# MANUAL

## INVENTORY AND WEAPONS MODDING SYSTEM (ICWM)

version 7.9.0b

# INTRODUCTION

English is not my first language, so if you do not understand what I mean, please contact me and I will try my best to assist you.

This manual was created to give the user a better understanding of ICWM. The Asset and its features will be discussed below. ICWM consists of 94% C # code. Users who know the basics of this programming language or similar languages will easily understand the logic of the Asset. Initially, it was planned to create only a modding system, but in the end, it became clear that it would be difficult to implement the relationship between the inventory and the modding system, therefore, they were combined into this single Asset.

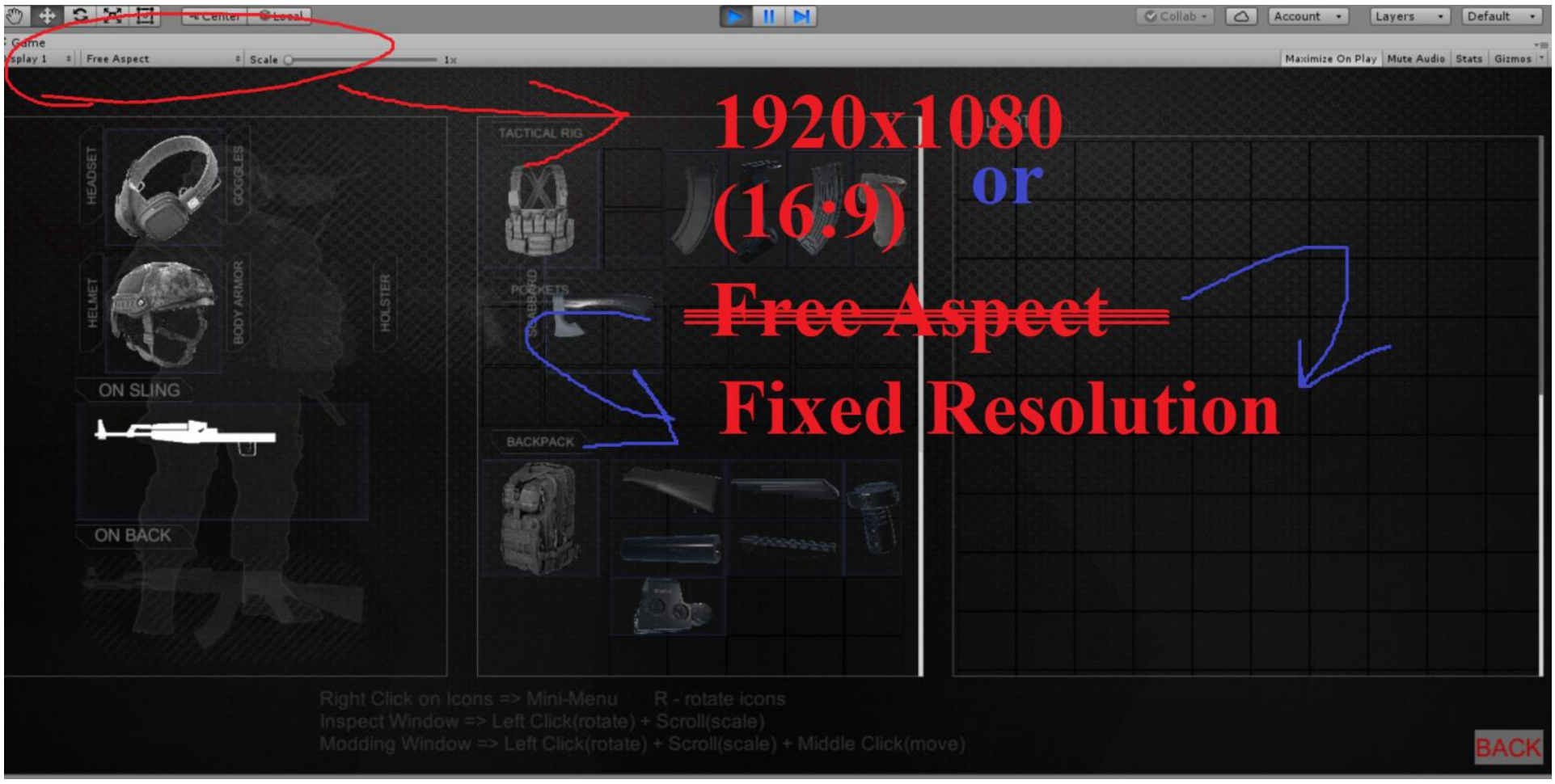
The Asset has its own set of requirements outlined below, and not following them ban lead to bugs occurring with ICWM. In the future, they will be simplified to automatism. This is a difficult task, because the entire logic of the system needs to be rebuilt.

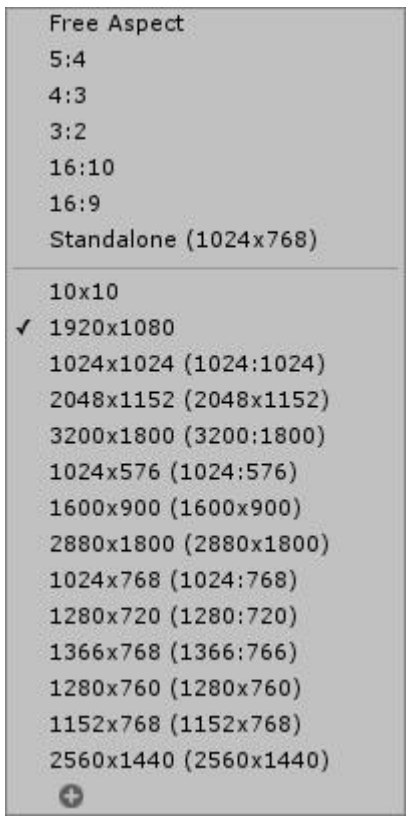
The Asset code is commented in sections where it is often not clear what the code is doing. The simpler code sections are left uncommented, as basic code knowledge is all that is required to interpret the functionality.

# Table of Contents

## BEFORE STARTING

On the second screen, you see the screen resolution that was tested.

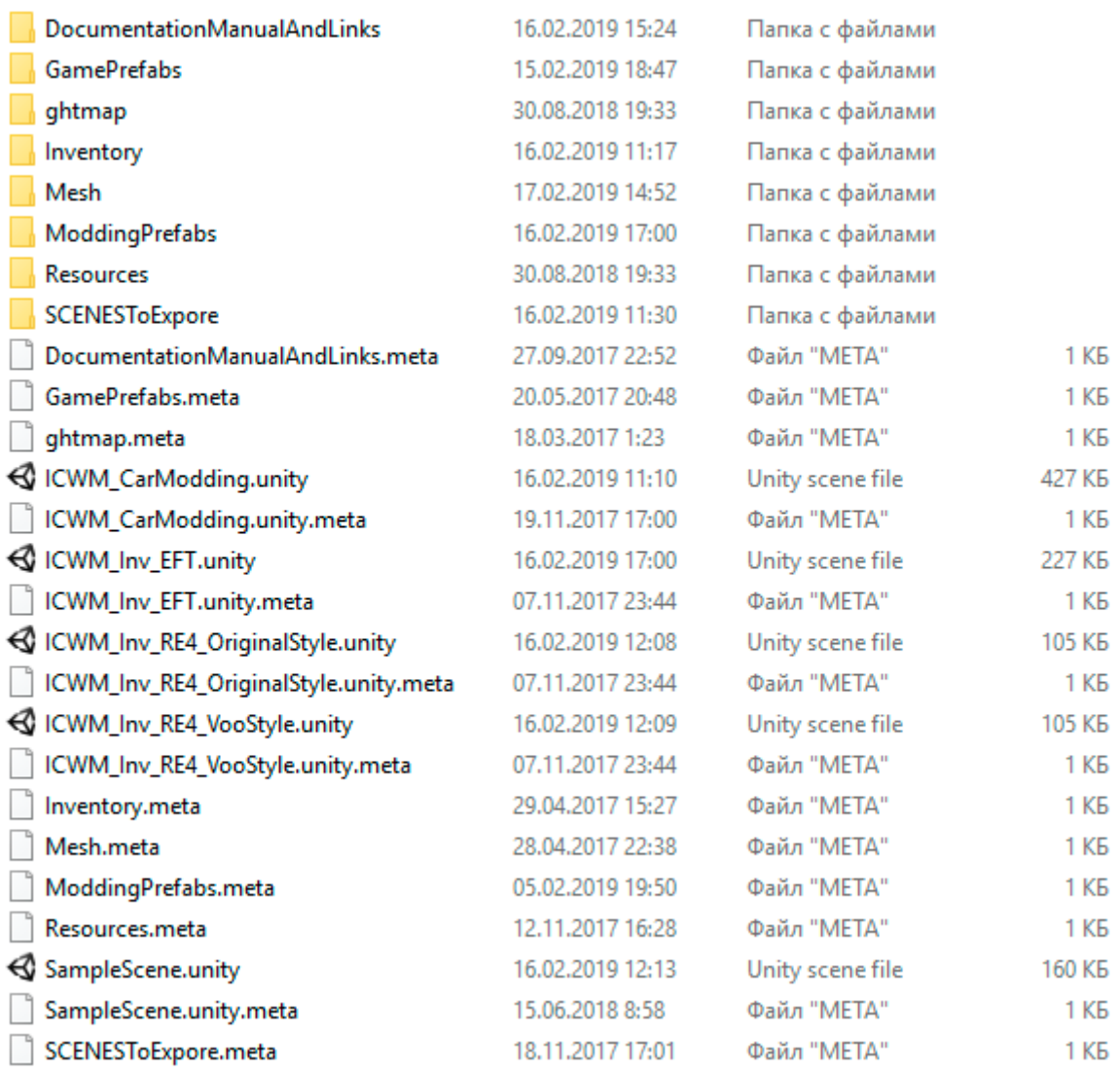


## HOW TO GET STARTED?

When you downloading an asset, you will have a folder named *AssetICWM*:

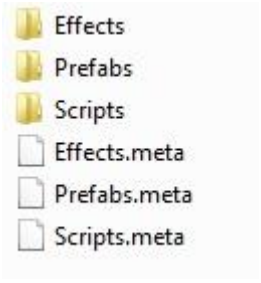

Open it:




Where:

1) DocumentationManualAndLinks – documentation;
2) GamePrefabs – all game prefabs;
3) ghtmap – baking of lightmap;
4) Inventory – everything that is necessary for this asset (consider later) (scripts, icons, materials…);
5) Mesh – all game prefabs, textures, materials;
6) ModdingPrefabs – all game prefabs, with witch the inventory interacts;
7) Resources – some material for the menu in the top panel and scripts;
8) ScensToExplore – scenes to study the characteristics of the Asset individually.

**Inventory:**

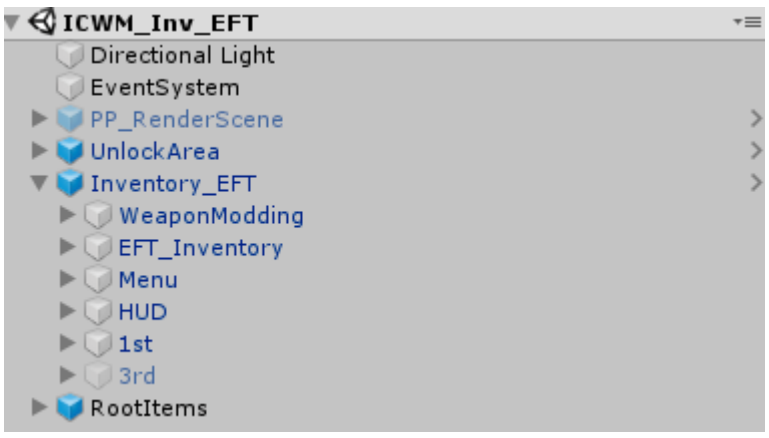



Where:

1) Effects – here all the icons, textures, materials that are used in inventory and modding
2) Prefabs – all prefabs that are necessary for the operation of the asset
3) Scripts – scripts

After running Unity scene (EFT_Inventory):


Consider in more detail the window with a hierarchy:




Where:

1) The elements PP_RenderScene и UnlockArea described in the Unlock System documentation;
2) WeaponModding – here everything happens that is connected with the modding of weapons;
3) EFT_Inventory – here everything happens that is connected with the inventory;
4) Menu – base menu (Press “Esc” to activate);
5) HUD – simple version HUD;
6) Player (1st – 3rd) – our player(FPS and TPS);
7) RootItems – all the items that are in the scene, and also there will spawn all the items that were thrown out of the inventory.


**WeaponModding:**

Where:

1) WeaponViewModding – here our weapons will spawn and icons will be created there
2) WeaponRenderScreen – here is rendered images for the weapon icon
3) PlayerInspectWindow – render for the inspect (Inspect)
4) Root… - here our attachments and weapons will spawn

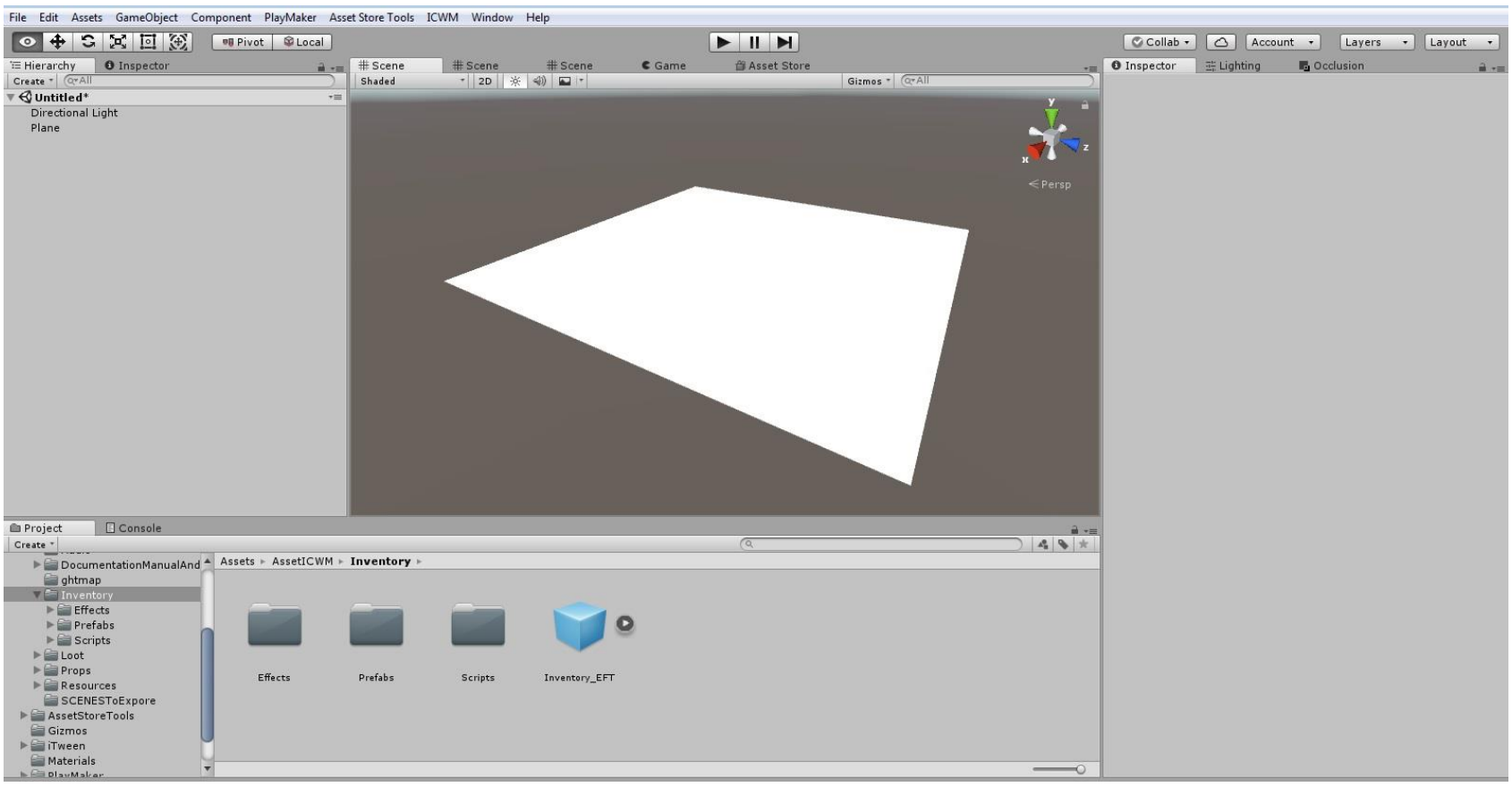
**Inventory:**

Where:

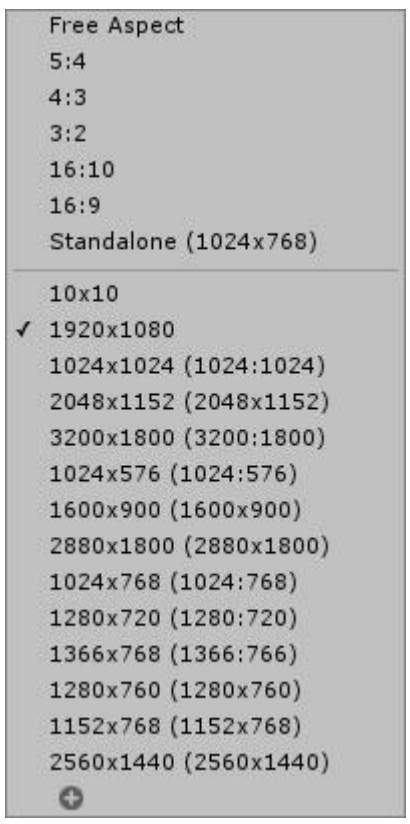
1) Background – background;
2) TopMenu – button bar at the top of the screen (inventory);
3) PageGear – the backpack area, player icons, another player's menu and so on;
4) PageShop – store menu;
5) Hotbar – …;
6) Active – all active buttons, images, etc., which will be visible in the modding mode;
7) Root… – – also, to spawn in them the necessary components.

# HOW TO ADD IT TO A NEW SCENE?

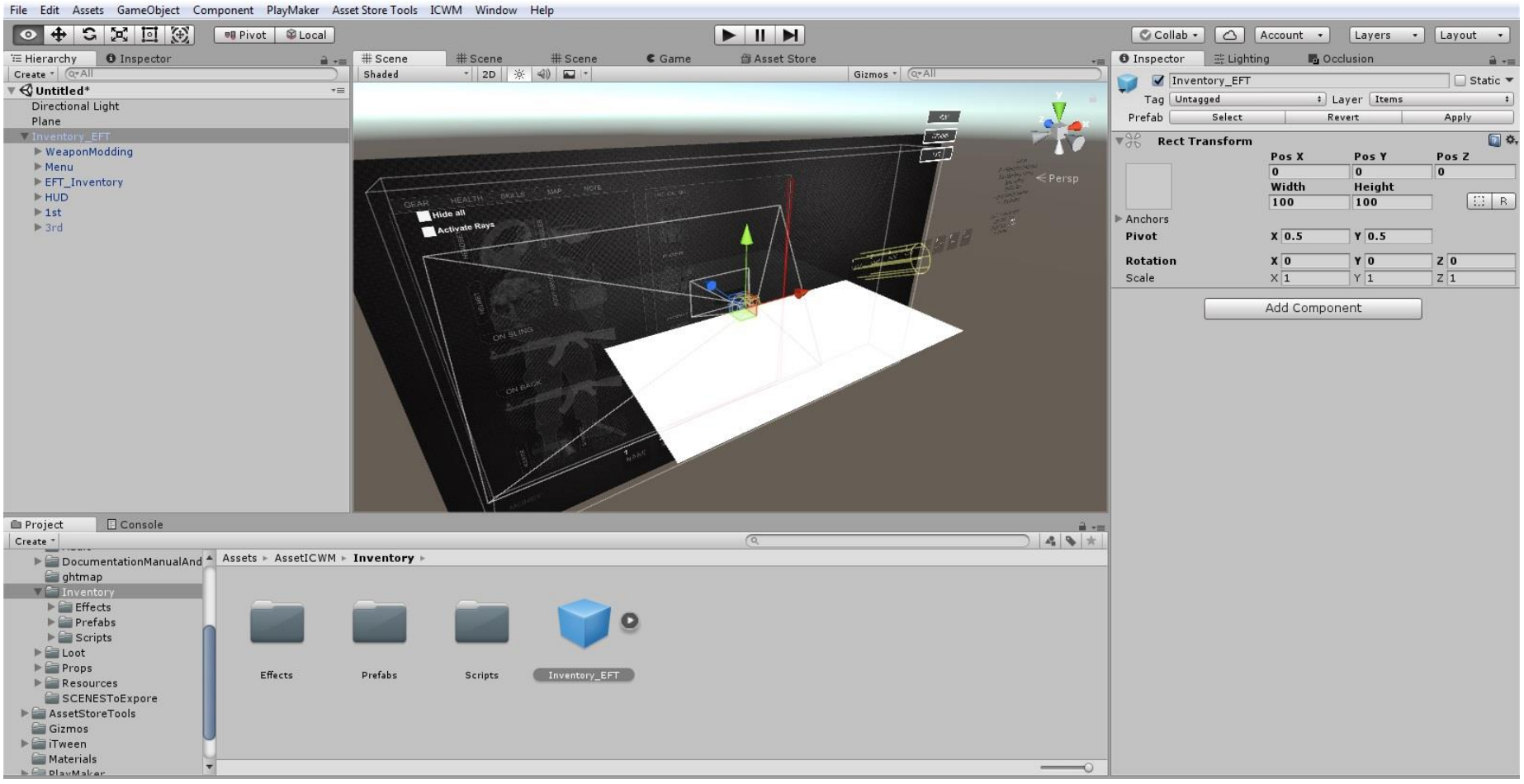
1) Create a new scene or open an old scene.




2) Set the appropriate screen resolution. Remove Free Aspect or Aspect Ratio and put Fixed Resolution, otherwise some items of inventory will not fit on the screen.

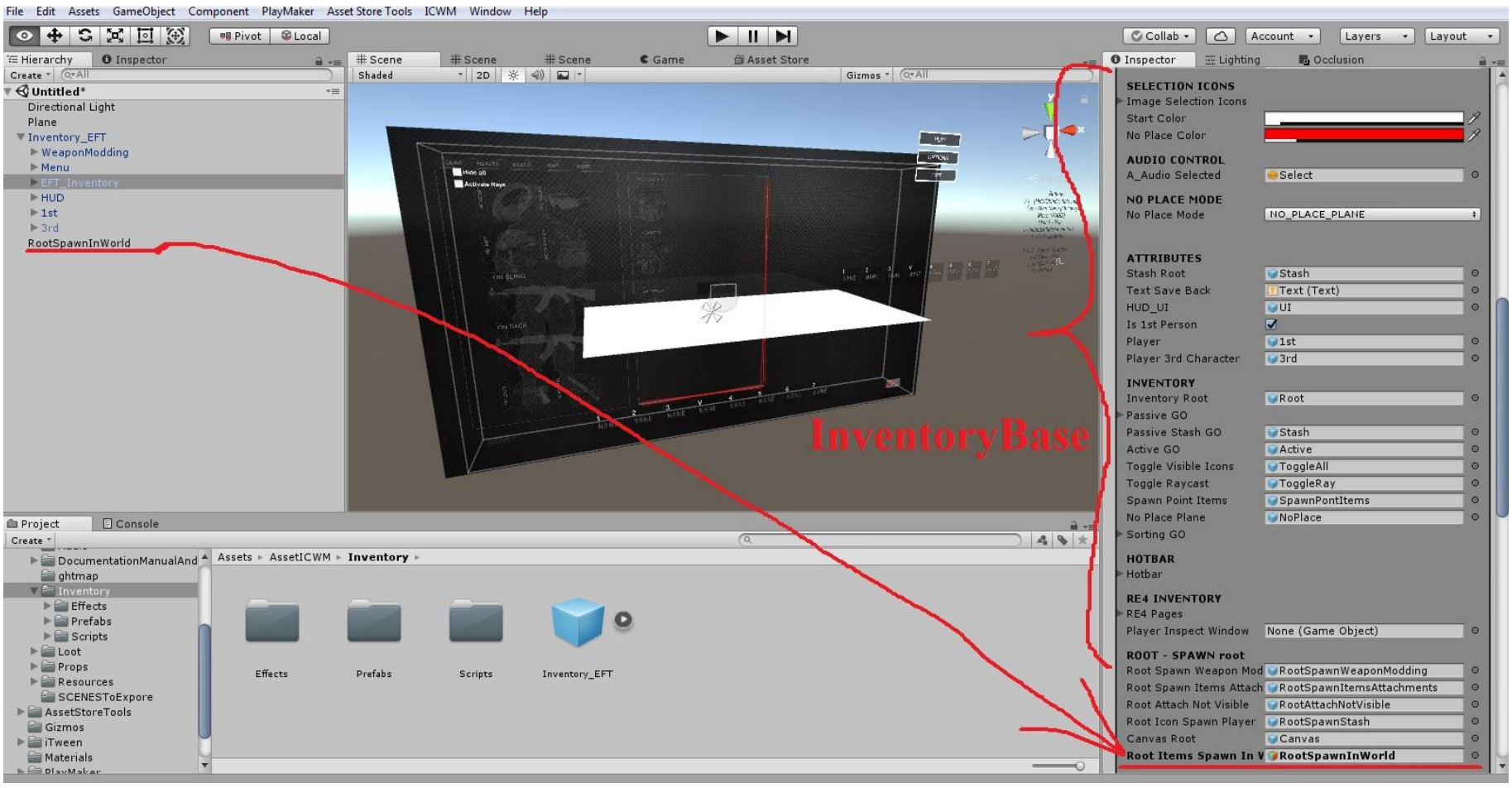


3) Add the necessary inventory from the Inventory folder to the scene.




4) Adjust the position of the player (1st or 3rd), so that it does not fall through the ground, add to the scene a plane with a collider;

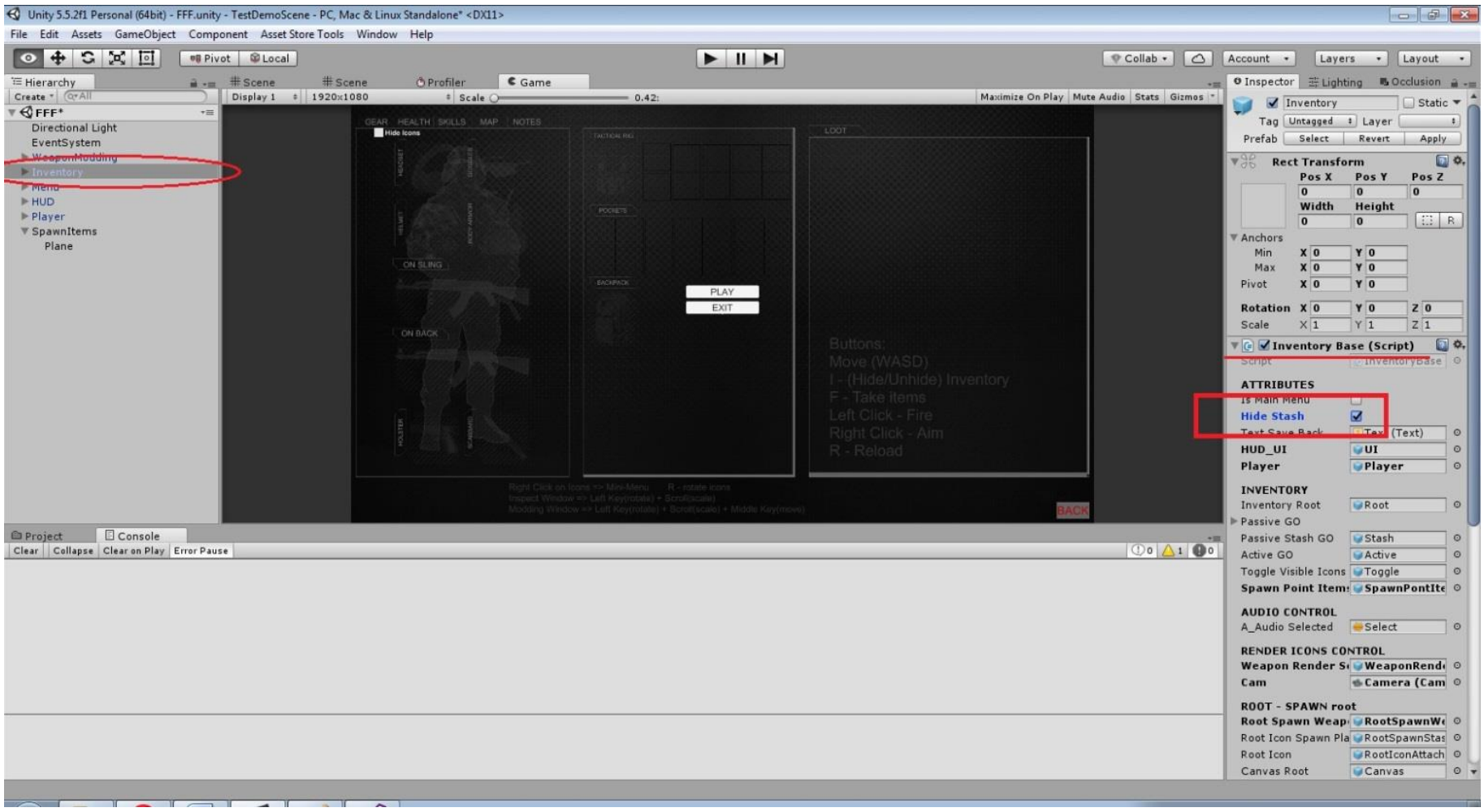
5) Create a GameObject and place it anywhere in the hierarchy to any place, then open the InventoryBase script in the inventory (see image) and drag your created GameObject to the location indicated in the image;




6) Play (if you did everything correctly, then there should be no errors). Oh yeah, do not forget to add the EventSystem to the scene (it's added in the same way as GameObject).

# HOW TO ACTIVATE THE STASH?

Very useful for testing.

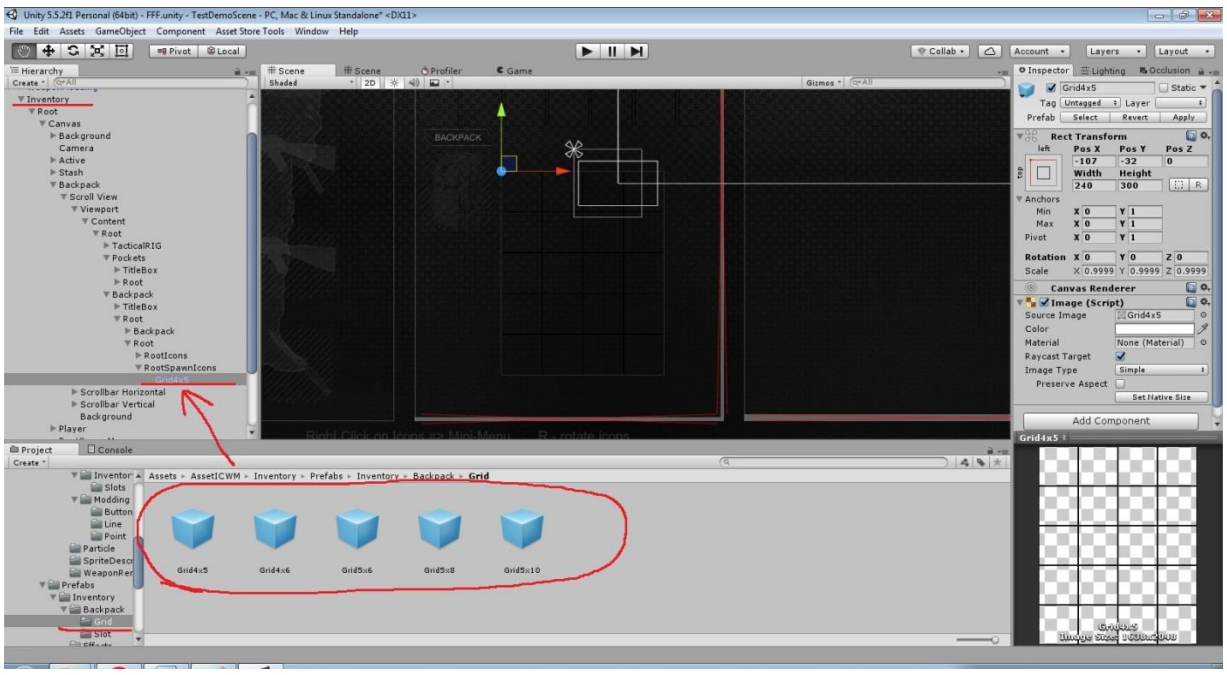


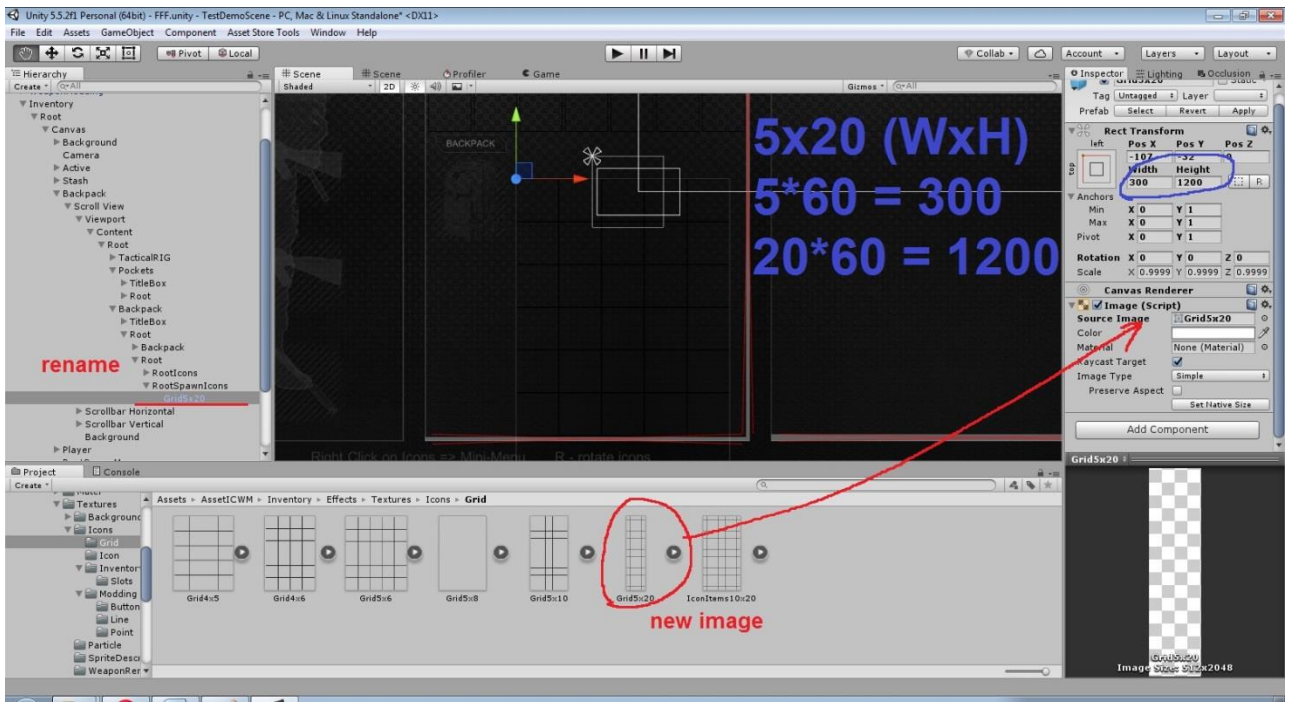
# HOW TO ADD A NEW BACKPACK SIZE?

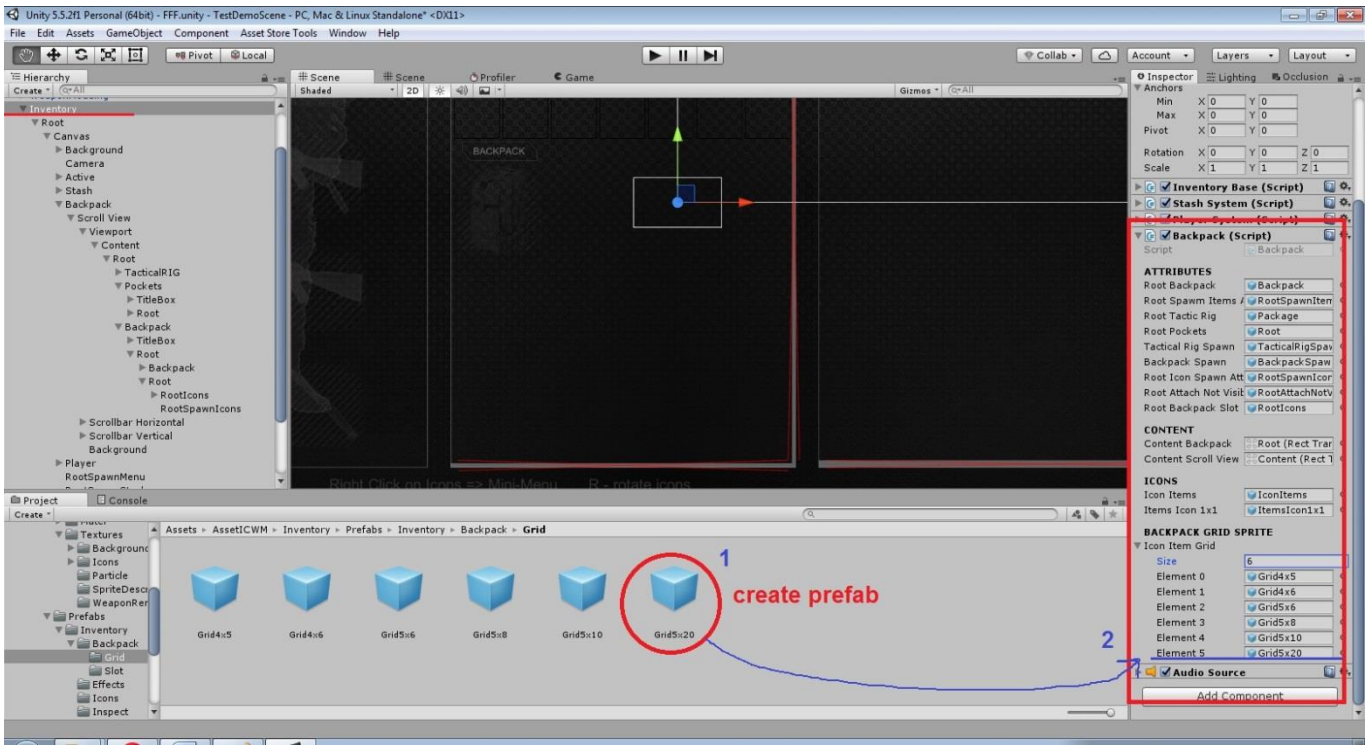
first that you have to do create the grid the required size, for example, create a size5x20.

You can use the examples common\Photoshop\IconItemsV35x6 and change it. I save in the format .png.

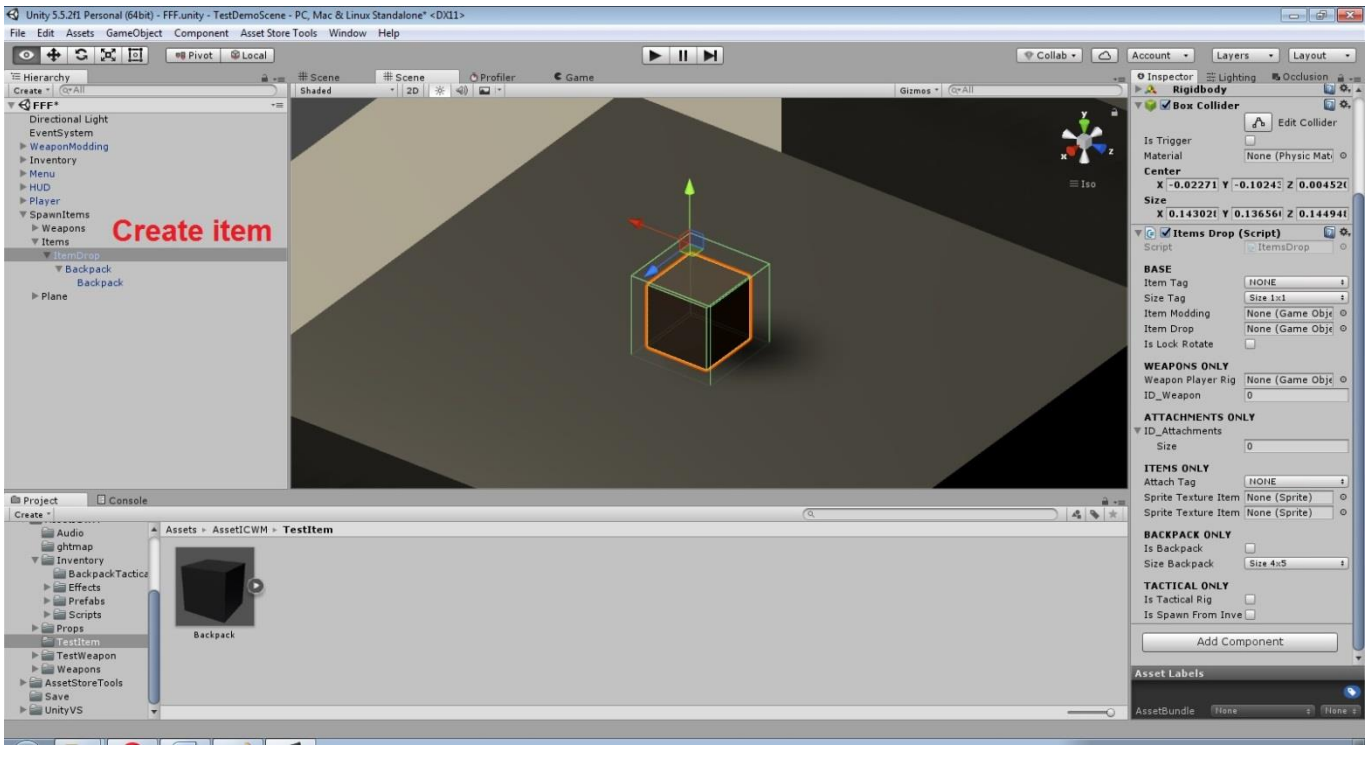
Now we have to create a grid prefab, for this we make use of the available prefab and change it slightly. Do not forget to change the size, the formula on the screen.






How to create a thing I already showed in the video.

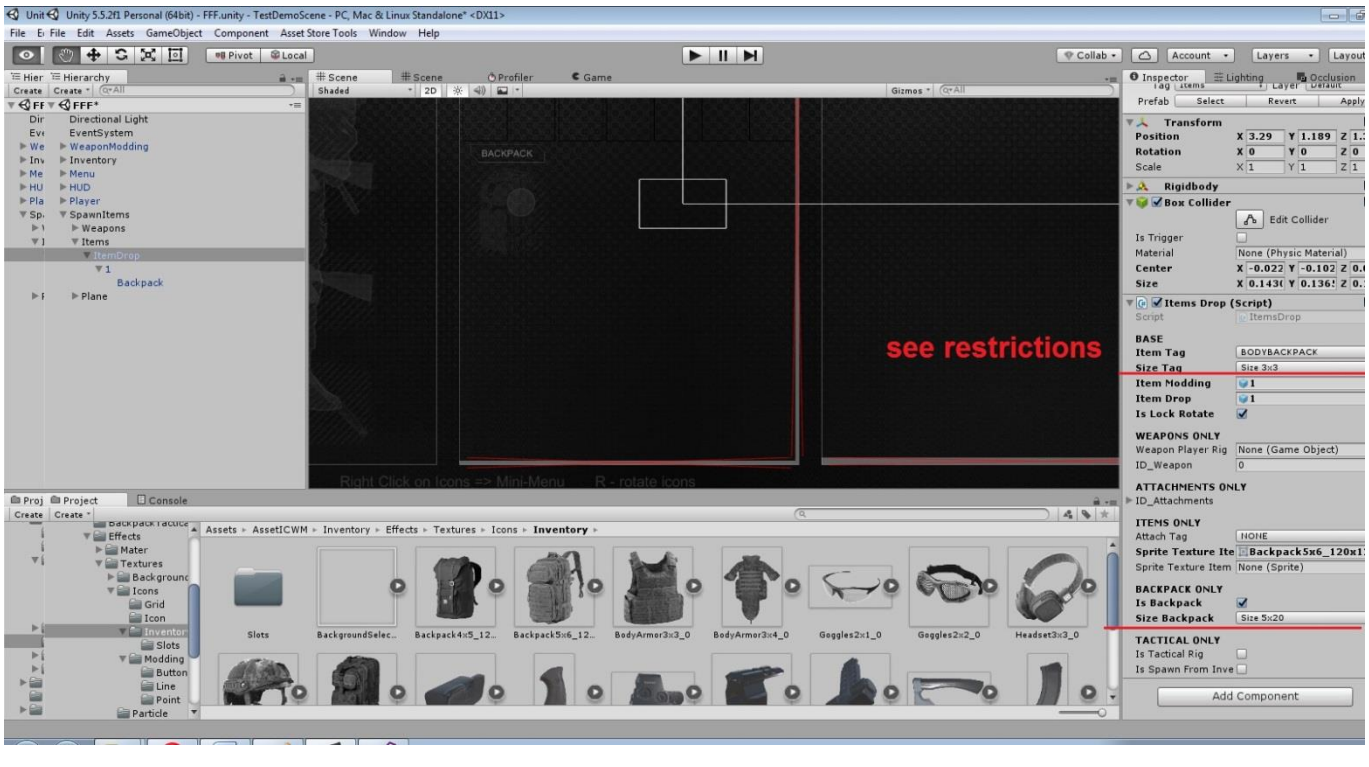



Now we will edit the script.

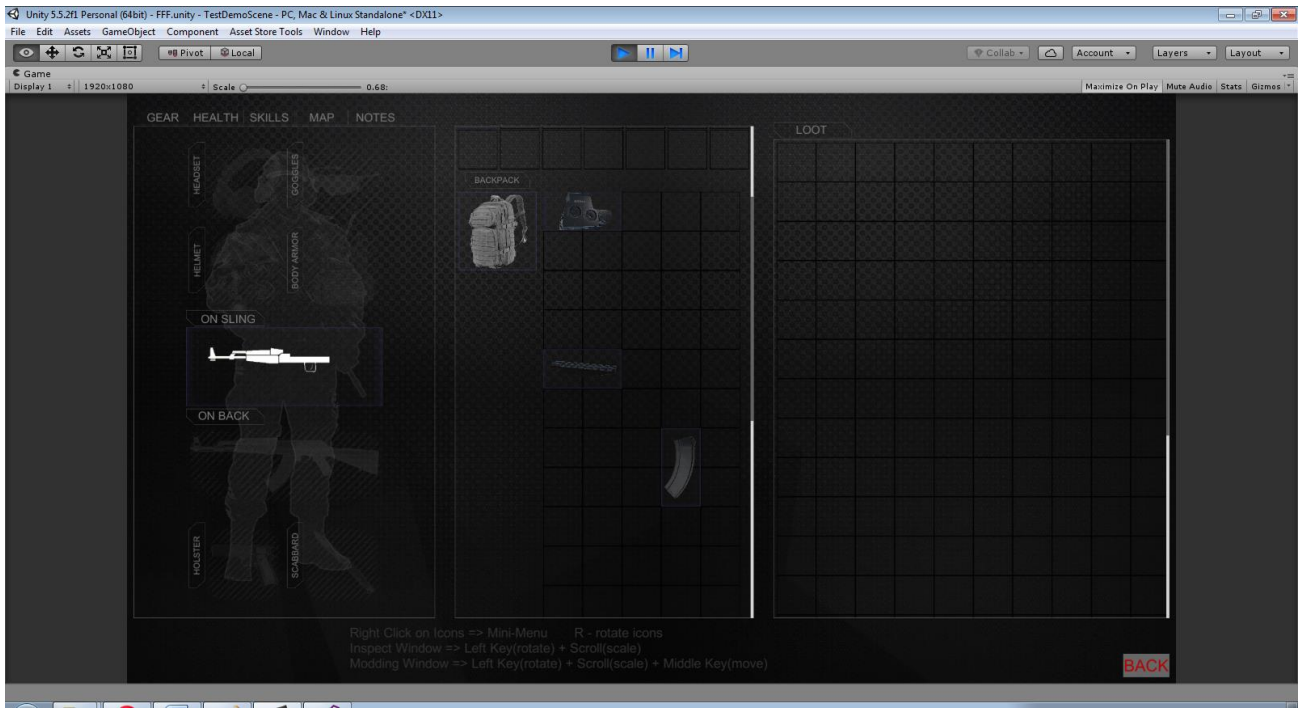
Open the script ClassList.

```
public enum SizeBackpack
    {
        Size4x5,
        Size4x6,
        Size5x6,
        Size5x8,
        Size5x10,
        Size5x20,
        NONE,
    }
public Vector2 GetSizeBackpack()
    {
        if (TagSizeBackpack == ClassList.SizeBackpack.Size4x5)
        {
            return new Vector2(4, 5);
        }
        else if (TagSizeBackpack == ClassList.SizeBackpack.Size4x6)
        {
            return new Vector2(4, 6);
        }
        else if (TagSizeBackpack == ClassList.SizeBackpack.Size5x6)
        {
            return new Vector2(5, 6);
        }
        else if (TagSizeBackpack == ClassList.SizeBackpack.Size5x8)
        {
            return new Vector2(5, 8);
        }
        else if (TagSizeBackpack == ClassList.SizeBackpack.Size5x10)
        {
            return new Vector2(5, 10);
        }
        else if (TagSizeBackpack == ClassList.SizeBackpack.Size5x20)
        {
            return new Vector2(5, 20);
        }
        else
        {
            return new Vector2(0, 0);
        }
    }
public int GetIndexSizeBackpack()
{
    if (TagSizeBackpack == SizeBackpack.Size4x5)
        return 0;
    else if (TagSizeBackpack == SizeBackpack.Size4x6)
        return 1;
    else if (TagSizeBackpack == SizeBackpack.Size5x6)
        return 2;
    else if (TagSizeBackpack == SizeBackpack.Size5x8)
        return 3;
    else if (TagSizeBackpack == SizeBackpack.Size5x10)
        return 4;
    else if (TagSizeBackpack == SizeBackpack.Size5x20)
        return 5;
    else
        return 0;
}
public int GetMasSizeBackpack()
{
    if (TagSizeBackpack == SizeBackpack.Size4x5)
        return 20; // 4 * 5
    else if (TagSizeBackpack == SizeBackpack.Size4x6)
        return 24;
    else if (TagSizeBackpack == SizeBackpack.Size5x6)
        return 30;
    else if (TagSizeBackpack == SizeBackpack.Size5x8)
        return 40;
    else if (TagSizeBackpack == SizeBackpack.Size5x10)
        return 50;
    else if (TagSizeBackpack == SizeBackpack.Size5x20)
        return 100;
    else
        return 0;
}
```

After that, you need to adjust your item.



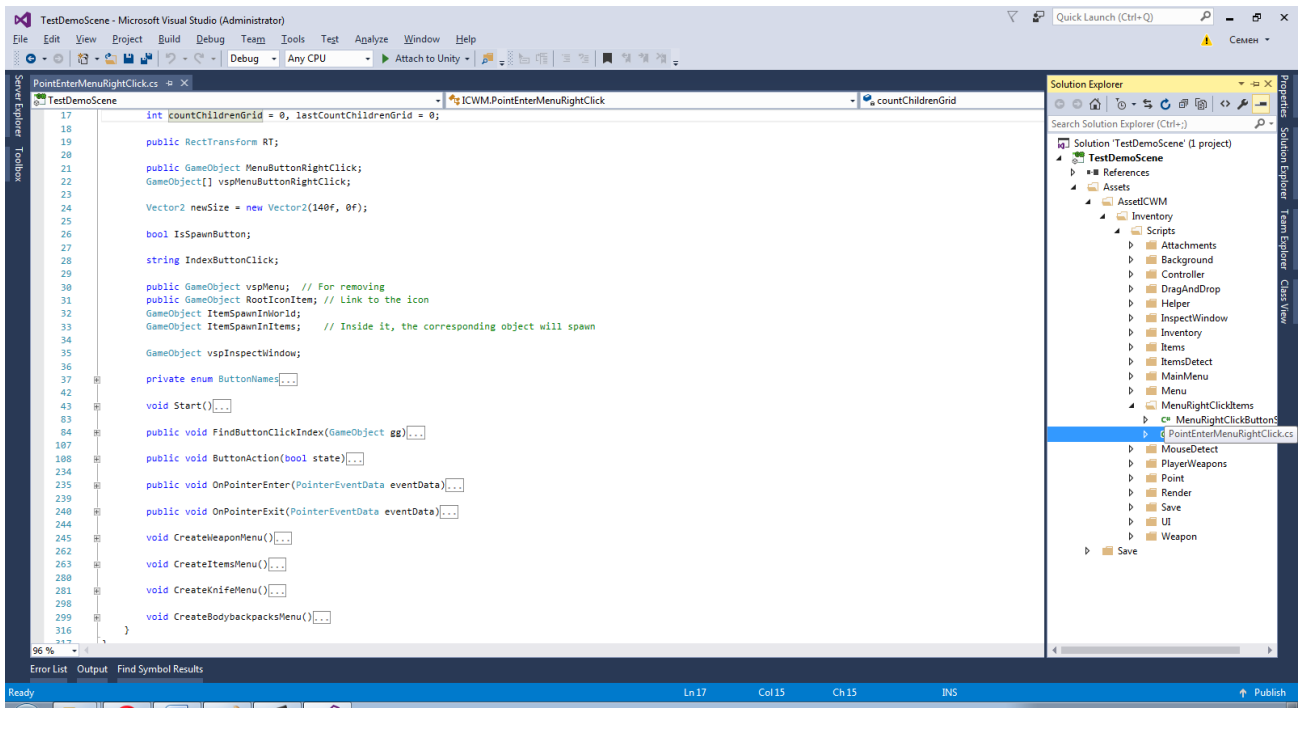

Ready!



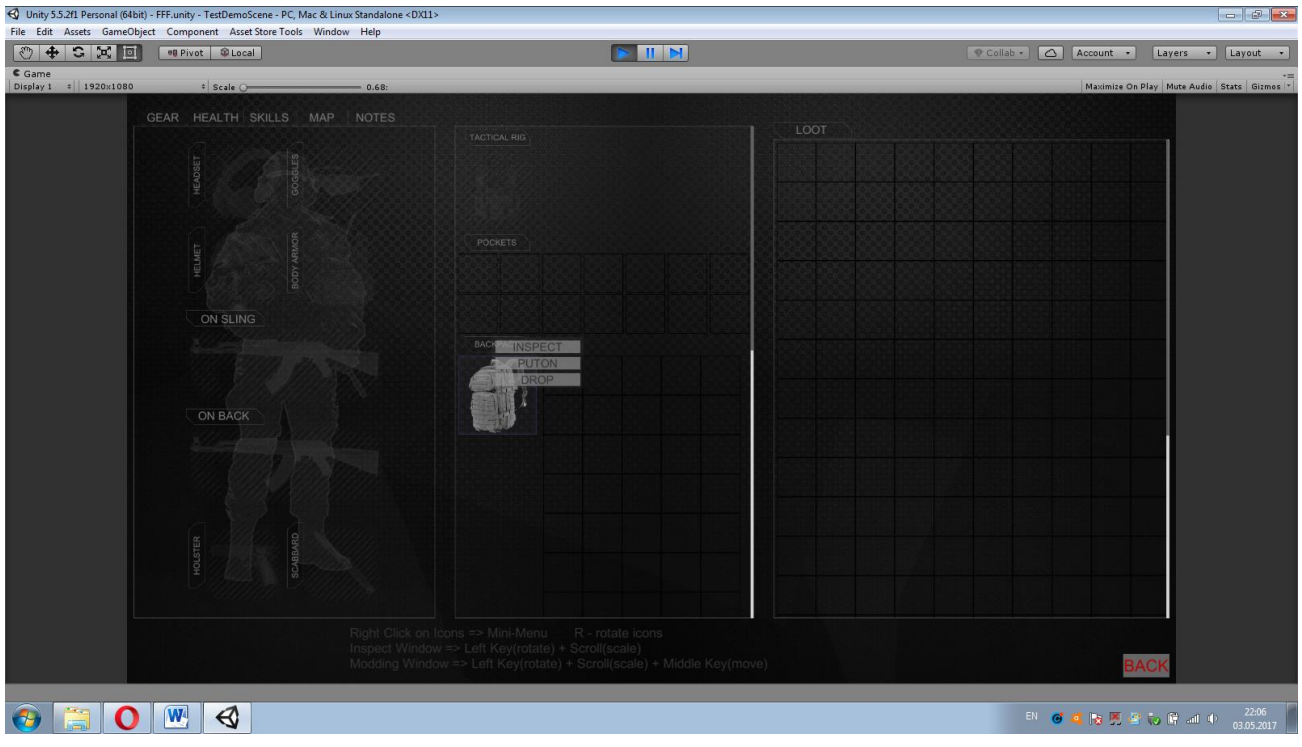
# HOW TO ADD A NEW BUTTON IN THE MINI-MENU?

Now we need to decide which class this button will belong to. All classes have different count of buttons in the menu. For example, add a new button to the backpack, body armor, helmet etc, because they all refer to the player I made them as one class.

Open the script *PointEnterMenuRightClick*.




```
private enum ButtonNames
    {
       DROP,
       MODDING,
       PUTON
    }

void CreateBodybackpacksMenu()
    {
       ButtonNames buttonName = ButtonNames.PUTON;

       vspMenuButtonRightClick = new GameObject[2];
       for (int i = 0; i < vspMenuButtonRightClick.Length; i++)
       {
          vspMenuButtonRightClick[i] = (GameObject)Instantiate(MenuButtonRightClick);
          vspMenuButtonRightClick[i].GetComponent<Transform>().SetParent(transform);
          vspMenuButtonRightClick[i].GetComponent<Transform>().localPosition = Vector3.zero;
          vspMenuButtonRightClick[i].GetComponent<Transform>().localScale = Vector3.one;

          vspMenuButtonRightClick[i].name = buttonName.ToString();
          vspMenuButtonRightClick[i].GetComponentInChildren<Text>().text = buttonName.ToString();
          if (i == 0)
             buttonName -= 2;
          else
             buttonName--;
       }
    }

public void ButtonAction(bool state)
    {

       if(IndexButtonClick == "PUTON")
       {
          // Action
       }

       IsClick = true;
       IndexButtonClick = null;
}
```



# HOW TO ADD A NEW ATTACHMENTS TAG?

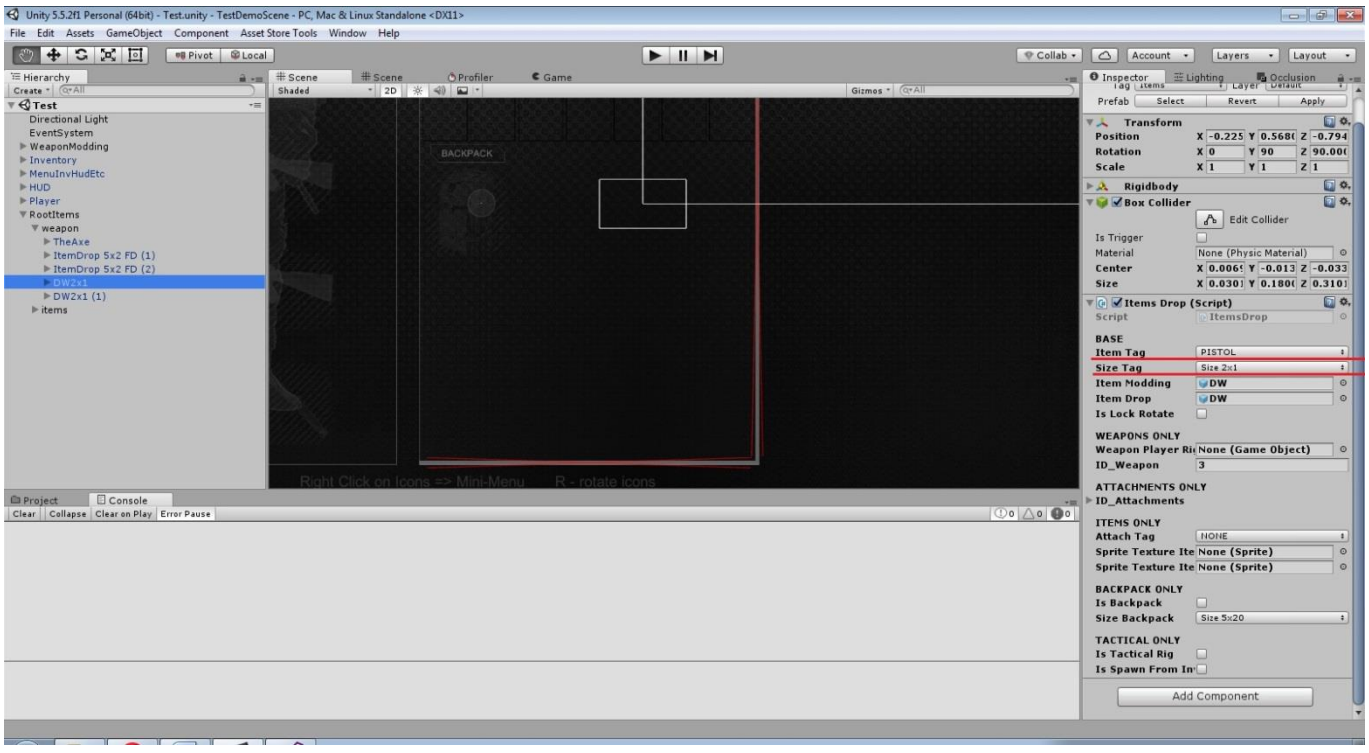
There is nothing difficult there. Everything is done in one action.

Open the script *ClassList* and add a new tag.

```
public enum TagAttchments
    {
      NONE,                              // The weapon does not have a mount
      ATTACH_BARREL_TOP,                 // Silencer, flash hider etc
      ATTACH_BOTTOM,                     //
      ATTACH_TOP,                        //
      ATTACH_TOP_PICATINNY,              //
      ATTACH_MAGAZINE,                   //
      ATTACH_BOTTOM_PICATINNY,           // Tactic grip etc
      ATTACH_BOTTOM_PICATINNY_FLASHLIGHT,//
      ATTACH_TOP_RECIVER,
      ATTACH_BOTTOM_PISTOLGRIP,
      ATTACH_TOP_HANDGUARD,
      ATTACH_BOTTOM_STOCK,
      ATTACH_BARREL,
      ATTACH_SLIDER,
    ATTACH_NEWTAG,
    }
```

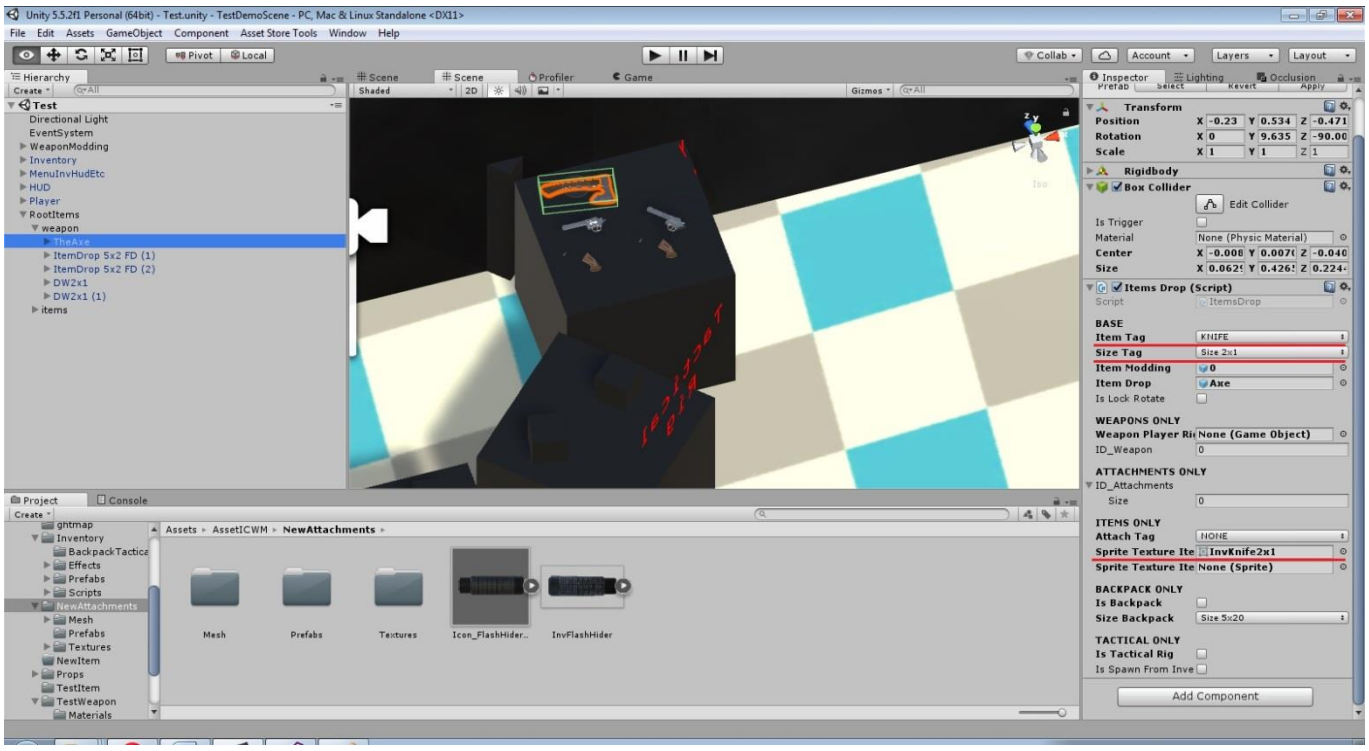
# HOW TO ADD A NEW PISTOL?

Watch the video «*How to add a new weapons*». You will need to adjust the script *ItemDrop*. Watch the restriction 1x2 и 2x1.



# HOW TO ADD A NEW KNIFE?

Watch the video «*How to add a new items*». You will need to adjust the script *ItemDrop*. Watch the restriction 1x2 и 2x1.

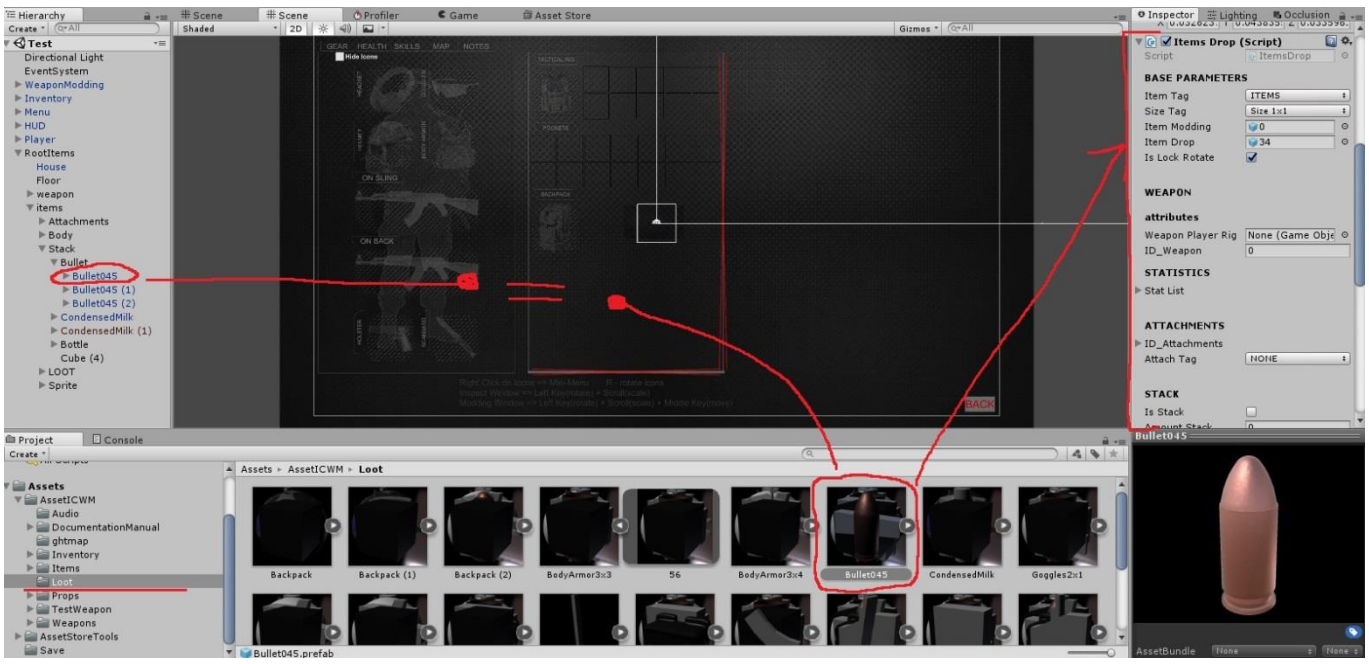


## HOW TO ADD NEW BODY ARMOR, TACTICAL RIG, BACKPACK, GOGGLES AND ETC?

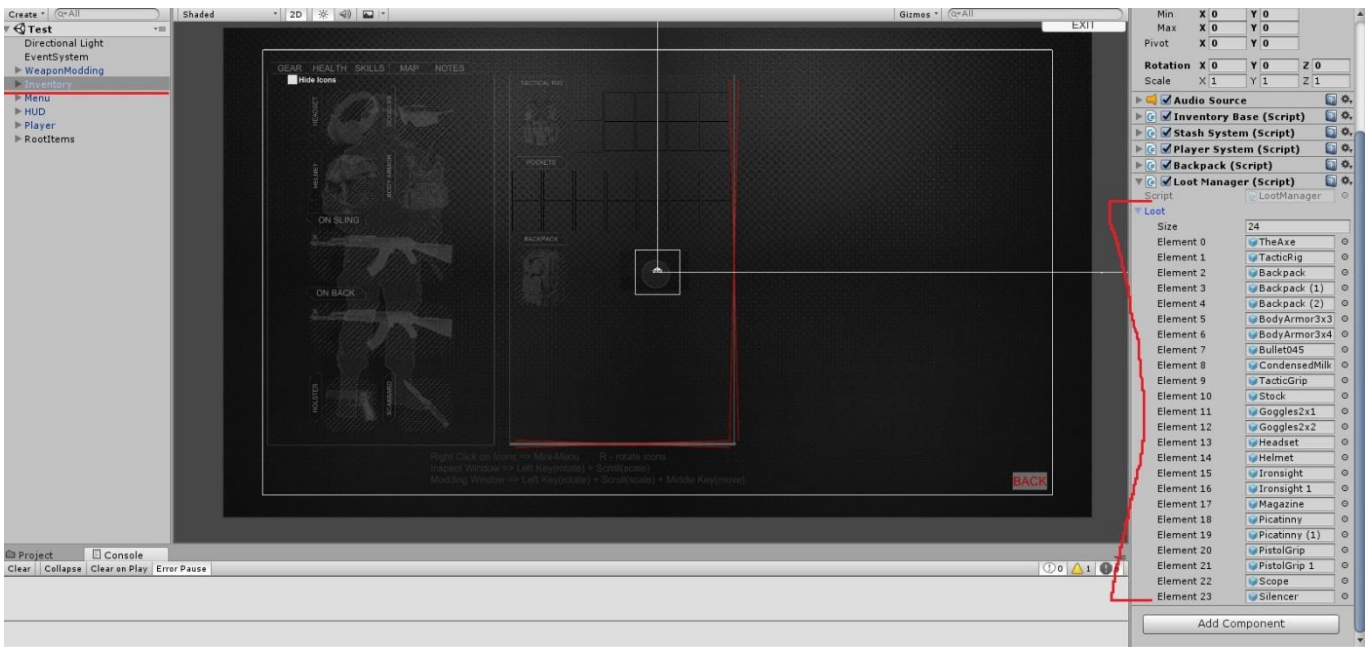
Watch the video «*How to add a new items*». You will need to adjust the script *ItemDrop*. Watch the restriction.

## HOW TO ADD ITEMS TO THE LOOT LIST

The item must be adjusted before placing it in the loot list.

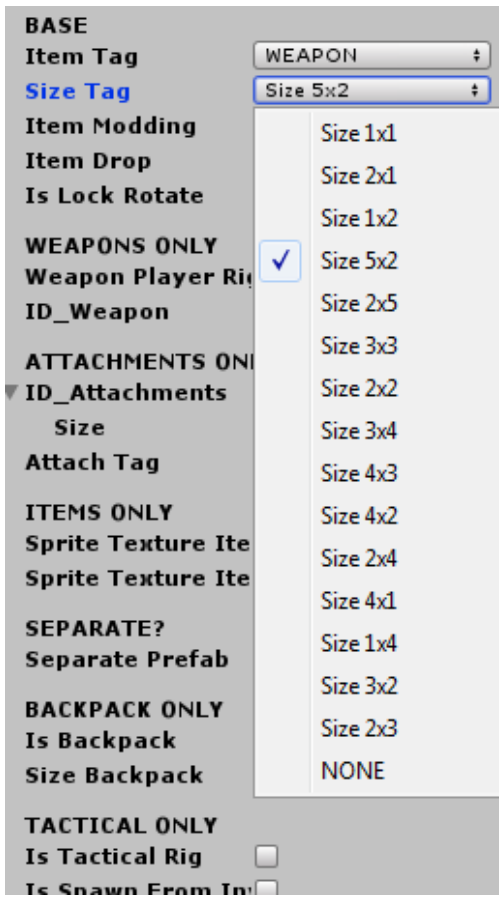



All the items that appear randomly in the loot box in the Loot array. You just need to add the item to the list.



# APPLICATION

**Icon Sizes:**




**Icons that can be rotated:**

1) 2x1
2) 1x2
3) 5x2
4) 2x5
5) 4x1 (New Update 2.0)
6) 1x4 (New Update 2.0)
7) 4x2 (New Update 2.0)
8) 2x4 (New Update 2.0)
9) 3x2 (New Update 2.0)
10) 2x3 (New Update 2.0)

**Size of the icon under the pistol:**

1) 2x1
2) 1x2

**The size of the icons for the weapons:**

1) 5x2
2) 2x5
3) 4x1 (New Update 2.0)
4) 1x4 (New Update 2.0)
5) 4x2 (New Update 2.0)
6) 2x4 (New Update 2.0)
7) 3x2 (New Update 2.0)
8) 2x3 (New Update 2.0)

**Size of the icon under the body armor:**

1) 3x3
2) 3x4

**Size of the icon for tactical rig:**

1) 3x3

**Size of the icon under the glasses or headset:**

1) 1x2
2) 2x1
3) 2x2

**Size of the icon under the helmet:**

1) 2x2

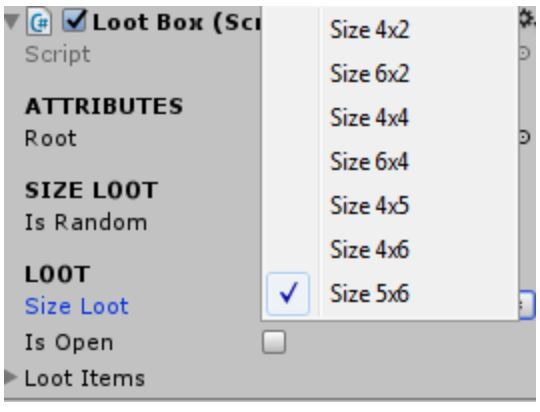
**Size of the icon for the backpack:**

1) 3x3
2) 3x4
3) 2x2 (New Update 2.0)
4) 4x3 (New Update 2.0)

**Icon size under the knife:**

1) 1x2
2) 2x1

**Size of the Loot Box** (New Update 2.0)




**Backpack sizes that cannot applied to EFT inventory:** (New Update 6.0)

1) 15x8
2) 11x7
3) 10x6