GREY GEAR™ SYSTEMS

FPS Integration Tool (Version 1.2)

User Manual (Edition 1)

Feel the FPS development capabilities at your fingertips!

Thanks for downloading the Grey Gear Systems FPS Integration Tool.

This allows you to implement First Person Shooter (FPS) game mechanics into your Unity project by supporting the systems and functionality automatically. Therefore allowing you to focus on developing the assets to then apply to the tool.

Carefully read this document as it is vital for using this tool properly and without issues.

Using this product requires a fundamental understanding of Unity for confusion to be avoided. If you are new, consider revising the tutorials from learn.unity.com.

Useful Links

Our Website The centre of our work.	**Youtube Tutorials** For you to Watch and Learn.
Post a Review Let us know what you think, Good or Bad.	**Quick Set-Up!** Get Demo Scenes running NOW!

Remember that updated editions of this user manual are made available via the https://greygearsystems.github.io/ website under Learning and Manual.

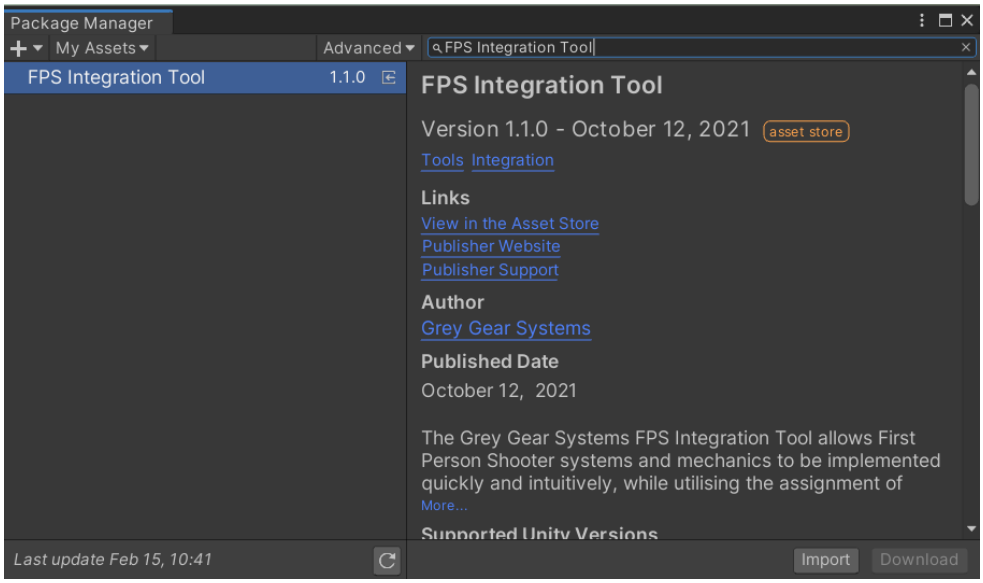
Now let's jump in…

Contents

Getting the Tool into Unity

If you have the GGS FPS Integration Tool folder in the Project tab already, you can skip this step. This is really for anyone who is viewing this PDF from the Grey Gear Systems website.

1. Firstly, open a new Unity project.
2. In Unity and from the toolbar, click Window then Package Manager.



3. From the top-left of the Package Manager tab, click the dropdown (that is sometimes labelled as Unity Registry) and select My Assets.
4. From the list on the left side of the Package Manager tab, select FPS Integration Tool.
5. Finally, click Import from the bottom-right of the tab. Then wait for the importing process to finish.

Package Layout

After importing the package into your Unity project, a folder named GGS FPS Integration Tool should exist under the Assets folder in the Project tab.

The GGS FPS Integration Tool folder contains the files used by the tool's systems.

Beware that editing the folder or file names, directories or content may impair the functionality of the tool.

Avoid saving your files inside this folder, even if produced when using this tool. This allows you to delete the tool for reimporting if it malfunctions.

The contents of the GGS FPS Integration Tool folder and subfolders involve:

Ammos - Ammo ScriptableObjects involved in weapons.

Animations - Animations used in Mouse, Movement Influence Layers, NonPlayer Zombies and so on.

Audio - Audio files used in weapons, movement and NonPlayer Zombies.

Code - Script files used in the tool. The Script subfolder contains the MonoBehaviour scripts that can be applied to GameObjects.

Default Scene - A very basic template scene that can be duplicated and it contains the main features of the tool already prepared. Review the Creating a New Scene section for more details.

Default Weapon - Used by the tool to generate template weapons which can be customised.

Demo Weapon Collection - The WeaponCollection ScriptableObject utilised in the included scenes.

Demos - Contains the demonstration scenes and their dependencies folders. They must be properly configured before running. For more information review the Setting-Up the Project to Play Demo Scenes section.

Editor Icons - Icons applied to ScriptableObjects.

Materials - Materials for the included weapons and other assets.

Menu Prefabs - Prefabs that are often used within the Inspector tab fields.

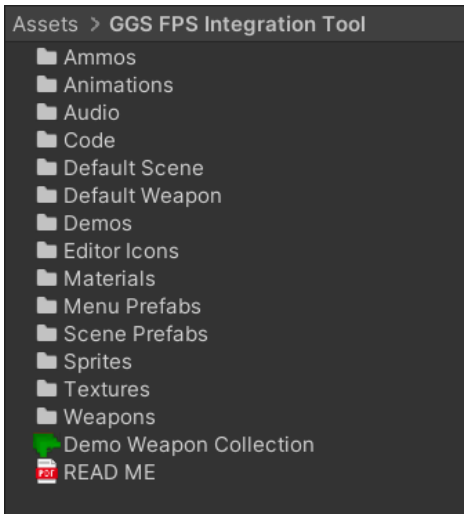
READ ME - This document.

Scene Prefabs - Prefabs usually used as a feature of a scene.

Sprites - Sprite images involved in the Heads-Up Display (HUD) within the Canvas GameObject.

Textures - Holds texture files for Materials.

Weapons - The included weapons used in the demonstration scenes which can also be used in your own scenes.



Step-By-Step Guides

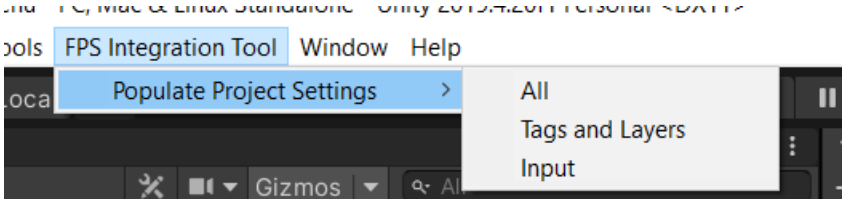
Although these guides are sectioned, you should follow each of them in their current order if you are starting from scratch.

Setting-Up the Project to Play Demo Scenes

This procedure is essential for your project and the included scenes to work correctly. As they require amendments within the Project Settings, the tool cannot work **out of the box** and ignoring this will cause errors.

Step 1

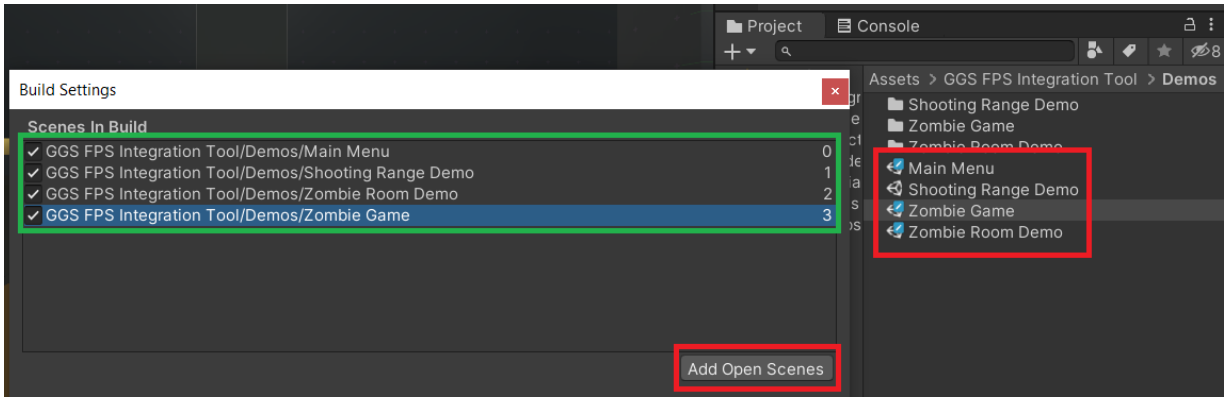
Apply the required Project Settings adjustments via the application toolbar option FPS Integration Tool > Populate Project Settings > All.



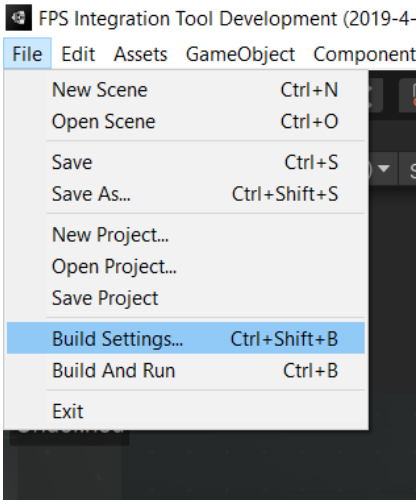
Then click Apply from the pop-up message and a notice in the Console tab that states the success of the procedure should appear.

Step 2

To allow the buttons of the Main Menu to change scenes, each level needs to be listed and ordered in a particular way within the Scenes in Build section of the Build Settings. To do this:



1. In the Project tab, access the GGS FPS Integration Tool > Demos folder to expose the four scene assets named Main Menu, Shooting Range Demo and so on.
2. Click the Main Menu scene to open it.
3. Then via the toolbar, go to File then Build Settings, which will launch a new window.



4. Under the area labelled Scenes in Build of the Build Settings window, click the Add Open Scenes button to include the currently opened scene (being the Main Menu) to the list.
5. Do the same for each of the other three levels being Shooting Range Demo, Zombie Room Demo and Zombie Game.

6. In the Scenes in Build section, the scene entries must have a ticked tickbox and be listed in the order of (each scene directory can be reordered through dragging):
    a. Main Menu
    b. Shooting Range Demo
    c. Zombie Room Demo
    d. Zombie Game

At this point, all four scenes should work without any dysfunction during Play Mode.

Demo Scene Gameplay

The scenes included in the FPS Integration Tool are designed to demonstrate the results of using this system.

They can also be used as templates, but if doing so, consider duplicating the scenes in a location outside the GGS FPS Integration Tool folder.

Upon running the scenes prior to additional customisations, the player is spawned in the environment and the HUD featuring health and ammo information is displayed. The Pistol is equipt from the start and can be swapped with other weapons once their collectable has been obtained.

Controls

The player controls set by default involves:

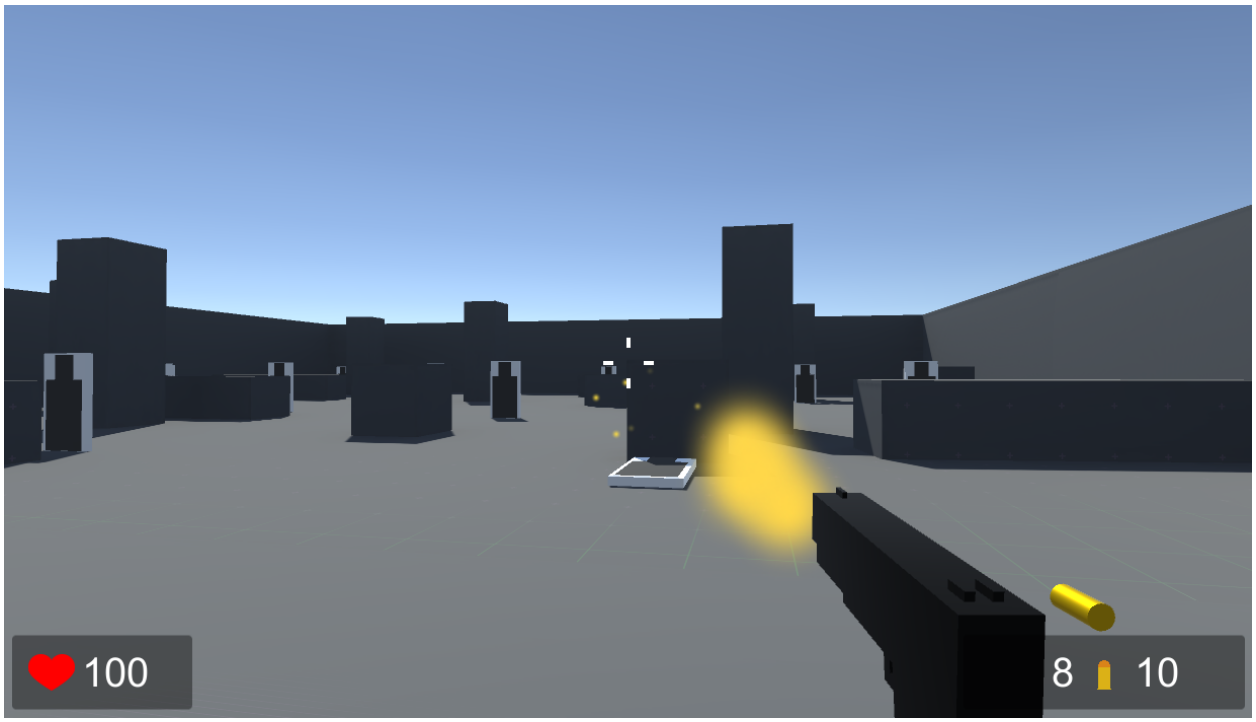
- Move Around - W, A, S, D Keys
- Jump - Space Bar
- Run - Hold Left Shift with the W Key
- Rotate Horizontally - Left & Right Mouse Movements
- Rotate Vertically - Forward & Back Mouse Movements
- Fire Weapon - Left Mouse Button
- Aim Weapon - Right Mouse Button
- Reload Weapon - R Key
- Switch Weapon - Q Key

- Toggle Pause state & Release Cursor - P or Escape Key

Shooting Range Demo

This scene features a shooting range where weapons can be easily collected from the weapon racks to be trialed on the targets, positioned at varying distances and exposure within the range itself.

As a goal, the player is required to shoot down 18 Targets with the weapons provided for the level to be completed. The progress is displayed on the top-left of the game view as part of the HUD.



Zombie Room Demo

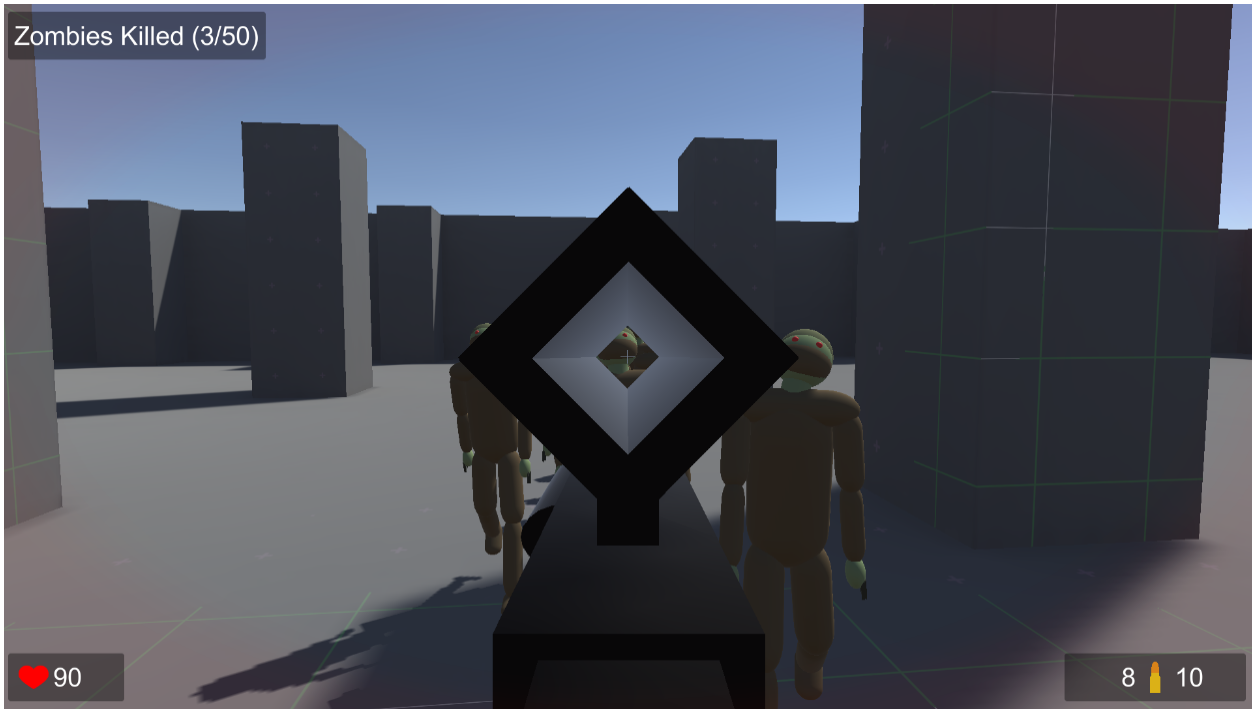
The scene consists of an uniform environment separated by pillars with 15 zombie entities and the weapons featured in the Shooting Range Demo which will spawn and respawn gradually over time.

The spawned zombies will slowly navigate to the player and will inflict damage when near.

Shooting a non-player will harm it and will die after enough damage is dealt. The weapons also conduct varying amounts of damage.

The player possesses a few lives and when killed, will respawn from one of the 4 spawn points. When killed without any lives remaining, the fail menu is displayed thus requiring the game to be restarted.

To succeed, 30 zombies have to be killed.



Zombie Game

This level simulates a wave-based zombie game mincing elements of a popular game mode featured in modern-day FPS games.

The scene features a dimly lit environment with 3 main sections that are accessed in turn, each separated by downsloping walkways sealed by doors.

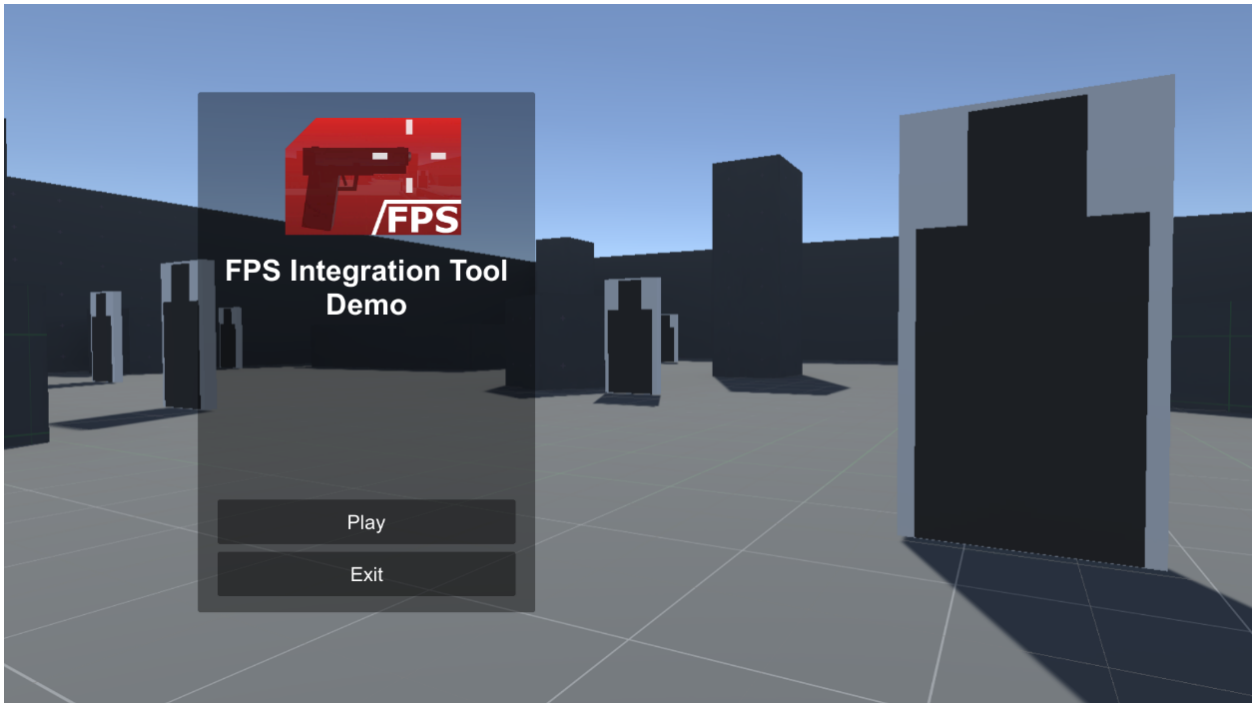
The objective is to complete each of the 5 waves, which is done so by killing all the hostilities in each and moving to the next area as required.

The waves become more difficult as the game progresses and the player has no lives thus the level is failed if killed.

Three different types of NonPlayer entities are involved being the regular zombie (from the Zombie Room Demo), the strong zombie (the harder version of the standard zombie) and the boss.

All the demo weapons are featured in this scene but their collectables are only accessible and respawn in certain areas. The health box that would heal 25 health by default upon collection also makes an appearance.

Main Menu

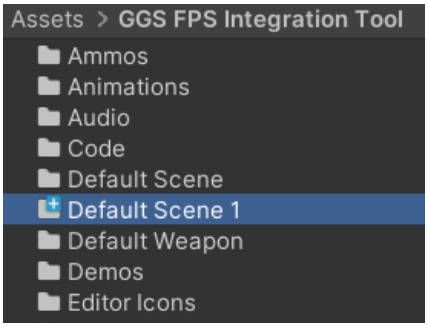


This contains the main menu system that all demo scenes are linked to and forms the supporting structure of the game, thus making it ready for exporting.

More information available in the Using the Main Menu Level Selection section.

Creating a New Scene

The easy way to create a scene with all the main features of the tool already operational is to duplicate the Default Scene from the GGS FPS Integration Tool folder, then move the new folder out into Assets and identically rename the new folder and its contents to your preference.



Step 1

Within the Project tab, go into the GGS FPS Integration Tool folder, select the Default Scene folder and click Edit > Duplicate from the toolbar. You can also use the Ctrl+D shortcut to duplicate selected objects in general.

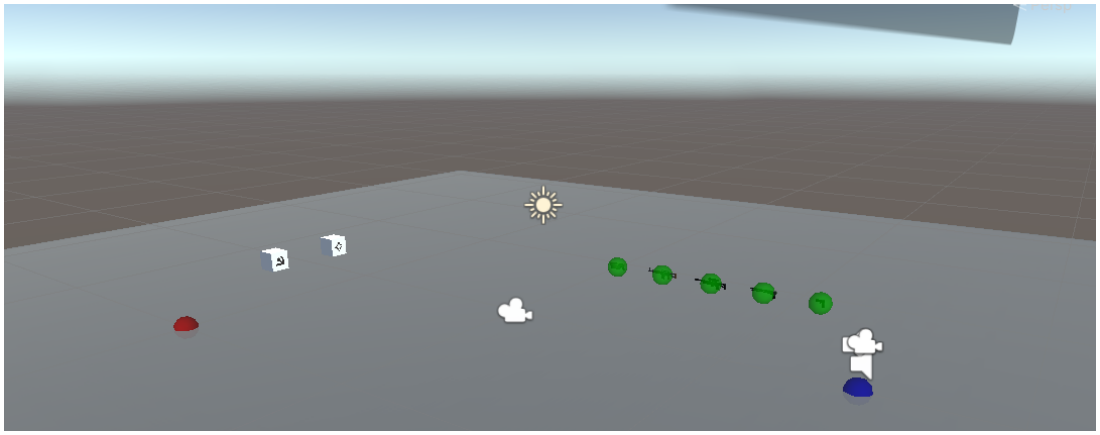
Step 2

Move the new duplicated folder (possibly named Default Scene 1) out of the GGS FPS Integration Tool folder into a preferred exterior alternative, or use the Assets instead.

Step 3

Identically rename the new default scene folder and both the subfolder and scene asset called Default Scene that are inside to something more relatable.

Opening the scene will reveal a very basic environment that includes the player, its dependencies and respawns for the player, weapon collectables and the zombie, as well as the Fail and Success Zones.



The scene can be played without any additional set-up.

Incorporating the Tool's Prefabs into a Scene

Beware that some Prefabs from the tool require setting-up if brought into a new scene. This section explains the steps needed to make such Prefabs function correctly.

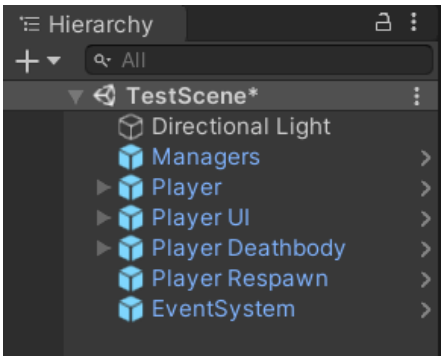
Create the Scene

1. Find a file location in the Project tab that is outside the GGS FPS Integration Tool folder.
2. Right-click in the Project tab and select Create then scene from the dropdowns.
3. Rename the new scene as you want.

Player

Step 1

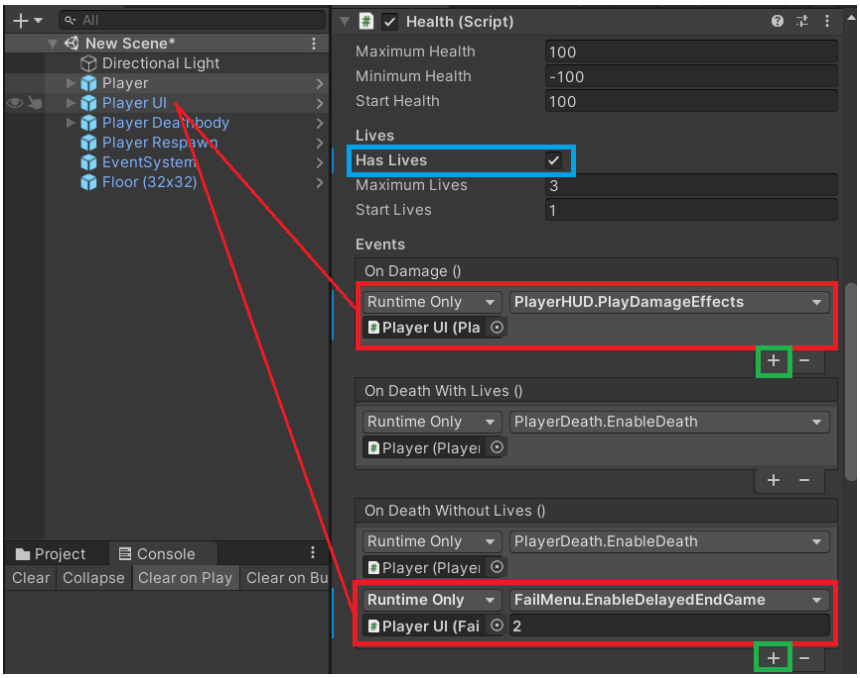
1. Open the scene and remove the Main Camera from the Hierarchy tab.
2. In the Project tab, under GGS FPS Integration Tool > Scene Prefabs, drag the Player, Player UI, Player Deathbody, Player Respawn, Managers and EventSystems Prefabs into the Scene tab.



3. Also drag in the Floor Prefab if one is not already in the scene.
4. If required, reposition the Player and the other included GameObjects so they are over the Floor GameObject in the scene.

Step 2

Observe this image as it outlines key aspects that require amending in the current and next steps.

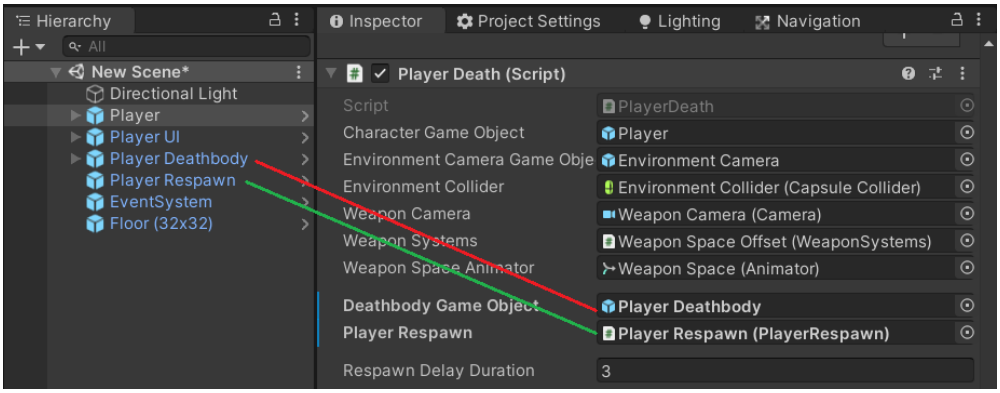


1. Select the Player GameObject from the Hierarchy tab, and in the Inspector tab, access the Health component and its Events section.
2. Within the On Damage () UnityEvents list, if it does not have any elements, click the plus icon to the bottom-right of it.
3. Drag the Player UI from the Hierarchy tab to the leftmost object field labelled None of that list element.
4. Click the No Function dropdown and hover over PlayerHUD then select PlayerDamageEffects.

Step 3

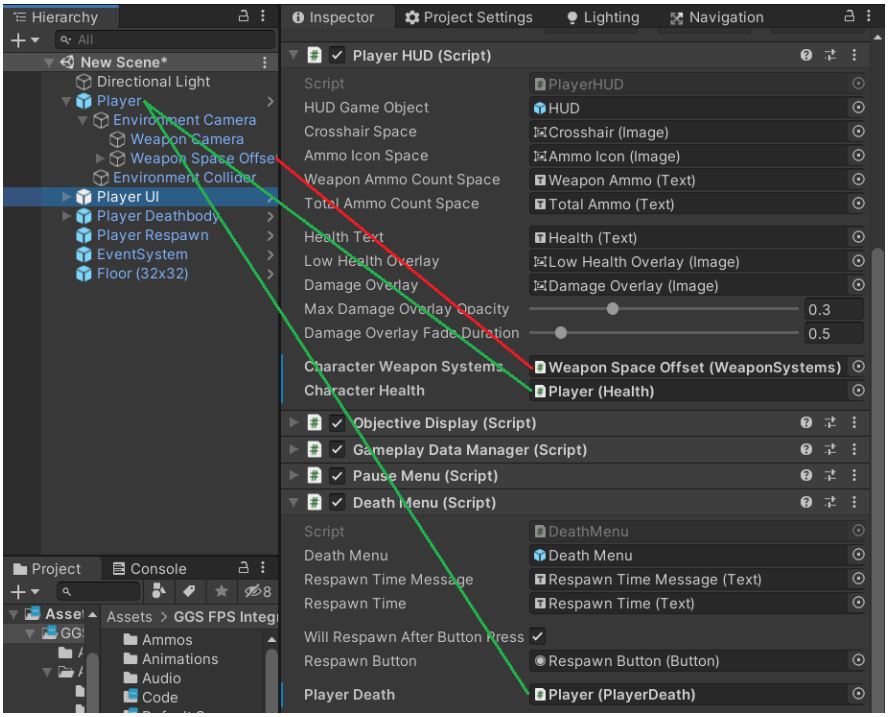
1. Now with either the On Death () or On Death Without Lives (), whichever is displayed, add a new element to it.
2. Drag the Player UI to the element's object field of the mentioned list.
3. Click the No Function dropdown, hover over FailMenu then select EnableDelayedEndGame then enter a number like 2 into the field below. EnableEndGame can be used also as it has the same behaviour apart from it involves no delay.

Step 4



1. In the Player Death component, just below the Health component, drag the Player Deathbody from the Hierarchy tab into the Deathbody GameObject field.
2. Then drag the Player Respawn GameObject to the Player Respawn field.

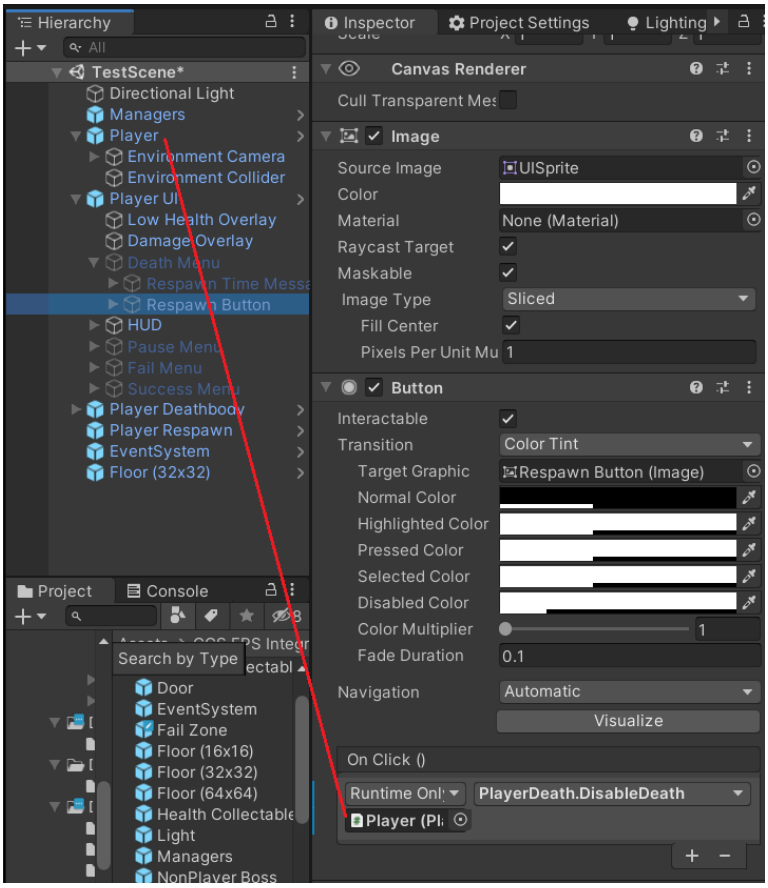
Step 5



1. Select the Player UI GameObject and navigate to the Player HUD component.

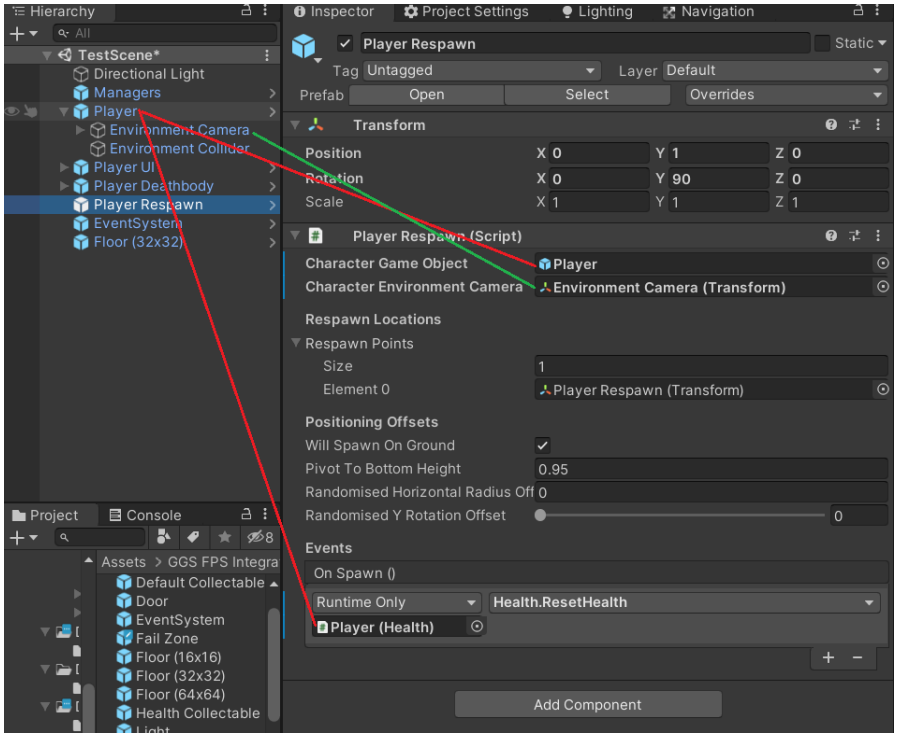
2. Expand the Player GameObject and its Environment Camera in the Hierarchy tab to expose the Weapon Space Offset.
3. Drag that Weapon Space Offset into the Character Weapon Systems field.
4. And apply the Player GameObject to the Character Health field.
5. Go to the Death Menu component within Player UI and assign the Player GameObject to the Player Death field.

Step 6



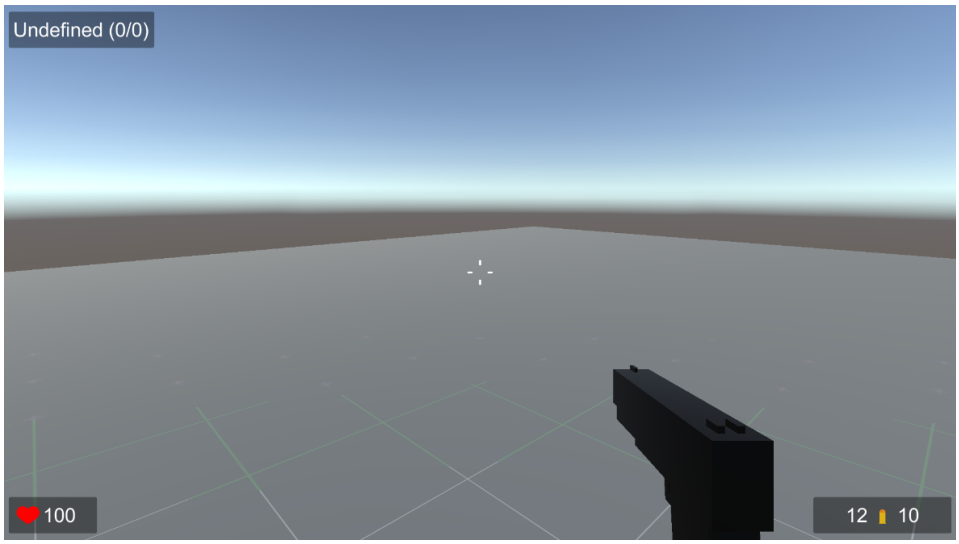
1. In the Hierarchy tab, expand Player UI and the same for its Death Menu GameObject, then select the Respawn Button.
2. In the Button component, in the On Click () UnityEvents list, create a new element if the list is empty and drag the Player GameObject into the leftmost object field.
3. Click the list element's function dropdown, hover over PlayerDeath and select DisableDeath.

Step 7



1. Select the Player Respawn GameObject and in the Player Respawn component, drag the Player GameObject to the Character GameObject field.
2. Apply the Environment Camera under the Player GameObject to the Character Environment Camera field.
3. In the On Spawn () UnityEvent list, drag the Player into the element's object field, then select the No Function dropdown, hover over Health then click ResetHealth.

At this point when running the project in Play Mode, you should have an FPS experience where the player can move around an area and fire a wielded weapon. The HUD should also be displayed and its values should update.



Auto Generate Lighting

If the scene appears dark, enable Auto Generate from the Lighting tab. Do this by navigating from the Toolbar options Window > Rendering > Lighting, and tick the box next to Auto Generate towards the bottom of the Lighting tab. After a duration, the scene should render thus lighting it up.

Please note that the lighting in the demo scenes are baked, therefore enabling Auto Generate is not required.

Collectable

Collectable Prefabs of the demo weapons can be accessed through the folders GGS FPS Integration Tool > Weapons, and in either of those subfolders, a collectable Prefab of the weapon can be dragged into the Scene tab to be included.

The Health Collectable Prefab is also available and can be found under FPS Integration Tool > Scene Prefabs.

For creating more bespoke collectables, consider the following instructions:

Step 1

Add the Collectable component to a new GameObject.

1. Create a new empty GameObject and position it at your preference.
2. Include the Collectable component by clicking Add Component then search for Collectable and select that option.

Step 2

Set-up the Collectable component:

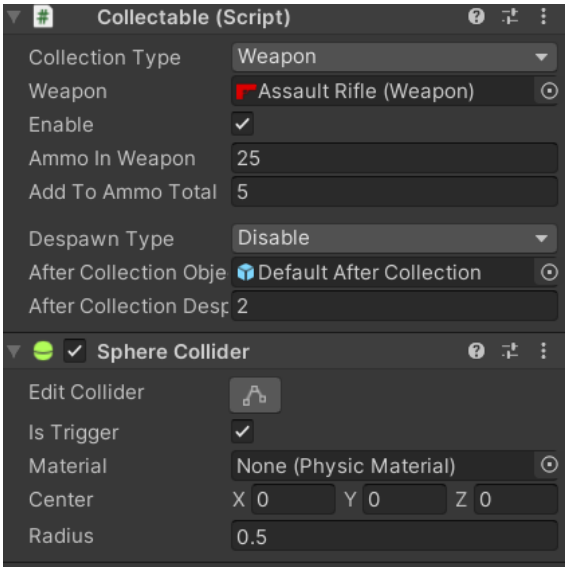
1. Choose the required Collection Type from the field out of the Weapon, Ammo, Health and Lives options. More information about this can be accessed in the Collectable section.
2. Complete the following fields that change depending on the Collection Type selected:
    - Weapon:
        i. Assign the required Weapon ScriptableObject to the Weapon field.
        ii. Tick the tickbox of the Enable field to allow use of the weapon on pick-up.
        iii. Set the Ammo In Weapon and Add To Ammo Total fields.
    - Ammo:
        i. Apply the ammo ScriptableObject to the Ammo field that you want to affect.
        ii. Set the Add To Ammo Total field.
    - Health & Lives - Edit the Health / Lives To Add fields as needed.
3. Complete the final three fields:
    - Despawn Type should be set to Disable.
    - After Collection Object field can be initialised with the Default After Collection Prefab.

- After Collection Despawn Time should be set to 2.

Step 3

Include a Sphere Collider trigger:

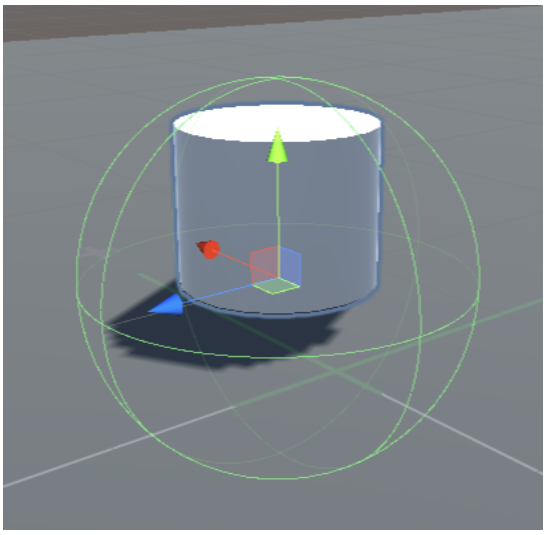
1. Apply the Sphere Collider component to the GameObject.
2. In the component, tick the Is Trigger tick box.



Step 4

Apply a model or visible asset as a child to the collectable GameObject but ensure they consist of no colliders.

Also remove any objects or components that are not needed to replicate the intended visual appearance or that might cause disturbance to the smooth running of the systems.



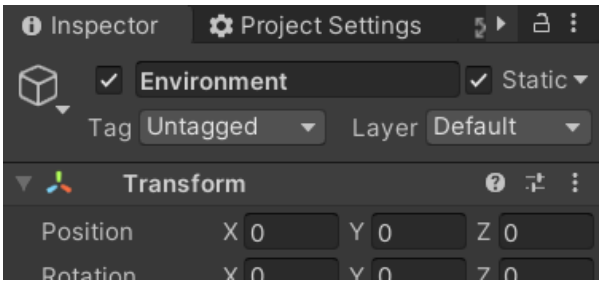
NonPlayer

The included non-player characters are entities that can be applied within the scene to behave as an enemy against the player. They are fully animated, produce audio, seeks the player to inflict damage, and can receive damage and be killed as well.

Introducing the non-player Prefabs requires these steps:

Step 1

1. For the non-player character to function appropriately with the environment, the level should feature a floor before introducing these entities.
2. Ensure that the environment's GameObjects use collider components that have their Is Trigger tickbox unticked.
3. Define all the GameObjects that make the environment and that will not move during Play Mode as Static, by selecting them and clicking the Static tickbox from the top of the Inspector tab. This is required for the AI navigation system to function.

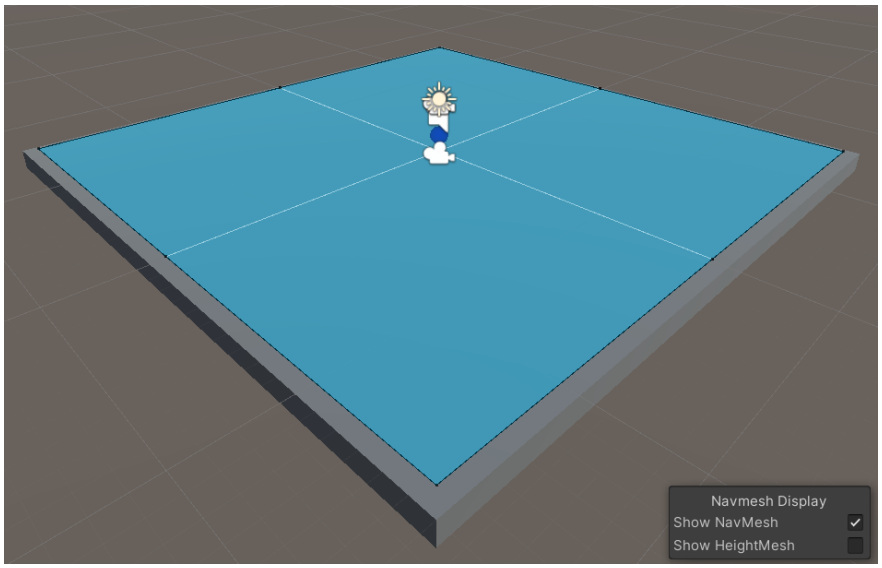


Step 2

From the Navigation tab, press the Bake button to define the area that the AI Agents can navigate to:

1. Open the Navigation tab via Window > AI > Navigation.
2. Select the Bake tab button at the top of the new Navigation tab.
3. Press the Bake button towards the bottom of the selected Bake tab.

At this point, blue surfaces should appear over the top of the static GameObjects.



Step 3

Bring a non-player character into the scene:

1. From the GGS FPS Integration Tool > Scene Prefabs folder, drag the desired NonPlayer character over the floor GameObject in the Scene tab.
2. Ensure the entity is not intersecting with any other GameObjects to prevent it from falling through the floor.

Step 4

Configure the non-player entity:

1. With the non-player character selected, access its Non Player Controller component.
2. Then drag the Player GameObject to the Target Player Transform field.
3. Apply the Player to the Target Player Death field.
4. Assign the Environment Collider under the Player gameObject to the Target Player Environment Collider.

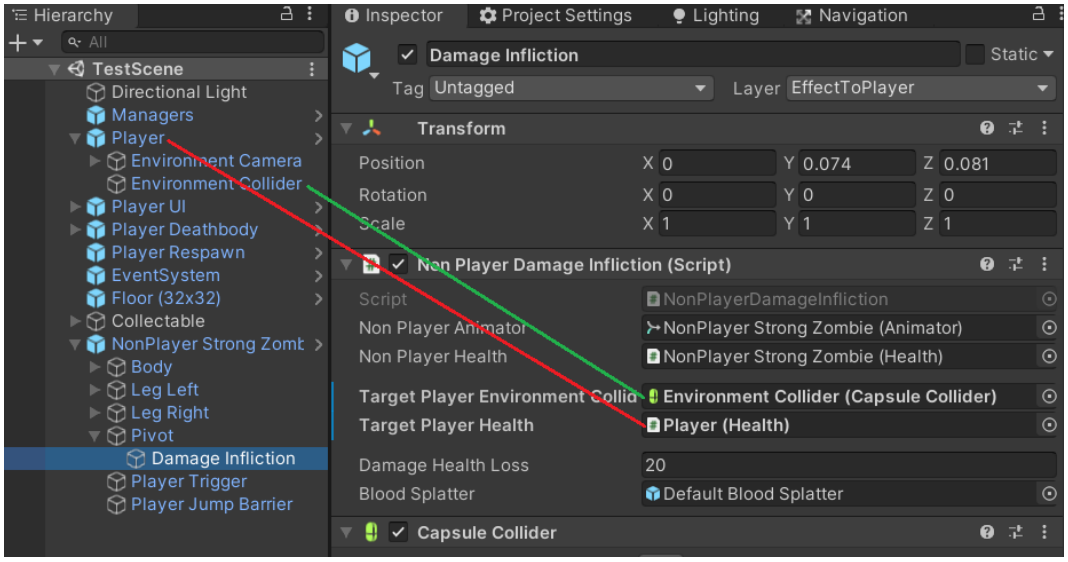
Step 5

Set-up its Non Player Damage Infliction component:

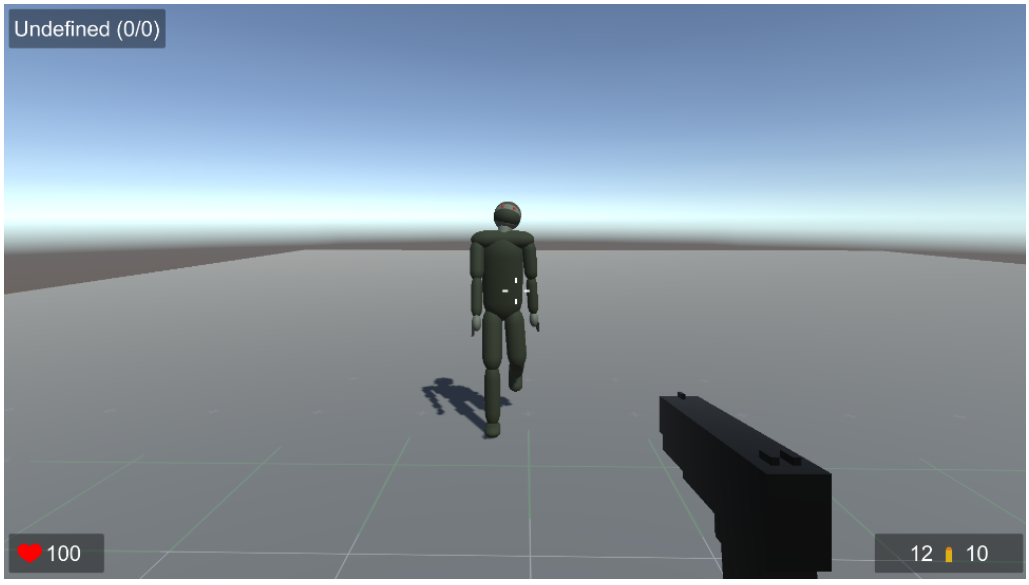
1. Via the Hierarchy tab, expand the NonPlayer character and its Pivot GameObject then select Damage Infliction.

2. Assign the Environment Collider GameObject, that is under Player, to the Target Player Environment Collider field in the Non Player Damage Infliction component.

3. Drag the Player GameObject to the Target Player Health field.



When in Play Mode, the non-player entity should navigate toward the player unless completely obstructed, while producing noises and displaying a walking animation. The entity and player should be able to harm and kill each other.



To involve additional non-player characters of the same kind, consider duplicating the one that has already been prepared, saving you from repeating the previous steps.

Respawn

There are different kinds of respawn components and Prefabs being the Player, NonPlayer and General Respawn designed for use with the player, non-player characters and collectables respectively.

Player

This tutorial can be skipped if the instructions of the Creating a New Scene have already been conducted:

1. Drag in the Player Respawn Prefab from the Scene Prefabs folder into the scene.
2. In the PlayerRespawn component, apply the Player GameObject to the Character GameObject field.
3. Then assign the Character Environment Camera field with the Player's Environment Camera GameObject.
4. In the On Spawn () UnityEvents list, add a new element if an empty one is not already available.
5. Then initialise the leftmost object field with the Player GameObject.
6. Click the functions dropdown and hover over Health and select ResetHealth.
7. Apply the Player Respawn GameObject to the required field of the PlayerDeath component of the Player.

NonPlayer

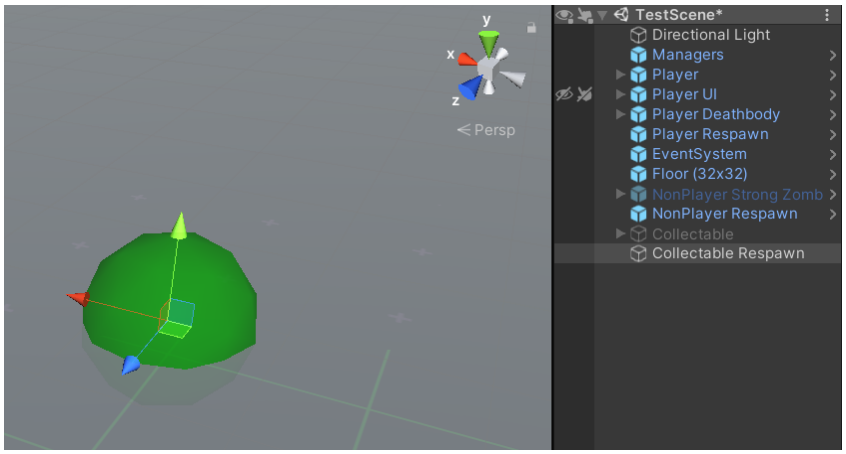
1. Drag in the NonPlayer Respawn Prefab from the Scene Prefabs folder into the scene and reposition the GameObject appropriately.
2. Within the Non Player Respawn component, expand the NonPlayer Characters foldout, change the Size field to 1 and drag the non-player GameObject from the Scene tab into the Element field.
3. To force the NonPlayer character to spawn from the respawn GameObject when starting, disable the entity and tick the Will Spawn On State tickbox.

General (Respawning a Collectable)

Although the weapon and the health collectable Prefabs are available for use and can be found under the GGS FPS Integration Tool folder then inside the Weapons and Scene Prefabs

subfolders respectively, a more bespoke alternative may be necessary and their integration is explained as follows:

1. Create a new GameObject, position it as necessary and add the General Respawn component to it.
2. Within that component, expand the Respawn Points foldout, set the Size field to 1 if needed and initialise the first Element field with the new GameObject.
3. Expand the Game Objects To Enable foldout and change the Size field to 1 also, then drag a collectable GameObject into the first Element field.
4. Tick the Will Auto Respawn tickbox and set the Auto Respawn Delay In Seconds field to around 3.
5. To make the collectable spawn from the respawn GameObject's position when starting, disable the collectable and tick the Will Spawn On Start tickbox.

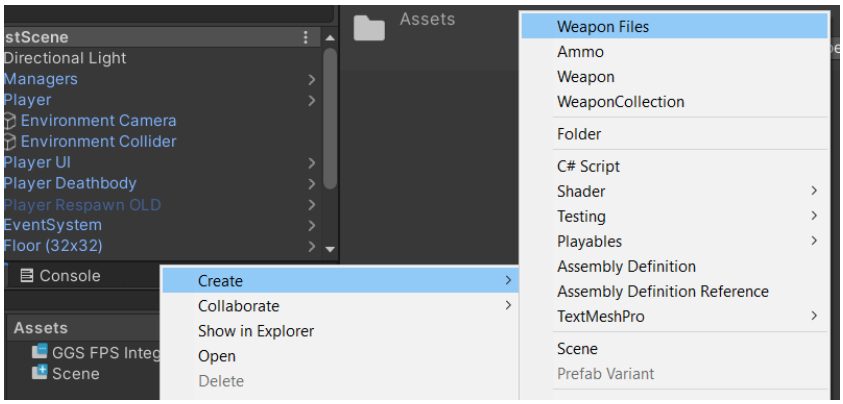


Creating a Weapon from Weapon Files

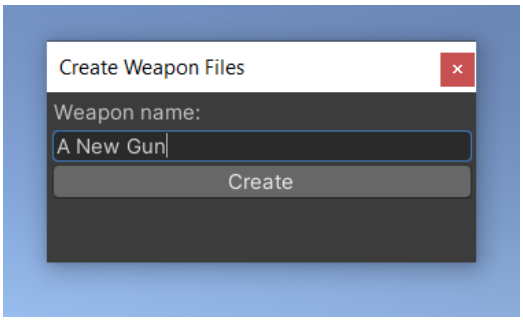
Step 1

Create a new weapon:

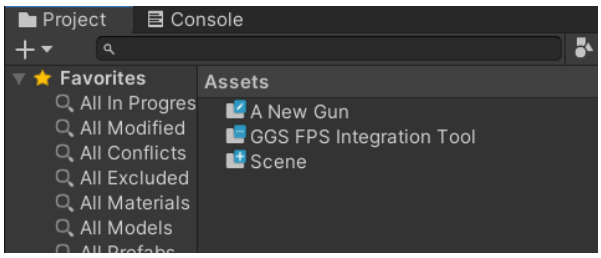
1. Right-click in the Project tab within the Assets folder and select Create > Weapon Files. For more information on the Weapon Files option, check the Weapon Files section.



2. A pop-up menu should appear within the Scene tab, then type in the new weapon's name into its field. The start and end characters of the name must not be spaces.
3. Then click the Create button.



You should have a folder in Assets with the name of your weapon.

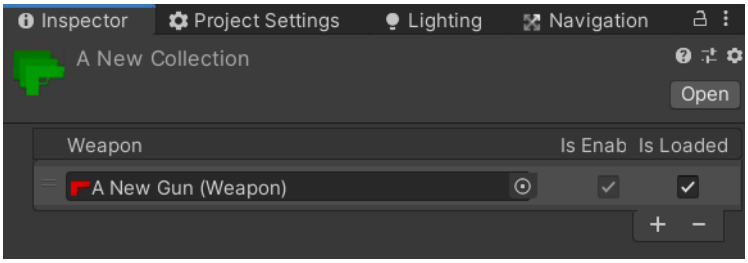


Step 2

Create a WeaponCollection ScriptableObject. More details about the Weapon Collection can be found in the WeaponCollection section.

1. Right-click in the Project tab within the Assets folder and select Create > WeaponCollection.

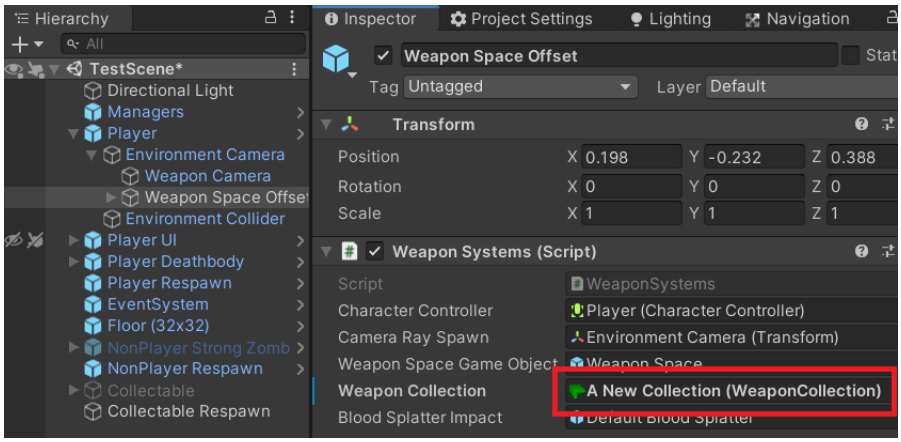
2. Rename and select the new WeaponCollection ScriptableObject to access its contents.
3. With the reorderable list, click the plus button to add a new element.
4. Apply the new weapon ScriptableObject to the left field and leave the two preceding tickboxes ticked.



Step 3

Apply the new WeaponCollection ScriptableObject to the WeaponSystems' Weapon Collection field.

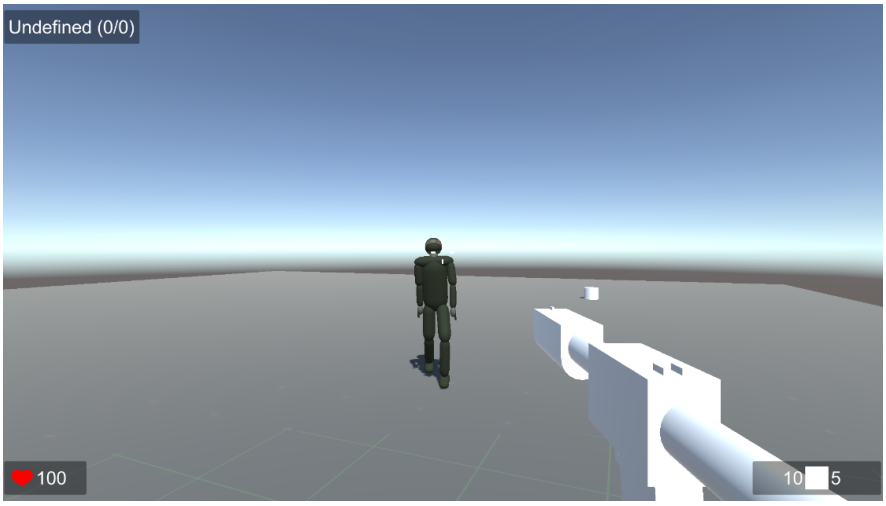
1. In the Hierarchy, expand the Player GameObject and its Environment Camera then select the Weapon Space Offset GameObject.
2. Assign the WeaponCollection to the Weapon Collection field.



Please note that any Weapon ScriptableObject that is assigned to a collectable without belonging to the WeaponCollection ScriptableObject that is currently in use will cause an error when the player attempts to pick-up the collectable.

Therefore, either replace the unlisted Weapon ScriptableObject or add the collectables' weapons to the WeaponCollection.

In Play Mode, you should have an FPS experience that features a generated copy of the Default Weapon involving standardised attributes.



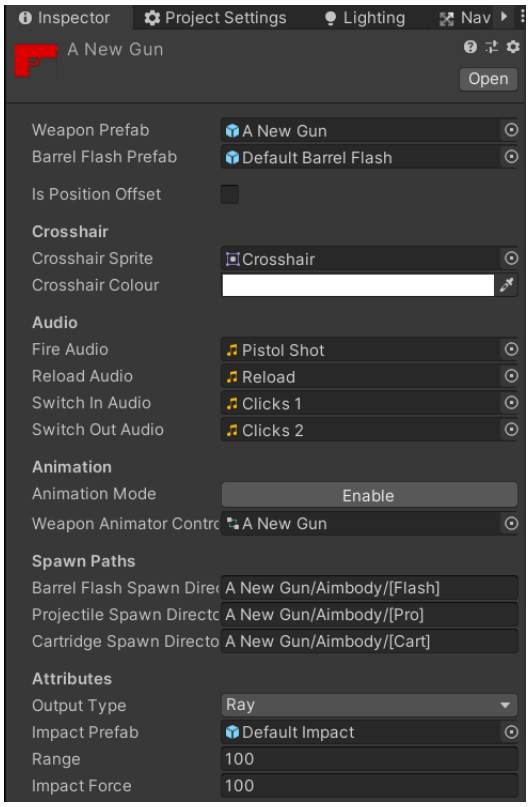
Editing Weapon Attributes

Step 1

Access the folder of your newly created weapon, and select its Weapon ScriptableObject.

Step 2

In the Inspector tab, the attributes of the ScriptableObject are displayed and can be changed to control the behaviour of the affected weapon.



For information on the Weapon ScriptableObject's fields, check the Weapon ScriptableObject section.

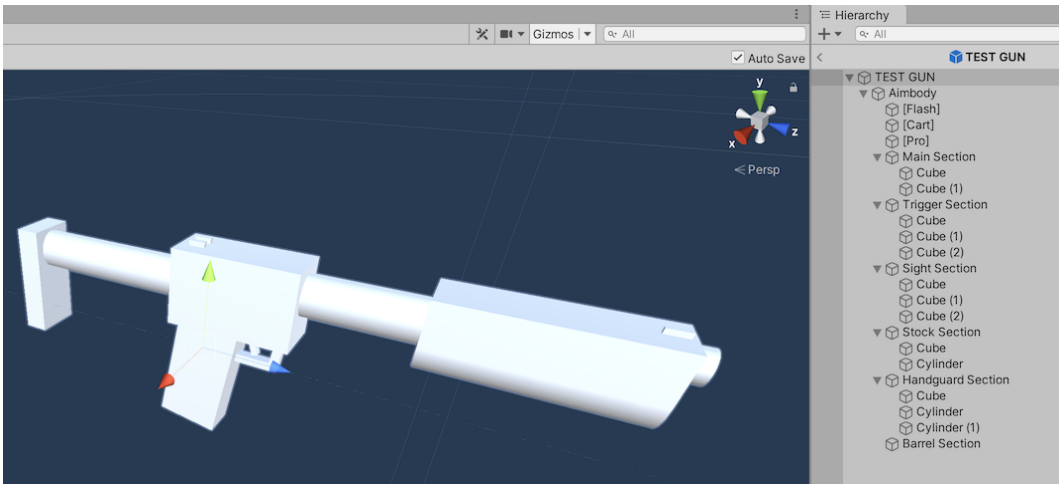
You should have a Weapon ScriptableObject with some adjusted attributes, and in Play Mode, you may notice differences in the weapon's behaviour.

Customising Weapon Prefabs

Step 1

Within the new Weapon folder, double-click the Weapon's Prefab.

Doing so will focus the Hierarchy tab and Scene tab around the Weapon Prefab, allowing you to edit it.



Step 2

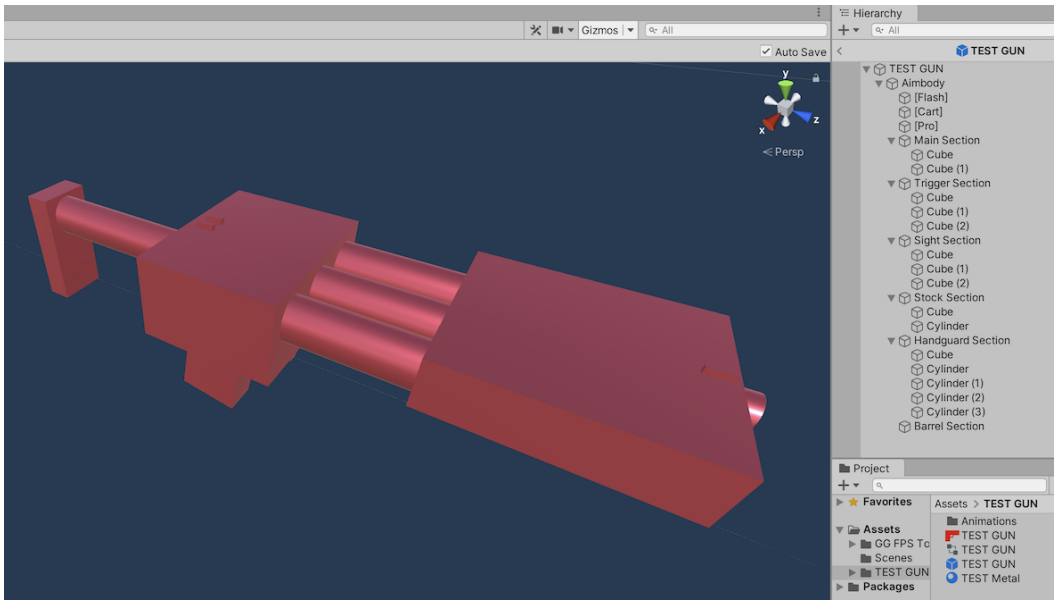
The structure of the Prefab is important to consider, the following is noted as it is in the Hierarchy tab with descriptions:

- **Weapon Name** - The Prefab itself, named after the weapon.
    - Aimbody - Needed for aiming animations.
        - [Flash] - The spawn point of the barrel flash with sound.
        - [Cart] - Spawn point of the cartridge ejection.
        - [Pro] - Spawn point of projectiles when firing.
        - Section - Grouping smaller GameObjects together into sections assists with managing the Prefab.
            - Primitive - An individual GameObject of the weapon; it can also be a mesh.
            - … - Any number of parts can be included.
        - … - Any number of sections can be used.

Anything other than the Sections and Primitives should not be edited, to allow easy implementation and smooth running of the weapon's animations and behaviour.

Step 3

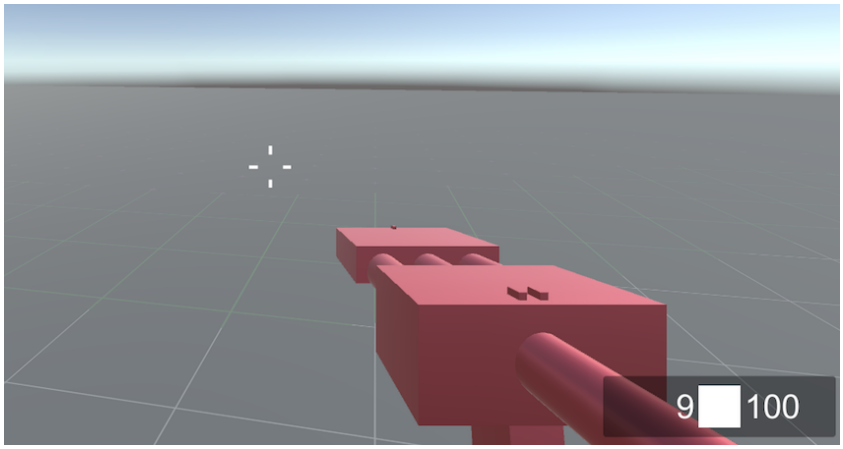
Experiment with selecting and changing the parts of the generated Weapon Prefab.



Prefab editing is automatically saved if the Auto Save box is ticked, at the top right of the Scene tab.

After customising the Weapon Prefab, run the project to view the changes to the weapon.

You now should have a weapon with an altered Prefab.



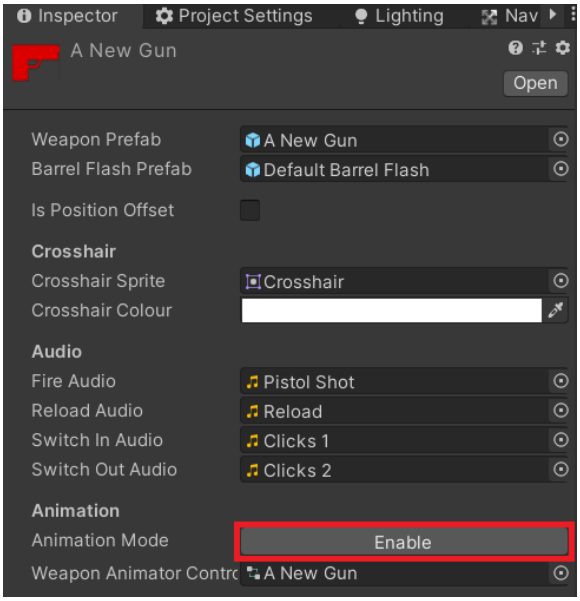
Models can also be imported from the .FBX format and be incorporated in a similar way as primitives.

Adjustments to Weapon Animations

Step 1

Prepare the Weapon Prefab for animating:

1. Select the Weapon ScriptableObject of the weapon you want to edit.
2. In the Inspector tab, click the button marked Enable that is labelled Animation Mode.

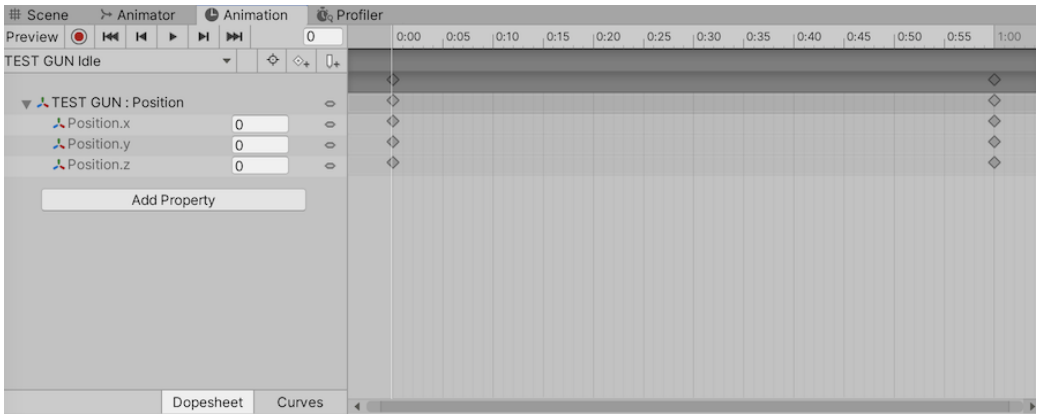


This spawns the weapon's Prefab and applies its Animator Controller to the Weapon Space GameObject.

Step 2

Prepare the animation system:

1. Open the Animation tab, which includes a timeline interface.



For more information on creating animations in Unity generally, review docs.unity3d.com/Manual/AnimationSection.html.

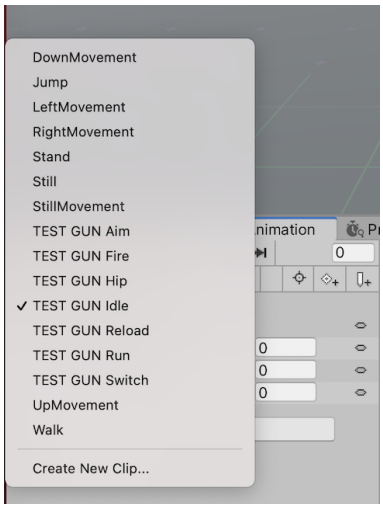
Step 3

There are important guidelines to consider when creating animations for the FPS Integration Tool and are described in the Weapon Space Animation Guidelines section.

The animations for the Weapon have already been generated through the Create > Weapon Files option.

Introducing the animations:

1. From the Hierarchy tab, select the weapon GameObject under Weapon Space. This ensures animations will be recorded correctly.
2. At the top-left of the Animation tab, click the Animation Clip name to select another from the dropdown.



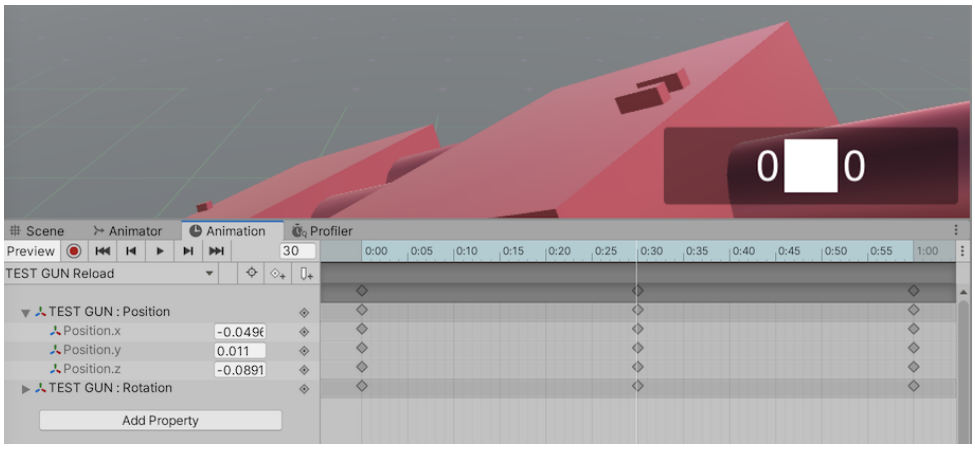
3. Click the play icon to play the looping animation. Alternatively, click (or drag across) the time bar (which displays timing numbers) to show the animation at specific times.

Step 4

Recording animations in general:

1. Utilise the [Keyframe Recording Mode] to easily produce animations through moving GameObjects via the Scene tab.

2. When adjusting GameObjects below the Weapon Space in the Hierarchy tab, the modification is recorded over the current position of the white line in the Animation tab's timeline.



3. Move this white line around the Timeline to affect different parts of the animation.

4. Once finished with editing, access the weapon's ScriptableObject and click the Animation Mode button marked Disable. This removes the weapon's GameObject and Animator Controller from the Weapon Space.

You now should have a weapon which plays adjusted animations.

Altering attributes of Non-Player Characters

When involved into the scene, non-player characters are highly customisable as appearance, animations, audio and attributes can be altered.

Remember that implementing complex model-related improvements to the non-players could cause an increase in computational overhead.

Speed

Step 1

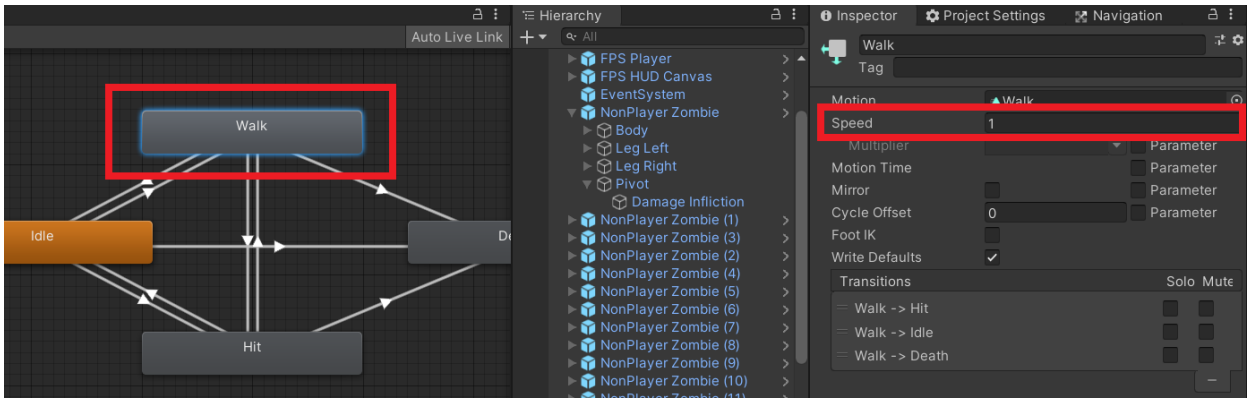
To adjust the movement speed of the NonPlayer GameObject in the scene, access the Nav Mesh Agent component and change the value of the Speed field.

Step 2

To maintain some realism, the animation speed of the NonPlayer entity has to be changed in synchronisation.

Note that changes to the NonPlayer Zombie animations and Animator Controller affects both NonPlayer Zombie and Strong Zombie.

1. In GGS FPS Integration Tool > Animations > NonPlayer Zombie or Boss, double-click the Zombie or Boss Animator Controller to view it in the Animator tab.
2. In the Animator tab, select the Walk Node, then in the Inspector tab, alter the Speed field to the same proportion as the Nav Mesh Agent component's speed.



For example, adjusting the speed from 1.2 to 2.4 in the Nav Mesh Agent means changing the Walk Node speed from 1 to 2. However, minor differentiations could be necessary to improve the accuracy of the animation.

If adjusting either the NonPlayer Zombie or Strong Zombie or just a single entity of such kind, certain animation clips and their Animator Controller would need duplicating, then reassign and alter them as necessary.

Damage To Player

To modify the damage the non-player character deals to the player, access its Non Player Damage Infliction component and edit the Damage Health Loss field to define how much health is deducted each time it attacks.

Damage From Player

This does not involve editing the non-player characters, but to control the damage dealt to the entity from the player:

- Configure the Damage Per Shot field from within the relevant Weapon ScriptableObject that uses the Output Type of Ray.
- Otherwise if using the Output Type of Projectile, alter the relevant fields of the required projectile Prefab's Projectile component.

Health

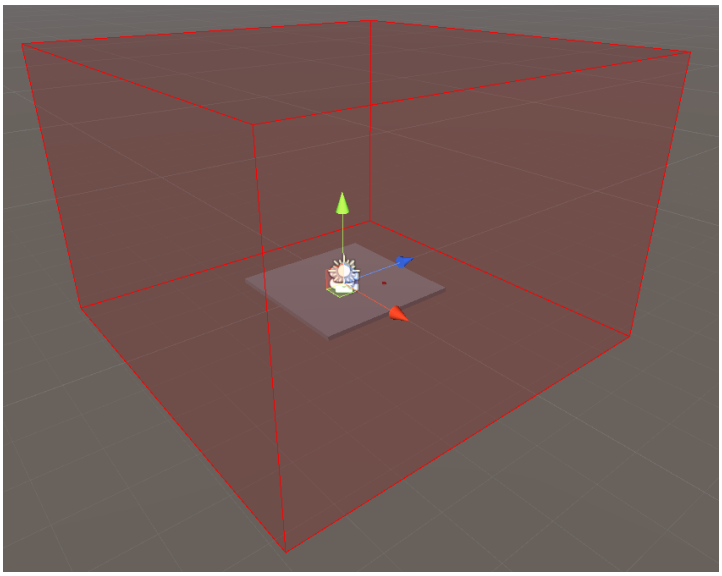
To amend the health of the non-player, access its Health component and change both the Maximum Health and Start Health fields.

Managing Boundaries in the Scene

The Managers Prefab from the Scene Prefabs folder contains a Boundary Manager component which instantly kills the player character and despawns or destroys general Rigidbody GameObjects if they pass the boundaries.

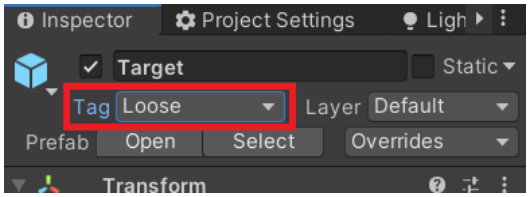
This ensures that GameObjects that have fallen off the level will not be falling forever.

The boundary is also displayed as a red box that encapsulates the playable areas of the scene.



So long as the Managers Prefab from the Scene Prefab folder is included in the scene, the boundary system should already be established.

Just remember, when making new GameObjects that use the Rigidbody component, thus could possibly fall off the level, set their tags to Loose.



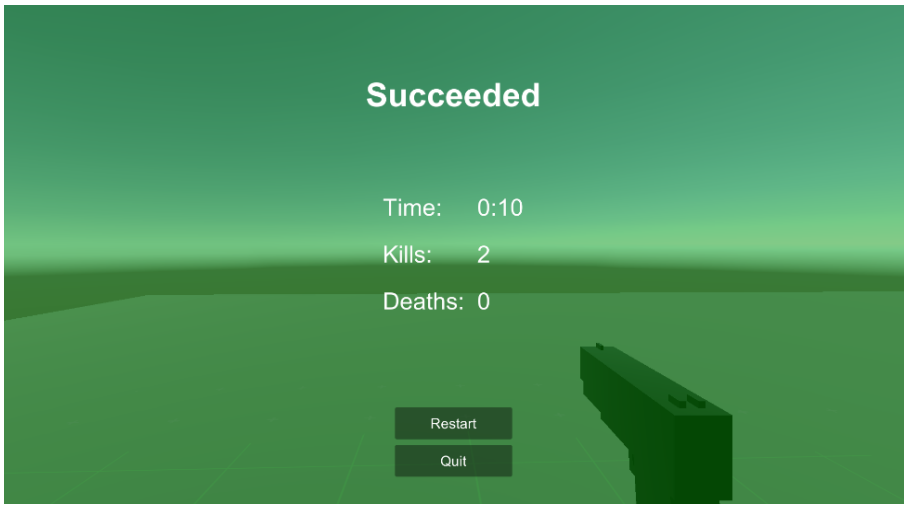
The Tag dropdown is accessible towards the top of the Inspector tab while a GameObject is selected.

If the size of the scene increases enough, the boundary will need adjusting to prevent GameObjects from being despawned in the playable area which is explained in the Boundary Manager section.

Implementing Success and Fail Game States

The scenes are linked together by a versatile menu system that utilises success and fail states which can resemble the player completing the level or encountering death respectively.

Both states can be easily activated in different ways and a scene can feature a combination of them.

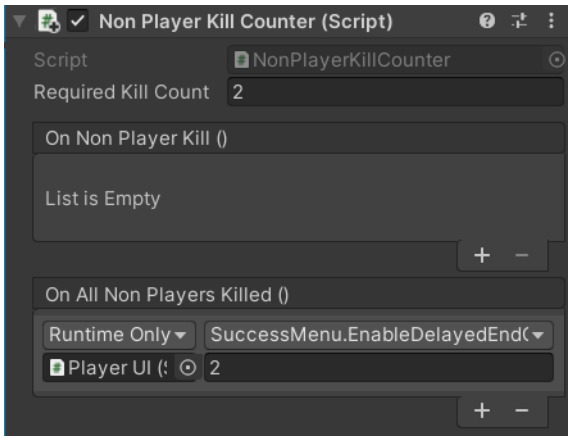


If following the step-by-step structure of these tutorials, note that it would be unnecessary to introduce all of these concepts into one level, thus choose which success and fail states to include then comply with their explanations.

Success

Kill Count

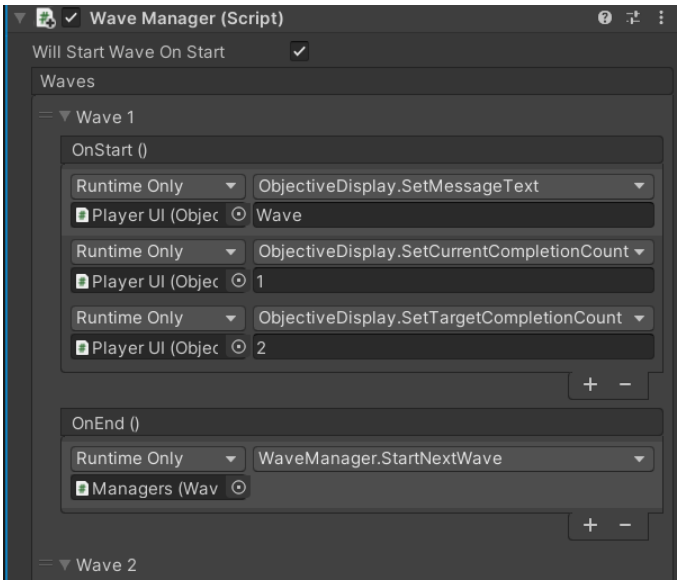
A system where the success state can be set once a chosen number of non-player kills has been reached can be adopted by:



1. Apply the Non Player Kill Detection component to the Managers GameObject of the scene.
2. Change the Required Zombie Kill Count field to the number of non-players that need to be killed to win.
3. In a spare On All Zombies Killed () UnityEvent element, drag the Player UI into the object field.
4. Click its No Function dropdown, hover over SuccessMenu and select EnableEndGame or EnableDelayedEndGame. Declare the end-game delay if the later function was selected.

Wave Completion

This tool enables projects to involve a wave-based structure which allows the invoking of success and fail states.

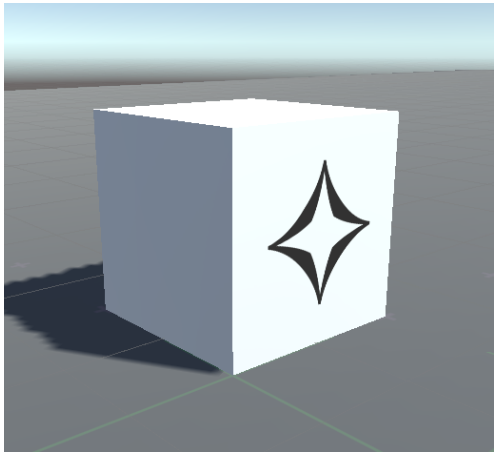


As the Waves reorderable list involves the UnityEvent field, SuccessMenu.EnableEndGame or EnableDelayedEndGame can be utilised by applying the Player UI to the object field and selecting such functions.

The full details on implementing this can be found in the Making Waves for a Zombie Game section.

Success Zone

The Success Zone is a Prefab that can be introduced to the level that is meant to be completed through the player reaching a destination.



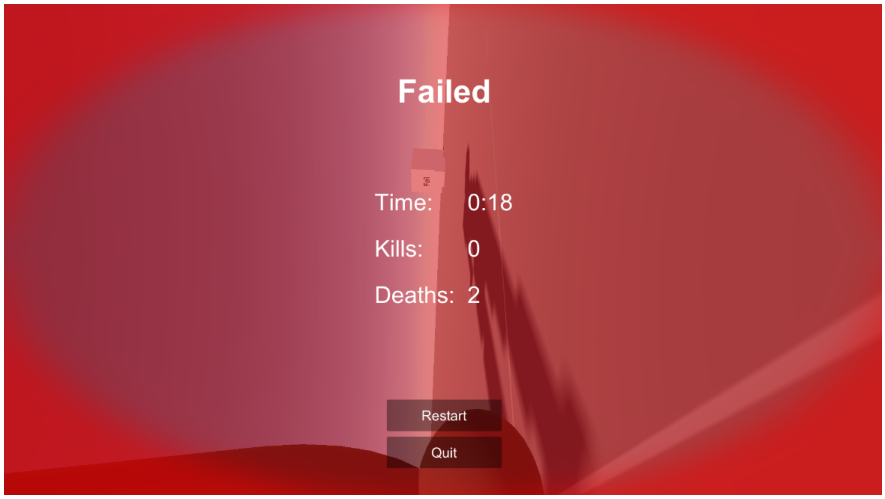
It works by using a Sphere Collider trigger and if the player intersects with it, the success state will be initiated. The Success Zone can be set-up in this way:

1. From the GGS FPS Integration Tool > Scene Prefabs, drag the Success Zone into the scene.
2. In the Trigger Action component of the Success Zone, expand Target Colliders, set the Size field to 1 and assign the player's Environment Collider to the element field.
3. Supply the Player UI to an element of the On Trigger Enter () field, then assign the SuccessMenu.EnableEndGame to its function dropdown.
4. Position the Success Zone GameObject in the scene and remove or replace its child GameObjects as needed.

Fail

Death

If the tutorials have been followed in order, the Player GameObject should already include a fail state that is triggered during death or a death without any remaining lives. If that is the case, this implementation can be skipped.



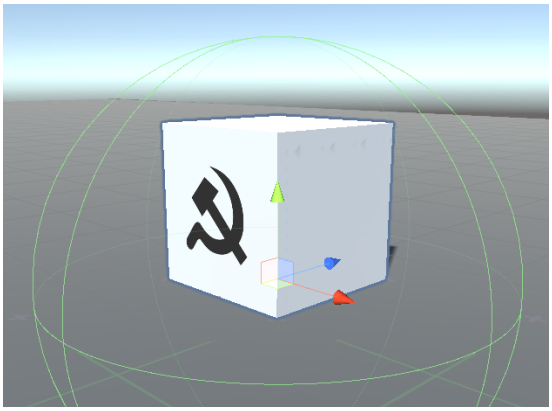
The Health component featuring UnityEvent fields can also call end-game states and failure of the level through excess player deaths can be utilised with these steps:

1. Access the Player GameObject then find its Health component.
2. In its On Death () or On Death Without Lives () UnityEvent fields (depending on which is displayed) create a new element if one is not already available and assign the Player UI to the element's object field.

3. Click its No Function dropdown then hover over FailMenu and select EnableDelayedEndGame.
4. In the field below the dropdown, set the value to around 2.

Fail Zone

This works identically to a Success Zone but triggers the fail state instead.



1. From the GGS FPS Integration Tool > Scene Prefabs, drag the Fail Zone into the scene.
2. In the Fail Zone and its Trigger Action component, expand Target Colliders, set the Size field to 1 and assign the player's Environment Collider to the element field.
3. Apply the Player UI to an element's object field belonging to a On Trigger Enter () UnityEvent element.
4. Assign the FailMenu.EnableEndGame to its No Function dropdown.
5. Reposition the Fail Zone GameObject and alter its child GameObjects as needed.

Making Waves for a Zombie Game

The FPS Integration Tool includes a wave system that allows the spawning of non-player entities and the manipulation of GameObjects to be controlled in an easily configurable sequence.

This feature was exercised in the Zombie Game scene and each of the versatile components listed below complement each other in producing such functionality:

Counter

The Counter is a component that can be applied to relevant GameObject and can coexist with many others in the scene.

It tracks a count that is incremented via other components that utilise UnityEvent fields and it also uses such fields which are executed when the count meets a set threshold.

Wave Manager

The Wave Manager component, usually added to the Managers GameObject, involves a reorderable list of UnityEvent fields used to structure the events that are invoked from the start and end of each wave.

Objective Display

This component is attached to the Player UI GameObject and allows the objectives and process to be updated; its message text, colour, current and target figures can be altered from UnityEvent elements.

This has to be considered when implementing a wave system as the Wave Manager does not handle the display of objectives automatically.

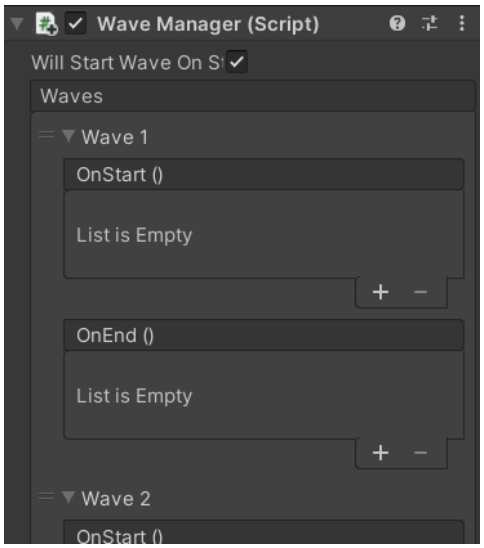
Implementing the Wave System

In this example, we will control the activation of three NonPlayer Respawns and a Weapon Collectable Respawn.

The start and end of the waves will be determined by the number of non-players killed and the success state will be triggered after the end of the final wave.

Step 1

Include and set-up the Wave Manager:



1. Apply the Wave Manager component to the scene's Managers GameObject.
2. Click the plus icon twice in the Waves reorderable list to create two new elements then expand their foldouts.

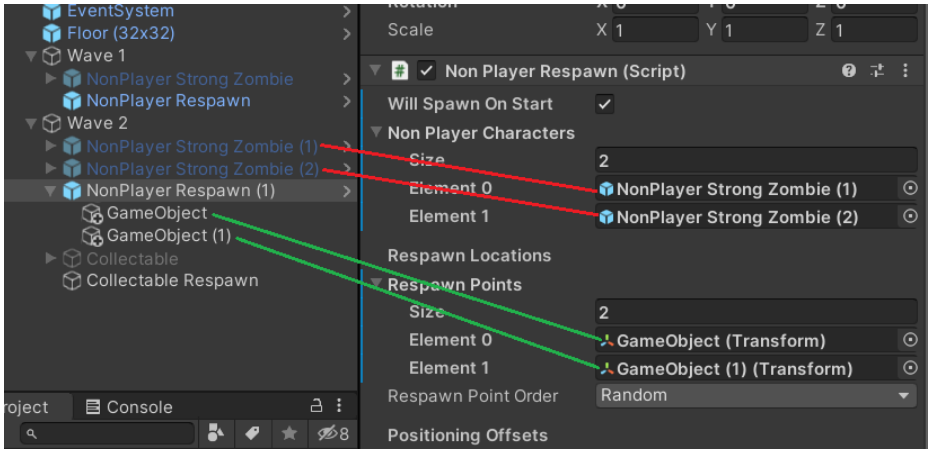
Step 2

Create and arrange the required GameObjects:

1. Create two new empty GameObjects and name them Wave 1 and Wave 2.
2. Add the NonPlayer Zombie and Respawn into the Wave 1 GameObject.
3. Duplicate two extra NonPlayer Zombies and one NonPlayer Respawn GameObject then apply them to the Wave 2 GameObject. Note that incremented preceding numbers will be added to the end of the duplicated names.
4. Also add the collectable GameObject and its respawn into Wave 2.
5. In Wave 2 and under its NonPlayer Respawn in the Hierarchy tab, create two new GameObjects. These will be the respawn points on the Respawn GameObject, so position them adequately.

Step 3

Edit the Non Player Respawn components of Wave 2:

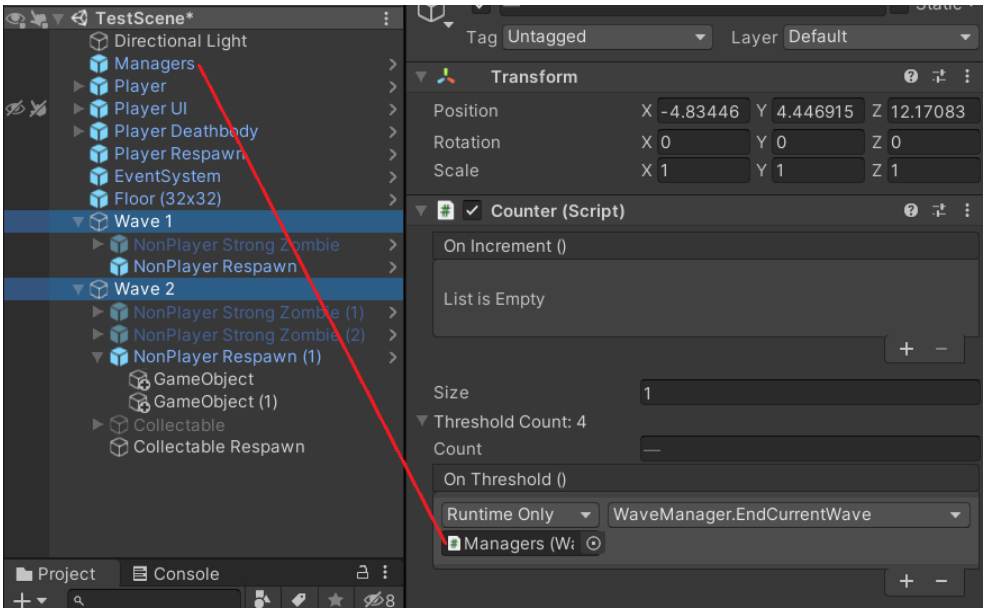


1. In the Non Player Respawn GameObject and component of Wave 2, expand the NonPlayer Characters foldout, set the Size field to 2 and assign the non-player character GameObjects of Wave 2 into the element fields.

2. Access the Respawn Points array of the component and change the Size field to 2 then apply the child GameObjects of the NonPlayer Respawn GameObject to the element fields.

Step 4

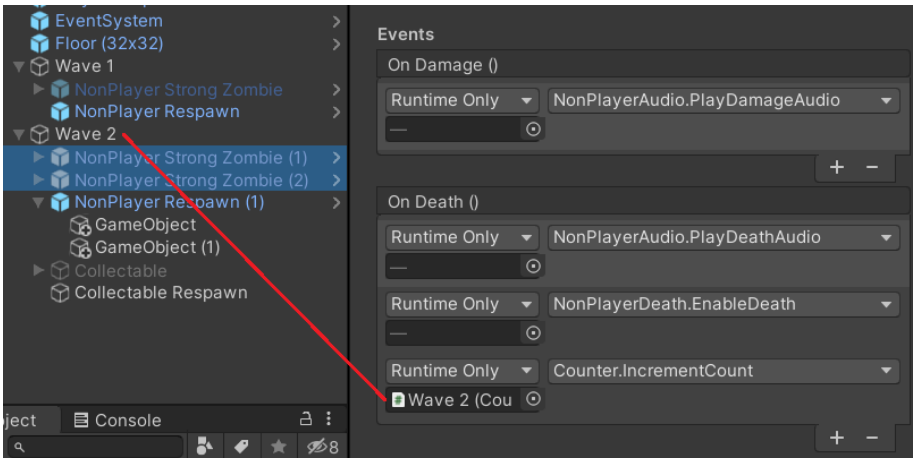
Set-up the Counter components for both waves:



1. For both Wave 1 and 2 GameObjects, include the Counter components and change their Size field to 1 and expand the Threshold Count foldouts.
2. For Wave 1, change its Count field to 2 and for the other, edit its count to 4.
3. In a Counter component's On Threshold () UnityEvent element for both waves, drop the Managers GameObject into the object field, then select WaveManager.EndCurrentWave from the No Function dropdown.

Step 5

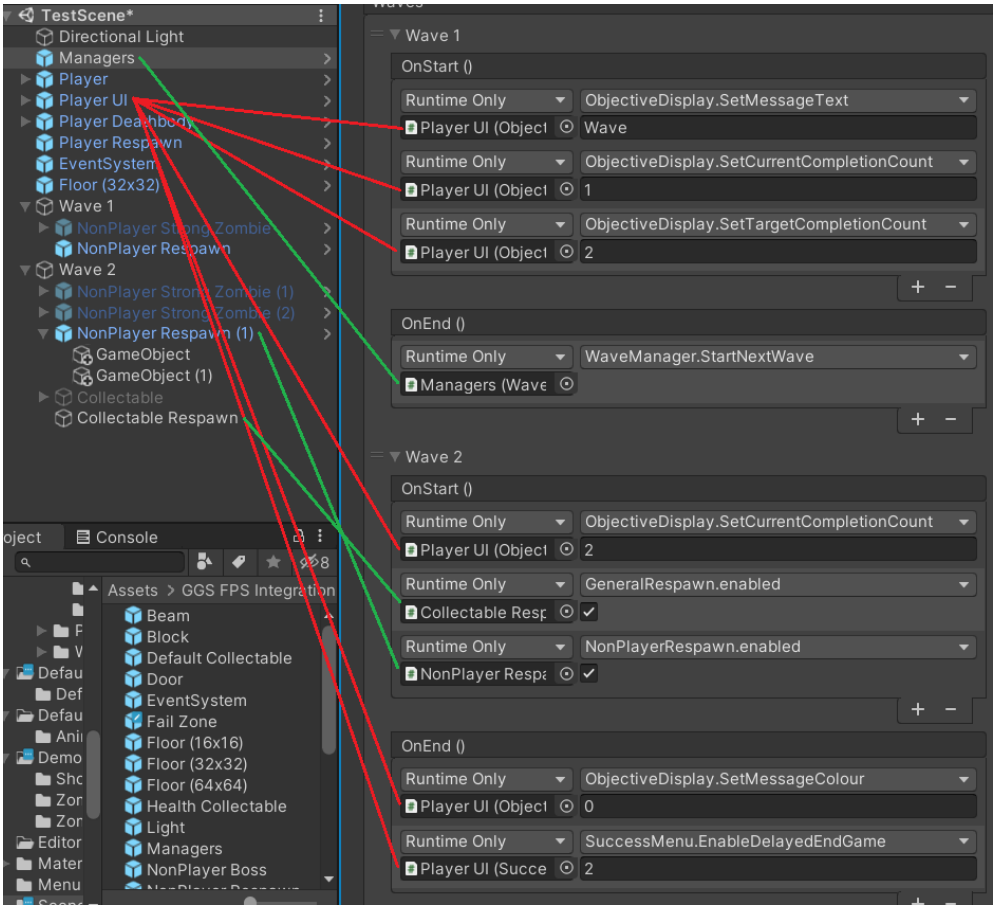
Link the Health component On Death () fields of the zombie entities to their wave's Counters:



1. With the non-player character within Wave 1, add an element to the Health component's On Death() UnityEvent field and assign the Wave 1 GameObject to the element and choose Counter.IncrementCount from its dropdown.
2. Do the same with each non-player entity in Wave 2 but ensure the Counter component of Wave 2 is used instead.

Step 6

Configure the Managers GameObject's Wave Manager UnityEvent fields as follows:



Note the entries of the elements will be described in the format Object Field > No Function Dropdown > Value.

1. In Wave 1:
    a. OnStart ():
        i. Player UI > ObjectiveDisplay.SetMessageText > "Wave"

ii. Player UI > ObjectiveDisplay.SetCurrentCompletionCount > 1

iii. Player UI > ObjectiveDisplay.SetTargetCompletionCount > 2

b. OnEnd ():

i. Managers > WaveManager.StartNextWave

2. Wave 2:

a. OnStart ():

i. Player UI > ObjectiveDisplay.SetCurrentCompletionCount > 2

ii. Collectable Respawn (of Wave 2) > GeneralRepsawn.enabled > Ticked (true)

iii. NonPlayer Respawn (of Wave 2) > NonPlayerRespawn.enabled > Ticked (true)

b. OnEnd ():

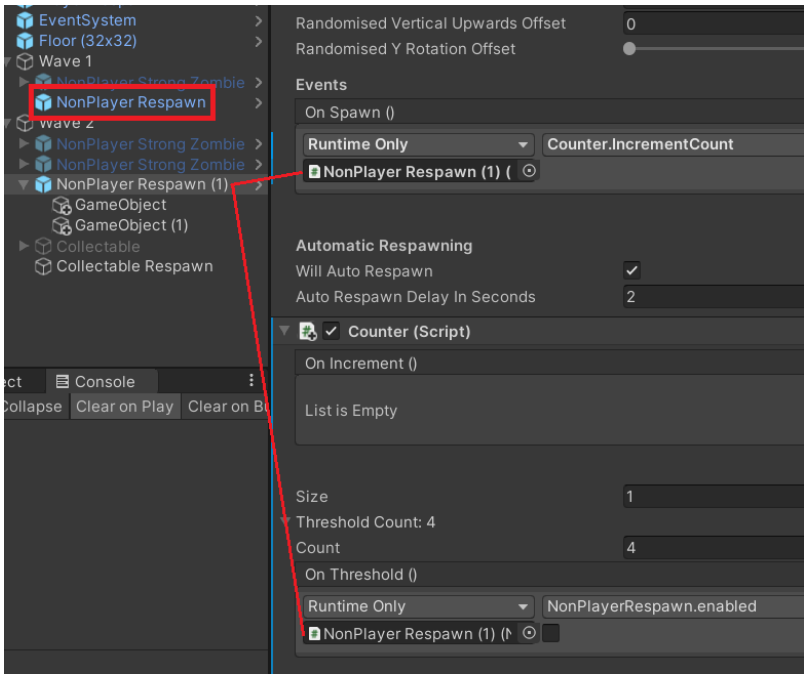
i. Player UI > ObjectiveDisplay.SetMessageColour > 0

ii. PlayerUI > SuccessMenu.EnabledDelayedEndGame > 2

The Objective Display functions influence the display of the objectives in the Player UI.

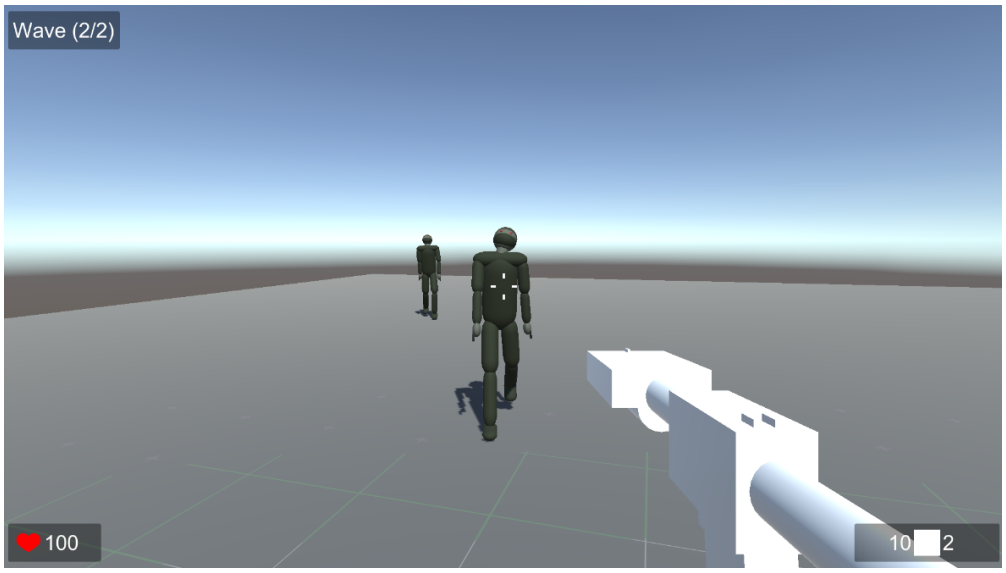
Step 7

Disable respawn components of the Wave 2 GameObject then attached and calibrate the Counters to the NonPlayer Respawn GameObjects of both waves:



1. Disable both the NonPlayer and General Respawn components of the non-player and collectable respawns in Wave 2 respectively. This can be done by unticking the tickbox from the component's header in the Inspector tab.
2. Add the Counter to the NonPlayer Respawn GameObjects in Wave 1 and 2.
3. Set the Size field to 1, the Threshold Count to 2 and 4 in the NonPlayer Respawn GameObjects of Wave 1 and 2 respectively.
4. Then in the new On Threshold () elements, supply the object fields with the same NonPlayer Respawn GameObjects that the Counters are attached to.
5. Then with their No Function dropdown, choose NonPlayerRespawn.enabled and untick its tickbox.
6. For both Wave 1 and 2, inside the NonPlayer Respawn GameObject, within a On Spawn () UnityEvent element of the Non Player Respawn component, add the related NonPlayer Respawn GameObject to the element's object field and select Counter.IncrementCount.

At this point, the wave system should work correctly.



The first wave features a zombie spawning once at a time and after 2 kills the next wave begins.

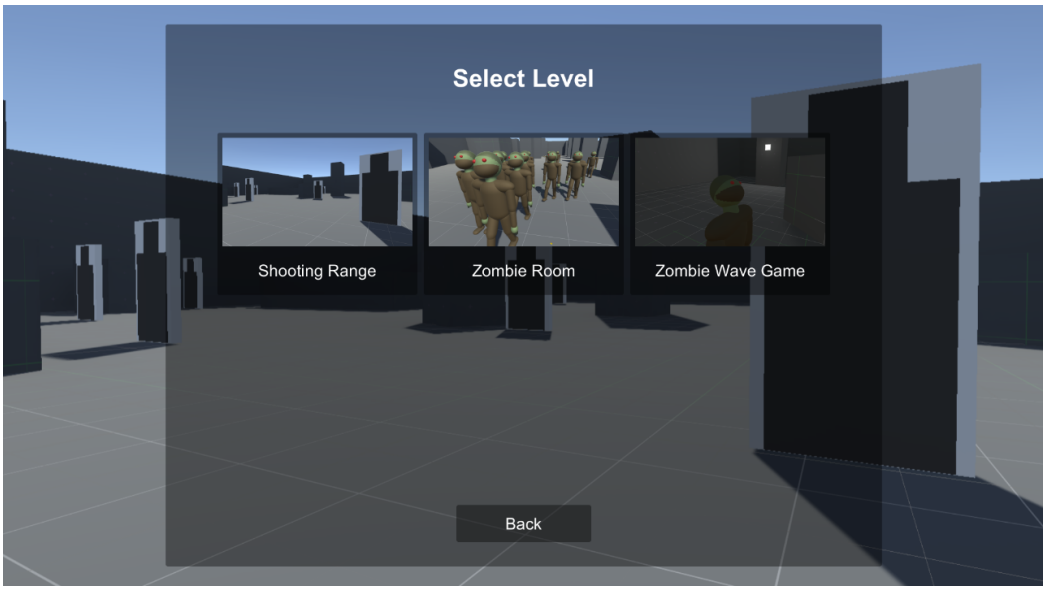
The second wave involves up to 2 zombies that spawn at a time from two different locations while also allowing a collectable to start instantiating.

When another 4 kills are gained, the objective display turns green and the success state is enabled soon after.

Using the Main Menu Level Selection

Up to six scenes can be added to the custom menu system that is featured in this tool, where a title, icon and description can accompany each scene.

Information of the Level Selection Organiser component can be sourced here.

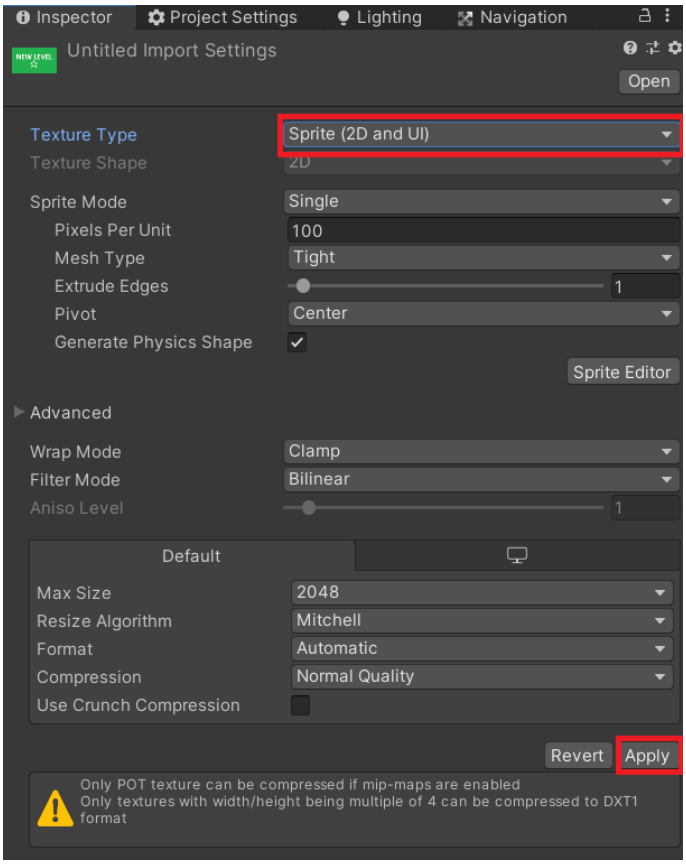


This can be achieved by following these steps:

Step 1

Import a 350 by 200 pixel image into Unity and make it into a Sprite:

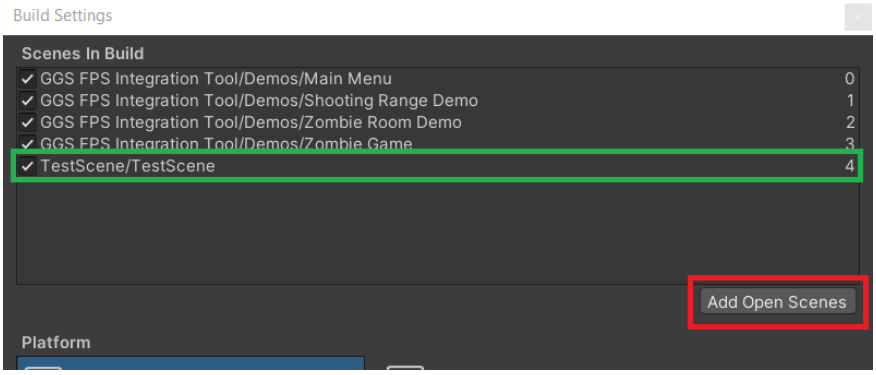
1. Create or source an image with the dimensions 350 by 200 pixels and import it into Unity by dragging the file into Unity's Project tab.
2. Select the imported image via the Project tab and in the Inspector, change the Texture Type dropdown from Default to Sprite (2D and UI), then click the Apply button towards the bottom of the menu.



Step 2

List the new scene in the Build Settings:

1. Open the new scene that will be included in the Level Selection Menu.
2. Open the Build Settings through File; shown on the toolbar.
3. In the Build Settings window, click the Add Open Scenes button. This will allow the opened scene to be accessible to the Main Menu during Play Mode and references the scene with an index starting from zero that should be noted for the next step.

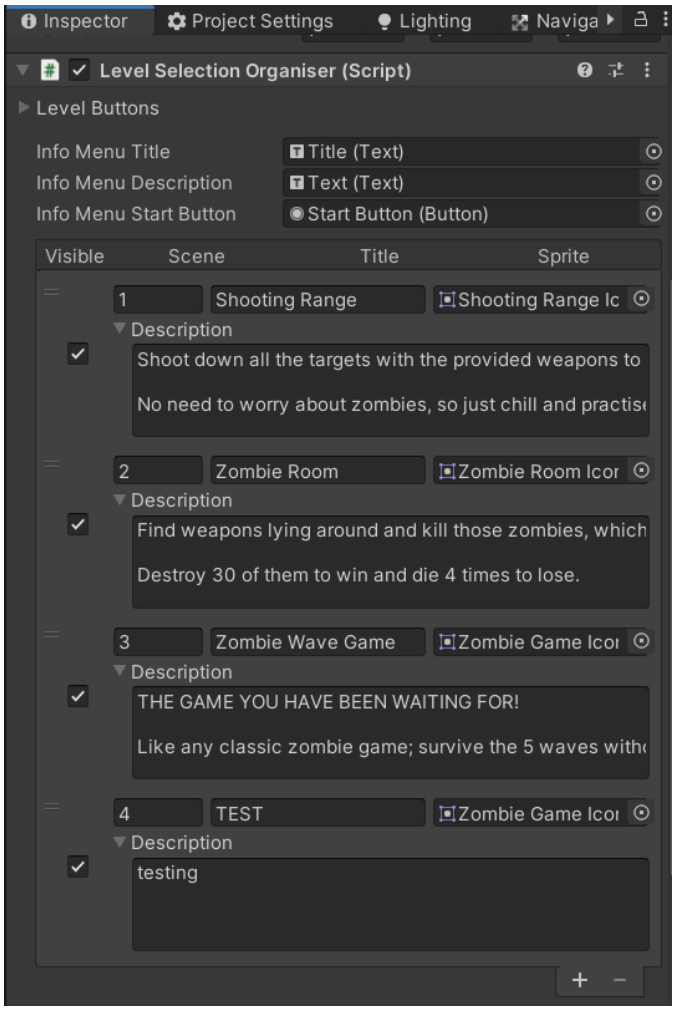


4. Ensure the relevant tickboxes that are next to the scenes' directories are ticked.

Step 3

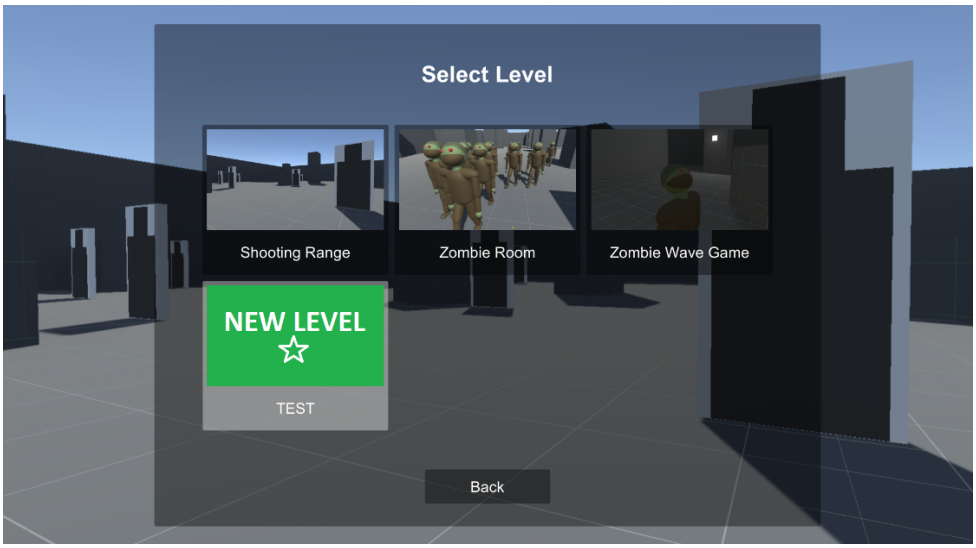
In the Main Menu scene, amend the Level Selection Organiser component of the Level Selection Menu GameObject to include the new level:

1. Within the Main Menu scene, found in the GGS FPS Integration Tool > Demos folder, and under UI Canvas in the Hierarchy tab, select the Level Selection Menu GameObject and find the Level Selection Organiser component.



2. Towards the bottom of the reorderable list, click the plus icon to create a new element if required.
3. Assign the new scene's index, as shown in the Build Settings, into the top-left field of the element.
4. Add the title in the centre field and the imported and configured image to the right field.
5. Include details about the level in the Description field; expand using its foldout.
6. Adjust the order of the elements and toggle the Visible tickbox to control how it is displayed in the Level Selection Menu.

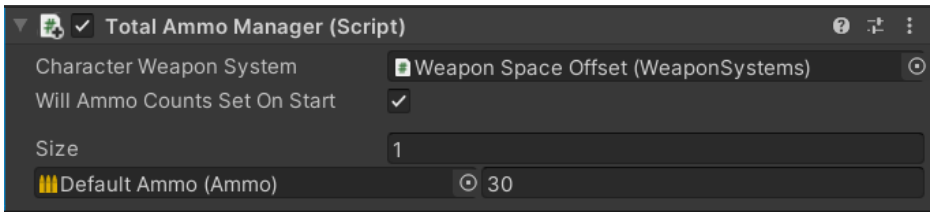
After enabling Play Mode and clicking the Play button in the first menu, the implemented level should be featured in the Level Selection Menu.



Once the scene's button is clicked, the Level Info menu is shown exposing the description and the Start button, which should lead to the new level.

Altering the Total Ammo on Start across Scenes

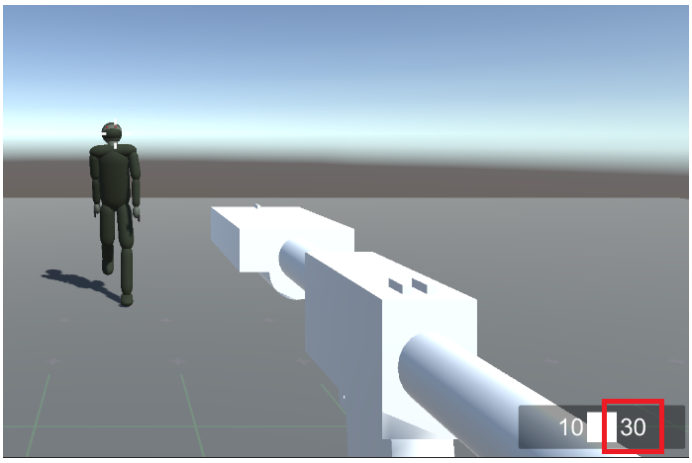
The Ammo ScriptableObjects containing the Start Ammo field outlines the default amount of ammo possessed by the player from the start of any scene.



To customise the amount of ammo supplied to the player from starting a certain scene:

1. Include the Total Ammo Manager component into the chosen scene's Managers GameObject.
2. Assign the Weapon Systems component of the player's Weapon Space Offset to the Character Weapon System field and tick the Will Ammo Counts Set On Start.
3. Declare the number of ammo amounts to alter via the Size field.
4. Complete the Ammo ScriptableObject fields with their corresponding total ammo counts granted from the beginning of the level.

Upon starting the level, the mentioned amount of ammo is applied to the target ammo type.



Weapon Space Animation Guidelines

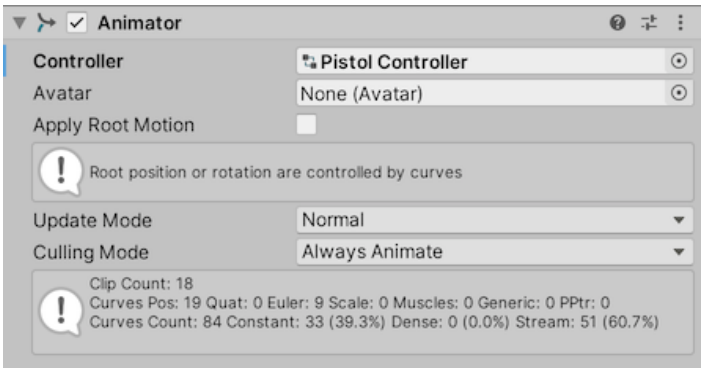
Although Unity's animation system is complex, it offers great flexibility and power for producing animations, which is why it is utilised by the FPS Integration Tool.

However, for animations to function appropriately with the tool, animations should follow the guidelines below.

Terminology

To avoid confusion with the elements that form Unity's animation system, the terms are explained:

- Animation Tab - Is the window for creating animations which features a Timeline.
- Animator Tab - Also a window, but used for structuring how a collection of animations are executed, which involves a flow diagram.
- Animator - Often refers to the component which applies the behaviours of an Animator Controller into the GameObject.

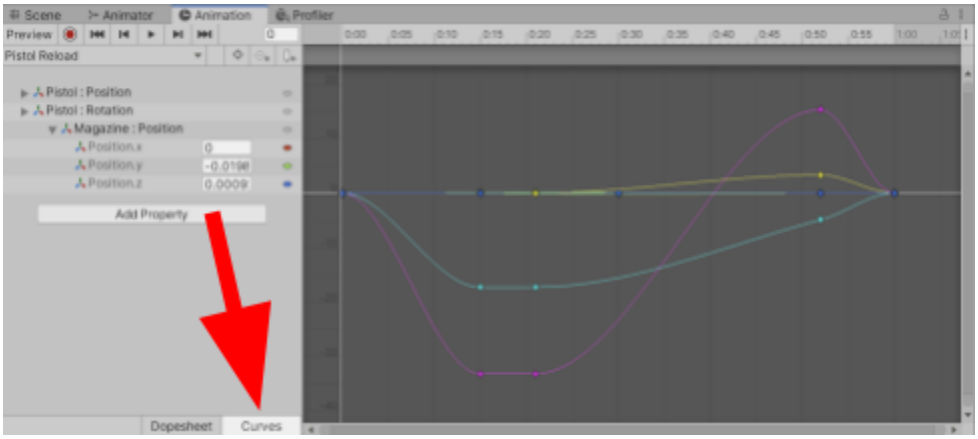


- Animator Controller - Is used in an Animator component to provide an animation system onto the GameObject.
- Animation Clip - Holds a particular animation to be used within an Animator Controller.

Animation Tab

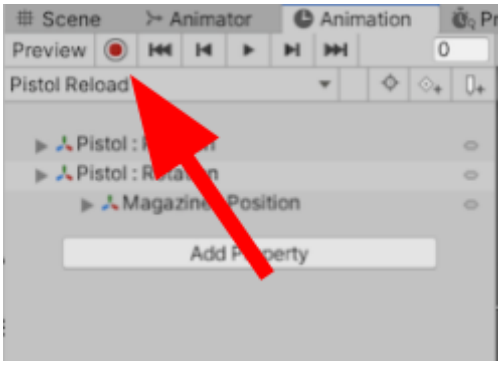
Terms related to the inner workings of the Animation tab:

- Dopesheet - The timeline interface that displays keyframes as dots.
- Curves - The timeline interface that uses curves between keyframes to describe characteristics of the transitions.



- Keyframe - A point in the Animation tab timeline that marks a state of a GameObject, such as a position or rotation. Gradual transitions occur between keyframes as the animation is played.
- Keyframe Recording Mode - The red recording button at the top-left of the Animation tab that allows GameObjects' states to be recorded if altered in the Scene tab. Often used

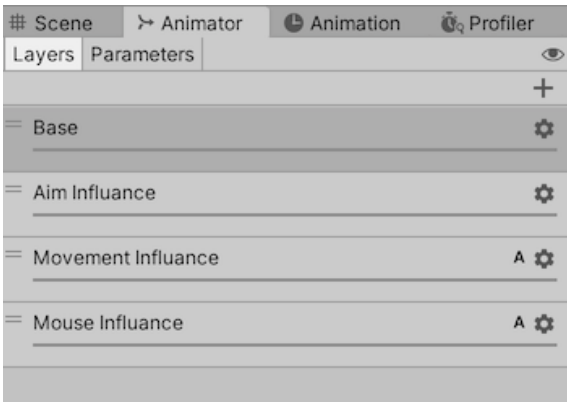
for animation development involving GameObject Transforms such as position, rotation and scale.



Animator Tab

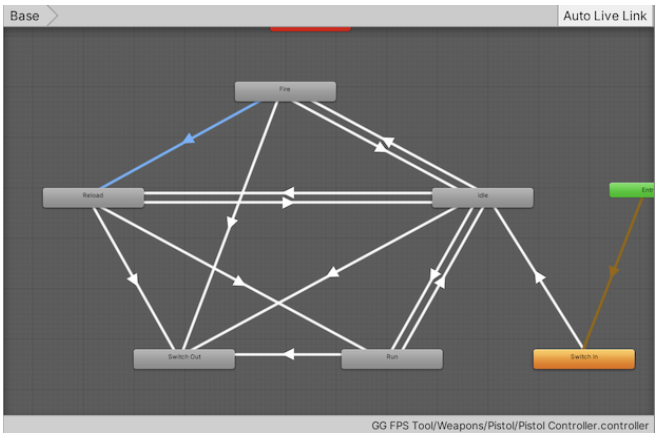
Terms for the Animator tab involves:

- Layers - Levels of animations that can influence others in additive or overriding ways, useful for combining animations which may need to play in parallel during runtime.



- Parameters - Variables of certain types that are often used to trigger transitions between animations.

- Nodes - States in the Animator Controller that plays a certain animation when executed.
- Transitions - Links between two Nodes that allows animations to transition once the specified Parameter conditions are met.



Layers (Animation)

In general, animations used with the weapon can be categorised as:

- Base - Where most of the animations related to the weapons' actions lie such as Fire, Switch and Reload, and where most animation editing takes place.
- Aim Influence - Features Hip and Aim animations which hold a constant position and uses transitions for switching between the two states, this is also often edited.
- Movement Influence - Supports animations that are controlled by the movement of the FPS Player GameObject.

- Mouse Influence - Is for animations controlled by mouse movements which rotates the FPS Player GameObject.

Remember, Layers for animations differ from Layers for GameObjects.

For more information on Layers for GameObjects, check the Layers (GameObject) section.

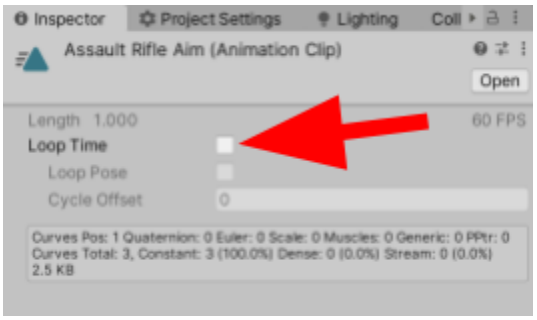
Hierarchy

To allow parallel animations provided by the Animator Controller Layers to affect the weapon correctly, the Weapon Space GameObject contains levels of GameObjects under it within the Hierarchy tab which are controlled by specific Layers:

- Weapon Space - Used by Mouse & Movement Influence Layers.
    - Weapon GameObject - Is spawned from the prefab during Play Mode, and is controlled by Base Layer animations.
        - Aimbody - Affected by the Aim Influence Layer.
            - Weapon Sections/Primitives - Are the parts of the weapon which can be individually animated within Base and Aim Influence Layers. Useful for creating more advanced animations, for example, moving the magazine during reloading.

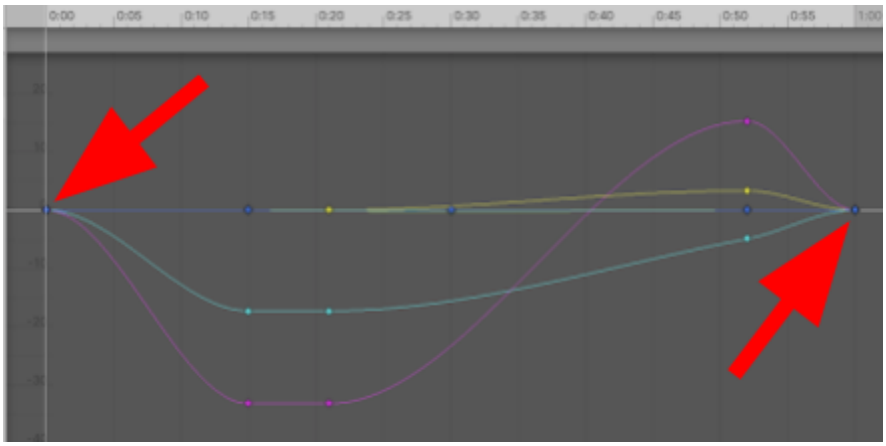
General

1. Animations should be 1 second long for easy management of speed within the Animator Controller Nodes.
2. With the Weapon ScriptableObject, ensure the correct Animation Clips and Animation Controller are applied to the fields.
3. Animation Timing Fields in Weapon ScriptableObjects should contain relatable values to certain Animator Controller Transitions to ensure Unity's animation system and the FPS Integration Tool are synchronised to prevent disordered outcomes.
4. Animation Clips' Loop Time should be unticked, unless the animation should repeat like for Partial Repeat reloading weapons.



5. If an Animator Controller becomes severely disrupted through editing, consider copying and pasting the Animator Controller of the Default Weapon to retry. Though ensure the Default Weapon files remain unchanged to prevent system malfunctions.

6. For creating realistic animations, the weapon's animated GameObjects should return to their initial position, rotation and scale by the end of the animation. Some types of animations are exempt from this rule, such as Switch animations.



7. Avoid editing the Animator Controller Parameters naming and Layer settings.

Idle Animations

Idle animations just involve the Weapon GameObject in its normal position throughout the animation.

Fire Animations

Often, firing animations rely on the incoming transition to replicate recoil, as firing often pushes the weapon back suddenly. When animating this, the animation should start with the weapon positioned slightly backwards then gradually returns to the initial position by the end.

Inconsistent Animations

Sometimes, firing animations may not play every time a round is fired even if the Speed field of the Fire Node matches the Fire Rate field in the corresponding Weapon ScriptableObject, this is due to the transition delay and can be remedied by slightly increasing the Node's speed.

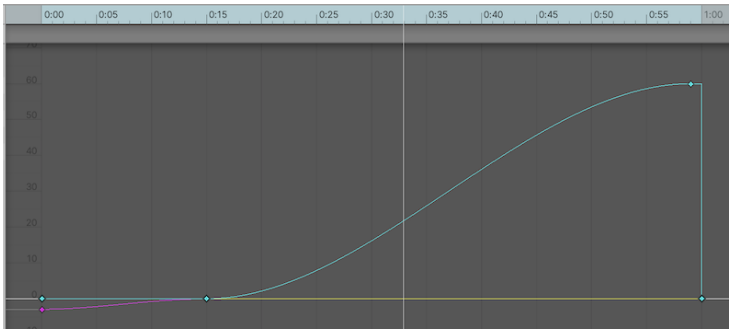
Revolving Chambers

Producing something like the Grenade Launcher's firing animation which involves the chambers partially rotating can cause difficulty with realistically animating the Weapon's GameObjects to return to their initial Transforms.

The solution to this involves:

1. In the timeline, set the Transforms to their initial states at the animation's end (1:00 second).
2. Animate the rotating chambers and ensure they rotate to a similar-appearing state. Consider calculating the amount of rotation by the equation **360 / Number of Rotational Symmetry = Degrees of Rotation**.
3. Position the rotation's ending Keyframe just before the resetting Keyframe (0:59 second), so there is minimal time between the rotation's and animation's end.
4. Switch the Animation tab from Dopesheet to the Curves setting.
5. Find the second to final Keyframe Node (on what should be the most expressive curve) then right-click it and select Broken from the Pop-Up Menu.
6. Move the Handle Node on the right side of that Keyframe Node downwards so it is directly below the Keyframe Node. Doing this turns the curve into a right angle, meaning the transition will be sudden instead of gradual.

The result should be a firing animation featuring a revolving section which rotates to the next chamber without any apparent reverting change at the animation's end.



Reload Animations

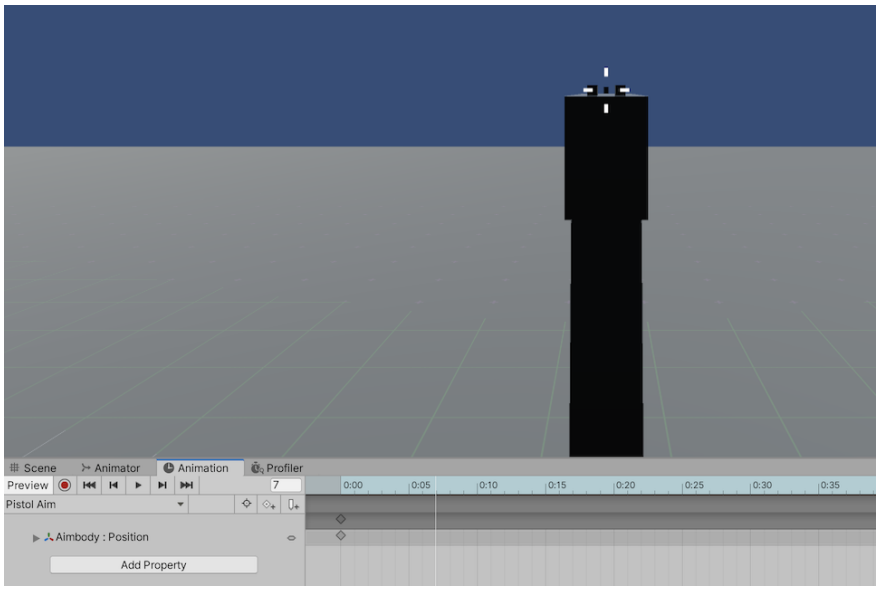
Reload animations should begin and end at the original idle positions, unless it uses **Partial** or **Partial Repeat** Reload Types enabled within the Weapon ScriptableObject, as this may need to repeat. Thus, the animations start and end should match each other and be positioned towards their target position instead of the idle position. Such animations utilise the Transitions to produce the idle to target movement.

Hip Animations

Hip animations are similar to the idle animations as they feature the initial Transforms throughout.

Aim Animations

Aim animations involve positioning the Weapon GameObject where it's own sight can be used to aim without the crosshair. When producing the Aim animation, ensure the sight accurately overlaps the pre-applied crosshair to ensure the raycasting matches the weapon sight direction. The whole animation is also constant.



Switch Animations

Switch animations should begin with the weapon off from view and end when at the idle position, by definition this is a Switch-In. For ease of animating, Switch-In animations can be

reused as a Switch-Out by setting their Node's Speed field to -1 (or the same number as the Switch-In Speed times -1).

Run Animations

Run animations are similar to Idle in that it resembles a constant position, but the Weapon GameObject is aimed away from ahead such as downwards. The jogging motion is managed automatically by the Movement Influence Layer.

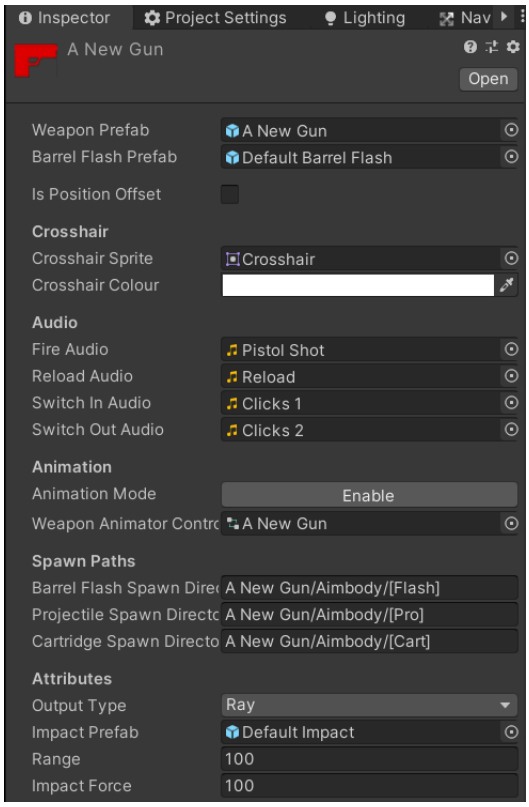
ScriptableObjects

ScriptableObjects are objects that behave like files rather than GameObjects. They are designed to hold values that can be modified via the Inspector tab, and can be applied to a field in another menu instead of being placed in the scene.

The FPS Integration Tool utilises this feature and includes these types of ScriptableObjects:

Weapon

Weapon ScriptableObjects contain data relating to a specific weapon's characteristics.



They are often assigned to WeaponCollection ScriptableObjects and are created via the Create Options.

The Weapon ScriptableObject involves these fields:

Field Name / Heading	Data Type	Description
Weapon Prefab	Prefab	The Prefab is instantiated when the weapon switches-in and such GameObject is destroyed when switching-out. The **Decollide** message box appears if the applied Prefab or its children contains any collider components, which could disrupt the raycast firing. Click the Decollide button and again in the pop-up menu to resolve this. The **Relayer** message box appears if the applied Prefab or its children are not in the FirstPerson layer, which could disrupt the weapon's rendering. Click the Relayer button and again in

		the pop-up menu to fix the issue.
Barrel Flash Prefab	Prefab	This Prefab is spawned when firing the weapon at the referencing GameObject defined from the Barrel Flash Spawn Directory field.
Is Position Offset	Tickbox (Boolean)	If ticked (true), the position of the Weapon Space Offset GameObject is adjusted with the Position Offset value for that weapon. If unticked (false), no position offset is applied.
Position Offset	Vector 3	**This field is displayed only when Is Position Offset is set to True.** The local space position offset for the Weapon Space Offset GameObject applied when switching to this weapon. This is useful for repositioning particular weapons without editing the weapon's animations.
Crosshair		
Crosshair Sprite	Sprite	The Sprite that is applied to the Crosshair Space of the Player UI when switching-in the weapon.
Crosshair Colour	Colour	Tints the Crosshair Sprite in the defined colour.
Audio		
Barrel Audio	Audio	The audio that is played when firing the weapon.
Reload Audio	Audio	The audio that is played when reloading the weapon.
Switch In Audio	Audio	The audio that is played when switching in the weapon.

Switch Out Audio	Audio	The audio that is played when switching out the weapon.
Animation		
Animation Mode	Toggle Button	When the button is clicked while displaying Enable, the weapon's Prefab and Animator Controller is applied to the Weapon Space GameObject. When clicked while displaying Disable, the Weapon's Prefab and Animator Controller are removed from the Weapon Space GameObject. If clicking the button of another Weapon ScriptableObject's Animation Mode while the initial weapon is in Animation Mode, the Prefab and Animator Controller is automatically replaced.
Weapon Animator Controller	Animator Controller	The Animator Controller of the weapon that is applied to the Weapon Space Animator component when the weapon is wielded.
Spawn Paths		
Barrel Flash Spawn Directory	String	Is the Hierarchy directory under the Weapon Space GameObject of the barrel flash spawn. Usually follows the format: 'Weapon Name'/Aimbody/[Flash] For example: MyWeapon/Aimbody/[Flash]
Projectile Spawn Directory	String	Is the Hierarchy directory under the Weapon Space of the projectile spawn. Usually follows the format: 'Weapon Name'/Aimbody/[Pro] For example: MyWeapon/Aimbody/[Pro]
Cartridge Spawn	String	Is the Hierarchy directory under the Weapon

Directory		Space of the cartridge spawn. Usually follows the format: ‘Weapon Name’/Aimbody/[Cart] For example: MyWeapon/Aimbody/[Cart]
Attributes		
Output Type	Options	Defines the type of simulated projectile fired from the weapon with the options Ray or Projectile. If set to Ray, a raycast is used to represent a projectile. If set to Projectile, a specified projectile GameObject is launched instead.
Impact Prefab	Prefab	**This field is displayed only when Output Type is set to Ray.** The Prefab that is spawned at the raycast’s hit location, often to simulate a bullet hole or equivalent.
Range	Float	**This field is displayed only when Output Type is set to Ray.** The maximum distance of the Raycast from the Camera Ray Spawn GameObject.
Impact Force	Float	**This field is displayed only when Output Type is set to Ray.** The force that is applied to a Rigidbody that was hit by the Raycast.
Damage Per Shot	Float	**This field is displayed only when Output Type is set to Ray.** The amount of damage per hit to inflict on GameObjects with Health components. Each raycast applies the damage equal to

		Damage Per Shot divided by Output Per Shot.
Projectile Prefab	Prefab	**This field is displayed only when Output Type is set to Projectile.** The Prefab that is launched from the projectile spawn point.
Launch Force	Float	**This field is displayed only when Output Type is set to Projectile.** The forward force applied to the newly spawned projectile GameObject.
Firing Type	Options	The type of firing response used when firing the weapon with the options Semi Automatic or Automatic. If set to Semi Automatic, the weapon is fired per button/key press, but firing will never exceed Shots Per Second. If set to Automatic, the weapon fires repeatedly at the Shots Per Second rate while the firing button/key is held down.
Shots Per Burst	Integer	Number of shots fired per fire button/key press.
Shots Per Second	Float	Maximum number of shots fired per second.
Output Per Shot	Integer	Number of raycasts or projectiles launched per shot. Exceeding the value of 10 is not recommended as it may cause unnecessary overhead.
Aiming Spread	Float Range	The Euler angle offset of the firing direction when aiming. Lower values provide more accuracy. Aiming Spread is often the lowest spread value.

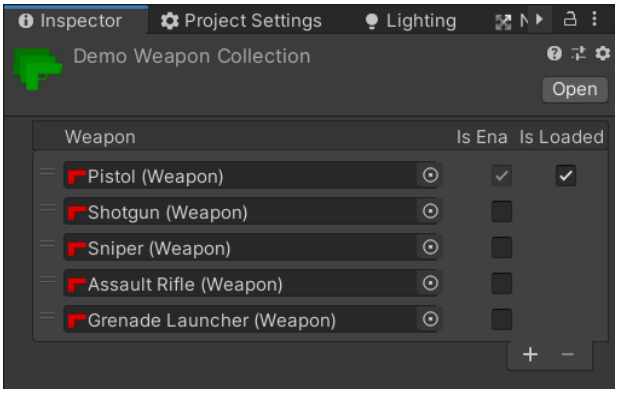
Hip Spread	Float Range	The Euler angle offset of the firing direction when not aiming. Lower values provide more accuracy. Hip Spread is often between Aiming Spread and Movement Spread values.
Movement Spread	Float Range	The Euler angle offset of the firing direction when not aiming while moving. Lower values provide more accuracy. Movement Spread is often the highest spread value.
Ammo	Ammo	The type of ammo used by the weapon.
Capacity	Integer	The maximum amount of ammo that can be loaded into the weapon.
Ammo Loss Per Shot	Integer	Amount of ammo decreased from the Capacity per shot.
Reloading Type	Options	Defines the weapon's nature of reloading with the options Full, Partial or Partial Repeat. If set to Full, the weapon is normally fully reloaded. If set to Partial, the weapon's ammo is normally increased by a set amount during reloading. If set to Partial Repeat, the weapon's ammo is normally increased by a set amount but the reloading process repeats until the weapon is fully loaded.
Ammo Added Per Reload	Integer	**This field is displayed only when Reloading Type is set to Partial or Partial Repeat.** The amount of ammo normally added to the weapon during reloading.
Animation Timing		

Aiming Transition Time	Float	Defines the duration in seconds taken from activating or deactivating the aiming to the effects being applied. The value of this field should be similar to the **Transition Duration (s)** field in the **Aim -> Hip** and **Hip -> Aim** Transitions of the Animator Controller's Aim Influence Layer.
Reloading Time	Float	Defines the duration of the reloading process in seconds. Should be proportionate to the Speed field of the Reload Node in the Base Layer of the Weapon's Animator Controller. As this Reloading Time and the Reload Node's Speed are based on Seconds and Speed Multiplication respectively, this conversion is required: Speed = 1 / Seconds Seconds = 1 / Speed
Switching Time	Float	Defines the duration of the switching process in seconds. Should be proportionate to the Speed field of the Switch-In and Switch-out Nodes in the Base Layer of the Weapon's Animator Controller. As this Switching Time and the Switch-In / Switch-out Node's Speed are based on Seconds and Speed Multiplication respectively, this conversion is required: Speed = 1 / Seconds Seconds = 1 / Speed
Running Recovery Time	Float	Amount of time firing is re-enabled after running. Should be similar to the Transition Duration (s) field in the Transitions exiting the Run Node in the Base Layer of the Weapon's Animator Controller.
Incremental Reload	Float	Amount of time firing is re-enabled after a Partial or Partial Repeat reloading is finished or

Recovery Time		interrupted. Should be similar to the Transition Duration (s) field in the Transitions exiting the Reload Node in the Base Layer of the Weapon's Animator Controller.
Cartridge Ejection		
Cartridge Prefab	Prefab	The Prefab that represents an ejected cartridge.
Ejection Trajectory	Vector3	The direction for the ejected cartridge to be launched towards.
Ejection Force	Float	The force applied to the ejected cartridge when launched.

WeaponCollection

The WeaponCollection ScriptableObject holds a reorderable list of Weapon ScriptableObjects that are used in the Weapon Systems component to define which weapons can be accessed by the Player GameObject.



WeaponCollection ScriptableObjects are created via the Create Options.

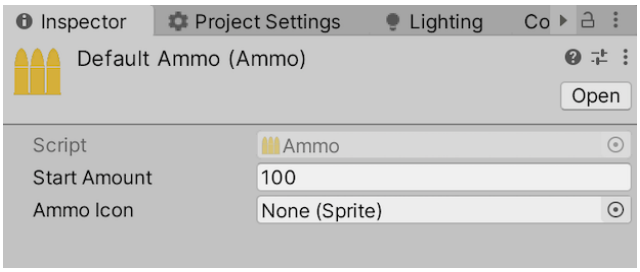
The WeaponCollection involves these fields:

Field Name /	Data Type	Description

Heading		
Weapon	Reorderable List	
=	Drag Handle	Allows the element to be reordered amongst other elements through dragging. The position of the element defines the order that it will be accessed when switching weapons.
Weapon	Weapon ScriptableObject	The weapon that will be loaded when switching to that element's order.
Is Enabled	Tickbox (Boolean)	If ticked (true), the weapon will be accessible from the start. If unticked (false), the weapon will need to be manually enabled after the scene starts.
Is Loaded	Tickbox (Boolean)	**This field is displayed only when Is Enabled is set to True.** If ticked (true), the weapon will be fully loaded from the start.
+	Add element	Adds another element to the reorderable list.
-	Remove element	Removes the selected element from the reorderable list.

Ammo

The Ammo ScriptableObject stores information about a type of ammo that weapons can refer to.



Field Name / Heading	Data Type	Description
Start Amount	Integer	Amount of this ammo available from the start.
Ammo Icon	Sprite	The sprite that is applied to the AmmoIcon GameObject when a weapon using this ammo is switched-in.

Components

The FPS Integration Tool comes with many MonoBehaviour scripts which provide certain functionality to GameObjects.

Many components are specifically designed for and are already implanted into Prefabs that can be introduced into scenes, thus the need to manually apply these tailored systems to GameObjects is kept minimal.

There are some components like Trigger Action, Moveable, Collectable, General Respawn, Non Player Kill Counter and so on, that are designed for more versatile operations and are meant to be attached to GameObjects more freely.

Some components may contain public properties and methods that can be accessed from your own scripts and sometimes through the UnityEvent fields of other components but this only applies to void methods with one parameter.

These are also described below but some are undocumented due to their purpose and sophistication.

The component included in the tool involves:

Player Controller

This script is pre-attached to the Player Prefab and provides movement mechanics with the related Character Controller as well as audio-visuals.

Field Name / Heading	Data Type	Description
Character Environment Camera	Camera	The camera that will be influenced with bobbing effects. The player's Environment Camera is meant to be assigned to this field.
Movement Inputs		
Movement X Axis Name	String	Name of the axis in the Input Manager used for tracking left and right Player GameObject movements.
Movement Y Axis Name	String	Name of the axis in the Input Manager used for tracking forwards and backwards Player GameObject movements.
Jump Button Name	String	Name of the key or button in the Input Manager that makes the player jump.
Run Button Name	String	Name of the key or button in the Input Manager to hold down to enable the player to run.
Mouse Inputs		
Mouse X Axis Name	String	Name of the axis in the Input Manager used for tracking horizontal mouse movements to rotate the player left and right.
Mouse Y Axis Name	String	Name of the axis in the Input Manager used for tracking vertical mouse movements to rotate the Player > Environment Camera GameObject upwards and downwards.

Movement Speeds		
Walking Speed	Float	The movement speed of the player when walking.
Running Speed	Float	The movement speed of the player when running. Often is set to a greater value than the Walking Speed field.
Mouse Sensitivities		
Mouse X Sensitivity	Float	The sensitivity multiplier for horizontal mouse movements.
Mouse Y Sensitivity	Float	The sensitivity multiplier for vertical mouse movements.
Aero Mobility		
Jump Force	Float	The upwards vector force applied to the player when jumping. A higher value leads to higher jumps.
Aero Mobility Multiplier	Float	The multiplier for effecting forwards, backwards, left and right player movements when it is airborne.
Gravity Multiplier	Float	Multiplier of the downwards force applied to the player when airborne.
Camera Bobbing Effects		
Walk Cycle Rate	Float	The per second rate of the walk cycle used for camera bobbing and footstep audio while walking or running.

Is Camera Bobbing Enabled	Tickbox (Boolean)	If ticked (true), camera bobbing will be activated. If unticked (false), camera bobbing will not occur.
Camera Movement Bob Intensity	Float	Defines the walk cycle intensity of the camera bobbing when moving.
Camera Landing Bob Intensity	Float	Defines the intensity of the camera bob when landing after being airborne.
Audio		
First Footstep Audio	Audio	The audio that is played in sequence of the walk cycle to resemble the initial footstep sound.
Second Footstep Audio	Audio	The audio that is played in sequence of the walk cycle to resemble the alternative footstep sound.
Jumping Audio	Audio	Sound that is played when jumping.
Landing Audio	Audio	Sound that is played when landing after being airborne.
Layers		
Non Jump Surface Layers	LayerMask	Used to define which layers of GameObjects the player cannot jump on.

Weapon Systems

This is pre-attached to the Weapon Space Offset GameObject of the Player Prefab and provides it with weapon related capabilities while allowing other vital systems to connect with it via the available fields.

Field Name / Heading	Data Type	Description

Character Controller	Character Controller	The Character Controller component of the player.
Camera Ray Spawn	Transform	Transform of the camera where the firing ray is projected in a forwards direction.
Weapon Space GameObject	GameObject	The GameObject that contains the spawned weapons when they are loaded.
Weapon Collection	WeaponCollection	The WeaponCollection ScriptableObject that is implemented to the FPS Player GameObject.
Blood Splatter Impact	Prefab	To instantiate at the Raycast impact when shooting a GameObject of the EffectFromPlayer Layer.
Button Names		
Fire Button Name	String	The name of the key or button in the Input Manager that is used to fire the wielded weapon if its Firing Type field is set to Semi Automatic.
Auto Fire Button Name	String	The name of the key or button in the Input Manager that is used to fire the wielded weapon if its Firing Type field is set to Automatic. Often set the same as Fire Button Name.
Reload Button Name	String	The name of the key or button in the Input Manager that reloads the wielded weapon.
Switch Button Name	String	The name of the key or button in the Input Manager that is used to switch the current weapon for the next.
Aim Button Name	String	The name of the key or button in the Input Manager used for aiming the wielded weapon.
Run Button Name	String	The name of the key or button in the Input Manager that is held down to enable running.

		Used with the input axes defined in the **Movement X Axis Name** and **Movement Y Axis Name** fields.
Selection Switch Button Names	Array	
Size	Integer	Number of **Element *** fields available for Strings.
Element * (Depends on array index)	Array of Strings	The name of the key or button in the Input Manager that selects a particular weapon to switch to.
Movement Axes		
Movement Y Axis Name	String	Name of the axis in the Input Manager used for tracking forwards and backwards Player GameObject movements.
Mouse Influence Axes		
Mouse X Influence Axis Name	String	Name of the axis in the Input Manager used for tracking horizontal mouse movements to effect weapon animations when rotating the player.
Mouse Y Influence Axis Name	String	Name of the axis in the Input Manager used for tracking vertical mouse movements to effect weapon animations when rotating the player.
Layer Names		
Fire Raycast Ignorable Layers	LayerMask	The layers of GameObjects that will be ignored by the firing raycasts.

Health

This component is designed to manage the health of GameObjects universally which includes player and non-player entities.

Field Name / Heading	Data Type	Description
Maximum Health	Float	The highest amount of health that can be held.
Minimum Health	Float	The lowest health value that can be reached.
Start Health	Float	Amount of health that the GameObject starts with.
Lives		
Has Lives	Tickbox (Boolean)	If ticked (true), allows different consequences to be executed depending on whether the GameObject possesses spare lives or not. A live is deducted for every death the GameObject endures. If unticked (false), lives will not be tracked or considered and any death has the same result.
Maximum Lives	Integer	**This field is displayed only when Has Lives is set to True.** The most number of lives the GameObject can own.
Start Lives	Integer	**This field is displayed only when Has Lives is set to True.** Amount of lives the entity starts with.
Events		
On Damage ()	UnityEvent	Calls the assigned methods when the

		GameObject's health is deducted.
On Death ()	UnityEvent	**This field is displayed only when Has Lives is set to False.** Calls the assigned methods when the GameObject's health reaches zero or less.
On Death With Lives ()	UnityEvent	**This field is displayed only when Has Lives is set to True.** Calls the assigned methods when the GameObject's health reaches zero or less and the entity has lives remaining. A live is taken after a death.
On Death Without Lives ()	UnityEvent	**This field is displayed only when Has Lives is set to True.** Calls the assigned methods when the GameObject's health reaches zero or less and the entity does not have lives remaining.

Properties

Name	Data Type	Accessors	Description
IsDead	Boolean	Read only	Is true if the HealthCount is zero or less.
HealthCount	Float	Read only	Amount of health possessed.
MaximumHealth	Float	Read only	Highest amount of health that can be reached.
MinimumHealth	Float	Read only	Lowest amount of health that can be reached.
LivesCount	Integer	Read only	Current number of lives in possession.

Methods

Return Type	Name & Signature	Description
Void	InflictDamage(float healthLoss, Collider hitCollider)	Used to apply an amount of damage to a Health component. The hitCollider parameter is used to determine if the affected GameObject has a damage multiplier to calculate.
Void	AddHealth(float amountToAdd)	Applies an amount of health to the recipient.
Void	AddLives(short amountToAdd)	Adds additional lives to the recipient.
Void	SetHealth(float health)	Defines the amount of health the component holds.
Void	ResetHealth()	Resets the HealthCount to its starting value.
Void	ResetHealthWithDelay(float delayInSeconds)	Resets the HealthCount to its starting value after a set duration in seconds.
Void	SetLives(int lives)	Defines the number of lives the component holds.
Void	ResetLives()	Resets the LivesCount to its starting value.
Void	ResetLivesWithDelay(float delayInSeconds)	Resets the LivesCount to its starting value after a set duration in seconds.
Void	DestroyGameObject(float secondsUntilDestruction)	Removes the GameObject from the scene after a duration of seconds. This should not normally be used on player or non-player entities.

Player Death

The Player Death component manages the player's death and activates the relevant functionality.

Field Name / Heading	Data Type	Description
Character GameObject	GameObject	The Player GameObject.
Environment Camera GameObject	GameObject	Environment Camera of the Player GameObject.
Environment Collider	Capsule Collider	Environment Collider of the Player GameObject.
Weapon Camera	Camera	Camera of the player's Weapon Camera.
Weapon Systems	WeaponSystems	WeaponSystems in the player's Weapon Space Offset.
Weapon Space Animator	Animator	Player's Weapon Space animator.
Deathbody GameObject	GameObject	The GameObject that behaves like the player's dead body upon death.
Player Respawn	PlayerRespawn	Player Respawn component from the Player Respawn GameObject.
Respawn Delay Duration	Float	Number of seconds until the player can respawn after dying.

Properties

Name	Data Type	Accessors	Description

IsDead	Boolean	Read only	Is true if EnableDeath has been called.

Methods

Return Type	Name & Signature	Description
Void	EnableDeath()	Activates the Deathbody, despawns the player and alters the UI.
Void	DisableDeath()	Disables the Deathbody and re-enables the player.

Player HUD

The component controls the player's Heads-Up Display (HUD) where information such as health, ammo and objectives are presented during gameplay.

Field Name / Heading	Data Type	Description
HUD Game Object	GameObject	The HUD GameObject of the Player UI.
Crosshair Space	Image	An area of the HUD for the crosshair to be displayed.
Ammo Icon Space	Image	An area of the HUD for showing the ammo icon of the current weapon.
Weapon Ammo Count Space	Text	Text element of the current weapon capacity.
Total Ammo Count Space	Text	Text element of the current amount of ammo possessed by the player in total.

Health Text	Text	Text element of the current amount of player health.
Low Health Overlay	Image	The tinting image becomes more opaque as the player's health gets lower.
Damage Overlay	Image	The covering image that is briefly visible when the player receives damage.
Max Damage Overlay Opacity	Float Range	Highest overlay opacity to be displayed when the player is damaged.
Damage Overlay Fade Duration	Float Range	Duration in seconds from the damage overlay being shown to it completely fading away.
Character Weapon Systems	WeaponSystems	Weapon Systems used by the player.
Character Health	Health	The Health component used by the player.

Methods

Return Type	Name & Signature	Description
Void	PlayDamageEffects()	Causes the damage overlay to flash.

Player Respawn

This deals with respawning the Player GameObject after the player has died and a respawn is granted.

Field Name / Heading	Data Type	Description

Character GameObject	GameObject	GameObject of the player.
Character Environment Camera	Transform	Transform of the Environment Camera in the Player GameObject.
Respawn Locations		
Respawn Points	Array	Array of Transforms resembling respawning locations.
Size	Integer	Size of the array.
Element *	Array of Transforms	Transform of the respawn point.
Position Offsets		
Will Spawn On Ground	Tickbox (Boolean)	If ticked (true), will attempt to place the Player GameObject on the surface below the respawn point when spawning the player.
Pivot To Bottom Height	Float	**This field is displayed only when Will Spawn On Ground is set to True.** Distance between the pivot of the player and the lowest point of the GameObject mesh. This works with placing the player on the below surface.
Randomised Horizontal Radius Offset	Float	The horizontal distance from the selected respawn location that the Player GameObject can spawn somewhere within.
Randomised Vertical Upwards Offset	Float	**This field is displayed only when Will Spawn On Ground is set to False.** The maximum height from the respawn location

		that Player GameObject can randomly spawn within.
Randomised Y Rotation Offset	Float Range	The most degrees of yaw rotation in either clockwise or anticlockwise that the Player GameObject could face from a random offset during spawning.
Events		
On Spawn ()	UnityEvent	Calls the assigned function once the player spawns. This is often used to reset the player's health.

Methods

Return Type	Name & Signature	Description
Void	RespawnAtNextPoint()	Spawn the GameObject at the next respawn point which is determined by the Respawn Point Order.
Void	RespawnAtSpecificPoint(short respawnPointIndex)	Spawn the GameObject at a specific respawn point by its index.
Void	ChangeRespawnPoints(Transform[] newRespawnPoints)	Change the respawn points held by this component through code.

Collectable

This component is designed to be applied to GameObjects that simulate collectables. When in contact with the Player GameObject, the collectable will despawn while providing something to the player.

Field Name / Heading	Data Type	Description

Collection Type	Options	Defines the purpose of the collectable with the options Weapon, Ammo, Health and Lives. If set to Weapon, this collectable can enable or disable the equipping of a weapon and affect the amounts of total ammo gained. Designed for weapon pick-ups. If set to Ammo, the collectable can only affect the total ammo possessed. Designed for ammo box pick-ups. If set to Health, the collectable can provide health to the player. If set to Lives, it can grant additional lives to the player.
Weapon	Weapon ScriptableObject	**This field is displayed only when Collection Type is set to Weapon.** The weapon that will be affected by the collection.
Enable	Tickbox (Boolean)	**This field is displayed only when Collection Type is set to Weapon.** Defines whether the weapon should be enabled or disabled on collection if not already in such status. If ticked (true), enable the specified weapon. If unticked (false), disable the specified weapon.
Ammo In Weapon	Integer	**This field is displayed only when Collection Type is set to Weapon.** Amount of ammo pre-loaded into the newly enabled weapon on collection. If this value exceeds its Capacity field, the value will be capped.
Ammo	Ammo ScriptableObject	**This field is displayed only when Collection Type is set to Ammo.**

		The ammo that will be affected by the collection.
Add To Ammo Total	Integer	**This field is displayed only when Collection Type is set to Weapon or Ammo.** Amount of ammo to be added to the total ammo in possession.
Health To Add	Float	**This field is displayed only when Collection Type is set to Health.** Amount of health given to the player on pick-up.
Lives To Add	Integer	**This field is displayed only when Collection Type is set to Lives.** Amount of extra lives granted to the player on pick-up.
Despawn Type	Options	Determines how the collectable will despawn with the options Disable and Destroy. If set to Disable, the collectable GameObject will be disabled on pick-up, which is effective for respawning. If set to Destroy, the collectable will be removed from the scene instead.
After Collection Object	Prefab	The GameObject that will be temporarily spawned once the collectable is collected.
After Collection Despawn Time	Float	Time in seconds until the After Collection Object despawns after spawning.

General Respawn

The component enables the respawning of general GameObjects that are not player or non-player related. This is often a solution for respawning collectables.

Field Name / Heading	Data Type	Description
Respawn Type	Options	Defines how the GameObject is respawned with the options Reset or Instantiate. If set to Reset, the GameObject to respawn is re-enabled. If set to Instantiate, a new GameObject is created from a Prefab during respawning.
GameObject To Enable	Array	**This field is displayed only when Respawn Type is set to Reset.** The GameObjects to respawned through re-enabling them.
Size	Integer	Size of the array.
Element *	Array of GameObject	A GameObject that can be spawned.
GameObject To Spawn	Prefab	**This field is displayed only when Respawn Type is set to Instantiate.** The Prefab that will be used to instantiate a new GameObject during spawning.
Will Spawn On Start	Tickbox (Boolean)	Defines if a NonPlayer GameObject is spawned as the scene begins or this component is enabled.
Respawn Locations		
Respawn Points	Array	Transforms of the respawn locations.
Size	Integer	Size of the array.
Element *	Array of Transforms	The Transform belonging to a respawn point that GameObjects can be spawned from.

Respawn Point Order	Options	**This field is displayed only when Size (of the Respawn Points array) is set to more than 1.** Determines how the respawn point is selected to spawn the next GameObject. Has the options Random or Sequential. If set to Random, the respawn point that spawns the GameObject is chosen at random. If set to Sequential, the respawn point that spawns the GameObject is chosen in a consistent order.
Positioning Offsets		
Will Spawn On Ground	Tickbox (Boolean)	Aims to place the GameObject on the surface below the respawn point when spawning.
Pivot To Bottom Height	Float	**This field is displayed only when Will Spawn On Ground is set to True.** Distance between the pivot and the lowest point of the GameObject mesh. This works with placing the object on the below surface.
Randomised Horizontal Radius Offset	Float	The horizontal distance from the selected respawn location that the GameObject can spawn somewhere within.
Randomised Vertical Upwards Offset	Float	**This field is displayed only when Will Spawn On Ground is set to False.** The maximum height from the respawn location that the GameObject can randomly spawn within.
Randomised Y Rotation Offset	Float Range	The most degrees of yaw rotation is either clockwise or anticlockwise that the GameObject could face from a random offset during spawning.

Events		
On Spawn ()	UnityEvent	Calls the assigned functions once the object spawns.
Automatic Respawning		
Will Auto Respawn	Tickbox (Boolean)	Defines that the non-player will respawn automatically and consistently after a duration of time.
Maximum Spawned GameObject In Existence	Integer	**This field is displayed only when Respawn Type is set to Instantiate.** **And when Will Auto Respawn is set to True.** States the maximum number of spawned GameObjects that need to exist at once. Thus pause spawning if this quantity is reached.
Auto Respawn Delay In Seconds	Float	**This field is displayed only when Will Auto Respawn is set to True.** The amount of time in seconds until the GameObject is spawned after activation of this component or from the last spawn. Objects will keep spawning until all members of the GameObject array are enabled.

Methods

Return Type	**Name & Signature**	**Description**
Void	DelayedRespawnAtNextPoint(float delayInSeconds)	Spawns a GameObject at the next respawn point, depending on the state of the Respawn Point Order field, after a duration of seconds.
Void	DelayedRespawnAtSpecificPoint(short	Spawns a GameObject at the specific respawn

	respawnPointIndex, float delayInSeconds)	point by index, after a duration of seconds.

Projectile

The Projectile component is designed to be used on projectile Prefabs and provides features to create all kinds of projectile behaviours.

Field Name / Heading	Data Type	Description
Collision Action	Options	Determines how the projectile behaves upon impacting a collider with the options Despawn, Drop and Stick. If set to Despawn, the projectile will despawn on collision which is useful for explosive projectiles. If set to Drop, it will continue being affected by gravity after the collision, thus will fall downwards. If set to Stick, it will attach itself to the impacted GameObject which is useful for replicating the characteristics of arrows.
Lifetimes		
Is Post Collision Lifetime Enabled	Tickbox (Boolean)	**This field is displayed only when Collision Action is set to Drop or Stick.** If ticked (true), the projectile's lifetime is split into before and after collision lifetime periods. Therefore the projectile's lifetime will restart from the Post Collision Lifetime figure upon collision. If unticked (false), the current lifetime count is maintained after collision.
Whole Lifetime	Float	**This field is displayed only when Is Post Collision Lifetime Enabled is set to False.**

		The projectile's existence time from spawning to despawning in seconds.
Pre Collision Lifetime	Float	**This field is displayed only when Is Post Collision Lifetime Enabled is set to True.** The projectile's existence time from spawning to despawning in seconds, if it does not collide with anything during that time.
Post Collision Lifetime	Float	**This field is displayed only when Is Post Collision Lifetime Enabled is set to True.** The projectile's existence time from the first collision to despawning in seconds.
Detonation		
Will Detonate On Collision	Tickbox (Boolean)	If ticked (true), the Prefab assigned to GameObject To Spawn On Detonation will be instantiated when the projectile collides with something. If unticked (false), no instantiations will occur from a projectile collision.
Will Detonate On Lifetime End	Tickbox (Boolean)	If ticked (true), the Prefab assigned to GameObject To Spawn On Detonation will be instantiated once the projectile's lifetime expires. If unticked (false), no instantiations will occur from the expiry of the projectile's lifetime.
GameObject To Spawn On Detonation	Prefab	**This field is displayed only when Will Detonate On Collision or Will Detonate On Lifetime End is set to True.** The prefab that will be spawned when the projectile's detonation is triggered.
Is Detonation GameObject Lifetime	Tickbox (Boolean)	**This field is displayed only when Will Detonate On Collision or Will Detonate On**

Limited		**Lifetime End is set to True.** If ticked (true), the GameObject spawned during detonation will despawn after a period of time. If unticked (false), the detonation GameObject will not despawn over time.
Detonation GameObject Lifetime	Float	**This field is displayed only when Is Detonation GameObject Lifetime Limited is set to True.** The amount of time in seconds until the detonation GameObject despawns.
Effects		
Damage On First Collision	Float	Amount of damage applied to Health components of GameObjects that have been directly hit by the projectile.
Maximum Detonation Area Damage	Float	Maximum amount of damage applied to Health components of GameObjects during detonation, before any multipliers are involved.
Detonation Area Radius	Float	Radius of the detonation area where GameObjects can be affected if within.
Detonation Explosion Force	Float	The degree of outwards force that is applied to in-range Rigidbody GameObjects.
Events		
On Spawn ()	UnityEvent	Calls the assigned method when the projectile is spawned.
On Collision ()	UnityEvent	Calls the assigned method when the projectile collides with something.
On Lifetime End ()	UnityEvent	Calls the assigned method when the projectile's

		lifetime expires.

Properties

Name	Data Type	Accessors	Description
RigidbodyExplosionUpwardsModifier	Float	Read/Write	The modification of upwards force to be applied to affected Rigidbody GameObjects upon an explosive effect.

Non Player Controller

This component coordinates the movement of the NonPlayer GameObject through guiding its navigation AI and manages the character's animations.

Field Name / Heading	Data Type	Description
Character Damage Infliction	NonPlayerDamageInfliction	The Non Player Damage Infliction component used by the non-player entity.
Character Player Contact Detection	ContactDetectionWithPlayer	The Contact Detection With Player component used by the non-player entity.
Target Player Transform	Transform	Transform of the player GameObject
Target Player Death	Player Death	The Player Death component of the player.
Target Environment Collider	Capsule Collider	Capsule Collider component of the player's Environment Collider.
Agent Off Mesh Link Speed	Float	The speed the non-player moves when passing over a Off Mesh Link, featured within the AI navigation system.

Non Player Audio

NonPlayerAudio manages the audio of the NonPlayer character and influences the variability of the audio in relation to pitch and timing.

Field Name / Heading	Data Type	Description
Periodic Audio Clip	AudioClip	The audio clip that is played in random intervals.
Damage Audio Clip	AudioClip	The audio clip that is played when damaged.
Death Audio Clip	AudioClip	The audio clip that is played during death.
Maximum Pitch	Float	Highest pitch that the audio clips can be played at.
Minimum Pitch	Float	Lowest pitch that the audio clips can be played at.

Methods

Return Type	Name & Signature	Description
Void	PlayDamageAudio()	Play the audio clip defined in Damage Audio Clip.
Void	PlayDeathAudio()	Play the audio clip defined in Death Audio Clip.
Void	PlayPeriodicAudio()	Play the audio clip defined in Periodic Audio Clip.
Void	PlayAudio(AudioClip audioClip)	Play the chosen audio clip.

Non Player Damage Infliction

This component is applied within non-player GameObjects where a collider with Is Trigger set to true is present and allows them to cause damage onto the player.

The frequency of the infliction is controlled by an Event defined in the non-player hit animation.

Field Name / Heading	Data Type	Description
Non Player Animator	Animator	The Animator of the parent GameObject.
Non Player Health	NonPlayerHealth	The NonPlayerHealth of the parent GameObject.
Target Player Environment Collider	Collider	For the player's Environment Collider.
Target Player Health	PlayerHealth	For the Player Health of the player GameObject.
Damage Health Loss	Float	The amount of damage that is inflicted onto the target player per attack.
Blood Splatter	Prefab	Prefab containing blood splatter particle effects that are generated during damage infliction.

Properties

Name	Data Type	Accessors	Description
BloodSplatterLifetime	Float	Read/Write	Duration in seconds until the blood splatter GameObject is destroyed.
IsDamaging	Boolean	Read only	Is currently in the process of damaging the player.

Methods

Return Type	Name & Signature	Description
Void	InflictDamageToTarget()	Apply the defined damage to the target player's health. Normally called by animation clips via an Event Array Relay component.

Non Player Death

The Non Player Death component manages the non-player entity during its death.

Field Name / Heading	Data Type	Description
Character GameObject	GameObject	The GameObject of the related non-player character.
Damage Infliction GameObject	GameObject	The non-player GameObject's Damage Infliction GameObject.
Despawn After Death Delay	Float	Amount of time in seconds from the non-player entity's death to its despawning. This delay allows death animations to play.
GameObjects To Disable On Death	Array	GameObjects to disable once the non-player character has died.
Size	Integer	Size of the array.
Element *	Array of GameObjects	A GameObject to disable upon the non-player's death. These will be re-enabled once it is respawned.
GameObjects To Disable On Despawn	Array	GameObjects to disable once the non-player character has despawned after its death.
Size	Integer	Size of the array.
Element *	Array of GameObjects	A GameObject to disable when the non-player despawns after dying. These will be re-enabled once it is respawned.

Properties

Name	Data Type	Accessors	Description
IsDead	Boolean	Read only	Is true if the non-player's EnableDeath() method has been invoked.
IsDespawned Dead	Boolean	Read only	After the Despawn After Death Delay has passed, the non-player is despawned.

Methods

Return Type	Name & Signature	Description
Void	EnableDeath()	Activates the death animation and disables certain components.
Void	DisableDeath()	Re-enables the GameObject's components after respawning.

Non Player Respawn

This component controls the respawning process of the non-player entity.

Field Name / Heading	Data Type	Description
Will Spawn On Start	Tickbox (Boolean)	Defines if a NonPlayer GameObject is spawned as the scene begins or this component is enabled.
NonPlayer Characters	Array	The GameObjects of the non-player entities that can be spawned.
Size	Integer	Size of the array.
Element *	Array of	A non-player GameObject that can be spawned.

	GameObjects	
Respawn Locations		
Respawn Points	Array	Transforms of the respawn locations.
Size	Integer	Size of the array.
Element *	Array of Transforms	The Transform belonging to a respawn point that entities can be spawned from.
Positioning Offsets		
Will Spawn On Ground	Tickbox (Boolean)	Aims to place the non-player GameObject on the surface below the respawn point when spawning.
Pivot To Bottom Height	Float	**This field is displayed only when Will Spawn On Ground is set to True.** Distance between the pivot of the non-player and the lowest point of the GameObject mesh. This works with placing the non-player character on the below surface.
Randomised Horizontal Radius Offset	Float	The horizontal distance from the selected respawn location that the non-player GameObject can spawn somewhere within.
Randomised Vertical Upwards Offset	Float	**This field is displayed only when Will Spawn On Ground is set to False.** The maximum height from the respawn location that non-player GameObjects can randomly spawn within.
Randomised Y Rotation Offset	Float Range	The most degrees of yaw rotation in either clockwise or anticlockwise that non-player GameObjects could face from a random offset during spawning.

Events		
On Spawn ()	UnityEvent	Calls the assigned function once the non-player entity spawns.
Automatic Respawning		
Will Auto Respawn	Tickbox (Boolean)	Defines that the non-player will respawn automatically and consistently after a duration of time.
Auto Respawn Delay In Seconds	Integer	**This field is displayed only when Will Auto Respawn is set to True.** The amount of time in seconds until the non-player character is spawned after activation of this component or from the last spawn. Entities will keep generating until all members of the NonPlayer Characters array are enabled.

Trigger Action

This component invokes functions assigned to its UnityEvent fields when a specified GameObject intersects with the collider of its own.

Field Name / Heading	**Data Type**	**Description**
Target Colliders	Array	The colliders that can involve the assigned On Trigger UnityEvents upon intersecting.
Size	Integer	Size of the array.
Element *	Array of Colliders	One of the listed colliders that can trigger the On Trigger UnityEvents.
On Trigger Enter ()	UnityEvent	Calls the assigned functions once every time a certain collider enters the trigger of the

		component's GameObject.
On Trigger Exit ()	UnityEvent	Calls the assigned functions once every time a certain collider leaves the trigger of the component's GameObject.
On Trigger Stay ()	UnityEvent	Calls the assigned functions for every frame that the target collider remains within the trigger of the component's GameObject.

Moveable

Can be applied to non-static GameObjects to allow them to move between their target and current positions.

Field Name / Heading	Data Type	Description
Target Position	Vector3	The position that the GameObject will move towards when invoked.
Transition Duration	Float	Amount to time in seconds for the GameObject to move from current to target positions and vice versa.

Properties

Name	Data Type	Accessors	Description
IsMovingTowardsTarget	Boolean	Read/Write	Is true if the moveable GameObject is transiting between the two positions. Is set to true to begin moving towards the target position. Becomes false again once it reaches its destination.

Boundary Manager

This component outlines the boundaries of the scene where player or non-player entities will be killed if they exit such limits. Additionally, GameObjects with Rigidbody components will be destroyed if they cross the boundaries.

Field Name / Heading	Data Type	Description
Centre	Vector3	States the central position of the boundary box.
Dimensions	Vector3	Represents the size of the boundary box on each axis.

Properties

Name	Data Type	Accessors	Description
PlayerGameObjectsTag	String	Read/Write	Tag name used for player GameObjects. Set to “Player” by default.
DeathBodyGameObjectsTag	String	Read/Write	Tag name used for player’s Deathbody. Set to “DeathBody” by default.
LooseGameObjectsTag	String	Read/Write	Tag name used for general Rigidbody GameObjects. Set to “Loose” by default.
GameObjectPositionCheckPerFrame	Integer	Read/Write	Determines the number of times per frame a GameObject that has crossed the boundary can be removed or managed. The higher this number, the more responsive this system will behave, but more overhead will occur. By default this is set to 1.

Total Ammo Manager

This allows the Start Amount of specific ammo types to be redefined for the scene that the component is involved in.

Field Name / Heading	Data Type	Description
Character Weapon System	WeaponSystems	WeaponSystems of the Player GameObject.
Will Ammo Counts Set On Start	Tickbox (Boolean)	If ticked (true), the entries will be applied to the player's total ammo counts. If unticked (false), no changes to the ammo counts will occur.
Size	Integer	The number of ammo types to adjust.
Leftside	Ammo	Type of ammo to edit.
Rightside	Integer	Quantity of the related ammo to apply when invoked or at the scene's beginning.

Methods

Return Type	Name & Signature	Description
Void	SetTotalAmmoCounts()	Applied the listed ammo amounts to the total ammo counts.

Collider Damage Multiplier

The Collider Damage Multiplier component can be added to GameObjects of non-player entities which are under the EffectFromPlayer layer and possess a collider.

Any attack from the player hitting the subject object will deal damage with an adjustable multiplier, thus useful to express weak points such as heads.

Field Name / Heading	Data Type	Description
Damage Multiplication	Float	Damage multiplier for the damage of the player's attack which hit the attached GameObject.

Objective Display

Controls the objective's details that are displayed in the HUD.

Field Name / Heading	Data Type	Description
Objective Text	Text	Text UI element of the Objective Display.
Objective Text Background	Image	Background UI element of the Objective Display.
Available Message Colours	Array	Lists the colours that can be applied to the message text of the displayed objectives.
Size	Integer	Size of the array.
Element *	Array of Colours	Available hues for recolouring the text.

Methods

Return Type	Name & Signature	Description
Void	SetMessageText(string text)	Sets the message of the display that titles the current objectives.
Void	SetMessageColour(int availableColorIndex)	Recolours the message's text from the list of Available Message Colours that are referenced by index.

Void	SetAreCountsDisplayed(bool areCountsDisplayed)	Toggles the showing of the message's preceding count figures and parentheses.
Void	SetCurrentCompletionCount(int count)	Sets the first count value that is for numbering the current progress of the objective.
Void	SetTargetCompletionCount(int count)	Sets the second count value which outlines the quantity of tasks that require satisfying to complete the objective.

Level Selection Organiser

The component is used within the Main Menu scene to allow customisation of the Level Selection Menu without requiring manual editing.

This populates the buttons in the Level Selection Menu and the UI elements of the Level Info Menu.

Field Name / Heading	Data Type	Description
Level Buttons	Array	Buttons that make up the Level Selection Menu which can be initialised with data.
Size	Integer	Size of the array.
Element *	Array of Buttons	A button object that can be populated with data.
Info Menu Title	Text	The text element representing the title of the Level Info Menu.
Info Menu Description	Text	The text element for the Level Info Menu's description.
Info Menu Start Button	Button	Button that starts the level from the Level Info Menu.

	Reorderable List	For organising and initialising the Level Selection Menu buttons.
=	Drag Handle	Allows the element to be reordered amongst other elements through dragging. The position of the element defines the order of the buttons in the menu from left-to-right, top-to-bottom.
Visible	Tickbox (Boolean)	If ticked (true), the details entered will be applied to a button. If unticked (false), such details will be skipped and the data of the proceeding element will be applied instead.
Scene	Integer	The index of the scene that will be accessed when clicking the start button from the Level Info Menu. The scene index can be viewed from the Scenes In Build section of the Build Settings.
Title	String	Displayed name of the level that is applied to the level selection button and info menu heading.
Sprite	Sprite	Image that is shown on the level selection button.
Description	String (Area)	The information about the level that is displayed on the Level Info Menu.
+	Add element	Adds another element to the reorderable list.
-	Remove element	Removes the select element from the reorderable list.

Methods

Return Type	Name & Signature	Description

Void	ChangeScene()	Start the scene that has been selected in the Level Selection Menu.

List of Other Components

Other components are involved to provide functionality to features of the FPS Integration Tool and its Prefabs. They are generally quite simple and their intended use can be summarised. Such components include:

Application State Events - Used for changing and restarting scenes and closing the application.

Assignable Respawn Points - Used for changing the respawn points.

Boss Shield - Functionality for the NonPlayer Boss' shield.

Contact Detection With Player - Prevents non-player entities from pushing the player when near.

Cursor Manager - Controls how the cursor behaves regarding locking and its visibility state.

Death Menu - Determines the display of the Death Menu. Requires the Player Death component to be assigned to the Player Death field.

End Game Menu - Is the parent class of the Fail and Success Menu.

Event Array Relay - Used to hold functions in UnityEvent fields that are called from animation clip events.

Fail Menu - Controls when the Fail Menu is shown.

Gameplay Data Display - Updates the UI elements displayed in the menus using values from the Gameplay Data Manager.

Gameplay Data Manager - Tracks the player's kills, deaths and passed time during the gameplay of a scene.

General UI - Manages the displaying of menus in the Main Menu scene.

Layer Collision Manager - Stored in a scene's Managers GameObject and alters the Layer Collision Matrix to allow collisions to behave correctly.

Non Player Kill Counter - Tracks the number of non-player characters killed during the scene's runtime.

Pause Menu - Defines when the Pause Menu is displayed and the time scale of the gameplay.

Respawn - The parent class for the Player, Non Player and General Respawn components.

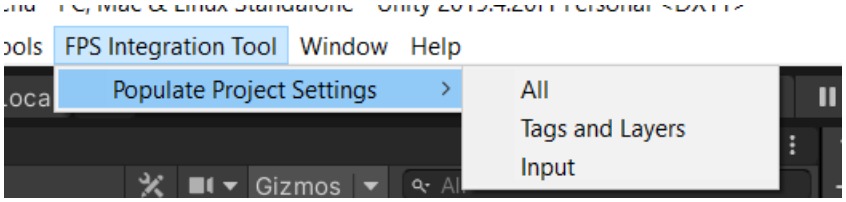
Success Menu - Controls when the Success Menu is shown.

Target Fall Counter - Tracks the number of shooting range targets that have fallen.

With this tool, editor scripts are utilised to implant added UI functionality to the components, while code files under the namespace Utilities allow complex systems to be separate from the components. Such scripts are not designed to be edited by the user.

Toolbar Options

The toolbar of Unity's application window is used by the FPS Integration Tool to provide important capabilities in a very accessible manner.



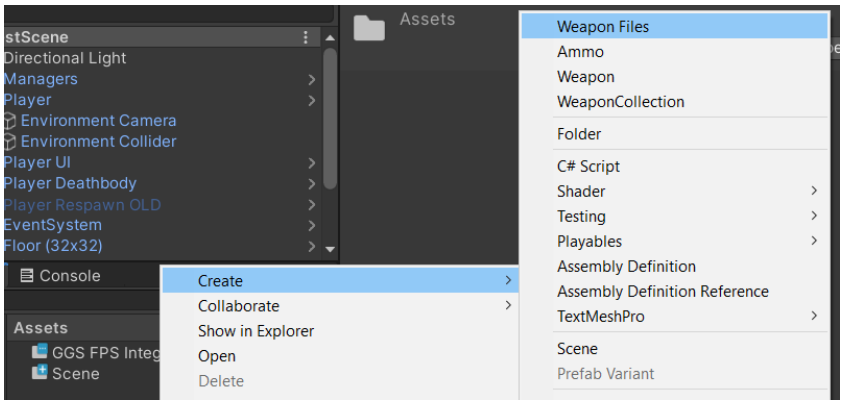
The structure and explanation of these options are shown:

- **FPS Integration Tool**
    - **Populate Project Settings** - For applying changes to the Project Settings that allows the tool's content to work correctly.
        - **All** - Changes both the Tags and Layers and Input sections of the Project Settings.
        - **Tags and Layers** - Updates the Tags and Layers settings.
        - **Input** - Adjusts the Input settings as needed.

Create Options

The Create Options allow ScriptableObjects and file systems to be easily created.

Access these options by right-clicking within a Project tab, hover over Create then select one of the options towards the top of the right-click menu.



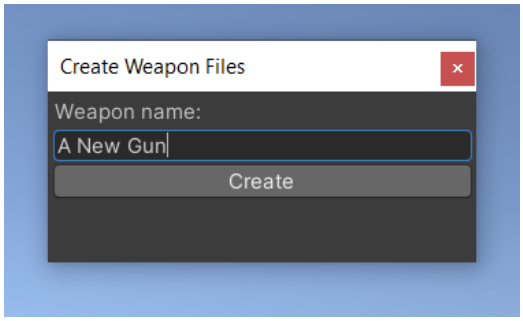
Weapon, WeaponCollection and Ammo ScriptableObjects can be created in this way.

Weapon Files

The Weapon Files option works differently compared to the individual ScriptableObject creation, it generates all the files that the new weapon depends on and links them together automatically.

This is achieved by this system duplicating content from the Default Weapon folder while adjusting the naming, similar to templating.

After selecting Weapon Files within the Create Options, a window menu should appear over the Scene tab titled Create Weapon Files with the field labelled **Weapon name**.



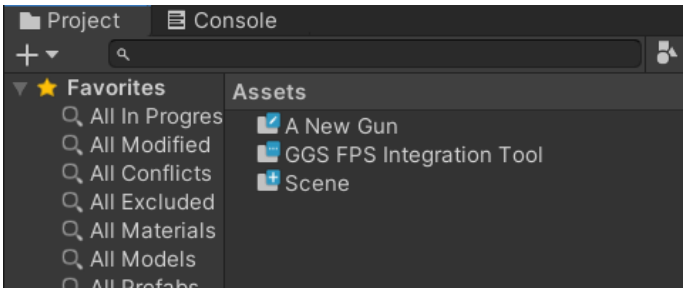
This window can be closed by clicking the usual window close icon.

Enter the name of your weapon and ensure the start and end characters of the string are not spaces.

This should reveal the Create button below the field, which if pressed will generate the weapon's files.

This will create a folder of the given name which contains these files:

- Animations folder - Contains the weapon's animations used in Base and Aim Influence Layers.
- The weapon's ScriptableObject
- The weapon's Animator Controller
- The weapon's Prefab



Layers (GameObject)

Layers are designed to easily categorise GameObjects and can be defined via Tags and layers in the Project Settings tab.

The FPS Integration Tool uses these layers:

Builtin Layer 0: **Default** - Used in the standard scenario.

Builtin Layer 5: **UI** - Used by Canvases and their child GameObjects.

User Layer 8: **FirstPerson** - For culling masks within the Environment Camera and Weapon Camera and is used on weapon Prefabs.

User Layer 9: **ThirdPerson** - Is applied to the Player Prefab and is used for the collisions with NonPlayer and EffectToPlayer.

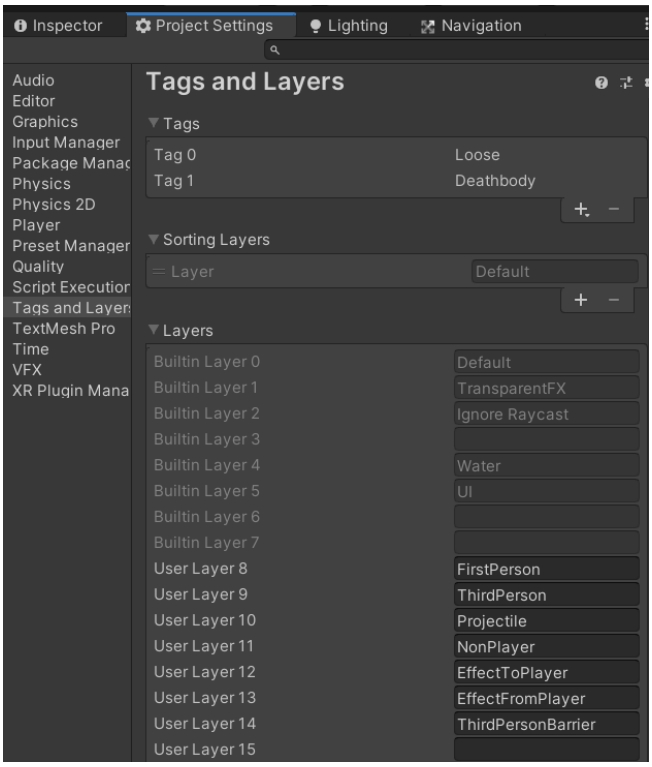
User Layer 10: **Projectile** - For preventing projectiles from colliding with the Player GameObject and other projectiles.

User Layer 11: **NonPlayer** - Used for simulating contact between non-player entities and the player.

User Layer 12: **EffectToPlayer** - For the colliders with Is Trigger enabled that deal damage to the player using the Non Player Damage Infliction component.

User Layer 13: **EffectFromPlayer** - For the closely fitted colliders of non-player GameObjects which detects raycasts from the player to simulate damage.

User Layer 14: **ThirdPersonBarrier** - This is used to block the player from accessing restricted areas while allowing firing rays, projectiles, non-player entities and other GameObjects to pass through as normal.



Defining the User Layer names in different fields from the ones described above may cause incorrect Layers being assigned to GameObjects and Prefabs, leading to issues with the gameplay.

To correct this malfunction through using the specified User Layer fields:

1. Delete the GGS FPS Integration Tool folder.
2. Apply User Layer names via the FPS Integration Tool > Populate Project Settings > All option.
3. Finally, reimport the GGS FPS Integration Tool folder by importing the package again.

If the intended User Layer fields are already taken by other layers from your project, manually changing the layers of the affected GameObjects and Prefabs would be the next best solution.

Intended use of the FPS Integration Tool

There are some conditions to consider when using this tool:

- This tool can be used in commercial and non-commercial projects so long as the product of such project is Built (where the project hosted in the Unity application is exported as an application itself known as a Build) from the Unity application.
- The tool (partly or wholly) cannot be distributed or used for creating other tools that would be published (whether it is via the Unity Asset Store or not).
- The tool should be credited within the credits (or with the listed contributors of the project) of the Built application, where the sentence '**Made with the Grey Gear Systems FPS Integration Tool. Available on the Unity Asset Store**' is displayed.
- Promoting the use of the FPS Integration Tool with your product (such as on websites) is recommended and greatly appreciated. The sentence '**Made with the Grey Gear Systems FPS Integration Tool. Available on the Unity Asset Store**' can also be used for this case.
- Writing an honest review of this tool on the Unity Asset Store would also be fantastic!

Thank you for cooperating and using the FPS Integration Tool!