GAME KIT CONTROLLER 3.03-3

By Two Cubes

Table of content

INTRODUCTION

Thanks for your purchase and interest in the asset, you won't regret it. The main purpose of this asset is to learn every aspect of a videogame and at the same time, make a solid controller, so anyone can make a quick prototype and try any idea, focusing on the game itself rather than spend time in systems or components.

Also, this asset will be updated forever or until it is perfect and allows to create literally any type of game, what happens first.

WARNING: THIS DOCUMENTATION COVERS THE FIRST STEPS AND BASIC USE OF THE MAIN COMPONENTS.

NEW VIDEO TUTORIALS ARE BEING MADE EVERY WEEK FOR THE ASSET, WHICH CAN BE FOUND ON THIS LINK:

https://www.youtube.com/watch?v=r0cKbNYUCZA&list=PLYVCbGEtbhxVjZ9C41fwTDynTpVkCP9iA

THERE IS AN OLD VERSION OF THE DOCUMENTATION WHICH IS NOT COMPLETE FOR THE LAST ASSET VERSION, LOCATED ON THE SAME FOLDER THAT THIS FILE, CALLED " Game Kit Controller Documentation 2.4d (Old version)".

THIS OLD DOCUMENTATION WILL BE UPDATED SOON AND IS CURRENTLY IN PROCESS. EVERY BULLET ELEMENT BELOW IS TO SHOW HOW THE ASSET ORGANIZES AND ALL OF THEM WILL BE EXPLAINED IN EVERY DOC UPDATE.

FOR NOW, THIS IS THE LATEST VERSION OF THIS FILE, WHICH WILL INCLUDE TUTORIALS FOR EVERY SYSTEM IN THE ASSET.

THIS FILE WILL BE UPDATED IN THE FORUM OF GKC EVERY WEEK UNTIL IS TOTALLY COMPLETE. YOU CAN FIND THE FORUM HERE:

https://forum.unity.com/threads/game-kit-controler-1st-3rd-controller-with-weapons-vehicles-2-4-released.351456/

OF COURSE, IN EVERY UPDATE THE LAST VERSION OF THIS DOC WILL BE INCLUDED.

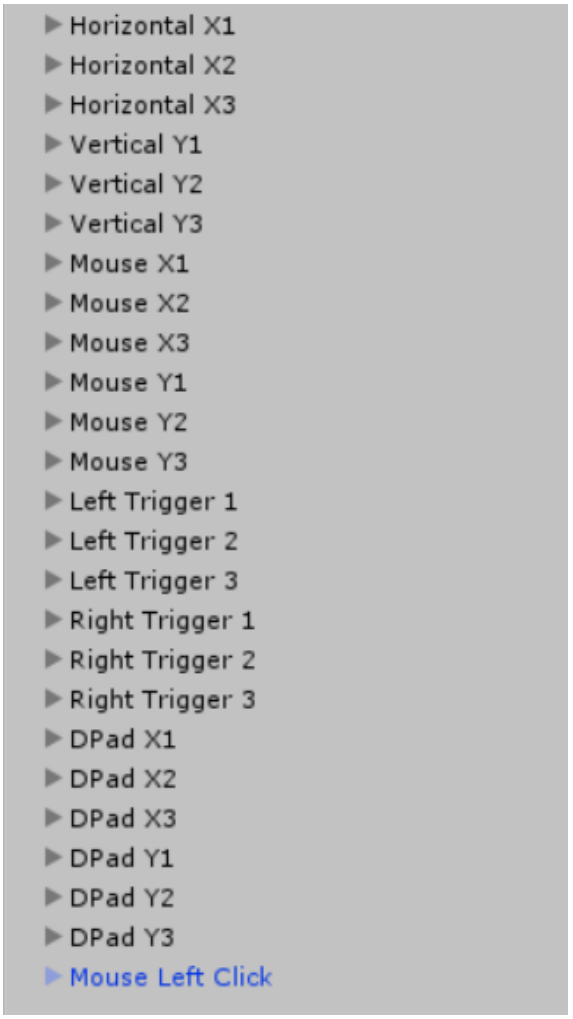
LAST DOCUMENT UPDATE: 10/06/22

FIRST RUN (IMPORT GKC WITHOUT PROJECT SETTINGS)

You can import this asset into an empty project or a project with previous content. If you import the asset into an existing project, you can uncheck the option to import the project settings, but in this case, you have to do the next (else, jump to First steps and basic use):

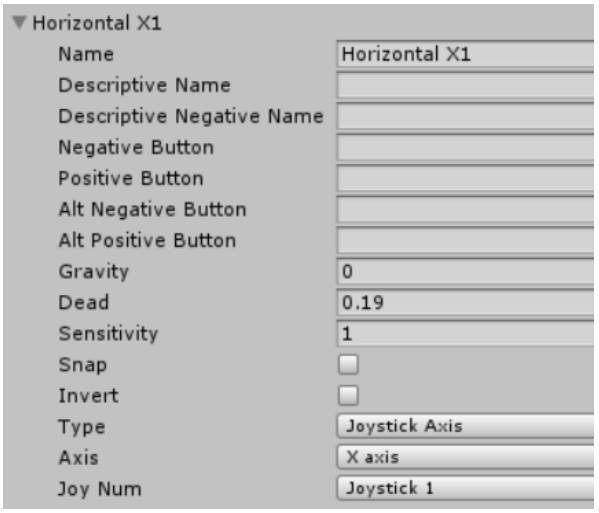
ADD AXES IN THE INPUT MANAGER

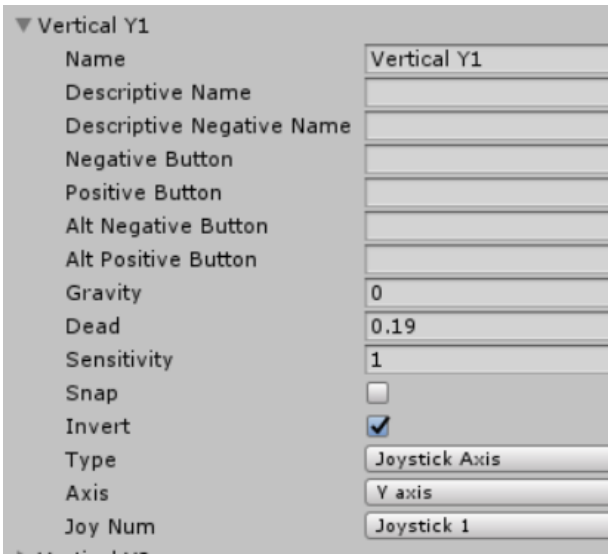
If you import the asset into a project without the Project.Settings, you have to add these axes (inside the yellow color) into the Input Manager of unity if you are going to use a gamepad, in Edit->Preferences->Input. These axes are used for the gamepad, but it needs to be configured if the settings of the asset are not imported (This step won't be necessary in the next update).

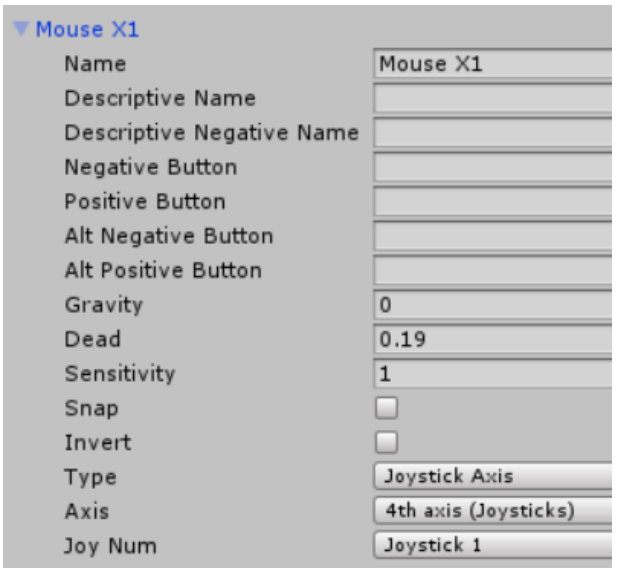


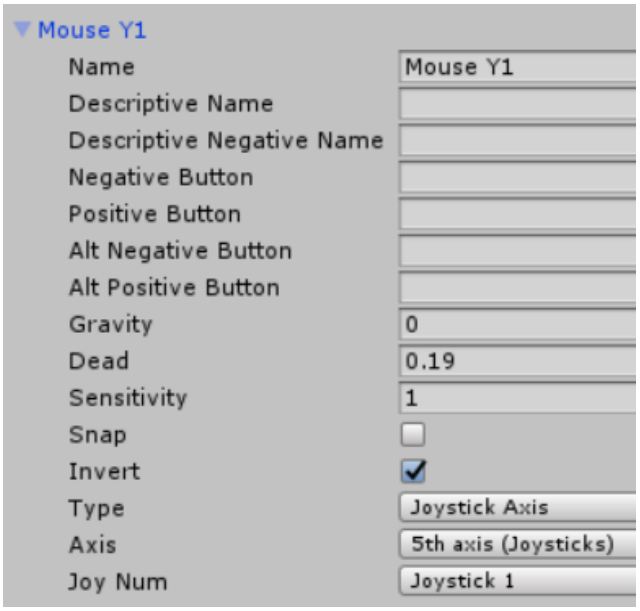
These are the input added for 4 players using gamepad, but you can add just one if you are not going to use the local multiplayer.

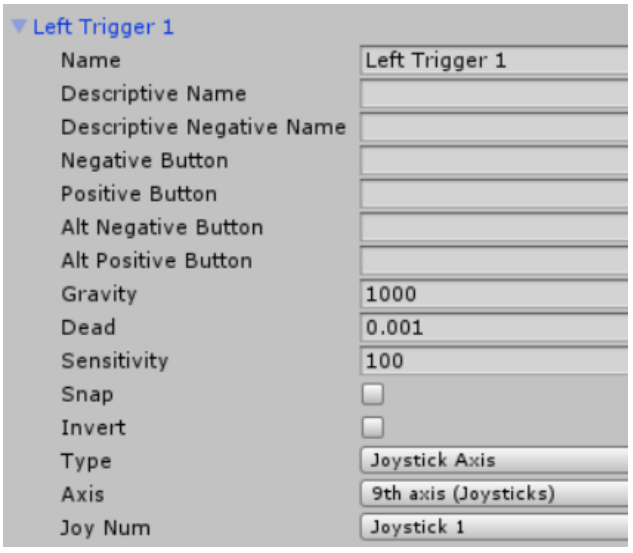
The values to configure are the following:

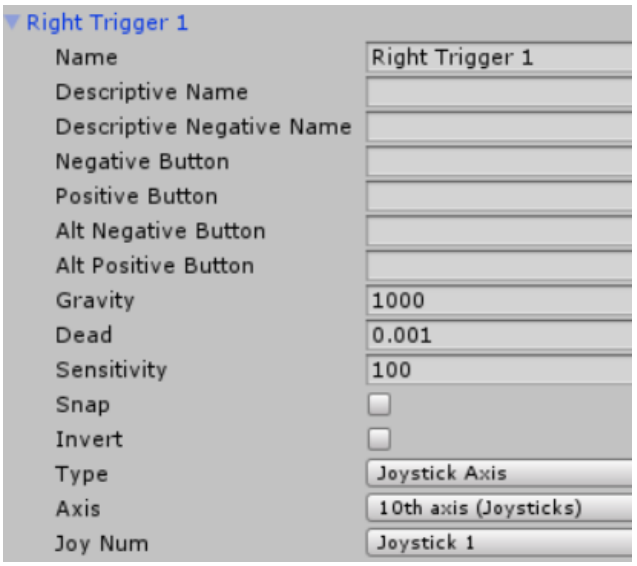


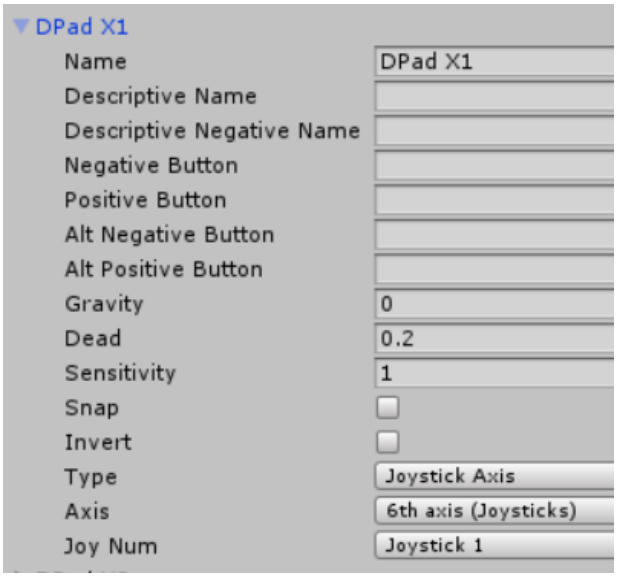


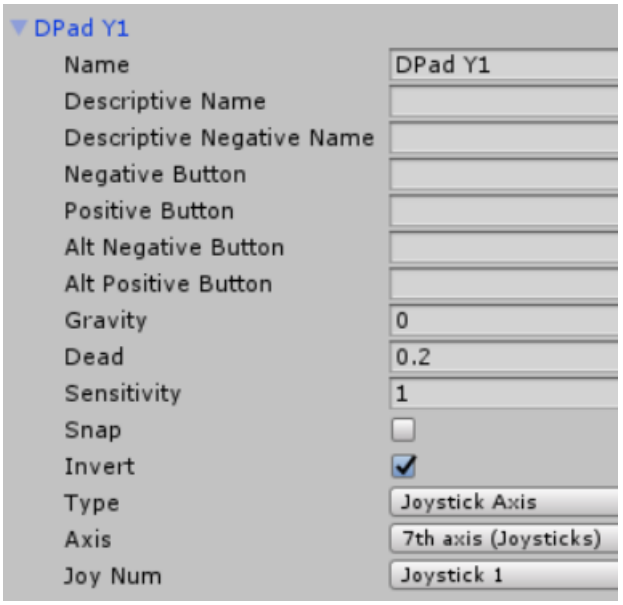


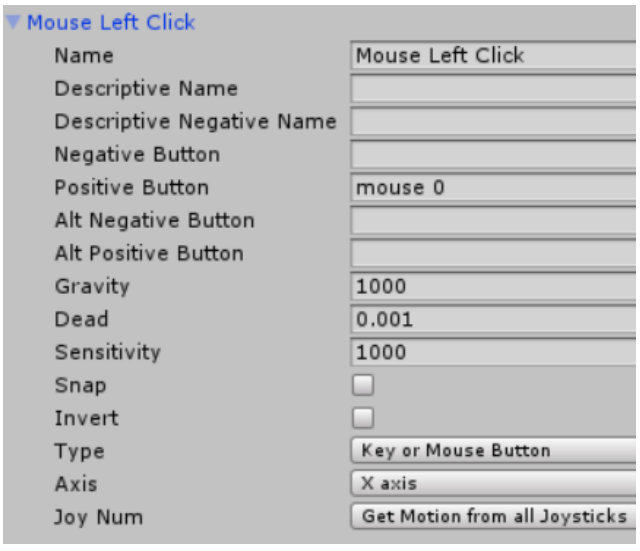

Mouse Y1
Name
Mouse Y1
Descriptive Name
Descriptive Negative Name
Negative Button
Positive Button
Alt Negative Button
Alt Positive Button
Gravity
0
Dead
0.19
Sensitivity
1
Snap
Invert
Type
Joystick Axis
Axis
5th axis (Joysticks)
Joy Num
Joystick 1


Left Trigger 1
Name
Left Trigger 1
Descriptive Name
Descriptive Negative Name
Negative Button
Positive Button
Alt Negative Button
Alt Positive Button
Gravity
1000
Dead
0.001
Sensitivity
100
Snap
Invert
Type
Joystick Axis
Axis
9th axis (Joysticks)
Joy Num
Joystick 1


Right Trigger 1
Name
Right Trigger 1
Descriptive Name
Descriptive Negative Name
Negative Button
Positive Button
Alt Negative Button
Alt Positive Button
Gravity
1000
Dead
0.001
Sensitivity
100
Snap
Invert
Type
Joystick Axis
Axis
10th axis (Joysticks)
Joy Num
Joystick 1


DPad X1
Name
DPad X1
Descriptive Name
Descriptive Negative Name
Negative Button
Positive Button
Alt Negative Button
Alt Positive Button
Gravity
0
Dead
0.2
Sensitivity
1
Snap
Invert
Type
Joystick Axis
Axis
6th axis (Joysticks)
Joy Num
Joystick 1


DPad Y1
Name
DPad Y1
Descriptive Name
Descriptive Negative Name
Negative Button
Positive Button
Alt Negative Button
Alt Positive Button
Gravity
0
Dead
0.2
Sensitivity
1
Snap
Invert
Type
Joystick Axis
Axis
7th axis (Joysticks)
Joy Num
Joystick 1

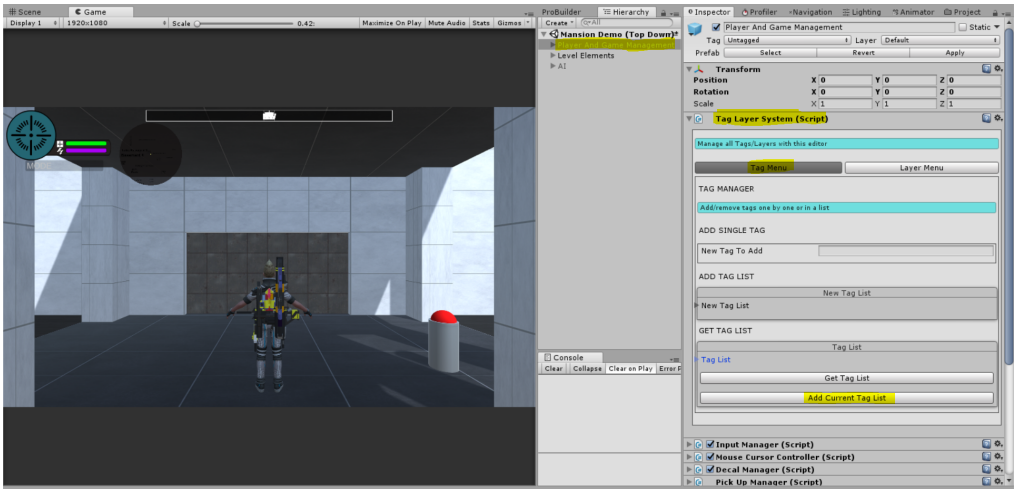

Mouse Left Click
Name
Mouse Left Click
Descriptive Name
Descriptive Negative Name
Negative Button
Positive Button
mouse 0
Alt Negative Button
Alt Positive Button
Gravity
1000
Dead
0.001
Sensitivity
1000
Snap
Invert
Type
Key or Mouse Button
Axis
X axis
Joy Num
Get Motion from all Joysticks

Right now, the input for 4 players have been configured, to being able to use the local multiplayer system. The settings are the same as above but configuring Joystick 2, 3, etc… and you can add as many gamepads as you need (though the limit is 16 players if I remember properly).

ADD THE TAGS AND LAYERS

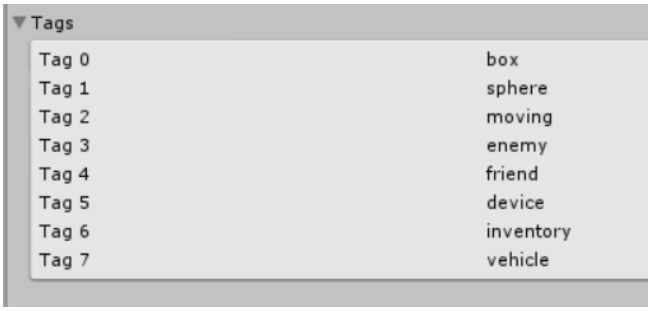
In case you import the asset to a project without the Project.Settings, you need to add the tags and layers used in this asset. To do this automatically, you need to go the demo scene in the asset, or drag and drop the prefab **Player And Game Management**. Then, go to the inspector of that gameObject and go to **Tag Layer System** inspector.

Open the Tag Menu and press the button Add Current Tag List.

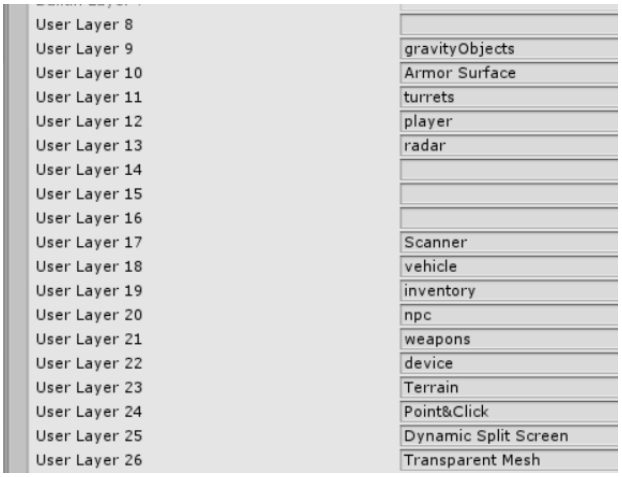


Do the same with the layers, pressing **Layer Menu** and the **Add Current Layer List**. After that, all the tags and layer used in the asset are assigned correctly to every object and prefab of the asset.

This is the full list of tags:



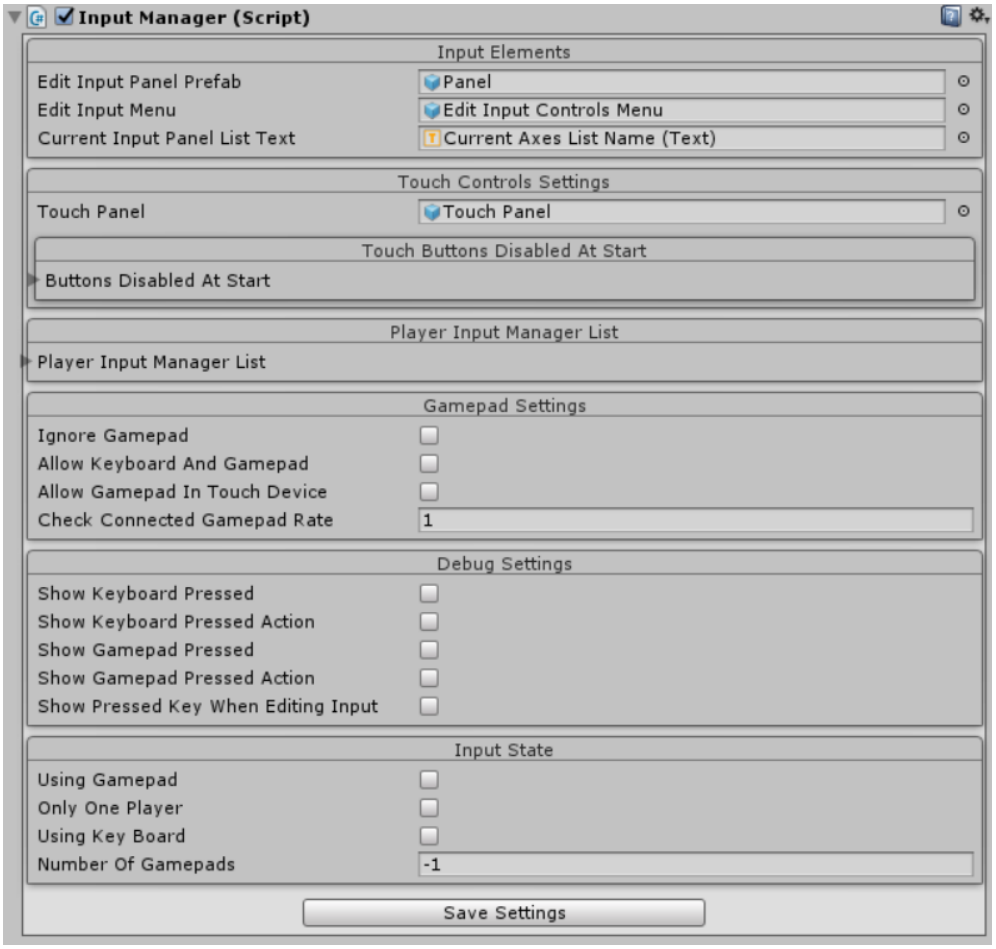
And this is the full list of layers:

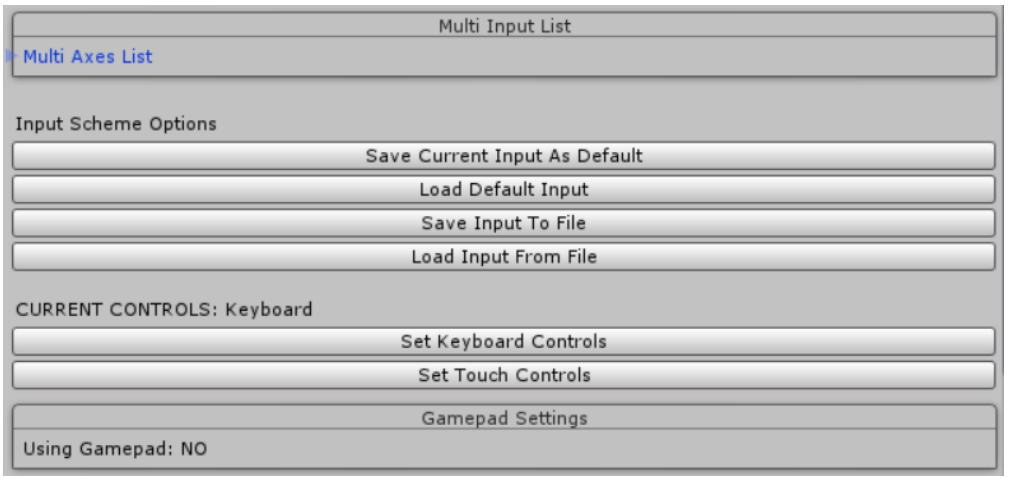


FIRST STEPS AND BASIC USE

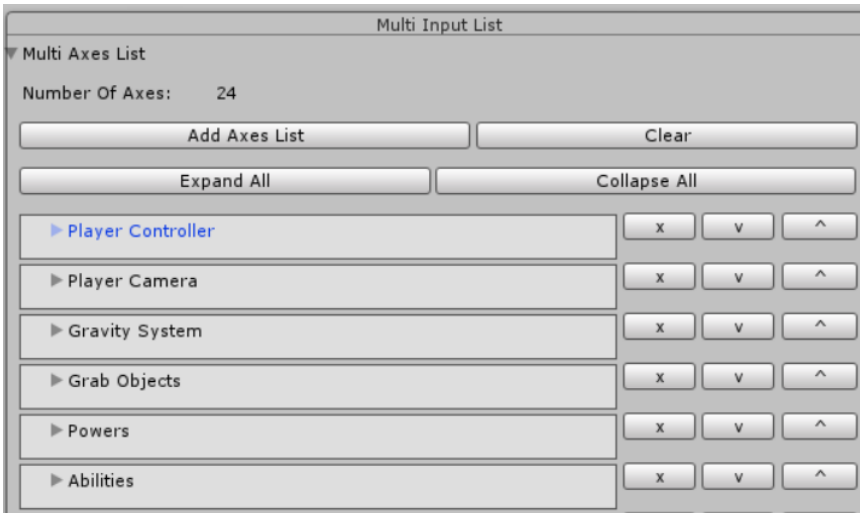
INPUT MANAGER

You can check all the actions in the pause menu by pressing Esc, then select Edit Control Input. In that window you can see all the actions defined in the custom **Input Manager**. This component is assigned in the gameObject **Player And Menu Management**.

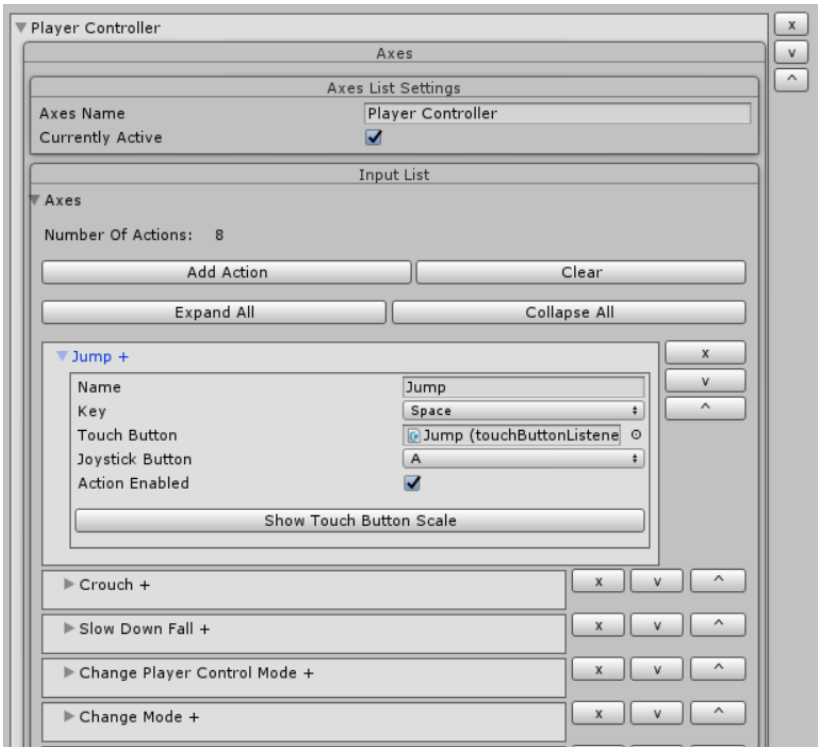




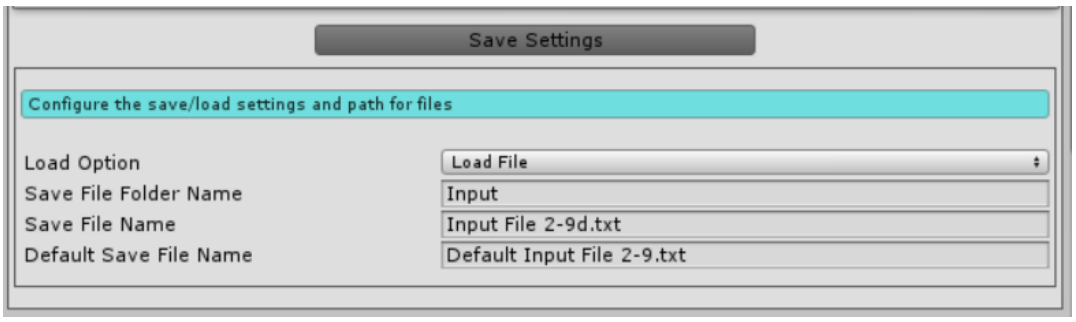
In the inspector you can add and remove as many groups of actions as needed. These groups are divided into Axes, so the input of the controller can be organized into a separate group from the camera, another group for weapons, powers, menus, vehicles, etc… For this, go to **Multi Axes List** and open it:



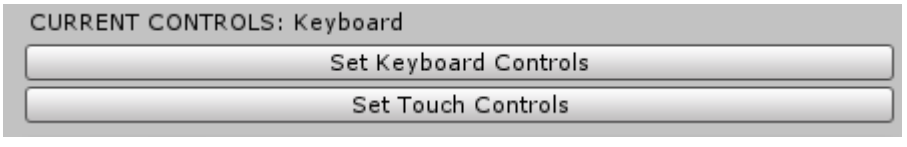
Inside every group, you can add, remove, enable and disable every action and assign a keyboard button, a gamepad button and a touch button.



Also, you can edit the path to save the current map button, by pressing the button **Save Settings**. This file will be stored in the persistent path by default or in a relative path to your project folder if you enable the option “Use Relative Path”. IMPORTANT: if you are making a game for mobile, set the option Use Relative Path to false.



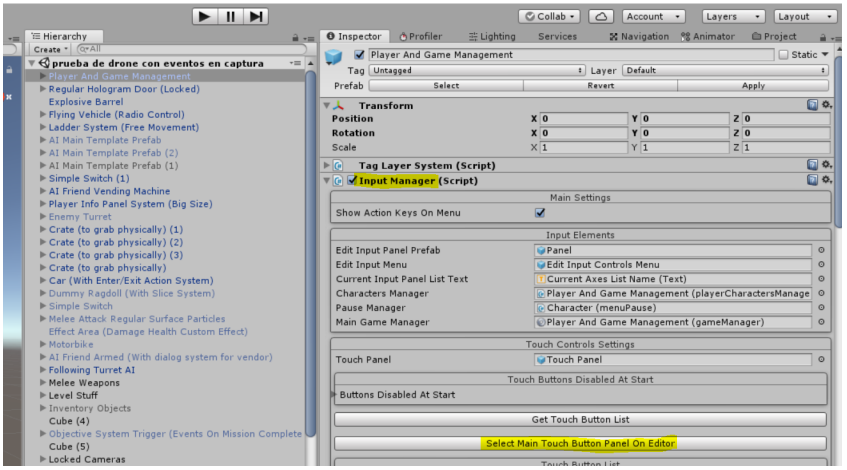
In the options below, you can select the keyboard or touch controls as input when the game starts. Also, in game, you can switch between these two modes by opening the menu with Escape and pressing the button **Switch Controls Type**. If you are in this touch controls mode, press Esc and then press **Switch Controls Type** again to change to keyboard controls. This option can be used in both modes. The label Current Controls shows what current control scheme is currently selected.



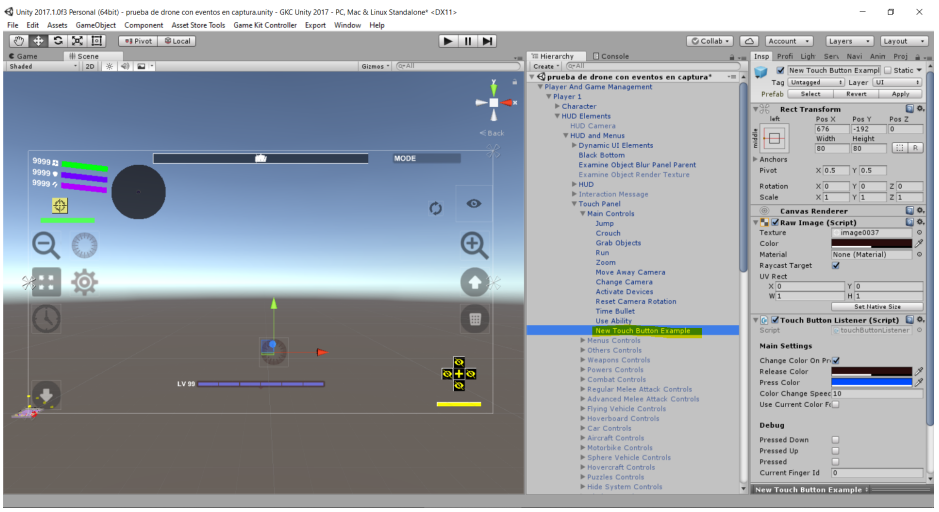
ADD A NEW TOUCH BUTTON

For this process, you can either configure a new input action or assign a touch button to an input action already configured.

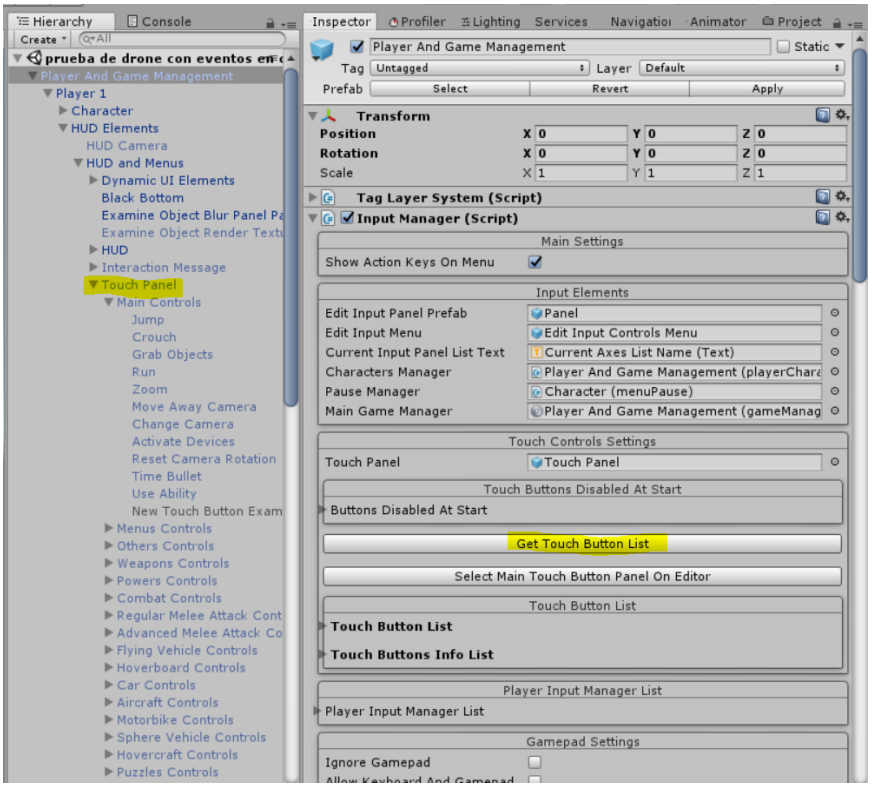
For this, go to the input manager component (from GKC) and press the button "Select Main Touch Button Panel On Editor".



Then, select the panel where you want to add a new touch button, or create a new panel and add the new touch button there. You can enable the touch button panel (and disable the rest of touch panels, so you can see better the main panel that you are configuring) and place the new button where you need on the screen, along with customizing its size, texture, color, etc...



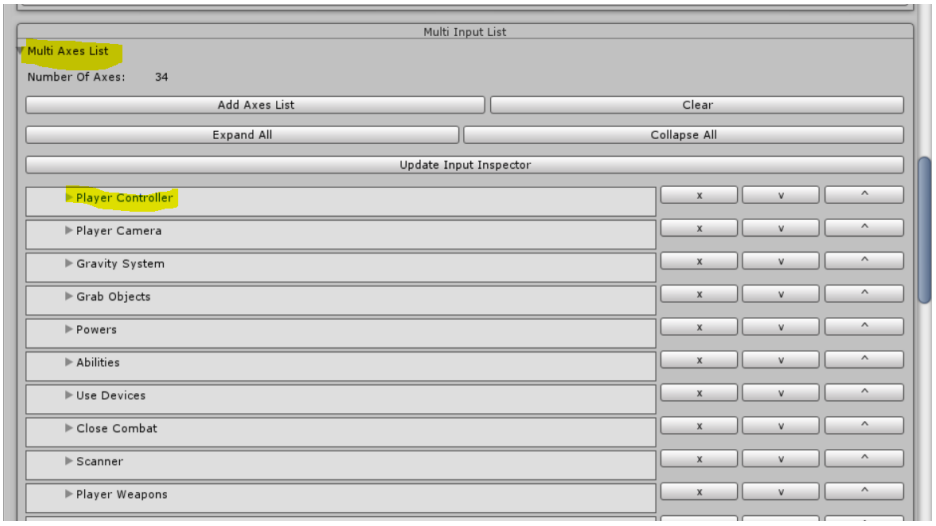
Now, press the button "Get Touch Button List". You will see that the main touch panel has been disabled again to hide it on the screen until it is used ingame:



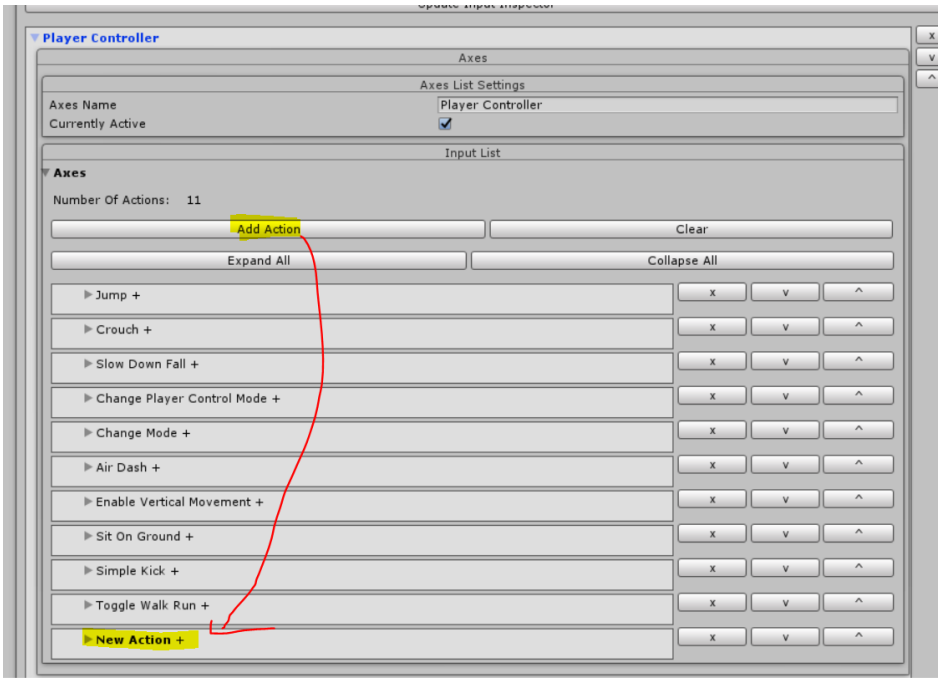
From there, in each input action, you can select the category of the touch button panel, and the button name. Now that a new button has been added, this will also appear when selecting the category of the touch buttons where this new button has been added (the category is basically the main panel where the touch button has been added, in the above example, it would be the “Main Controls” category, which is the name of the panel it self.

Each panel is like the category for regular controls, weapons, powers, examine devices, use photo mode, etc… so through event options, you can change the current touch control scheme (you have categories, and you can define control schemes, which are combinations of these touch panels) to combine for example regular controls with weapons and similar.

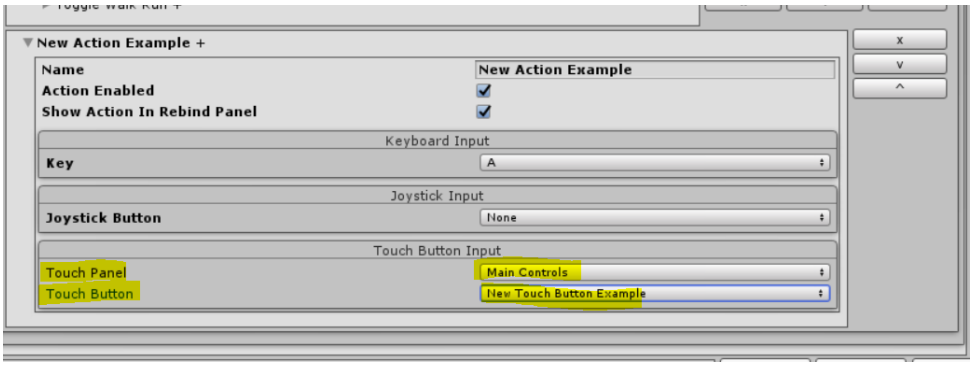
Back to configure the touch button in a specific action, open the Multi Axes List and select the category of input and the action input it self to set the new touch button:



For this example, let's add a new action and set the new touch button there:



So in the Touch Panel field, you select the category of the touch controls where the new touch button was added and the new touch button it self:



And now, that touch button will activate that input on play mode and in build as well. If you assigned that touch button to a previous action, there are no more extra steps. If you added a new action, you only need to follow the steps in the previous part of this doc to set the action events which that button will trigger in the player input system component, in player controller gameObject.

You can also check the tutorial video for this process, which you can see here:

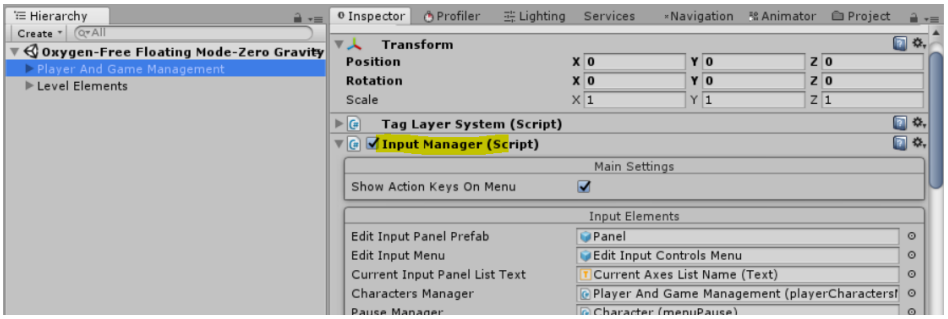
https://www.youtube.com/watch?v=_7C7wYB0iIA

CHECK AND FILTER INPUT ACTIONS AND KEYS/GAMEPAD USED FOR IT

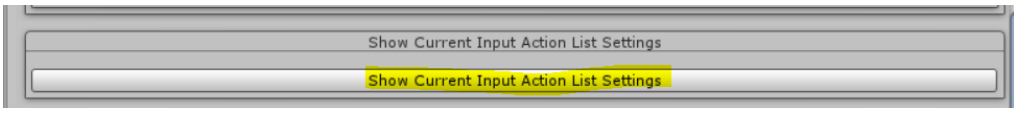
The custom system of input on GKC has an option which allows to show all the input actions configured so far on the settings, including to show the name of each action, its category and the input key used for it, including keyboard and gamepad as well.

This allows you to see better all the actions that your current game uses and the free keys that haven't been assigned yet.

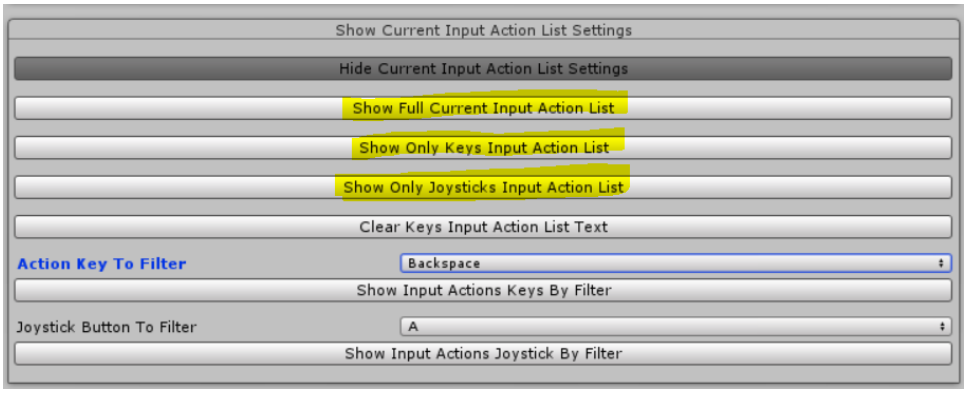
To see this, go to the Input Manager component in the main player's prefab parent:



And in the bottom section, you will find the button "Show Current Input Action List Settings":

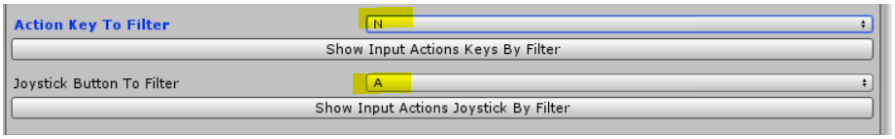


Press and select the input actions that you want to see, gamepad or keyboard and press the button for either of both options to show all the input configured:

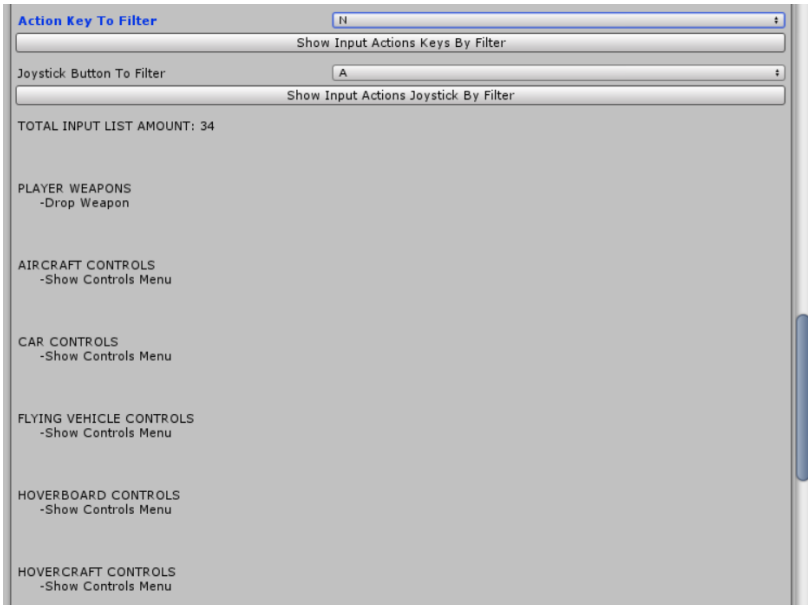


This will show each category of input, each action input and the keyboard key and/or gamepad button on each action:

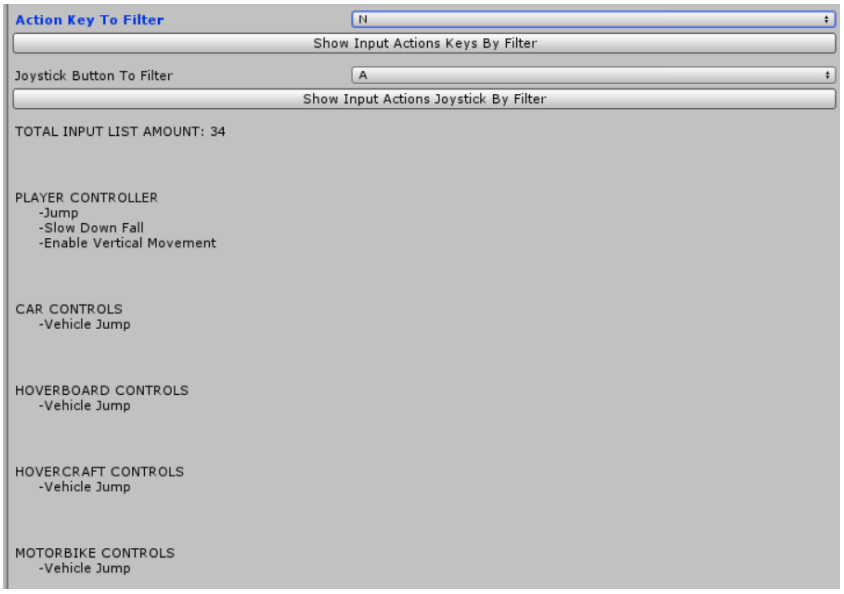
You can also filter the actions to show by keyboard key or gamepad button in the buttons below, by selecting a keyboard key or a gamepad button on the fields “Action Key To Filter” and “Joystick Button To Filter”:



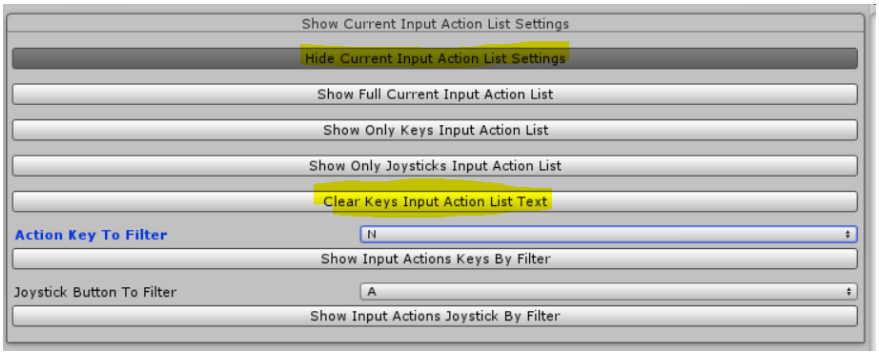
Here an example searching for key “N” on keyboard:



And here searching for button “A” on gamepad:



You can also clear the info shown in the button “Clear Keys Input Actions List Text” and close this panel in the “Hide Current Input Action List Settings”:

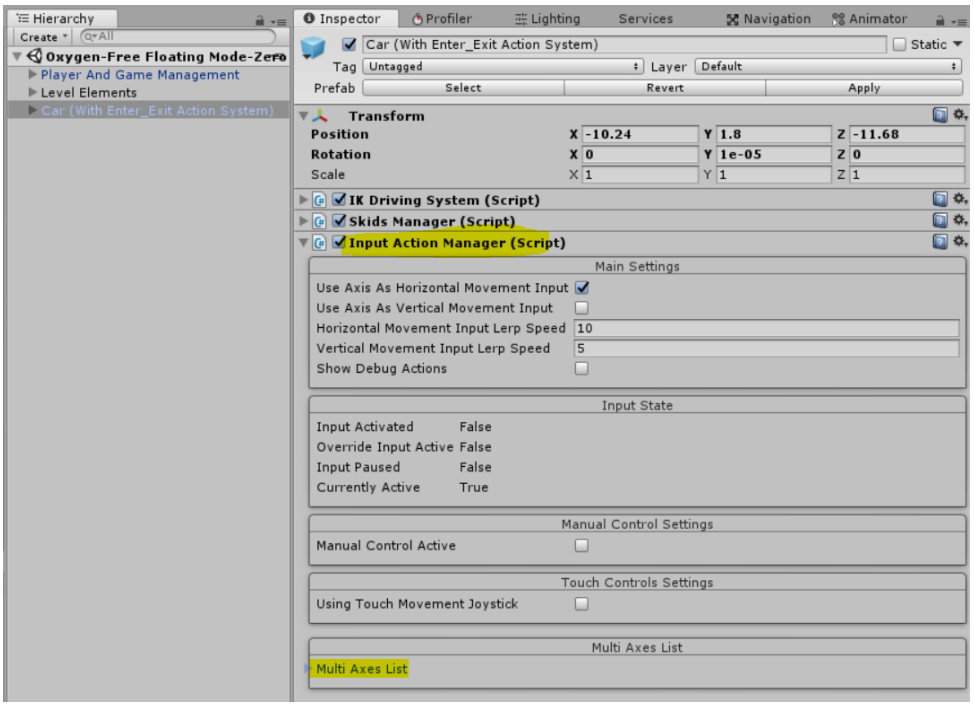


CONFIGURE INPUT ON VEHICLES

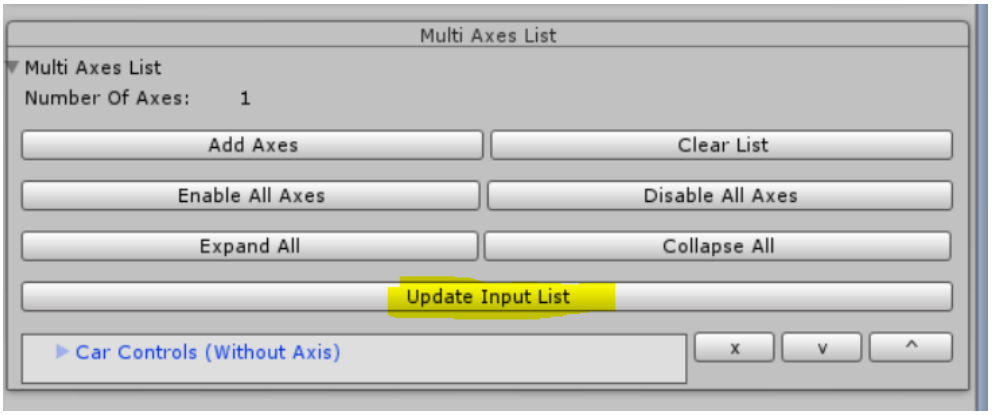
The customization of input on vehicles is almost exact to the process for the settings of input in the Player Input Manager component in the player.

First of all, you basically need to add or configure the input actions on the Input Manager component of GKC.
It is recommended to add a new input action category for each new vehicle, if the new one that you want to add has a different input or special actions not present in the previous one. Of course, you can add the new actions in a previous category of vehicle input as well.

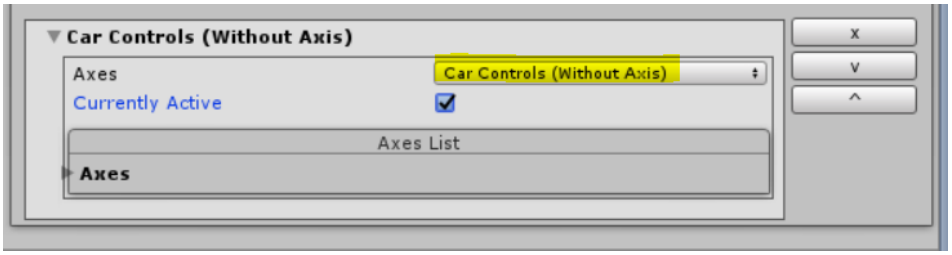
Once you have the new actions or the new category added to configure a new vehicle control, go to any of the GKC vehicles and in the main object it self, go to the component called “Input Action Manager”, more specifically, in the “Multi Axes List”:



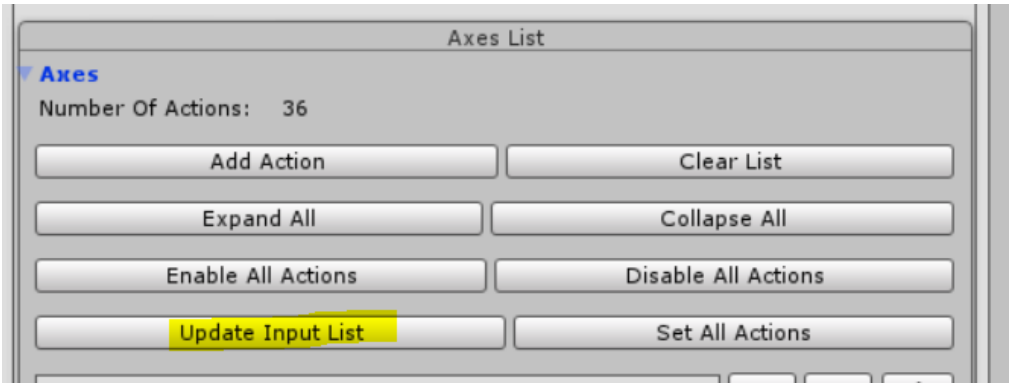
If you added a new input category in Input Manager or just added or modified some input actions, press the button “Update Input List”:



Now, in the field “Axes” below that button, select the input category for this vehicle, in the case you added a new one. You will see a list with all the input categories configured in Input Manager, so select one of them. If you just added or modified input actions, no need to select a different input category in that field:

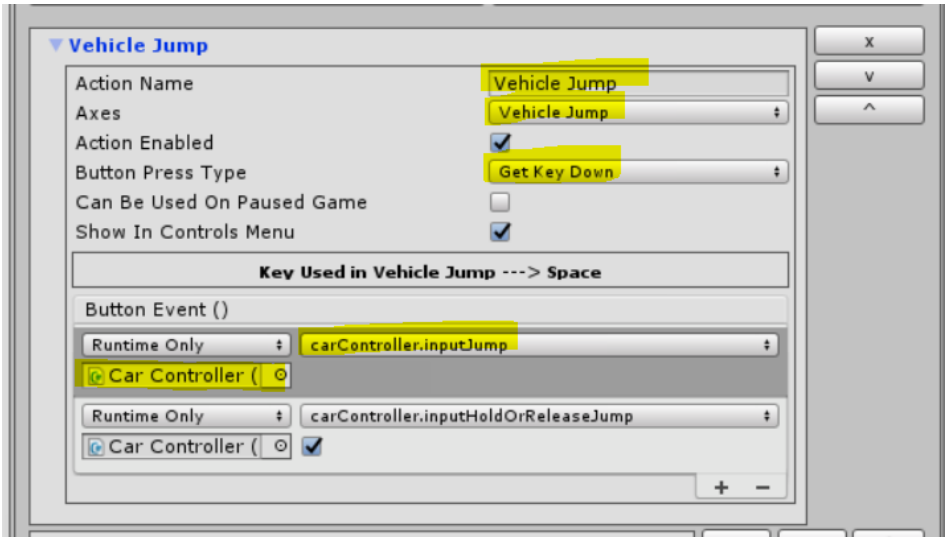


Now, inside Axes, press the button “Update Input List”:



And now, select each input that is on this list, and remove, edit or add new input actions based on what you need. On each input action, you only need to select the input action name on the field “Axes” according to the name of the input from Input Manager, and the type of press to use to activate this action (down, hold or up).

Finally, configure or add a new event in the “Button Event”, drop the object with the component that has the function to activate, and configure the function it self on the event field. You can set as many events as you need on each input action:



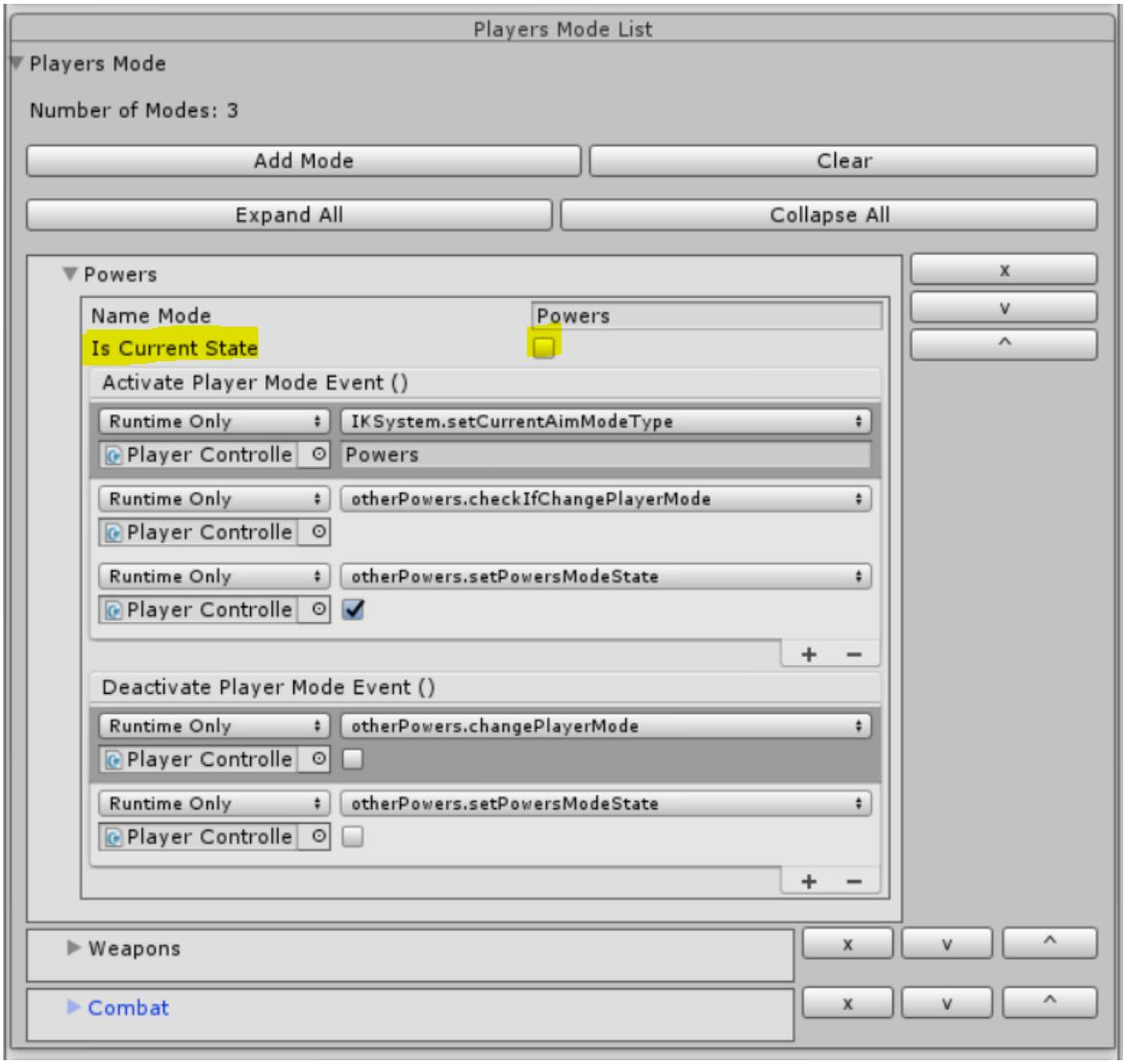
Don’t forget to set a name for the action in the field “Action Name” as that name will be shown in the ingame control panel for that vehicle (it can be opened with N by default, and it can be disabled as well, to not be used).

And the same applies for each of the rest of the actions. Make sure to press the buttons to update the input action list if you add/remove/modify the input action on the Input Manager of GKC for the vehicle that you are going to use.

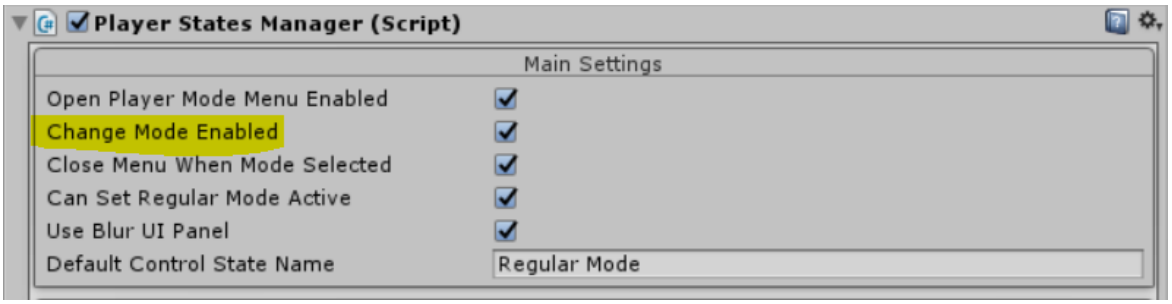
PLAYER CONTROL MODE

The player has 3 modes: weapons, powers and combat. To change between them, press H (by default). You can see the current state by checking the name in the upper left corner of the screen, below the health and energy bar.

The weapon mode is the default mode. In **Player States Manager inspector** (in the gameObject Player Controller) you can choose the default mode at the beginning of the game.



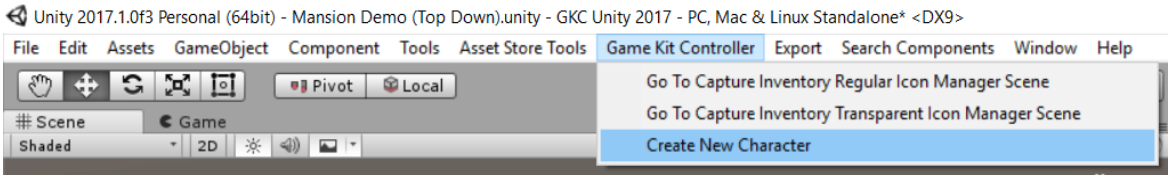
Also, you can configure if the player can change between these modes or not, so the game can always has the same mode or all of them. This option is called Change Mode Enabled.



CREATE A NEW CHARACTER

To set your model, follow these steps:

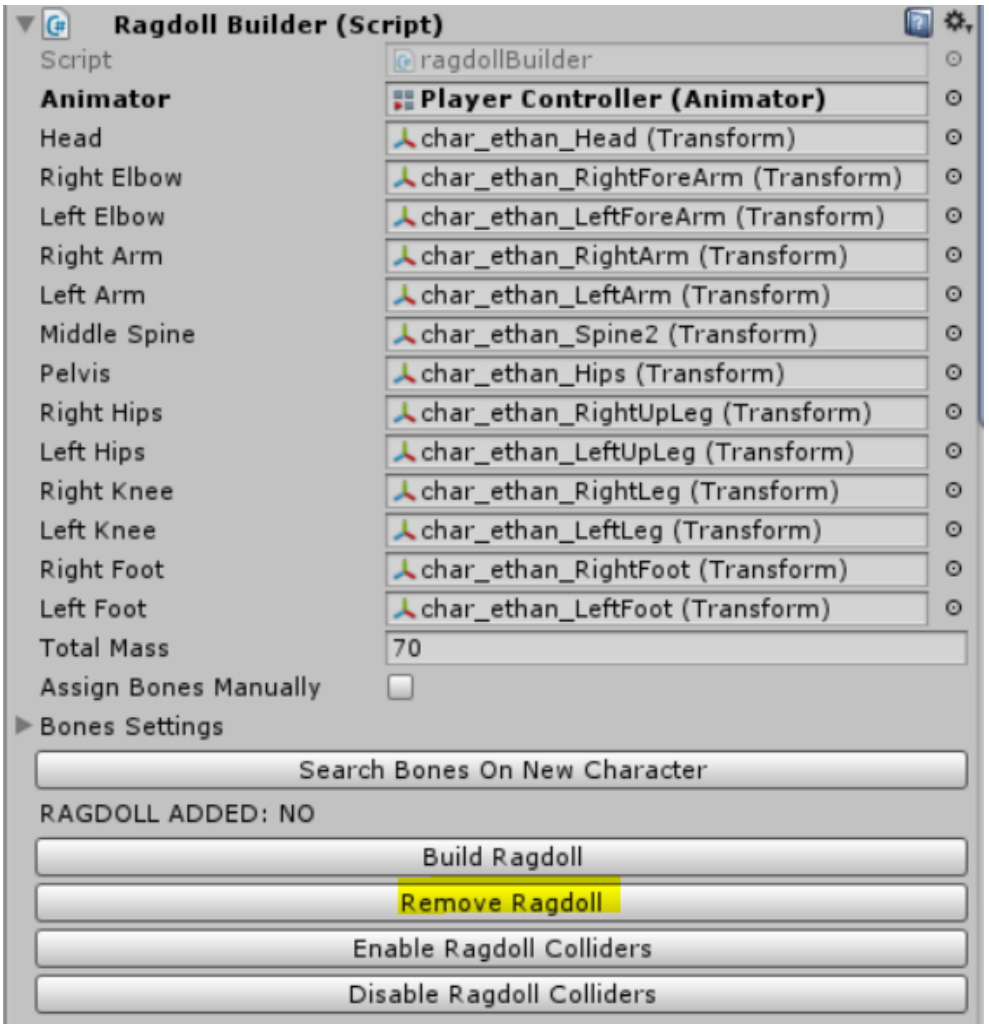
- In the tools bar press **Game Kit Controller** and **Create New Character**.



- In that window select a humanoid model and press the **Create Player** button. If you are in the Scene window, set the camera in the position where you want to place the player once is created.

- The new player will be placed in the scene, in the position where the editor camera is looking.
- This process creates the ragdoll of the new player automatically. Here you can see how it looks.

- This ragdoll can be configured or removed. For this go to **Ragdoll Builder** inspector, in the GameObject called “Character” and press the button Remove Ragdoll. You can also configure its scale and mass in the option Bone Settings.



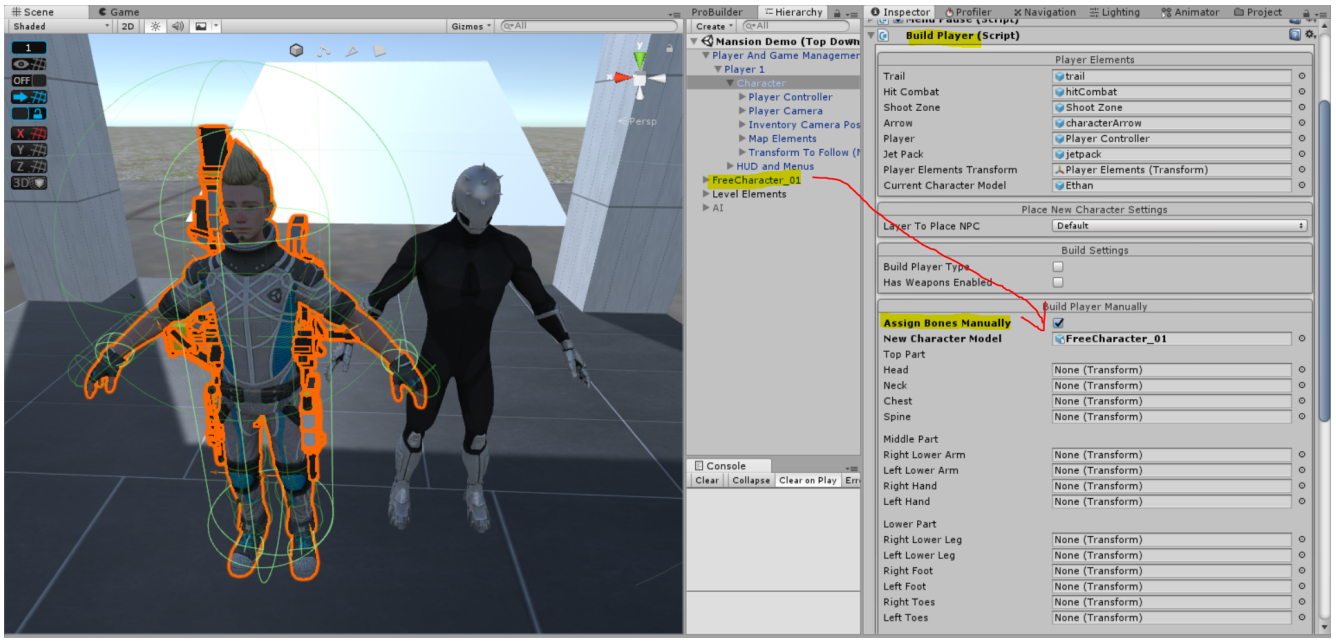
- Then you will see the messages in the console about the weapons.

- These messages is to tell the user to configure the correct weapon parent, which is the place where every weapon is parented while is not being used, like the back, the leg, etc… By default, the weapon is placed in the back. To configure a better parent check the next steps.

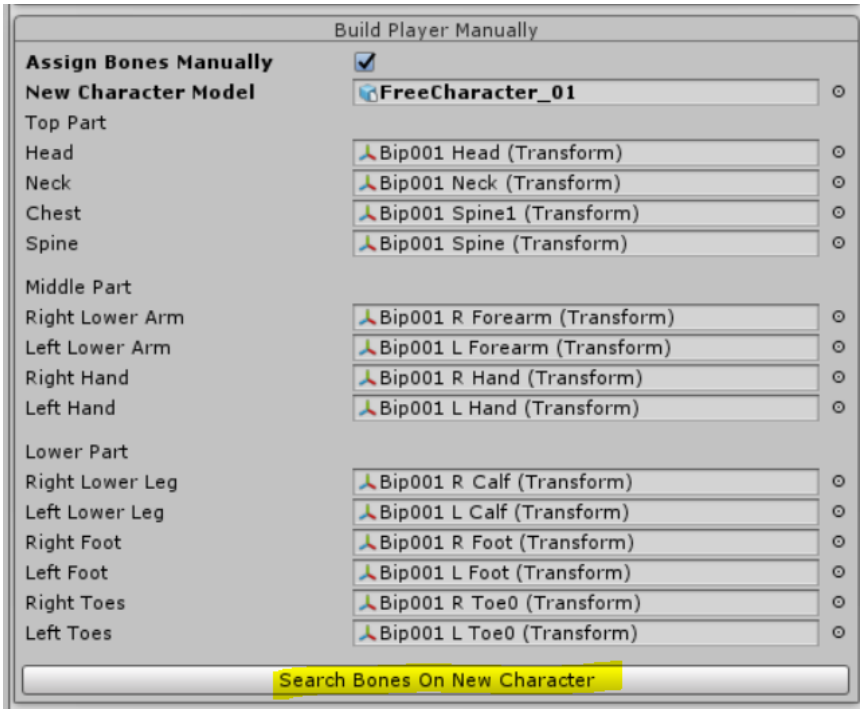
CREATE A NEW CHARACTER WITH THE MANUAL CREATOR

There is also a manual mode to create a new character. This mode is used when the automatic system hasn't assigned the bones as the character needs (due to special skeleton hierarchy). For this, follow these steps:

- Drag and drop a player's prefab in the scene.
- Drag and drop the new model you want to assign in the player.
- Open **Build Player** inspector, set to true the field Assign Bones Manually and assign the new model in the field New Character Model:



- Press the button **Search Bones On New Character** and the bones needed will be assigned in the inspector automatically:

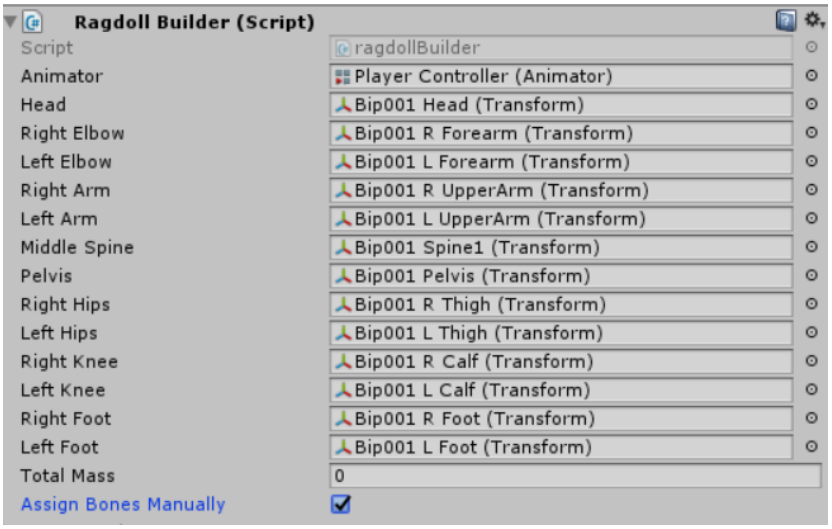


- Then, assign the bones (in case that weren't assigned correctly with the automatic mode) and press the button Build Character, so the old player's model is replaced by the new one and all the fields needed are assigned.

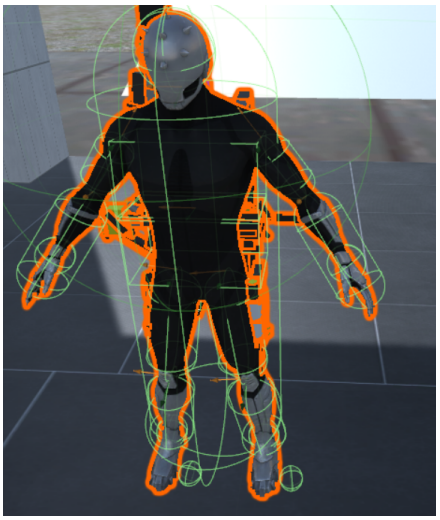
The same can be done with the ragdoll (by default the ragdoll is created with the above process), so bones can be assigned manually to add the ragdoll parts on them properly in case the system didn't create it correctly. For this, follow these steps:

- Go to the **Ragdoll Builder** inspector and set to true Assign Bones Manually.
- Make sure the animator of the player is assigned in the field Animator.

- Press the button Search Bones On New Character and the bones will be assigned automatically, like in the build player:



- Assign the bones that weren't assigned properly and press the button **Build Ragdoll**. The ragdoll will be added like this:

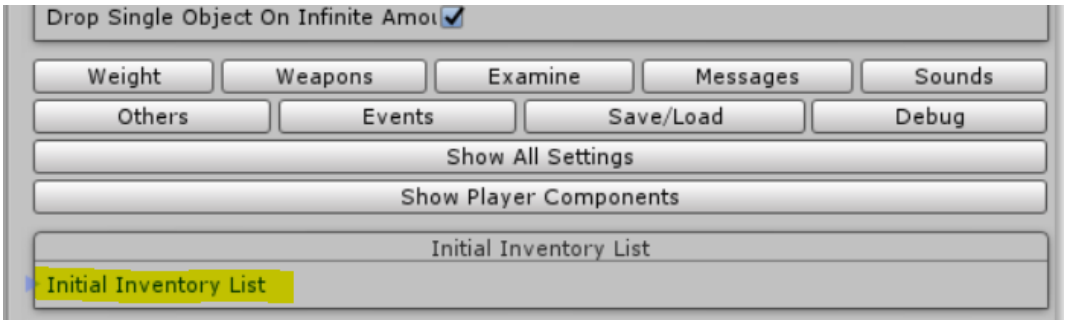


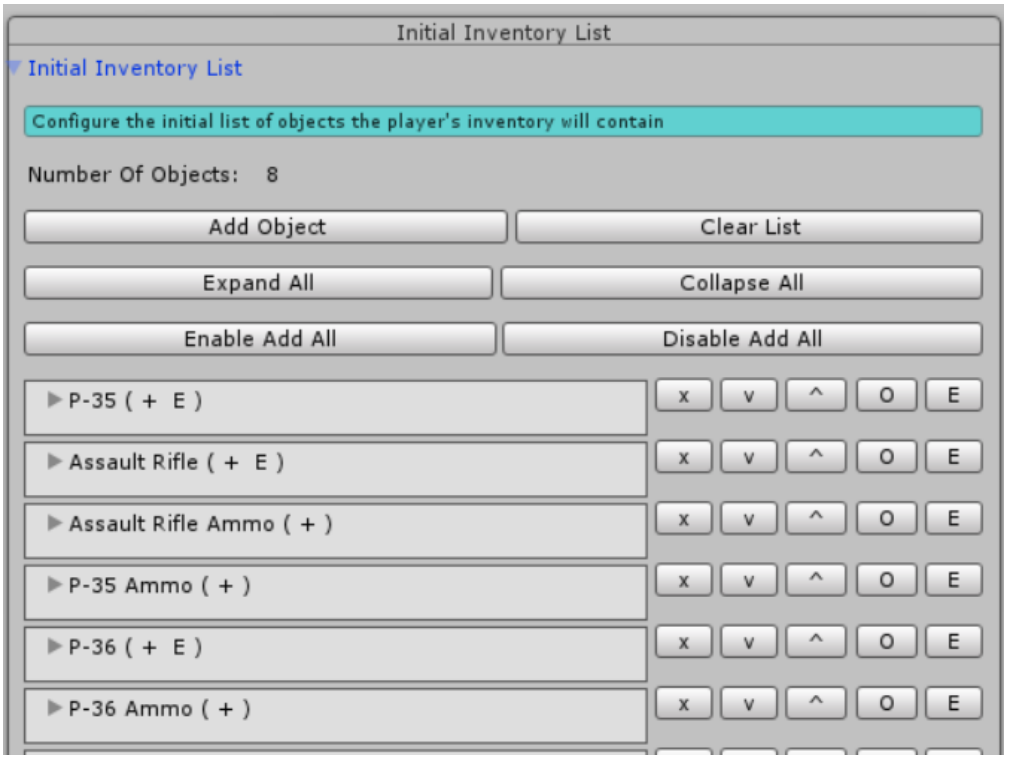
SET THE DEFAULT WEAPONS IN THE NEW MODEL

To manage the player's weapons, go to **Player Controller** gameObject, and to the inspector **Player Weapons Manager**.

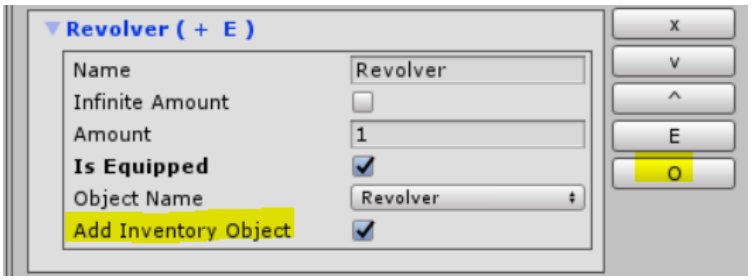
IF YOU USE THE INVENTORY TO STORE AND MANAGE WEAPONS, follow these steps:

- Open **Inventory Manager** inspector in Player Controller gameObject and open the Initial Inventory List:

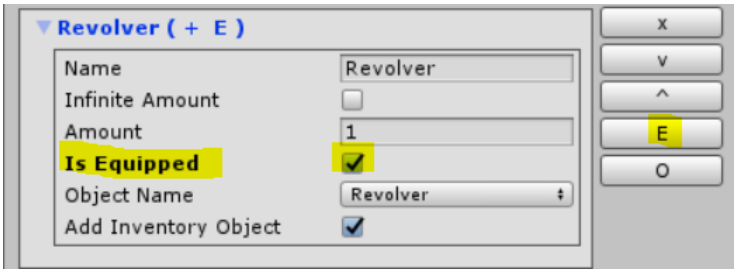




- In that list you can configure the initial inventory in the player at the start of the game. You can add, remove, set as equipped and set if the inventory object will be added at the start or not (to avoid removing an object from the list to not being included in the inventory of the player at the start). By default all the weapons available in the player are included.
- To include every weapon, press the button O to enable or disable if the weapon will be in the inventory at the start or no (the field **Add Inventory Object** will be as true or false):



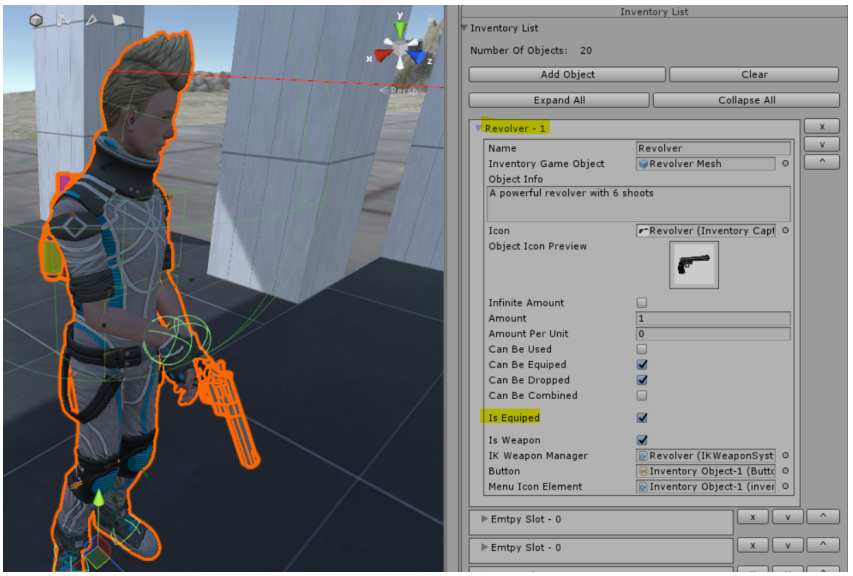
- To set that weapon as equipped at the start of the game (so the player can carry and fire that weapon), press the button E and the field **Is Equipped** will be set to true or false:



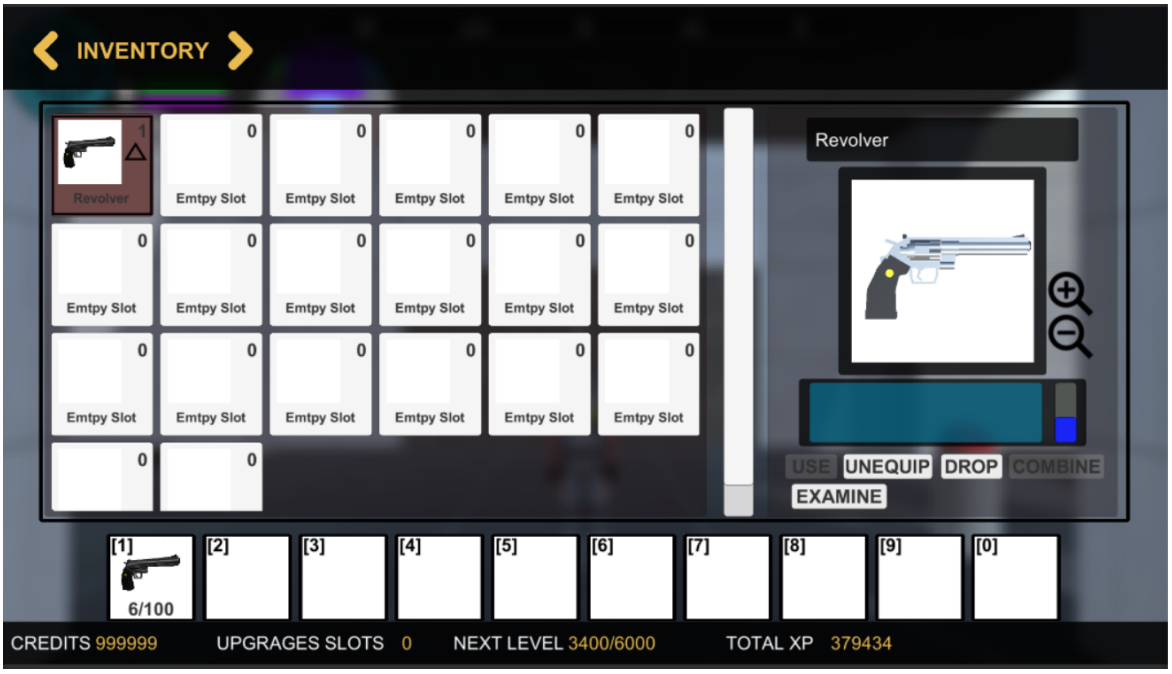
- Currently, the revolver is configured to be added to the inventory and equipped.

Do this setting with the rest of weapons, so you can add as many as you need or remove by setting Add Inventory Object as false, so the player starts with no weapons, but he can add weapons later in the game using the weapons pickups.

For this example, only the revolver has been added and equipped, so you can see that the inventory object is included in the Inventory List, below the Inventory Manager List, in the same inspector.



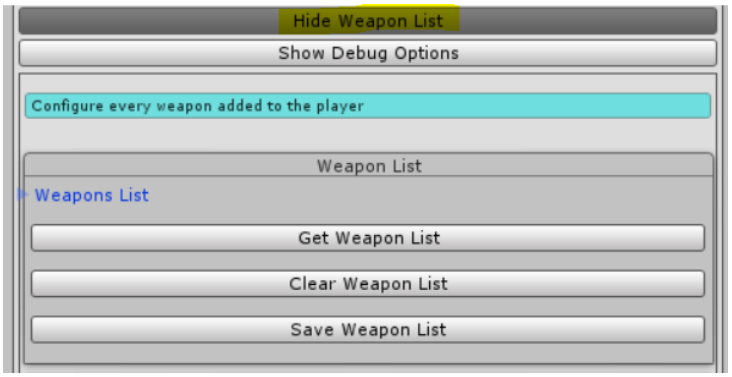
The rest of empty slots are configured according to inventory space configured in the inspector, in this case, 20 slots with a limit of 10 units per slot.



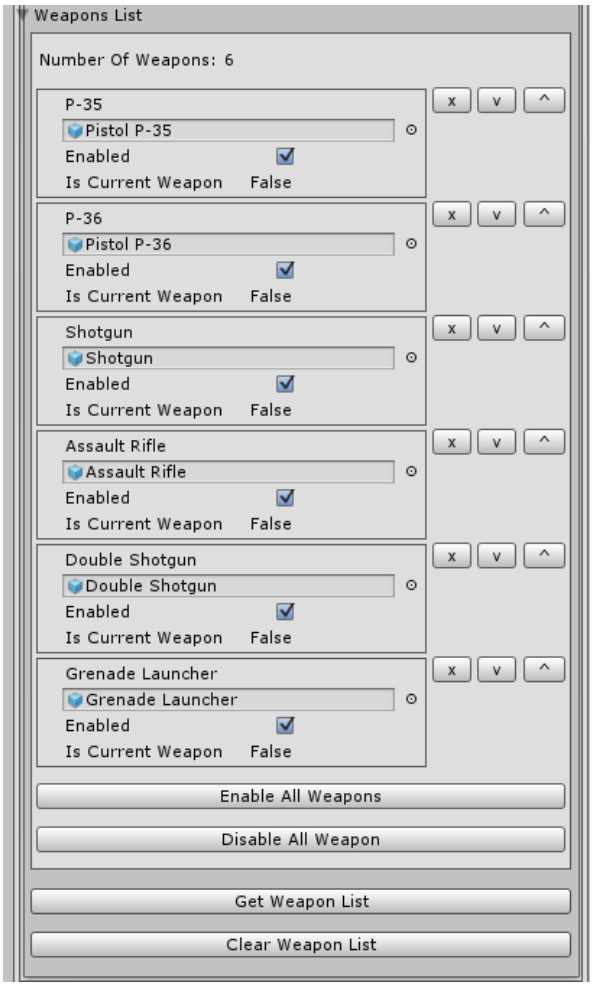
IF YOU DON'T USE THE INVENTORY TO STORE AND MANAGE WEAPONS, follow these steps:

IF YOU DON'T NEED TO USE THE WEAPONS, follow these steps:

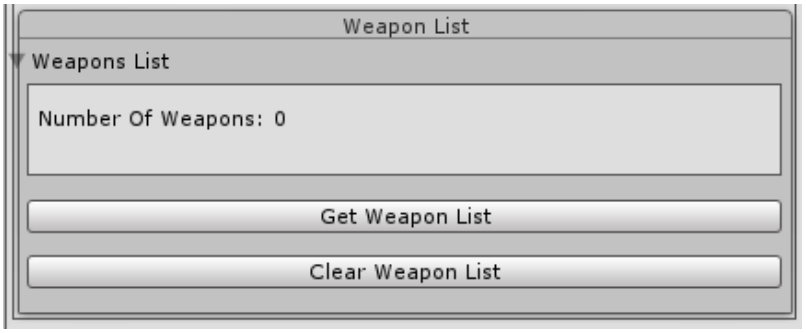
- In the **Player Weapons Manager** inspector, press the button **Show Weapon List**.



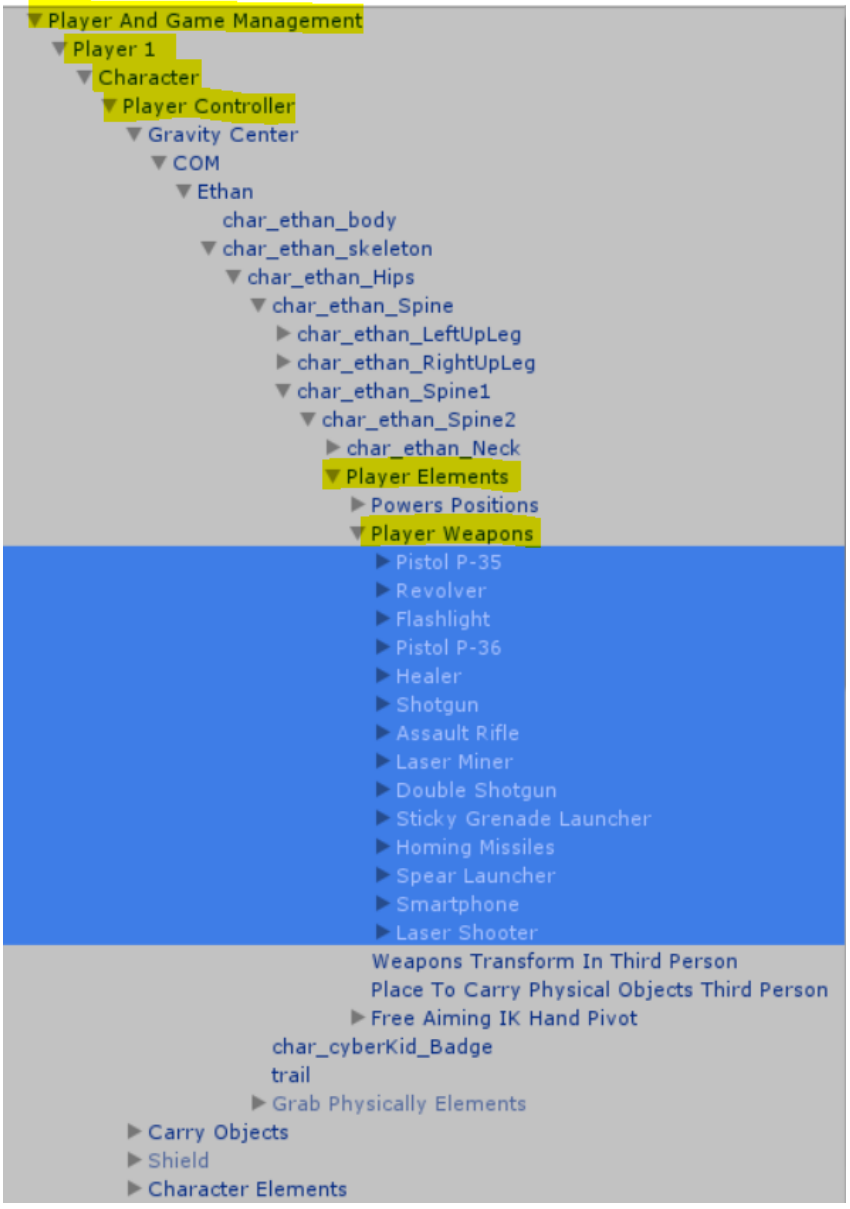
- Open the **Weapons List**:



- Press the button **Disable All Weapons** or **Clear Weapon List**.



- After this, you can also remove the weapons gameObjects from the player, located inside his body. The image below shows where they are located in the hierarchy. Remove all the objects inside the gameObject Player Weapons.

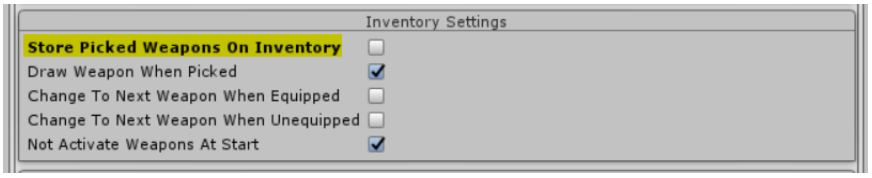


- Also, make sure to remove the weapons objects configured in the **Inventory Manager** inspector in the Player Controller gameObject, to avoid adding weapons that doesn't exist in the player body anymore.

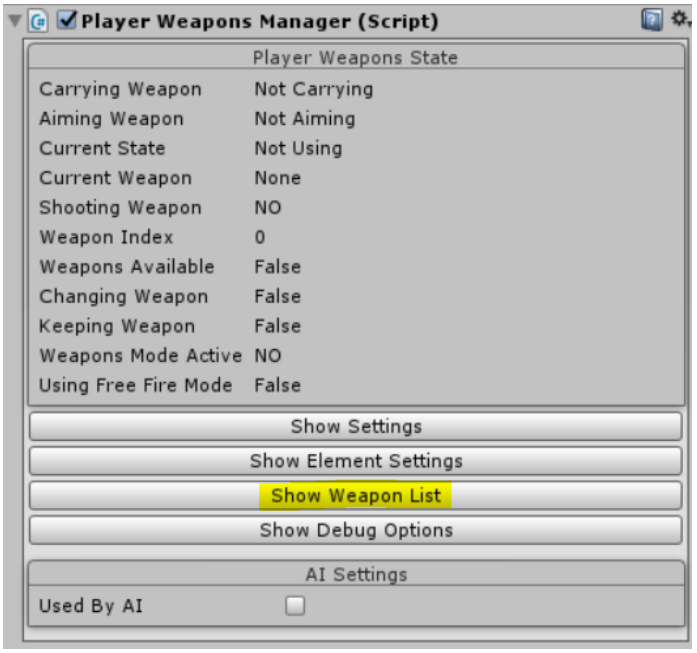
IF YOU NEED TO USE THE WEAPONS, follow these steps:

- Press the button Show Settings in the **Player Weapons Manager** inspector.

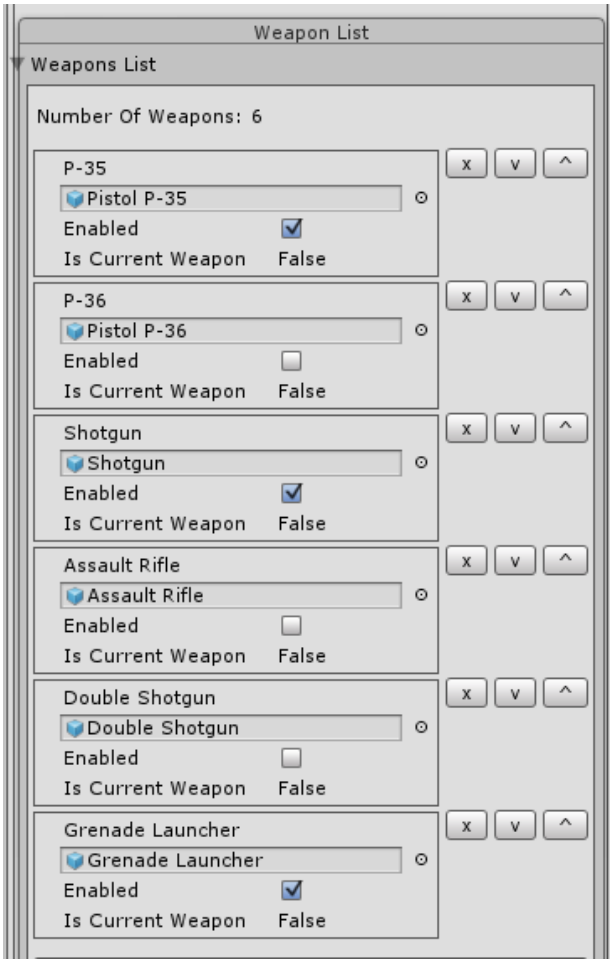
- In the Inventory Settings, set to false the field **Store Picked Weapons On Inventory**:



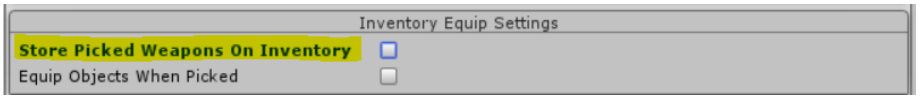
- Now, press the button **Show Weapon List**.



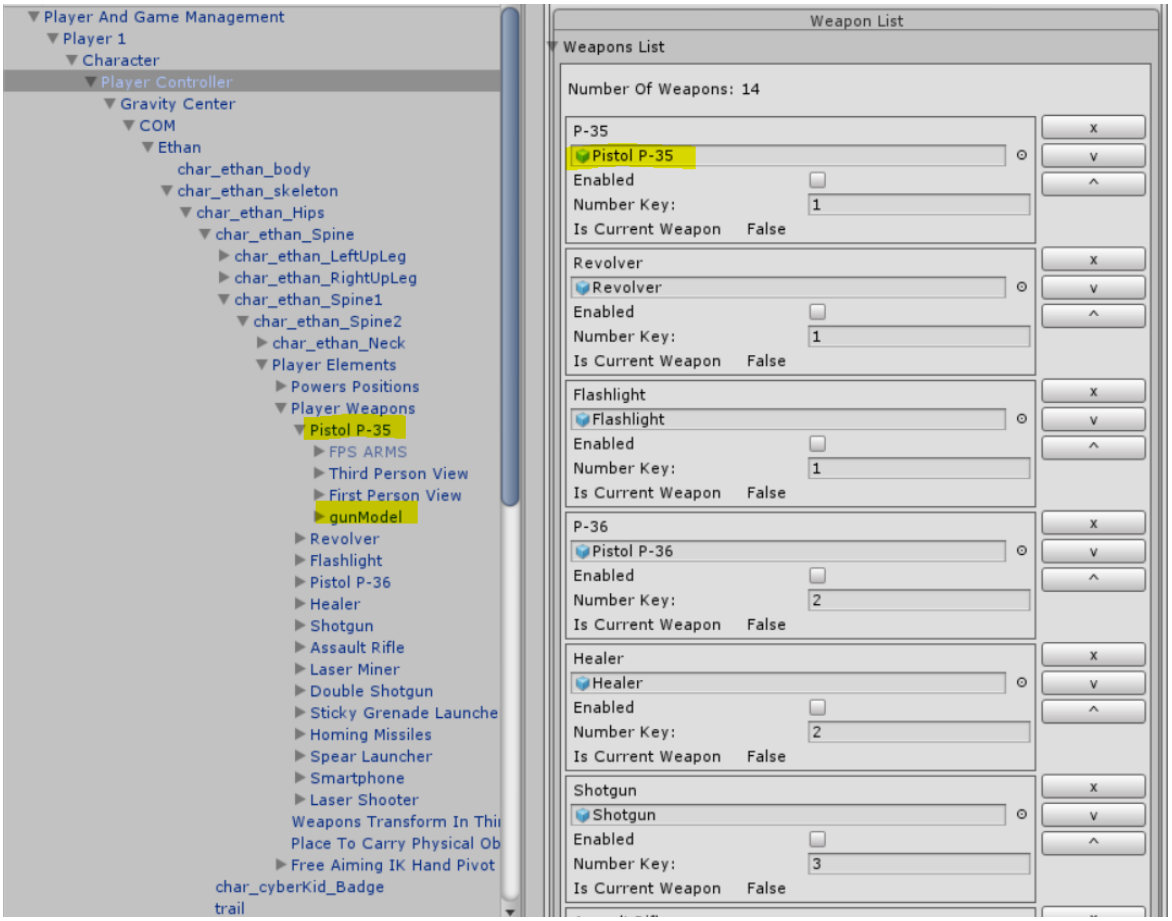
- You can now press **Enabled** (or disabled) option in every weapon to make it usable in game (else the weapon model in the player is disabled, but you can use it if you pick the weapon in the level, but you first need to complete these steps to avoid problems. You can remove every weapon in the list, as explained below, so even if the player finds that weapon to pick in the level, he will be unable to use it).



- Go to **Inventory Manager** inspector in the Player Controller gameObject and in the Inventory Equip Settings, set to false the field Store Picked Weapons On Inventory:

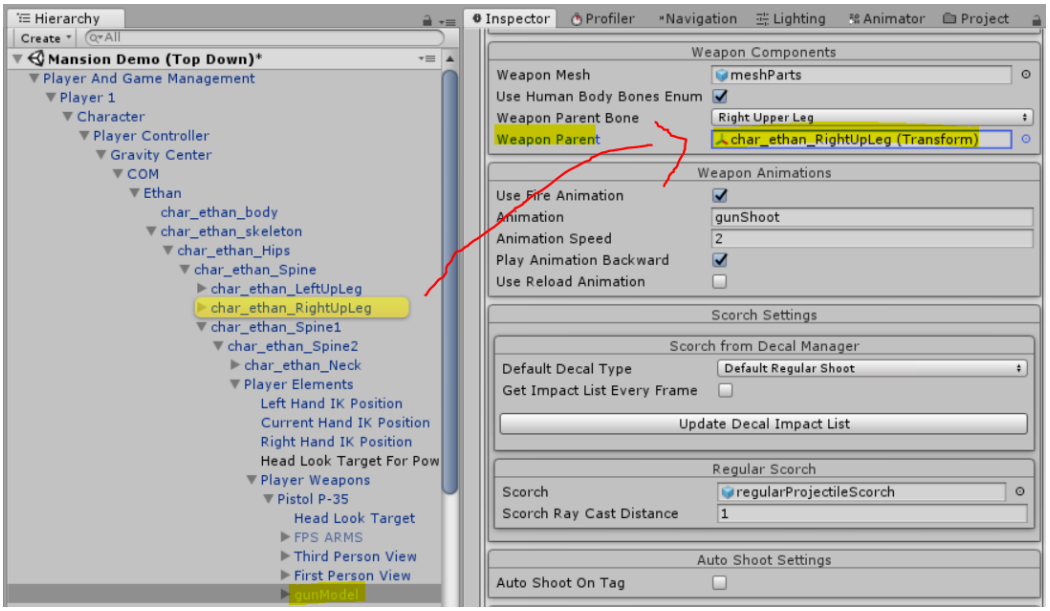


- Finally, return to **Player Weapons Manager** inspector. You can assign in every weapon where it will be attached inside the player's body (the back, the legs, the waist and by default is the back), so if the weapon is in the leg, it is placed inside of it. To this, go to any weapon (by selecting any weapon in the above list).



- Go to gunModel inside the weapon, and open the inspector **Player Weapon System**.

- Press the button **Show Weapon Settings**, go to section **Weapon Components** and set the bone that you want in the label **Weapon Parent** (for example in the demo, the pistols are parented in every leg and the rest of weapons in the back of the player).

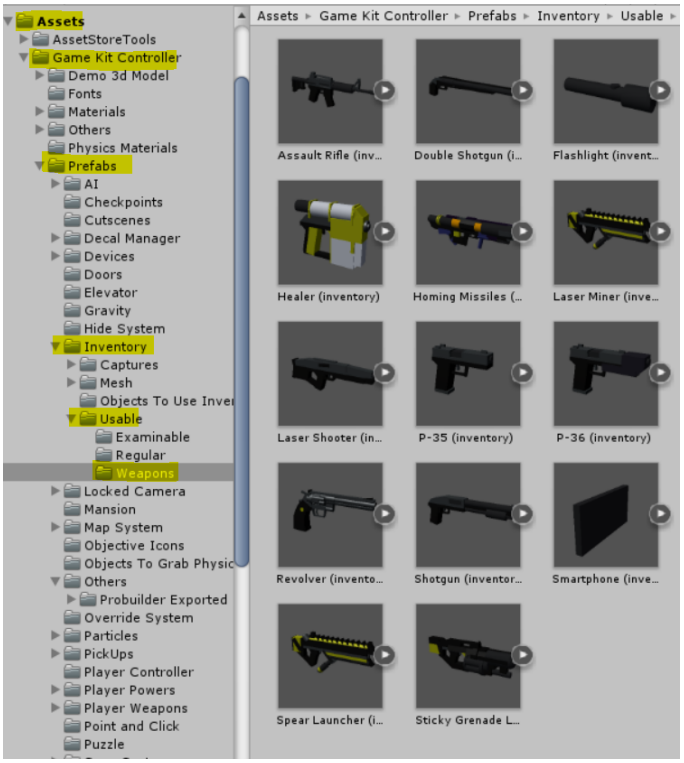


Like this, the weapons can be activated and used in the player without manage them through the inventory, so you can use them similar to games like Doom (2016) which hasn't an inventory.

Also, for there are two types of weapons pickups, one to just activate the weapons and other to store them as inventory.

The first type can be found in the prefabs folder **Prefabs/Player Weapons/Weapons**:

The second type can be found in the prefabs folder **Prefabs/Inventory/Usable/Weapons**:



So according to the type of weapons management that you use in your project, use the proper weapon pickup prefabs.

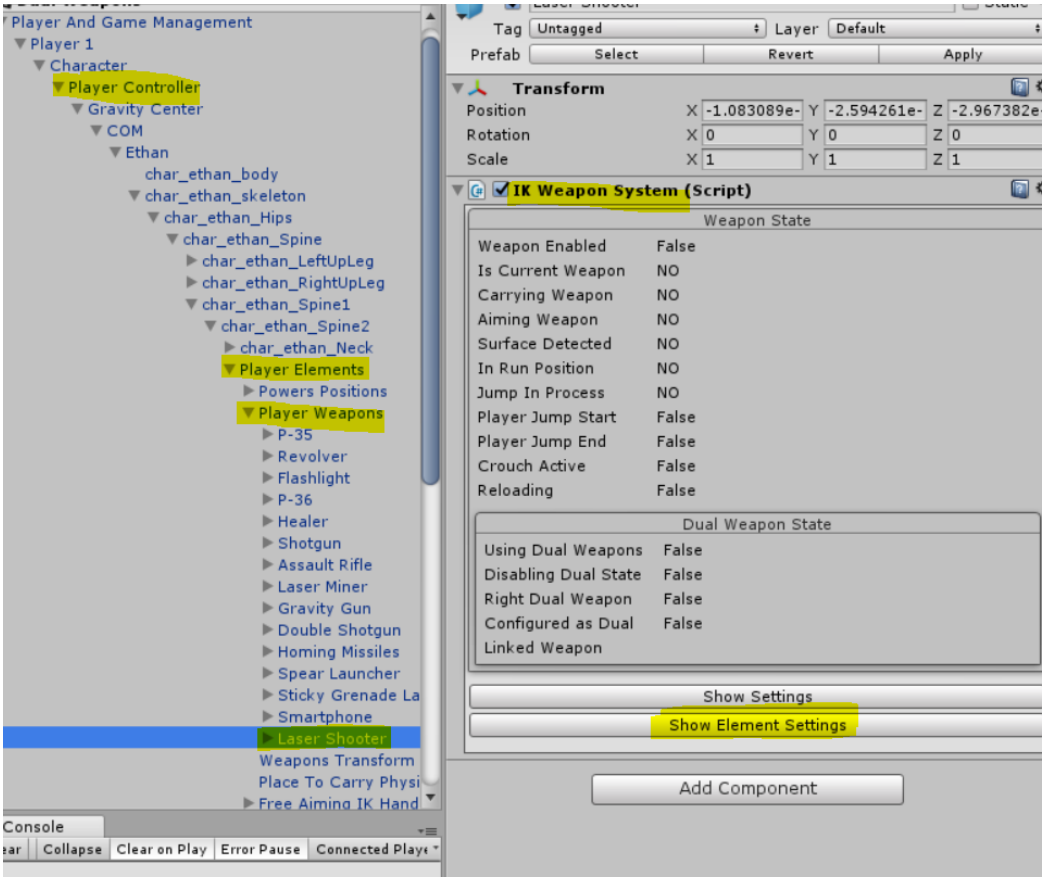
ADJUST WEAPONS TO PLAYER'S HAND FOR QUICK DRAW WEAPON ACTION

In third person view, there is an option on weapons which allows to make a quick draw of the weapon placing it directly on the dominant player's hand. There is also an option to carry a weapon on the player's dominant hand when he is not aiming, without IK.

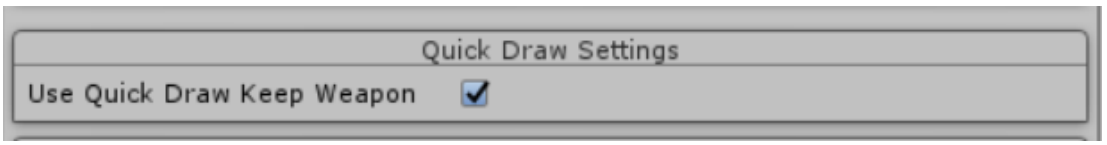
For both options, it is necessary to adjust the position of the weapon on the player's dominant hand, since every model has different bone orientation, this needs to be done manually if you use any of the above options (there are a **few weapons using these options by default, like shotgun, laser miner, double shotgun, homing missiles, spear launcher and laser shooter**).

This is a very quick an easy step, with the next steps:

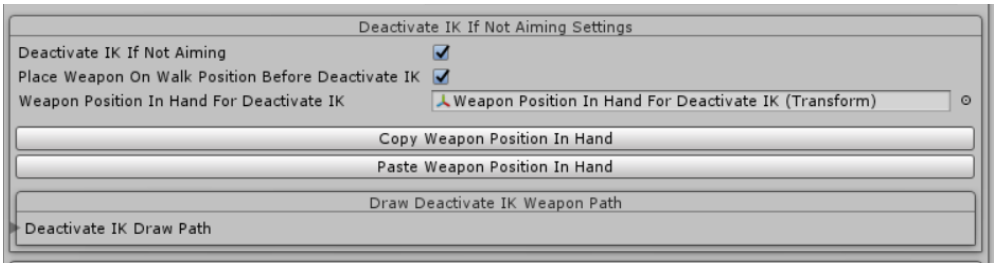
- Select the weapon that you want to adjust in the hierarchy, open the **IK Weapon System** inspector and press the button **Show Element Settings:**



- Press the button **Show Third Person Settings** and **Show Single Weapon Settings**.
- The option to use quick draw weapon is in **Quick Draw Settings**:



- And the option to carry a weapon on player's dominant hand while not aiming is in **Deactivate IK If Not Aiming Settings:**



- This is also where the hand position is adjusted to place the weapon on it while not aiming it
- To adjust the position, press play, draw the weapon which you are adjusting, aim the weapon and then stop aiming. You may see that the weapon is placed with an incorrect position and rotation when the player draws the weapon. Don't worry, this is why these steps are being done:

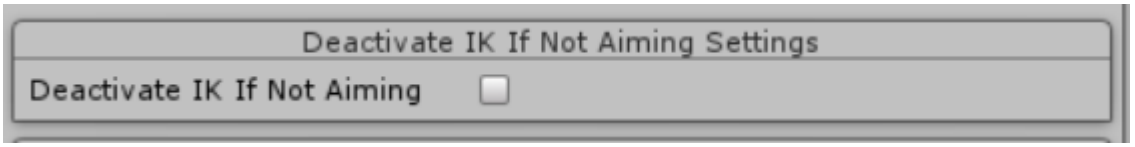
- Without keeping the weapon, press the button **Copy Weapon Position In Hand**
- Stop the game and press the button **Paste Weapon Position In Hand**

And that is it, the weapon now will be placed properly on player's hand when the quick draw action is enabled on that weapon, looking like this when the player draws that weapon with the position adjusted:

The quick draw can be also used without deactivating the IK system on player's arms, like for example, the double shotgun, which by default is configured like that:

This weapon has the option to make a quick draw enabled and deactivate IK option disabled, so the weapons is first placed in player's dominant hand and then moved to a walk position, like in the capture above.

For this, set to false the field **Deactivate IK If Not Aiming**, located in the same part that is has been used for this explanation:

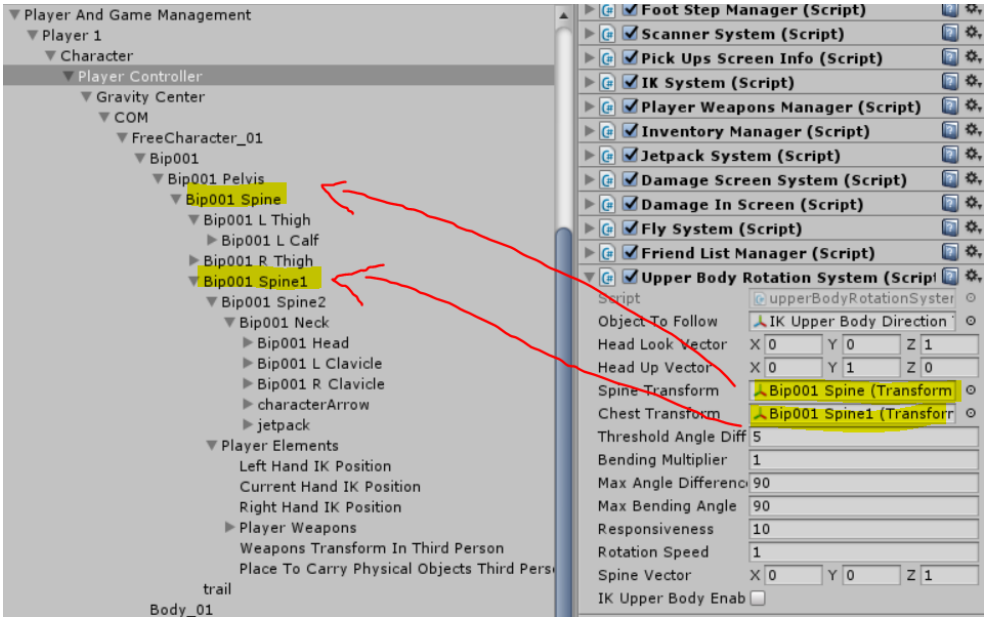


ADJUSTING PLAYER'S UPPER BODY FOR WEAPONS

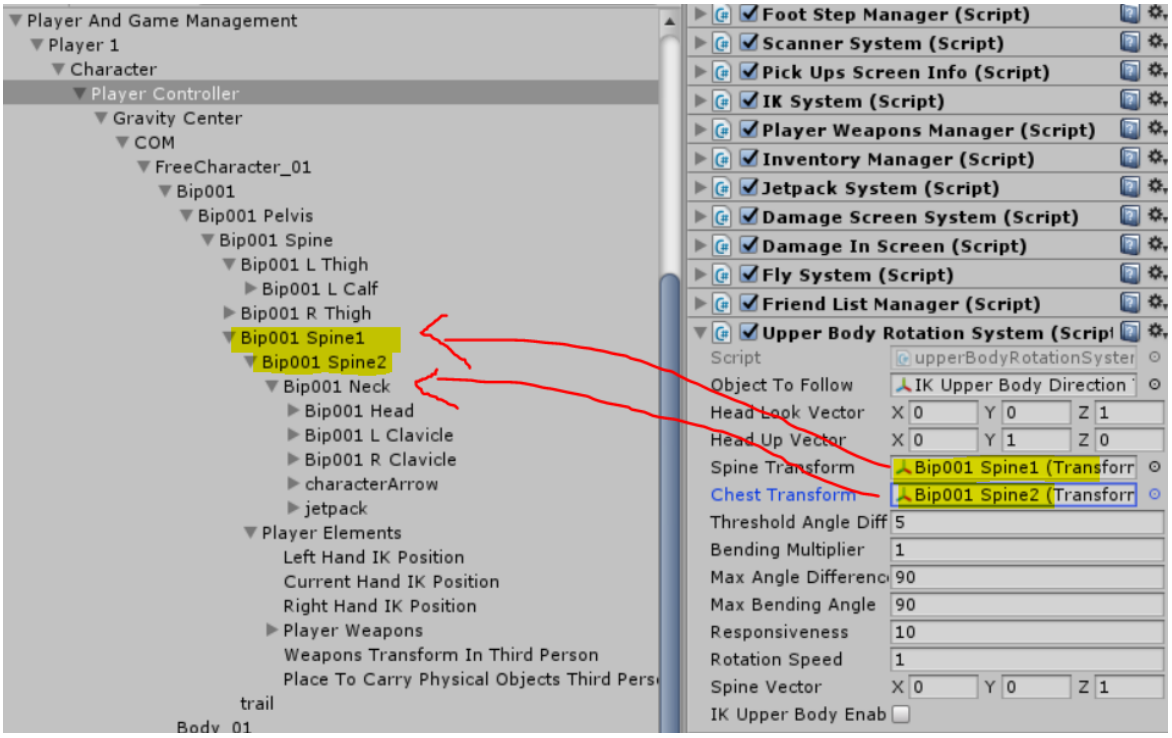
The player's upper body rotation can be configured for aim mode in third person. This is made in the inspector **Upper Body Rotation System** in the **Player Controller** gameObject, allowing to configure how fast the rotations are done and the clamp values used for the spine rotation. The object to follow is the actual transform followed for the player's torso, which is inside the main camera, so you can move it up or down to adjust the orientation when the player aims.

In the case you configure a new model, and something like this happens:

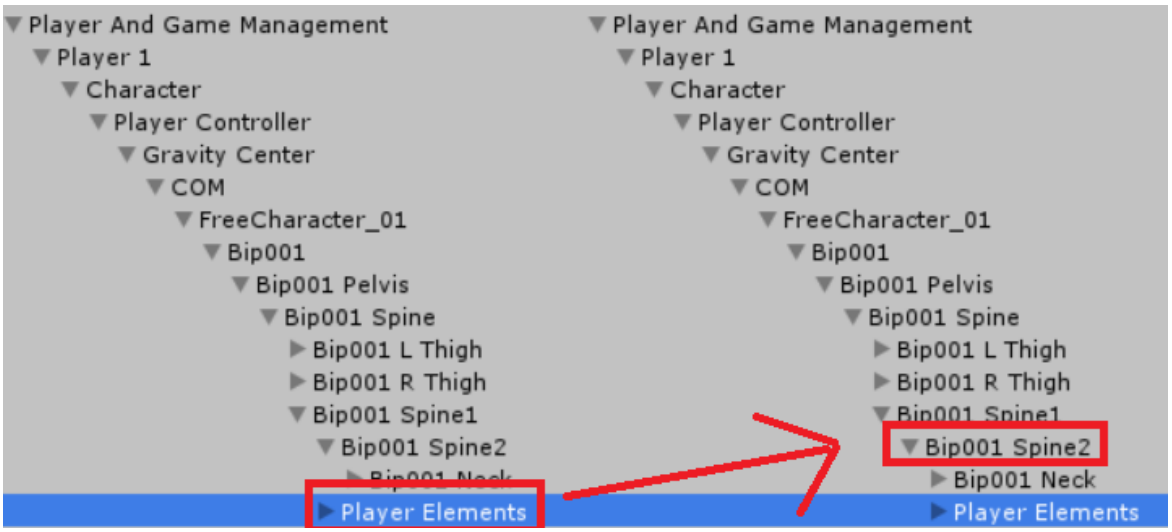
The cause is that the spine bones are not correctly configured in the **Upper Body Rotation System**, in the gameObject **Player Controller**, due to the skeleton hierarchy is slightly different in this case. In the example above, the system has taken the hips as first bone, this is incorrect.



But you only need to set the correct bones. To do this go to the inspector and make sure to place the bones which belong to the spine and the chest, like this:



But if this happens, you need to make another step, which is to move the **Player Elements** gameObject from its current hierarchy position to inside the Chest transform configured in the inspector, in this case, the Spine2.

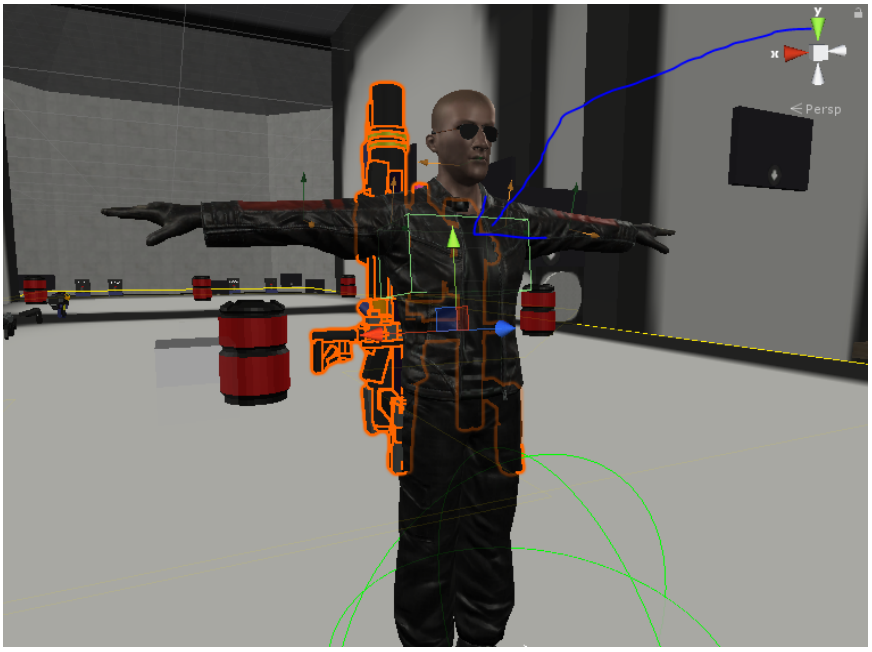


Like this, the player weapon should move correctly, rotating the player's upper body and without any issue.

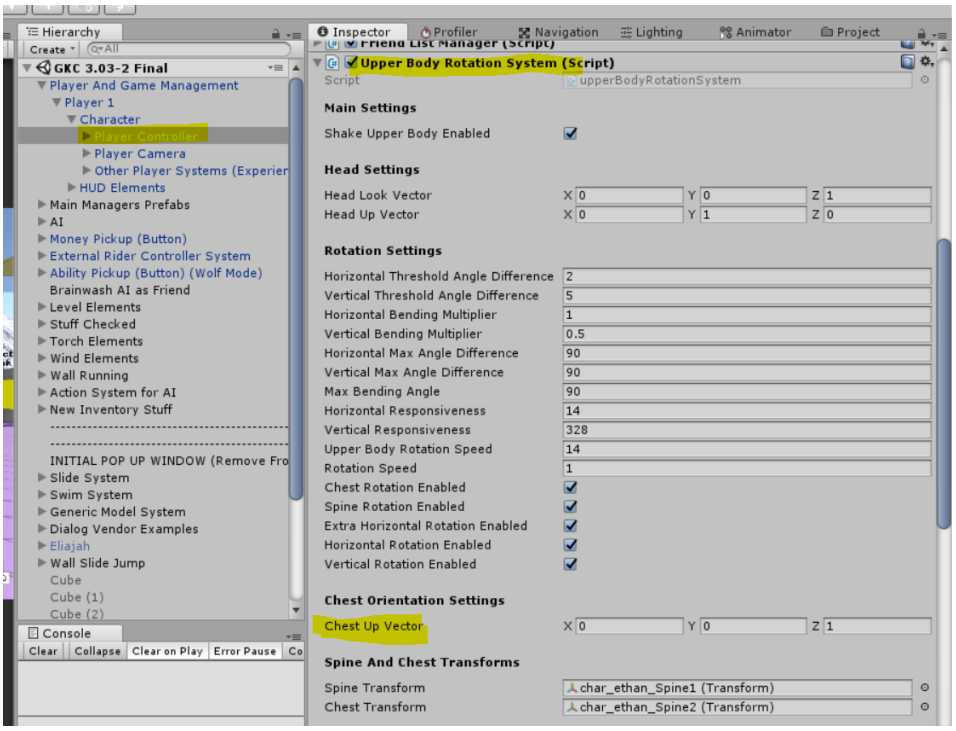
FIXING PLAYER SPINE ROTATION

If the player spine doesn't rotate correctly when you move the mouse up and down in aim mode in third person, the cause is due to the spine orientation in the model.

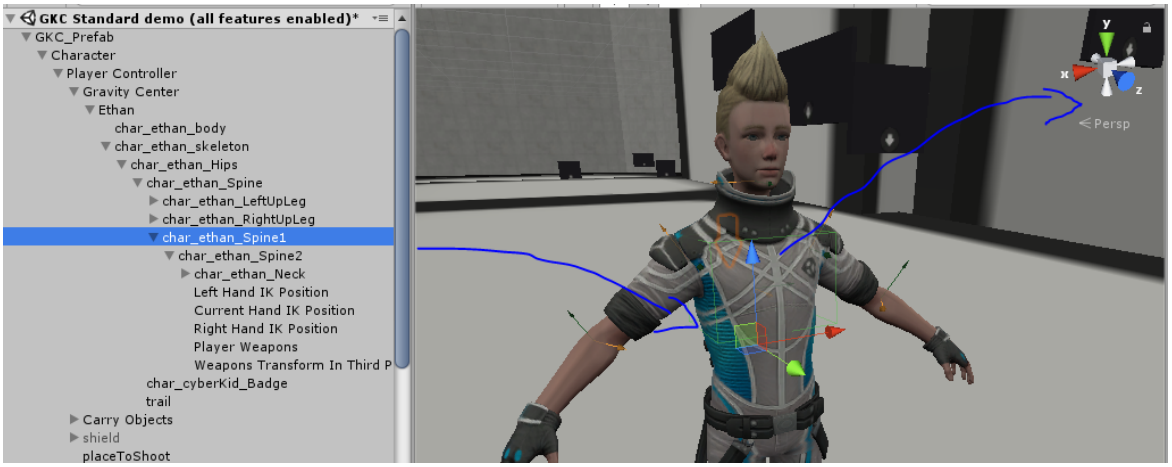
For example, in this model, the local Y axis of this model spine, is the Y axis, which can be different in every model.



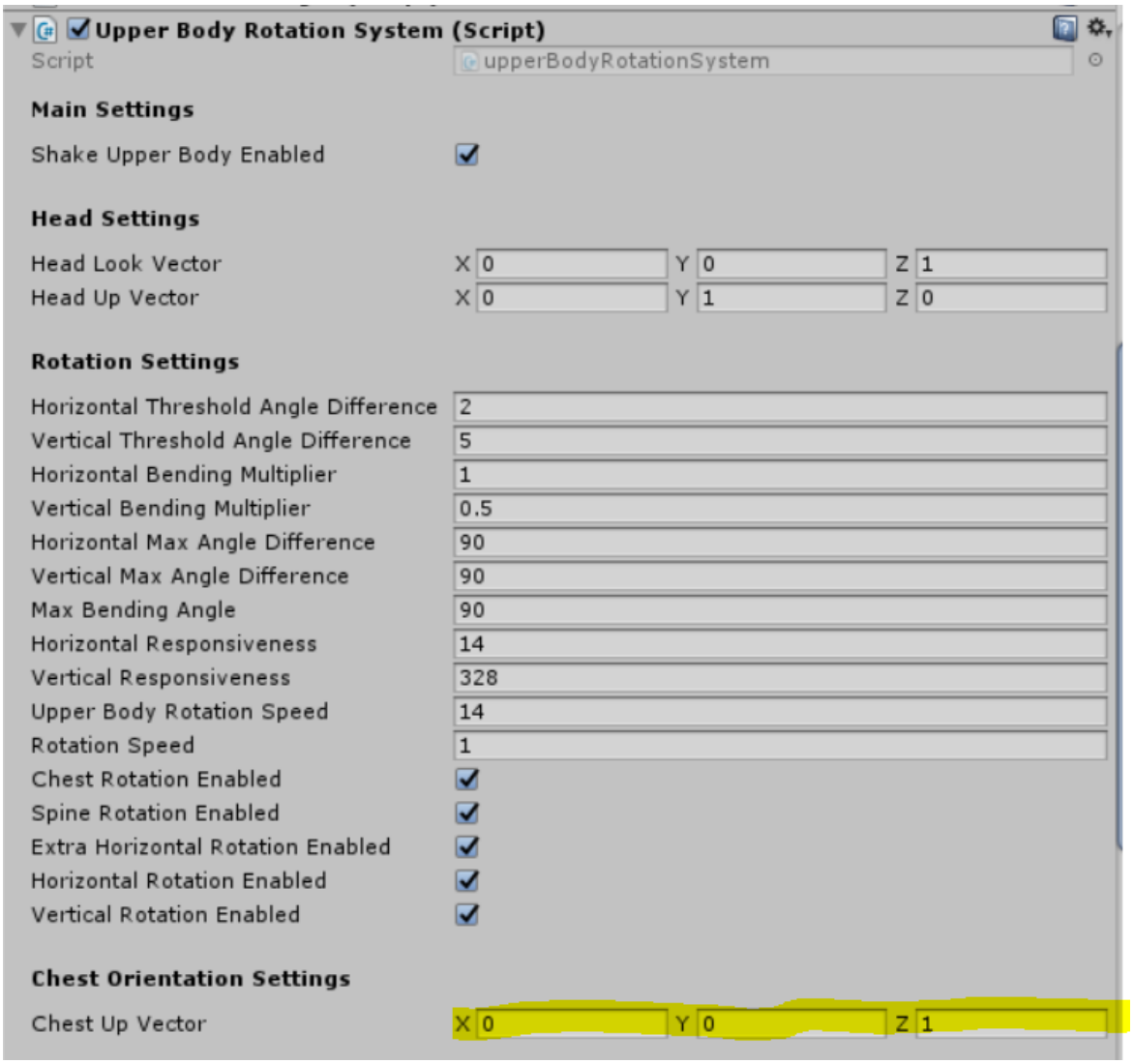
To fix this, go to the **Upper Body Rotation System** inspector in the gameObject **Player Controller** and configure the correct values in the field **Spine Up Vector**. It needs to be the same up transform orientation of the transform assigned in the field Spine Transform (mixamorig: Spine in this example). In this case, the value is (0,1,0) instead of (0,0,1) which is the default value.



In the default model of GKC, its spine orientation is Z for the local Y axis.

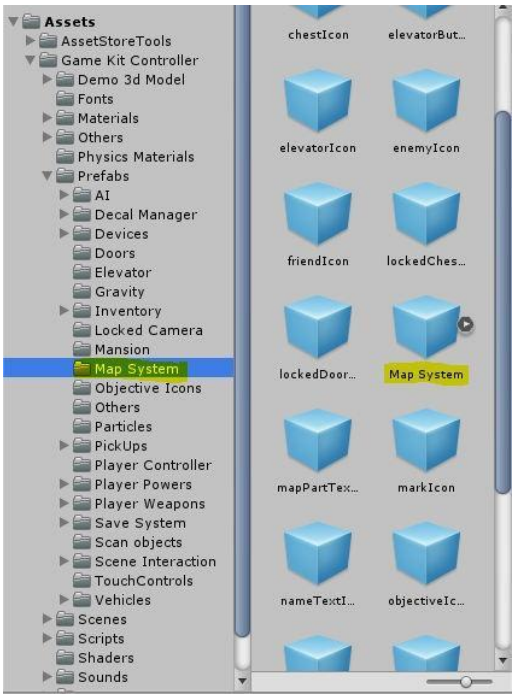


And the value used for this is (0,0,1):

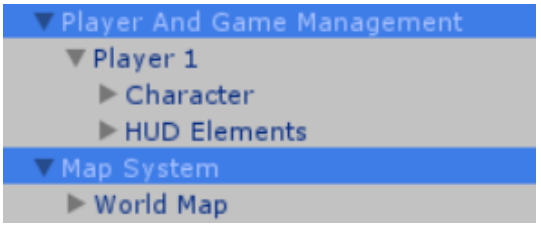


SET THE MAP SYSTEM

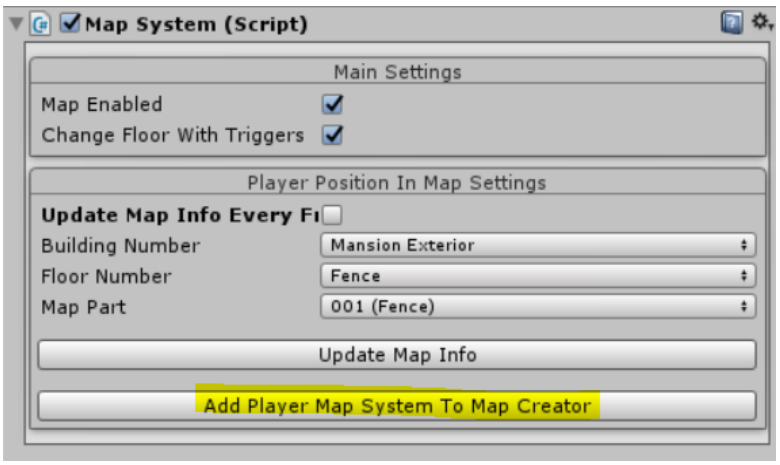
Once that a new player has been created, if you want to use the map system, you need to drag and drop the **Map System prefab** (with the same name) in the folder **Assets/Game Kit Controller/Prefabs/Map System**.



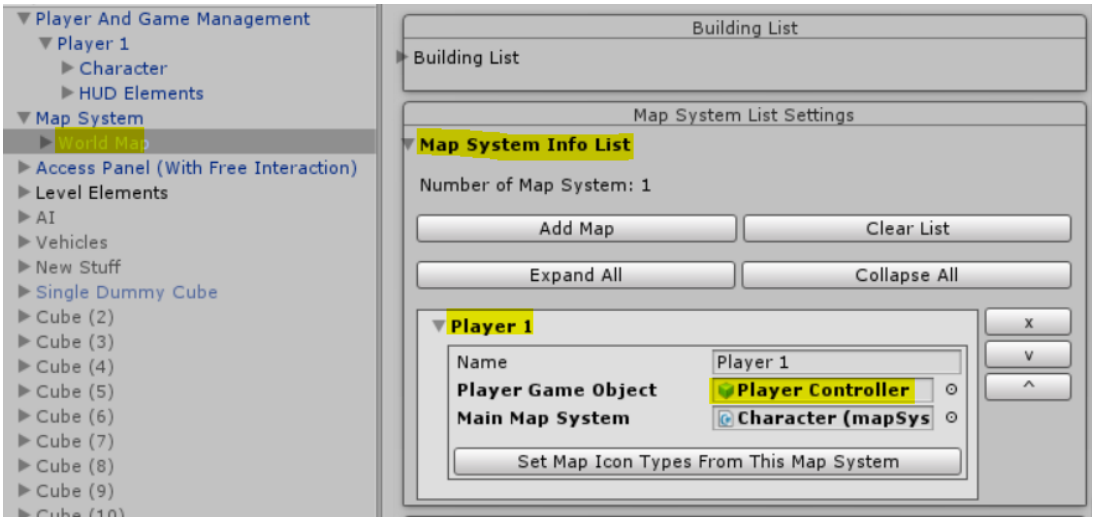
Once the hierarchy contains these two elements:



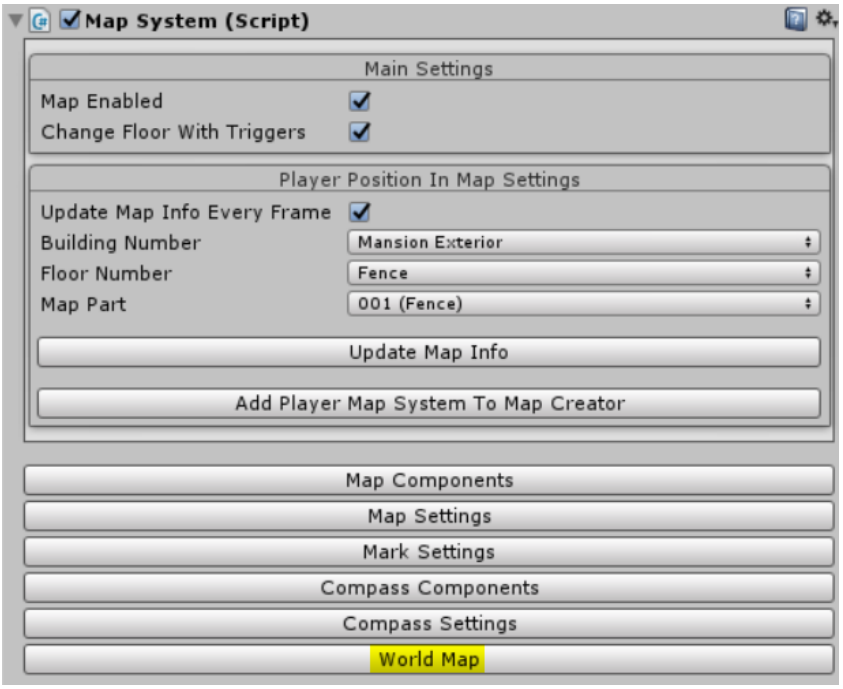
Then, go to **Character** gameObject, **Map System** inspector and press the button **Add Player to the Map System Creator**, to add this player map system to the main map manager, which is Map Creator.



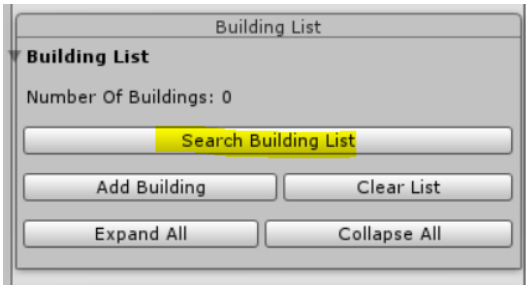
In the **Map Creator** inspector, you can see how the current player has been added to the main map manager:

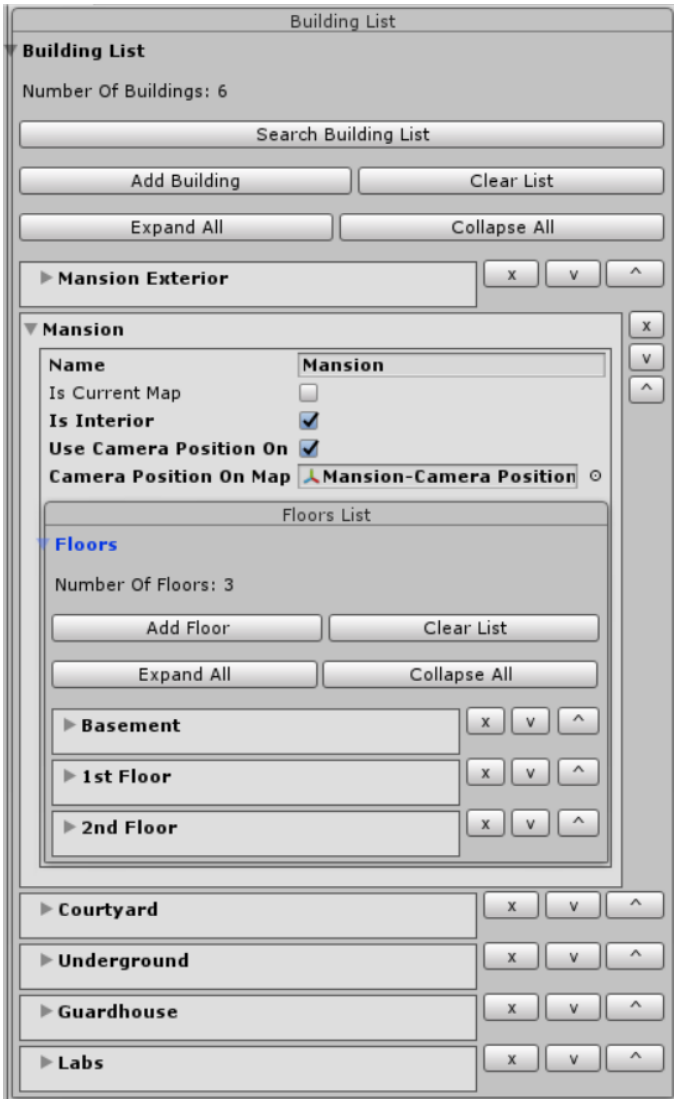


Now go back to **Character** gameObject, **Map System** inspector and press the button **World Map**.



Go to **Building List**, and press the button **Search Building List**. The buildings and floors configured in the Map Creator inspector will be configured automatically.





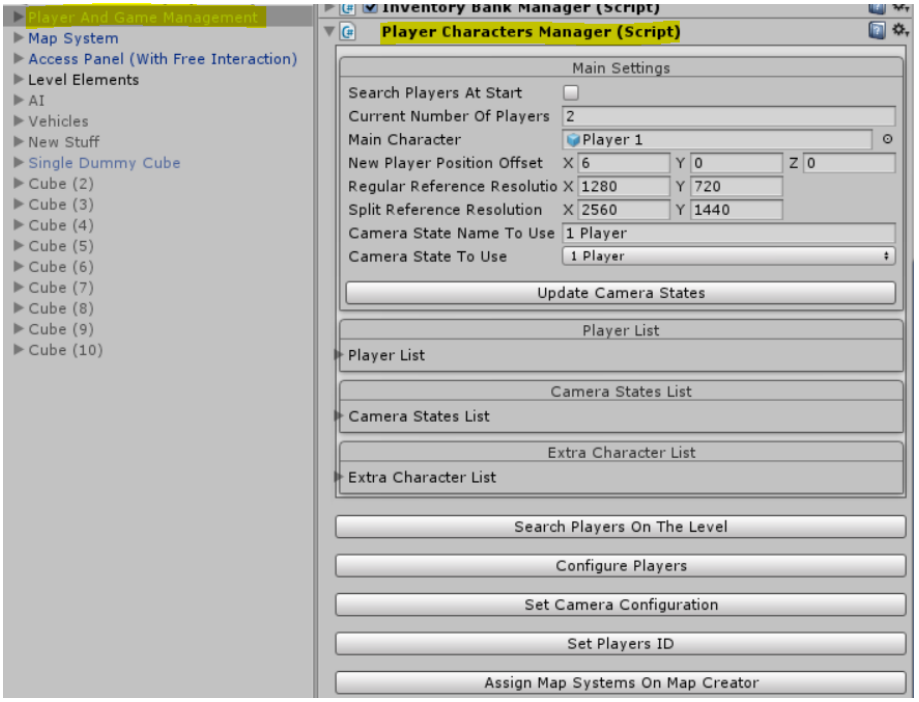
If you don’t need to use the map system, don’t configure the above steps and it will be disabled.

CONFIGURE LOCAL MULTIPLAYER

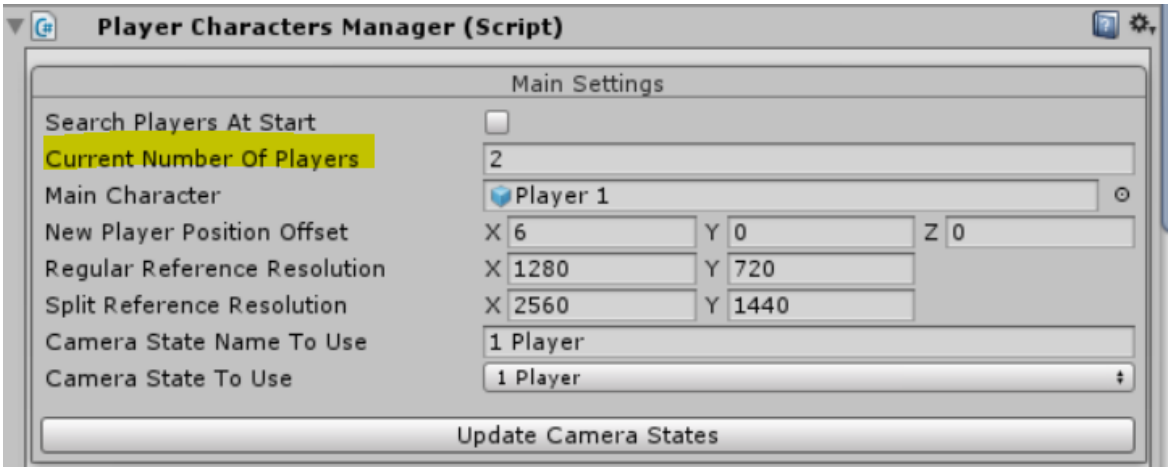
In update 2.9, local multiplayer has been added, with split screen system as the current way to configure the cameras. This local multiplayer is on beta state on 2.9 and will be complete in 3.0. But all the systems work properly without issues or errors, just need some extra additions and checks, like allowing to configure the players ingame.

For now, the players can only be configured in editor time. For this, follow these steps:

Go to the gameObject called **Player and Menu Management** and to the inspector **Player Characters Manager.**

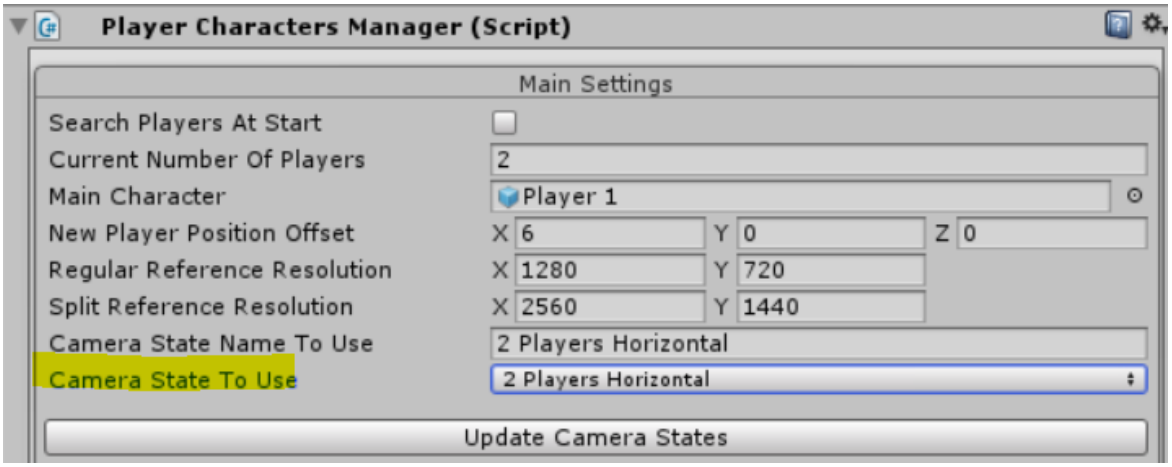


There, in the field **Current Number Of Players**, configure the number of players to use (for now use only from 1 to 4).

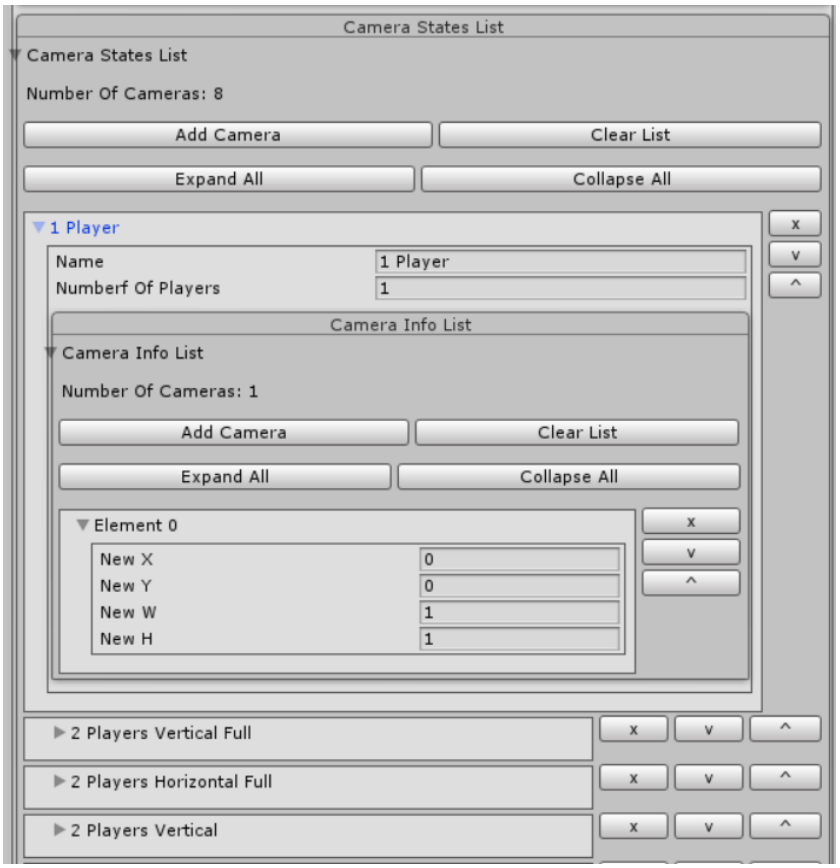


In this example, lets just use 2. After that, in **Camera State To Use**, select the screen camera configuration for the split screen. For 2 players, there are different options like full screen with

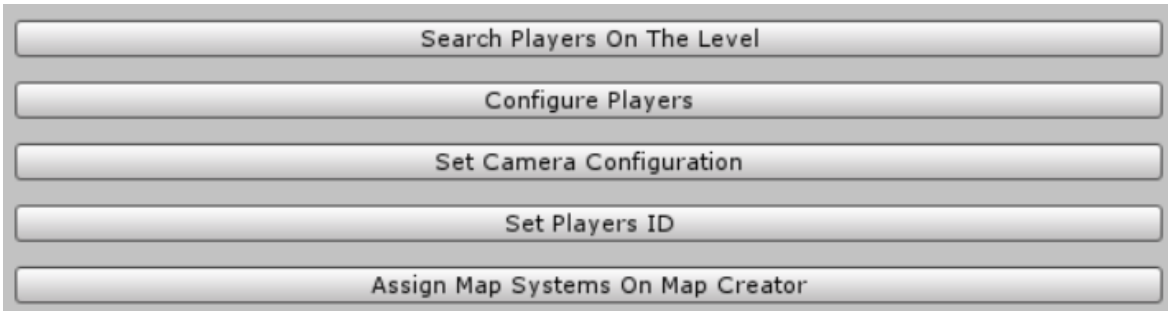
left and right side with every player, vertical, with a player on top and the other on the bottom, etc…



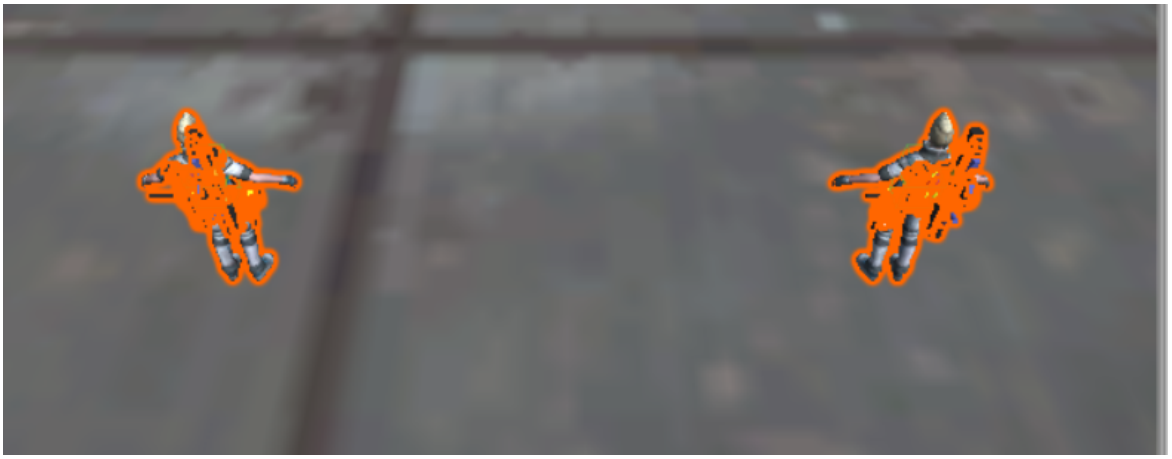
You can configure new camera distributions on the **Camera States List**, which is configure the amount of main cameras (one per player) and the X, Y, W and H values of every camera in the screen, setting these values for new players added:



After that, press these buttons in the next order: **Configure Players**, **Set Camera configuration**, **Set Players ID** and **Assign Map System On Map Creator**. This will add the new players configured in the amount above, configure the split screen cameras, set the players ID to manage input for everyone of them and add them to the main map system manager.



Here you can see the new player added along the previous one:

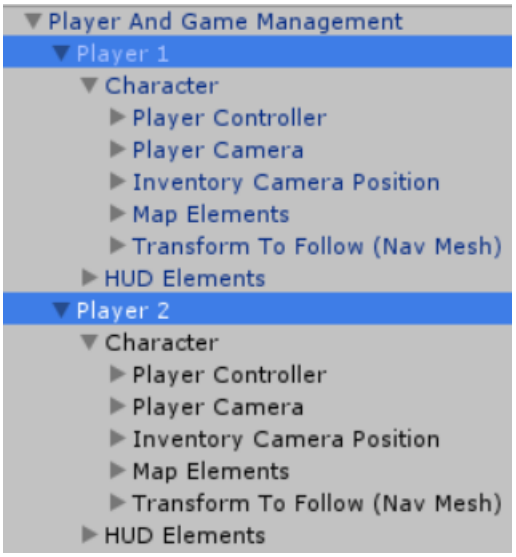


Here the camera configuration:

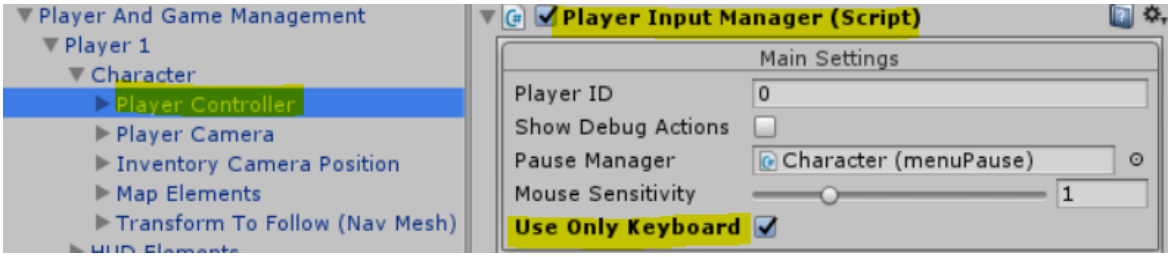
You can select another type of camera setting, just by selecting a new one in the field **Camera State to Use** and pressing the button **Set Camera Configuration** again. Here the same players in vertical:

You can also remove players by selecting a lower amount in the field **Current Number of Players** and pressing the above buttons in the same order (and make sure to select the Camera State To Use according to the number of players in scene). So you can return to have just one player or add more.

In the hierarchy, you can see how the new players are added inside the main parent:



By default, all players are controlled with gamepad, so you will need the same amount of gamepads as players to play with them. There is an option to allow one player to use the keyboard. For this, go to any of the players to the gameObject called **Player Controller**, and in the inspector **Player Input Manager** set to true the field **Use Only Keyboard**.



After that, return to **Player Character Manager** and press **Set Players ID** button again.

Now if you go back to **Player Controller** gameObject and select the inspector **Player Controller**, you can see that the Player ID is 0:

When usually would 1. This is to make the system to know that this player won't use a gamepad but the keyboard. Follow the same steps to configure the use of gamepad again (set as false the field Use Only Keyboard and press the button Set Players ID again).

CONFIGURE INPUT REBIND MENU PANELS AND ACTIONS

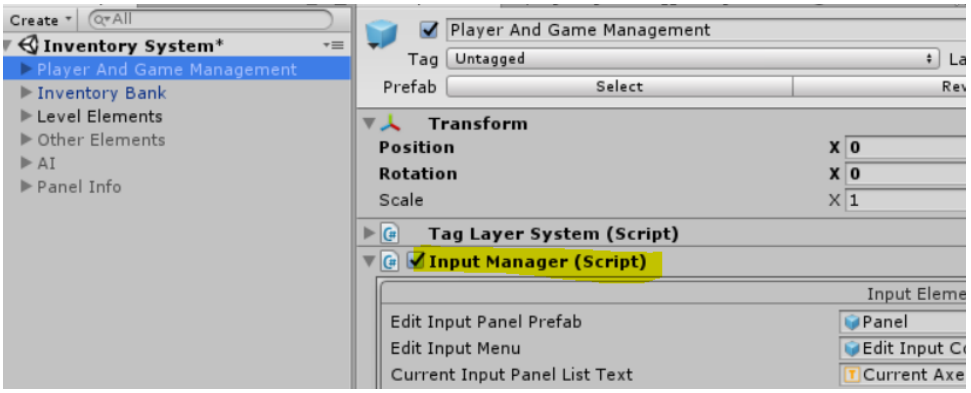
The input can be edited ingame at any moment, allowing to change for example the key used for jump.

For this, a menu is configured for this system, and inside this menu, multiple panels for each group of actions and different actions buttons are created for each action in those groups.

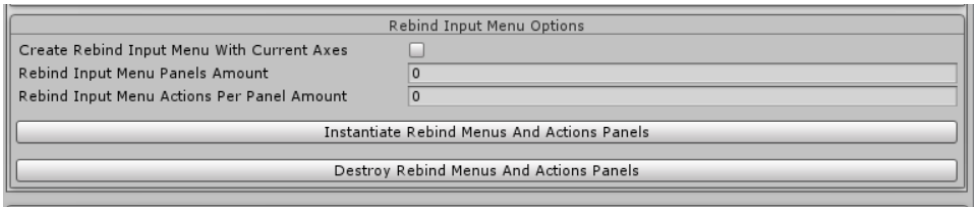
This by default is made at the start, instantiating these elements in that moment, but these elements can be configured and instantiated in the editor, before starting the game, to avoid those instantiations at the start.

To configure this, follow these steps:

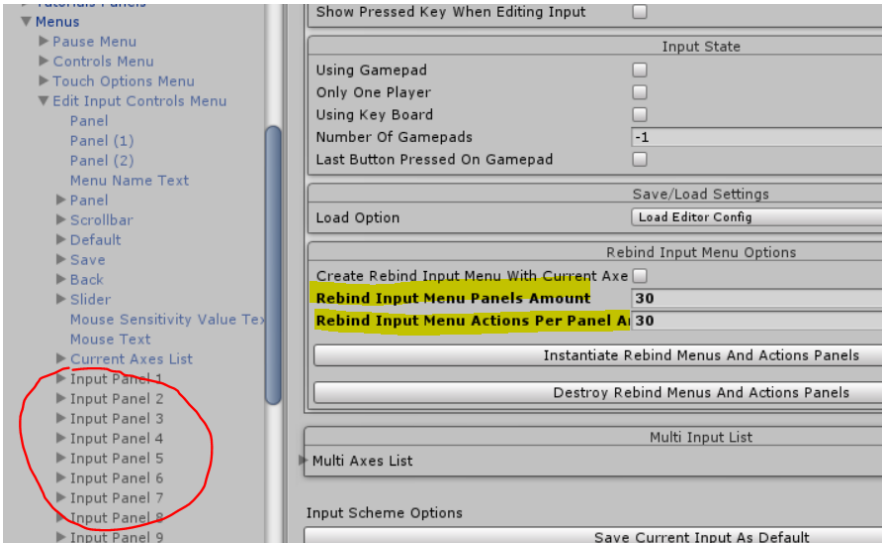
- Go to Input Manager inspector:



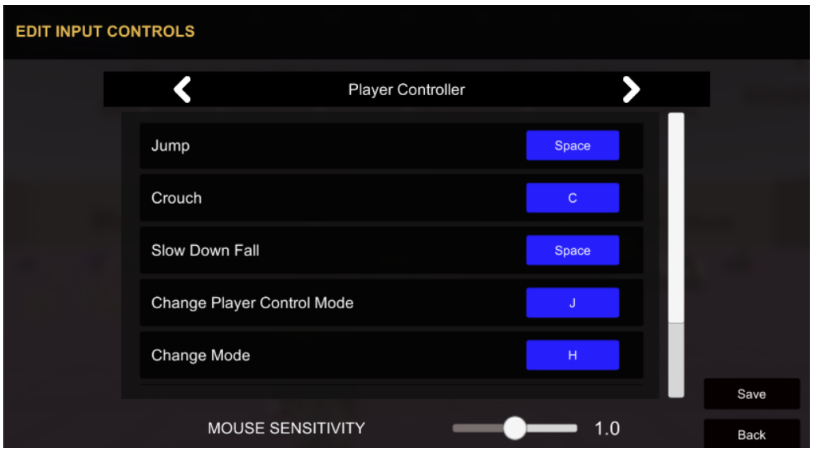
- Go to the section Rebing Input Menu Options:



- And set the amount of panels and actions buttons inside each panel, to use it as a pooling list of elements to be used for the rebind menu and press the button **Instantiate Rebind Menus And Actions Panels**:



You can see how these panels and elements are created inside the Edit Input Controls menu.



You can also, destroy these panels to create other amount or instantiate them at the start with the button **Destroy Rebind Menus and Actions Panels**.

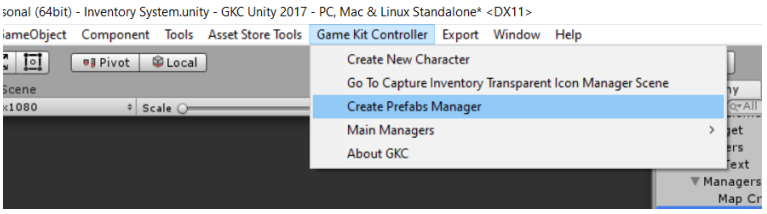
The options **Create Rebind Input Menu With Current Axes** will use the current amount of input categories and actions in each category to create the exact amount of panels for the rebind menu, instead of having a pooling list with panels and buttons that maybe will be left as not used due to the actions amount are lower that the amount of these panels.

PREFABS MANAGER

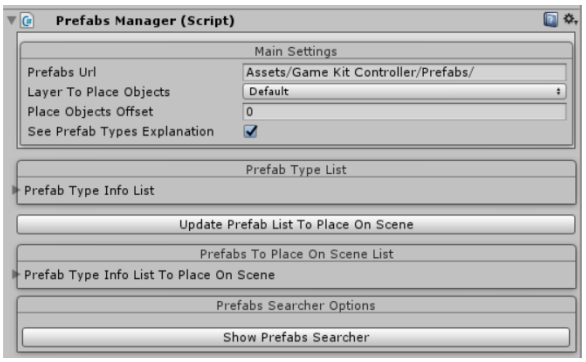
There is a system called **Prefabs Manager**, which allows to manage, instantiate and add prefabs to the scene easily, avoiding the need to search between folders.

For this follow these steps:

Go to the upper part of unity, in the options of the top, **Game Kit Controller->Create Prefabs Manager**.



If there was a Prefabs Manager previously in the scene, it will select in the editor, else a new one will be placed on the scene and selected in the editor.

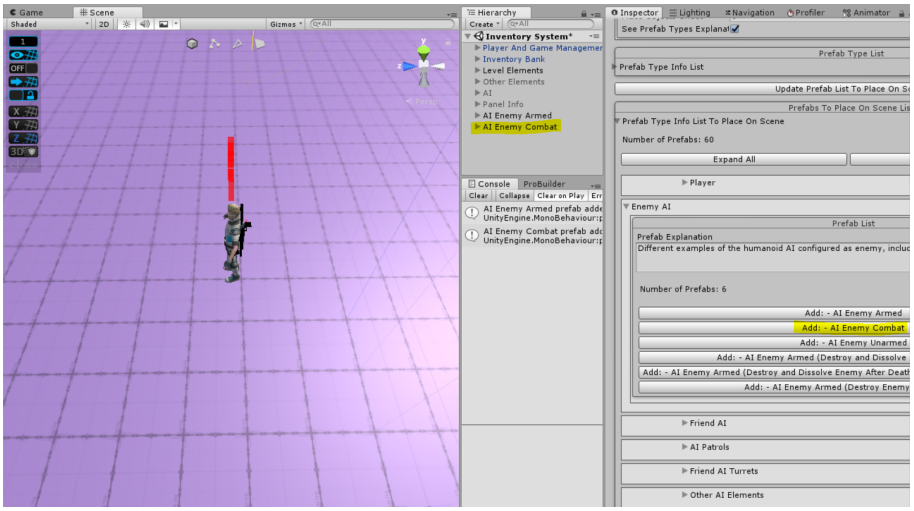


Inside the **Prefab Type Info List**, you can find the current prefab categories configured and the different prefabs configured in each one along an explanation of what each category does.

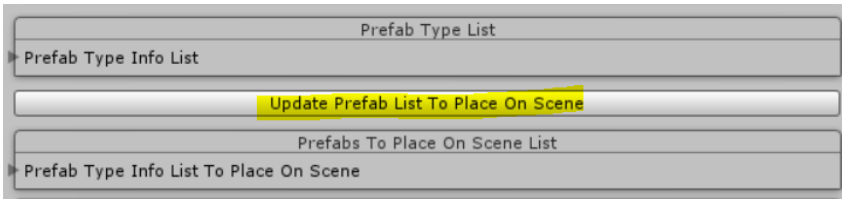
This list is mostly used to configure the prefabs of the project, where you can add your own, so you can use this system as well to place your new prefabs into the scene.

Once you have your prefabs configured here (using the buttons to configure a path for the prefabs folder and using the buttons to get the prefabs objects in those folders), you can use the next list, **Prefab Type Info List To Place On Scene**, which is the one used to place the prefabs in the scene, being a more simplified version, with only the needed elements to make it simpler to use.

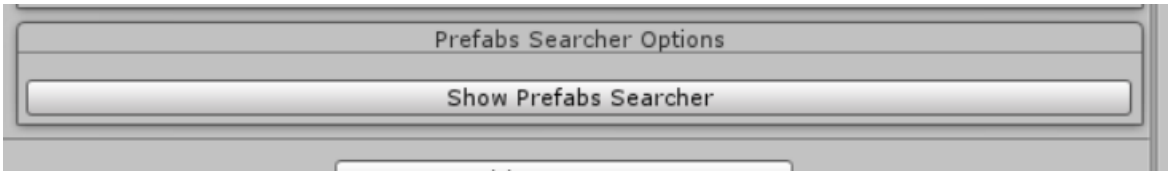
In this list, you can select any category and press the button **Add: "Name of the prefab"**, to place in on the scene, placing the prefab in the position of the level where the current camera editor in the window Scene is looking at, using a raycast to detect the surface:



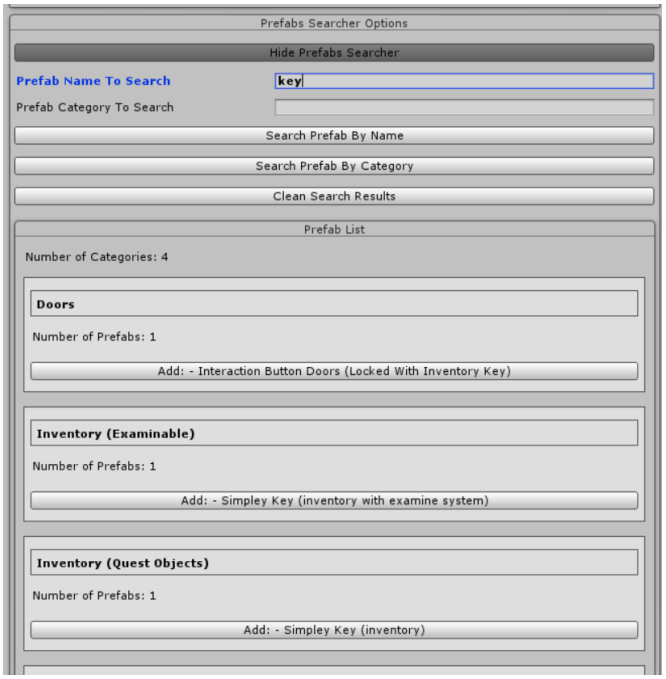
Also, if you add new prefabs or remove others from the first list, make sure to press the button **Update Prefab List To Place On Scene**, so the main second list used to place prefabs in scene contains the same elements



You can also use a searcher to filter the categories and prefabs objects by name, so you can easily find the prefab you need to use. For this, press the button **Show Prefabs Searcher**:

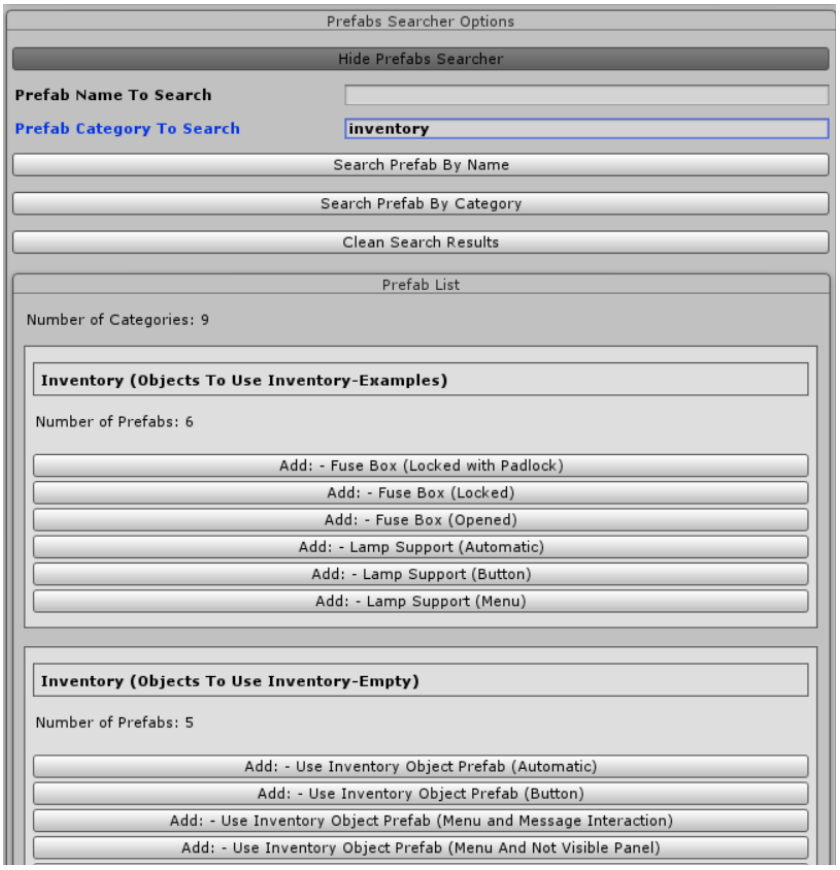


And in the field Prefab Name To Search, type the name (or part of the name) of the prefab you are searching and press the button **Search Prefab By Name**, to get a list of similar results:



So for the results found, you can press the button below each results to place them in the scene.

You can also search by categories, instead of by prefab name, with the other text field and pressing **Search Prefab By Category:**

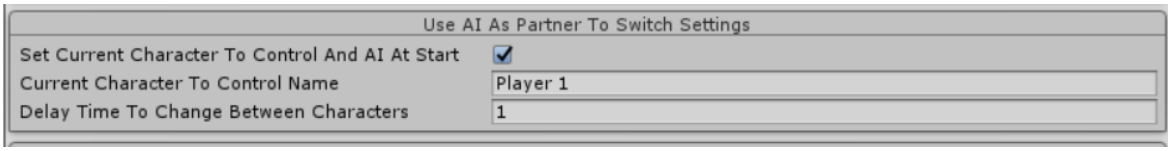


CONFIGURE THE SWITCH CHARACTER COMPANION

This system allows to configure additional characters in the game, setting them as AI which follows the player and can receive orders, like any regular friend AI would do, but in this case, the player can switch the control of these characters, setting the previous controller as AI and ther previous AI as the new controller used by the player. Any number of characters can be configured from 1 to any amount, so this can be used to make games similar to resident evil revelations 2, combining the gameplay with characters with different abilities who need to work together to keep moving in the game.

To configure this, it is the same steps as in the local multiplayer, so check that part of the doc to add 1 or any extra amount of characters.

Once you have done that, in the same **player characters manager** component, set the field **Set Current Character To Control** And AI At Start as true and in the field **Current Character To Control Name**, write the name of the character which will be used as the default player, setting the rest as AI.

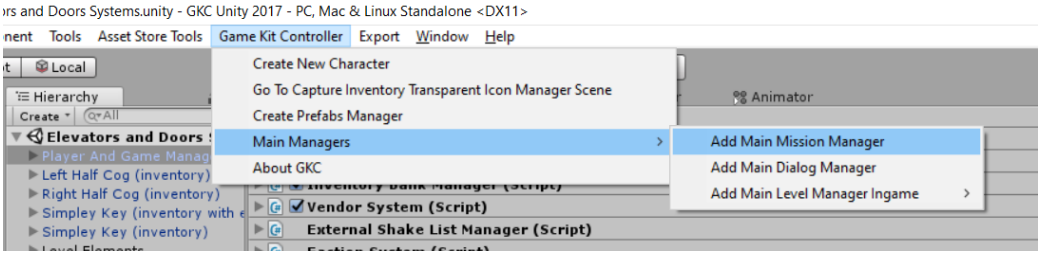


Once the game starts, you can switch between characters. By default, the key to change between characters is . (you can customize these actions keys in the input manager at any moment).

CONFIGURE THE MAIN MISSION MANAGER

This system allows to configure the whole mission system and is used to set some settings automatically on it.

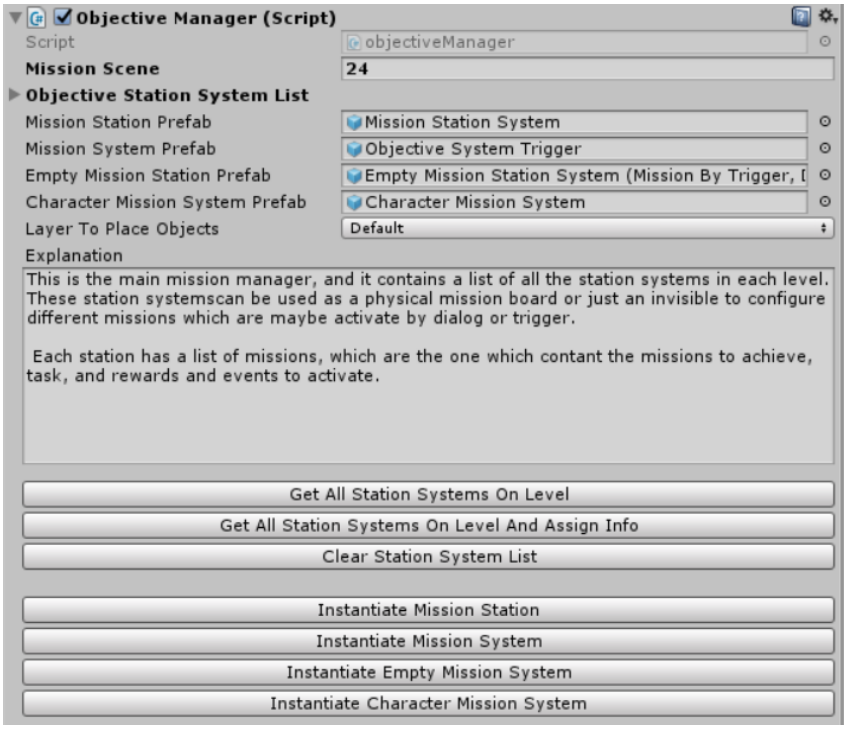
For this, go to the upper tool bar of unity, select **Game Kit Controller->Main Managers->Add Main Mission Manager**.



Then, this system is placed on the level, which is the main mission manager, and it contains a list of all the station systems in each level. These station systems can be used as a physical mission board or just an invisible to configure different missions which are maybe activate by dialog or trigger.

Each station has a list of missions, which are the one which contains the missions to achieve, task, and rewards and events to activate.

Also, it allows to place different mission elements on the level with just a click.



One of them is the **Mission Station**, which can be used as a physical mission board or just an invisible to configure different missions which are maybe activate by dialog or trigger.

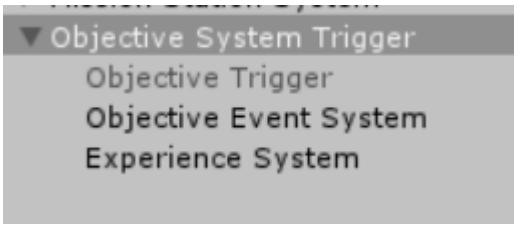
Each station has a list of missions, which are the one which contains the missions to achieve, task, and rewards and events to activate.

Other element is the **Mission System.** which is the component where each mission itself is configured and managed.

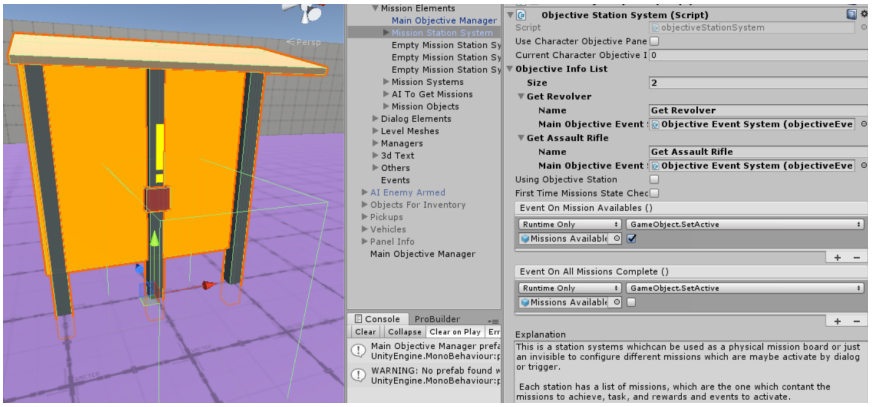
And finally the **Character Mission System**, used to place missions on NPC, giving them to the player directly when the player uses the interaction button, showing the mission panel on characters. To trigger missions with dialog, other way (which will be explained later in videos and on this doc) is used, and which you can already see in the advanced mission demo.

The process to configure missions in the main mission manager is the next:

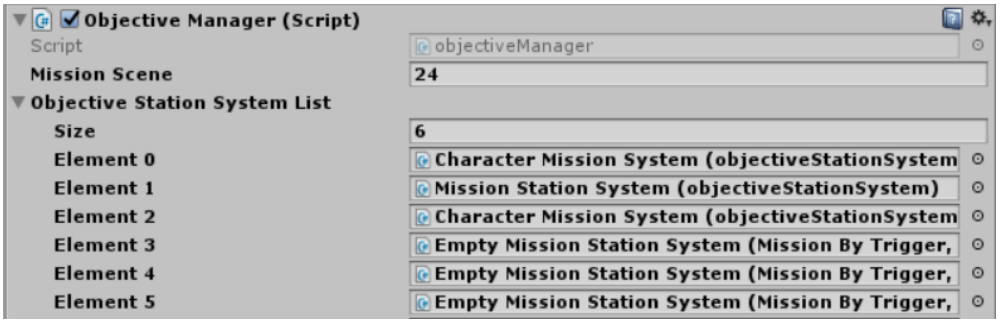
Configure all the missions that you need in the **Objective Event System** components:



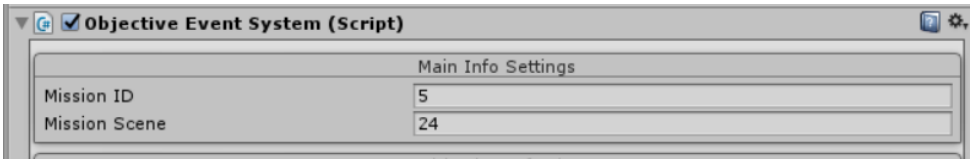
Them, in the **Station System or the Empty Station System** (Objective Station System component), assign these missions according to where you want them to be found by the player, like a mission board, with a character, by trigger, etc...so you assign these missions to one Objective Station System component or other.



Then, go to the **Main Mission Manager** (Objective Manager component), assign the Scene number of the level assigned in the unity Build Settings panel and press the button **Get All Station Systems On The Level And Assign Info**. You will see how all the Station System appear on the **Objective Station System List**.



With it, all **Objective Event System** (each mission configured), gets a Mission Id and the same Mission Scene value than in the main manager.



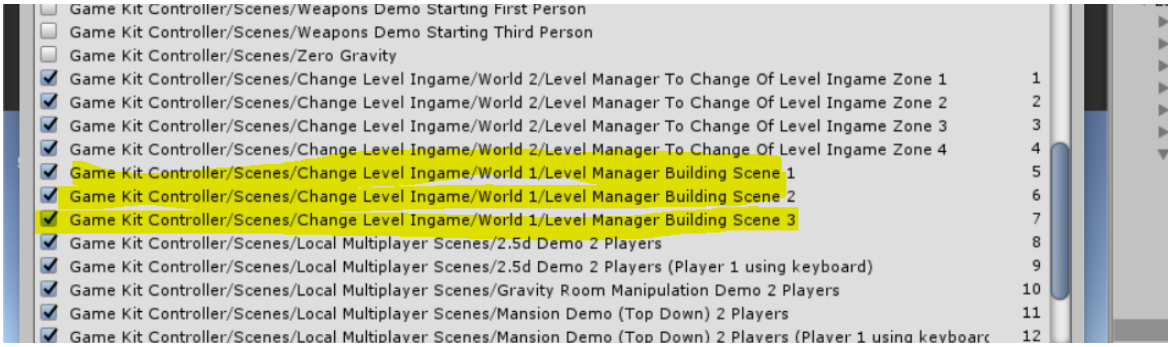
This also allows the system to save the info of the missions found by the player, with those already complete and those to finish yet.

CONFIGURE A MISSION TO START IN A SCENE AND FINISH IN ANOTHER SCENE

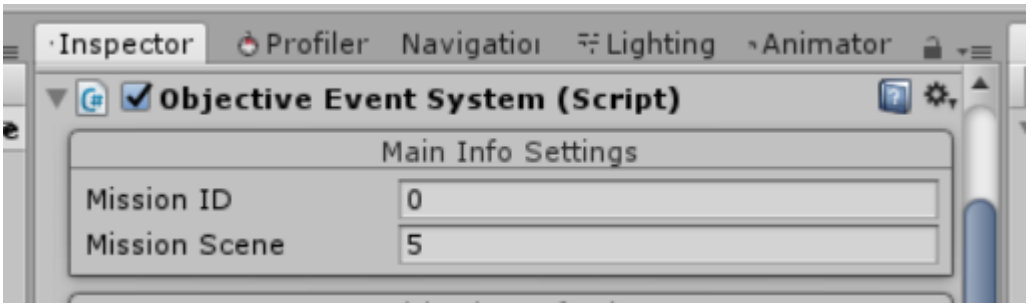
This is very simple, you only need to configure the same info on the Objective Event System component on the scene where it starts and where it ends (the scene where the mission is complete doesn't require the trigger of the mission to be active, since the mission was already activated on another scene).

For this, make sure to configure the same scene ID and mission ID on the fields of the **Objective Event System** components on both scenes, so the system can link the info of one side in the other.

Here an example, where the build settings with the scene index are configured like this:

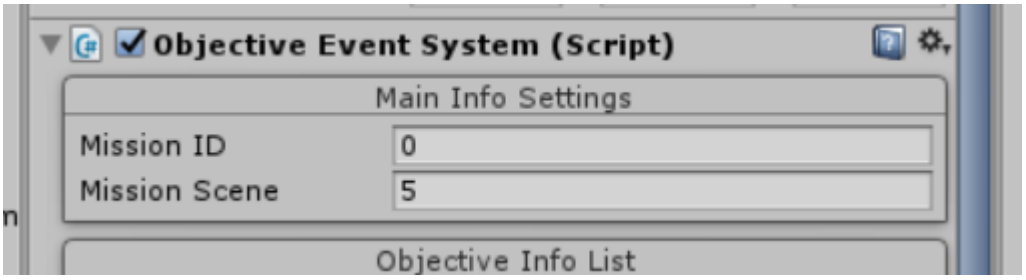


In the scene Level Manager Building Scene 1, this is the mission ID and scene ID values configured on the objective event system component:



The scene index is 5, since the scene uses that indes on the above build settings and the mission ID is 0 (it could be any other value, but in this case, this is the unique ID assigned to it on this scene).

On the second scene, where the mission is complete, which in this case is Level Manager Building Scene 2, the values on the objective event system are these:



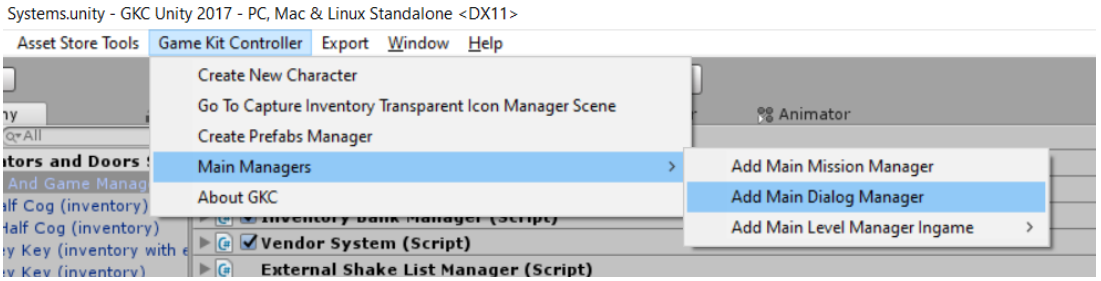
Like previously, same values, being the mission ID a unique value on this new scene (make sure that for example in this case, no other objective event system has the mission ID field with 0, since each mission needs an unique value for this).

It is recommended to check the section CONFIGURE THE MAIN SCENE MANAGER to see more about how the different scenes are managed.

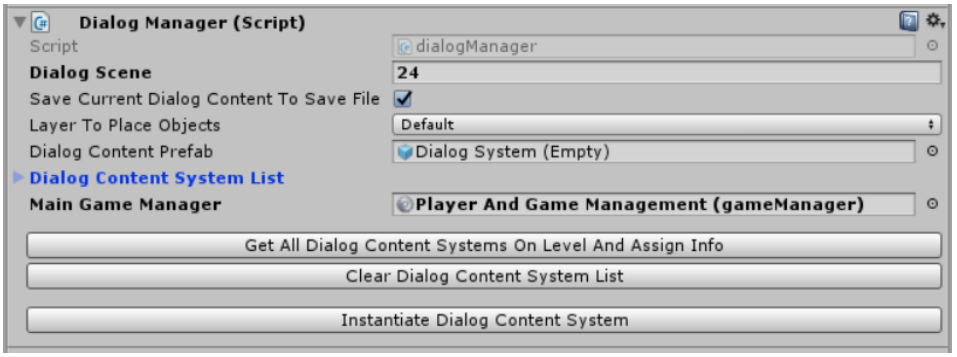
CONFIGURE THE MAIN DIALOG MANAGER

This system is used to save the info of the dialogs, so if a player talks with an NPC with different dialogs, this system will save the current element to play once the player returns to play.

For this, go to the upper tool bar of unity, select **Game Kit Controller->Main Managers->Add Main Dialog Manager**.



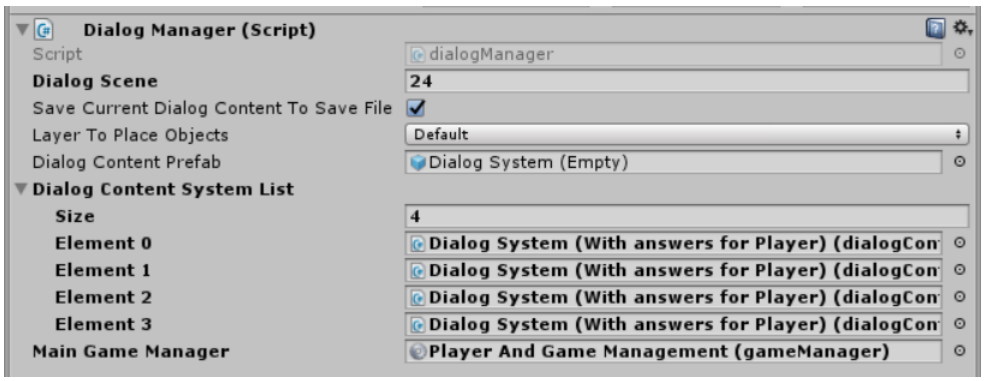
Then, this system is placed on the level, which is the **Dialog Manager**, and it contains the dialog prefab which can be placed in the level at any moment on any NPC, so the player can talk with them.



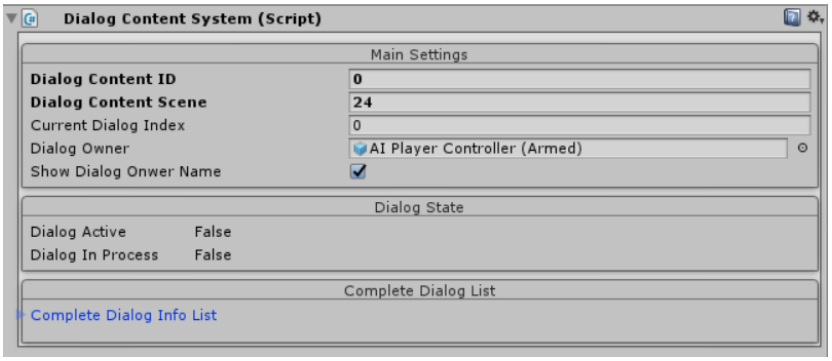
The process to configure is like the previous mission manager. First configure the dialogs that you need (you can check the video tutorial for dialog).

Once you have that ready, go to the main dialog manager, configure the Scene for the current level and press the button **Get All Dialog Content Systems On Level And Assign Info**.

Like that, all dialogs configured will appear on the list.



And also, each dialog will receive the info of the Scene configured.

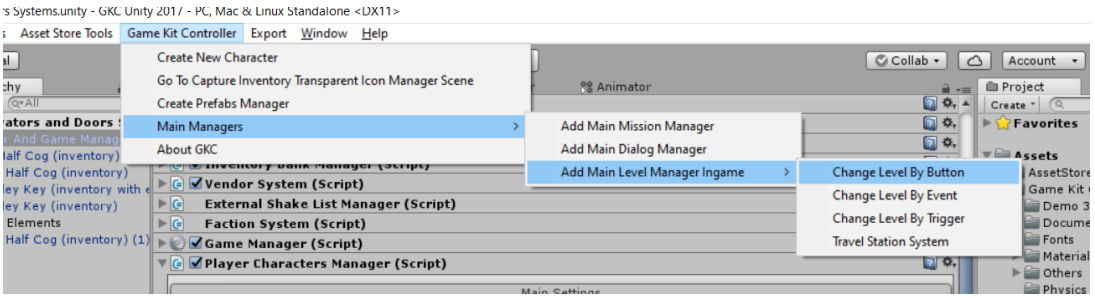

Dialog Content System (Script)
Main Settings
Dialog Content ID
0
Dialog Content Scene
24
Current Dialog Index
0
Dialog Owner
AI Player Controller (Armed)
Show Dialog Onwer Name
Dialog State
Dialog Active
False
Dialog In Process
False
Complete Dialog List
Complete Dialog Info List

CONFIGURE THE MAIN SCENE MANAGER

This system allows to connect different Scenes between each other ingame, so the player can change from one Scene to another using some element of the level, like a button to press, a trigger to enter, an event, like pick an object, using a fast travel station, etc…

Think about this system like in borderlands games, where the player reaches the limit of the level, with some kind of gate where he press the interaction button, and he moves to another new level, being a different zone. And of course, the player can return to the previous one, using the gate where he appears in the zone.

To configure this Scene manager, go to the upper tool bar of unity, select **Game Kit Controller->Main Managers->Add Main Level Manager Ingame**.



In this menu, you can place the different types of gates to change of Scene, according to what you need. The current types are **by Button**, so the player can use the interaction button when he is on the range of the gate to trigger the change of level.

By event, this can be triggered by any object with an event attached, for example, picking ammo from the ground or a weapon or an inventory object can activate the change of Scene.

By trigger, so basically, when the player enters inside a trigger, it will change of Scene automatically.

And also, **a travel station**, a physical trave station which can be placed anywhere on the level and will allow the player to select a Scene to travel (these scenes can be unlocked once the player reaches the travel station of each Scene or by default, have unlocked all the Scene available to travel, according to your type of game).

Once you place one of these **changes of Scene objects** (from now, we will call this object **CSO**) , you have to go to the component **Level Manager ingame**. This component is used to configure the current ID of this CSO (**Level Manager ID**) and the Scene number to travel (**Scene Number To Load**), and inside of that Scene, the ID of the CSO to use as a place to appear **(Level Manager ID to Load**).

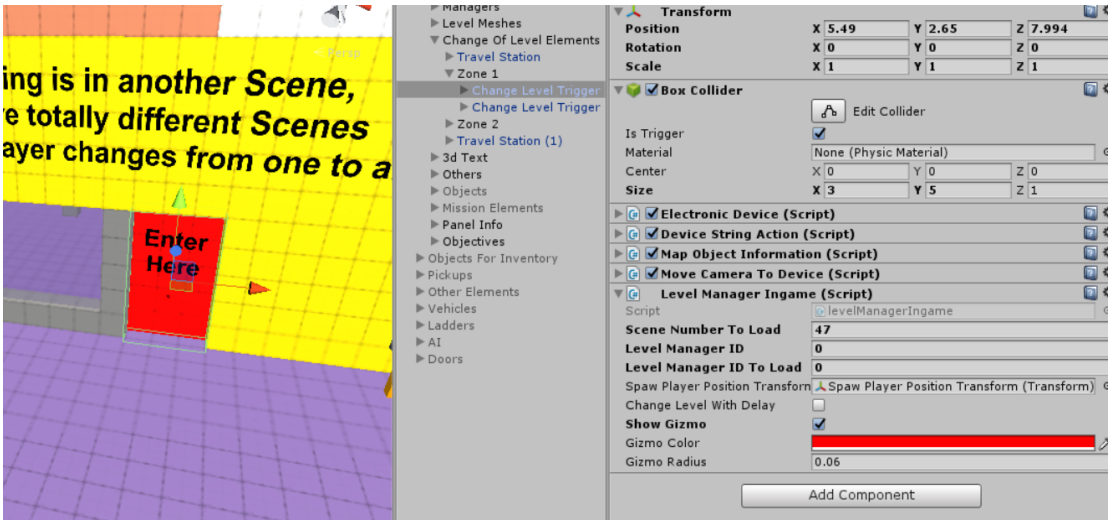
It is like an address, and you can have one or many CSO in each Scene, so each one has its own direction.

You can see an example of this in the Scenes Level Manager Building Scene 1 and 2, where in the first door in the exterior of the building, there is one of these CSO, configured with its ID and the scene and ID to use in that scene, which belongs to the object in the second scene, which has that CSO inside the building, with the ID of destiny.

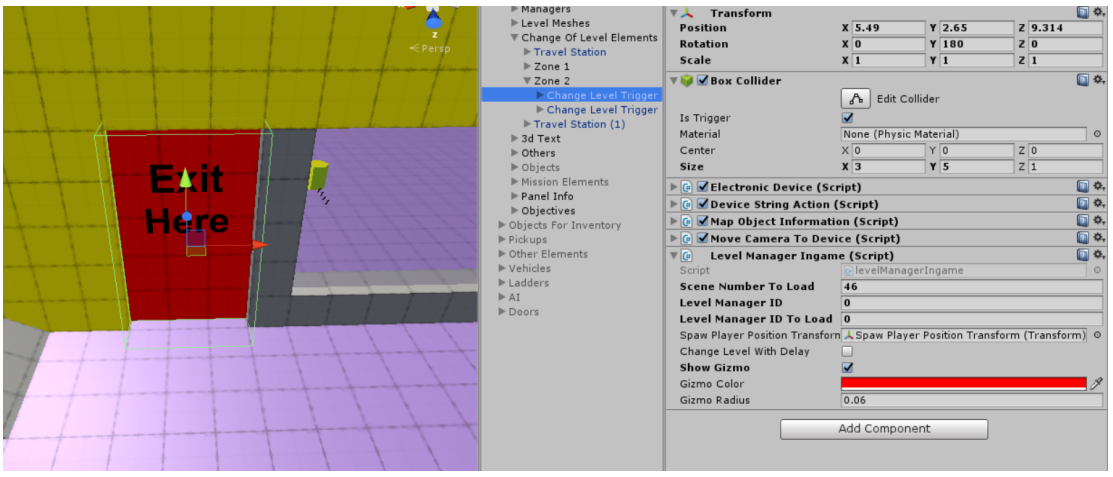
It is very simple but can be confusing for the first time, so take a look at the exterior and interior of this door.

In the exterior, the Scene Number To Load is the scene which contains the interior of the building. The Level Manager Id is 0, since this is the first CSO of this Scene and Level Manager ID to Load is 0 as well since that other object is the first CSO of the second scene, which has the interior of the building.

In this case, the exterior zone is the Scene number 46 and the interior of the building is the Scene number 47.



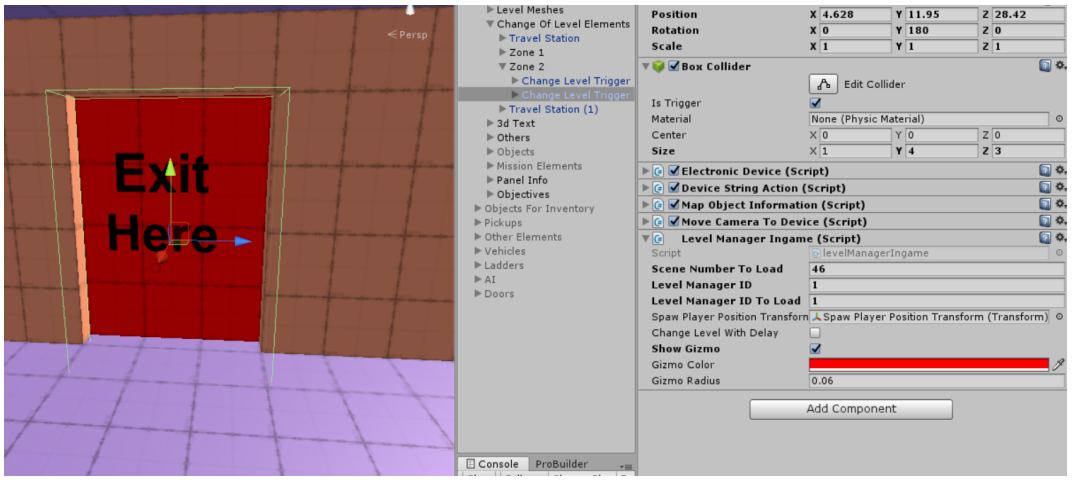
In the Scene number 47, you can see the settings of the other component, which is in this case, is going to travel to the Scene 46. Its Level Manager is 0, which is the address configured in the other scene and its Level Manager ID to Load is 0 too, since this will travel to the above CSO.

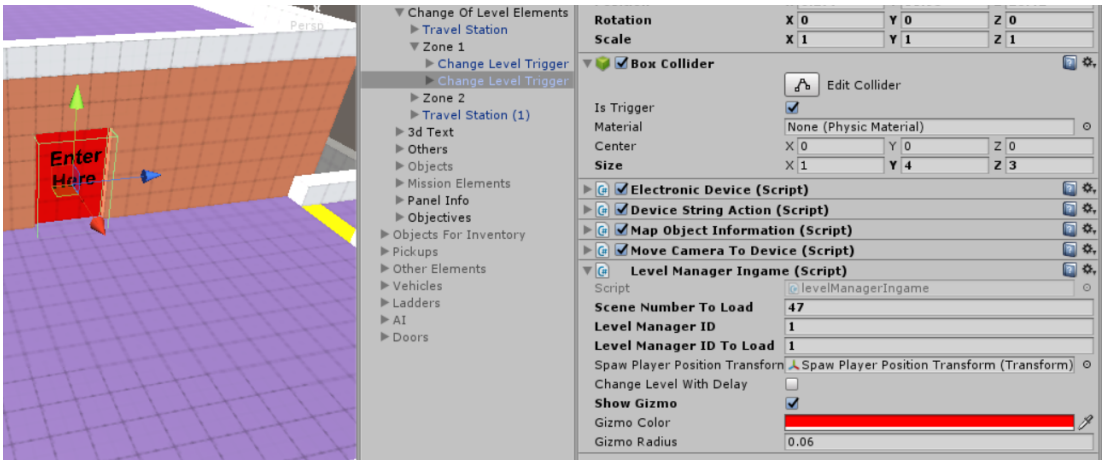


In the component **Level Manager Ingame** you can see the field Spawn Player Position Transform which is the reference position and rotation used to place the player on the level once he changes

of Scene, so each CSO can be considered as “connected” to other CSO configured in another Scene.

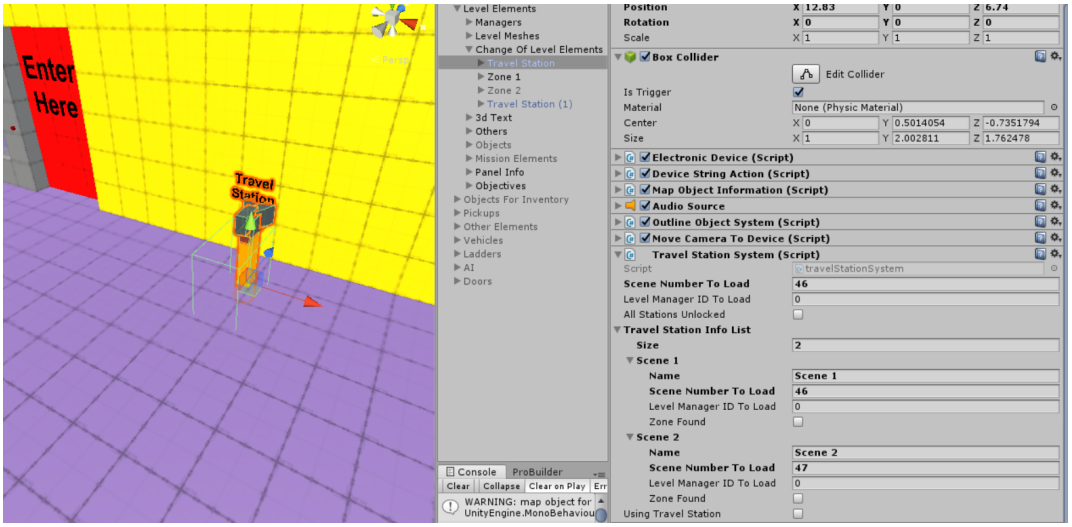
You can see the same example in the second door of this demo example, but in this case, the ID is 1 instead of 0 since this is the second CSO on the level, so the interior points to the exterior object and vice versa.

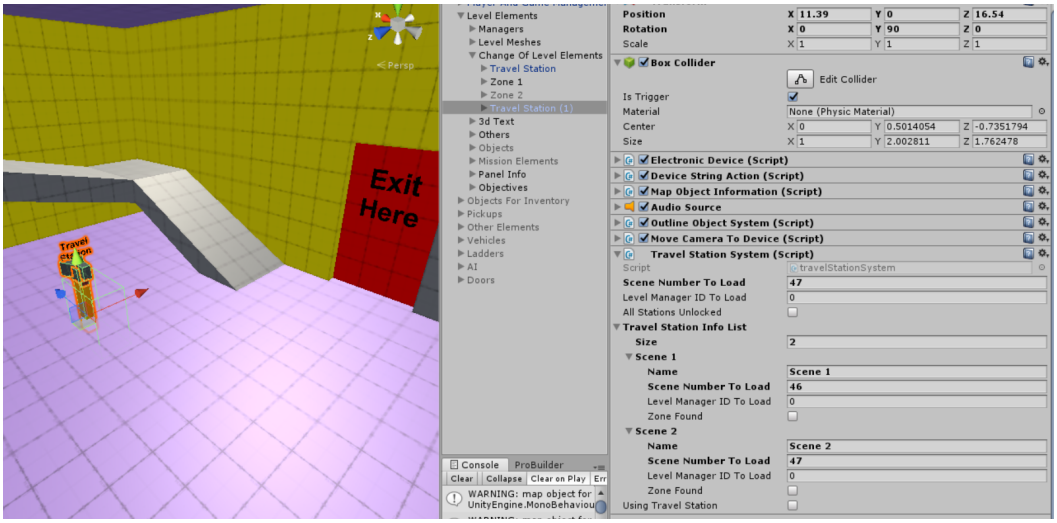




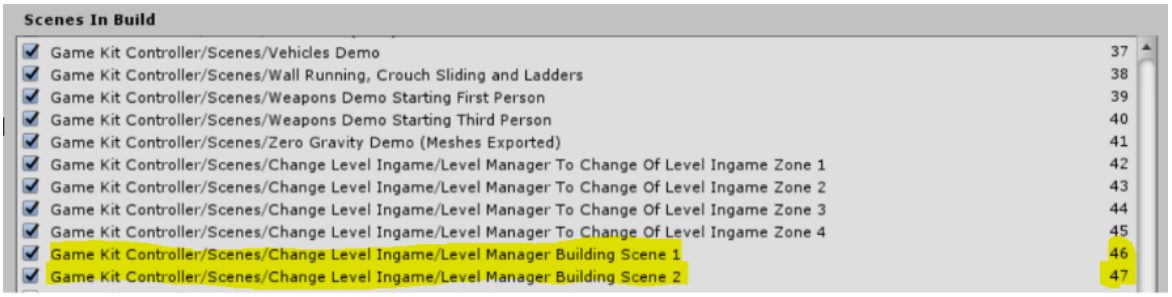
The **travel station** work in the same way. Once you have all you Scenes on your game, you can configure the component Travel Station System, with the list of Scenes info, each one with the Scene number to travel and the CSO ID to go. For example, if a Scene has three gates, you select one of them as the default position of that zone to appear when the player travels to that Scene (this also works the same in borderlands, having a fixed zone to appear in one zone, even if that zone has many gates).

In this case, you can see how the station has 2 elements, one per scene, pointing to the first door of the Scene, with the ID 0, so if the player travels to the Scene 1 it will appear in the exterior, at the side of the door and it he travels to the Scene 2, it will appear inside the building, at the side of the door too.



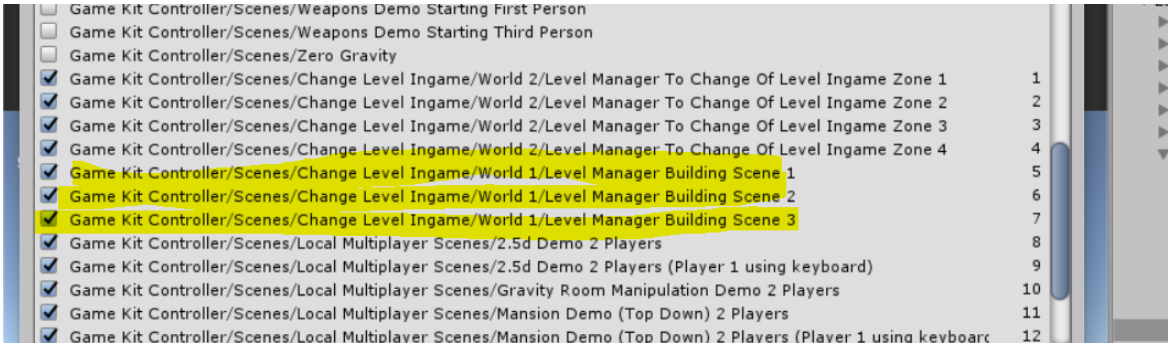


Here you can see also in the **Build Settings**, how the Scene numbers are 46 and 47 for this couple of examples, being each one the exterior and the interior zones of this place.

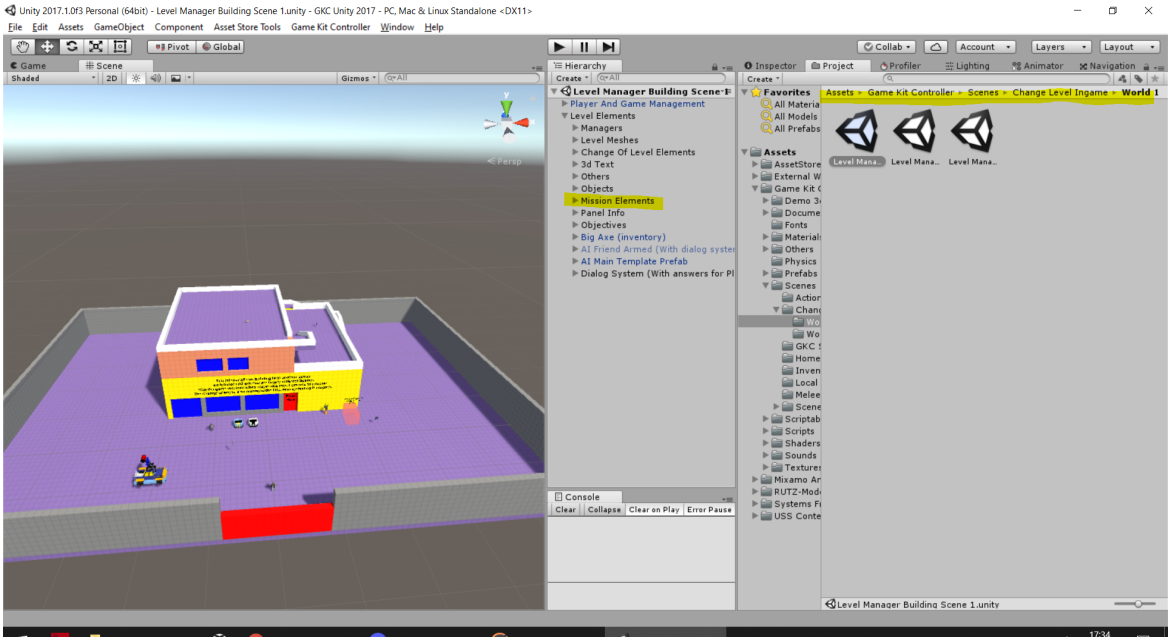


TEST SCENE MANAGER

In order to try the default demo scenes for the scene manager, you can configure the build settings to set the scene index like in the image below, with the same index for each scene:



Using for this the scenes located on this folder:

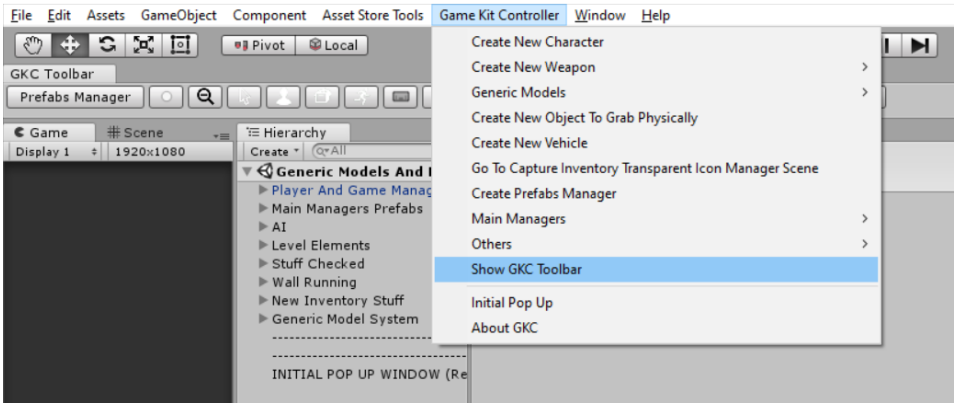


Once that is configured, open the first scene and play there with entering into the house and returning outside and walking also outside of the perimeter through the big red door behind the player.

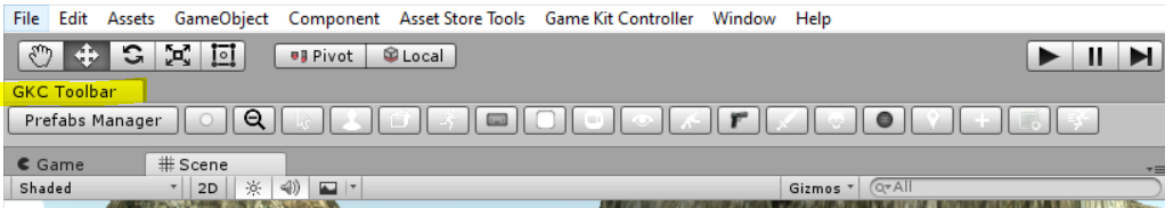
TOOLBAR SYSTEM

There is a new toolbar system which can be used to make shortcuts and quick actions. It is very customizable and it has many actions as shortcuts in order to make the asset easier and faster to use or simplify some steps, like select the main player, focus on a certain component, move the player around, toggle camera view, etc...

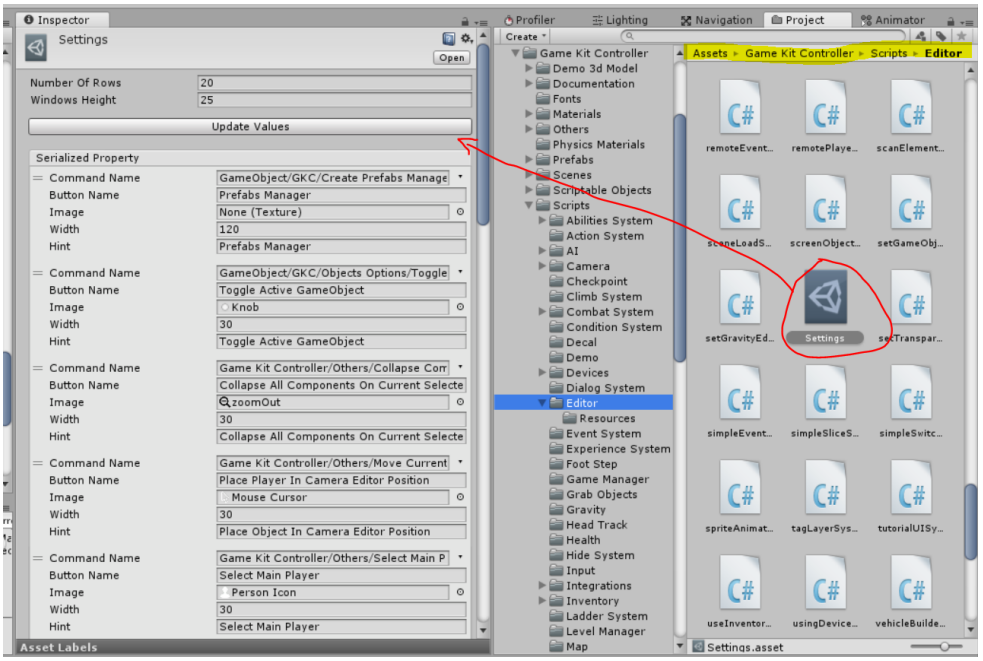
It can be activated here:



And you can place it anywhere on the editor:



And you can locate the file where the settings are configured, including number of rows for buttons and new actions:

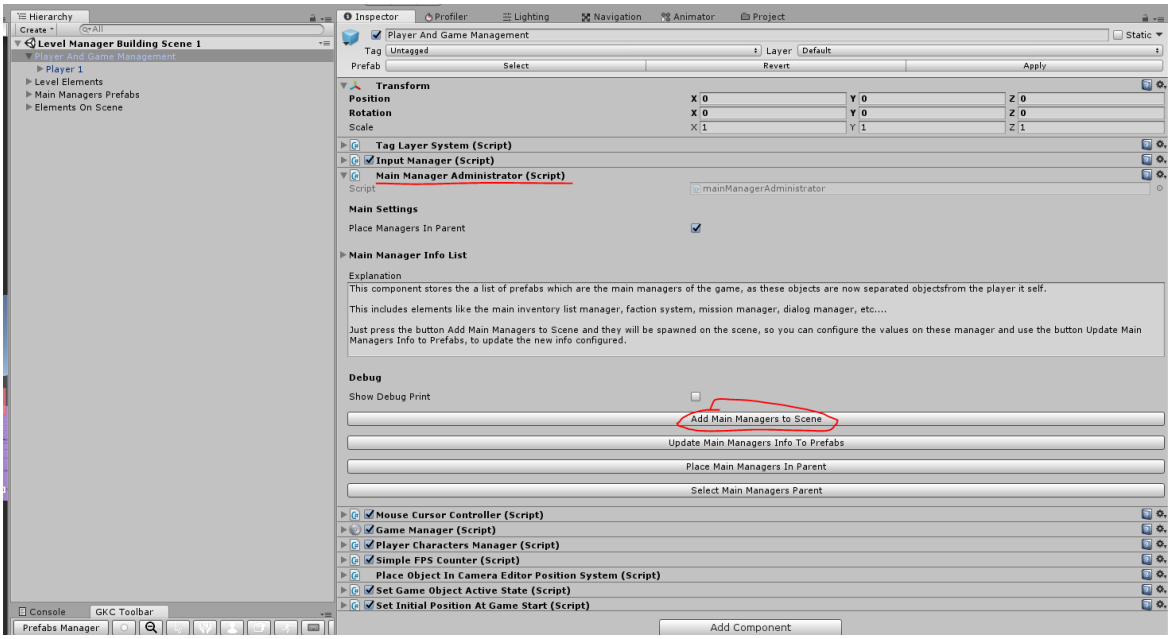


MAIN MANAGER ADMINISTRATOR

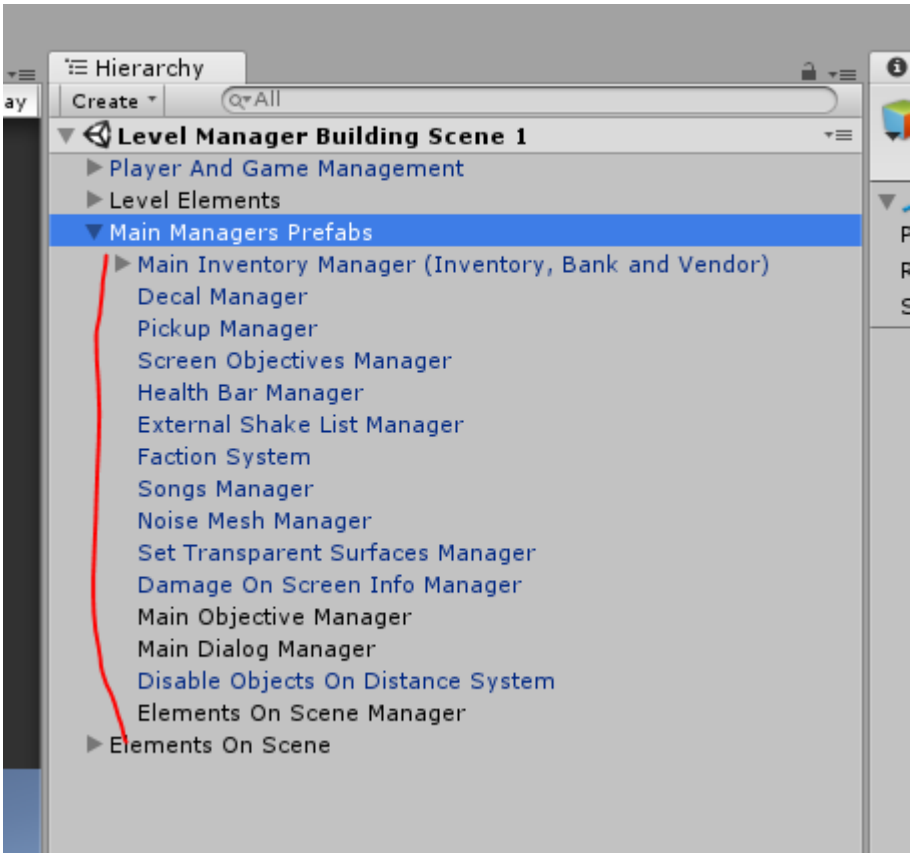
GKC has many main manager, including the main inventory manager, which handles all the inventory info objects created on the asset, or the faction system to configure the relations between groups of AI and player and if they are enemies, friend or neutral and more main systems used to handle different parts of the asset.

This main manager is located on the player prefab parent, on the component Main Manager Administrator, where you can see a list of prefabs assigned there which are the managers themselves. The system automatically drops them on scene at the start of the game if they haven't been dropped manually on the editor.

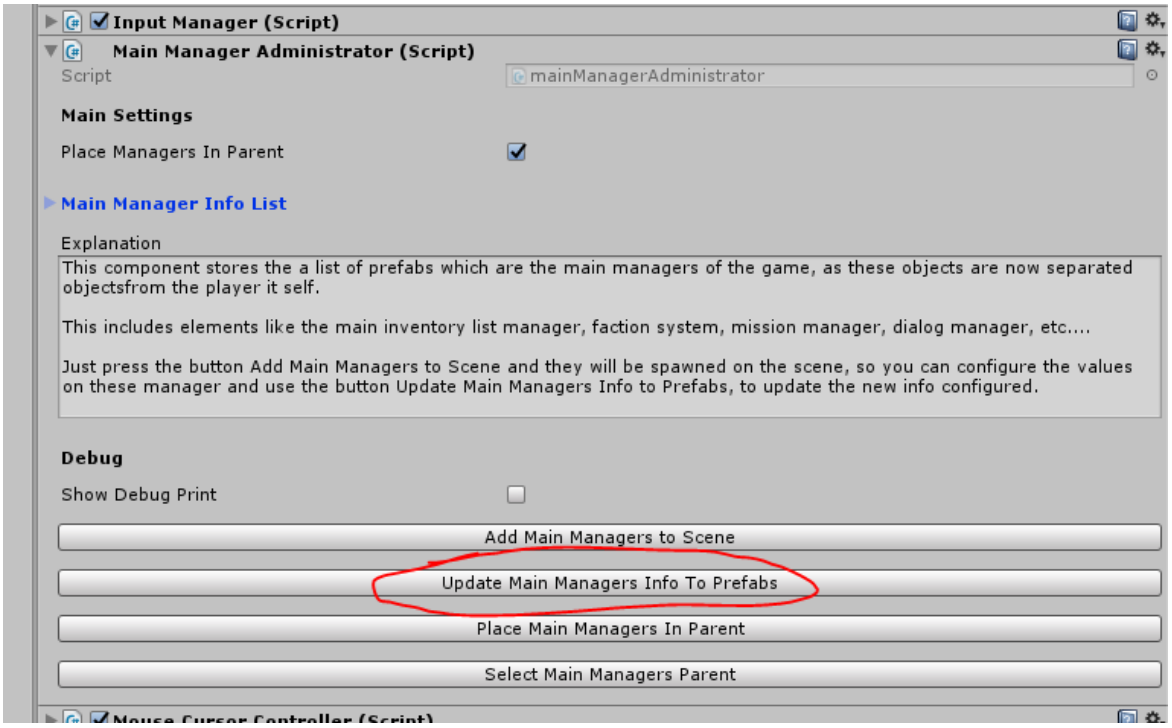
So for this, you can just press the button Add Main Managers to Scene and they will be placed in a new parent object on the hierarchy:



This parent is called Main Managers Prefabs:



Some of these managers are linked to their original prefabs in the project, so you can apply the changes made on them in a scene, like creating a new inventory object, and that info will be applied to the rest of scenes where you have your player prefab placed. You can use the button Update Main Managers Info To Prefabs to update such info.

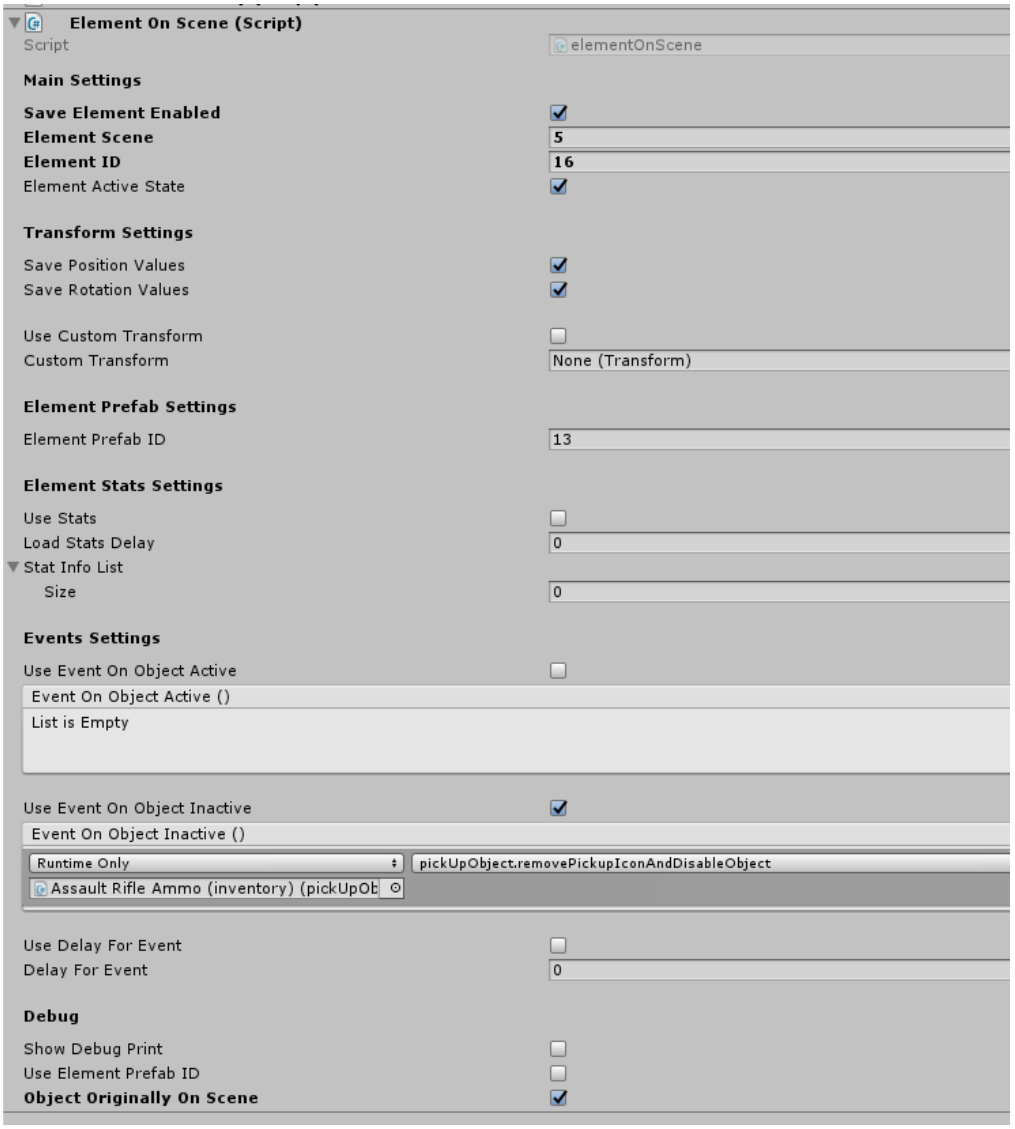


SAVE SYSTEM FOR ELEMENTS STATE ON SCENE

The save system has been improved, allowing now to save the state of objects on scene and keep that info in between games and load/save game and change of scenes, which can be used to keep AI dead, objects spawned on scene, or pickups taken, doors unlocked or locked, places where inventory objects were used, vehicles moved to different scenes, etc….

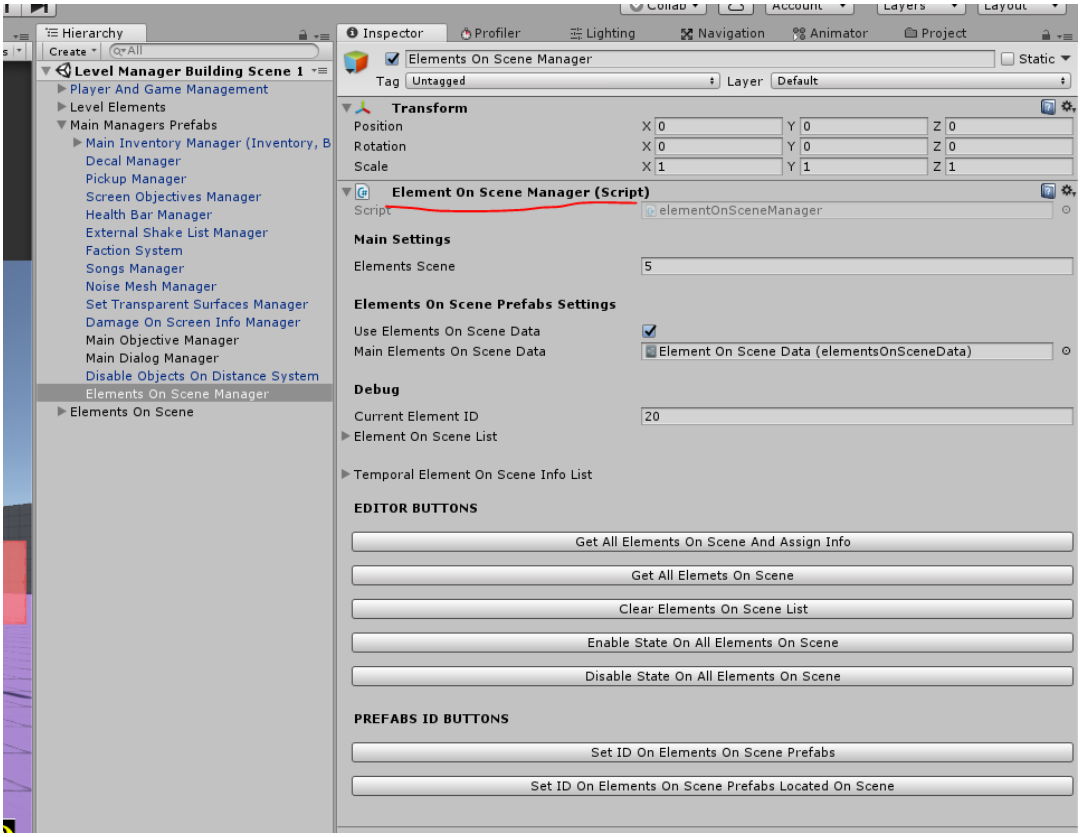
For this, there is a main manager which takes all the objects on scene with a component attached on them, called Element On Scene, which allows to set an id value and configure what elements are saved, including position/rotation and events options to initialize the state when the scene is loaded, so if a pickup is taken, that object will be disabled when the scene is loaded, or if an inventory object like a key was used, it will keep that door unlocked through those events, calling the function unlock on the door.

Here you can see an example of this, with an object having this component and some info, like scene value (which is the scene index from build settings where the object is located) and element ID, to know the id of this object, unique in each one of them.

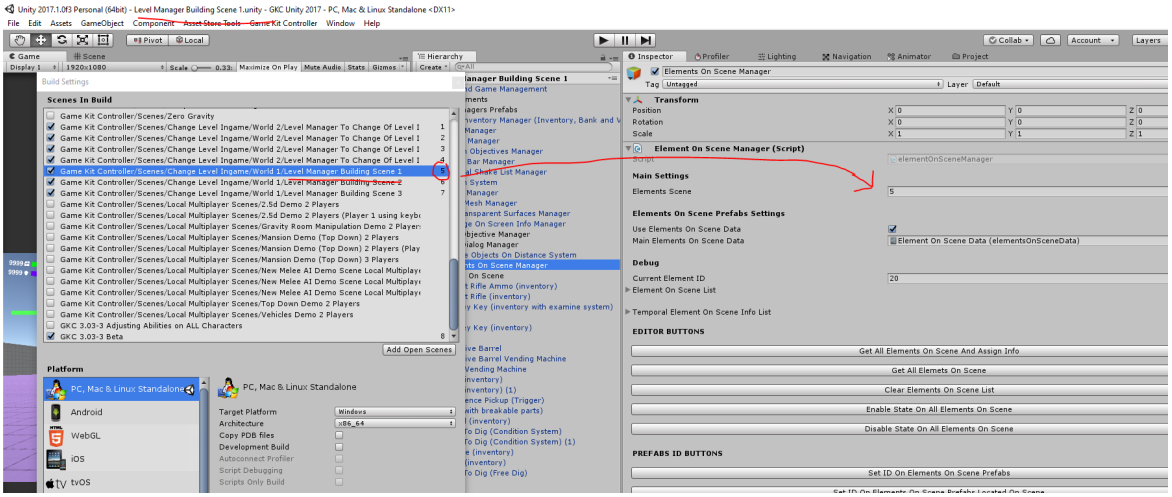


That logic can be used for any type of object to keep its state in between scenes and loaded games.

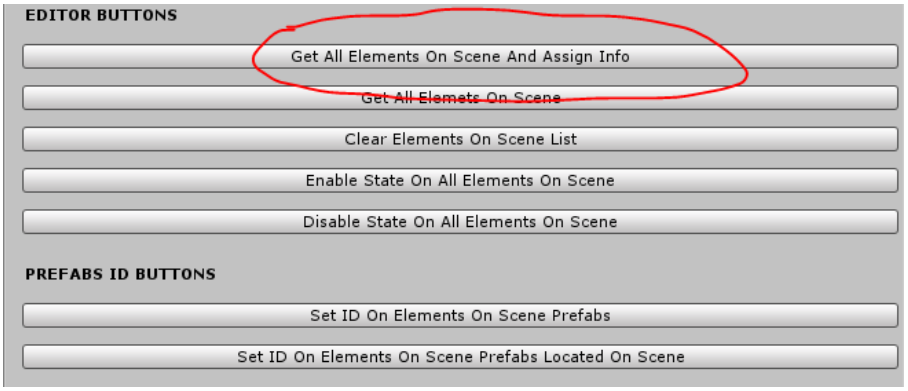
To configure this, go to the Main Manager Prefabs parent (which you can see more info about it on the section MAIN MANAGERS ADMINISTRATOR) and locate the Elements On Scene Manager object:



The logic to configure it is very similar to other managers like mission, or dialogue, so in the Element Scene field, configure the scene index from that scene on the respective build settings, in this case, the scene is configured on the index 5, so set that in the Element On Scene Manager component, like here:



Once that is done and that you put the objects and elements you want to use on that scene, press the button Get All Elements On Scene And Assign Info.



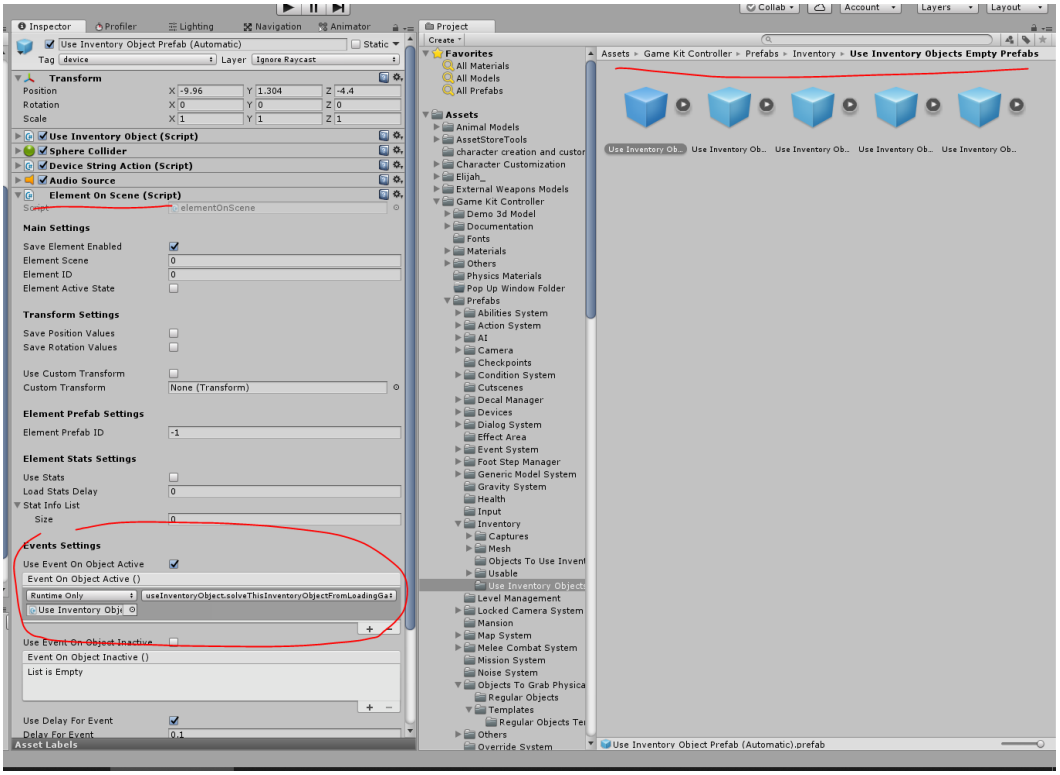
That will assign the scene index and an unique ID for each object on scene. If you remove or drop more objects with info to save, make sure to press the button again, to assign the ID to each one.

All the inventory objects and most of the interaction elements have this component attached to them, so you can either disable or remove them in case you don't want to save that info if your game or certain elements don't need to.

In the same way, you can attach it to any new object or element that you create to save the state of it, for example, for any simple rigidbody, like a box, to keep the info of its position/rotation or any extra value which requires the initialization through the event options on the component.

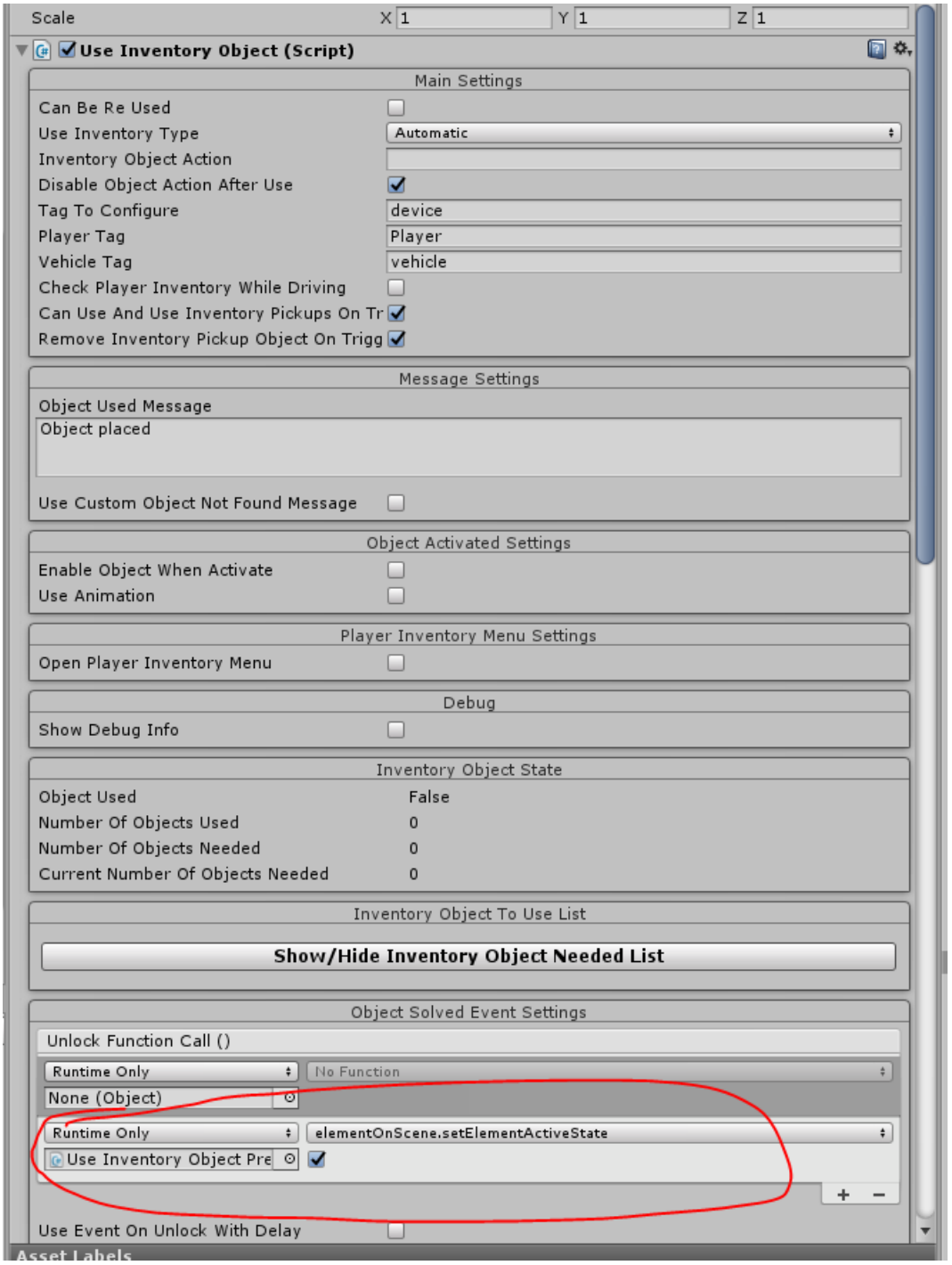
The same scene manager examples of the previous pages contains the main save system configured for all the elements on it, allowing to check how it works and to try them ingame, following the same steps to configure the build settings with the scene index and the scenes related to these demo examples.

Another example of this is the prefabs where inventory objects are used, as they have assigned this component too, so they will keep the info in between scenes, allowing to use the events to initialize their values if the inventory objects on them were used or not:



You can see how the events for the object active calls to the solve inventory object element.

And here how the usage of inventory objects when complete, will call to the element on scene info to set it as active:

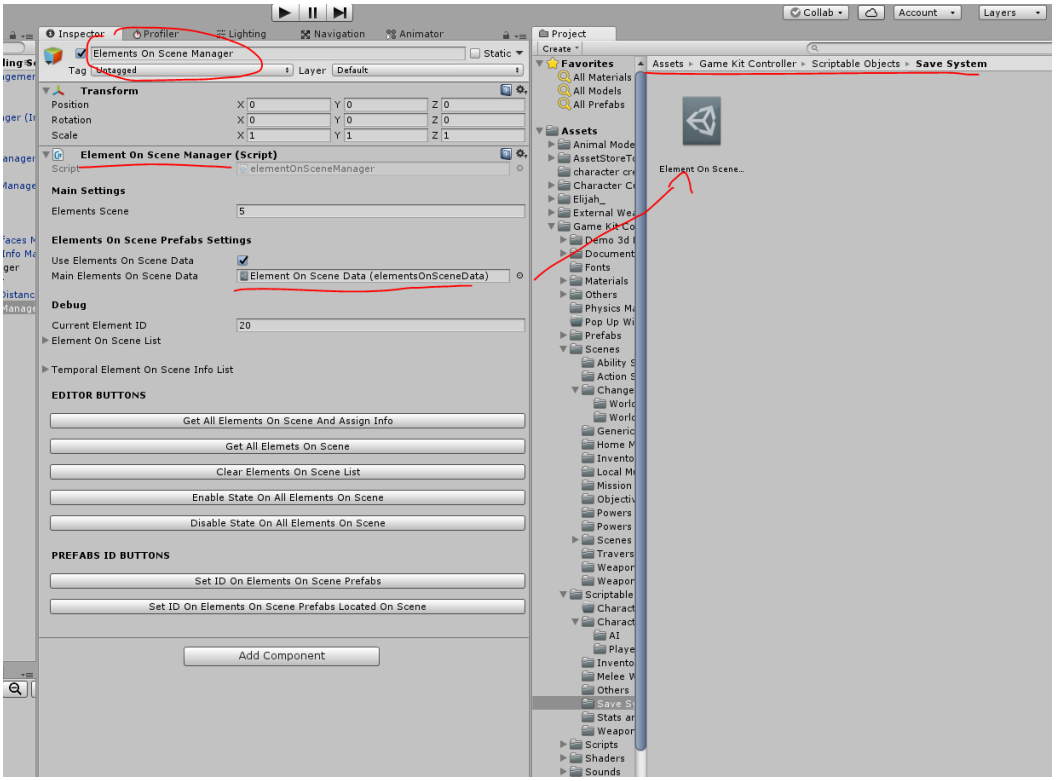


That will make it to call the element on scene event to initialize the object as already complete, so a door remains unlocked, or a device remains active if energy was given to it to turn it on, and so on. So you can pretty much configure any event call to initialize the GKC elements or your own as well.

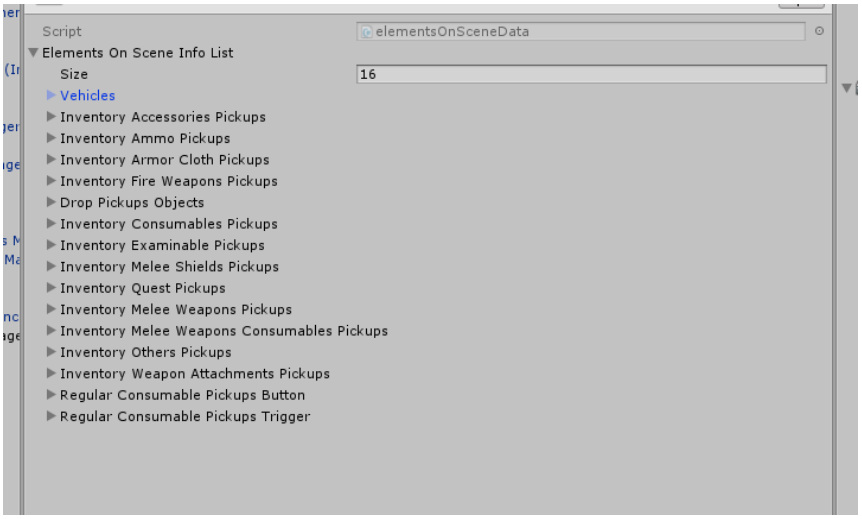
A scriptable object also handles prefabs to organize elements which can be spawned ingame for example, pickups, so even if the object was not present on scene on the editor, the system is able to recognize that a certain object was spawned ingame, to remain there if the game is loaded.

So if for example, you break a crate with pickups inside, these pickups can remain there, instead of being removed and the crate can remain destroyed, to avoid using it again.

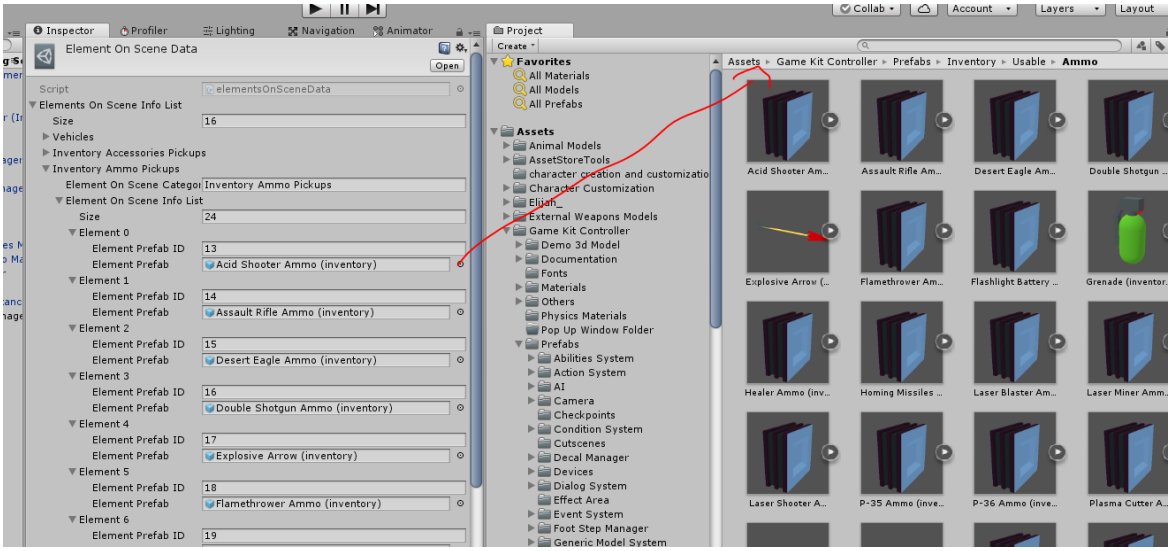
This scriptable object is assigned on the main element on scene manager, and can be found here:



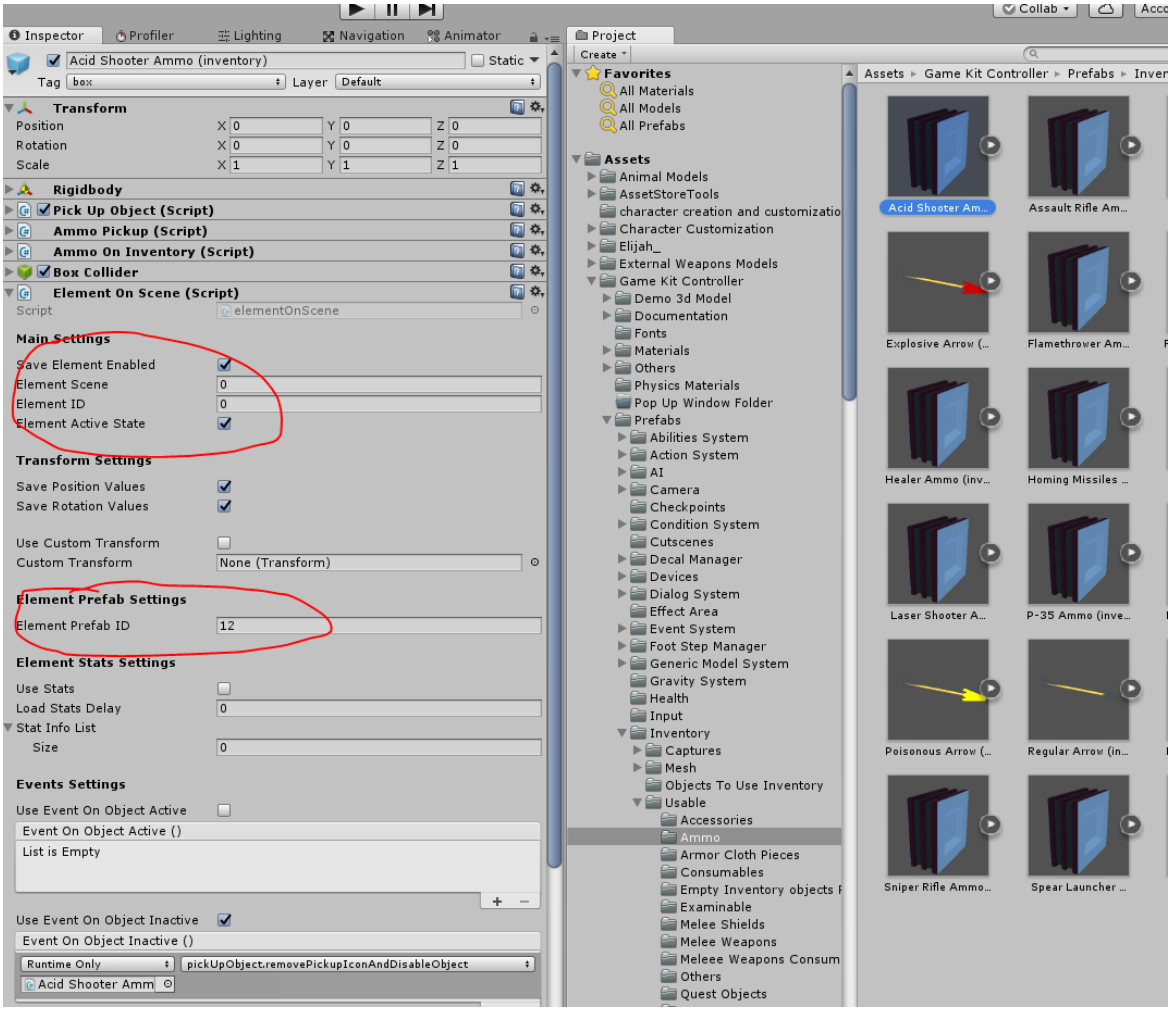
You can configure any new category or use the current ones to organize the prefabs of this type which can be spawned or picked or destroyed from scene as you need:



And in each category, it is a matter of dropping those prefabs which the element on scene component attached to be recognized as element to save state, like inventory ammo pickups:

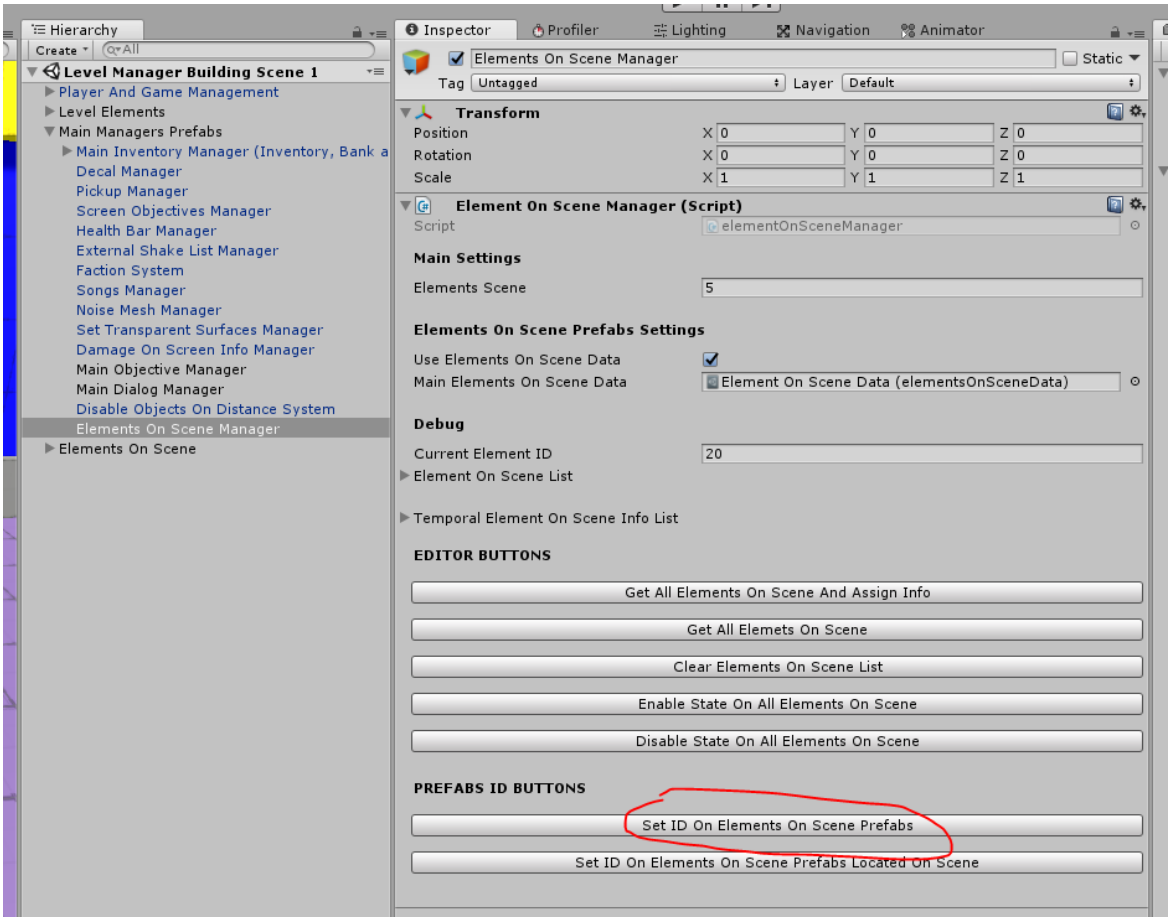


For example, this pickup has a prefab ID assigned automatically by the main manager:



So the system is able to locate it and spawn it if needed.

Once the prefabs that you need are placed on the scriptable object, go to the main element on scene manager and press the button Set ID On Elements On Scene Prefabs.



Make sure to do this if you add or remove prefab objects of the main scriptable object.

HIERARCHY ON GKC PREFABS

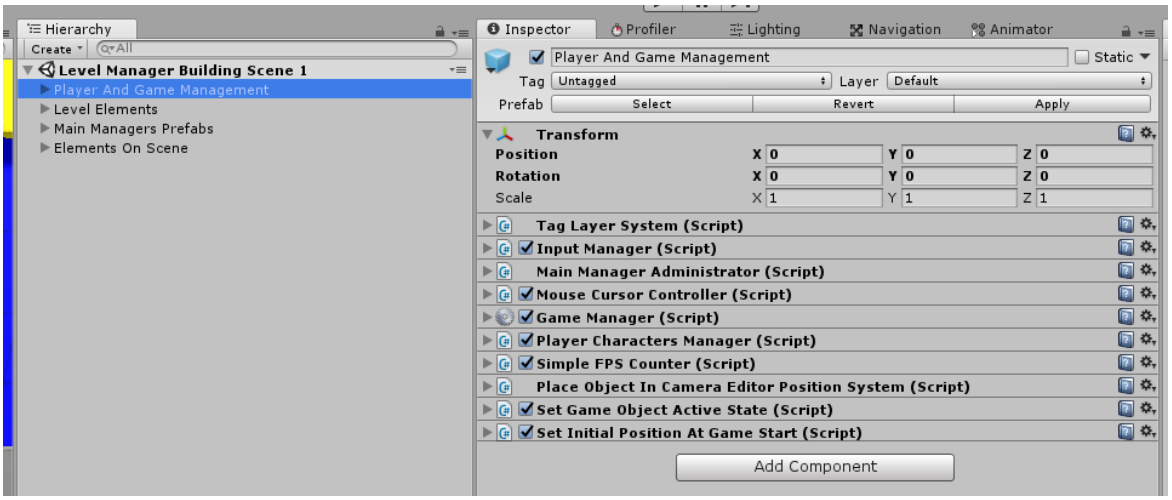
IMPORTANT: THIS HIERARCHY HAS BEEN IMPROVED FOR THE NEW VERSION 3.03-1A, HAVING A MORE ORGANIZED STRUCTURED, THE PREVIOUS ELEMENTS KEEP THE SAME NAME AND USE AND THERE ARE A FEW NEW COMPONENTS, WHICH WILL BE EXPLAINED PROPERLY SHORTLY AFTER RELEASING THIS NEW UPDATE ON THE STORE.

Main Player

Player And Game Management

The main player prefab is called “Player And Game Management” and it can be found in the prefabs folder called “Player Controller”.

It is composed of different elements and here you can see each one of the most important in its hierarchy:



In this level of the hierarchy is where the main managers in the game are located, in the Main Manager Administrator in certain (which is explained a few pages above, on the MAIN MANAGER ADMINISTRATOR section)

Some of the most important components are:

-Input Manager: all the input keys and actions are configured on this component, including the settings for touch buttons and gamepad.

-Pick up manager: it allows to configure all the pickups that are going to be used on the game, as you can add new ones or replace/remove previous and set any type of pickup that you need, organized on categories

-Inventory List Manager: it is used to configure all the inventory objects that will be used on the game. Along with this, the vendor system and inventory bank system are used to configure the info that will be managed in the ingame vendor system and the inventory bank, to store objects in fixed places, similar to games like borderlands, resident evil series and others.

-Faction system: used to configure the different group of characters in the game and its relation between them, including all the type of AI and their relation with them and the player too, to set if they will attack each other, or if their relation is of friendship or neutral.

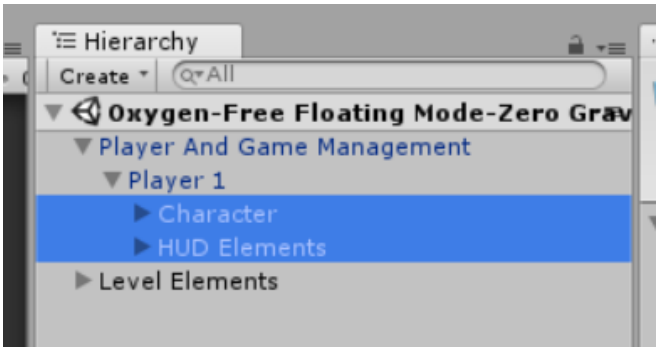
-Game Manager: it contains some settings related to the save of info, like the path used for it and the main name for the files along with other minor settings.

-Player Characters Manager: used for the management of multiple players in the same scene, for the use of local multiplayer and the switch companion AI.

IMPORTANT: the position and rotation of this object, Player 1 and Character and HUD elements must be (0,0,0) for both fields, if you drop the player prefab in the scene, that will be the values by default.

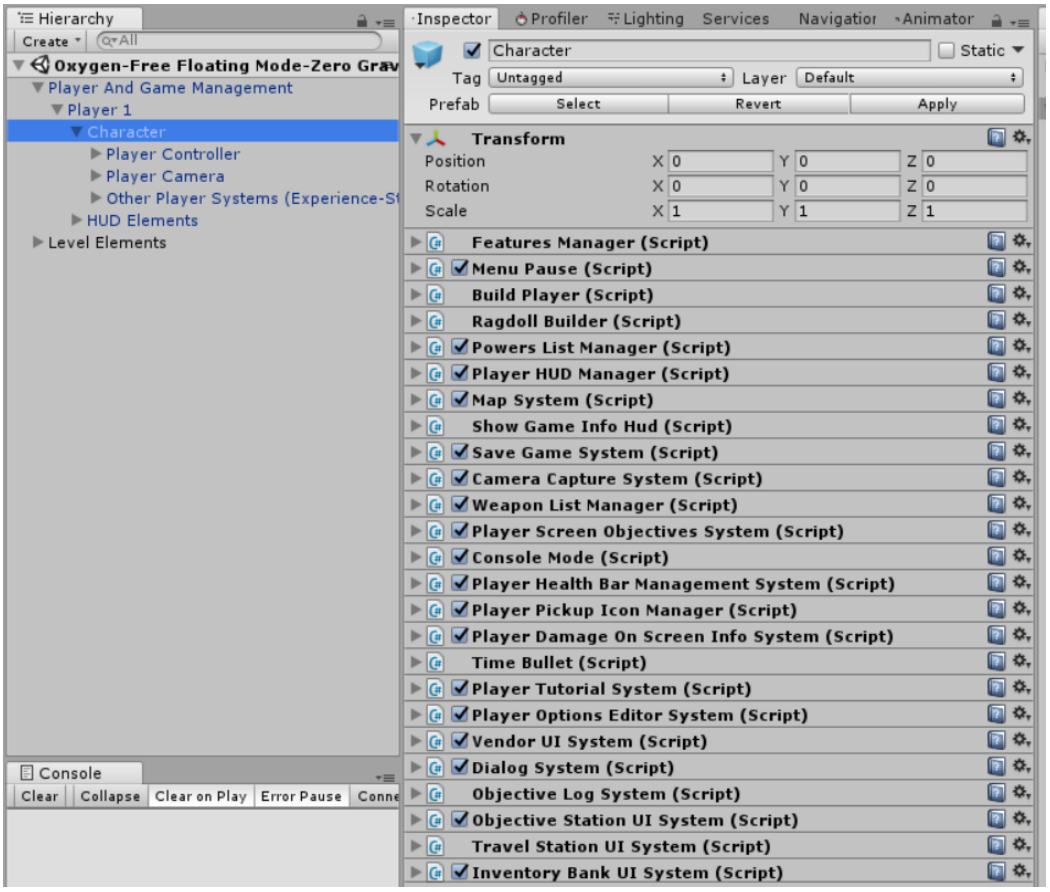
Player 1

On this level, the Elements located are Character, which contains the player it self and the HUD elements, which contains the elements related to the UI and the canvas which contain them.



Character

It contains three main elements, the Player Controller, the Player Camera and third object used to place and manage multiple new systems as they are adding to the asset for the player. This is done like that to make them easier to manage and find, instead of having a single gameObject with a lot of components on it. This also allows components to have components more modular, and make it easy to remove or disable any of them.



The Character object itself contains these main components:

-Menu Pause: it is used to manage the main pause menu and ingame menus, allowing to add new ones (like inventory, map, mission log, etc…)

-Build Player: it is used for the character creator wizard when a new character or player is configured, in order to place and tweak the new model as a new character. It is also where the "Manual Character Creator is located", you can see more of this in this doc.

-Ragdoll Builder: used to add automatically a ragdoll to the character, it is used by default when a new character is created, but it can be used at any moment too.

-Map system: used to manage the map of the scene along with the map creator.

-Save Game System: it manages the main save system menu that the player can use at any moment in game. Of course, the game can use only save stations in fixed places to save the game, instead of allowing you to save on any moment.

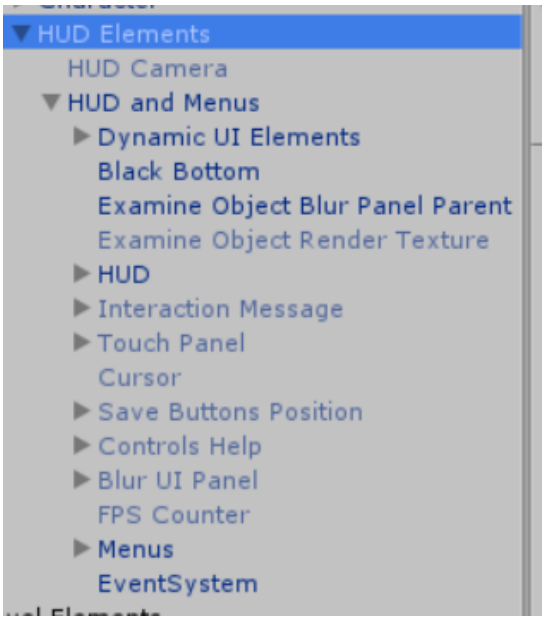
-Camera Capture System: used for different elements, including the photo mode ingame, the smartphone camera tool and the save system to take captures of the scene when the player is saving, using the main camera to take such capture

-Player Options Editor System: used to configure the different ingame settings, related to stuff like graphic settings, mouse sensitivity, movement and camera rotation and orientation values, show tutorials ingame or disable them, game volume, etc…

-**Player Tutorial System:** used to manage the player tutorial UI and the different tutorials configured on it.

The rest of the systems after that are used to manage different elements, such as the dialog UI, vendor UI, the mission log menu, etc…

HUD elements



It contains all the UI elements in the game, including the main canvas, configured in HUD and Menus gameObject.

Inside of it, you can see where the dynamic elements of the game are located (icons which follow positions of objects on screen or that are disabled/enabled to show info about anything needed).

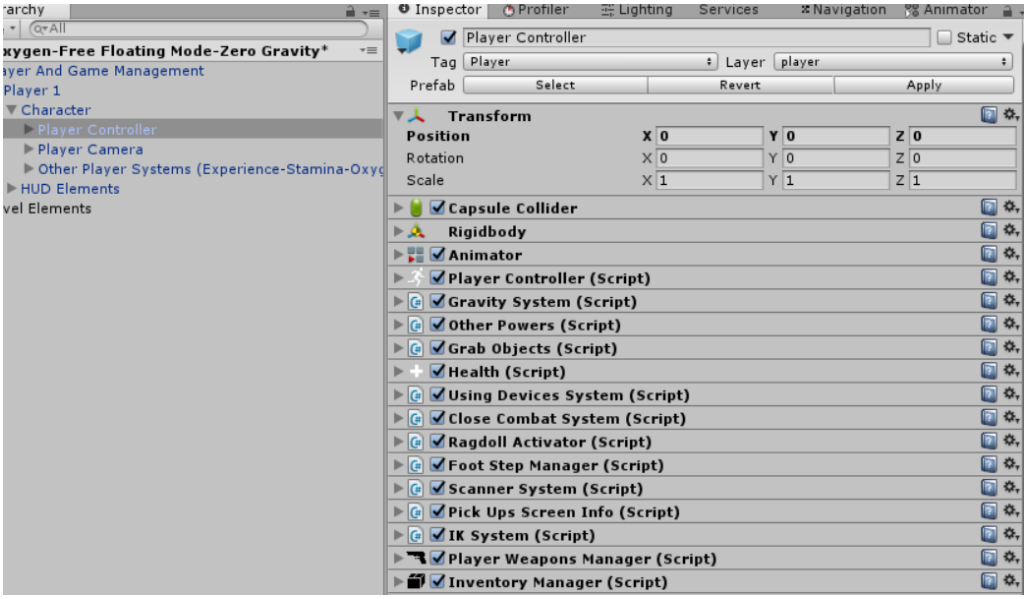
The gameObject HUD contains the main info of the game screen, including the info of the player (health bar, stamina bar, weapons HUD, abilities info, etc…), the ingame menus, like inventory, map, mission log, experience/skills menus, etc…

After that, you can see the touch panels which contain the different touch panels and its buttons for mobile devices.

And finally, the pause menus, inside Menus gameObject, along with the settings, die and exit from the game menu.

Player Controller

This is the main player gameObject it self, though the humanoid AI uses most of the same components than the player to work, which allows to make these components easy to manage and extend, and have available any element/system that is added to the player for the AI, without major modifications or additions. This contains the next main elements:



-Capsule collider, Rigidbody, Animator: objects used on any type of controller on unity

-Player Controller: it manages the movement and animation of the character, including ground and air detection, collisions, velocities, being physics based driven, but with a custom gravity system to apply force in the local down direction of the player, allowing to manage its gravity with different directions

-Gravity System: used to manage the gravity control on the character

-Other Powers: used to configure the different powers that the player can use to fire energy from his hands and use other type of abilities, which can be extended to add new ones, similar to the weapon system or the ability system it self

-Grab Objects: system used to grab objects in a similar way to dead space, with a type of telekinesis, and to grab and carry objects physically on player's hands, using IK for that as well, and managing the collisions and forces of that object, including to drop or throw it.

-Health: it is the main damage detection system, which can be configured on any object that can be damaged, including event options on death, damage, and other states. It has also options to use a shield to avoid damage and allows to detect damage on specific body parts of any object or character, like a hand, arms, chest, head, etc….triggering any event on each part as needed.

-Using Devices System: used to manage interaction with devices and objects on the game, detecting and storing all of them and setting the current that to use according to the settings, like closer to player, closer to the center of screen, the one that is visible on screen, etc...like vehicles, computers, buttons, pickups, chests, etc….

-Close Combat System: it manages the close combat, usually configured as kicks and punches, managing the different attacks configured on it and the combo logic. It also manages the AI to trigger attacks on the AI characters.

-Ragdoll Activator: it manages the ragdoll configured in the character when it dies or the ragdoll state is triggered by external events. It also allows to use dead animations instead of activating the ragdoll directly.

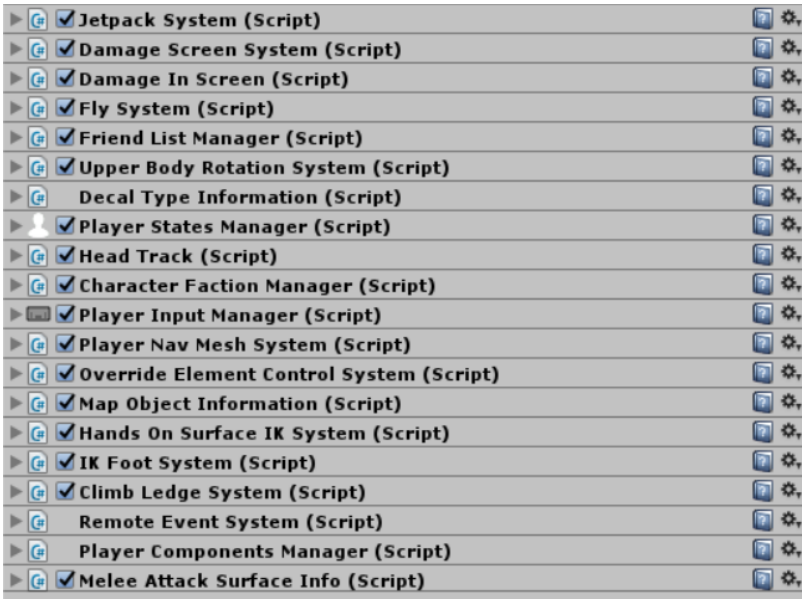
-Foot Step Manager: used to detect the current surface where the character is located and play different sounds, particles and foot prints over it, including terrains, and configure different states, for walk, run, crouch, etc… in order to configure different volumes for the steps and noise values for the AI.

-Pickups Screen Info: used to show the info (name and/or icons) of the pickups taken by the player ingame.

-IK System: used to manage the different elements which have IK options for the character on it, like seats on vehicles when driving, using weapons, grabbing objects, etc….

-Player Weapons Manager: used to manage and configure all the weapons used on the character, including to change between them, activate/deactivate each one according to if the player equips them or not, edit its attachments and all the regular actions on fire weapons, like reload, fire, drop or pick weapons, etc…

-Inventory Manager: used to manage the inventory that the player carries with it, including to configure the initial inventory that the player has at the beginning of the game, which can be empty or contain certain items or weapons, which can be configured as equipped at the start too. It also allows to combine, drop, use, examine and more actions with the objects, including to show the weapons slots equipped ingame.



-Damage Screen System and Damage In Screen: used to make the damage visible on screen, with a red screen and arrows showing the direction of the damage, and showing damage icons numbers on screen for each object that receives damage and which is visible on the screen (similar to borderlands) respectively.

-Friend List Manager: used to manage the different friendly AI found by the player, to give orders to it. It can be activated with K by default.

-Upper Body Rotation System: it manages the upper body orientation when using weapons or powers that requires the player to aim in third person, so its upper body can rotate separately from the lower body to aim up, down, and to the sides, as any other third person game does.

-Player States manager: it allows to configure different modes for the character and change between them ingame through input or events, like having a weapons mode, powers mode, close

combat mode, melee weapons mode, simple mode without nothing extra or special to activate, etc...Basically, a list of states with events to trigger for each one.

-Head Track: it manages the head and upper body orientation to make them look to a certain position, with certain clamp limits and according to the forward direction of the character, using IK for both body parts.

-Character Faction Manager: used to configure the faction where that character belongs, so the AI is able to identify each character and attack them if they are identified as enemies or leave them alone if the other character is friend or neutral.

-Player Input Manager: once all the input actions are configured in Input Manager, the player gets that list of actions and to each one, an event is configured along with setting the type of press key (down, hold or release) to activate that action. So it makes the player call the jump function when the key jump is pressed through an event calling such function. It also manages the gamepad and mobile input, along with the axis for movement and camera.

-**Player Navmesh System:** used to activate the movement on the character based on navmesh navigation, used most of the time for point and click control type.

-Map Object Information: used to configure an object as an element to be shown in the map system as an icon, like a door, an AI, the player, an elevator, etc...

-Hands On Surface IK System and IK Foot System: IK system based used to place the feet of the character on the ground according to the collider shape of that mesh, so in stairs, slopes, etc.. the feet are placed visually on top of the mesh, instead of being always flat in the same height, looking awkward when the component is disabled. The one for the hands makes the character to place his hands dynamically on close surfaces without animations, making both systems to feel that character more alive and interact with objects of the scene organically.

-Climb Ledge System: a system used to detect dynamically when the player is on the air and close to a ledge that could be grabbed, so the player can hang from it, climb it or lose it or jump from it on the air again. It also allows it to grab to a flat wall (instead of the ledge at top) by getting close to the wall in the air and pressing the E (interaction key).

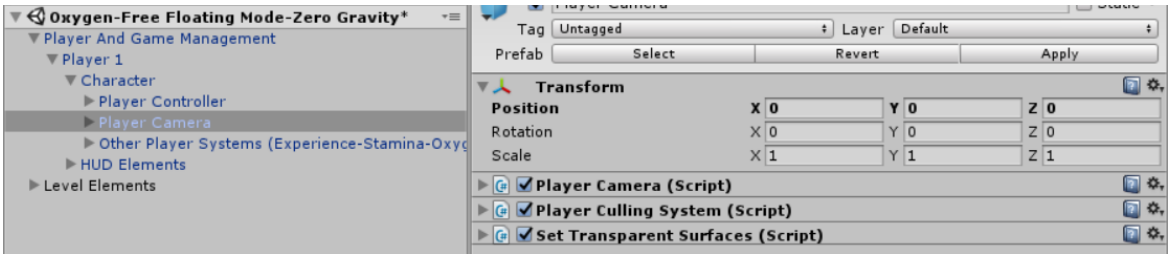
-Remote Event System: a system which allows to configure a list of events and assign a name to each element of the list. Each one of these elements can be triggered remotely by another object through an effect area, on trigger or by certain components or other events, as the system is made in order to avoid having direct references to the object with the event to trigger, so each object can be different prefabs without any relation.

An example of this is that the player can activate its sleep spell ability which basically uses a sphere cast to detect objects inside a radius and trigger their remote event system, searching for the element with the name “Activate Sleep Spell Cast”. That has an event configured on each AI to call to the ragdoll state to make them fall and be in that state x amount of time.

So that type of reaction is triggered without any direct reference to any specific component or object of the AI and the player, even if the AI is spawned dynamically at run time. This can be used on multitude of situations, for example, for an ability hack which basically detects an object by raycast and call its remote event system component to call the event called “Hack”, to unlock a door, make a device to explode, disable the energy of a generator, make a vehicle to activate its self destruct function, etc...

Player Camera

It contains the main camera used for the controller, including:



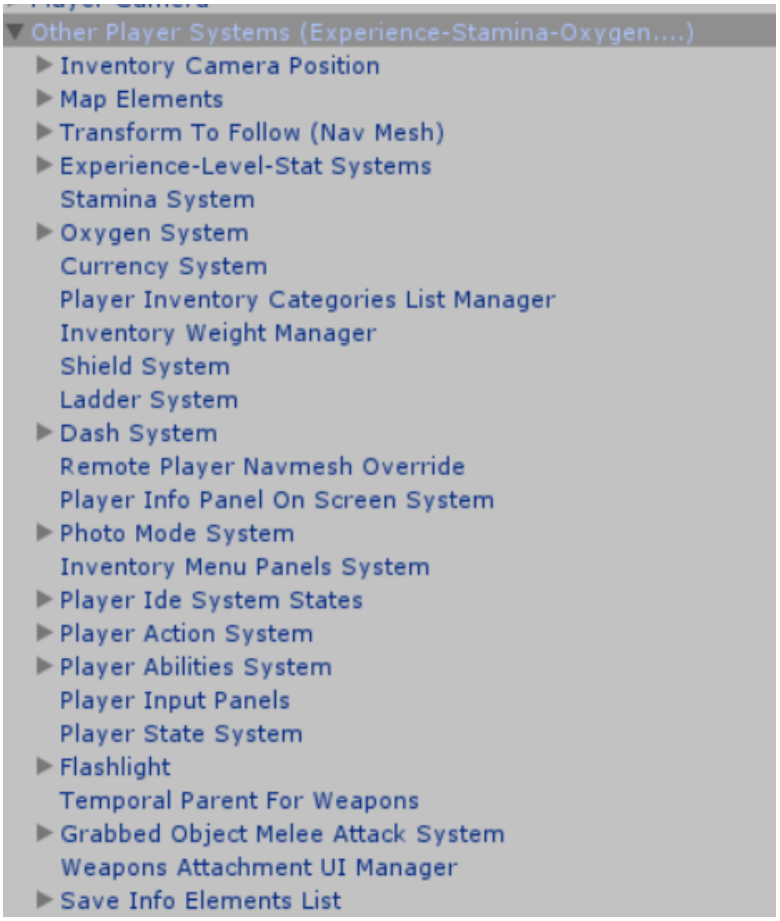
-Player Camera: it is the main camera manager for the player, taking care of the camera on free view (third, first person camera) and in locked view through the locked camera system (fixed camera, top down, isometric, point and click, 2.5d, etc…). It also manages the different camera states configured on the camera and the collision with surfaces, along the lock-on targets on screen.

-Player Culling System: used to fade the player's materials smoothly when the camera is close enough to it, so the camera view is not blocked by its meshes, something made in a lot of games in third person.

-Set Transparent Surfaces: used to detect objects between the player and the main camera with a certain layer configured on it (Transparent Mesh), allowing to keep collisions with them but seeing through them and fade their meshes to keep the player visible on screen. This is used on certain objects of the scene in different games, like columns, certain walls, trees, etc…. and in most games with an isometric or top down view.

Other Player Systems

This gameObject contains a multitude of different systems, which have been added in previous updates, having placed each new system/component in its own empty gameObject, to make these systems easy to manage, locate and disable/remove/extend and get a more modular asset.



Most of them have a name very related to its function on the asset:

-Inventory Camera Position and Map Elements: it contains the camera used for the inventory menu, to examine objects in 3d view on that menu panel and the camera used for the map menu respectively.

-Transform to follow: it is an empty transform used as reference for the player navmesh system, as a reference for the target position to reach.

-Experience-Level-Stat Systems: it contains the stats and skills systems used to manage different values (remain health, money, stamina, energy, etc…) on the player and its skills configured (liek double jump, grab objects, throw objects, gravity control, powers, abilities, etc…) and save/load its info between games, along with getting values in game for those systems that could need to access to that info quickly and without need to get references to each specific component.

It also contains the main experience system, to get XP and configure each level of the player, for RPG elements, setting the skills to unlock or the increase of stats values to obtain on each new level. It also has the experience menu, to see the statistics of the player and the skills menu, to unlock or improve abilities configured on that system. Of course, these elements are configured in different components in this object.

-Player Inventory Categories List Manager: it allows to customize categories for the inventory menu, so the player can filter ingame the objects on its inventory for categories, instead of seeing all the objects at once in the inventory grid.

For example, to check only the weapons that he carries, or its ammo, the main quest objects, consumables, etc…

-Shield System: used to absorb the damage received by the character until its value reaches 0 and can be only refilled after x time without receiving damage or getting shield pickups on the scene, in a very similar way to borderlands games.

-Remote Player Navmesh Override: it allows to change between free control of the player and movement based on navmesh, setting an specific position of the scene to be reached by the player, though events for example.

-Player Info Panel On Screen System: used to show panels in the screen for text, allowing to configure any type of text panel, with different size, font, design, etc… and be activated by the prefabs called "Info Panel", showing info ingame on certain parts of the scene without pausing the game, in an easy way.

-Inventory Panels Menu System: used to configure different designs of the inventory menu, mostly used to show a full inventory menu or a more simplified with the inventory grid in the side of the screen, to allow to see the actual world on the scene to use inventory objects on it, similar to last resident evil games and remakes.

-Player Action System: used to add and configured new actions on the character based on animations, and which can be activated by input at any moment ingame, by events, or on certain parts of the scene, like pressing a button to interact with it, sit on a chair, open a door, open a chest, pick an object, etc….It can be also used combined with the ability system, to make spell cast, like throw fire, or electricity, poison, and any type of magic or power and it can be used for other stuff, like the melee action system, reload weapons, enter/exit from vehicles, etc….

-Player Abilities System: it allows to configure any type of ability by input, allowing to create new abilities based on scripts or activate and use any of the current powers/abilities or actions in the player, like grab objects, float in the air, activate the gravity control ability, teleport, throw fire, use the grappling hook, etc… Any of these abilities can be used also separately, using its own input, instead of the default used for the ability system itself, triggered by Q by default.

-Player Input Panels: used to configure the different panels used on screen which shows the current actions input available according to the state/mode of the player, like using weapons, driving, using powers, examining an object, using the melee system, close combat, etc…

It also contains the different control schemes for the touch controls, having a list of different modes to activate different touch panels and hide others, so each player state/mode can have its own touch input panel, like a touch panel to drive, to use weapons, melee, close combat, etc… and avoid the need to have all those touch buttons always active.

-Temporal Parent For Weapons: an empty object used to place all the weapons gameObjects used on the player when he is not carrying a certain weapon on his hands or just equipped, for those weapons that are basically disabled while not used.

-Grabbed Melee Attack System: it contains the main melee combat system and its management of grabbed melee weapons (which are grabbed by the player through the grab objects system). It includes the slice mode, the damage reaction, to play animations based on the damage received by other melee weapons, and the melee weapons manager for the grabbed weapons that the character carries currently.

-Weapons Attachment UI Manager: it contains the UI elements used for the attachment system to avoid having those references configured on the attachment system on each weapon.

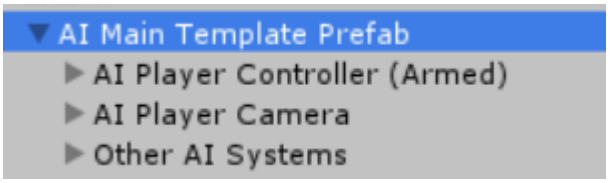
-Save Info Elements List: it contains the different components used for the save system in the game, for elements like inventory, weapons, vendor system, skills/experience, ingame settings and more, allowing to add new components easily to manage new info to save, being easy to extend this info now that before (and more improvements will be added to the main save system itself in next updates).

The rest of objects contain a single component and the name describes the tasks that each one makes.

AI SETTINGS

The main AI prefab is called AI Main Template Prefab, and it has this hierarchy, in a very similar way to the player itself.

The AI also uses the player camera gameObject, but just as a direction reference for the forward movement of the AI character towards its target on the navigation. Also, the camera component is disabled on the AI, so that won't affect the performance by any means.

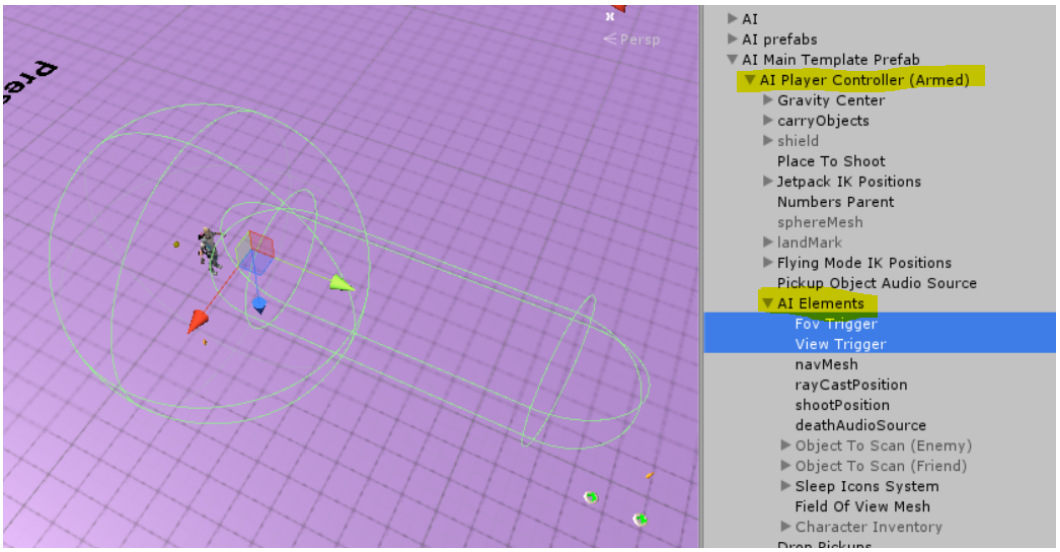


AI Target detection and management

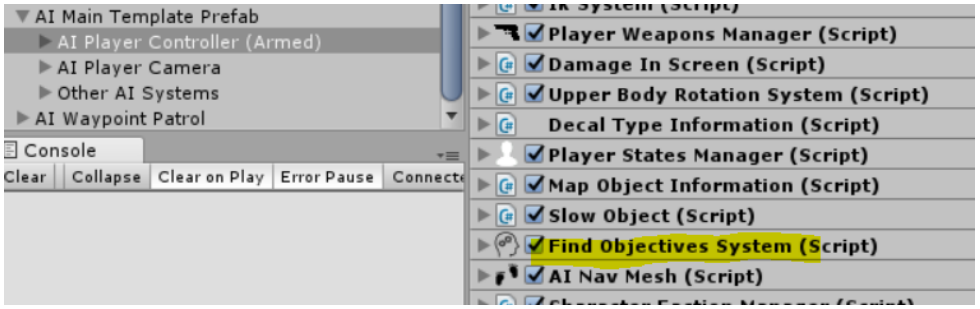
The AI uses two main elements to detect targets, a sphere trigger called "Fov Trigger" used as a range detection, storing all the possible targets inside of it and removing those which exits from it.

The other element is a capsule trigger called "View Trigger" used to detect and check possible threats using a delay for it, instead of identify the target immediately, like the sphere trigger does, so if the target remains inside the capsule trigger x amount of time, the AI will set that possible threat as a target to attack, but if the target exits from that trigger, the AI will lost track of it and return to its regular tasks.

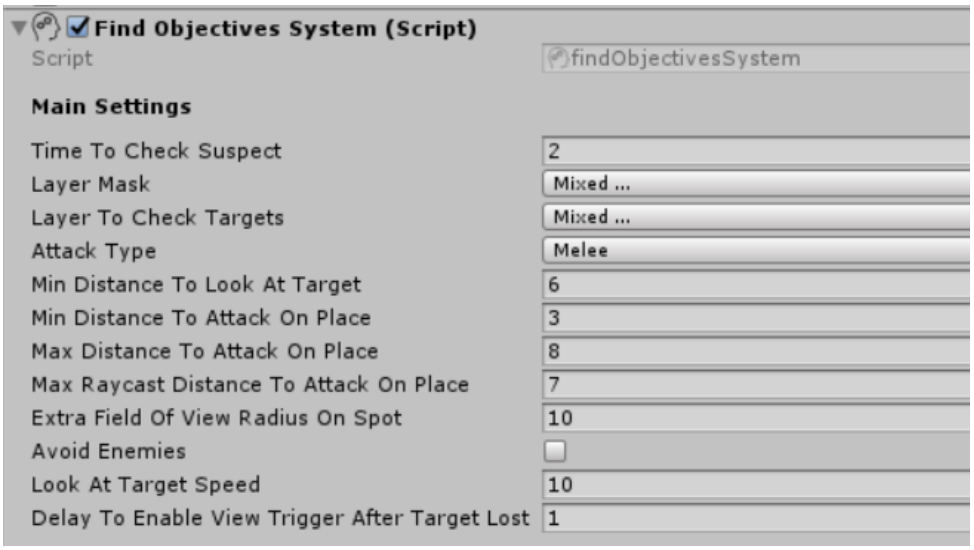
You can edit these triggers to make them bigger or use any other trigger shape instead.



These two triggers send the signal of objects detected to the component "Find Objectives System", which takes those objects and process it, to see if they are enemies or not to attack, obtaining the faction of the possible targets and checking if they are enemies or not, and activate the attack state according to that.

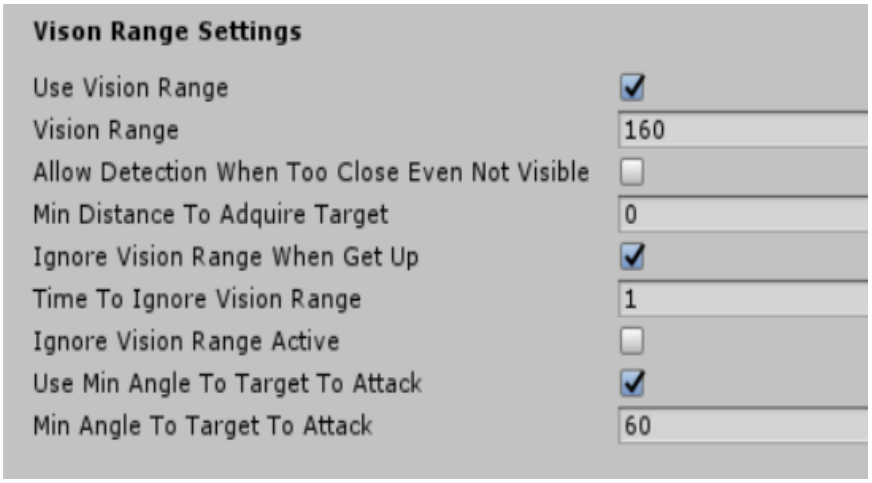


This component has different settings, including raycast to check targets, minimum distances to start to attack the current target and more.

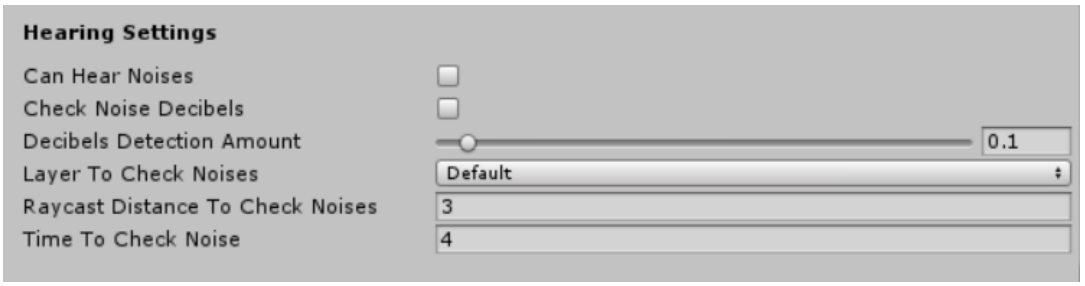


One important setting is “Avoid enemies”. This is combined with the setting “None” in the field “Attack Type”, which will make this AI to run away from targets that usually would attack, moving in the opposite direction.

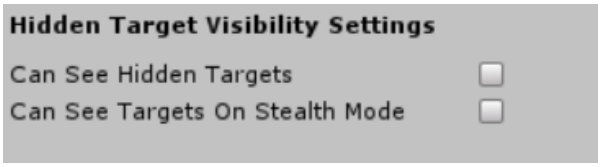
The vision range settings allows to configure the total range of view around the AI, taking the forward direction as the center and increasing that vision range to both sides. So you can customize the vision range to be for example 180 degrees, so the AI will see in forward and sides direction, but nothing besides that point, so any target behind the AI won’t be attacked until the direction of the AI or the position of that target reaches a visible range on that view.



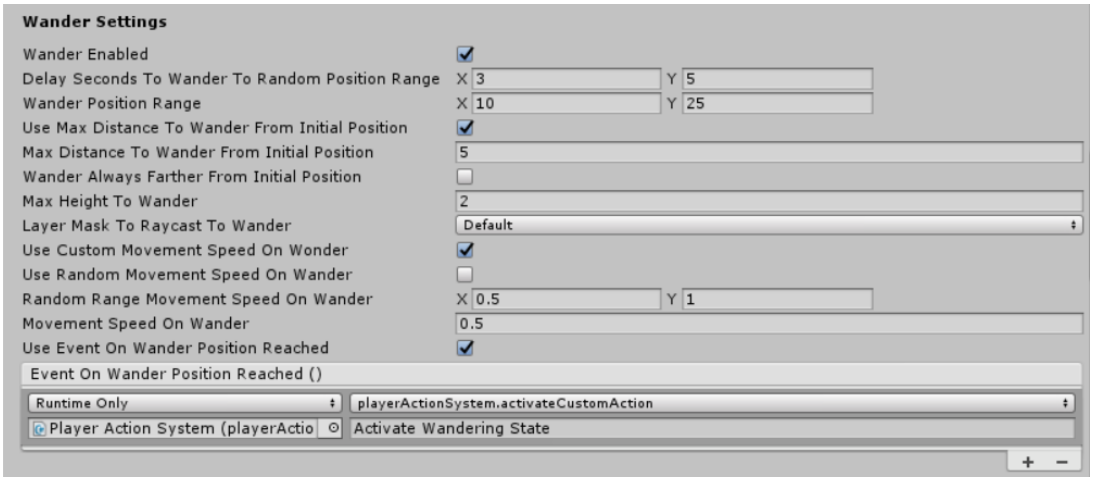
The AI can also hear noises, made from firing weapons, steps, explosions, objects dropped or thrown, etc… allowing to also detect decibels, so if a noise has a lower decibel value that the AI can detect, it won't be able to hear it. When a sound is detected, the AI will go to the position where the noise was caused, to investigate, and after x time, it will return to its previous task, according to the settings configured.



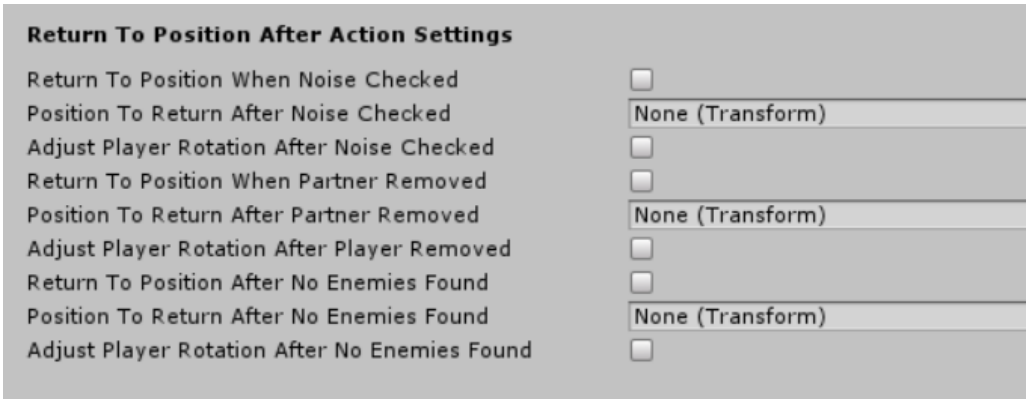
There are also options to configure if the AI can detect any hidden object on a fixed or free place (like inside a locker or in the grass crouching) and if the target is in stealth mode (through an ability for example).



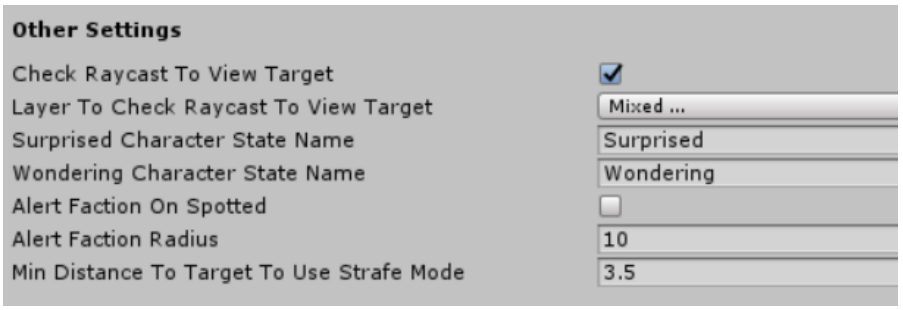
The AI can also wander around, moving to random positions inside a range and with options to avoid to get to far away or move farther in each new position, along with event options, to for example, play random animations in each "rest" delay for the AI while waits to move to the next wander position.



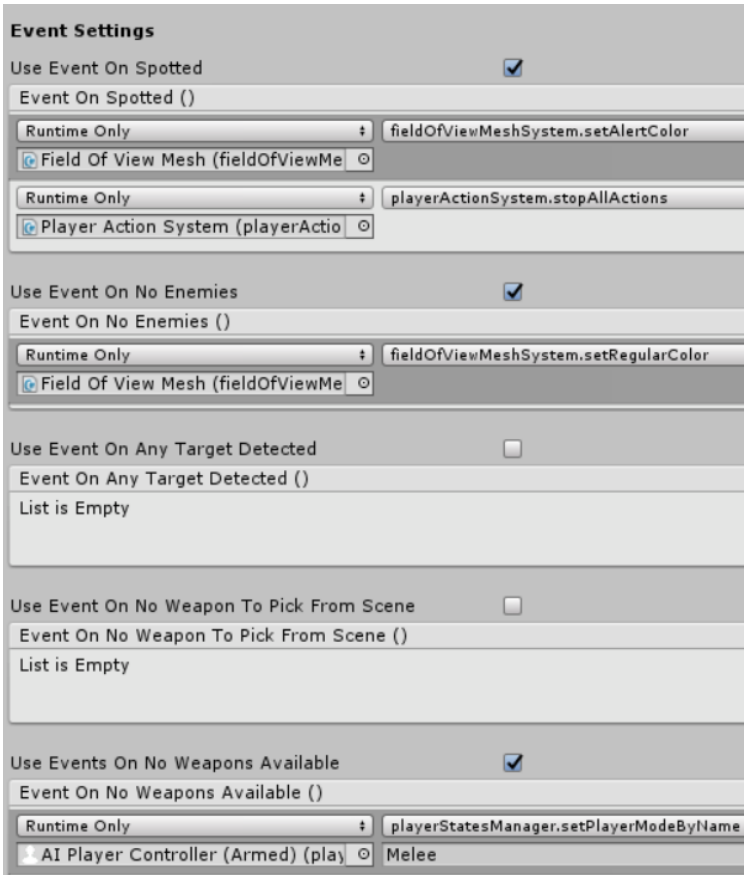
There are also options to configure if the AI returns to the original position where it was located at the start in certain occasions, like after investigating a noise, or losing track of a target, etc...



And other settings allow, for example, make the AI to alert other partners of its faction when a target to attack is located, so the rest of the AI of that faction will also attack that target.

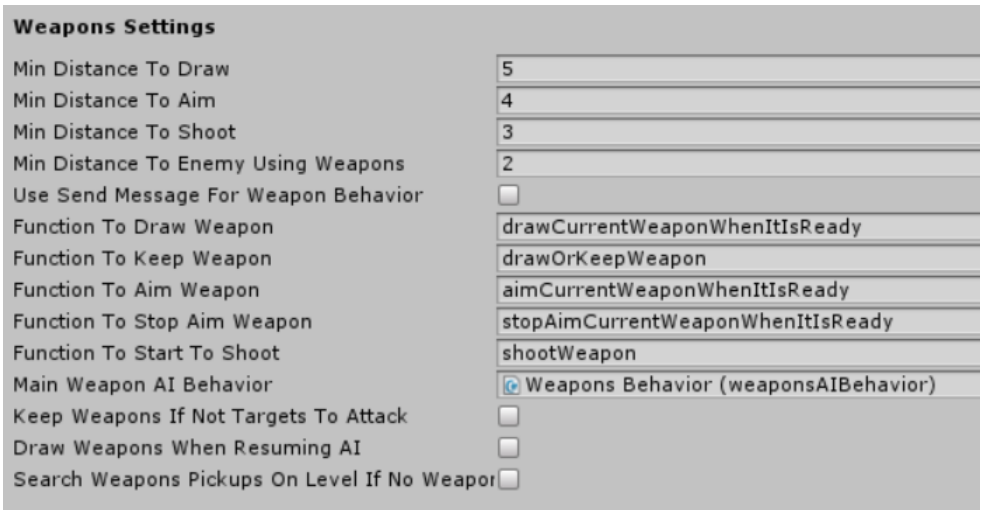


Finally, there are different event options, to be triggered on a target detection or a targe lossed and more similar situations.

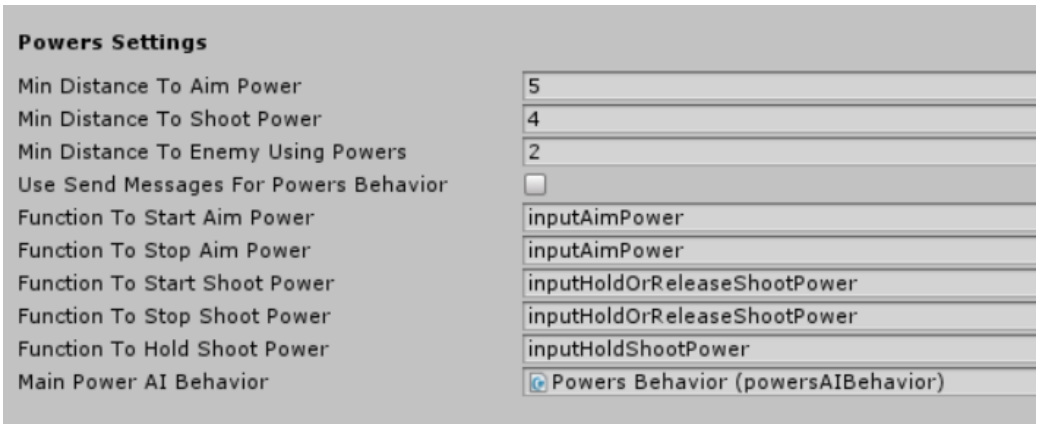


AI Attack mode

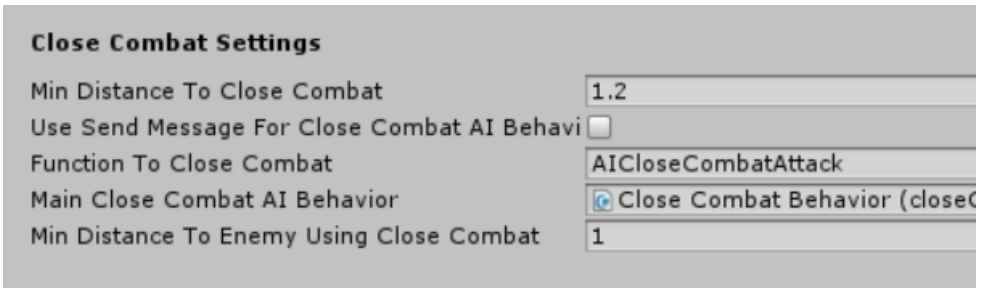
In the same component, you have the options for the different attack modes, including the fire weapon settings, for distances to start to attack:



Use powers:



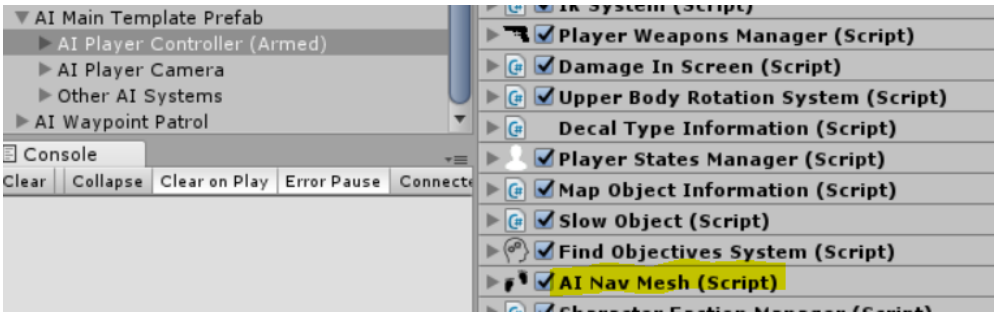
Close Combat:



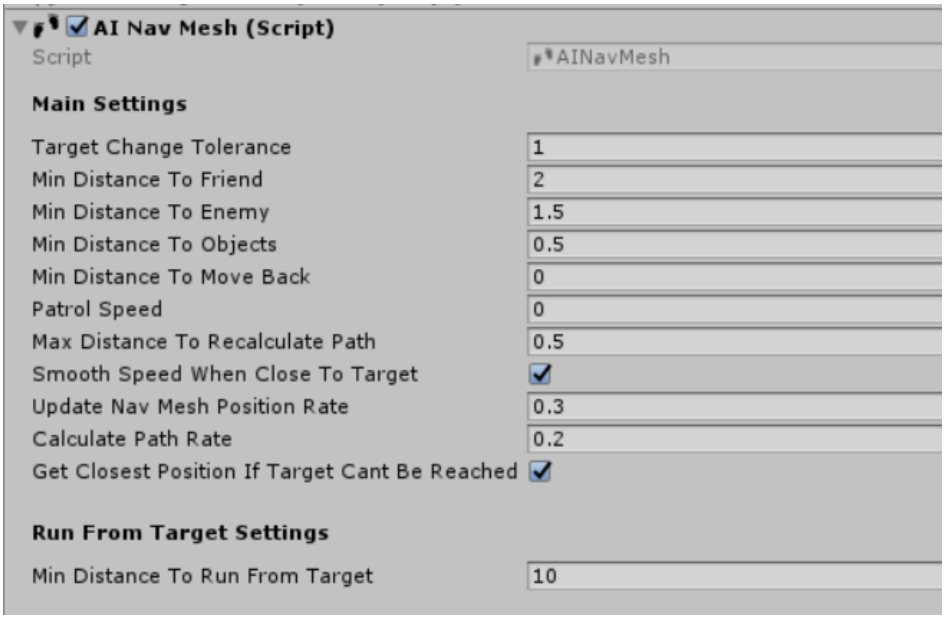
And melee system:

AI Navigation

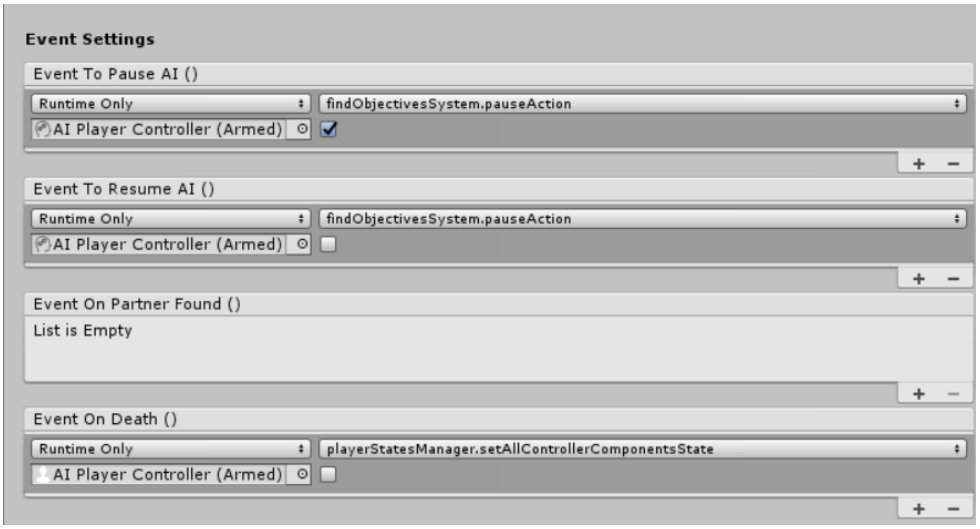
The component “AI Nav Mesh” is the one that manages the navigation of the AI through the scene using navmesh, making them to move to a certain position given a target transform to reach.



It has options to configure how close the AI will move toward each type of target, being an enemy, a friend or an object.

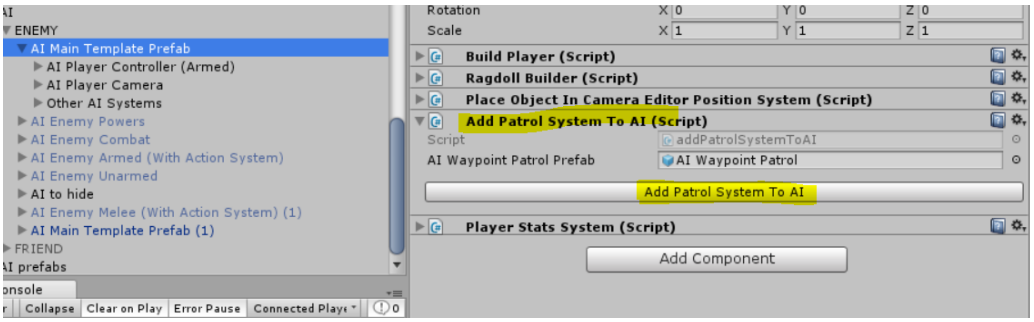


It also has events to trigger on different situations, according to what you need.

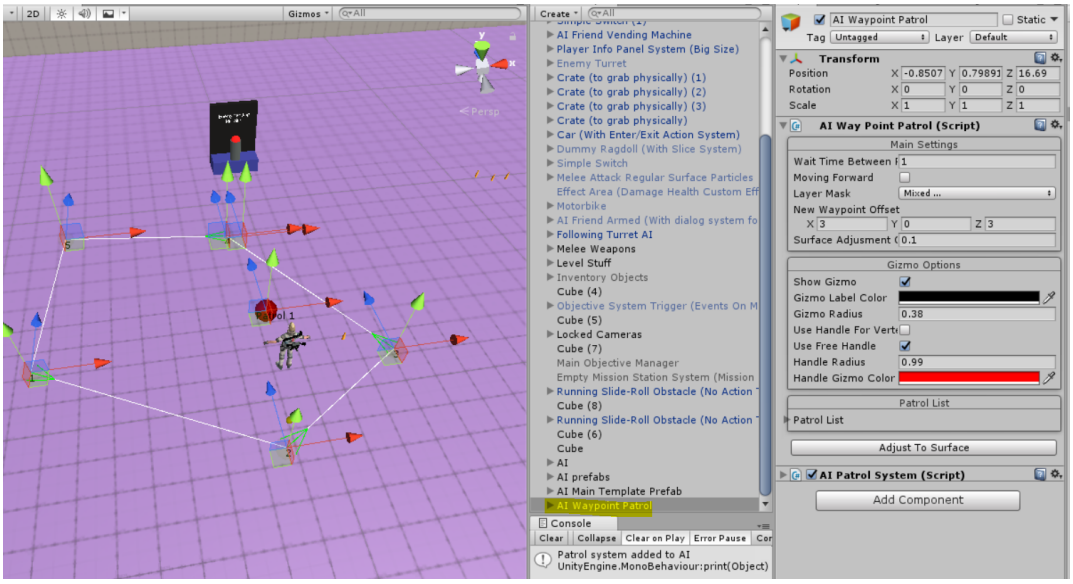


AI Patrol state

Any AI can use a patrol waypoint system to move around a path while searching for possible targets. For this, go to the main AI prefab parent and in the component “Add Patrol System To AI”, press the button “Add Patrol System To AI”.

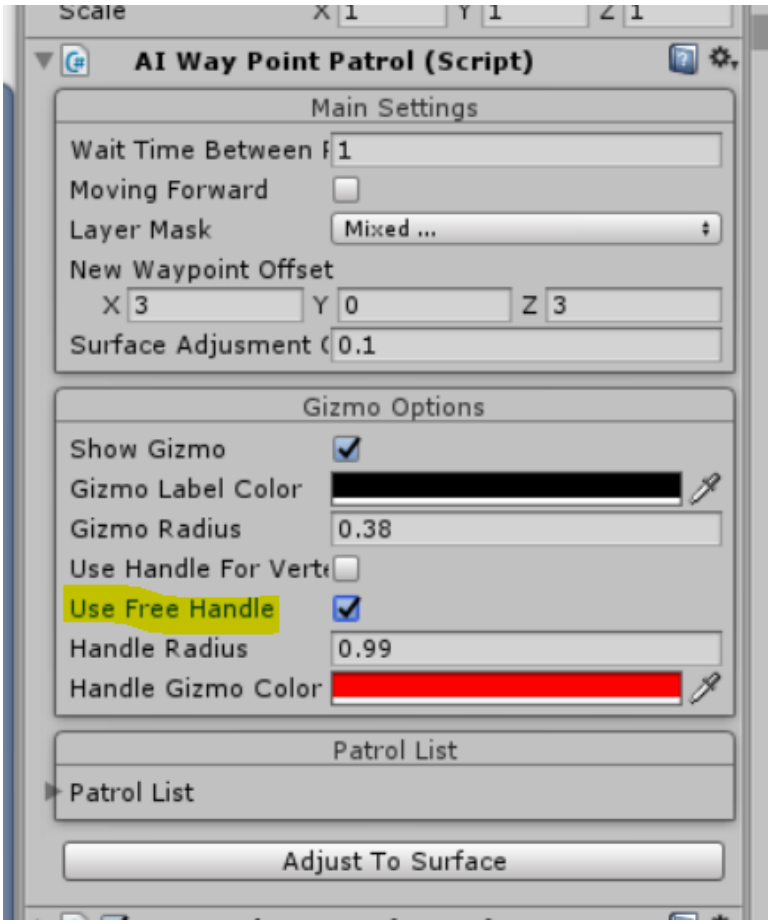


A new object will be added to the scene, called “AI Waypoint Patrol”:

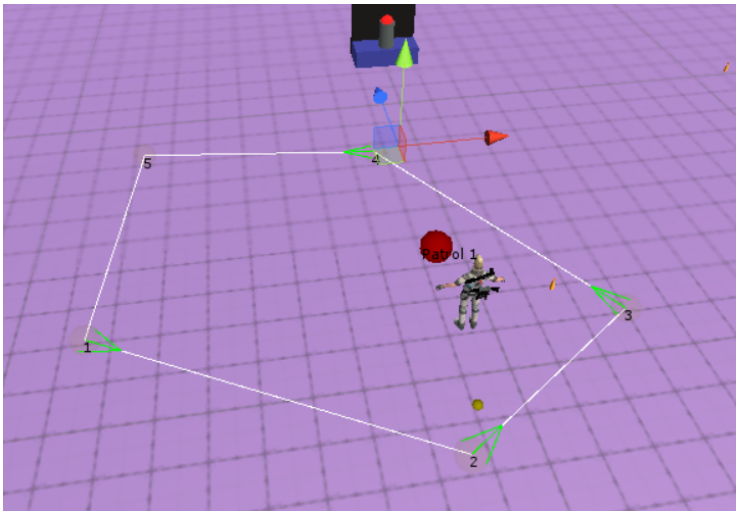


By default, it has a few waypoints configured and you can see that transform gizmos are shown in each waypoint, which is made to allow to move each waypoint separately without the need to select one by one and move them.

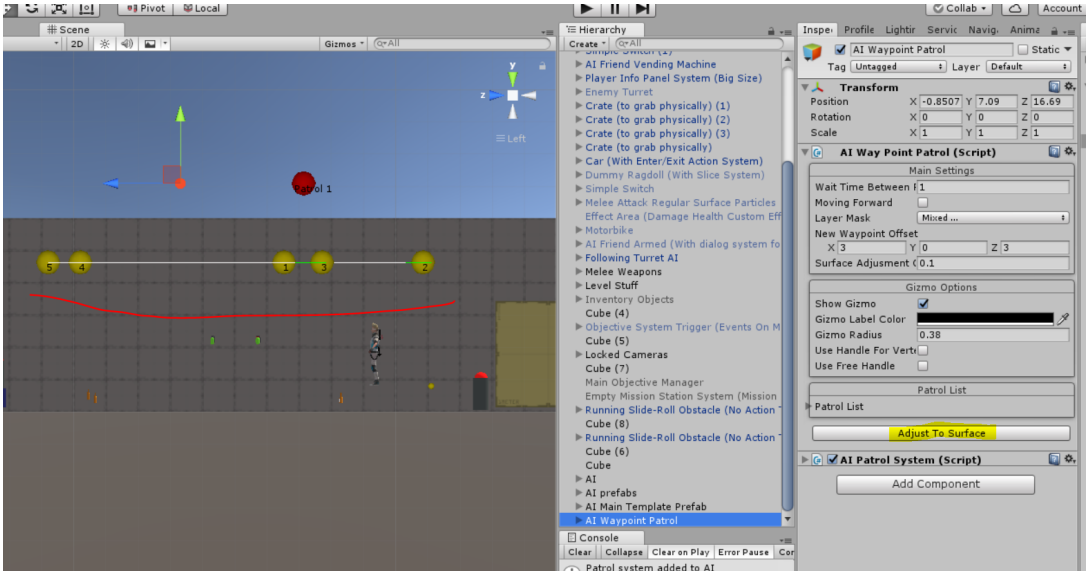
You can disable this gizmo on the option “Use Free Handle” in the component “AI Waypoint Patrol”:



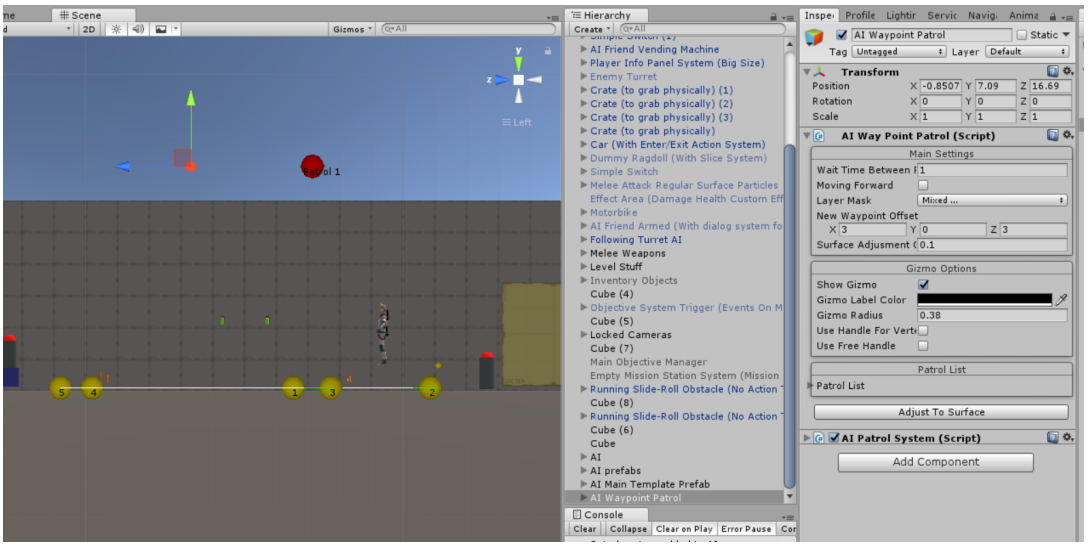
So the waypoint in the scene will only show the path itself and its direction.



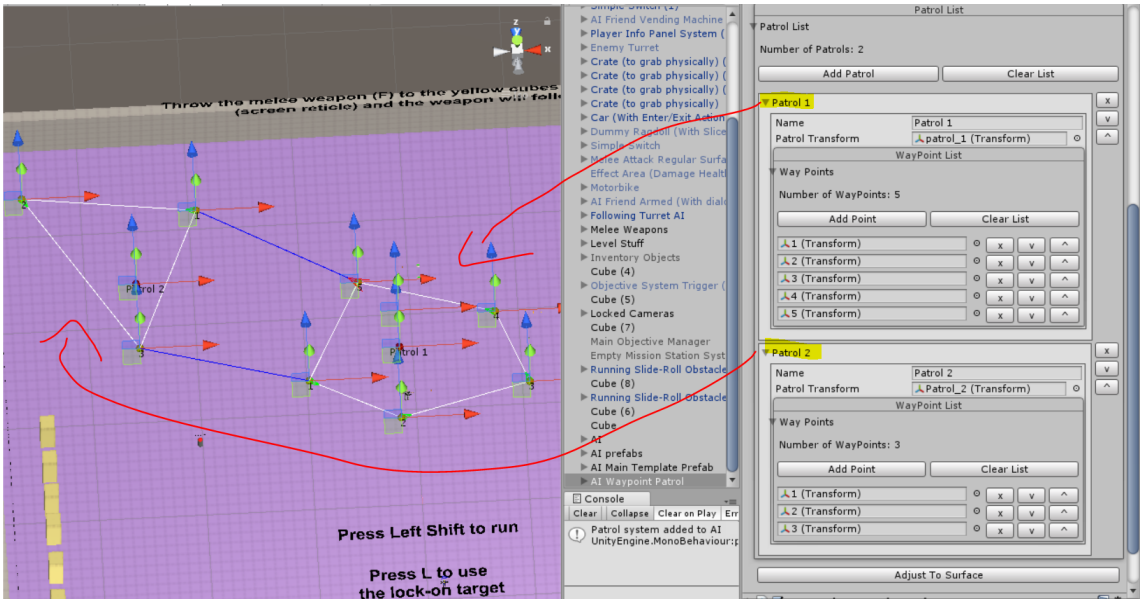
Also, it is recommended that each time that waypoints are added or placed in a new position, press the button “Adjust To Surface”:



So all the waypoints are adjusted to the ground where they are located, in order to manage the navigation values properly, looking like this after pressing the button, in this lateral view of the scene:

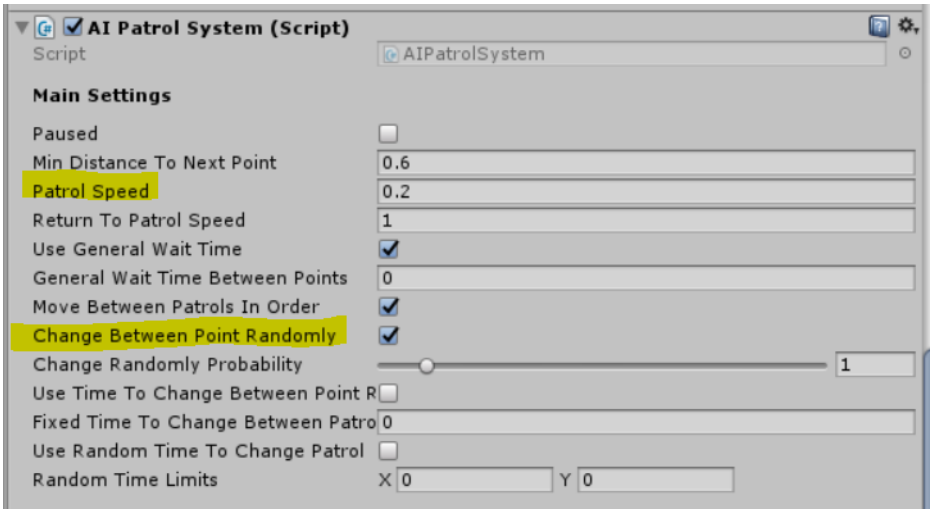


You can edit the waypoint system, to add or remove more points or new zones to walk, so the AI can have more than one zone where to move around. Of course, you can just use a single patrol and set as many points as you need.



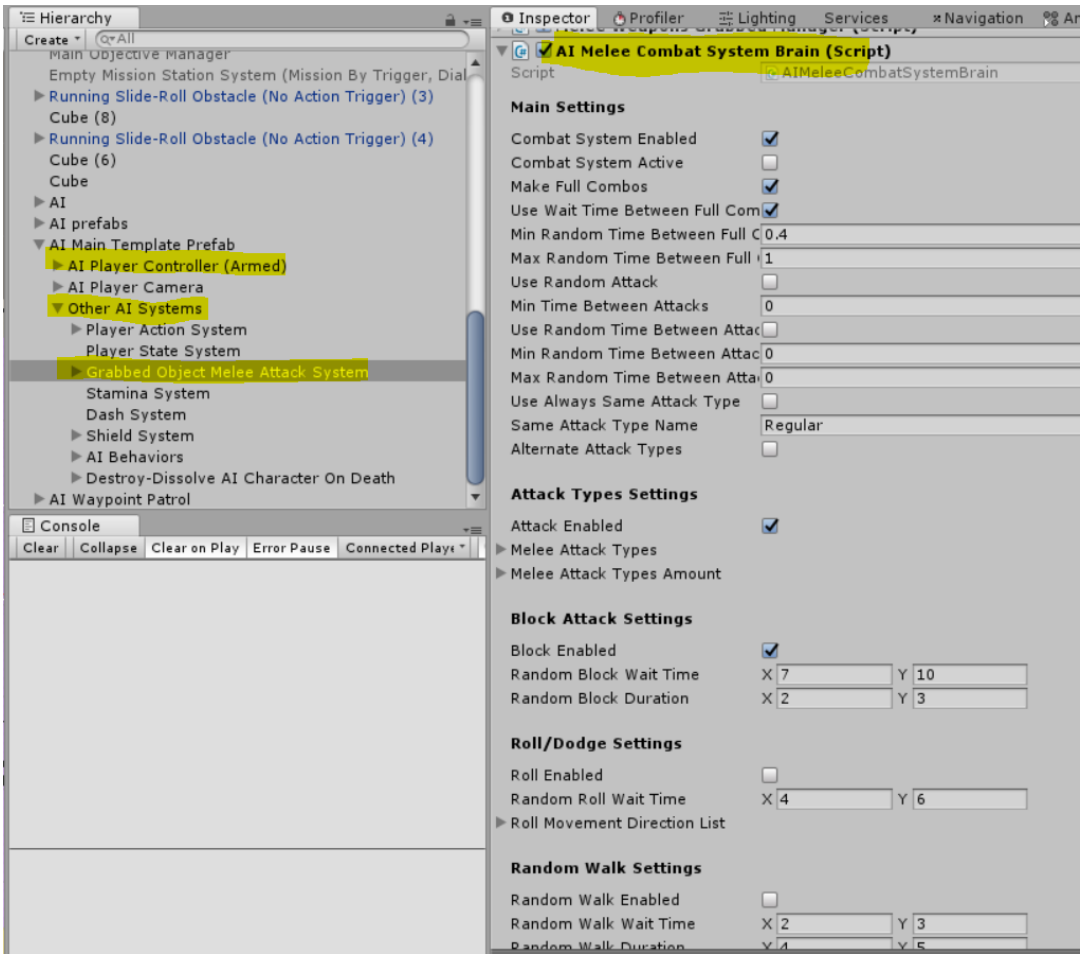
Once the waypoints are configured, go to the "AI Patrol System" component, and apply the settings that you prefer, like the speed of the AI while moving on the waypoints, the lower the value, the slower the character will move, changing from run by default to walk instead through them.

Also, you can also configure that the AI changes randomly to another waypoint, instead of moving through them in order, which is what they do by default.



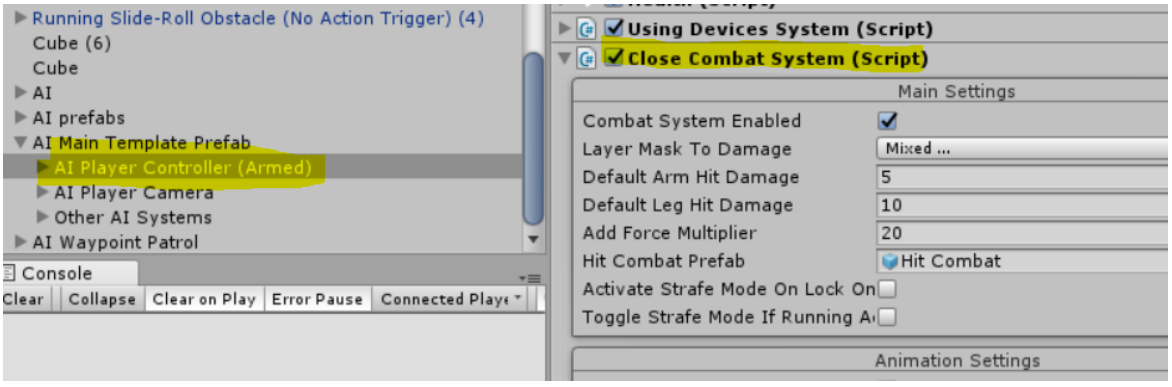
AI Melee System

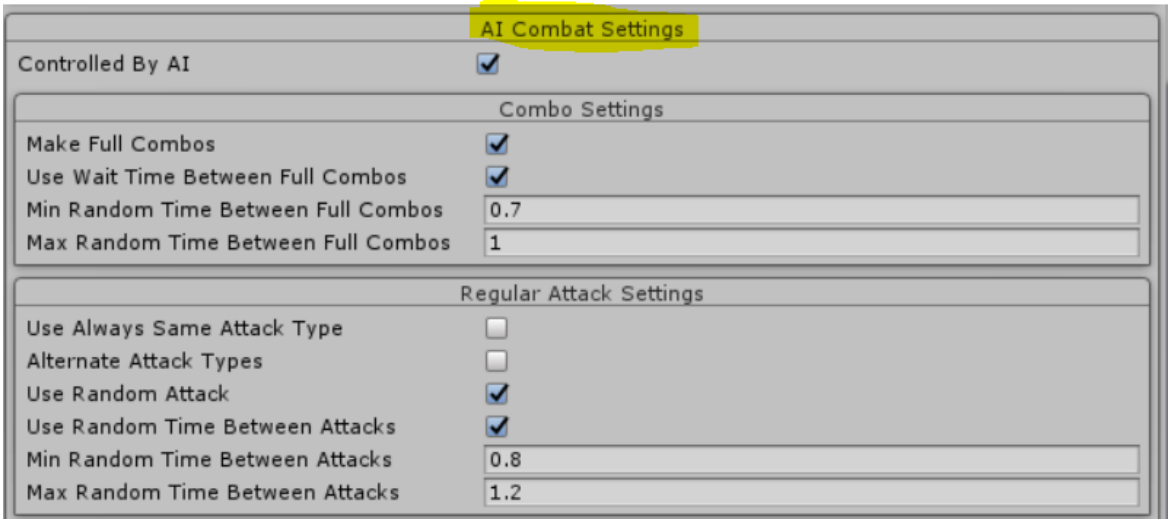
The settings to configure the different elements to attack, block, move around and other elements of the melee system on the AI are located in the component called “AI Melee Combat System Brain”, in the hierarchy shown in the below image:



AI Close Combat System

The settings to configure the different elements the attack elements of the close combat system on the AI are located in the component called “Close Combat System”, in the hierarchy shown in the below image:




AI Combat Settings
Controlled By AI
Combo Settings
Make Full Combos
Use Wait Time Between Full Combos
Min Random Time Between Full Combos
0.7
Max Random Time Between Full Combos
1
Regular Attack Settings
Use Always Same Attack Type
Alternate Attack Types
Use Random Attack
Use Random Time Between Attacks
Min Random Time Between Attacks
0.8
Max Random Time Between Attacks
1.2

EXTERNAL FILES

Animations examples for the action system and other elements in the asset

The action system can be used with any type of humanoid animation. To work on this system, example animations from mixamo were used as placeholders to configure multiple examples of the action system. Due to mixamo animations can't be included on asset, you can find the animations used on these examples on the public repository of Game Kit Controller on github:

https://github.com/sr3888/GKC-Public-Repository

These animations are already with the regular settings applied to work on root motion properly (like rotation based on current pose, loop time, adjust speed, etc…). No extra setting is needed to try these examples.

All those animations and copyrights belong to mixamo. You can get those animations and more at the official website of mixamo:

https://www.mixamo.com/#/

Of course, on this action system, you can use your own animations, from the asset store or any other store or web.

Slice systems used on some elements of the asset

There are a couple of systems in the asset that uses a slice system for 3d meshes and skinned mesh renders, for the melee weapon system, allowing to cut surfaces and characters, similar to games like metal gear rising. Those systems can be found in the public repository of GKC, which is the same limb from a few lines above.

You only need to import those 2 packages and open the script file called sliceSystemUtils.cs and uncomment all the commented lines on that script, in order to use the slice system. This is not mandatory to be done if you won't use the slice system in any part of your project.

POSSIBLE ISSUES AND FIXES

Change from .NET 3.5 to 4.0 in Unity 2019

If you are using unity 2019, maybe you can see this message on the console:

This issue is happening to most assets which haven't used 2019 as starting version, so when importing the asset, the default configuration of .NET is 3.5 when Unity needs 4.0 to work properly, causing these errors on screen in any project/asset from a lower version.

To fix this, follow these steps: Go to Edit --> Project Settings --> Player --> Other Settings --> Configuration --> Scripting Runtime Version --> .NET 4.x Equivalent

You can see more info in this link: https://www.corrosionhour.com/change-net-version-to-4-in-unity/

TUTORIAL VIDEOS

Here you can see the tutorial video list, every week new videos will be published:

https://www.youtube.com/watch?v=r0cKbNYUCZA&list=PLYVCbGEtbhxVjZ9C41fwTDynTpVkCP9iA

YOUTUBE CHANNEL

You can see preview videos about new progress and new features added in each new version on the channel Two Cubes Studio:

https://www.youtube.com/channel/UCs21at6NKl_ieSIbE_2unuA

SOUNDS EFFECTS

Some of the sound effects in the asset have been obtained from a free sounds effects page: http://www.freesfx.co.uk.

INTERESTING ASSETS FROM THE STORE

The models from Rutz Studios are very interesting and this artist has made the new model character that is used on the main player controller of GKC:

https://assetstore.unity.com/publishers/45909

The swim animations used on the swim system are made by US Studios, which you can find here for more animations packs and animations for swim as well:

https://assetstore.unity.com/packages/3d/animations/uss-animset-true-swimming-181530

UNITY FORUM, SUGGESTIONS, ISSUES OR PROBLEMS

Feel free to post any comments in the asset forum, including support, suggestions any anything you consider important.

Asset forum:
https://forum.unity.com/threads/game-kit-controler-1st-3rd-controller-with-weapons-vehicles-2-3-8-released-features-poll.351456/

Also, you can send me a PM there, my username is sr388:
https://forum.unity.com/members/sr388.660567/

SUPPORT

EMAIL

If you have any doubt problem or suggestions, please send me an email:

santimonti90@gmail.com

DISCORD

Community and Support: Official discord channel with a growing base of users. Get direct support in real time from the developer and helping people as well, with remote sessions:

https://discord.gg/kUpeRZ8

SOCIAL MEDIA

Twitter and Instagram, where gifs and videos with progress of the asset are posted a few times each week:

Twitter: @santimonti90

https://twitter.com/santimonti90

Instagram: @santimonti388

https://www.instagram.com/santimonti388

PATREON

If you want to give an extra support to the developer to help to keep the development of the asset, you can join the official patreon of Game Kit Controller, called **Two Cubes Studio:**

https://www.patreon.com/twocubesstudio