



HORROR I.VI KIT

DOCUMENTATION

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

ABOUT HORROR FPS KIT	1
FEATURES.....	3
MANUAL PROJECT SETUP	4
MAIN MENU SETUP	6
GAME SETUP	6
ADDING NEW INPUTS.....	7
MENU CONTROLLER.....	10
OPTIONS CONTROLLER	11
SAVE/LOAD MANAGER	12
SETTING UP	12
ADDING CUSTOM SAVEABLES	13
SAVING GAME DATA	15
SAVEGAMEHANDLER ERROR MESSAGES	15
ADDING RUNTIME SAVEABLES	16
ADDING NEW CUTSCENES.....	18
ADDING NEW OBJECTIVES	19
ADDING NEW FLOATING OBJECTS.....	21
INVENTORY	22
ADDING NEW INVENTORY ITEMS	22
INVENTORY CONTAINERS	24
COMBINABLE ITEMS	25
INVENTORY CUSTOM ACTIONS	26
CHANGING SCRIPTS UI TEXTS.....	28
PLAYER BODY (Body Animator).....	29
ADDING NEW INTERACTIVE ITEMS	30
SURFACE DETAILS.....	31
DYNAMIC OBJECTS.....	32
DRAGGABLE OBJECTS	33
ADDING NEW ZOMBIE AI.....	34
ADDING NEW PLAYER WEAPONS	37
SETTING UP VHS PLAYER.....	40
ADDING CUSTOM PLAYER MODEL	41
EMERALD AI 2.0 INTEGRATION.....	43
HDRP/URP PARTIALL SUPPORT	43
CREDITS.....	44

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

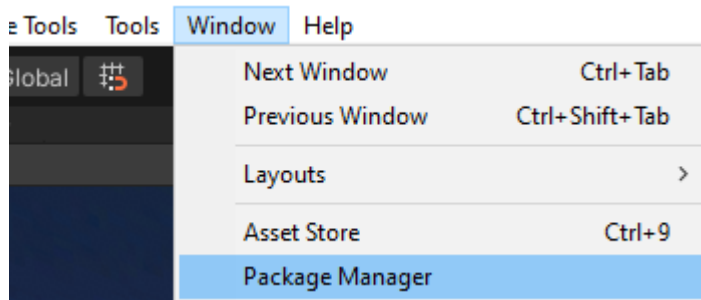
- Rebindable Keyboard Controls at Runtime
- Zombie AI (Sleep, Wander, Scream, Agony, Hunger, Attack, Attract, Footsteps)
- Scene Preloader (Background Change, Tips)
- Jumpscare (Animation, Scare Effects, Scared Breath, Customizable)
- UI Menu (Main Menu, Pause Menu, Load Menu..)
- Notifications (Pickup,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

MANUAL PROJECT SETUP

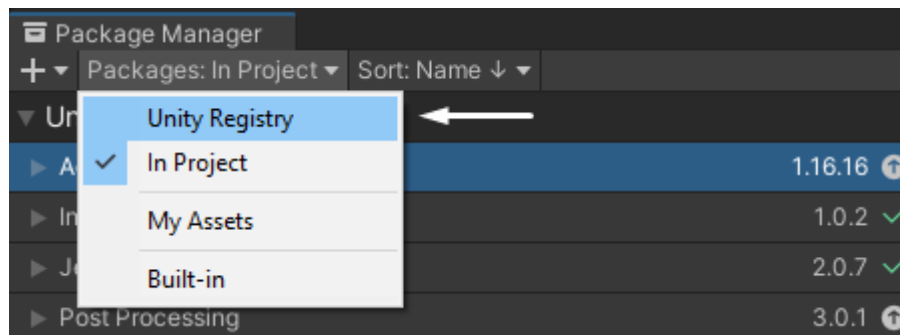
We strongly recommend importing asset into an empty project and importing all tags and layers!

Make sure that you have imported **Input System**, **Post Processing** and **Timeline** from **Package Manager**!

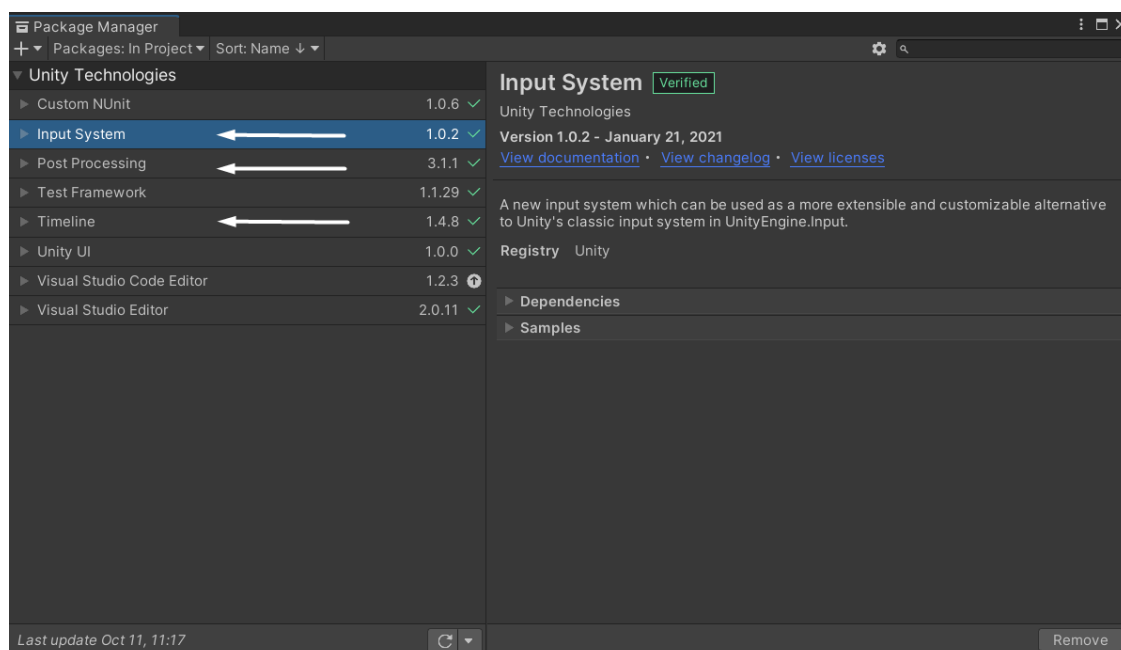
1. Open **Package Manager** from **Window** drop-down.



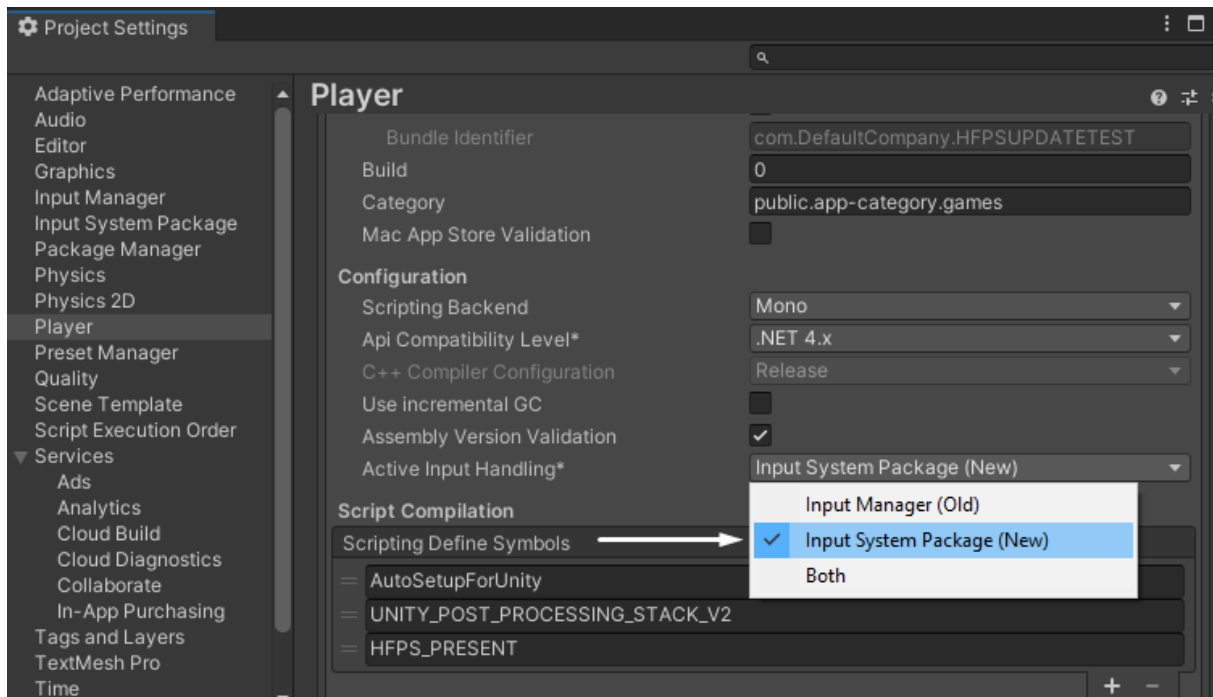
2. Select **Unity Registry** from the **Packages** drop-down menu.



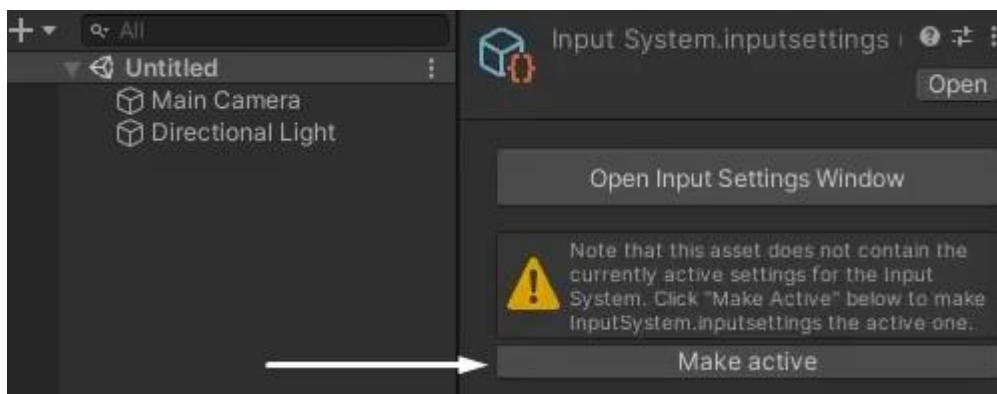
3. Find the packages mentioned above and import them into a project.



4. Go to **Player Settings** and set **Active Input Handling** to **Input System Package**.



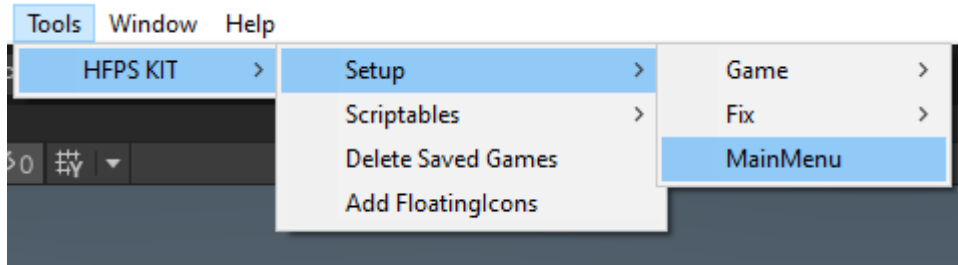
5. Go to **Horror FPS KIT\HFPS Assets\Scriptables\InputSystem** folder and click activate in **InputSystem.inputsettings**.



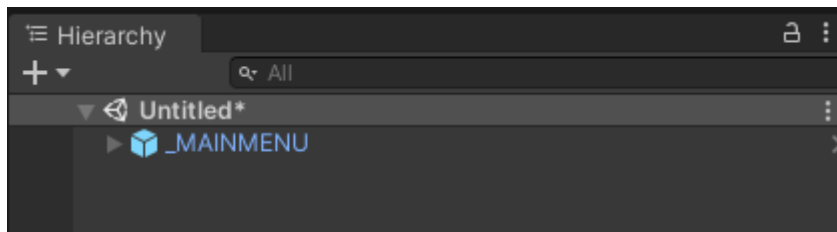
- If this button does not exist, ignore this step.
6. If you see this message after clicking the Play button - **"[Device Init] Supported Input Device is not connected!"**, or the game does not respond to any inputs just restart the Unity.
- If you have trouble setting up a project along with HFPS, here is a video on how to set up a project properly:
<https://www.youtube.com/watch?v=iX5voeju3MQ>

MAIN MENU SETUP

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



- You should now see the **_MAINMENU** object added to your scene.

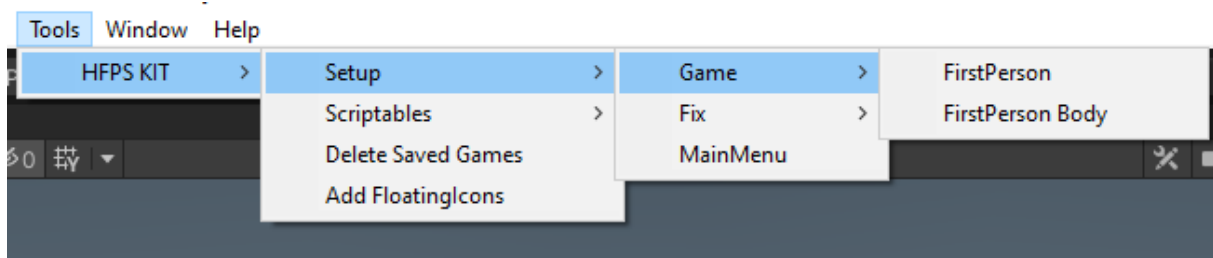


- Add a Camera object to your scene.

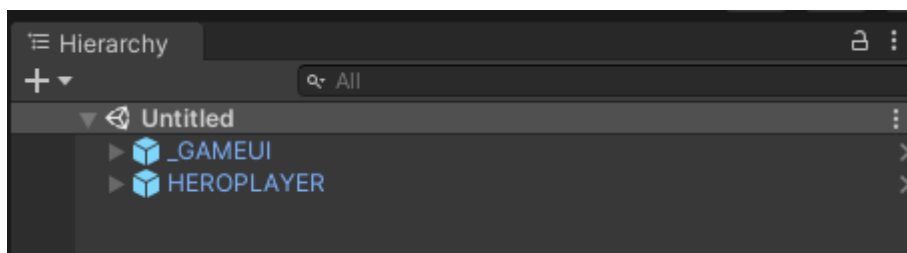
GAME SETUP

The steps are the same as when setting up the main menu.

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

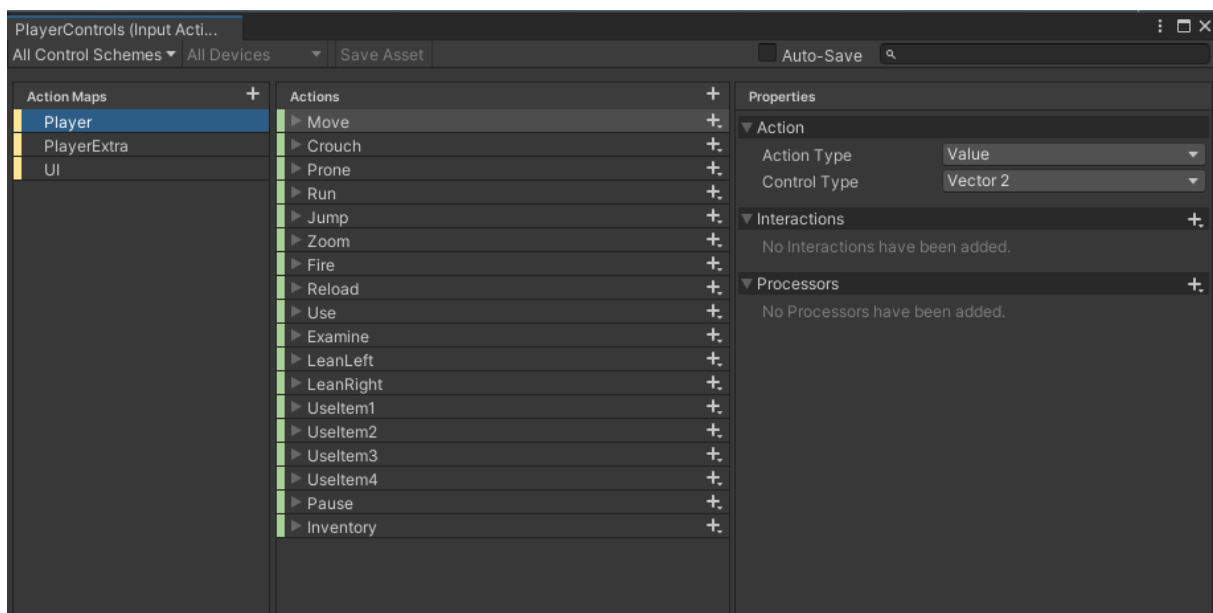


3. You should now see the **_GAMEUI** and **HEROPLAYER/FPSPLAYER** objects added to your scene.

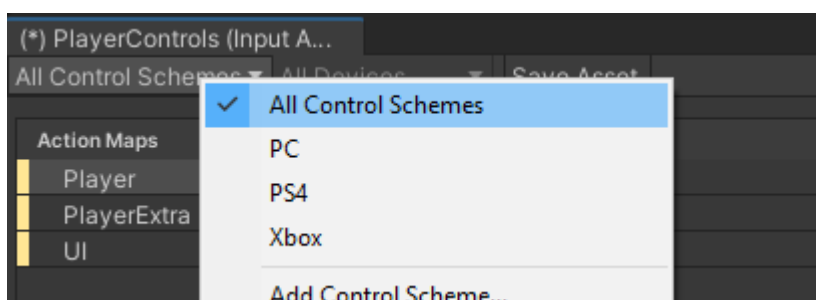


ADDING NEW INPUTS

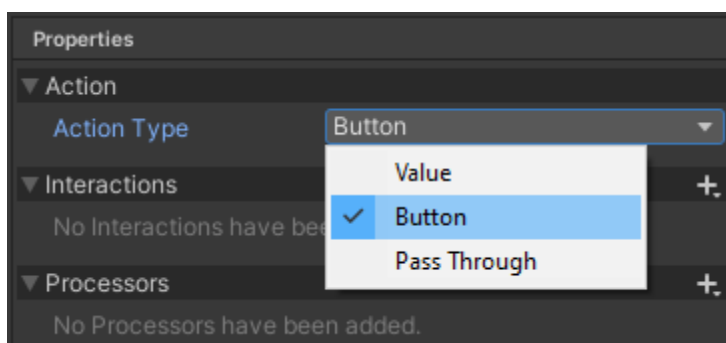
1. Open the **PlayerControls** asset window by double-clicking on it.



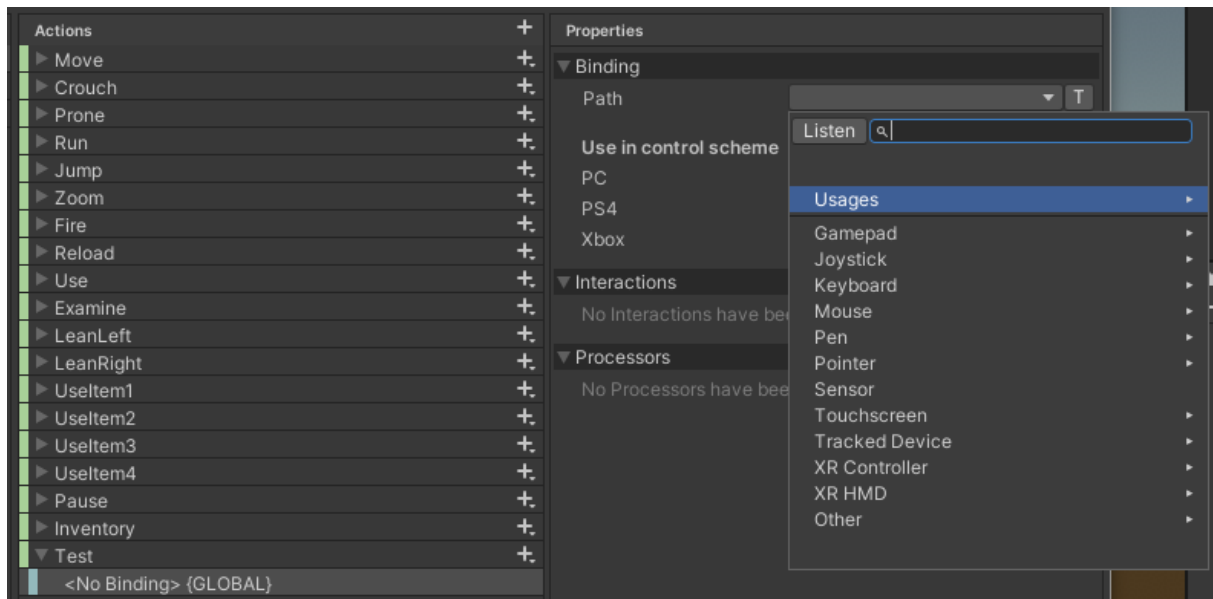
2. Clicking on the + button you will add a new action.
3. In the top panel select the control scheme.



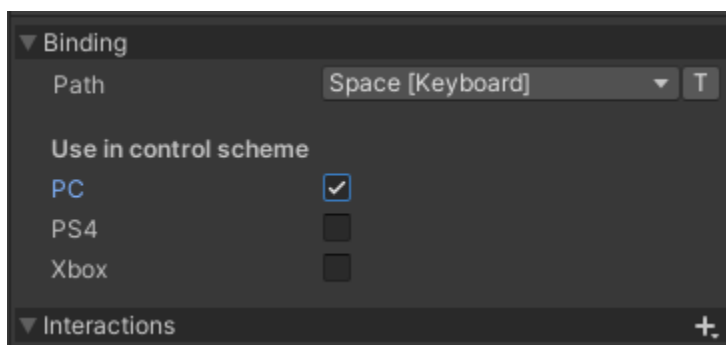
4. After clicking on the action, you can select the **Action Type**.



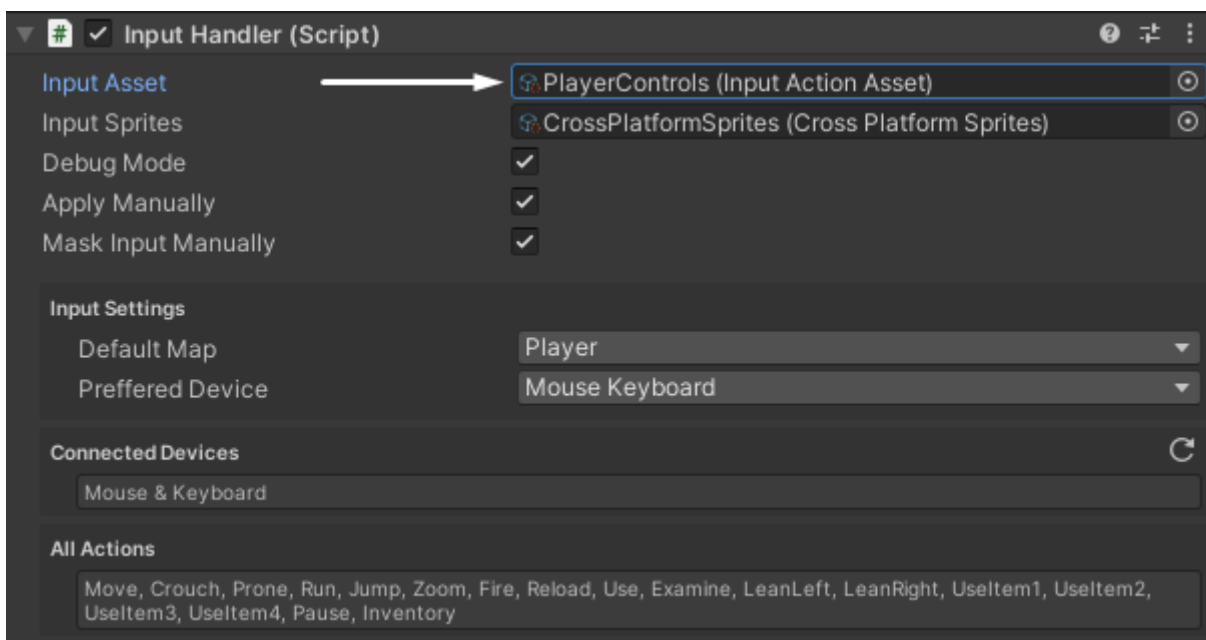
- Now select the binding path. The binding path will vary depending on the currently selected action type!



- After selecting the binding path, select the control scheme that belongs to the input.



- Now click the **Save Asset** button to save all input changes.
 - If you create a new **Input Action Asset**, you must assign it to an **Input Handler** to use it.

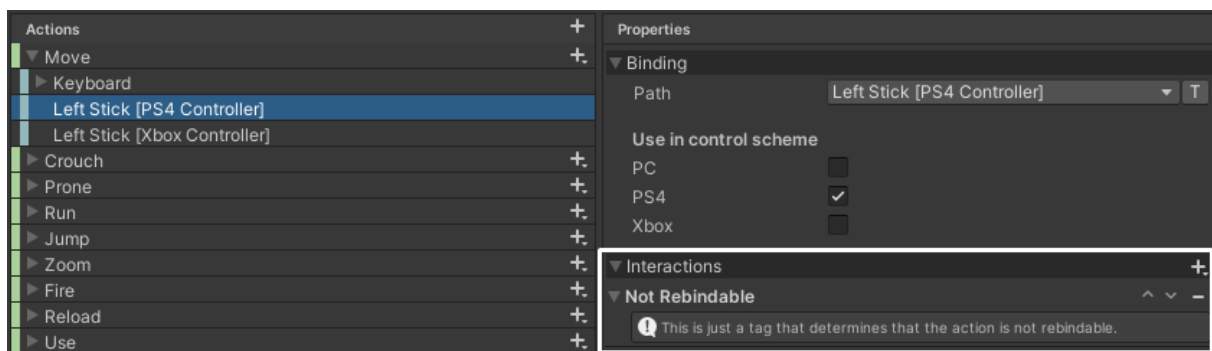


- After adding a new action, you can use it using the **Input Handler** functions.

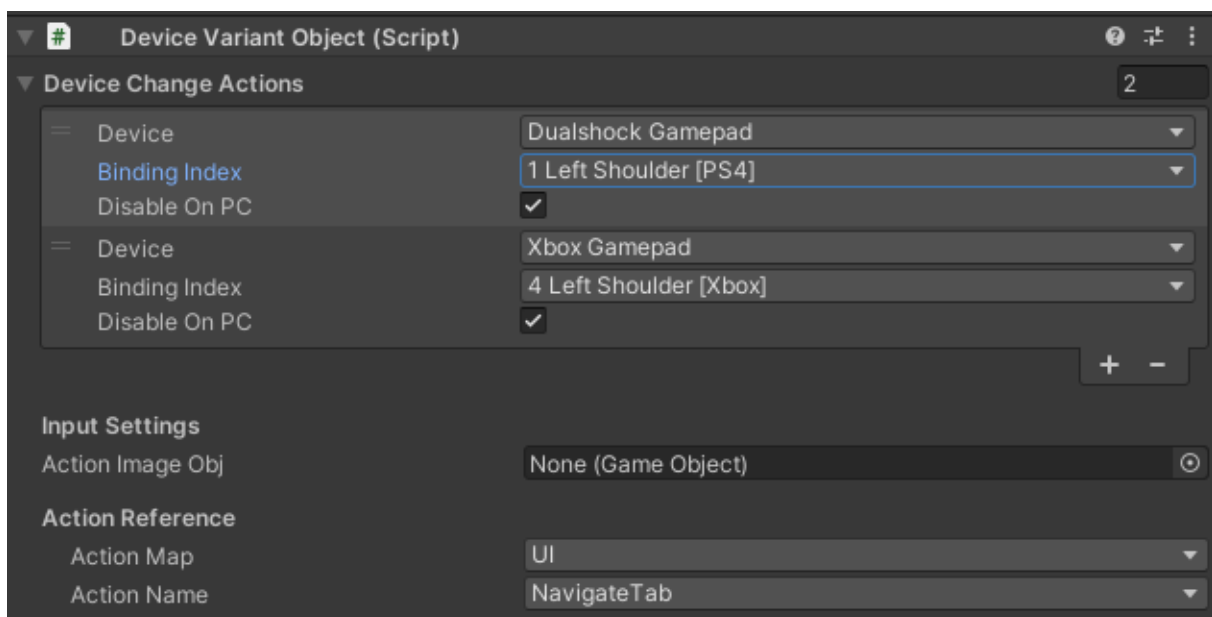
```
1. if (InputHandler.ReadButtonOnce(this, ActionName))
2. {
3.     Debug.Log($"Action {ActionName} is pressed once!");
4. }
```

```
1. Vector2 movement;
2. if ((movement = InputHandler.ReadInput<Vector2>(ActionName)) != null)
3. {
4.     Debug.Log($"Action {ActionName} is updating {movement}!");
5. }
```

- When using the **ReadInput<T>** function, pay attention to using the return type that belongs to the action specified by the **Action Type**.
- You can specify which inputs cannot be rebound by adding a **Not Rebindable** interaction.

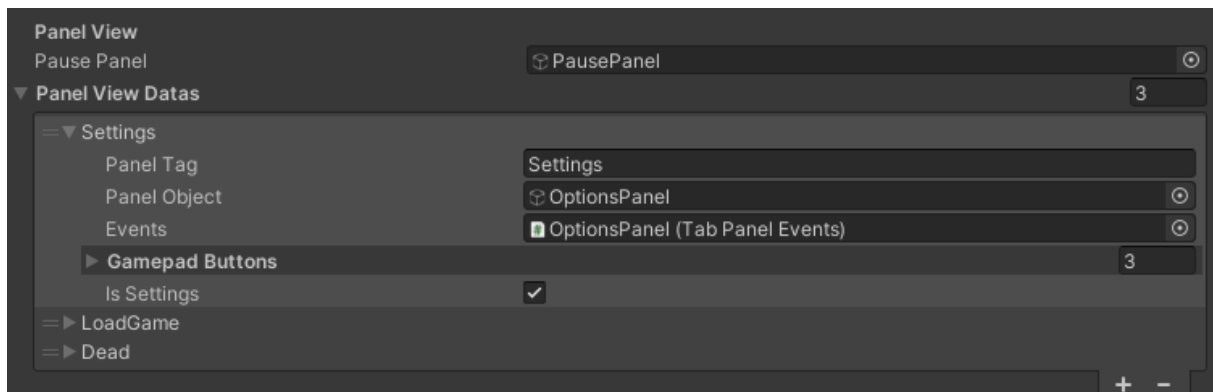


- If you have problems adding new inputs, you can see the official tutorial for adding new inputs from Unity:
<https://www.youtube.com/watch?v=5tOOstXalKE>

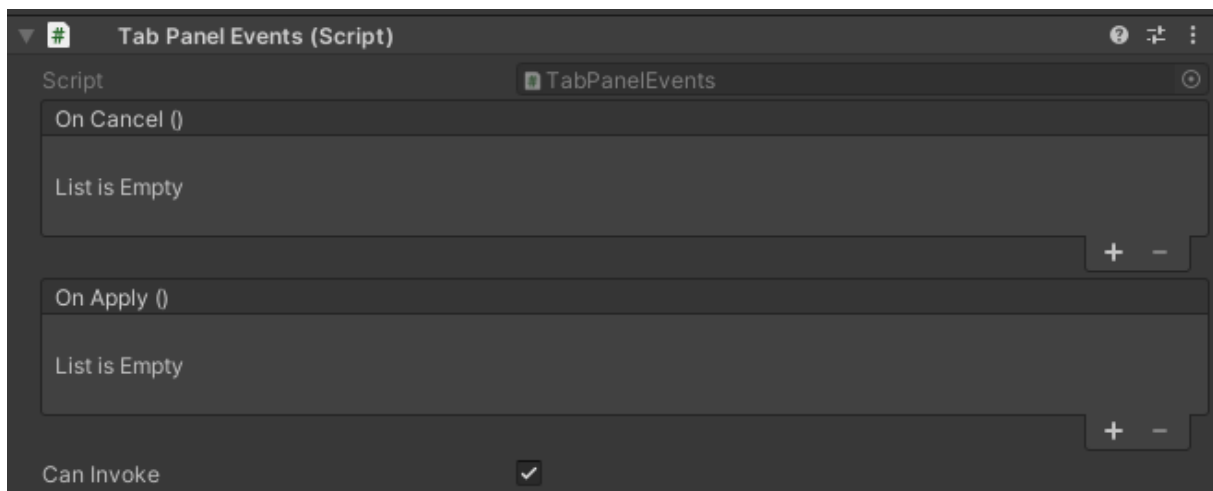


- You can define input icon using the **Device Variant Object** script, which changes the icon depending on the currently selected input controller.

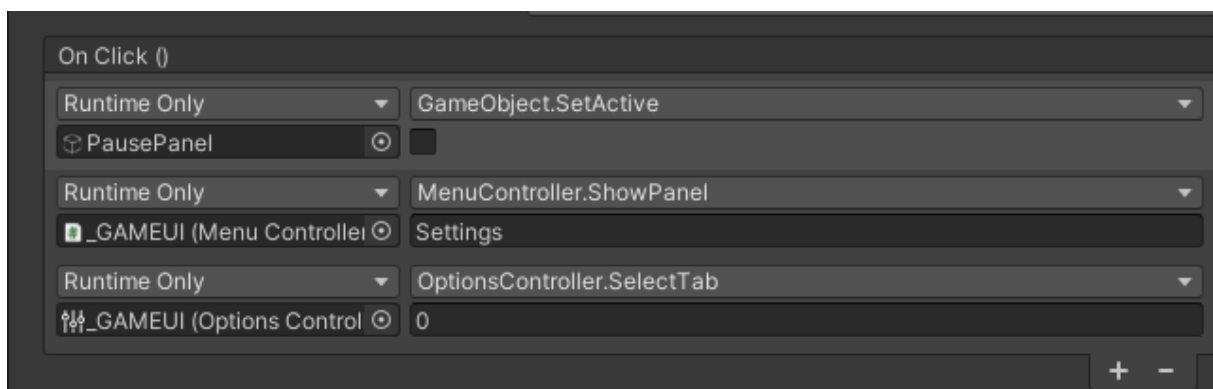
MENU CONTROLLER



- Using the Menu Controller script, you can easily define new menu panels that can be controlled using the keyboard or gamepad.



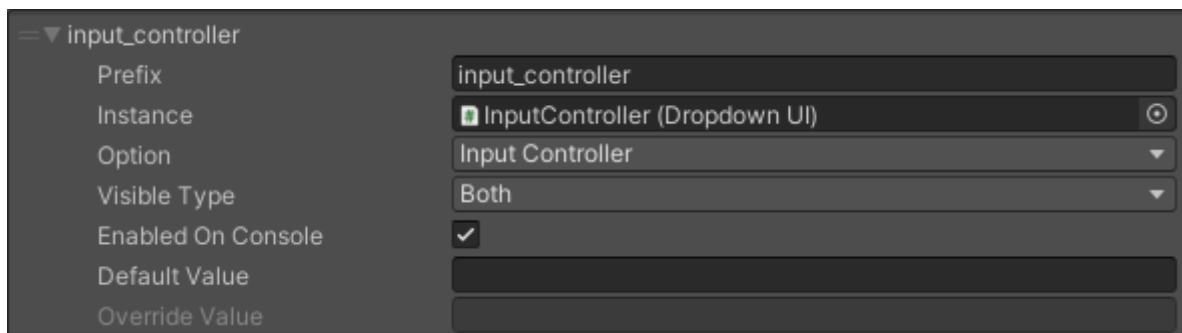
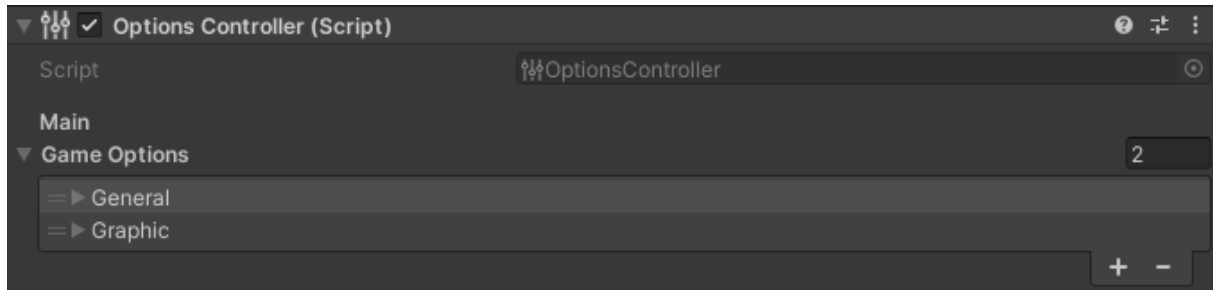
- **TabPanelEvents** defines what happens when you press **Cancel** or **Apply** button defined in the Input Action Asset. Tab Panel Events is used only when the panel is active.



- To display the panel, call the **ShowPanel()** function.
- When returning to the general menu, it is necessary to call the **ShowGeneralMenu()** and **ResetPanels()** functions.

OPTIONS CONTROLLER

- The Options Controller script gives you methods to define new game settings, which may vary depending on the current platform.



- You can select one of the default option types, or select a custom option type that you can manage using the Options Controller events and functions to get a value from an option.
- You can also specify whether the option can be displayed and used on the keyboard or gamepad controller.
- To listen for an options changes, use the OnOptionsUpdated event.

```
1. OptionsController.OnOptionsUpdated += OnOptionsUpdated;
```

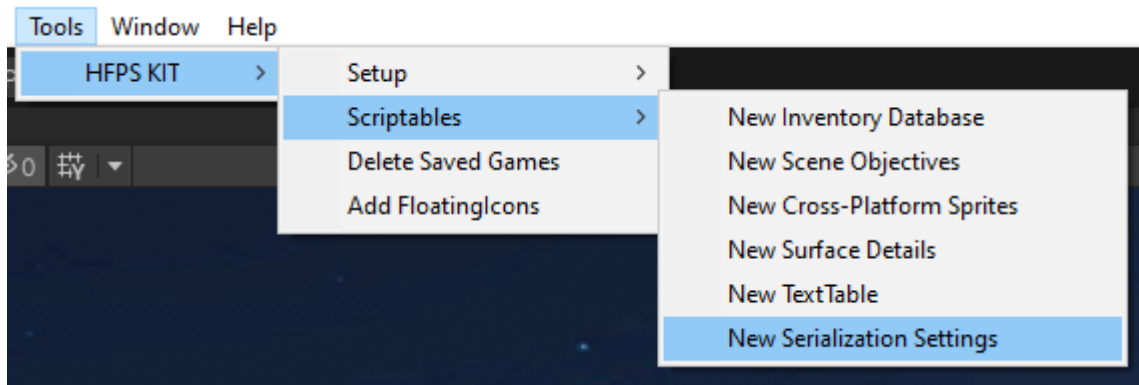
- After adding a function for the event, you can use the GetOptionValue() function to get the value of the option.

```
1. private void OnOptionsUpdated()
2. {
3.     if (OptionsController.GetOptionValue("KEY", out float value))
4.     {
5.         //your code goes here
6.     }
7. }
```

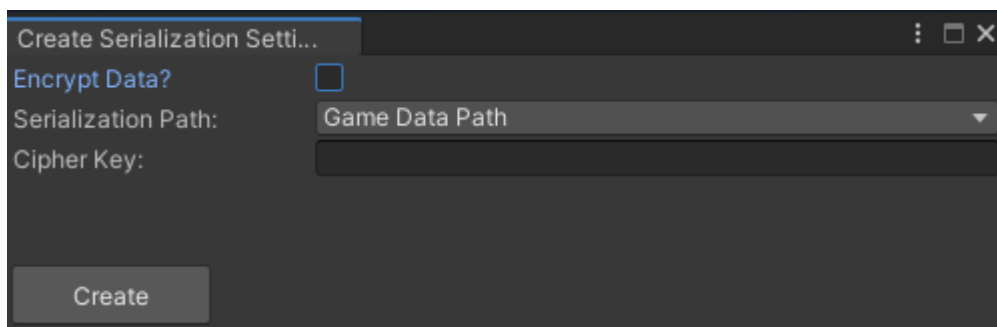
SAVE/LOAD MANAGER

SETTING UP

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

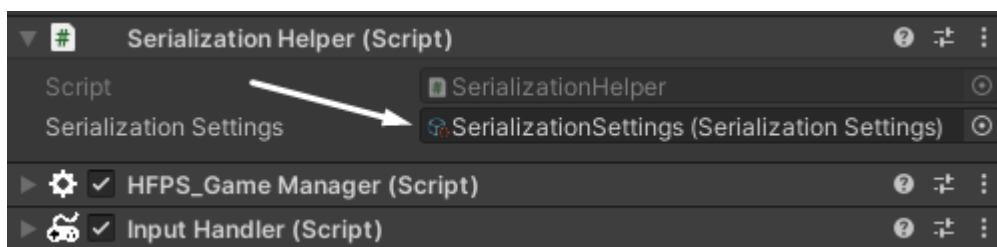


- This will show up a **Serialization Settings** window.



- **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 **Serialization Helper** script.



- The serialization path is used as the default path for storing saved games or settings files.

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;

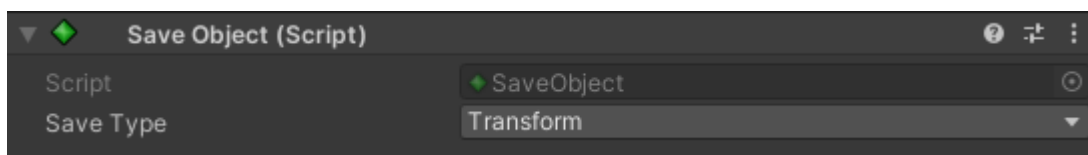
    [SaveableField, HideInInspector]
    public float exampleFloat = 10.5f;
}
```

- You can also set your own field key.

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

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

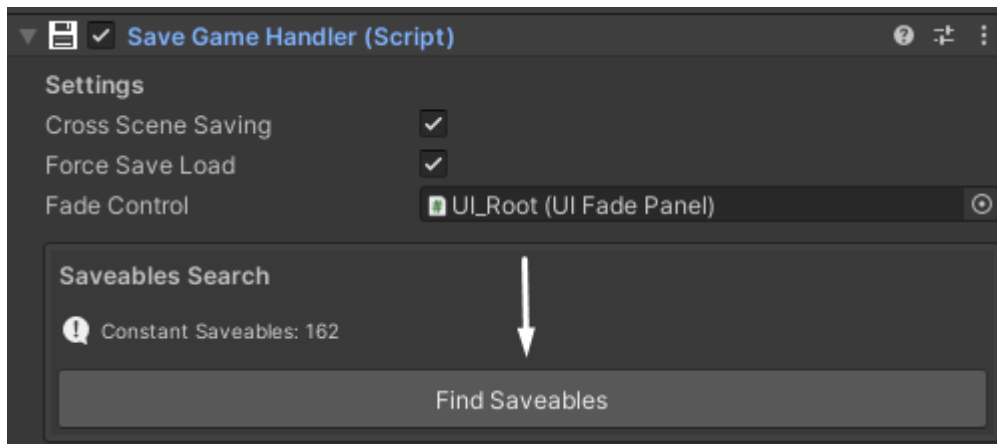
3. Adding **SaveObject.cs** script to **GameObject**.



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

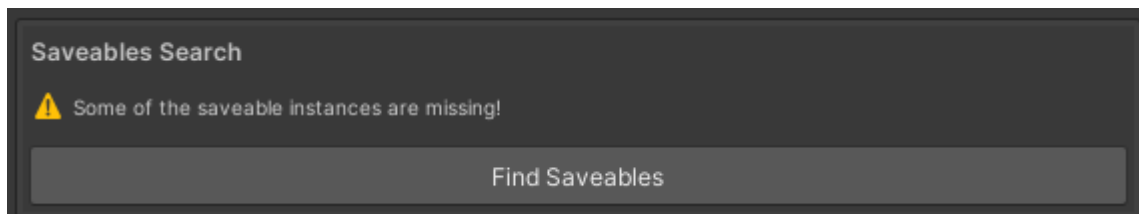
SAVING GAME DATA

- By clicking the **Find Saveables** button, the script will find all **Saveables** defined by the methods in the **Adding Custom Saveables** section.

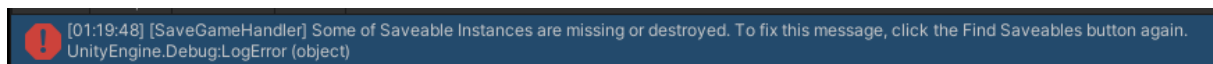


- To load a new scene and keep all inventory and equipped items, you must use the **LoadNextScene()** function located in the **HFPS_GameManager** script.
- There is a script called **NextLevelTrigger** that you can use to load next scene without losing the inventory or equipment by going on a trigger.

SAVEGAMEHANDLER ERROR MESSAGES



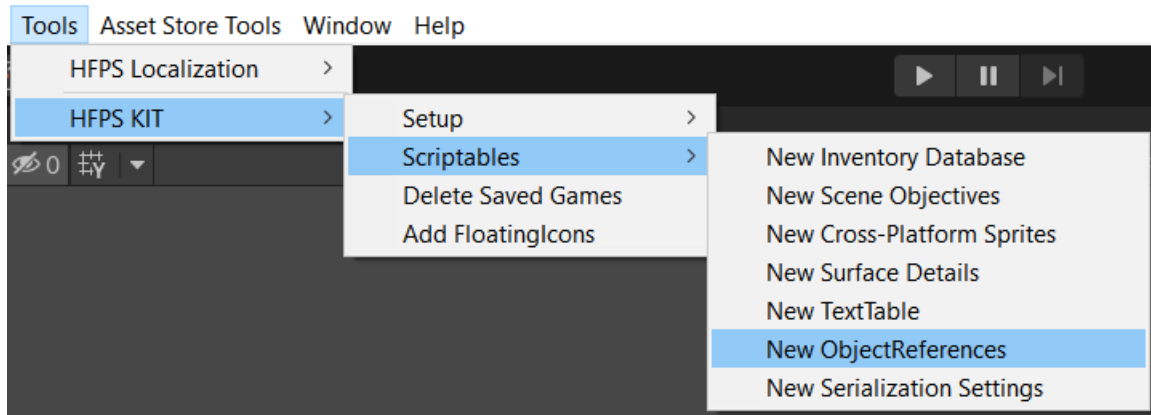
- The referenced **Saveable** has been removed from the scene.
- You must click **Find Saveables** again to clear this message.



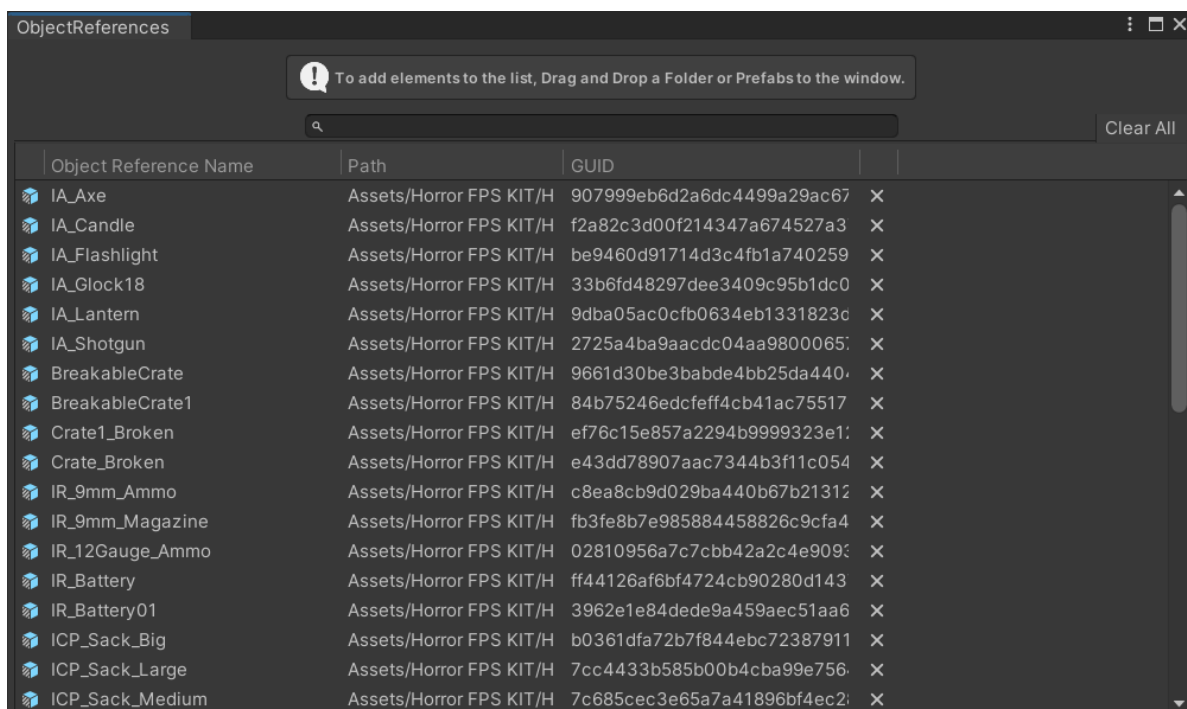
- The script loaded a **Saveable** that was saved in a file but was removed from the scene.
- You must click **Find Saveables** again to clear this message.
- If the **Force Save Load** option is turned off, loading will be prevented if this error message is shown.

ADDING RUNTIME SAVEABLES

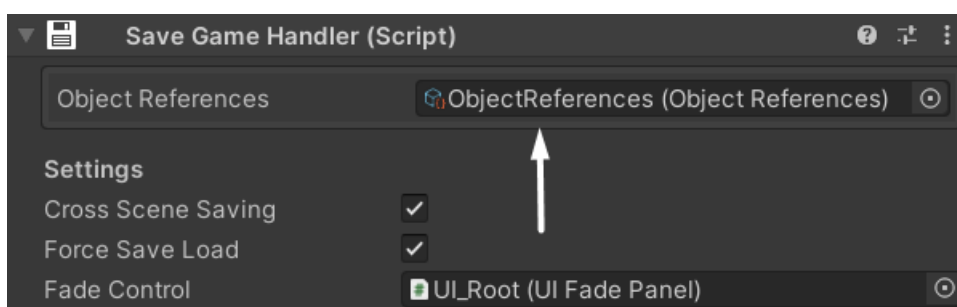
1. Create or open an existing **Object References** asset.



2. Open **Object References Window** and add instantiable prefabs by dragging folders or prefabs to the window.



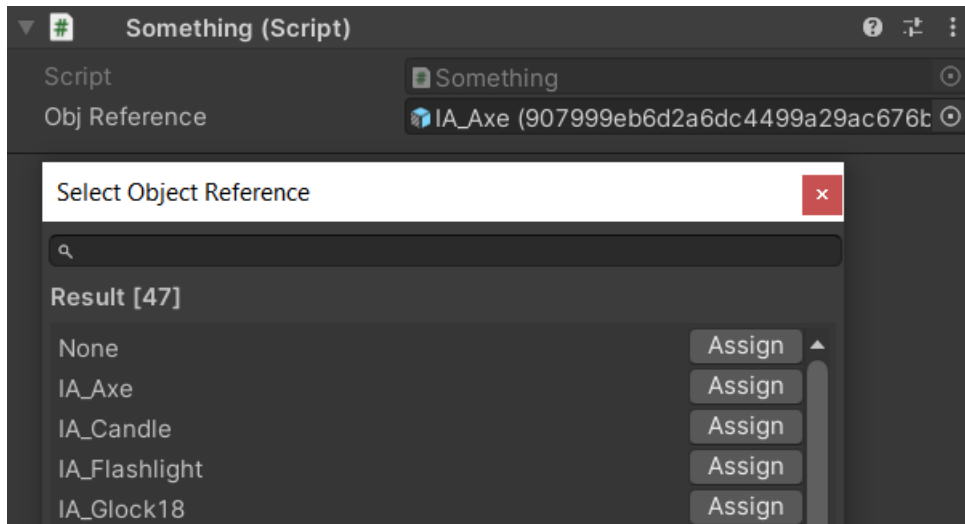
3. Assign **Object References** asset to **SaveGameHandler** script. This will allow you to open the object reference selection tool.



4. Add the **ObjectReference** variable.

```
1. using HFPS.System;
2.
3. public class Something : MonoBehaviour
4. {
5.     public ObjectReference ObjReference;
6. }
```

5. Assign **Object Reference** object by opening the **Object Reference Selection Tool**.



- This variable allows you to select the **Object Reference** defined in the first steps.
6. After selecting Object Reference, you can instantiate it at runtime using the **InstantiateSaveable()** function located in the **SaveGameHandler** script.

```
1. void Start()
2. {
3.     SaveGameHandler handler = SaveGameHandler.Instance;
4.     GameObject go = handler.InstantiateSaveable(ObjectReference, position, rotation);
5.
6.     //go represents the instantiated object
7. }
```

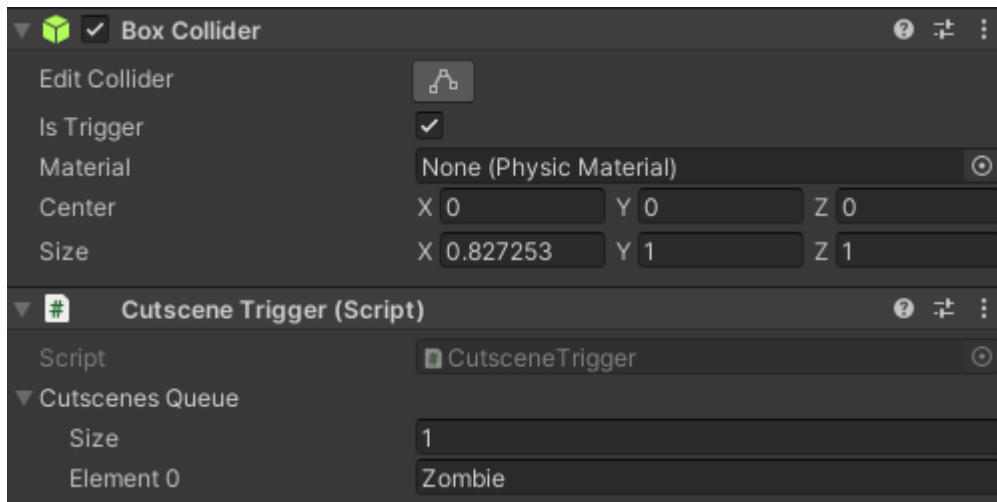
- Using the Object References we can store a specific GUID of the asset which we can use to retrieve an object that was created at runtime.

ADDING NEW CUTSCENES

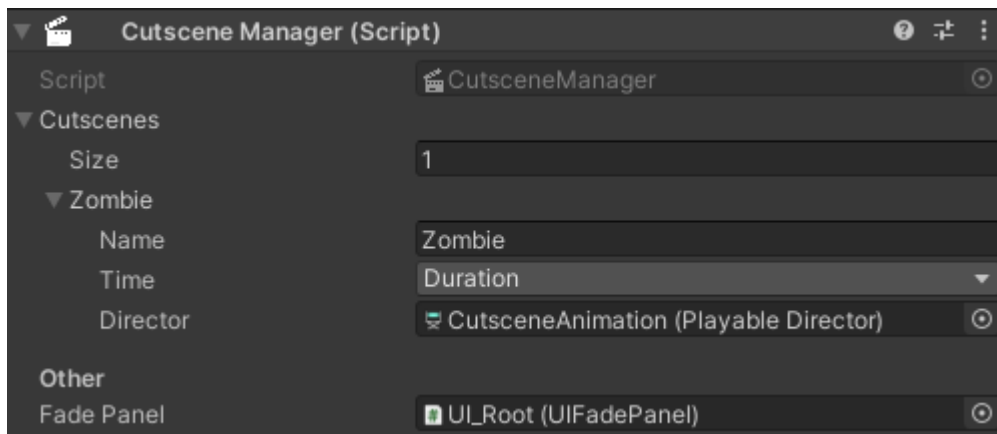
1. Create a new **Timeline Animation**. You can follow the tutorial from **Brackeys**.

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

2. Create a **Cutscene Trigger**.



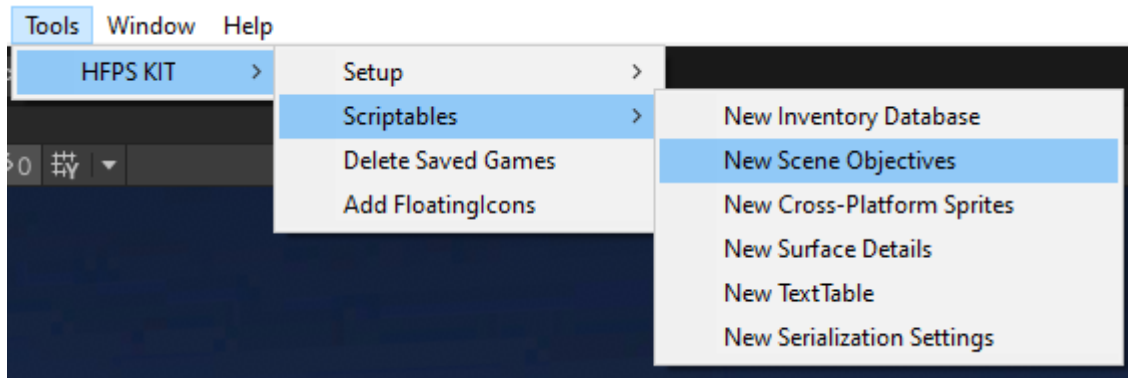
- You can **queue** the cutscenes by adding more **Elements**.
3. Assign your new **Cutscenes** to the **Cutscene Manager** script.



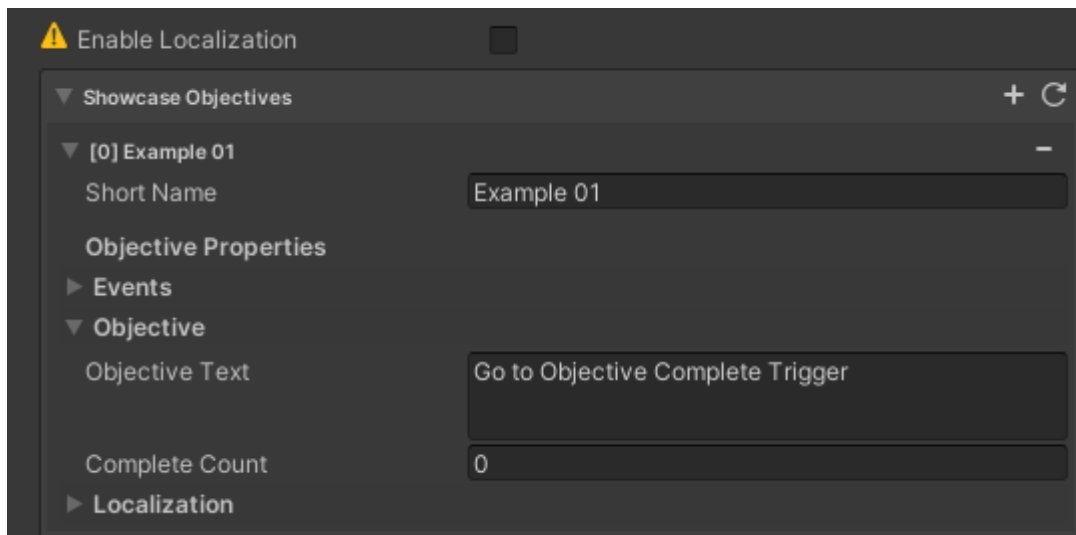
- If you set the **Time** parameter to **Manual**, you must manually skip or cancel the cutscene sequence by invoking the **Cutscene Manager -> SkipCutscene()** or **AbortCutscenes()**.

ADDING NEW OBJECTIVES

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

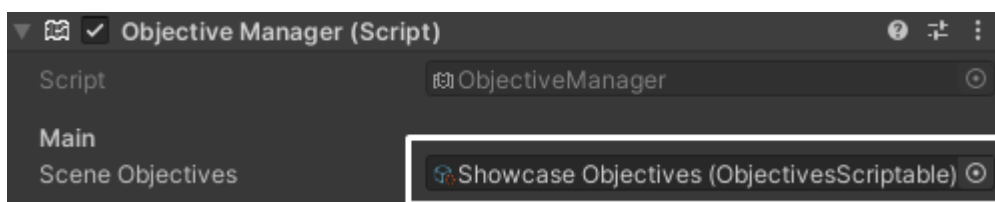


2. Clicking the + button you will add new objective to the **Scene Objectives** asset.

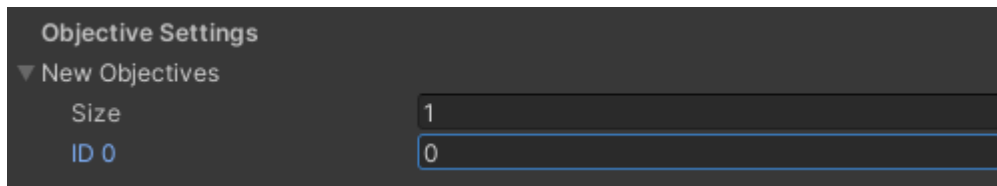


- If you reorder the objectives, clicking the refresh button will refresh the objectives IDs.
- Each objective has a unique ID that allows you to easily define which objective you want to run. Unique is a number placed before the name of the objectives: **[0]** Example 01.
- As you may have noticed, there are settings that allows localization. By clicking on the **Enable Localization** button, the system will allow you to select the localization key, however the **HFPS Localization System** package is required, otherwise the localization will not work!

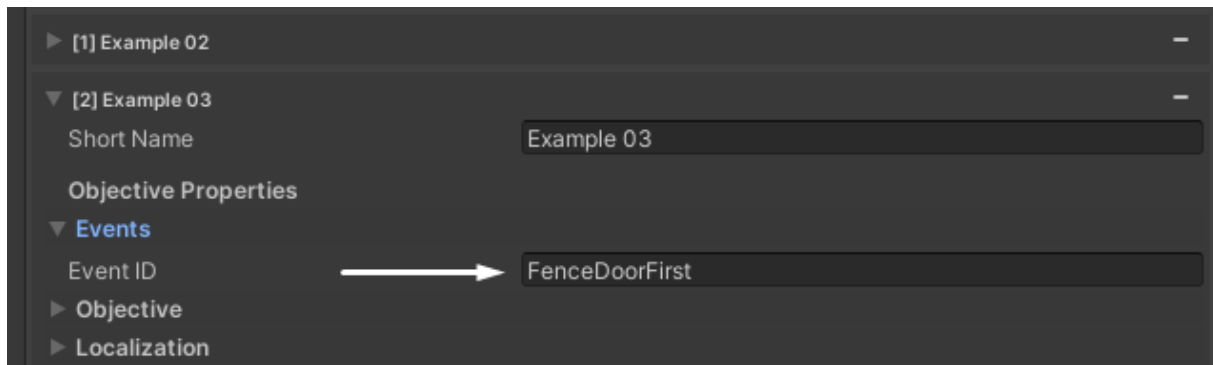
3. Remember to always assign objectives of your scene in the **Objectives Manager** script.



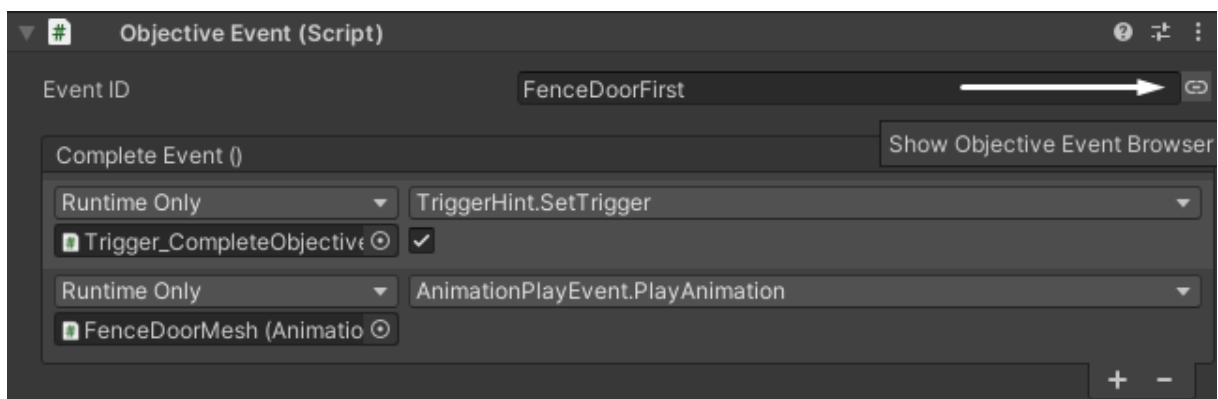
4. Create a new trigger, add **TriggerObjective.cs** script and set ID to objective which you want to run.



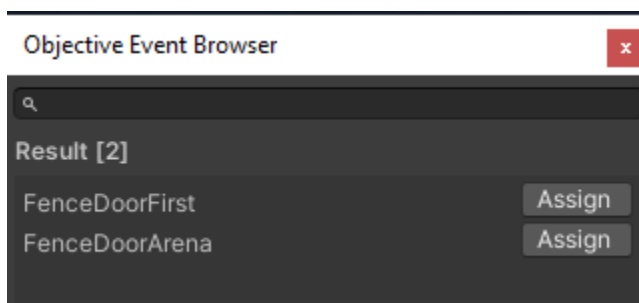
5. The Scene Objectives asset also allows you to define an event that will be called when the objective is completed.



6. After defining an event, you must add the Objective Event script to an empty GameObject and select the event you want to use.



7. Clicking the **Link** button, you will display the **Objective Event Browser Window**.

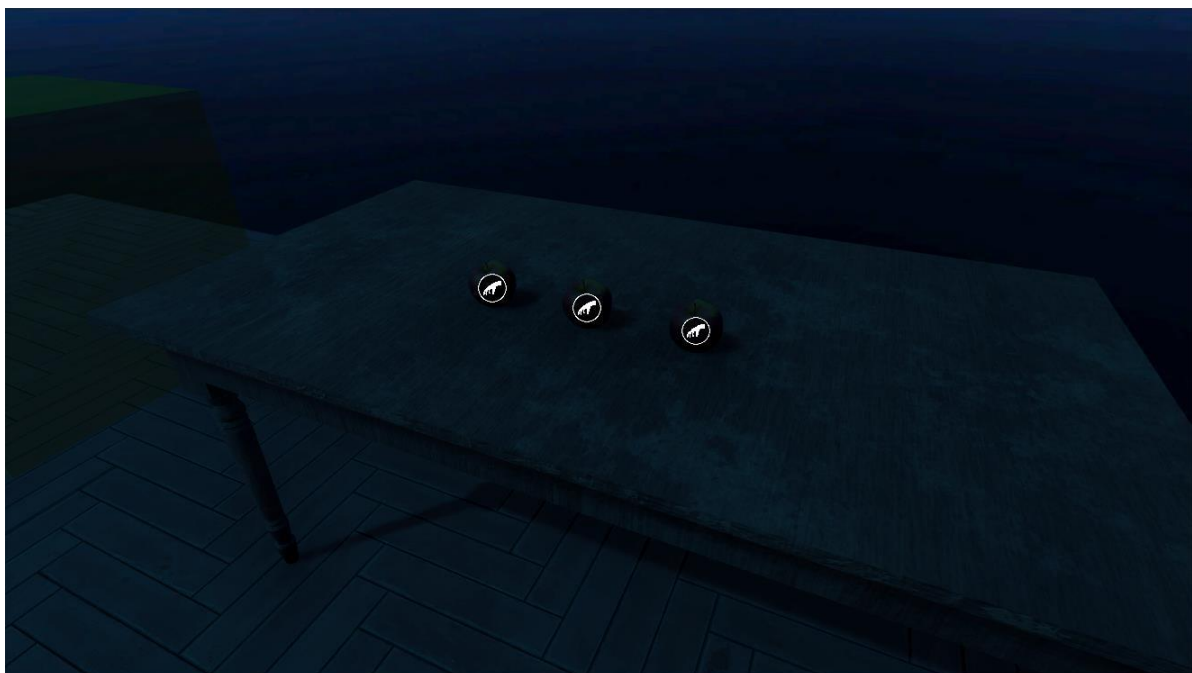
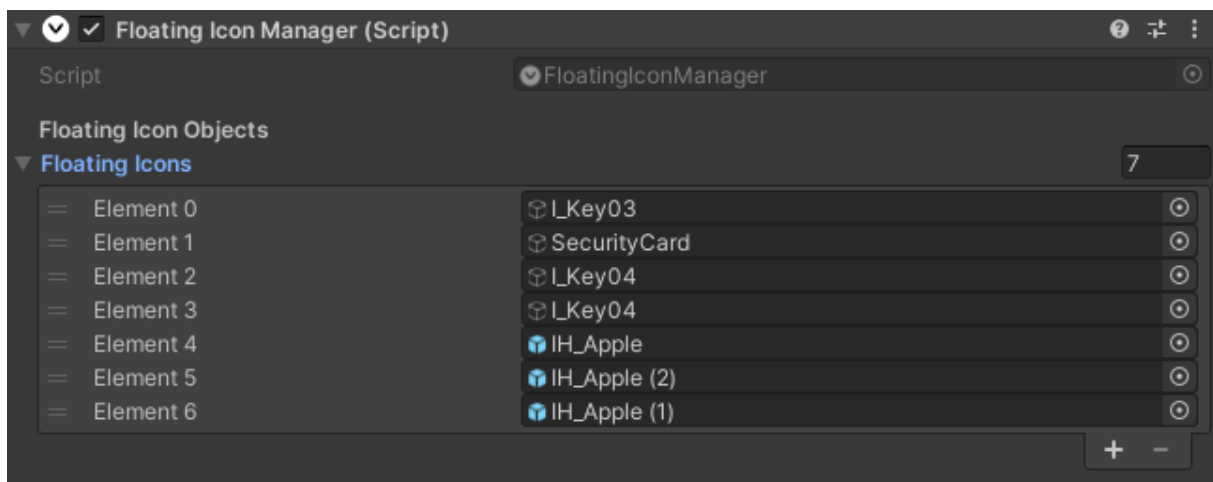
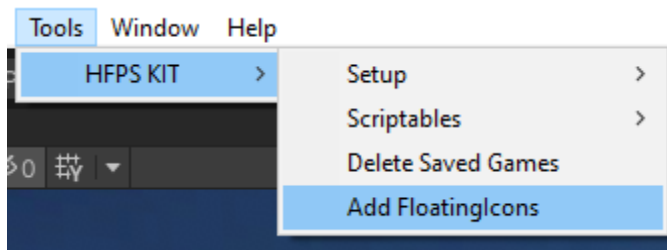


Complete Count: The objective will be fully completed if you complete the objective n-times.

To show progress to the objective, insert "{0}/{1}" to the objective text.

ADDING NEW FLOATING OBJECTS

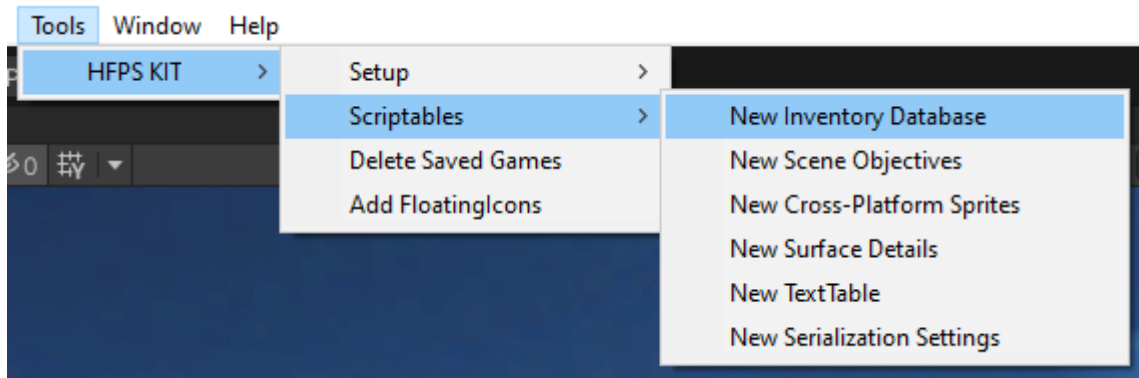
1. When you select all the **GameObjects** you want to define to display the floating icon above object, pressing the **Add FloatingIcons** button will automatically add the objects to the list.



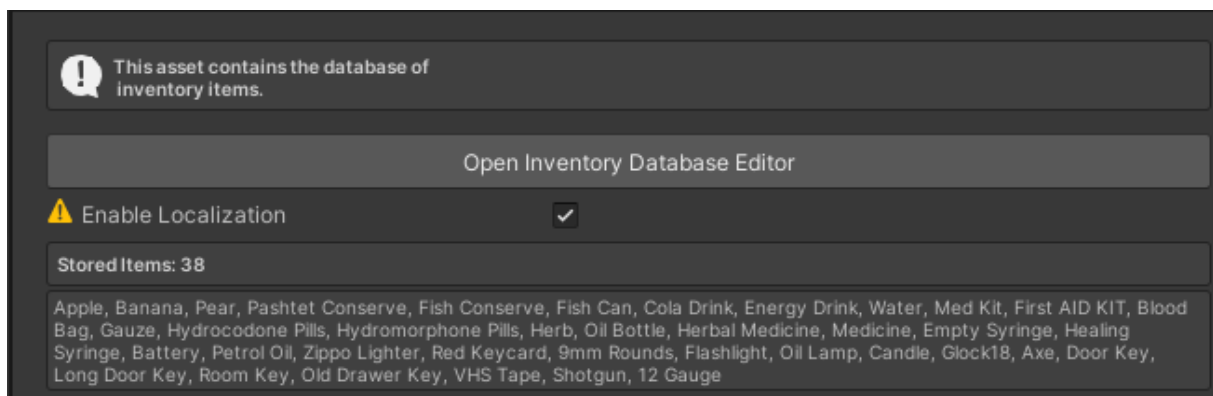
INVENTORY

ADDING NEW INVENTORY ITEMS

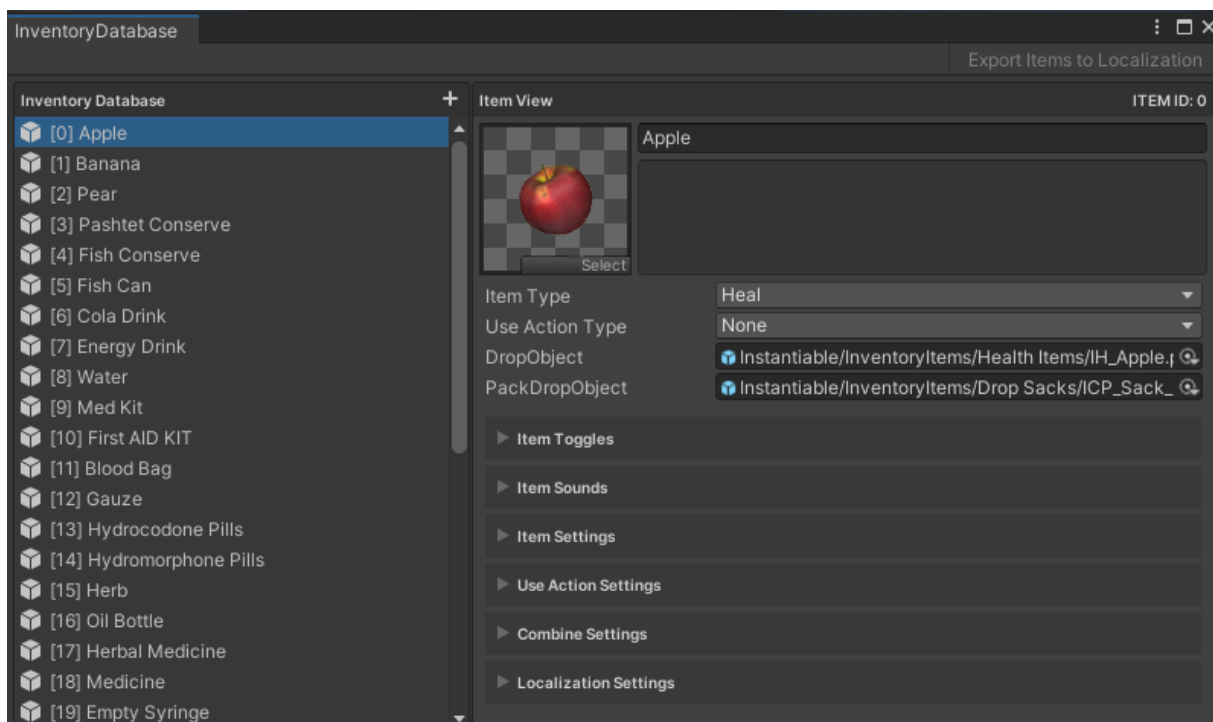
1. Create or Open an existing **Inventory Database** asset.



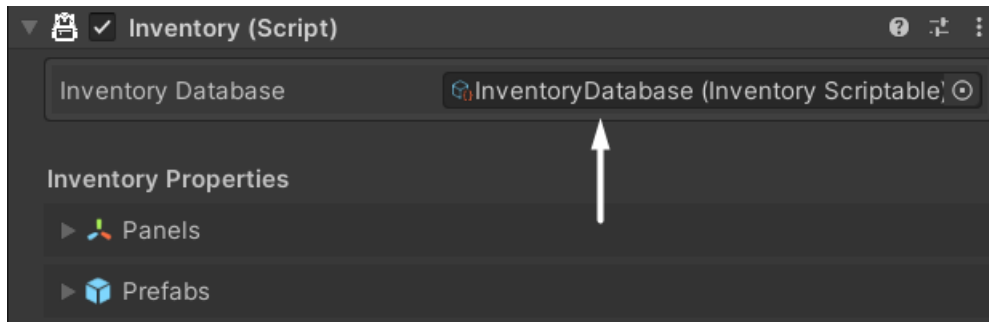
2. To open the **Inventory Database Window**, double-click the asset or click the **Open Inventory Database Editor** button.



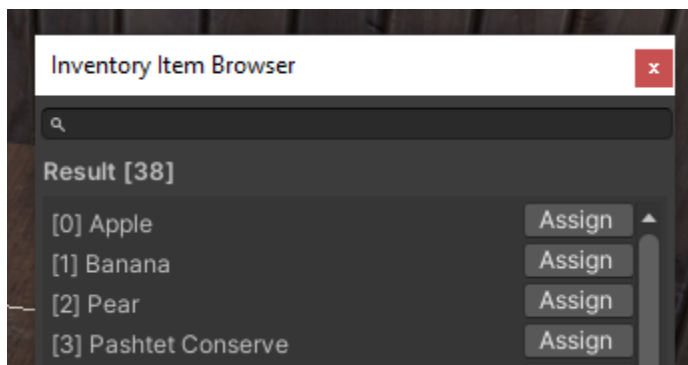
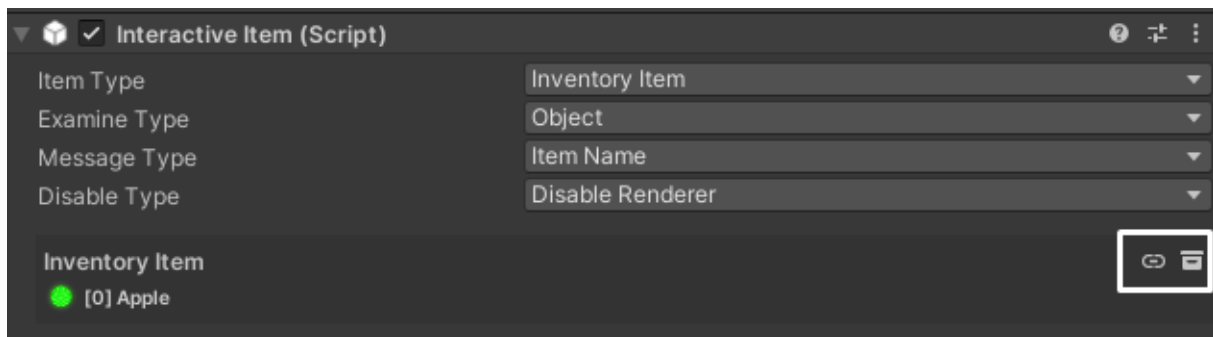
- This will show up the window where you can manage all inventory items.



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



4. To interact with a newly created item, add the **Interactiveltem.cs** script to the object and select the item you want to use in the **Inventory Item Browser**.



- 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.

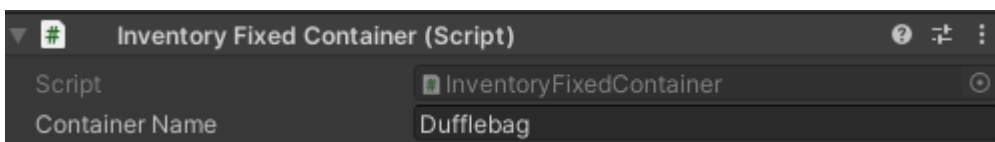


INVENTORY CONTAINERS

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

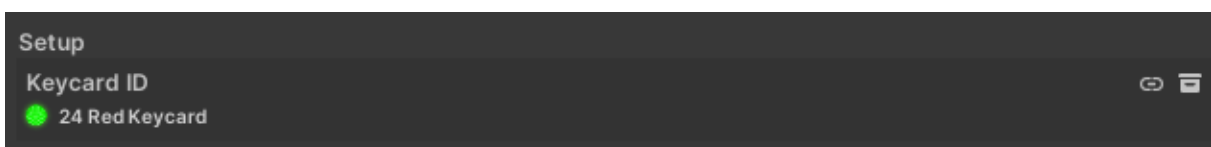


- To store an **Item** to a **Container**, simply add the **InventoryContainer.cs** script to it.
- You can also define **Fixed Containers** that can contain the same items as another container by adding the **InventoryFixedContainer.cs** script.



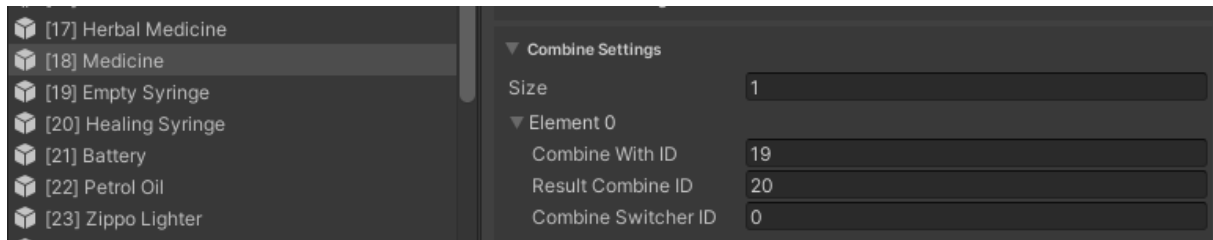
- To convert an int field to an Inventory Selector field, you can use the **[InventorySelector]** attribute in the int field.

```
1. [InventorySelector]
2. public int InventoryID;
```



COMBINABLE ITEMS

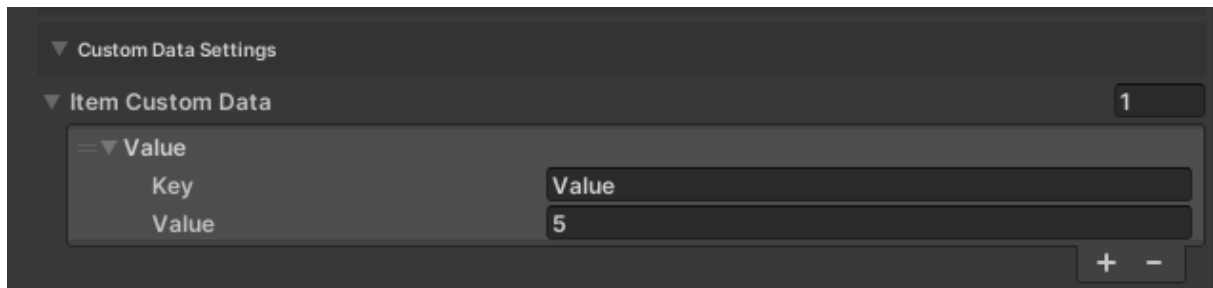
- Inventory has a feature that allows to combine two different objects to get another useful item.
- This can be done by defining the **Combine Settings** in the Inventory Database Editor.



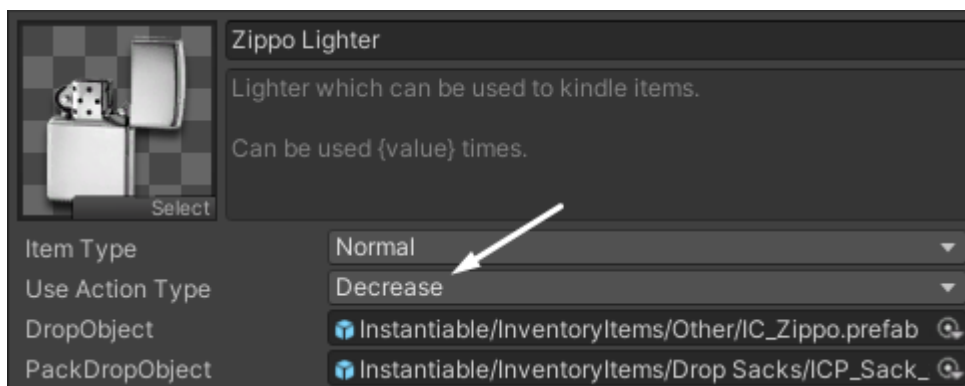
INVENTORY CUSTOM ACTIONS

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

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

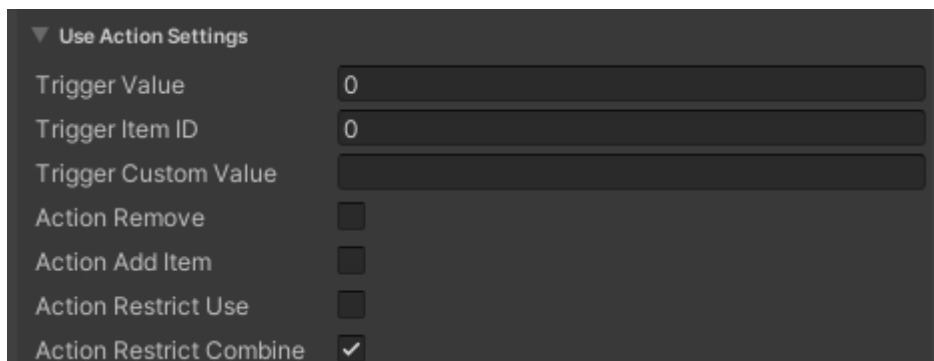


- **Predefined Keys:** Value, Path, Tag
2. Define **Use Action Type** in the Item Settings.



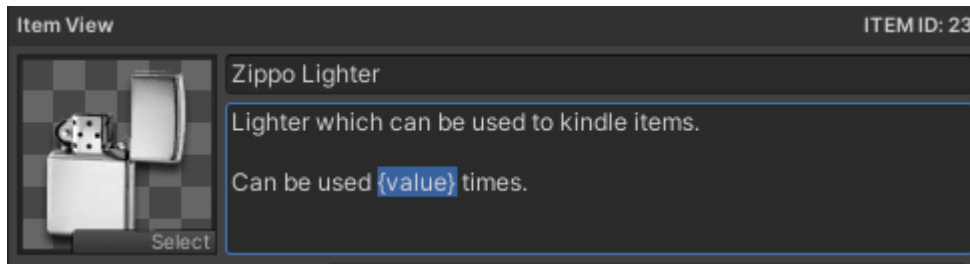
- We set **Use Action Type** to **Decrease** to decrease the value each time zippo is combined with a candle.

3. Define **Use Action Settings**



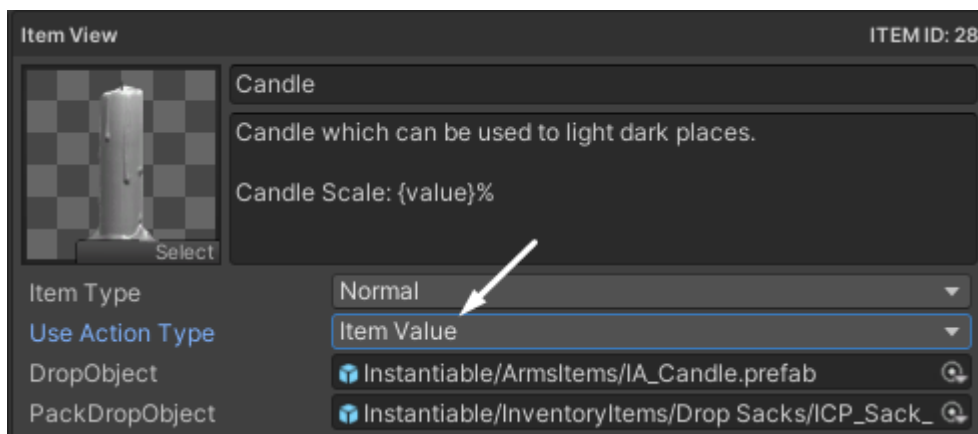
- We have now defined that when the Zippo Lighter is used 5 times, you will not be able to light the candle again.

- 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 want to use the **Item Value** action type, you can follow the same steps, but this action type is specific in that you can set the value manually by script using the **ItemValueProvider** interface.

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



2. Add **ItemValueProvider** interface and its functions.

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

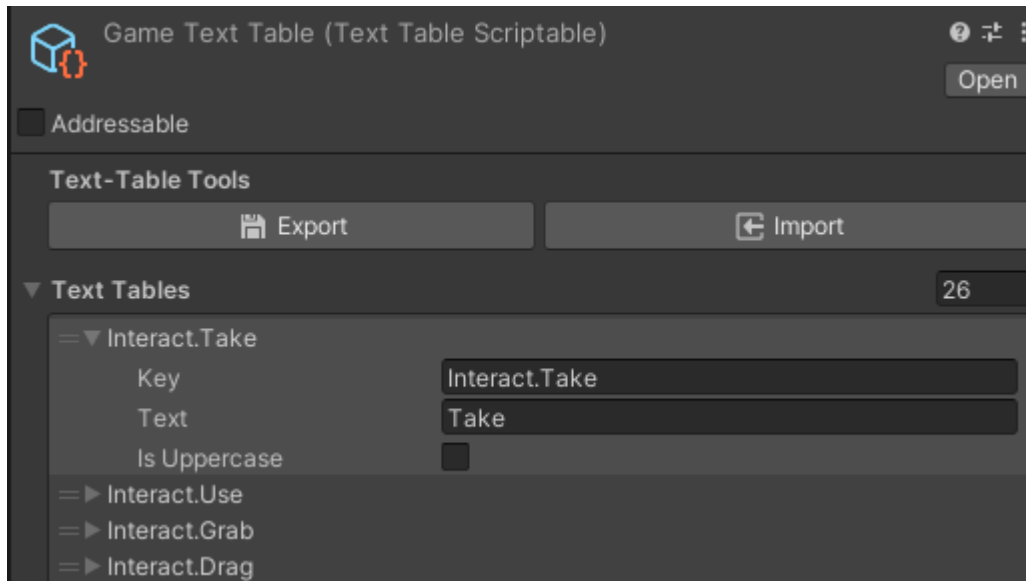
```
    public string OnGetValue()
    {
        throw new NotImplementedException();
    }

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

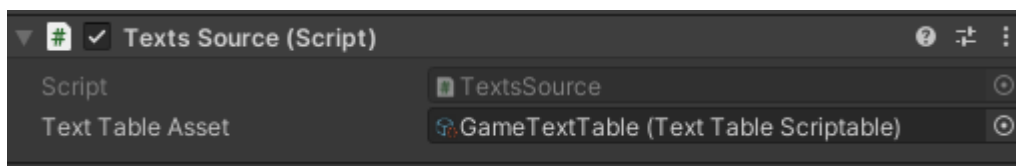
- In the Item Toggles drop-down menu, we set the action to be applied when the item is combined. When the item is combined, `OnSetValue ()` is called.
- OnGetValue()** is different, you must update the value manually using **Inventory.GetItemData().data** or setting the custom data in **AddItem()** function.

CHANGING SCRIPTS UI TEXTS

- To change the UI script texts, find the **GameTextTable** asset.



- From there, you can change or create new user interface texts, which you want to use in scripts.

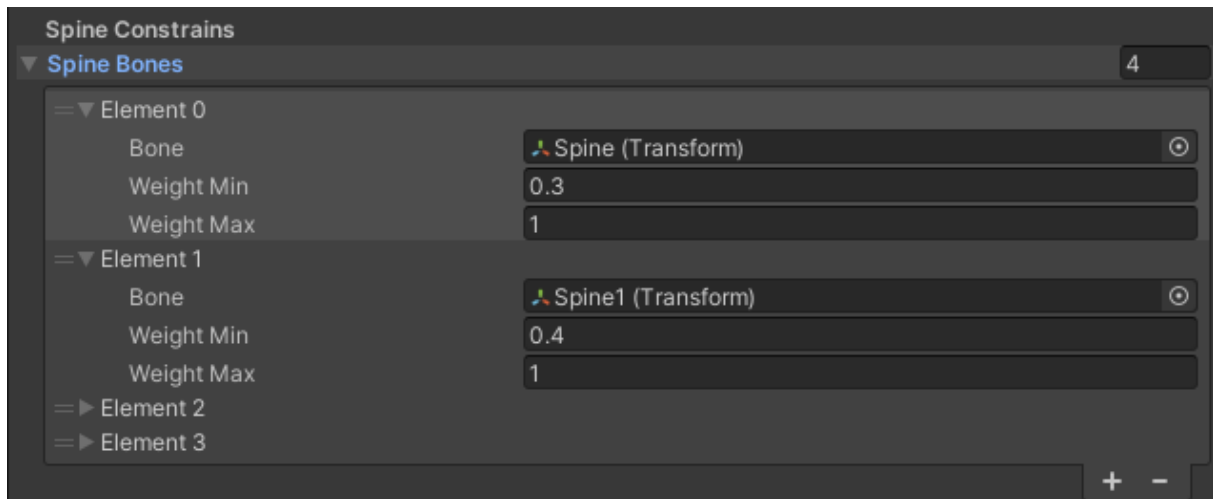


- The newly created **TextTable** asset can be assigned in the **Text Source** script.
- You can easily use the TextsSource functions to retrieve text from a text table.
- The following code shows a basic example of retrieving texts.

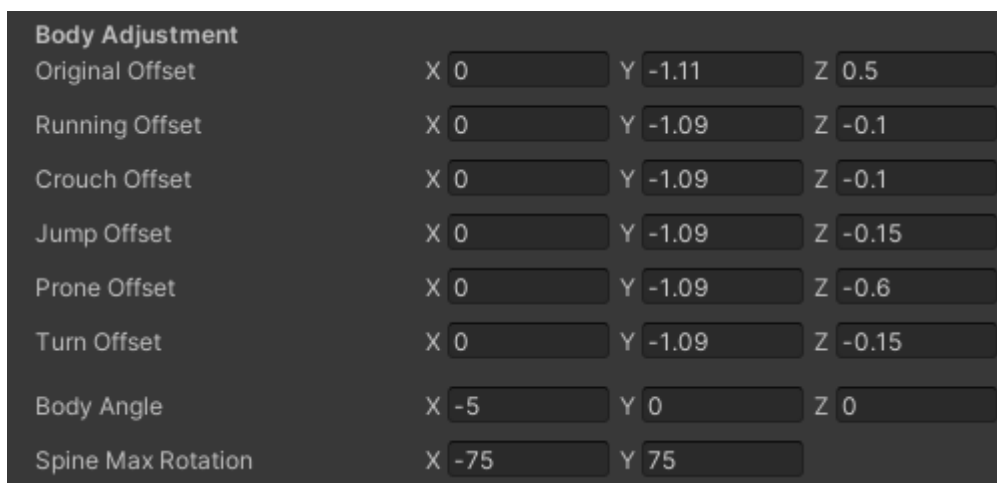
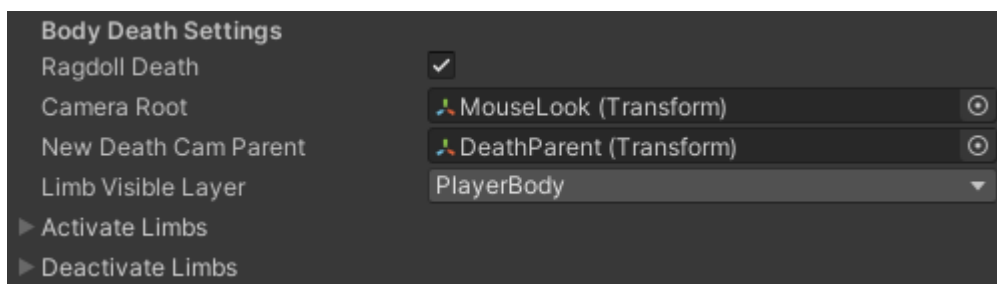
```
1. using HFPS.Systems;
2.
3. void Awake()
4. {
5.     TextsSource.Subscribe(OnInitTexts);
6.     //or TextsSource.OnInitTexts += OnInitTexts;
7. }
8.
9. private void OnInitTexts()
10. {
11.     string text = TextsSource.GetText("Interact.Take", "Take");
12.     Debug.Log(text);
13. }
```

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.



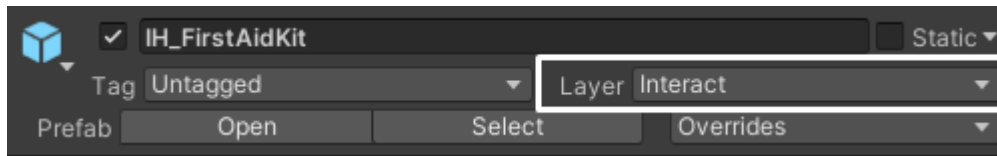
- Using the **Spine Bones**, you can easily define the entire spine and its min/max rotation by adjusting the weights.



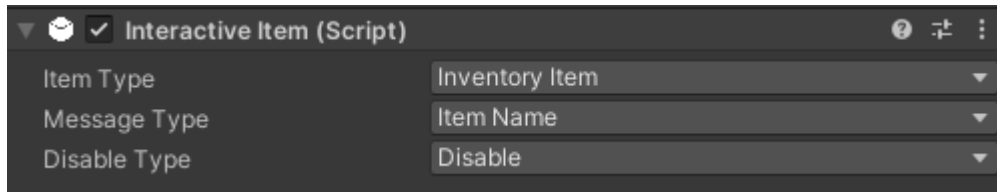
- You can adjust **Body Death**, **Body Adjustments** and **Other** settings depending to your needs.

ADDING NEW INTERACTIVE ITEMS

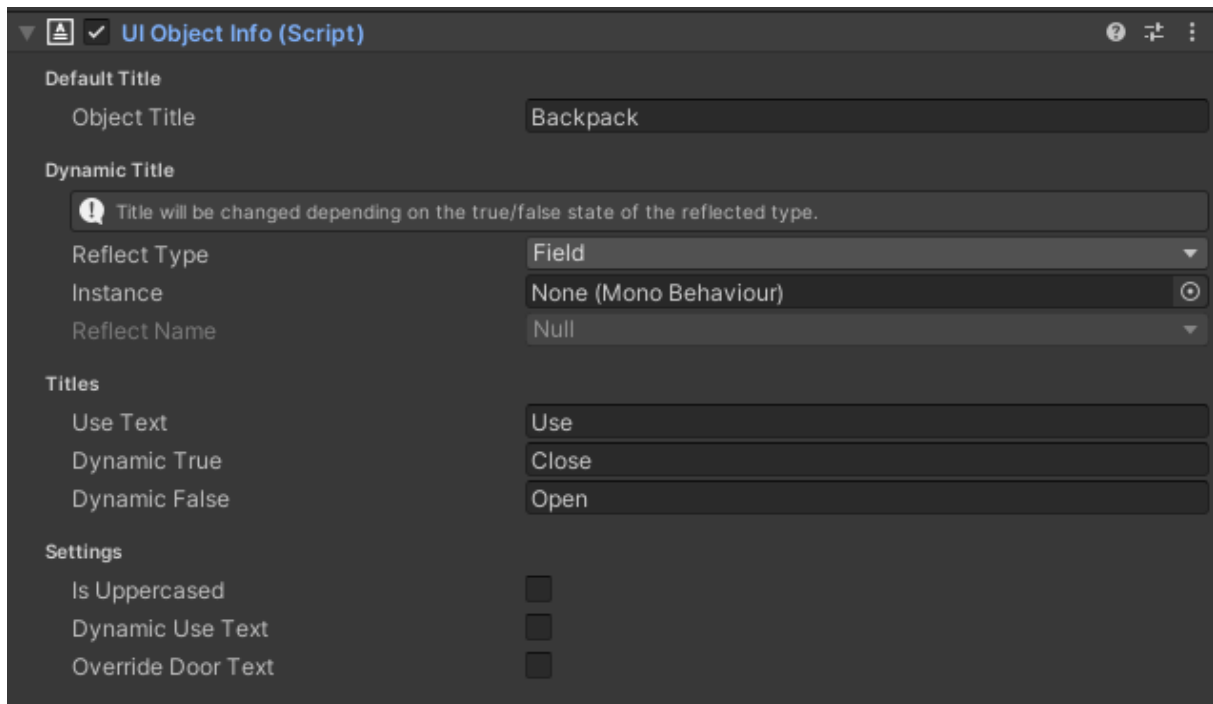
1. Set the object layer to **Interact**.



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



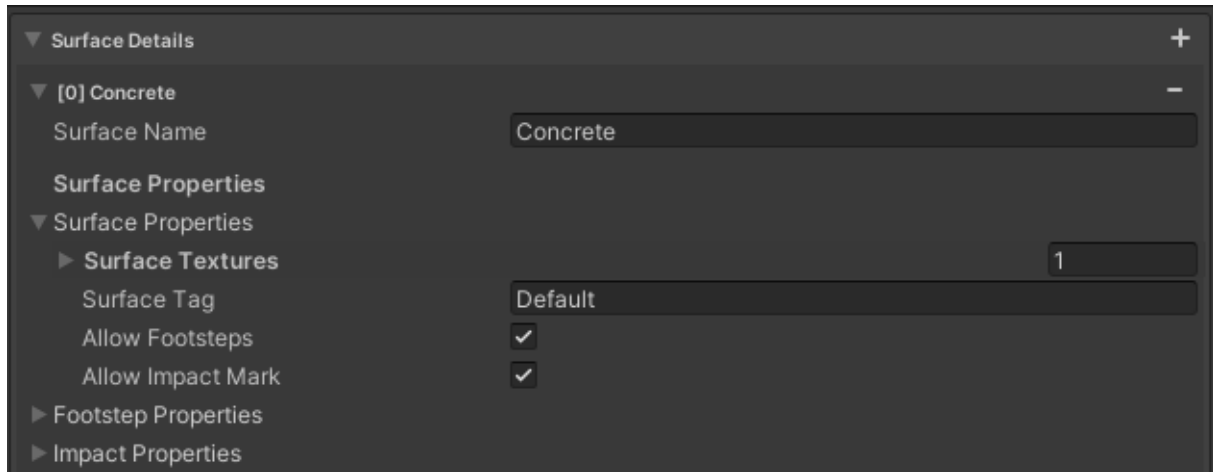
3. To change the user interface texts, add the **UIObjectInfo.cs** script.



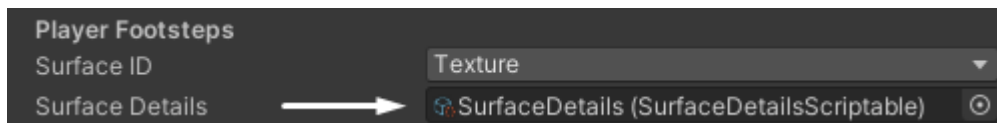
- To change interaction titles depending on another script field, assign **Instance** and **Reflect Name**.
- To create your own interactive object with your own script, create a public **UseObject()** function.

SURFACE DETAILS

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



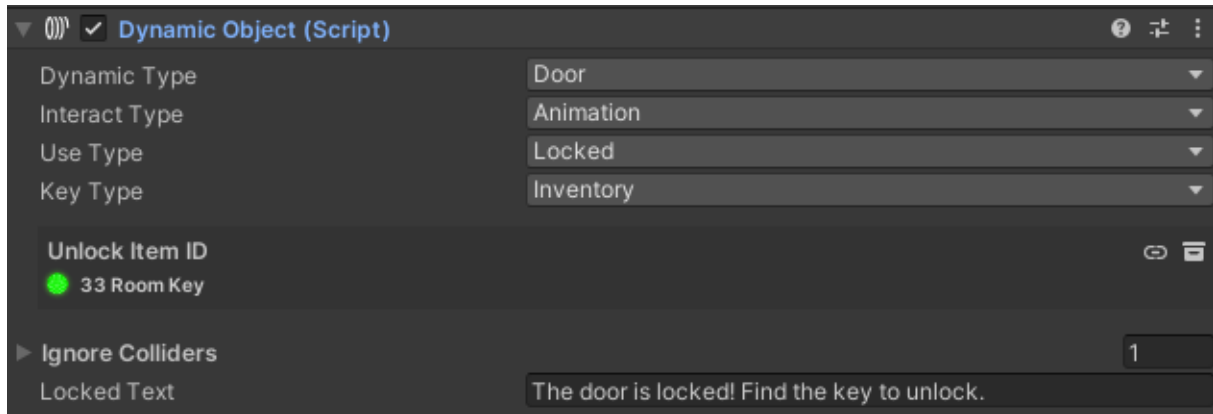
- You can use **Surface Textures** to define different footsteps or bullet marks for different textures.
- **Surface Details** also support terrain textures, so you can easily define each surface of the game.



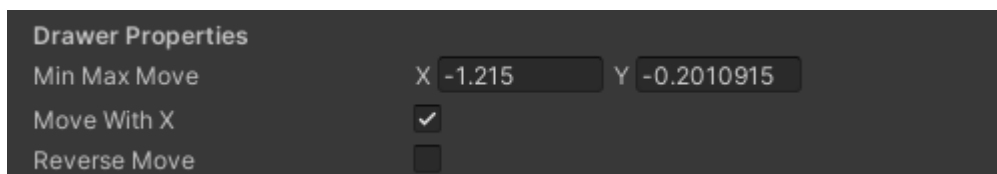
- Using surface details is very simple, just move your surface data into a script.
- To use classic surface detection, change the **Surface ID** to **Tag**.
- The **Surface Details Asset** also contains all the required functions, so obtaining elements is very easy.
- Each function of **Surface Details**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

DYNAMIC OBJECTS

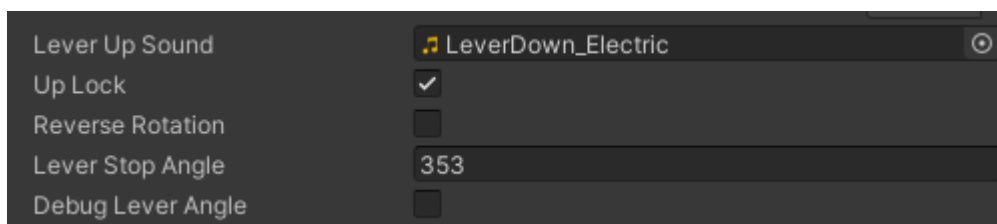
- Using a **DynamicObject.cs** script, you can easily define interactable **Doors**, **Drawers**, **Levers**, **Valves**, or **Moving Interactions**.



- 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**.



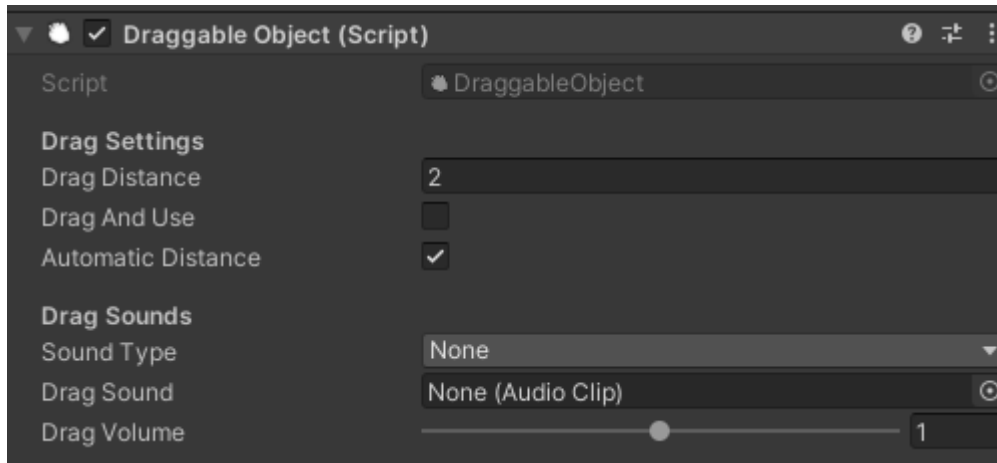
- By defining a new dynamic **Drawer** type with an interact **Mouse** type, you must assign a minimum and maximum pull position by changing the **Min Max Move** field.



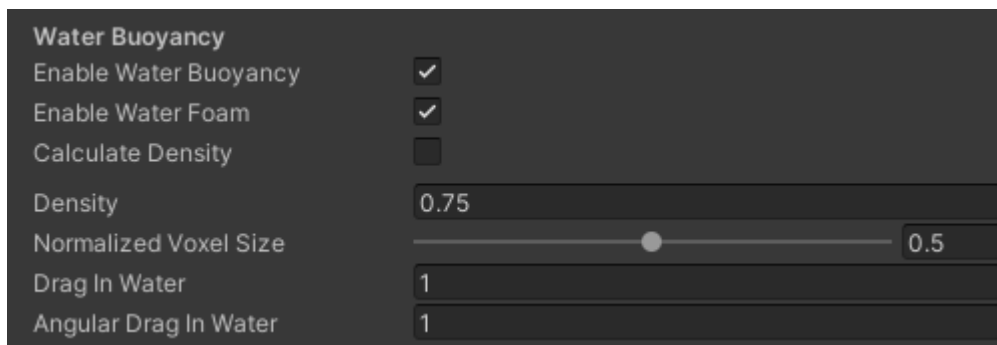
- The same must be done with the dynamic **Lever** type by changing the **Lever Stop Angle** field. If you have trouble defining the stop angle, check the **Debug Lever Angle**.

DRAGGABLE OBJECTS

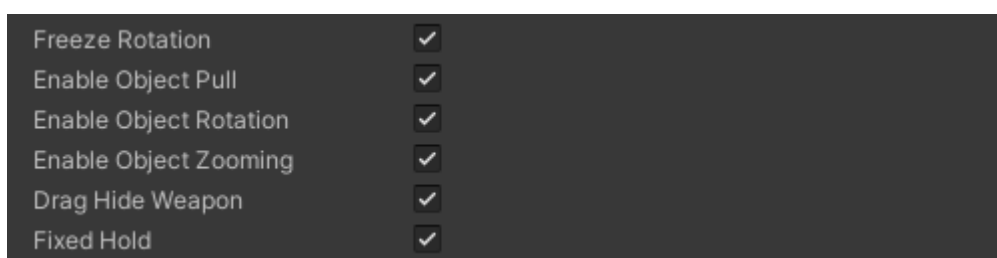
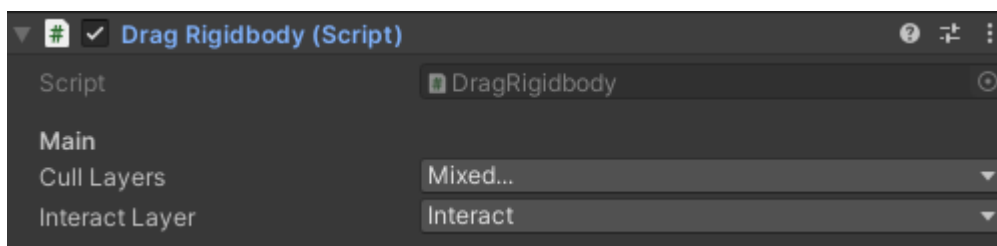
- Defining draggable objects is very simple, just change the object layer to **Interact** and add the **DraggableObject.cs** script.



- 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.

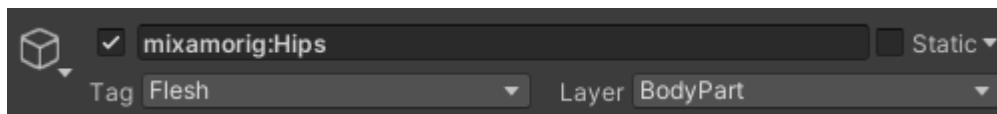


ADDING NEW ZOMBIE AI

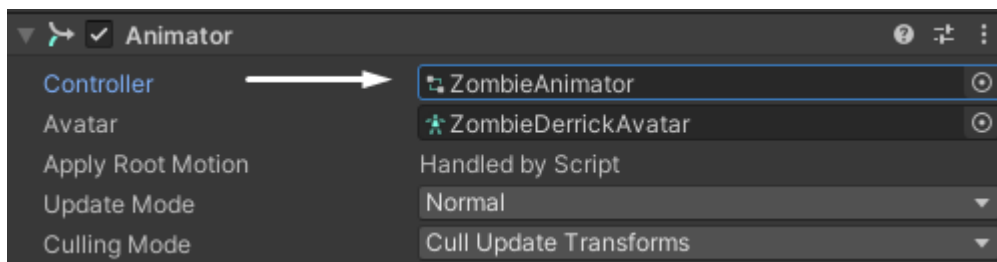
1. Convert your **Zombie Character** to a **Ragdoll** (GameObject -> 3D Object -> Ragdoll).
2. Adjust the ragdoll colliders to match your character model.
3. Change the zombie root layer to **Zombie**.



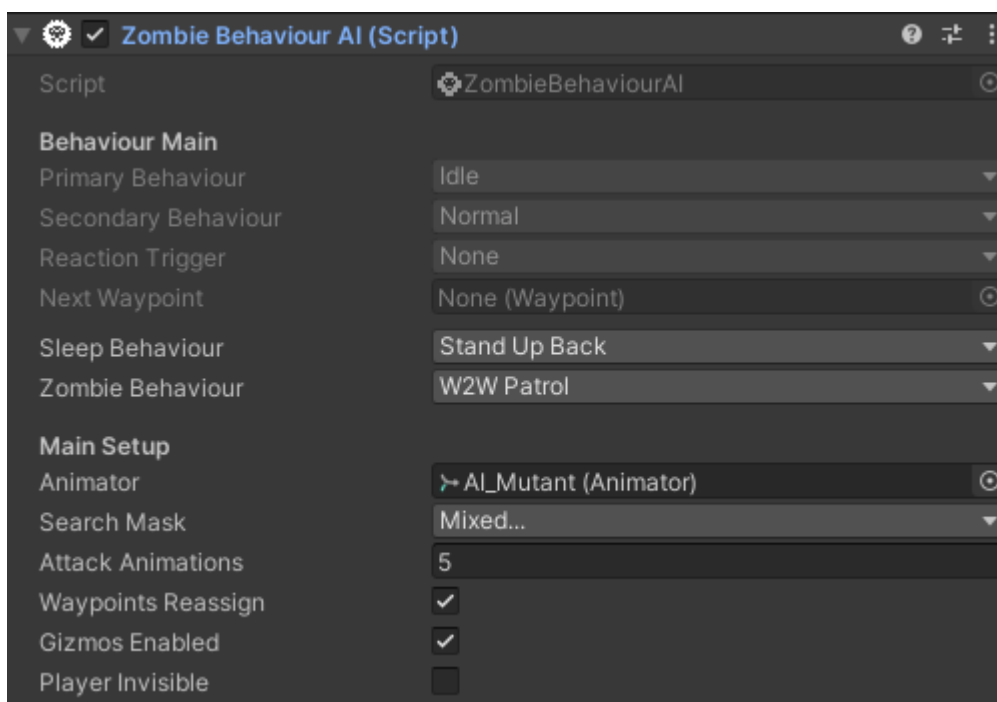
4. Then change the zombie hips layer to **BodyPart** and tag to **Flesh**.



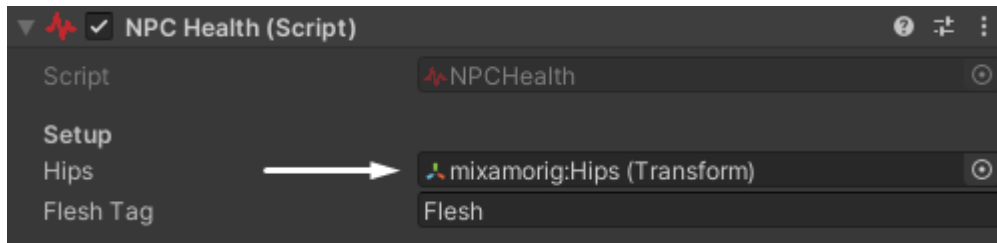
5. Assign an **Animator Controller Asset** to an **Animator** component.
 - It is recommended to use the default **ZombieAnimator Controller** as it contains all required connections and behaviour scripts.



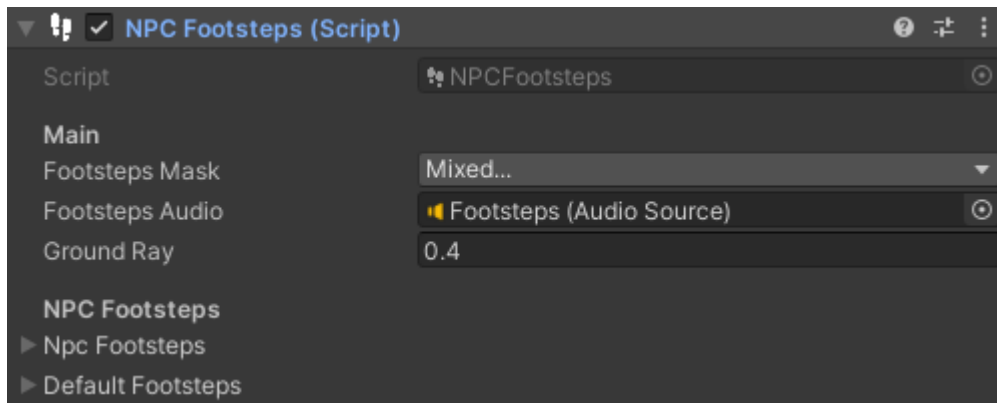
6. Add **ZombieBehaviourAI.cs** script to zombie root.



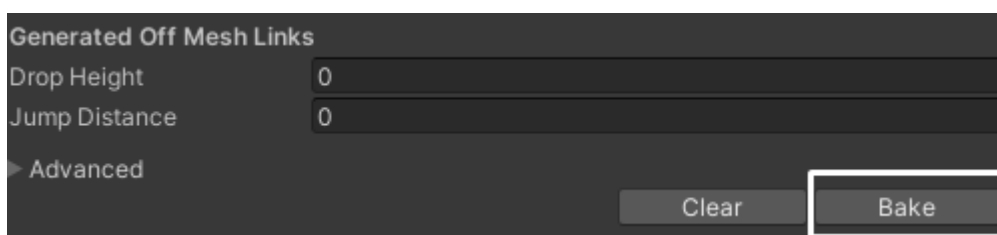
7. Add **Capsule Collider** component to the zombie root and adjust the collider to match your character model.
8. Assign **Zombie Animator** component to an **Animator** field and define a **Search Mask**.
9. Add the **NPCHealth.cs** script to the zombie root and assign zombie hips.



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

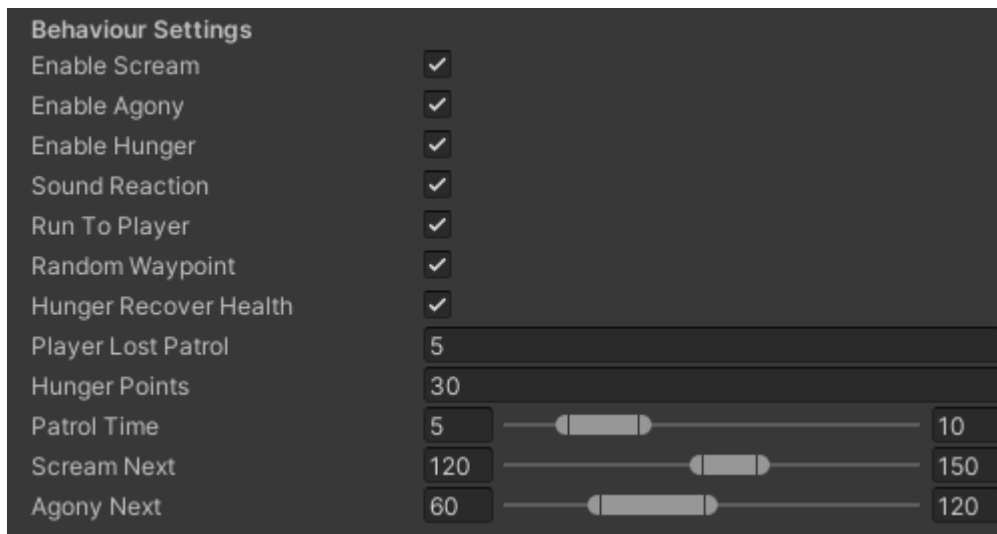


11. Create a new empty object, add an **Audio Source** component, change **Spatial Blend** to **3D** and move the object as the zombie root child object.
12. Add a **Nav Mesh Agent** component and adjust it to fit your character's model. Make sure that the **Stopping Distance** is not zero and **Auto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

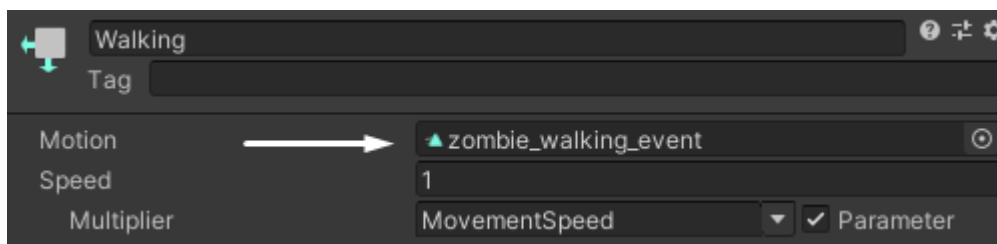


14. Add **WaypointGroup.cs** script to the empty object.
 - Waypoints are added automatically by adding empty objects to the **Waypoint Group** child object.

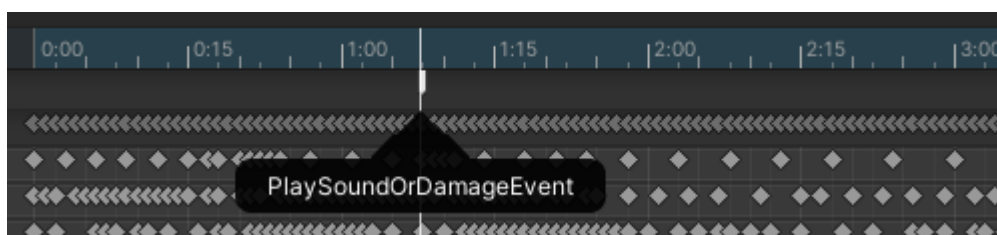
- You can define your own character behaviour by changing the **Behaviour Settings**.



- 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.



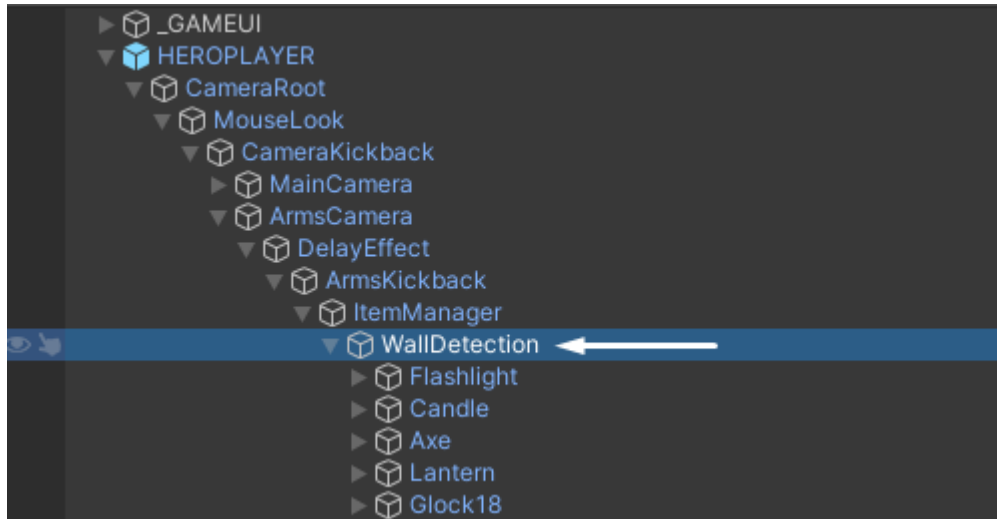
- To damage the player, you must create an animation event that calls the **PlaySoundOrDamageEvent()** function with a parameter of 0.

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

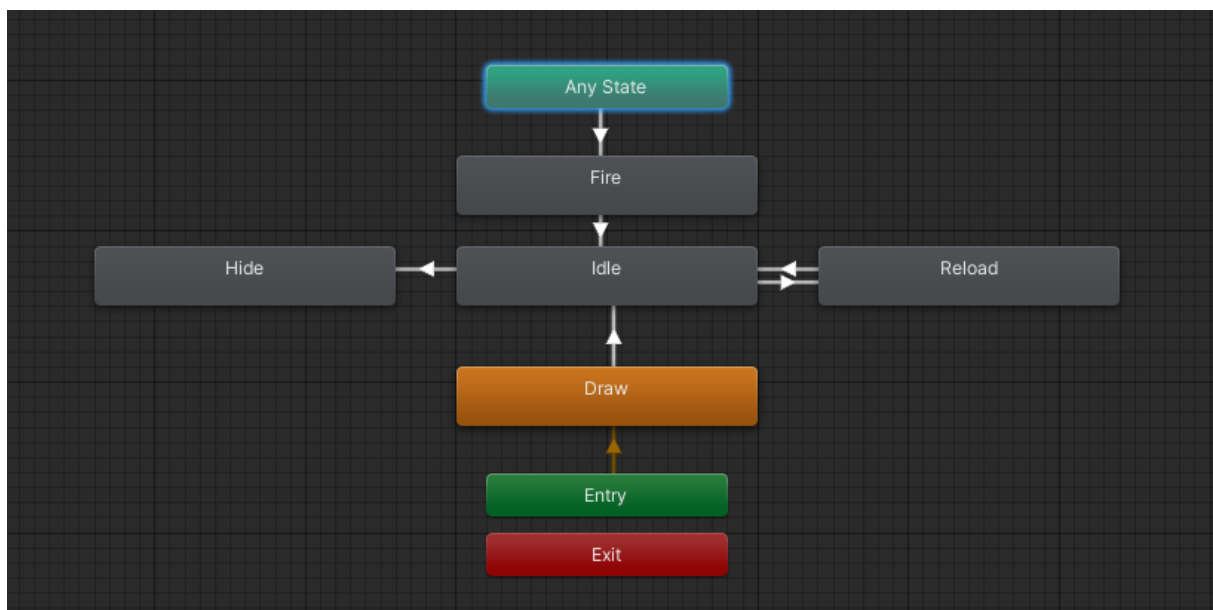
1. Locate the **WallDetection** object in the **PLAYER** object, which contains all the usable objects.



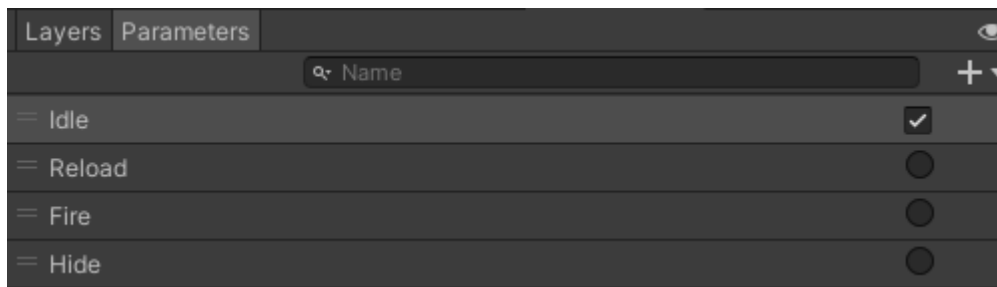
2. Duplicate one of these items and replace the disabled object inside the duplicated object with your object.
3. Set a new layer of object to the **CamWeapon**.



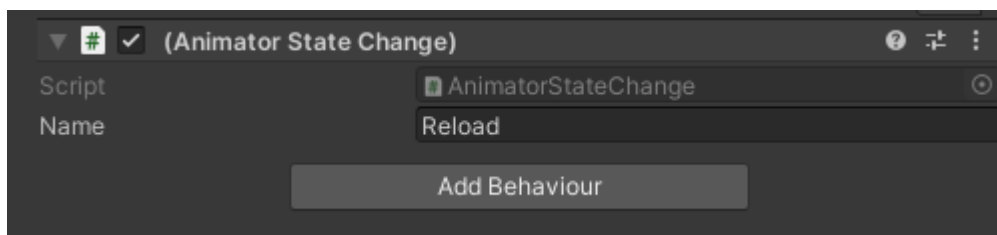
4. Create a weapon **Animator** asset.



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

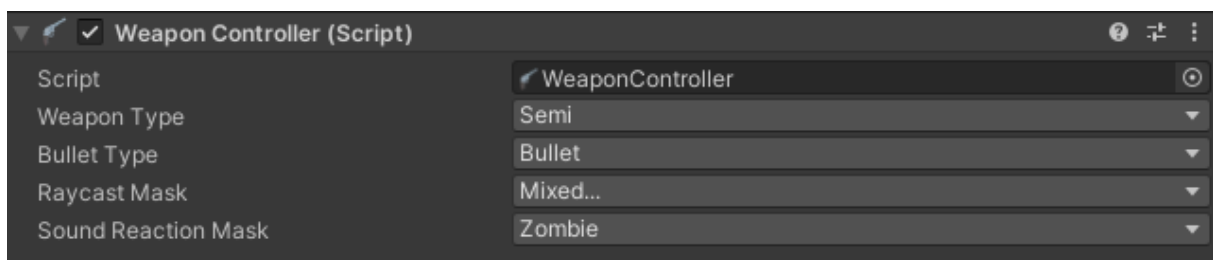


6. Create all required state transitions.
7. Add **AnimatorStateChange** behaviour to **Draw, Hide, Reload** states with the name **Draw, Hide, Reload** that the **WeaponController** script uses.

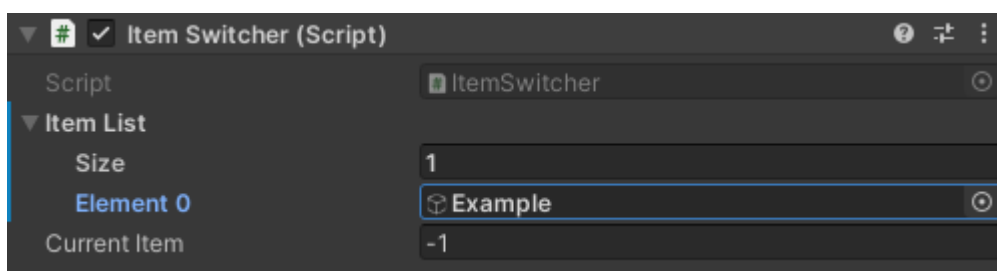


- This behavior script finds and sends state data to the **WeaponController** script using the **IOnAnimatorState** interface.
- It is important to write exactly **Draw, Hide or Reload** in the name field, depending on the state.

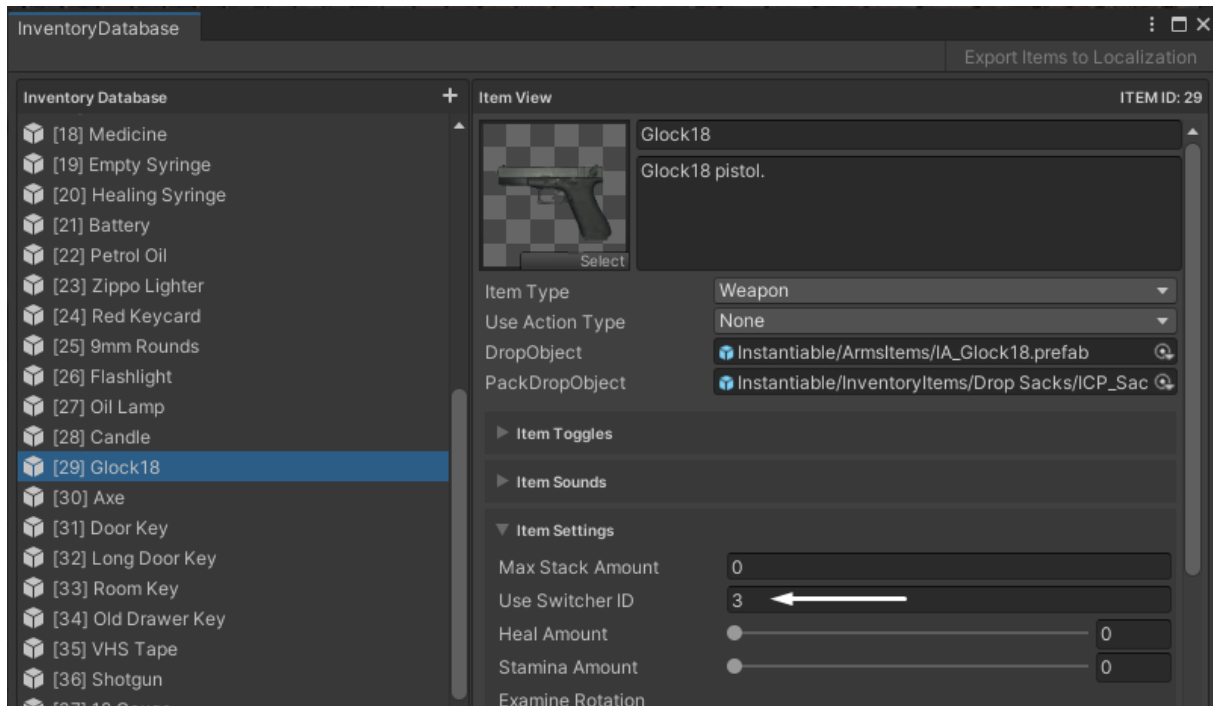
8. Add the **WeaponController.cs** script to the weapon root object and set the weapon parameters.



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.



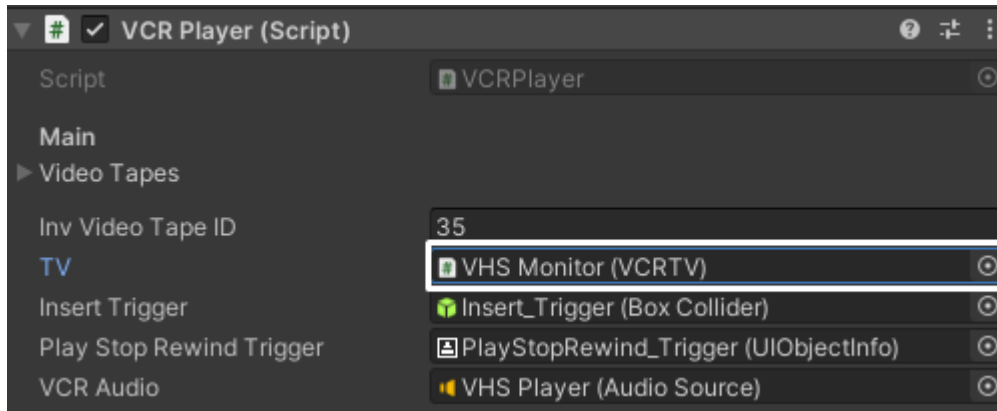
- You can also define your own usable weapons/items by adding the **SwitcherBehaviour** subclass to your own script.

```
Počet odkazů: 1
public class WeaponController : SwitcherBehaviour, ISaveableArmsItem, IOnAnimatorState
{
}
```

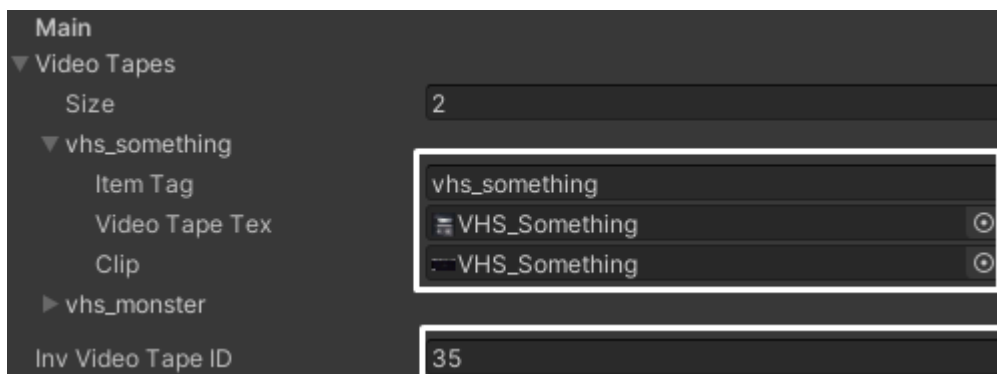
- You can add an **ISaveableArmsItem** interface to a script to save and load usable object data.

SETTING UP VHS PLAYER

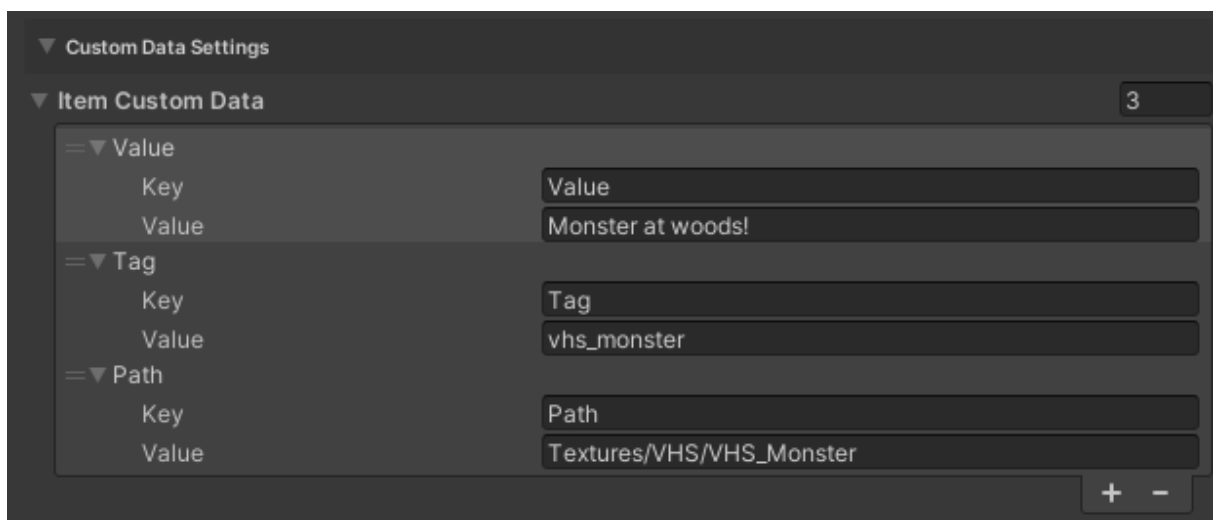
- Supported video formats: [Video file compatibility](#)
- Move **VHS Player** and **VHS Monitor** from **Resources** folder to the scene.
 - Connect **VHS Player** with **VHS Monitor**.



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



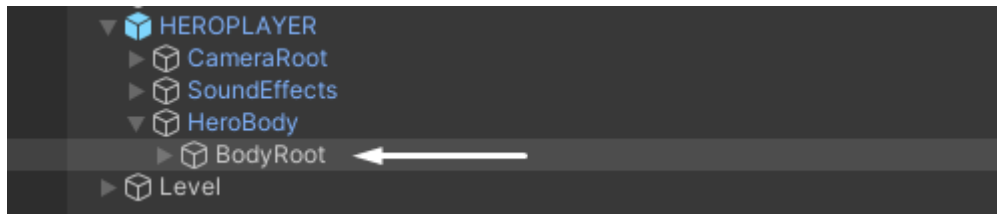
- In **Interactive Item** script, set VHS Tape **Item Custom Data**.



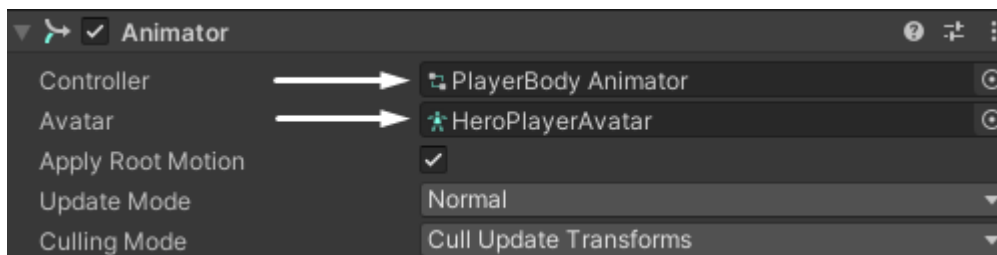
- The **Path** value is the path inside the **Resources** folder. All keys must be same!

ADDING CUSTOM PLAYER MODEL

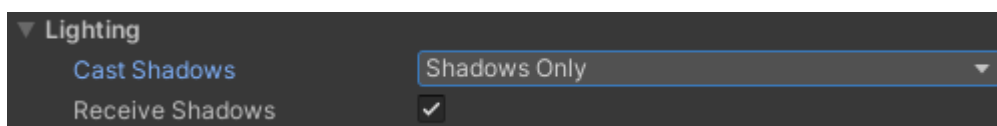
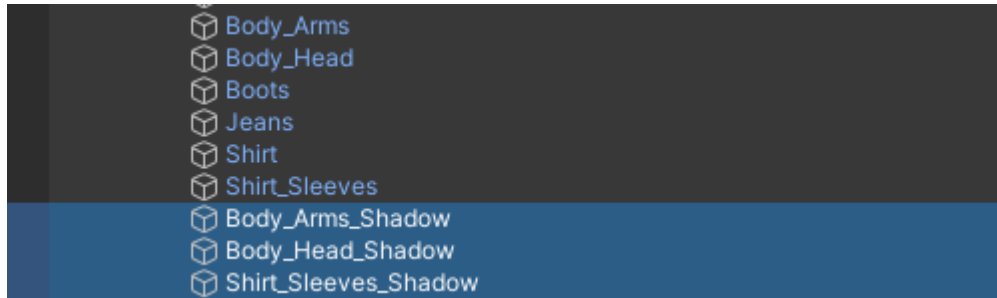
1. Replace the **BodyRoot** object with your **Player** model.



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



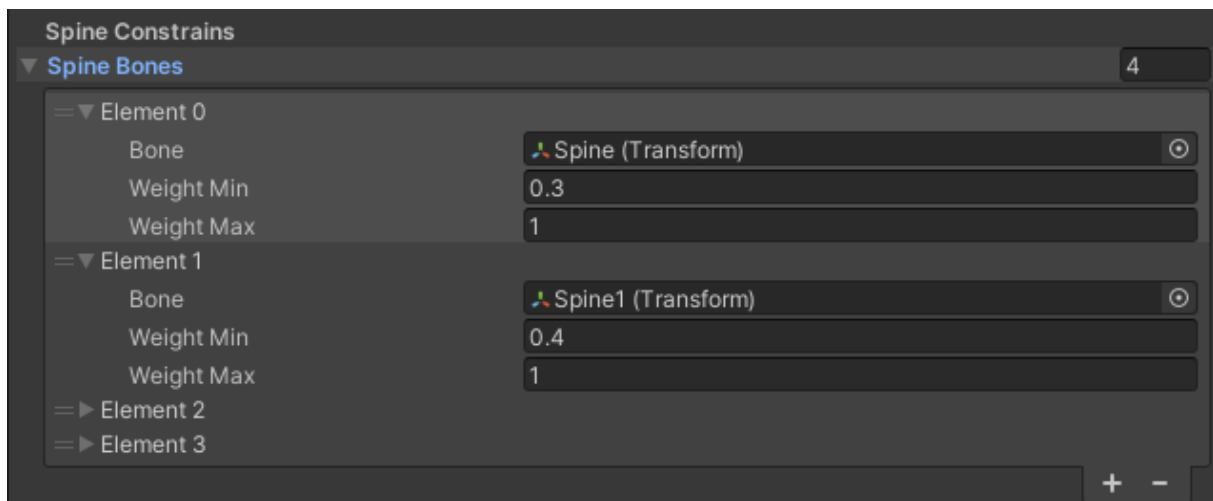
3. If you want your hands and head to be invisible to the **Main Camera**, you must duplicate these objects and change their **Cast Shadows** field to **Shadows Only** in the **Lighting** drop-down menu.



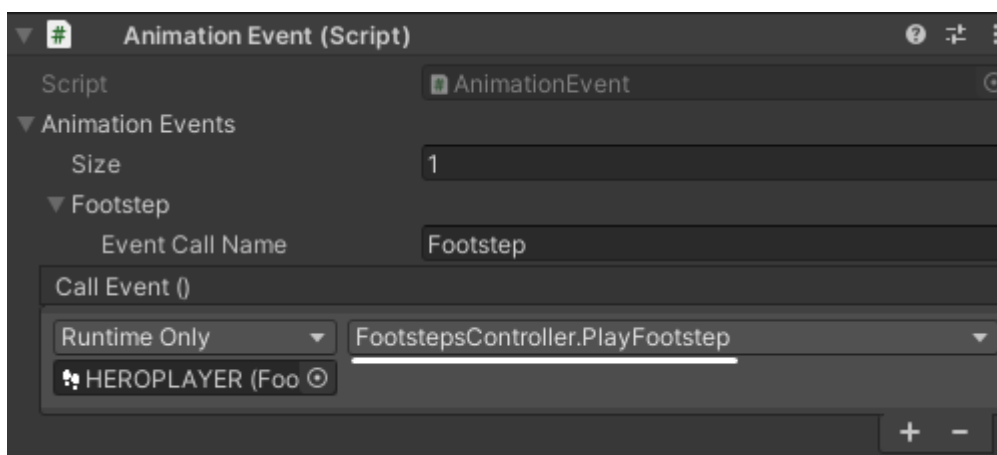
4. Next, you need to change the layer of the original duplicated objects to **PlayerInvisible** and the other player objects to **PlayerBody**.



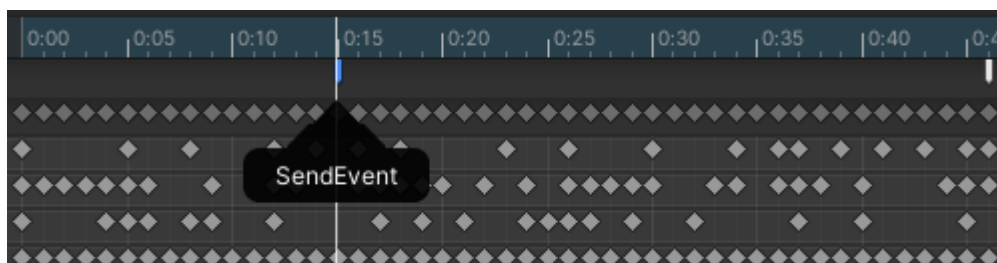
5. Assign a player's Spine Bones in the **Body Animator**.



6. To perform event-based footsteps, you must add the **AnimationEvent.cs** script to the **HeroBody -> HeroRoot** object 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 2.0 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 **EmeraldAISendDamage.cs** script right after the enemy **EmeraldAISystem** script.
5. Follow the **Emerald AI** instructions and create enemies.

HDRP/URP PARTIAL 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](#) – HFPS Website
- [Horror FPS KIT](#) – HFPS AssetStore

