot.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active reloader immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: **Component** means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. **GameObject** activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Magazine Size	Integer	The value it should report for its magazine size if any modules query it.

See Also

[Modular Firearms](#)

PatternBallisticShooter MonoBehaviour

Overview

The PatternBallisticShooter module instantly fires in a preset pattern. The pattern is defined in 2D with a max distance of 1m from the origin on each axis. The distance setting specifies how far from the firearm's muzzle the full pattern size is achieved.
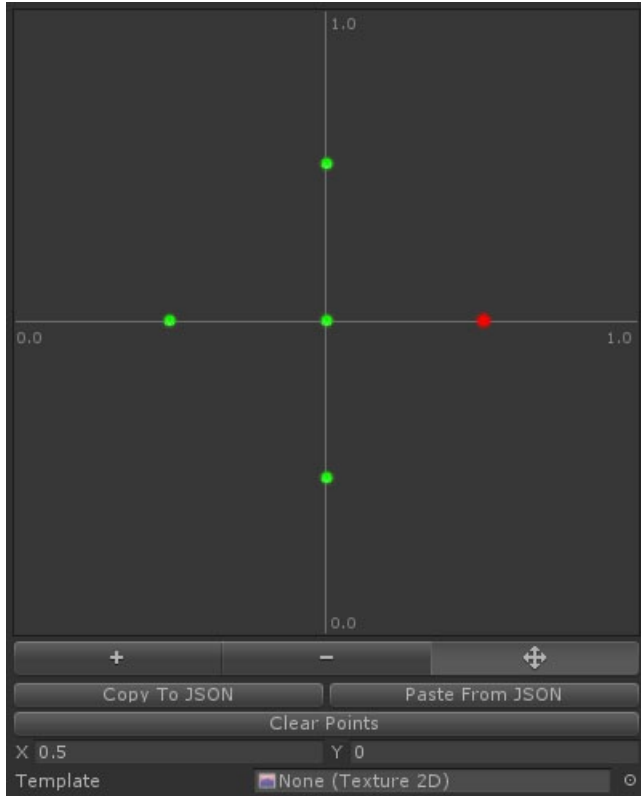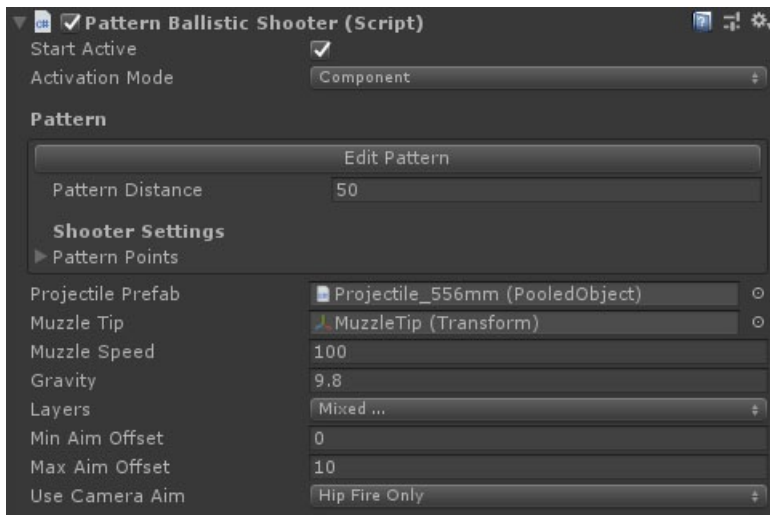
Pattern Editor

The pattern editor allows you to specify the gunshot pattern. The **+**, **-** and move buttons switch the editor mode. In **+** mode, clicking will add a new point. In **-** mode, clicking will remove the clicked point. In **move** mode, you can select a point by clicking it, and move a point by click-dragging it. The editor also has the following controls:

CONTROL	DESCRIPTION
Copy To JSON	Copy all points to the clipboard in JSON format. You can paste this to any text editor to make changes.
Paste From JSON	If the clipboard contains valid JSON, replace all the points with those on the clipboard.
Clear Points	Remove all points from the pattern.
Position	The position of the selected point (clamped to a -1 to 1 range on each axis).
Template	Load a background image to use as a template for placing points.

Inspector

Properties

| NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active shooter immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: **Component** means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. **GameObject** activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Pattern Distance	Float	The distance from the muzzle tip at which the pattern will match the diagram.
Pattern Points	Vector2 Array	The points that make up the pattern.
Projectile Prefab	PooledObject	The projectile to spawn.
Muzzle Tip	Transform	The transform that the bullet actually fires from.
Muzzle Speed	Float	The speed of the projectile.
Gravity	Float	The gravity for the projectile.
Layers	LayerMask	The physics collision layers the shot can hit.
Min Aim Offset	Float	The maximum angle from forward the shooter can fire when accuracy is 1.
Max Aim Offset	Float	The maximum angle from forward the shooter can fire when accuracy is 0.
Use Camera Aim	Dropdown	When set to use camera aim, the gun first casts from the FirstPersonCamera's aim transform, and then from the muzzle tip to that point to get more accurate firing. Options are: **HipFireOnly**, **HipAndAimDownSights**, **AimDownSightsOnly**, **Never**.

See Also

PatternHitscanShooter MonoBehaviour

Overview

The PatternHitscanShooter module instantly fires in a preset pattern. The pattern is defined in 2D with a max distance of 1m from the origin on each axis. The distance setting specifies how far from the firearm's muzzle the full pattern size is achieved.
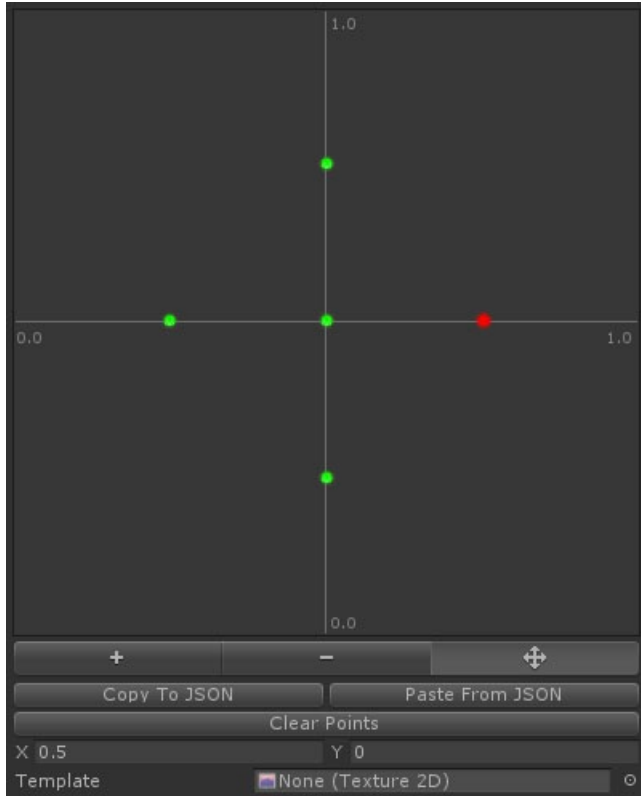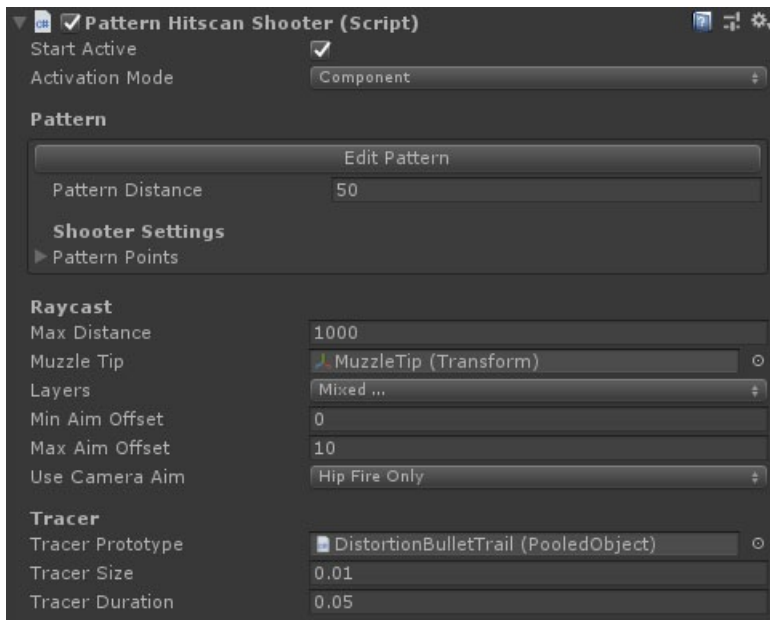
Pattern Editor

The pattern editor allows you to specify the gunshot pattern. You can select a point by left clicking and move a point by left-click dragging.

CONTROL	DESCRIPTION
Template	Load a background image to use as a template for placing points.
Copy To JSON	Copy all points to the clipboard in JSON format. You can paste this to any text editor to make changes.
Paste From JSON	If the clipboard contains valid JSON, replace all the points with those on the clipboard.
Add New Point	Add a new point to the pattern (the new points will be placed at the center and selected automatically.
Remove Point	Remove the selected point.
Position	The position of the selected point (clamped to a -1 to 1 range on each axis).

Inspector

Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active shooter immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: **Component** means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. **GameObject** activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Pattern Distance	Float	The distance from the muzzle tip at which the pattern will match the diagram.
Pattern Points	Vector2 Array	The points that make up the pattern.
Max Distance	Float	The maximum distance that the weapon will register a hit.
Muzzle Tip	Transform	The transform that the bullet actually fires from.
Layers	LayerMask	The physics collision layers the shot can hit.
Min Aim Offset	Float	The maximum angle from forward the shooter can fire when accuracy is 1.
Max Aim Offset	Float	The maximum angle from forward the shooter can fire when accuracy is 0.
Use Camera Aim	Dropdown	When set to use camera aim, the gun first casts from the FirstPersonCamera's aim transform, and then from the muzzle tip to that point to get more accurate firing. Options are: **HipFireOnly**, **HipAndAimDownSights**, **AimDownSightsOnly**, **Never**.
Tracer Prototype	Pooled Object	The optional pooled tracer prototype to use (must implement the IPooledHitscanTrail interface).

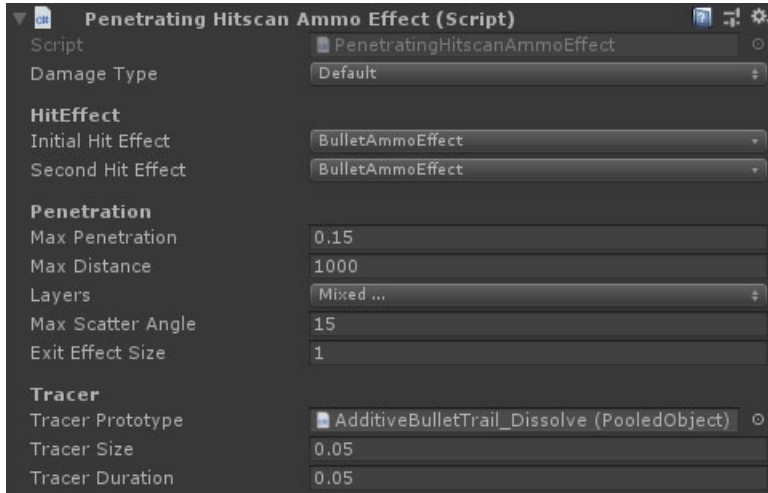| NAME	TYPE	DESCRIPTION
Tracer Size	Float	The size (thickness/radius) of the tracer line.
Tracer Duration	Float	How long should the tracer object stay visible.

See Also

Modular Firearms

PenetratingHitscanAmmoEffect MonoBehaviour

Overview

The PenetratingHitscanAmmoEffect module allows projectiles to bounce off surfaces based on the impact angle and the distance travelled. If you are using a projectile shooter, then there is a separate PenetratingProjectileAmmoEffect that should be used instead of this.

This ammo effect is also used alongside another effect to apply damage and visual elements.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Damage Type	DamageType	The type of damage the weapon should do with this ammo.
Initial Hit Effect	BaseAmmoEffect	The effect of the ammo when it first hits.
Second Hit Effect	BaseAmmoEffect	The effect of the ammo after it has penetrated something.
Max Penetration	Float	The maximum thickness of object the bullet can penetrate.
Max Distance	Float	The maximum distance that the weapon will register a hit (includes the distance travelled up to the penetration).
Layers	LayerMask	The layers bullets will collide with.
Max Scatter Angle	Float	Randomises the deflected bullet direction within this cone angle.
Exit Effect Size	Float	Uses the surface system to show a bullet hit effect on exit. Set this to zero if you don't want it to happen.
Tracer Prototype	PooledObject	The optional pooled tracer prototype to use (must implement the IPooledHitscanTrail interface)
Tracer Size	Float	How size (thickness/radius) of the tracer line.

NAME	TYPE	DESCRIPTION
Tracer Duration	Float	How long the tracer line will last.

See Also

Modular Firearms

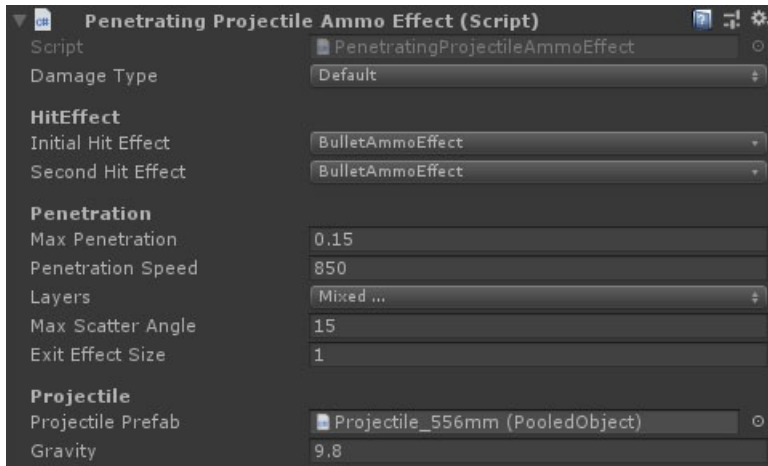# PenetratingProjectileAmmoEffect MonoBehaviour

Overview

The PenetratingProjectileAmmoEffect module allows projectiles to penetrate through surfaces based on the impact angle and the speed the projectile is travelling. A new projectile will be spawned on the other side of the surface, with its speed and direction altered based on the impact.

If you are using a hitscan shooter, then there is a separate PenetratingHitscanAmmoEffect that should be used instead of this.

This ammo effect is also used alongside another effect to apply damage and visual elements.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Damage Type	DamageType	The type of damage the weapon should do with this ammo.
Initial Hit Effect	BaseAmmoEffect	The effect of the ammo when it first hits.
Second Hit Effect	BaseAmmoEffect	The effect of the ammo after it has penetrated something.
Max Penetration	Float	The maximum thickness of object the bullet can penetrate.
Penetration Speed	Float	The speed at which max penetration will be reached. The speed will also be clamped to this so that (for example) hitscan weapons don't have infinite speed.
Layers	LayerMask	The layers bullets will collide with.
Max Scatter Angle	Float	Randomises the deflected bullet direction within this cone angle.
Exit Effect Size	Float	Uses the surface system to show a bullet hit effect on exit. Set this to zero if you don't want it to happen.
Projectile Prefab	PooledObject	The projectile to spawn.

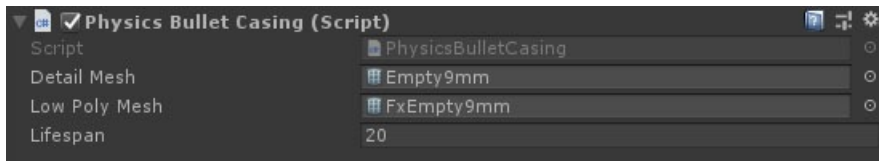| NAME	TYPE	DESCRIPTION
Gravity	Float	The gravity for the projectile.

See Also

Modular Firearms

PhysicsBulletCasing MonoBehaviour

Overview

The PhysicsBulletCasing behaviour is a bullet casing that can be ejected from firearms and will physically react with the world.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Detail Mesh	Mesh	The detail mesh to show while the bullet is in the first person view.
Low Poly Mesh	Mesh	The low poly mesh to switch to when not in the first person view.
Lifespan	Float	How long should the casing remain before being returned to the pool.

See Also

Modular Firearms

PooledObject

Unity Rigidbody

PlayAudioToolAction MonoBehaviour

Overview

The PlayAudioToolAction behaviour is used to play and audio clip on the start or end of a wieldable tool's actions.

Inspector
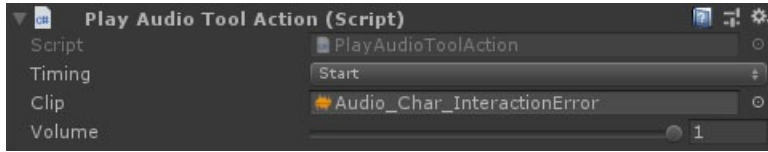
Properties

NAME	TYPE	DESCRIPTION
Timing	Dropdown	When should the clip be played.
Clip	AudioClip	The clip to play.
Volume	Float	The volume for the clip.

See Also

Wieldable Tools

PlayerTracker MonoBehaviour

Overview

The PlayerTracker behaviour is a guided projectile tracker component which directs the projectile towards the current player character. Definitely more useful on enemy weapons than player weapons.

Inspector
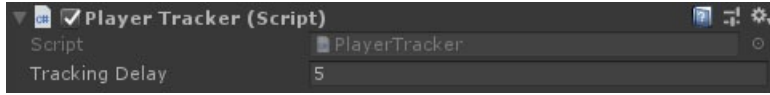
Properties

NAME	TYPE	DESCRIPTION
Tracking Delay	Float	The time from firing to starting to steer towards the player character.

See Also

Modular Firearms

Hitscan vs Projectiles

PooledExplosion MonoBehaviour

Overview

The PooledExplosion behaviour manages the lifecycle of a pooled particle object, and handles applying damage and force to items in its area of effect.

Inspector
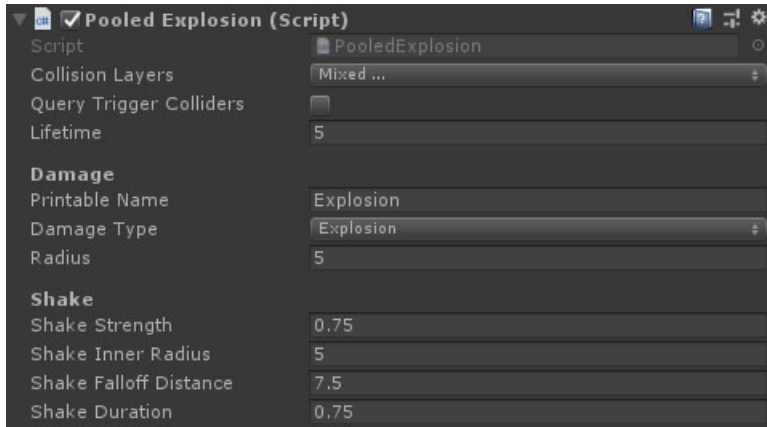
Properties

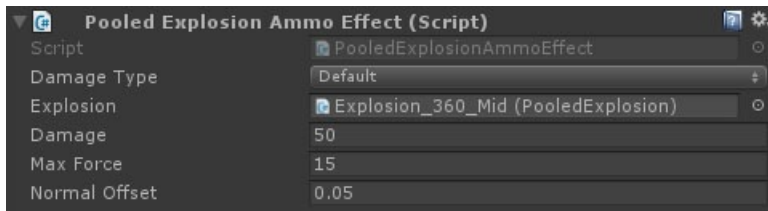| NAME	TYPE	DESCRIPTION
Collision Layers	Float	The valid collision layers the explosion will affect.
Query Trigger Colliders	Boolean	Should the explosion be tested against trigger colliders.
Lifetime	Float	Duration the object should remain active before being returned to the pool.
Printable Name	Float	A description of the damage, used when logging and displaying game events.
Damage Type	Float	The damage type the explosion applies (enables filtering damage types).
Radius	Float	The radius of the explosion.
Shake Strength	Float	The strength of the camera (and other) shake due to the explosion.
Shake Inner Radius	Float	The inner shake radius of the explosion. Any shake handlers within this radius will be affected at full strength, falling off to 0 outside this based on the falloff distance.
Shake Falloff Distance	Float	The distance beyond the inner radius where the shake effect drops off to 0.
Shake Duration	Float	The duration of the shake effect.

See Also

Explosions

PooledExplosionAmmoEffect MonoBehaviour

Overview

The PooledExplosionAmmoEffect spawns a PooledExplosion and passes it the relevant data to inflict damage and add force.

Inspector

Properties

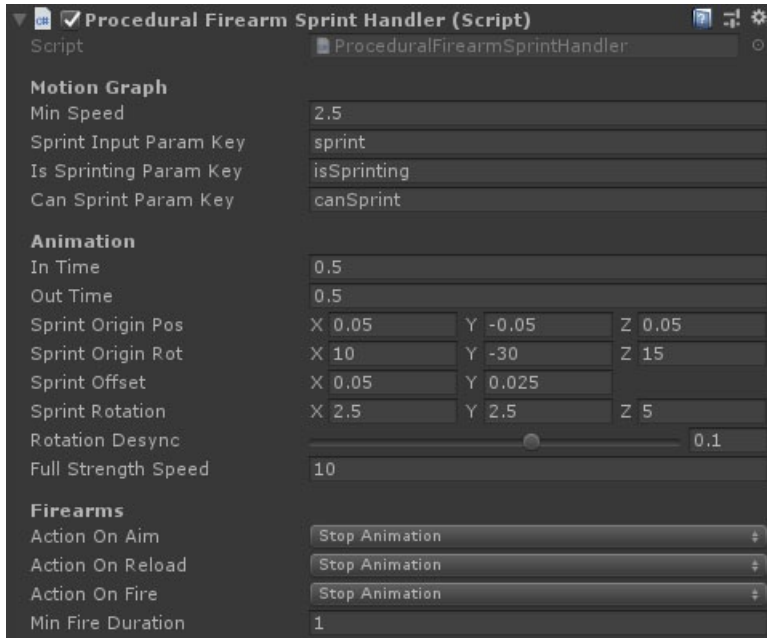| NAME	TYPE	DESCRIPTION
Explosion	PooledExplosion	The explosion object to spawn at the impact location.
Damage	Float	The damage does at the center of the explosion. Drops off to zero at the edge of the radius.
Max Force	Float	The maximum force to be imparted onto any objects in the explosion's radius. Requires either a Rigidbody or an impact handler and drops off to zero the further from the center the object is.
Normal Offset	Float	An offset from the hit point along its normal to spawn the explosion. Prevents the explosion from appearing embedded in the surface.

See Also

Modular Firearms

Pooled Explosion

PooledExplosionSpawner MonoBehaviour

Overview

The PooledExplosionSpawner .

Inspector

Properties

NAME	TYPE	DESCRIPTION
Explosion	PooledExplosion	The explosion object to spawn.
Offset	Vector3	An offset from the object's origin that the explosion will spawn.
Damage	Float	The damage does at the center of the explosion. Drops off to zero at the edge of the radius.
Max Force	Float	The maximum force to be imparted onto any objects in the explosion's radius. Requires either a Rigidbody or an impact handler and drops off to zero the further from the center the object is.

See Also

Explosions

PooledExplosion

ProceduralFirearmSprintHandler MonoBehaviour

Overview

The AnimatedFirearmSprintHandler behaviour is attached to a modular firearm to model different sprinting mechanics. This version uses procedural animation to move the firearm into a sprint pose and add an additive transform bob effect.

Inspector

Properties

Motion Graph

NAME	TYPE	DESCRIPTION
Min Speed	Float	The minimum speed the character must be moving for the sprint animation to play.
Sprint Input Param Key	String	(Required) The switch parameter on the motion graph which is set by the input handler to tell the character when to sprint.
Is Sprinting Param Key	String	(Required) A switch parameter on the motion graph which the graph sets when the character is sprinting.
Can Sprint Param Key	String	(Optional) A switch parameter on the motion graph which tells the character if it can sprint or not.

Animation

NAME	TYPE	DESCRIPTION
In Time	Float	The time taken to blend into the sprint animation.
Out Time	Float	The time taken to blend out of the sprint animation to idle.
Sprint Origin Pos	Vector	The neutral weapon / item position in sprint pose before bob is applied.

NAME	TYPE	DESCRIPTION
Sprint Origin Rot	Vector	The neutral weapon / item rotation in sprint pose before bob is applied.
Sprint Offset	Vector	The peak position offset of the step cycle on the z and y axes (z does not change).
Sprint Rotation	Vector	The peak rotation of the step cycle on each axis.
Rotation Desync	Float	The offset in terms of one full step cycle (left and right) for the timing of the rotation. Positive means after the position, Negative means before.
Full Strength Speed	Float	The speed at which the full sprint animation strength is reached. This fades out the rotation aspect as the character slows down.

Firearms

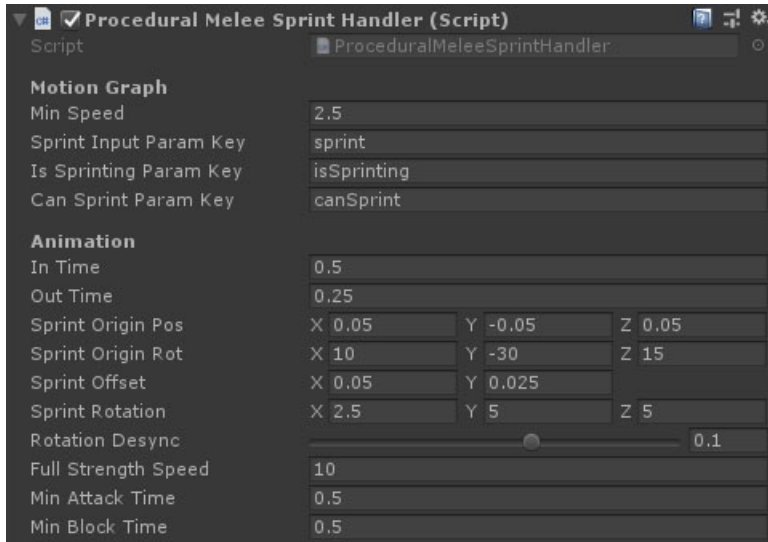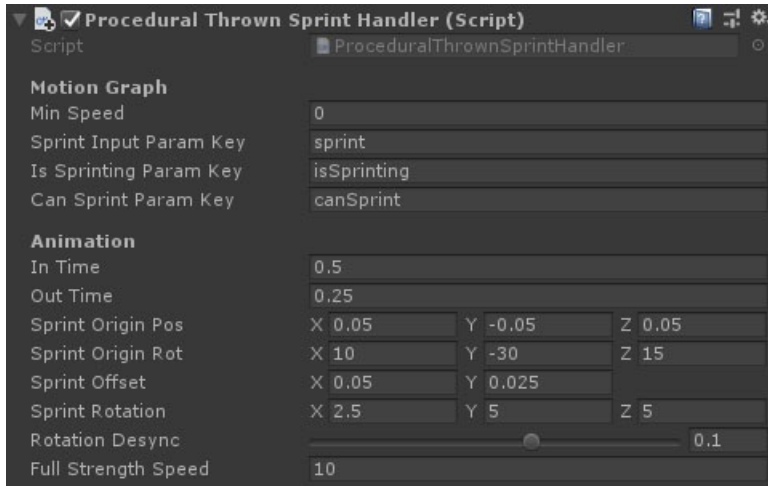| NAME	TYPE	DESCRIPTION
Action On Aim	Dropdown	What to do when the firearm enters / exits ADS. **Stop Sprinting** completely stops the character from sprinting until they leave ADS. **Stop Animation** stops the firearm sprint animation without affecting the character movement.
Action On Reload	Dropdown	What to do when the firearm is reloaded. **Stop Sprinting** completely stops the character from sprinting until the reload animation completes. **Stop Animation** stops the firearm sprint animation without affecting the character movement.
Action On Fire	Dropdown	What to do when the firearm trigger is pulled while sprinting. **Stop Sprinting** completely stops the character from sprinting until the weapon stops firing. **Stop Animation** stops the firearm sprint animation without affecting the character movement (both of these have a slight delay before firing to allow the weapon to be aligned). **Cannot Fire** means that the firearm trigger is blocked until the character stops sprinting.
Min Fire Duration	Float	The minimum amount of time the firearm sprint animation will be paused or sprinting blocked when the trigger is pulled. Prevents rapid tapping of the trigger popping in and out of sprint.

See Also

Modular Firearms

The Motion Graph

Additive Transforms & Effects

ProceduralMeleeSprintHandler MonoBehaviour

Overview

The ProceduralMeleeSprintHandler behaviour is attached to a melee weapon to model different sprinting mechanics. This version uses procedural animation to move the firearm into a sprint pose and add an additive transform bob effect.

Inspector

Properties

Motion Graph

NAME	TYPE	DESCRIPTION
Min Speed	Float	The minimum speed the character must be moving for the sprint animation to play.
Sprint Input Param Key	String	(Required) The switch parameter on the motion graph which is set by the input handler to tell the character when to sprint.
Is Sprinting Param Key	String	(Required) A switch parameter on the motion graph which the graph sets when the character is sprinting.
Can Sprint Param Key	String	(Optional) A switch parameter on the motion graph which tells the character if it can sprint or not.

Animation

NAME	TYPE	DESCRIPTION
In Time	Float	The time taken to blend into the sprint animation.
Out Time	Float	The time taken to blend out of the sprint animation to idle.
Sprint Origin Pos	Vector	The neutral weapon / item position in sprint pose before bob is applied.
Sprint Origin Rot	Vector	The neutral weapon / item rotation in sprint pose before bob is applied.

NAME	TYPE	DESCRIPTION
Sprint Offset	Vector	The peak position offset of the step cycle on the z and y axes (z does not change).
Sprint Rotation	Vector	The peak rotation of the step cycle on each axis.
Rotation Desync	Float	The offset in terms of one full step cycle (left and right) for the timing of the rotation. Positive means after the position, Negative means before.
Full Strength Speed	Float	The speed at which the full sprint animation strength is reached. This fades out the rotation aspect as the character slows down.

Melee Weapon

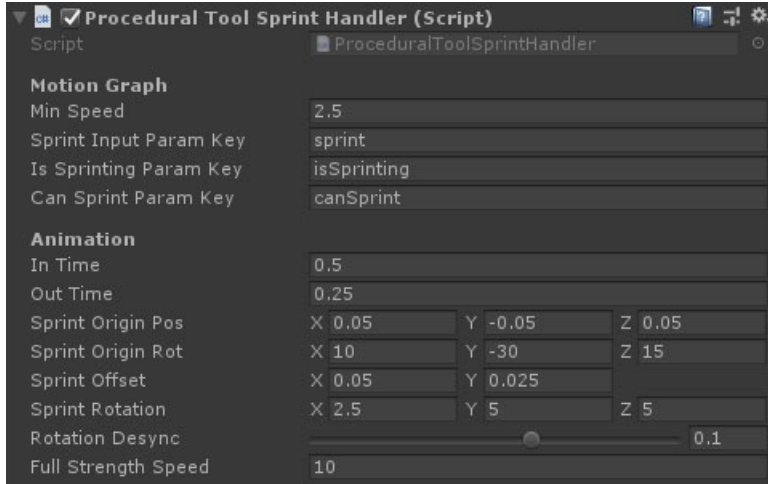| NAME	TYPE	DESCRIPTION
Min Attack Time	Float	The minimum amount of time the sprint animation will be paused after an attack.
Min Block Time	Float	The minimum amount of time the sprint animation will be paused when blocking (prevents tapping block).

See Also

Melee Weapons

The Motion Graph

Additive Transforms & Effects

ProceduralThrownSprintHandler MonoBehaviour

Overview

The ProceduralThrownSprintHandler behaviour is attached to a thrown weapon to model different sprinting mechanics. This version uses procedural animation to move the firearm into a sprint pose and add an additive transform bob effect.

Inspector

Properties

Motion Graph

NAME	TYPE	DESCRIPTION
Min Speed	Float	The minimum speed the character must be moving for the sprint animation to play.
Sprint Input Param Key	String	(Required) The switch parameter on the motion graph which is set by the input handler to tell the character when to sprint.
Is Sprinting Param Key	String	(Required) A switch parameter on the motion graph which the graph sets when the character is sprinting.
Can Sprint Param Key	String	(Optional) A switch parameter on the motion graph which tells the character if it can sprint or not.

Animation

NAME	TYPE	DESCRIPTION
In Time	Float	The time taken to blend into the sprint animation.
Out Time	Float	The time taken to blend out of the sprint animation to idle.
Sprint Origin Pos	Vector	The neutral weapon / item position in sprint pose before bob is applied.
Sprint Origin Rot	Vector	The neutral weapon / item rotation in sprint pose before bob is applied.
Sprint Offset	Vector	The peak position offset of the step cycle on the z and y axes (z does not change).

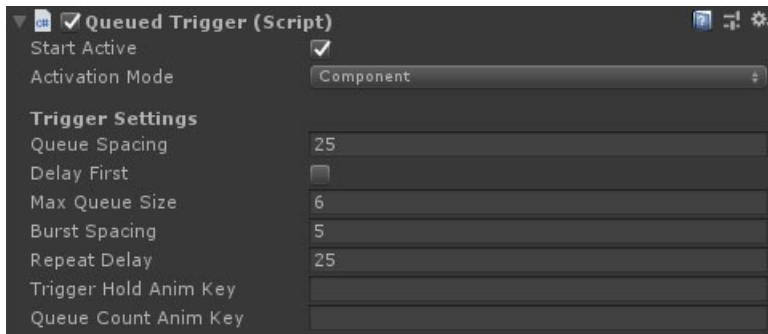| NAME	TYPE	DESCRIPTION
Sprint Rotation	Vector	The peak rotation of the step cycle on each axis.
Rotation Desync	Float	The offset in terms of one full step cycle (left and right) for the timing of the rotation. Positive means after the position, Negative means before.
Full Strength Speed	Float	The speed at which the full sprint animation strength is reached. This fades out the rotation aspect as the character slows down.

See Also

Thrown Weapons

The Motion Graph

Additive Transforms & Effects

ProceduralToolSprintHandler MonoBehaviour

Overview

The ProceduralToolSprintHandler behaviour is attached to a wieldable tool to model different sprinting mechanics. This version uses procedural animation to move the tool into a sprint pose and add an additive transform bob effect.

Inspector

Properties

Motion Graph

NAME	TYPE	DESCRIPTION
Min Speed	Float	The minimum speed the character must be moving for the sprint animation to play.
Sprint Input Param Key	String	(Required) The switch parameter on the motion graph which is set by the input handler to tell the character when to sprint.
Is Sprinting Param Key	String	(Required) A switch parameter on the motion graph which the graph sets when the character is sprinting.
Can Sprint Param Key	String	(Optional) A switch parameter on the motion graph which tells the character if it can sprint or not.

Animation

NAME	TYPE	DESCRIPTION
In Time	Float	The time taken to blend into the sprint animation.
Out Time	Float	The time taken to blend out of the sprint animation to idle.
Sprint Origin Pos	Vector	The neutral weapon / item position in sprint pose before bob is applied.
Sprint Origin Rot	Vector	The neutral weapon / item rotation in sprint pose before bob is applied.
Sprint Offset	Vector	The peak position offset of the step cycle on the z and y axes (z does not change).

NAME	TYPE	DESCRIPTION
Sprint Rotation	Vector	The peak rotation of the step cycle on each axis.
Rotation Desync	Float	The offset in terms of one full step cycle (left and right) for the timing of the rotation. Positive means after the position, Negative means before.
Full Strength Speed	Float	The speed at which the full sprint animation strength is reached. This fades out the rotation aspect as the character slows down.

See Also

Wieldable Tools

The Motion Graph

Additive Transforms & Effects

QueuedTrigger MonoBehaviour

Overview

The QueuedTrigger module charges up while the fire button is held down and fires once it hits full charge.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active trigger immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: **Component** means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. **GameObject** activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Charge Duration	Float	How long does it take to charge the trigger.
Uncharge Duration	Float	How long does it take to uncharge the trigger, assuming it hasn't gone off.
Repeat	Boolean	Once the shot is fired, start charging the next shot if this is true.
Repeat Delay	Float	The time between a shot firing and starting charging the next shot.
Audio Source	AudioSource	The source to play the audio from (needs its own as it must be interrupted and seeked).
Trigger Audio Charge	AudioClip	The audio clip to play on charge.
Trigger Audio Release	AudioClip	The audio clip to play on release.

See Also

Modular Firearms

RandomObjectMuzzleEffect MonoBehaviour

Overview

The RandomObjectMuzzleEffect module enables one out of a pool of game objects each shot, and then disables it again after a brief wait.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active muzzle effect immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: **Component** means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. **GameObject** activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Muzzle Flashes	GameObject Array	The muzzle flash game objects.
Muzzle Flash Duration	Float	The duration the flash should remain active. Keep this longer for objects with particle effects.
Firing Sounds	AudioClip Array	The audio clips to use when firing. Chosen at random.

See Also

Modular Firearms

RaycastTargetingSystem MonoBehaviour

Overview

The RaycastTargetingSystem behaviour is a targeting system that passes the current raycast hit point to any guided projectiles that have been registered with it. Those projectiles require a TargetingSystemTracker component attached.

Inspector
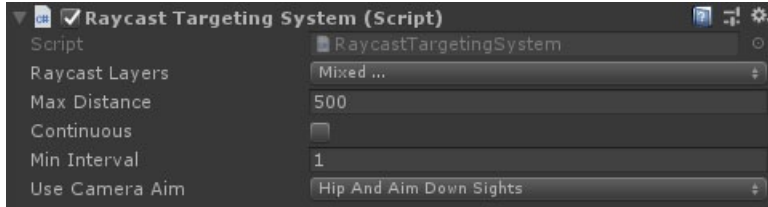
Properties

NAME	TYPE	DESCRIPTION
Raycast Layers	LayerMask	The layers to cast against.
Max Distance	Float	The max distance for targets to home in on.
Continuous	Boolean	Should the targetying system update the projectiles each tick or just pass the initial hit point.
Min Interval	Float	The targeting system will only check for new targets after the min interval has elapsed since the last time, passing the last target to any new trackers before that. This is useful for burst launchers so that each missile tracks the same target.
Use Camera Aim	Dropdown	When set to use camera aim, the targeting system casts forward from the FirstPersonCamera's aim transform. If not then it casts from the transform the targeting system is attached to.

See Also

Modular Firearms

Hitscan vs Projectiles

TargetingSystemTracker Behaviour

RechargingAmmo MonoBehaviour

Overview

The RechargingAmmo module is a unique ammo for the firearm it is attached to.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active ammo module immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: **Component** means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. **GameObject** activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Printable Name	String	The name to show on the HUD.
Effect	Dropdown	A dropdown which shows all the AmmoEffect modules on the GameObject. This is the effect the bullets will have on hitting a target.
Starting Ammo	Int	The amount of ammo the weapon starts with.
Max Ammo	Int	The maximum amount of ammo the weapon can carry.

See Also

Modular Firearms

RecoilPushback MonoBehaviour

Overview

The RecoilPushback behaviour connects to a firearm's recoil module and pushes the wielding character backwards when they fire.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Force	Float	The force to push the character back with.
Pitch Mode	Dropdown	How character aim pitch affects the pushback. **Ignore** will push the character backwards horizontally with the set force, **ScaleForce** will push the character back horizontally based on pitch, while **Full3D** will push the character directly away from their aim direction (including pitch).
Airborne Only	Boolean	Should the recoil only push back when the character is not grounded.

See Also

Modular Firearms

ReloaderCountdown MonoBehaviour

Overview

The ReloaderCountdown behaviour attaches to the firearm's reloader and tracks the magazine size. As it approaches zero then it plays audio in sequence. This can be used to add a warning that the magazine is almost empty, or for stylistic audio such as an M1 Garand's iconic ping on the last round.

Inspector
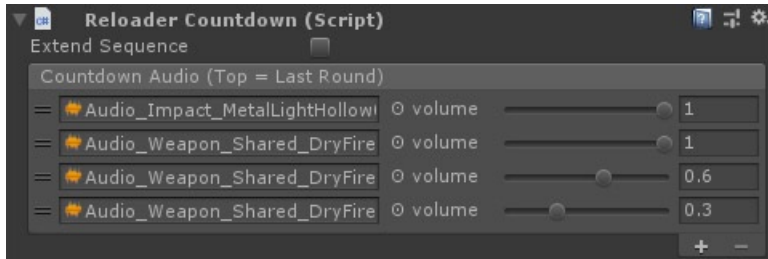
Properties

NAME	TYPE	DESCRIPTION
Countdown Audio	Countdown Audio Array	The audio clips to play as ammo is consumed. First element is for the last round in the magazine, second = penultimate, and so on.
Extend Sequence	Boolean	If set, then the last clip will be used for all ammo until the count is within the sequence.

Countdown Audio Element

NAME	TYPE	DESCRIPTION
Clip	AudioClip	The audio clip to play.
Volume	Float	The volume to play the clip at.

See Also

Modular Firearms

RenderTextureScope MonoBehaviour

Overview

The RenderTextureScope is used to model a weapon scope using a separate camera attached to the weapon. The shader provided with NeoFPS provides parallax effects that move the scope ring and reticule as the camera viewing the scope lens goes off the scope's axis, and fades the lens to an opaque glass at the extremes (disabling the scope camera when fully opaque. Render texture based scopes incur a performance penalty because they require rendering the scene twice, along with communicating back and forth with the GPU each frame.

Inspector
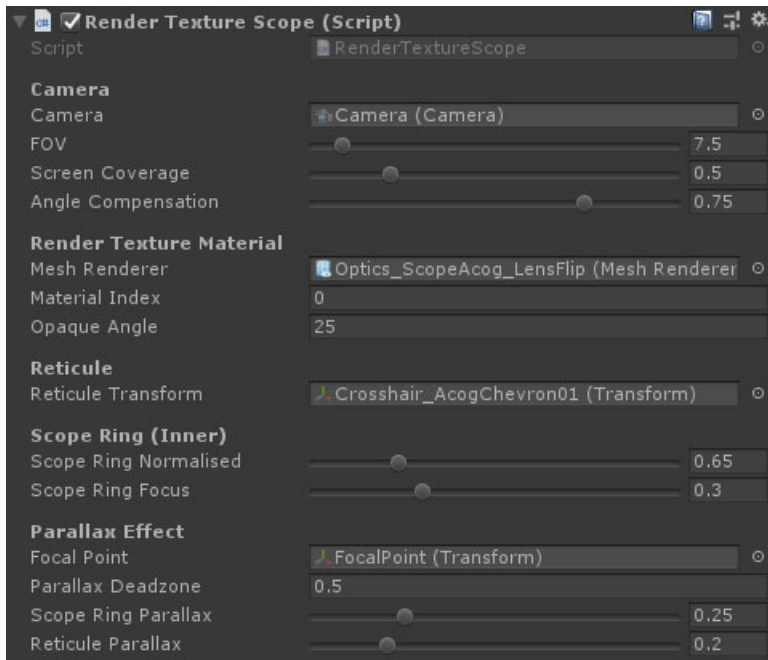
Properties

The RenderTextureScope properties are split into a number of sections:

Camera

NAME	TYPE	DESCRIPTION
Camera	Camera	The scope camera.
FOV	Float	The field of view for the scope camera.
Screen Coverage	Float	The amount of the screen height the scope takes up (multiplier). Smaller screen coverage needs a smaller render texture and therefore performs better.
Angle Compensation	Float	Rotate the camera to adapt to the parallax effect. At 1, the image will track (roughly) with the scope ring. At 0, the scope ring will be completely detached from the image.

Render Texture Material

NAME	TYPE	DESCRIPTION
Mesh Renderer	MeshRenderer	The mesh renderer of the scope lens which will receive the render texture.

NAME	TYPE	DESCRIPTION
Material Index	Integer	The material index is used to specify which material to modify from a multi-part mesh.
Opaque Angle	Float	The angle in degrees off the axis of the scope where the camera stops rendering and the material becomes completely opaque.

Reticule

NAME	TYPE	DESCRIPTION
Reticule Transform	Transform	The transform of the reticule geometry. Local position (0,0,0) should be centered on the camera and slightly in front.

Scope Ring (Inner)

NAME	TYPE	DESCRIPTION
Scope Ring Normalised	Float	The start of the scope ring in terms of the normalised radius. At 1, the ring is a circle that touches the edges of the texture.
Scope Ring Focus	Float	The focus level of the scope ring. Setting this lower will expand the blur of the ring out from the normalised start radius.

Parallax Effect

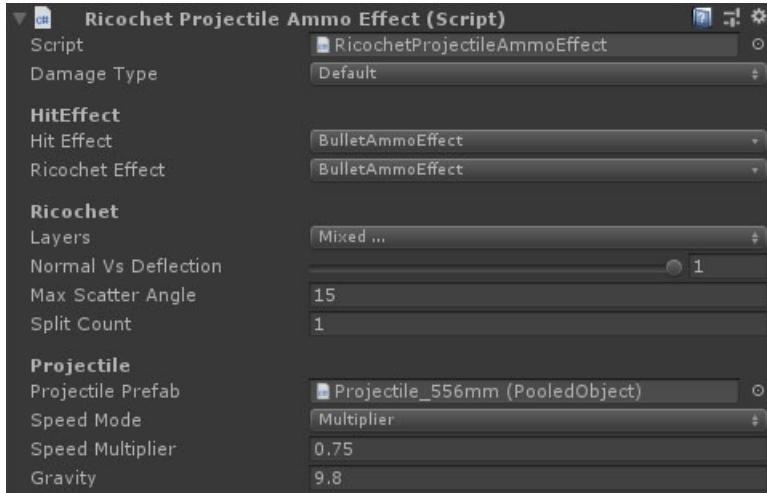| NAME	TYPE	DESCRIPTION
Focal Point	Transform	A transform that represents the lens of the scope that the player looks into. Used to determine the off-axis angle for parallax and fade. This should be placed near the lens of the scope.
Parallax Deadzone	Float	An angle range from the scope axis where there is no parallax. This prevents the reticule from going off center during small movements like idle animations or procedural breathing.
Scope Ring Parallax	Float	The amount of parallax to add to the scope ring as the eye-line moves away from the scope axis.
Reticule Parallax	Float	The amount of parallax to add to the reticule as the eye-line moves away from the scope axis. Values above the scope ring parallax will make the reticule seem like it is further forward than the ring.

See Also

Modular Firearms

Scopes & Optics

RicochetHitscanAmmoEffect MonoBehaviour

Overview

The RicochetHitscanAmmoEffect module allows projectiles to bounce off surfaces based on the impact angle and the distance travelled. If you are using a projectile shooter, then there is a separate RicochetProjectileAmmoEffect that should be used instead of this.

This ammo effect is also used alongside another effect to apply damage and visual elements.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Damage Type	DamageType	The type of damage the weapon should do with this ammo.
Initial Hit Effect	BaseAmmoEffect	The effect of the ammo when it first hits.
Second Hit Effect	BaseAmmoEffect	The effect of the ammo after it has penetrated something.
Max Distance	Float	The maximum distance that the weapon will register a hit (includes the distance travelled up to the penetration).
Layers	LayerMask	The layers bullets will collide with.
Normal Vs Deflection	Float	A blending value based on ricocheting directly along the hit normal (0) vs the reflection of the inbound ray (1).
Max Scatter Angle	Float	Randomises the deflected bullet direction within this cone angle.
Split Count	Integer	The number of shots to split the ricochet into (1 is no split).
Tracer Prototype	[PooledObject][4]	The optional pooled tracer prototype to use (must implement the IPooledHitscanTrail interface)
Tracer Size	Float	How size (thickness/radius) of the tracer line.
Tracer Duration	Float	How long the tracer line will last.

See Also

Modular Firearms

Surfaces

RicochetProjectileAmmoEffect MonoBehaviour

Overview

The RicochetProjectileAmmoEffect module allows projectiles to bounce off surfaces based on the impact angle and the speed the projectile is travelling. If you are using a hitscan shooter, then there is a separate RicochetHitscanAmmoEffect that should be used instead of this.

This ammo effect is also used alongside another effect to apply damage and visual elements.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Damage Type	DamageType	The type of damage the weapon should do with this ammo.
Initial Hit Effect	BaseAmmoEffect	The effect of the ammo when it first hits.
Second Hit Effect	BaseAmmoEffect	The effect of the ammo after it has penetrated something.
Layers	LayerMask	The layers bullets will collide with.
Normal Vs Deflection	Float	A blending value based on ricocheting directly along the hit normal (0) vs the reflection of the inbound ray (1).
Max Scatter Angle	Float	Randomises the deflected bullet direction within this cone angle.
Split Count	Integer	The number of shots to split the ricochet into (1 is no split).
Projectile Prefab	PooledObject	The projectile to spawn.

NAME	TYPE	DESCRIPTION
Speed Mode	Dropdown	ow the ricochet speed of the projectile is calculated. **Multiplier** multiplies the contact speed by a set value, **FixedSpeed** uses a set speed, **AngleBasedMultiplier** multiplies the contact speed by a value based on the contact angle, **AngleBasedSpeed** uses a set speed based on the contact angle.
Speed Multiplier	Float	A multiplier applied to the bullet speed after ricochet. Only visible when **Speed Mode** is set to **Multiplier**.
Fixed Speed	Float	The speed of the projectile after ricochet. Only visible when **Speed Mode** is set to **FixedSpeed**.
Straight On Multiplier	Float	A multiplier applied to the bullet speed after ricochet when the entry velocity was perpendicular to the surface. Only visible when **Speed Mode** is set to **AngleBasedMultiplier**.
Glancing Multiplier	Float	A multiplier applied to the bullet speed after ricochet if the initial bullet was travelling parallel to the surface. Only visible when **Speed Mode** is set to **AngleBasedMultiplier**.
Straight On Speed	Float	The speed of the projectile after ricochet when the entry velocity was perpendicular to the surface. Only visible when **Speed Mode** is set to **AngleBasedSpeed**.
Glancing Speed	Float	The speed of the projectile after ricochet if the initial bullet was travelling parallel to the surface. Only visible when **Speed Mode** is set to **AngleBasedSpeed**.
Gravity	Float	The gravity for the projectile.

See Also

Modular Firearms

Surfaces

SampleParticleExplosion MonoBehaviour

Overview

The SampleParticleExplosion behaviour is a basic implementation of an explosion effect involving a central fireball and a number of after effects.

Inspector
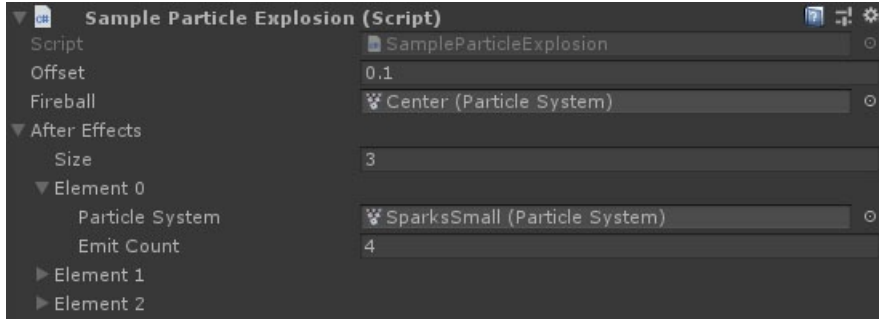
Properties

NAME	TYPE	DESCRIPTION
Offset	Float	The distance along the normal to offset the exposion (for raising from a surface).
Fireball	ParticleSystem	The central fireball of the explosion.
After Effects	After Effects Array	Any extra emitters for explosion after effects (smoke, sparks, etc).

After Effects

NAME	TYPE	DESCRIPTION
Particle System	ParticleSystem	The after effect particle system.
Emit Count	Int	The amount of particles to emit.

See Also

Explosions

Unity ParticleSystem

ScopedAimer MonoBehaviour

Overview

The ScopedAimer module will raise the weapon and then switch to a HUD Scope when the weapon is fully raised.

In The Scene View

With a gameobject selected that uses a HeadMoveAimer component, the aimer guide handle will be visible in the scene view. This is used to represent the aim point of the aimer, with the blue arrow pointing forwards, and the green arrow pointing up.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active aimer immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: **Component** means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. **GameObject** activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Aim Up Audio	AudioClip	An audio clip to play when the weapon is raised.
Aim Down Audio	AudioClip	An audio clip to play when the weapon is lowered.
Hip Accuracy Cap	Float	The highest accuracy the firearm can achieve while not aiming down sights.
Aimed Accuracy Cap	Float	The highest accuracy the firearm can achieve while aiming down sights.
Can Aim While Reloading	Boolean	Should the weapon be lowered when reloading or can it stay aimed.
On Aim Up	UnityEvent	An event called when the weapon is fully raised.
On Aim Down	UnityEvent	An event called when the weapon is fully lowered.
Aim Offset	Vector3	The offset for the root transform to move to align the weapon sights with the camera.
From Transform	Transform	Dragging a transform in here will calculate the offset from the root transform to this transform, and set the **Aim Offset** property with the result
Hud Scope Key	String	The key for the specific scope to show.
Fov Multiplier	Float	A multiplier for the camera FoV for aim zoom.
Aim Time	Float	The time it takes to reach full aim, or return to zero aim.
Position Spring Multiplier	Float	A multiplier for procedural spring position animation on the weapon. Used to reduce movement when the firearm is close to the camera.
Rotation Spring Multiplier	Float	A multiplier for procedural spring rotation animation on the weapon. Used to reduce movement when the firearm is close to the camera.

NAME	TYPE	DESCRIPTION
Aim Anim Bool	String	The animator parameter key for a bool used to control aiming state in animations.
Block Trigger	Boolean	If true then the gun cannot fire while transitioning in and out of aim mode. This is used to prevent gunshots interrupting the animation. This property will only be shown if the Aim Anim Bool property is true.
Crosshair Up	FpsCrosshair	The crosshair to show when aiming down sights.
Crosshair Down	FpsCrosshair	The crosshair to show when not aiming down sights.

See Also

Modular Firearms

First Person Camera

HudScope

SemiAutoTrigger MonoBehaviour

Overview

The SemiAutoTrigger fires a single shot as fast as the player can press the trigger. If the trigger is held down it will also repeat at a slow rate.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active trigger immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: **Component** means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. **GameObject** activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Cooldown	Int	Cooldown between trigger pulls (number of fixed update frames).
Repeat Delay	Int	How many fixed update frames before firing again (0 = requires fresh trigger press).

See Also

[Modular Firearms](#)

SharedPoolAmmo MonoBehaviour

Overview

The SharedPoolAmmo module uses ammo that is stored in the character inventory and can be shared between weapons.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active ammo module immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: **Component** means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. **GameObject** activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Effect	Dropdown	A dropdown which shows all the AmmoEffect modules on the GameObject. This is the effect the bullets will have on hitting a target.
Ammo Type	SharedAmmoType	The ammo type to use.

See Also

Modular Firearms

SharedAmmoType

ShieldBoosterToolAction MonoBehaviour

Overview

The ShieldBoosterToolAction behaviour is used to add an action that restores a character's shield slots to a wieldable tool.

Inspector
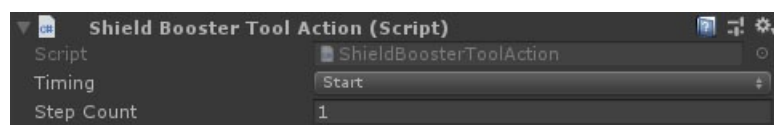
Properties

NAME	TYPE	DESCRIPTION
Timing	Dropdown	When should the tool apply the shield boost.
Step Count	Integer	The number of shield steps to recharge.

See Also

Wieldable Tools

Health and Damage

SimpleBallisticShooter MonoBehaviour

Overview

The SimpleBallisticShooter module spawns an BallisticProjectile. It is not affected by accuracy, and has no options to use the camera aim target. If you want these extra features then you can alternatively use the [BallisticShooter][4]

Inspector

Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active shooter immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: **Component** means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. **GameObject** activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Projectile Prefab	PooledObject	The projectile to spawn.
Muzzle Tip	Transform	The position and direction the projectile is spawned.
Layers	LayerMask	The physics collision layers the shot can hit.
Muzzle Speed	Float	The speed of the projectile.
Layers	LayerMask	The layers that will be checked against when casting for valid interaction targets.
Gravity	Float	The gravity for the projectile.

See Also

Modular Firearms

BallisticProjectile

SimpleParticleMuzzleEffect MonoBehaviour

Overview

The SimpleParticleMuzzleEffect behaviour spawns triggers a particle system to emit each shot.

Inspector
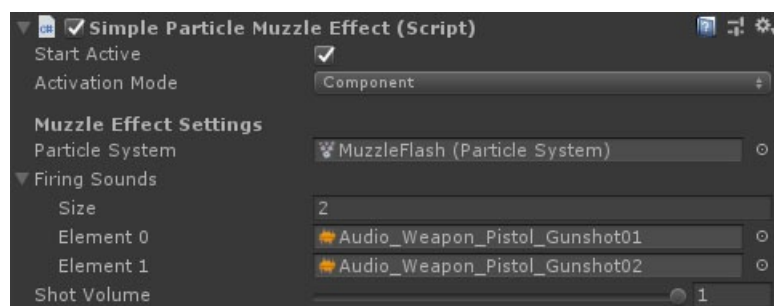
Properties

NAME	TYPE	DESCRIPTION
Particle System	ParticleSystem	The particle system to play.
Firing Sounds	AudioClip Array	The audio clips to use when firing. Chosen at random.

See Also

Modular Firearms

SimpleReloader MonoBehaviour

Overview

The SimpleReloader behaviour triggers a reload animation and waits until it completes before taking ammo from the firearm's ammo module and adding it to the magazine.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active reloader immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: **Component** means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. **GameObject** activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Magazine Size	Float	The number of rounds that can be fit in the magazine at once.
Starting Magazine	Float	The number of rounds in the magazine on initialisation.
Reload Delay Type	Dropdown	The delay type between starting and completing a reload. The options are **None**, **Elapsed Time**, **External Trigger**.
Reload Duration*	Float	The time taken to reload.
Reload Anim Trigger	String	The AnimatorController trigger key for the reload animation.
Reload Audio	AudioClip	The audio clip to play while reloading.

* This property is only visible if the reload delay type is set to ElapsedTime.

See Also

Modular Firearms

FirearmAnimEventsHandler

Unity Animator

Unity AnimatorController

SimpleSteeringMotor MonoBehaviour

Overview

The SimpleSteeringMotor behaviour is used as part of a guided projectile to steer and move it towards its target. As time goes on its turning circle improves to prevent it getting stuck orbiting its target.

Inspector
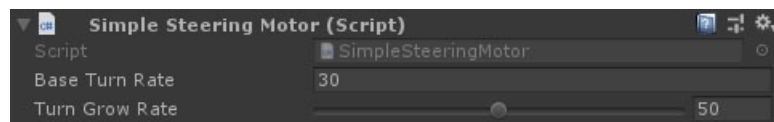
Properties

NAME	TYPE	DESCRIPTION
Base Turn Rate	Float	The turn rate of the projectile (degrees per second).
Turn Grow Rate	Float	An increase to the turn rate based on elapsed time in flight (base turn rate + turn grow rate * elapsed time).

See Also

Modular Firearms

Hitscan vs Projectiles

SpreadBallisticShooter MonoBehaviour

Overview

The SpreadShooter module is in effect a number of BallisticShooter modules in one.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active shooter immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: **Component** means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. **GameObject** activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Projectile Prefab	PooledObject	The projectile to spawn.
Muzzle Speed	Float	The speed of the projectile.
Gravity	Float	The gravity for the projectile.
Muzzle Tip	Transform	The transform that the bullet actually fires from.
Layers	LayerMask	The physics collision layers the shot can hit.
Min Aim Offset	Float	The maximum angle from forward the shooter can fire when accuracy is 1.
Max Aim Offset	Float	The maximum angle from forward the shooter can fire when accuracy is 0.
Use Camera Aim	Dropdown	When set to use camera aim, the gun first casts from the FirstPersonCamera's aim transform, and then from the muzzle tip to that point to get more accurate firing. Options are: **HipFireOnly**, **HipAndAimDownSights**, **AimDownSightsOnly**, **Never**.

NAME	TYPE	DESCRIPTION
Bullet Count	Int	How many pellets are fired each shot.
Cone	Float	The spread of the cone in degrees.

See Also

Modular Firearms

SpreadHitscanShooter MonoBehaviour

Overview

The SpreadHitscanShooter module is in effect a number of HitscanShooter modules in one.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active shooter immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: **Component** means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. **GameObject** activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Max Distance	Float	The maximum distance that the weapon will register a hit.
Muzzle Tip	Transform	The transform that the bullet actually fires from.
Layers	LayerMask	The physics collision layers the shot can hit.
Min Aim Offset	Float	The maximum angle from forward the shooter can fire when accuracy is 1.
Max Aim Offset	Float	The maximum angle from forward the shooter can fire when accuracy is 0.
Use Camera Aim	Dropdown	When set to use camera aim, the gun first casts from the FirstPersonCamera's aim transform, and then from the muzzle tip to that point to get more accurate firing. Options are: **HipFireOnly**, **HipAndAimDownSights**, **AimDownSightsOnly**, **Never**.
Bullet Count	Int	How many pellets are fired each shot.
Cone	Float	The spread of the cone in degrees.

NAME	TYPE	DESCRIPTION
Tracer Prototype	Pooled Object	The optional pooled tracer prototype to use (must implement the IPooledHitscanTrail interface).
Tracer Size	Float	The size (thickness/radius) of the tracer line.
Tracer Duration	Float	How long should the tracer object stay visible.
Shots Per Tracer	Int	How many pellets are required per tracer line.

See Also

Modular Firearms

SpringRecoilHandler MonoBehaviour

Overview

The SpringRecoilHandler module uses the additive transform system to move and rotate the firearm as it recoils.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active recoil handler immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: **Component** means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. **GameObject** activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
On Recoil	UnityEvent	An event that fires every time the weapon recoils.
Hip Accuracy Kick	Float	The accuracy decrement per shot in hip fire mode (accuracy has a 0-1 range).
Hip Accuracy Recover	Float	The accuracy recovered per second in hip fire mode (accuracy has a 0-1 range).

NAME	TYPE	DESCRIPTION
Sighted Accuracy Kick	Float	The accuracy decrement per shot in sighted fire mode (accuracy has a 0-1 range).
Sighted Accuracy Recover	Float	The accuracy recovered per second in sighted fire mode (accuracy has a 0-1 range).
Weapon Kicker	AdditiveKicker	The additive kicker behaviour on the object.
Weapon Wander	Float	How much of the weapon recoil rotation is directed sideways instead of up.
Weapon Rotation	Float	The rotation angle of the weapon recoil while firing from the hip. At 0 wander, this is how many degrees the weapon pitches up.
Weapon Rotation Aimed	Float	The rotation angle of the weapon recoil while aimed. At 0 wander, this is how many degrees the weapon pitches up.
Weapon Knock Back	Float	Knock-back is movement backwards, towards the camera.
Head Wander	Float	How much of the head recoil rotation is directed sideways instead of up. At negative values this will be the opposite of the guns sideways rotation.
Head Rotation	Float	The head rotation angle of the weapon recoil when firing from the hip. At 0 wander, this is how many degrees the weapon pitches up.
Head Rotation Aimed	Float	The head rotation angle of the weapon recoil when aimed. At 0 wander, this is how many degrees the weapon pitches up.
Bypass Move Multiplier	Boolean	Should the movement recoil effect be affected by the weapon and head spring multipliers? These affect everything, from bob to impacts to shake. If the multiplier is at zero then the recoil will be disabled either way.
Bypass Rotate Multiplier	Boolean	Should the rotation recoil effect be affected by the weapon and head spring multipliers? These affect everything, from bob to impacts to shake. If the multiplier is at zero then the recoil will be disabled either way.
Rotation Duration	Float	The amount of time the angle recoil effect takes to return to zero.
KnockBack Duration	Float	The amount of time the knockback effect takes to return to zero.

See Also

Modular Firearms

Additive Transforms and Effects

StandardShellEject MonoBehaviour

Overview

The StandardShellEject behaviour launches a pooled bullet casing object away from the specified transform.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active ejector immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: **Component** means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. **GameObject** activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Shell Eject Proxy	Transform	A proxy transform where ejected shells will be spawned.
Shell Prefab	PooledObject	The shell prefab object to spawn.
Delay Type	Dropdown	The delay type between firing and ejecting a shell. Options are **None**, **Elapsed Time**, **External Trigger**.
Delay*	Float	The delay time between firing and ejecting a shell.
Out Speed	Float	The ejected shell speed directly out from the ejector.
Back Speed	Float	The ejected shell speed back over the wielder's shoulder.
Inherit Velocity	Float	How much of the character's velocity should be added to the ejected shells.
Angular Velocity A	Vector3	The minimum angular velocity on each axis (will be picked at random between this an B).
Angular Velocity B	Vector3	The maximum angular velocity on each axis (will be picked at random between this an A).

* This property will only be visible if the delay type is set to Elapsed Time

See Also

Modular Firearms

PooledObject

SurfaceBulletPhysicsAmmoEffect MonoBehaviour

Overview

The SurfaceBulletPhysicsAmmoEffect module allows bullets to ricochet and penetrate objects based on the surface. This ammo effect is used alongside another effect to apply damage and visual elements. Since both ricochets and penetrations reduce the speed of the resulting projectile, this effect works well when paired with the AdvancedBulletAmmoEffect which allows damage to fall-off with speed.

The penetration and ricochet settings for each surface are specified using a SurfaceBulletPhysicsInfo scriptable object.

This module only works with projectile based shooters.

Inspector
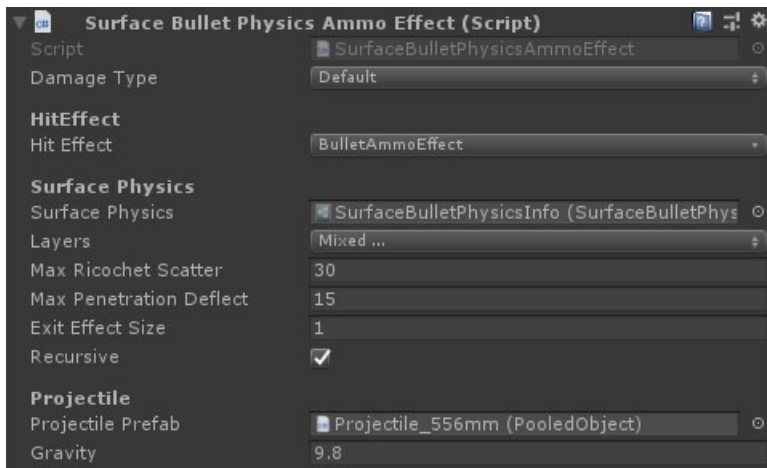
Properties

NAME	TYPE	DESCRIPTION
Damage Type	DamageType	The type of damage the weapon should do with this ammo (does nothing for this effect).
Hit Effect	BaseAmmoEffect	The effect of the ammo when it hits something.
Surface Physics	SurfaceBulletPhysicsInfo	The per-surface bullet physics info
Layers	[LayerMask][unity-layers]	The layers bullets will collide with.
Normal Vs Deflection	Float	A blending value based on ricocheting directly along the hit normal (0) vs the reflection of the inbound ray (1).
Max Ricochet Scatter	Float	Randomises the deflected bullet direction within this cone angle, dependent on surface settings.
Max Penetration Deflect	Float	Randomises the penetrating bullet direction within this cone angle, dependent on surface settings.
Exit Effect Size	Float	Uses the surface system to show a bullet hit effect on exit. Set this to zero if you don't want it to happen.
Recursive	Boolean	Should the bullet keep ricocheting / penetrating after the first time until it has slowed or travelled far enough.

NAME	TYPE	DESCRIPTION
Projectile Prefab	PooledObject	The projectile to spawn.
Gravity	Float	The gravity for the projectile.

See Also

Modular Firearms

Surfaces

TargetingSystemTracker MonoBehaviour

Overview

The TargetingSystemTracker behaviour is a guided projectile tracking system which defers its target tracking to a targeting system such as a TargetLockTrigger or LaserTargetingSystem.

Inspector
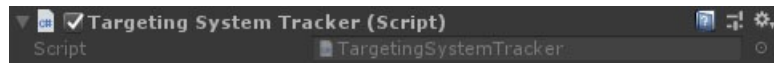
Properties

The TargetingSystemTracker behaviour has no properties exposed in the inspector.

See Also

Modular Firearms

Hitscan vs Projectiles

TargetLockTrigger MonoBehaviour

Overview

The TargetLockTrigger module charges up while the fire button is held down and fires once it hits full charge.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active trigger immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: **Component** means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. **GameObject** activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Charge Duration	Float	How long does it take to charge the trigger.
Uncharge Duration	Float	How long does it take to uncharge the trigger, assuming it hasn't gone off.
Repeat	Boolean	Once the shot is fired, start charging the next shot if this is true.
Repeat Delay	Float	The time between a shot firing and starting charging the next shot.
Audio Source	AudioSource	The source to play the audio from (needs its own as it must be interrupted and seeked).
Trigger Audio Charge	AudioClip	The audio clip to play on charge.
Trigger Audio Release	AudioClip	The audio clip to play on release.

See Also

Modular Firearms

TargetTrackingAmmoEffect MonoBehaviour

Overview

The TargetTrackingAmmoEffect behaviour is used to tag targets for guided projectiles to home in on. The projectiles must use a TargetingSystemTracker component.

Inspector
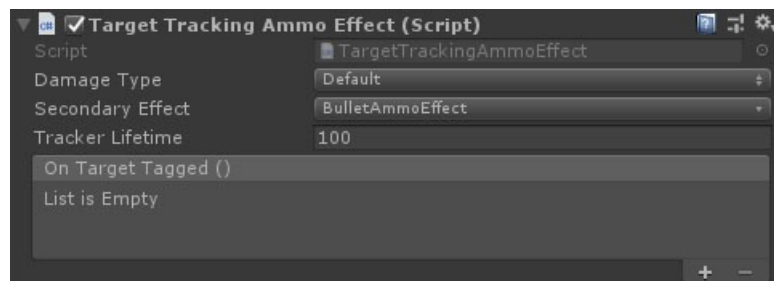
Properties

NAME	TYPE	DESCRIPTION
Damage Type	Dropdown	The type of damage the impact counts as. This is a common property for all ammo effects and actually has no effect in this case.
Secondary Effect	AmmoEffect	An optional secondary ammo effect to allow the tracking bullet to deal damage, etc.
Tracker Lifetime	Float	The amount of time (seconds) the tracking effect will last for.
On Target Tagged	UnityEvent	An event fired when an object is successfully tagged. You can use this to perform actions like switching ammo effects.

See Also

Modular Firearms

Hitscan vs Projectiles

ThrownWeapon MonoBehaviour

Overview

The ThrownWeapon behavior is used for weapons such as hand grenades.

Inspector
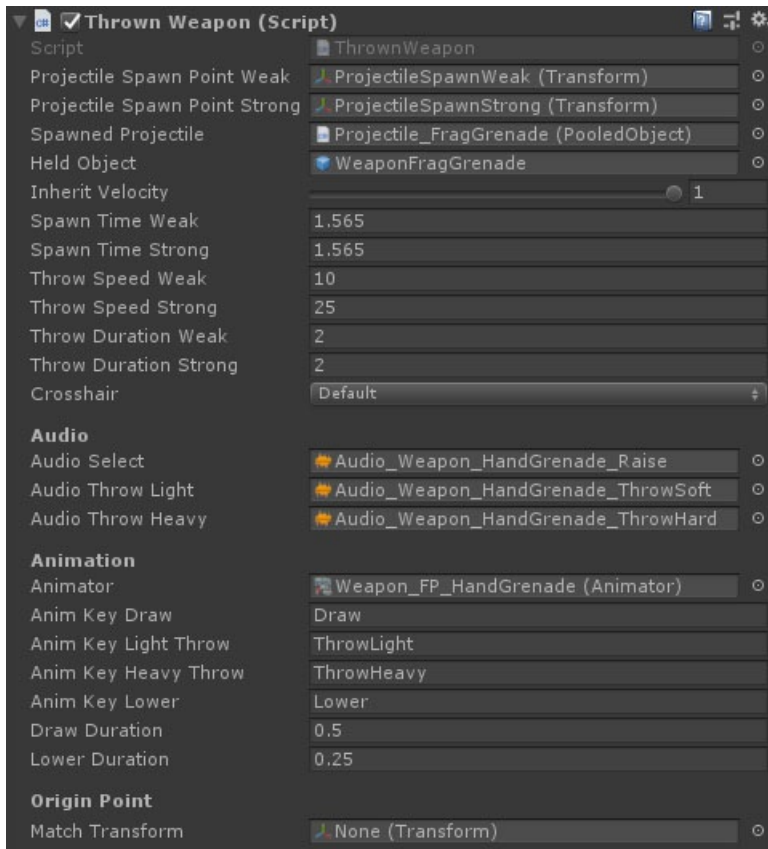
Properties

NAME	TYPE	DESCRIPTION
Animator	Animator	The animator component of the weapon.
Projectile Spawn Point Weak	Transform	A proxy transform for setting the position and rotation of the spawned projectile (weak throw).
Projectile Spawn Point Strong	Transform	A proxy transform for setting the position and rotation of the spawned projectile (strong throw).
Spawned Projectile	PooledObject	The prefab to throw.
Held Object	GameObject	The weapon game object. This is deactivated and swapped with the pooled object during the throw animation.
Inherit Velocity	Float	How much of the character velocity does the thrown weapon inherit (think Counter Strike).
Spawn Time Weak	Float	The point in the animation (seconds) to swap the animated weapon with the pooled physics weapon (weak throw).
Spawn Time Strong	Float	The point in the animation (seconds) to swap the animated weapon with the pooled physics weapon (strong throw).

NAME	TYPE	DESCRIPTION
Throw Speed Weak	Float	The throw speed of the projectile (weak throw).
Throw Speed Strong	Float	The throw speed of the projectile (strong throw).
Throw Duration Weak	Float	The full duration of the weak throw animation.
Throw Duration Strong	Float	The full duration of the strong throw animation.
Draw Duration	Float	The time it takes to raise the weapon.
Crosshair	FpsCrosshair	The crosshair to show when the weapon is drawn.
Audio Rotator	AudioRotator	An audio rotator for weapon noises.
Audio Select	AudioClip	The audio clip when raising the weapon.
Audio Throw Light	AudioClip	The audio clip for a weak throw.
Audio Throw Heavy	AudioClip	The audio clip for a strong throw.
Animator	Animator	The animator component of the weapon.
Anim Key Draw	String	The key for the AnimatorController trigger property that triggers the draw animation.
Anim Key Light Throw	String	The key for the AnimatorController trigger property that triggers the light throw animation.
Anim Key Heavy Throw	String	The key for the AnimatorController trigger property that triggers the heavy throw animation.
Anim Key Lower	String	The AnimatorController trigger key for the weapon lower animation (blank = no animation).
Lower Duration	Float	The time taken to lower the item on deselection.

Origin Point

The NeoFPS weapons assume that the camera is placed at the origin. For many assets or 3rd party weapons, the origin is at the character feet or hips and the camera is a child object of the weapon's hierarchy. To align the object to correctly work with NeoFPS you can drag the camera object of the weapon into the **Match Transform** field under the **Origin Point** heading. This will move everything below the spring object to match up.

See Also

Thrown Weapons

PooledObject

Unity Animator

Unity AnimatorController

ThrownnWieldableStanceManager MonoBehaviour

Overview

The ThrownnWieldableStanceManager behaviour is used to specify poses or stances for a thrown weapon. The entire weapon will be moved to match the stance, and can optionally set an animator bool parameter too.

The stance will be exited temporarily whenever the weapon is thrown.

Inspector

Properties

Use the **Add Stance** buttons to add a new stance to the manager.

Stances

Individual stances have the following properties:

NAME	TYPE	DESCRIPTION
Name	String	The name of the stance.
Animator Bool Key	String	An optional name of a bool parameter in the weapon's Animator.
Position	Vector	The position to move the weapon to in this stance.
Rotation	Vector	The rotation of the weapon in this stance.
In Position Blend	Dropdown	The easing method for blending between the source position and stance position on entering the stance. Options are: **Lerp, EaseIn, EaseOut, EaseInOut, SwingAcross, SwingUp, Spring, Bounce** and **Overshoot**.
In Rotation Blend	Dropdown	The easing method for blending between the source rotation and stance rotation on entering the stance. Options are: **Lerp, Slerp, EaseIn, EaseOut, EaseInOut, Spring, Bounce, Overshoot**.
In Time	Float	The time taken to enter the stance.

NAME	TYPE	DESCRIPTION
Out Position Blend	Dropdown	The easing method for blending between the stance position and idle position on exiting the stance. Options are: **Lerp, **EaseIn**, **EaseOut**, **EaseInOut**, **SwingAcross**, **SwingUp**, **Spring**, **Bounce** and **Overshoot**.
Out Rotation Blend	Dropdown	The easing method for blending between the stance rotation and Idle rotation on exiting the stance. Options are: **Lerp, **Slerp**, **EaseIn**, **EaseOut**, **EaseInOut**, **Spring**, **Bounce**, **Overshoot**.
Out Time	Float	The time taken to enter the stance.

See Also

Thrown Weapons

TimedExploder MonoBehaviour

Overview

The TimedExploder behaviour is attached to such as hand grenades, spawning an explosion after a set time. The item should be a PooledObject and will be returned to the pool when the explosion is spawned.

Inspector
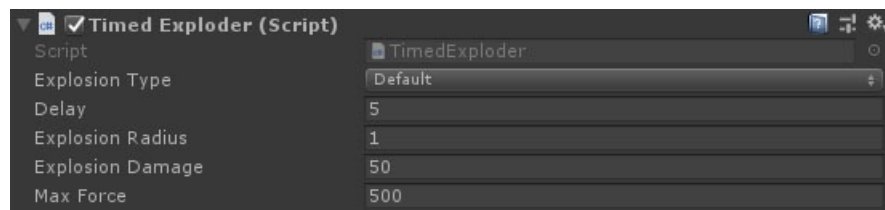
Properties

NAME	TYPE	DESCRIPTION
Explosion Type	ExplosionType	The id of the explosion to spawn.
Delay		The delay before exploding.
Explosion Radius		The radius of the explosion.
Explosion Damage		The damage the explosion does at its center.
Max Force		The max force to be imparted onto any objects in the explosion radius. The force falls off as distance from the center increases. Requires either a Rigidbody or an impact handler.

See Also

Explosions

UnityEventToolAction MonoBehaviour

Overview

The UnityEventToolAction behaviour is used to add a Unity event to a wieldable tool to aid in connecting to other systems.

Inspector
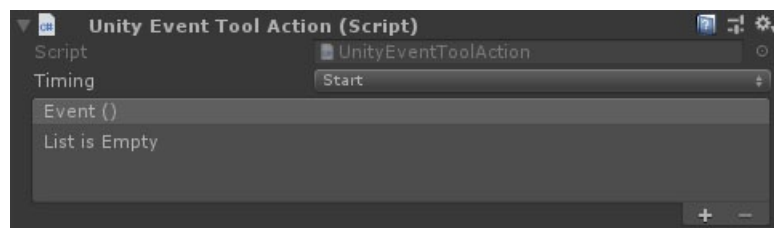
Properties

NAME	TYPE	DESCRIPTION
Timing	Dropdown	When should the event be triggered.
Event	UnityEvent	The event to fire.

See Also

Wieldable Tools

WeaponMoveAimer MonoBehaviour

Overview

The WeaponMoveAimer module moves the entire weapon up to align the center of the camera with the weapon sights.

In The Scene View

With a gameobject selected that uses a HeadMoveAimer component, the aimer guide handle will be visible in the scene view. This is used to represent the aim point of the aimer, with the blue arrow pointing forwards, and the green arrow pointing up.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Start Active	Boolean	Should this module register as the active aimer immediately on start.
Activation Mode	Dropdown	How does module activation work. Options are: **Component** means the component will be enabled / disabled - use this when you have multiple modules on an object such as the firearm root. **GameObject** activates / deactivates the object - use this for attachments where you want the geo to appear/disappear.
Aim Up Audio	AudioClip	An audio clip to play when the weapon is raised.
Aim Down Audio	AudioClip	An audio clip to play when the weapon is lowered.
Hip Accuracy Cap	Float	The highest accuracy the firearm can achieve while not aiming down sights.
Aimed Accuracy Cap	Float	The highest accuracy the firearm can achieve while aiming down sights.
Can Aim While Reloading	Boolean	Should the weapon be lowered when reloading or can it stay aimed.
On Aim Up	UnityEvent	An event called when the weapon is fully raised.
On Aim Down	UnityEvent	An event called when the weapon is fully lowered.
Aim Offset	Transform	A target aim transform. The weapon will be moved to align this transform to the camera when aiming down sights. Offsets are calculated on Awake, so moving this transform at after this point has no effect.
Aim Position	Vector3	The offset for the weapon transform to move to align the weapon sights with the camera. This property will only be visible if no **AimOffset** transform is set.
Aim Rotation	Vector3	An euler angle offset for the weapon to move to align the weapon sights with the camera. This property will only be visible if no **AimOffset** transform is set.
Fov Multiplier	Float	A multiplier for the camera FoV for aim zoom.
Input Multiplier	Float	A multiplier for the camera input when aiming down sights. By default, this value is locked to the FoV multiplier. Clicking the lock icon next to the 2 multipliers allows you to modify it separately for situations such as render texture scopes.
Aim Time	Float	The time it takes to reach full aim, or return to zero aim.

NAME	TYPE	DESCRIPTION
Transition	Dropdown	The transitions easing to apply. Note: custom transition requires inheriting a new class and overriding the custom transition methods. **Lerp** moves between states with uniform speed, **SwingUp** moves across and then up, **SwingAcross** moves up and then across, **EaseInOut** smoothes the beginning and end of the transition for a more natural feel, **Overshoot** moves past the target position and then back, **OvershootIn** overshoots when raising the weapon, but with a smoother return to lowered, **Spring** overshoots and wobbles back to position, **SpringIn** has a smoother return to lowered position, **Bounce** bounces off the target pose briefly, **BounceIn** has a smoother return to lowered, **Custom** uses virtual methods which you can override in a custom script.
Position Spring Multiplier	Float	A multiplier for procedural spring position animation on the weapon. Used to reduce movement when the firearm is close to the camera.
Rotation Spring Multiplier	Float	A multiplier for procedural spring rotation animation on the weapon. Used to reduce movement when the firearm is close to the camera.
Aim Anim Bool	String	The animator parameter key for a bool used to control aiming state in animations.
Block Trigger	Boolean	If true then the gun cannot fire while transitioning in and out of aim mode. This is used to prevent gunshots interrupting the animation. This property will only be shown if the Aim Anim Bool property is true.
Crosshair Up	FpsCrosshair	The crosshair to show when aiming down sights.
Crosshair Down	FpsCrosshair	The crosshair to show when not aiming down sights.

See Also

Modular Firearms

First Person Camera

WieldableFlashlight MonoBehaviour

Overview

The WieldableFlashlight behaviour is used to add a toggleable flashlight to firearms.

Inspector
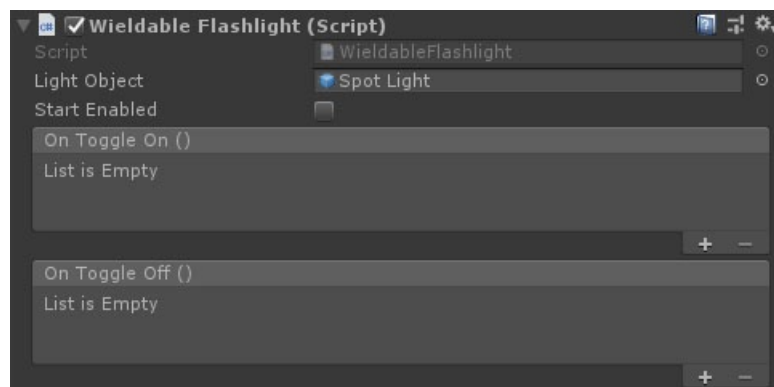
Properties

NAME	TYPE	DESCRIPTION
Light Object	GameObject	A child object with a light component attached.
Start Enabled	Boolean	Should the flashlight be on from the start.
On Toggle On	UnityEvent	An event fired when the laser is switched on.
On Toggle Off	UnityEvent	An event fired when the laser is switched off.

See Also

Modular Firearms

Attachments

WieldableLaserPointer MonoBehaviour

Overview

The WieldableLaserPointer behaviour is used to add a toggleable laser pointer to firearms with configurable colour and intensity.

Inspector
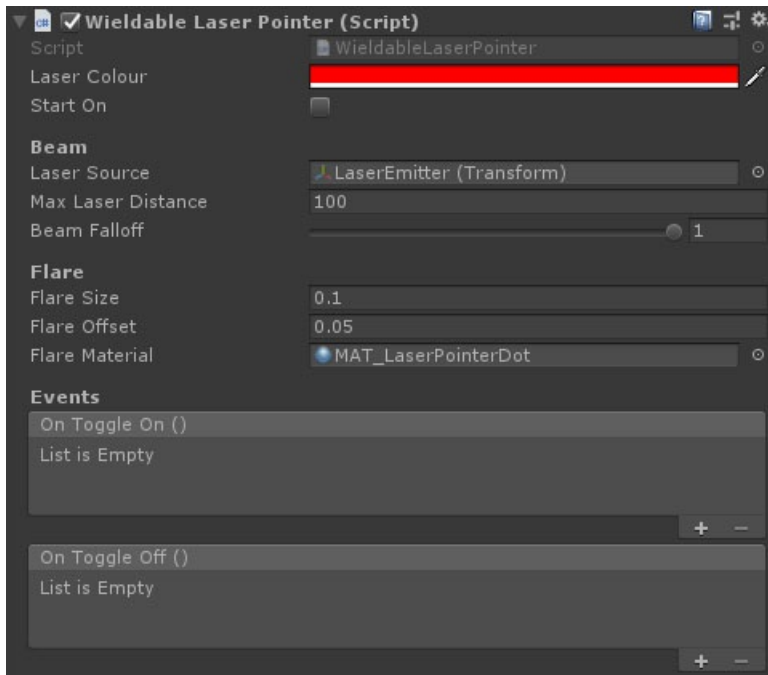
Properties

The WieldableLaserPointer properties are split into a number of sections:

NAME	TYPE	DESCRIPTION
Laser Colour	Color	The colour of the laser.
Start On	Boolean	Should the laser be on from start.

Beam

NAME	TYPE	DESCRIPTION
Laser Source	Transform	The transform the laser will be projected from (forwards).
Max Laser Distance	Float	The furthest distance the laser pointer will be visible.
Beam Falloff	Float	The alpha falloff of the beam over its length.

Flare

NAME	TYPE	DESCRIPTION
Flare Size	Float	The size of the laser hit point flare.
Flare Offset	Float	The distance the flare should be pushed forward towards the camera from the point of impact (prevents intersection with walls).

NAME	TYPE	DESCRIPTION
Flare Material	Material	The material to use for the laser impact flare. NeoFPS includes a shader for laser flares with variable intensity and colour.

Events

NAME	TYPE	DESCRIPTION
On Toggle On	UnityEvent	An event fired when the laser is switched on.
On Toggle Off	UnityEvent	An event fired when the laser is switched off.

See Also

Modular Firearms

Attachments

WieldableTool MonoBehaviour

Overview

The WieldableTool behaviour is used to create tools and items that are equipped and used in a character's hands. They have both a primary and secondary fire, and are built out of modules and actions.

Inspector
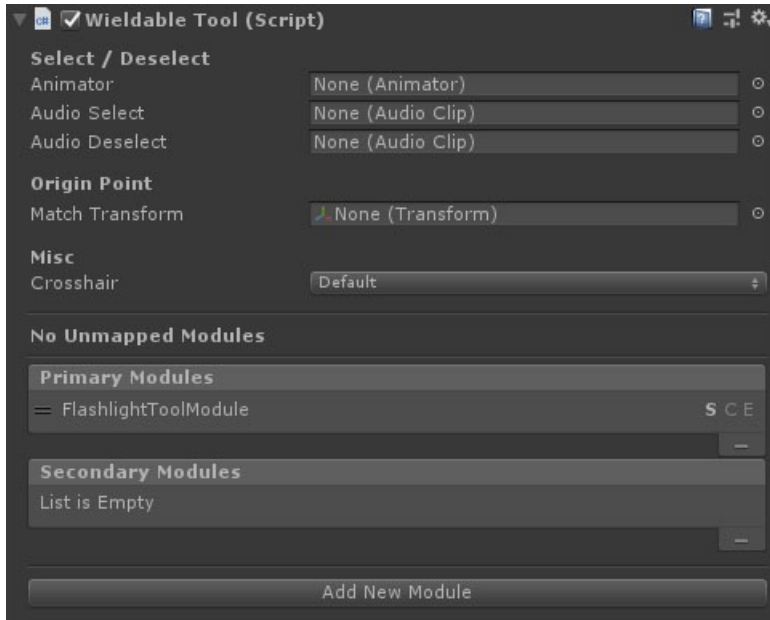
Properties

NAME	TYPE	DESCRIPTION
Animator	Animator	The animator component of the tool.
Anim Key Draw	String	The key for the AnimatorController trigger property that triggers the draw animation.
Draw Duration	Float	The time it takes to raise the tool. This option will only be visible when the anim key is set.
Anim Key Lower	String	The AnimatorController trigger key for the tool lower animation (blank = no animation).
Lower Duration	Float	The time taken to lower the item on deselection. This option will only be visible when the anim key is set.
Audio Select	AudioClip	The audio clip when raising the tool.
Audio Deselect	AudioClip	The audio clip when lowering the tool.
Crosshair	[FpsCrosshair] [3]	The crosshair to show when the weapon is drawn.

Unmapped Modules

The unmapped modules are the tool modules and actions that exist on this object but have not yet been assigned to either the primary fire. The buttons will assign the module to the relevant fire mode, or remove it from the object.

Primary and Secondary Modules

The primary and secondary modules array assign each of the attached modules to either the primary or secondary fire modes, including specifying the order they trigger. The **S**, **C** and **E** characters specify whether the module's actions are triggered on the fire mode's start (S), end (E) or continuously while the trigger is held (C). A number of modules have options you can set for which they should use.

Add New Module

The "Add New Module" button is a fast way to add a module component to the tool. Clicking it will list all the available module types for you to choose from.

Origin Point

The NeoFPS weapons assume that the camera is placed at the origin. For many assets or 3rd party weapons, the origin is at the character feet or hips and the camera is a child object of the weapon's hierarchy. To align the object to correctly work with NeoFPS you can drag the camera object of the weapon into the **Match Transform** field under the **Origin Point** heading. This will move everything below the spring object to match up.

See Also

Wieldable Tools

WieldableToolStanceManager MonoBehaviour

Overview

The WieldableToolStanceManager behaviour is used to specify poses or stances for a [wieldable tool](#). The entire tool will be moved to match the stance, and can optionally set an animator bool parameter too.

The stance will be temporarily exited when aiming and reloading.

Inspector

Properties

Use the **Add Stance** buttons to add a new stance to the manager.

Stances

Individual stances have the following properties:

NAME	TYPE	DESCRIPTION
Name	String	The name of the stance.
Animator Bool Key	String	An optional name of a bool parameter in the weapon's [Animator](#).
Position	Vector	The position to move the weapon to in this stance.
Rotation	Vector	The rotation of the weapon in this stance.
In Position Blend	Dropdown	The easing method for blending between the source position and stance position on entering the stance. Options are: **Lerp, EaseIn, EaseOut, EaseInOut, SwingAcross, SwingUp, Spring, Bounce** and **Overshoot**.
In Rotation Blend	Dropdown	The easing method for blending between the source rotation and stance rotation on entering the stance. Options are: **Lerp, Slerp, EaseIn, EaseOut, EaseInOut, Spring, Bounce, Overshoot**.
In Time	Float	The time taken to enter the stance.

NAME	TYPE	DESCRIPTION
Out Position Blend	Dropdown	The easing method for blending between the stance position and idle position on exiting the stance. Options are: **Lerp, **EaseIn**, **EaseOut**, **EaseInOut**, **SwingAcross**, **SwingUp**, **Spring**, **Bounce** and **Overshoot**.
Out Rotation Blend	Dropdown	The easing method for blending between the stance rotation and Idle rotation on exiting the stance. Options are: **Lerp, **Slerp**, **EaseIn**, **EaseOut**, **EaseInOut**, **Spring**, **Bounce**, **Overshoot**.
Out Time	Float	The time taken to enter the stance.

See Also

Wieldable Tools

SharedAmmoType ScriptableObject

Overview

The SharedAmmoType scriptable object is an ammo type that can be stored in the inventory and used by multiple firearms.

Inspector
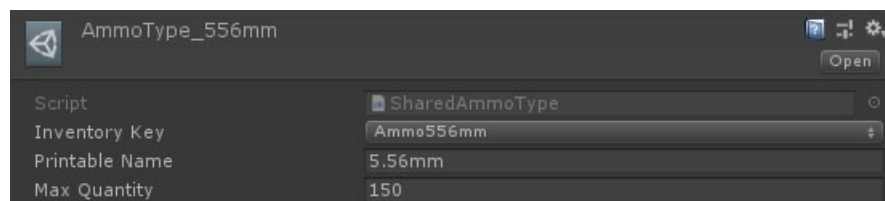
Properties

NAME	TYPE	DESCRIPTION
Inventory ID	ID Picker	The inventory item key. Clicking this button will open the inventory database item picker.
Printable Name	String	The name to be printed on the HUD.
Max Quantity	Int	The maximum quantity a character can carry.

See Also

Modular Firearms

FpsInventoryAmmo

SurfaceBulletPhysicsInfo ScriptableObject

Overview

The SurfaceBulletPhysicsInfo scriptable object specifies how bullets react to different surface types using a SurfaceBulletPhysicsAmmoEffect. Individual surfaces can have different settings for ricochet and penetration.

Inspector
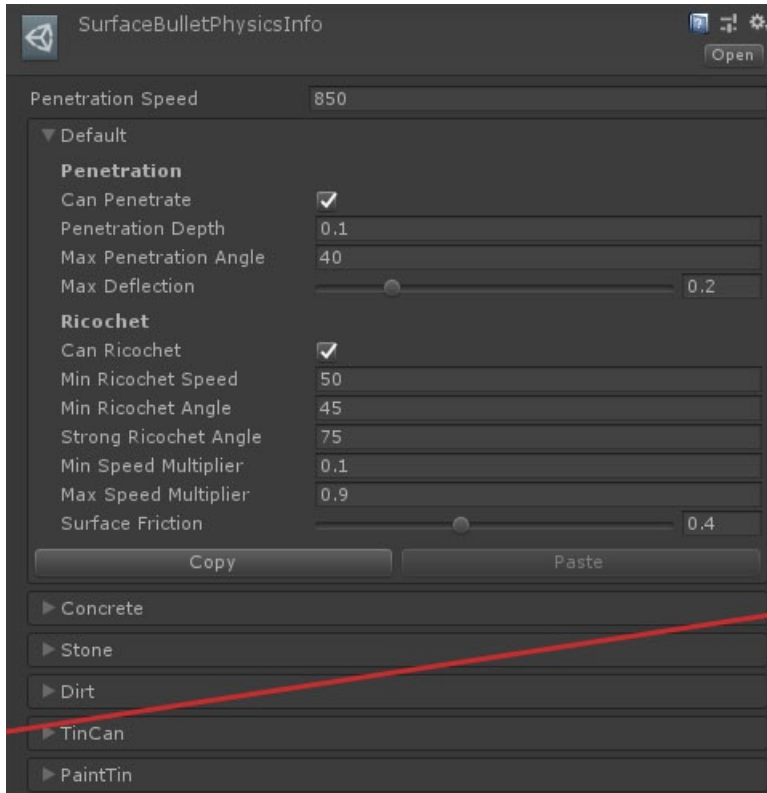
Properties

NAME	TYPE	DESCRIPTION
Penetration Speed	Float	The speed at which the penetration values below are accurate. As speed drops away, projectiles cannot penetrate as far.

Per-Surface Properties

NAME	TYPE	DESCRIPTION
Can Penetrate	Boolean	Can the projectile penetrate this surface type.
Penetration Depth	Float	The maximum depth the projectile can penetrate this surface when travelling at the **Penetration Speed** specified above.
Max Penetration Angle	Float	The maximum angle of incidence with the surface where a projectile can penetrate.
Max Deflection	Float	A 0-1 value representing how much the projectile direction can be altered by moving through the surface.
Can Ricochet	Boolean	Can the projectile ricochet off this surface.

NAME	TYPE	DESCRIPTION
Min Ricochet Speed	Float	The speed below which a projectile cannot ricochet.
Min Ricochet Angle	Float	The minimum angle of incidence that a projectile will ricochet. Below this and the projectile will penetrate or be destroyed.
Strong Ricochet Angle	Float	The angle of incidence where the projectile is deflected with the least speed loss.
Min Speed Multiplier	Float	The multiplier applied to the projectile speed when it hits at the minimum angle (as close to straight on as it can ricochet).
Max Speed Multiplier	Float	The multiplier applied to the projectile speed when it hits at the maximum glancing angle.
Surface Friction	Float	Surface friction is used to add randomness to the ricocheted projectile's direction. Lower friction means a more predictable ricochet.

See Also

Modular Firearms

FpsInventoryAmmo

Health and Damage

Overview

Character health and damage are handler by 2 elements in NeoFPS: A health manager and a number of damage handlers. Damage handlers react to damage from weapons and other scripts, modify that damage, and then pass it on to the health manager. For more information, see the BasicDamageHandler and EventDamageHandler.
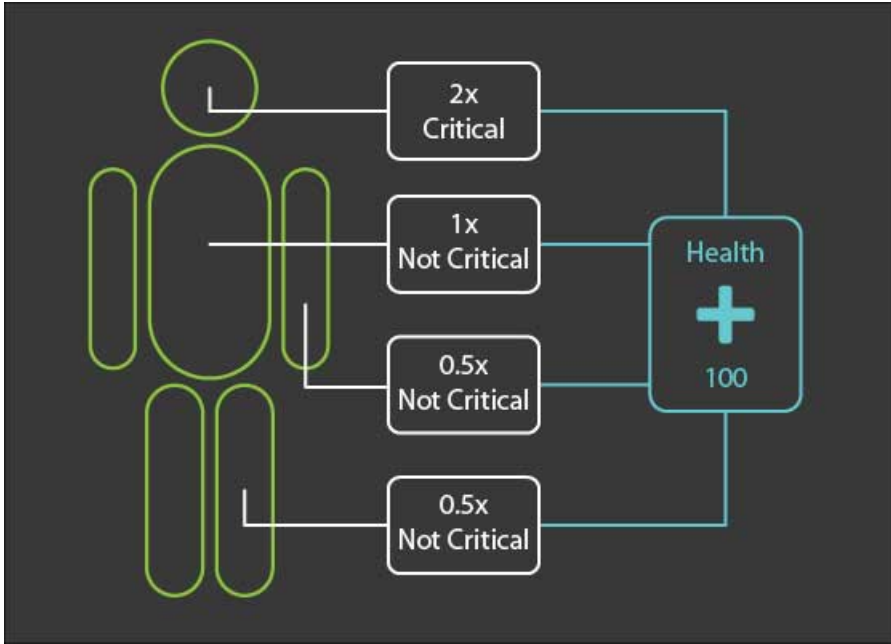
Incoming damage can have a source attached, which can be used to work out the position or direction damage came from as well as the type of damage.

Health Pickups

Characters can regain health by collecting health pickups. These have a number of options for how they heal, and can be applied to either interactive objects, or trigger based pickups. The simplest way to create a health pickup is using the Pickup Wizard in the NeoFPS Hub.

Shields & Armour

Alongside health, NeoFPS has systems for simple energy shields and inventory based armour. When the character or object takes damage, first shields will absorb that damage until they are depleted, then armour will mitigate some of the damage (and be destroyed in the process).

Shields require the character to have a ShieldSystem behaviour attached, and the damage handlers are replaced with ShieldedDamageHandlers. You can then specify a number of shield steps, each with a set charge. Damage is taken from each step in order, and if any step is not fully broken it will recharge after a set period. To restore broken steps you can use a ShieldPickup.

Armour is implemented by using ArmouredDamageHandler components. These allow you to specify the inventory key for an armour item, and then any damage will be (partially) absorbed and taken away from that inventory item instead.

If you want to use both shields and armour, you can use the ShieldedArmouredDamageHandler.

The simplest way to create shield booster pickups or armour (inventory item) pickups is using the Pickup Wizard in the NeoFPS Hub.

Healing and Damage Zones

You can define areas that heal or damage characters over time by using the HealZone and DamageZone behaviours.

Damage Filters

Damage in NeoFPS can be filtered by team and by type. This allows for gameplay mechanics such as ignoring friendly fire, or blocking damage from explosions but not bullets.

The damage filter can be used for up to 8 teams and 8 damage types.

Damage handlers have an incoming damage filter, while damage sources have an outgoing damage filter. In code a collision can be checked via the following code:

```
if (inDamageFilter.CollidesWith (sourceDamageFilter, friendlyFire))
{
    // React to damage here
}
```

The friendly fire parameter is a `bool`. If this is true then the team section of either filter will be ignored. If not then the team has to be valid as well as the damage type.

Damage Source

The damage source allows for feedback on the location of the source and, when combined with damage filters, effects such as returning damage to melee attackers.

The damage source specifies outgoing damage filter and the character controller if relevant.

See Also

ArmouredDamageHandler MonoBehaviour

Overview

The ArmouredDamageHandler behaviour adds an armour system to the BasicDamageHandler. Armour will mitigate some or all of the damage received by the damage handler, but it is consumed in the process. Armour is an inventory item that is picked up in the scene.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Multiplier	Float	The value to multiply any incoming damage by. Use to reduce damage to areas like feet, or raise it for areas like the head.
Critical	Boolean	Does the damage count as critical. Used to change the feedback for the damage taker and dealer.
Inventory Key	FpsInventoryKey	The inventory key of the armour type.
Damage Mitigation	Float	The amount of damage the armour should nullify.
Armour Damage Multiplier	Float	A multiplier used to modify how much armour is destroyed by the incoming damage.

See Also

Health and Damage

BasicDamageHandler MonoBehaviour

Overview

The BasicDamageHandler behaviour takes damage and passes it to a parent HealthManager

Inspector
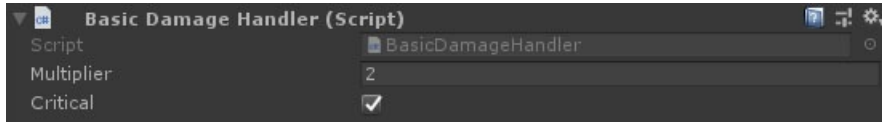
Properties

NAME	TYPE	DESCRIPTION
Multiplier	Float	The value to multiply any incoming damage by. Use to reduce damage to areas like feet, or raise it for areas like the head.
Critical	Boolean	Does the damage count as critical. Used to change the feedback for the damage taker and dealer.

See Also

Health and Damage

BasicHealthManager MonoBehaviour

Overview

The BasicHealthManager behaviour manages a character's health, firing events when health changes and on death.

Inspector
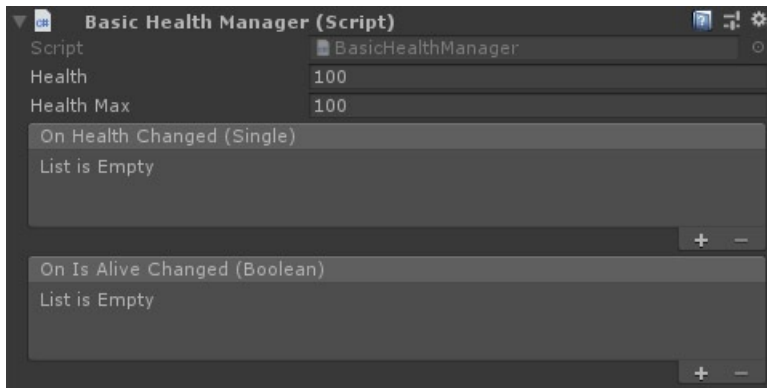
Properties

NAME	TYPE	DESCRIPTION
Health	Float	The starting health of the character.
Health Max	Float	The maximum health of the character.
Can Damage Self	Boolean	Can the character damage itself (eg with explosives).
On Health Changed	Unity Event	An event called whenever the health changes.
On Is Alive Changed	Unity Event	An event called whenever the alive state of the health manager changes.

See Also

Health and Damage

DamageZone MonoBehaviour

Overview

The DamageZone behaviour applies damage to a character over time whilst they are inside it. It is used alongside a CharacterTriggerZone, so it should be placed on a GameObject on the **TriggerZones** layer with a collider with **IsTrigger** set to true.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Damage Per Second	Float	The amount of damage to apply to the player character per second.
Damage Type	DamageType	The type of damage to apply.
Damage Description	String	A description of the damage to use in logs, etc.

See Also

Health and Damage

EventDamageHandler MonoBehaviour

Overview

The EventDamageHandler behaviour takes damage and passes it to a parent HealthManager as well as invoking a damage event.

Inspector
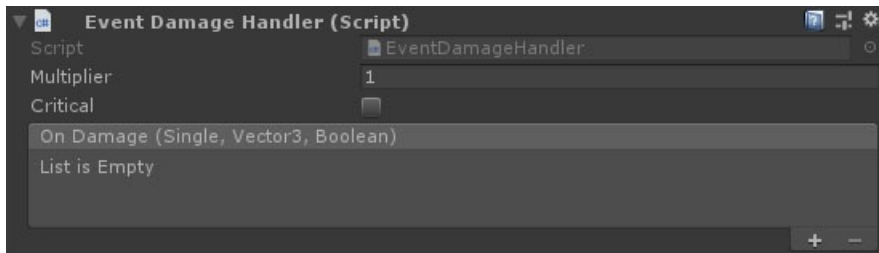
Properties

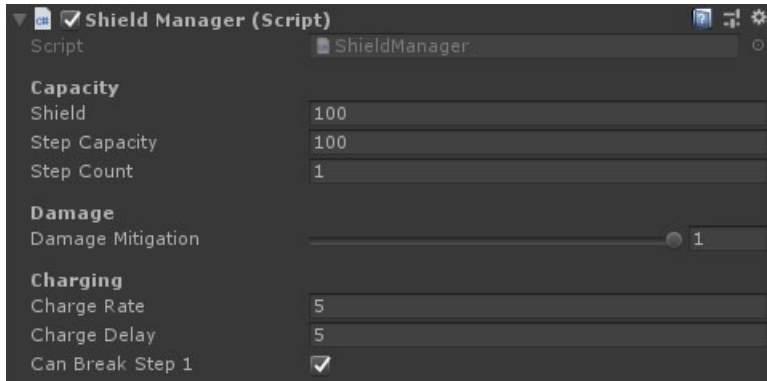| NAME	TYPE	DESCRIPTION
Multiplier	Float	The value to multiply any incoming damage by. Use to reduce damage to areas like feet, or raise it for areas like the head.
Critical	Boolean	Does the damage count as critical. Used to change the feedback for the damage taker and dealer.
On Damage	UnityEvent	An event that is invoked when damage is taken. The parameters are: `float damage, Vector3 direction, bool critical`

See Also

Health and Damage

Unity Events

HealthPickup MonoBehaviour

Overview

The HealthPickup behaviour restores health to the character that picks it up. It can be set to provide a set amount of healing, or some factor of the character's total or missing health.

Inspector
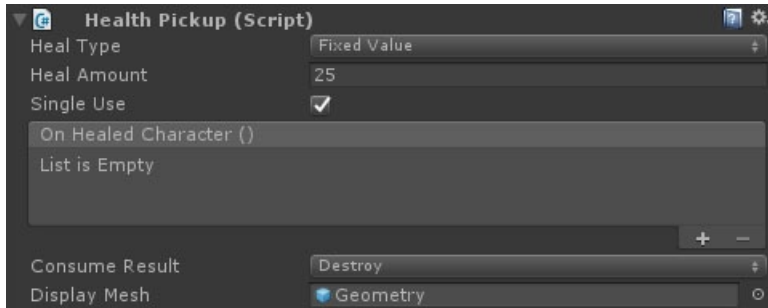
Properties

NAME	TYPE	DESCRIPTION
Heal Type	Dropdown	How the heal is applied. **FixedValue** adds the health. **Factor** adds amount * max health. **MissingFactor** adds amount * missing health.
Heal Amount	Float	The amount to heal by. If heal type is **Factor** or **MissingFactor**, this will be a value between 0 and 1.
On Healed Character	[UnityEvent](#)	An event called when the pickup heals a character.
Single Use	Float	If the character needs less healing than the pickup amount, should the pickup still be destroyed or should the remainder be available to use again?
Consume Result	Dropdown	What to do to the pickup object once its item has been used (fully, or single use). Available options are **Destroy**, **Disable** and **Respawn**.
Respawn Duration*	Float	How long to wait before respawning if the consume result is set to **Respawn**.
Display Mesh	GameObject	The object containing the display mesh of the pickup. This should be a child of the object this behaviour is applied to, so that if this is disabled the pickup will still respawn if required.

* This property is only visible if the consume result is set to Respawn.

See Also

[Health and Damage](#)

HealZone MonoBehaviour

Overview

The HealZone behaviour restores health to a character over time whilst they are inside it. It is used alongside a CharacterTriggerZone, so it should be placed on a GameObject on the **TriggerZones** layer with a collider with **IsTrigger** set to true.

Inspector
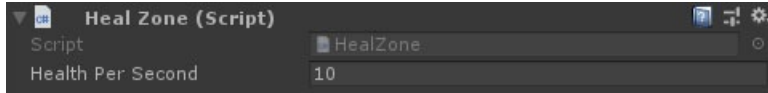
Properties

NAME	TYPE	DESCRIPTION
Health Per Second	Float	The amount of health to apply to the player character per second.

See Also

Health and Damage

RechargingHealthManager MonoBehaviour

Overview

The RechargingHealthManager behaviour will recharge to full health, with a set delay after being damaged. New damage interrupts the healing and resets the timer.

Inspector
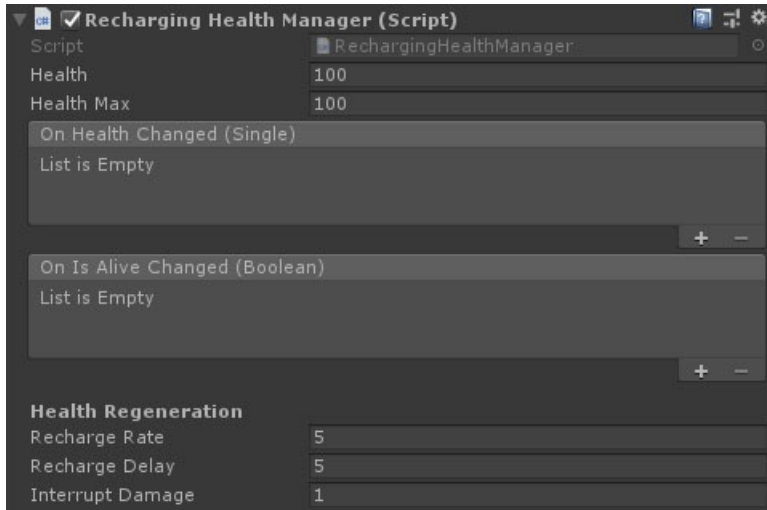
Properties

NAME	TYPE	DESCRIPTION
Health	Float	The starting health of the character.
Health Max	Float	The maximum health of the character.
On Health Changed	Unity Event	An event called whenever the health changes.
On Is Alive Changed	Unity Event	An event called whenever the alive state of the health manager changes.
Recharge Rate	Float	The recharge speed for health regeneration.
Recharge Delay	Float	The delay between taking damage and starting health regen.
Interrupt Damage	Float	Health recharge will be interrupted if damage greater than this amount is received.

See Also

Health and Damage

ShieldedArmouredDamageHandler MonoBehaviour

Overview

The ShieldedArmouredDamageHandler behaviour combines the ShieldedDamageHandler and ArmouredDamageHandler systems. Damage is first absorbed by the character's shields. Any damage that is not shielded (because the shields have been depleted) is then mitigated by the armour, before being passed to the character's health manager.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Multiplier	Float	The value to multiply any incoming damage by. Use to reduce damage to areas like feet, or raise it for areas like the head.
Critical	Boolean	Does the damage count as critical. Used to change the feedback for the damage taker and dealer.
Inventory Key	[FpsInventoryKey] [4]	The inventory key of the armour type.
Damage Mitigation	Float	The amount of damage the armour should nullify.
Armour Damage Multiplier	Float	A multiplier used to modify how much armour is destroyed by the incoming damage.

See Also

Health and Damage

ShieldedDamageHandler MonoBehaviour

Overview

The ShieldedDamageHandler behaviour adds an energy shield system to the BasicDamageHandler. Energy shields completely absorb damage until they are broken. The shield capacity and charge is handled by a ShieldManager behaviour on the character.

Inspector
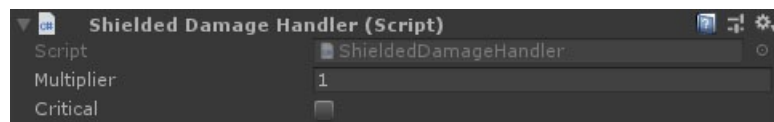
Properties

NAME	TYPE	DESCRIPTION
Multiplier	Float	The value to multiply any incoming damage by. Use to reduce damage to areas like feet, or raise it for areas like the head.
Critical	Boolean	Does the damage count as critical. Used to change the feedback for the damage taker and dealer.

See Also

Health and Damage

ShieldManager MonoBehaviour

Overview

The ShieldManager behaviour is used to track a character's energy shields. Shields completely absorb damage via the ShieldedDamageHandler and ShieldedArmouredDamageHandler, but are depleted in the process. A character's shields are split into steps. The steps automatically recharge if partially depleted after a short delay. If a shield step is completely depleted, then that step is broken and will not recharge. You will need to collect a ShieldPickup to restore broken steps.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Shield	Float	The starting shield amount.
Step Capacity	Float	The shield capacity of each shield step / block.
Step Count	Integer	The number of shield steps / blocks.
Damage Mitigation	Float	The amount of damage (multiplier) that the shield negates.
Charge Rate	Float	The recharge speed for shield regeneration.
Charge Delay	Float	The delay between taking damage and starting shield regen.
Can Break Step 1	Bolean	Shield steps only recharge if the shield value is greater than their starting level. If this property is false, step 1 will always recharge, even if it hits zero.

See Also

Health and Damage

ShieldPickup MonoBehaviour

Overview

The ShieldPickup behaviour is used to restore broken shield steps to a ShieldManager.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Step Count	Integer	The number of shield steps to recharge.
On Shields Recharged	UnityEvent	An event called when the pickup heals a character.
Single Use	Boolean	If the character needs less recharging than the pickup amount, should the pickup still be destroyed or should the remainder be available to use again?
Consume Result	Dropdown	What to do to the pickup object once its item has been used (fully, or single use). Options are **Destroy** to destroy the whole object, **Disable** to disable the render mesh and trigger so it can be reset later, and **Respawn** to disable and the reset automatically after a brief wait.
Respawn Duration	Float	How long to wait before respawning if the consume result is set to **Respawn**.
Display Mesh	GameObject	The display mesh of the pickup. This should not be the same game object as this, so that if this is disabled the pickup will still respawn if required.

See Also

Health and Damage

The Player HUD

Overview

The player heads up display is made up of a number of UI elements that each present specific information about the player character or their inventory.

The Player Character Watcher

Most of the HUD elements subcsribe to a player character watcher behaviour that is added to an object in their parent hierarchy. This is a behaviour that implements the `IPlayerCharacterWatcher` interface and tells the HUD elements when the player character changes so that they can attach to the relevant property or system attached to the character. If a behaviour that implements this interface is not found, then the HUD elements will log an error to the console on start. The NeoFPS samples use the SoloPlayerCharacterEventWatcher behaviour, though you are free to implement your own if required.

HUD Layout

The demo HUD makes use of Unity's UI auto layout components to properly order items. This means that when a player character is spawned that does not have a certain feature attached or enabled, then the relevant HUD item will be hidden, and the other HUD elements in its group will be positioned to fill the gap.

HUD Hider

The HudHider is used to hide the HUD elements when the player is not looking through a first person camera. This can be used for things like cutscenes.

Included HUD elements

The following HUD elements are included with NeoFPS for direct use or for reference when designing your own player HUD.

Health Counter

The health counter shows how much health your character currently has. For more information see HudHealthCounter.

Shield Meter

![Shield Meter](../images/screenshot-hud-shield meter.jpg)

The shield meter shows the status of the player character's energy shields. These can be split into multiple shield steps which recharge unless broken. For more information see HudShieldMeter.

Damage Markers

The damage markers highlight which direction damage comes from. The relevant edge of the screen is highlighted and then fades out over time. For more information see HudDamageMarkers.

Death Popup
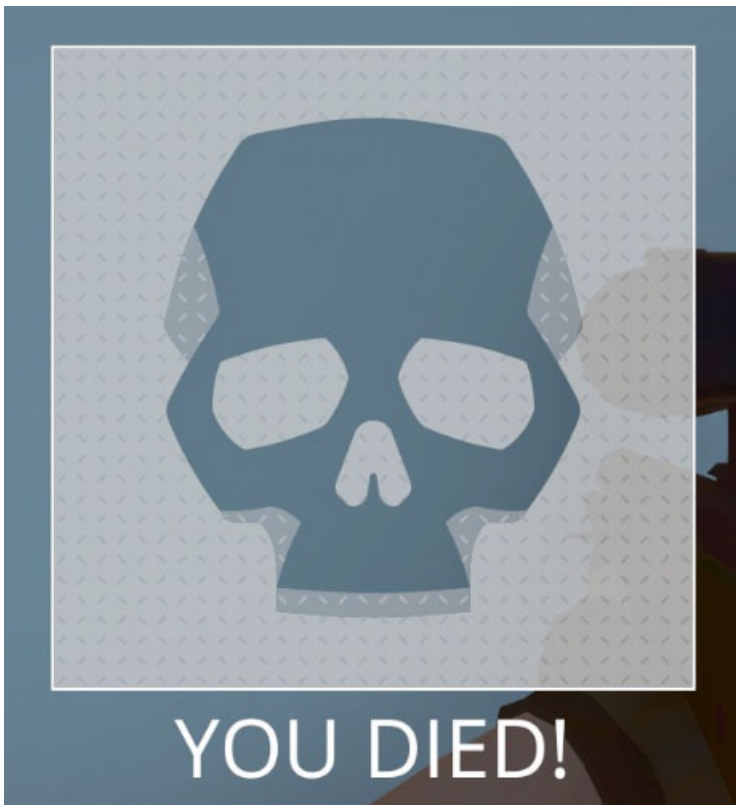
The death popup appears when the player character is killed and before they respawn. For more information see HudDeathPopup.

Ammo Counter
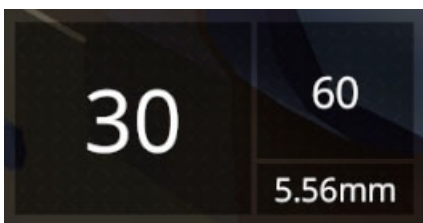
The ammo counter shows the current and available ammo along with the ammo type and weapon mode. For more information see HudAmmoCounter.

Firearm Mode
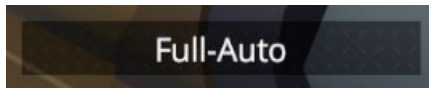
The firearm mode readout will appear when the player selects a firearm that has multiple modes. It displays the current mode. For more information see HudAmmoCounter.

Crosshair
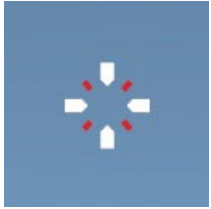
The crosshair is used for aiming and expands and contracts based on the current accuracy of the weapon. Multiple crosshairs can be used and referenced by ID. For more information see HudCrosshair.

You can also use the advanced crosshair system which has a more extensible system for drawing individual crosshairs, and adds hit and critical hit markers. For more information see HudAdvancedCrosshair.

Scopes
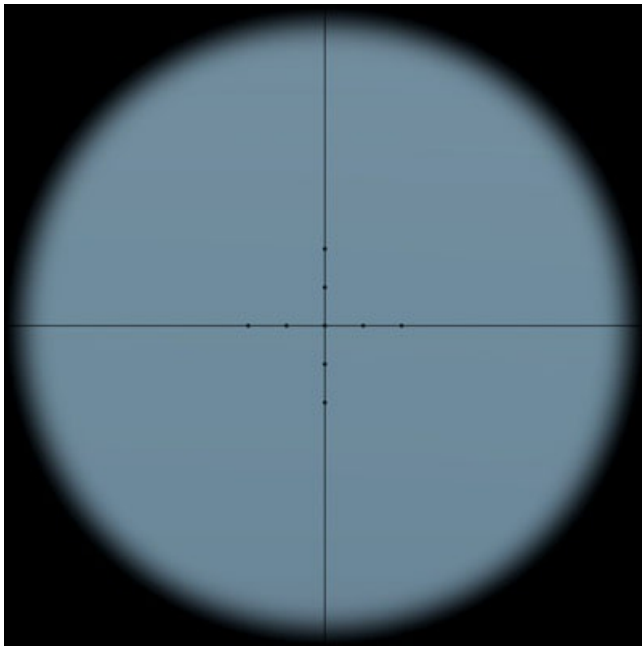
The scope is an image overlay that is used for certain weapons. Multiple scope images can be used and are referenced by ID. For more information see HudScope.

Interaction Tooltip
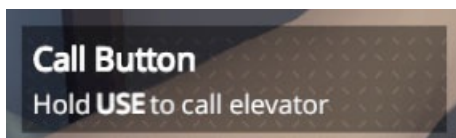
The interaction tooltip displays the name of the interactive object you are currently aiming at, along with the action performed when you interact with it. For more information see HudInteractionTooltip.

Progress Bar

The progress bar is used to show the interaction progress for items that require you to hold down the interact control. For more information see HudProgressBar.

Stamina Bar

The stamina bar shows how much health your character currently has. For more information see HudStaminaBar.

Inventory - Standard

The standard inventory is the kind of inventory you would see in a classic PC FPS. Each weapon has a set slot in a sequence that matches the number keys. This HUD inventory is also used for the swappable inventory. For more information see HudInventoryStandardPC and HudInventoryItemStandard.

Inventory - Stacked

The stacked inventory groups items together into stacks of a similar type. This is the inventory used in FPS games such as Half-Life. For more information see HudInventoryStackedPC, HudInventoryItemStacked and HudInventoryStackedSlot.

Meters and Counters
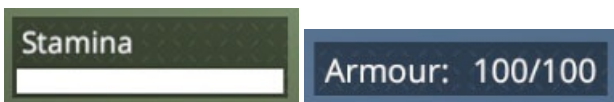
A number of different HUD meters and counnters are provided to track various properties. These include:

BEHAVIOUR	DESCRIPTION
HudInventoryItemCounter	Shows the quantity of a specific item in the player character's inventory.
HudInventoryItemMeter	Shows the quantity vs max quantity of a specific item in the player character's inventory.

BEHAVIOUR	DESCRIPTION
HudMotionGraphParameterMeter	Shows the value in the relevant motion graph parameter on the character as a meter based on a set range.
HudMotionGraphParameterReadout	Shows the value in the relevant motion graph parameter on the character.
HudOxygenMeter	Shows the amount of oxygen remaining while swimming.
HudSlowMoCharge	Shows the charge of the slow-mo system. Sometimes called focus or adrenaline.
HudStaminaBar	Shows the player character's stamina / fatigue levels.

Target Lock

The target lock HUD shows icons over objects in the world that are locked onto. Locks can be partial (a second icon overlays and shows the lock progress) and full (only a location marker is shown). For more information see HudTargetLockMarkers and HudTargetLock

# See Also

Health and Damage

Modular Firearms

Inventory

HudAdvancedCrosshair MonoBehaviour

Overview

The HudAdvancedCrosshair behaviour displays a Unity UI based crosshair when an appropriate wieldable is selected, expanding based on accuracy. It can also show a hit marker briefly when the player deals damage to something, including highlighting the marker for critical hits.

Inspector
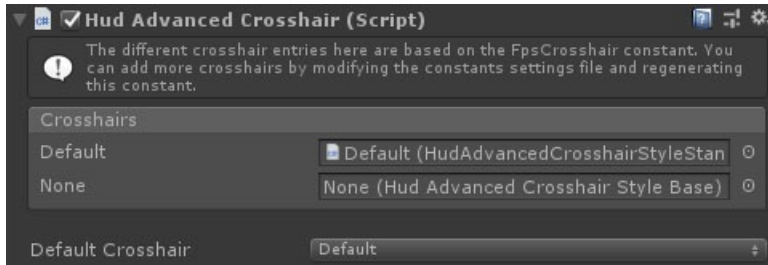
Properties

NAME	TYPE	DESCRIPTION
Crosshairs	HudAdvancedCrosshairStyleStandard Array	The individual crosshairs.
Default Crosshair	FpsCrosshair	The default crosshair to use (when no valid crosshair driver is being wielded).

See Also

Weapons

Unity UI

HudAdvancedCrosshairStyleStandard MonoBehaviour

Overview

The HudAdvancedCrosshairStyleStandard behaviour represents a specific crosshair style for the HudAdvancedCrosshair behaviour. It handles animating the hit marker and changing its size based on accuracy. You can also write your own crosshair styles for more complex effects if desired.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Crosshair Rect	RectTransform	The parent rect transform of the crosshairs (will be expanded and contracted based on accuracy).
Max Accuracy Size	Float	The size the UI element will reach at 100% accuracy.
Min Accuracy Size	Float	The size the UI element will reach at 0% accuracy.
m_OnlyShowCriticals	Boolean	Should the hit marker only show critical hits, or any hit that dealt damage.
m_CritColour	Color	The colour of the hit markers if for critical hits. Non-critical will use the crosshair colour.
m_HitmarkerDuration	Float	The amount of time the hit marker will be visible.
m_Animation	Dropdown	The animation easing function of the hit marker size. Available options are: **Lerp**, **EaseIn**, **EaseOut**, **EaseInOut**, **Spring** and **Bounce**.
m_HitMarkerRect	RectTransform	The parent rect transform of the hit marker.
m_HitStartSize	Float	The starting size of the hit marker.
m_HitEndSize	Float	The size of the hit marker just before it vanishes.

See Also

HudAdvancedCrosshair

Unity UI

HudAmmoCounter MonoBehaviour

Overview

The HudAmmoCounter behaviour is a simple [Unity UI](#) based counter for firearm ammo or item stack counts. It also displays a description of the firearm's ammo type.

Inspector
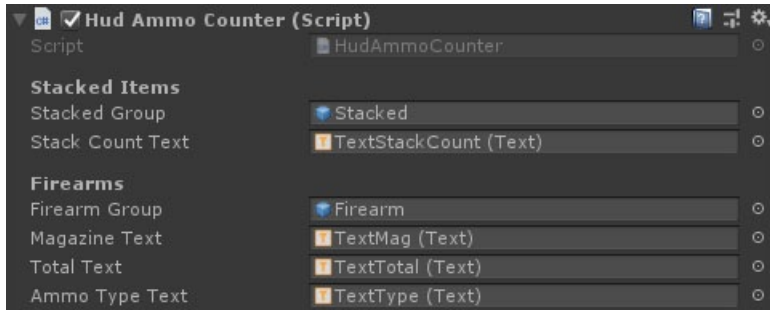
Properties

NAME	TYPE	DESCRIPTION
Stacked Group	GameObject	The group used for wieldable item stacks such as hand grenades.
Stack Count Text	Text	The stack count text entry for displaying count and total.
Firearm Group	GameObject	The group used for firearms.
Magazine Text	Text	The text entry for the current ammo in the firearm magazine.
Total Text	Text	The text entry for the total ammo the character is carrying.
Ammo Type Text	Text	The text entry for displaying the ammo type.

See Also

[Modular Firearms](#)

[Unity UI](#)

HudCrosshair MonoBehaviour

Overview

The HudCrosshair behaviour displays a Unity UI based crosshair when an appropriate wieldable is selected, expanding based on accuracy.

Inspector
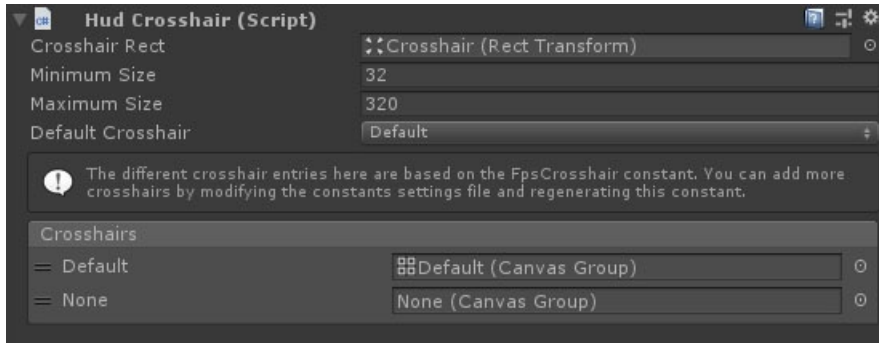
Properties

NAME	TYPE	DESCRIPTION
Crosshair Rect	RectTransform	The parent rect transform of the crosshairs (will be expanded and contracted based on accuracy).
Minimum Size	Float	The size the UI element will reach at 100% accuracy.
Maximum Size	Float	The size the UI element will reach at 0% accuracy.
Default Crosshair	FpsCrosshair	The crosshair to show by default.
Crosshairs	RectTransform Array	The individual crosshairs to match the FpsCrosshair generated constants values.

See Also

Modular Firearms

Unity UI

HudDamageMarkers MonoBehaviour

Overview

The HudDamageMarkers behaviour displays images at the edge of the screen based on the direction the player character takes damage from and then fades out over time.

Inspector
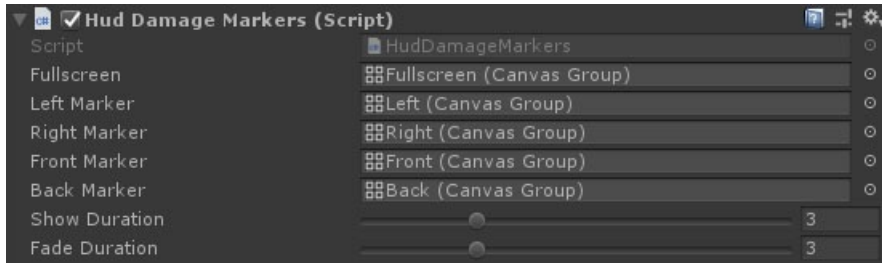
Properties

NAME	TYPE	DESCRIPTION
Fullscreen	CanvasGroup	The group for that encompasses all of the markers.
Left Marker	CanvasGroup	The group for fading out the left hand marker.
Right Marker	CanvasGroup	The group for fading out the right hand marker.
Front Marker	CanvasGroup	The group for fading out the front marker (top of the screen).
Back Marker	CanvasGroup	The group for fading out the back marker (bottom of the screen).
Show Duration	Float	How long should the markers remain fully visible.
Fade Duration	Float	How long do the markers take to fade out.

See Also

Health and Damage

Unity UI

HudDeathPopup MonoBehaviour

Overview

The HudDeathPopup behaviour is a simple Unity UI Image popup that displays when the player character dies.

Inspector
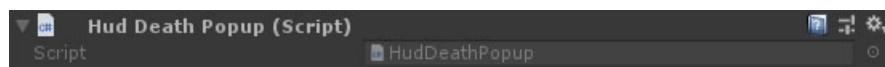
Properties

The HudDeathPopup behaviour has no properties exposed in the inspector.


See Also

Health and Damage

Unity UI

HudFirearmMode MonoBehaviour

Overview

The HudFirearmMode behaviour is used to display a description of the equipped firearm's current mode if appropriate.

Inspector
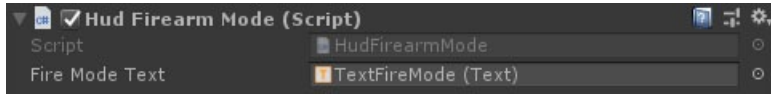
Properties

NAME	TYPE	DESCRIPTION
Fire Mode Text	Text	The text entry for displaying the weapon fire mode.

See Also

Modular Firearms

Unity UI

HudFIrearmOverheatBar MonoBehaviour

Overview

The HudFIrearmOverheatBar behaviour displays a bar showing the current heat of the selected weapon. If the weapon overheats, then it will show a warning. If the selected weapon does not have a FirearmOverheat behaviour attached then this HUD element will be disabled.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Bar Rect	RectTransform	The heat bar rect transform. This will be scaled along the x-axis based on the heat.
Cooldown Marker Rect	RectTransform	The rect transform of the cooldown marker. The cooldown marker shows the cooling threshold once a weapon has overheated. It will be hidden if the weapon cannot overheat, or if the threshold is 0.
Overheated Warning	GameObject	An object that is displayed while the weapon is overheating.

See Also

FirearmOverheat Monobehaviour

Unity UI

HudHealthCounter MonoBehaviour

Overview

The HudHealthCounter behaviour is a basic numeric Unity UI based health counter.

Inspector
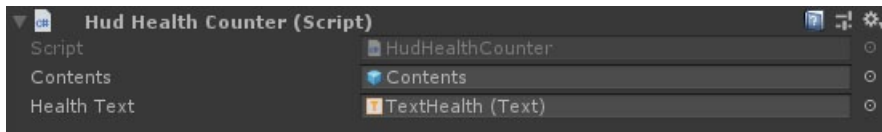
Properties

NAME	TYPE	DESCRIPTION
Contents	GameObject	The health counter game object (hidden when the player is not controlling a character.
Health Text	Text	The text readout for the current character health.

See Also

Health and Damage

Unity UI

HudHider MonoBehaviour

Overview

The HudHider behaviour is used to hide the HUD canvas when there is no first person camera active.

Inspector
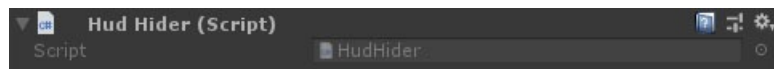
Properties

The HudHider behaviour has no properties exposed in the inspector.

See Also

[Health and Damage](#)

[Unity UI](#)

HudInteractionTooltip MonoBehaviour

Overview

The HudInteractionTooltip behaviour is used to display the name of the interactive object that the player character is aiming at, along with a simple description of what interacting with it does.

Inspector
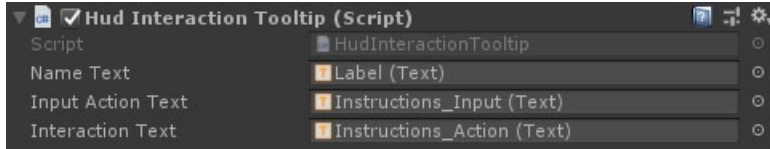
Properties

NAME	TYPE	DESCRIPTION
Name Text	Text	The UI text element to show the highlighted object's name.
Input Action Text	Text	The UI text element to show the input action required (press or hold).
Interaction Text	Text	The UI text element to show the interaction result (eg pick up).

See Also

Interactive Objects

Unity UI

HudInventoryItemCounter MonoBehaviour

Overview

The HudInventoryItemCounter behaviour displays a simple text counter showing the quantity of a specific item in the player character's inventory.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Counter Text	Text	The UI text element to output the quantity to.
Format	Dropdown	How to format the quantity. Available options are: **Quantity** prints the number of items, but hides itself when at 0; **QuantityAlwaysVisible** prints the number of items, including at 0; **QuantityOfMaximum** prints the stored number and maximum, eg. 1/10; **Percent** displays the quantity as a percentage of the maximum.

See Also

Inventory

Unity UI

HudInventoryItemMeter MonoBehaviour

Overview

The HudInventoryItemMeter behaviour displays a simple meter bar that represents the quantity vs max quantity of a specific item in the player character's inventory.

Inspector
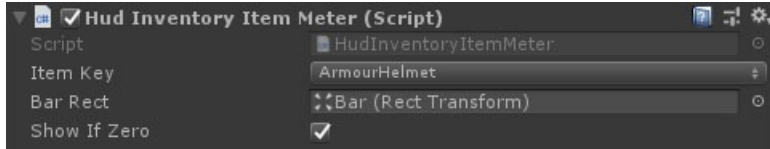
Properties

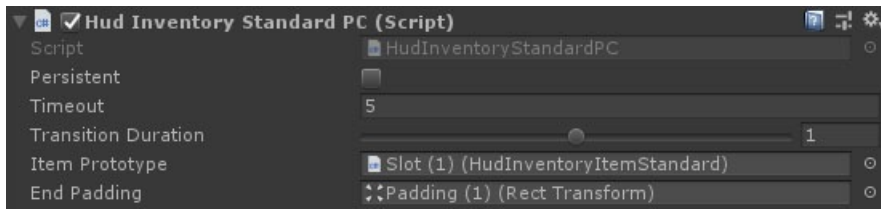| NAME	TYPE	DESCRIPTION
Bar Rect	RectTransform	The rect transform of the filled bar.

See Also

Inventory

Unity UI

HudInventoryStackedPC MonoBehaviour

Overview

The HudInventoryStackedPC behaviour represents the stacked inventory on the HUD. Layed out for keyboard users.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Persistent	Boolean	Does the HUD inventory stay visible at all times, or fade out?
Timeout	Float	The duration the HUD inventory stays fully visible before fading out.
Transition Duration	Float	The duration the fade out lasts.
Stack Prototype	HudInventoryStackedSlot	A prototype of a single quick-slot stack for duplicating.
End Padding	Transform	The padding transform to pad the layout group and push the item slots together.

See Also

HudInventoryStackedSlot

Inventory Examples

Unity UI

HudInventoryItemStacked MonoBehaviour

Overview

The HudInventoryStackedSlot behaviour represents an individual quick-slot item in the HudInventoryStackedPC.
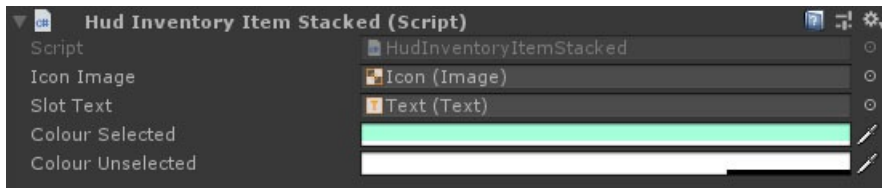
Inspector

Properties

NAME	TYPE	DESCRIPTION
Icon Image	Image	The UI image that is used to display the correct UI icon for the item.
Slot Text	Text	The UI text to display the slot number.
Colour Selected	Color	A colour for the item in the HUD when selected.
Colour Unselected	Color	A colour for the item in the HUD when not selected.

See Also

HudInventoryStackedPC

Inventory Examples

Unity UI

HudInventoryStackedSlot MonoBehaviour

Overview

The HudInventoryStackedSlot behaviour represents a quick-slot stack in the HudInventoryStackedPC.

Inspector
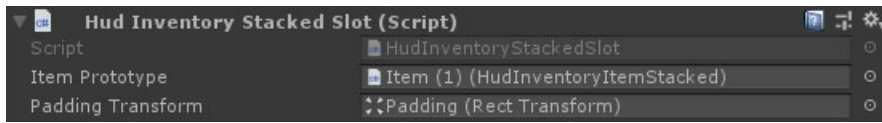
Properties

NAME	TYPE	DESCRIPTION
Item Prototype	HudInventoryItemStacked	The item slot prototype to duplicate for the stack.
Padding Transform	RectTransform	The padding transform to pad the layout group and push the item slots together.

See Also

HudInventoryItemStacked

HudInventoryStackedPC

Inventory Examples

Unity UI

HudInventoryStandardPC MonoBehaviour

Overview

The HudInventoryStandardPC behaviour represents the standard inventory on the HUD. Layed out for keyboard users.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Persistent	Boolean	Does the HUD inventory stay visible at all times, or fade out?
Timeout	Float	The duration the HUD inventory stays fully visible before fading out.
Transition Duration	Float	The duration the fade out lasts.
Item Prototype	HudInventoryItemStandard	A prototype of a single quick-slot item for duplicating.
End Padding	Transform	The padding transform to pad the layout group and push the item slots together.

See Also

HudInventoryItemStandard

Inventory Examples

Unity UI

HudInventoryItemStandard MonoBehaviour

Overview

The HudInventoryItemStandard represents a quick-slot item in the HudInventoryStandardPC.

Inspector
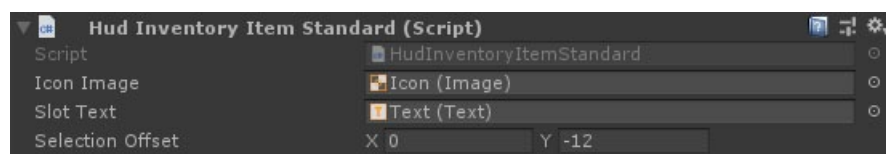
Properties

NAME	TYPE	DESCRIPTION
Icon Image	Image	The UI image that is used to display the correct UI icon for the item.
Slot Text	Text	The UI text to display the slot number.
Selection Offset	Vector2	An offset to highlight quick slot items when selected.

See Also

Inventory Examples

Unity UI

HudCrosshair MonoBehaviour

Overview

The HudMotionGraphParameterMeter behaviour displays is used to track a [parameter](#) on the player character's motion graph and display it as a meter bar on the [Unity UI](#). An example usage is for jetpack fuel on the jetpack character.

Inspector
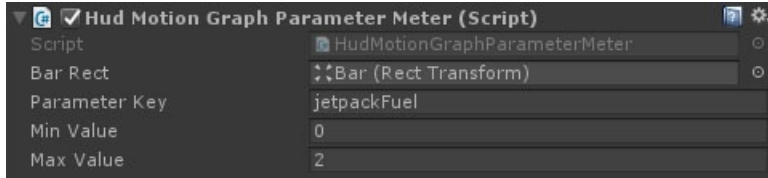
Properties

NAME	TYPE	DESCRIPTION
Bar Rect	[RectTransform][unity-recttransform]	The rect transform of the meter's bar. This will be scaled based on the value.
Parameter Key	String	The name of the motion graph [float parameter](#) to watch.
Min Value	Float	The minimum value at which the meter bar will have zero width.
Max Value	Float	The maximum value at which the meter bar will have reached its full width.

See Also

[Unity UI](#)

[Motion Graph Parameters And Data](#)

HudCrosshair MonoBehaviour

Overview

The HudMotionGraphParameterReadout behaviour displays is used to track a parameter on the player character's motion graph and display it as a text readout on the Unity UI.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Bar Rect	[RectTransform][unity-recttransform]	The rect transform of the meter's bar. This will be scaled based on the value.
Parameter Key	String	The name of the motion graph float parameter to watch.

See Also

Unity UI

Motion Graph Parameters And Data

HudOxygenMeter MonoBehaviour

Overview

The HudOxygenMeter behaviour displays a simple meter bar for the player character's oxygen level when swimming.

Inspector
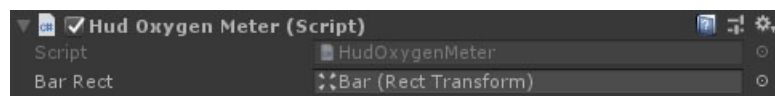
Properties

NAME	TYPE	DESCRIPTION
Bar Rect	RectTransform	The rect transform of the filled bar.

See Also

SlowMoSystem

Unity UI

HudProgressBar MonoBehaviour

Overview

The HudProgressBar behaviour is used to display a progress bar whent the character interacts with an interactive object in the world.

Inspector
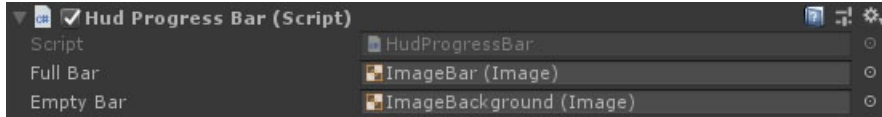
Properties

NAME	TYPE	DESCRIPTION
Full Bar	Image	The image for the completed progress (overlaps the empty bar. Should be scaled so 100 wide fills the empty bar.
Empty Bar	Image	The image for the empty bar.

See Also

Interactive Objects

Unity UI

HudScope MonoBehaviour

Overview

The HudScope behaviour displays a UI based scope.

Inspector
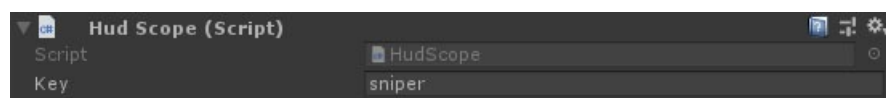
Properties

NAME	TYPE	DESCRIPTION
Key	String	The scope name. Used to allow different weapons to use different scopes.

See Also

[Modular Firearms](#)

[Unity UI](#)

HudShieldMeter MonoBehaviour

Overview

The HudShieldMeter behaviour displays a meter representing the player character's shield strength if it has a ShieldManager attached. Shields are broken into multiple recharging steps and the current step can be broken if fully depleted. The steps are represented in the HUD by the HudShieldMeterStep behaviour.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Step Prototype	HudShieldMeterStep	A shield meter step (in the object hierarchy) that can be duplicated for multiple steps.

See Also

ShieldManager

Unity UI

HudShieldMeterStep MonoBehaviour

Overview

The HudShieldMeterStep behaviour represents a single shield step for the HudShieldMeter behaviour.

Inspector
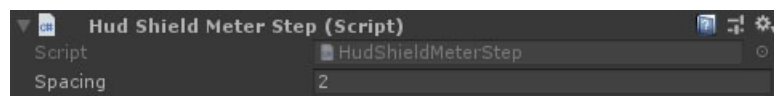
Properties

NAME	TYPE	DESCRIPTION
Spacing	Integer	The spacing between steps.

See Also

ShieldManager

Unity UI

HudSlowMoCharge MonoBehaviour

Overview

The HudSlowMoCharge behaviour displays a simple meter bar for the player character's slow-mo charge (sometimes called focus or adrenaline in games) if it detects they have a SlowMoSystem attached.

Inspector
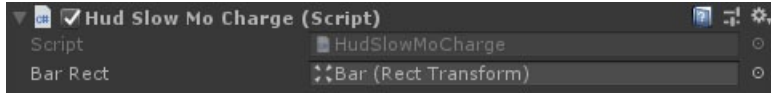
Properties

NAME	TYPE	DESCRIPTION
Bar Rect	RectTransform	The rect transform of the filled bar.

See Also

SlowMoSystem

Unity UI

HudStaminaBar MonoBehaviour

Overview

The HudStaminaBar behaviour is used to display the player character's stamina levels if it detects they have a StaminaSystem attached.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Bar Rect	RectTransform	The rect transform of the filled bar.

See Also

StaminaSystem

Unity UI

HudTargetLock MonoBehaviour

Overview

The HudTargetLock behaviour is represents a single target lock marker. It can be used for full and partial target locks. The markers are managed by the HudTargetLockMarkers behaviour.

Inspector
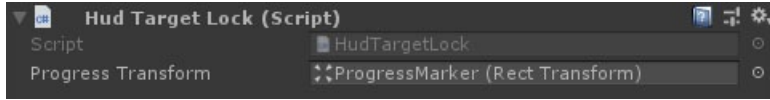
Properties

NAME	TYPE	DESCRIPTION
Progress Transform	RectTransform	The transform used to show the progress of the target lock. This size will reduce as the lock strength increases.

See Also

[The Player HUD1

Unity UI

HudTargetLockMarkers MonoBehaviour

Overview

The HudTargetLockMarkers behaviour connects to a player character or weapon's targeting system and displays target markers for each lock. Each target lock marker must have a script implementing the `ITargetLock` interface, such as the HudTargetLock behaviour.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Source	Dropdown	Where to get find the targeting system on the player character. **Weapon** checks the selected weapon. **Character** checks the root of the player character. **CharacterChildren** checks any child objects of the child character. If CharacterChildren is selected, then be careful not to use a targeting system on any weapons as these could conflict.

See Also

[The Player HUD1

Unity UI

HudToggle MonoBehaviour

Overview

The HudToggle behaviour is used to hide or show the HUD (useful for taking screenshots).

Inspector
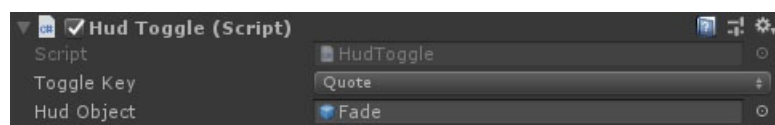
Properties

NAME	TYPE	DESCRIPTION
Toggle Key	Dropdown	The keyboard key which toggles the HUD visibility.
HUD Object	GameObject	The gameobject containing the HUD UI. Must be a child of this object.

See Also

[The Player HUD1

Unity UI

Save Games

Overview

NeoFPS uses a comprehensive but flexible system for saving game state which is designed to achieve the same effect you see in top single-player first person shooters. The player should be able to quick-save the game at any point without a perceptible freeze, and then re-load to the exact same state down to the level of bullets in flight and rigidbodies mid-fall.

The system can also be set up to save as much or as little information as desired. The mentioned quick-saves might be the desired effect for a narrative game, but a rogue-lite style FPS might want to save limited information on inventory and character skills only, while a first person platformer might want to use checkpoints that save the active spawn point and score.

Quick, Auto and Manual Saves

The save system has 2 user controlled save types that are enabled by default. On standalone builds, the player can quick-save using the **F5** key, and quick-load using the **F9** key (these can be remapped in the game options). The player can also manually save the game from the in-game menu and give the save a specific name to help understand what point in the game they had reached (commonly called a hard save).

In most modern FPS games it is also common to save the game state at specific points in a scene or after specific actions. The save system can trigger auto-saves via scripts, and includes an example behaviour called AutoSaveBehaviour.

Each of the above save types can be enabled or disabled in the SaveGameManager settings.

Save Browser

The NeoFPS samples provides a save game browser in the main menu, implemented in the NeoFPS sample UI system. The same functionality can be recreated fairly simply in another UI, though it requires some scripting knowledge.

The save browser lists all the of the available saves, along with the following details:

- **Name** - For manual saves, this can be set by the player. For quick and auto saves, this is set in the scene save info.
- **Save Type** - Quick, auto or manual save.
- **Save Date & Time** - The moment the save button was hit.
- **Thumbnail** - An image that represents the save. For quick and auto saves, the save system can take a screenshot on save, get a texture from the scene save info, or get a texture from the save game manager. Manual saves cannot use a screenshot (it would just be a menu).

There is also a version of the save browser in the in-game menu which only listss manual saves, and allows you to create a new manual save or choose one of the existing ones to overwrite.

From the main menu, you can also use the **Continue** button to load from the latest save (this can be set to specific types or all types in the manager).

Overrides and Persistence

Besides the standard game saves, the save game system can also be used for other purposes where serializing and deserializing scene or object date is required (either to disk or to a memory buffer). One example of this is persisting character data between scenes as the player progresses through the game. In this situation, the game mode tells the save manager to save the data for the player and character objects, and then reloads them when the new scene has loaded.

In the case of persistence, it is only desired to save and load specific data such as the character health and inventory. Information such as character position, aim and animation states could actually cause some major problems if they did not match the new scene. To achieve this, the NeoFPS save system supports a feature called overrides. You can specify different save modes, and then each object can override what information is saved for each mode.

For more information see Overrides and Persistence.

Controlling What Data Is Saved

The NeoFPS save system requires some setup for objects to control what information is saved and loaded. You can specify individual child objects and components to either include in or exclude from the save.

Components can come in 2 forms. Neo-serialized components are MonoBehaviours which implement the `INeoSerializableComponent` interface. This is the best method of serialization. If you want to serialize a component that you do not have control over the script for (such as an inbuilt Unity component, or a script from another asset), there must be a formatter created for it which understands what data to save and load.

For more information see Serializing Data.

Objects that are instantiated at runtime can also be recreated from their prefabs as long as that prefab is registered with either the save game manager or the scene save info for the scene it is instantiated in. Unity does not provide information on whether an object is being created as part of a scene load, or a runtime instantiation. This means that in order for the objects to be correctly registered in the scene hierarchy, there are some rules about how to instantiate prefabs and where.

For more information see Runtime Objects.

Troubleshooting

In the event that data is not being saved properly there are a few common mistakes to look out for, and the save game inspector can be used to examine saved data and check where the problem lies.

For more information see Troubleshooting Saves.

See Also

Serializing Data

Runtime Objects

Overrides And Persistence

Troubleshooting Saves

Serializing Data

Overview

The core of the NeoFPS save games system is the pairing of the SceneSaveInfo and NeoSerializedGameObject. The latter is used to identify which objects, along with which of their components, to save and load. The former is required for each saved scene and inside the save files it acts as the container for all the root serialized objects in the scene.

NeoSerializedGameObject

The NeoSerializedGameObject is placed on every object that needs to be saved, or that has components that need saving. On its own it can save the transform properties of the object along with its name. It is also used to filter all of the child objects, neo-serialized components and non-neo-serialized components.

Child Objects

By default, a NeoSerializedGameObject has its child object filter set to **Exclude**. This means that it **will save all** child NeoSerializedGameObjects, and to block a child from being saved, it must be added to the filter list. You can also set the filter to **Include**. This means that **none** of the child objects will be saved, unless they are specifically added to the filter list.

Child objects do not need to be direct children of the parent NeoSerializedGameObject. Other GameObjects can sit inbetween them in the hierarchy. The exception to this is when objects are instantiated at runtime. When the hierarchy is rebuilt while loading from save, any runtime instantiated objects will be recreated as direct child objects of their parent NeoSerializedGameObject. Therefore it is advisable to make sure that this relationship is in place from the start. For more information on the instantiation of objects at runtime, see Runtime Objects.

Neo-Serialized Components

Neo-serialized components are any monobehaviour which implements the `INeoSerializableComponent` interface, requiring the following 2 methods:

```
public void WriteProperties(INeoSerializer writer, NeoSerializedGameObject nsgo, SaveMode saveMode);
public void ReadProperties(INeoDeserializer reader, NeoSerializedGameObject nsgo);
```

The **nsgo** parameter is the object the component is attached to (useful if you want to store references to other components), while the **saveMode** parameter is used for overrides (see Overrides And Persistence).

Other Components

Components which do not implement the `INeoSerializableComponent` interface require a "formatter" to handle writing and reading their data. The following is an example implementation:

```
public class RigidbodyFormatter : NeoSerializationFormatter<Rigidbody>
{
  // Register the formatter with the save manager.
  [RuntimeInitializeOnLoadMethod(RuntimeInitializeLoadType.BeforeSceneLoad)]
#if UNITY_EDITOR // Required for the formatter to be recognised when selecting components to serialize
  [UnityEditor.InitializeOnLoadMethod]
#endif
  static void Register()
  {
    NeoSerializationFormatters.RegisterFormatter(new RigidbodyFormatter());
  }

  private static readonly NeoSerializationKey k_VelocityKey = new NeoSerializationKey("velocity");

  protected override void WriteProperties(INeoSerializer writer, Rigidbody from)
  {
    writer.WriteValue(k_VelocityKey, from.velocity);

    // Write other properties here...
  }

  protected override void ReadProperties(INeoDeserializer reader, Rigidbody to)
  {
    Vector3 v;
    if (reader.TryReadValue(k_VelocityKey, out v, Vector3.zero))
      to.velocity = v;

    // Read other properties here...
  }
}
```

Inside the `ReadProperties()` method you can also perform logic such as simulating a particle system or starting a coroutine.

Formatters allow you to save components which you do not have source control over, such as in-built Unity components or the scripts from other assets.

SceneSaveInfo

The SceneSaveInfo behaviour must be placed in each scene that needs to be saved. It allows you to specify the scene name, along with a thumbnail to show in the save browser if the manager is not set to use a screenshot. It also acts as a root container for each of the NeoSerializedGameObjects in the scene (though they should not be its child in the scene hierarchy).

The SceneSaveInfo component can also be used to register prefabs that can be instantiated at runtime and rebuilt by the save system. For more information on this see Runtime Objects.

The Binary Serializer / Deserializer

NeoFPS uses a custom written binary serializer and deserializer for save games. It is designed with the following principles in mind: it should be very fast to gather and write the save data. It should be very robust when reading the data.

Speed is important when writing saves so that the save system does not cause large frame hitches and interfere with the flow of the game. To achieve this, the serializer uses pre-allocated buffers and keeps the save size small by storing the values using a hash instead of a name. This is very different from save systems which prioritise human readability by saving in JSON or XML formats, but makes a huge difference to performance. The actual file writing from the buffers to the save file is performed on a separate thread to the rest of the game.

Robustness is most important when reading because there is no guarantee that the saved data will be up to date with the game when it is run. The player might save the game and then not return to it until a long time has passed and the game has received multiple content updates. The deserializer builds a map of all the properties that is queried by the gameobjects and components in the scene immediately after the scene itself has loaded. Firstly, any objects are destroyed or instantiated as required to recreate the hierarchy in the save file (objects are only destroyed here if they were destroyed in the save so new content is not affected).

After that, all the objects and their components are asked to take the data they need from the save data in turn. This allows the components to handle missing data gracefully while redundant data will just be ignored. Speed is also less important at the point of deserialization as it is usually behind a loading screen and not during ongoing play.

The custom serializer uses what's called "unsafe" code to achieve high performance. This means that it uses pointers to move around the buffers and break values into individual bytes. This requires the code to be placed in an assembly definition which enables unsafe code for its contents. This also means that the serializer cannot work for WebGL builds. For this situation, another serializer / deserializer combo has been created called the SafeSerializer. This relies on the .NET binary formatter and streams. It has a higher performance overhead than the custom version but is able to run on all platforms.

Property Keys And Hashes

For the best performance, when writing your own formatter and `INeoSerializableComponent` scripts, you can store the property key hashes instead of saving and loading using the key names. If you save or load a value using a string then it will be hashed there and then. If this is done thousands of times in a frame then it can be a very expensive process. To make storing the hash easier, you can use the `NeoSerializationKey` helper class. Some examples of writing a value are as follows:

```
public class ExampleNeoComponent : MonoBehaviour, INeoSerializableComponent
{
 // Create a static readonly key helper once which can be used multiple times
 private static readonly NeoSerializationKey k_PropertyKey = new NeoSerializationKey("propertyName");

 public int propertyValue = 5;

 public void WriteProperties(INeoSerializer writer, NeoSerializedGameObject nsgo, SaveMode saveMode)
 {
  // Use the property name directly. Easiest, and good for prototyping, but poor performance
  writer.WriteValue("propertyName", v);

  // Hash the property name. This is what the serializer does behind the scenes in the previous case
  writer.WriteValue(NeoSerializationUtilities.StringToHash("propertyName"), v);

  // Use the key that was hashed once on creation. Best performance
  writer.WriteValue(k_PropertyKey, v);
 }

 // ...
}
```

Component And Object References

The save system can serialize references to other NeoSerializedGameObjects or the components on them. It cannot serialize references to components on an object that does not have a NeoSerializedGameObject component attached and is not also serialized.

The following helper functions exist to aid in saving or loading references:

```
public class ExampleNeoComponent : MonoBehaviour, INeoSerializableComponent
{
 public GameObject testGameObject;
 public NeoSerializedGameObject testNsgo;
 public Transform testTransform;
 public NeoComponent testComponent;

 public void WriteProperties(INeoSerializer writer, NeoSerializedGameObject nsgo, SaveMode saveMode)
 {
  writer.WriteGameObjectReference("testGameObject", testGameObject, nsgo);
  writer.WriteNeoSerializedGameObjectReference("testNsgo", testNsgo, nsgo);
  writer.WriteTransformReference("testTransform", testTransform, nsgo);
  writer.WriteComponentReference("testComponent", testComponent, nsgo);
 }

 public void ReadProperties(INeoDeserializer reader, NeoSerializedGameObject nsgo)
 {
  reader.TryReadGameObjectReference("testGameObject", out testGameObject, nsgo);
  reader.TryReadNeoSerializedGameObjectReference("testNsgo", out testNsgo, nsgo);
  reader.TryReadTransformReference("testTransform", out testTransform, nsgo);
  reader.TryReadComponentReference("testComponent", out testComponent, nsgo);
 }
}
```

See Also

NeoSerializedGameObject MonoBehaviour

SceneSaveInfo MonoBehaviour

Runtime Objects

Overview

The NeoFPS save system is able to save and load objects that are instantiated at runtime, providing they adhere to the following rules:

1. The prefab must have a NeoSerializedGameObject on its root object.
2. The prefab must be registered with the save sytem
3. The object has to be instantiated by the the scene's SceneSaveInfo object or a parent NeoSerializedGameObject that is correctly set up.

Registering Prefabs

Prefabs can be registered in 2 places: on the SaveGameManager asset (*NeoFPS/Resources/FpsSettings_SaveGames.asset* by default), or on the scene's SceneSaveInfo object. Prefabs registered on the save game manager asset will be available at all times, however they (and all the assets such as textures and audio clips they reference) will be loaded into memory at all times. If the prefab is only supposed to be available in a few scenes then it should be registered in those scenes only.

The process for registering a prefab is the same for both situations. The SaveGameManager asset and the SceneSaveInfo monobehaviour both have the following section in the inspector:
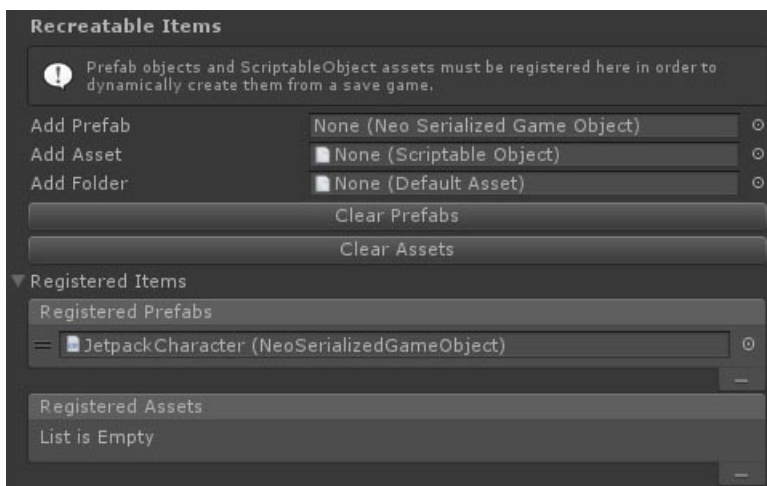
To register a prefab, you can drag it from the project view into the **Add Prefab** field. Alternatively you can drag a folder from the project view onto the **Add Folder** field and every prefab in that folder or its subfolders that has a NeoSerializedGameObject on its root will be registered. Each item will only be registered once, and duplicate items will be skipped.

To view the registered prefabs, you can expand the **Registered Items** foldout at the bottom of the inspector. The order of the items has no effect. You can remove an item by selecting the marker on the left (**=**) and hitting the **-** button at the bottom of the list. You can also click the **Clear Prefabs** button to remove all registered prefabs from the list.

Instantiating Prefabs

Instantiating a serializable prefab must be done via the following methods on a NeoSerializedGameObject or SceneSaveInfo monobehaviour:

```
public T InstantiatePrefab<T>(T prototype);
public T InstantiatePrefab<T>(T prototype, Vector3 position, Quaternion rotation);
public T InstantiatePrefab<T>(int prefabID, int serializationKey);
```

For the first two methods, if the prefab that's instantiated does not have a NeoSerializedGameObject at its root then it will be instantiated the standard Unity way. Otherwise it will be given unique serialization ID, registered with the parent object or scene and flagged as a runtime object.

The second method is usually used by the save game manager when rebuilding the scene hierarchy, but it can also be used in special situations where you don't want the save system to generate unique IDs.

Destroying Objects

You can destroy an object using the standard `Destroy()` method. If the object was instantiated at runtime then it will simply be unregistered by the parent object or scene. If the object was in the scene hierarchy on load, then it willl be flagged for destruction by the scene or parent object.

Changing Hierarchy

Each NeoSerializedGameObject is attached to its parent or the scene on load or instantiation. If you want to move an object from one NeoSerializedGameObject parent to another, then you can use the `NeoSerializedGameObject.SetParent()` method. Using the `transform.SetParent()` method as standard will cause unexpected results as the object will still be registered with its original parent. This means that on saving and loading using the save system, the object will be recreated as though it was still attached to the original.

Runtime instantiated objects must be a direct child of their parent NeoSerializedGameObject. The save game `InstantiatePrefab()` and `SetParent()` methods will enforce this, but the object hierarchy must not be changed separately.

Loading Process

When a scene is loaded from a save game, the system starts by loading the scene as standard. As soon as the scene is available, the NeoSerializedGameObject hierarchy is traversed and any changes to the hierarchy made. In order from the root NeoSerializedGameObject objects to the children, any objects that are flagged for destruction in the save are destroyed. Any runtime instantiated objects registered with the parents are instantiated via their prefab ID and assigned their original serialization ID. Once the hierarchy has been rebuilt, it is traversed again to load all the stored component data.

See Also

Troubleshooting

Overrides and Persistence

Overview

The NeoFPS save system has multiple uses. In the samples it is also set up to save character data for persistence when changing scenes, but it can also be extended to for other purposes.

Persistence

When progressing through a game you often have levels which continue on from each other. In this situation, you need certain data to carry over into the new scene, while other data can be discarded. For example, you would want the character to spawn at the position of the spawn point in the new scene, in a neutral state, but you would probably want them to retain their health and the contents of their inventory from the previous scene.

The actual persistence of the data is handled by the game mode. This tells the save game manger to save the player object and character object to a byte buffer using the persistence save mode. If you want to extend the persistence system or create your own, then it can be done with the following methods:

```
public static bool SaveGameManager.SaveGameObjectsToBuffer(NeoSerializedGameObject[] objects, SaveMode
saveMode);
public static bool SaveGameManager.LoadGameObjectsFromBuffer(NeoSerializedGameObject[] objects);
```

These both take an array of objects and will serialize them in order. As long as the objects in the save method match the objects in the load method (in terms of source prefab and order), then the requested data will be transferred between them.

Adding New Save Modes

The available save modes are set using the NeoFPS generated constants system. The first option is always the default save mode, as used by the standard quick, auto and manual saves. It is advised to keep the second option as persistence and add any new modes after this unless you're sure it isn't required for your project.

Using Overrides To Customise Saved Data

Each NeoSerializedGameObject can be set up to save different data for each game mode. The main filters and transform options are used for the default mode.
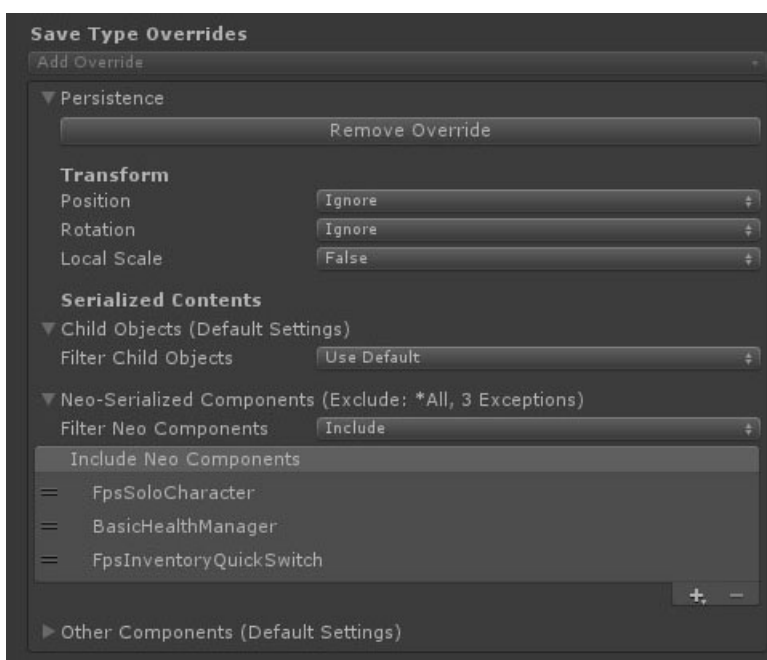
At the bottom of its inspector, you can see a section called **Save Type Overrides**. The dropdown button below this (**Add Override**) allows you to add a new override based on the available modes.

The override settings will appear below this and mirror the save settings and filters in the main inspector. Each property has a **Use Default** option so that not every property needs to be overriden. It can be limited to just transform properties, children or components as required.

See Also

Inventory Examples

Troubleshooting The Save System

Overview

Serializing and deserializing game state is a complex topic, and it can be tricky to work out why something isn't being saved the way you would expect it to. The NeoFPS save system tries to give the visual feedback in the inspector to help diagnose issues, along with a tool called the NeoSave File Inspector, which you can use to check data that has been saved to a file.

Troubleshooting NeoSerializedGameObjects

The following is an example of the serialization info shown in the inspector for a NeoSerializedGameObject:
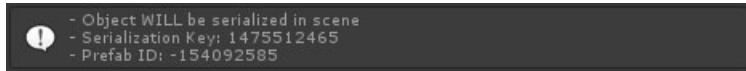
While this is an example of an object that will not be correctly saved by the save system:
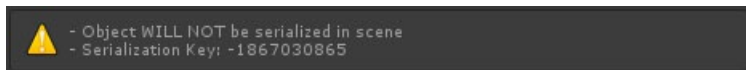
If an object says it will not be saved then that is generally down to one of four reasons:

1. The scene does not have a valid SceneSaveInfo.
2. The object's heirarchy does not have a NeoSerializedGameObject at its root.
3. The object or one of its parents is blocked from saving by NeoSerializedGameObject child object filters higher up the hierarchy.
4. The object was instantiated at runtime, but using default Unity instantiation and not via the NeoFPS save system.

NeoSave File Inspector
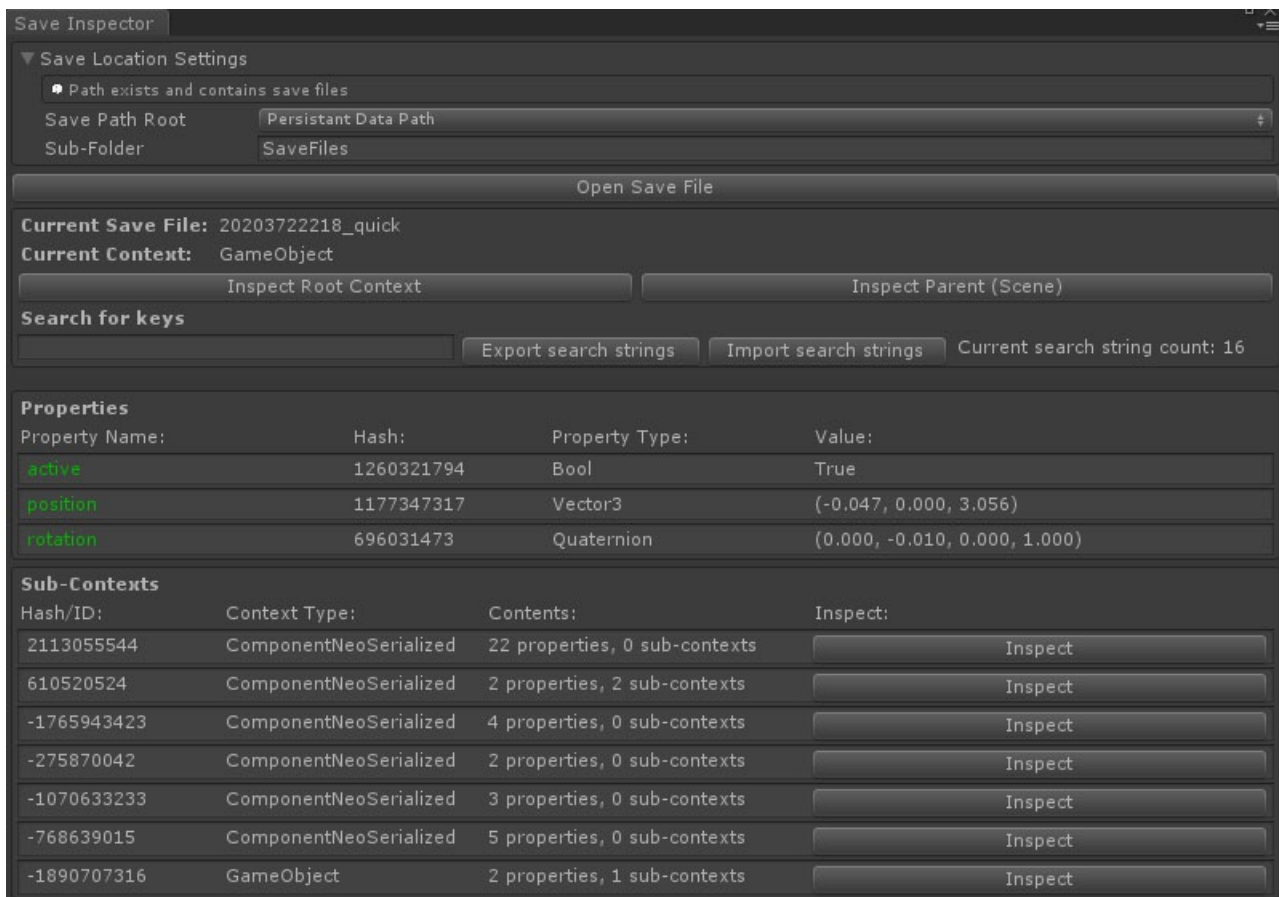
The NeoSave File Inspector can be used to open save files and inspect their contents. It can be found in the menus at *Tools/NeoFPS/NeoSave File Inspector*, or by clicking the **Save Game Inspector** button from within the SaveGameManager

settings.

Save Location Settings
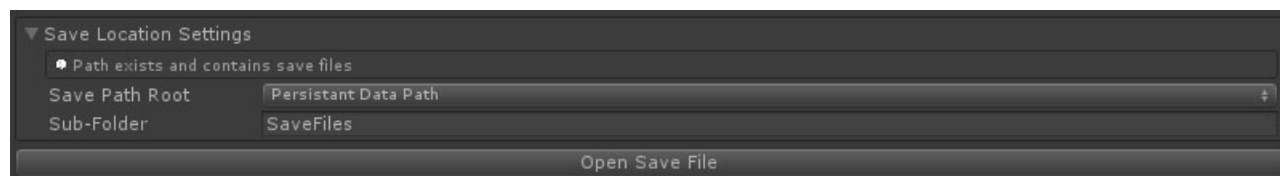
At the top of the editor is a foldout section for exploring to the project's save files. This matches the SaveGameManager settings and will display an info box that shows if the folder exists and contains save files.

Searching For Strings
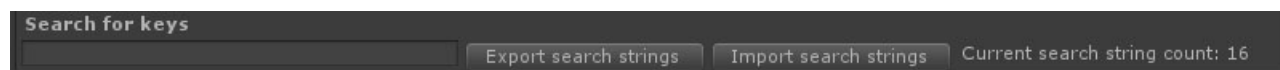
NeoFPS save files are designed to use a low footprint and fast format, and not to be human readable. Data is not organised using property names, but hashes instead. This can make it tricky to understand what values in the save file inspector correspond to what properties in a component or object. To aid in this, you can register property names with the inspector, and when a hash matches one of those property names, then the name will be displayed in green besides the hash value.

You can add new property names via the **Search For Keys** text field. This will add the name and its hash to the inspector. The inspector automatically registers a number of common property names that are used in the base NeoSerializedGameObject and save file metadata. You can also export and import the property names via the **Export Search Strings** and **Import Search Strings** buttons. This is useful if you are making frequent use of the save file inspector and have specific classes you want to focus on.

Navigating The File
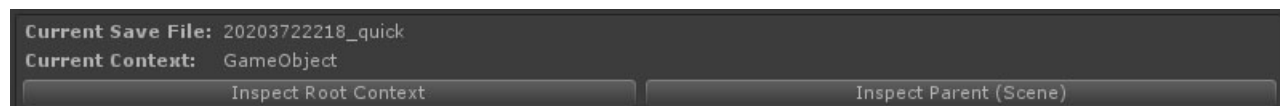
Near the top of the inspector is a section which displays information on your current location in the save file, and allows you to navigate up the save hierarchy.

NeoFPS save files group data into contexts, These context are things like *Scene*, *MetaData*, *GameObject* and *ComponentNeoSerialized* or *ComponentNeoFormatted*. These contexts are stored in a tree hierarchy, similar to the scene hierarchy. At the root of the save file is always the *Root* context. At any time while inspecting the save file contents, you can return to the save file root by using the **Inspect Root Context** button. You can also move back up the hierarchy towards the root one node at a time using the **Inspect Parent** button. The brackets next to this display the context type of the parent.

The **Current Context** readout tells you what context you are currently inspecting.

The **Properties** section displays the values that are currently saved for the inspected context and contains the following info:

- **Property Name** displays the name of the property if a property name has been registered with the inspector that matches the property's hash (see above).
- **Hash** is the hashed property name that was used to save the data.
- **Property Type** is the type of the value (it will also say if it is an array).

- **Value** is the saved value. In the case of arrays, you can expand this section to see the individual values (large arrays will show the first 50 values, followed by a "..." to signify there is more data in the file).

The **Sub-Contexts** section shows the child contexts that the current context contains and shows the following info:

- **Hash/ID** is the ID value that was used to identify the context when saved. This could be a generated ID, a hashed name, or a sequential number based on the context type. The only rule is that it is unique within the current parent context.
- **Context Type** is the type of the context.
- **Contents** shows the number of properties and child contexts for this context.
- The **Inspect** button will change the currently inspected context to this one. You can reach the previous context using the **Inspect Parent** button.

Loading vs Initialisation Order

Another area where it can be complicated to enforce correct behaviour is in the execution order of loading objects from save data vs initialising them using the standard Unity `Awake()` and `Start()` methods.

Unfortunately Unity only provides limited functionality to detect when a scene has been loaded, and events are only fired once Unity has initialised all objects with `Awake()`. It is possible to halt async scene loads once all objects have been created, but the scene state is not sufficent for searching for or instantiating objects.

The NeoFPS save system attempts to get around this by using Script Execution Order with the SceneSaveInfo behaviour to ensure that it is the first object to be awakened in the scene, which in turn triggers the loading from save data before the rest of the scene objects are awake. The reasoning behind this is that the save system can recreate the scene hierarchy and destroy any objects that require it before they are initialised. Runtime objects that are instantiated will call Awake() immediately, while scene objects will call Awake() once the hierarchy has been fully rebuilt.

If you find that your components are being loaded after they have already been initialised then check in the Script Execution Order settings that the top value is **NeoSaveGames.SceneSaveInfo**.

It is also a good idea to move any common initialisation logic out of `Awake()` and into a separate method that can be called from inside both `Awake()` and `ReadProperties()` with a check that it does not occur twice. For example:

```csharp
public class ExampleBehaviour : MonoBehaviour, INeoSerializableComponent
{
 private bool m_Initialised = false;

 void Awake()
 {
  // Do not initialise if already loaded by save system
  if (!m_Initialised)
  {
   // Perform non-shared initialisation logic here

   Initialise();
  }
 }

 void Initialise()
 {
  // Perform shared initialisation logic here

  m_Initialised = true;
 }

 public void ReadProperties(INeoDeserializer reader, NeoSerializedGameObject nsgo)
 {
  if (!m_Initialised)
  {
   // Read the relevant data here

   Initialise();
  }
  else
   Debug.LogError("Reading propertes after component was initialised. Check script execution order of
SceneSaveInfo.cs");
 }
}
```

See Also

[Inventory Examples][2]

AutoSaveBehaviour MonoBehaviour

Overview

The AutoSaveBehaviour behaviour is used to trigger auto-saves or checkpoint saves via the NeoFPS save system. It can be triggered via unity events, and used with the NeoFPS contact triggers and interactive objects.

Inspector
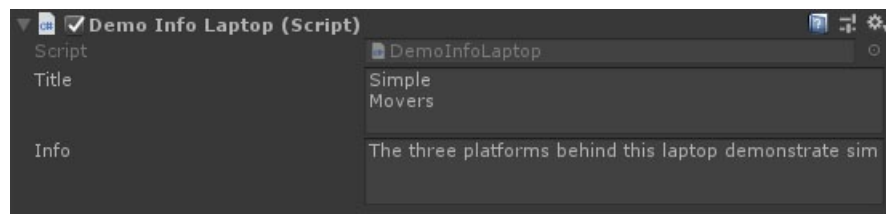
Properties

NAME	TYPE	DESCRIPTION
One Shot	Boolean	Can the autosave be triggered multiple times by this behaviour.
Cooldown	Float	The time in seconds before the autosave can be triggered again. This property will only appear if **One Shot** is set to false.
Retry Attempts	Integer	If the save fails (because another save or load event is in progress), the behaviour will try every second this manny times.

See Also

Save Games

Unity Events

Interaction With The World

NeoSerializedGameObject MonoBehaviour

Overview

The NeoSerializedGameObject behaviour is a key component of the NeoSave system. This component must be added to any objects that need to be saved and is used to specify which of its components and child objects are serialized.

Inspector
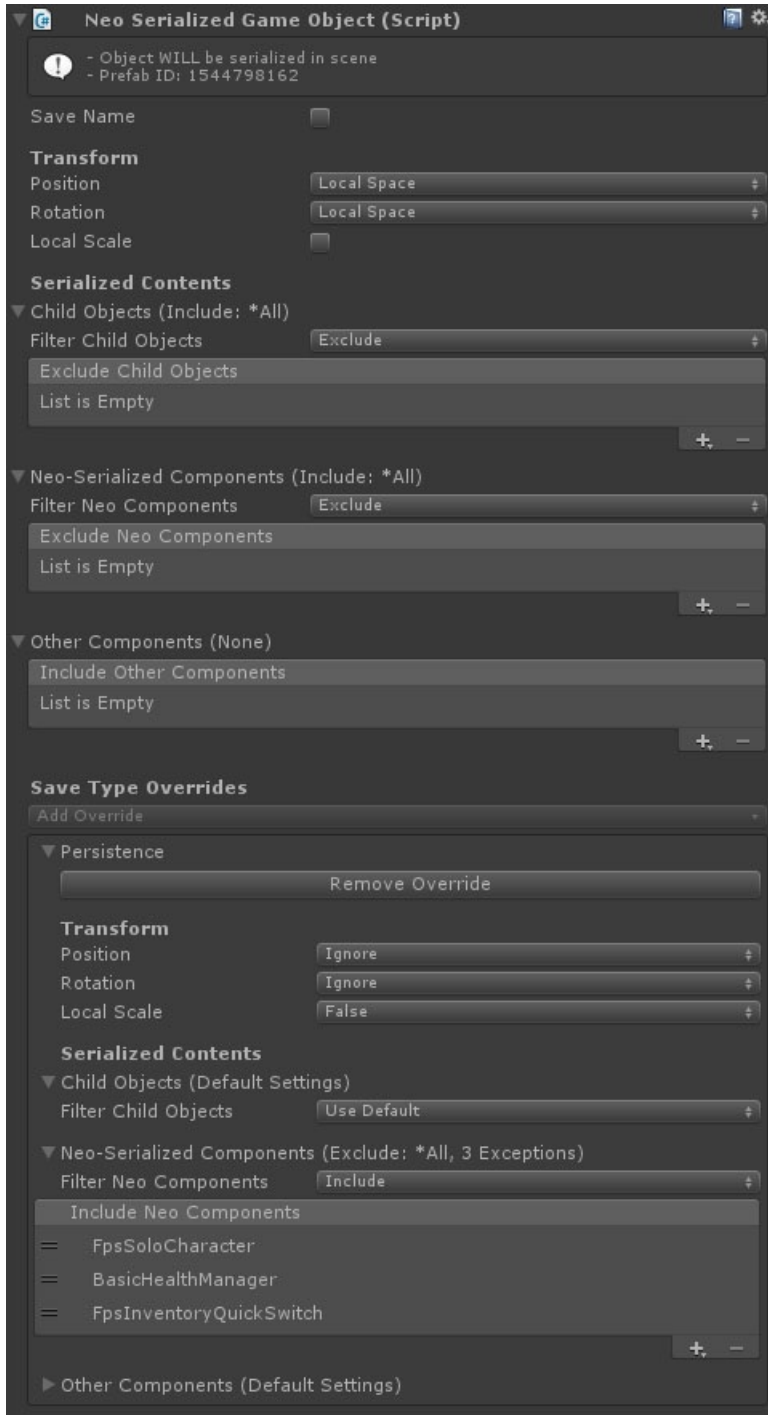
The information box at the top of the inspector gives details on the state of the object. It will list the object's serialization key, if the object was instantiated at run-time and whether the object is correctly set up to be saved. This requires that the root object has a valid and enabled NeoSerializedGameObject attached and none of its NeoSerializedGameObject parents are disabled or filtered out.

Properties

NAME	TYPE	DESCRIPTION
Save Name	Boolean	Save and load the object's name. This is not required in most cases.
Position	Dropdown	How to save and load the object's transform position. The avialable options are **Local Space**, **World Space** and **Ignore**.
Rotation	Dropdown	How to save and load the object's transform rotation. The avialable options are **Local Space**, **World Space** and **Ignore**.
Local Scale	Boolean	Should the object transform local scale be saved and loaded or ignored. Saving scale in world space is not an option .

Child Objects Foldout

NAME	TYPE	DESCRIPTION
Filter Child Objects	Dropdown	How to filter child objects for saving and loading. If set to **Exclude** then any objects added to the list below will **not** be saved. If set to **Include** then **only** objects in the list below will be saved.

To add objects to the list, click the **+** button and the available objects will be shown in a dropdown.

Neo-Serialized Components Foldout

NAME	TYPE	DESCRIPTION
Filter Neo Components	Dropdown	How to filter the neo-serialized components attached to this game object. If set to **Exclude** then any objects added to the list below will **not** be saved. If set to **Include** then **only** objects in the list below will be saved.

To add objects to the list, click the **+** button and the available components will be shown in a dropdown. A neo-serialized component is any behaviour that implements the `INeoSerializedComponent` interface. For more information see Serializing Data.

Other Components Foldout

Components that do not implement the `INeoSerializedComponent` interface can be serialized by adding them to this list. In order for a component (that is not a neo-serialized component) to appear in this list, it must be attached to this gameobject and have a valid formatter registered with the save system. For more information see Serializing Data.

Save Type Overrides

The NeoSave system allows you to define save modes using the generated constants system. By default, the samples use the save modes **Default** for standard game saves and **Persistence** for persisting certain data such as character health and inventory between scenes. You can add overrides here by clicing the **Add Override** button. This will show the available modes except for **Default** and any that have already been overriden. You can then override the filters and properties for that save mode as required or set them to use default. For example, the characters override the persistence save mode to only save the health manager and inventory components while not saving data like position, aim and motion graph states.
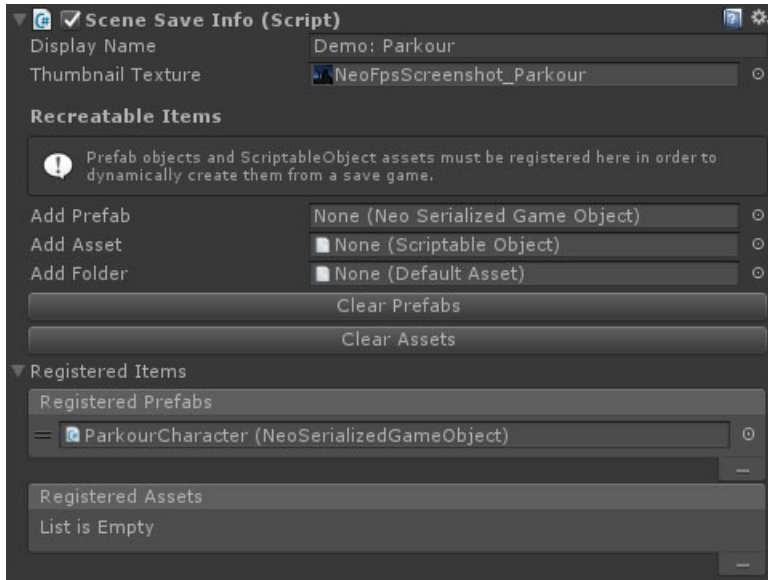
See Also

Serializing Data

SceneSaveInfo MonoBehaviour

Overview

The SceneSaveInfo behaviour stores information about the current scene for the save system. This includes the display name and thumbnail for the [save game browser][2] along with the objects registered for runtime instantiation and serialization.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Display Name	String	The title of the scene to display in the save browser.
Thumbnail Texture	Texture2D	An image to display in the save browser, depending on how the thumbnails are set up.

Recreatable Items

This section in the inspector is used to register objects with the save system so that they can be instantiated at runtime. This is only required for objects which need to be instantiated for this scene only. For more information see Runtime Objects.

See Also

Runtime Objects

SaveGameManager ScriptableObject

Overview

The SaveGameManager asset is the central access point for the save games system. It specifies the various save game settings as well as registering objects for instantiation at runtime.

Inspector
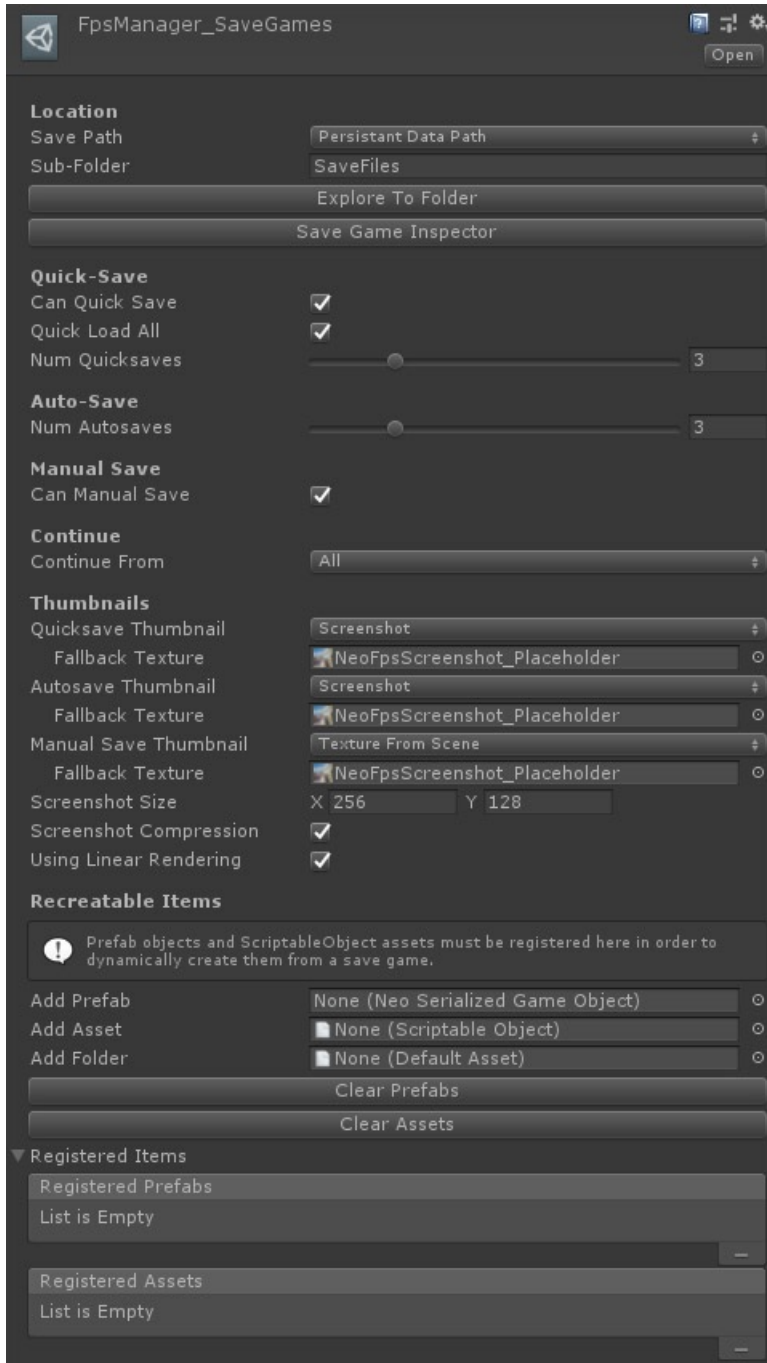
Properties

Location

NAME	TYPE	DESCRIPTION
Save Path	Dropdown	The available save file locations.
Sub-Folder	String	A sub-folder within the save path folder to place save files in.

You can use the **Explore To Folder** button to open the save games folder in the OS file explorer.

You can also use the **Save Game Inspector** button to show the save game inspector (also availabe via *Tools/NeoFPS/Save Game Inspector*). This allows you to open a save file and browse its contents. For more information see Troubleshooting

Quick-Save

NAME	TYPE	DESCRIPTION
Can Quick Save	Boolean	Sets whether the quick-save system is enabled in this project.
Quick Load All	Boolean	If true, quick loading will load the latest quick/auto/manual save. If not then it will only load the latest quick-save.
Num Quicksaves	Integer	The number of quicksaves to maintain. If the number exceeds this value, the oldest saves will be deleted.

Auto-Save

NAME	TYPE	DESCRIPTION
Num Autosaves	Integer	The number of autosaves to maintain. If the number exceeds this value, the oldest saves will be deleted.

Manual Save

NAME	TYPE	DESCRIPTION
Can Manual Save	Boolean	Sets whether the manual system is enabled in this project.

Continue

NAME	TYPE	DESCRIPTION
Continue From	Dropdown	What type of saves the game can be continued from. Available options are **None**, **All** and **Auto Save Only**.

Thumbnails

NAME	TYPE	DESCRIPTION
Quicksave Thumbnail	Dropdown	Where to get the thumbnail texture for quick-saves. Options are **None**, **Texture**, **TextureFromScene**, **Screenshot**.
(Fallback) Texture	Texture2D	If the thumbnail type is set to **Texture** then this is the texture that will be serialized to save files. If the thumbnail type is set to **Texture From Scene** or **Screenshot** then this texture will be used in the event those methods fail.
Autosave Thumbnail	Dropdown	Where to get the thumbnail texture for auto-saves. Options are **None**, **Texture**, **TextureFromScene**, **Screenshot**. The Texture / Fallback Texture property will appear below this if required.
Manual Save Thumbnail	Dropdown	Where to get the thumbnail texture for manual saves. Options are **None**, **Texture**, **TextureFromScene**, **Screenshot**. The Texture / Fallback Texture property will appear below this if required.
Screenshot Size	Vector2Int	If any of the thumbnail types are set to **Screenshot** then this property defines the resulting size of the saved screenshot texture.

NAME	TYPE	DESCRIPTION
Screenshot Compression	Boolean	This property defines whether the resulting texture will be compressed before serialization.
Using Linear Rendering	Boolean	Set this property to true if the project is set to use linear color space rendering. Screenshots need adapting to the correct color space or they can appear washed out.

Recreatable Items

This section in the inspector is used to register objects with the save system so that they can be instantiated at runtime. This is only required for objects which need to be instantiated for this scene only. For more information see [Runtime Objects][4].

See Also

Serializing Data

Troubleshooting

NeoSceneManager ScriptableObject

Overview

The NeoSceneManager asset specifies the default loading screen scene and exposes events to the inspector.

Inspector
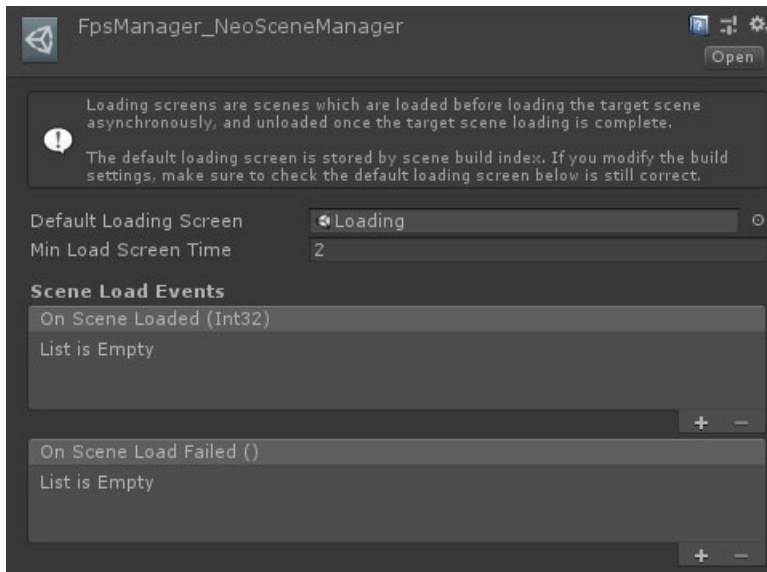
Properties

NAME	TYPE	DESCRIPTION
Default Loading Screen	Scene	The loading screen scene to use by default if none are specified. This is stored by index, so if changing the build settings, make sure to check this after.
Min Load Time	Float	The minimum amount of time the loading screen should stay visible before the scene is fully activated.
On Scene Loaded	Unity Event	An event called when the new scene is correctly loaded.
On Scene Load Failed	Unity Event	An event called when the new scene fails to load.

See Also

Unity Events

Samples

Overview

NeoFPS comes with a number of sample assets to demonstrate features and provide refererence for your own implementations.

Demo Facility
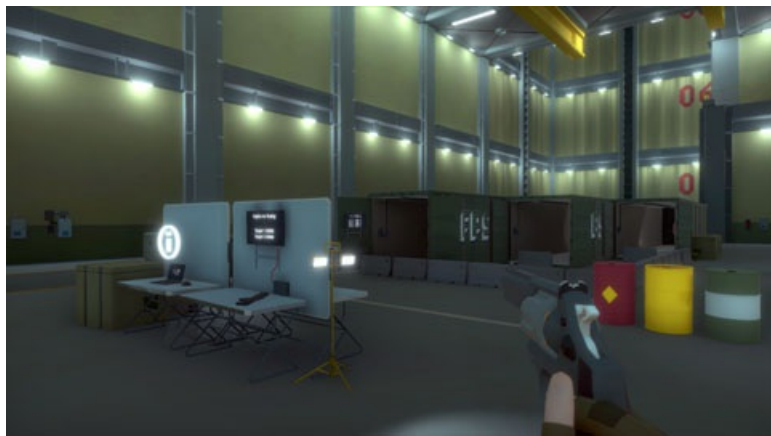
The demo facility is an early stages demo that is intended to grow with NeoFPS. As more features are added it will be expanded into a small demonstration of a story driven FPS' game mechanics.

In this version it contains a firing range and a number of environment elements.

Feature Demos

Firing Range

The firing range demo features 4 ranges to experiment with the weapons in NeoFPS:

- A quick reaction target to test reflexes
- An occupied building with popup targets inside and out
- Popup targets over a large area to test accuracy at different ranges
- Multi-range sliding targets to test scopes and aimers

Parkour

The parkour demo uses a custom motion graph that is set up to show off the parkour features of NeoFPS, and provides an obstacle course to try it out in. The scene has obstacles to demo:

- Wall Running
- Crouch Slides
- Jumping Between Walls
- Climbing
- Vertical Wall Run

Jetpacks & Guided Missiles

The jetpacks and guided missiles puts you in control of a character with tribes inspired movement including jetpacks, skiing and aim-hover. You also have a shoulder mounted missile launcher which can lock onto up to 6 targets at once using the ability button.

Inventories

The inventories demo comprises 3 scenes with a different inventory type in each. Use the scene switch buttons to switch to the other scenes and compare.

Swimming

The swimming demo uses a custom motion graph that is set up to show off one implementation of swimming in NeoFPS. It has a number of water zones to demonstrate features like moving surface heights, flow, and changing depths.

Moving Platforms

The moving platforms demo features a number of moving and rotating platforms, including complex rotations and waypoint based platforms.

NeoCharacterController
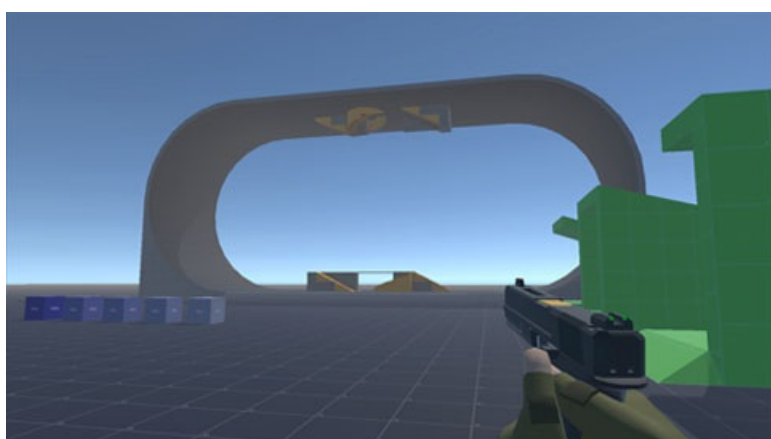
The NeoCharacterController demo shows some of the features of the NeoFPS character controller including pushing rigidbodies, variable gravity, and slope effects.

Persistence
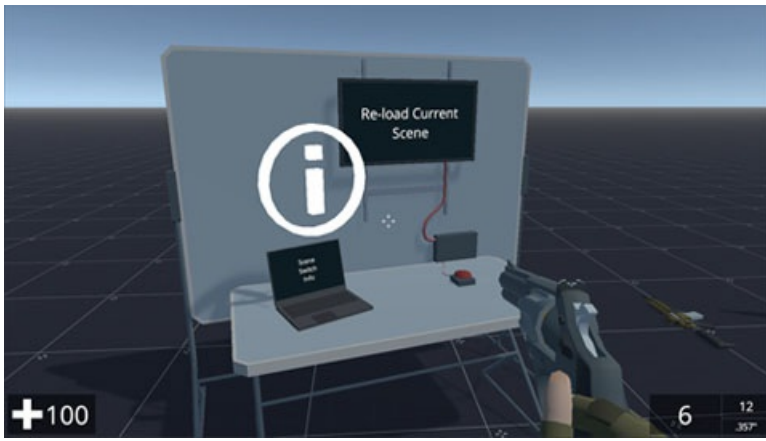
The persistence demo shows how the NeoFPS save system can be used to persist data like character health and inventory across scene changes.

Doors

The doors demo features the main door types in NeoFPS. It is worth exploring the scene hierarchy and tweaking values in the door behaviours to see what happens. The demo also contains a number of locked doors to demonstrate how the locks system works. This includes:

- Inventory keys
- Keypad locks
- Lockpicking
- Destructible locks

Ladders
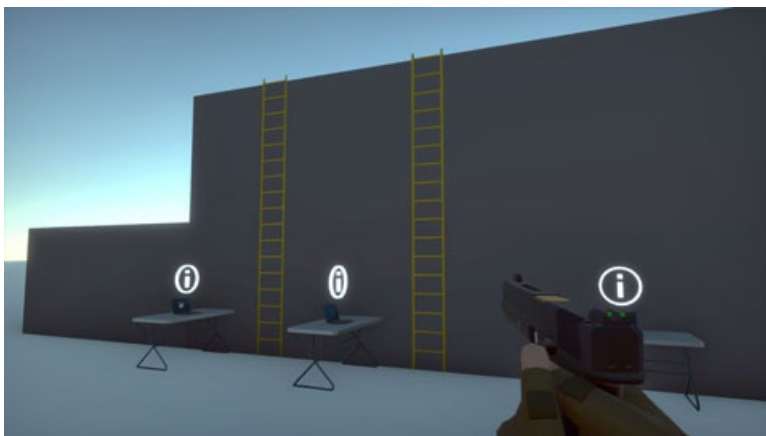
The ladders demo features the main ladder types for comparison and experimentation. For consistency across a project, a number of the properties that affect ladder climbing are accessible through the relevant motion graph states instead of the ladders themselves.

Minimal Character
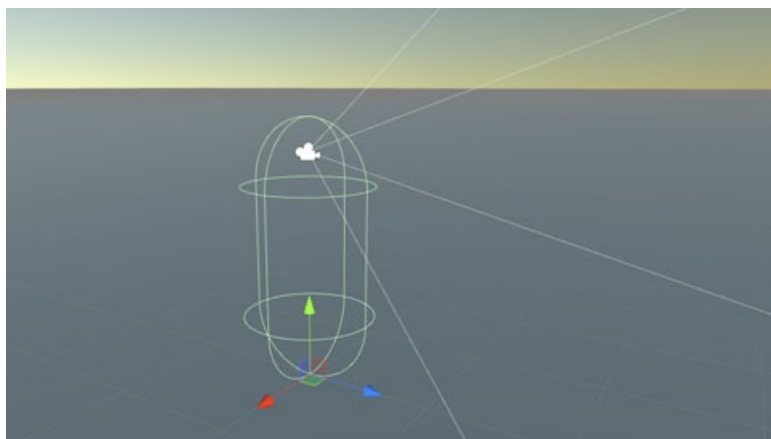
The minimal character demo uses a character that is stripped down to only the fundamental features of NeoFPS: movement and aim. It serves as an example of how you can pick and choose which features of NeoFPS to make use of and acts as a bare minimum foundation to build on using either NeoFPS' solutions, third party solutions, or your own custom solutions.

Since the minimal character demo uses default Unity input instead of NeoFPS' bindable input, it does not handle the sample menus and UI system. Because of this, the minimal character demo will not be visible in the "Select Level" section of the game main menu, and is intended only as an example in the editor.

See Also

AI

Overview

The NeoFPS Demo Facility sample includes a number of simple AI turrets and cameras. This is a very limited AI system that was mainly created to test out using the Modular Firearm system to create non-player weapons. The turret is set up with similar modules to the sample assault rifle, with some modifications such as a larger magazine size.

Both the cameras and turrets work by tracking side to side. If the player enters the AI's detection range and view area, then the AI will enter a suspicious state. After a brief period, it will upgrade to an alerted state and the turret will start shooting at the player. If the player leaves AI's view, then it will seek randomly from point to point until it times out and returns to an idle state, or detects the player again and re-enters the alerted state.

The turret and camera systems are not considered optimised for use in a complete game. For alternative AI solutions for NeoFPS you can check the integrations page on the NeoFPS website.

See Also

Modular Firearms

Sample UI

Overview
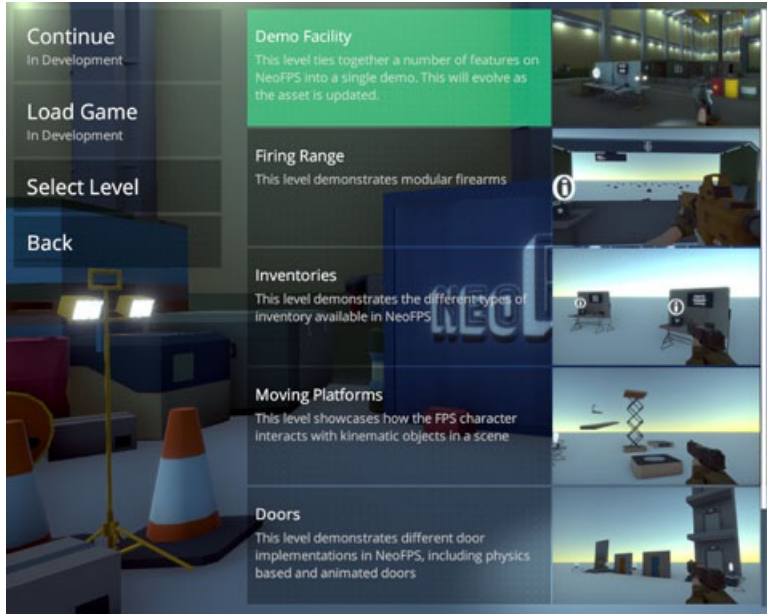
The sample UI is a tweaked implementation of the main Unity UI controls for more consistent menus that work with mouse navigation as well as keyboard or gamepad.

The original Unity UI source can be found here.

The main reasoning is to create a set of menu controls with multiple elements that can work as one. For example, a slider menu item in the options menu could comprise of a slider, a **+** button, a **-** button and a text input field. These all work together to drive and display the correct value. When using a mouse, dragging on the slider or clicking the buttons will change the value and that will be shown in the input field. Clicking the input field will capture keyboard input, which will set the new value and position the slider to match. When using a gamepad, highlighting the menu entry and hitting submit will give it focus. Until the user cancels or hits submit again the direction controls will update the value (eg. right is increase, left is decrease) instead of navigating to the next item.

The sample UI also has basic layout functionality using menu panels and navigation bars.

Lastly, the sample UI also implements popups that can be displayed from any script. This is used with the DemoInfoLaptop to provide info popups when used.

Using the sample UI is not a requirement for any of the features of NeoFPS. It is only intended for use in the NeoFPS samples and needs extra work to be considered a complete solution for use in other games. It can be useful, however, in demonstrating how to block character input whilst accessing UI elements

UI Styles

The sample UI uses UiStyle scriptable object assets to define a consistent style for the UI elements. The examples include the following assets:

- **UiStyleMenus** asset is used for menu buttons and items
- **UiStyleMenuGroups** asset is used for the foldouts and dropdowns of menu items
- **UiStyleNavigation** asset is used for the navigation menus (SinglePlayer, Options, etc)

See Also

Unity UI

ApplyRandomDamage MonoBehaviour

Overview

The ApplyRandomDamage behaviour applies a random amount of damage between the min and max provided.

Inspector
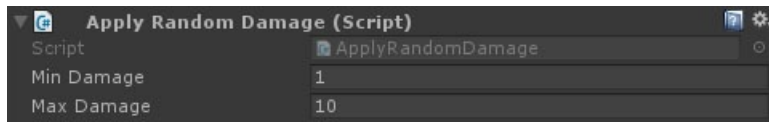
Properties

NAME	TYPE	DESCRIPTION
Min Damage	Float	The minimum damage to apply each time.
Max Damage	Float	The miaximum damage to apply each time.

See Also

[Health and Damage](#)

CameraSeeker MonoBehaviour

Overview

The CameraSeeker behaviour is a very simple fixed position AI which scans back and forth looking for a player, and enters an alert mode when it finds them.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Look Transform	Transform	The transform used to detect look angles (should be the last in the chain, eg. camera body).

NAME	TYPE	DESCRIPTION
Horizontal Servo Transform	Transform	The transform used to rotate on the horizontal axis.
Vertical Servo Transform	Transform	The transform used to rotate on the vertical axis.
Rotation Speed Idle	Float	The camera rotation speed when idle (degrees per second).
Pause Idle	Float	The length of the pause at each extreme of rotation while idle.
Idle Rotations	Vector2	The rotation points for the camera when idling.
Rotation Speed Hostile	Float	The camera rotation speed when hostile (degrees per second).
Pause Hostile	Float	The length of the pause at each extreme of rotation when hostile (degrees per second).
Suspicious Time	Float	The duration the camera will be suspicious before engaging.
Hunting Time	Float	The duration the camera will stay in hunting mode before going idle.
Detection Range	Float	The maximum range of the camera in meters.
Min Angles	Vector2	The minimum angles the camera can reach (negative y is down).
Max Angles	Vector2	The maximum angles the camera can reach (negative y is down).
Camera Light	Light	The halo glow for the camera.
Colour Idle	Color	The halo colour for the idle state.
Colour Suspicious	Color	The halo colour for the suspicious state.
Colour Engaged	Color	The halo colour for the engaged state.
On Idle	UnityEvent	An event invoked when the seeker enters the idle state.
On Suspicious	UnityEvent	An event invoked when the seeker enters the suspicious state.
On Engaged	UnityEvent	An event invoked when the seeker enters the engaged state.
On Hunting	UnityEvent	An event invoked when the seeker enters the hunting state.
On Killed	UnityEvent	An event invoked when the seeker is killed.
Starting Health	Float	The health of the seeker.

See Also

DemoFacilityTarget MonoBehaviour

Overview

The DemoFacilityTarget is a simple target that sends damage to a tracker for displaying to the player.

Inspector
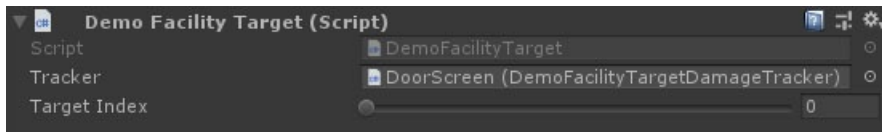
Properties

NAME	TYPE	DESCRIPTION
Tracker	DemoDamageTracker	The target damage tracker.
Target Index	Int	The target index in the tracker.

See Also

DemoFacilityTargetDamageTracker

Health and Damage

DemoFacilityTargetDamageTracker MonoBehaviour

Overview

The DemoFacilityTargetDamageTracker behaviour tracks damage to a number of DemoFacilityTarget objects and prints it to a [Unity UI] readout.

Inspector
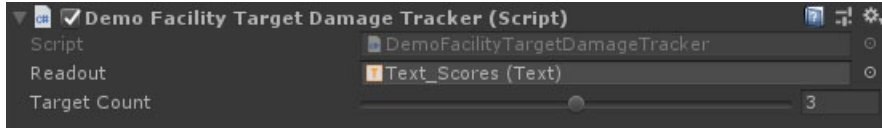
Properties

NAME	TYPE	DESCRIPTION
Readout	Text	The text readout for target damage.
Target Count	Int	The number of targets to track.

See Also

DemoFacilityTarget

Unity UI

DemoInfoLaptop MonoBehaviour

Overview

The DemoInfoLaptop behaviour is an info point that can be interacted with and displays a text popup using the demo UI.

Inspector
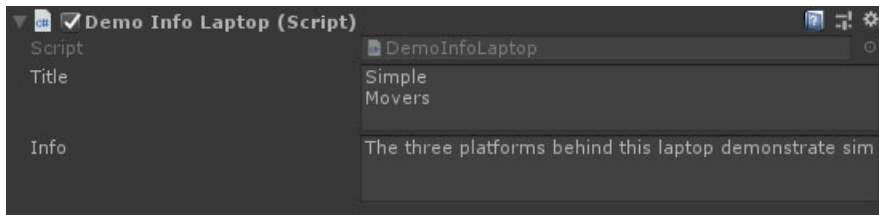
Properties

NAME	TYPE	DESCRIPTION
Title	String	The title to display on the laptop screen.
Info	String	The info to display in the popup when interated with.

See Also

Sample UI

Interactive Objects

Unity UI

DoorsDemoElevatorReadout MonoBehaviour

Overview

The DoorsDemoElevatorReadout behaviour subscribes to the events on an ElevatorController, and prints it to a Unity UI.

Inspector

Properties

The DoorsDemoElevatorReadout behaviour has no properties exposed in the inspector.

See Also

ElevatorController

Unity UI

FiringRangeMovingTarget MonoBehaviour

Overview

The FiringRangeMovingTarget behaviour is a variant of the FiringRangeTarget that also moves.

Inspector

Properties

The FiringRangeMovingTarget inherits from the FiringRangeTarget. Check the reference for information on its properties.

NAME	TYPE	DESCRIPTION
Move Offset	Vector3	The offset to move to.
Move Duration	Float	The time taken to reach the offset.

See Also

Health and Damage

FiringRangeTarget

FiringRangeSequencer

FiringRangeReadout MonoBehaviour

Overview

The FiringRangeReadout behaviour recieves hit and miss events from the targets in the firing range demo and prints them to a Unity UI.

Inspector
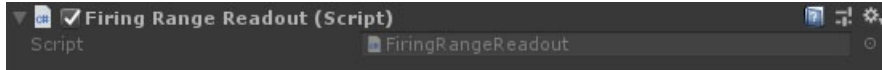
Properties

The FiringRangeReadout behaviour has no properties exposed in the inspector.

See Also

FiringRangeSequencer

Unity UI

FiringRangeSequencer MonoBehaviour

Overview

The FiringRangeSequencer behaviour manages all the targets in a demo firing range. It works in waves, controlling the number of targets that pop up at once along with the spacing.

Inspector
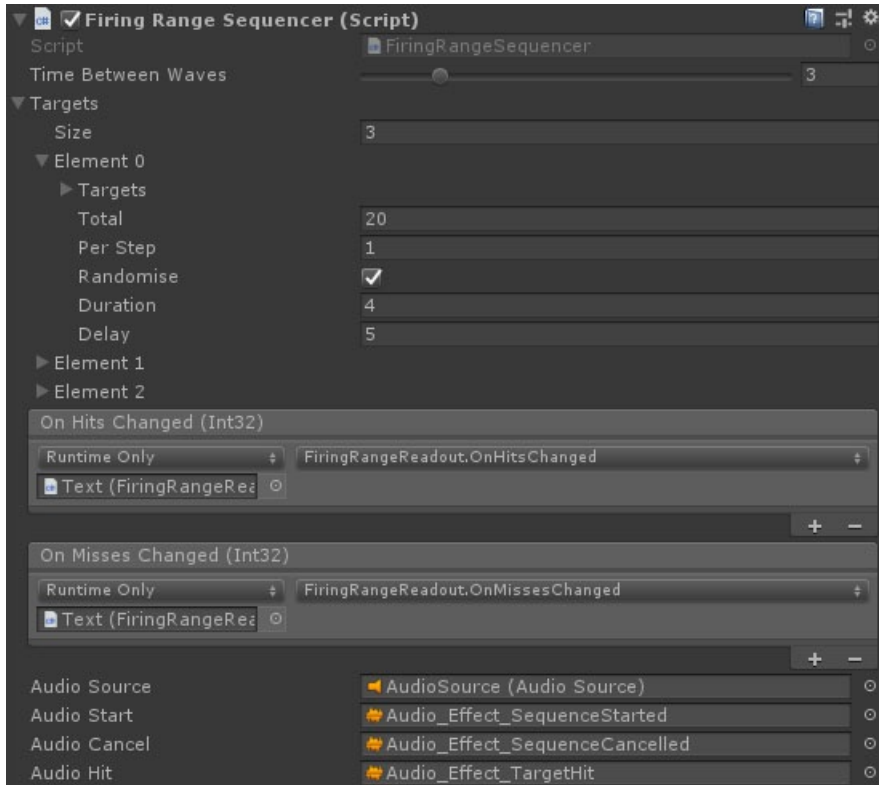
Properties

NAME	TYPE	DESCRIPTION
Time Between Waves	Float	The pause in between each wave.
Targets	Target Group Array	The targets for each wave.
On Hits Changed	UnityEvent	An event that is invoked when a target is hit.
On Misses Changed	UnityEvent	An event that is invoked when a target is missed.
Audio Transform	Transform	The transform to attach the audio rotator to.
Audio Source	AudioSource	The audio source for playing one shot firing range audio clips.
Audio Start	AudioClip	The audio clip to play when the sequence starts.
Audio Cancel	AudioClip	The audio clip to play when the sequence is cancelled.
Audio Hit	AudioClip	The audio clip to play when a target is hit.

Target Group

NAME	TYPE	DESCRIPTION
Targets	FiringRangeTarget Array	The targets for this wave.
Total	Int	The total number of targets to pop up this wave.
Per Step	Int	The number of targets to pop up for each step of the wave.
Randomise	Boolean	Should the targets be chosen at random or in sequence.
Duration	Float	The duration a target should stay up.
Delay	Float	The delay between steps.

See Also

FiringRangeTarget

FiringRangeMovingTarget

FiringRangeTarget MonoBehaviour

Overview

The FiringRangeTarget behaviour is a pop-up target that sends hit or miss events to the sequencer.

Inspector
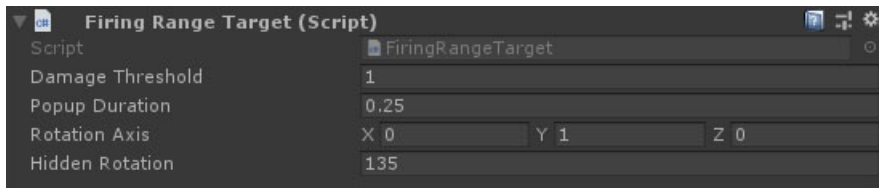
Properties

NAME	TYPE	DESCRIPTION
Damage Threshold	Float	The damage threshold for the target to drop and register as a hit.
Popup Duration	Float	The duration the target will be visible. If the target is not hit in this time it registers as a miss.
Rotation Axis	Vector3	The axis to rotate the target around when it pops up.
Hidden Rotation	Float	The rotation of the target around the specified axis when it is fully hidden.

See Also

Health and Damage

FiringRangeSequencer

InfoPopupTrigger MonoBehaviour

Overview

The InfoPopupTrigger behaviour is an info point that can be interacted with and displays a text popup using the demo UI.

Inspector
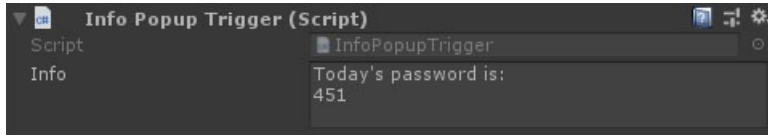
Properties

NAME	TYPE	DESCRIPTION
Title	String	The title to display on the laptop screen.
Info	String	The info to display in the popup when interated with.

See Also

Sample UI

Interactive Objects

Unity UI

KeypadPopup MonoBehaviour

Overview

The KeypadPopup behaviour is an info point that can be interacted with and displays a text popup using the demo UI.

Inspector
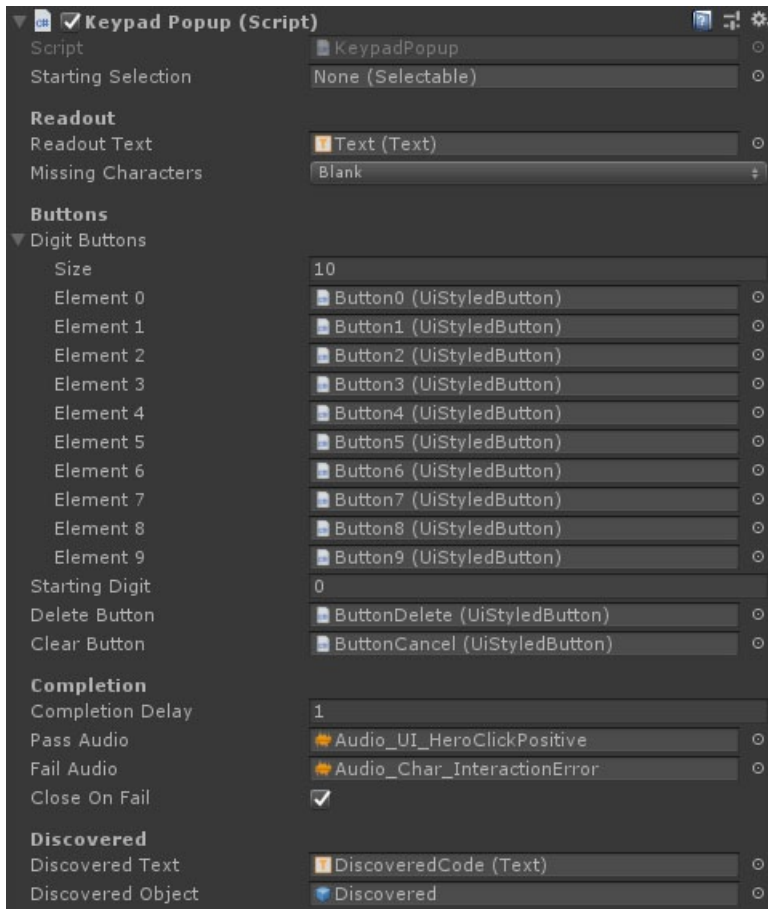
Properties

NAME	TYPE	DESCRIPTION
Title	String	The title to display on the laptop screen.
Info	String	The info to display in the popup when interated with.

See Also

Sample UI

Interactive Objects

Unity UI

LoadingScreen MonoBehaviour

Overview

The LoadingScreen behaviour is a very simple example implementation of a loading screen with gameplay hints and instructions.

The first time the loading screen is shown, it will display a warning about exiting while performing a save. After this, a random hint will be displayed.

Inspector
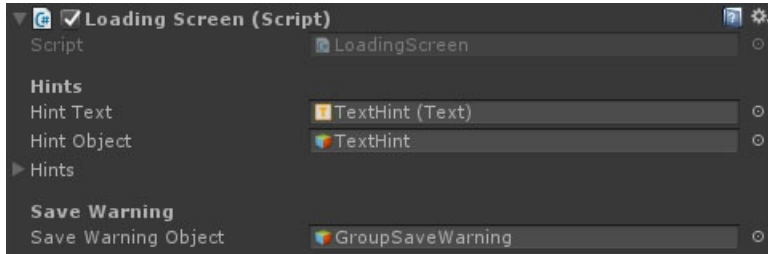
Properties

NAME	TYPE	DESCRIPTION
Hint Text	Text	The text output for the gameplay hints.
Hint Object	GameObject	The object that contains the hints UI. The first time the screen is shown, this object will be hidden.
Hints	String Array	An array of hints to display, chosen at random.
Save Warning Object	GameObject	The object that contains a warning about the "saving" icon. This will only be shown the first time.

See Also

Unity UI

MinimalDemoCharacter MonoBehaviour

Overview

The MinimalDemoCharacter behaviour ties together the NeoCharacterController, MotionController and MouseAndGamepadAimController to create a simplified playable character with all the movement and aim features of the main NeoFPS demo character.

The minimal character uses Unity's input system to control the character instead of NeoFPS' bindable input system, while all of NeoFPS' features outside movement and aiming are removed.

Inspector
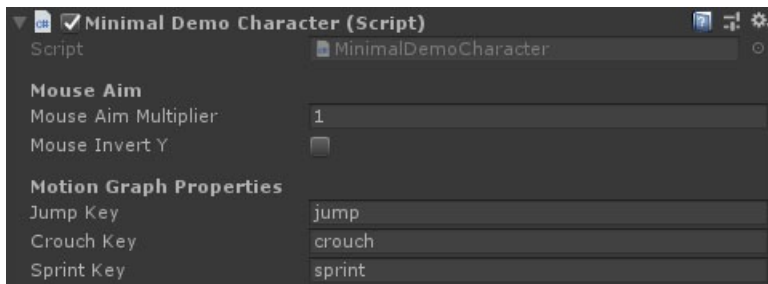
Properties

NAME	TYPE	DESCRIPTION
Mouse Aim Multiplier	Float	The mouse aim sensitivity.
Mouse Invert Y	Boolean	Is the mouse vertical aim rotation flipped.
Jump Key	String	The key to the jump trigger property in the character motion graph.
Crouch Key	String	The key to the crouch switch property in the character motion graph.
Sprint Key	String	The key to the sprint switch property in the character motion graph.

See Also

NeoCharacterController

MotionController

MouseAndGamepadAimController

Samples

OutOfBoundsRespawn MonoBehaviour

Overview

The OutOfBoundsRespawn behaviour simply watches the player character, and if their height is lower than a set value, it respawns them.

Inspector
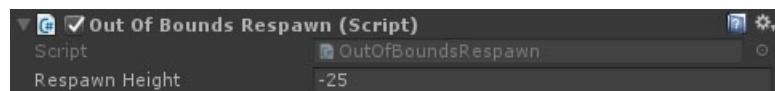
Properties

NAME	TYPE	DESCRIPTION
Respawn Height	Float	If the player character's position is lower than this height then it will trigger a respawn.

TurretSeeker MonoBehaviour

Overview

The TurretSeeker behaviour is a very simple fixed position AI which scans back and forth looking for a player, similarly to the CameraSeeker. Once it spots a player it will enter a suspicious mode, and then start shooting after a brief pause.

The TurretSeeker uses the Modular Firearm system to model gun behaviour using the same modules as the player's first person weapons.

Inspector

Properties

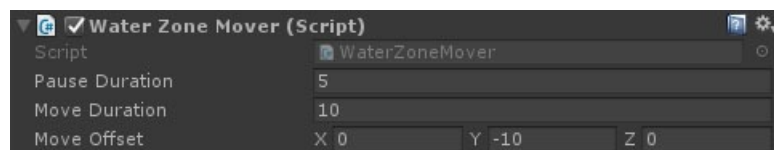| NAME	TYPE	DESCRIPTION
Look Transform	Transform	The transform used to detect look angles (should be the last in the chain, eg. camera body).
Horizontal Servo Transform	Transform	The transform used to rotate on the horizontal axis.
Vertical Servo Transform	Transform	The transform used to rotate on the vertical axis.
Rotation Speed Idle	Float	The camera rotation speed when idle (degrees per second).
Pause Idle	Float	The length of the pause at each extreme of rotation while idle.
Idle Rotations	Vector2	The rotation points for the camera when idling.
Rotation Speed Hostile	Float	The camera rotation speed when hostile (degrees per second).
Pause Hostile	Float	The length of the pause at each extreme of rotation when hostile (degrees per second).
Suspicious Time	Float	The duration the camera will be suspicious before engaging.
Hunting Time	Float	The duration the camera will stay in hunting mode before going idle.
Detection Range	Float	The maximum range of the camera in meters.
Min Angles	Vector2	The minimum angles the camera can reach (negative y is down).
Max Angles	Vector2	The maximum angles the camera can reach (negative y is down).
Camera Light	Light	The halo glow for the camera.
Colour Idle	Color	The halo colour for the idle state.
Colour Suspicious	Color	The halo colour for the suspicious state.
Colour Engaged	Color	The halo colour for the engaged state.
On Idle	UnityEvent	An event invoked when the seeker enters the idle state.
On Suspicious	UnityEvent	An event invoked when the seeker enters the suspicious state.
On Engaged	UnityEvent	An event invoked when the seeker enters the engaged state.
On Hunting	UnityEvent	An event invoked when the seeker enters the hunting state.
On Killed	UnityEvent	An event invoked when the seeker is killed.
Starting Health	Float	The health of the turret.

See Also

Modular Firearms

CameraSeeker

WaterZoneMover MonoBehaviour

Overview

The WaterZoneMover behaviour is used in the swimming demo to raise and lower one of the water zones.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Pause Duration	Float	The pause between moving up and down.
Move Duration	Float	The time it takes to go from the original position to the offset position or back again.
Move Offset	Vector3	The offset from the original position to move in local space.

Utilities

Animation Behaviours

Animation graph behaviours are attached to states in the unity animator controller graphs. NeoFPS uses 2 behaviours that allow animations to sync the active state of objects to specific clips, and to swap animated objects with dynamic physics objects. These are used in the modular firearms during reload animations but can have a number of uses besides.

For more information, see the AnimStateObjectActivator, AnimStateObjectSwapper, and AnimStateObjectSwapperTarget references.

Pooling

Pooling is an important optimisation in any game. It means pre-instantiating a pool of objects before they are needed that can be spawned and recycled at any time. Instantiation and the garbage collection resulting from object destruction are relatively expensive operations and it is best practice not to instantiate or destroy objects during gameplay. The NeoFPS pooling utilities can specify pool sizes for individual prefabs that will be instantiated at startup. These can then be retrieved and returned as needed.

The PoolManager specifies the pooled objects to be made available across all scenes, along with a number of default settings, while the ScenePoolHandler is responsible for preallocating and accessing pools within the active scene. Retrieve a PooledObject with the following function:

```
public T GetPooledObject<T> (PooledObject prototype) where T : MonoBehaviour
```

The PooledObject can be returned to the pool with the following function:

```
void PooledObject.ReturnToPool ()
```

If a prototype PooledObject is not managed by the PoolManager already then a new pool of the default size will be instantiated. This is an expensive operation so it is best to try and anticipate the required objects in advance.

Object Lifecycle

NeoFPS uses a number of simple utilities that manage the active state and lifecycle of an object. See the TimedDisabler, ObjectLifecycleManager and ObjectLifecycleMgrDictionary for more information. You can also use a TemporaryPooledObject which has the lifecycle built in for objects that should only be visible for a brief period.

Teleporters

There are 2 teleporter setups provided in NeoFPS:

- The Teleporter behaviour is used in the NeoFPS demos to quickly get around the scenes. It is used for physical teleporter objects that you step into and interact with to teleport to another location. The target location needs to be another teleporter, and that teleporter will be disabled for a bried period on teleporting into it.
- The TeleportZone1Way behaviour is used for simpler navigation around scenes. On stepping into the teleport zone a timer will start and an *onEntered* UnityEvent will fire that you can use to trigger particle and audio effects. Once the timer completes, the character is teleported to the target transform and an *onTeleported* event is fired. If the character leaves the zone before the timer completes, then the timer resets and an *onCancelled* event is fired.

Slow-Motion System

NeoFPS implements a simple slow-motion system that allows you to create slow-mo effects in response to various triggers. The SlowMoSystem behaviour can be added to the player character to create an adrenaline or focus style system as used in games like F.E.A.R. This system has a slow-mo charge which is drained while in slow motion and the recharges when back in normal time. You can also zero the drain or recharge rates for more control over when the effect starts or stops.

To trigger the slow-mo effect, NeoFPS implements a number of sample mechanisms. These include:

- The SetTimeScale Motion Graph Behaviour triggers slow-mo when entering a specific motion state.
- The SlowMoZone behaviour is used to define a trigger zone. When the player character enters, time is slowed. When they exit, time speed is restored.
- The TimedSlowMoZone behaviour is similar, but the slow-mo effect only lasts for a short period (based on the drain rate).

The slow-mo system works by modifying the following script property: `NeoFpsTimeScale.timeScale`. This is similar to Unity's `Time.timeScale`, but fires events that allow other systems to monitor the time scale. This allows things like [audio sources][unity-audiosource] to react by modifying their pitch (and therefore speed) via the AudioTimeScalePitchBend behaviour. If you want to create extremely simple slow-mo effects (such as when liberating an outpost in far cry), you can modify this property directly.

See Also

Unity AnimatorController

Unity State Machine Behaviours

AnimStateObjectActivator StateMachineBehaviour

Overview

This is used to control when gameobjects are active from within an animation graph. For example, controlling a shotgun shell's visibility during a reload animation.

Inspector
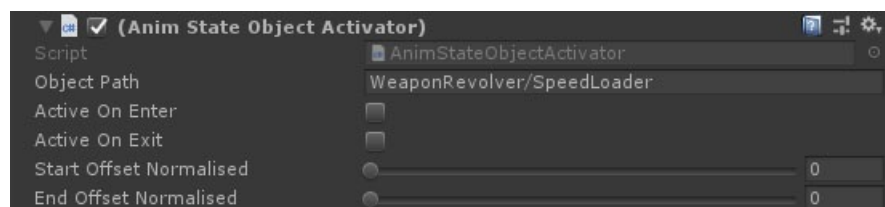
Properties

NAME	TYPE	DESCRIPTION
Object Path	String	The path to the relevant object in the object hierarchy the animator is attached to.
Active On Enter	Boolean	Should the object be active when the animation state is entered.
Active On Exit	Boolean	Should the object be active when the animation state is exited.
Start Offset Normalised	Float	The normalised offset (0 to 1) from entry of the state for the object state to be changed (0 is the start, 1 is the end).
End Offset Normalised	Float	The normalised offset (0 to 1) from the end of the state for the object state to be changed (0 is the end, 1 is the start).

See Also

Unity Animator

Unity AnimatorController

AnimStateObjectSwapper StateMachineBehaviour

Overview

The AnimStateObjectSwapper is a StateMachineBehaviour. It is attached to animator controller graph states and will trigger an AnimatedObjectSwapper to swap an object out.

Inspector
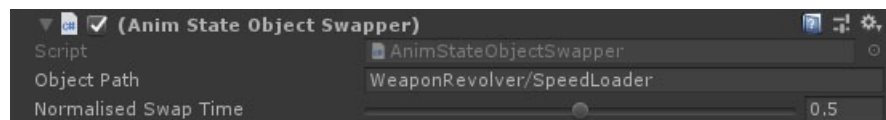
Properties

NAME	TYPE	DESCRIPTION
Object Path	String	The path to the object containing the AnimatedObjectSwapper relative to the object hierarchy the animator is attached to.
Normalised Swap Time	Float	The normalised offset (0 to 1) from entry of the state for the object swap to occur. (0 is the start, 1 is the end).

See Also

AnimatedObjectSwapper

Unity Animator

Unity AnimatorController

Unity StateMachineBehaviour

AbilityBasedSlowMoSystem MonoBehaviour

Overview

The AbilityBasedSlowMoSystem behaviour swaps out an object with a pooled physics object. It is paired with an AnimStateObjectSwapper state machine behaviour that triggers the swap.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Target Transform	Transform	The object to swap.
Pooled Object	PooledObject	The pooled object to swap the target object with.
Spawn Velocity	Vector3	The velocity to spawn the physics object at relative to the target object rotation.
Spawn Angular Velocity	Vector3	The angular velocity of the spawned physics object.

See Also

AnimStateObjectSwapper

PooledObject

Unity Animator

Unity AnimatorController

Unity Rigidbody

AnimStateObjectSwapperTarget MonoBehaviour

Overview

The AnimStateObjectSwapperTarget behaviour swaps out an object with a pooled physics object. It is paired with an AnimStateObjectSwapper state machine behaviour that triggers the swap.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Target Transform	Transform	The object to swap.
Pooled Object	PooledObject	The pooled object to swap the target object with.
Spawn Velocity	Vector3	The velocity to spawn the physics object at relative to the target object rotation.
Spawn Angular Velocity	Vector3	The angular velocity of the spawned physics object.

See Also

AnimStateObjectSwapper

PooledObject

Unity Animator

Unity AnimatorController

Unity Rigidbody

BillboardOrientation MonoBehaviour

Overview

The BillboardOrientation behaviour is used to orient an object towards the currently active camera.

Inspector
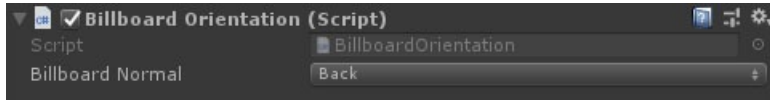
Properties

NAME	TYPE	DESCRIPTION
Billboard Normal	Dropdown	The billboard surface normal direction (this will be turned towards the camera).

BulletTrailCleaner MonoBehaviour

Overview

The BulletTrailCleaner behaviour is attached to pooled projectiles with a trail renderer and clears the trail when the projectile is disabled so that it can be recycled without old trail data bleeding into new.

Inspector
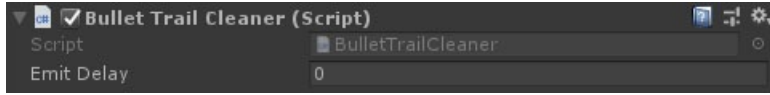
Properties

NAME	TYPE	DESCRIPTION
Emit Delay	Float	The delay before enabling the trail render.

See Also

Interactive Objects

Unity Events

CheckpointTrigger MonoBehaviour

Overview

The CheckpointTrigger behaviour is used to signify progression checkpoints in your levels. Crossing a checkpoint can enable the connected spawn points and disable the old spawns, as well as auto-saving the game.

Inspector
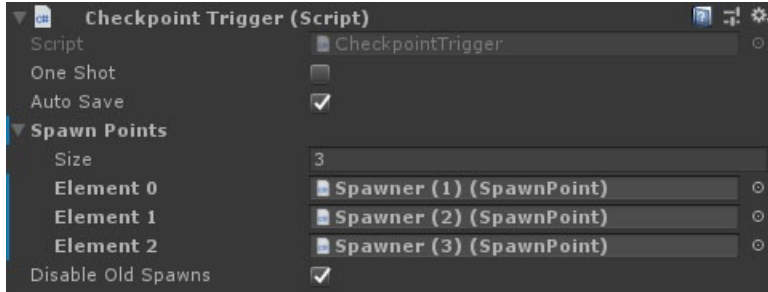
Properties

NAME	TYPE	DESCRIPTION
One Shot	Boolean	Should the checkpoint trigger fire multiple times (eg allow back-tracking).
Auto Save	Boolean	Save the game progress using the auto save feature.
Spawn Points	SpawnPoint Array	A list of spawn points to enable at this checkpoint.
Disable Old Spawns	Boolean	Should previous spawn points be disabled (guaranteeing that the player spawns here).

See Also

Save Games

DemoButtonPresser MonoBehaviour

Overview

The DemoButtonPresser behaviour is used to procedurally animate a simple button. It has a public `Press()` function that can be attached to unity events, for example on the InteractiveObject **OnUsed** event.

Inspector
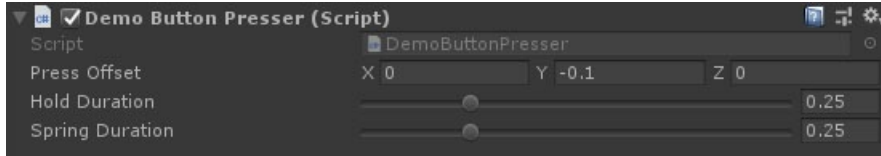
Properties

NAME	TYPE	DESCRIPTION
Press Offset	Vector3	The button position offset when pressed.
Hold Duration	Float	The duration to hold the button down.
Spring Duration	Float	The duration to spring back to the original position.

See Also

Interactive Objects

Unity Events

NeoFpsSceneSwitcher MonoBehaviour

Overview

The NeoFpsSceneSwitcher behaviour is used to trigger loading a new scene.

The NeoFPS scene management system loads a scene with the following steps:

1. Load the loading screen scene which displays a loading screen over the game view.
2. If the switcher is set to persist data, then ask the current game mode to save the relevant game data to a memory buffer.
3. Unload the current scene.
4. Load the new scene asynchronously in the background.
5. If data was persisted, ask the new scene's game mode to load it.
6. Unload the loading screen scene.

Inspector
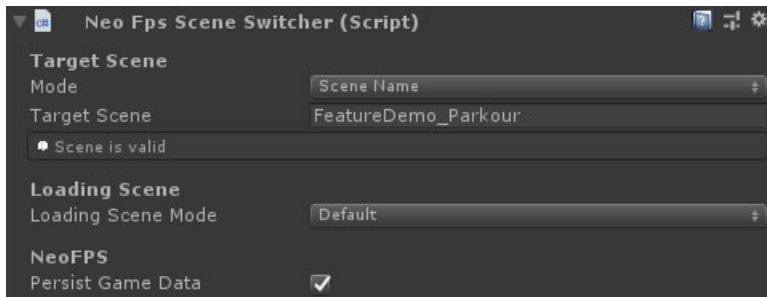
Properties

NAME	TYPE	DESCRIPTION
Mode	Dropdown	How the scene to load is specified. The options are **Scene Name**, **Scene Index**, **Next Scene Index**. If the last option is used then the next scene listed in the build options will be loaded.
Target Scene	String/Integer	The name or index of the scene to load, matching the entry in the project build options.
Loading Scene Mode	Dropdown	How to choose the loading screen scene to load. The options are **Default** (specified in the NeoSceneManager options, **Scene Name** and **Scene Index**.
Loading Scene	String/Integer	The name or index of the loading screen scene to load, matching the entry in the project build options.
Persist Game Data	Boolean	Should game data (eg, character health and inventory) be persisted to the new scene?

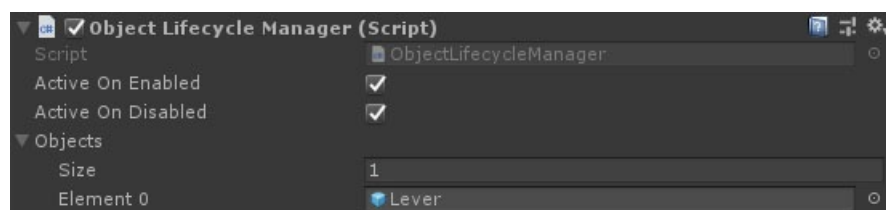## See Also

Runtime Objects

NeoSceneManager

Game Modes

ObjectLifecycleManager MonoBehaviour

Overview

The ObjectLifecycleManager is used to manage the active state of attached objects.

Inspector

Properties

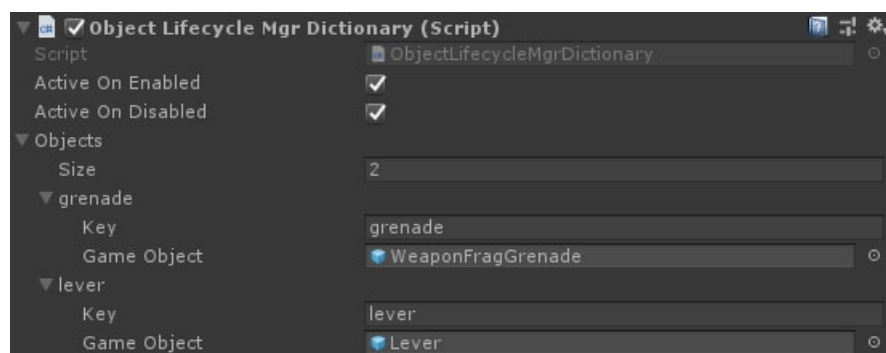| NAME	TYPE	DESCRIPTION
Active On Enabled	Boolean	Should all the attached objects be enabled when this is.
Active On Disabled	Boolean	Should all the attached objects be disabled when this is.
Objects	GameObject Array	The objects being managed.

See Also

ObjectLifecycleMgrDictionary MonoBehaviour

Overview

The ObjectLifecycleMgrDictionary behaviour allows scripts to manage the active state of attached objects by name.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Active On Enabled	Boolean	Should all the attached objects be enabled when this is.
Active On Disabled	Boolean	Should all the attached objects be disabled when this is.
Objects	Entry Array	The objects being managed.

Entry

NAME	TYPE	DESCRIPTION
Key	String	A string key for the object.
Game Object	GameObject	The object to manage.

See Also

ObjectMover MonoBehaviour

Overview

The ObjectMover behaviour is attached to an object to interpolate from one position to another.

Inspector
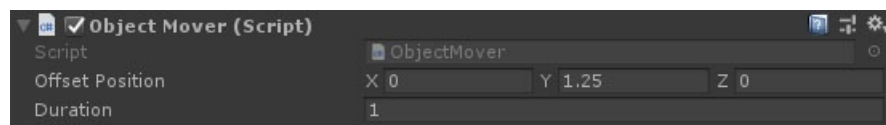
Properties

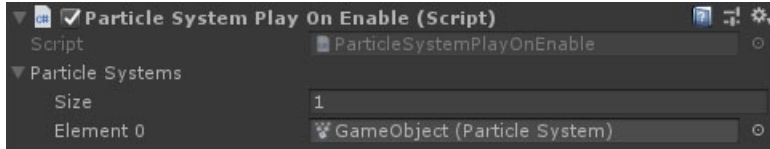| NAME	TYPE	DESCRIPTION
Offset Position	Vector3	The offset from the starting position to move to.
Duration	Float	The time taken to move from position to position.

See Also

ParticleSystemPlayOnEnable MonoBehaviour

Overview

The ParticleSystemPlayOnEnable behaviour is a simple utility component that is used to get the effect of "Play On Awake" on a particle system that is recycled and reused multiple times by the pooling system.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Particle Systems	Particle System Array	The particle systems to play on enabling the object.

See Also

PoolManager

Utilities

PooledObject MonoBehaviour

Overview

The PooledObject behaviour is used to manage object lifecycle for a reusable object. It is managed by the PoolManager.

Inspector

Properties

The PooledObject behaviour has no properties exposed in the inspector.

See Also

PoolManager

Utilities
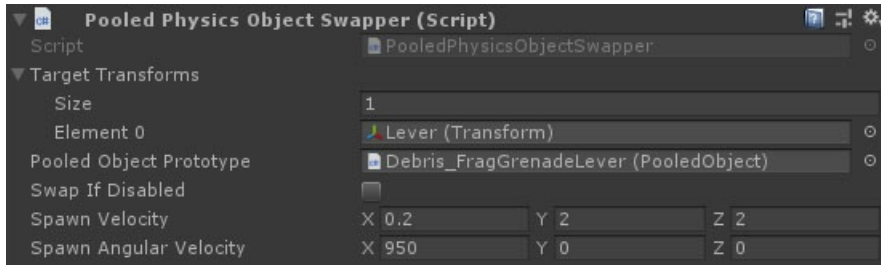
PooledObjectSwapper MonoBehaviour

Overview

The PooledPhysicsObjectSwapper behviour swaps out a number of objects with pooled objects.

Inspector
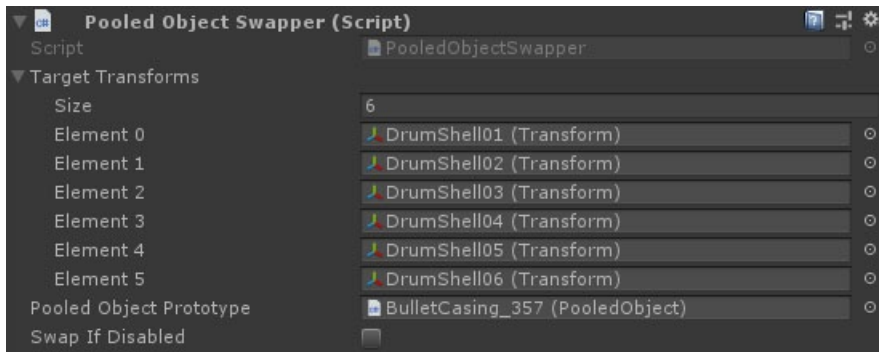
Properties

NAME	TYPE	DESCRIPTION
Target Transforms	Transform Array	The objects to swap.
Pooled Object Prototype	PooledObject	The pooled object to swap the target objects with.
Swap If Disabled	Boolean	Swap the target objects if they are disabled.

See Also

PooledObject

PooledPhysicsObjectSwapper

PooledPhysicsObjectSwapper MonoBehaviour

Overview

The PooledPhysicsObjectSwapper behviour swaps out a number of objects with pooled physics objects. An example use is in the revolver where it swaps the empty shells for physics objects during a reload.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Target Transforms	Transform Array	The objects to swap.
Pooled Object Prototype	PooledObject	The pooled object to swap the target objects with.
Swap If Disabled	Boolean	Swap the target objects if they are disabled.
Spawn Velocity	Vector3	The velocity to spawn the physics object at relative to the target object rotation.
Spawn Angular Velocity	Vector3	The angular velocity of the spawned physics object.

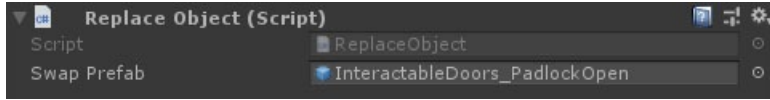## See Also

PooledObject

PooledObjectSwapper

Unity Rigidbody

ReplaceObject MonoBehaviour

Overview

The ReplaceObject behaviour simply replaces the object it's attached to with the prefab in its settings. It needs triggering via code or using an event on something like an interactive object or trigger zone.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Swap Prefab	GameObject	The object to replace this one with.

See Also

SlowMoSystem

CharacterTriggerZone

ScenePoolHandler ScriptableObject

Overview

The ScenePoolHandler behaviour manages PooledObject prefab spawning and reuse within a scene.

Inspector
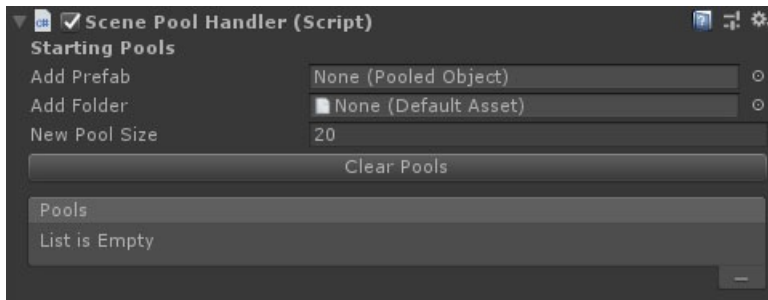
Properties

NAME	TYPE	DESCRIPTION
Add Prefab	PooledObject	Adds a prefab to the pools list below, using the new pool size setting.
Add Folder	Folder	Adds all the prefabs within the folder and all its sub-folders to the pools list below, using the new pool size setting.
New Pool Size	Integer	The pool size to use for any new tools added to the pools list below.
Pools	Pool Info Array	The pools to set up at initialisation.

Pool Info

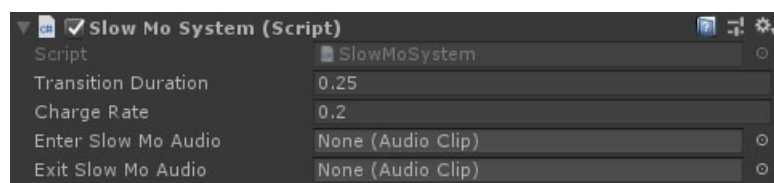| NAME	TYPE	DESCRIPTION
Prototype	PooledObject	The prefab object to spawn.
Count	Int	The number of objects to instantiate for the pool.

See Also

PooledObject

PoolManager

Utilities

SlowMoSystem MonoBehaviour

Overview

The SlowMoSystem behaviour is added to a the player character and exposes a simple API to allow slow-motion effects.

Inspector

Properties

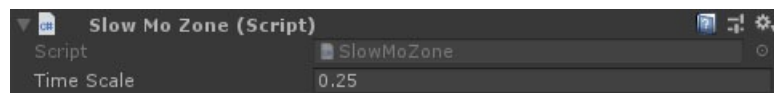| NAME	TYPE	DESCRIPTION
Transition Duration	Float	The (real, unscaled) time taken to transition time scales.
Charge Rate	Float	The amount of charge added per (real, unscaled) second.
Enter Slow Mo Audio	AudioClip	The sound to play on entering slow-mo.
Exit Slow Mo Audio	AudioClip	The sound to play on exiting slow-mo.

See Also

InteractiveObject

SlowMoZone MonoBehaviour

Overview

The SlowMoZone behaviour derives from CharacterTriggerZone and is used to define an area that will cause time to slow down when entered, and return to normal once exited.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Time Scale	Float	The time scale inside this zone.

See Also

SlowMoSystem

CharacterTriggerZone

Teleporter MonoBehaviour

Overview

The Teleporter behaviour is used to teleport a character from one point to another. It is triggered via an InteractiveObject.

Inspector
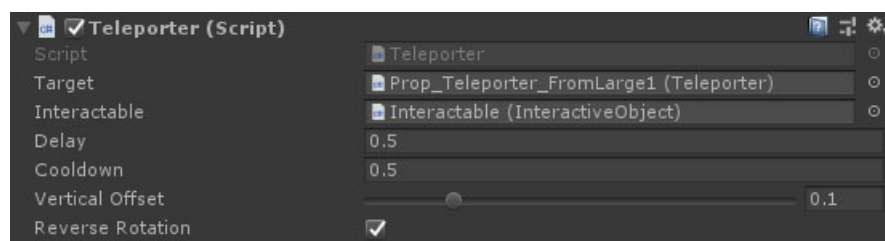
Properties

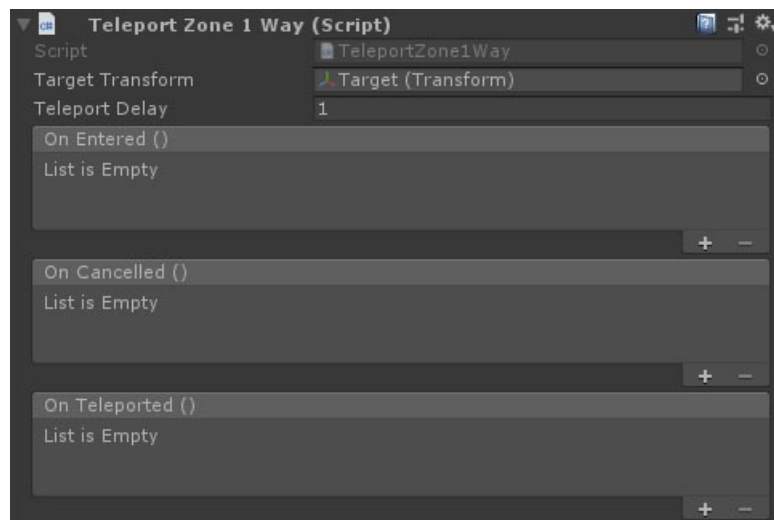| NAME	TYPE	DESCRIPTION
Target	Teleporter	The target teleporter to teleport to.
Interactable	InteractiveObject	The interactive object that triggers the teleport. Is disabled for cooldown.
Delay	Float	The delay between triggering and teleport.
Cooldown	Float	The cooldown blocks teleport until it has completed.
Vertical Offset	Float	An amount to raise the character on teleport to prevent ground overlap.
Reverse Rotation	Boolean	Should the relative rotation be reversed on teleport. For example walking in to teleporter A translates to walking out of teleporter B.

See Also

InteractiveObject

TeleportZone1Way MonoBehaviour

Overview

The TeleportZone1Way behaviour is used to teleport a character to a target transform position and rotation. It can be set with an optional delay and exposes events to control charge up and teleport / cancel effects.

Inspector

Properties

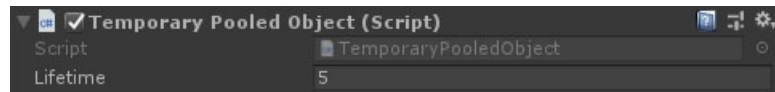| NAME	TYPE	DESCRIPTION
Target Transform	Transform	The transform to teleport to. The character will match its position and orientation.
Teleport Delay	Float	The time between entering the trigger zone and teleporting. Allows for effects and feedback to communicate what's going on.
On Entered	UnityEvent	An event fired as soon as a character enters the teleport trigger zone.
On Cancelled	UnityEvent	An event fired if the character leaves before the teleport delay is over.
On Teleported	UnityEvent	An event fired after the character is teleported.

See Also

InteractiveObject

TemporaryPooledObject MonoBehaviour

Overview

The TemporaryPooledObject behaviour is used to manage object lifecycle for a reusable object. It is spawned from the PoolManager, and will be returned after the specified time has elapsed.

Inspector

Properties

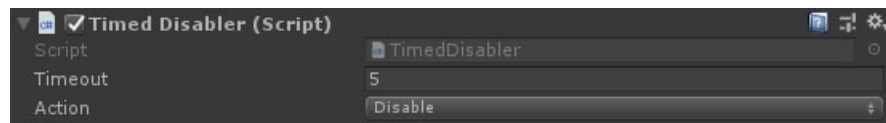| NAME	TYPE	DESCRIPTION
Lifetime	Float	The duration the object will stay active before returning to the pool.

See Also

PoolManager

Utilities

TimedDisabler MonoBehaviour

Overview

The TimedDisabler behaviour is attached to an object to disable or destroy it after a set time.

Inspector

Properties

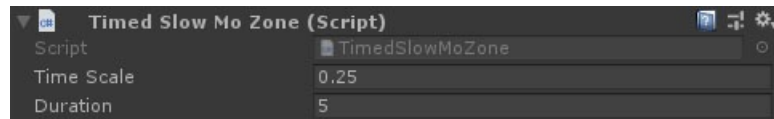| NAME	TYPE	DESCRIPTION
Timeout	Float	The time after starting that the action will be performed.
Action	Dropdown	The action to performm on timeout. Options are **Disable** and **Destroy**

See Also

TimedSlowMoZone MonoBehaviour

Overview

The TimedSlowMoZone behaviour derives from CharacterTriggerZone and is used to define an area that will cause time to slow down when entered, and return to normal after a set period of (unscaled) time has passed.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Time Scale	Float	The time scale inside this zone.
Duration	Float	The duration the slow-mo effect should last.

See Also

SlowMoSystem
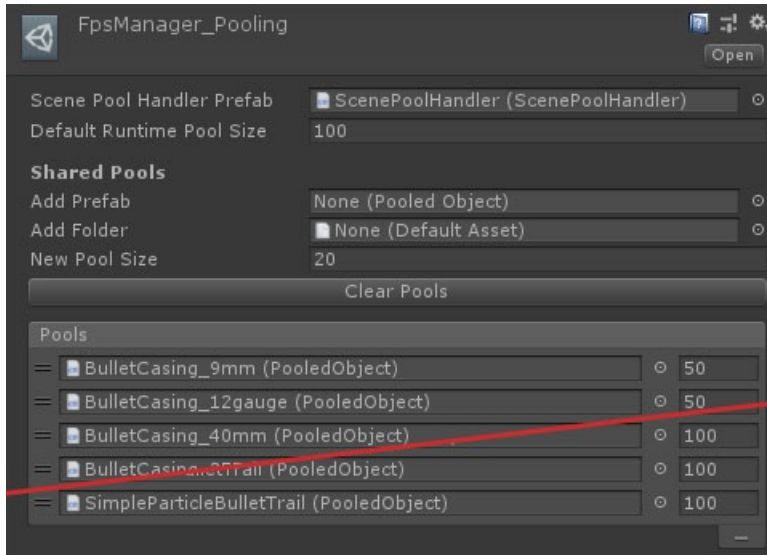
CharacterTriggerZone

PoolManager ScriptableObject

Overview

The PoolManager scriptable object specifies the PooledObject prefabs available across all scenes, and handles creation or registration of the ScenePoolHandler after scene loads.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Scene Pool Handler Prefab	ScenePoolHandler	A scene pool handler prefab to be instantiated in the scene if none already exists.
Default Runtime Pool Size	Int	The size of any new pools that are added to the array after initialisation.
Add Prefab	PooledObject	Adds a prefab to the pools list below, using the new pool size setting.
Add Folder	Folder	Adds all the prefabs within the folder and all its sub-folders to the pools list below, using the new pool size setting.
New Pool Size	Integer	The pool size to use for any new tools added to the pools list below.
Pools	Pool Info Array	The pools to set up at initialisation.

Pool Info

NAME	TYPE	DESCRIPTION
Prototype	PooledObject	The prefab object to spawn.
Count	Int	The number of objects to instantiate for the pool.

See Also

PooledObject

ScenePoolHandler

Utilities

Surfaces

Overview

The NeoFPS surfaces system is an in-development system for querying impact surface materials. It is used in a number of places in the NeoFPS framework, including Footsteps and Bullet Hit Effects.

Surface Manager

The SurfaceManager scriptable object is located in the NeoFPS/Resources folder, and accessible through the NeoFPS hub. The surface manager associates specific surfaces with effects and audio clips for use in bullet hits, etc. You can also place a SurfaceFxOverrides behaviour in the scene in order to override the surface effects on a scene by scene basis.

Adding New Surface Types

The `Surface` identifier is a generated constant that lists the available surfaces in the game. Adding new surfaces is as simple as editing the ConstantsSettings and regenerating the constant. Keep in mind that in the inspector generated constants are stored by index. This means that reordering or removing existing surfaces will not be correctly picked up in inspector properties. This means that it is best to try and decide on the available surfaces early in a project if possible.

Once you have added new surface types, you can then access these when assigning audio clips in the SurfaceAudioData assets used for things like footstep and bullet hit audio, and the SurfaceHitFxData asset used to specify impact visual effects among others.

Surface Behaviours

Surface behaviours (currently just SimpleSurface) are attached to physics objects that need a surface ID such as environment or prop colliders. This means that if the object is shot, hit or stepped on the surface behaviour will tell the relevant system the surface they need to react to with audio or graphics effects.

New surface behaviours will be added in future updates.

Adding New Bullet Hit VFX

The SurfaceManager holds a reference to the SurfaceHitFxData asset that specifies all the bullet hit VFX for different surface types. This is currently shared for all bullet and melee impacts, but the future you will be able to specify separate hit FX data for different attack types. The architecture of this has not been decided yet.

Each of the entries in the SurfaceHitFxData asset references a prefab with a component that implements the `BaseHitFxBehaviour` base class. The demos all use the ParticleSystemHitFxBehaviour which works by moving a group of particle systems to align with the hit point, and calling `Emit()` on them. This means that all of the particles per surface type reuse the same particle systems instead of having hundreds of separate systems all working at once. Another option that helps when replacing the impact FX with other assets is the PooledObjectHitFxBehaviour. This uses the pooling system to recycle objects for efficiency. This can be combined with the ParticleSystemPlayOnEnable component to trigger particle systems each time a pooled object is activated.

You can also implement your own FX by writing a script with a class that inherits from the `BaseHitFxBehaviour` base class.
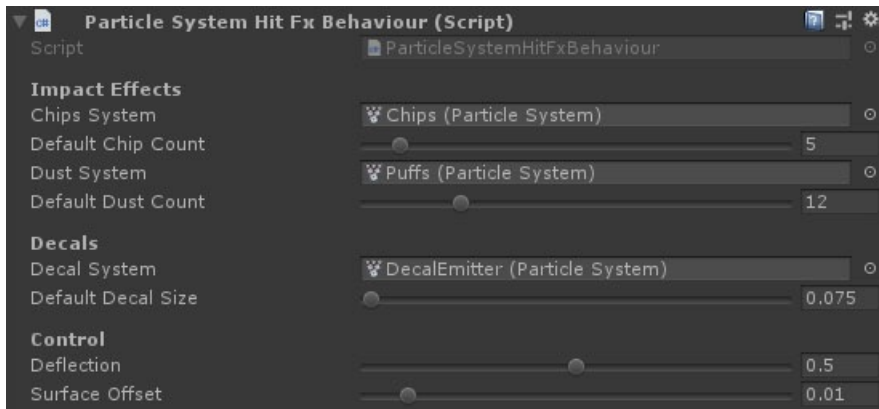

See Also

SurfaceManager

SimpleSurface

ParticleSystemHitFxBehaviour MonoBehaviour

Overview

The ParticleSystemHitFxBehaviour ties together a number of particle systems to create surface impact effects.

Inspector

Properties

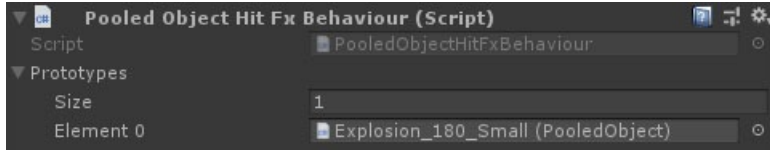| NAME	TYPE	DESCRIPTION
Chips System	ParticleSystem	The particle system for chunks of surface material (eg. wood splinters, paint chips).
Default Chip Count	Int	The number of chip particles for a bullet hit with size 1.
Dust System	ParticleSystem	The particle system for fine dust spray or similar.
Default Dust Count	Int	The number of dust particles for a bullet hit with size 1.
Decal System	ParticleSystem	The particle system used to place a decal.
Default Decal Size	Float	The amount of deflection for the hit system (0 follows the impact normal, 1 follows the reflected impact ray).
Deflection	Float	The amount of deflection for the hit system (0 follows the impact normal, 1 follows the reflected impact ray).
Surface Offset	Float	The distance from the surface to place the decal (prevents z-fighting).

See Also

Surfaces

SurfaceHitFxData

PooledObjectHitFxBehaviour MonoBehaviour

Overview

The PooledObjectHitFxBehaviour spawns an object at the hit location & rotation using the NeoFPS pooling system. This can be used with a TemporaryPooledObject to ensure the spawned object is returned to the pool after a brief delay.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Prototypes	PooledObject	The pooled objects to choose from.

See Also

Surfaces

SurfaceHitFxData

SimpleSurface MonoBehaviour

Overview

The SimpleSurface behaviour is attached to objects in the scene and assigns them a surface tag for responding to effects.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Surface	FpsSurfaceMaterial	The surface material ID.

See Also

Surfaces

Generated Constants
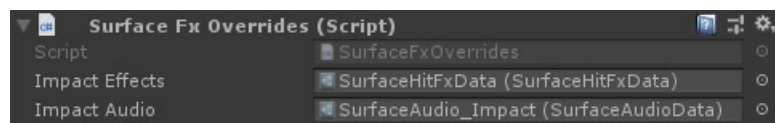
SurfaceFxOverrides MonoBehaviour

Overview

The SurfaceFxOverrides behaviour is used to override the surface effects of the SurfaceManager for a specific scene.

Inspector

Properties

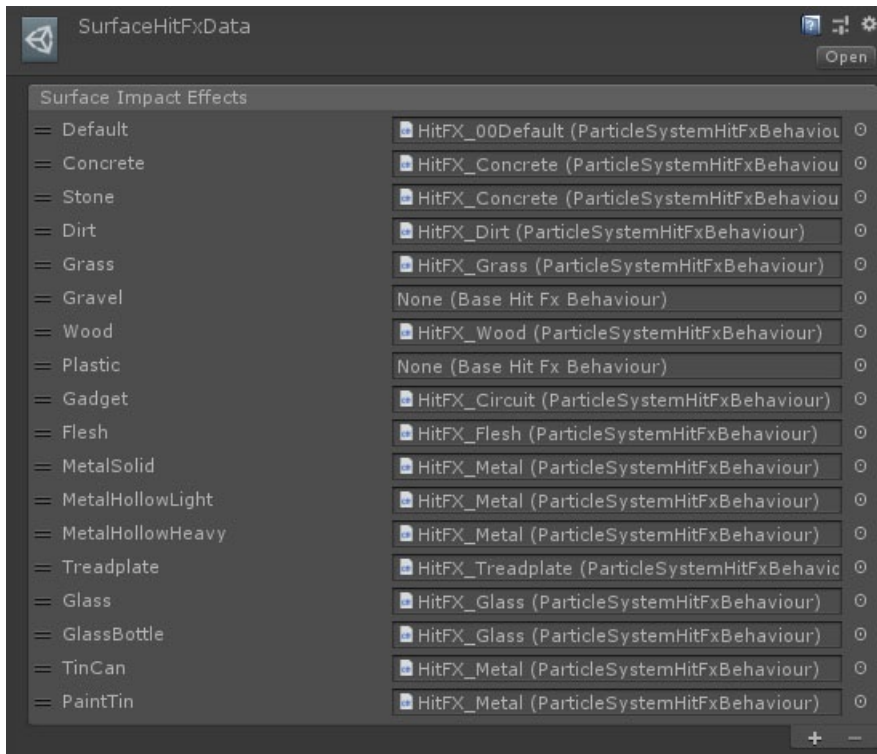| NAME	TYPE	DESCRIPTION
Impact Effects	SurfaceHitFxData	The impact special effects for things like bullet hits.
Impact Audio	SurfaceAudioData	The audio library for impact audio, eg. bullet hits.

See Also

Surfaces

SurfaceManager

SurfaceHitFxData

SurfaceAudioData

SurfaceHitFxData ScriptableObject

Overview

The SurfaceHitFxData scriptable object specifies per-surface visual effects for impacts.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Data	HitFx Array	Per-surface hit effects. Must inherit from the `BaseHitFxBehaviour` class. Entries correspond to the values in the FpsSurfaceMaterial generated constant.

See Also

Surfaces
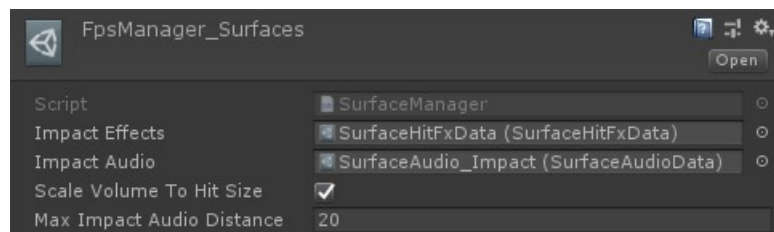
ParticleSystemHitFxBehaviour

SurfaceManager ScriptableObject

Overview

The SurfaceManager scriptable object is a central manager that handles visual and audio impact effects with various surfaces. You can also override this on a scene by scene basis by using a SurfaceFxOverrides behaviour.

Inspector

Properties

NAME	TYPE	DESCRIPTION
Impact Effects	SurfaceHitFxData	The impact special effects for things like bullet hits.
Impact Audio	SurfaceAudioData	The audio library for impact audio, eg. bullet hits.
Scale Volume To Hit Size	Bool	Should impact audio be louder for heavier hits.
Max Impact Audio Distance	Float	The maximum distance from the player character to play impact audio.

See Also

Surfaces

SurfaceHitFxData

SurfaceAudioData