



HORROR FPS KIT 1.5

DOCUMENTATION

THANKS FOR BUYING OUR 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	4
PLAYER FUNCTIONS	4
SYSTEMS	4
OBJECT PICKUPS	4
DYNAMIC FUNCTIONS	4
MORE FUNCTIONS	4
PROJECT SETUP	5
MAIN MENU SETUP	5
GAME SCENE SETUP	6
CONFIG MANAGER	7
HOW TO CONNECT CONFIG MANAGER WITH OTHER SCRIPTS	8
CONFIG MANAGER FUNCTIONS	8
INPUT MANAGER (REBINDABLE INPUT)	9
ADDING NEW INPUT	9
DESERIALIZE INPUT USING CONFIG HANDLER	10
TYPE PARSER	11
SETTING UP PARSER	11
INTERACT MANAGER	12
DEFINING NEW INTERACTIVE ITEM	12
DYNAMIC OBJECTS	13
DYNAMIC DOOR	13
DYNAMIC DRAWER	14
DYNAMIC LEVER	15
DYNAMIC VALVE	16
DYNAMIC MOVABLE INTERACT	16
DRAGGABLE OBJECTS	17
OBJECTIVE MANAGER	18
ADDING NEW OBJECTIVES	18
WATER BUOYANCY	20
PLAYER BODY (BodyAnimator)	22

INVENTORY	23
ADDING NEW ITEMS	23
INVENTORY SLOTS EXPAND	25
INVENTORY CONTAINERS	25
COMBINABLE ITEMS	26
INVENTORY CUSTOM ACTIONS	27
SAVE/LOAD MANAGER	29
SETTING UP SAVE/LOAD MANAGER	29
ADDING CUSTOM SAVEABLES	29
SAVING USING ATTRIBUTE	30
SAVING GAMEOBJECT DATA	31
SAVING GAME DATA	32
GRAPHIC SETTINGS	34
ADDING NEW PLAYER WEAPONS OR ITEMS	35
AI ZOMBIE BEHAVIOUR	37
ADDING NEW ZOMBIE	37
ZOMBIE IMPACT SOUND ATTRACT	39
JUMPSCARES	40
SMALL JUMPSCARE	40
JUMPSCARE WITH EFFECT	41
AMBIENCE ZONE	42
EXAMINE MANAGER	43
ADDING EXAMINE OBJECTS	43
FLOATING OBJECTS	44
ADDING NEW FLOATING OBJECTS	44
HOW TO UPDATE POSTPROCESSING?	45
FIXES FOR MOST KNOWN ERRORS	45
PACKAGE MANAGER ERROR	45
BUG, ERROR REPORT	46
CREDITS	46
LINKS	46

FEATURES

PLAYER FUNCTIONS

- Player Controller (Walk, Run, Jump, Crouch, Crawl, Ladder Climbing)
- Full Player Body (Body Animator)
- Player Footsteps With Sounds
- Weapons (Firearm, Melee)
- Player Lean (Wall Detection)
- Fall Damage
- Zoom Effect
- UI Crosshair

SYSTEMS

- Save/Load System (Player Data, Scene Data, Items Data, Encrypt)
- Objective System (New, Complete, Complete And New)
- Inventory System (Add, Remove, Move, Replace, Use, Combine, Drop, Store)
- Inventory Shortcuts System
- Drag Rigidbody System (Rotate, Zoom, Throw)
- Examine And Paper Read System (Rotate, Examine)
- Wall Detect System (Hide Weapon)
- Interact System

OBJECT PICKUPS

- Custom Object Pickup (Events, Hints, Objective Trigger)
- Light Objects (Flashlight, Oil Lamp, Candle)
- Backpack Pickup

DYNAMIC FUNCTIONS

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

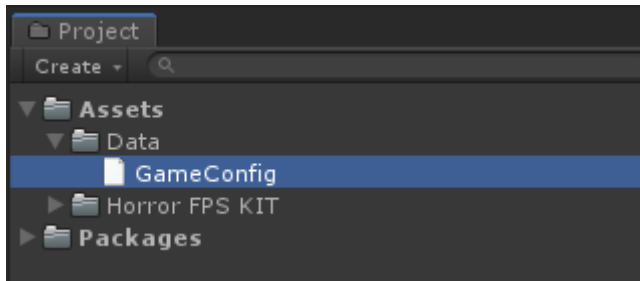
MORE FUNCTIONS

- Scene Loader (Background Change, Tips)
- Rebindable Input Manager
- AI Zombie (Walk, Run, Attack, Patrol, Attract)
- Water Buoyancy
- UI Menus (Main Menu, Load Game, Pause Menu, Dead Menu)
- Jumpscare (Event, Scare Effect, Scared Breath, Animation)
- Interactive Light (Lamp, Switch, Animation)
- Floating Object Icon
- Ambience Sound Trigger (In, Out)
- Notifications (Pickup, Hint, Message)
- Props, Collectable Objects, And Much More...

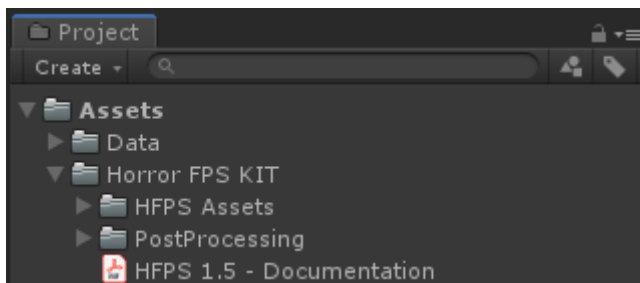
PROJECT SETUP

We highly recommend to import asset to a empty project and also import all Tags and Layers!

1. Import **Asset** to a **Empty Project** (All Project Settings will be overwritten!)

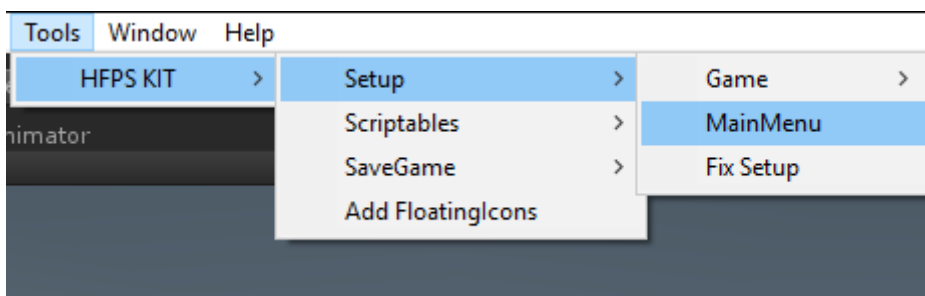


2. Move **Data** folder to your project **Assets** folder or **Run** game.

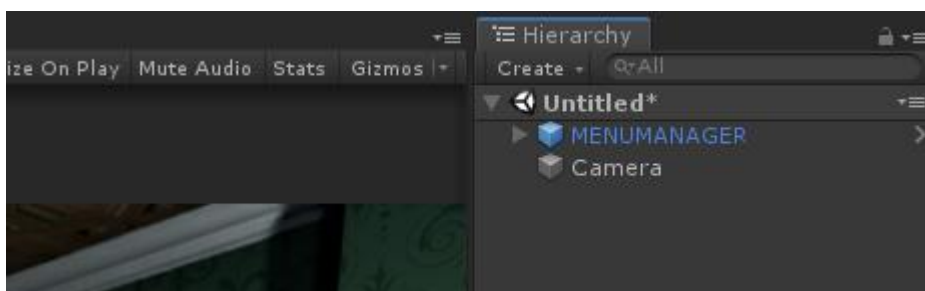


MAIN MENU SETUP

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

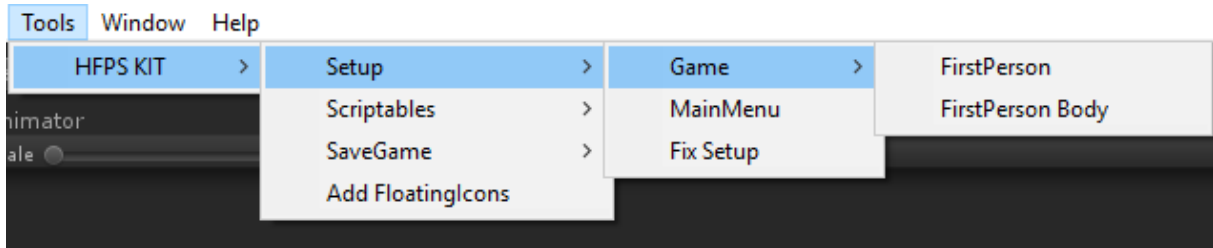


3. Now check if you have a **MENUMANAGER** object in your scene.

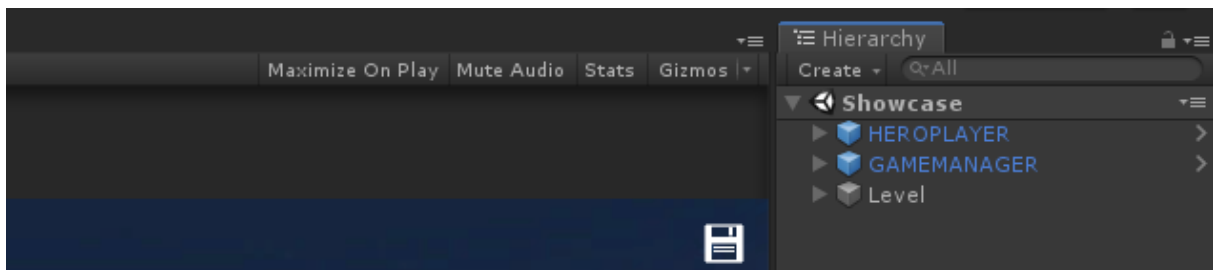


GAME SCENE SETUP

1. Open new empty scene.
2. Go to **Tools -> HFPS KIT -> Setup -> Game** and choose (FirstPerson or Body)



3. Check if your scene has a **PLAYER** and **GAMEMANAGER** object.



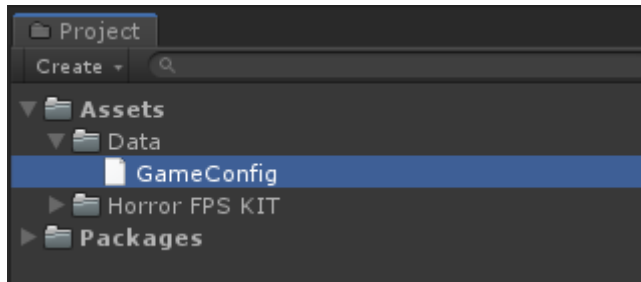
4. Move **PLAYER** to a floor.
5. We recommend run **HFPS** from **Main Menu** and let it to create a needed files.



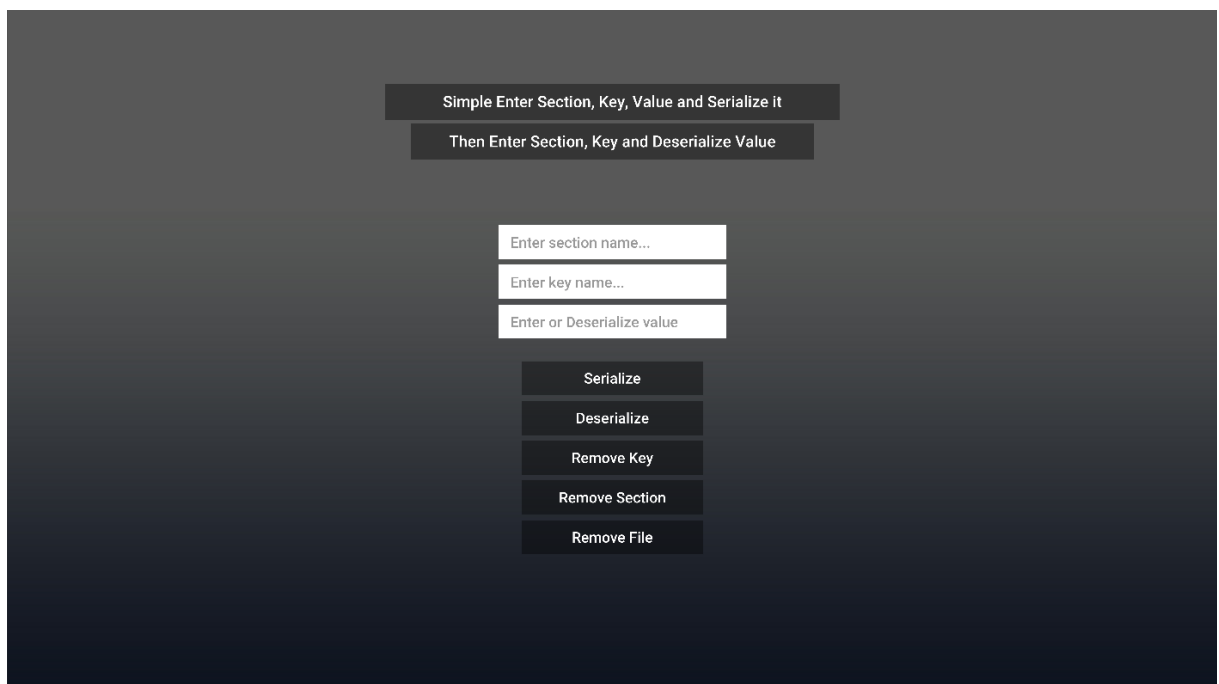
CONFIG MANAGER

It's a simple **Serialization Manager** which is used to **Serialize** and **Deserialize** basic game settings (**Input, Graphics, Game Settings**).

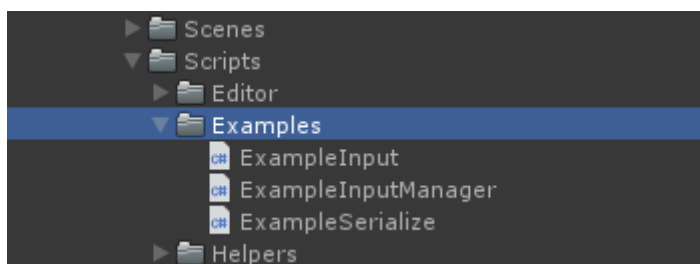
- All config files are stored inside project or inside exported game to folder named **Data**.



- You can easily view or edit config settings by the **ExampleSerialization** scene



- Also the example scripts for serialization are included in **Scripts** folder.



HOW TO CONNECT CONFIG MANAGER WITH OTHER SCRIPTS

1. Add namespace to your script

```
using ThunderWire.Configuration;
```

2. Setup **Config Folder** and **Config Name**

```
ConfigManager.SetFilePath(FilePath.GameDataPath);  
ConfigManager.SetFilename("Config Name");
```

CONFIG MANAGER FUNCTIONS

ConfigManager.EnableDebugging(bool); - Enable Debugging

ConfigManager.SetFilename(string); - Set Config Name

ConfigManager.SetFilePath(FilePath); - Set Config Path

ConfigManager.Serialize("Section", "Key", "Value"); - Serialize to config file

ConfigManager.Deserialize("Section", "Key"); - Deserialize from config file

ConfigManager.Deserialize<TYPE>("Section", "Key"); - Deserialize to Type

ConfigManager.ContainsSection("Section"); - Check if Config has Section

ConfigManager.ContainsSectionKey("Section", "Key",); - Check if Section has Key

ConfigManager.ContainsKeyValue("Section", "Key", "Value",); - Check if Key has Value

ConfigManager.RemoveSectionKey ("Section", "Key"); - Remove Key from Section

ConfigManager.RemoveSection ("Section"); - Remove Section from File

ConfigManager.GetSectionKeys ("Section"); - Get Count fo Section Keys

ConfigManager.ExistFile ("ConfigFolder", "ConfigName "); - Check if Config Exists

ConfigManager.ExistFileInFolder ("File", "Folder "); - Check if Config Exists in Folder

ConfigManager.ExistFileWithPath("FullPath", "File"); - Check if Config Exists is Path

ConfigManager.RemoveFile(FilePath, "File"); - Remove file from Path

ConfigManager.DuplicateFile(FilePath, "File", "Name"); - Duplicate File in Path

ConfigManager.GetFolderPath(FilePath); - Get Folder Path

ConfigManager.GetFilepathRoot(FilePath); - Get Folder Path Root

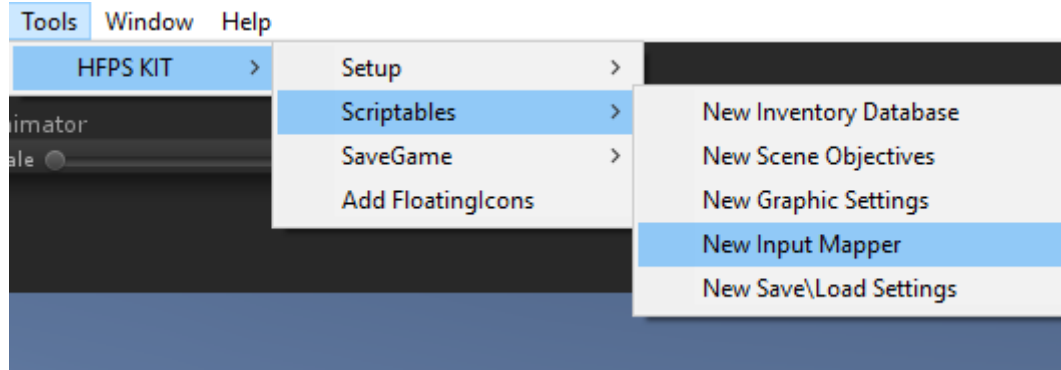
ConfigManager.GetFileAndPath(FilePath, "File"); - Get Fullpath to File

ConfigManager.GetFileAndPathFolder(FilePath, "Folder", "File"); - Get File in Filepath and Folder

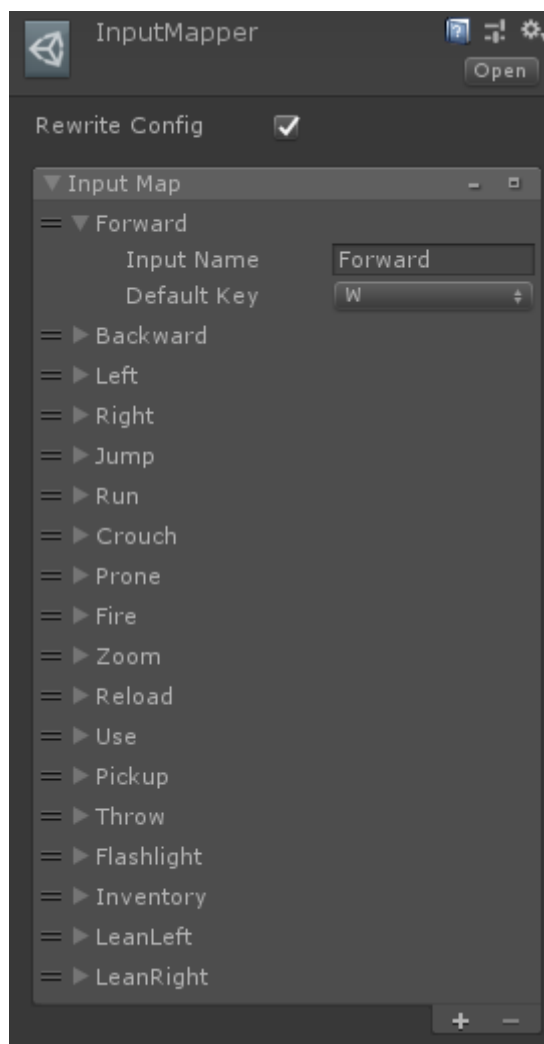
INPUT MANAGER (REBINDABLE INPUT)

ADDING NEW INPUT

1. Create or Open exist **Input Mapper Settings**.



2. Now you can **Edit** or **Create** New Inputs.



- Input Controller will automatically create Inputs in that order.
- If you select **Rewrite Config**, script will automatically rewrite all changed Inputs in config file with input mapper default key, otherwise all inputs from config file will be set as default.

DESERIALIZE INPUT USING CONFIG HANDLER

- For example you can open **ExampleInput.cs** script from **Examples** folder.

1. Add namespace to your script

```
using ThunderWire.Helpers;
```

2. Define **ConfigHandler.cs**

```
public ConfigHandler configHandler;
```

3. Define new key

```
private KeyCode useKey;
```

4. Write a parsing function to **Update()**.

```
void Update()
{
    if (configHandler.GetKeysCount() > 0 && !isSet)
    {
        useKey = Parser.Convert<KeyCode>(configHandler.Deserialize("Input", "Use"));
        isSet = true;
    }

    if (Input.GetKeyDown(useKey) && !isPressed)
    {
        Debug.Log("Use Key Pressed!");
        isPressed = true;
    }
    else if (isPressed)
    {
        isPressed = false;
    }
}
```

- Also with the same way you can deserialize Input using a **Input Controller** script, which is much easier to use.
- For example, open a **ExampleInputManager.cs** script.

TYPE PARSER

It's a useful tool, which can convert string to a specified type using one function.

SUPPORTED PARSE TYPES

Vector2, Vector3, Vector4, Quaternion, int, uint, Long, uLong, float, double, bool, char, short, byte, Color, KeyCode

SETTING UP PARSER

All you have to do is add a **Helpers** namespace:

```
using ThunderWire.Helpers;
```

Then you can **Parse** your string to a correct format using this command:

```
Parser.Convert<TYPE>("string");
```

VECTOR 2:

```
Parser.Convert(string x, string y);
```

VECTOR 3:

```
Parser.Convert(string x, string y, string z);
```

VECTOR 4 or QUATERNION:

```
Parser.Convert<TYPE>(string x, string y, string z, string w);
```

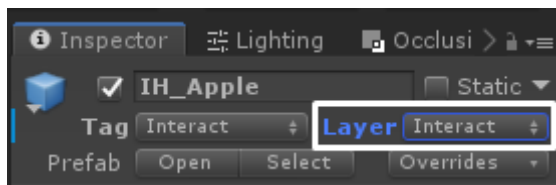
TYPE MUST BE VECTOR4 or QUATERNION! OTHERWISE YOU WILL GET A ERROR.

INTERACT MANAGER

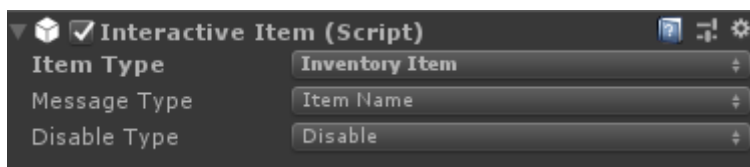
- All items with a **Interact** layer will be defined as a **Interactive object**.
- You can change basic **Interact** settings in a **InteractManager** script located in a **MouseLook** object.

DEFINING NEW INTERACTIVE ITEM

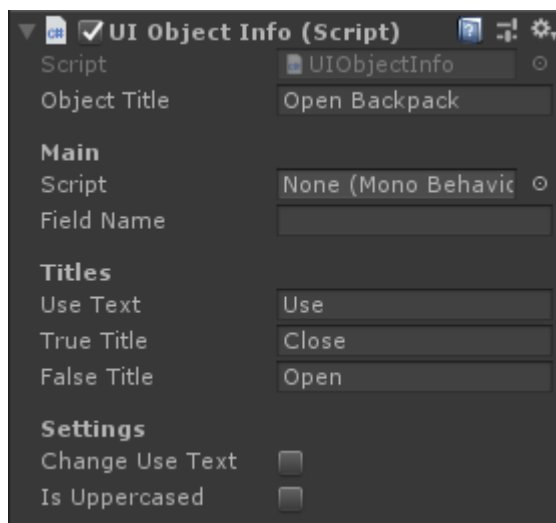
1. Set object layer to a **Interact** layer (Also you can set object tag to **Interact**)



2. Add **Interactiveltem.cs** script to a object



- With this script you can set basic object properties such as **Item Type**, **Message Type** and **Disable Type**.
- If you want add **Event** after object pickup, you need to add **ItemEvent.cs** script to object where is **Interactiveltem** script.
- You can also change a UI Info of a Interactive Item by adding a **UIObjectInfo.cs** script.

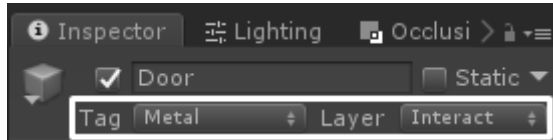


- With this script you can change a main **Interact Titles**.
- You can also change a **True/False** title by defining a **Script** which will be inspected and a **Field Name**.
- For example, have look at the **Dynamic Door Prefabs**.

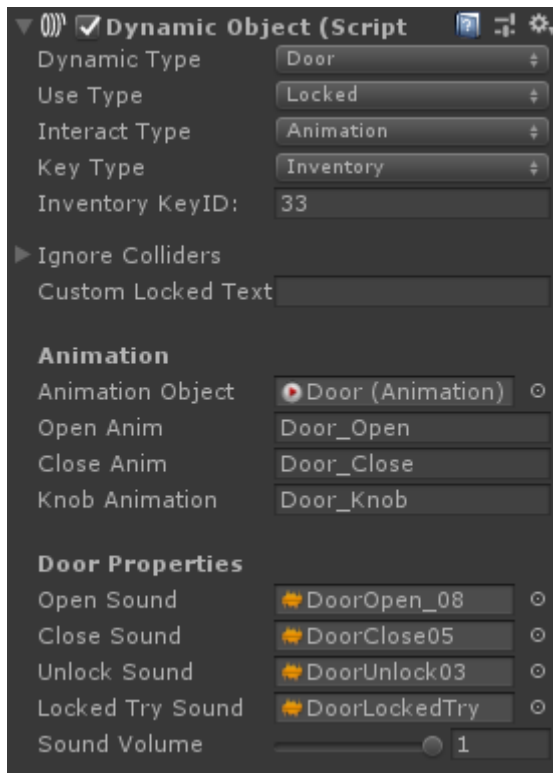
DYNAMIC OBJECTS

DYNAMIC DOOR

1. Set door object layer to a **Interact** layer (you can set a object tag to a door material type).

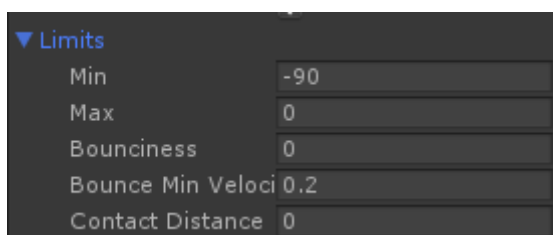


2. Add collider, **DynamicObject.cs** script and set **Dynamic Type** to a **Door**.



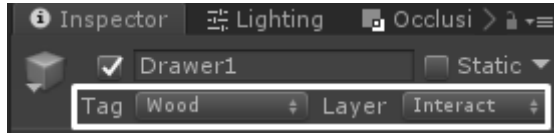
- You can set a door **Use Type** to a **Normal**, **Locked** or **Jammed**.
- With **Interact Type** you can set a type how you will open a door.
- Also there is an option for a **Key Type** where you can define how door will be unlocked.

3. If you set door **Interact Type** to a **Mouse** type, then you will need to add required additional components (**Rigidbody** and **Hinge Joint**).
4. In **Hinge Joint** component you'll need to set a rotate **Limits**.

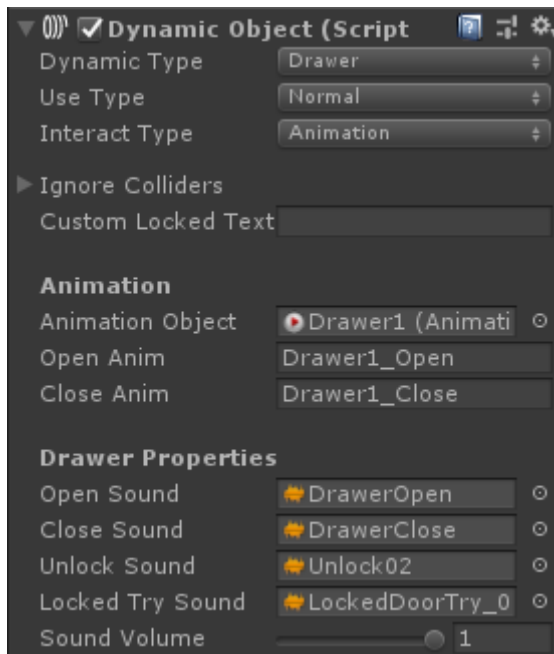


DYNAMIC DRAWER

1. Set drawer object layer to a **Interact** layer (you can set a object tag to a drawer material type).

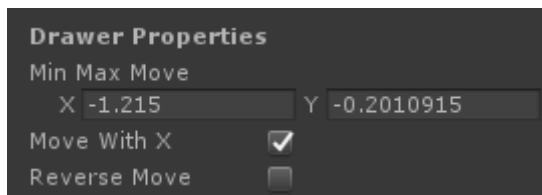


2. Add collider, **DynamicObject.cs** script and set **Dynamic Type** to a **Drawer**.



- **Dynamic Drawer** has a same **Type** properties as a Door.

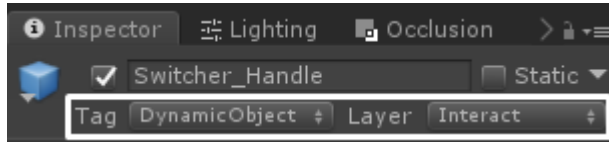
- **Mouse Interact Type** has some additional **Drawer Properties**.



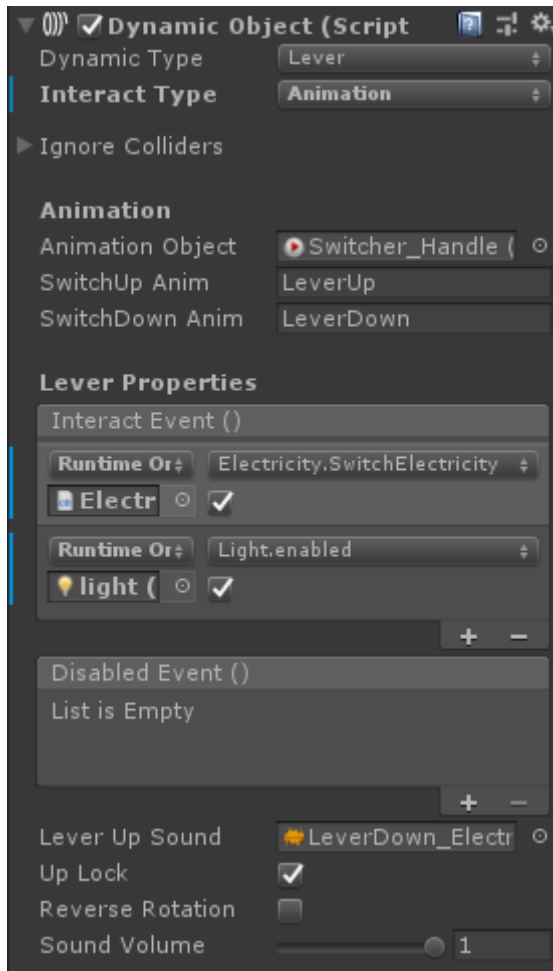
- With **Min Max Move** you can set drawer open and close limits.
- Drawer drag orientation is normally driven by a drawer object **X** position but some models have a different orientation so you can simply deselect **Move With X** statement and drag your drawer with a **Z** position.

DYNAMIC LEVER

1. Set lever object layer to a **Interact** layer (you can set a object tag to a lever material type)



2. Add collider, **DynamicObject.cs** script and set **Dynamic Type** to a **Lever**.



- You can set a **Lever Up Interact Events** or **Lever Down Interact Events**.
- Also there is an option to keep lever up locked.

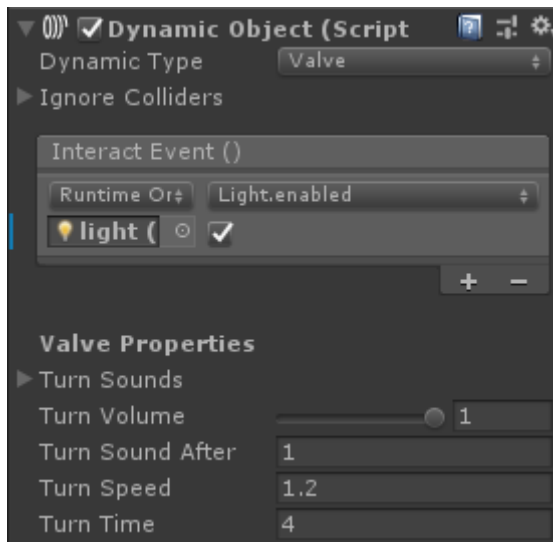
- If you set lever **Interact Type** to a **Mouse Type**, then you will need to add required additional components (**Rigidbody** and **Hinge Joint**).
- Also you'll need to set a rotate **Limits**.
- **Dynamic Lever** works with same way as a **Dynamic Door**, but it has a **Event Functions**.

DYNAMIC VALVE

1. Set valve object layer to a **Interact** layer (you can set a object tag to a valve material type or just set tag to a DynamicObject).



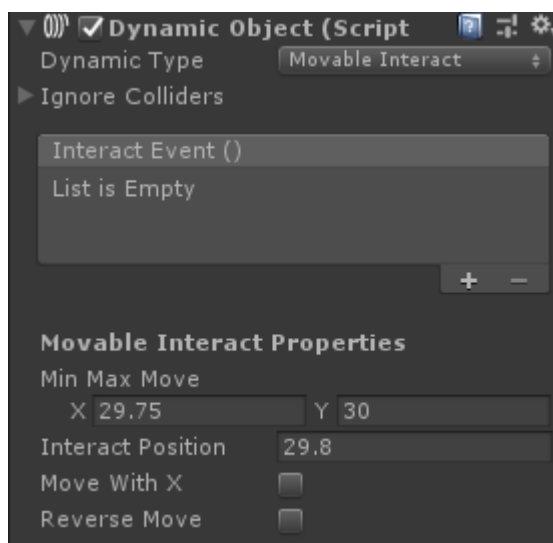
2. Add collider, **DynamicObject.cs** script and set **Dynamic Type** to a **Valve**.



- When you'll turn valve for certain time, the **Interact Event** will be **Invoked**.

DYNAMIC MOVABLE INTERACT

1. Set object layer to a **Interact** layer, add collider, **DynamicObject.cs** script and set **Dynamic Type** to **Movable Interact**.

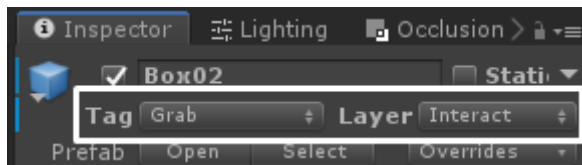


- **Movable Interact** is basically a drawer with a **Interact Event**.

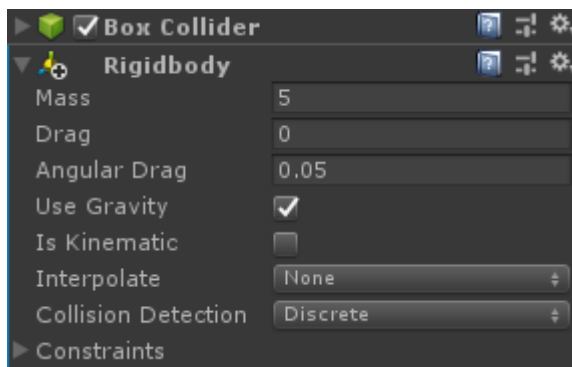
DRAGGABLE OBJECTS

- You can **Rotate**, **Zoom** and **Throw** dragged object.
- Dragged object cannot be **Interactive**!

1. Set object layer to **Interact** and tag to **Grab**.



2. Add required components (**Rigidbody** and **Collider**).

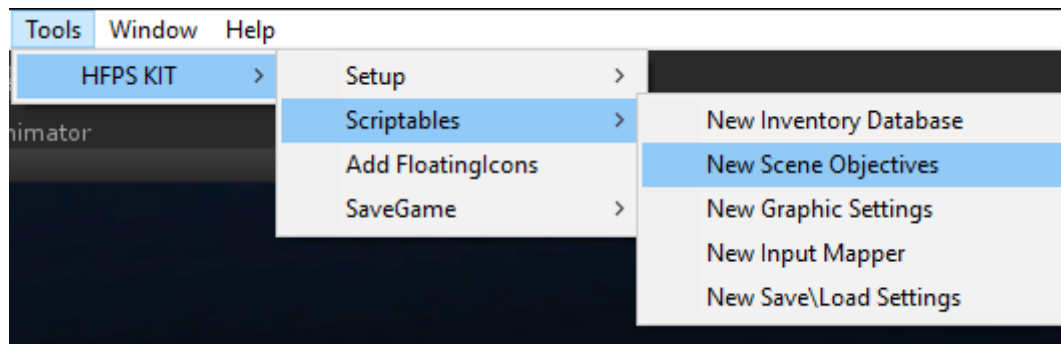


OBJECTIVE MANAGER

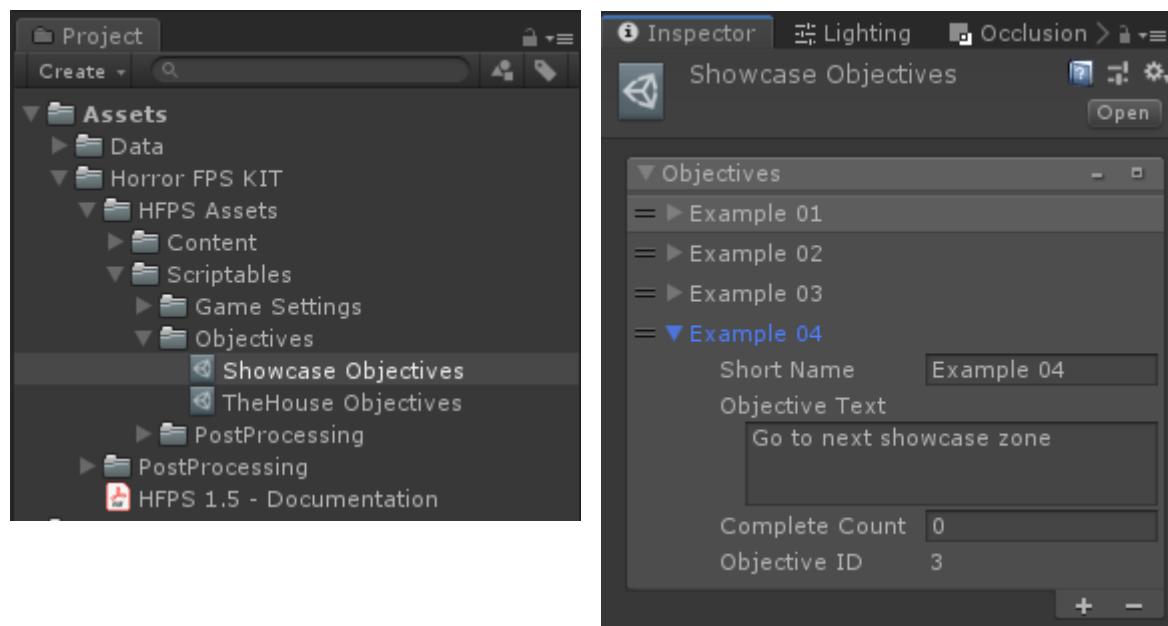
- It's a main script, which contains all functions for game objectives.
- Functions: **New Objective**, **Complete Objective**, **New and Complete Objective**

ADDING NEW OBJECTIVES

1. Create a new **Scene Objectives** asset.

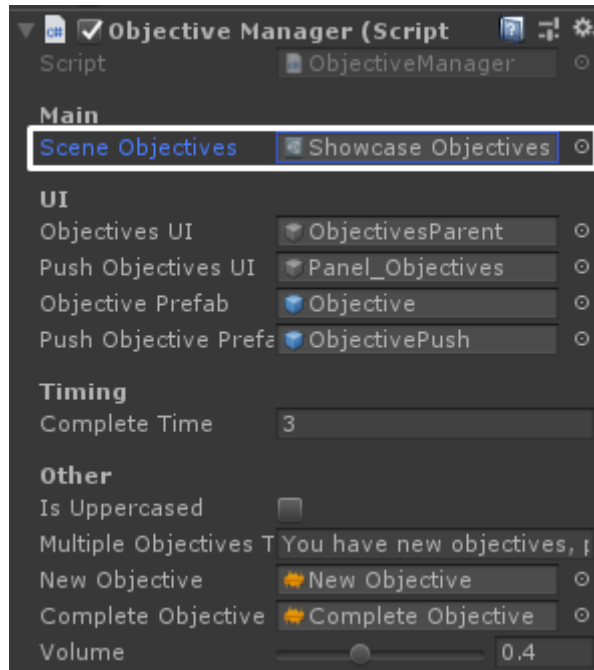


2. This will create a new scriptable asset, where you can add and set main objective properties.

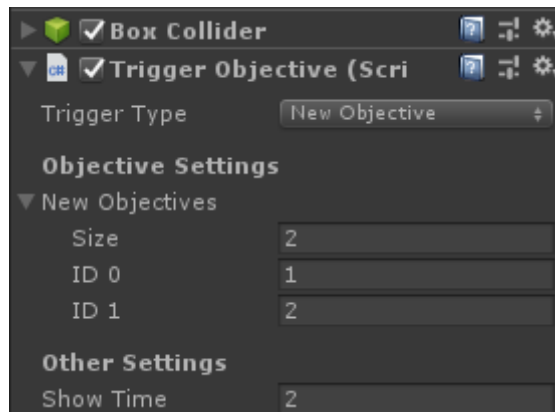


- Every objective has a unique **Objective ID** with which you can easily define which objective you want to trigger.
- Also when you change the order of the objectives, script will automatically reset **Objective ID** with a correct order.

3. Assign new scene objectives asset to the **Objective Manager** script located in **GAMEMANAGER** object.



4. Create a new trigger object with a **TriggerObjective.cs** script where you set a basic particular **Objective** properties.



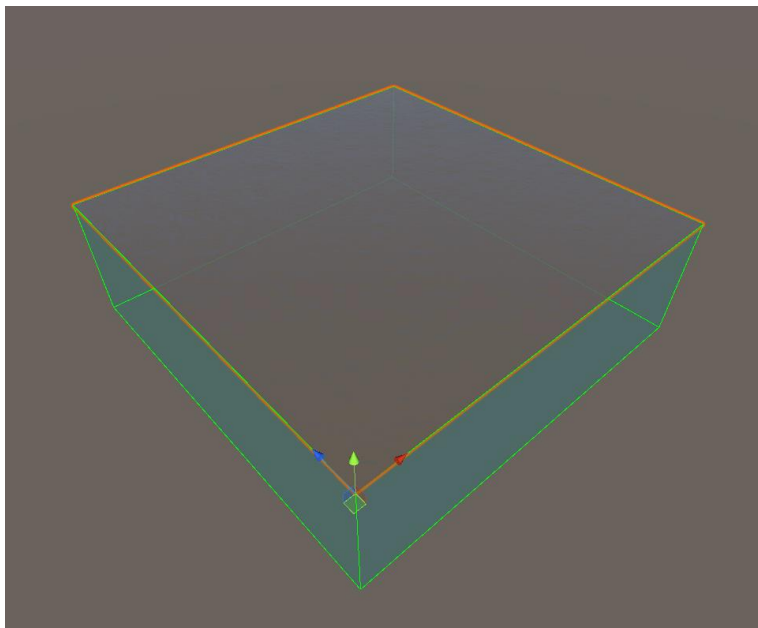
- Objective **Complete Count** means, that the objective will be completed only when, you trigger objective for a number of times.
- If you want to count, how much times you will need to trigger an objective, you need to format your **Objective Text** with a **format** tag "{0}/{1}".
- Example: Collect three apples {0}/{1}

WATER BUOYANCY

1. Drag & Drop Water Object.

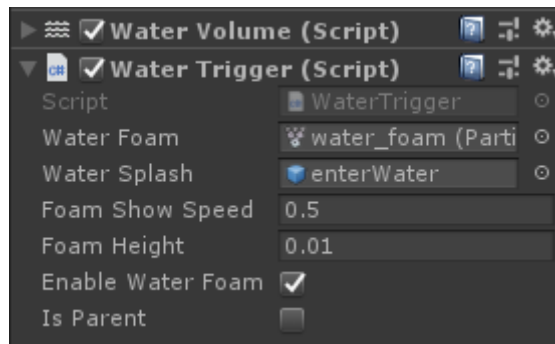


2. Add **WaterVolume.cs** script and change water object tag to a **Water Volume**
3. Script will automatically create a plane instance of an object.

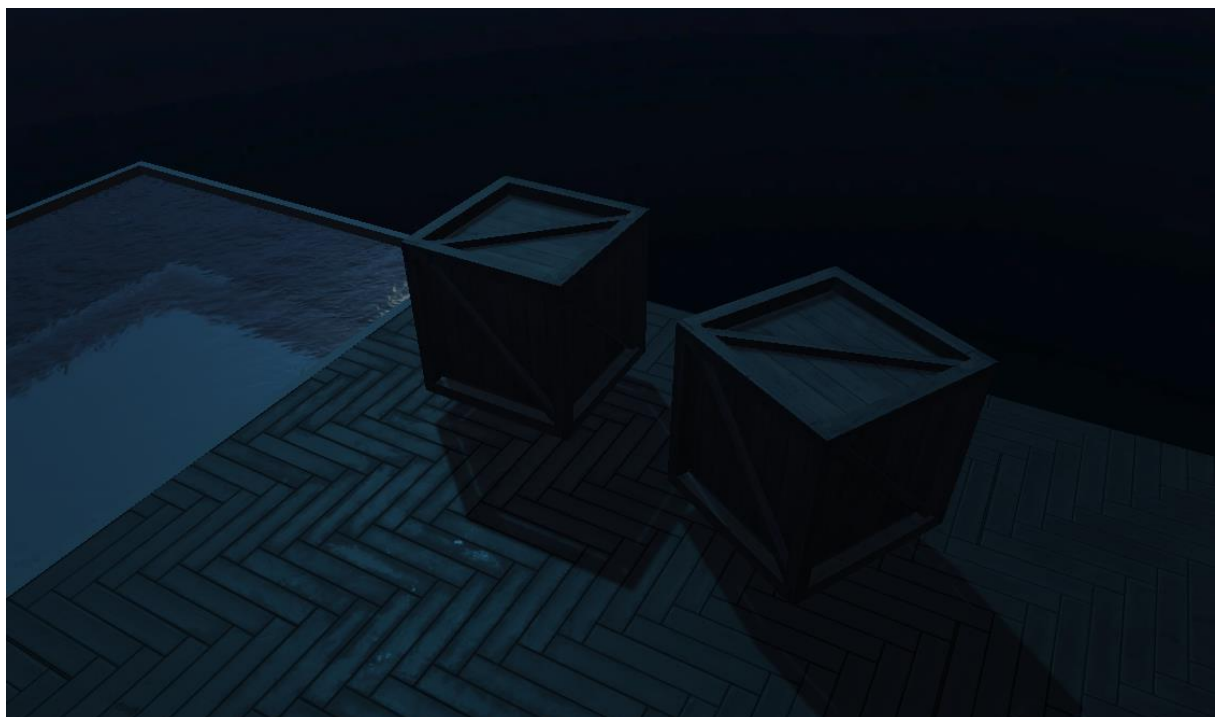
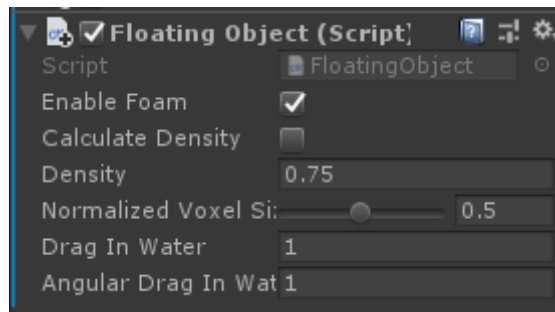


- To change plane dimensions use script **Rows** and **Columns**.
- If you need a smaller water object, change a **Quad Segment Size**.

4. Add **WaterTrigger.cs** script which controls object foams and player in water state.



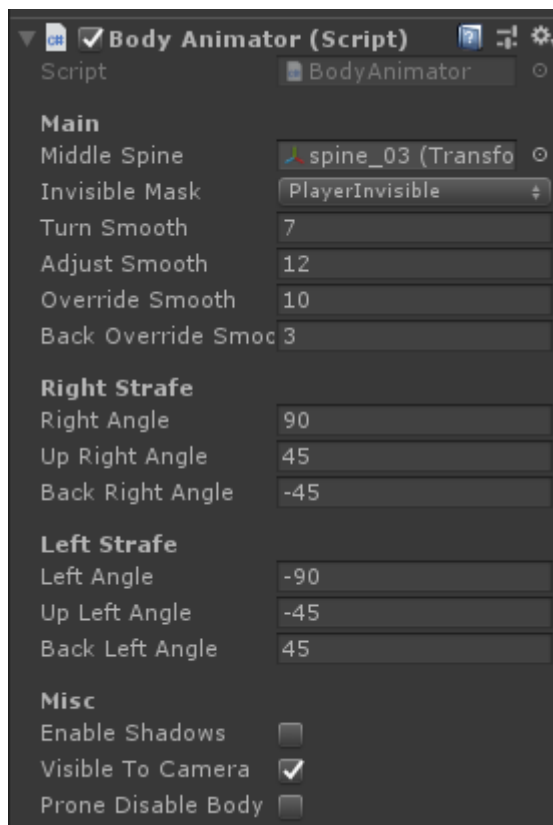
5. Add **FloatingObject.cs** script to floatable objects.



- **Object Density > Water Density = Lower Object Buoyancy**

PLAYER BODY (BodyAnimator)

- Body Animator is a main script, which contains all main functions for a **First Person** body animations.
- Script is not connected with other scripts, so you can simply remove the **HeroBody** object, if you doesn't want to have a FirstPerson Body.



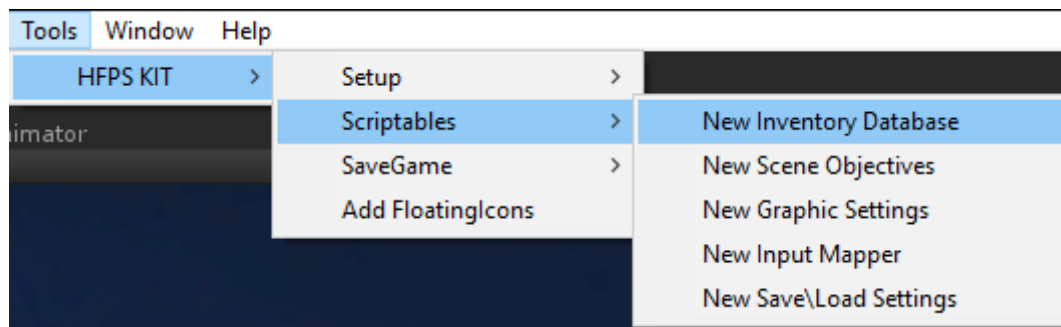
- Script supports all character models marked as a **Humanoid**.
- The **Middle Spine** field controls a character's spine rotation when you walking to sides, so it's required it to define correctly.
- You can also set strafe angles and body adjustments, to correctly control a body animations.

INVENTORY

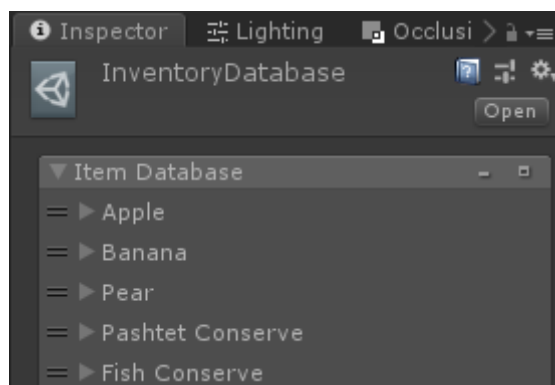
- You can show **Inventory** menu by pressing **TAB** button, also you can change this button directly in **Input Mapper** or in **Pause** or **Main Menu**.

ADDING NEW ITEMS

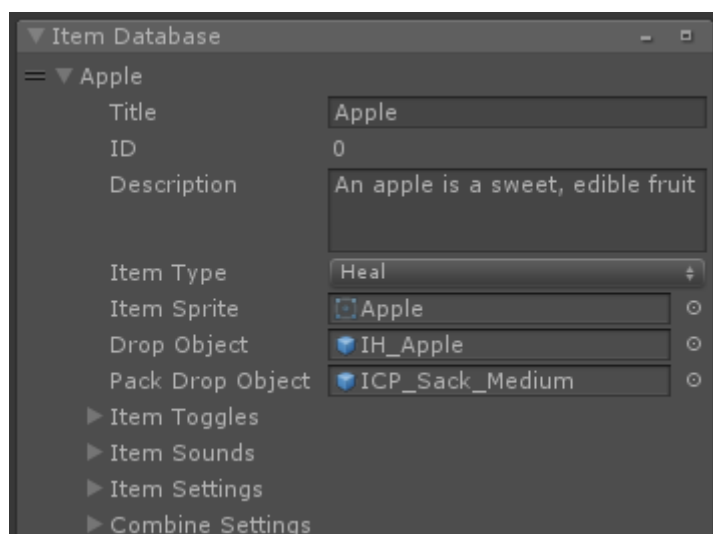
1. Create new **Inventory Database** asset.



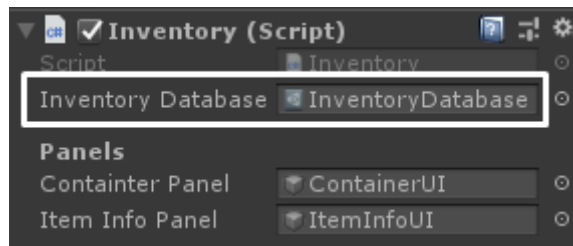
2. This will create a new **Inventory Database** inside a **Scriptables** folder.



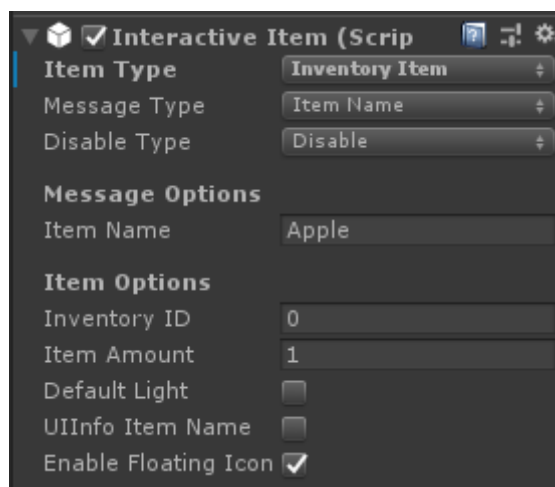
- Inside this scriptable asset you can define new **Inventory Items**.



3. Assign new Inventory Database to the **Inventory** script located in **GAMEMANAGER** object.



4. Add **Interactiveltem.cs** script to item object and set **Item Type** to a **Inventory Item**.



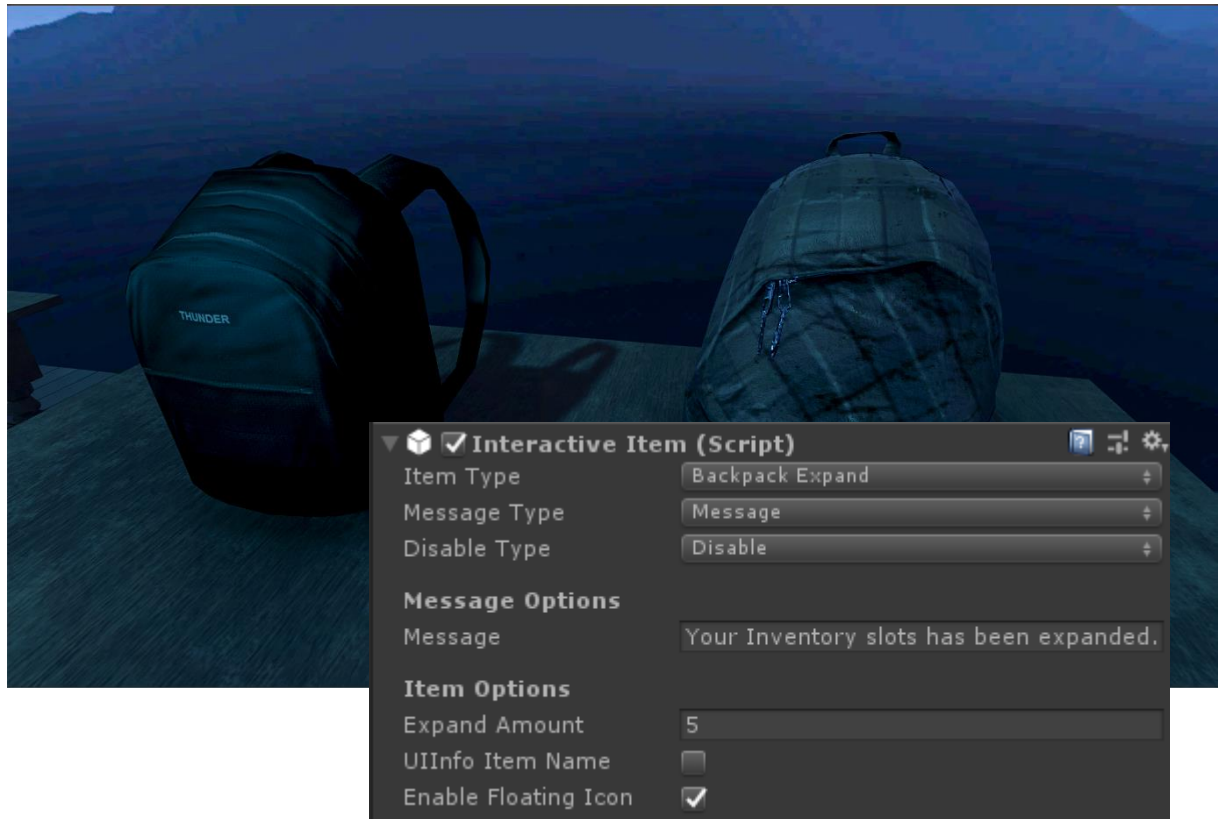
5. Set **Inventory ID** to a particular number of your item which is defined in **Inventory Database**.

- **Inventory Database** item ID is read only and is defined automatically with a database script.
- If you change order of any Item, it will automatically reset all item IDs with a current items order.



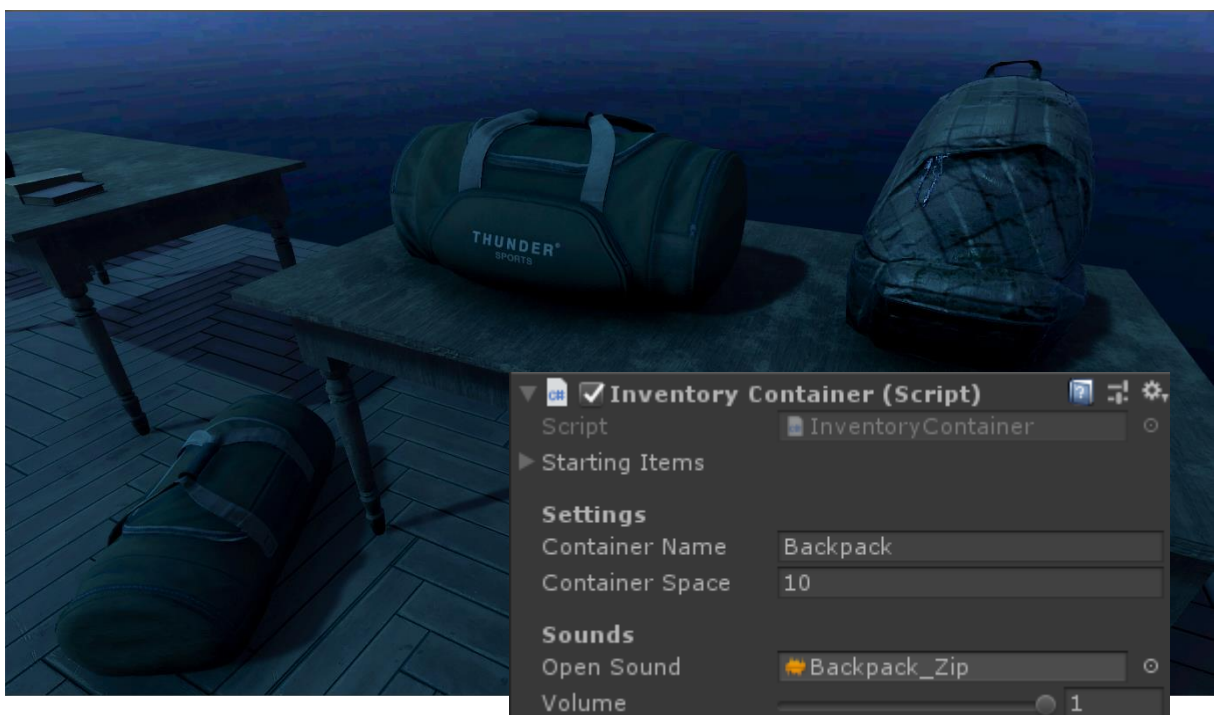
INVENTORY SLOTS EXPAND

- You can expand inventory slots by picking up an object where **Item Type** is set to a **Backpack Expand**.

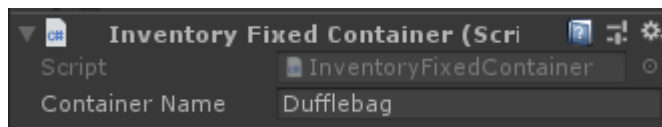


INVENTORY CONTAINERS

- With this method you can store inventory items to an object (**Backpack**).
- The main script for this method is an **InventoryContainer.cs**.

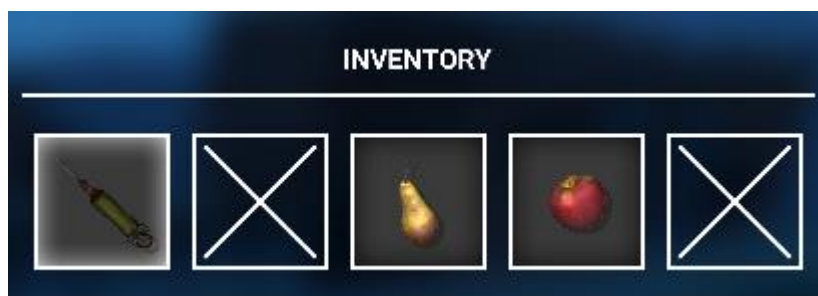
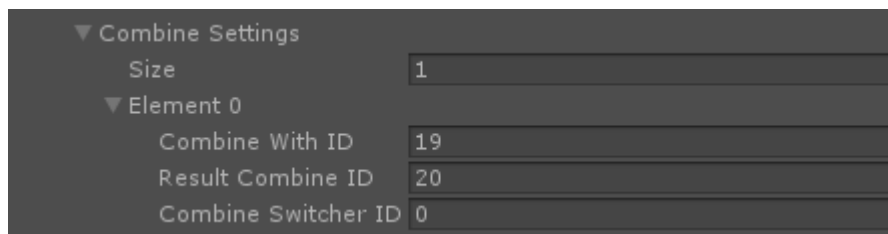


- Also if you want to access stored items everywhere, where is a specific object, you can do that by adding an **InventoryFixedContainer.cs** script to a objects.



COMBINABLE ITEMS

- Inventory has a function to combine two different objects with order to get a other useful item.
- This can be done by changing a **Combine Settings** on an item inside **Inventory Database**.

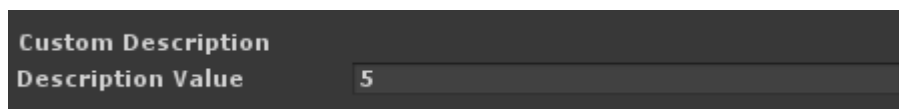


- An example **Inventory Database** with all properties is included in asset.

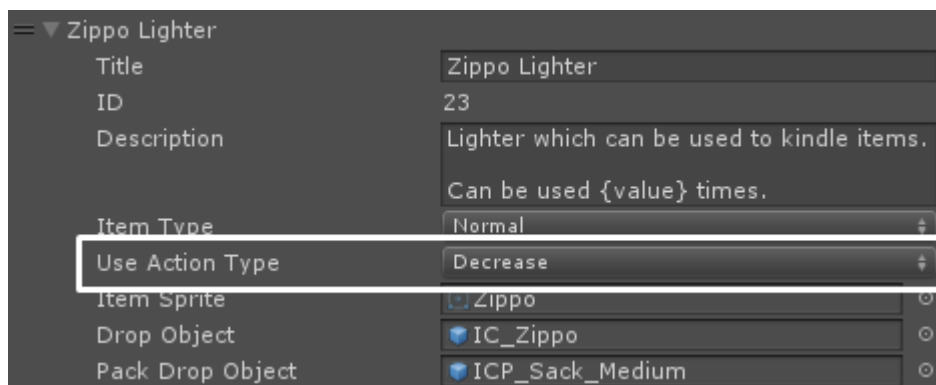
INVENTORY CUSTOM ACTIONS

- If you need to set custom actions to items after combine, you need to set item custom actions in **Inventory Database** asset.
- You can set actions such as Increase, Decrease, Custom Value.
- **Starting value** can be set in **Interactiveltem** script and **Trigger value** in **Inventory Database**.
- To **Activate Custom Actions** you must select in Item toggles **Do Action Use** or **Do Action Combine**.

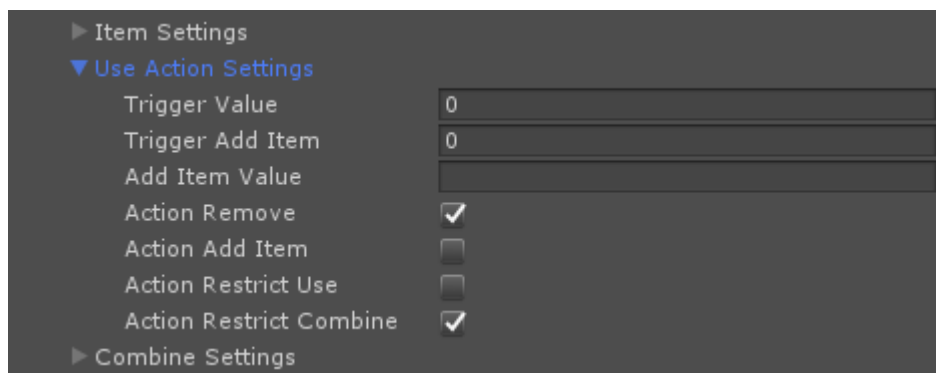
1. Set value in **Interactiveltem** script.



2. Define **Use Action Type** in **Inventory Database**



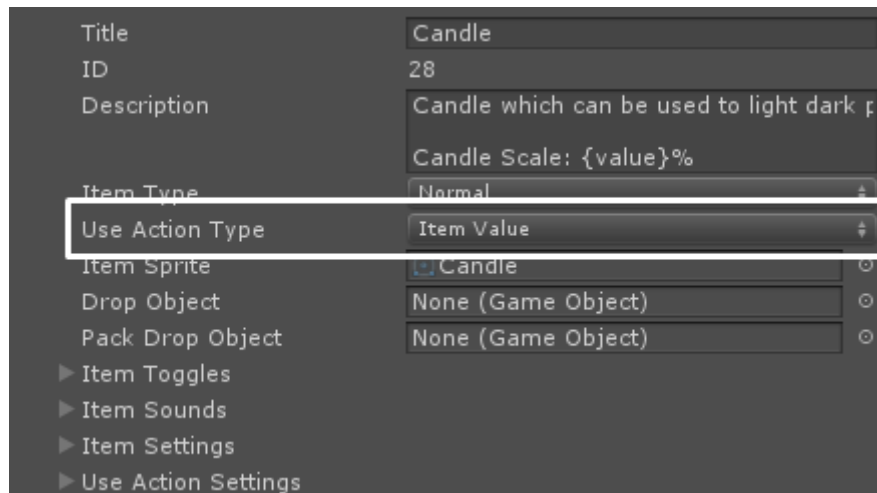
3. Set **Use Action Settings**



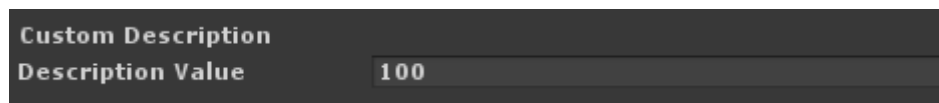
- To show current value in **Item Description**, you need to add **{value}** tag to the **Description** text as is shown in the screenshot.

- To use **Item Value** action type, you need to open **Switcher Item** script and add **ItemValueProvider** interface.

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



2. Set value in **InteractiveItem** script.



3. Add **ItemValueProvider** interface.

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

4. Fix Interface Errors and implement new functions **OnGetValue** and **OnSetValue**.

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

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

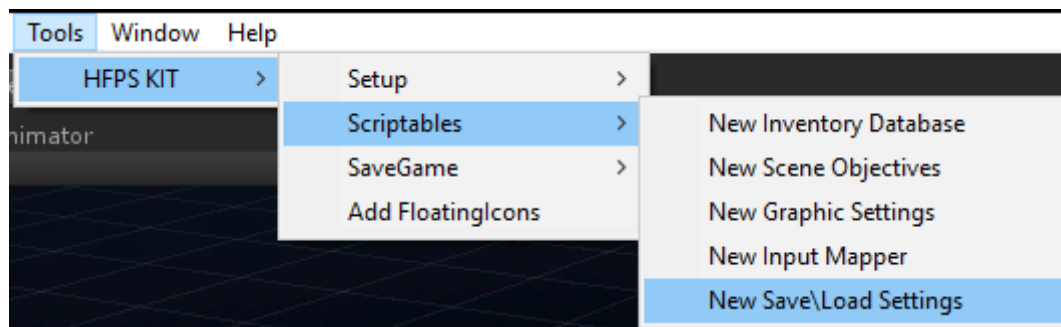
- With these functions you can simply define custom values.
- OnGetValue** must be controlled manually with **Switcher Item** script.

SAVE/LOAD MANAGER

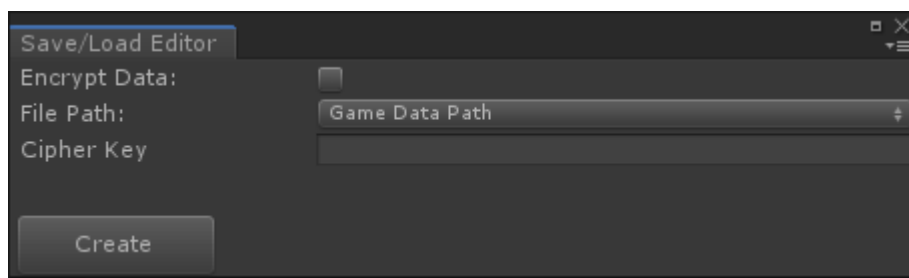
- Saving and Loading game data is a main function in many AAA game titles, with **SaveGameHandler.cs** you can save your game data too.

SETTING UP SAVE/LOAD MANAGER

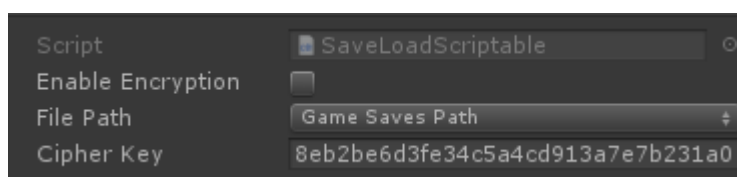
1. Create a new **SaveLoadSettings**.



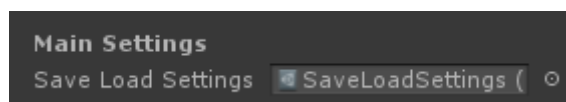
- This will show up a Save/Load Editor window.



- In this window you can set a basic saving settings.



- The **Cipher Key** is for security reason encrypted.
2. New created settings asset you'll need to define in **SaveGameHandler** script located in **GAMEMANAGER** object.



ADDING CUSTOM SAVEABLES

1. Add **ISaveable** interface to your custom script.

```
public class ExampleSaveable : MonoBehaviour, ISaveable
{
}

```

2. Fix errors caused by interface (VS 17: ALT + ENTER or CTRL + .).

```
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 a two new functions **OnLoad** and **OnSave**.
- These functions are main functions for saving and loading custom data.

3. Define, which data you want to save.

```
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 }
        };
    }
}

```

- Token key must be same as a key, with which you save data.

SAVING USING ATTRIBUTE

- This method allows you to simply save a script data using only attribute.

- The field must be always **public**, otherwise the manager will not find it.

1. Add **SaveableField** attribute to a field, which 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 field **custom key**.

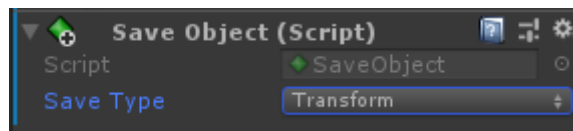
```
[SaveableField("theBoolean"), HideInInspector]
public bool exampleBool = true;

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

SAVING GAMEOBJECT DATA

- With this method you can save a basic **GameObject** data.
- **Transform, Rigidbody, Position, Rotation and Object/Renderer Active.**

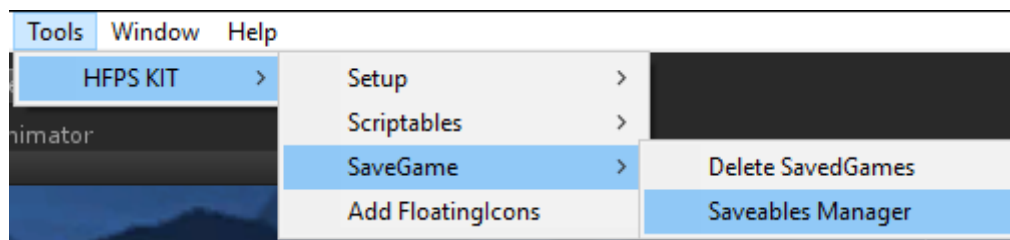
1. Add **SaveObject.cs** script to object.



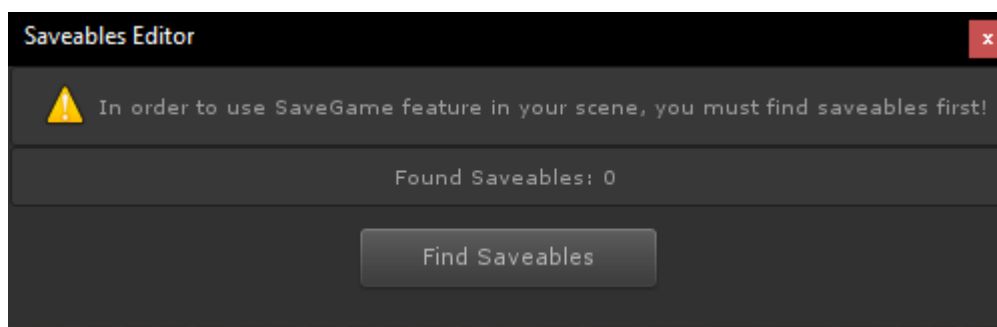
- In this script you can set which data from **GameObject** you want to save.

SAVING GAME DATA

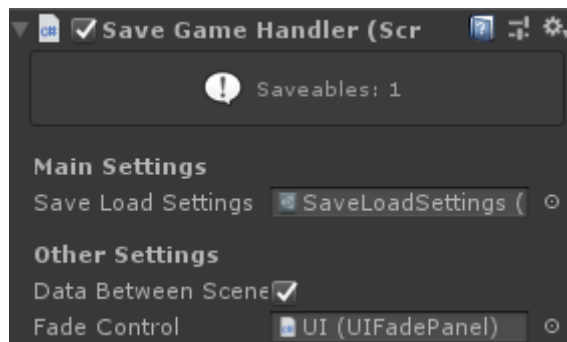
1. Select **Saveables Manager** from **Tools** menu.



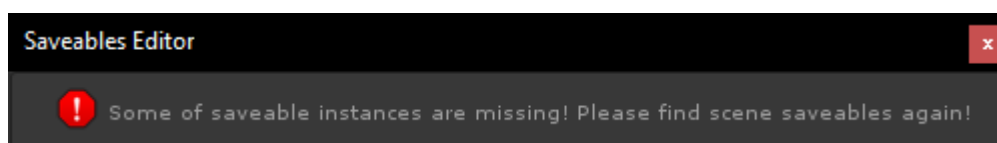
- This will show up a **Saveables Editor** window.
- Editor will not show up, when **SaveGameHandler** script will not be found.



2. Click **Find Saveables** button, editor will automatically find all scene saveable objects.



- If editor find any saveable objects, message will change to a **“SaveGame Handler is set up successfully!”**.
- The **SaveGameHandler** script also displays how many items will be saved.
- If you delete saveable object, you will be prompt to find saveables again.



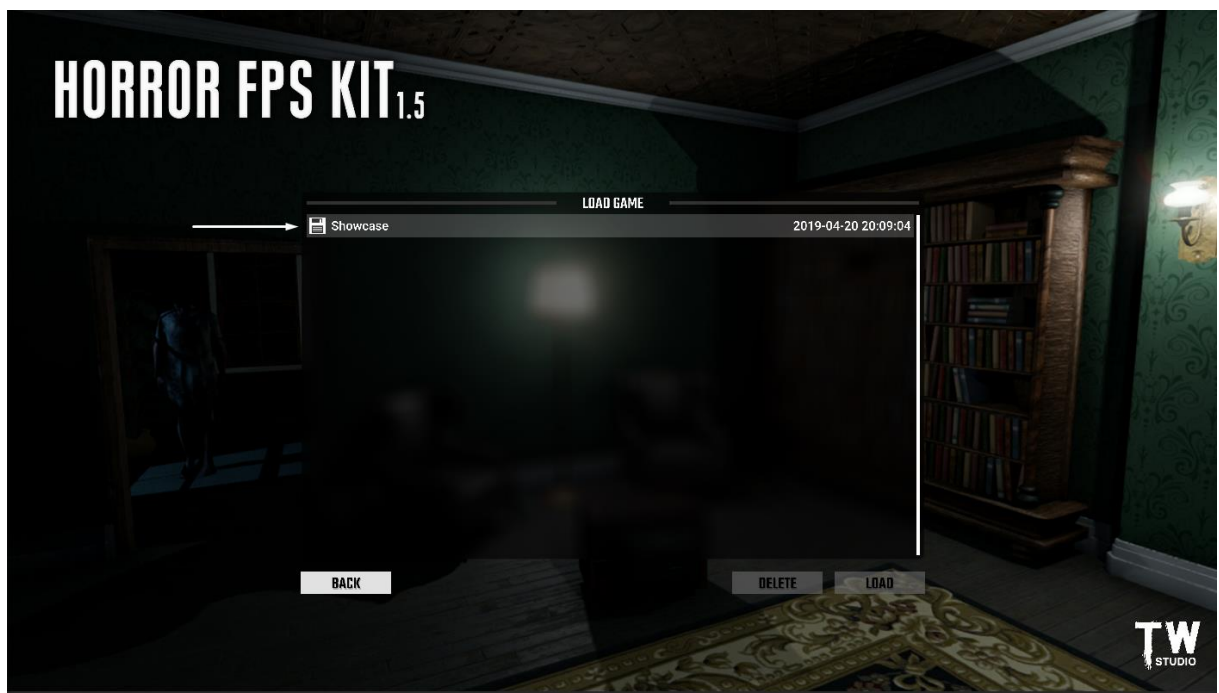
3. Select **Save Game** from **Pause Menu**.



- Icon of diskette indicates, that the game is saved correctly.



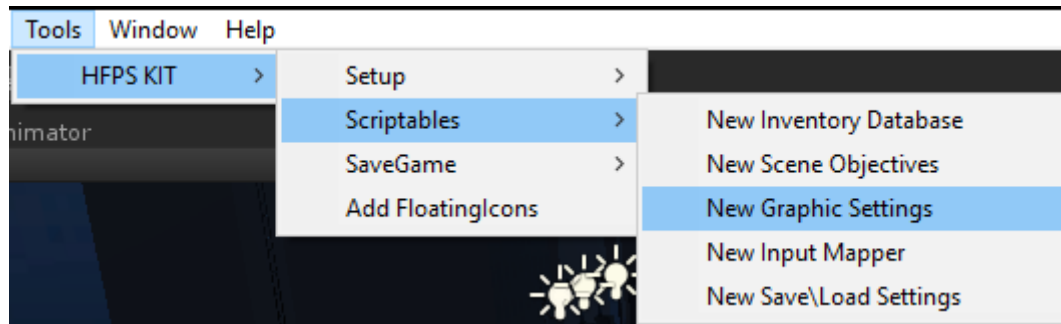
4. Then you would be able to **Load** saved game from a **Main Menu**.



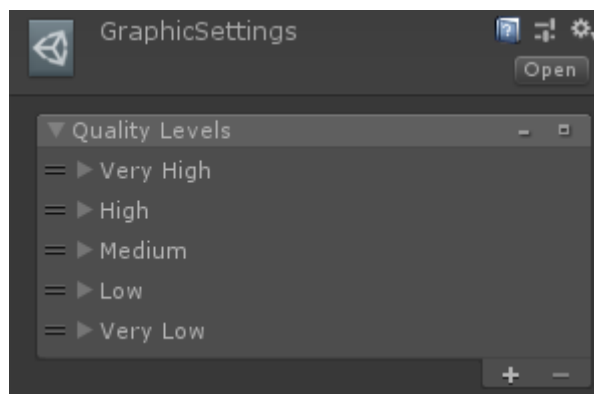
- Also you can **Delete** saved game, if you don't want it.

GRAPHIC SETTINGS

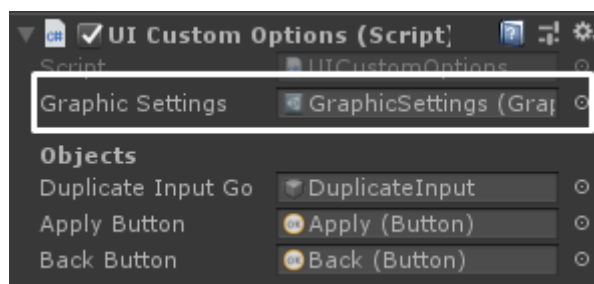
1. Edit or Create new **Graphic Settings** asset.



2. Add new or Edit current **Graphic Settings** in the **Scriptables** folder.



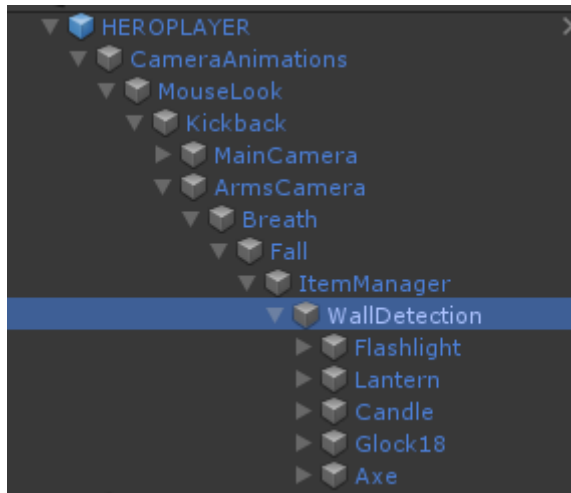
3. Assign **GraphicSettings** to the **UICustomOptions** script located in **GAMEMANAGER** object.



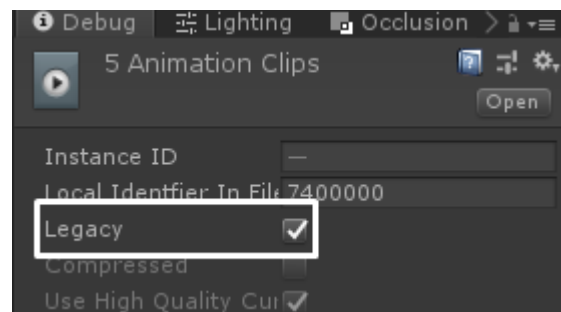
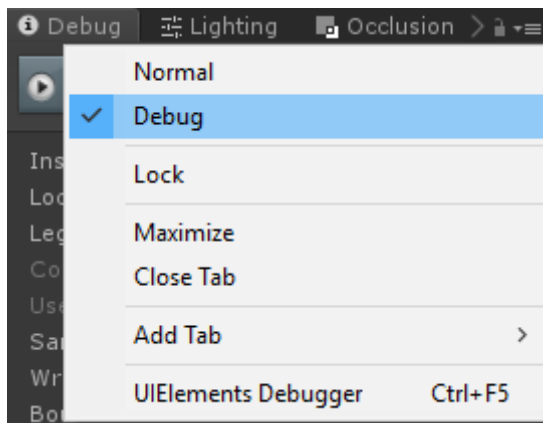
- Changed graphic settings in runtime will be saved to a **config** file.

ADDING NEW PLAYER WEAPONS OR ITEMS

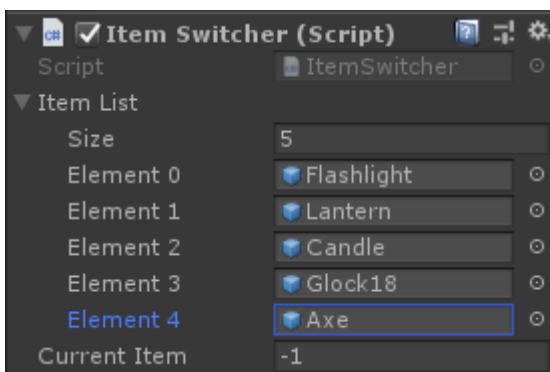
1. Find a **WallDetection** object in **Player** object, which contains all **Weapon/Item** objects.



2. Duplicate one of these items and replace disabled object inside duplicated object with your own Weapon or Item.
3. Mark your **Animations** as a **legacy** animations and add it to your weapon/item root object.



4. Add your new Weapon/Item to an **ItemSwitcher** Item List.



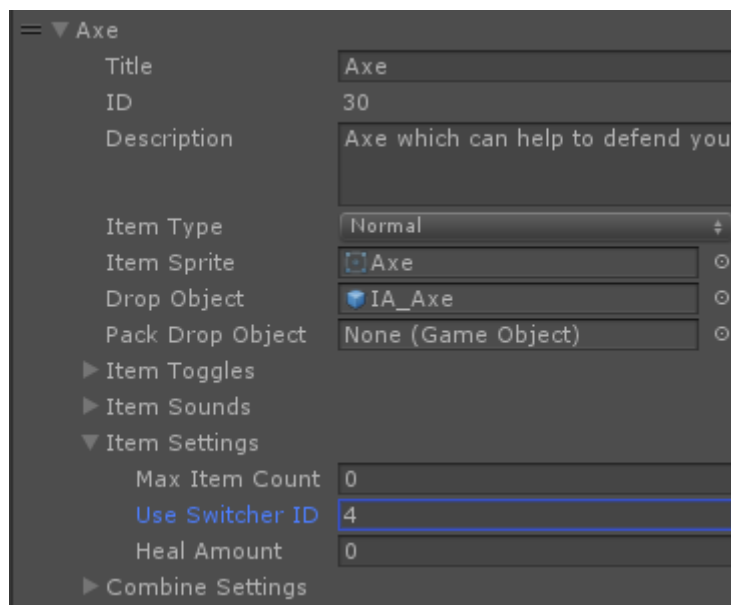
- Custom Weapon/Item script require an **ISwitcher** interface.

```
public class MeleeController : MonoBehaviour, ISwitcher, ISwitcherWallHit
{

```

- Also, if you want extra functions, when you hit wall, you can add an **ISwitcherWallHit** interface.
- These interfaces will automatically create required functions, which will be driven by **ItemSwitcher** script.

5. Do not forget to add your item to an **Inventory Database** and set **Use Switcher ID**.

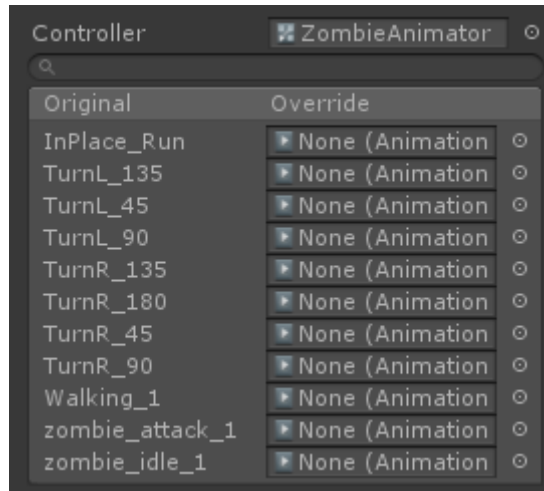


The screenshot shows the Unity Hierarchy panel for an 'Axe' item. The configuration is as follows:

Property	Value
Title	Axe
ID	30
Description	Axe which can help to defend you
Item Type	Normal
Item Sprite	Axe
Drop Object	IA_Axe
Pack Drop Object	None (Game Object)
Item Toggles	
Item Sounds	
Item Settings	
Max Item Count	0
Use Switcher ID	4
Heal Amount	0
Combine Settings	

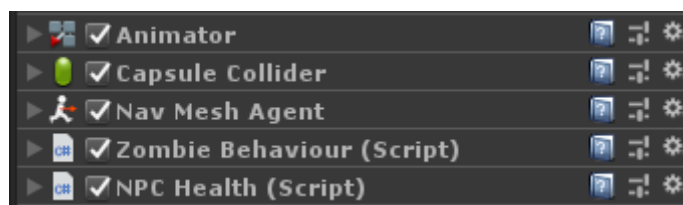
AI ZOMBIE BEHAVIOUR

- If you want to add new animations, just create new **Animator Override Controller** and assign your own animations.

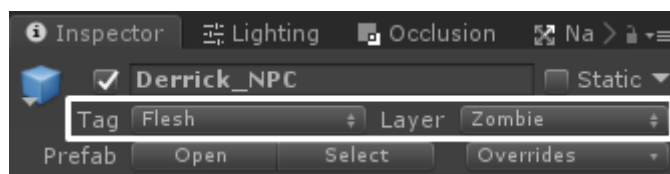


ADDING NEW ZOMBIE

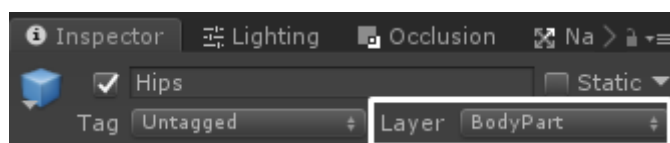
1. Convert your **Zombie Character** to a **Ragdoll** (GameObject -> 3D Object -> Ragdoll).
2. Add required scripts to a Zombie Character.



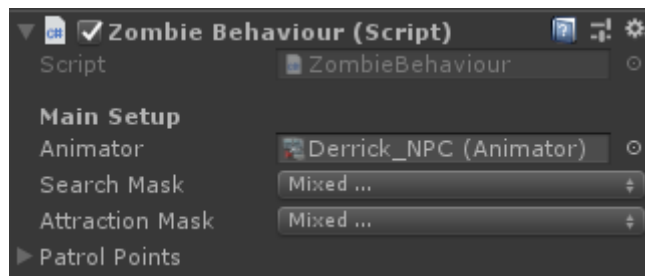
3. Set zombie **Tag** to **Flesh** and **Layer** to **Zombie**



4. Zombie hips **Layer** set to a **BodyPart**

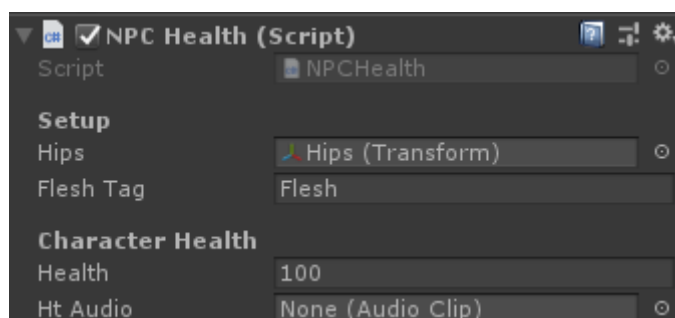


5. In **ZombieBehaviour** script assign zombie Animator and set **masks**



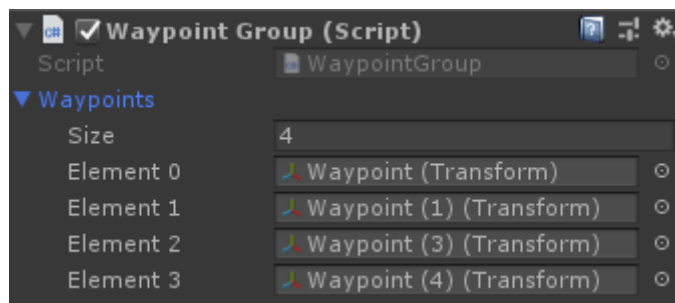
- For example, look at the **Derrick** prefab.

6. Assign **Hips** in **NPCHealth** script and set zombie **Health** points.

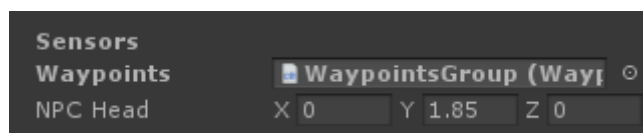


7. Create empty GameObject and add **WaypointGroup.cs** script.

8. By adding empty GameObjects to a object, where is **WaypointGroup** script, you will automatically define new waypoints.

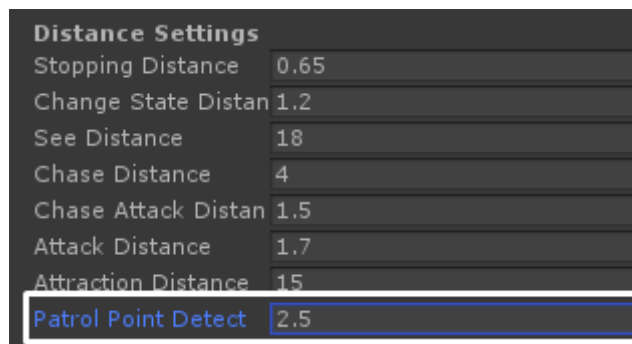


9. Inside the **Sensors** section, assign **Waypoints** and **NPC Head** position.



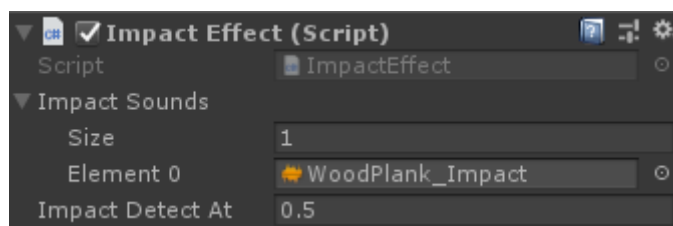
- **ZombieBehaviour** script contains a bunch of settings, with which you can play.

- **Intelligence** slider defines main zombie intelligence settings as is (attracted state, go to patrol point state, look state, sound detect state).
- ❖ **Intelligence 1** = zombie can be attracted and can go to a patrol point.
- ❖ **Intelligence 2** = zombie will turn to a direction where you fired, zombie hears dropped objects.
- **Patrol Point** is point, where zombie go, if distance between last seen position and patrol point position will be in range of **Patrol Point Detect** distance.



ZOMBIE IMPACT SOUND ATTRACT

1. Add **ImpactEffect.cs** script to a draggable object.



- If Impact Volume is greater than **Impact Detect At** distance, impact will attract zombie.

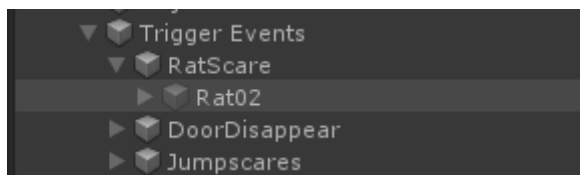
JUMPSCARES

SMALL JUMPSCARE

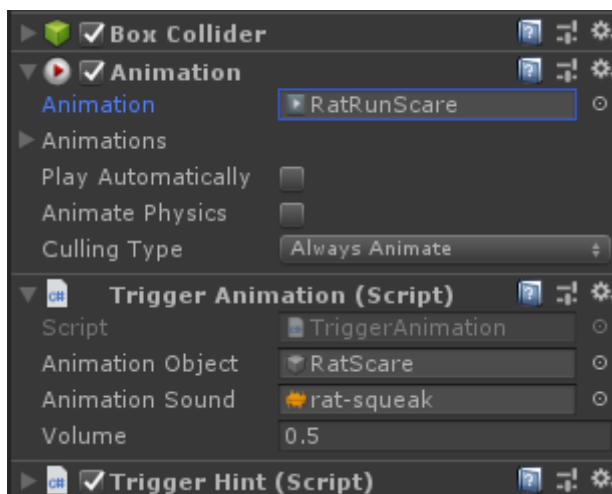
1. Create a **Trigger** object.



2. Create an Empty GameObject and move creature to it.



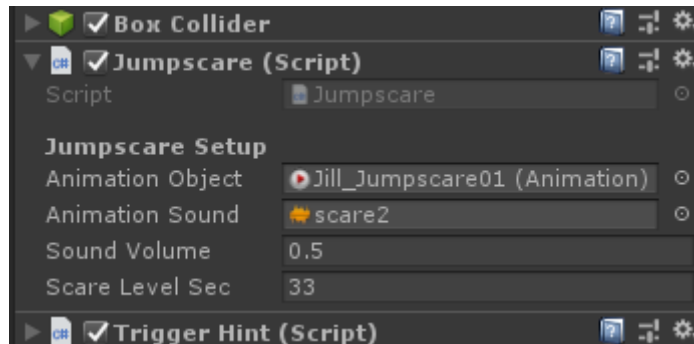
3. Create **Jumpscare Animation** and add **TriggerAnimation.cs** script



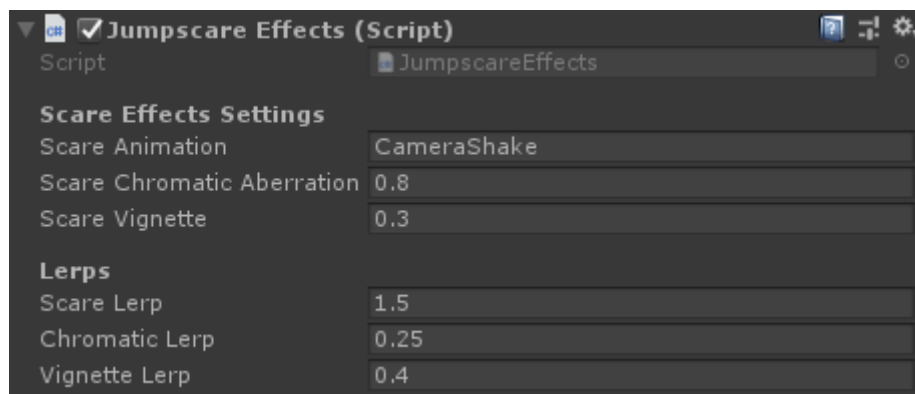
- Also you can add **TriggerHint.cs** script, if you want to show hint when jumpscare animation will be played.

JUMPSCARE WITH EFFECT

- You can create it with same steps as **Small Jumpscare** but instead of adding **TriggerAnimation.cs** script add **Jumpscare.cs** script.



- You can set, how long will be player scared by setting **Scare Level Sec** in seconds (**Scared Breathing**)
- **Jumpscare** script is linked with a **JumpscareEffects.cs** script in MouseLook object.
- Jumpscare effects control **Camera Shake**, **Scared Breath** and **Scare Effects**.

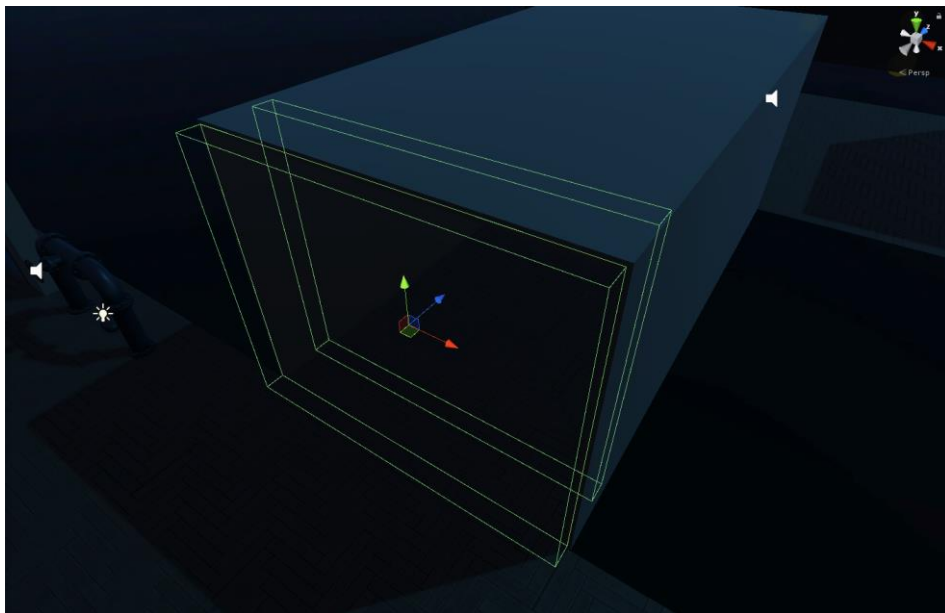


- If you have problems with creating jumpscare animation, you can go to my **Youtube Channel** and watch my [Jumpscare Tutorial](#)

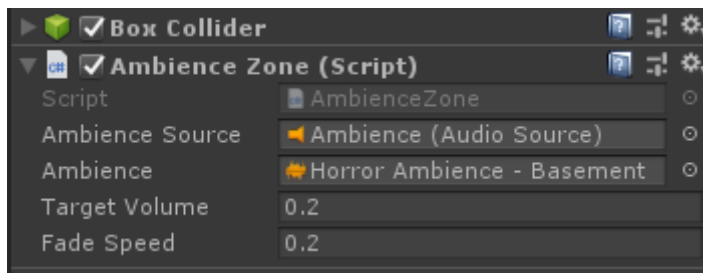
AMBIENCE ZONE

- This will allow you to change game **Ambience Sound**, when you go to particular room.

1. Create two **Trigger** objects.



2. Add **AmbienceZone.cs** script to these objects.

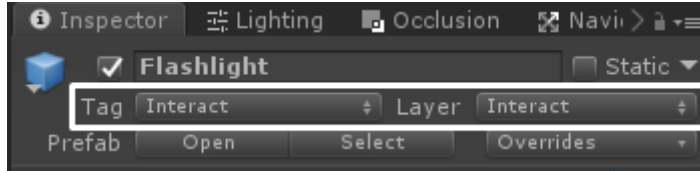


- Assign **Ambience Source** with an audio source from **Player** object.
- To first object set a **default** ambience and to second set the **room** ambience.

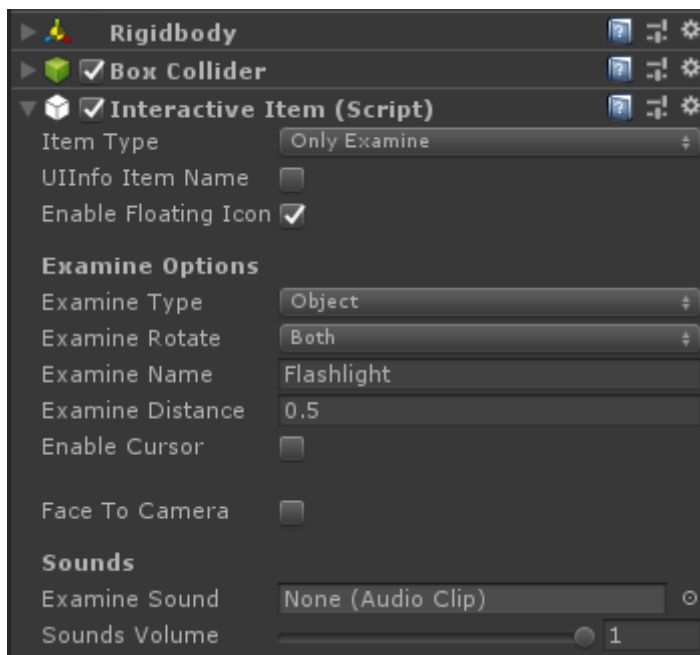
EXAMINE MANAGER

ADDING EXAMINE OBJECTS

1. Set object layer to a **Interact** layer (you can also set object tag to a **Interact**).



2. Add **Interactiveltem.cs** script and set **Item Type** to **Only Examine**



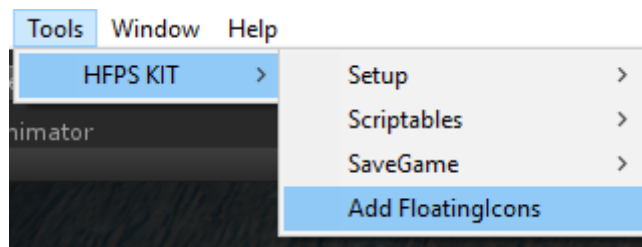
- With **Only Examine** type you can only **Examine Objects**, but you can also set **Examine Type** at the other **Item Types**.
- When you set **Examine Type** to **Advanced Object**, you will get more **Examine Options**.
- This script also allows you to set **Examine type** to a **Paper** type.
- Also there is a feature to set **Examine Rotate** directions.
- With **Face To Camera** feature you can set object starting rotation.

FLOATING OBJECTS

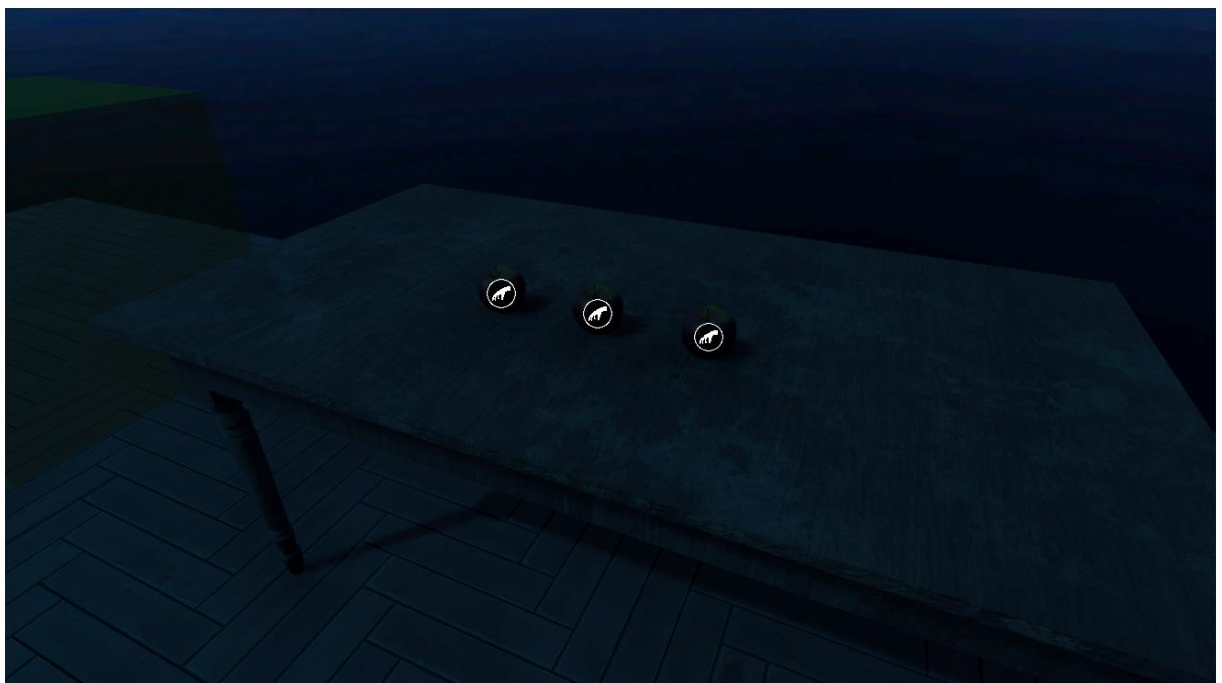
- **Floating Object** feature allows you to better find a hidden objects or mark interactable objects.

ADDING NEW FLOATING OBJECTS

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



- This will automatically add all selected objects to a **Floating Icons** list.



HOW TO UPDATE POSTPROCESSING?

- If you want to update **PostProcessing v2** to new version, you need to know these facts.
- To do not cause a script errors, backup a **PostProcessing Custom Shaders** folder located inside PostProcessing folder (**PostProcessing/Shaders/Custom**).
- Open new empty scene, remove old PostProcessing version and replace it with new version.
- Move backup of **Custom Shaders** folder to same location as before!
- Always use **PostProcessing** which is inside **Assets** folder and not from **PackageManager!**
- [Latest PostProcessing](#)

FIXES FOR MOST KNOWN ERRORS

PACKAGE MANAGER ERROR

Library\PackageCache\com.unity.package-manager-ui@2.0.7\Editor\AssemblyInfo.cs(7,12): error CS0246: The type or namespace name 'XmlNamespacePrefixAttribute' could not be found (are you missing a using directive or an assembly reference?)

- If you met with this or similar error, you don't need to worry, this error is caused by **Unity Package Manager** and can be fixed easily.
- You can simply fix this error by going to **Help -> Reset Packages to defaults**.

SCRIPT INCOMPATIBILITY WITH .NET

- If you met with error about script incompatibility or error which saying that script is not a part of a C# 4.0 language specification, you need to follow these steps below.
- **This can also fix a Save/Load problems.**
- You need to go to **Edit -> Project Settings -> Player** and set **Scripting Runtime Version** to **.NET 4.x Equivalent** and **Api Compatibility Level** to **.NET 4.x**, then you need to **restart Unity**. **With every change you need to restart Unity!**

BUG, ERROR REPORT

- If you found bug or error, please send a message to our email address:
thunderwiregames@gmail.com
- Or visit our website [Customer Support](#) page or [Contact Page](#)

CREDITS

- This kit is developed and designed by **ThunderWire Studio** © All Rights Reserved.
- Almost all assets are created by **ThunderWire Studio**, besides ones downloaded with royalty free license.

LINKS

- [ThunderWire Studio](#) - Youtube Channel
- [ThunderWire Studio](#) - Developer Website
- [ThunderWire Studio](#) - AssetStore
- [Horror FPS KIT](#) - Website
- [Horror FPS KIT](#) - AssetStore

