
## THANKS FOR BUYING HORROR FPS KIT!

If you like our assets, you can also visit our channel:
www.youtube.com/c/ThunderWireGamesIndie
and check out our tutorials and game developments.
Also, you can check out our website for future asset releases www.twgamesdev.com

## ABOUT HORROR FPS KIT

**HORROR FPS KIT** is an advanced and easy-to-use horror game template with many features essential to creating your own **horror** game, including gameplay features seen in **AAA** horror games of the last decade. It contains a lot of ready-to-use assets, just drag and drop into a scene.

# SUMMARY

# FEATURES

## PLAYER FUNCTIONS

- Player Controller (Walk, Run, Jump, Crouch, Prone, Ladder Climbing)
- Fully Animated Player Body (Movement)
- Texture and Tag based Footsteps
- Player Lean and Wall Detection
- Weapons (Glock, Axe)
- Zoom Effect
- Item Switcher

## SYSTEMS

- Save/Load System (Player Data, Scene Data, Inventory, Encryption)
- Cross-Platform Input (PC, PS4, XONE)
- Advanced Menu System (Cross-Platform Support)
- Inventory System (Add, Remove, Move, Use, Drop, Examine, Store, Shortcuts)
- Objectives System (Add, Complete, Complete and Add, Events)
- Cutscenes System (Queue)
- Examination System (Rotate, Double Examine, Papers)
- Drag Rigidbody System (Rotate, Zoom, Throw)
- Interact Manager (Pickup, Messages)
- Floating Icons System

## OBJECT PICKUPS

- Custom Objects Pickup
- Light Items (Flashlight, Hinge Lantern, Candle)
- Backpack Inventory Expand Pickup

## DYNAMIC OBJECTS

- Door, Drawer, Lever, Valve, Movable Interact
- Types (Mouse, Animation, Locked, Jammed)
- More Objects (Keypad Lock, Keycard Lock, Padlock)

## MORE FEATURES

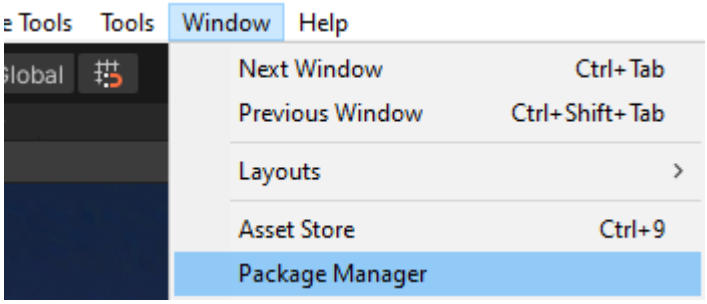
- Rebindable Keyboard Controls at Runtime
- Zombie AI (Sleep, Wander, Scream, Agony, Hunger, Attack, Attract, Footsteps)
- Scene Preloader (Background Change, Tips)
- Jumpscares (Animation, Scare Effects, Scared Breath, Customizable)
- UI Menu (Main Menu, Pause Menu, Load Menu..)
- Notifications (Pickup, Hint, Messages)
- Realistic VHS Player
- CCTV System
- Ambience Zones
- Water Buoyancy
- Interactive Lights (Lamps, Switchers, Animation)
- Props, Collectable Items, much more..

# PROJECT SETUP

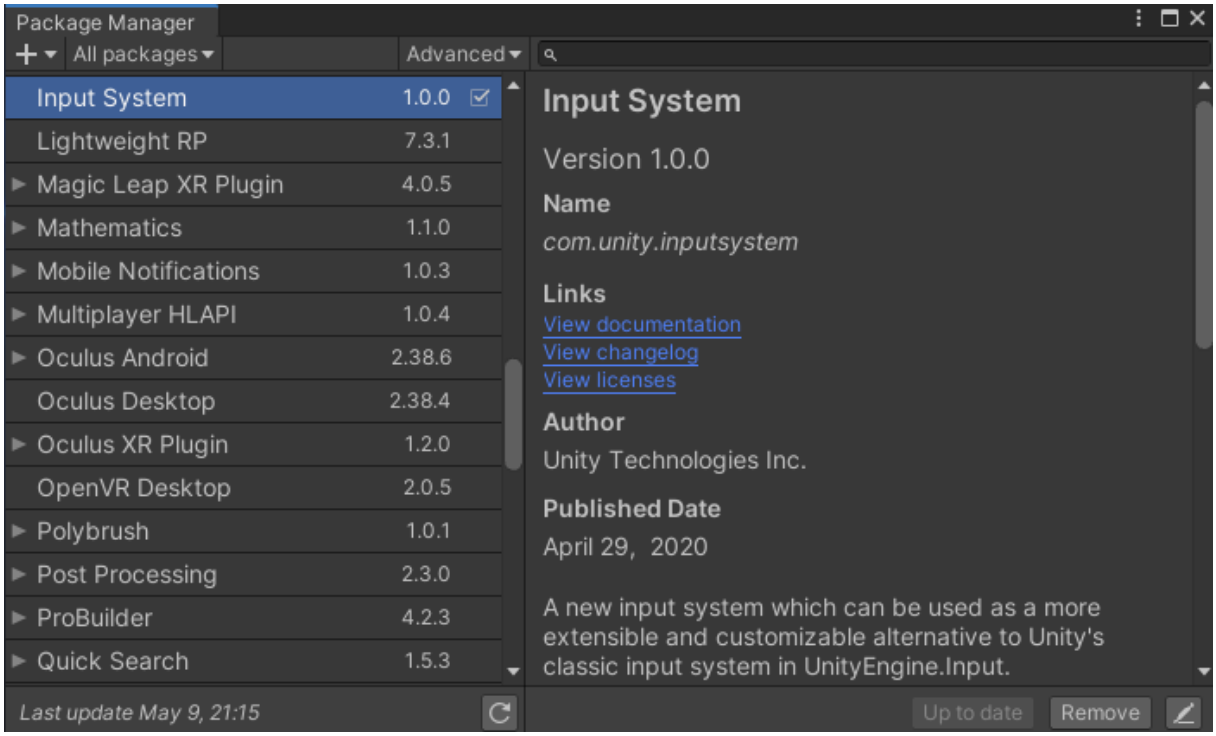
**We strongly recommend importing asset into an empty project and importing all tags and layers!**

Make sure that you have imported **Input System** and **PostProcessing** from **Package Manager!**

1. Open **Package Manager** from **Window** drop-down.




2. Find **Input System**, **PostProcessing** and Import to a Project.

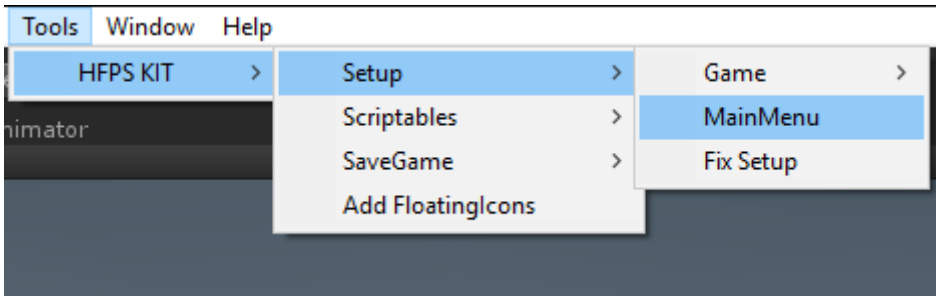



- If you haven't found an **Input System** or **PostProcessing** in **Package Manager**, select **Show preview packages** from the **Advanced** drop-down menu.

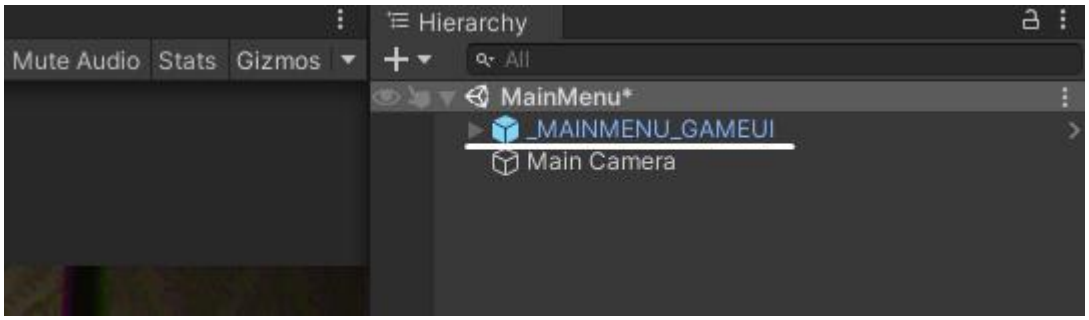
3. Go to **Player Settings** and set **Active Input Handling** to **Input System Package**.
4. Go to **Horror FPS KIT\HFPS Assets\Scriptables\InputSystem** folder and click activate in **InputSystem.inputsettings**.

## MAIN MENU SETUP

1. Open new empty scene.
2. Go to **Tools -> HFPS KIT -> Setup -> MainMenu**.



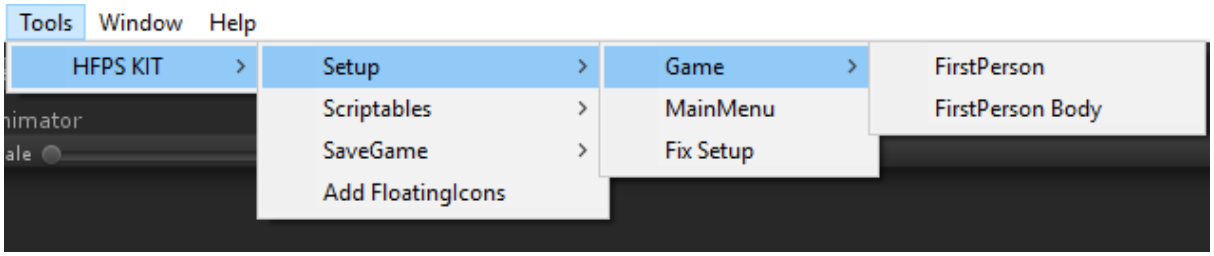

3. Now check if you have a **_MAINMENU_GAMEUI** object in your scene.




## GAME SETUP

**Steps are same as in Main Menu Setup.**

1. Open a new empty scene.
2. Go to **Tools -> HFPS KIT -> Setup -> Game**.

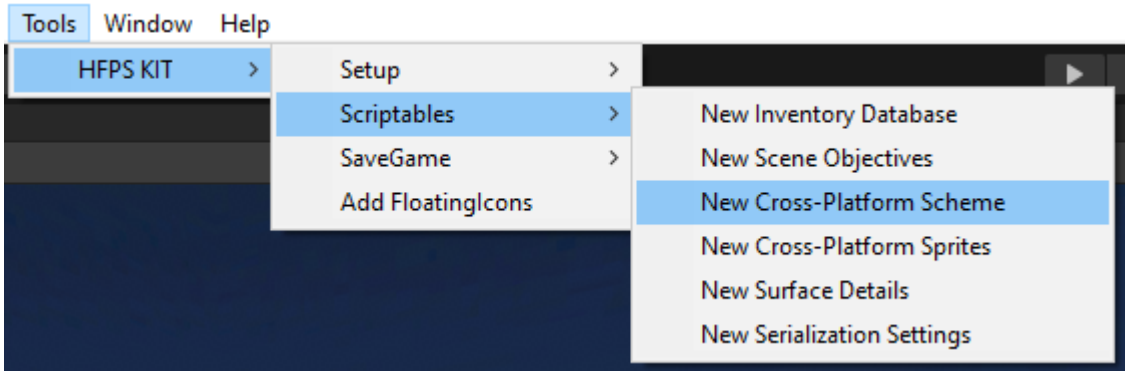



3. Check if your scene has a **PLAYER** and **_GAMEUI** object.
4. Move **PLAYER** from default position to a floor.
5. Done!

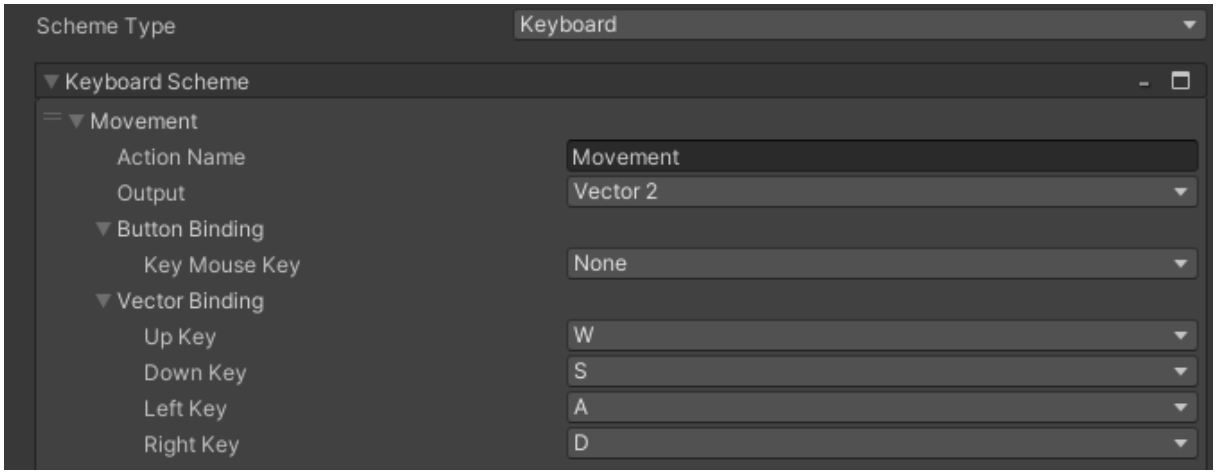
# ADDING NEW CROSS-PLATFORM INPUTS

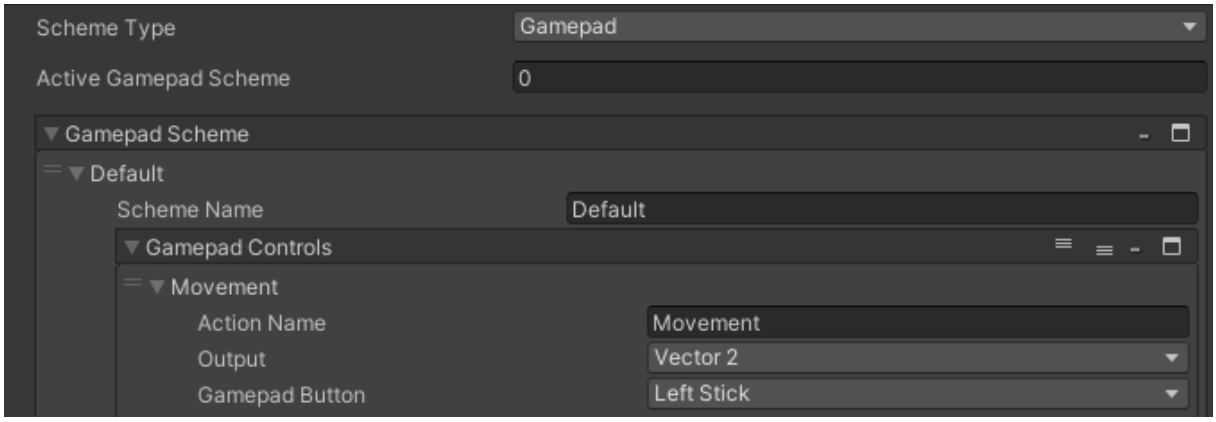
**Latest Package Manager Input System is Required!**

1. Create or open an existing **Cross-Platform Scheme**



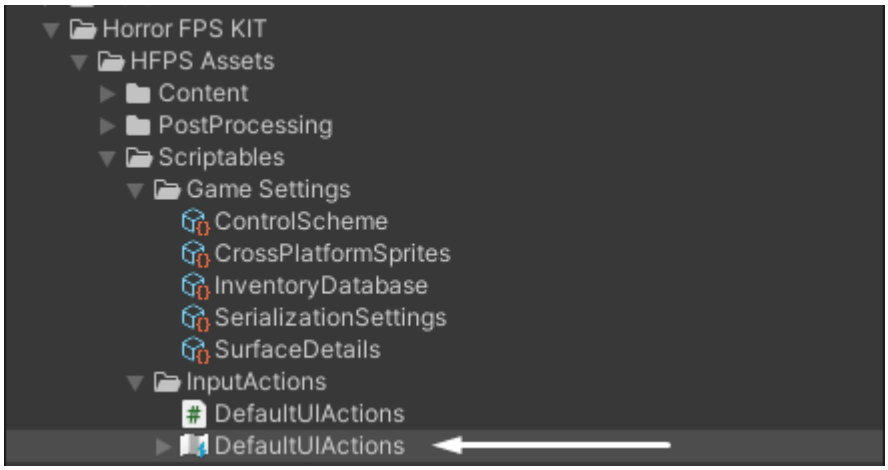

2. You can **edit** or **create** new inputs.

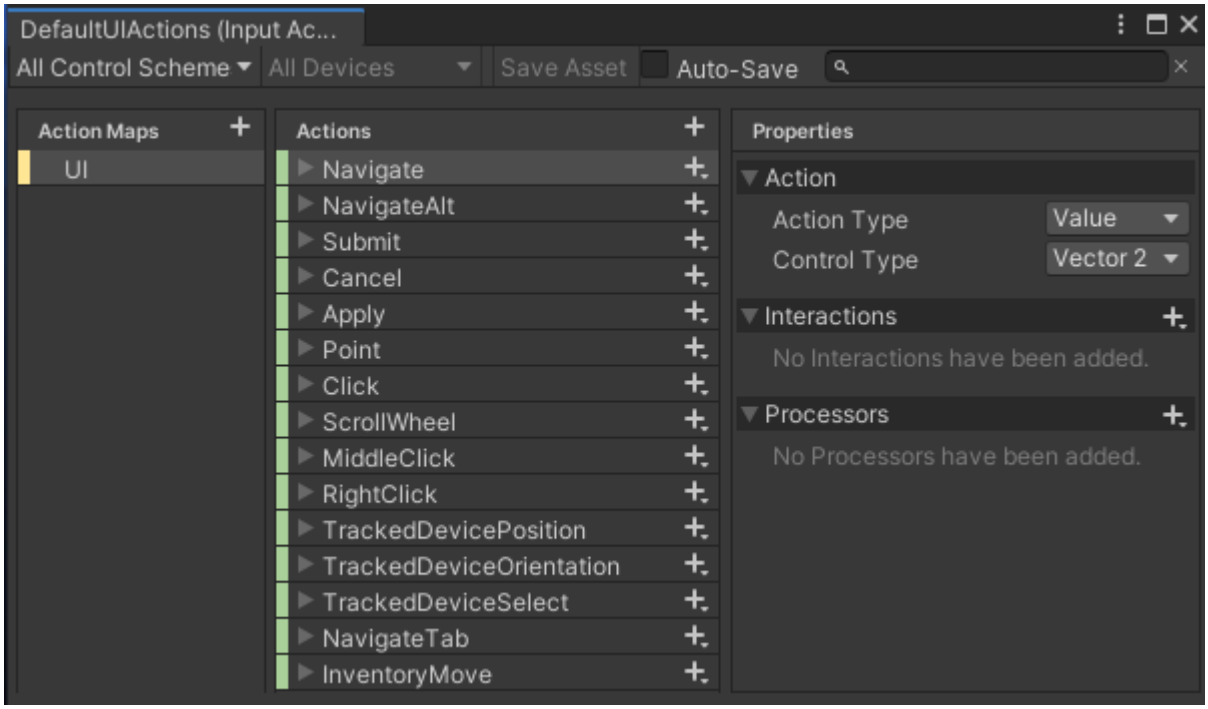


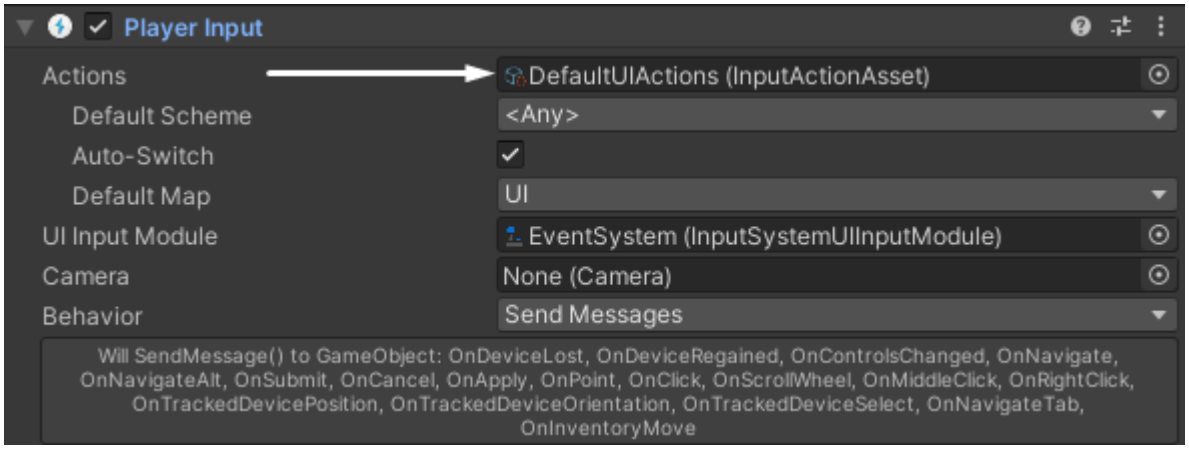




- Supported Output Types: **Bool, Vector2**
- **Output** means which type is returned when a specific input is pressed.
- **Active Gamepad Scheme** defines, which scheme will be preferred.
- The **Player's Inputs** are controlled from the **Control Scheme** and the **UI Inputs** are controlled from the **Input System -> Input Actions Scheme**.

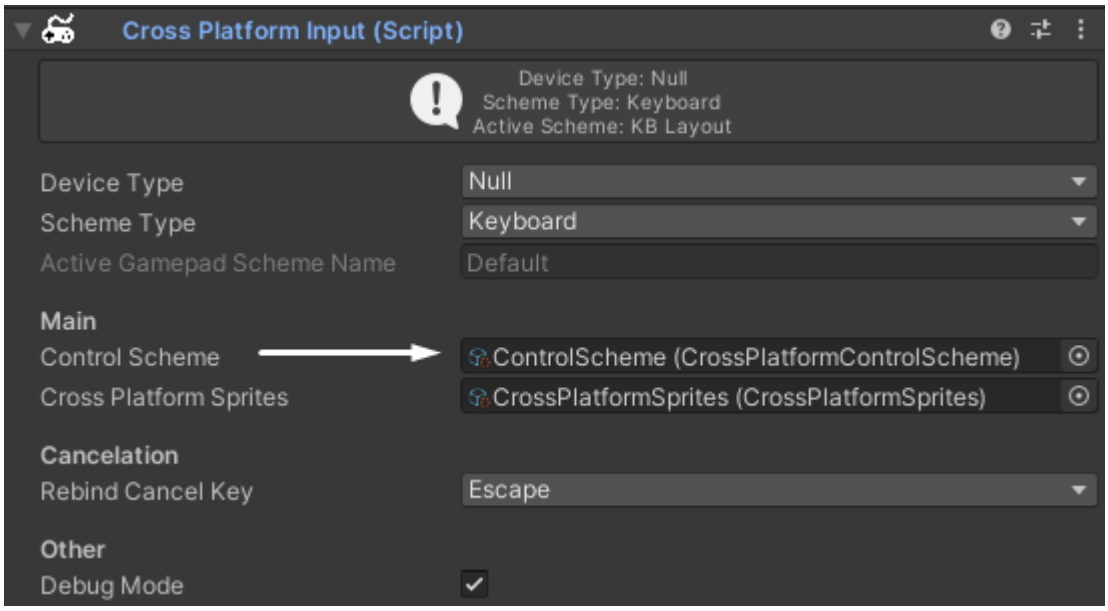
- To use **UI Inputs**, use **Input System -> Player Input** script.



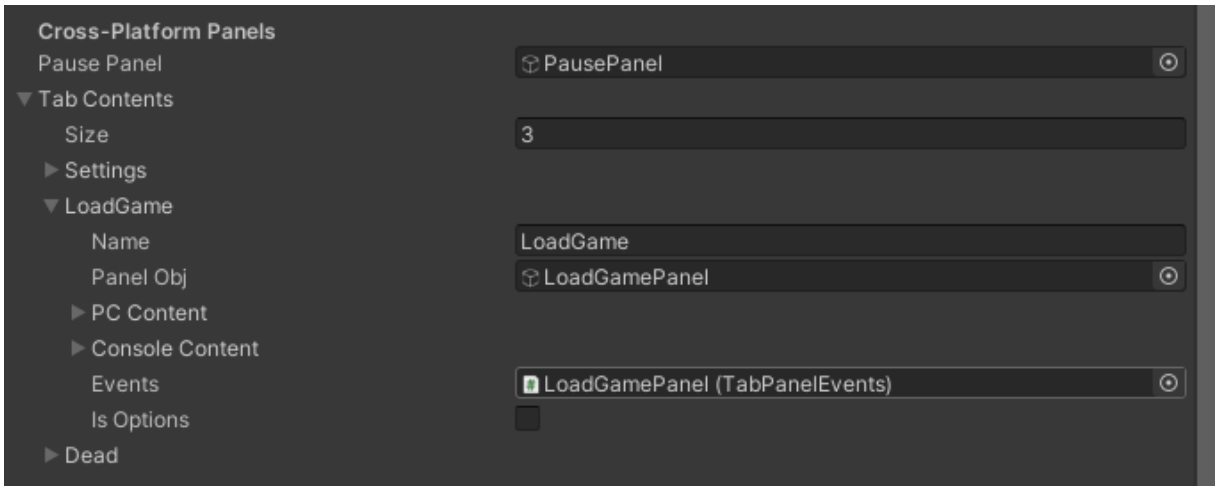




5. New created **Control Scheme** must be assigned in main **Cross-Platform Input** script which is in **_GAMEUI** object.

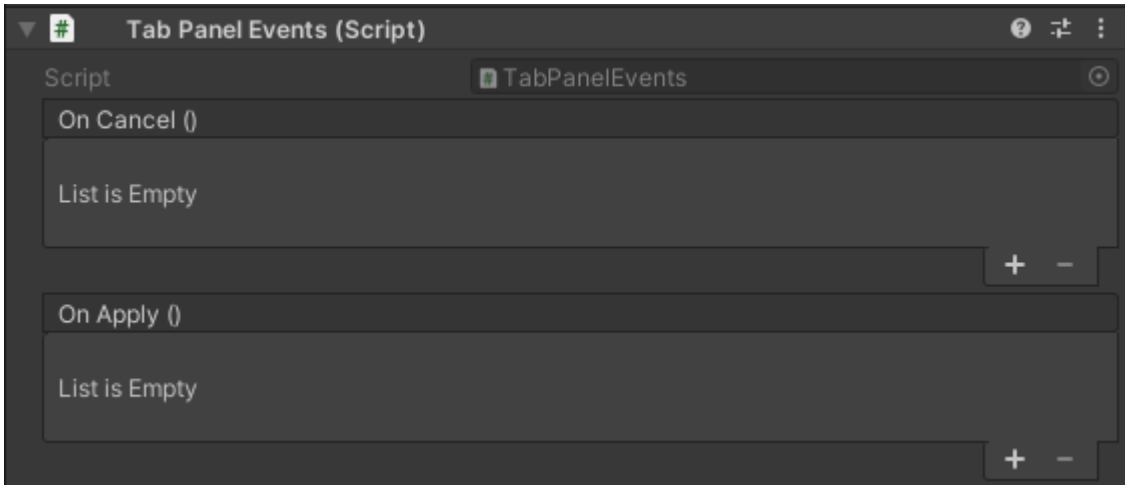



- You can also change **Cross-Platform Button Sprites** in **CrossPlatformSprites** asset.

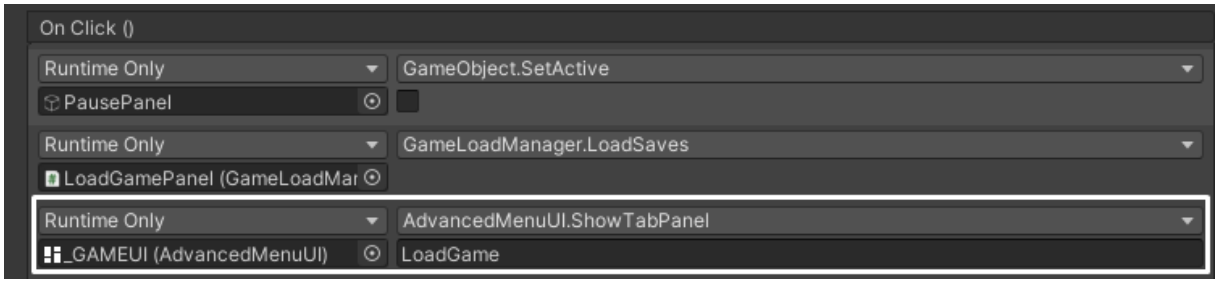
# ADVANCED MENU UI




- With **Tab Contents** you can easily define new menu panels.
- You can use **PC and Console Content** to define which objects will be enabled or disabled on different platforms.



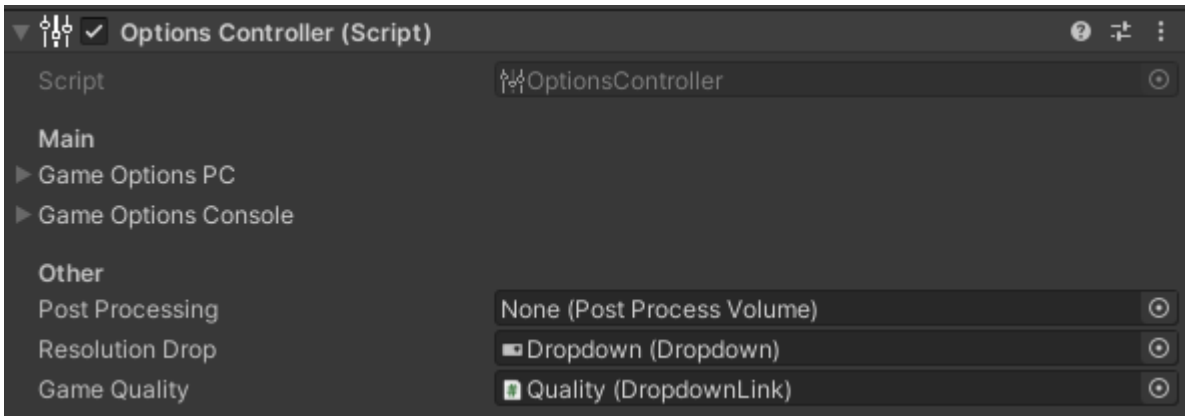

- **TabPanelEvents** defines what happens when you press **Cancel** or **Apply** button.

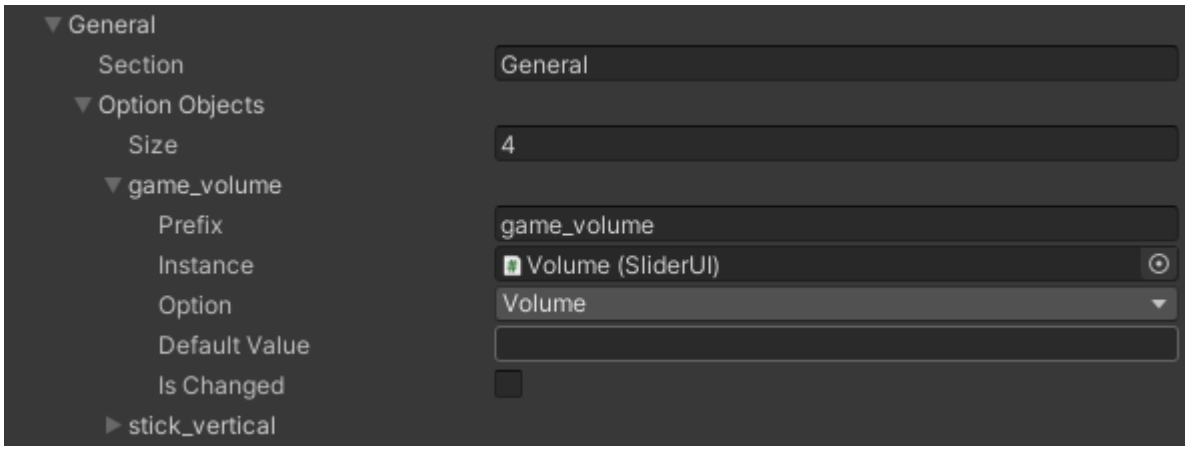



- To display the panel, call the **ShowTabPanel()** function with the **Name** parameter.
- To deselect the panel, call the **DeselectPanel()** function.

# OPTIONS CONTROLLER

- With Option Controller, you can define **Option Settings** on different platforms.




- You can use **Default Option Types,** or you can create **Custom**.
- To create **Custom Option**, add **IJsonListener** interface to your script.
- When your **Custom Option** is changed, the **OnJsonChanged** function is called.

```
public void OnJsonChanged()
{
    if (jsonHandler)
    {
        JObject root = jsonHandler.Json();

        if (root[OptionsController.OPTIONS_PC_PREFIX] != null)
        {
            if (root[OptionsController.OPTIONS_PC_PREFIX]["general"][mousePrefix] != null)
            {
                float sensitivity = (float)root[OptionsController.OPTIONS_PC_PREFIX]["general"][mousePrefix];
```

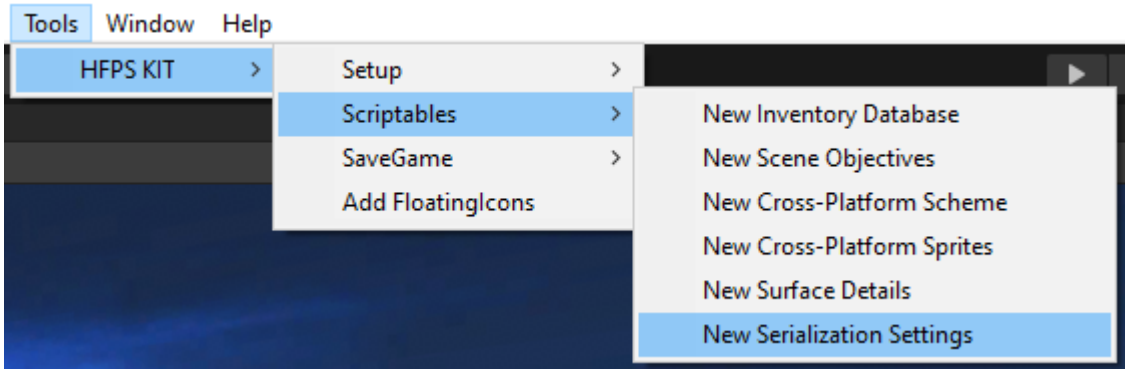
- With that method you can easily create handler for your **Custom Option**.
- As example you can open **MouseLook.cs** script.

# SAVE/LOAD MANAGER

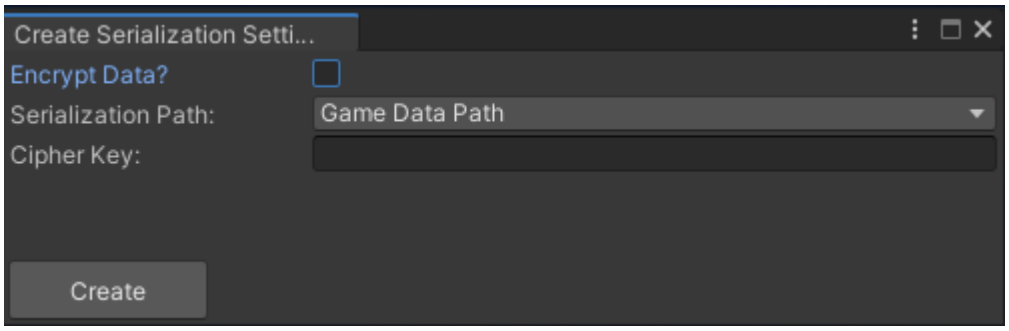
Saving and Loading game data is a major feature of many AAA game titles. With **SaveGameHandler.cs** you can save game data as well.

## SETTING UP

1. Create new **Serialization Settings** asset.

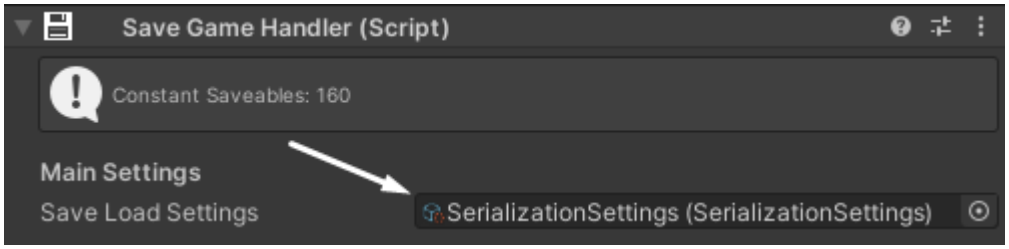



- This will show up a **Serialization Settings** window.




- **Cipher Key** can be a random word or text. It is used to **Save** and **Load** encrypted **Game Data**.

2. Assign your new **Serialization Settings** into **SaveGameHandler** script.



## ADDING CUSTOM SAVEABLES

There are 3 methods, how you can define new **Saveables**.

1. Adding **Isaveable** interface.

```
public class ExampleSaveable : MonoBehaviour, ISaveable
{

}
```

- After adding the **Isaveable** interface, you need to create the required functions.
- It can be done by pressing (**ALT + Enter or Ctrl + .**) in **Visual Studio**.

```
public class ExampleSaveable : MonoBehaviour, ISaveable
{
    public void OnLoad(JToken token)
    {
        throw new System.NotImplementedException();
    }

    public Dictionary<string, object> OnSave()
    {
        throw new System.NotImplementedException();
    }
}
```

- This will create two new functions, **OnLoad** and **OnSave**.
- These functions are the main functions for **Saving** and **Loading** game data.

```
public class ExampleSaveable : MonoBehaviour, ISaveable
{
    private bool exampleBool = true;
    private float exampleFloat = 10.5f;

    public void OnLoad(JToken token)
    {
        exampleBool = token["exampleBool"].ToObject<bool>();
        exampleFloat = token["exampleFloat"].ToObject<float>();
    }

    public Dictionary<string, object> OnSave()
    {
        return new Dictionary<string, object>()
        {
            { "exampleBool", exampleBool },
            { "exampleFloat", exampleFloat }
        };
    }
}
```

- The **Token Path** must be the same as the **Key**, you use to store the data.
- **Dictionary Values** can be **Nested**.

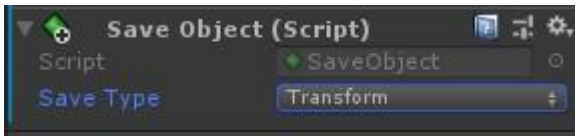
2. Adding **SaveableField** attribute to the field that you want to save.

```
public class ExampleSaveable : MonoBehaviour
{
    [SaveableField, HideInInspector]
    public bool exampleBool = true;

    [SaveableField, HideInInspector]
    public float exampleFloat = 10.5f;
}
```

- You can also set your own field key.

```
[SaveableField("theBoolean"), HideInInspector]
public bool exampleBool = true;

[SaveableField("theFloat"), HideInInspector]
public float exampleFloat = 10.5f;
```

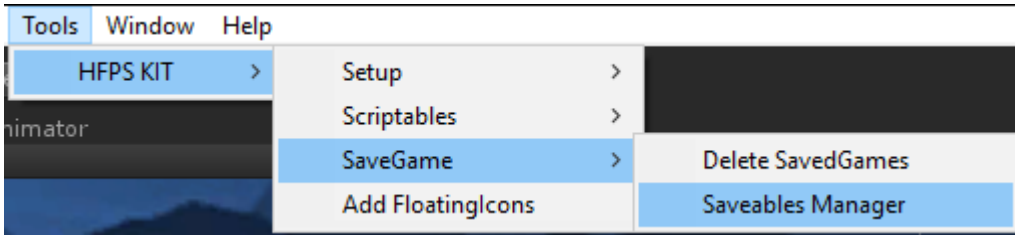
3. Adding **SaveObject.cs** script to **GameObject**.



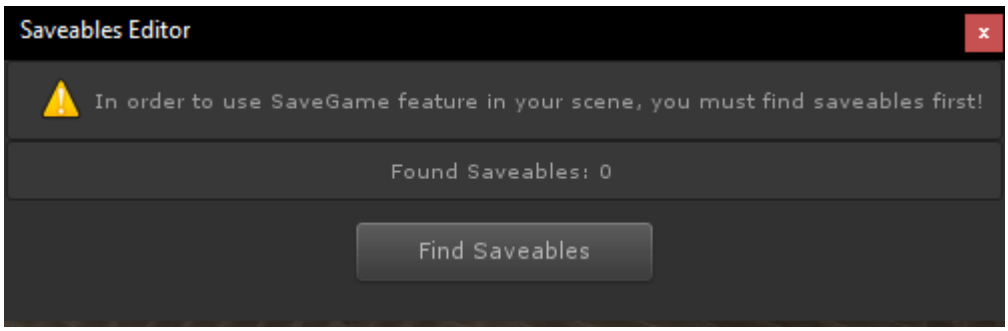

- This script allows you to define what you want to save from the object.
- **Supported Types:** Transform, Rigidbody, Position, Rotation and Object/Renderer Active.

## SAVING GAME DATA

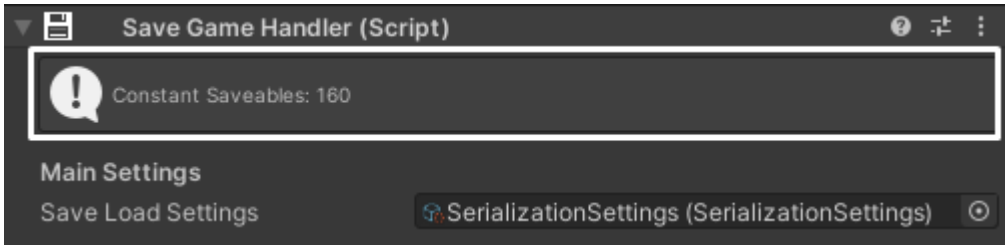
1. Select **Saveables Manager** from the **Tools** menu.



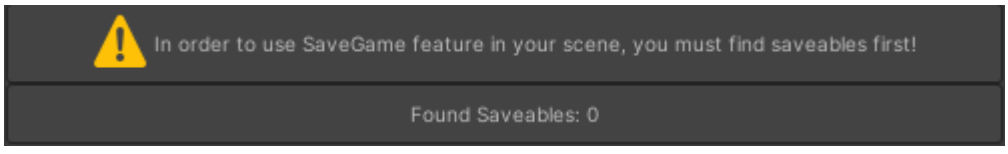

- This will show up a **Saveables Editor** window.
- Editor will not show up, when **SaveGameHandler** script will not be found.



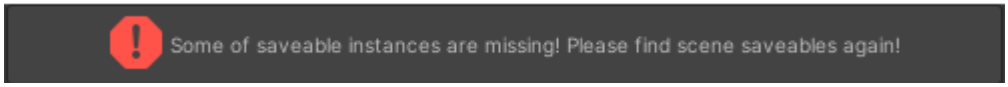

2. By clicking **Find Savablebles**, the editor automatically finds all objects that can be saved.




## SAVEABLE SCRIPT MESSAGES




- You did not click the **Find Saveables** button or the scene has no **Saveables**.




- The referenced **Saveable** has been removed from the scene.
- You must click **Find Savablebles** again to clear this message.

# ADDING RUNTIME SAVEABLES

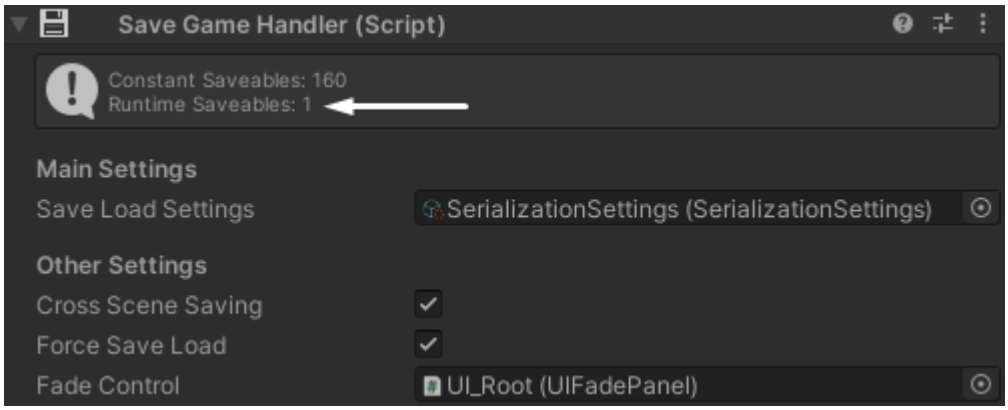
- **This feature is still WIP, it will change in later updates.**

1. Use the **InstantiateSaveable()** function from **SaveGameHander** script.

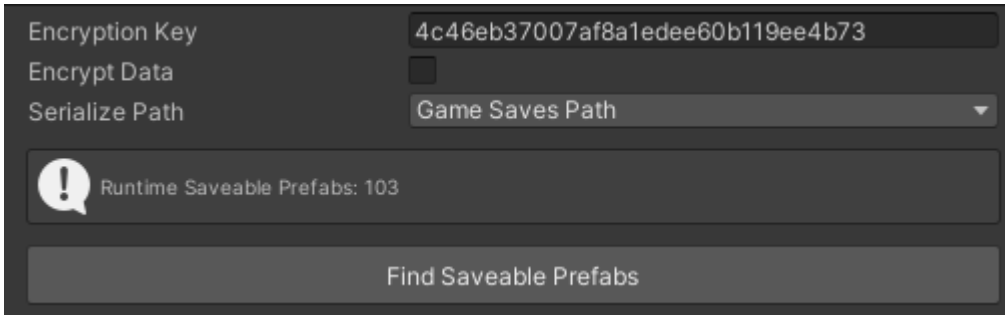
```
void Example()
{
    SaveGameHandler.Instance.InstantiateSaveable(Prefab, transform.position, transform.eulerAngles);
}
```

- This feature allows you to **Instantiate** the **Prefab** and allows you to **Save** and **Load** it's **Savablebles** at **Runtime**.

2. If you successfully **Instantiate** the **Saveable**, it will appear in the **SaveGameHandler** script.




3. To find the **Runtime Saveable Prefabs**, you must go to **Serialization Settings** asset, and click **Find Saveable Prefabs**.

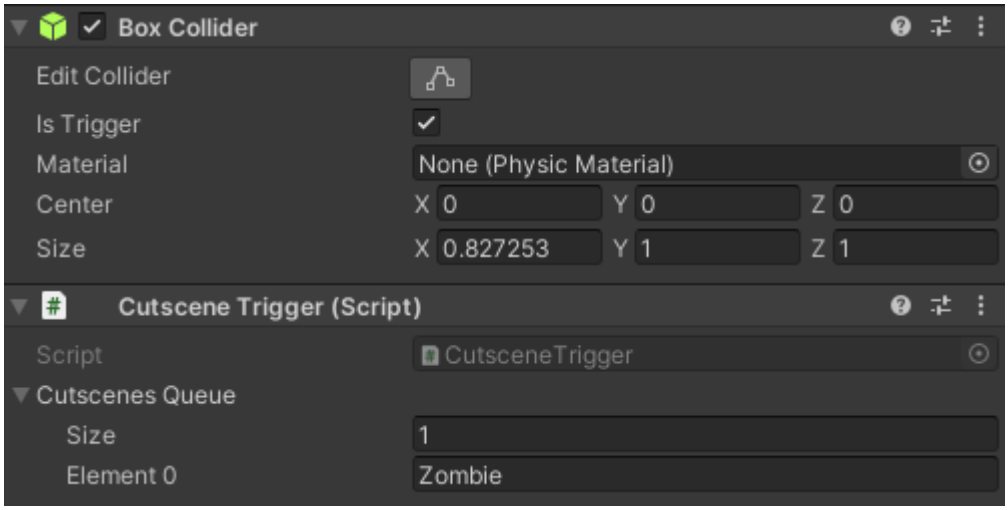


# ADDING NEW CUTSCENES

1. Create a new **Timeline Animation**. You can follow the tutorial from **Brackeys**.

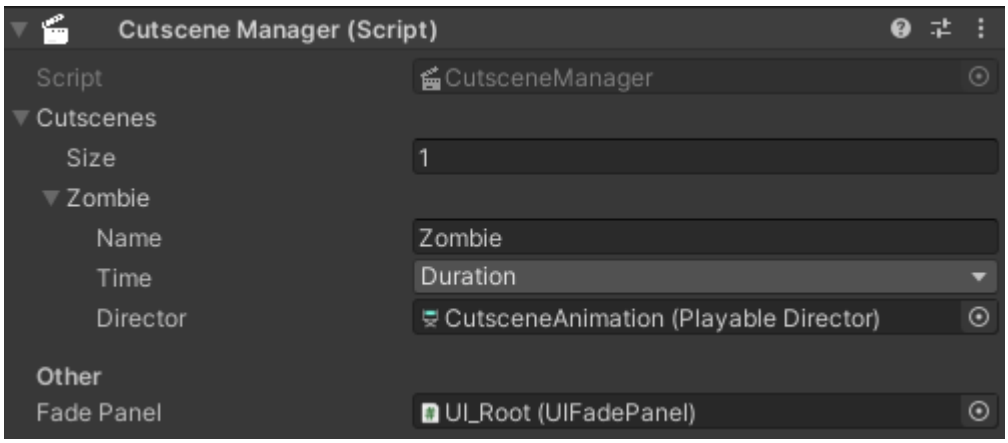
   https://www.youtube.com/watch?v=G_uBFM3YUF4

2. Create a **Cutscene Trigger**.




- You can **queue** the cutscenes by adding more **Elements**.

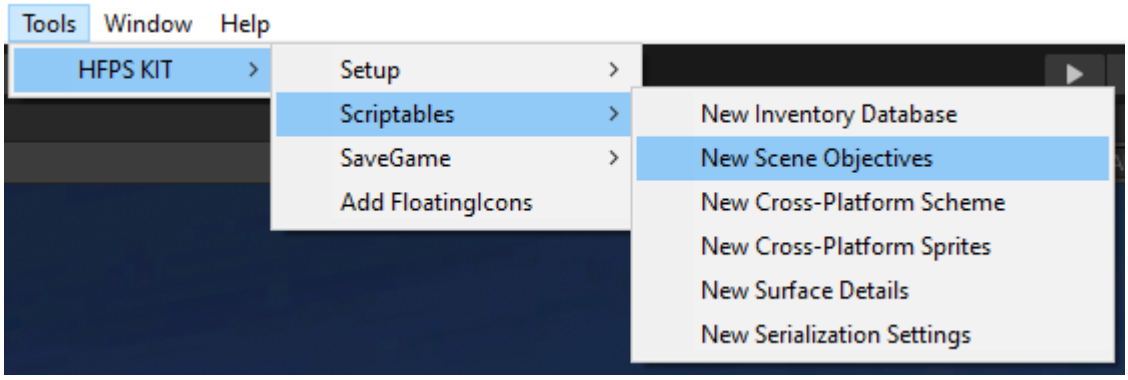
3. Assign your new **Cutscenes** to the **Cutscene Manager** script.



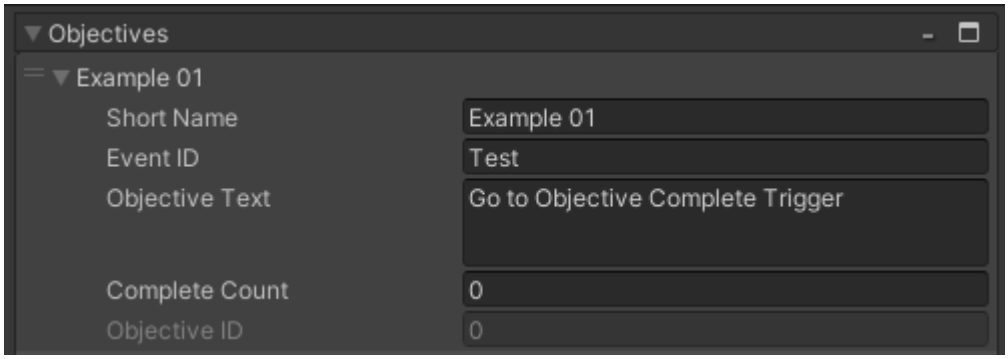

- If you set the **Time** parameter to **Manual**, you must manually skip or cancel the cutscene sequence by invoking the **Cutscene Manager -> SkipCutscene()** or **AbortCutscenes()**.

# ADDING NEW OBJECTIVES

1. Create or open an existing **Scene Objectives**




2. Add new **Objectives** to the **Scene Objectives** asset.

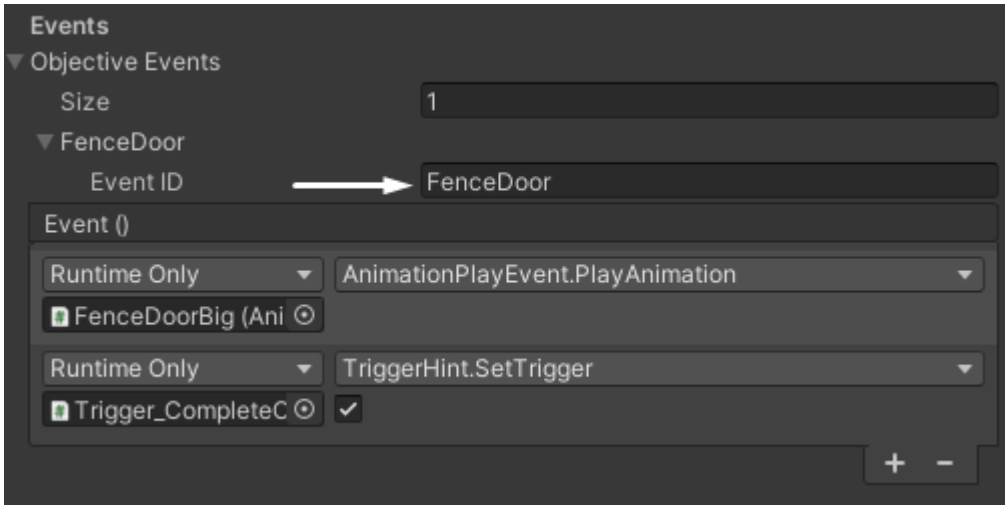



- Each objective has a unique ID that allows you to easily define which objective you want to run. The script also automatically sets the correct order of objectives.

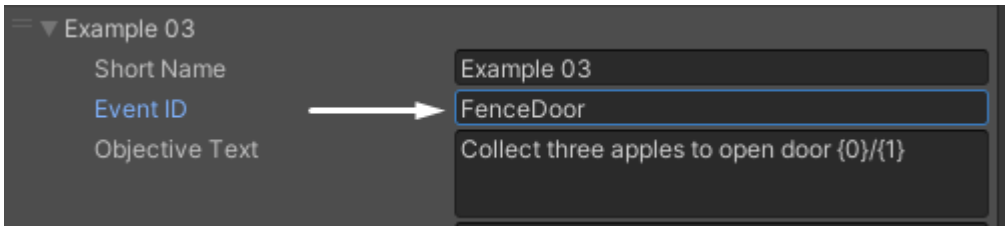
3. Remember to always assign objectives of your scene in the **Objectives Manager** script.


4. Create a new trigger, add **TriggerObjective.cs** script and set ID to objective which you want to run.

5. If you want, you can also set up objective events by adding **Objective Events** in the **Objectives Manager** script.




- Don’t forget to assign **Objective Events** and **Scene Objectives Event ID** to the same value.

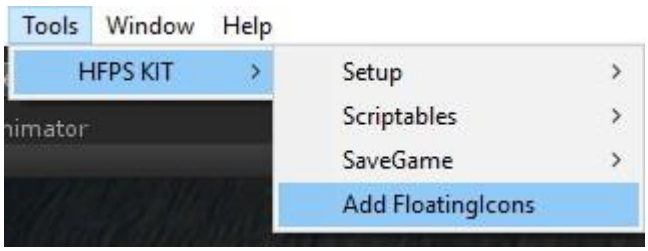



**Complete Count:** The objective will be fully completed, if you complete the objective n-times.

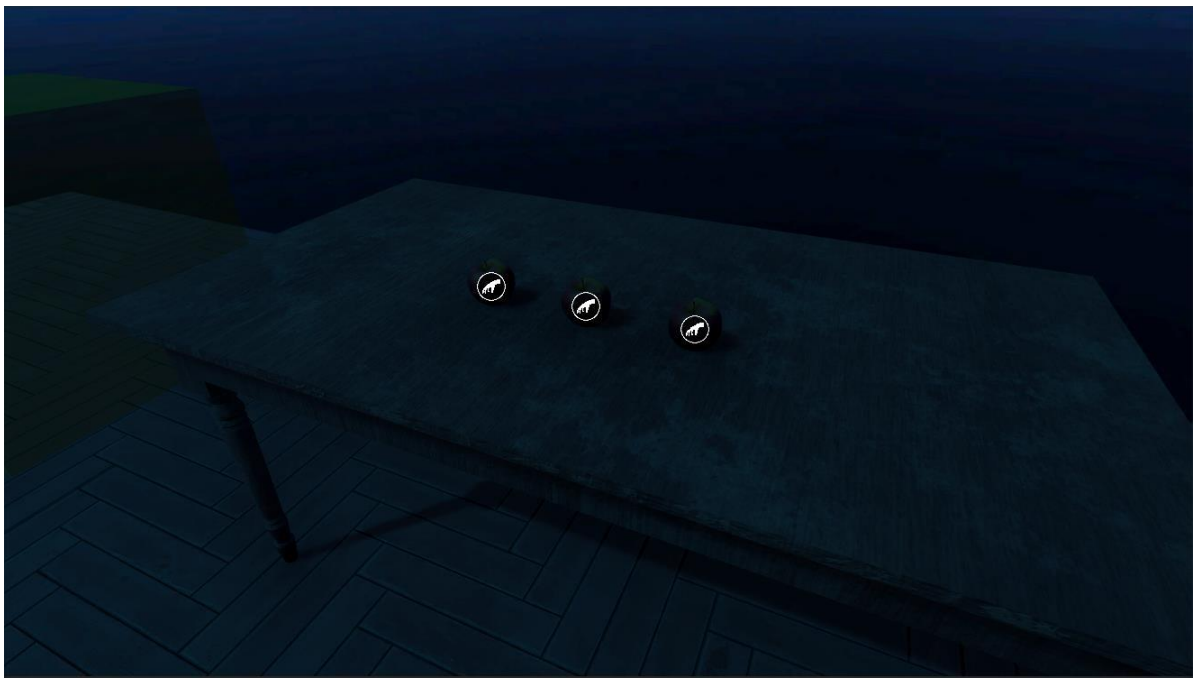
To show the progress of the objective, include **“{0}/{1}”** in the objective text.

# ADDING NEW FLOATING OBJECTS

1. Select **Add FloatingIcons** option from Tools menu.



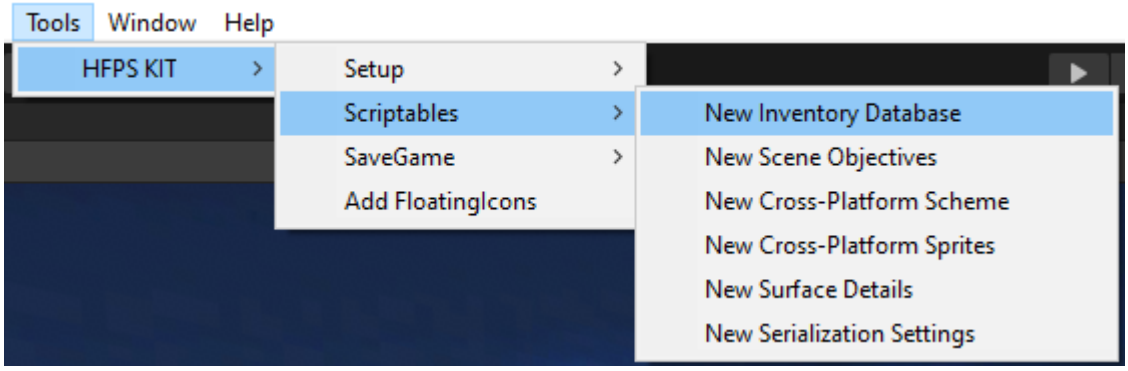

2. This will automatically add all selected objects to the **Floating Icons** list.

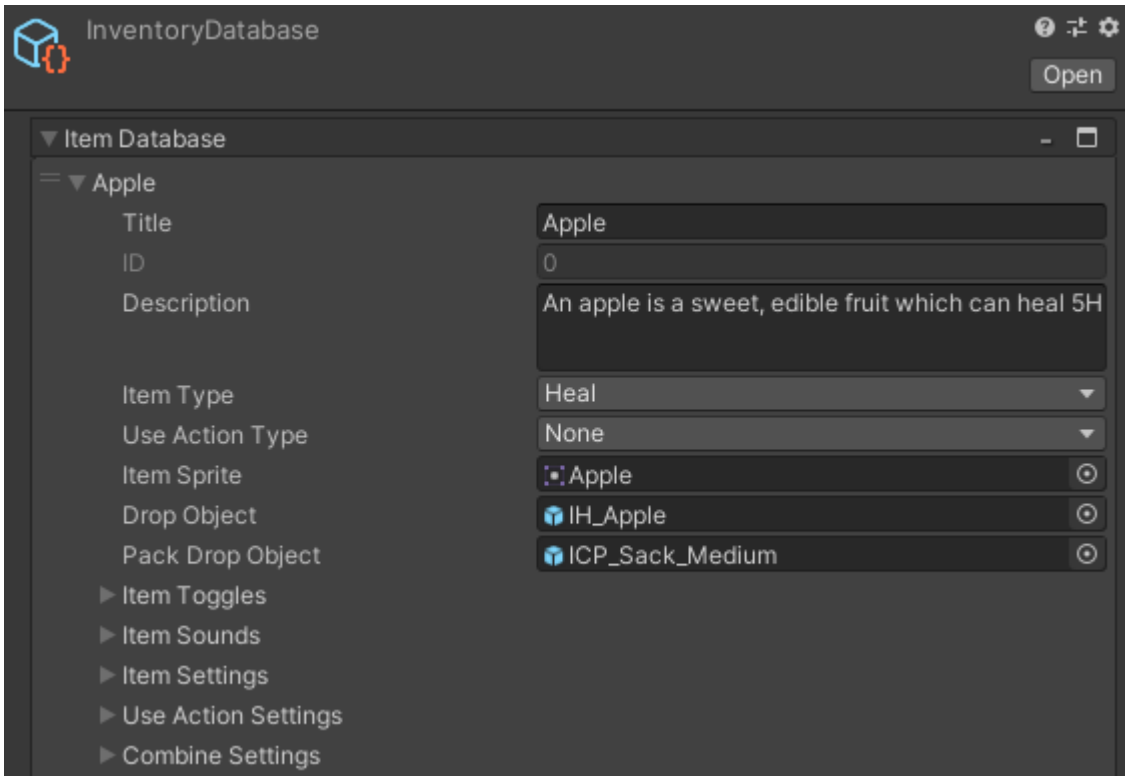
# INVENTORY

## ADDING NEW INVENTORY ITEMS

1. Create or open an existing **Inventory Database** asset.

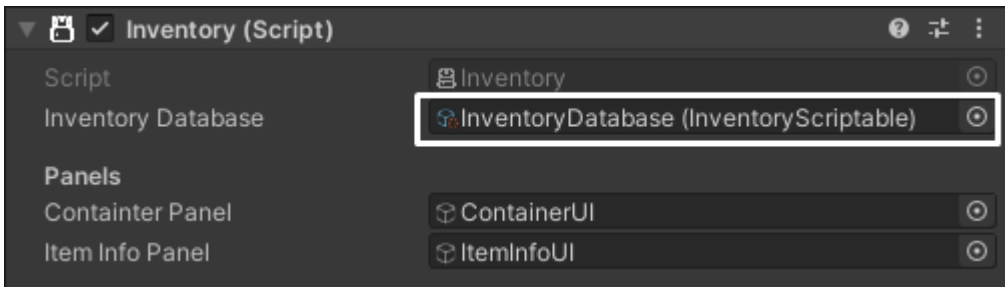



2. Add your new **Item** to the **Inventory Database**.

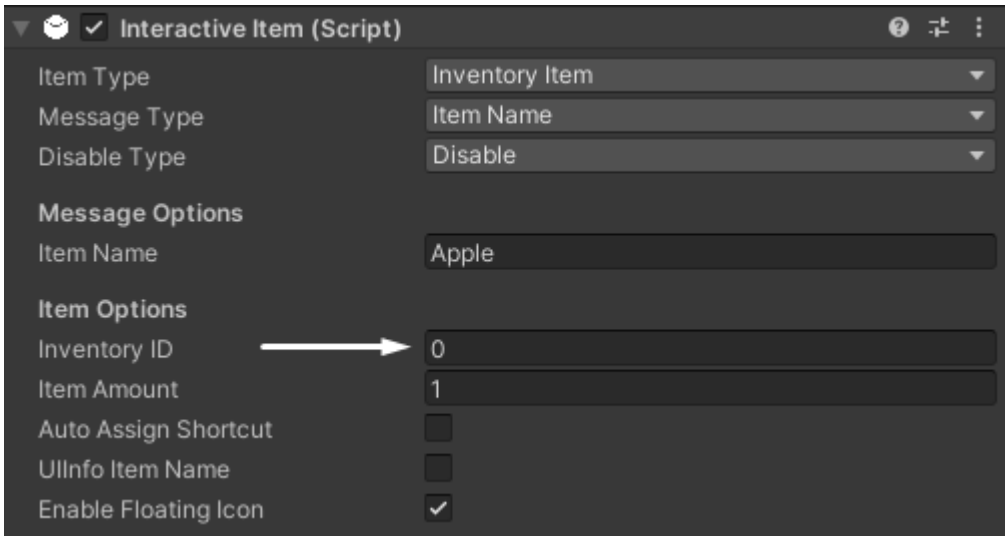



- In the item's drop-down menu, you can change various settings that will change what the item will represent in the game.

3. New created **Inventory Database** must be assigned in main **Inventory** script which is in **_GAMEUI** object.




4. To interact with a newly created item, add the **InteractiveItem.cs** script to the object and set the **Inventory ID** to the **ID** that is in the **Inventory Database** in the newly created item drop-down menu.




- The **Item ID** is read-only and is defined automatically by the database script.
- If you change the order of any item, all item **IDs** will be automatically updated with the current item order.

## INVENTORY CONTAINERS

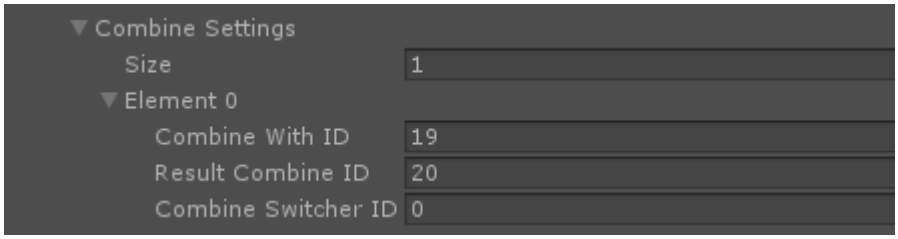
- All items from **Inventory Database** can be stored inside **Inventory Containers**.


- To store an **Item** to a **Container**, simply add the **InventoryContainer.cs** script to it.

- You can also define **Fixed Containers** that can contain the same items as another container by adding the **InventoryFixedContainer.cs** script.

## COMBINABLE ITEMS

- Inventory has a feature that allows to combine two different objects to get another useful item.
- This can be done by changing the **Combine Settings** in the Item drop-down menu.

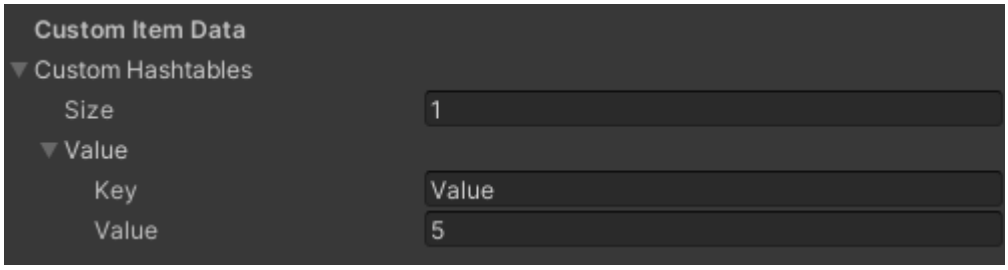




## INVENTORY CUSTOM ACTIONS

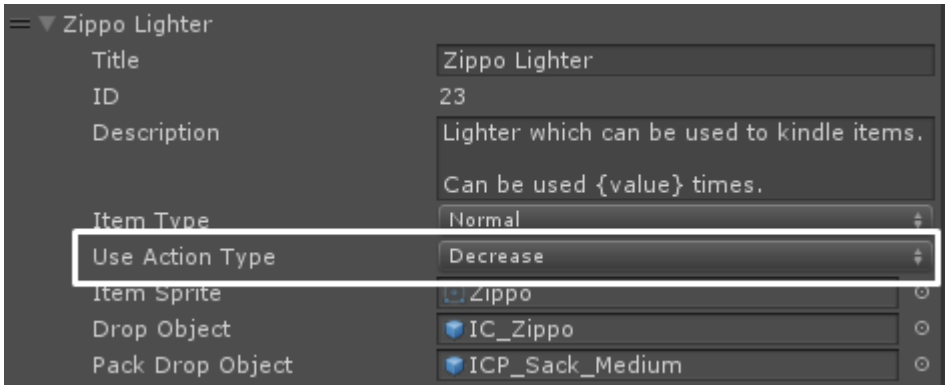
- To activate **Custom Actions,** you must enable **Do Action Use** or **Do Action Combine** inside **Item Toggles** drop-down.

1. Define your **Custom Item Data** inside **Interactive Item** script.

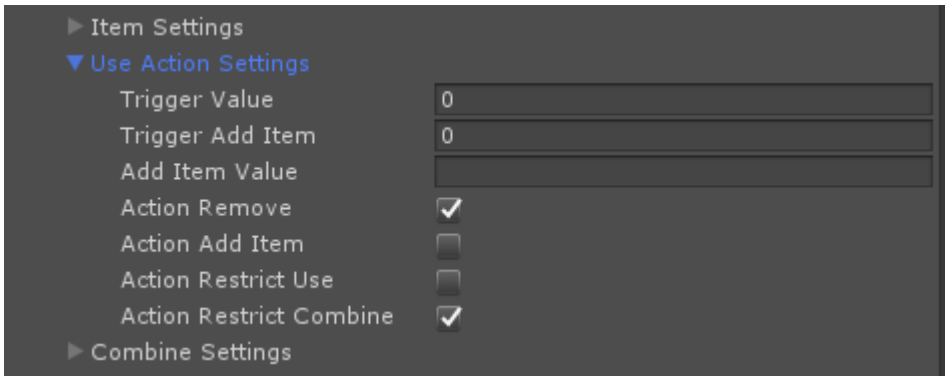



- **Predefined Keys**: Value, Path, Tag
- Use the **Value** key to define the item value in the description.

2. Define **Use Action Type** inside the Item.



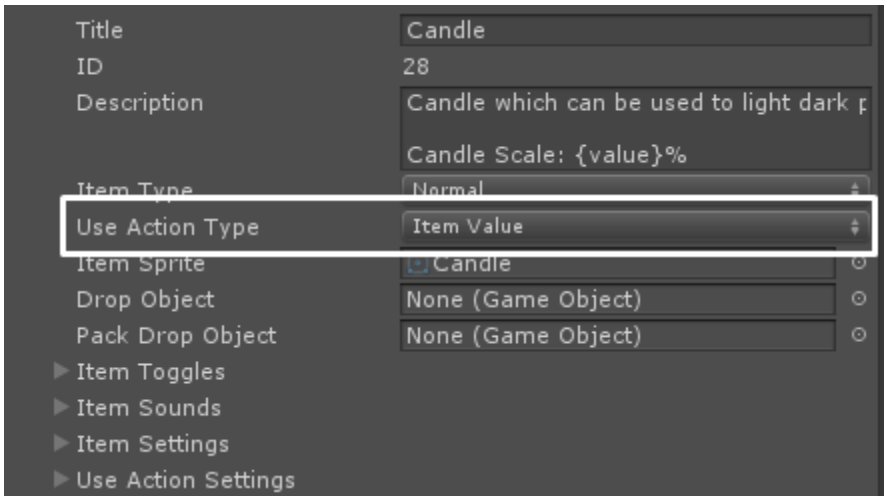

3. Set **Use Action Settings**




- To display the current value in the item description, you must add a **{value}** tag to the description text, as shown in the screenshot.
- If you use Zippo Lighter Item 5 times, you will not be able to combine it again.

- To use **Item Value** action type, you must open the script which controls your item and add **IitemValueProvider** interface.

1. Set **Use Action Type** to **Item Value**




2. Add a hashtable with the **"Value"** Key.


3. Add **IitemValueProvider** interface and its functions.

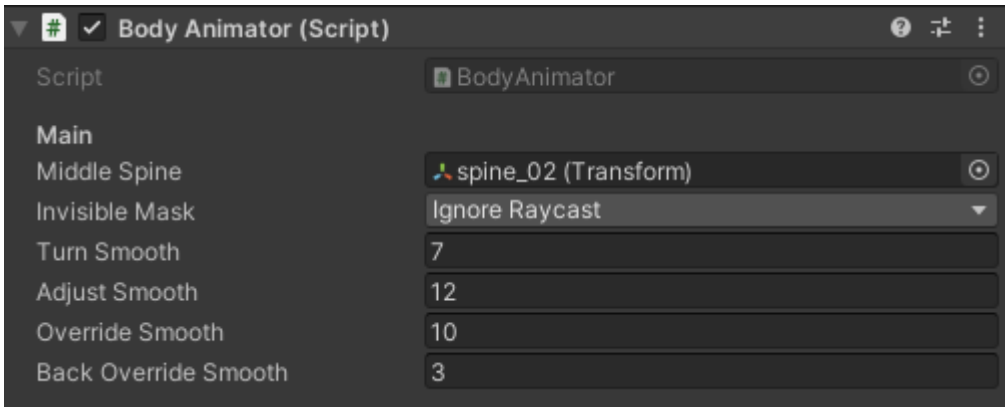
```
public class CandleItem : MonoBehaviour, ISwitcher, ISaveableArmsItem, IItemValueProvider {
```

```
public string OnGetValue()
{
    throw new System.NotImplementedException();
}

public void OnSetValue(string value)
{
    throw new System.NotImplementedException();
}
```

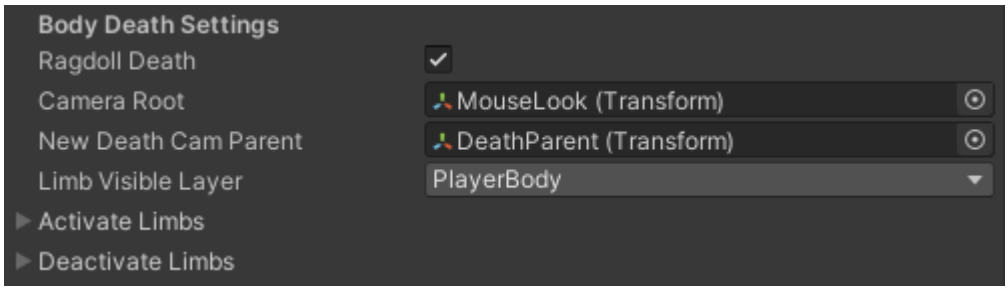
**Increase:** The hashtable **Value** parameter is incremented by 1.
**Decrease:** The hashtable **Value** parameter is decremented by 1.
**Item Value:** The hashtable **Value** parameter changes according to the **IitemValueProvider** interface.

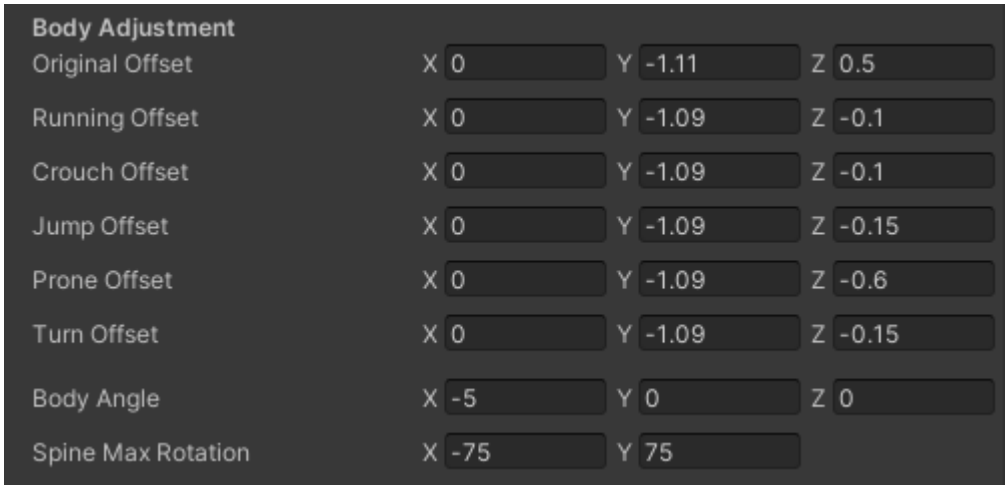
# PLAYER BODY (Body Animator)

- HFPS scripts are not linked to **Body Animator**, so if you don't want to have a first-person body, you can simply delete the **HeroBody** object.




- The **Middle Spine** field controls the rotation of the character's spine as you walk sideways, so it needs to be defined correctly.

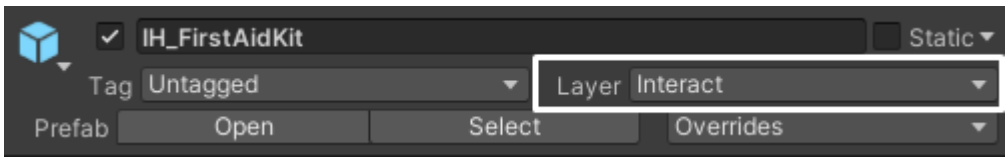


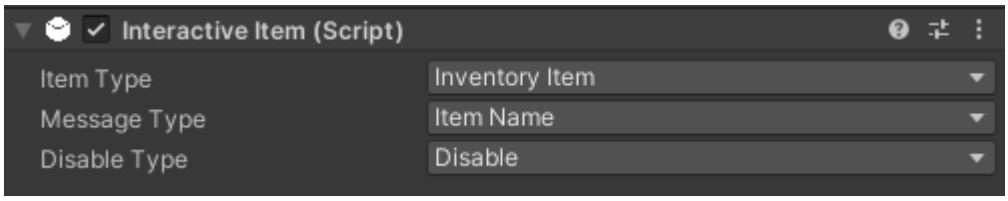

- You can adjust **Body Death**, **Body Adjustments** and **Other** settings depending to your needs.

# ADDING NEW INTERACTIVE ITEMS

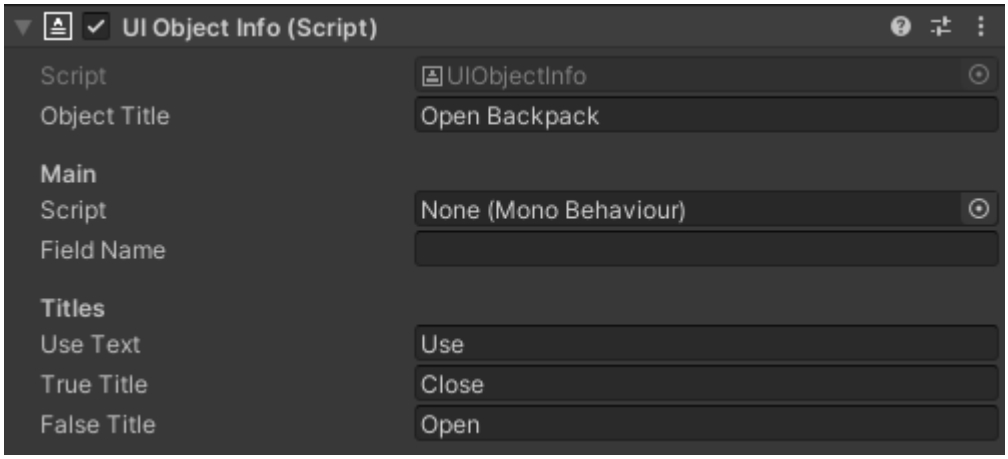
1. Set the object layer to **Interact**.




2. Add the **InteractiveItem.cs** script to the object.



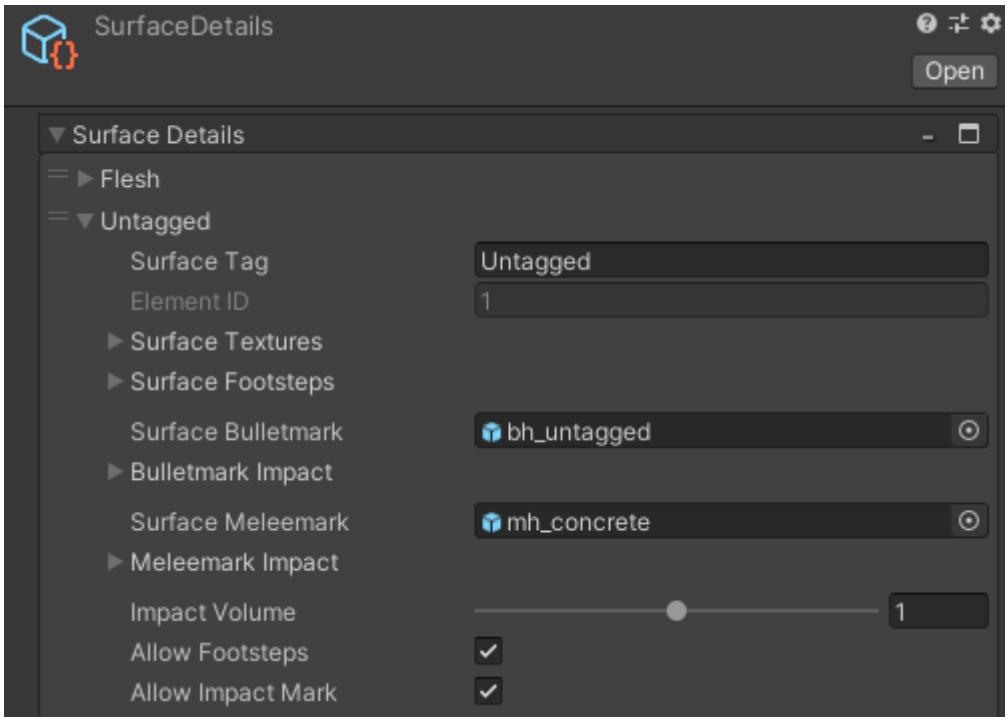

3. To change the user interface texts, add the **UIObjectInfo.cs** script.



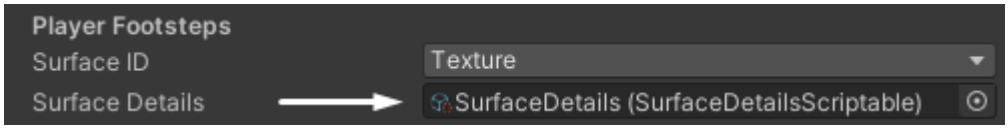

- To change interaction titles depending on another script field, assign **Script** and **Field Name** fields. The field must be public to use this feature.
- To create your own interactive object with your own script, create a public **UseObject()** function.

# SURFACE DETAILS

- With surface details, you can easily define new materials with different material surface settings.



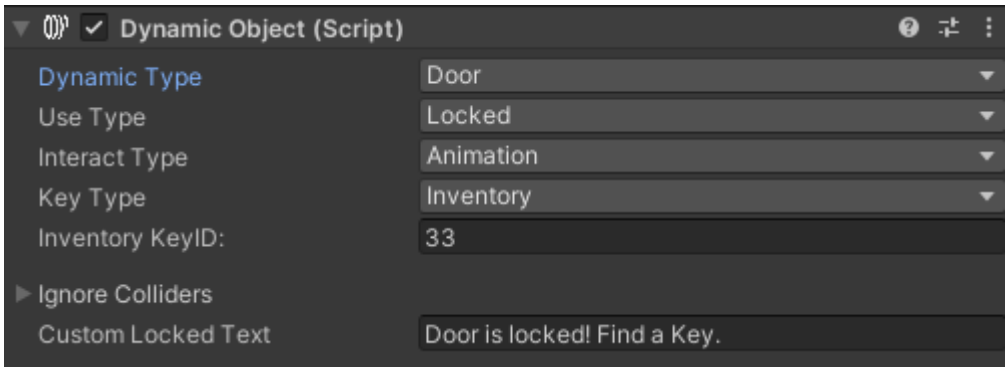

- You can use **Surface Details** to define different footsteps or bullet marks for different textures.
- **Surface Details** also support terrain textures, so you can easily define each surface of the game.



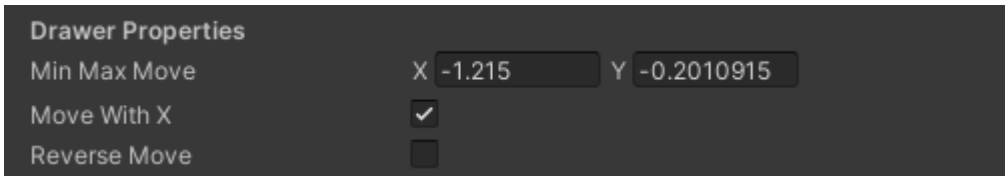

- Using surface details is very simple, just move your surface data into a script.
- To use classic surface detection, change the **Surface ID** to **Tag**.
- The **Surface Details Asset** also contains all the required functions, so obtaining elements is very easy.
- Each function of **Surface Details** is well commented.

# DYNAMIC OBJECTS

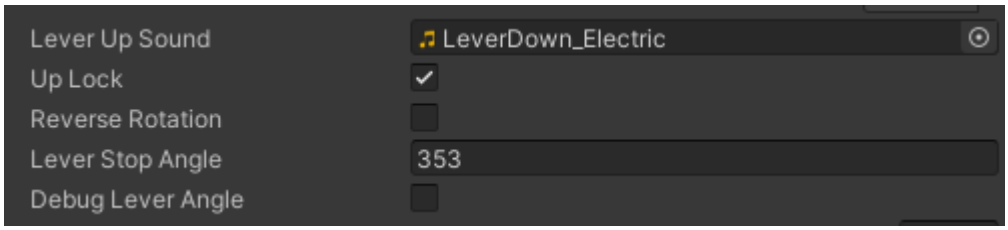
- Using a **DynamicObject.cs** script, you can easily define interactable **Doors**, **Drawers**, **Levers**, **Valves**, or **Moving Interactions**.




- You can define a new dynamic type by changing the **Dynamic Type** field.
- By changing the **Use Type**, you can determine whether the object is **Normal**, **Locked**, or **Jammed**.
- By changing the **Interact Type**, you can determine whether the object can be opened or closed using **Animation** or by dragging the **Mouse**.
- The **Key Type** determines how you want to unlock the **Locked Use Type**.



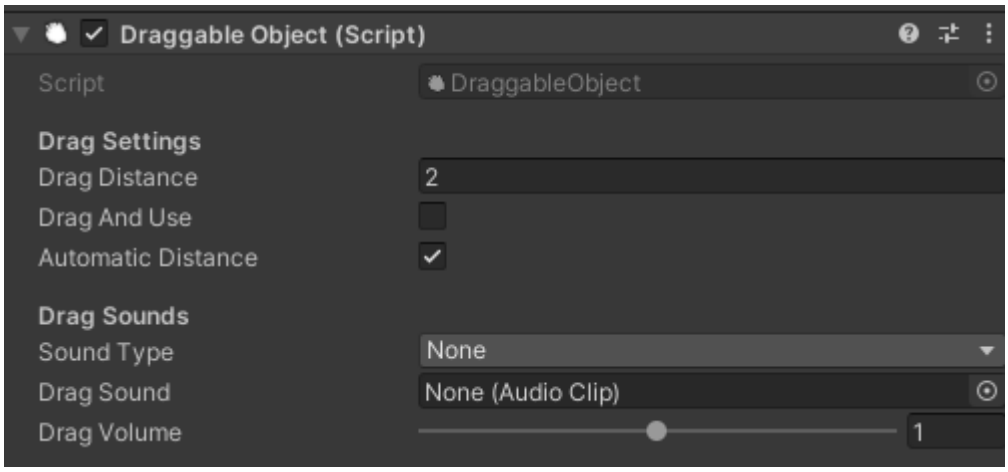

- By defining a new dynamic **Drawer** type with an interact **Mouse** type, you must assign a minimum and maximum pull position by changing the **Min Max Move** field.



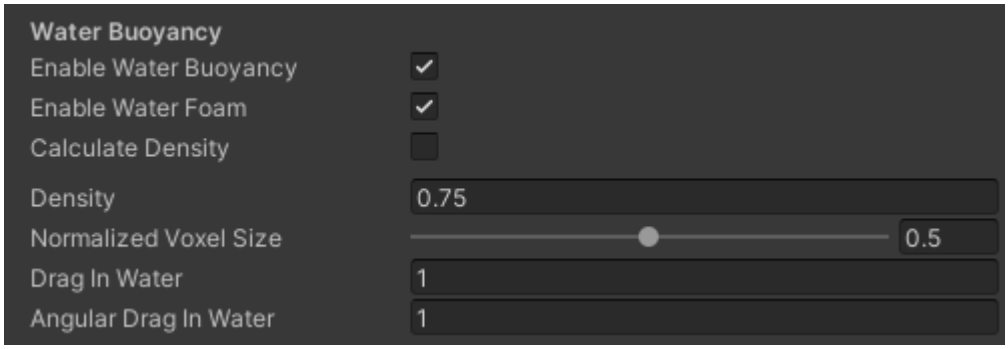

- The same must be done with the dynamic **Lever** type by changing the **Lever Stop Angle** field. If you have trouble defining the stop angle, check the **Debug Lever Angle**.

# DRAGGABLE OBJECTS

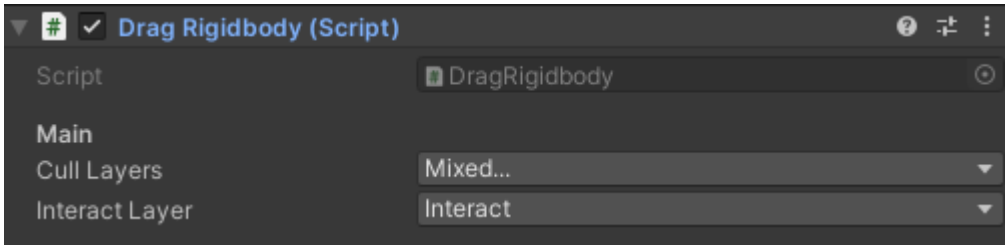
- Defining draggable objects is very simple, just change the object layer to **Interact** and add the **DraggableObject.cs** script.

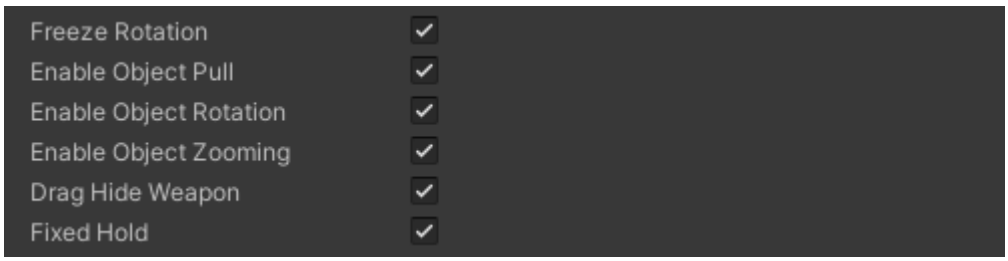



- You can also determine the water behaviour of draggable objects.
- Object Density > Water Density = Lower Object Buoyancy.



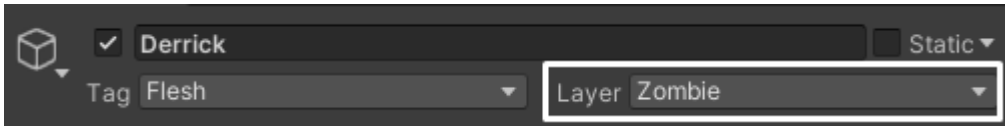

- You can change other settings in the **Drag Rigidbody** script located in the **PLAYER** -> **MouseLook** object.



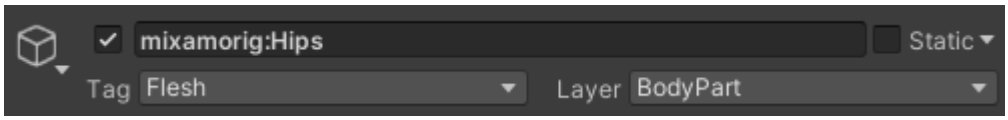



# ADDING NEW ZOMBIE AI

1. Convert your **Zombie Character** to a **Ragdoll** (GameObject -> 3D Object -> Ragdoll).
2. Adjust the ragdoll colliders to match your character model.
3. Change the zombie root layer to **Zombie**.

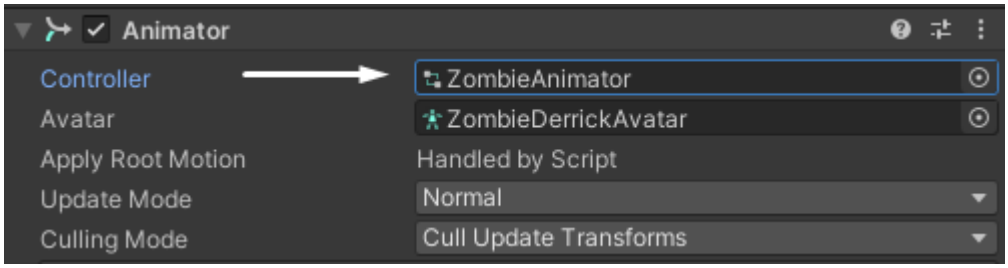



4. Then change the zombie hips layer to **BodyPart** and tag to **Flesh**.

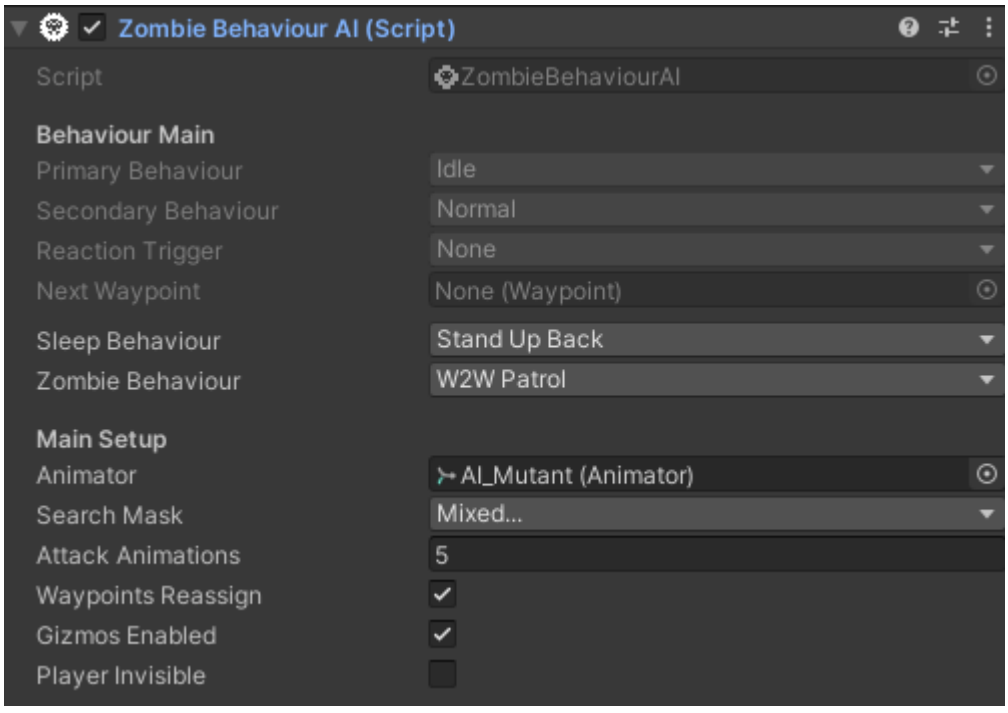



5. Assign an **Animator Controller Asset** to an **Animator** component.

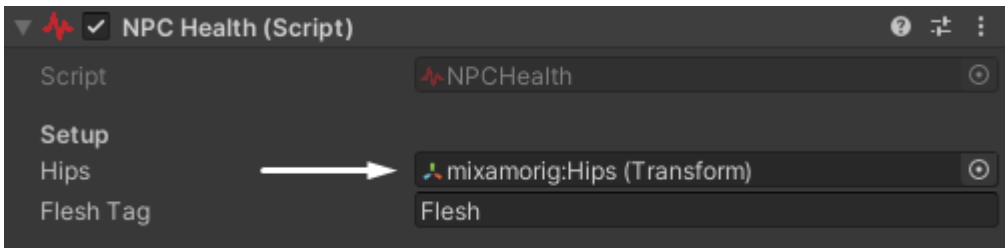
- It is recommended to use the default **ZombieAnimator Controller** as it contains all required connections and behaviour scripts.



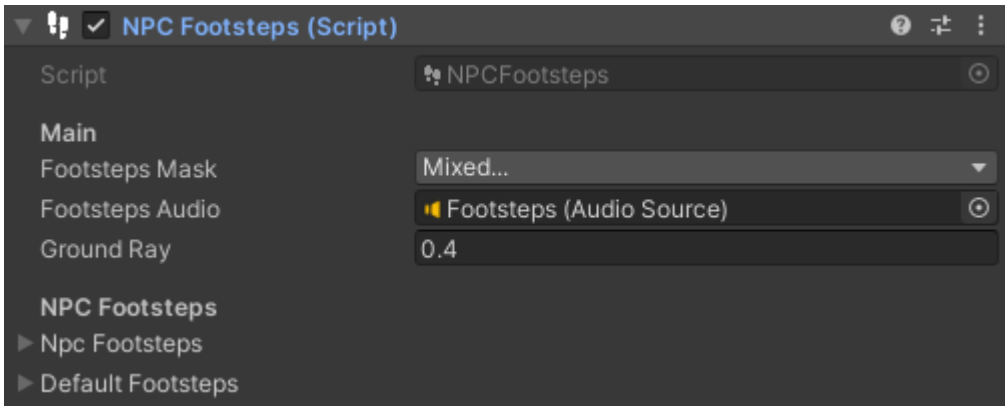

6. Add **ZombieBehaviourAI.cs** script to zombie root.



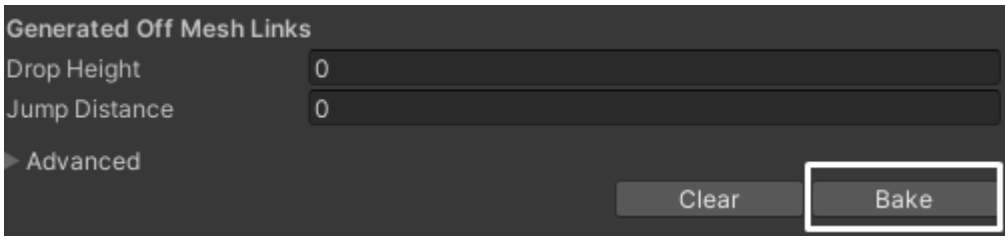
7. Add **Capsule Collider** component to the zombie root and adjust the collider to match your character model.
8. Assign **Zombie Animator** component to an **Animator** field and define a **Search Mask**.
9. Add the **NPCHealth.cs** script to the zombie root and assign zombie hips.




10. Add the **NPCFootsteps.cs** script to the zombie root.




11. Create a new empty object, add an **Audio Source** component, change **Spatial Blend** to **3D** and move the object as the zombie root child object.
12. Add a **Nav Mesh Agent** component and adjust it to fit your character's model. Make sure that the **Stopping Distance** is not zero and **Auto Braking** is enabled.
13. Go to **Window -> AI -> Navigation -> Bake** and click **Bake**.

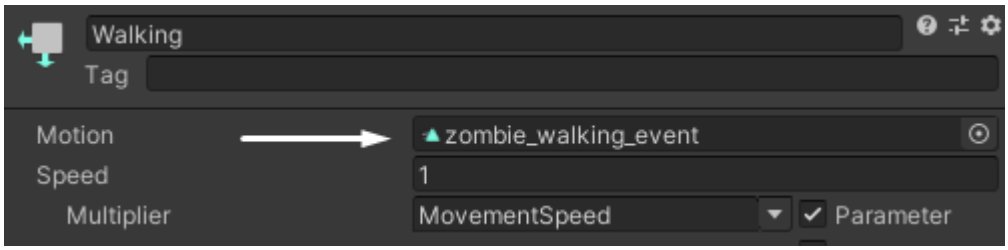



14. Add **WaypointGroup.cs** script to the empty object.

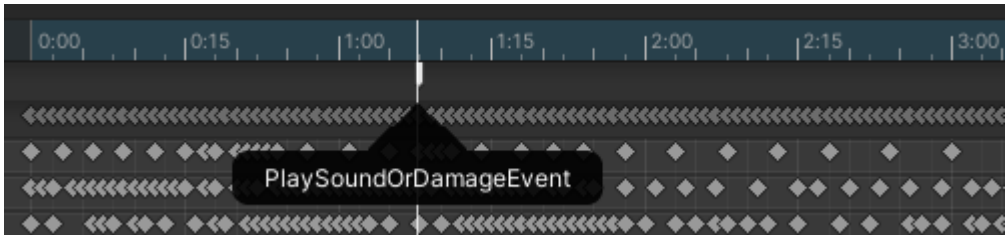
- Waypoints are added automatically by adding empty objects to the **Waypoint Group** child object.

- You can define your own character behaviour by changing the **Behaviour Settings**.


- If you want, you can also define hunger points by adding the **HungerPoint.cs** script to the object.
- Zombie will search for **Hunger Point** objects, if **Hunger** is enabled and the **Hunger Points** field in the **AI Zombie Behavior** script is zero.




- If you have your own zombie animations, it is recommended to change the default animations of the zombie controller asset.



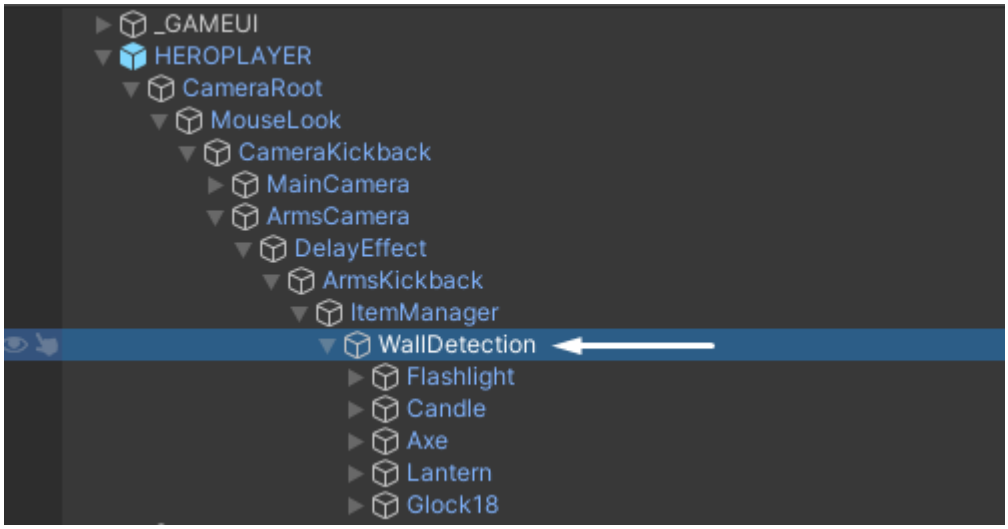

- To damage the player, you must create an animation event that calls the **PlaySoundOrDamageEvent()** function with a parameter of 0.

**Sleep Behaviour:** The zombie starts with a sleep animation if the player is in close distance or if it makes any impact sound, the zombie gets up and changes his sleep behavior to None.

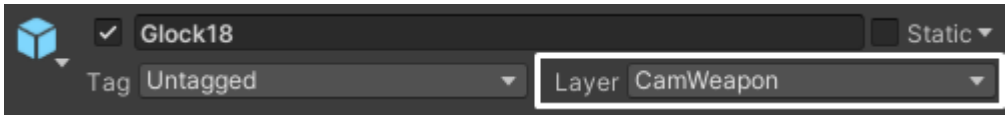
**Zombie Behaviour:** The basic awake behavior of the zombie. Waypoint to Waypoint, Waypoint to Waypoint with Patrol, Waypoint to Waypoint with Idle.

# ADDING NEW PLAYER WEAPONS

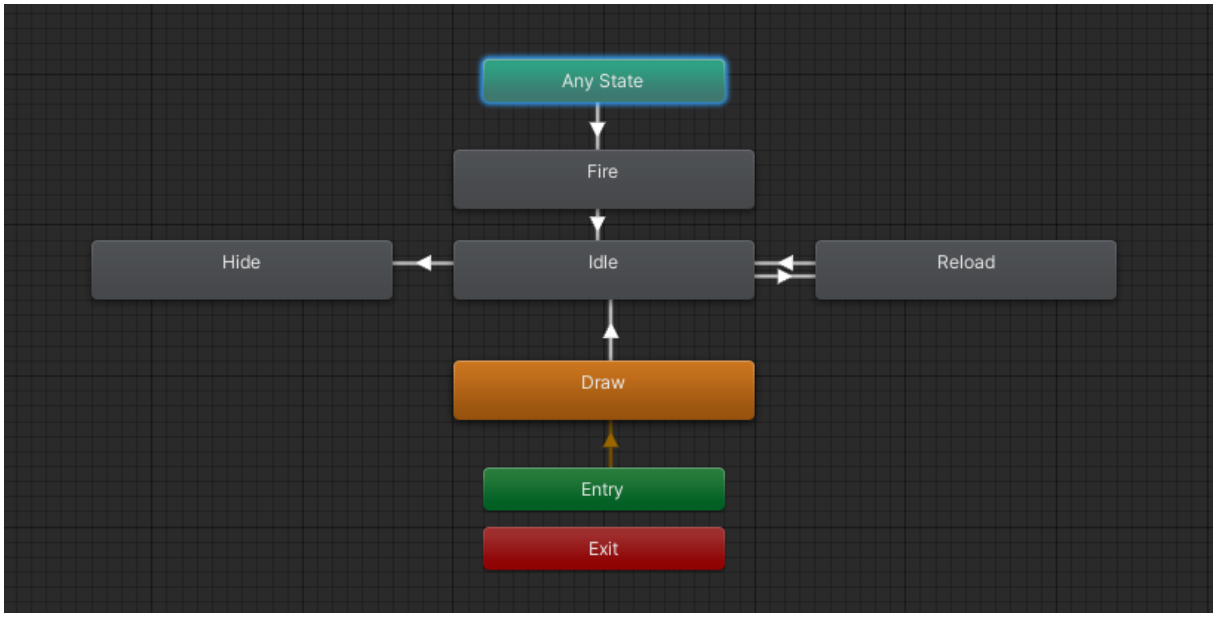
1. Locate the **WallDetection** object in the **PLAYER** object, which contains all the usable objects.



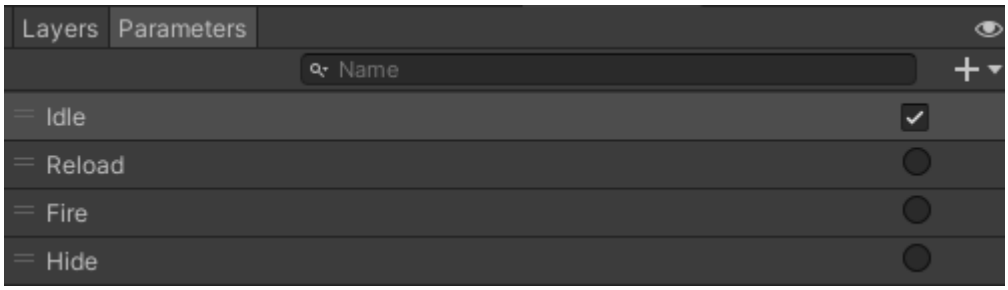

2. Duplicate one of these items and replace the disabled object inside the duplicated object with your object.
3. Set a new layer of object to the **CamWeapon**.



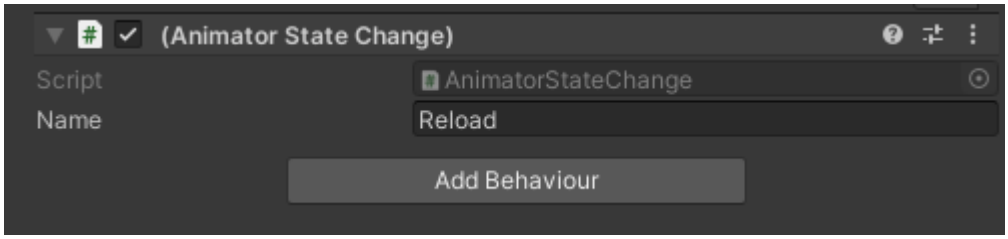

4. Create a weapon **Animator** asset.



5. Add the required parameters: **(bool) Idle = true**, **(trigger) Reload**, **Shoot**, **Hide**

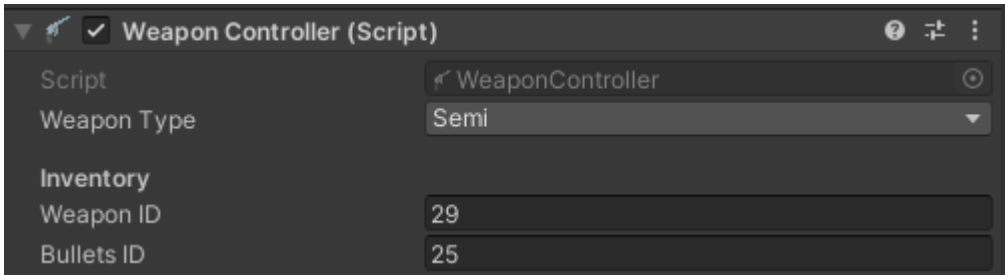



6. Create all required state transitions.
7. Add **AnimatorStateChange** behaviour to **Draw, Hide, Reload** states with the name **Draw, Hide, Reload** that the **WeaponController** script uses.

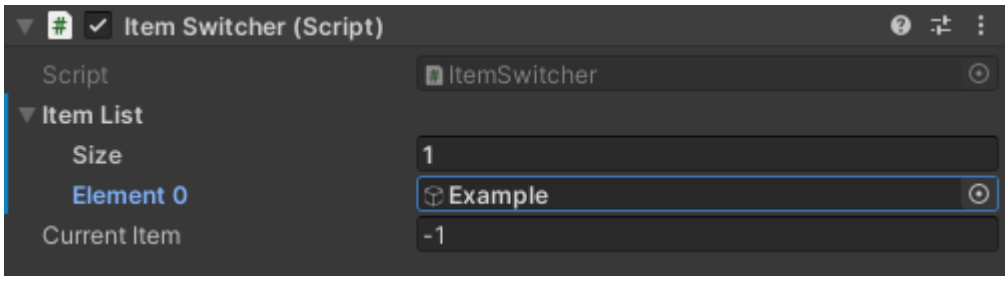



- This behavior script finds and sends state data to the **WeaponController** script using the **IOnAnimatorState** interface.
- It is important to write exactly **Draw, Hide or Reload** in the name field, depending on the state.

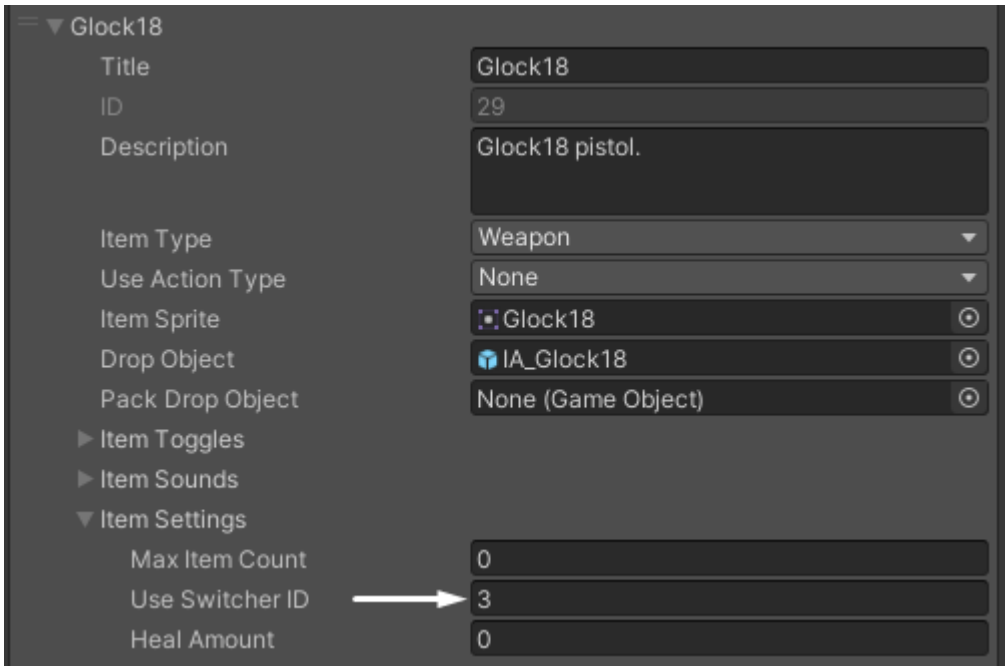
8. Add the **WeaponController.cs** script to the weapon root object and set the weapon parameters.



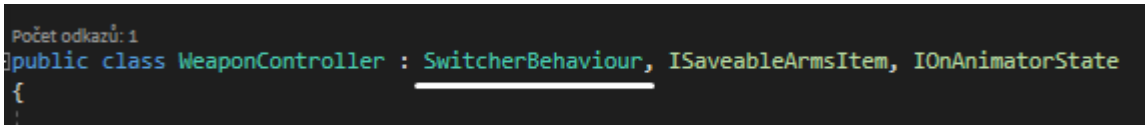

9. Go to the **ItemManager** object and add a new weapon to the **ItemSwitcher -> Item List**.



10. To use your weapon from the inventory, you must define the **Use Switcher ID** field from the **Item Settings** drop-down menu. **Weapon ID** is the ID of the **Item List** element.




- You can also define your own usable weapons/items by adding the **SwitcherBehaviour** subclass to your own script.

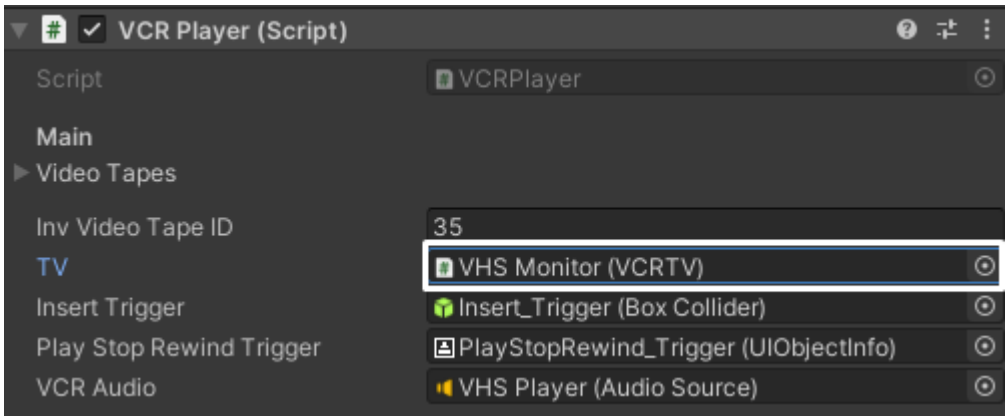



- You can add an **ISaveableArmsItem** interface to a script to save and load usable object data.

# SETTING UP VHS PLAYER

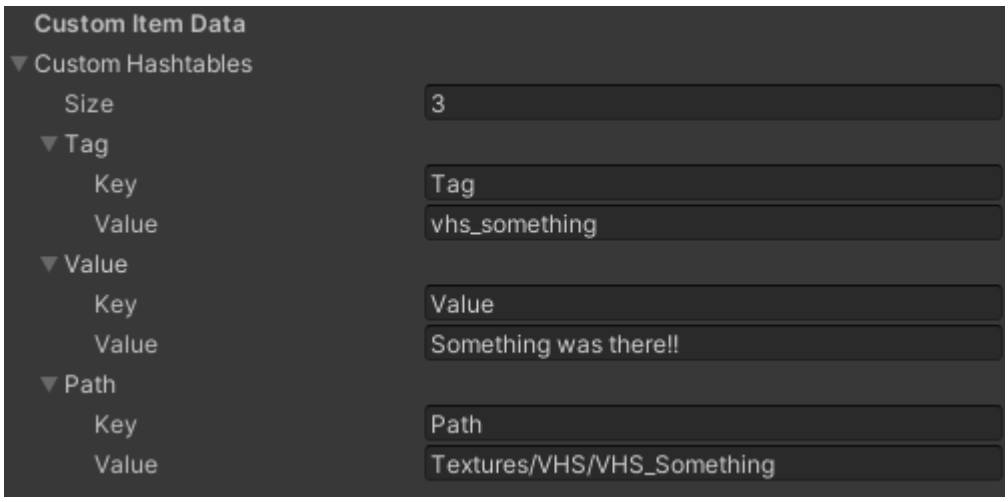
- Supported video formats: [Video file compatibility]

1. Move **VHS Player** and **VHS Monitor** from **Resources** folder to the scene.
2. Connect **VHS Player** with **VHS Monitor**.




3. Set **Inventory Video Tape ID** and add your video tapes.

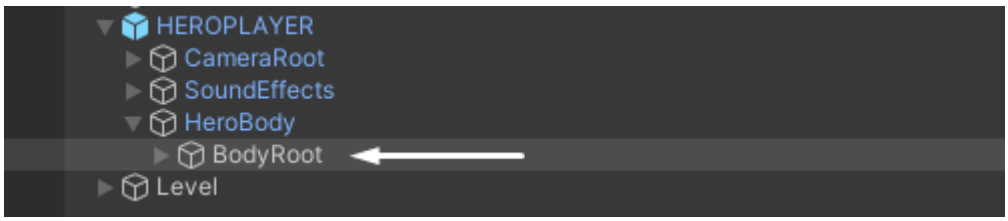

4. In **Interactive Item** script, set VHS Tape **Custom Item Data**.



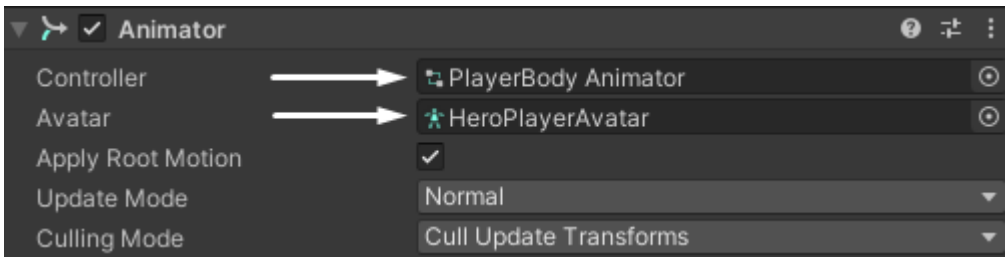

- All **Key** field values must be the same as in the screenshot.
- The **Path** value is the path inside the **Resources** folder.

# ADDING CUSTOM PLAYER MODEL

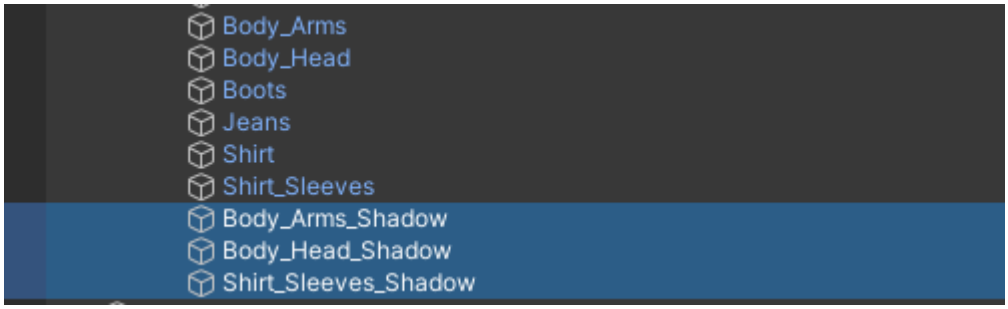
1. Replace the **BodyRoot** object with your **Player** model.

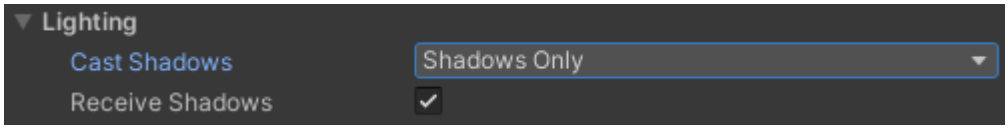



2. Assign the **PlayerBody Animator** asset to the **Controller** field and make sure you have the **Humanoid Avatar** assigned to the **Avatar** field.

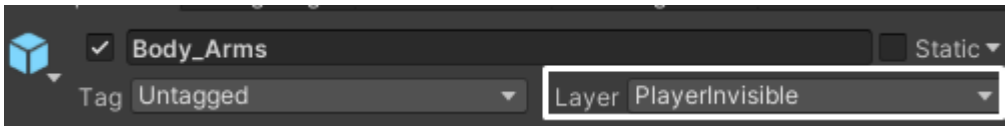


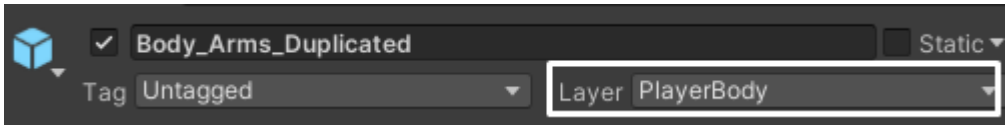

3. If you want your hands and head to be invisible to the **Main Camera**, you must duplicate these objects and change their **Cast Shadows** field to **Shadows Only** in the **Lighting** drop-down menu.



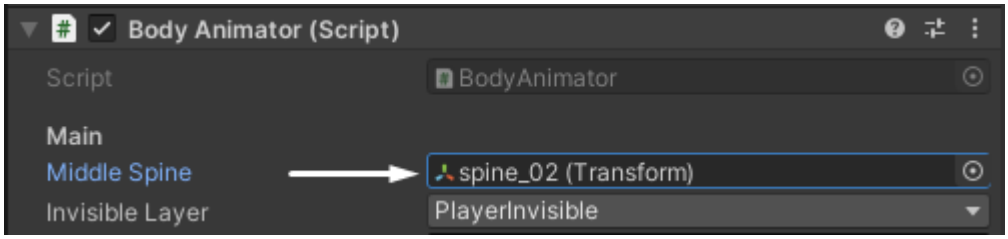




4. Next, you need to change the layer of the original duplicated objects to **PlayerInvisible** and the other player objects to **PlayerBody**.

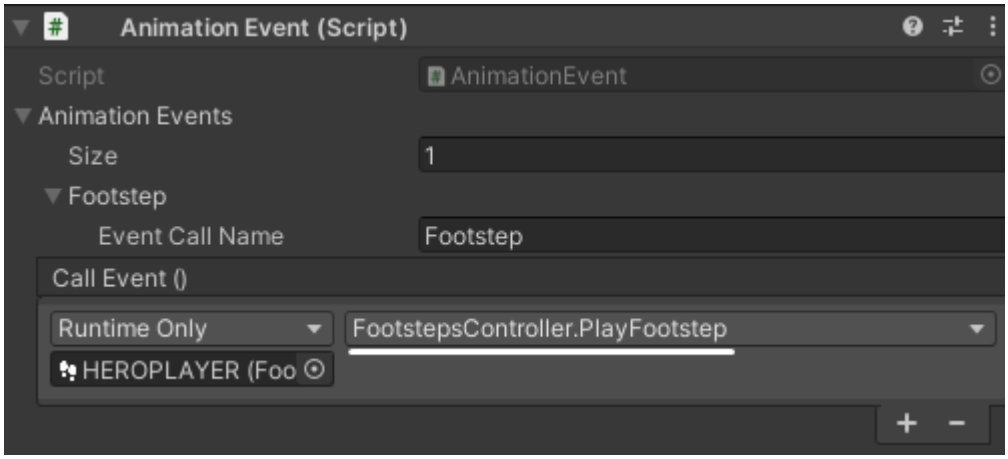


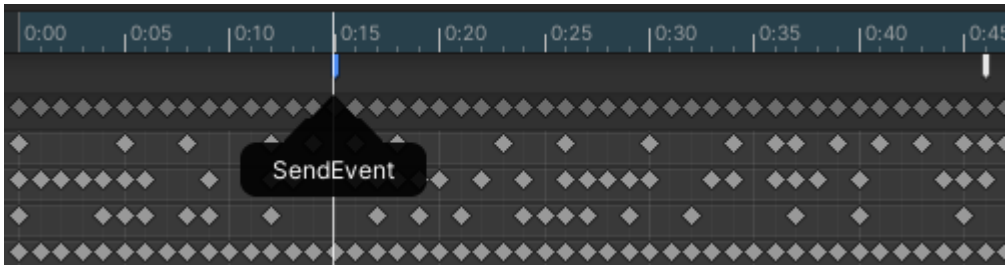
5. Assign a player's middle spine to the **Body Animator -> Middle Spine** field.




6. To perform event-based footsteps, you must add the **AnimationEvent.cs** script to the player's body and create a **FootstepsController -> PlayFootstep()** event.




7. Then, at a specific animation time, create an event that points to the **AnimationEvent -> SendEvent()** function.




8. As the **SendEvent()** parameter set the text of the **Event Call Name**.

# EMERALD AI INTEGRATION

1. Open an empty scene and import **Emerald AI** first and then **Horror FPS KIT**.
2. Go to **Horror FPS KIT\HFPS Assets\Content\Scripts\Core\Examples** and unzip the **EmeraldAI.zip** package.
3. Replace default **EmeraldAIPlayerDamage.cs** script with an extracted version. If the script already contains the **DamageHFPSPlayer()** function, uncomment it and skip this step.
4. Move **EmeralAISendDamage.cs** script right after the enemy **EmeraldAISystem** script.
5. Follow the **Emerald AI** instructions and create enemies.

# HDRP/URP PARTIALL SUPPORT

- The **Horror FPS KIT** has partial support for **HDRP** or **URP**, which means that the kit is compatible with these rendering pipelines, but after setting up on these pipelines, pink materials or post-processing problems may occur.

- If you choose to use the **URP** rendering pipeline, some scenes may contain pink materials. This problem can be easily solved by changing all shaders of pink materials to **URP** default shaders. The reason you need to do this is because Unity has chosen not to support other shaders, such as **URP** or **HDRP** shaders.

- If you choose to use the **HDRP** rendering pipeline, you will experience the same problem with pink materials as in **URP**, but there will also be an incompatibility with post-processing. This is because the **HDRP** rendering pipeline uses its own post-processing system and the camera **blur effect** will not be compatible anymore. You can also easily solve the problem with the pink materials, but if you want to use the **blur effect**, you have to write a brand new **shader** and **script**, or you can use another method instead of the blur effect, such as a transparent black panel or something else.

# CREDITS

- This kit was developed and designed by **ThunderWire Studio**
  © All rights reserved.
- Almost all assets are created by **ThunderWire Studio**, except those that are downloaded under a royalty free license.

## BUG, ERROR REPORT

- If you find any problem with the kit, send a message to our e-mail address and carefully describe the problem you found: **thunderwiregames@gmail.com**
- Or visit our **Customer Support** page or **Contact** page and email us there.

## USEFUL LINKS

- **ThunderWire Studio** - Youtube Channel
- **ThunderWire Studio** - Developer Website
- **ThunderWire Studio** - AssetStore
- **Horror FPS KIT** - Website
- **Horror FPS KIT** – AssetStore