Unity Game Templates

Table of Contents

Introduction

The 2D side scroller game template is created on Unity and is implemented specially for non-programmers. The key aim of producing premade Unity game templates is to assist students in understanding the mechanism and concepts of game design with little to no knowledge of programming languages like C#. With these game templates, we hope to eliminate the coding aspect of the game design and instead focus on the user interface and object template aspect of it, to increase the simplicity and usability of the game template. This would translate to a unique game development experience where the user would be able to utilize a drag and drop approach for game development. The development will be based entirely on the user learning and experience, and for the editor to be used in the working industry in the future.

This 2D platform game template is made by using the perspective of someone who has never used unity before and to teach them every aspect of how this game should be played.

Every detail on how to utilise this game is mentioned below in full detail.

What is Unity?

Unity is a completely free gaming engine to develop video games for web plugins, desktop platforms, consoles and mobile devices. The newly implemented 2D side Scroller game template is played and can be further designed on the Unity platform.

It has always been problematic to learn the basics of game design without learning the actual coding involved with it like C#. When students are more focused on the programming aspect of game design, they tend to lose focus and interest in the creative aspect of it. This problem persists when a first-time user wants to understand unity as a platform but falls short of creating any functional games because of the lack of strong programming skills.

With this project, we aim to eliminate the programming aspect of game design and instead make this process simpler for non-programmers. This is performed through pre-packaging all the function scripts into “templates”. All that the user will need to worry about is assigning the game assets to those functions. This would lead to a better interactive and knowledgeable experience for the user, where they would be able to have hands-on experience in game development and design.

System Requirements

- ★ No less than 64 MB RAM memory
- ★ 610 MB free hard drive space for installation
- ★ Windows® 7 64-bit / Windows® 8 64-bit / Windows® 8.1 64-bit / Windows® 10 64-bit
  or
  Mac computer running minimum macOS 10.12.6 and Xcode 9.4 or higher.
- ★ keyboard, mouse
- ★ Unity (Version 2021.1.16+)

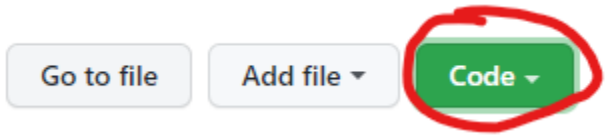
Getting Started

Downloading the Files

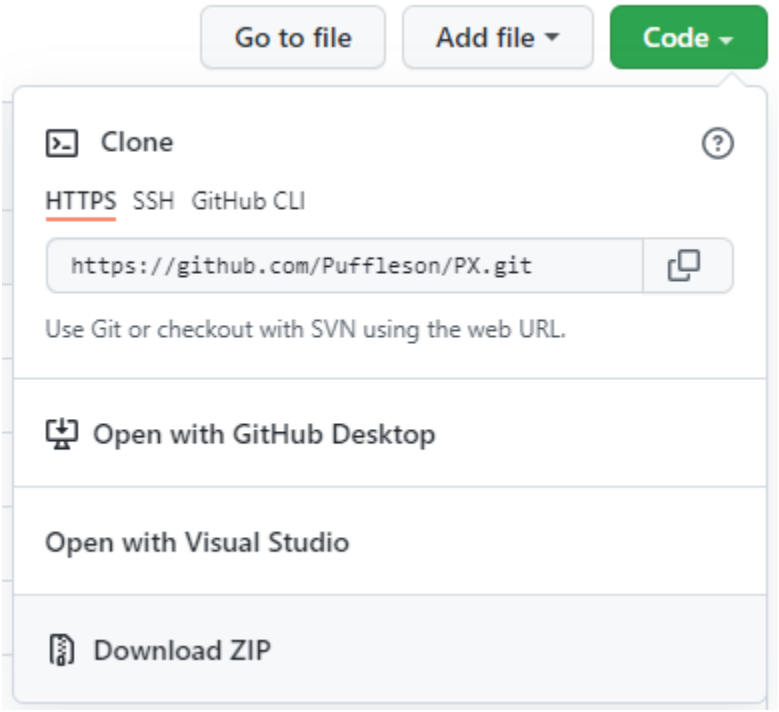
For the entire Unity Project, go to:

- **https://github.com/Puffleson/PX-Unity-Game-Templates**

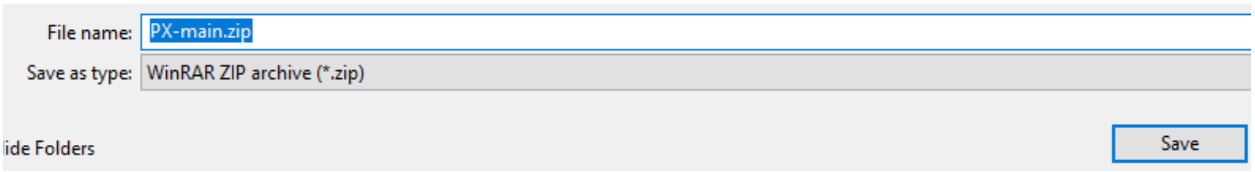
Click on **Code** (the green button)



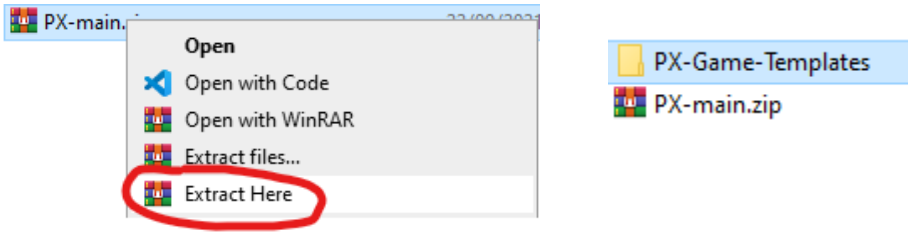
It will expand a dropdown menu. Click on **Download Zip**



Save it to any location. The zip file is around **539 MB** on disk in size.

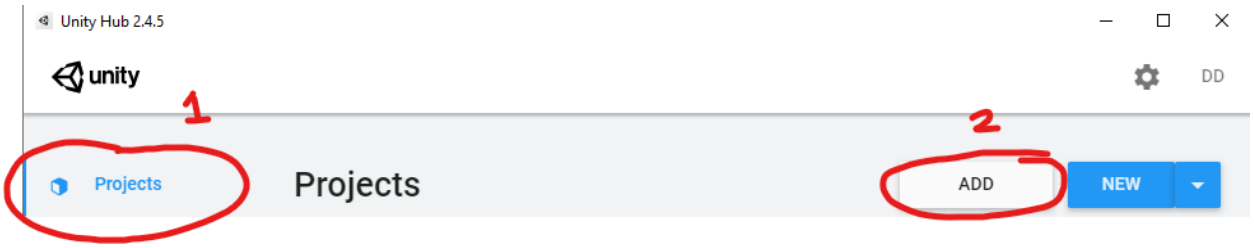


Right click the downloaded zip file and click **Extract here**.

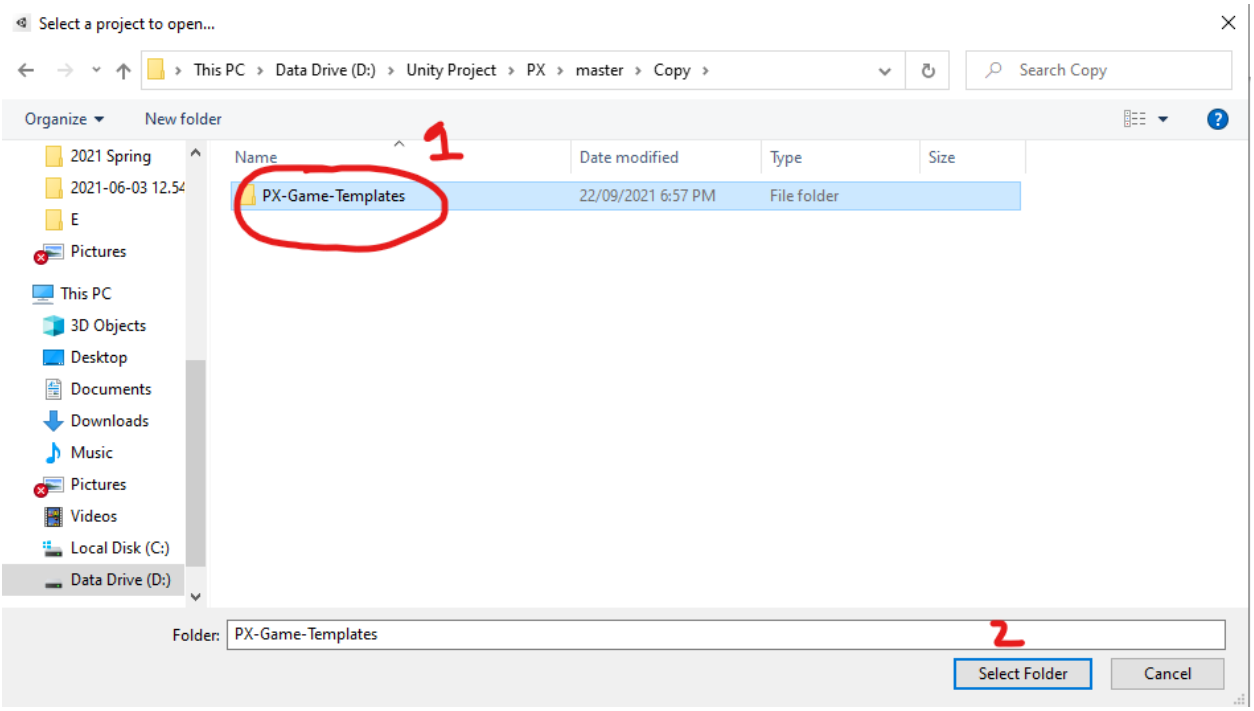


Adding the project to Unity

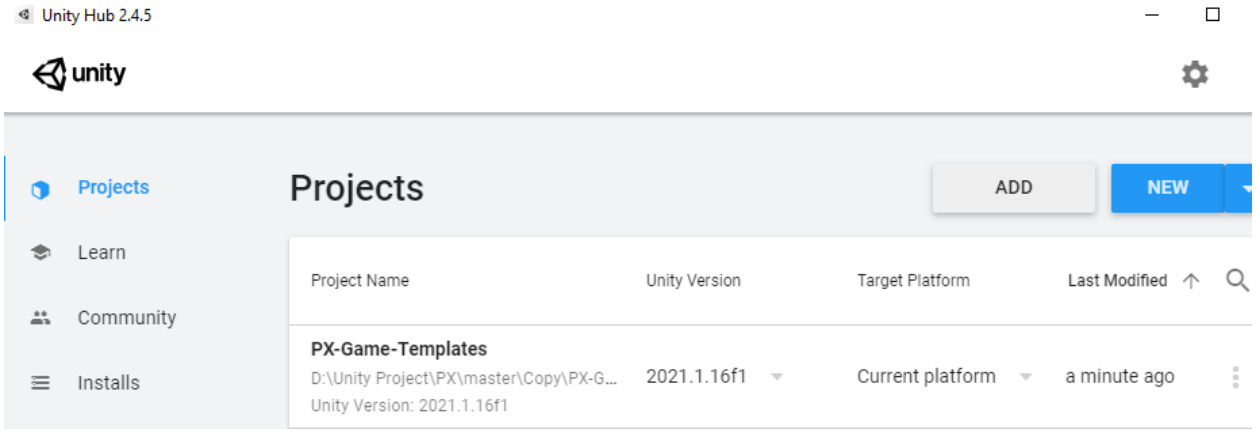
Open Unity Hub. In **Projects**, click **Add**. This will open up a new window for you to find the project.



Find the project. Click on the project and click **Select Folder**.

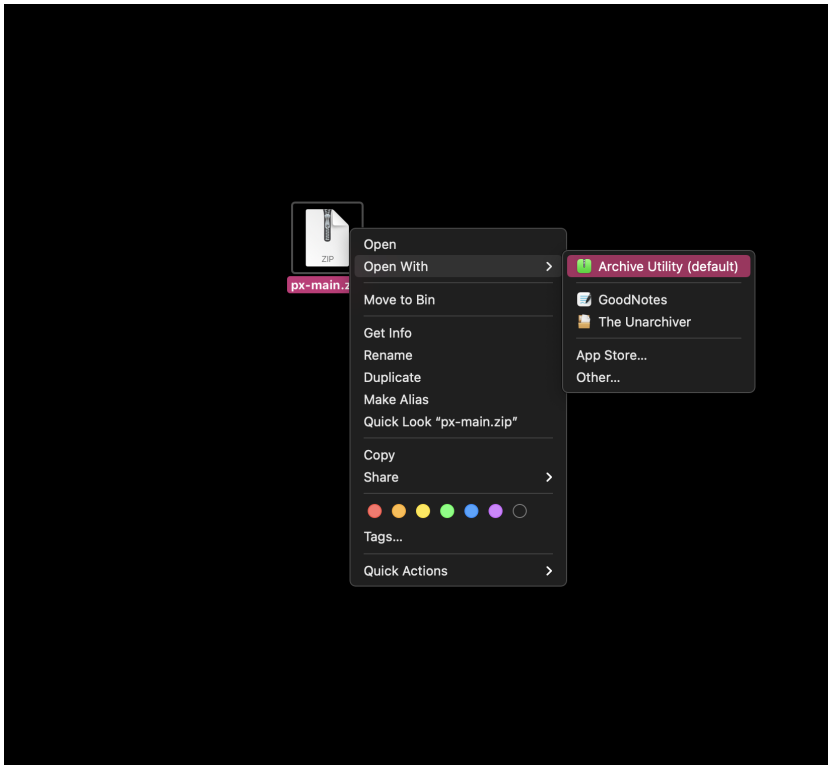


This will add the project to your Unity Hub.

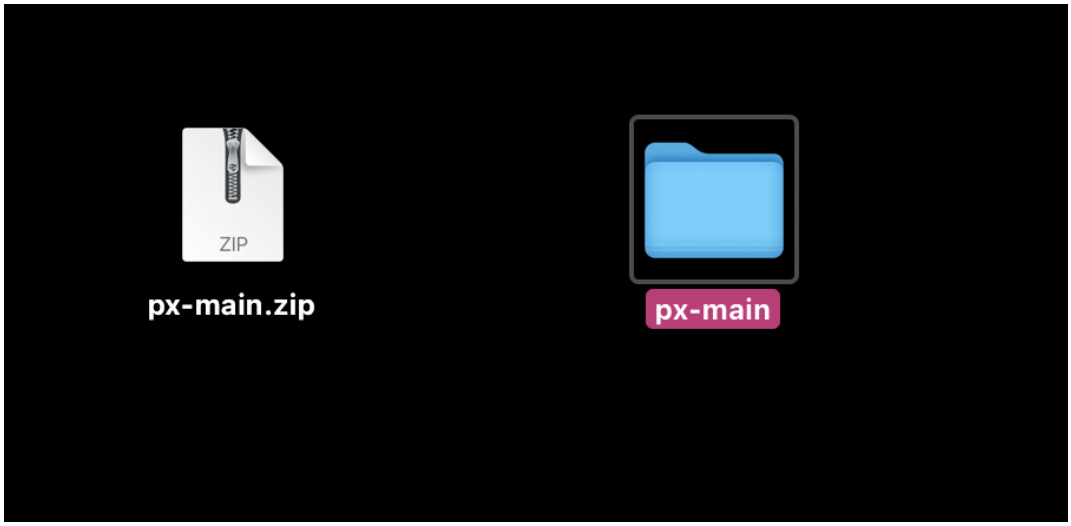


getting started for Mac.

After downloading the file , right click on the zip file and select open with > archive utility.

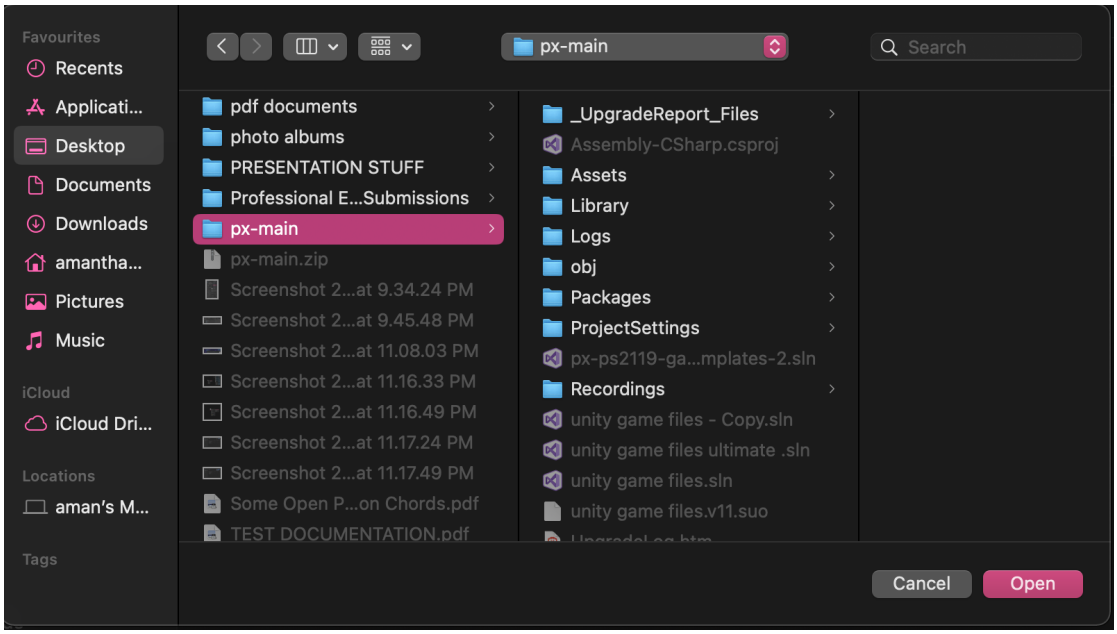


The files now extracted will appear.

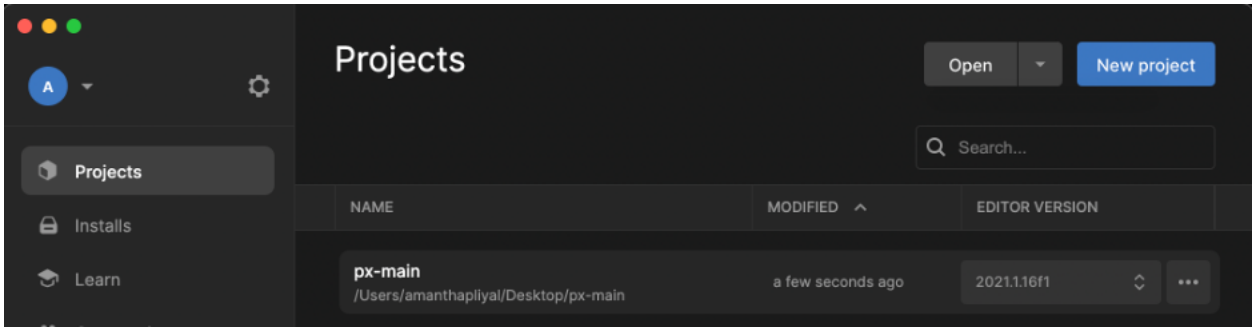


Open unity hub , now select the down arrow next to open and select “add project from disk”.

Now navigate and look for your project folder you just unzipped. Once you find it , select and click open(I saved and unzipped at desktop).



Your project will be added to the unity hub.

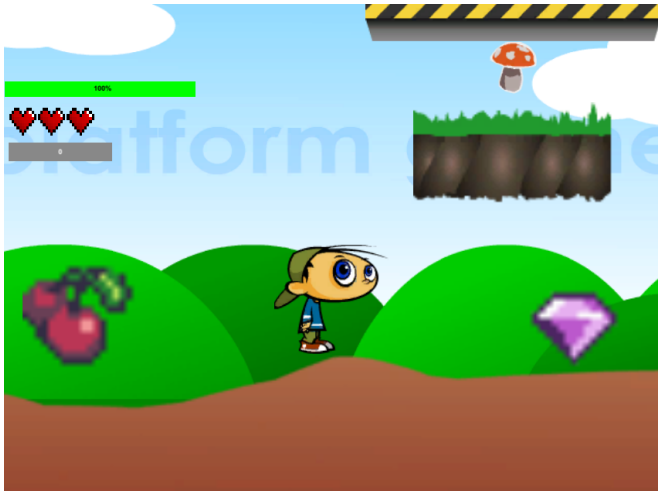


Creating a New Scene

Overview

A scene is a screen that the player will see. This can be a level, a menu, or a win/lose screen. All game creation happens in a scene and is separated from other scenes.You can have as many scenes as you want.

For example:

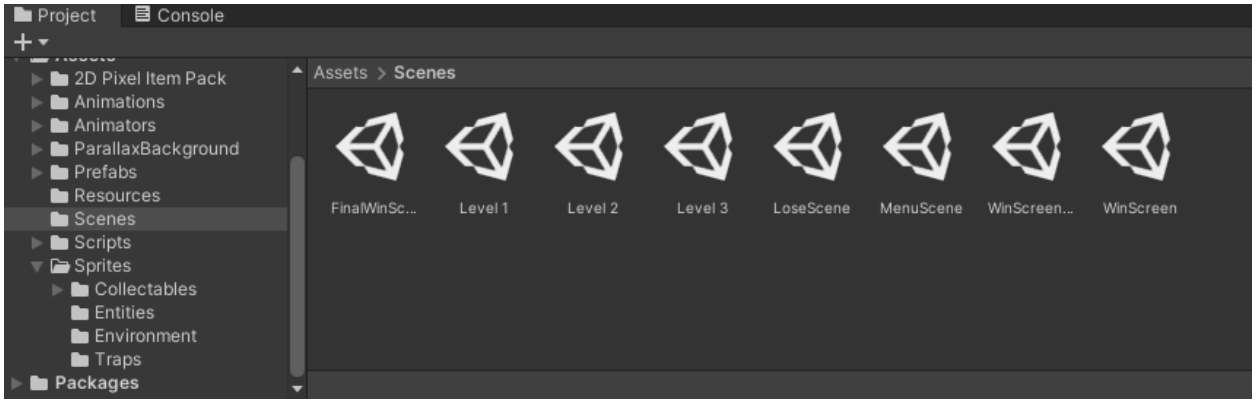


Level 1 Scene

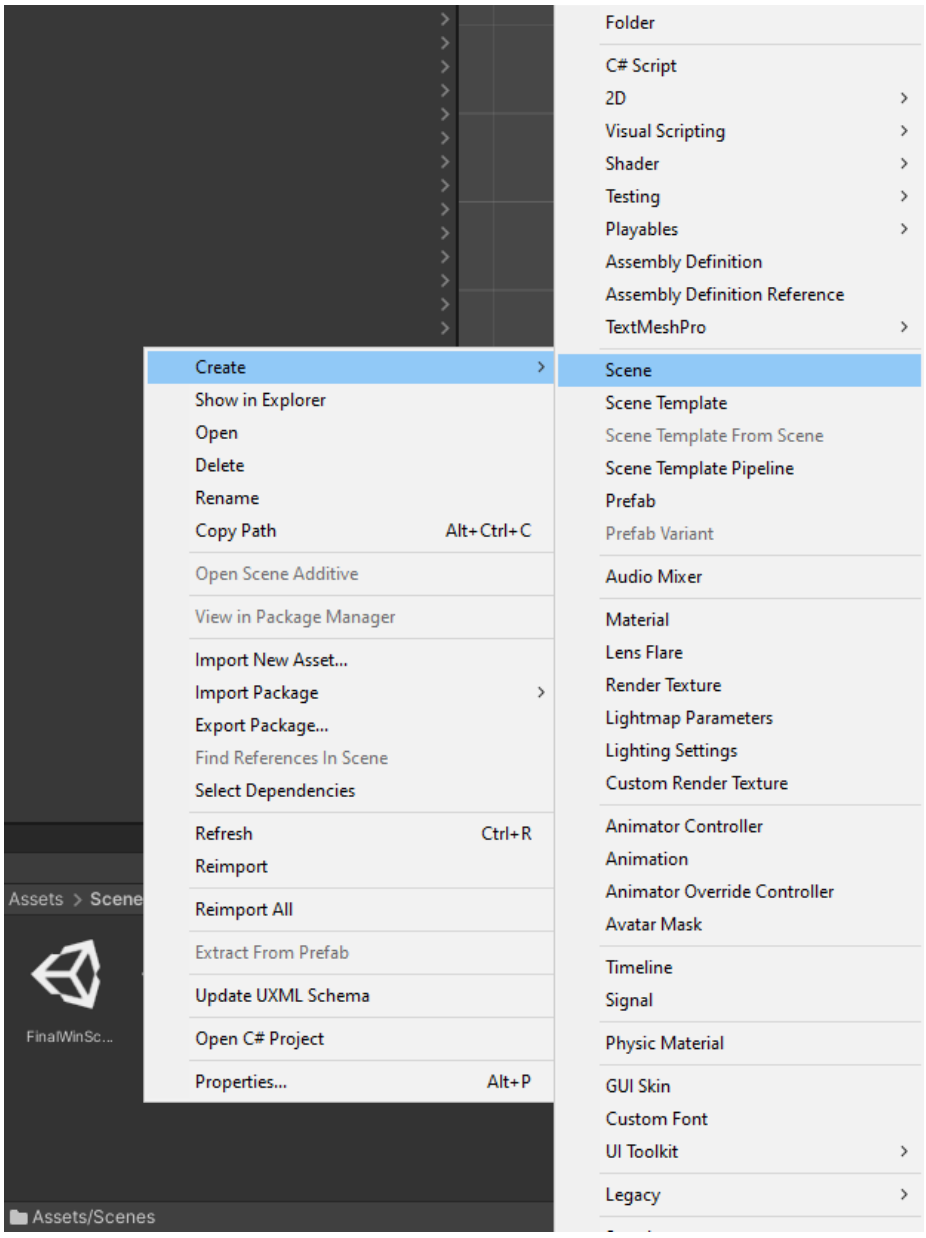
Level 2 Scene

Creating a New Scene

In your **Project** window.

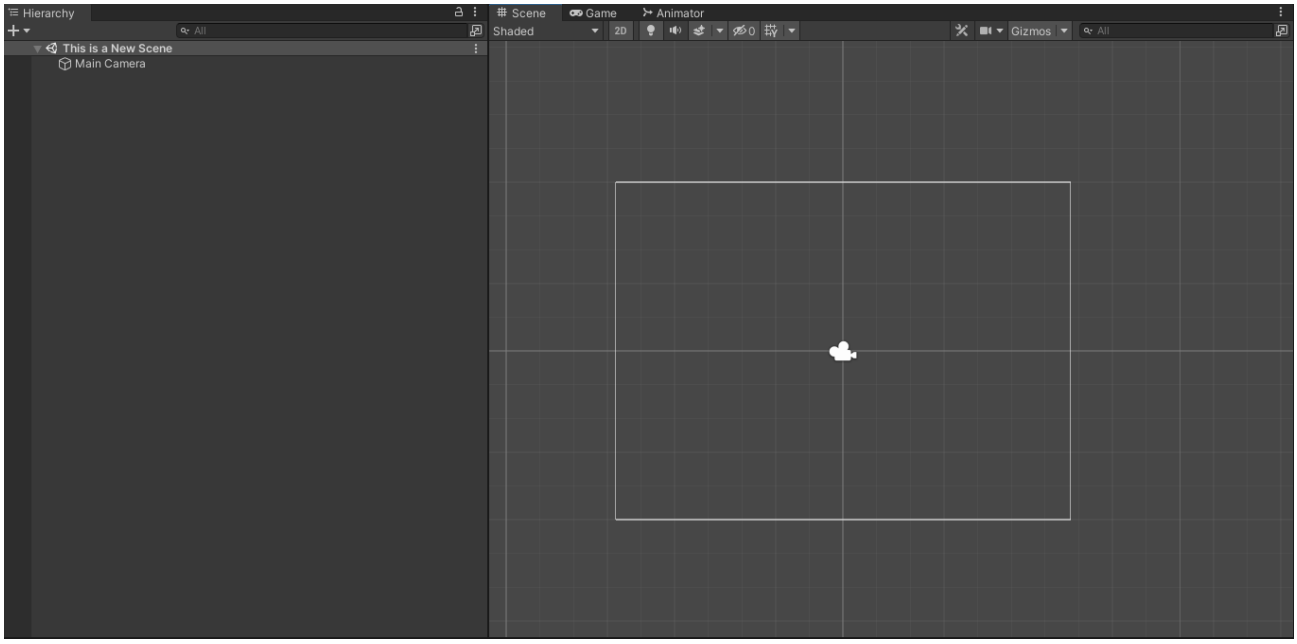


Right click an empty space. **Create > Scene**



You can name it anything you want. E.g. '**Level 6'**, **A Totally New Level**

Double click it to open and view it.

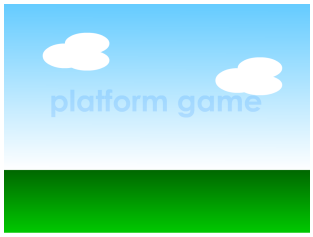


Environment

Overview

A game environment will have at least one of each:

- **Background** - Visual image
- **Mid parallax** - Visual image
- **Foreground** - Interactable and walkable image



Background

Mid Parallax

Foreground

Background

Overview

If the three game objects for the background are; background, mid, foreground, then the ‘Order in Layer’ are **0**, **1**, and **2** respectively.

- **0** - back
- **1** - middle
- **2** - front

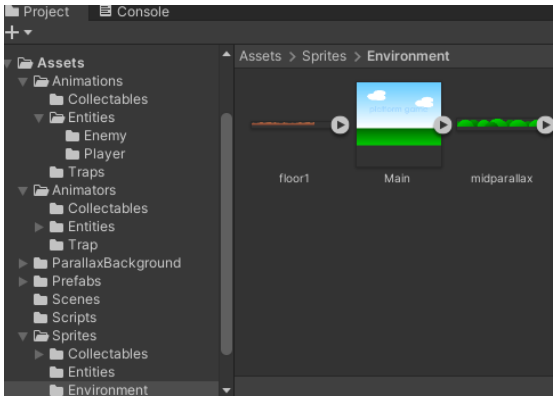
Required Files:

- Sprites (image file)

<u>Adding Background Sprite into Game Scene</u>

Main

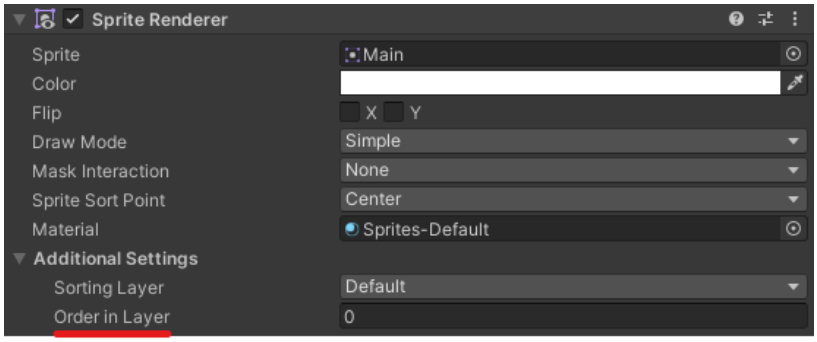
Click and drag the main background sprite into ‘**Main Camera**’.



Anytime the camera moves, the main background will also move.

Set ‘Order in Layer’ to **0**.

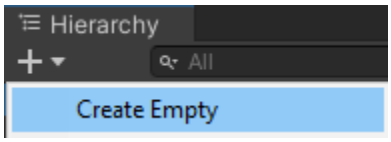
Any game objects with a ‘Order in Layer’ number than ‘0’ will be in front of it and any with less will be behind it.



Mid

Add new Gameobject to Game Scene

Click '**+**' and '**Create Empty**' in the Hierarchy window.

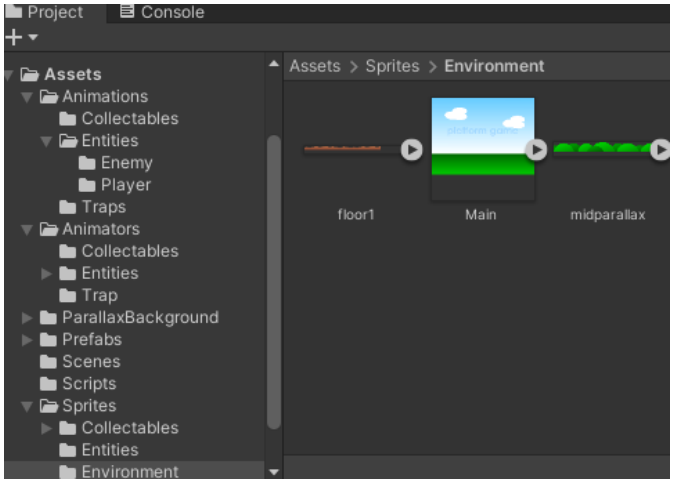


Rename the new Gameobject to **'Environment**'.

This is where you will put your 'Mid' and 'Foreground' game objects.

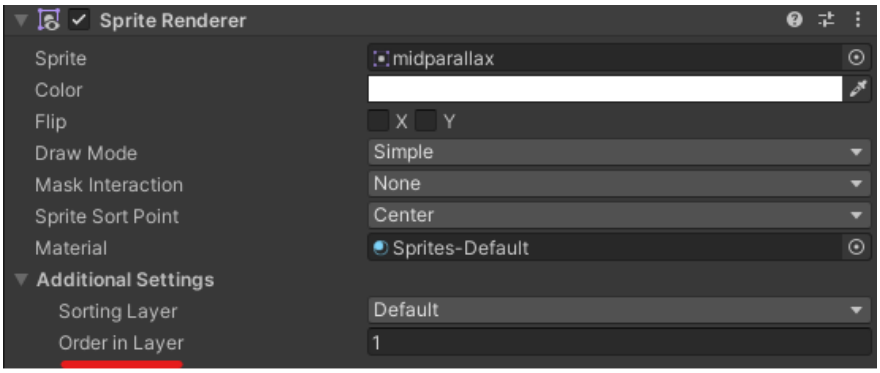
Adding Mid Sprite into Game Scene - Adding to Parent Gameobject

Click and drag your mid parallax sprite from the 'Project' window into the new '**Environment'** game object.



Set 'Order in Layer' to a number higher than **Background** but smaller than **Foreground**.

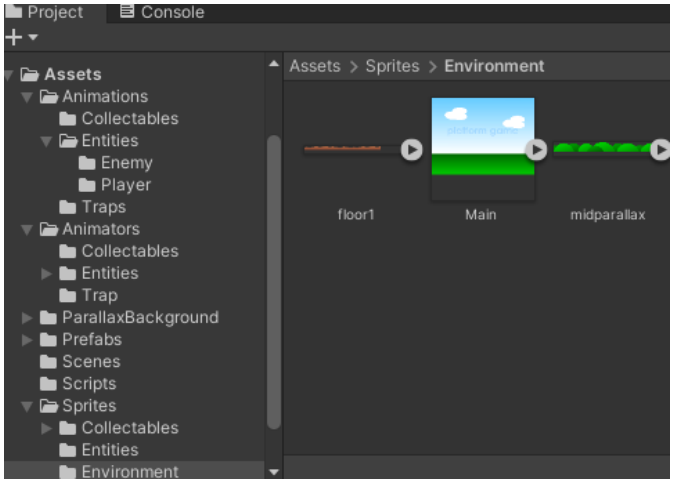
This sprite will be in front of Background but behind Foreground.



Foreground

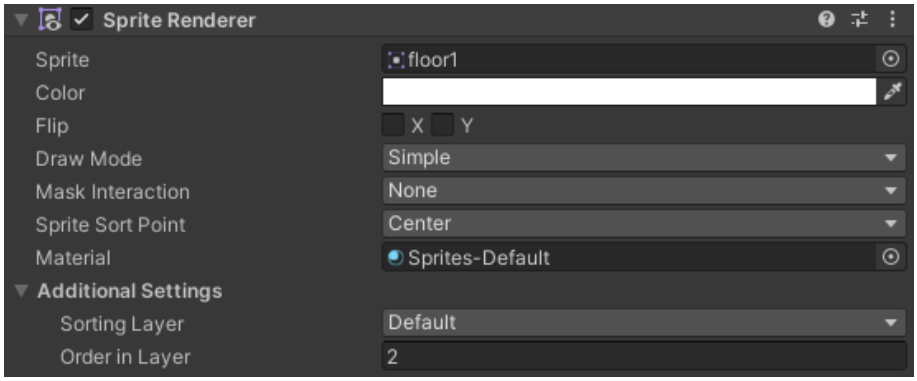
Adding Mid Sprite into Game Scene - Adding to Parent Gameobject

Click and drag your mid parallax sprite from the ‘Project’ window into the new ‘**Environment’** game object.



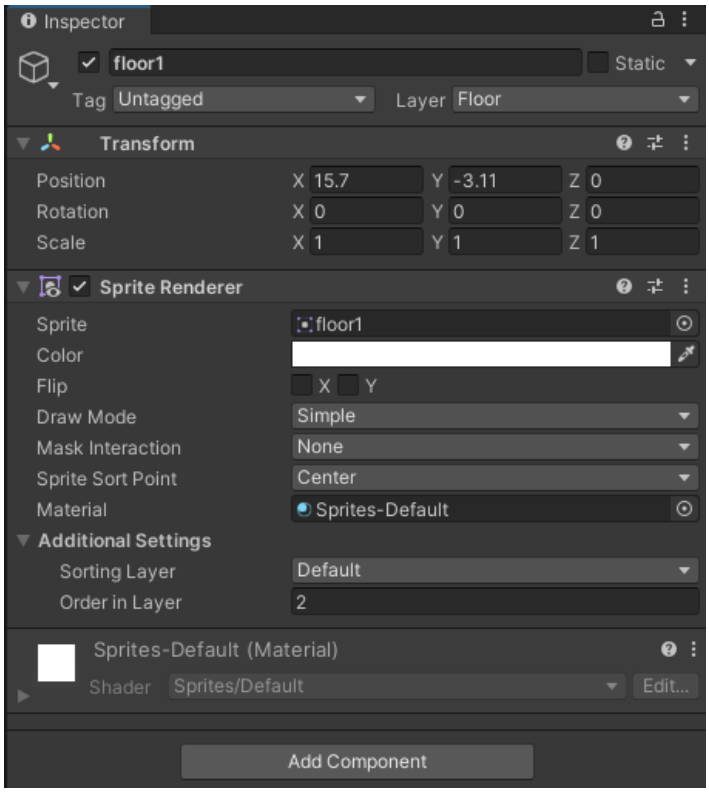
Set ‘Order in Layer’ to a number higher than **Background** and **Mid Parallax**.

This sprite will be in front of Background and Foreground.



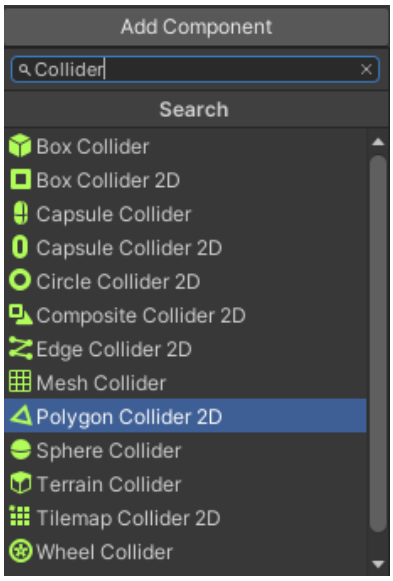
Adding Physics Colliders to Foreground Gameobject
Click on your Foreground game object in the Hierarchy.

On the right, in the Inspector, Click **Add Component**.



Search and choose **PolygonCollider2D**.

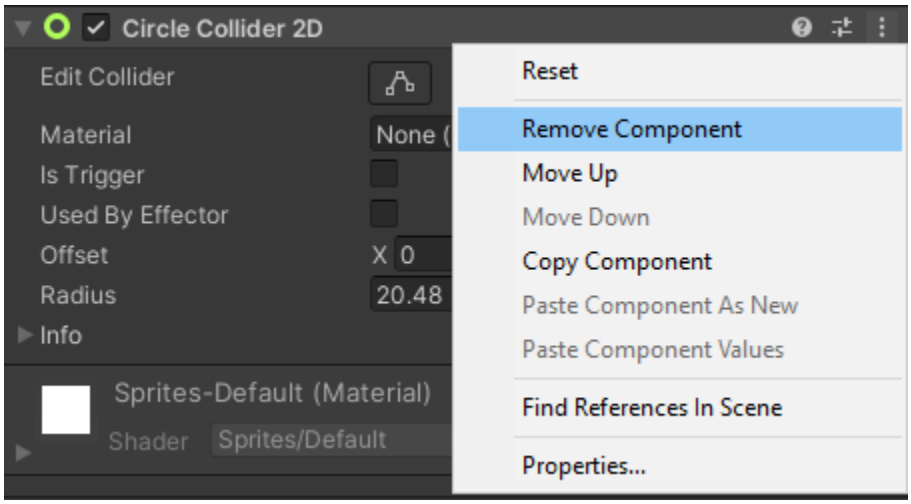
Colliders adds Physics to a game object and will, as the name suggests, 'COLLIDE' with any game object that has a RigidBody2D component. In this case, it will be your Player.



Removing a Component

If you chose the wrong collider.

Click on the **3 Dots** on the top right of that component and click **Remove Component.**



Moving Platforms

HorizontalObjectMovement / VerticalObjectMovement

Overview

The chosen game object will move between two game object points indicated by the designer. This can be horizontal or vertical movement depending on the script you are using.
These scripts will use the waypoints the user created to dictate where the objects should move between.

C# Scripts:

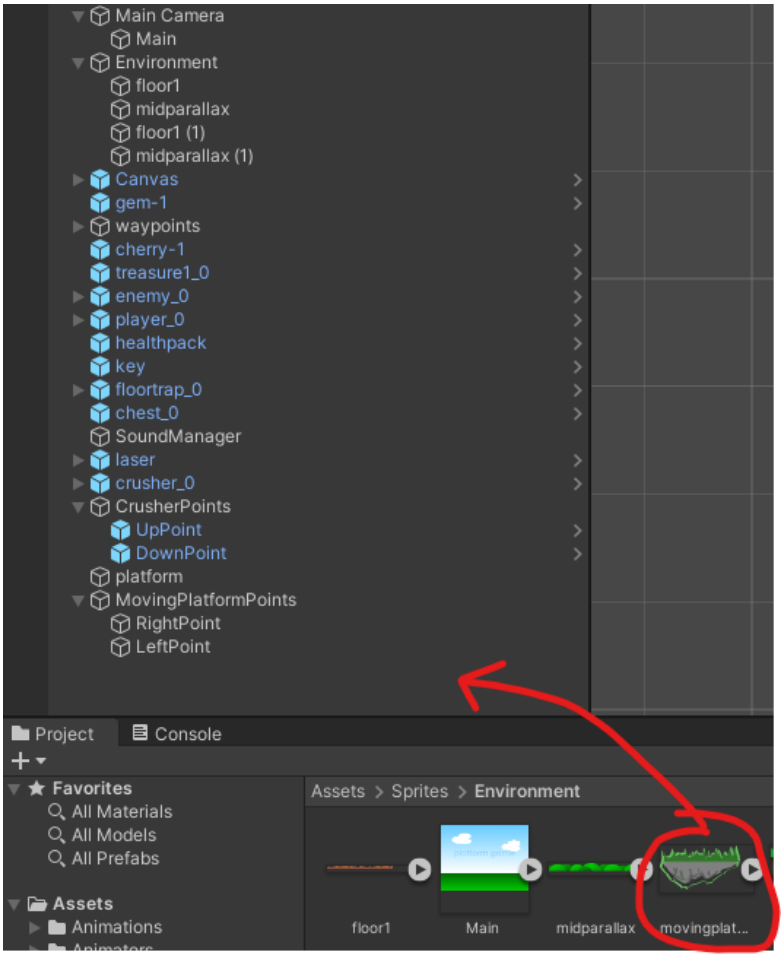
- HorizontalObjectMovement
- VerticalObjectMovement

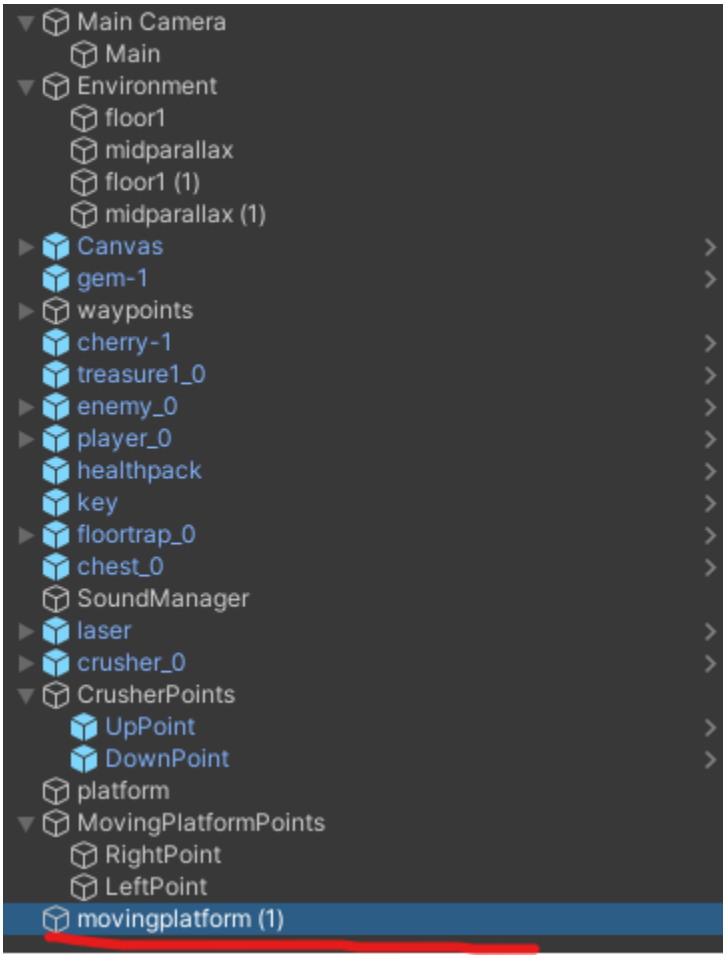
Required Files:

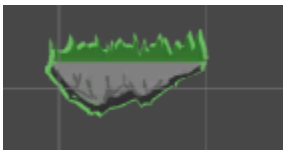
- Sprite (image file)

Adding Object Sprite

Drag the sprite image file you want to be your moving platform into your **Hierarchy**.







***NOTE - Make sure that is the sprite size you want.**
If not, you are required to recreate and draw it as changing the sprite size in the Unity Editor will distort the player model as well.

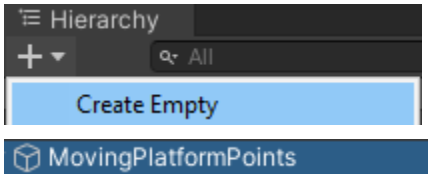
This will make the player model larger. **Which IS NOT WHAT WE WANT.**

Normal Size

Platform Size Changed in Unity.

Adding the Waypoints for the Object to Move Between

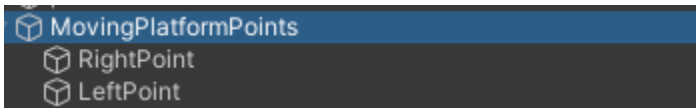
Create an empty game object for the waypoints.

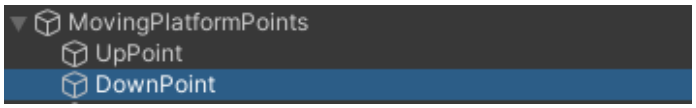


Create empty game objects as children of the parent game object. In this case for **MovingPlatformPoints**. **Right Click 'MovingPlatformPoints' > Create Empty**.

If you want it to move between two vertical points, you can name it differently. E.g. 'UpPoint' and 'DownPoint'.

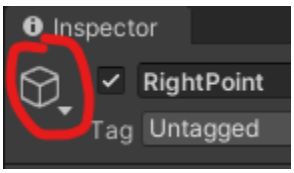
These are only for naming so it makes it easier to identify for yourself. Name it anything you want.

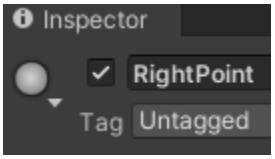




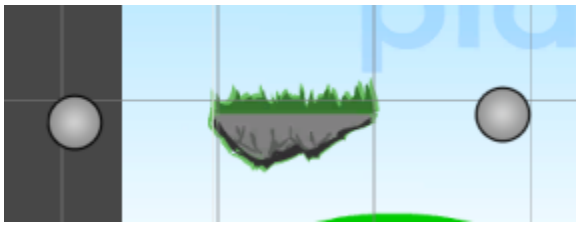
Click on a waypoint in your **Hierarchy.**

In your **Inspector**, click the box icon. Choose an icon so that we can see and distinguish the waypoint.



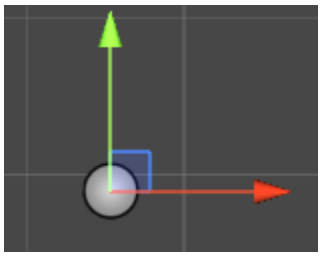


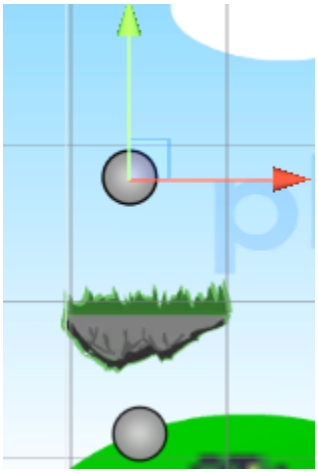
***NOTE - This does not show in game, it is only for the user to see during game creating.**

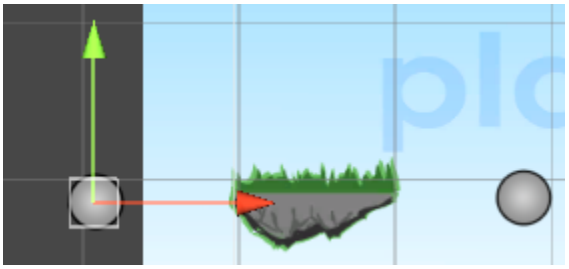


Moving the Waypoint

Use the arrows to position the waypoints in the desired locations. Horizontally or vertically.





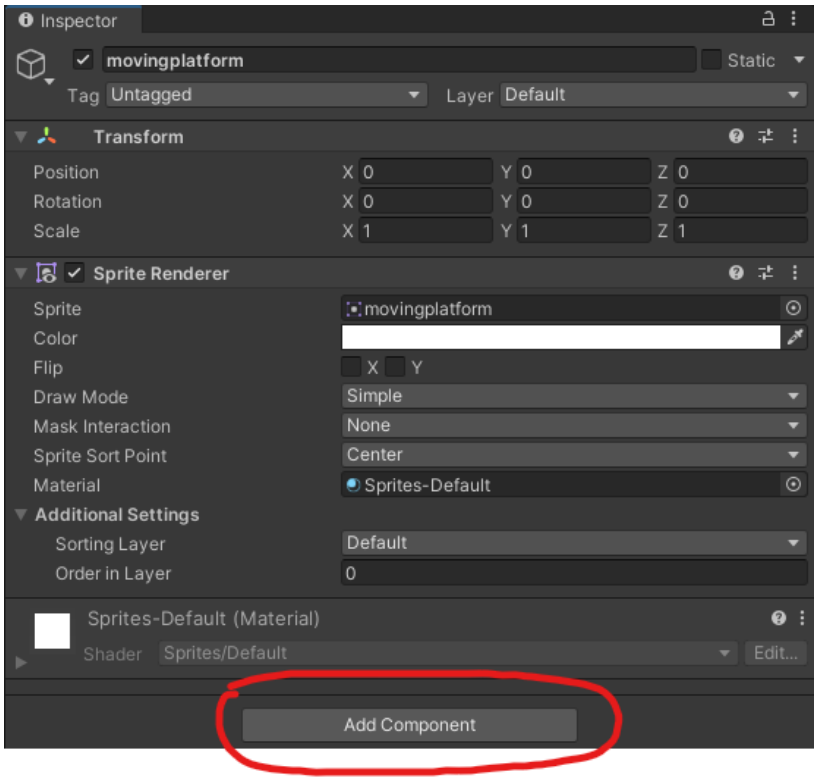


Adding Collider2D Component to Gameobject

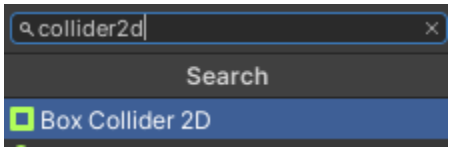
This is to allow the player to stand on the moving platform.

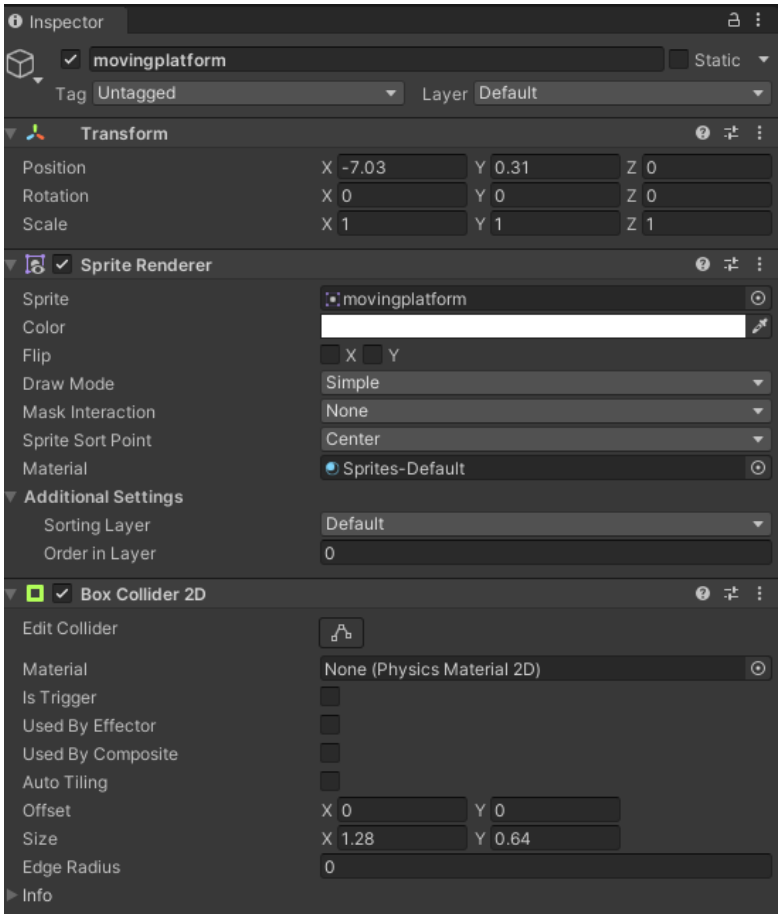
Click your moving platform game object in the **Hierarchy**.

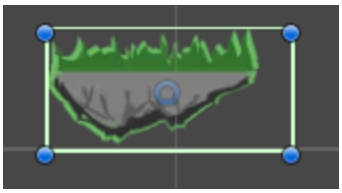
In your **Inspector**, click '**Add Component**' at the bottom.



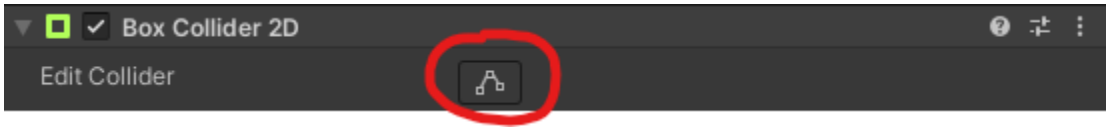
Search 'collider2D' and click **Box Collider 2D**.

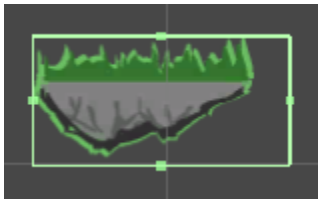




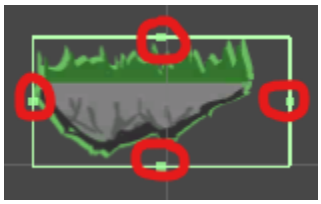


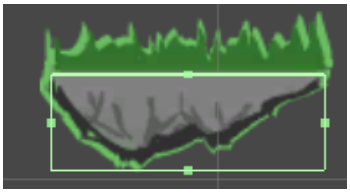
To edit the box collider. Click the icon next to **Edit Collider** in the **Inspector**.





Drag the points to your desired size.

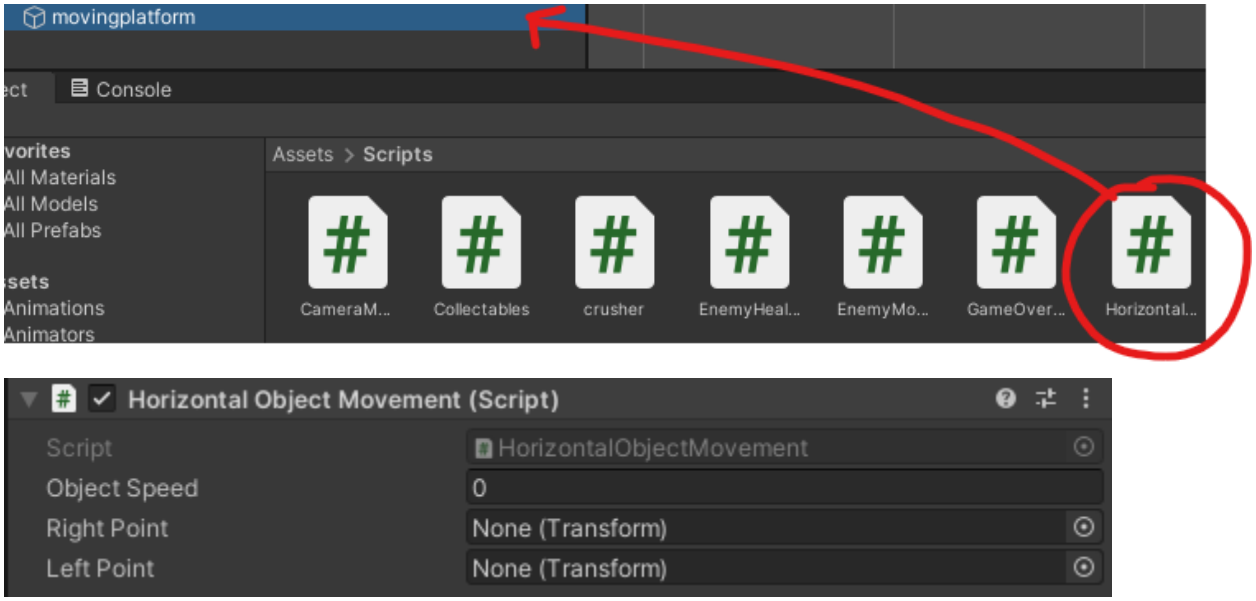




Implementing the Script

Horizontal moving platform

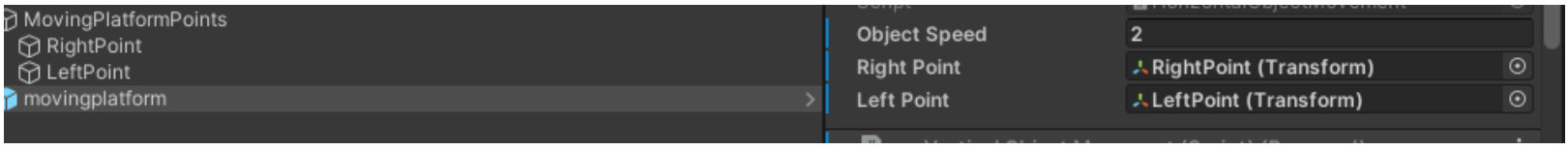
Find **HorizontalObjectMovement** and drag it into your game object.



Variables:

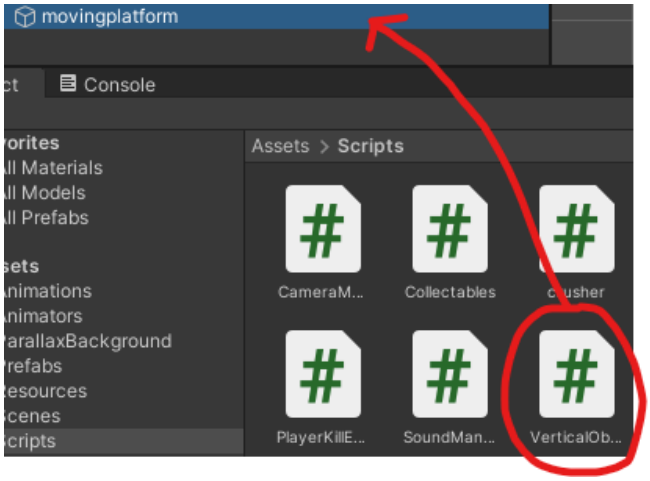
- **Object Speed** - Number indicating how fast the moving platform should go.
- **Right Point** - Right Waypoint
- **Left Point** - Left Waypoint

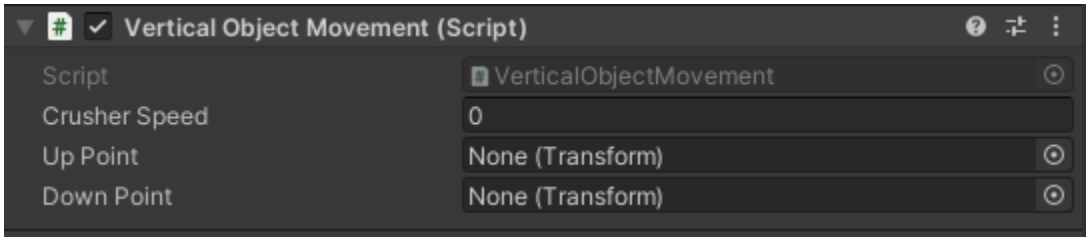
Click can drag the waypoints into their respective variables.



Vertical moving platform

Find **HorizontalObjectMovement** and drag it into your game object.





Variables:

- **Object Speed** - Number indicating how fast the moving platform should go.
- **Up Point** - Up Waypoint
- **Down Point** - Down Waypoint

Click can drag the waypoints into their respective variables.

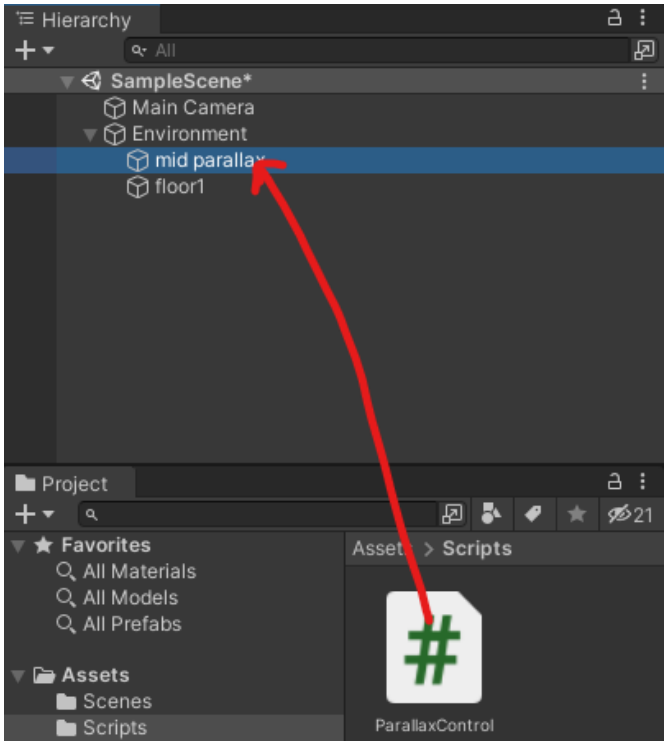
ParallaxControl

Mid Parallax

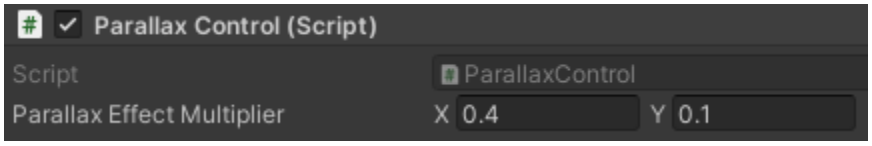
C# Scripts:

- ParallaxControl

Click and drag **ParallaxControl** from the 'Project' window into the desired game object background in the 'Hierarchy'.



You will then see it in your 'Inspector' when you click on the game object where you dragged the script into ('mid parallax'). It will look like this:

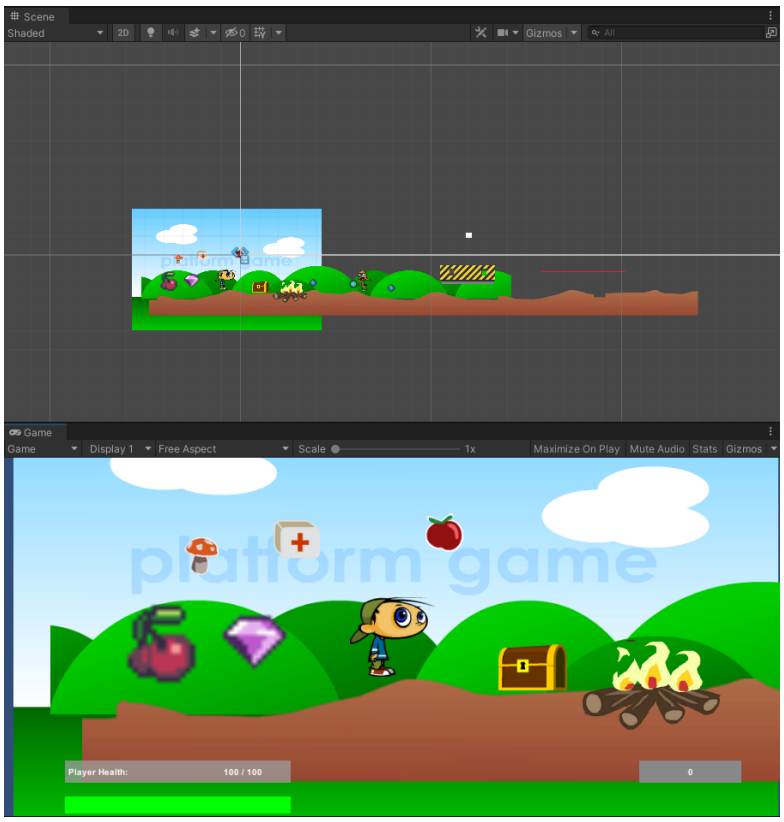


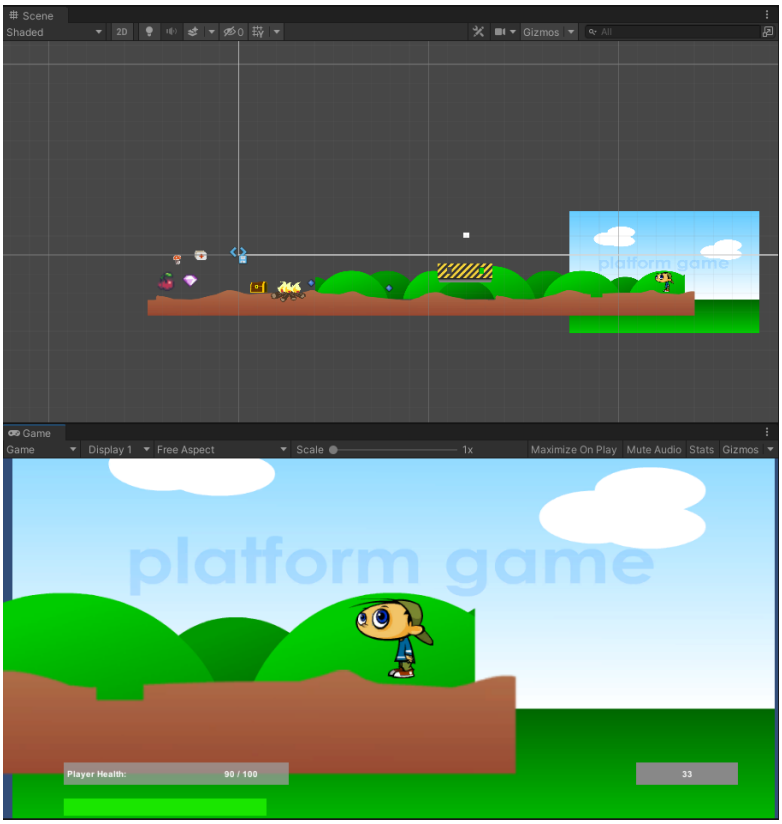
Variable:

- **Parallax Effect Multiplier** - Controls the intensity of the parallax effect on the X (left and right) and Y (up and down) axis. **The higher the number, the more parallax effect.**

Example:

In this example, the green grass follows the background when the player walks.





***NOTE - How the 'mid parallax' game object (green grass) moves when the player moves.**

Player/Main Character

Overview

The main game object that the player will control is through user input. Gameplayer will be based around the ‘Player’ game object with scripts (C# template code files) and references (drag-and-drop into Inspector components).

C# Scripts:

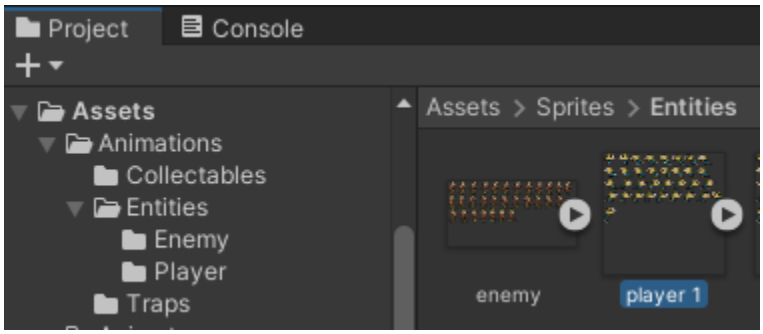
- PlayerController
- PlayerHealthUI
- PlayerKillEnemy

Required Files:

- Sprites (image file) - ***only sprites of the player facing one side is required (either left or right, not both)***
- Animations (.anim)
- Animation Controllers/Animator (.controller)

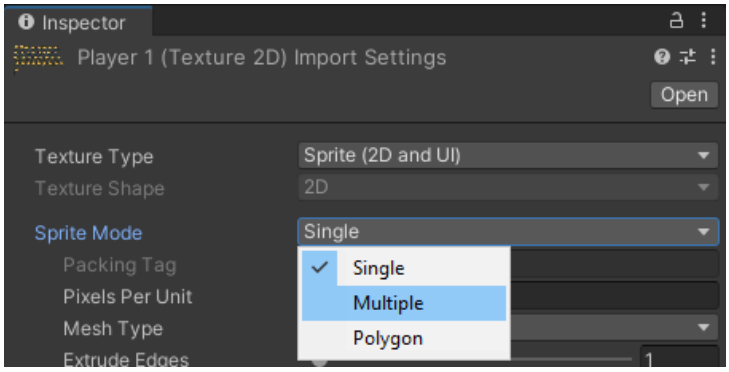
Player Sprites

In your Project window, click on the player sprite sheet.



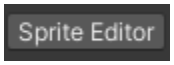
In the Inspector, click the dropdown for **Sprite Mode** and choose **Multiple.**

This allows for multiple crops and slices of the sprite sheet.

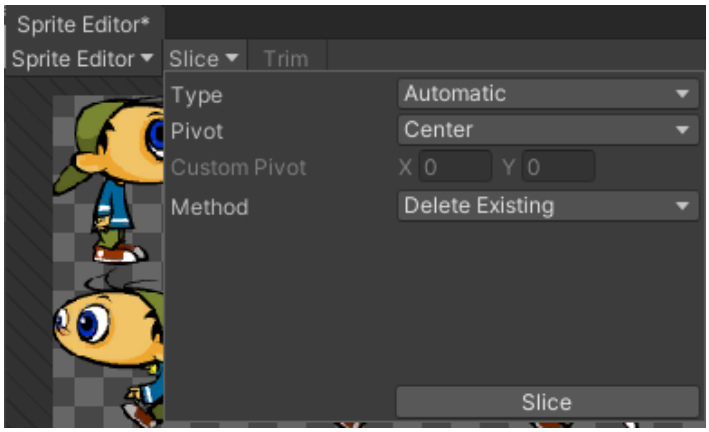


Click **Apply**

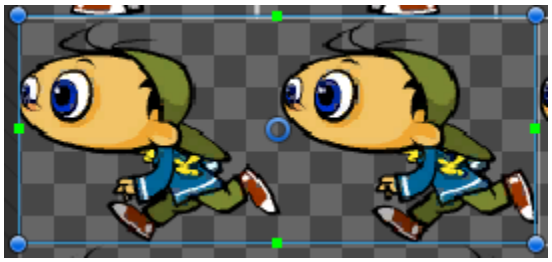
Click **Sprite Editor.**



Click the **Slice** dropdown and **Slice** to automatically separate the sprites into different png images.



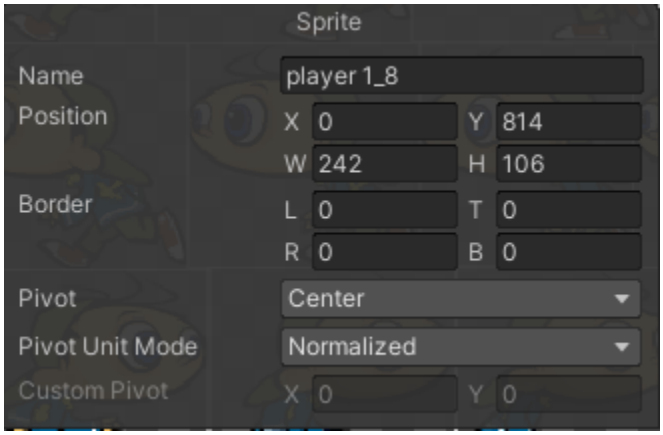
If it slices two sprites that are meant to be separate as one.



Click the box and delete it. Then click and drag around it to manually draw a box around it.

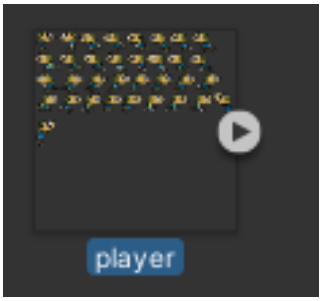
Click on the newly made slice box and rename it to the order it should be in.

***NOTE - if there is another sprite with the same name, the new name will not save. Please change the name of that slice and rename again.**

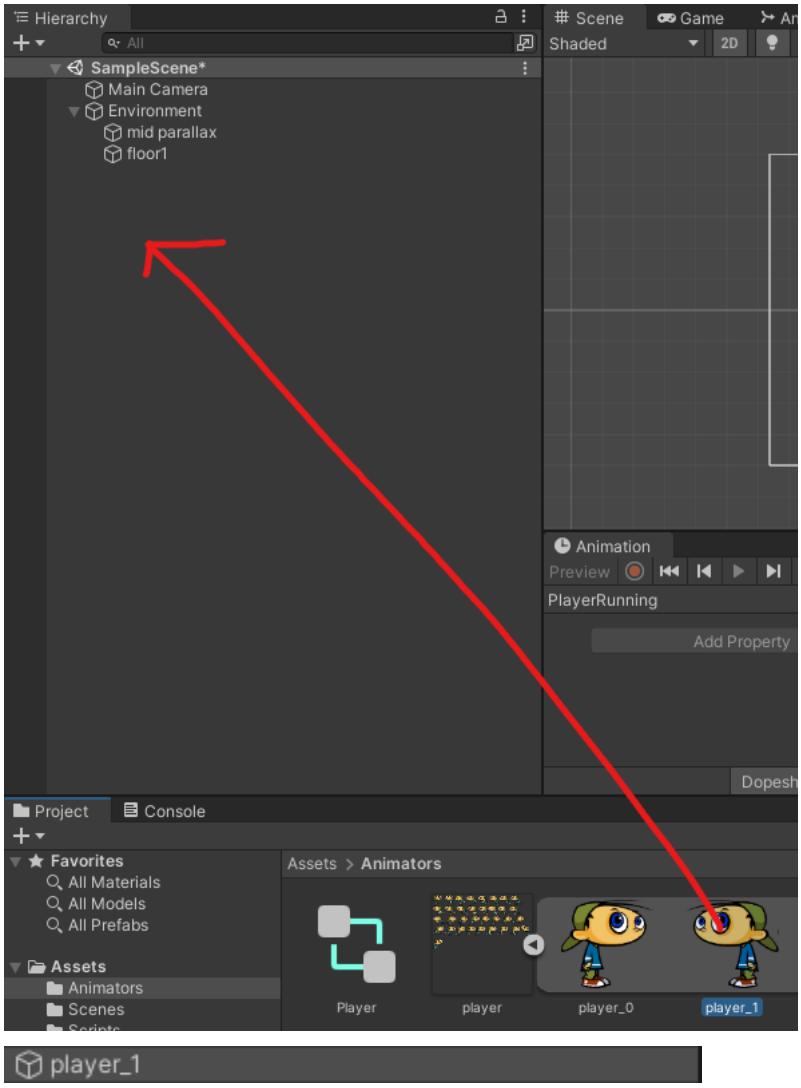


Make Player Gameobject

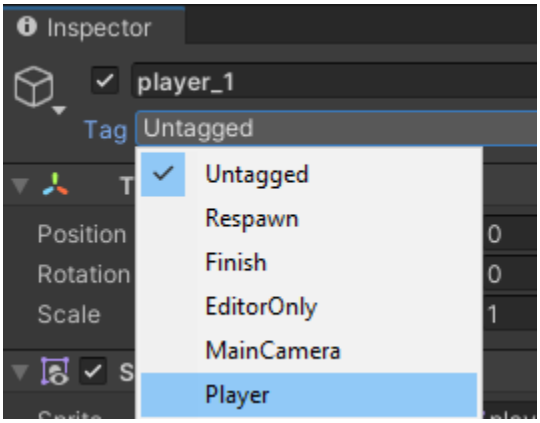
Go to your file with the player sprite. Click the arrow on the right to expand and see the sliced sprites.

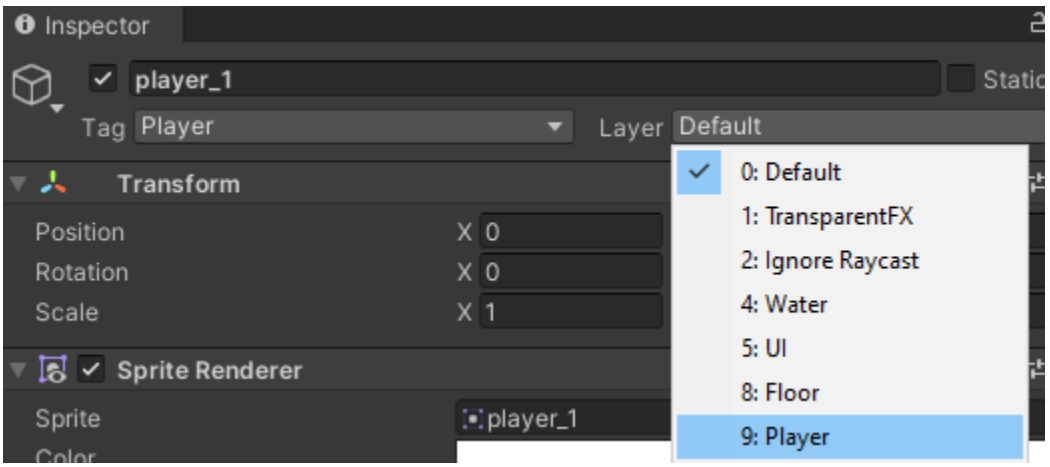


Click and drag an image slice into the Hierarchy to create a game object instance of it.



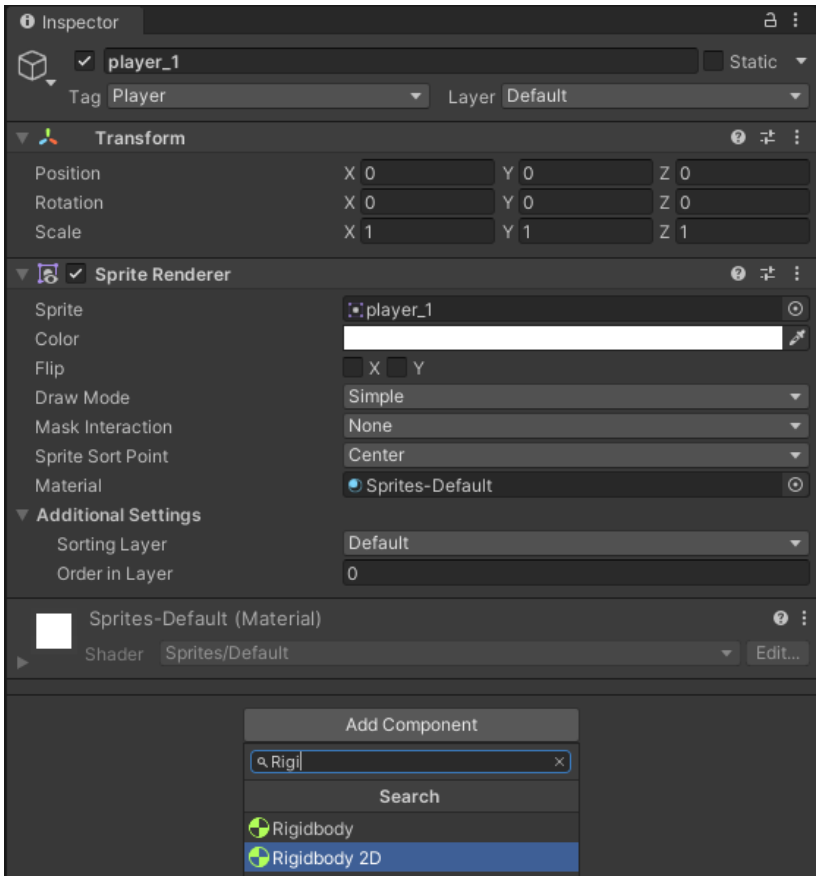
In the **Inspector**, click **Tag** and choose **Player.**
Create a new layer with a number 8 or higher, name it **Player**, and choose the **Player** layer.





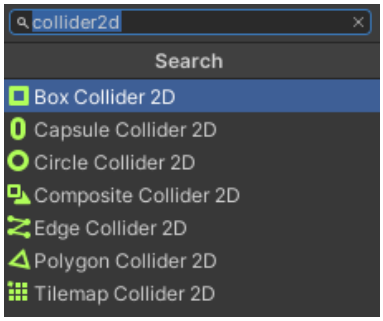
Add Component, search **Rigidbody2D** and add it.

Rigidbody2D adds physics to the game object.

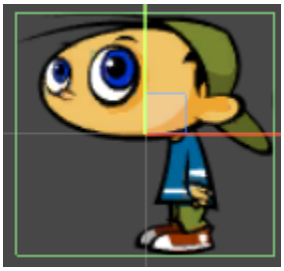


Add Component, search **Collider2D** and add a collider of your choice for the Player.

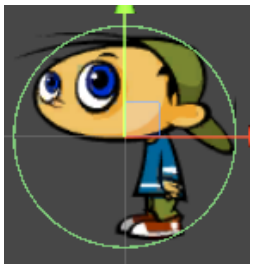
Recommendations are: BoxCollider2D, CapsuleCollider2D, PolygonCollider2D.



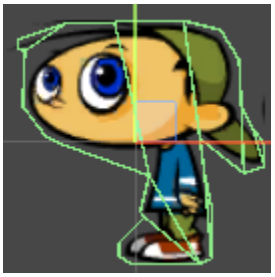
BoxCollider2D



CapsuleCollider2D

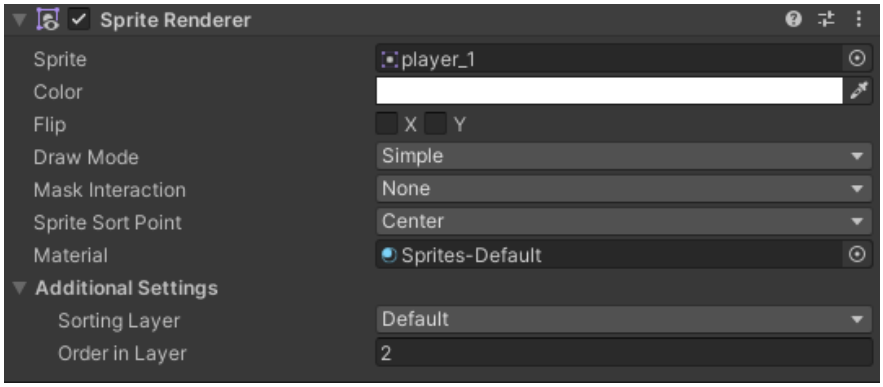


PolygonCollider2D



Player Layer

Change the ‘Order in Layer’ for the player game object in Inspector to the same layer as your **Foreground**. In this case, the **Foreground** layer order would be **2**, as such would be the Player.



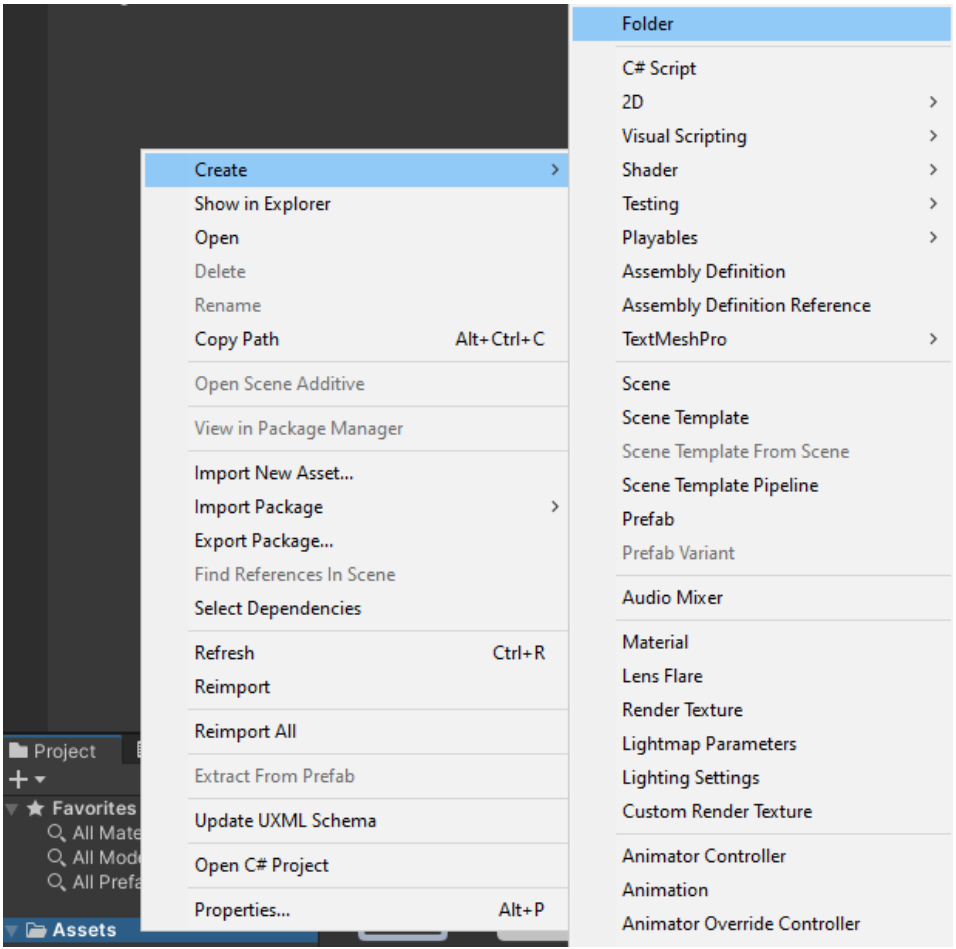
Animations

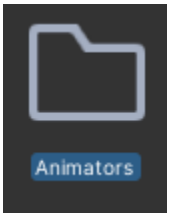
Animators

Animators are the file which controls the state and transitions between a game object’s animations.

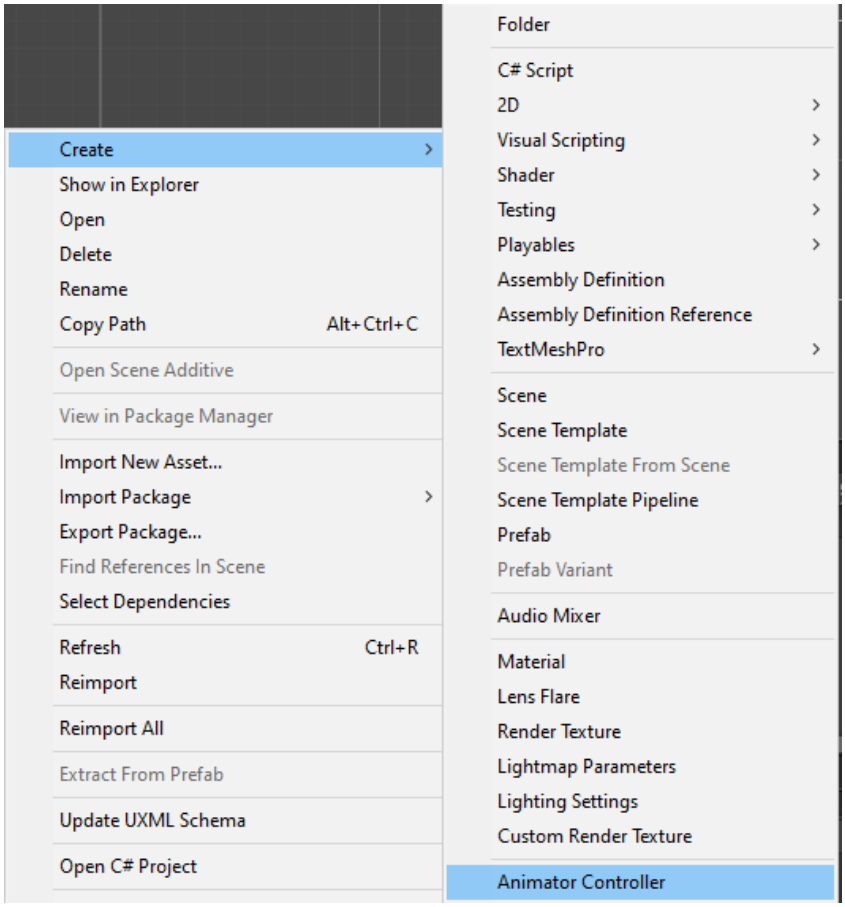
For example, PlayerIdle -> PlayerJump, or, PlayerRunning -> PlayerIdle.

In the Project window, right-click the **Assets** folder and click **Create > Folder**, and rename it ‘**Animators**’

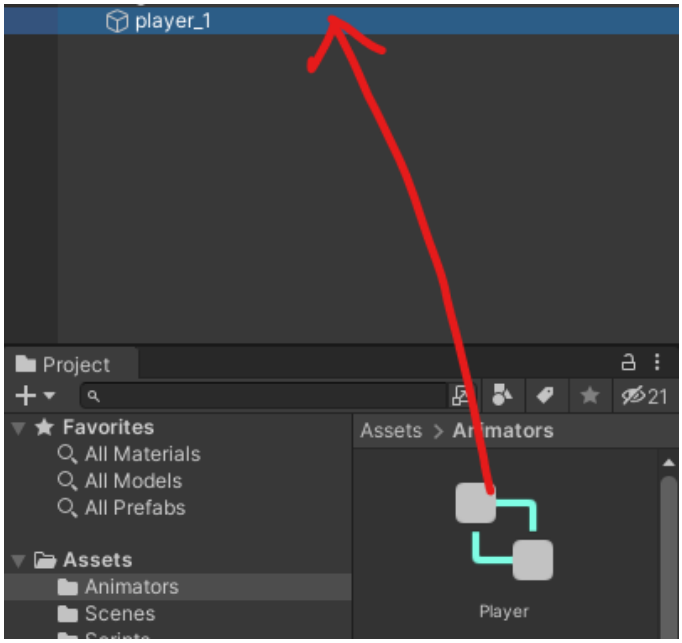




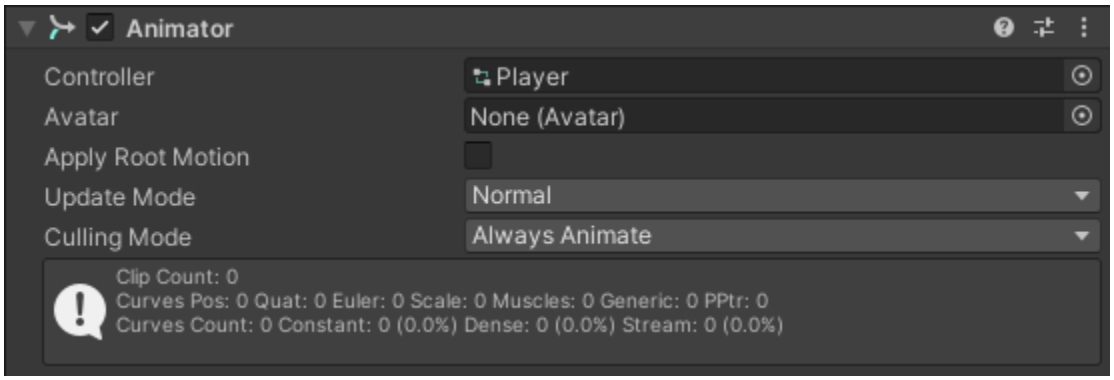
Right-click the file **Animators** and click **Create > Animator Controller.** Rename to **Player**.



Click and drag the **Player** animator file into the player game object.

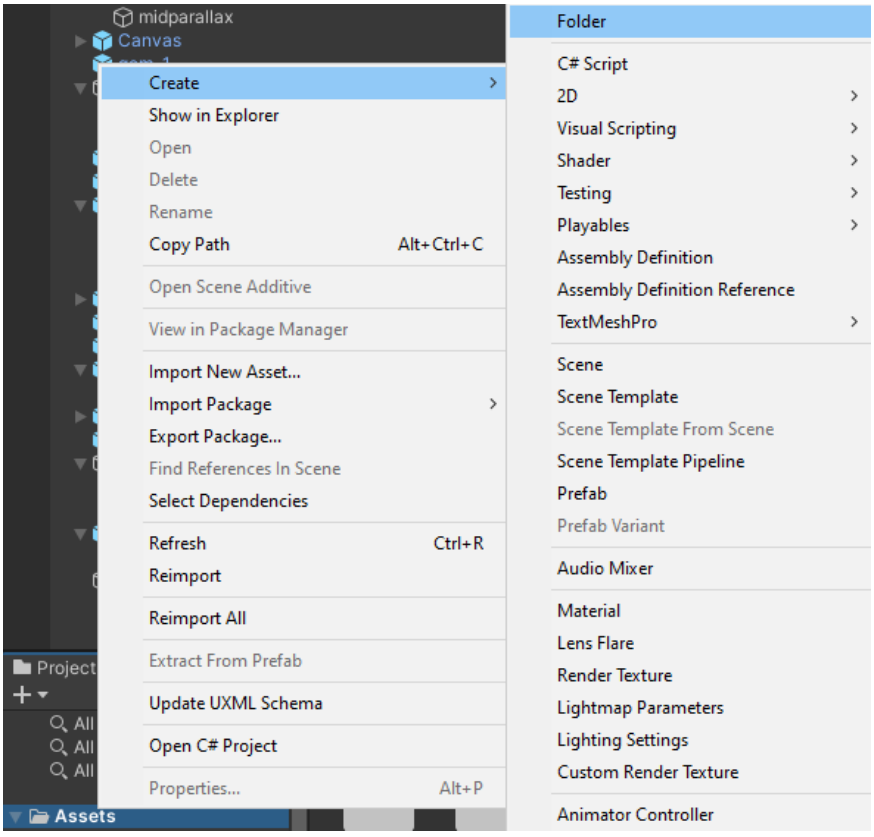


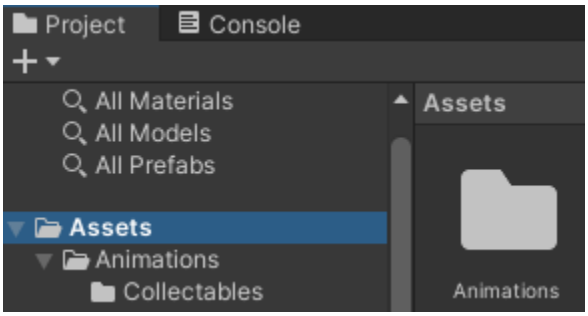
It will show up in the **Inspector** when the player game object is clicked in the **Hierarchy** as shown in the image below.



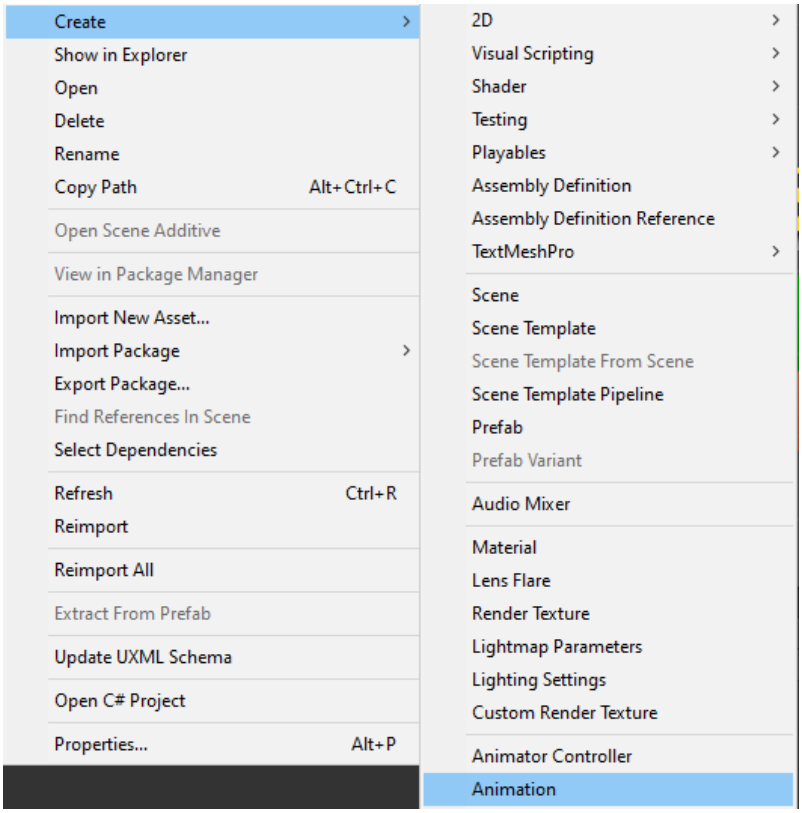
Animation

Make a new **Animations** folder to store your animation files.

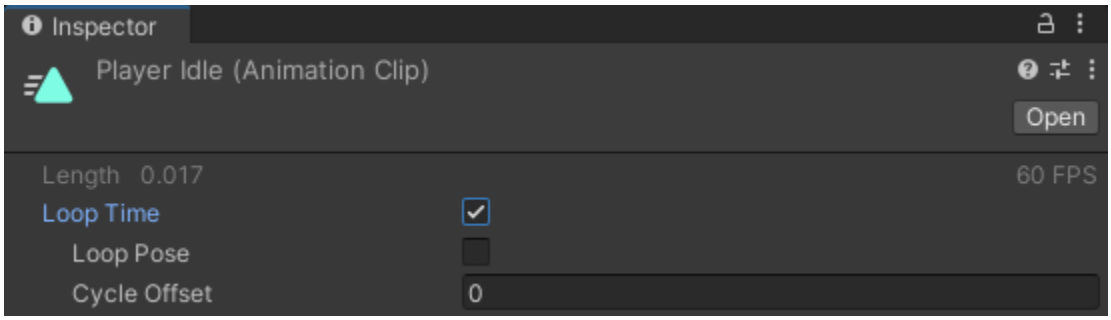




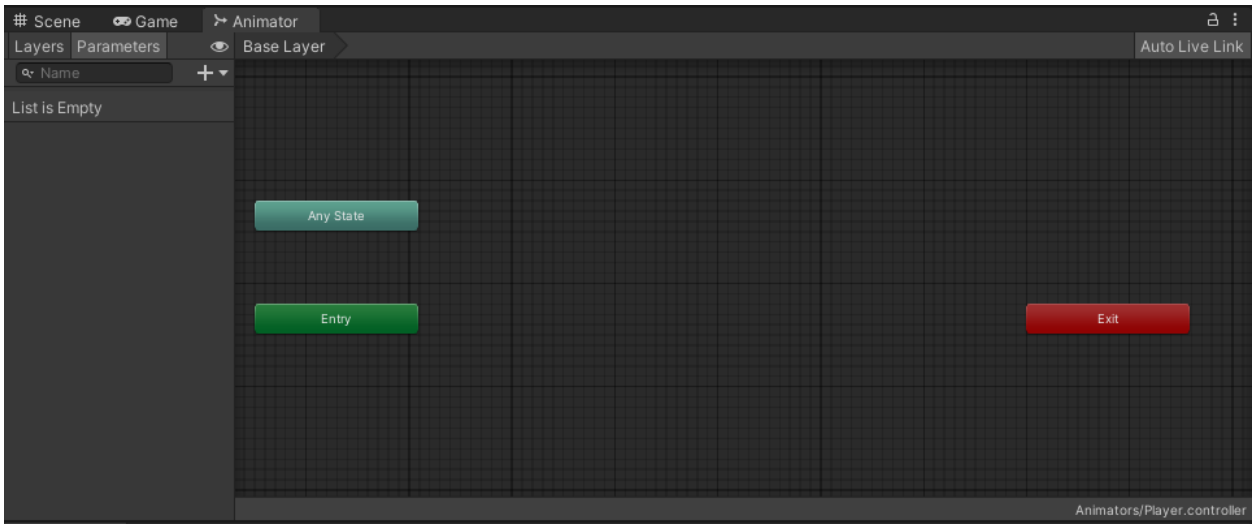
Right-click the **Animations** folder and click **Create > Animation**. Name it **PlayerIdle.**



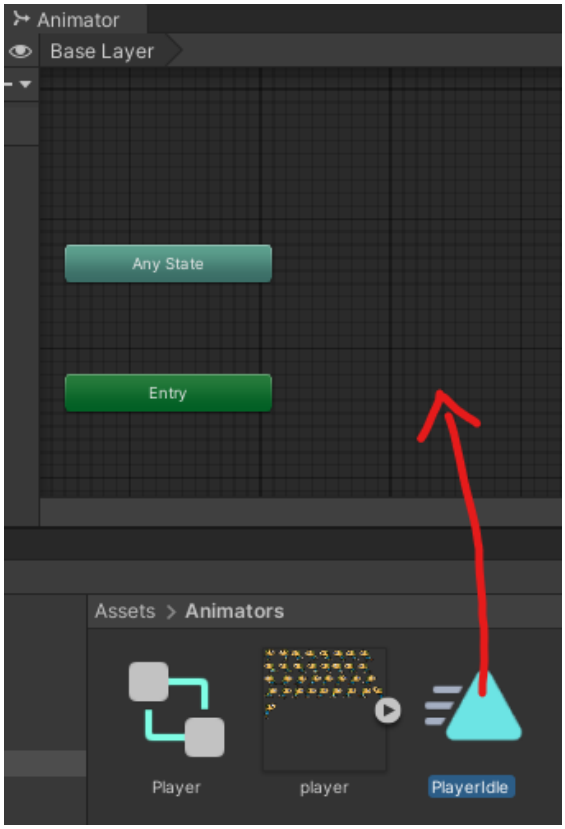
Click on **PlayerIdle** and check **Loop Time**. This will allow this animation clip to loop infinitely instead of playing just once.

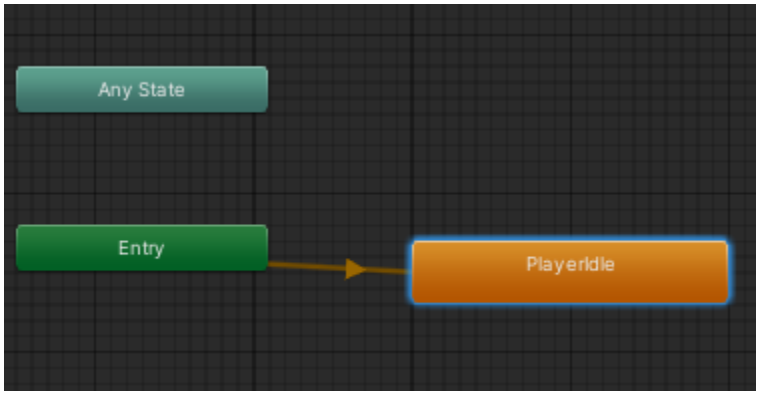


Double click the **Player** animator file in **Projects > Animator.** An **Animator** window should pop up.

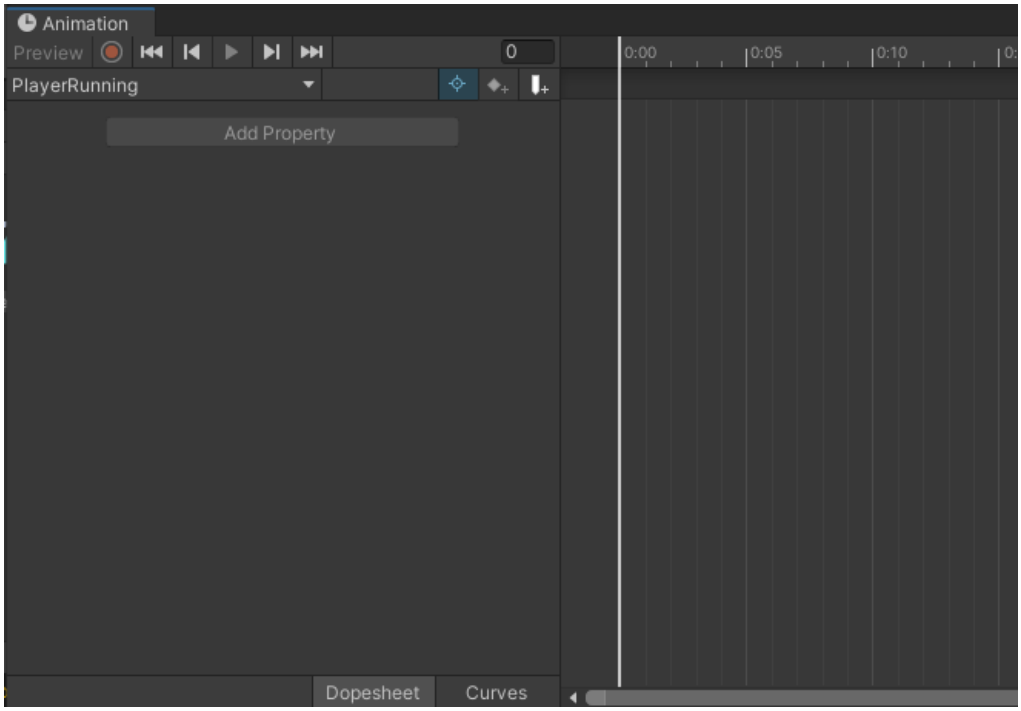


Click and drag **PlayerIdle** into the **Animator** window.

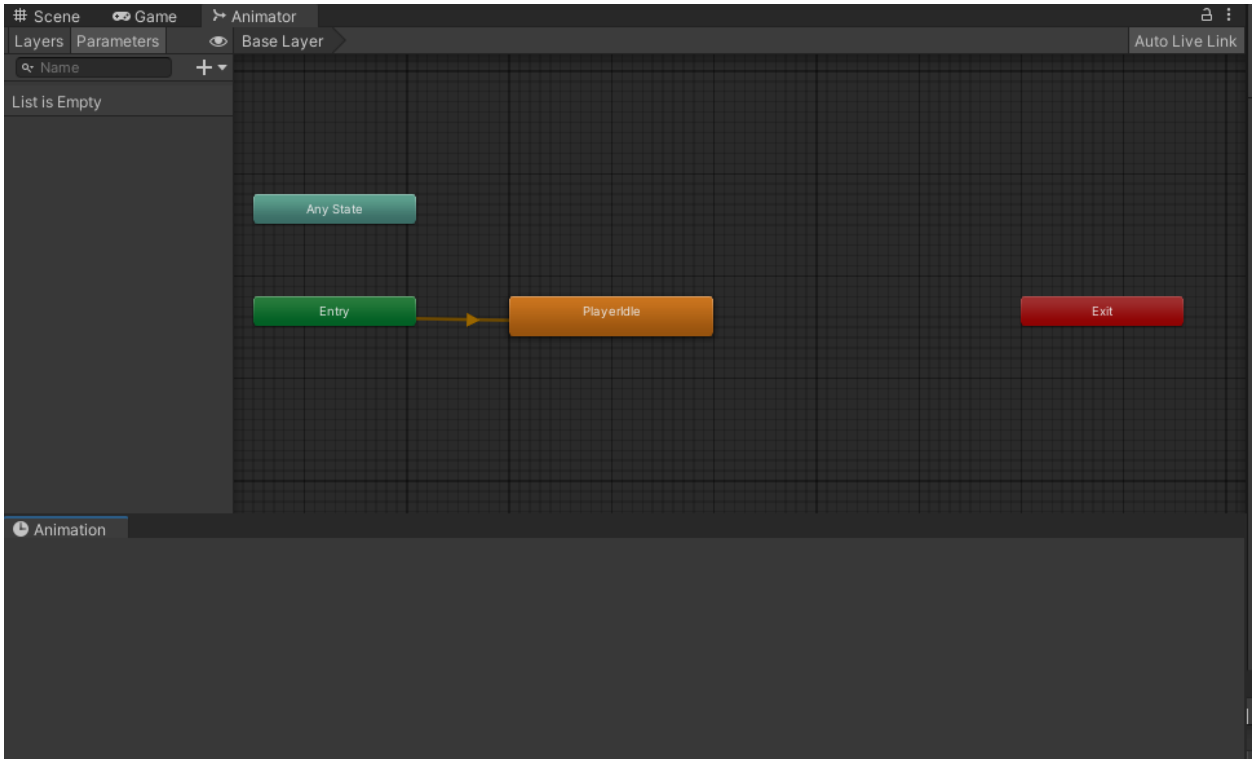


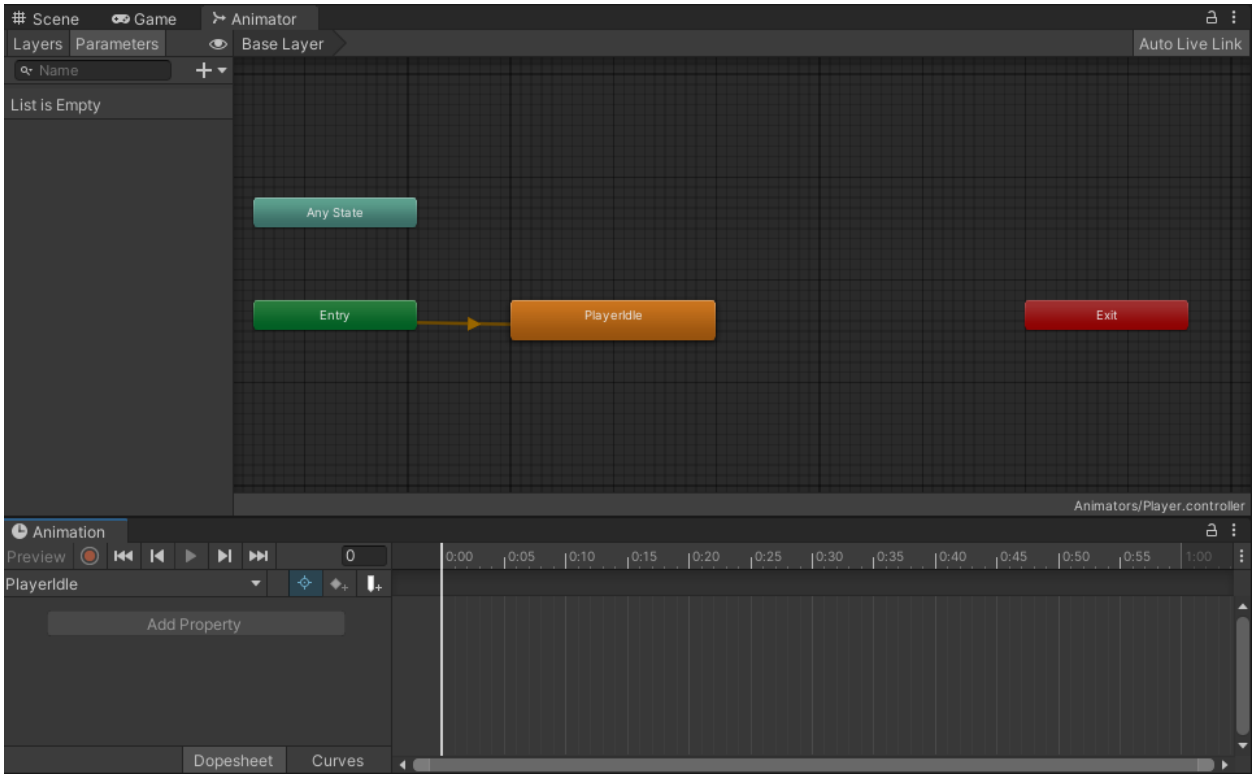


Double click **PlayerIdle.** An **Animation** window should pop up.



Click and drag the window under the **Scene** window to dock it.

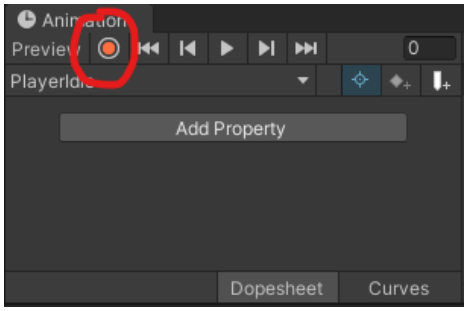




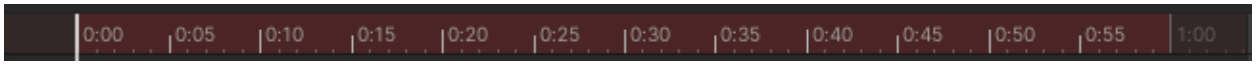
Running Animation

Repeat for **PlayerRunning** animation.
The new animation file will be placed in the same **Player** animator controller file.

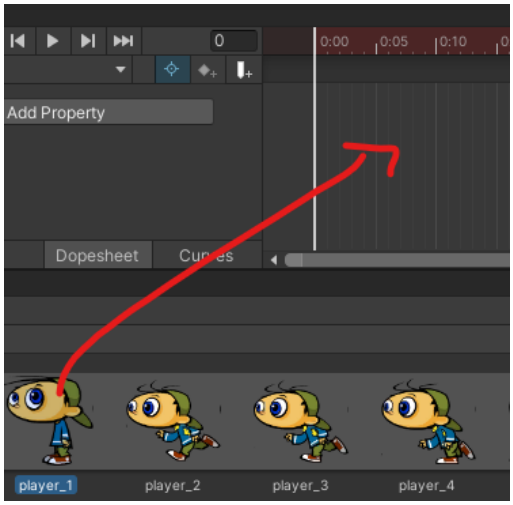
In the **Animation** window, click on the **Red Circle**, which is the record button.

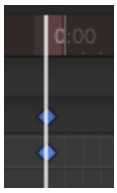


The time frame should turn Red. This means you can start dragging in images for the animation.

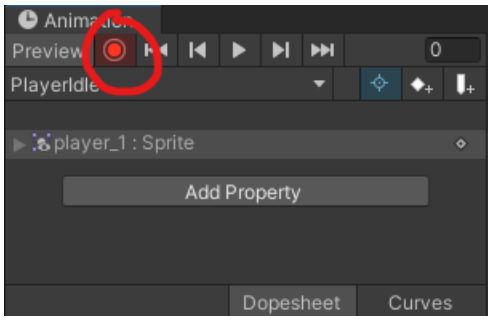


Drag it into the **Animation** window.

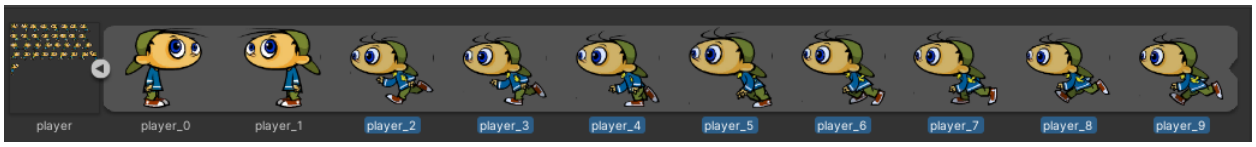




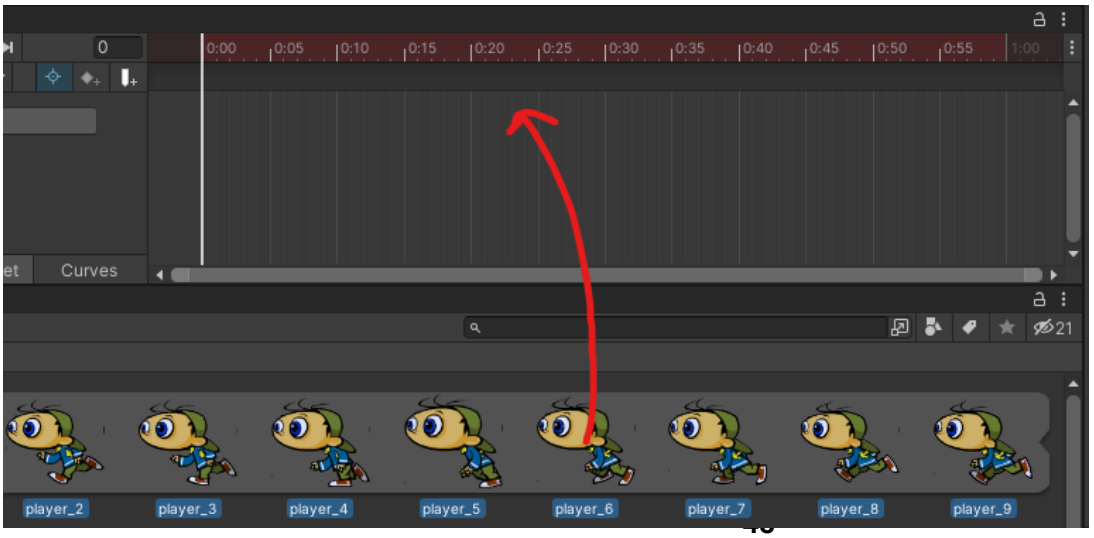
Press the Red button again to stop the recording for **PlayerIdle**.

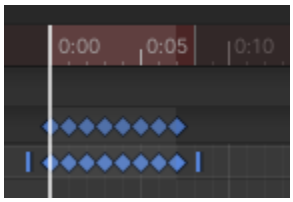


In your sprites folder, expand the sprite sheet right the arrow and select all the sprites you want to be your running animation.

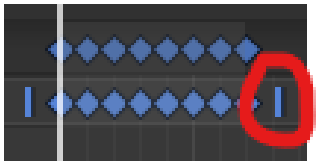


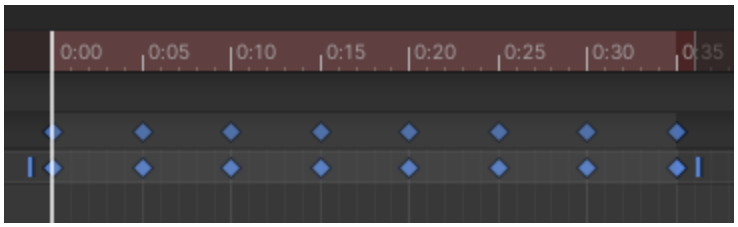
Drag it into the **Animation** window.



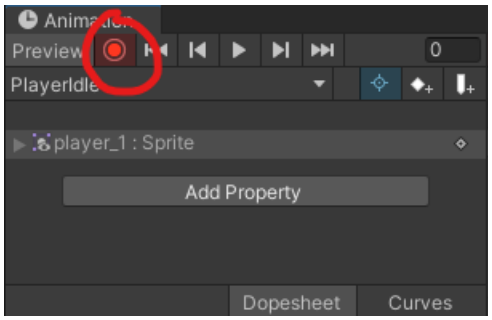


Click and drag the bar on the right to increase the time it takes to run the entire animation.



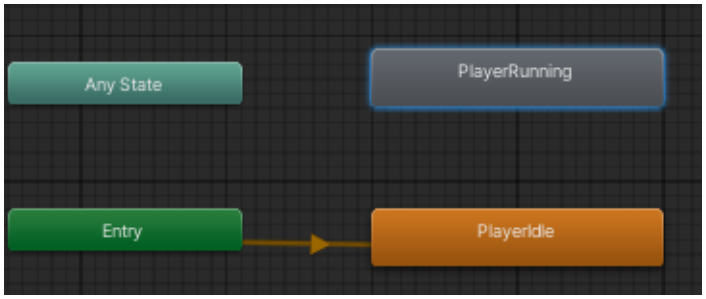


Once you are happy, press the Red button again to stop the recording.

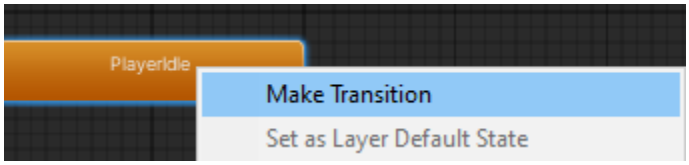


Animation Transitions - PlayerIdle to PlayerRunning

Click and drag **PlayerRunning** into the **Player** animator window.

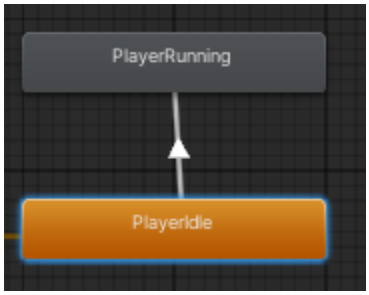


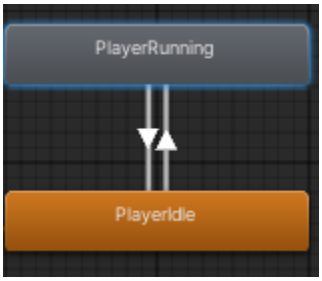
Right-click **PlayerIdle** in the animator window and click **Make Transition.**



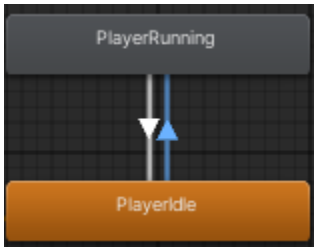
Click on **PlayerRunning**. Do the same for **PlayerRunning** (Rightclick **PlayerRunning > Make Transition).**

This indicates that PlayerIdle can transition to the PlayerRunning clip and vice versa.



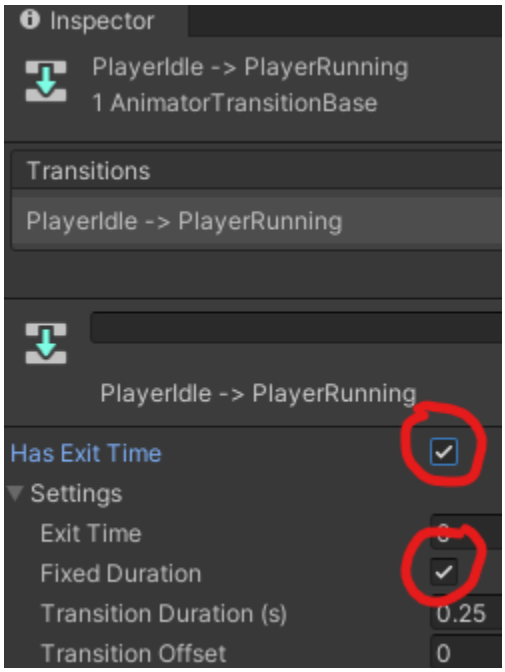


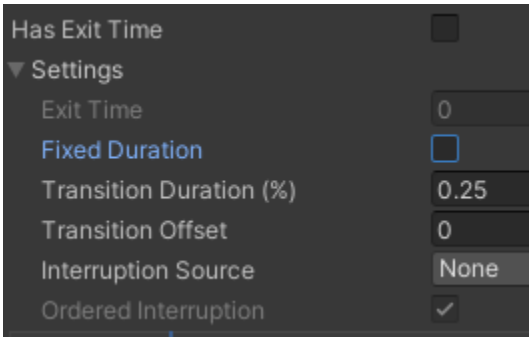
Click on the transition arrow.



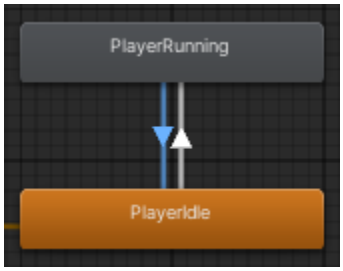
In the **Inspector,** expand **Settings** and uncheck **Has Exit Time** and **Fixed Duration**.

This removes the frames in between each transition and allows the transitions to happen instantly instead of waiting for each clip to finish before transitioning to the next.



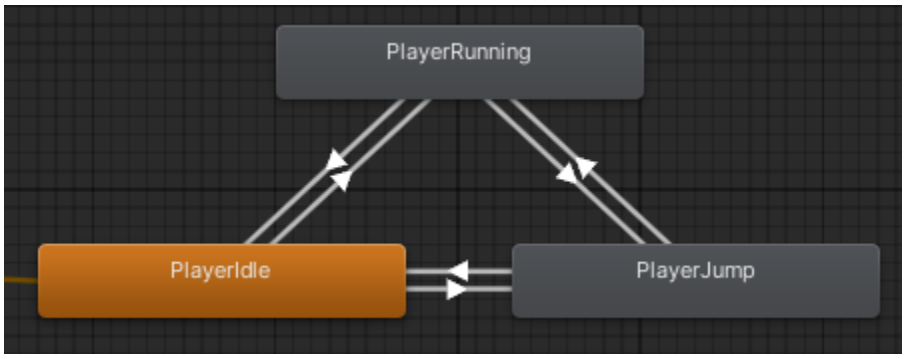


Do the same for the other transition.



Do the same if you have a running and/or falling animation.

Make sure you have the appropriate transitions for each situation.



Animator Conditions

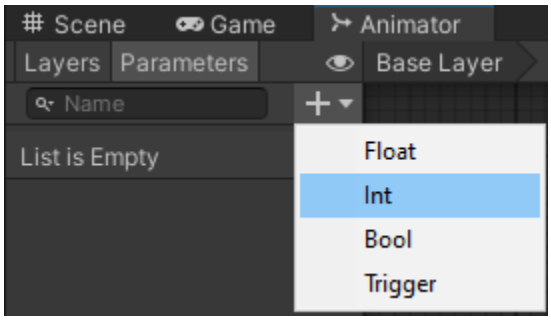
These will determine the conditions that will allow for the transition between **Animation** clips.
If a transition has a condition, it must first be met before being allowed to initiate.

In the **PlayerController** script, the animation playing is determined by the player's current state:

- **Idle** - 0
- **Run** - 1
- **Jump** - 2
- **Fall** - 3

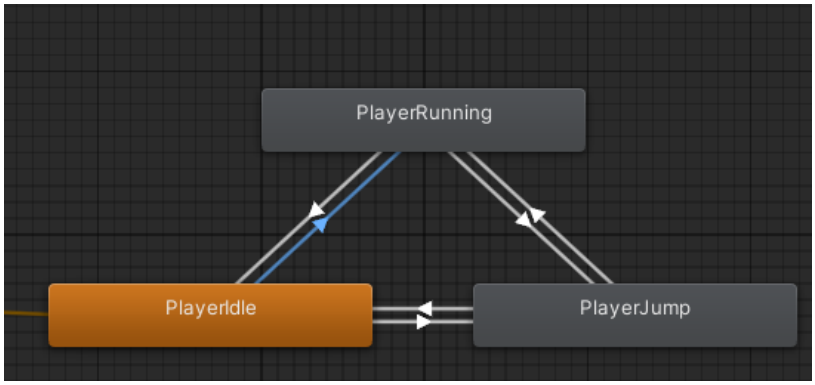
If the player does not press a button and is currently standing still, his state is **"Idle"** which is the equivalent to the number **0**.
If the player press space and jumps, his state is **"Jump"** which is the equivalent to the number **2** and will transition to the **PlayerJump** animation.

In the **Animator** window, click **Parameters >** '**+**' **> Int**.

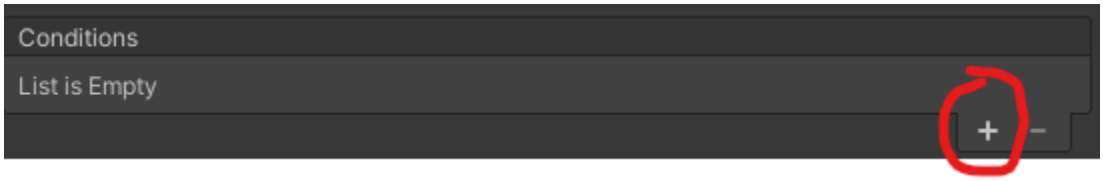


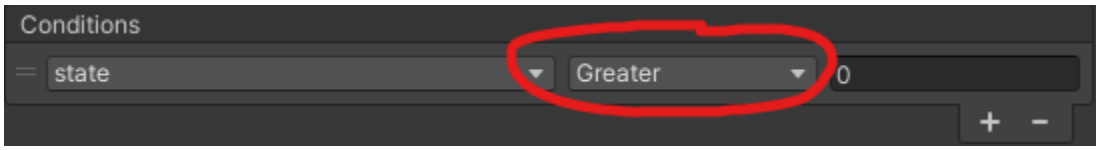
Rename it as '**state'**.

Click on the transition from **PlayerIdle** to **PlayerRunning.**

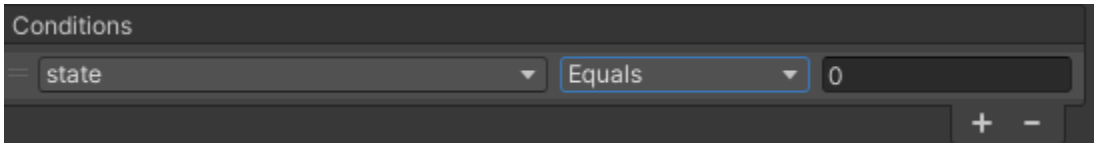


In the **Inspector**. Click the '**+**' icon to add a new transition condition that must be met.

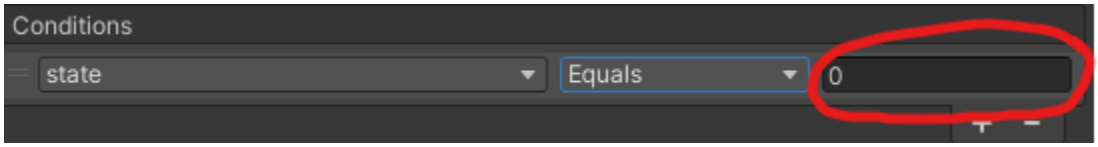


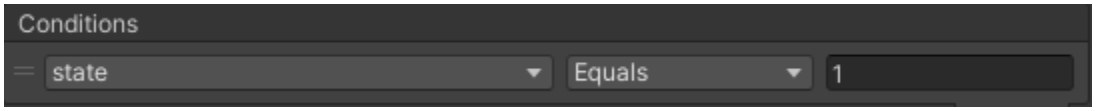


Change the dropdown from **Greater** to **Equals**.



Change the number from **0** to **1**.

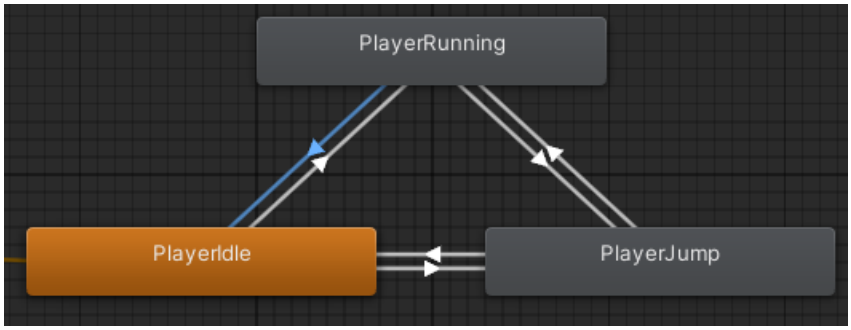




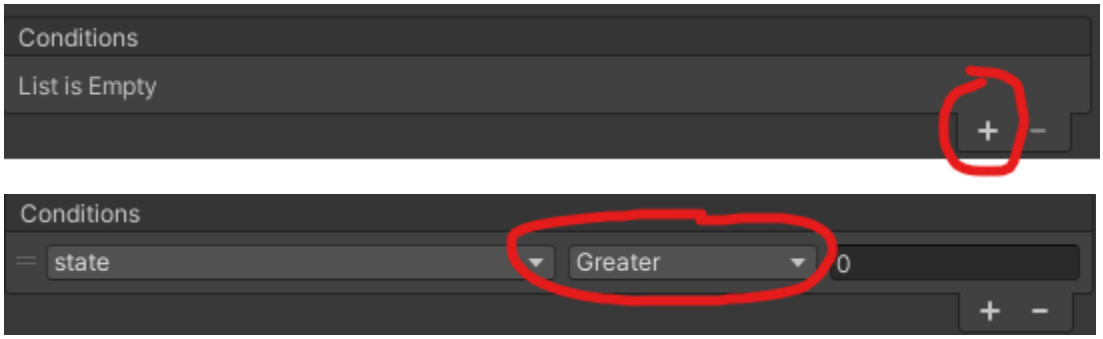
***Note - This means, the character's current state must be "1", also known as "Run", in order for the transition of the "PlayerIdle" animation to change to "PlayerRunning".**

In other words, the player must be currently running in order for the animation "PlayerRunning" to be played. The number 1 is a way to indicate that in the **PlayerController C# script.**

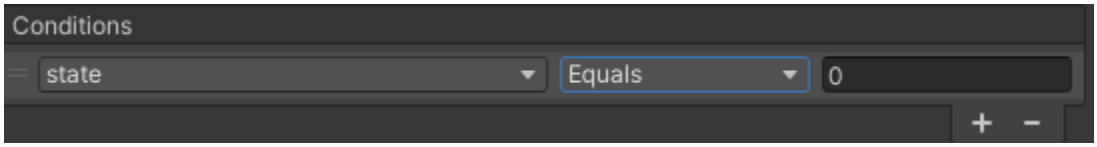
Click on the transition from **PlayerRunning** to **PlayerIdle.**



In the **Inspector**. Click the '**+**' icon to add a new transition condition that must be met.



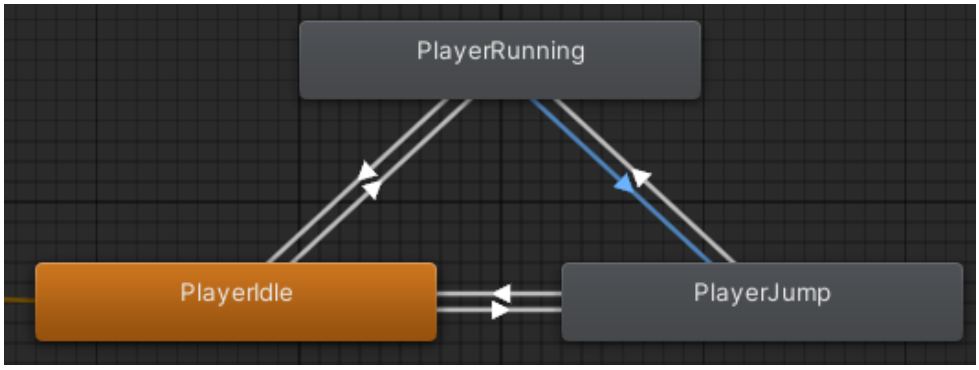
Change the dropdown from **Greater** to **Equals**.



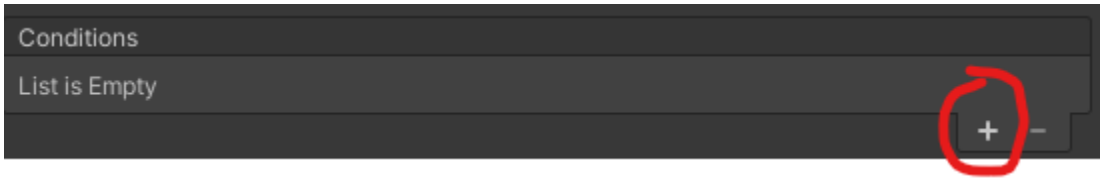
***Note - This means, the character's current state must be "0", also known as "Idle", in order for the transition of the "PlayerRunning" animation to change to "PlayerIdle".**

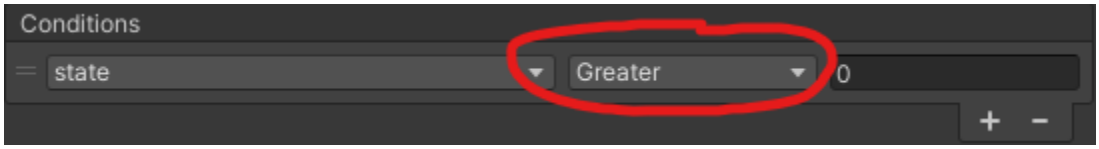
In other words, the player must be currently idle/standing still in order for the animation "PlayerIdle" to be played. The number 0 is a way to indicate that in the **PlayerController C# script.**

Click on the transition from **PlayerRunning** to **PlayerJump.**

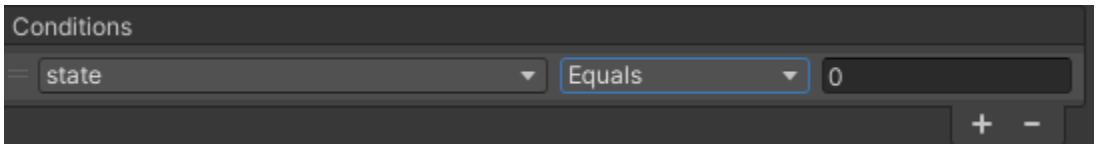


In the **Inspector**. Click the '**+**' icon to add a new transition condition that must be met.

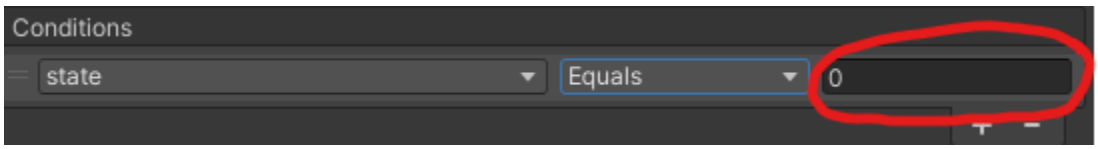


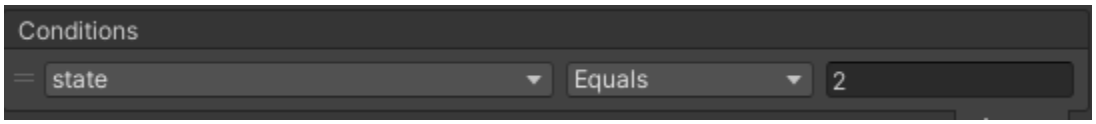


Change the dropdown from **Greater** to **Equals**.

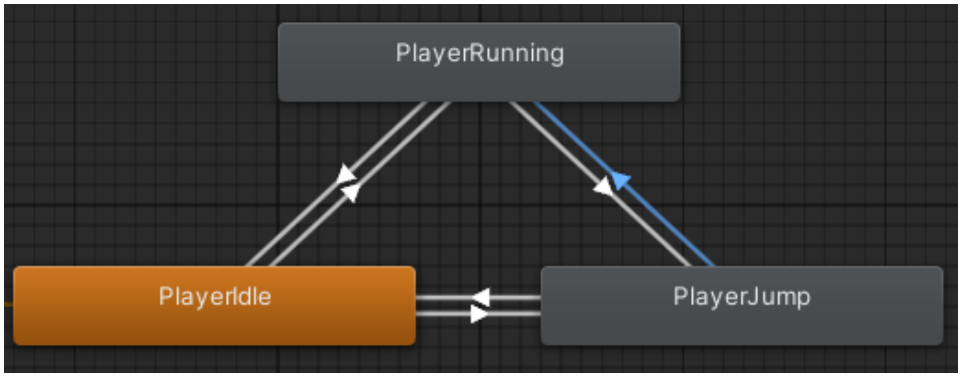


Change the number from **0** to **2**.

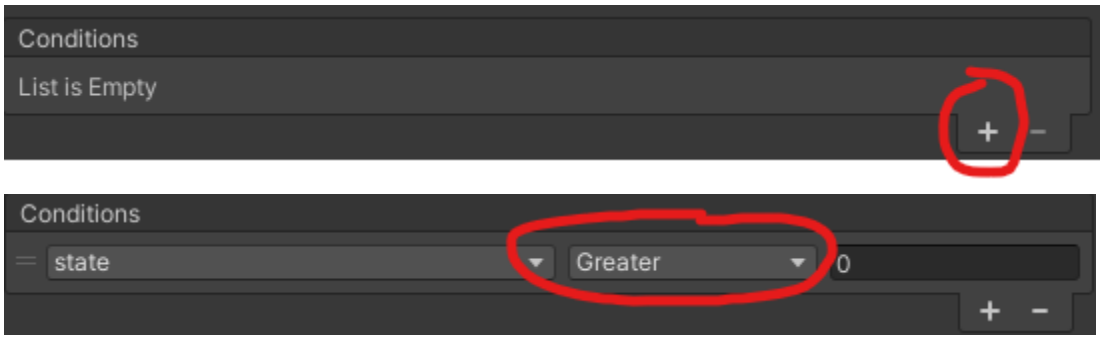




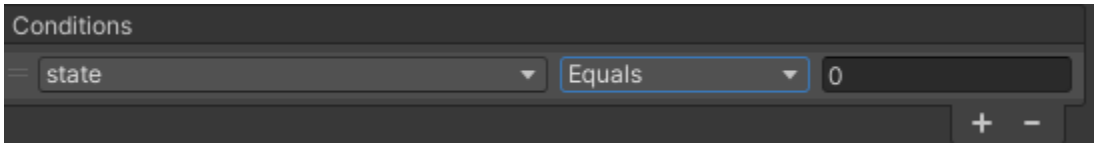
Click on the transition from **PlayerJump** to **PlayerRunning.**



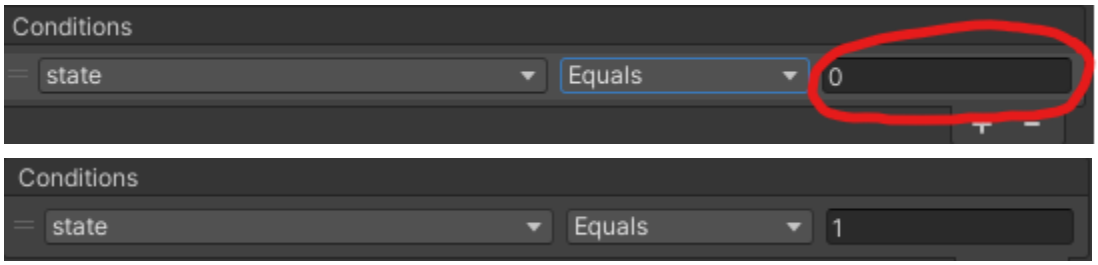
In the **Inspector**. Click the '**+**' icon to add a new transition condition that must be met.



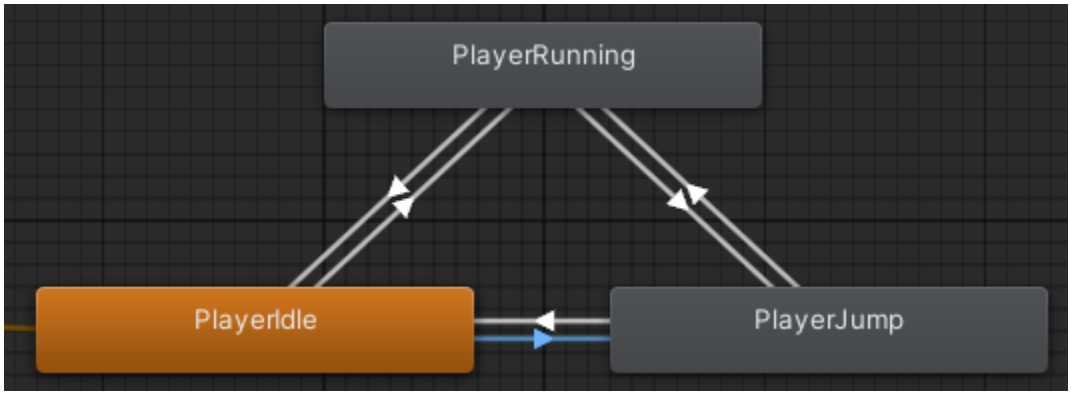
Change the dropdown from **Greater** to **Equals**.



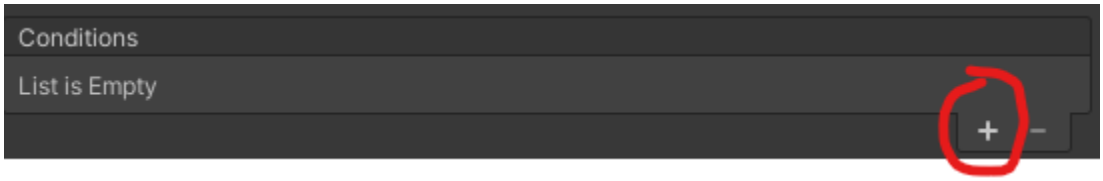
Change the number from **0** to **1**.

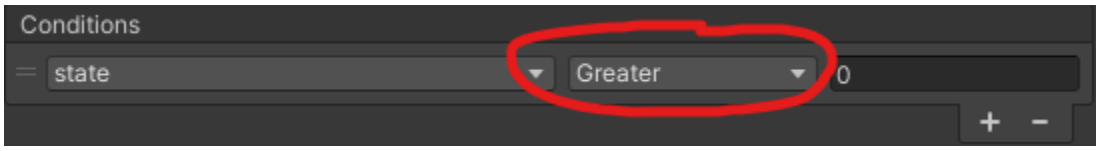


Click on the transition from **PlayerIdle** to **PlayerJump.**

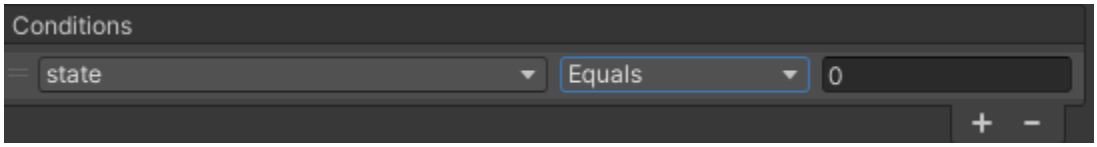


In the **Inspector**. Click the '**+**' icon to add a new transition condition that must be met.

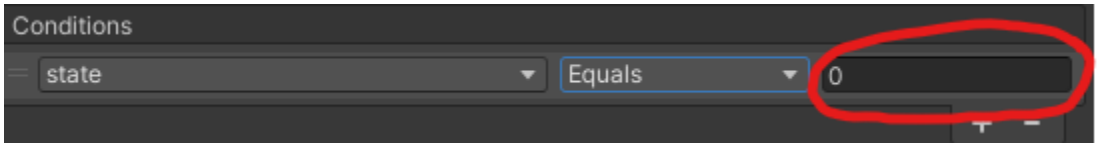


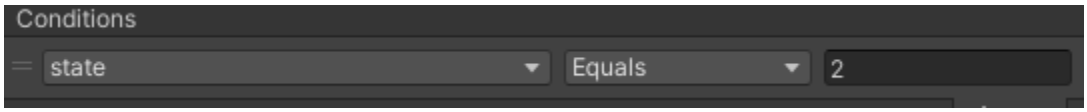


Change the dropdown from **Greater** to **Equals**.

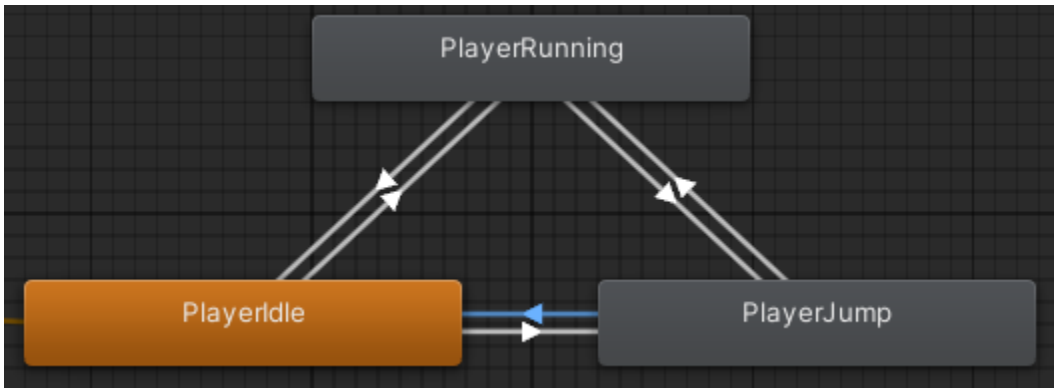


Change the number from **0** to **2**.

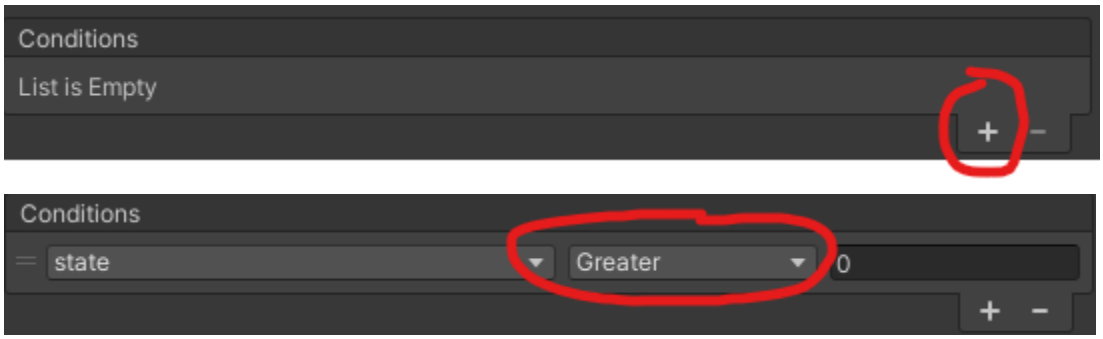




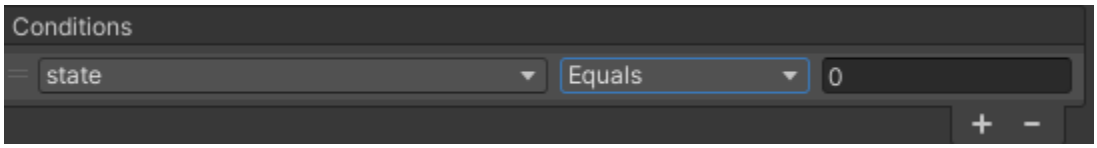
Click on the transition from **PlayerJump** to **PlayerIdle.**



In the **Inspector**. Click the '**+**' icon to add a new transition condition that must be met.

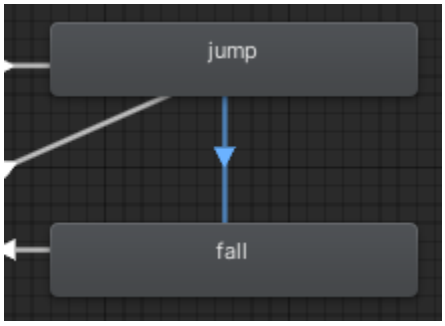


Change the dropdown from **Greater** to **Equals**.

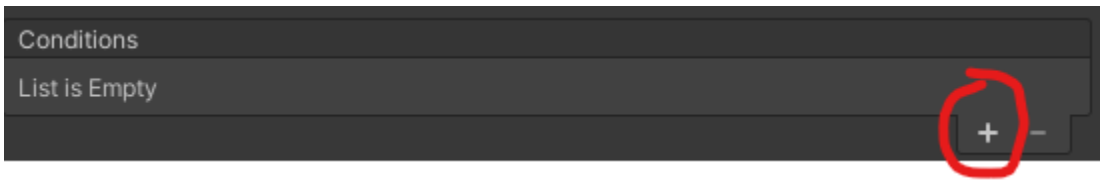


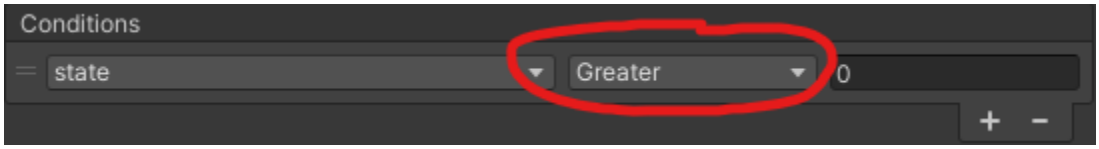
If you have a falling animation.

Click on the transition from **PlayerJump** to the falling animation file.

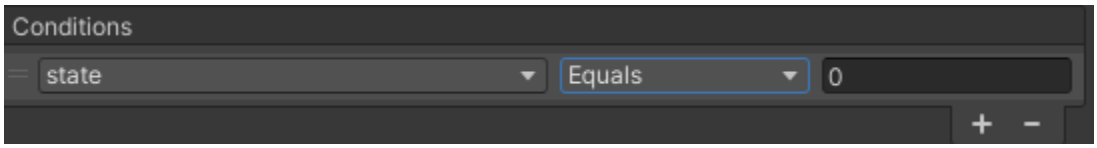


In the **Inspector**. Click the '**+**' icon to add a new transition condition that must be met.

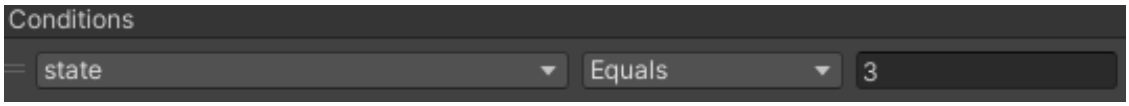




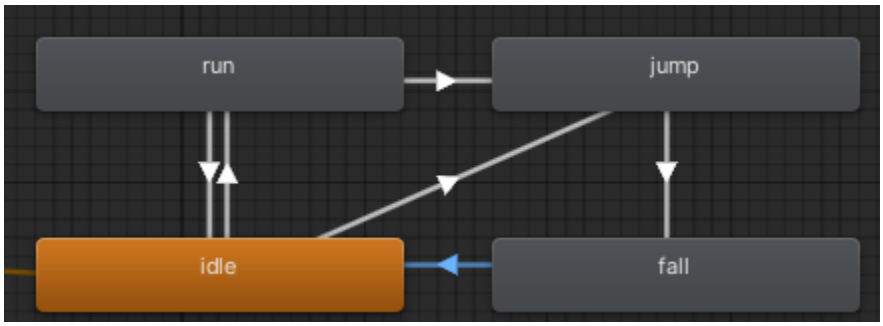
Change the dropdown from **Greater** to **Equals**.



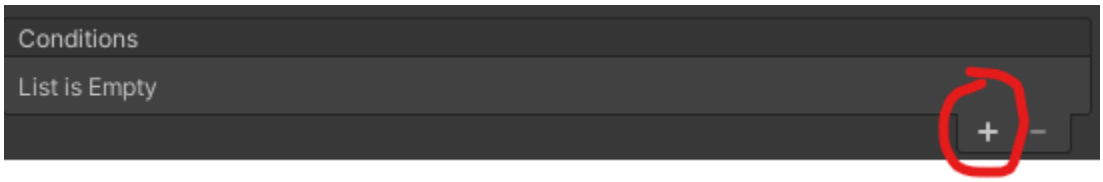
Change the number from **0** to **3**.

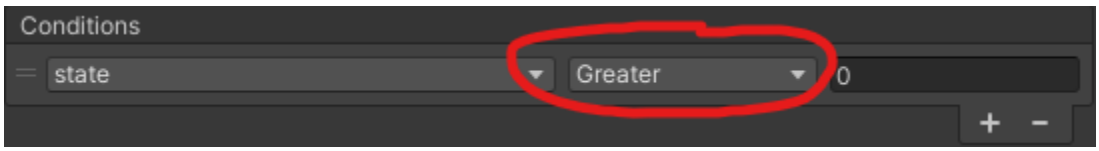


Click on the transition from **PlayerJump** to **PlayerIdle.**

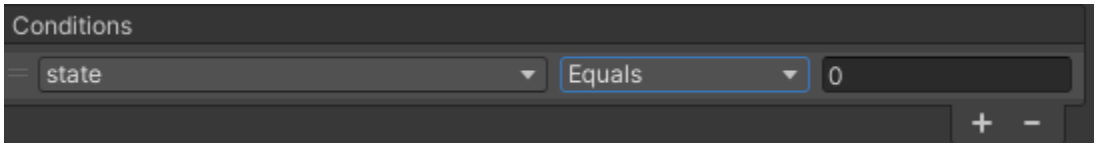


In the **Inspector**. Click the '**+**' icon to add a new transition condition that must be met.





Change the dropdown from **Greater** to **Equals**.



PlayerController

Adding PlayerController to the ‘player_0’ Gameobject

Click and drag **PlayerController** to the main player object.

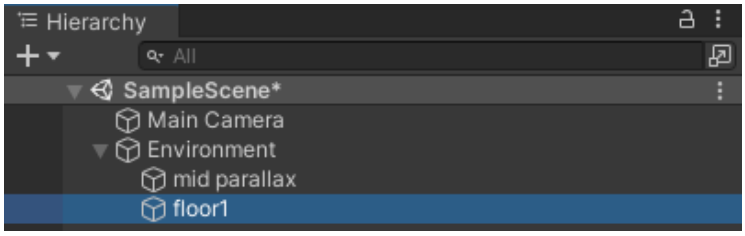
Variables:

- **Ground** (Layer) - Foreground Layer
  - ***Players only jump if they are touching the ‘Floor’ layer***
- **Speed** (float) - the movement speed of the player.
- **Jump Force** (float) - the player’s jump power
- **Max Health** (float) - how much health the player has at the beginning

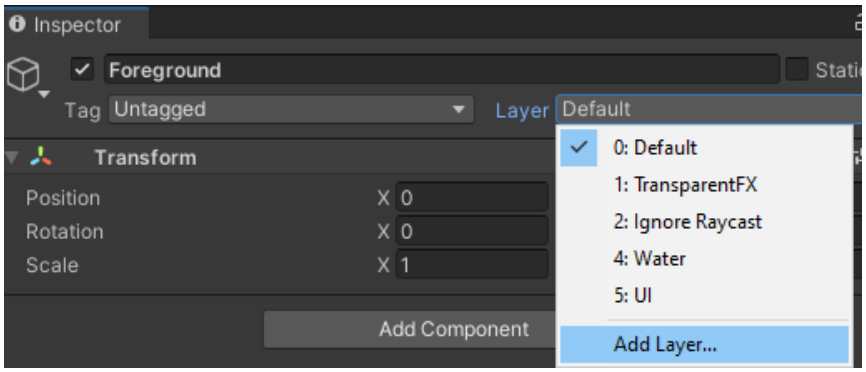
Experiment with the Speed and Jump Force to see what best suits your game.

Adding Layer to Foreground - To allow Player to jump

Click on your foreground game object in your Hierarchy. In this case, the example foreground is called ‘**floor1**’

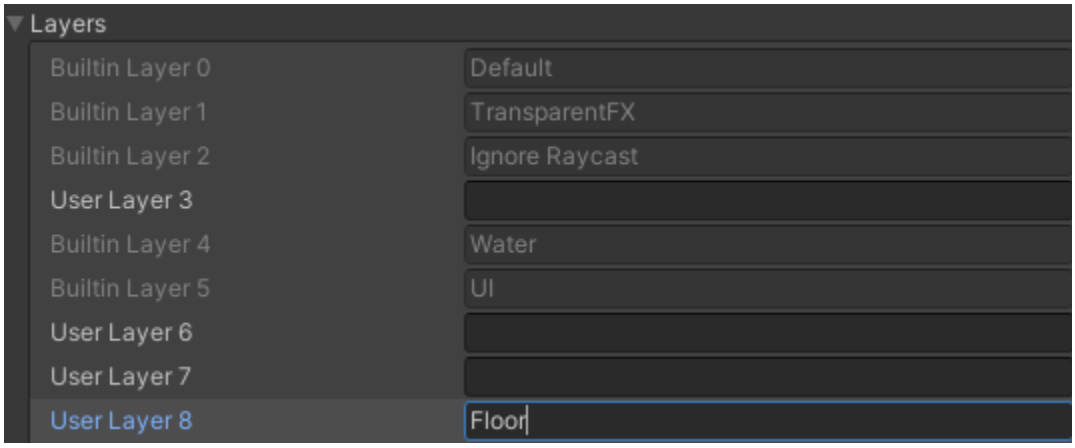


Click on ‘Add Layer…’.

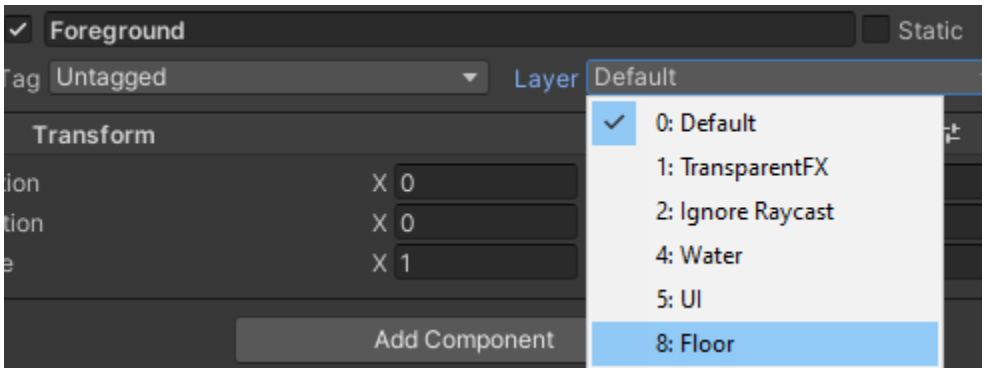


Type in ‘Floor’ in a ‘User Layer’ 8 or higher.

We use layer 8 or higher because the Unity Editor utilises layers 0 - 7 for default in-built settings.



Click on the game object again and in 'Layers', choose the new layer.



Go back to the **PlayerController** component in the player object **Inspector** and change **Ground = Nothing** to **Ground = Floor**.

The C# script only allows the player to jump if it is touching the layer, in this case, 'Floor'.

PlayerHealthUI

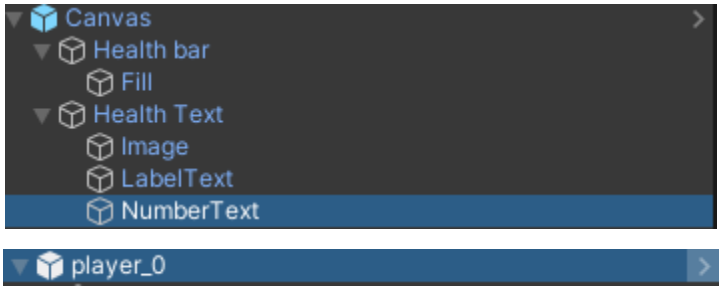
***Note - Requires a Canvas with UI objects.**

Overview

Calculates and sets current player health and player score to UI game objects. Visualises and drains the player health bar. The health bar colour changes from Max value colour to Min value colour.

Required objects:

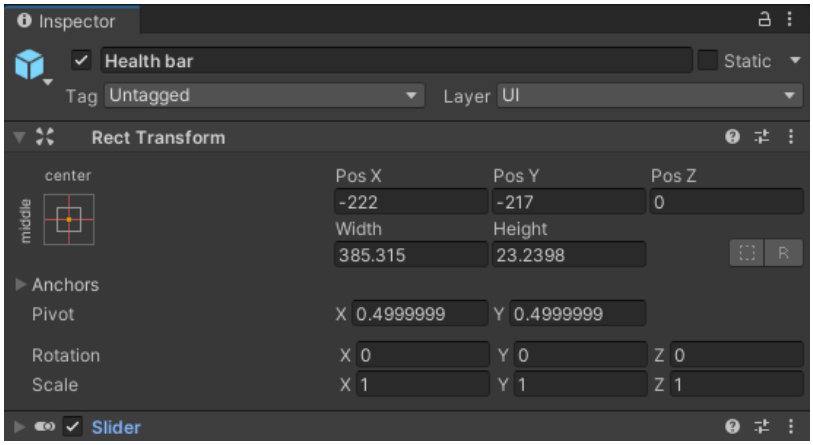
- Health bar object with a ‘slider’ component
- Image object that fills the bar
- Text object that indicates the player’s health
- Player game object - with **PlayerController**



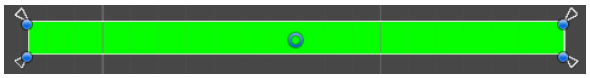
Click and drag **PlayerHealthUI** into the main player object.

Variables:

- **Slider** - game object that has the ‘Slider’ component and is the health bar background.



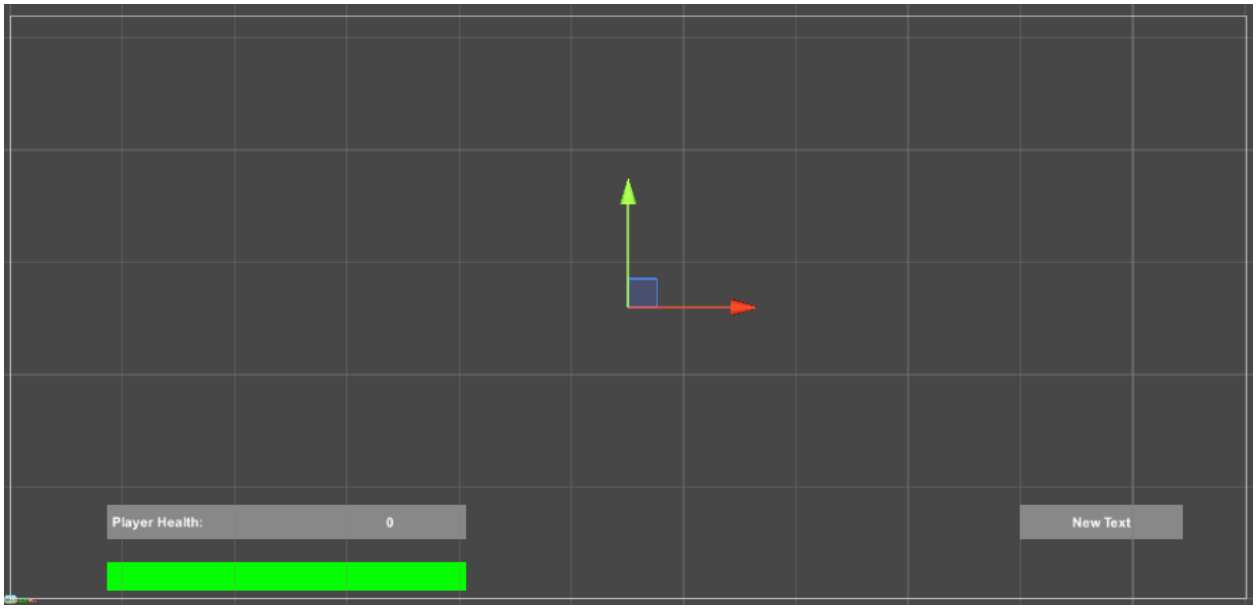
- **Fill** - the Image object to the size of the Slider game object. The colour and how the health bar will be visualised



- **Full Health Colour** - the health bar colour at max value
- **Zero Health Colour** - the health bar colour at min value
- **Health Number Text** - Text object

- **Player Script** - PlayerController
    - ***Drag the player object from the 'Hierarchy' and it will search for the PlayerController attached to that object.***

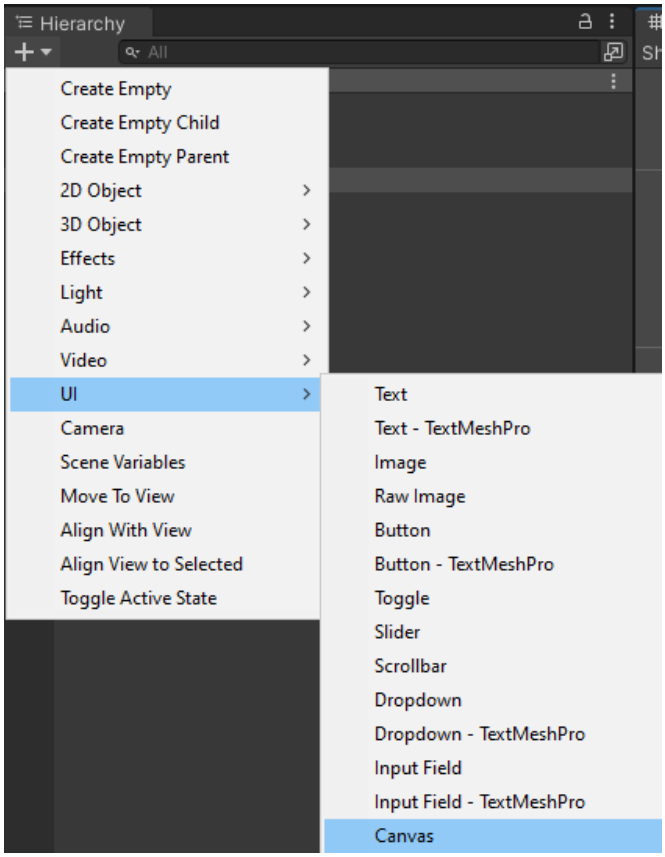
Final UI Image



Making a UI - Character Healthbar

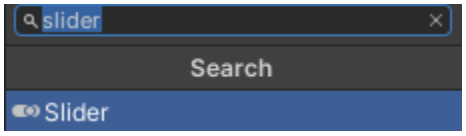
In the **Hierarchy**, click **'+' > UI > Canvas**.

All UI elements must be in a canvas.

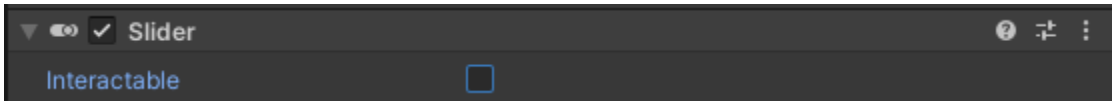


Create a child game object in that canvas. Right-click **Canvas > Create Empty**. Rename to **Healthbar** for convenience identification purposes.

Click on **Healthbar** and in the **Inspector**, click **Add Component**, search **Slider** and add '**Slider**'.

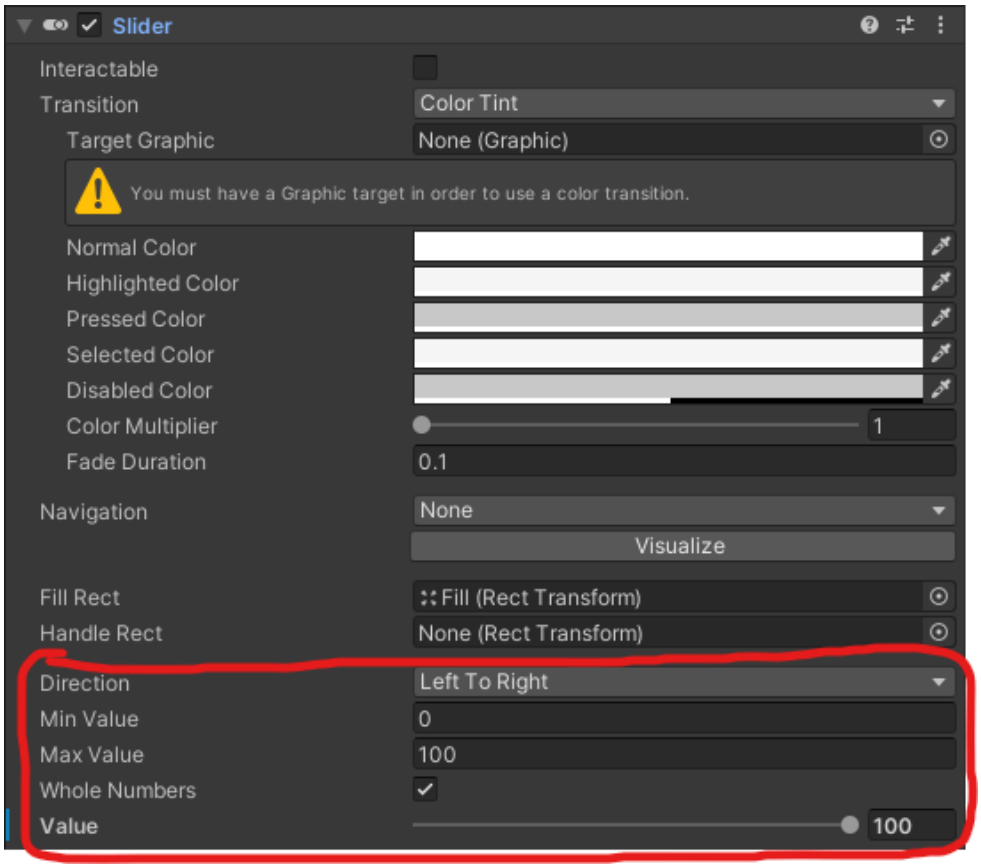


Untick **Interactable**.



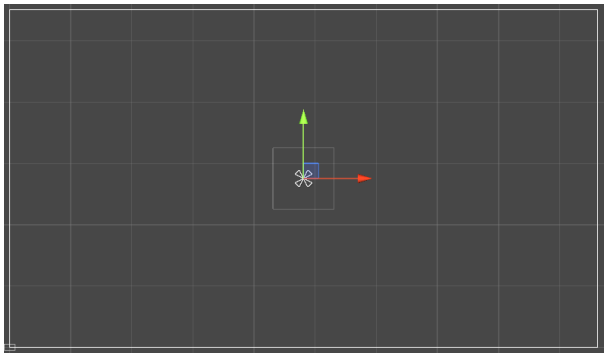
Slider Variables

- Direction - the direction the colour of the health bar will decrease to
- Min Value - Minimum player health
- Max Value - Maximum player health
- Value - the current value

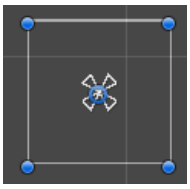


Change the **Max value** to the **Maximum player health** and **Min Value** to **Minimum Player Health**.
Increase **Value** to the maximum value.

Click on the **Healthbar** object in the **Hierarchy**, hover over the **Scene** window and press the **F** key on your keyboard to move and look at the game object.

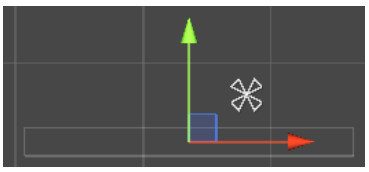


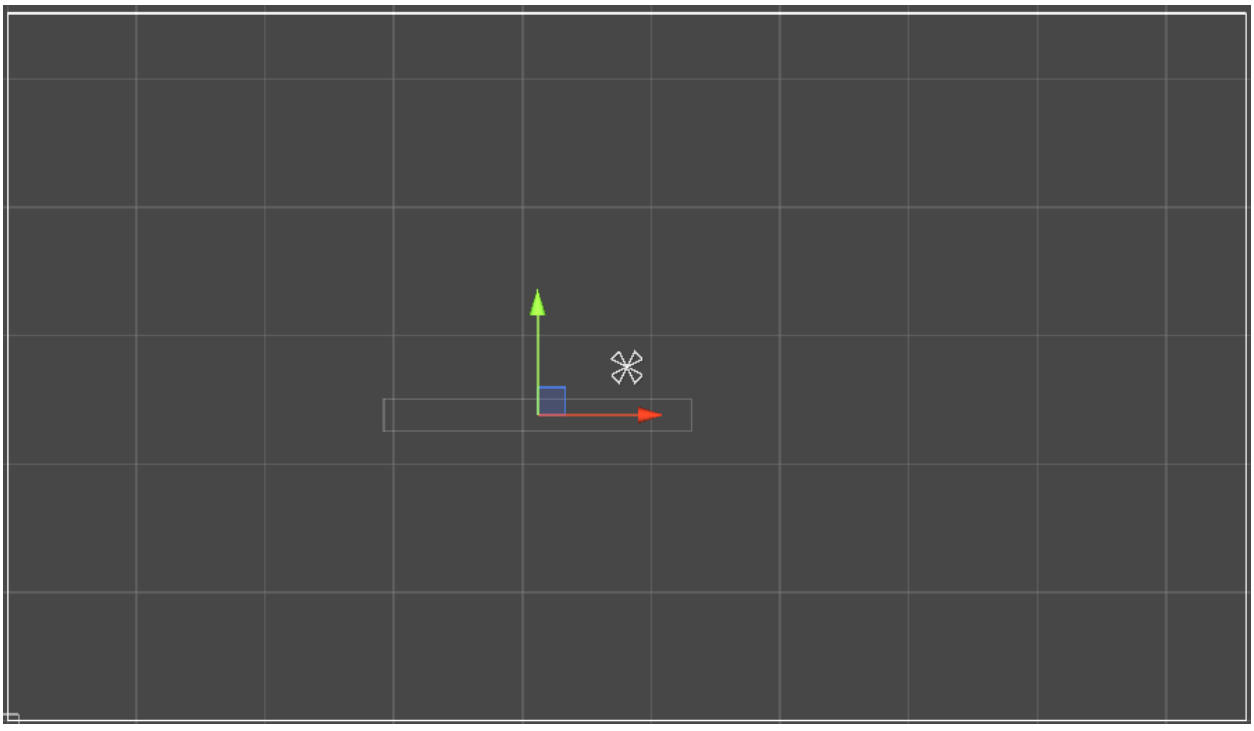
Click on the **Rect tool** at the top left above the **Hierarchy** to start adjusting its size.

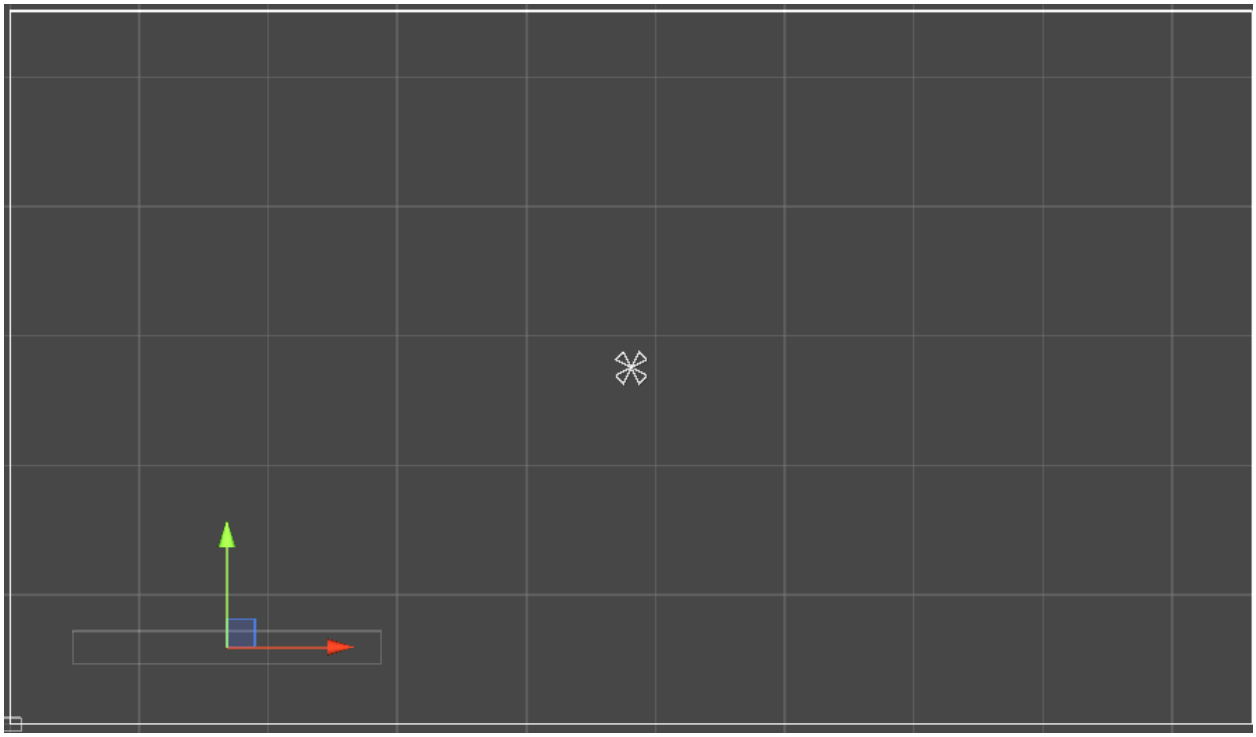


Adjust the size by either using the corners or the edges.

Click on the **Move tool** to start moving the game object.

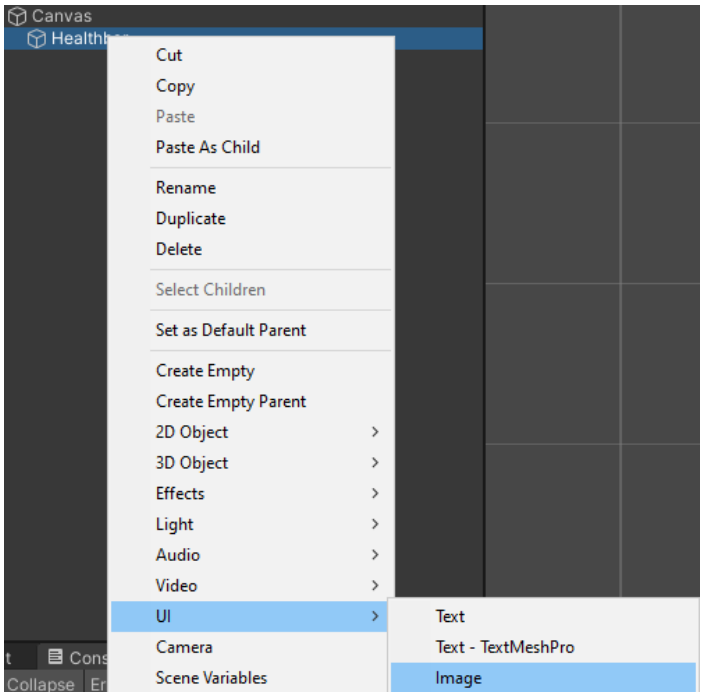


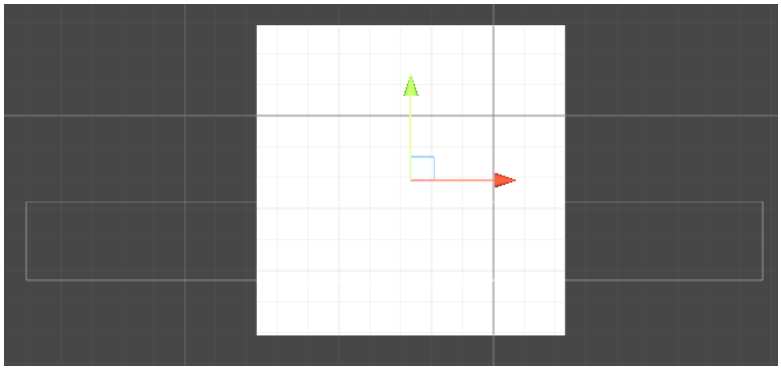




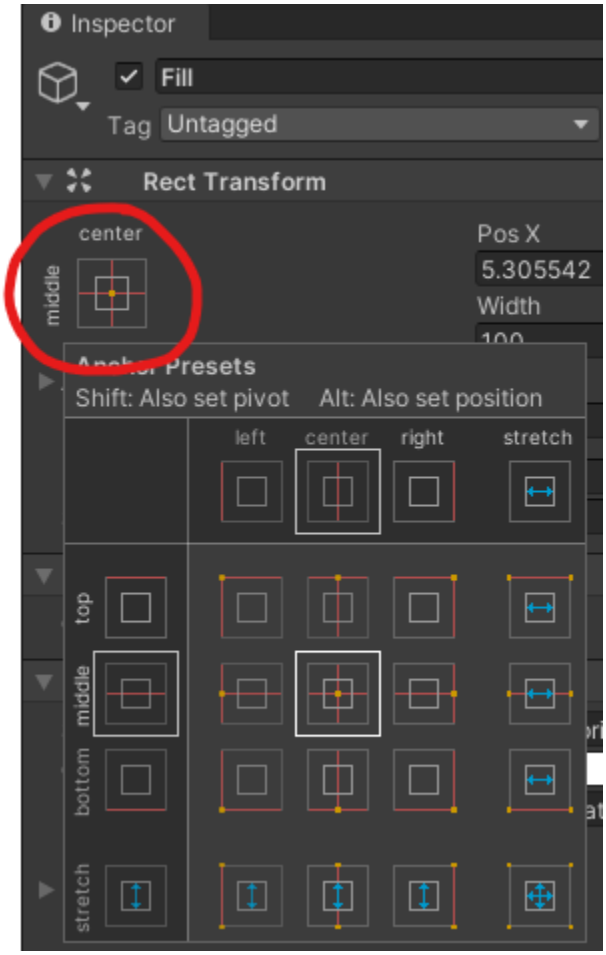
The canvas is ‘Main camera’ relative, meaning it will follow the camera and not stay still.

We will now make the **Image** game component to fill the **Healthbar**.
Rightclick **Healthbar > UI > Image**. Rename it to **Fill**.



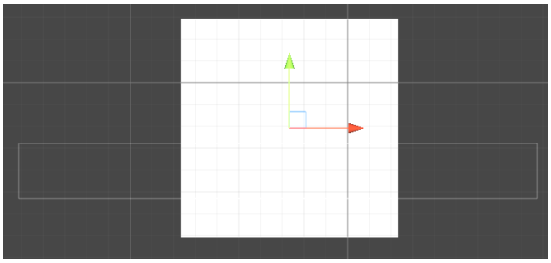


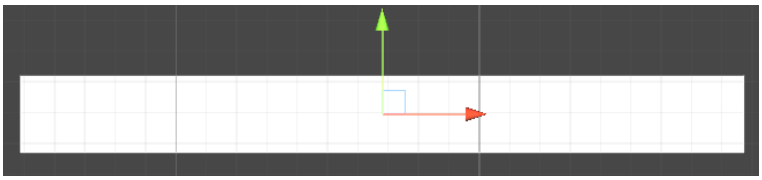
Click on the game object in the **Hierarchy,** '**Fill',** and in the **Inspector,** click the icon under **Rect Transform.**



Hold **Alt** on your keyboard and click the bottom right icon.

This will change the object's size to fill the edges of its parent object.

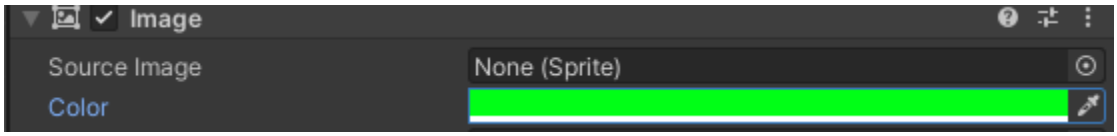




Before **After**

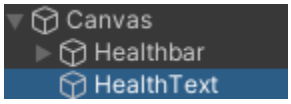
Change the colour of the image to be the colour at player maximum health.

Note down the Hexadecimal for that colour for a later script, PlayerHealthUI.

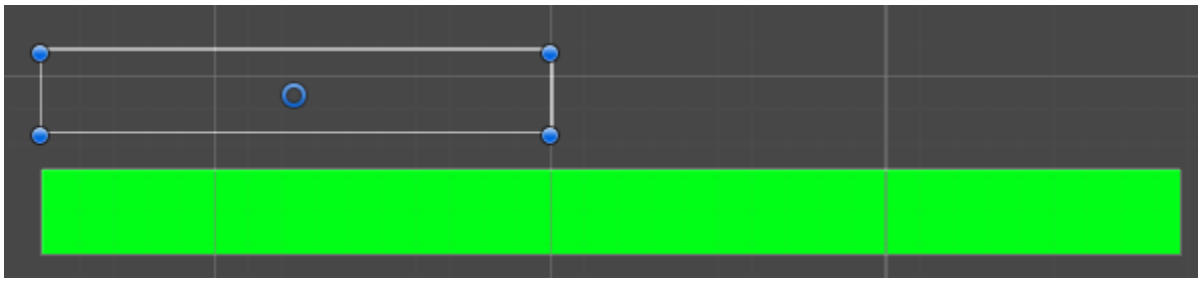


Making a UI - Character Health Text

Make a new game object child for **Canvas**. Right-click **Canvas > Create Empty.**



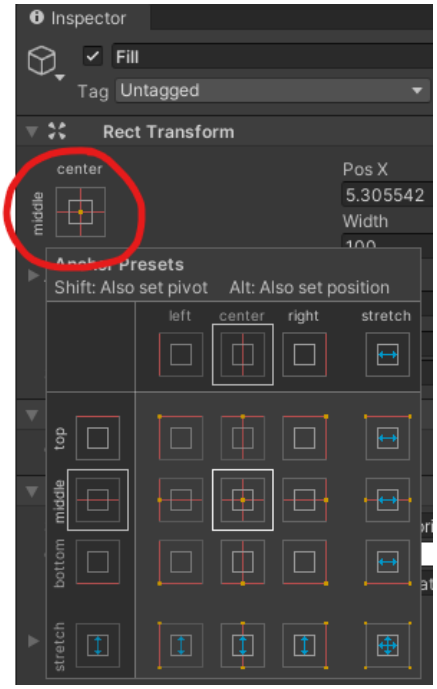
Drag and resize above the healthbar.



Make a new **Image** game object child for **HealthText**. Right-click **HealthText > UI > Image.**

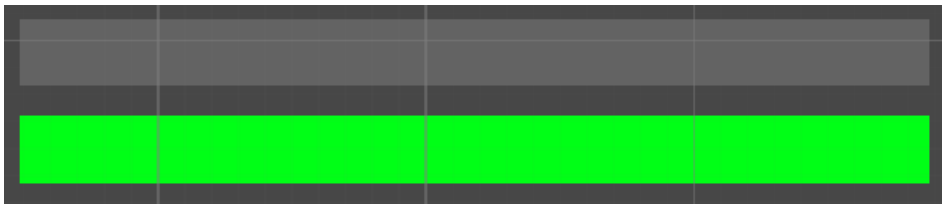
Rename to **Background,** change colour above the health bar. **This will be the background for the Text.**

Click on the game object in the **Hierarchy,** '**Fill',** and in the **Inspector,** click the icon under **Rect Transform.**

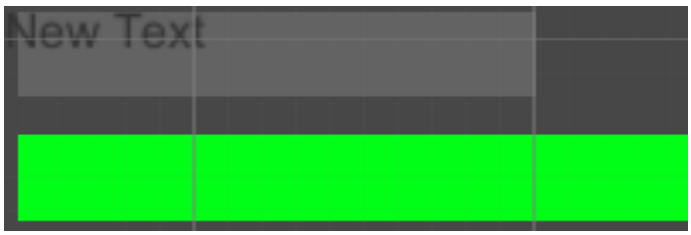


Hold **Alt** on your keyboard and click the bottom right icon.

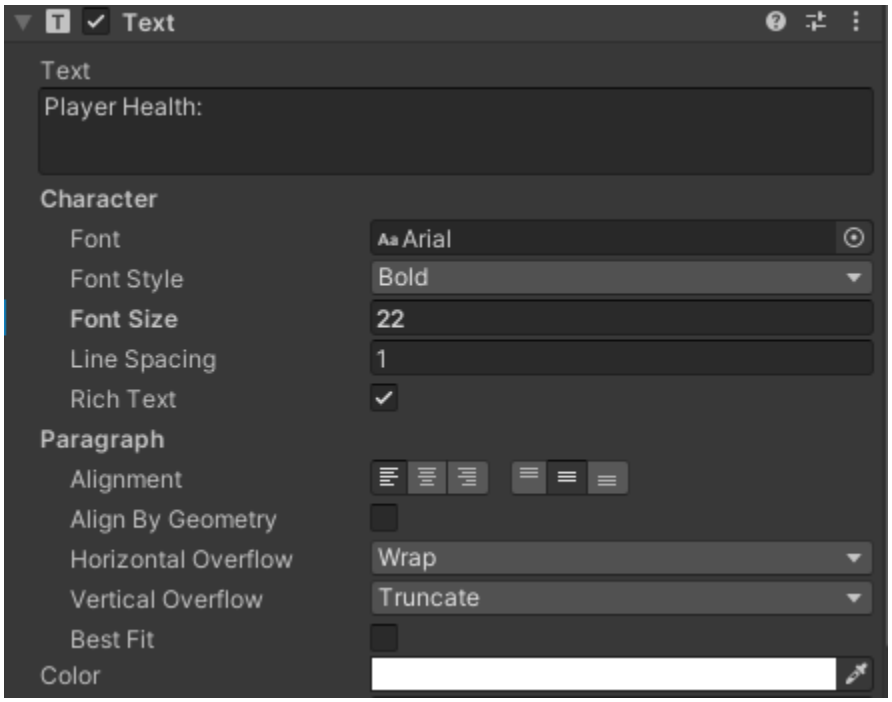
This will change the object's size to fill the edges of its parent object.



Make a **Text** game object. Right-click **Healthtext** in the **Hierarchy > UI > Text**. Rename to **Label.**

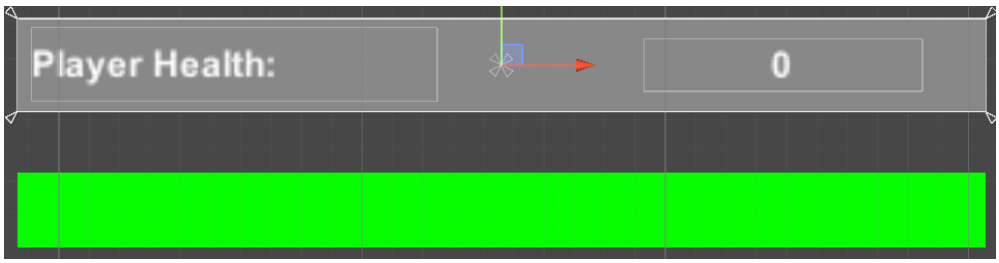


Click on the object **Label**. In the **Inspector**, change the text size, alignment, colour to your liking. **You may also change the font style.**



Change the text's **Text** to "**Player Health:**".

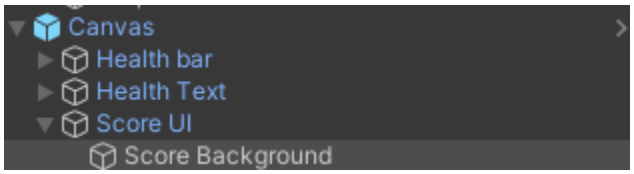
Do the same for the number text value which will display the player's **health** as a text value. **You may also change the font style.**



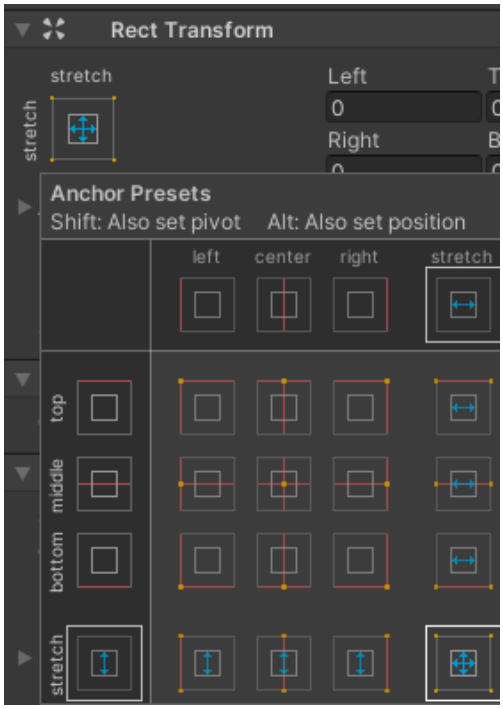
Making a UI - Character Score UI

Create an empty child game object in **Canvas**. Rename it to **Score UI**.

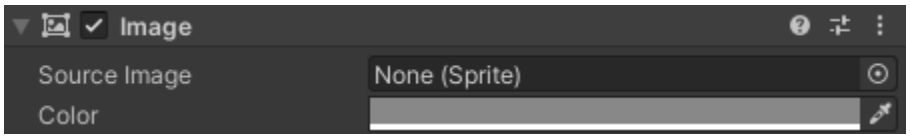
Make an **Image** object child inside **Score UI**. Right-click **ScoreUI > UI > Image**. This will be the background for the Score interface.



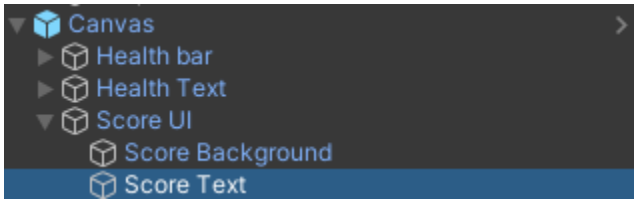
Change its **Rect size**.



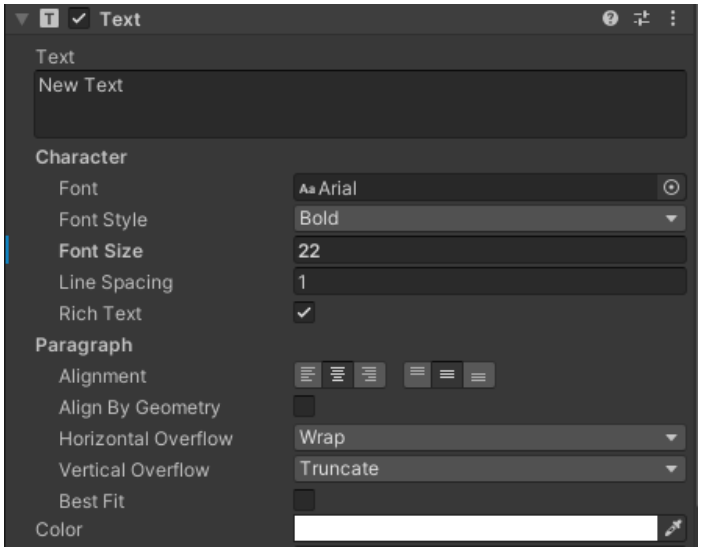
Change the colour of the background.



Add a **Text** object child inside **ScoreUI** for the display score number.



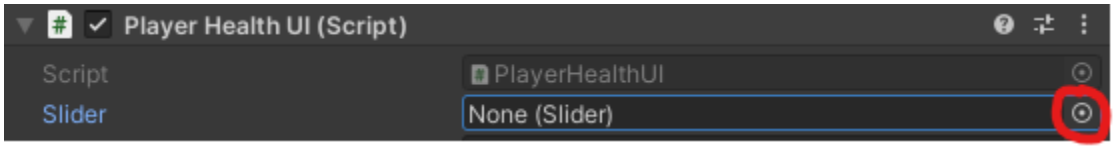
Change the colour, text alignment, and size. **You may also change the font style.**



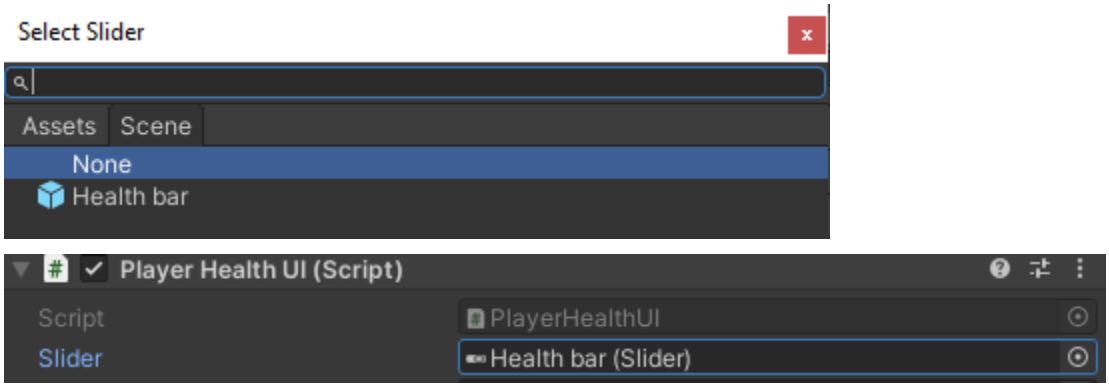
Attaching Game Objects to a C# Script component

Slider

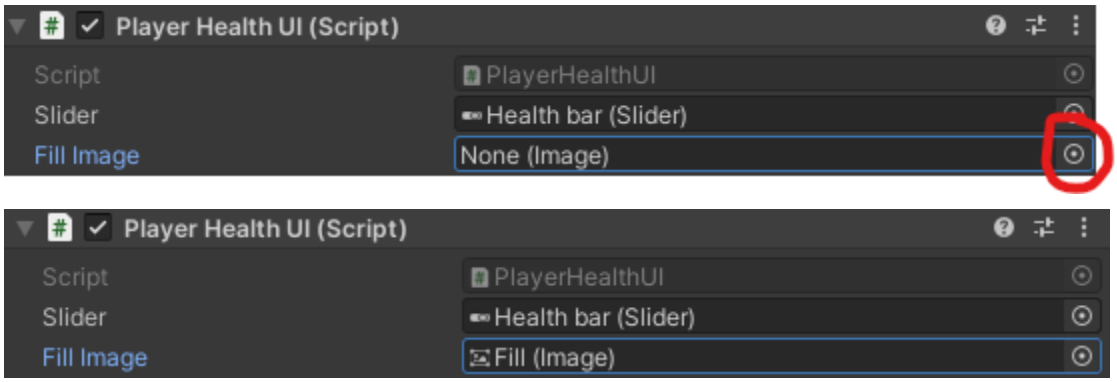
Click the **Circle** button on the right of the variable.



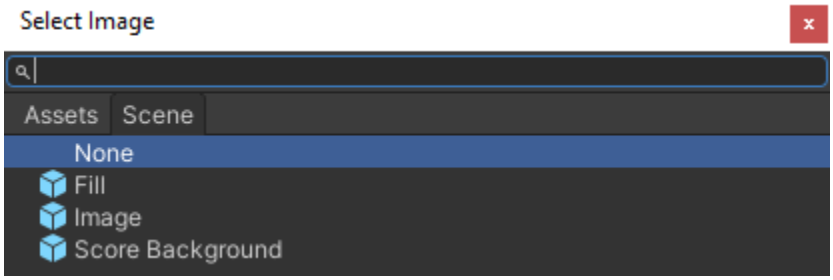
Click on the desired game object. In this case, double click '**Health bar'.**



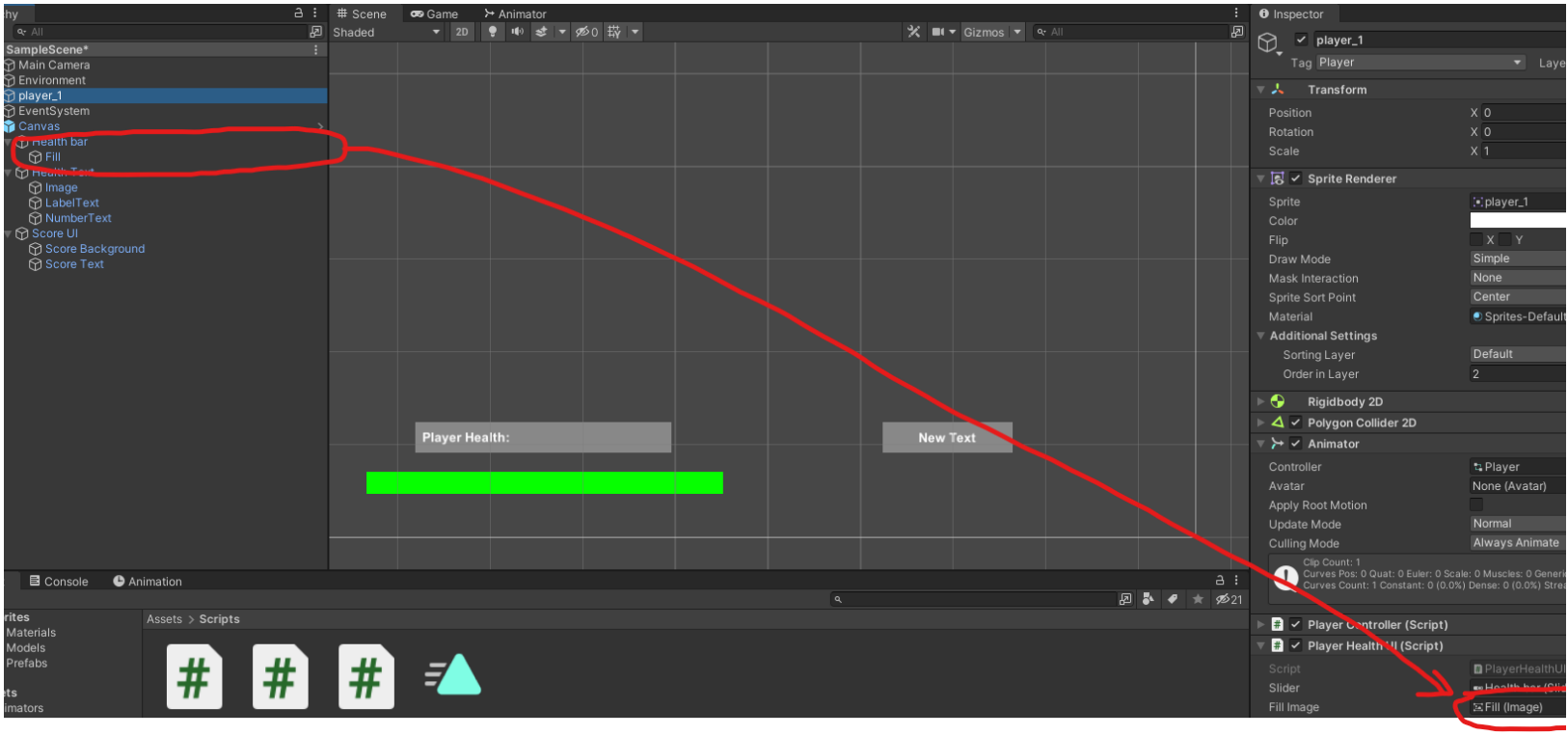
Click on Fill image and choose **Fill**.



Sometimes there may be multiple game objects and you may not know which is the correct one.

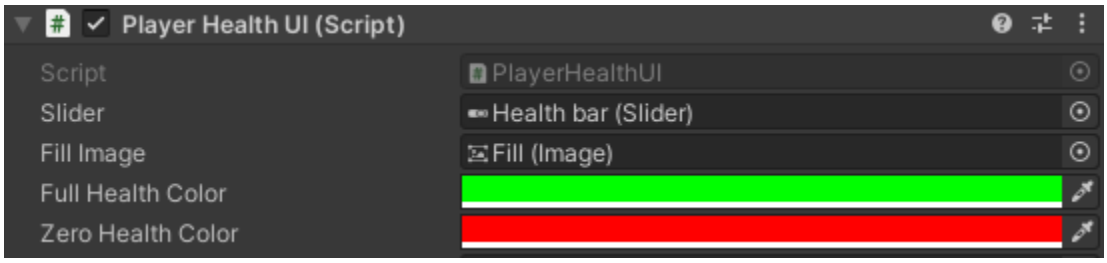


If that is the case, drag the desired object from the Hierarchy into the variable section.



Change the Full health colour to the colour that was chosen for the **Fill**.

Choose and change the colour for when the player's health reduces to 0.



Find and select **NumberText** for **HealthNumberText.**
Click and drag the game object **OR** choose the game object by clicking the **Circle** button.

Click and drag the player game object from the **Hierarchy** into the variable section.

Enemy

Overview

Enemies will have Collider2D that deal damage to the player, detect when the player kills it, and the path detection to move back and forth. If placed on a platform, it will walk back and forth once it reaches the edge.

C# Scripts:

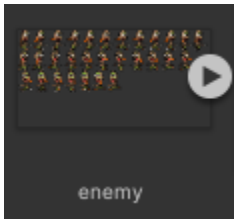
- EnemyMovement
- EnemyHealthAndDamage
- PlayerKillEnemy

Required Files:

- Sprites (image file) - ***only sprites of the player facing one side is required (either left or right, not both)***
- Animations (.anim)
- Animation Controllers/Animator (.controller)

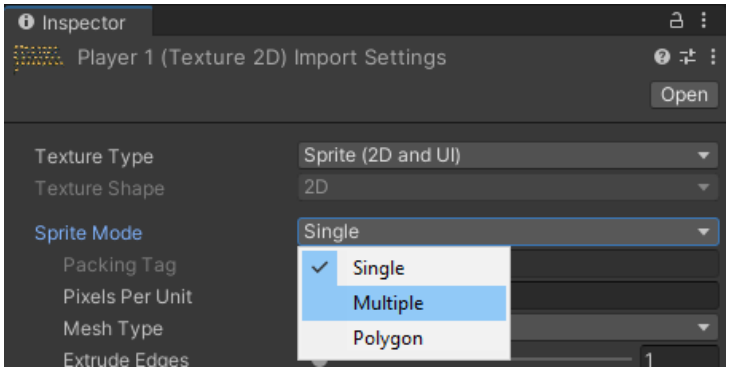
Enemy Sprites

In your Project window, click on the player sprite sheet.



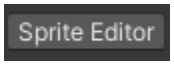
In the Inspector, click the dropdown for **Sprite Mode** and choose **Multiple.**

This allows for multiple crops and slices of the sprite sheet.

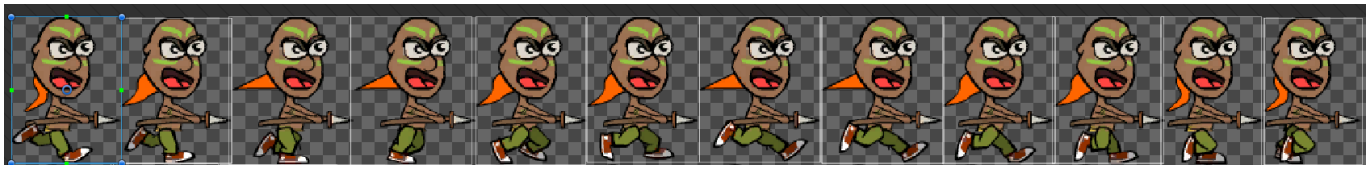


Click **Apply**

Click **Sprite Editor.**



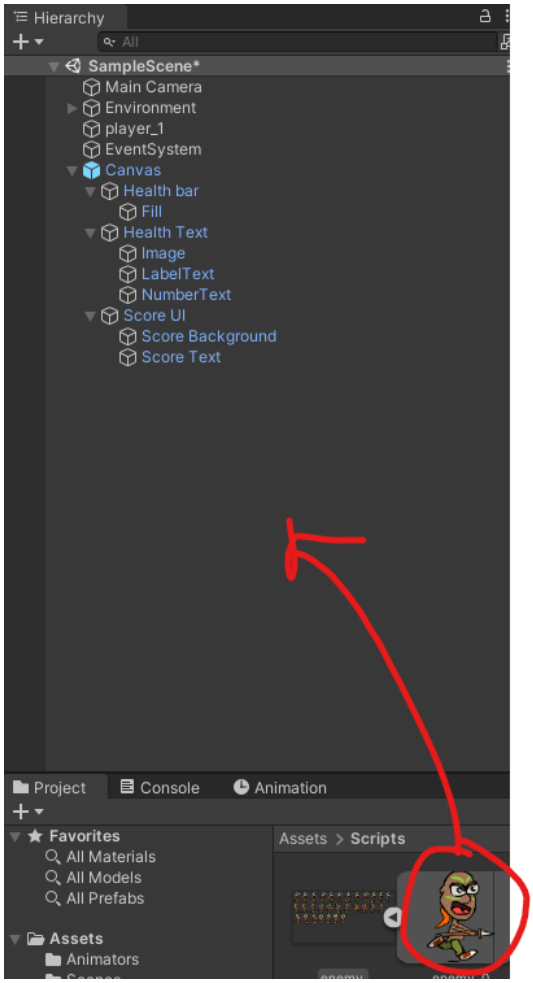
Click the **Slice** dropdown and **Slice** to automatically separate the sprites into different png images.



Make Enemy Gameobject

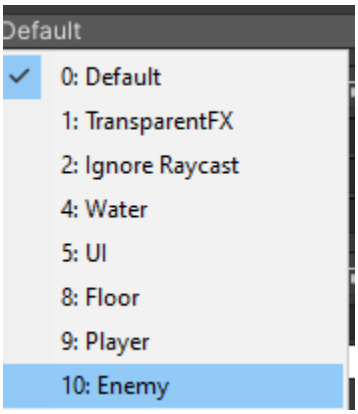
Go to your file with the player sprite. Click the arrow on the right to expand and see the sliced sprites.

Click and drag an image slice into the Hierarchy to create a game object instance of it.



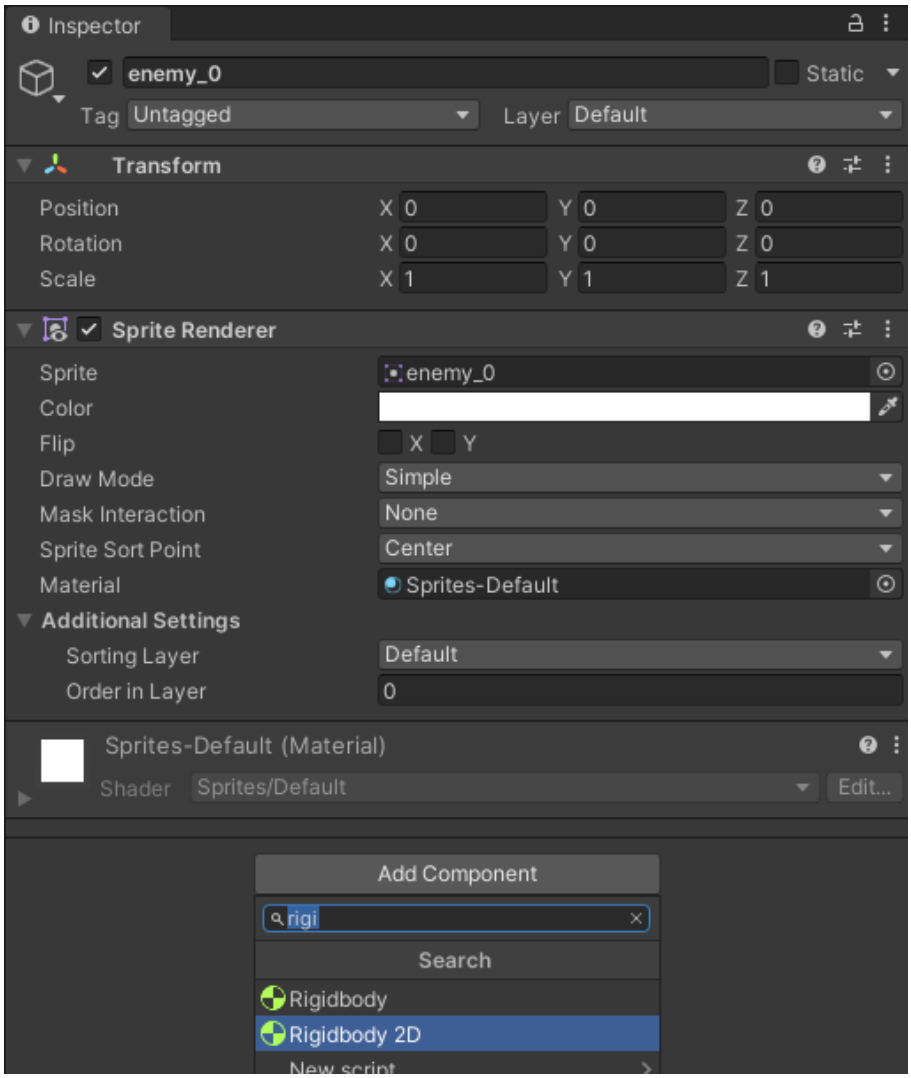
In the **Inspector**, click **Tag** and choose **enemy_0.**

Create a new layer with a number 8 or higher, name it **Enemy**, and choose the **Enemy** layer.



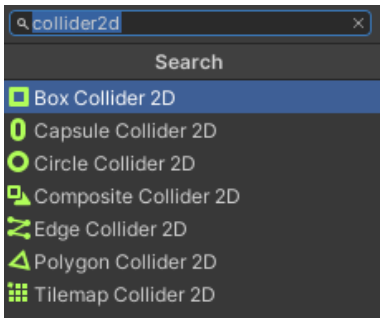
Add Component, search **Rigidbody2D** and add it.

Rigidbody2D adds physics to the game object.



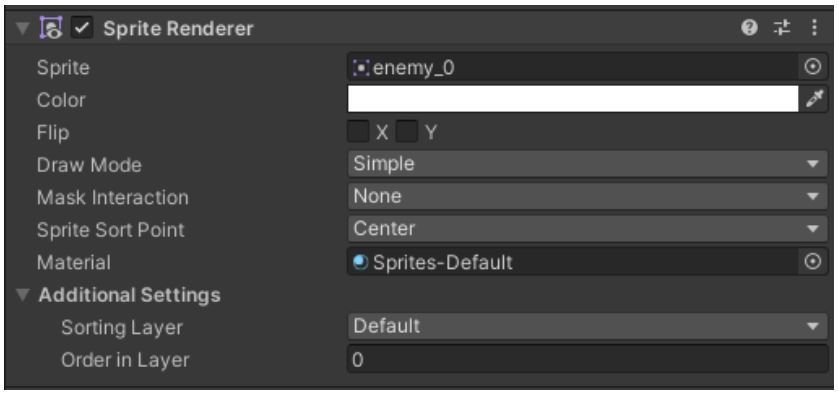
Add Component, search **Collider2D** and add a collider of your choice for the Player.

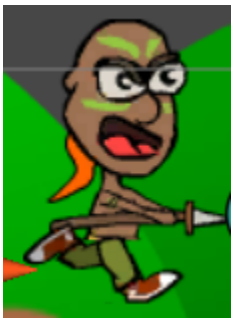
Recommendations are: BoxCollider2D, CapsuleCollider2D, PolygonCollider2D.



Enemy Layer

Change the ‘Order in Layer’ for the player game object in Inspector to the same layer as your **Foreground**. In this case, the **Foreground** layer order would be **2**, as such would be the Enemy..

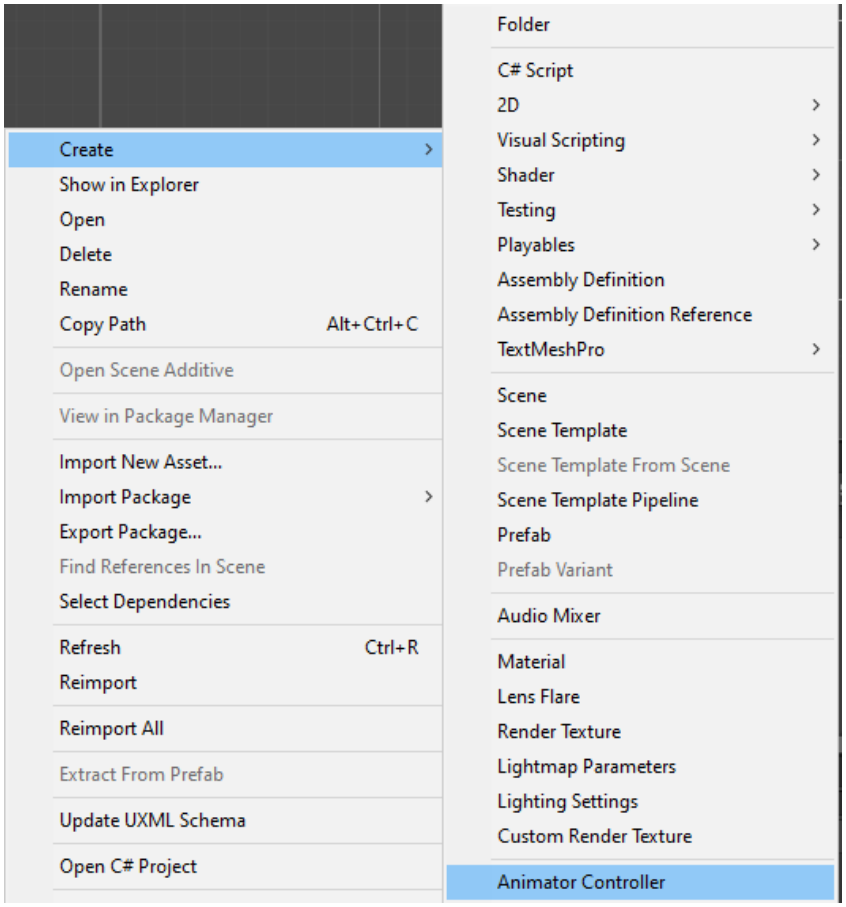




Animations

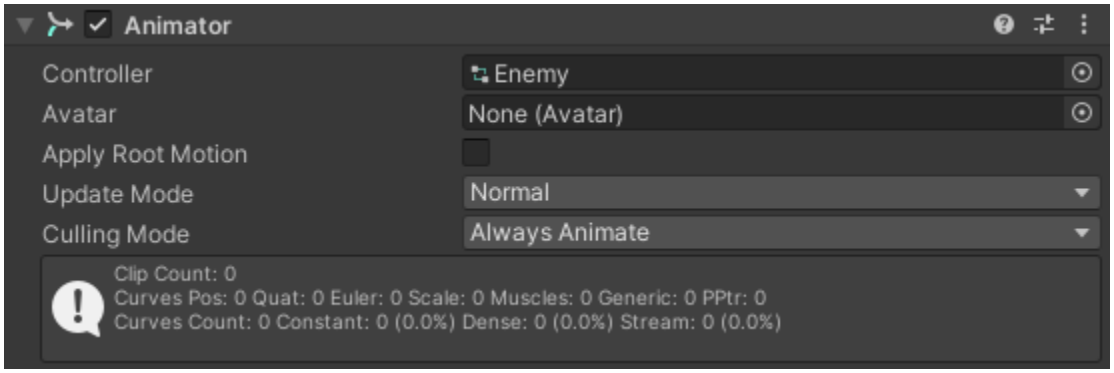
Animators

Right-click the file **Animators** and click **Create > Animator Controller.** Rename to **Enemy**.



Click and drag the **Enemy** animator file into the player game object.

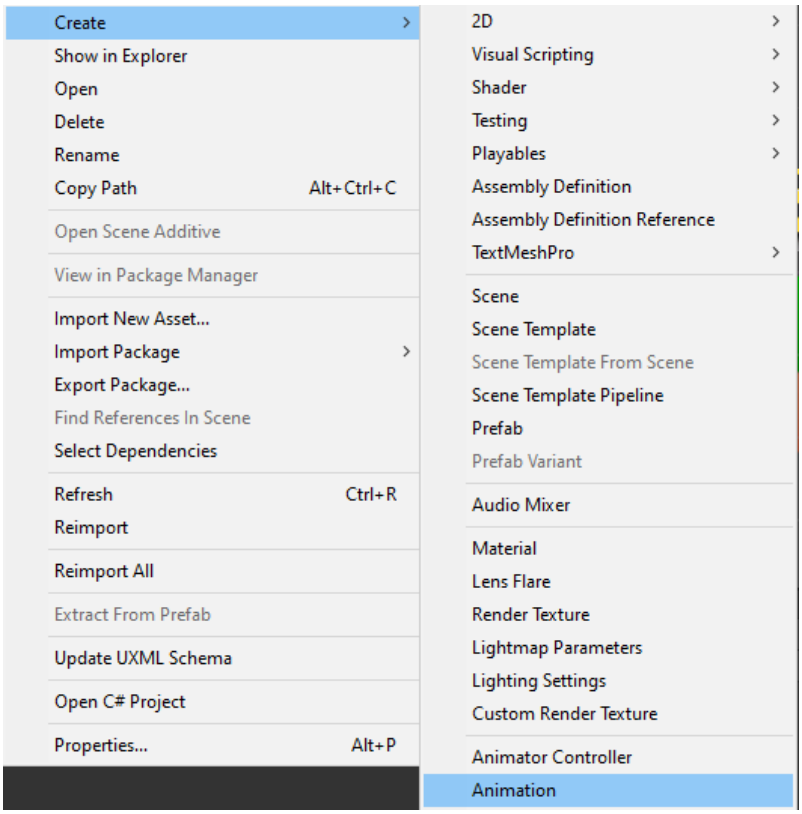
It will show up in the **Inspector** when the enemy game object is clicked in the **Hierarchy** as shown in the image below.



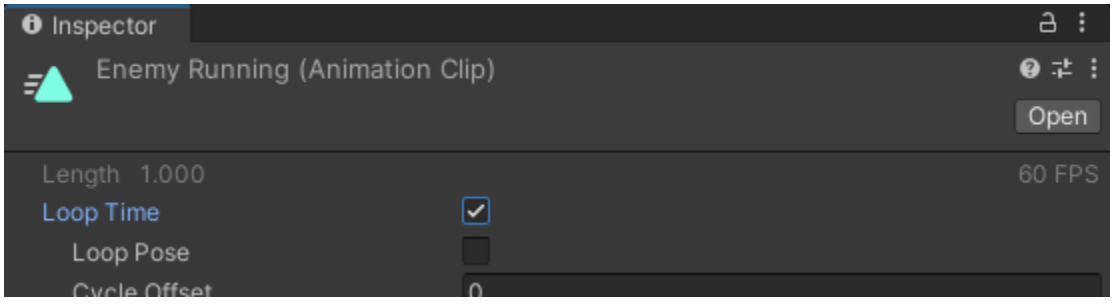
Animation

Make a new **Animations** folder to store your animation files.

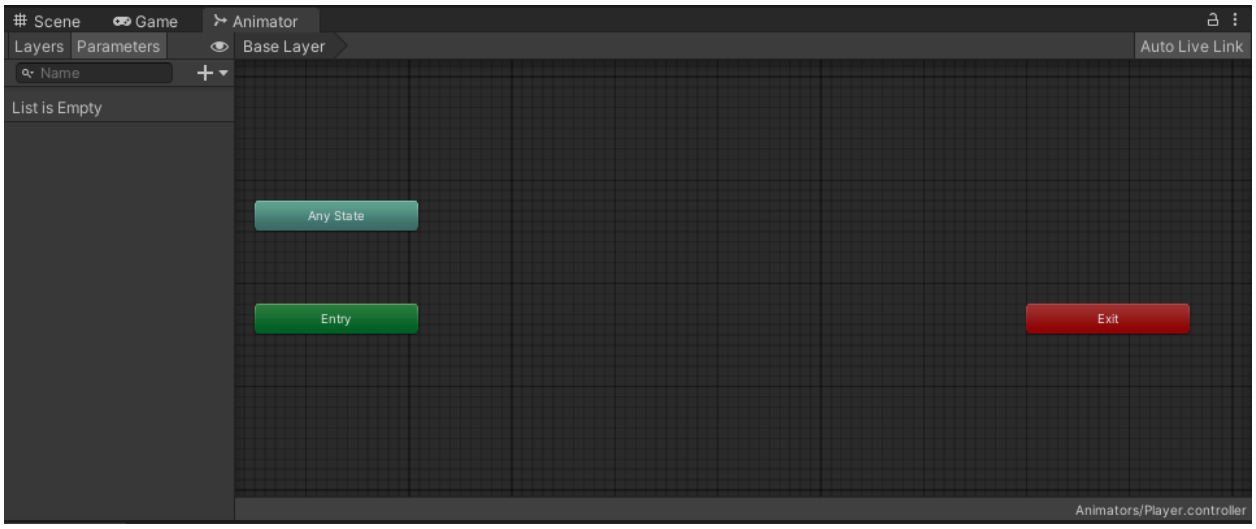
Right-click the **Animations** folder and click **Create > Animation**. Name it **EnemyRunning.**



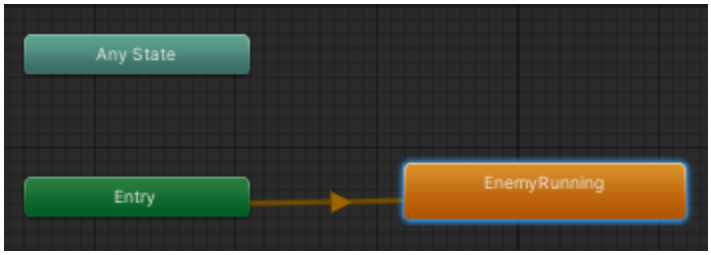
Click on **PlayerRunning** and check **Loop Time**. This will allow this animation clip to loop infinitely instead of playing just once.



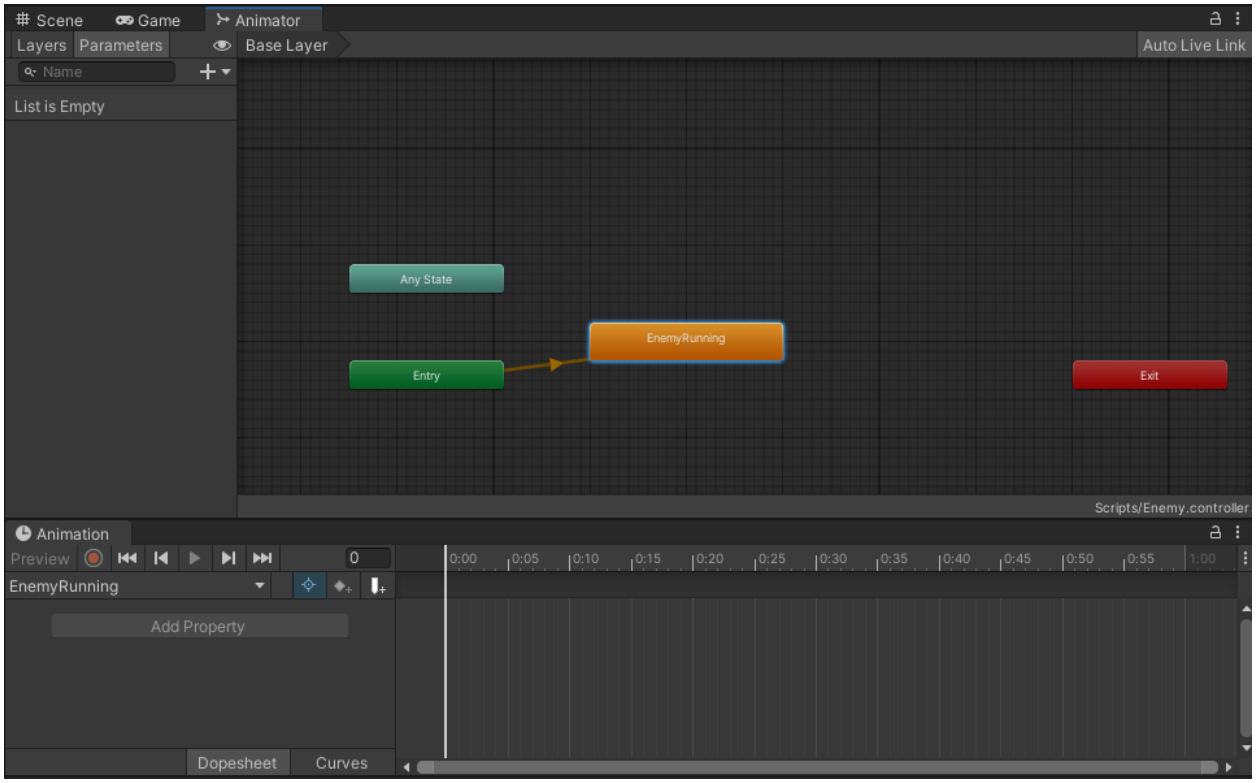
Double click the **Enemy** animator file in **Projects > Animator.** An **Animator** window should pop up.



Click and drag **EnemyRunning** into the **Animator** window.



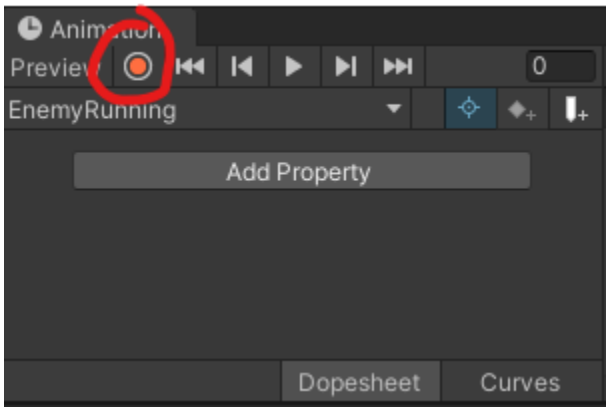
Select **enemy_0** in the **Hierarchy**. Double click **EnemyRunning.**



Running Animation

Repeat for PlayerRunning animation.
The new animation file will be placed in the same **Player** animator controller file.

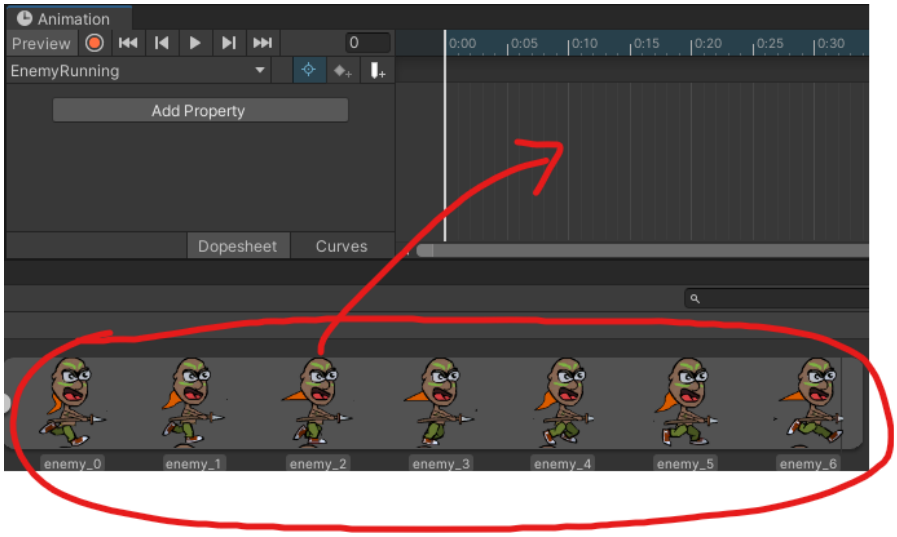
In the **Animation** window, click on the **Red Circle**, which is the record button.

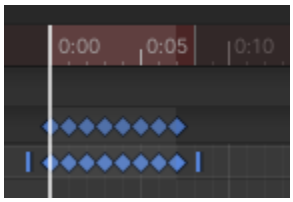


The time frame should turn Red. This means you can start dragging in images for the animation.

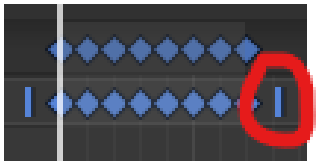
In your sprites folder, expand the sprite sheet right the arrow and select all the sprites you want to be your running animation.

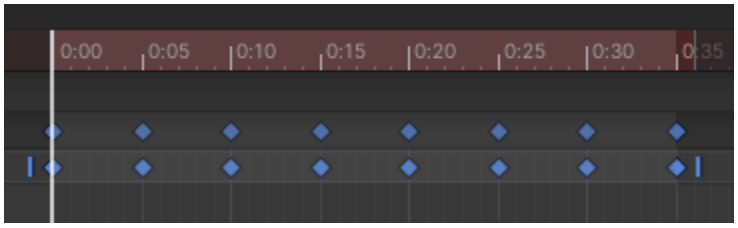
Drag it into the **Animation** window.



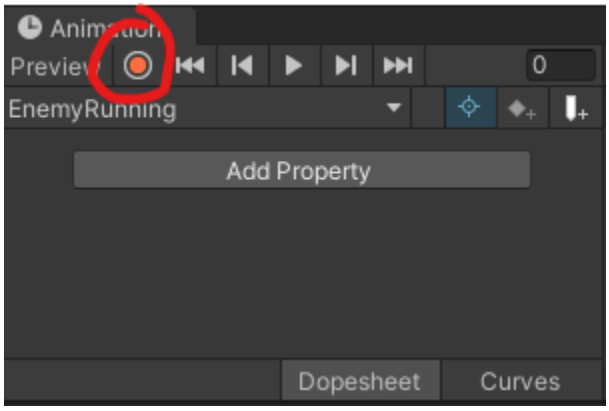


Click and drag the bar on the right to increase the time it takes to run the entire animation.





Once you are happy, press the Red button again to stop the recording.



EnemyMovement

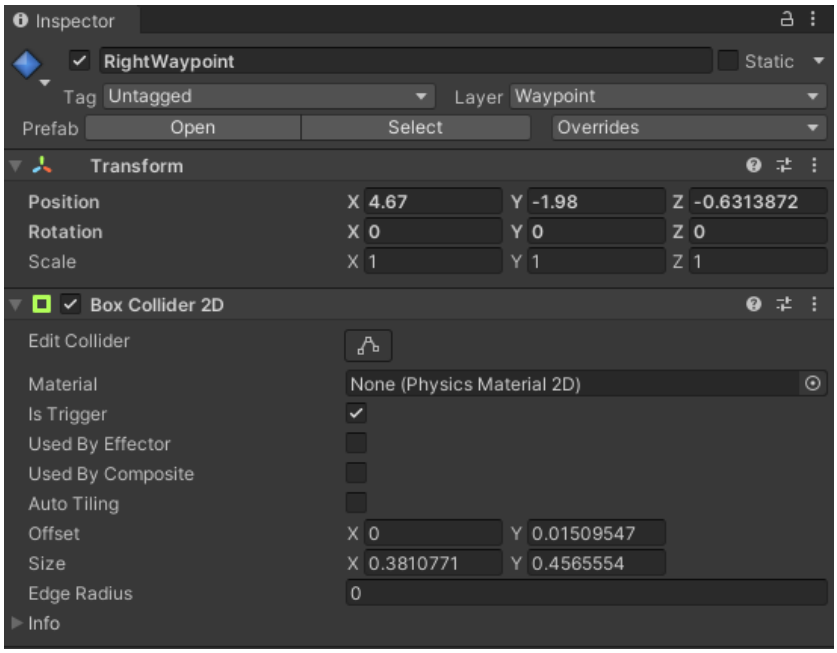
Overview

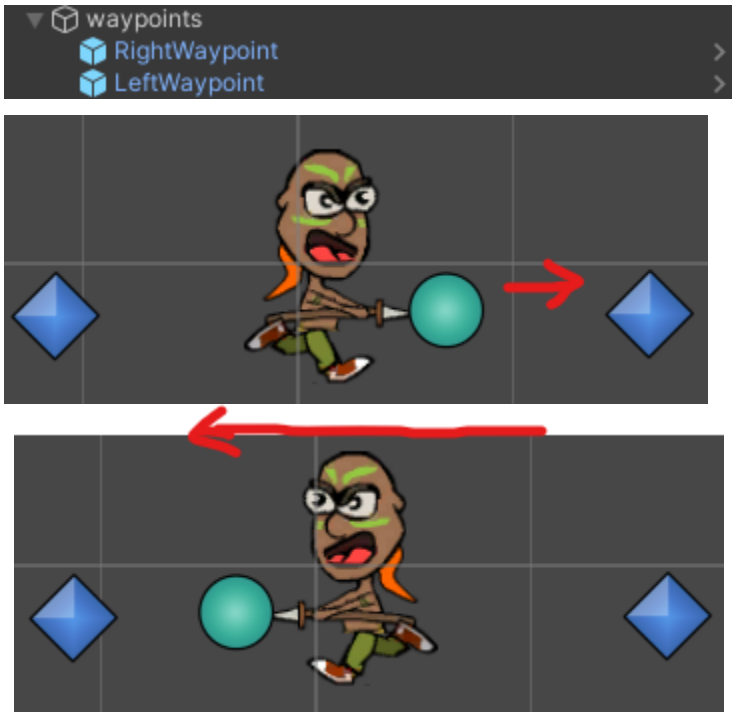
The enemy will move at the allocated speed and it will have a game object point that detects when it hits a wall and/or waypoint object placed by the user.

The enemy will first move to the right then the left on game start.

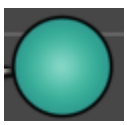
Required Objects:

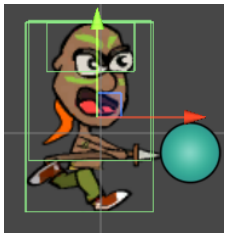
- **Waypoints** empty game objects with:
  - ‘Layer’ - **Waypoint**.
  - Box Collider 2D - **Is Trigger** ticked





- **WaypointDetect** - Detects waypoints or walls placed down to instruct the enemy to turn and move in the opposite direction.
  - **Place the ‘Waypoint’ object in front and outside of any BoxCollider2D.**

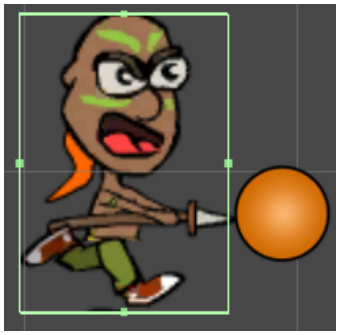




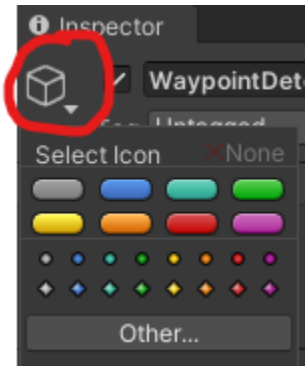
Making WaypointDetection Object

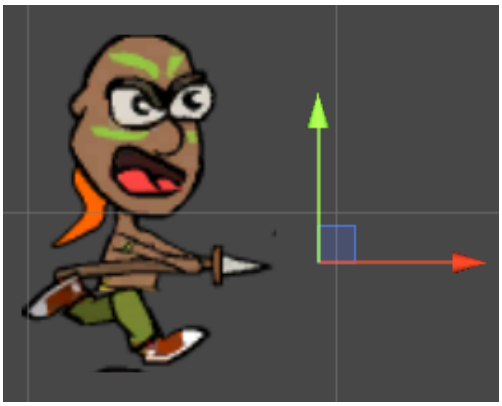
Create a new empty game object as a child of the **enemy** game object.
Name it **WaypointDetect** or anything you would like.

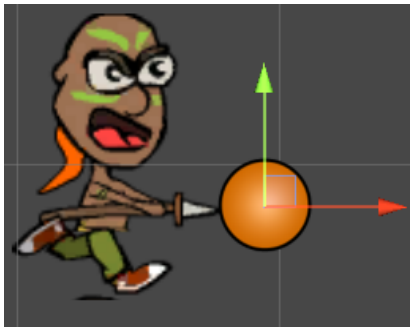
Move **WaypointDetect** so that it is just in front of the enemy sprite but also outside of its collider.



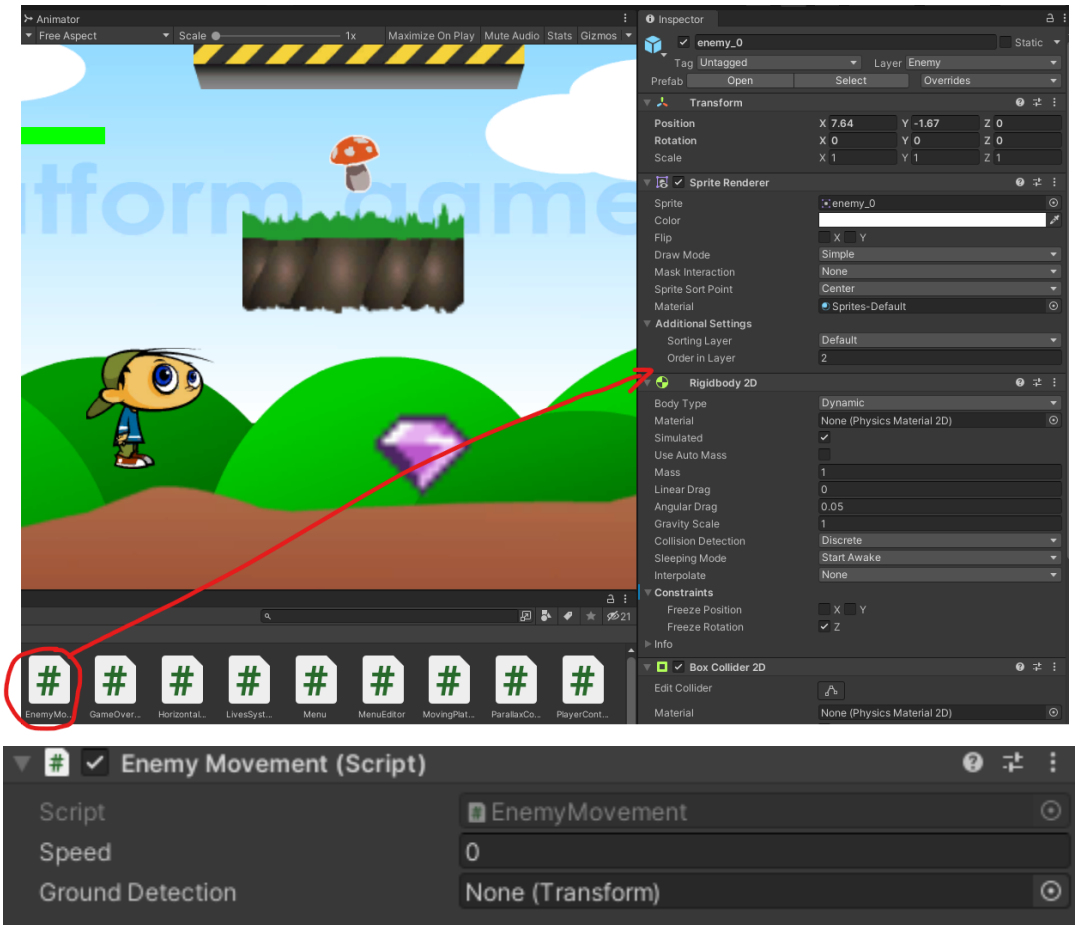
Click on **WaypointDetect** and in the **Inspector** change its image icon to any you would like**.**



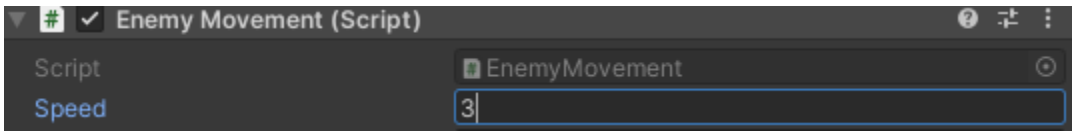




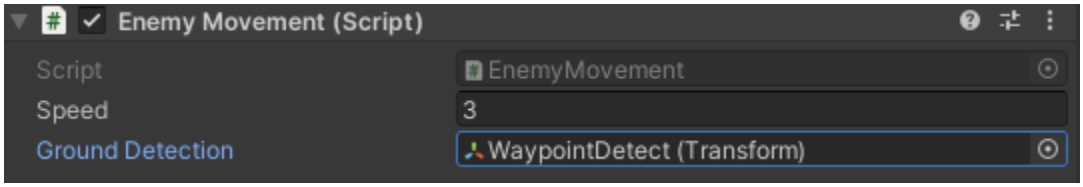
Click and drag the **EnemyMovement** script from your **Project** window into the enemy game object.



Choose a movement speed number.



Click and drag the **WaypointDetect** game object from the **Hierarchy** into the **Ground Detection** variable section.



What does Ground Detection do?

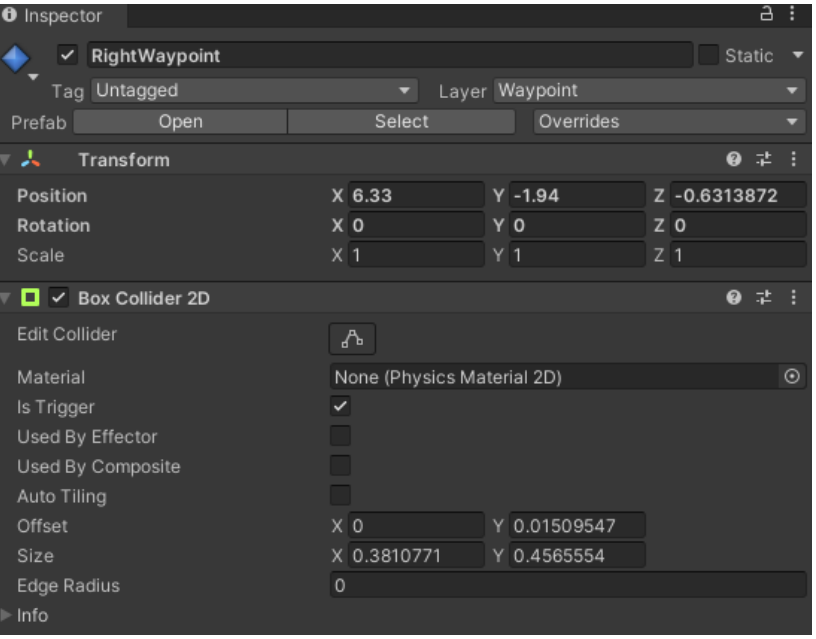
Ground detection takes **WaypointDetection**'s current position and checks if it is inside a wall, waypoint, or if there is no floor indicating the edge. If those happen, then the enemy should turn around and walk the other way.

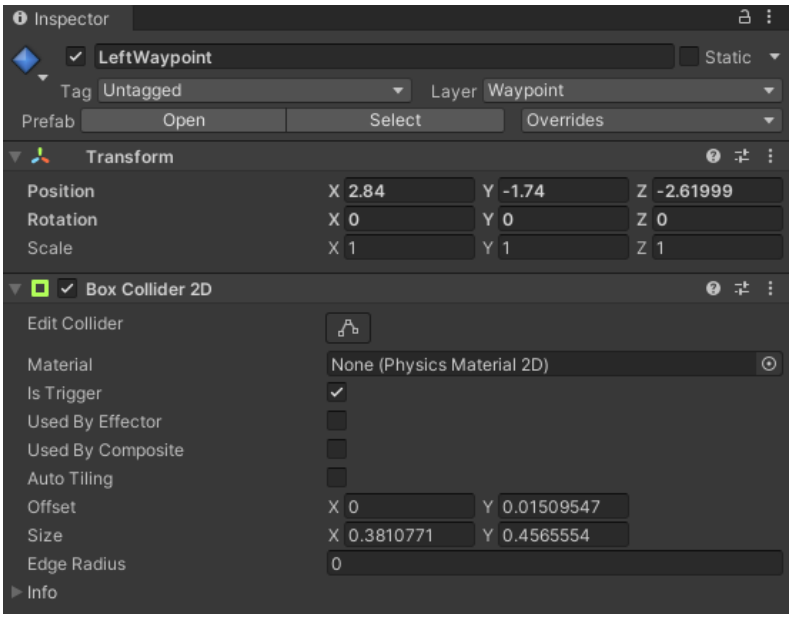
Making Waypoints

Create an empty game object for the waypoints.

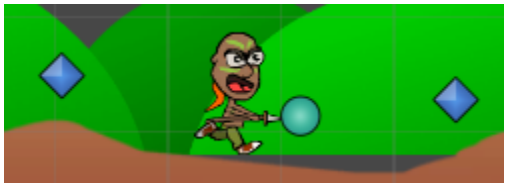
Create empty game objects as children of the parent game object. In this case for **waypoints**.

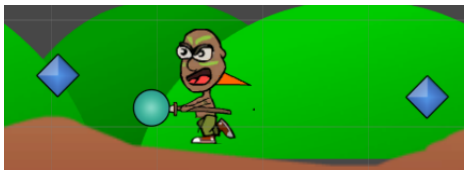
For each of the waypoints, **Add Component > BoxCollider2D**.
Tick '**Is Trigger'**.
Change the image icon so that they are easier to identify in the **Scene** window.





Place them in front and behind the enemy.





EnemyHealthAndDamage

Overview

If the player walks into the ‘Damage’ game object with the Collider2D, the player will take damage based on the damage specified by the user in the script variable.

Required Objects:

- New game object child - with a **BoxCollider2D**

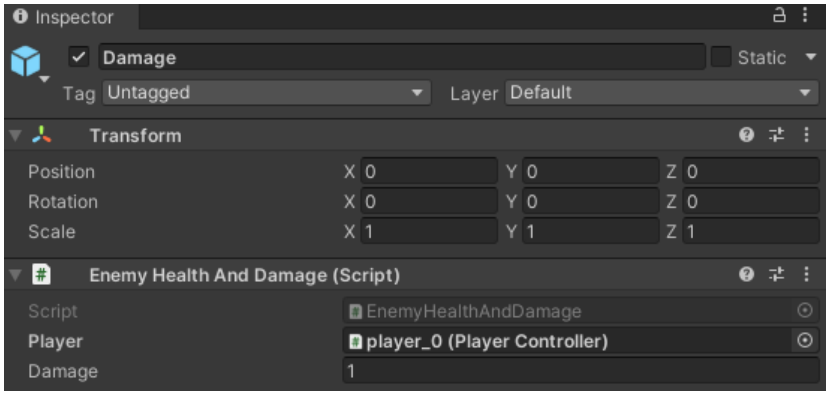
- Player Object - with **PlayerController** script attached

Click and drag **EnemyHealthAndDamage** into the new game object child.

Variables:

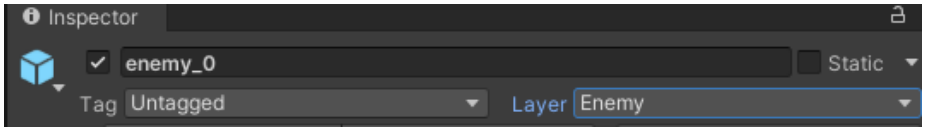
- Player - **PlayerController** script
  - **Click and drag the player game object into the ‘Player’ section of ‘Enemy Health And Damage’**

- Damage (float) - how much damage the enemy deals with the player's health each frame.

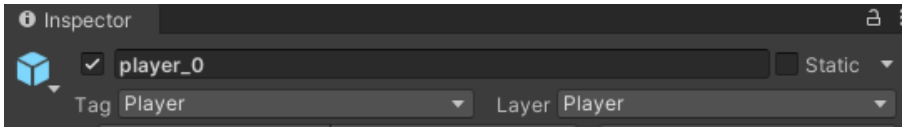


Making the ‘Enemy’ Layer not collide with ‘Player’ Layer

Create a new ‘Layer’ called **Enemy** for the main enemy object.



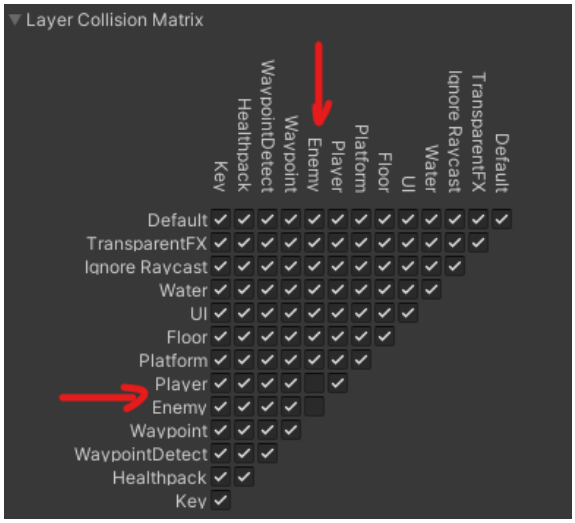
Create a new ‘Layer’ called **Player** for the main player object.
Set the Tag for the main player object as **Player**.



Go to ‘**Edit > Project Settings > Physics2D**’.
Scroll down and expand ‘**Layer Collision Matrix’**.

Ensure Layers for **‘Enemy-to-Enemy’** and ‘**Enemy-toPlayer’** are unticked.

If not unticked.
Untick the Layers for ‘Enemy-to-Enemy’ and ‘Enemy-toPlayer’



What does this do?

This makes it so any game object's collider with Enemy and Player cannot collide with each other. In short, they cannot interact with each other.

This allows the player to walk into the enemy and take damage. If unticked, the player will not take damage as the damage overlaps the main collider and will instead push the enemy away.

ENEMY-PLAYER TICKED

UNTICKED

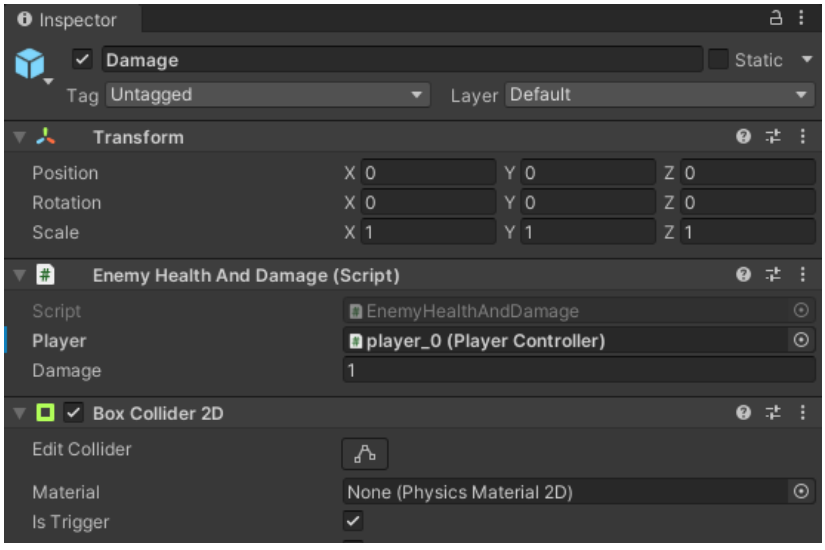
Enemy Damage Collider Game Object

Create a new game object as a child for the **enemy** game object.

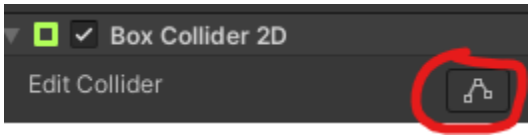
Name it. In the example, we called it **Damage** to make it easy to understand.

In the **Inspector** for **Damage**, **Add Component > BoxCollider2D**.

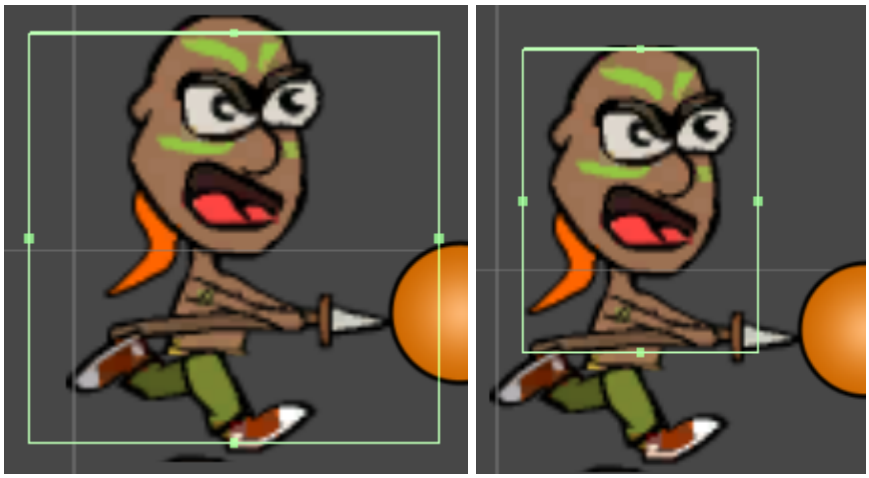
Tick '**Is Trigger**'



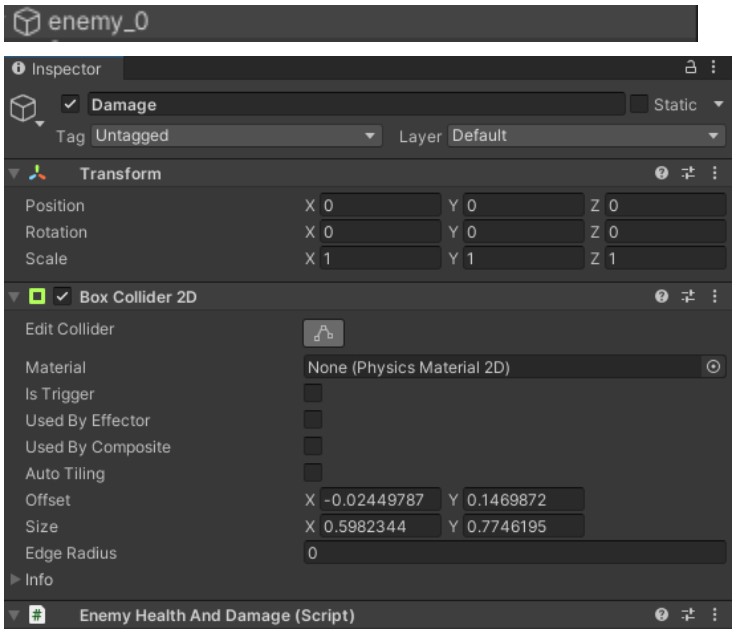
In the **Inspector** click the button next to **Edit Collider** to manually change the collider's size.



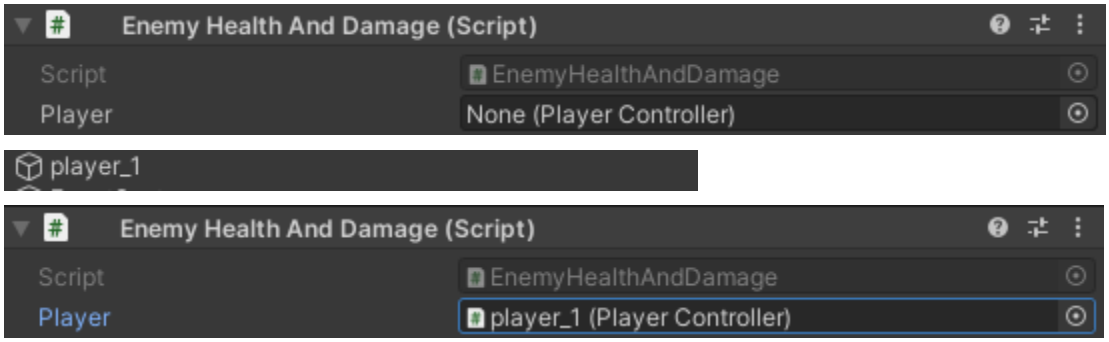
Readjust the size to fit the head to the waist.



Drag the **EnemyHealthAndDamage** script from your **Project** window into the **Damage** game object in the **Hierarchy.**



Drag the **player** game object from the **Hierarchy** into the **Player** variable section



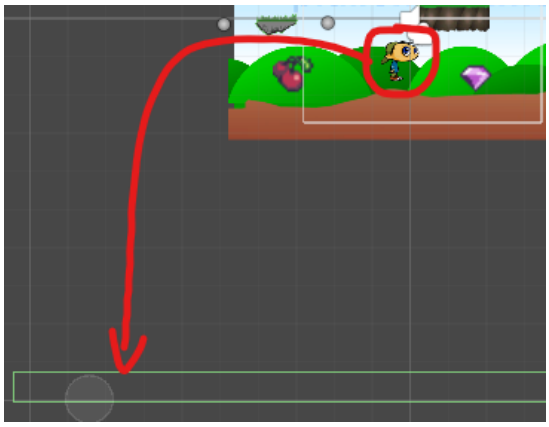
The 'EnemyHealthAndDamge' script will reference functions from the 'PlayerController' script with contains the 'takeDamage()' function which allows the player to take damage.

Change the **Damage** variable number to any you would like.

Death From Falling Outside the Map

Overview

When the player jumps/falls off the environment, they will instantly die and lose a life.



Scripts:

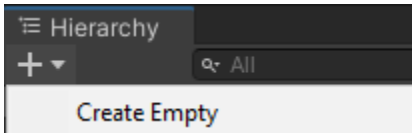
- EnemyHealthAndDamage

Gameobject:

- BoxCollider2D

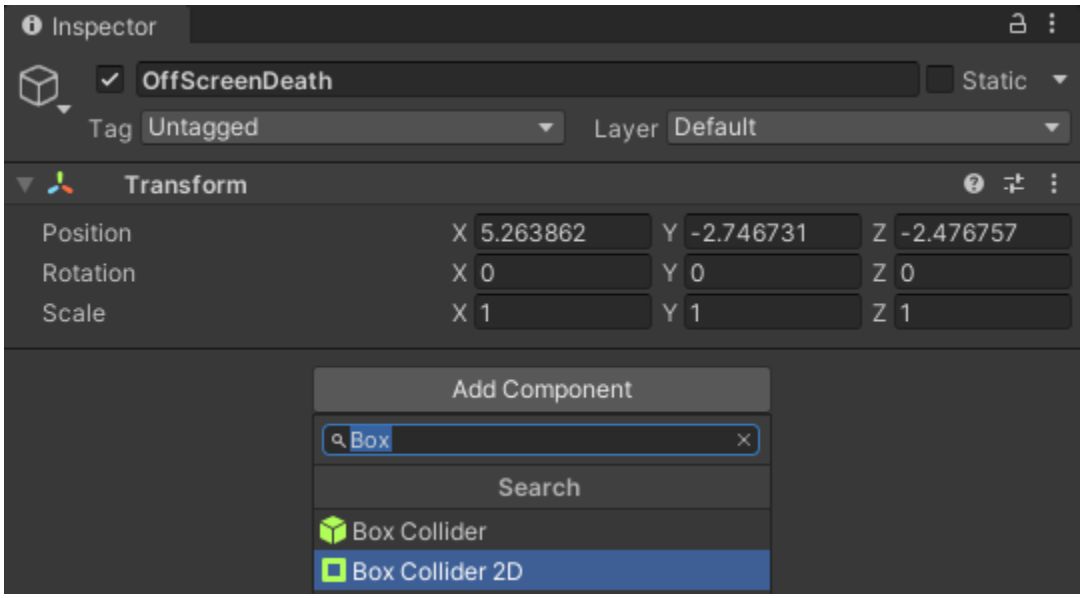
Creating the Game Object

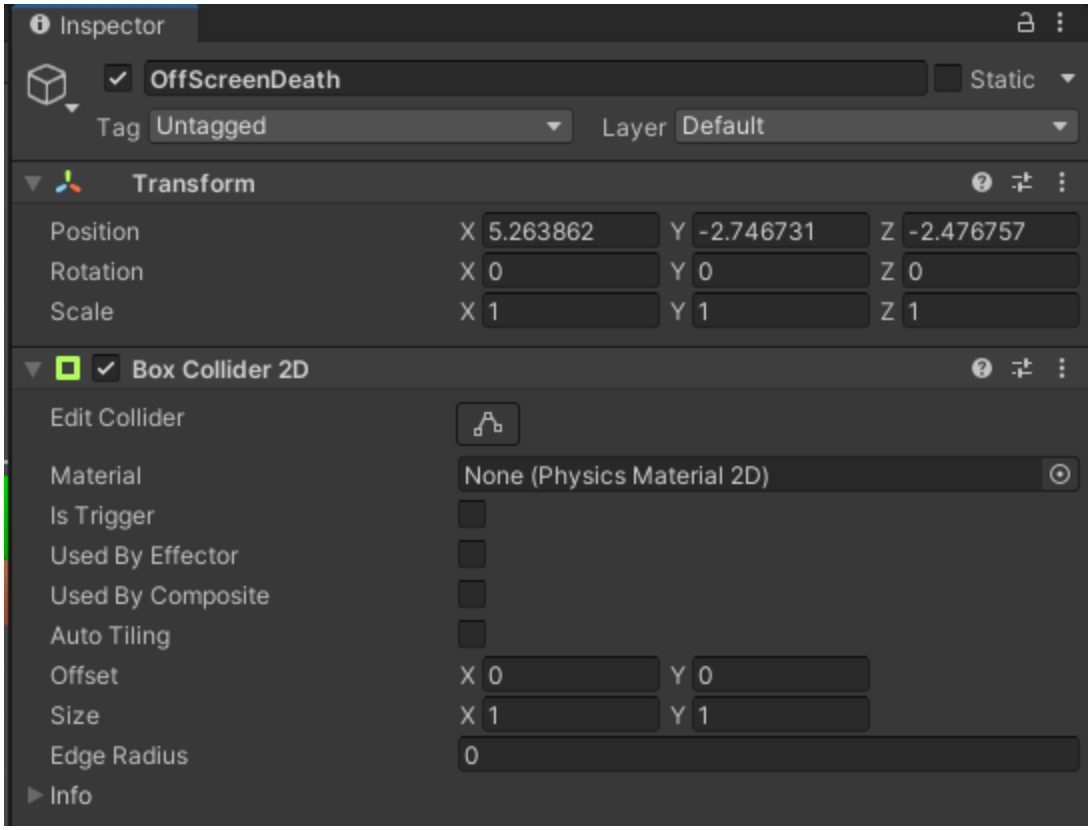
In the **Hierarchy**, click the **+** icon and **Create Empty**.



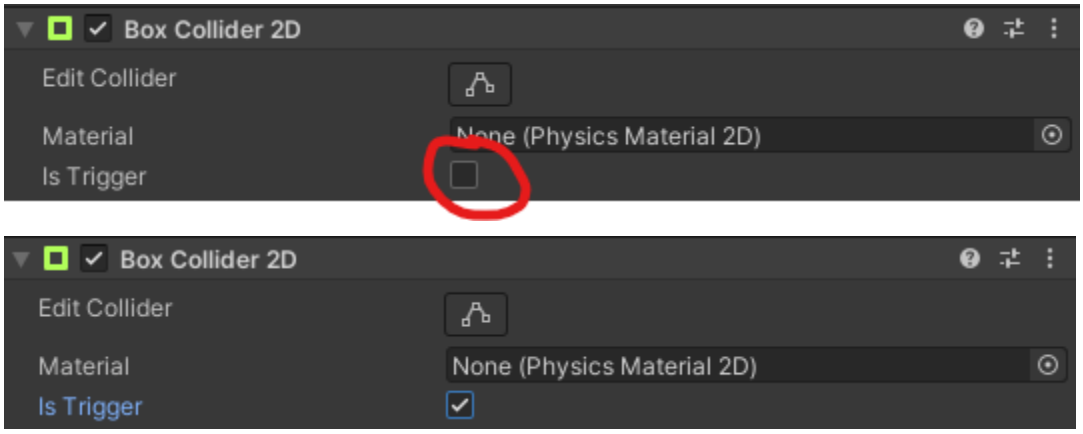
Name it anything you want. In the sample game it is called **OffScreenDeath**.

In the **Inspector**, click **Add Component**, search **Box Collider 2D,** and click it.

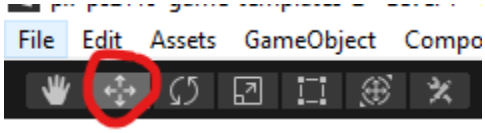




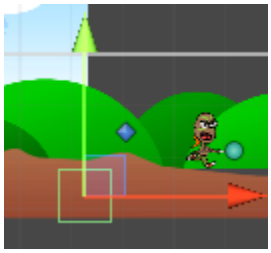
Click check the **Is Trigger** box.

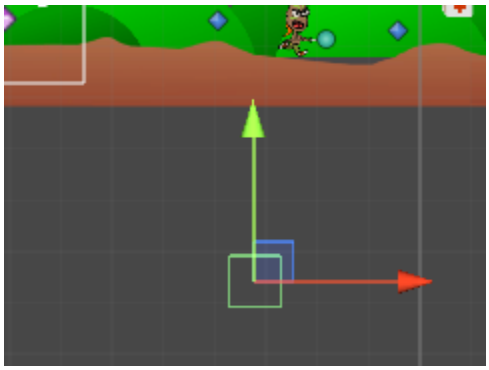


Click the **Move Tool**.

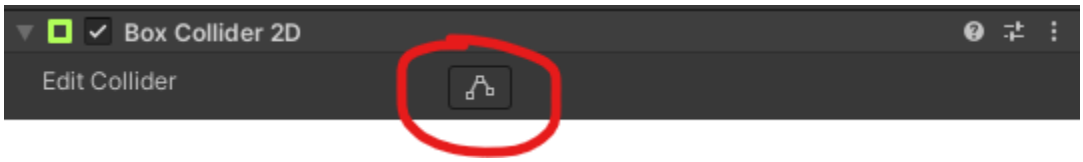


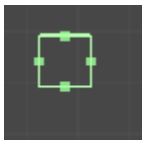
Use the arrows to position your game object below the map.



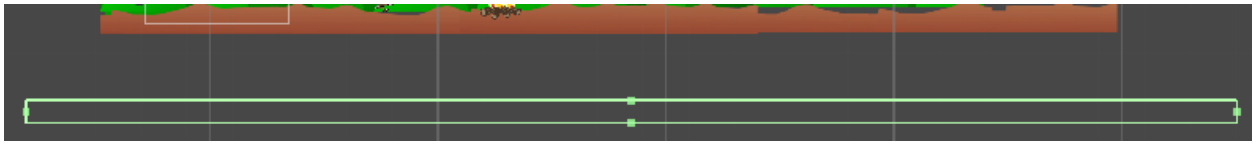


Click **Edit Collider**



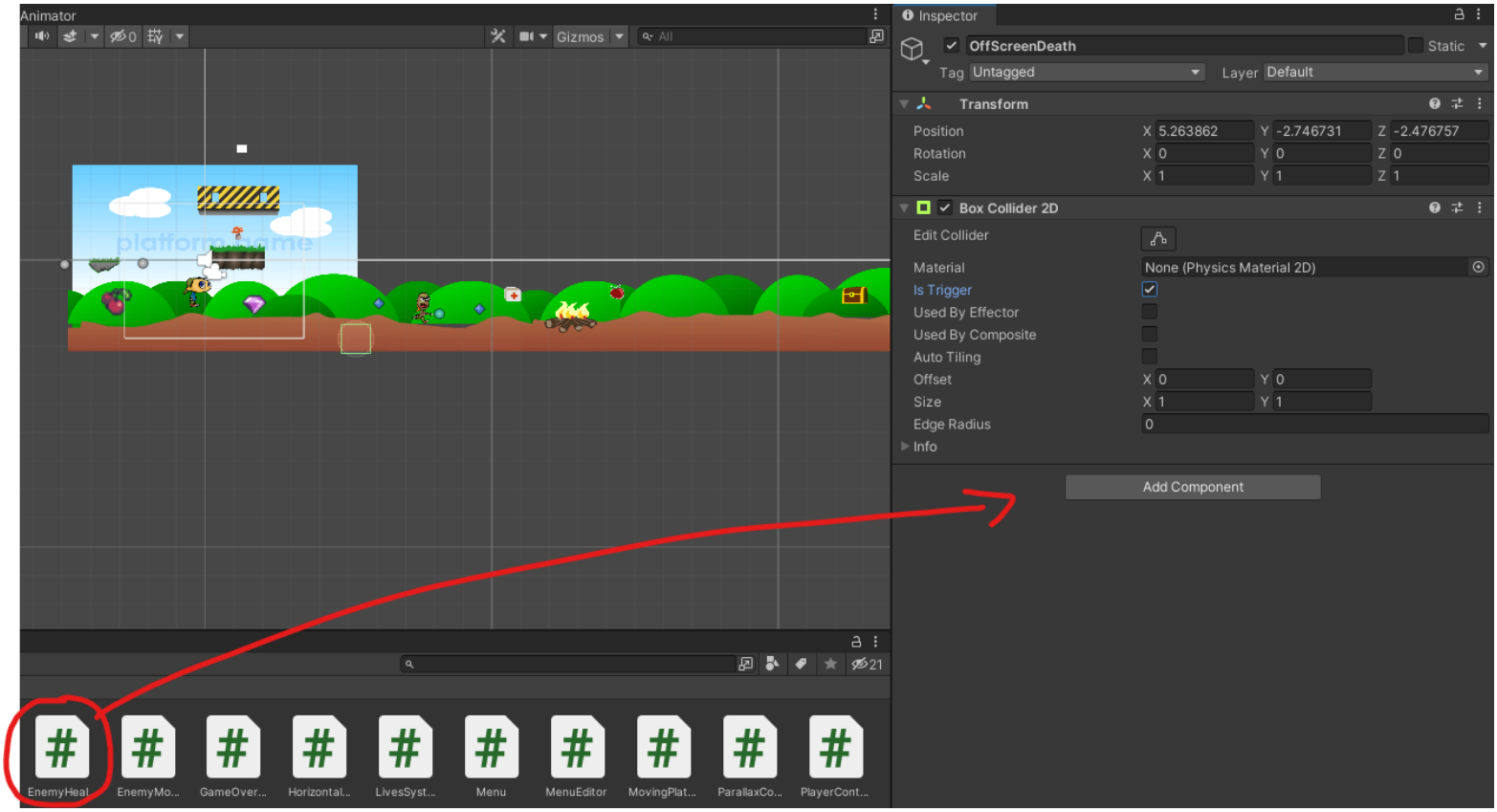


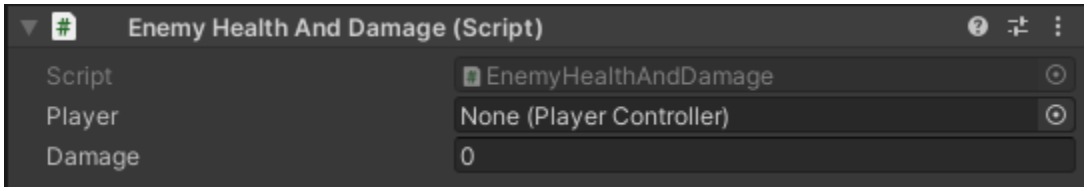
Use the squares on each edge to resize the collider. Resize to be bigger than the floor object.



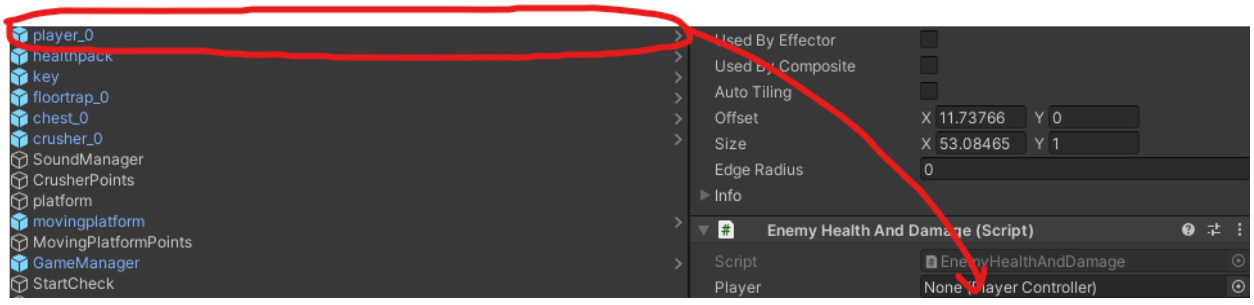
Adding EnemyHealthAndDamage Script

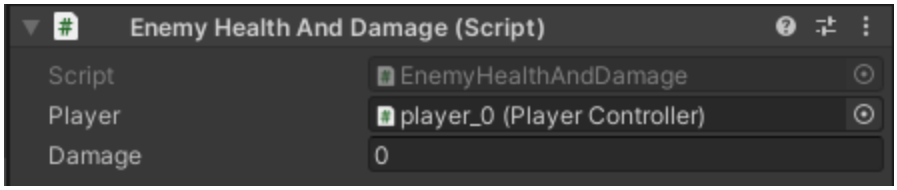
Find the **EnemyHealthAndDamage** script. Drag it into the **Inspector** of **OffScreenDeath**.



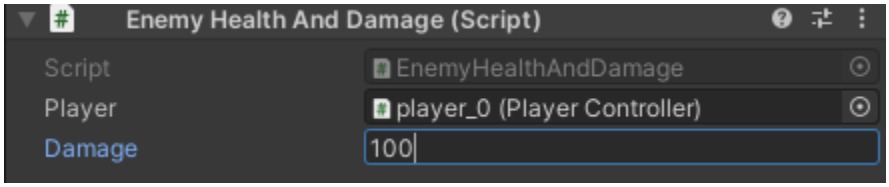


Drag the player object with the script **PlayerController** into the **Player** variable





Change the value **Damage** to the player's max health to make the player take full damage when they fall and touch the collider.



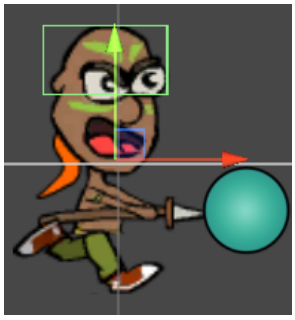
PlayerKillEnemy

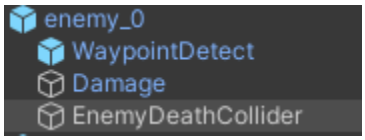
Overview

This script allows the enemy to be killed by the player. If any colliders from any game object walk into the game object with the script, it will destroy the game object specified by the user.

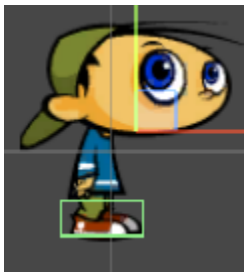
Required Objects:

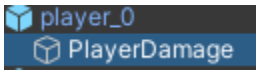
- A new game object that has a **BoxCollider2D** for the enemy.

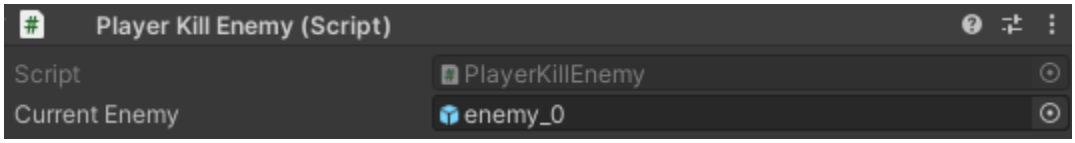




- A new game object that has a **BoxCollider2D** for the player.







Variables

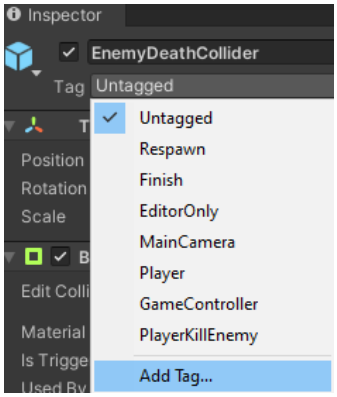
- **Current Enemy** - the enemy game object that the script will destroy

Creating and Implement Object Collider and ‘PlayerKillEnemy’ Script

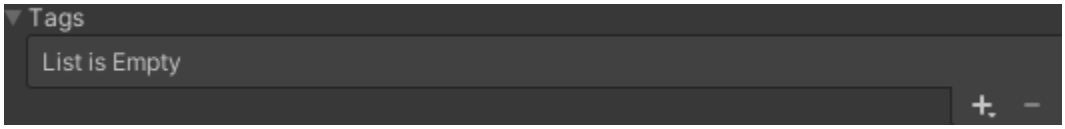
Create a new game object child for the **player** game object.
Name it to what you desire. In the image, it is named **PlayerDamage**.

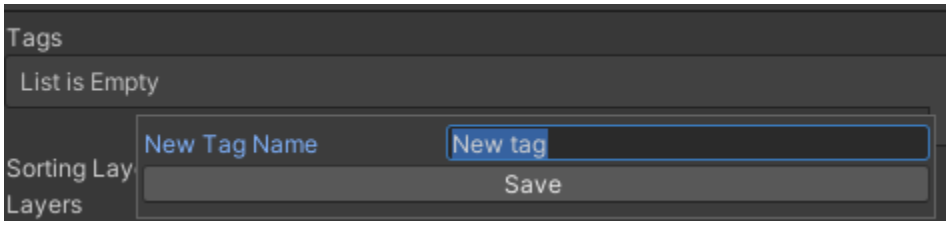
IF PlayerKillEnemy Tag does not exist.

Create a new **Tag.** Click the Tag dropdown and click **‘Add Tag…**’

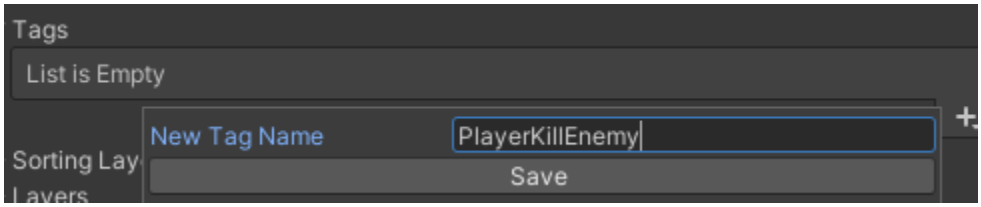


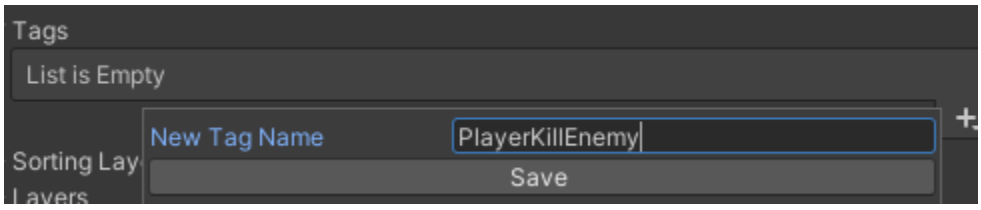
In **Tags,** click **+**.



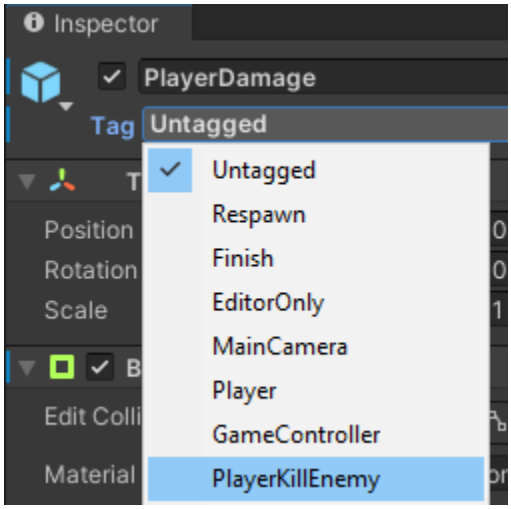


Name the tag **PlayerKillEnemy**. Click **Save**

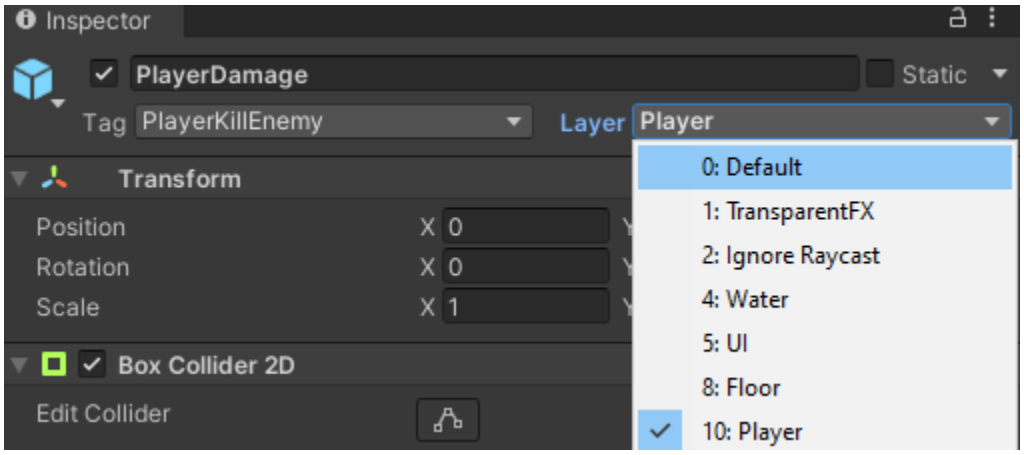




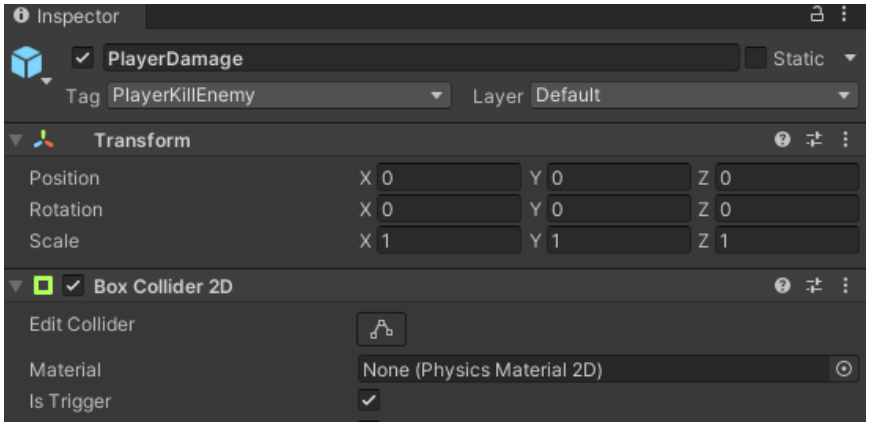
Go back to the **PlayerDamage Inspector.**
Click the **Tag** dropdown and choose **PlayerKillEnemy**.



Change the layer **'Player'** to **'Default**'.
When creating a child object, it automatically sets its layer to be the same as its parent object.

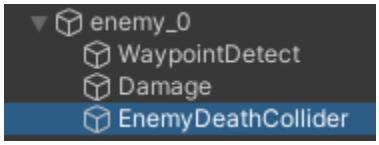


In the **Inspector** for **PlayerDamage**, **Add Component > BoxCollider2D**.
Tick **'Is Trigger**'

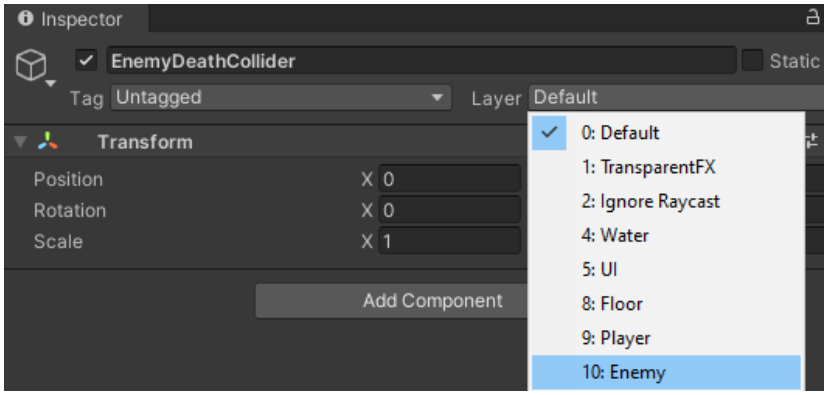


Click **Edit Collider** and adjust the size to the player's feet so that when the enemy's head hits the player's feet, it dies.

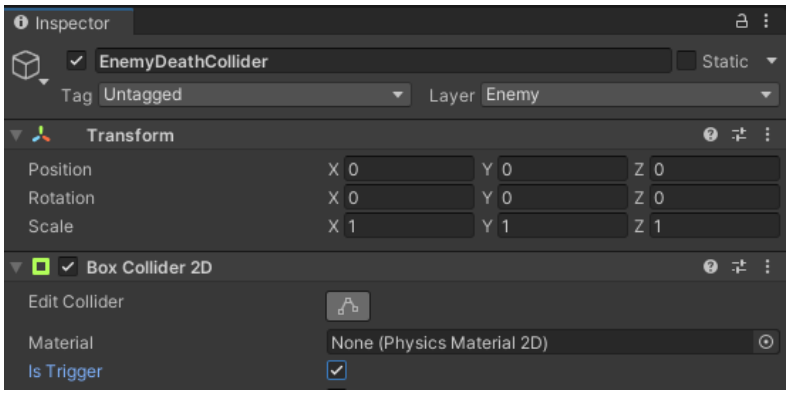
Create a new game object child for the **enemy** game object.
Name it to what you desire. In the image, it is named **EnemyDeathCollider**.



Change the **Layer** to '**Enemy'**.

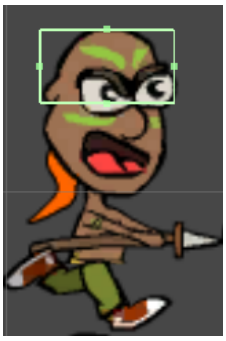


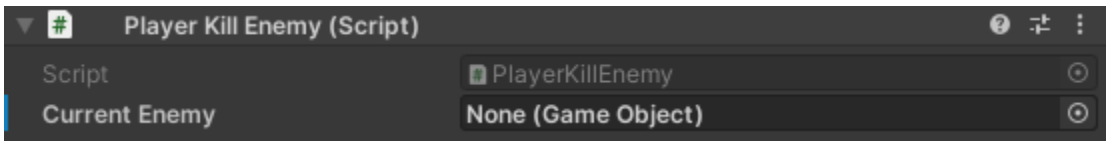
In the **Inspector** for **PlayerDamage**, **Add Component > BoxCollider2D**.
Tick **'Is Trigger**'



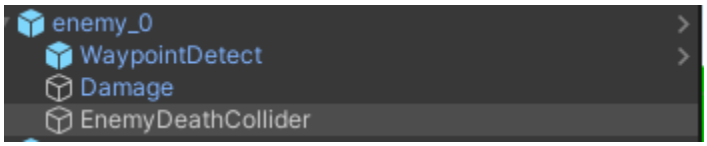
Click **Edit Collider** and adjust the size to the player's feet so that when the player jumps on the enemy's head, it dies.

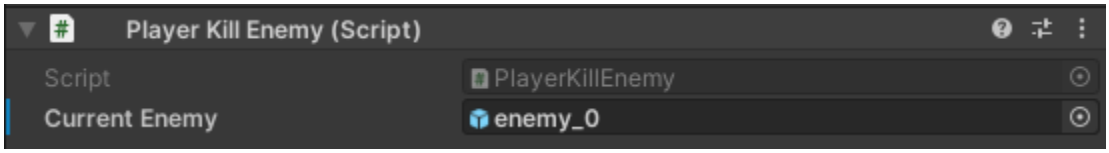
Drag the **PlayerKillEnemy** script into **EnemyDeathCollider**.



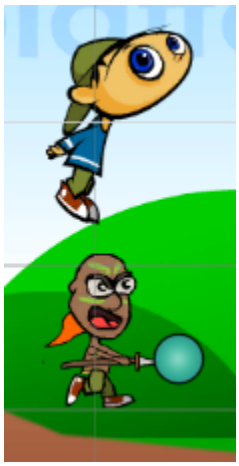


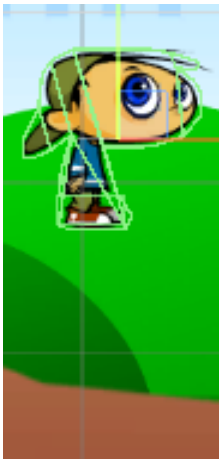
Drag the **enemy** game object you want to destroy when the script activates into the **Current Enemy** variable section. In this case the enemy game object parent.





Demonstration





Collectables

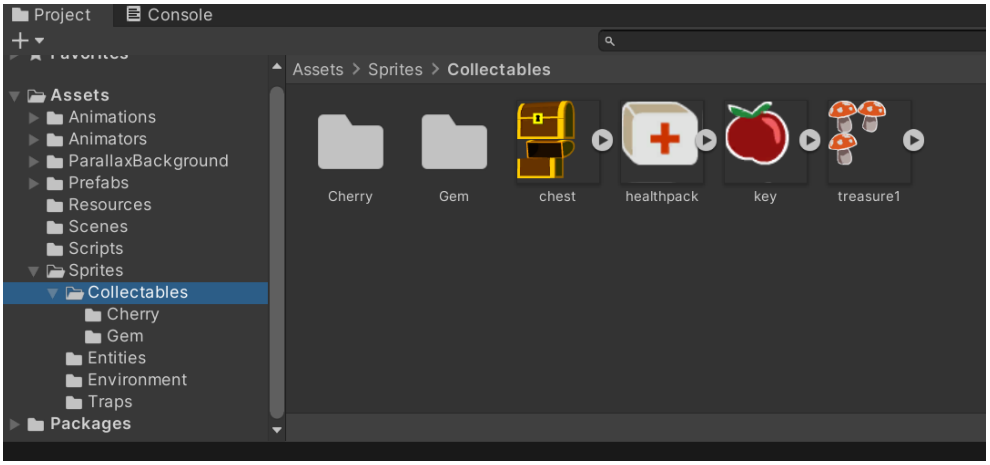
HealthPack Treasure Cherry Gem Key

Overview

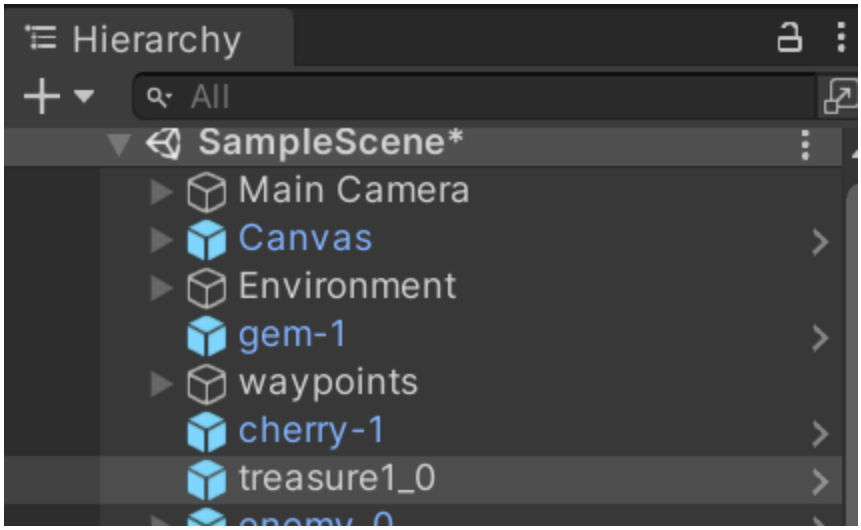
In most games, you will find a number of things that you can acquire as a player which are mainly collectables. This is a feature that is added for a number of reasons. They can be used to stimulate exploration or to break up the monotony of running between areas. Either way, they are fun and engaging additions to your game.

Creating collectables

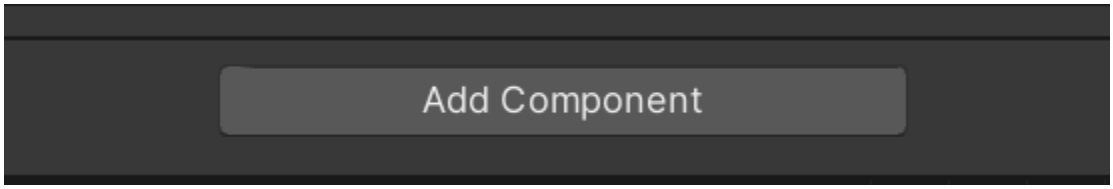
1. Download an image that suits a collectable.
2. Click on the ‘Collectables’ folder that is located under the ‘Sprites’ folder in the Project Window. Drag and drop your downloaded collectable to this folder.



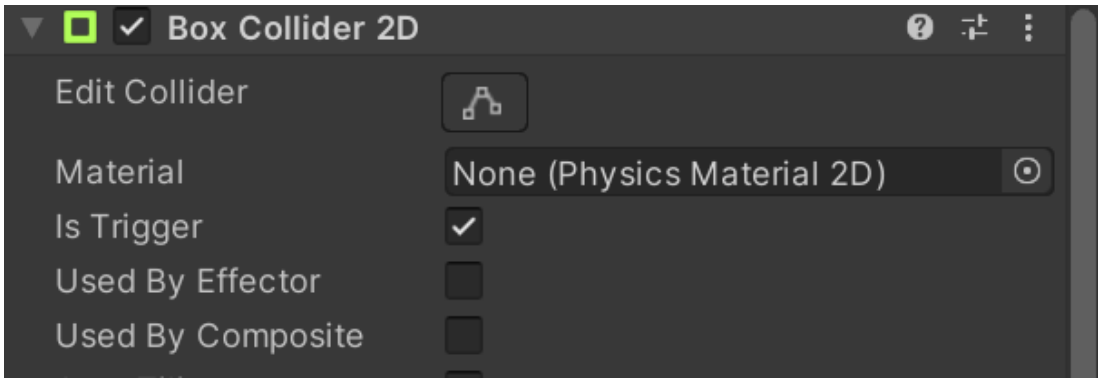
3. Create a new game object and name it to whatever you like. To demonstrate, I will use the ‘treasure1_0’ as an example.



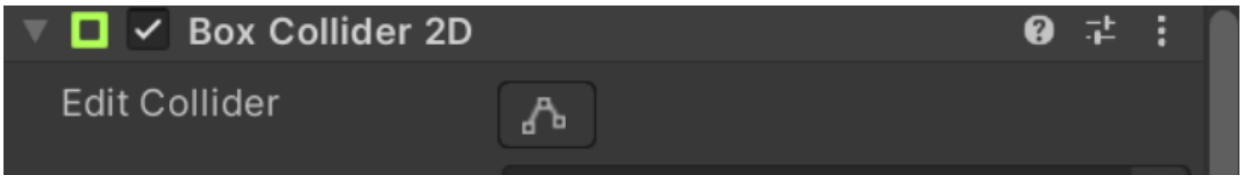
4. In the ‘Inspector’ for ‘treasure1_0’ (on the right-hand side), click ‘Add Component’. Then type and select ‘BoxCollider2D’.



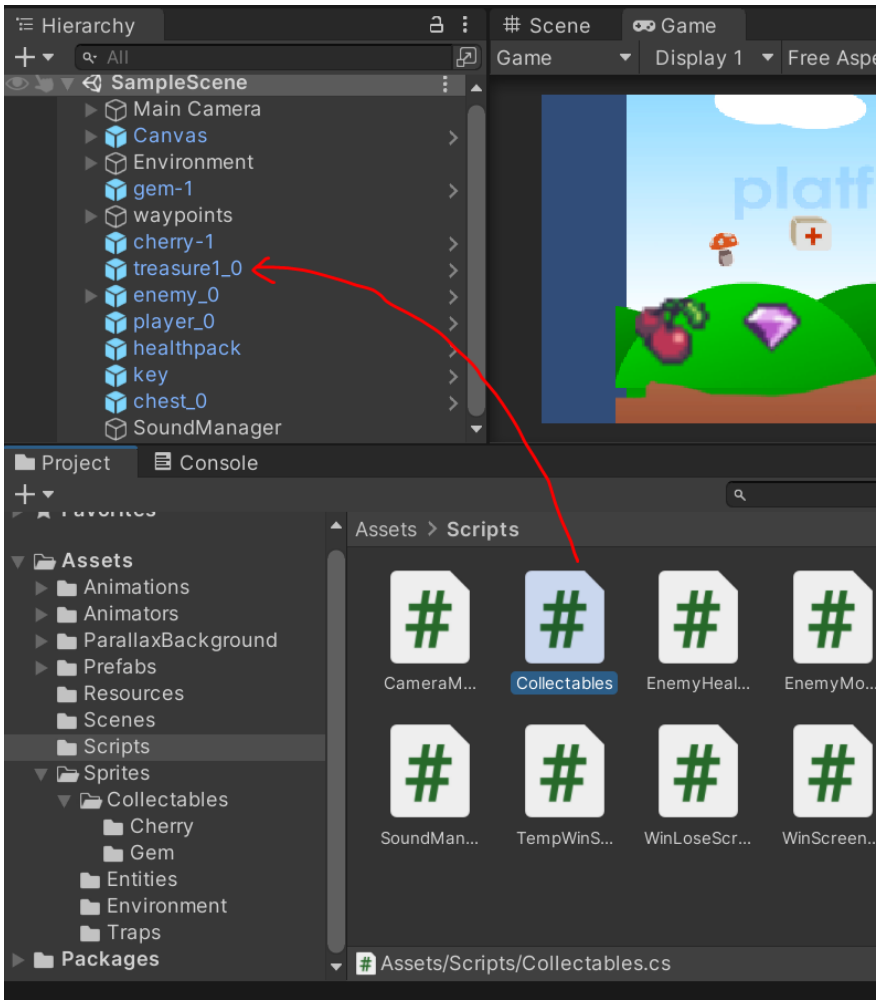
5. On the Box Collider 2D, Tick ‘Is Trigger



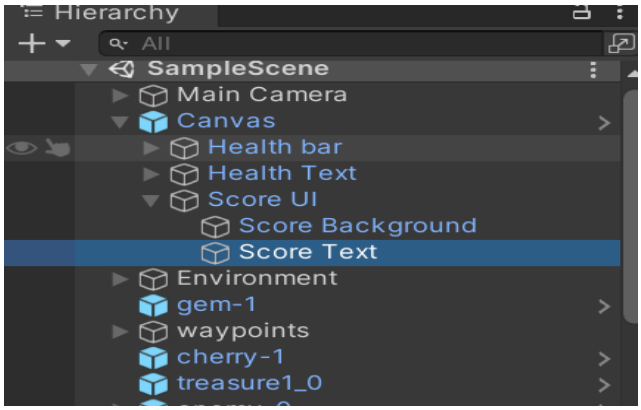
6. In the **Inspector** click the button next to **Edit Collider** to manually change the collider’s size.

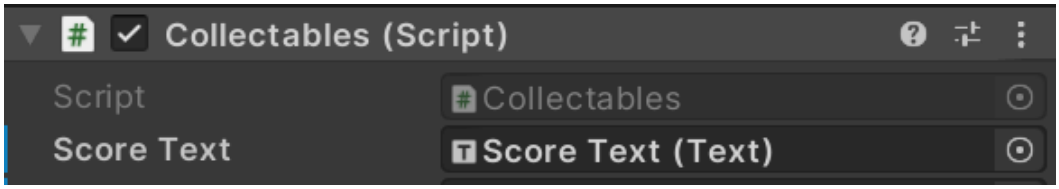


7. Go to ‘Scripts folder’ on Project. Then drag and drop the ‘Collectables’ Script to your game object in the Hierarchy. So for example: as shown below, I will drag the collectable script to the ‘treasure1_0’ game object.

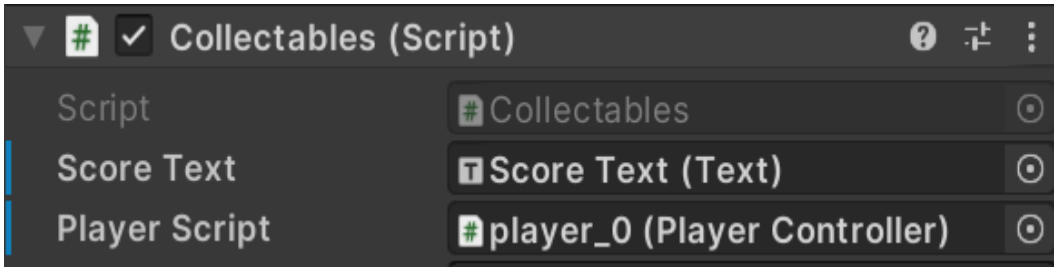


8. Drag the **Score text** game object ‘underScore UI’ from the Hierarchy and place it into the **Player** variable section.

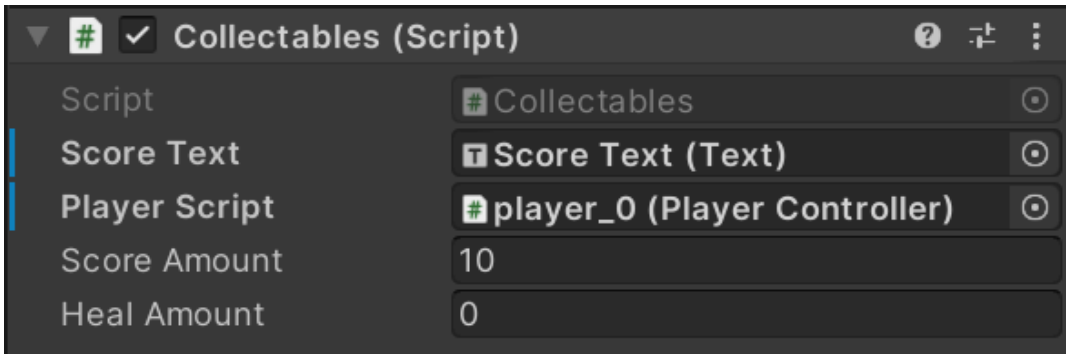




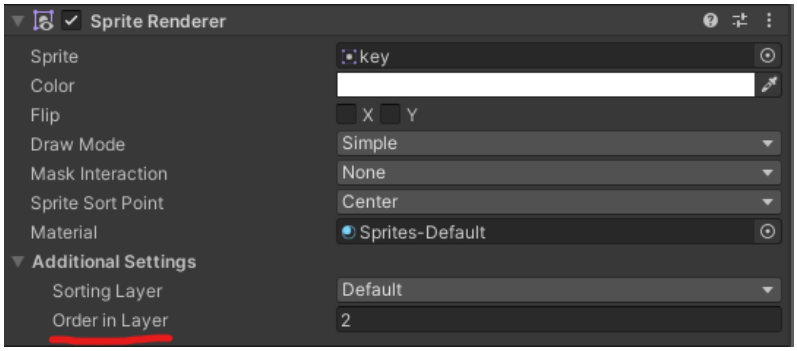
9. Drag the **player** game object from the **Hierarchy** into the **Player** variable section



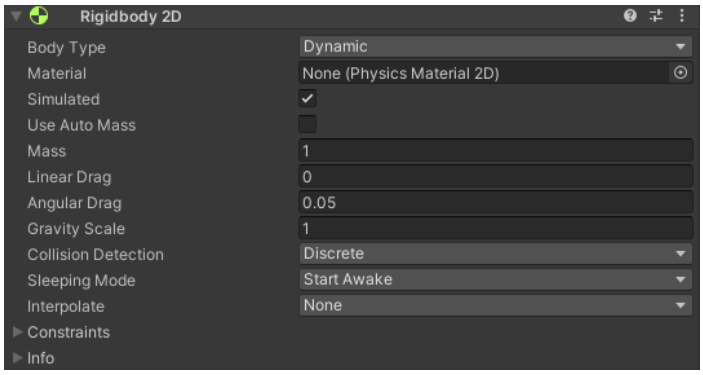
10. Change the **Score Amount** variable number to any you would like.
    a. If your collectable has the ability to heal the player, feel free to change **Health Amount**



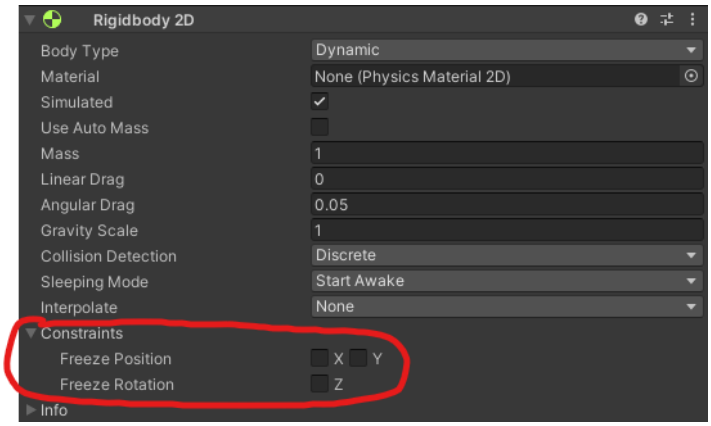
11. Set **Order in Layer** to **2** as it is in the foreground.



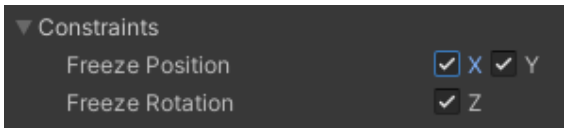
12. Add a **Rigidbody2D** component. **Add Component > Rigidbody2D**.



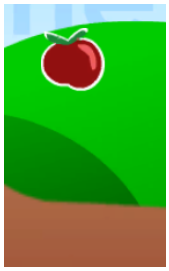
13. Expand **Constraints** in the **Rigidbody2D** component.



14. Tick **Freeze Position** for **X** and **Y**, and **Freeze Rotation** for **Z**.



This does not allow the object to move in the X-axis (horizontal / left and right) and Y-axis (vertical / up and down), as well as rotate.

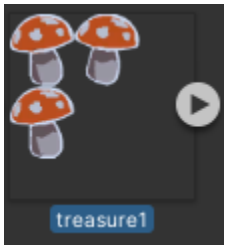


Frozen

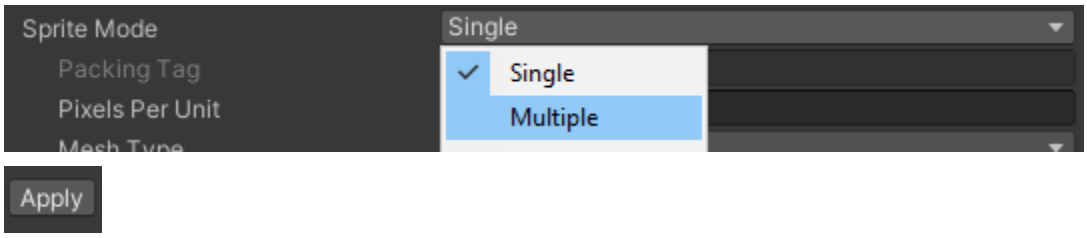
If your Collectable has animations.

Sprites

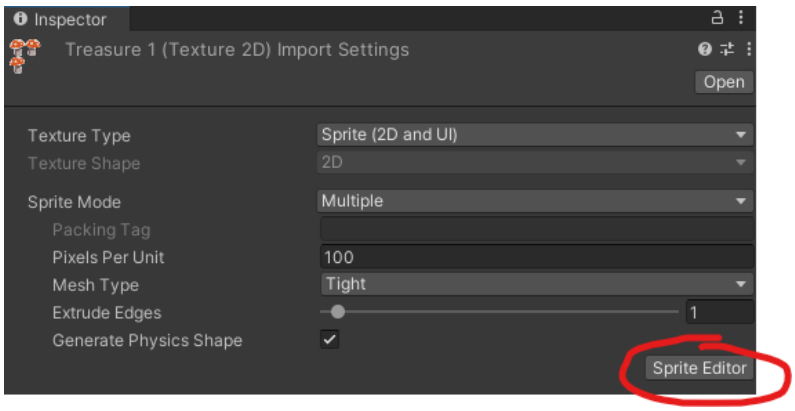
1. Upload your sprites to the Project window

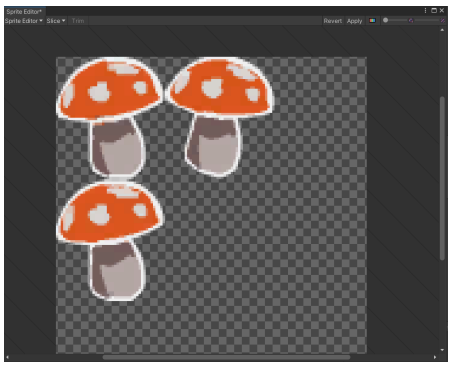


2. Click on the sprite and change Sprite Mode from **Single** to **Multiple** to allow multiple crops of the sprite. Hit apply.

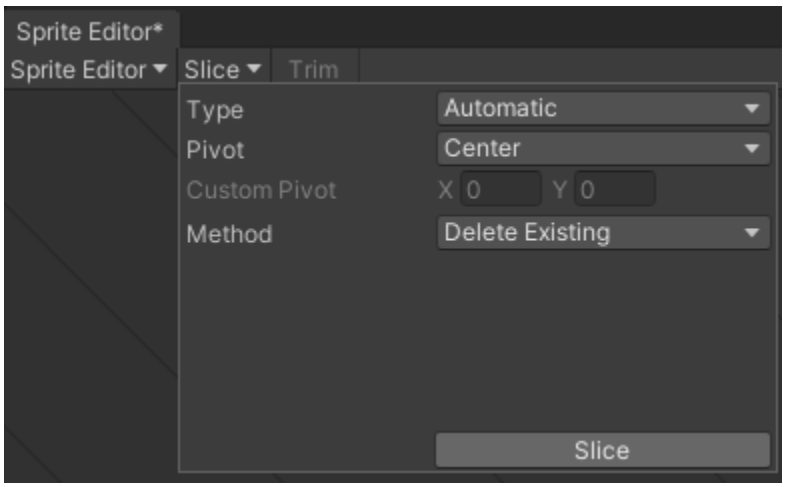


3. Click on **Sprite Editor** and a new window will pop up.





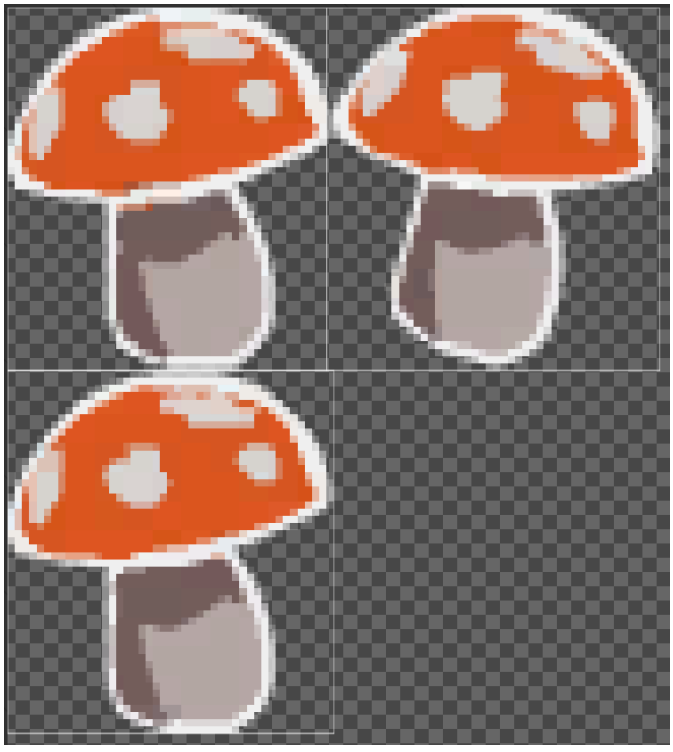
4. Click on **Slice** dropdown **> Slice** to Automatically cut the sprites up into separate images.



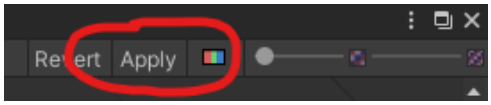
If the 'Slice' was not cut correct where there are joined boxes or the sprites are not separated.

Click on the box delete the Slice.

Manually click and drag a box around each sprite to draw boxes in the order that they are meant to be in the animation.



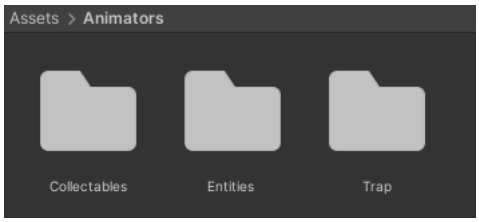
5. Click **Apply**



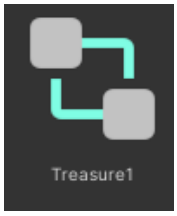
Animation

Animators

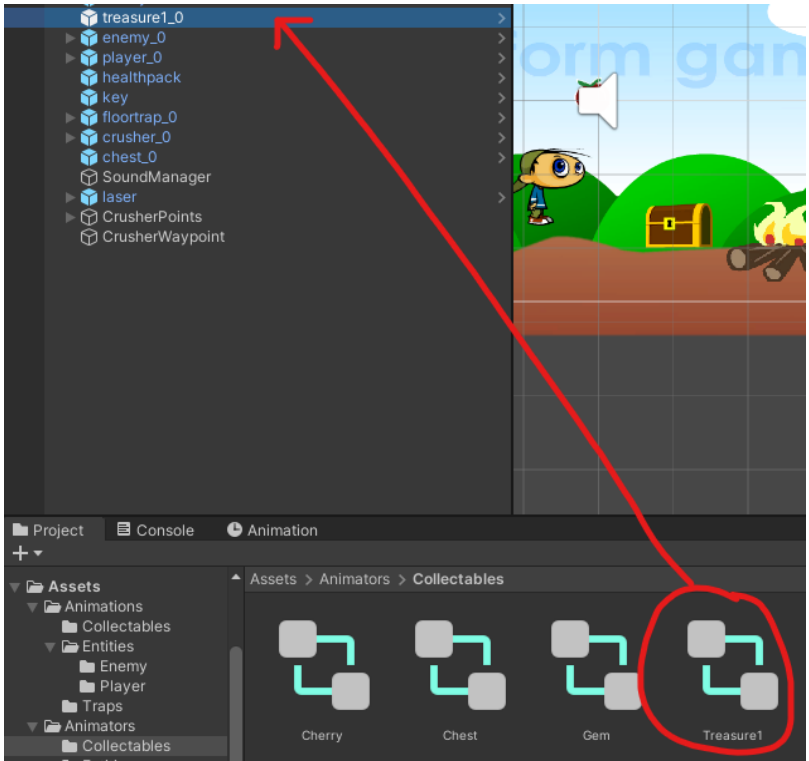
1. Create a new folder called **Collectables** to store your Collectable Animators. Right-click **Assets > Create > Folder**.



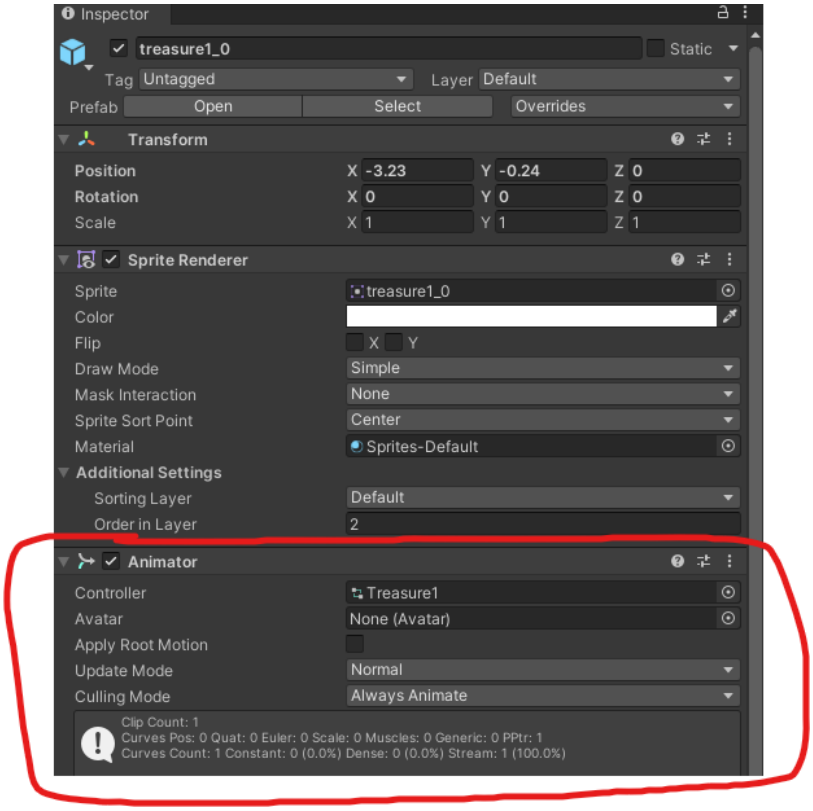
2. Right-click the new **Collectables** folder and create an **Animator Controller**. Right-click **Collectables > Create > Animator Controller**. Rename the new Animator Controller to the same name as the game object it will be for.



3. Click and drag the new **Animator Controller** into the game object it is for.

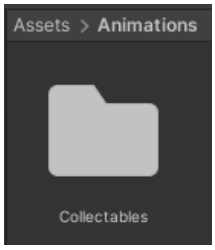


In the Inspector it should appear like this:

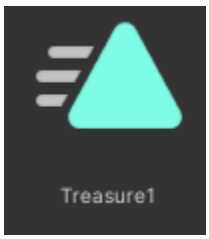


Animation File

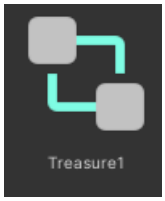
1. Create a new folder called **Collectables** to store your animation folder.



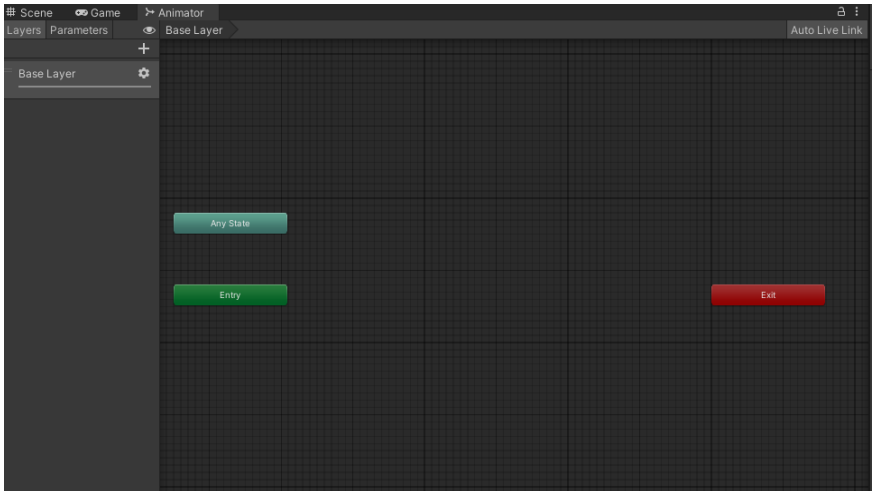
2. Right-click the new **Collectables** folder and create an **Animation**. Right-click **Collectables > Create > Animation**. Rename the new Animation to the same name as the game object it will be for.



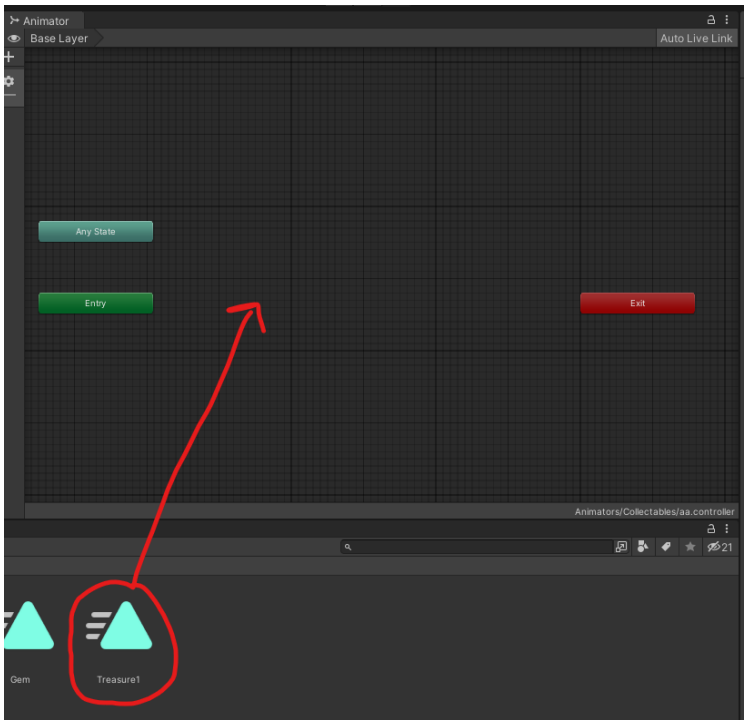
3. Go back to the **Animator Controller** file for your object.



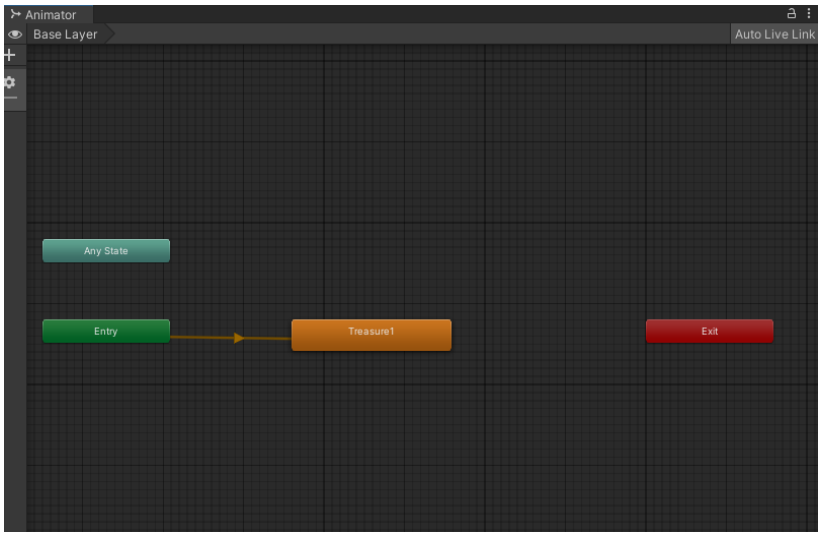
4. Double click it to open an **Animator** window.



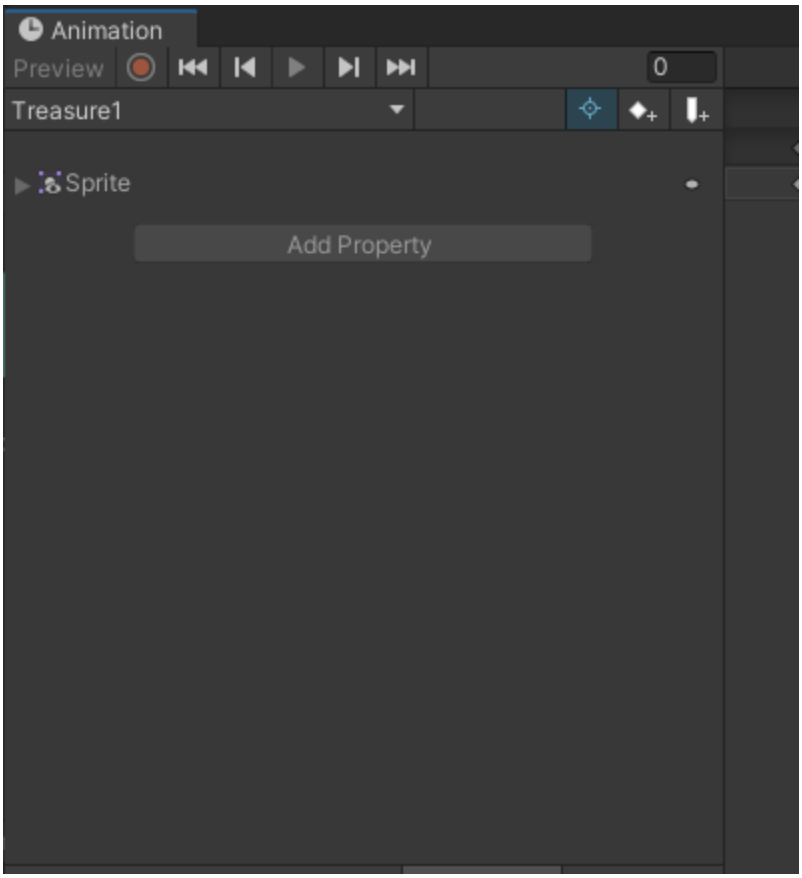
5. Go back to where you stored your **Animation** file. Drag the animation file into the **Animator** window.



It will appear like this:

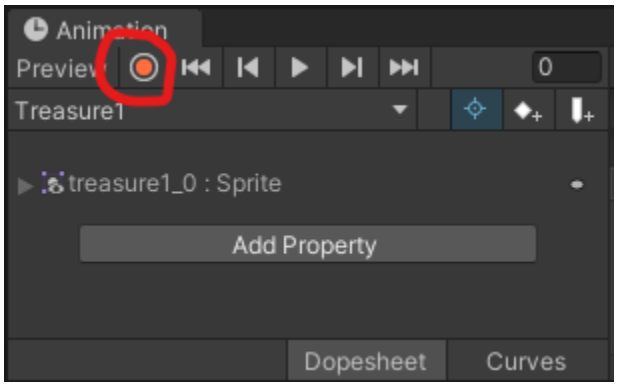


6. Double click the **Animation** file in your **Project** window to open up the **Animation** window to record and edit your animations.



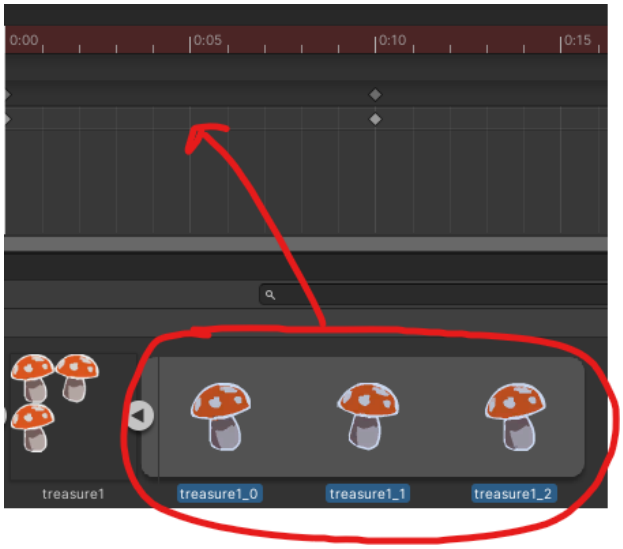
7. In your **Hierarchy** click on the game object for your Animation.

8. Click the record button in the **Animation** window.

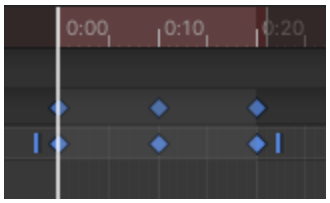


9. Go to your folder with the sprite. Click on the arrow next to the image to expand it and see your **'Sliced'** up sprites.

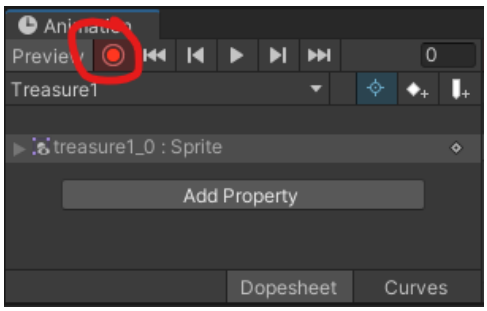
10. Select and drag the sprites into the **Animation** window.



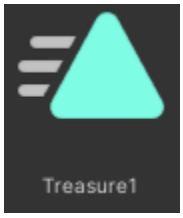
11. Adjust the points in the **Animation** window to your liking.

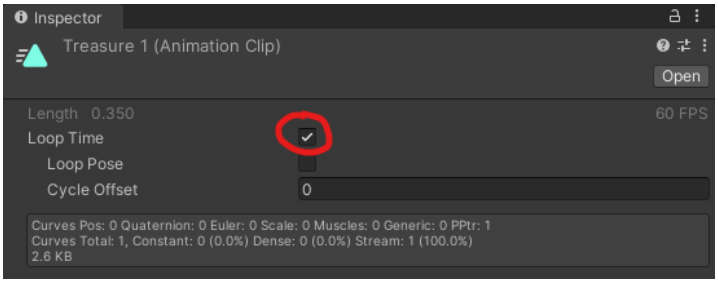


12. Click the record button again to stop recording.



13. Go back to your **Animation** file, click it, and in the **Inspector** check **Loop Time** to allow the animation to loop/repeat.





Traps and Obstacles

Overview

Traps and obstacles are at the core of level design , traps and obstacles will also have the Collider 2D activated and will deal damage to the player , they will be attached to C# scripts to enable their core functionalities.

Scripts Used

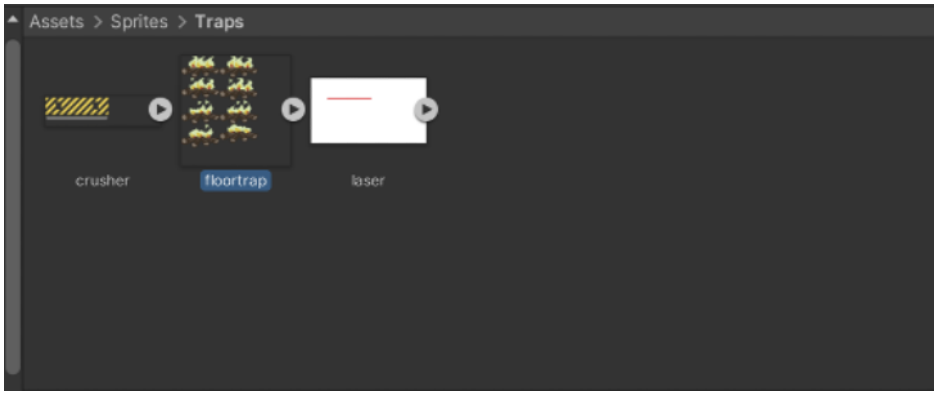
- Laser
- Crusher
- Enemy Health And Damage

Files required

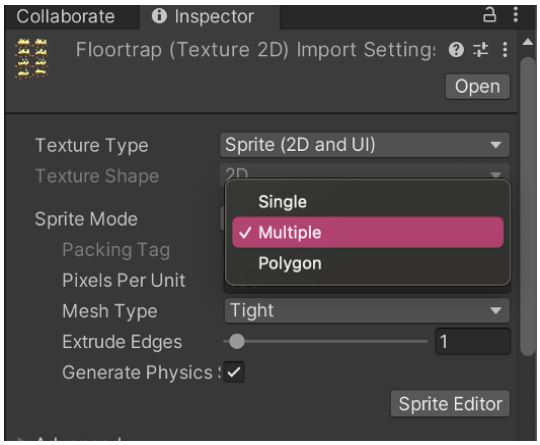
- Crusher and floortrap sprite sheet

Floor trap

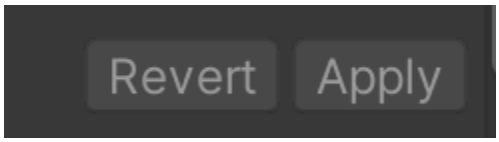
1. In the project window at the bottom screen and select the floor trap sprite sheet.



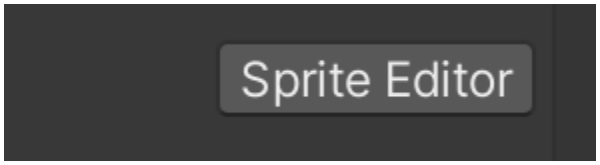
2. On the right-hand side inspector box, all the details of the sprite will pop up. Now change the sprite mode to multiple from the dropdown menu.



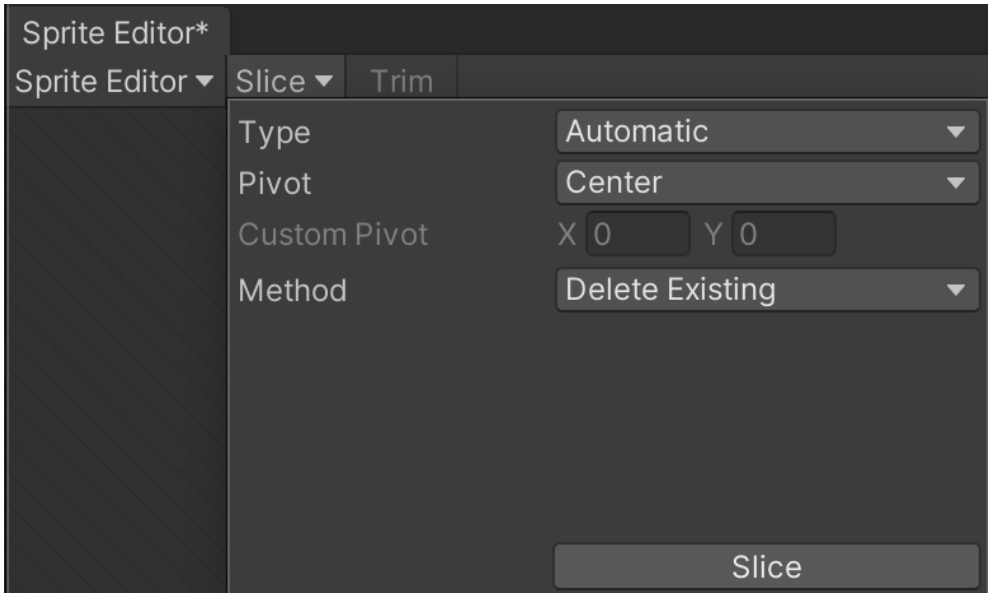
3. Once selected click on apply at the bottom.



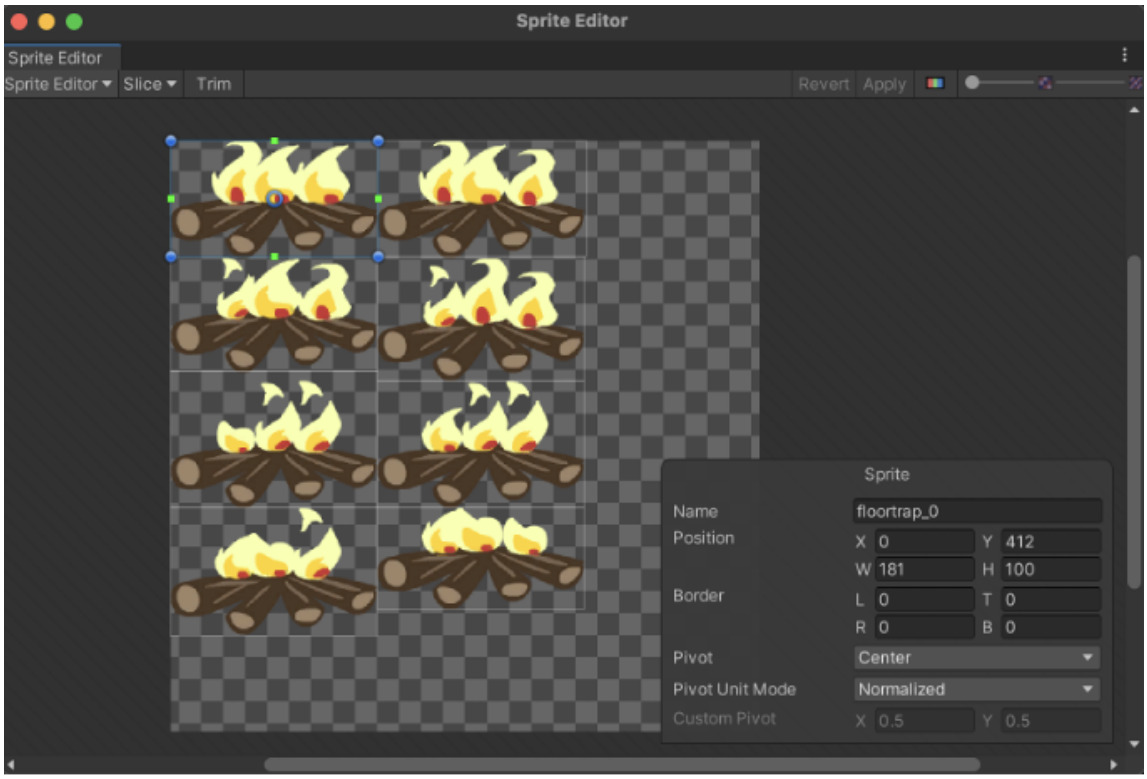
4. Now click on the sprite editor



5. Sprite editor window will open, now select the slice option from the top menu bar and select the type “automatic” and click on slice.



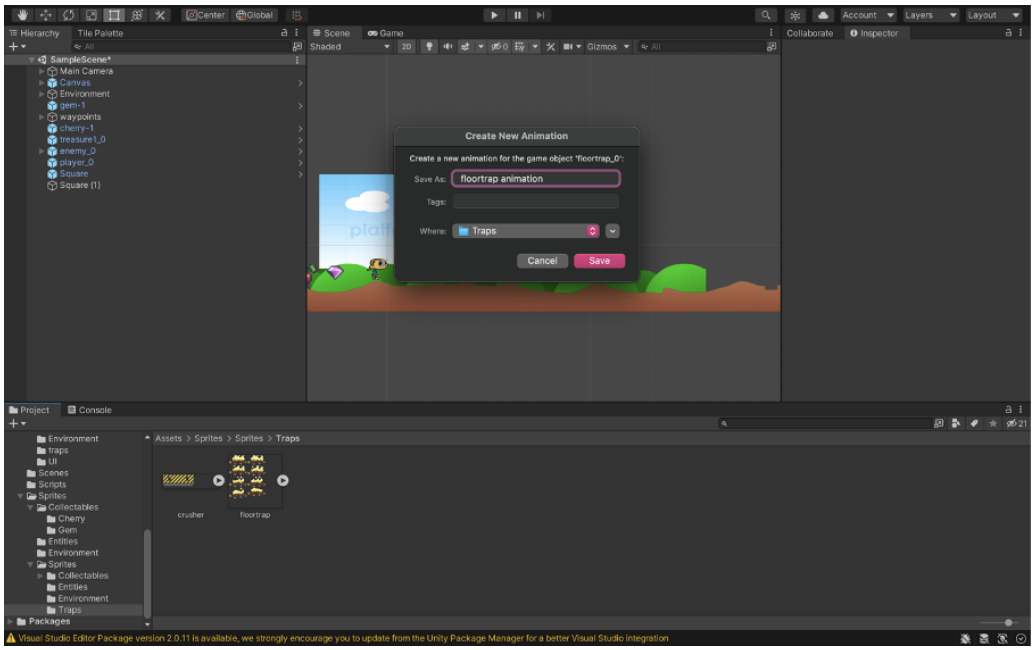
6. If the sprites are not sliced evenly , or there is disparity in shape size , then you're your mouse across the sprite to cover it in a box.Once done your sprite should be split into separate .png files.



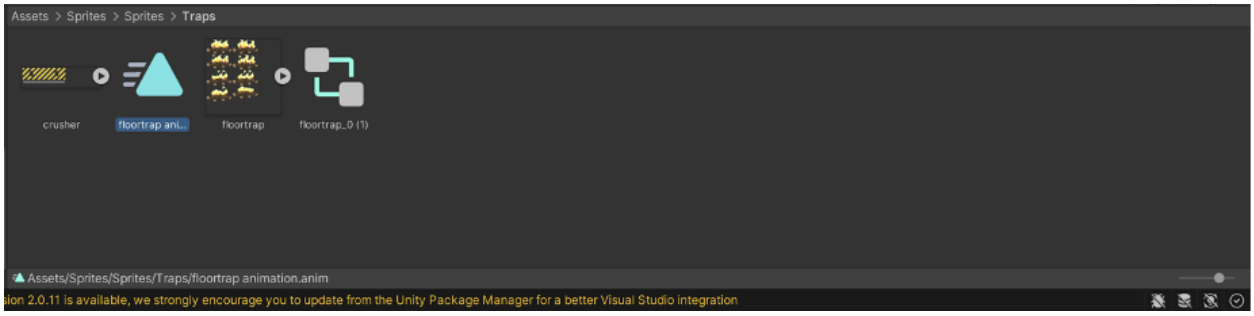
<u>Turning floortraps into gameobjects</u>

1. Navigate to the floortrap sprite, and click on the expand arrow attached, and the sliced sprite will display.

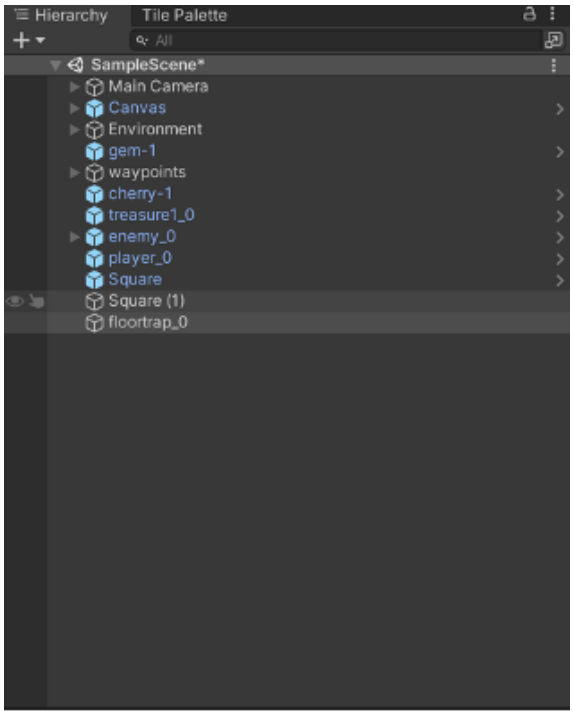
2. Now drag the floortrap sliced image into the scene, once dragged, a “create a new animation” box will pop up.



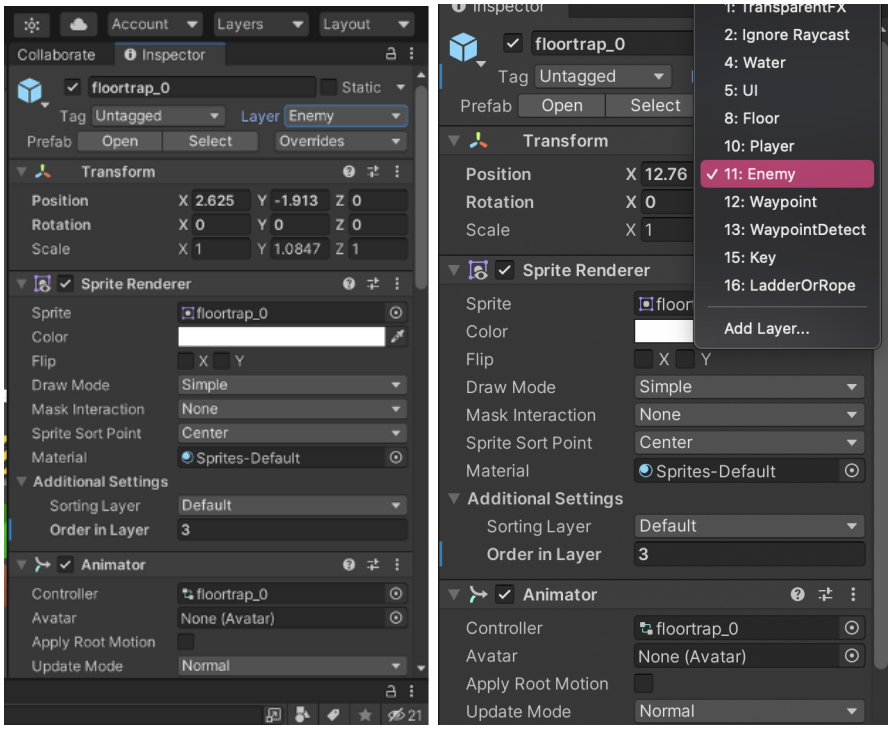
3. One you hit save, the animation clip and the animator will automatically be created. You can cut and paste these animation entities into the animation folder created earlier.



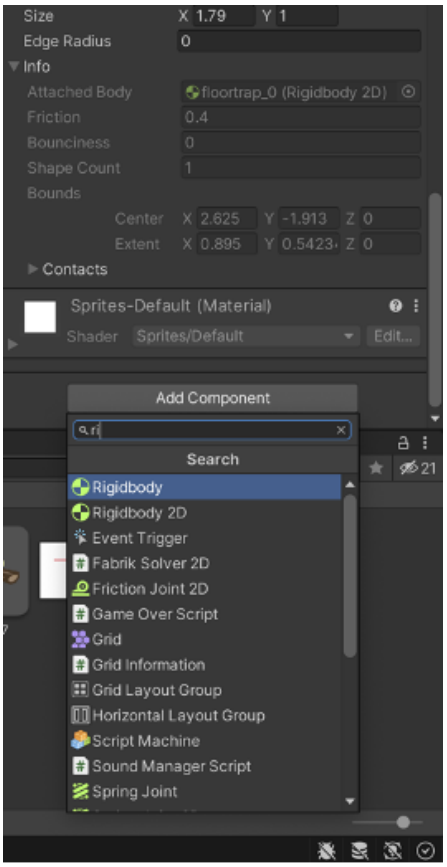
4. Also the floortrap_0 game object will show in the hierarchy box at the top left.



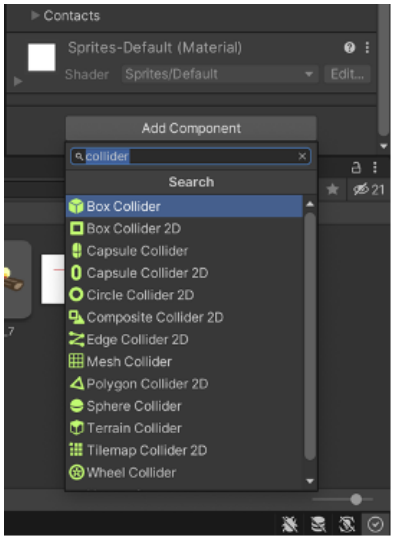
5. Now with the floor trap game object selected from the hierarchy and select the layer enemy from the dropdown menu.



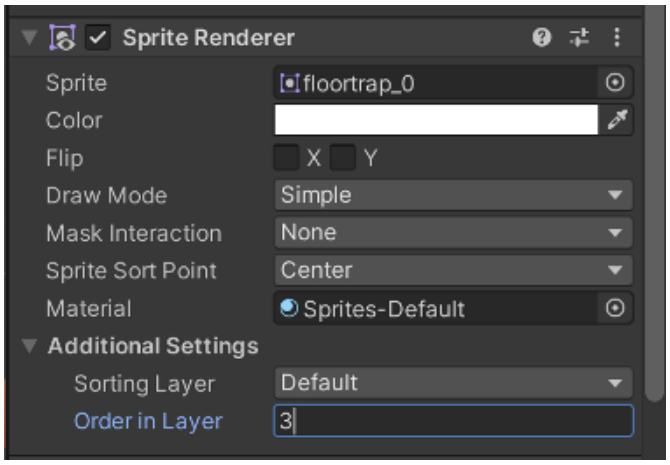
6. Now scroll to the bottom of the inspector and click on the add component button, search for “Rigid body 2d” and select it.



7. Again, click on add component and search for Collider2D, choose a collider of your liking, for this example we will use the Boxcollider2D.

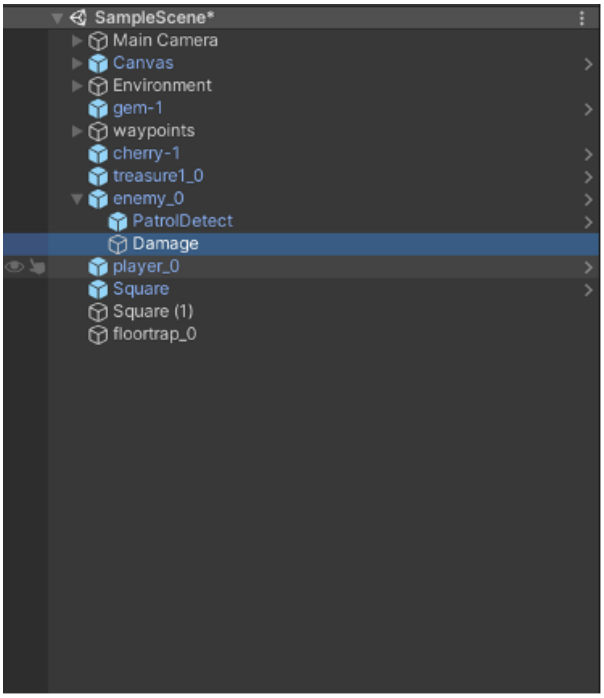


8. In the sprite renderer, set the “order in layer” to layer 3. This is because the enemy and player layer are set in 3, so to avoid traps acting as a blockade, we set “order in layer” to 3.



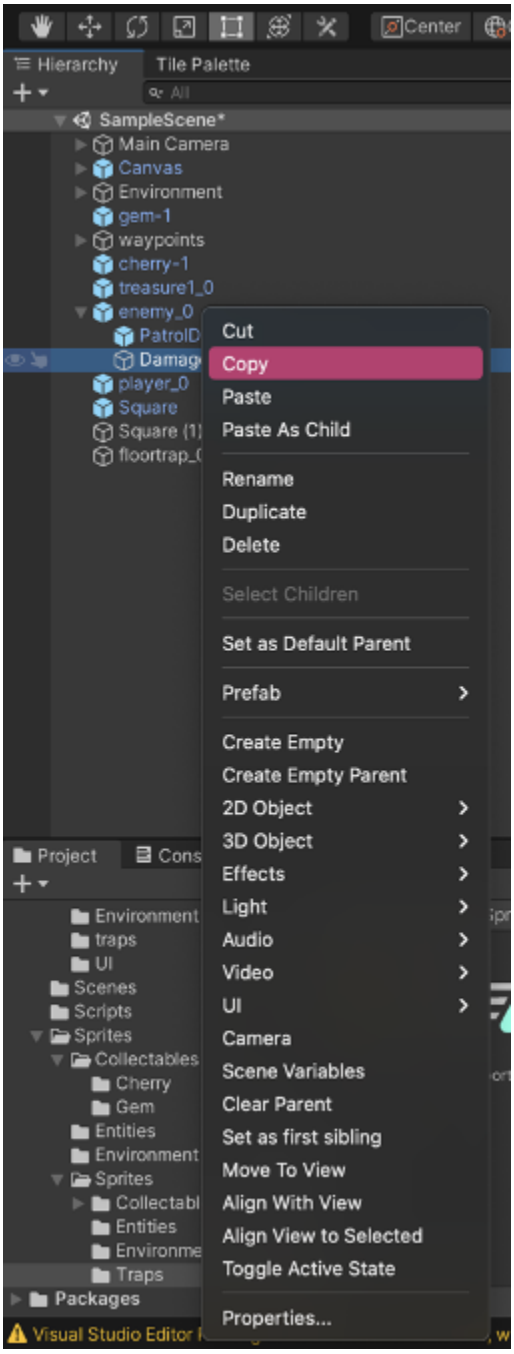
Damage functionality

1. After completing the above step, the floor trap will be animated when the game runs, but when the player walks over it, no damage would be dealt. Now we will add the damage functionality to the script.For the functionality to work, you can reuse the same “damage” game object created for the enemy.

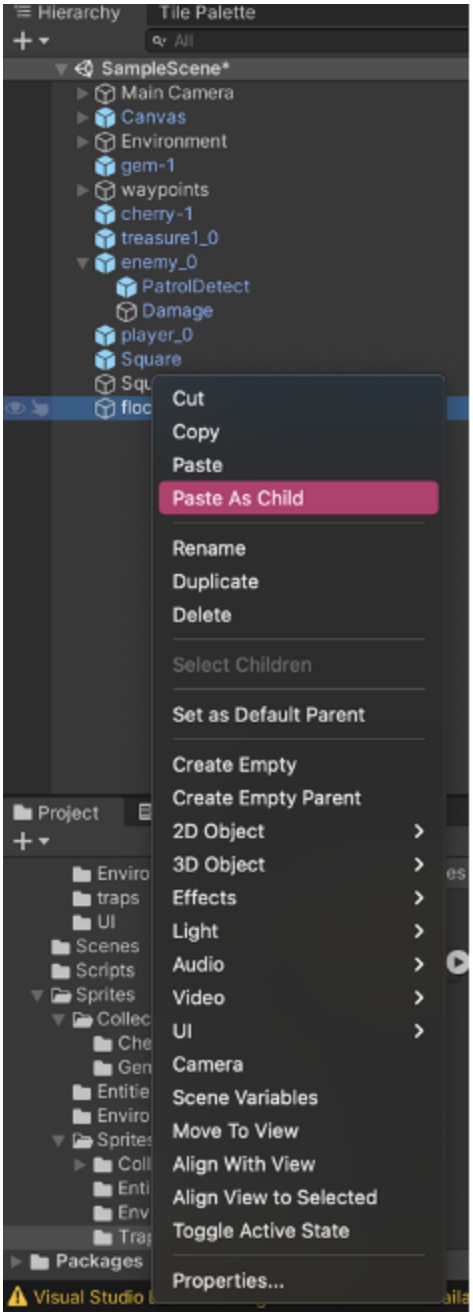


The “damage” game object is attached to the “enemy health and damage script”.

2. Just right click on the “damage” object, the child of “enemy_0” , and click copy.



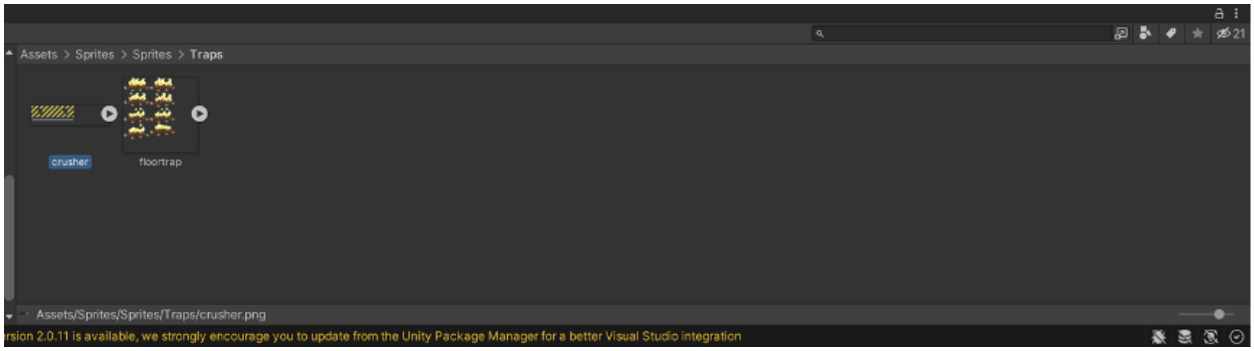
3. Then right click on the “floortrap_0” game object and select, “paste as child”



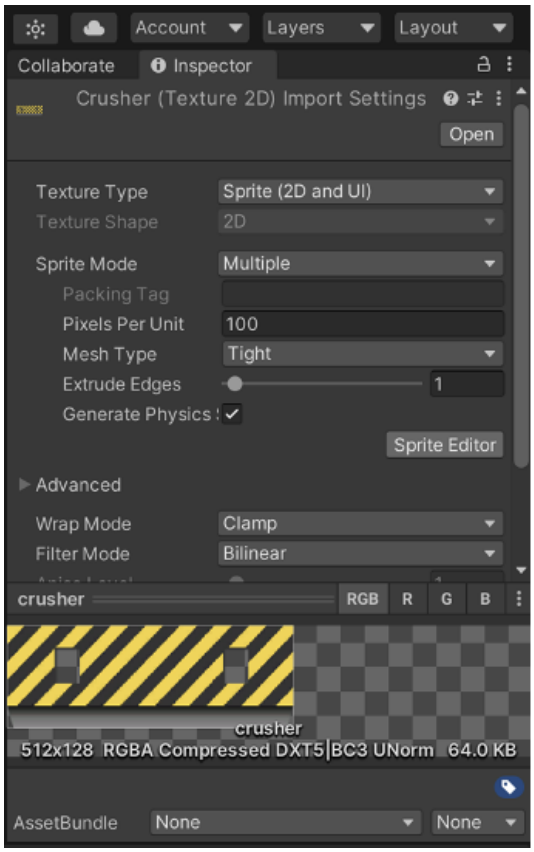
4. You have now created your first obstacle!!

Crusher:

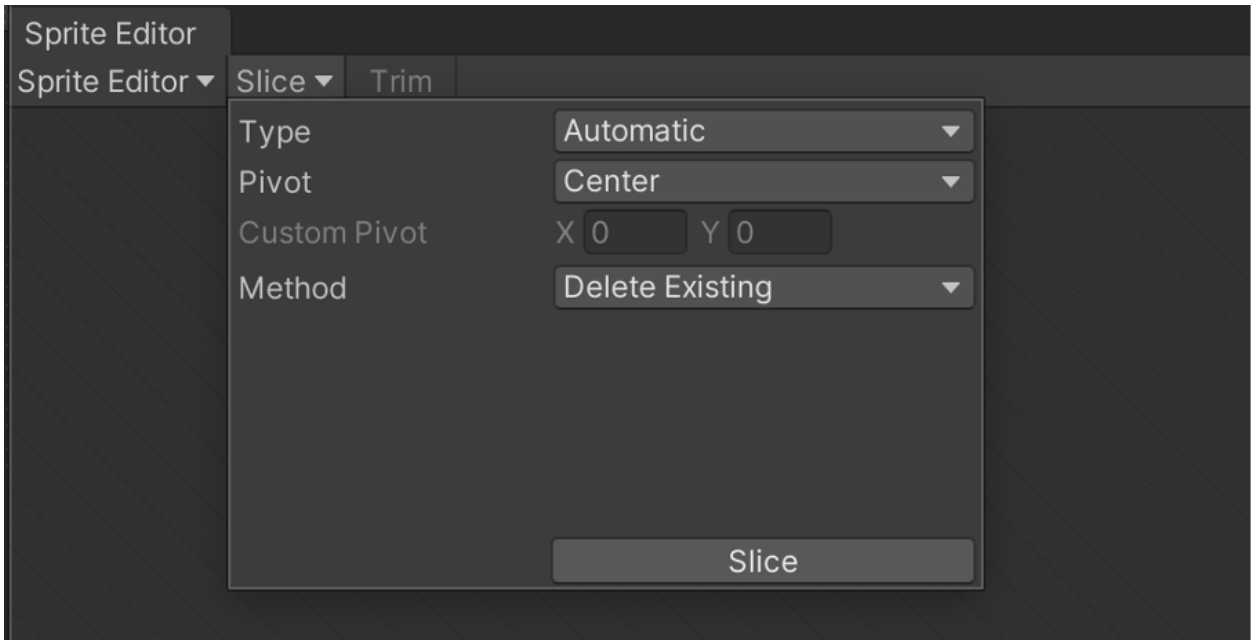
1. Select the crusher sprite from the bottom project tab.



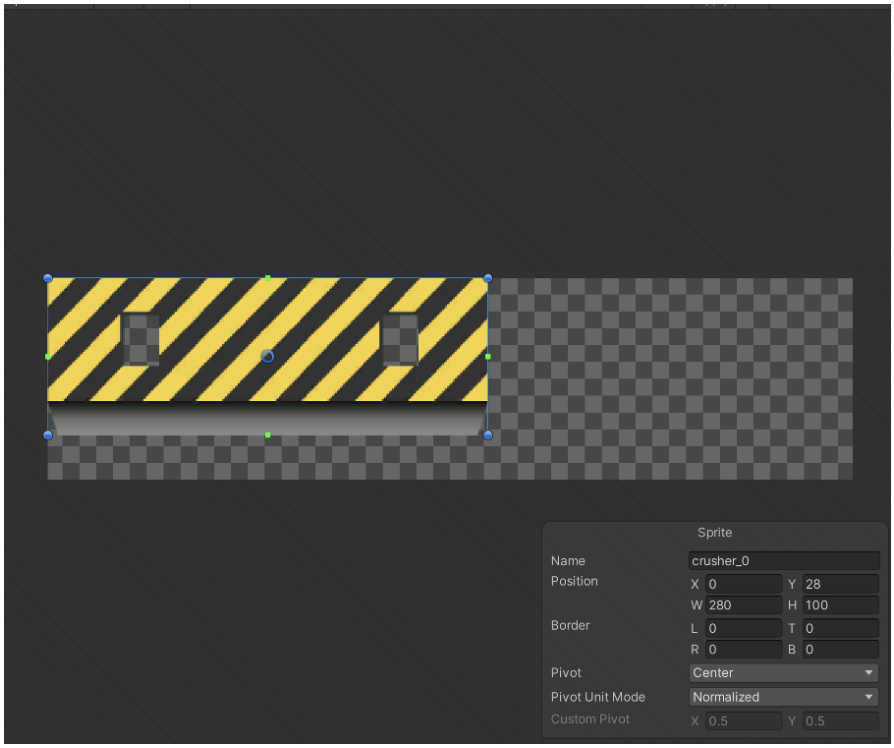
2. From the inspector in the upper righthand side select multiple for the “sprite mode”, click on the sprite editor.



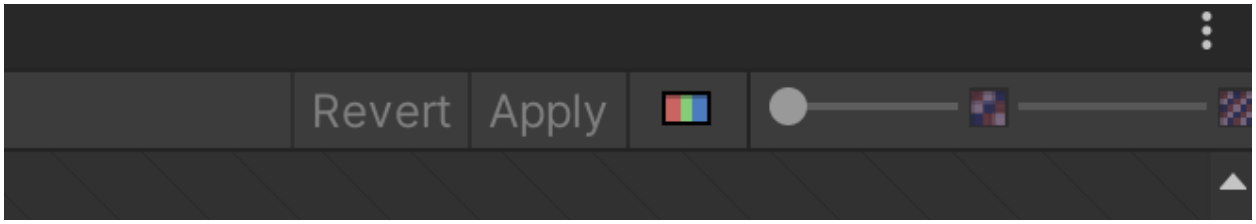
3. Select the slice option and make sure that the automatic option is selected , and then press slice.



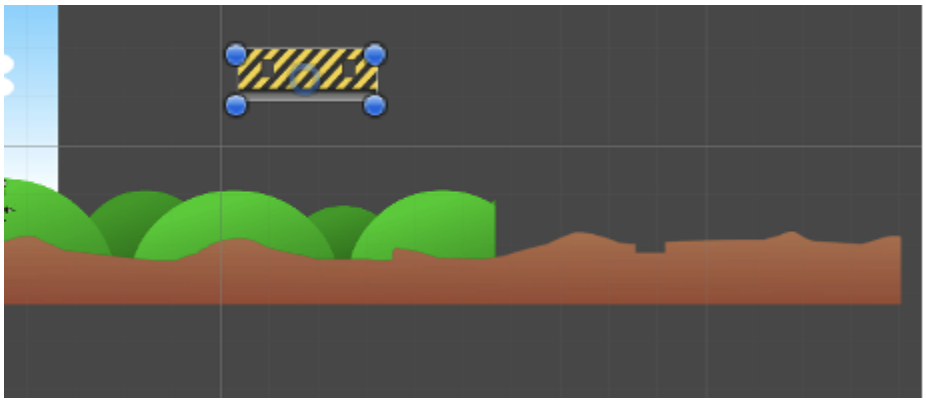
4. Unity will automatically slice precisely to the borders of the crusher.



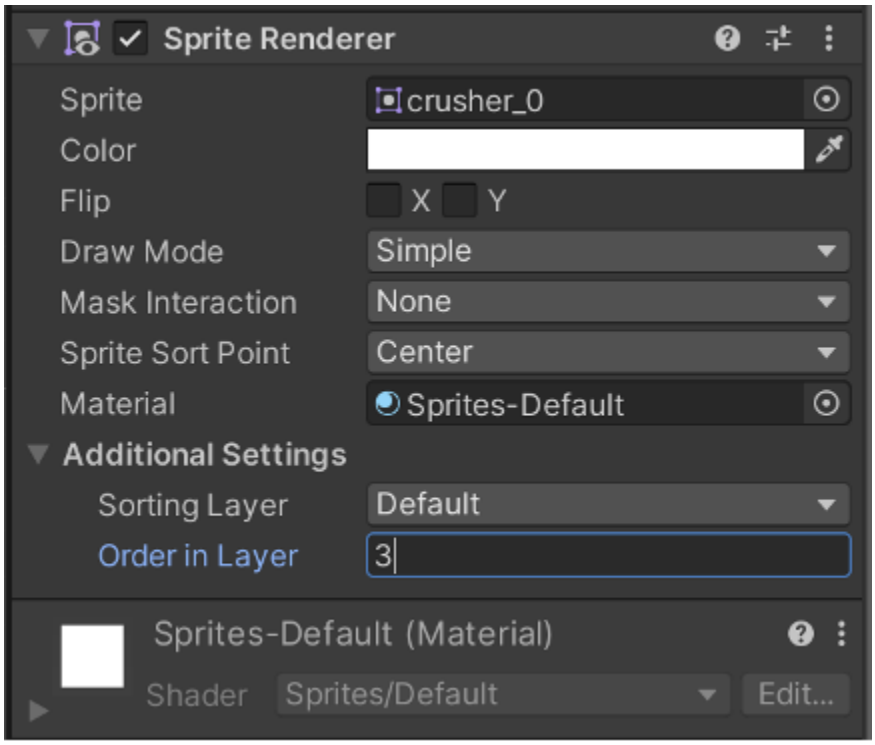
5. Then click on apply for the changes to take place.



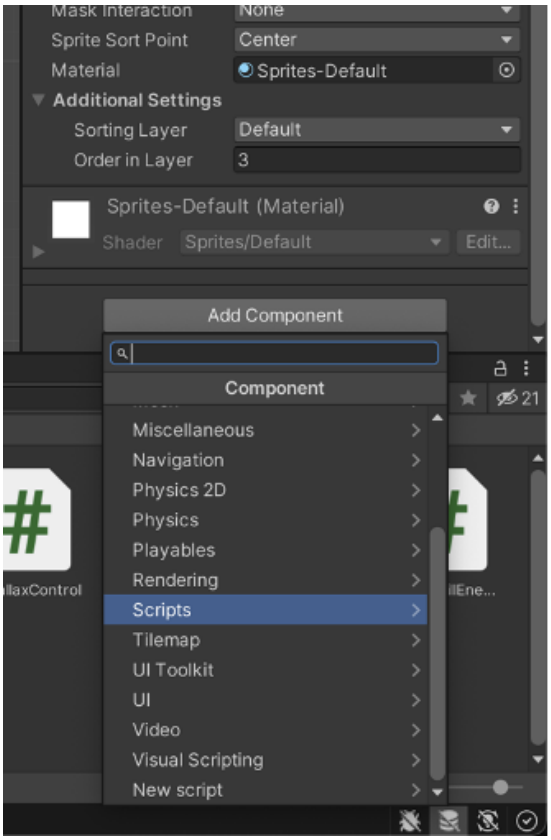
6. Drag the crusher to the hierarchy at the top left, you will notice that the crusher game object appears in the game scene.



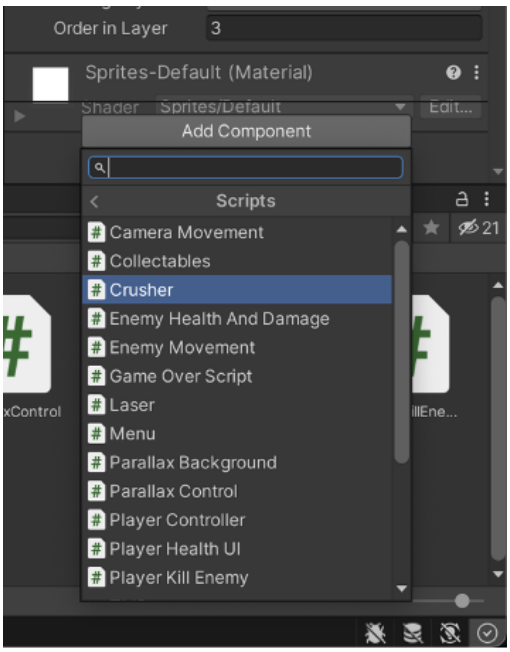
7. Now having selected the crusher game object from the hierarchy, navigate to the sprite renderer component in the inspector and “set order in layer” to 3



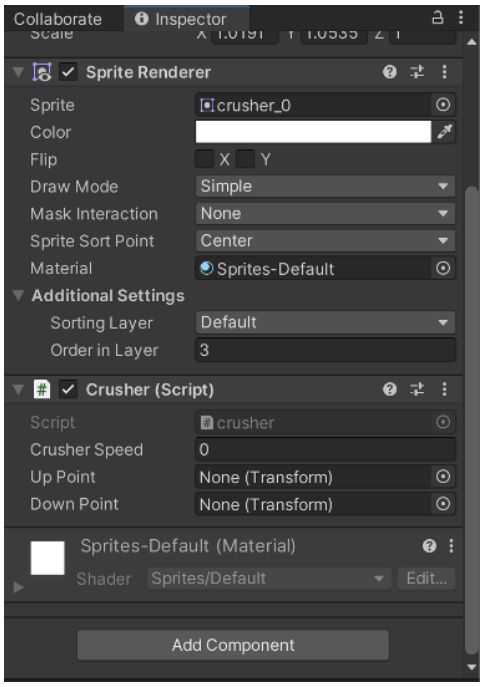
8. Now navigate to the bottom of the inspector and click on “add component” and scroll to look for the “scripts” option.



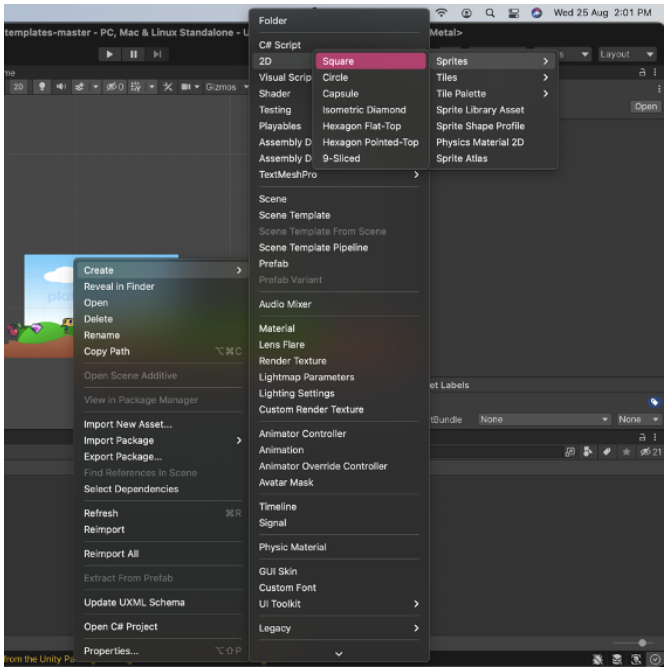
9. Now select the crusher script from the options, all the scripts are pulled from the assets > scripts folder.



10. You will set that the component will be added to the crusher.

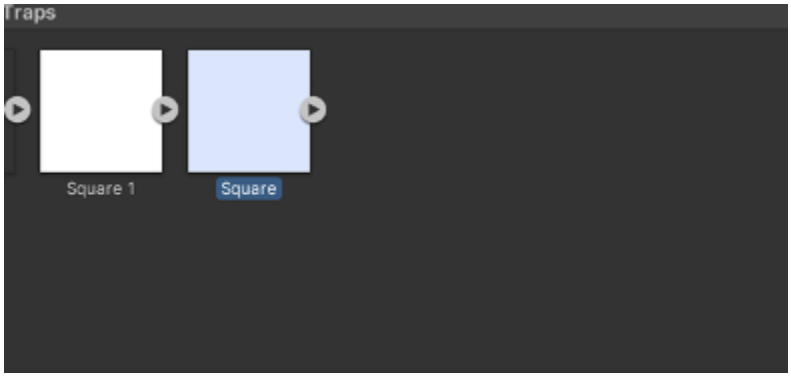


11. Now right click in the project folder and create a sprite, choosing create > 2D > sprite > square. Do this two times to create two squares.

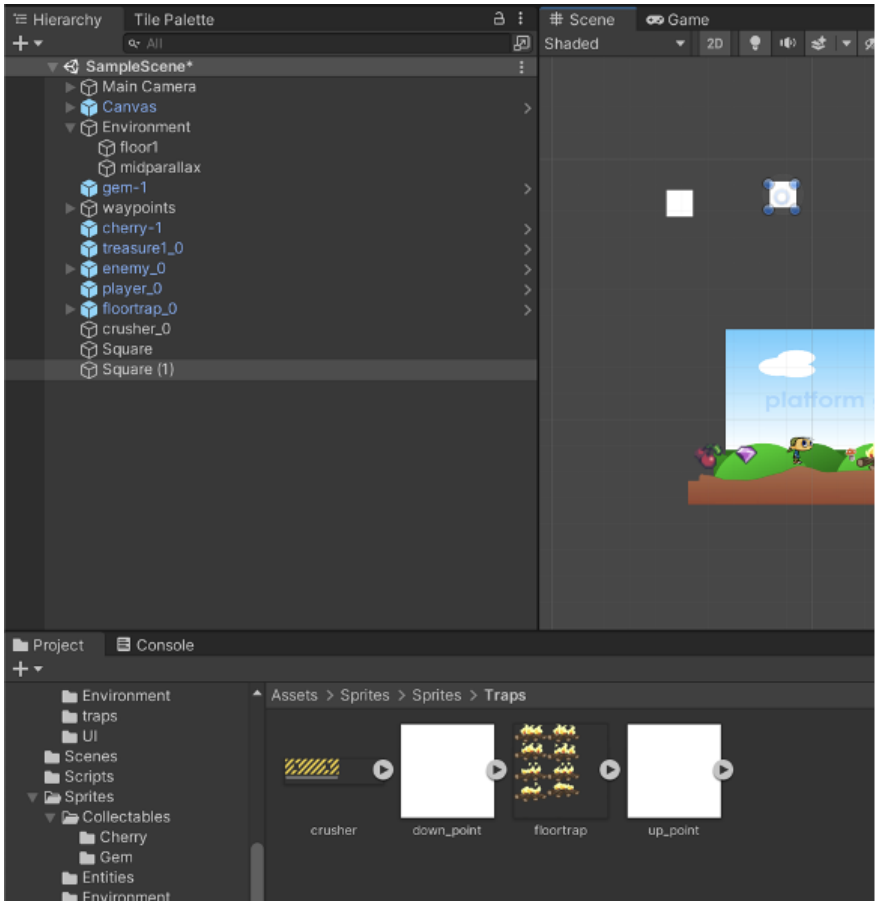


you can select any other shapes other than square, these sprites act as reference point for the crusher to move up and down.

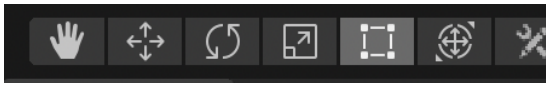
12. the square will show in the bottom projects tab as sprites



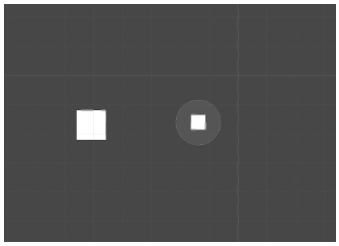
13. Rename these sprites as “up_point” and “down_point” respectively , then drag them to the game scene.



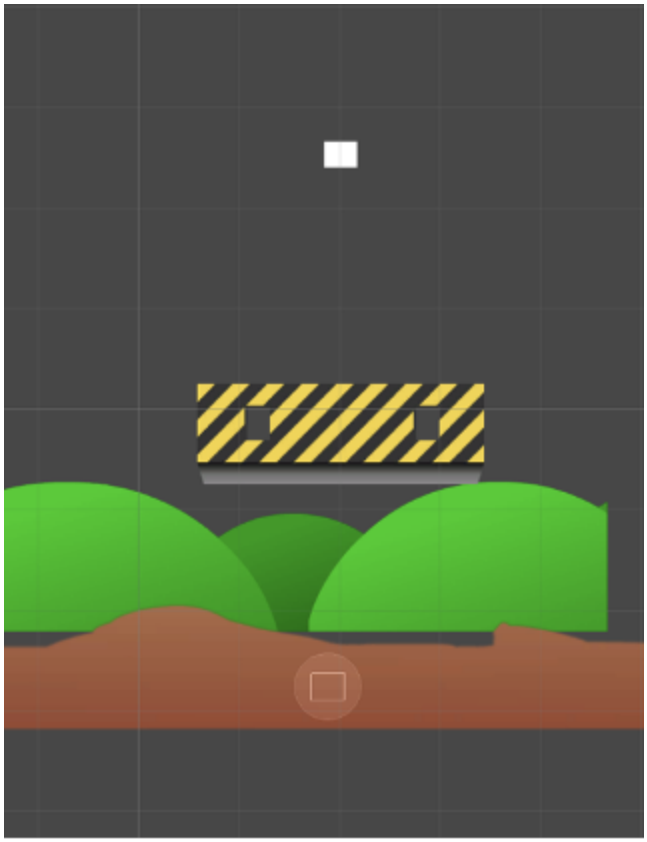
14. Choose the “rect tool” from the top left section.



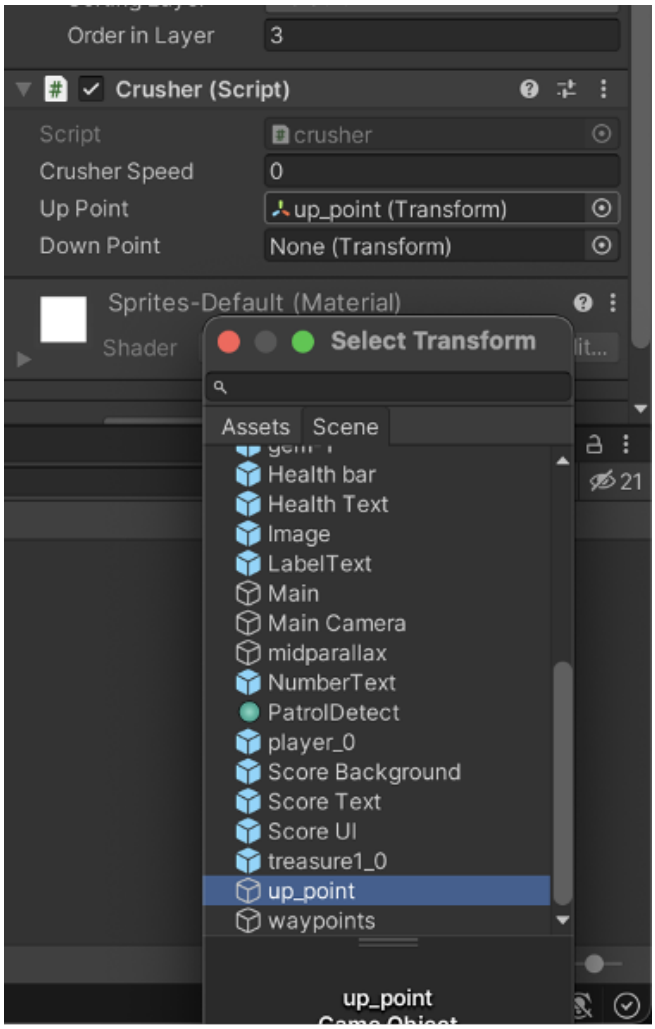
15. Reduce the size of the sprite by dragging the corner of the sprite inwards.



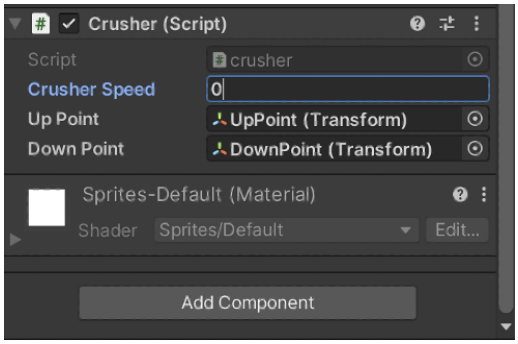
16. Then place the square sprites along the vertical axis, any place along the game floor. Up_ point sprite should be placed above the floor and the down_point behind the floor. The crusher will oscillate up and down along these squares.



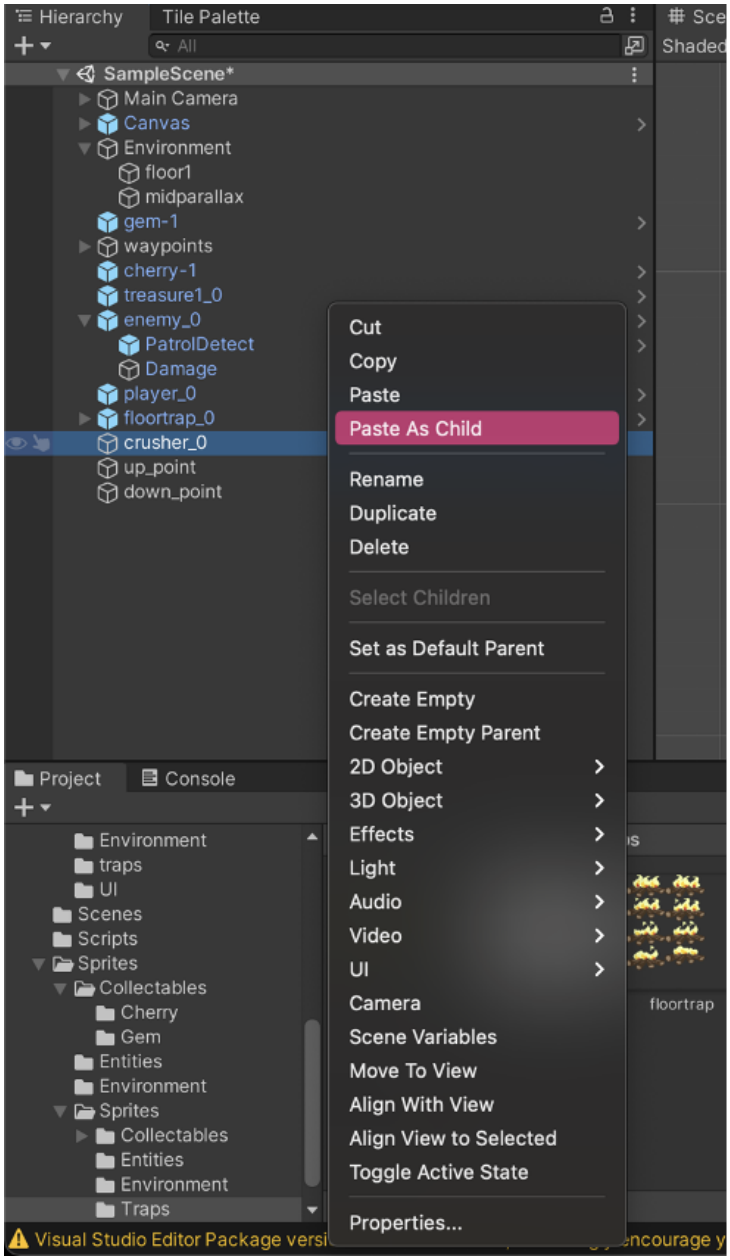
17. Now in the script component of the game object in the inspector, select the up point as the up_point game object and down point as the down_point game object.



18. And set the crusher speed to your liking.

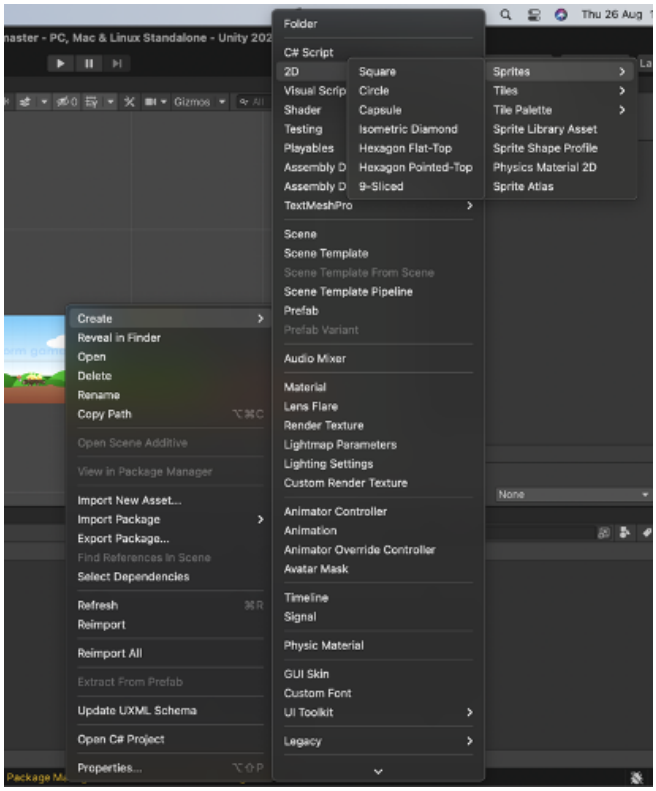


19. Now as the last part again, copy the damage game object from the enemy and “past as a child” inside the crusher game object in the hierarchy.

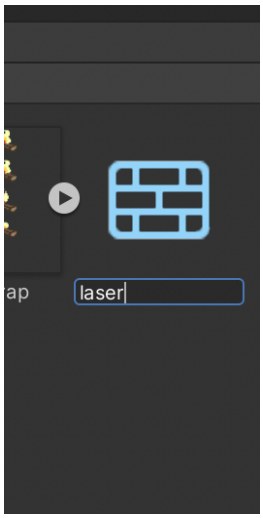

Hierarchy
Tile Palette
All
SampleScene*
Main Camera
Canvas
Environment
floor1
midparallax
gem-1
waypoints
cherry-1
treasure1_0
enemy_0
PatrolDetect
Damage
player_0
floortrap_0
crusher_0
up_point
down_point
Cut
Copy
Paste
Paste As Child
Rename
Duplicate
Delete
Select Children
Set as Default Parent
Create Empty
Create Empty Parent
2D Object
3D Object
Effects
Light
Audio
Video
UI
Camera
Scene Variables
Move To View
Align With View
Align View to Selected
Toggle Active State
Properties...
Project
Console
Environment
traps
UI
Scenes
Scripts
Sprites
Collectables
Cherry
Gem
Entities
Environment
Sprites
Collectables
Entities
Environment
Traps
floortrap
Visual Studio Editor Package versi

Laser

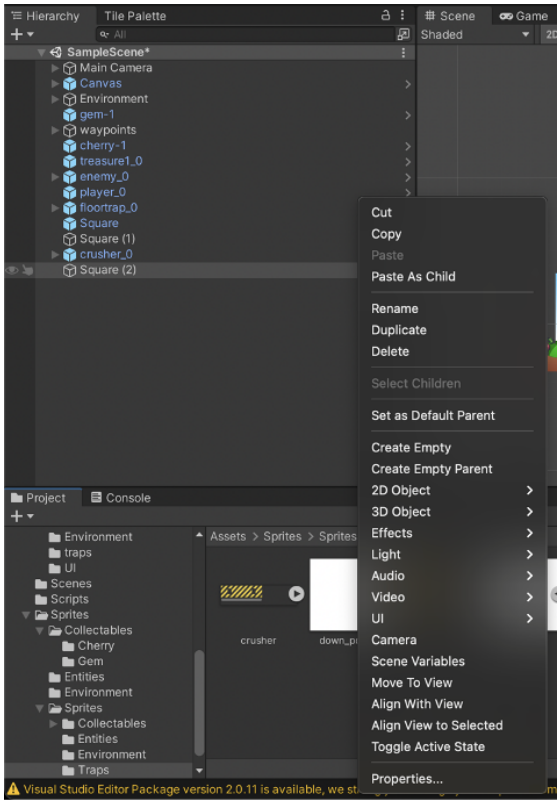
1. Right click on the bottom projects area and create a 2D square sprite.



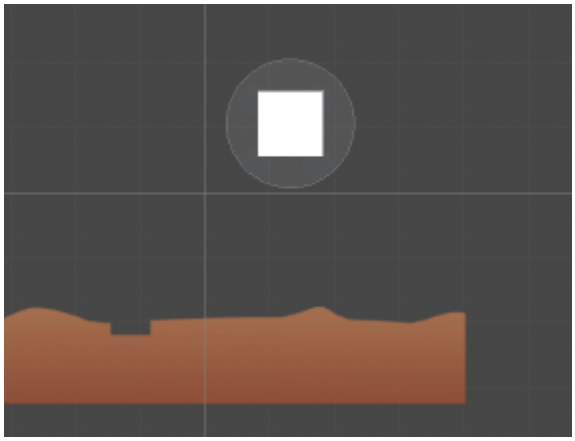
2.The sprite will have a default shape name .Rename the sprite to laser and press enter.



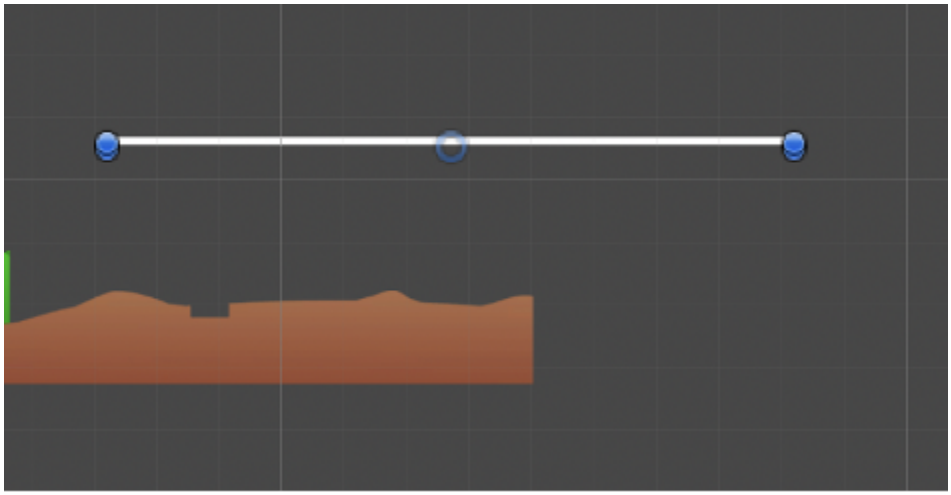
3.Drag the laser sprite form the projects area to the hierarchy at the top left and rename the gameobject to laser



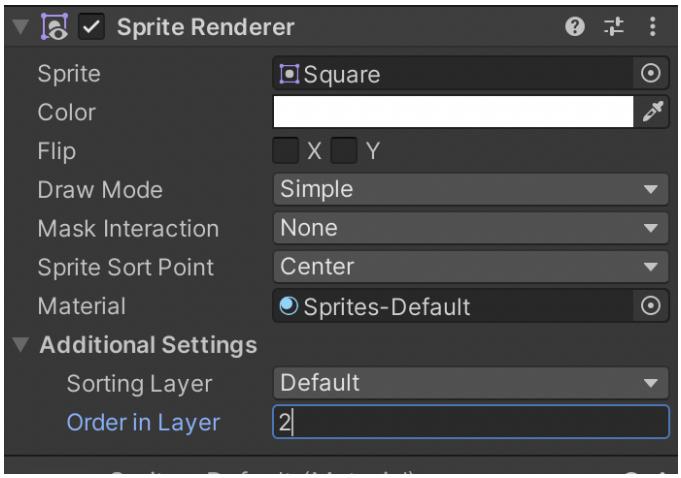
4.The square sprite will appear in the game scene.



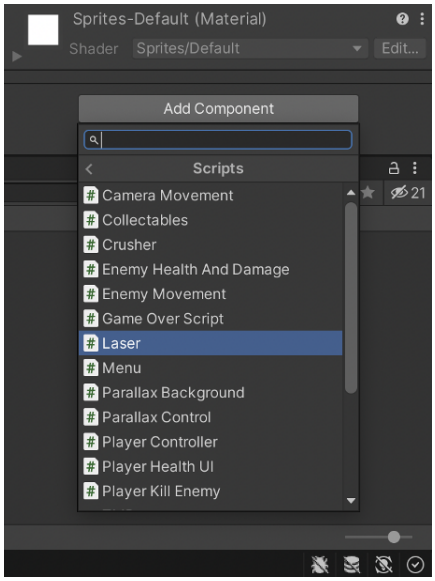
5. Now point your mouse to the corner of the sprite to click and drag to change the shape of the sprite using the “rect tool” from before.



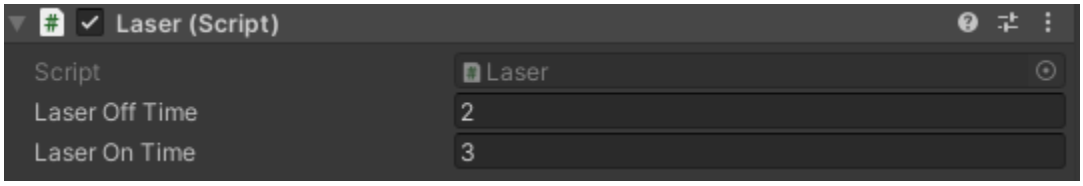
6.While keeping the sprite selected, change the value “order in layer” to 2 under the “sprite renderer component”.



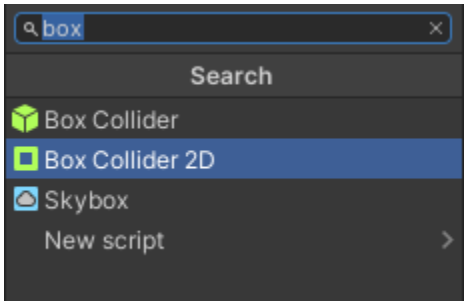
7.Now click on add component at the bottom of the inspector window and scroll down to scripts and select laser.

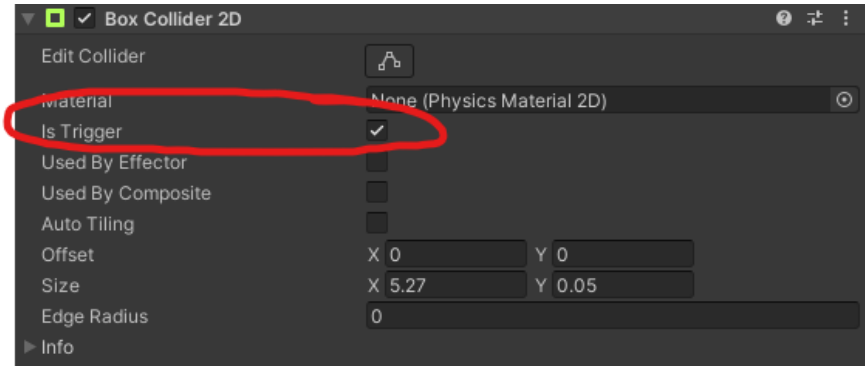


- **Laser Off Time - the amount of seconds the laser will be off for**
- **Laser On Time - the amount of seconds the laser will be active for**

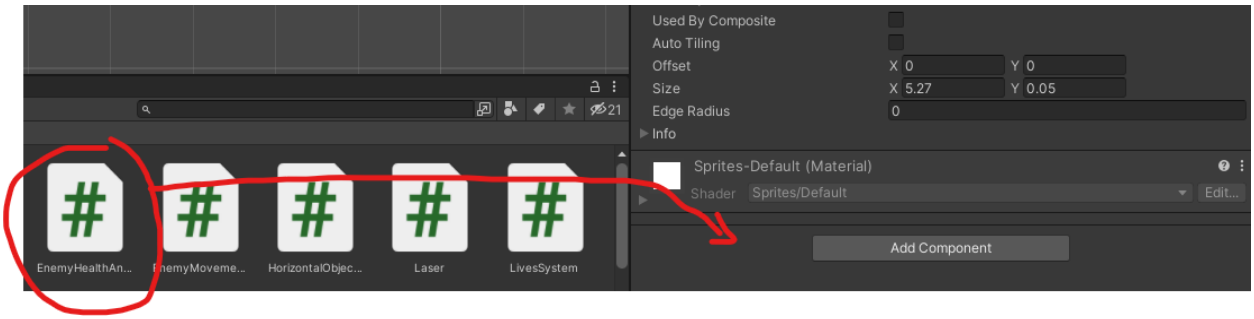


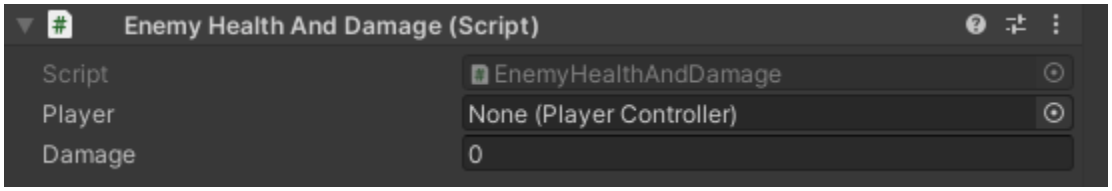
8. Search and add the **Box Collider 2D** component. Within the **Box Collider 2D** component, tick the **IsTrigger** checkbox.



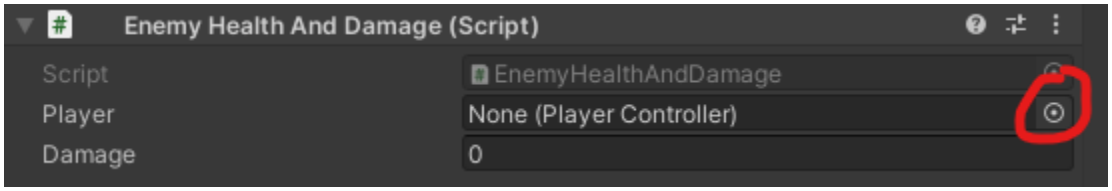


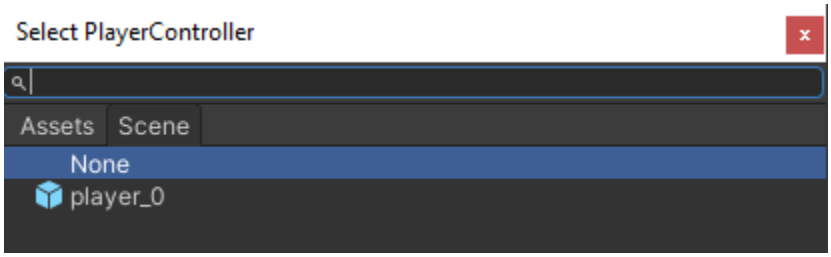
9. Drag the **EnemyHealthAndDamage** script into your '**laser**' **Hierarchy**.



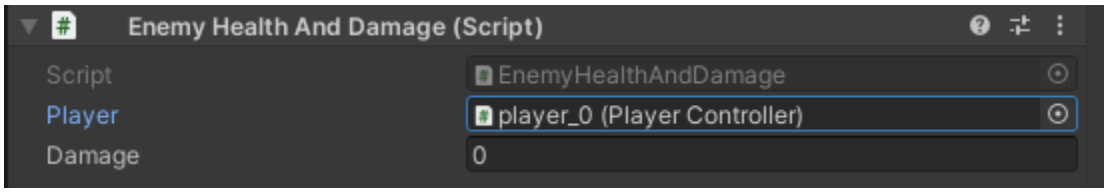


10. For the **Player** component section, click the circle next on the right. A new popup will appear.

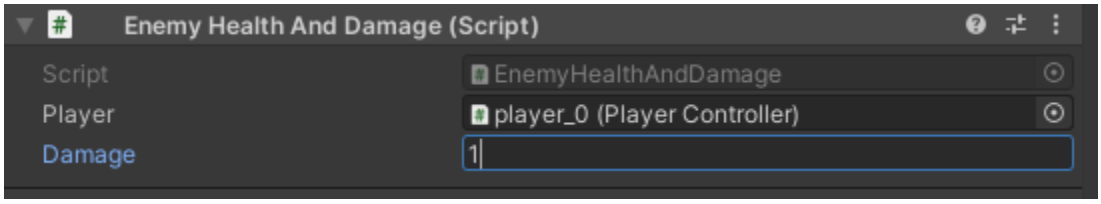




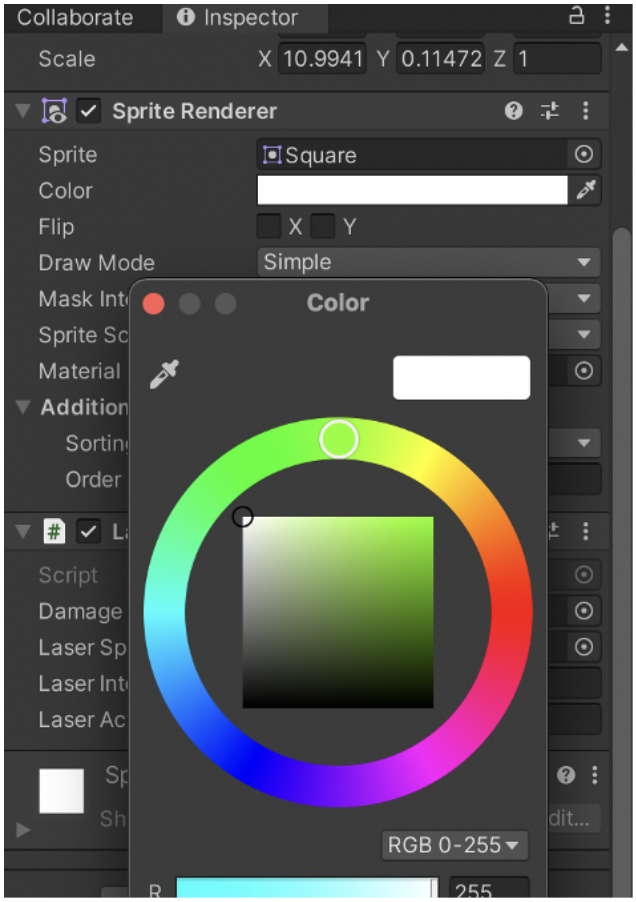
11. In **Scene** of that popup, click your player game object with the **PlayerController** script. In this case it would be for us, **player_0**.



12. Type in the amount of Damage your laser will do per in-game frame.



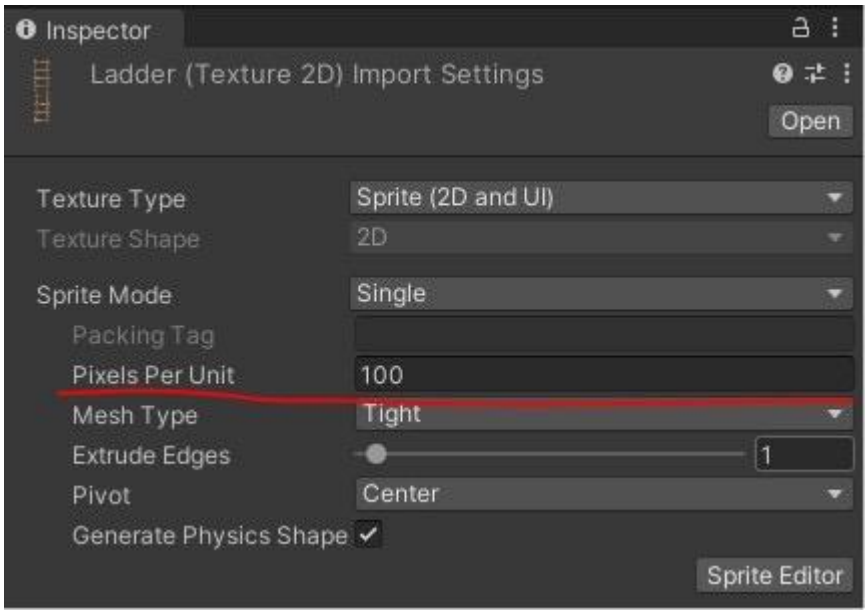
13. You can customize the color of the laser to anything by simply choosing a color from the color option under the inspector's sprite renderer component.



Creating Ladders:

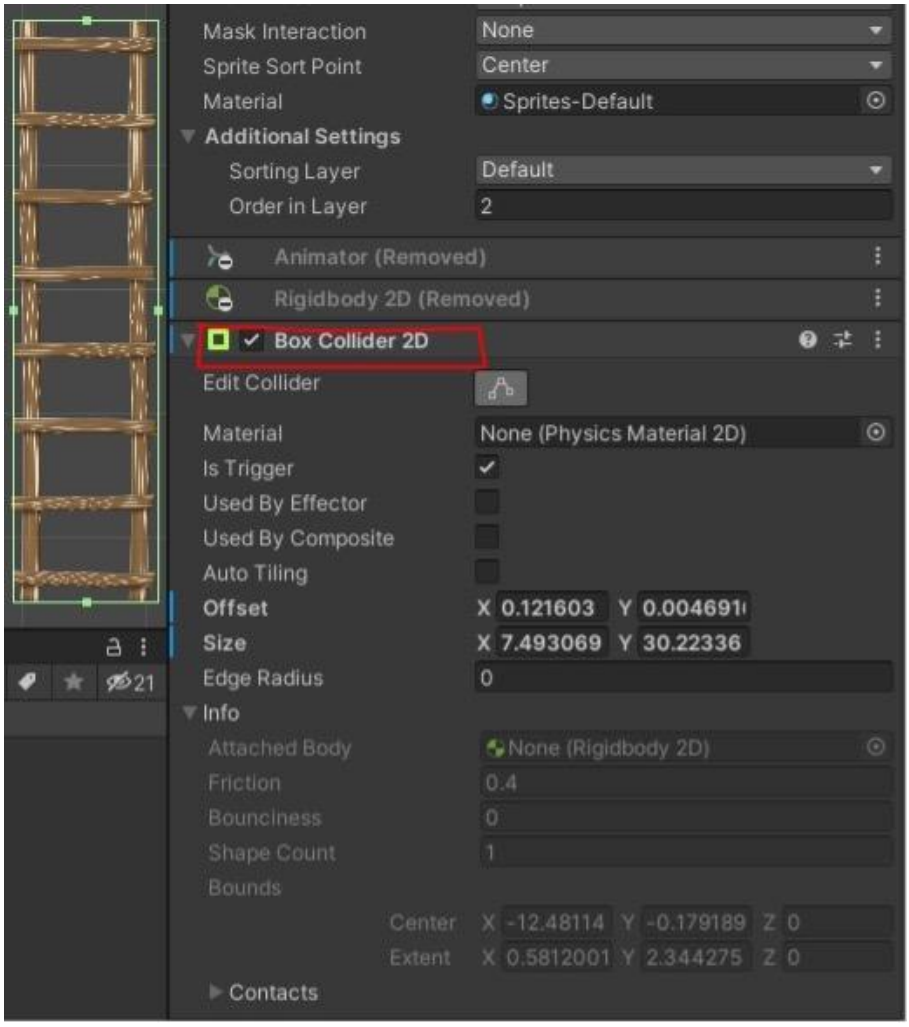
1) Download an image that suits the ladder.
2) Drag n drop that image in the sprites folder of the project.

3) Rename the file name to “Ladder”.
4) Adjust its size according to the scene by changing the “Pixels Per Unit” value.

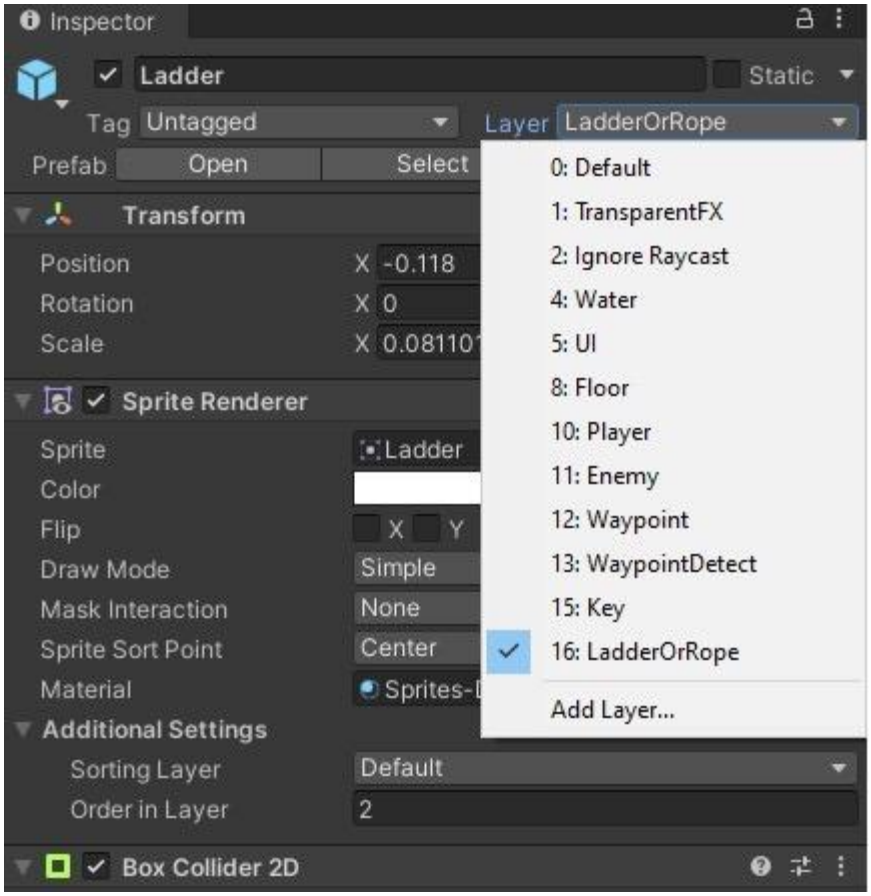


5) Now, drag n drop the created sprite into the scene so that it can be converted into a scene object.

6) Apply **“Box Collider 2D”** component to the object and adjust the collider size according to the sprite and check **“Is Trigger”** so that the player can detect the collision for it to be used as a trigger to perform a specific function.



7) Now change the Layer of the object from “Default” to “LadderOrRope”.



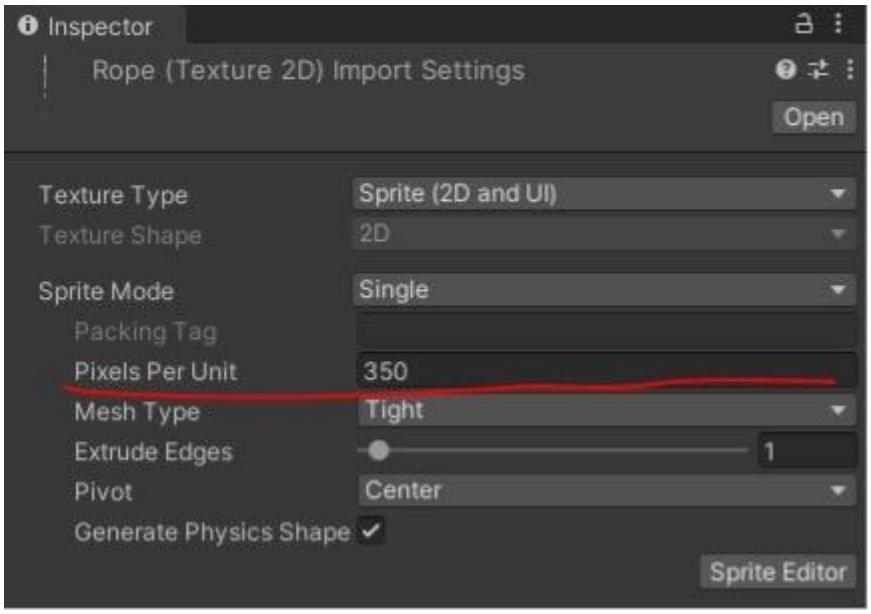
Creating Swinging Ropes:

1. Download an image that suits the rope.

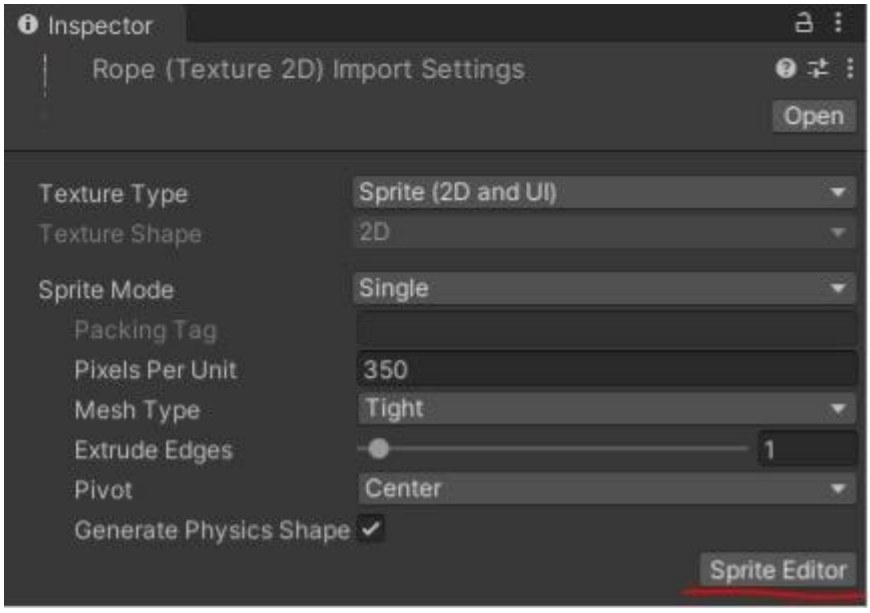
2. Drag and drop that image in the sprites folder of the project.

3. Rename the file name to “Rope”.

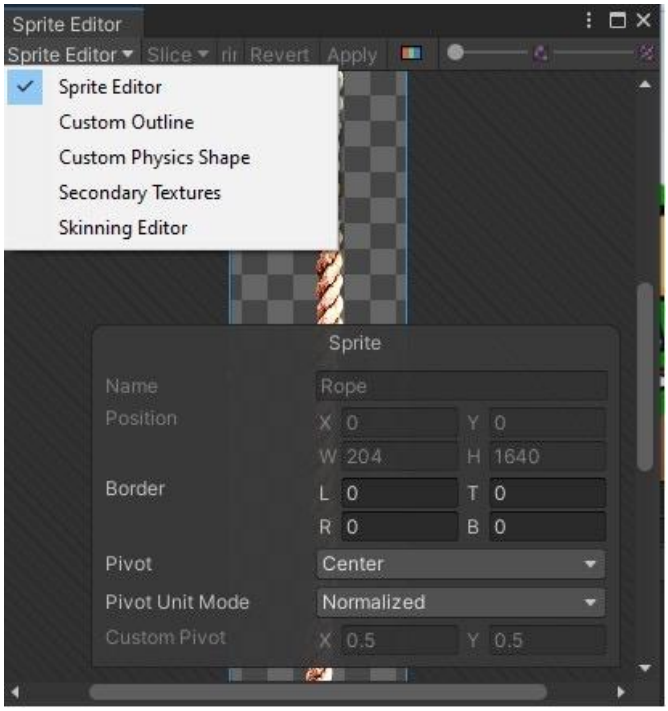
4. Adjust its size according to the scene by changing the “**Pixels Per Unit”** value.

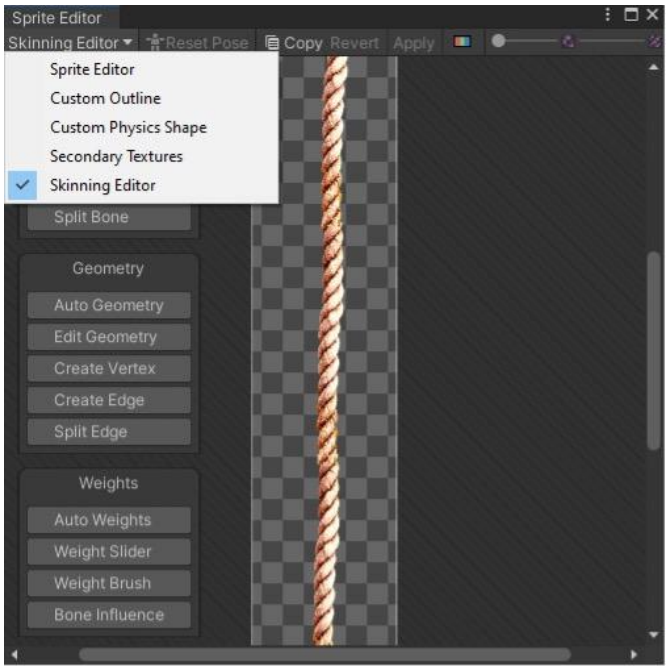


5. Now, with the use of Unity`s “Sprite Editor”,

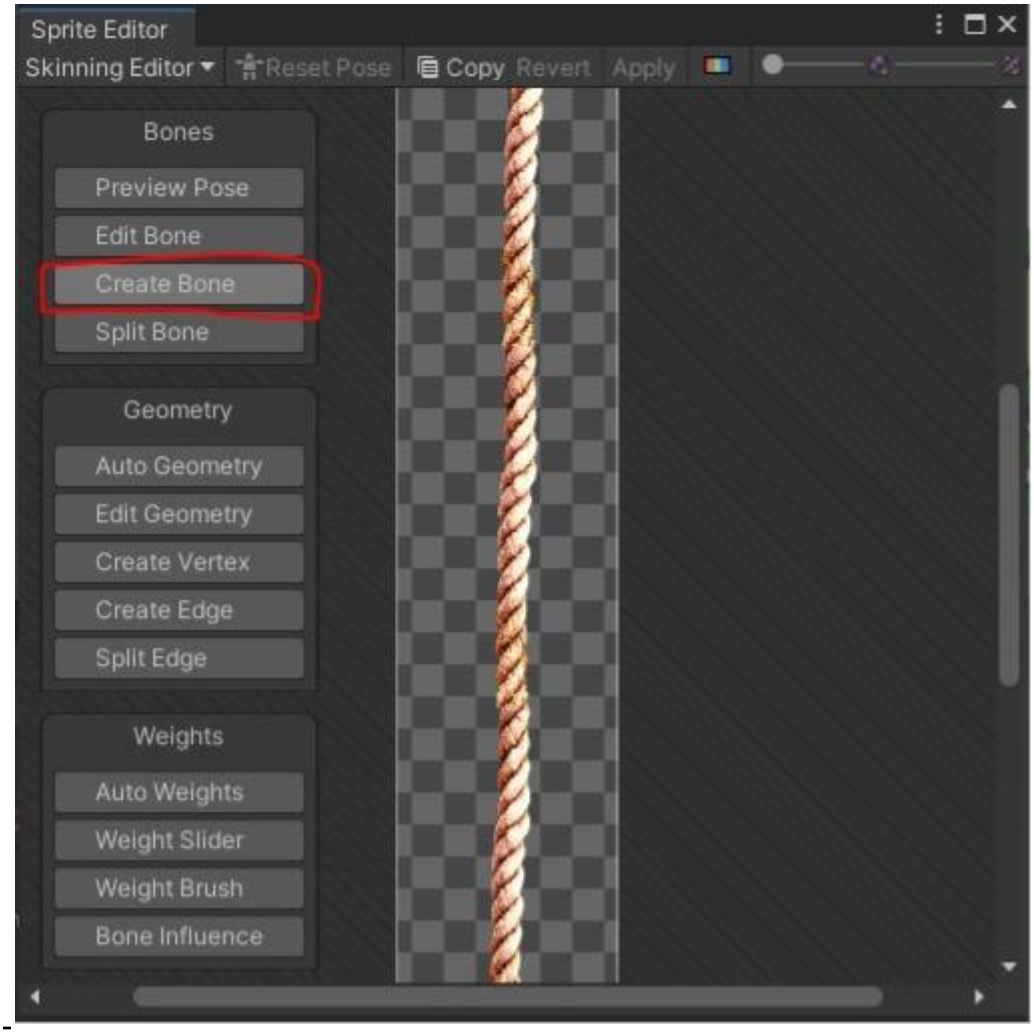


6. Apply the basic bone system for creating the swing effect by clicking on “Sprite Editor”, then on the top left change the “Sprite Editor” to the “Skinning Editor”

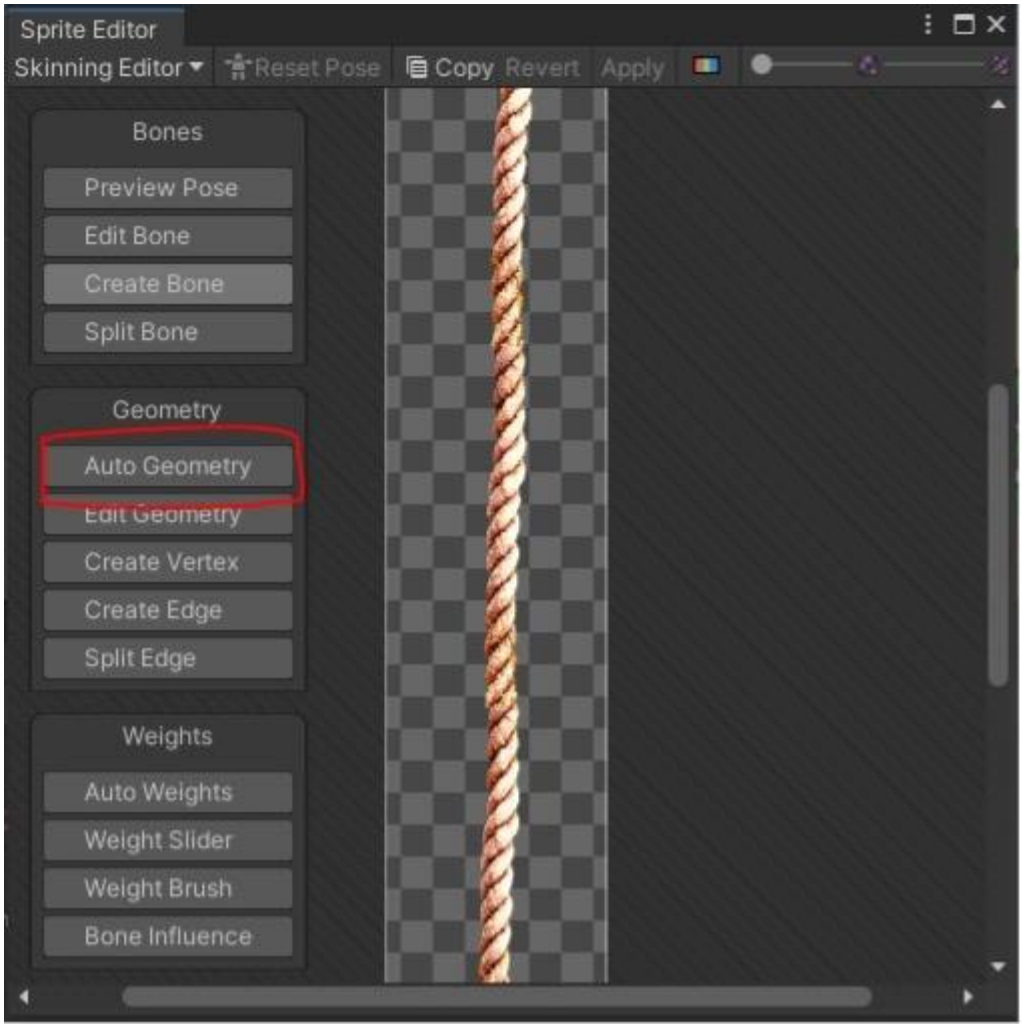




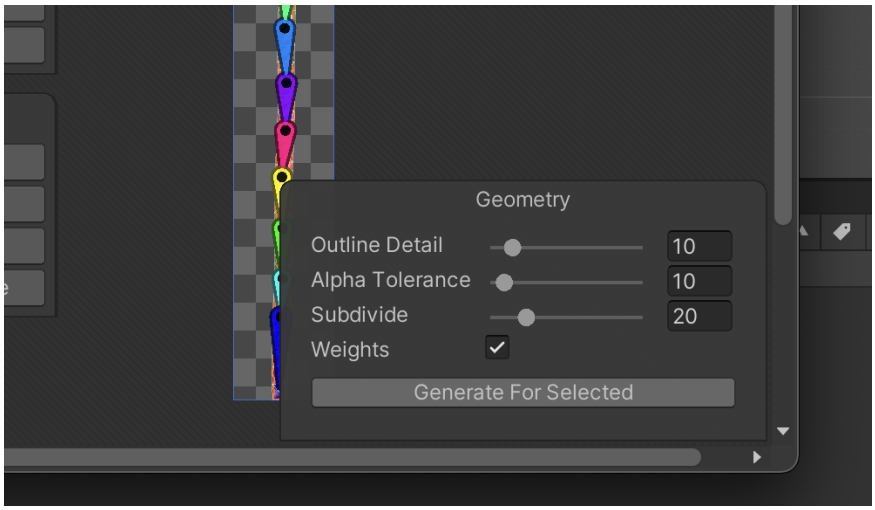
7. then in the bones section click on create bone



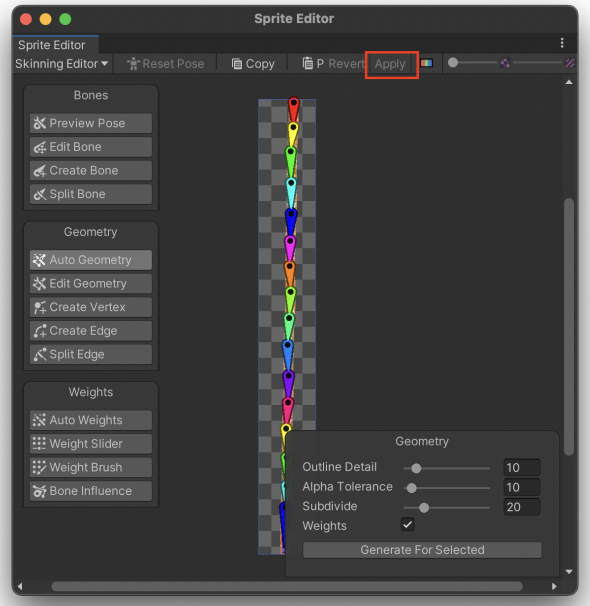
8. Now create multiple bones such that the whole sprite is covered from top to bottom then click on “Auto Geometry” and click on “Generate for Selected”.



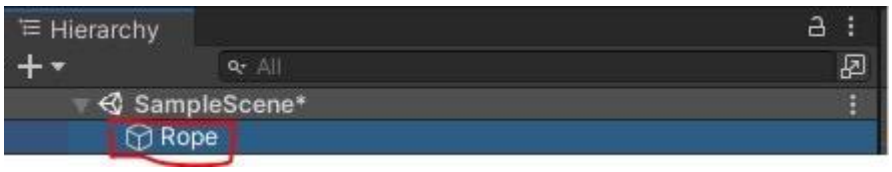
9. and click on “Generate for Selected” .



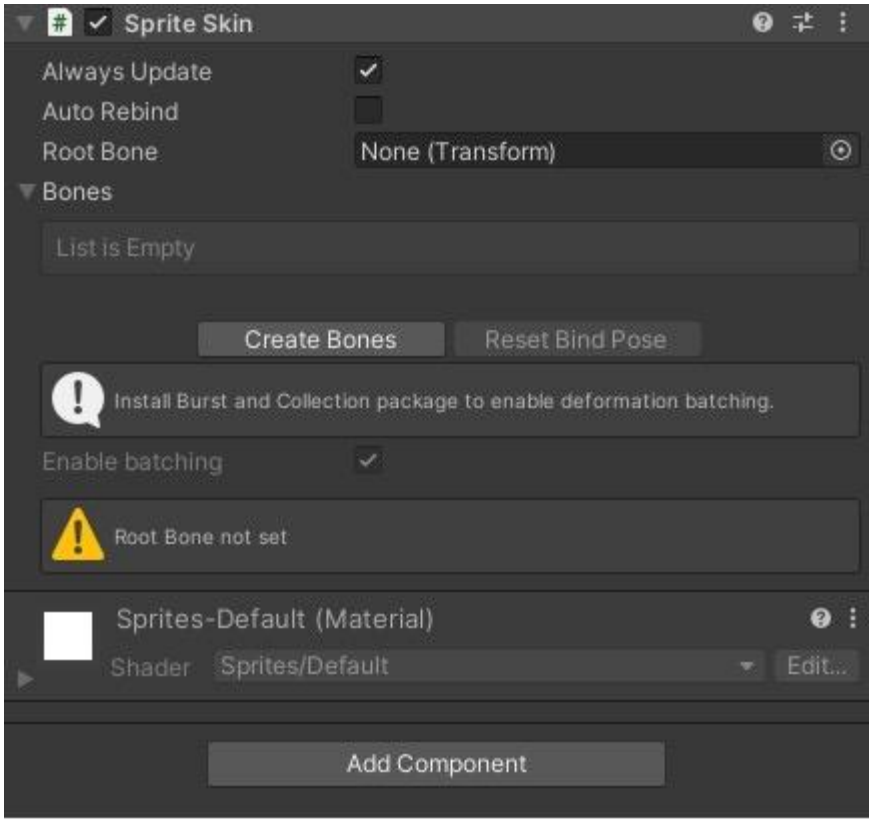
10. Now click on apply the changes.



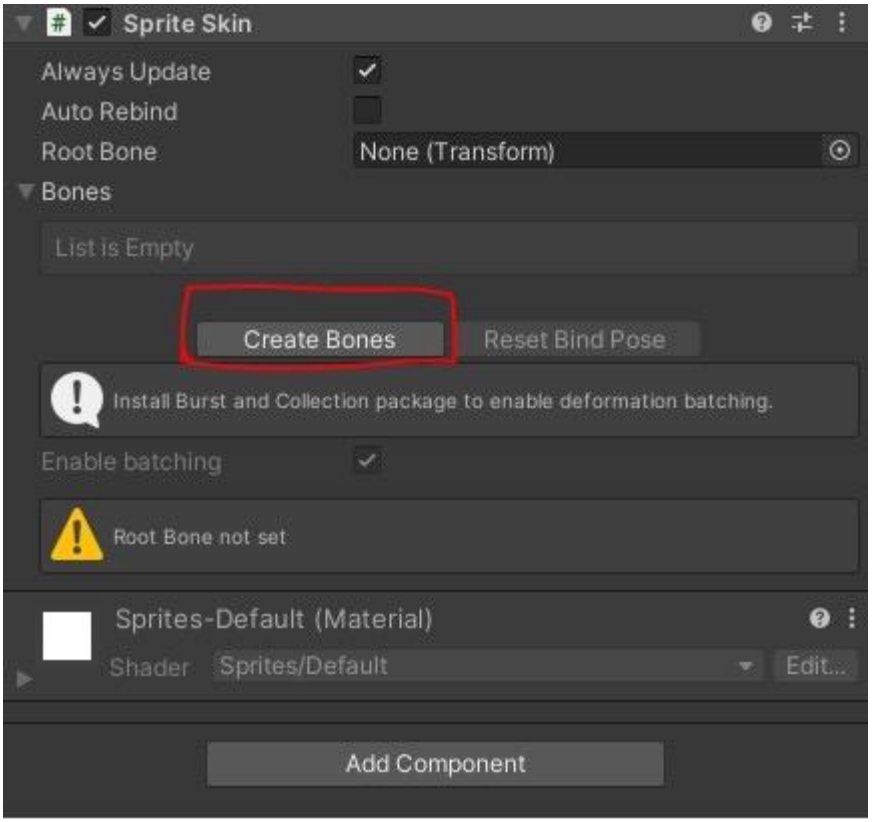
11. Now, drag and drop the created sprite into the scene so that it can be converted into a scene object.

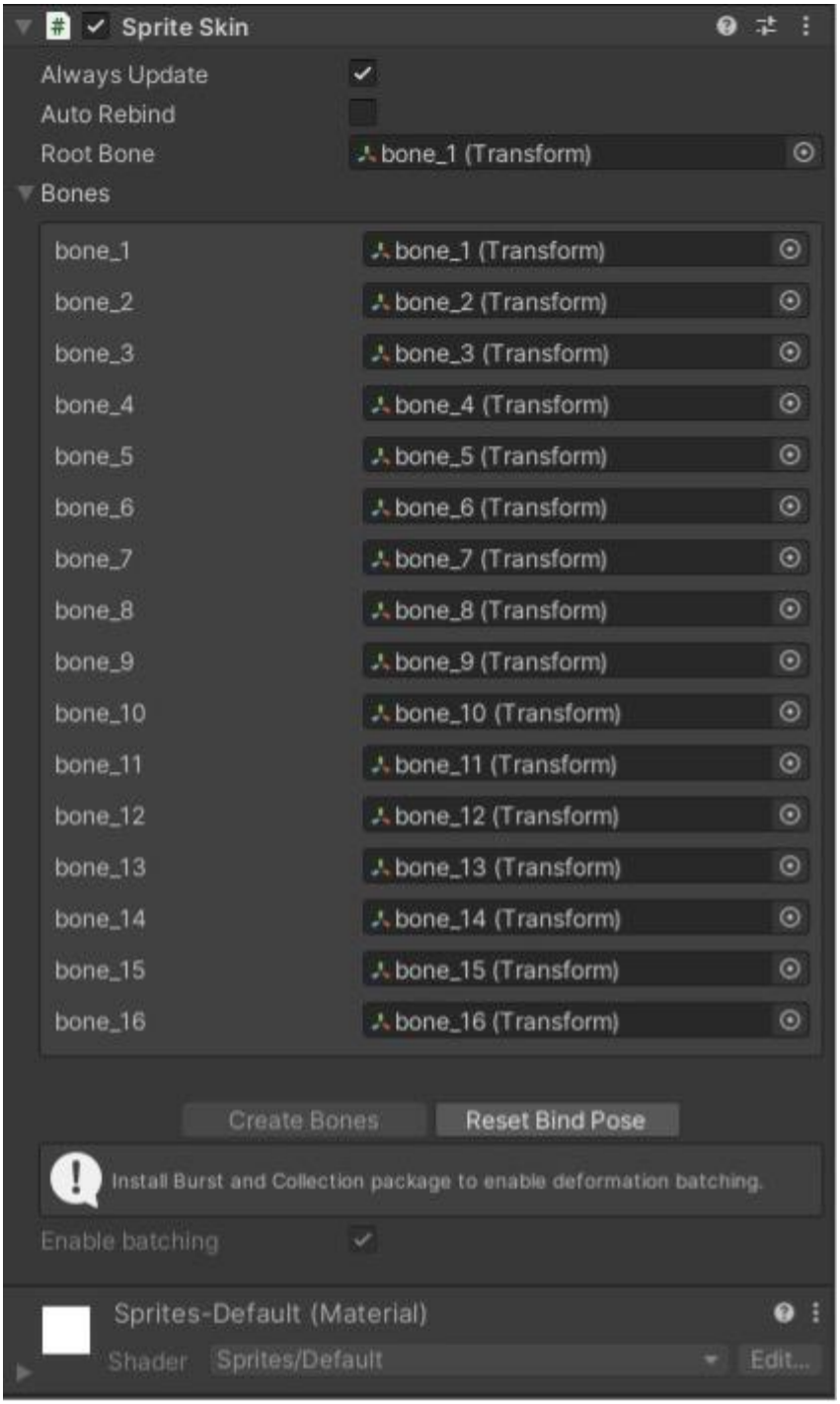


12. Now, select the object and click on Add component and search for “Sprite Skin” Script which is a built-in function from Unity to apply the created bones on the object.

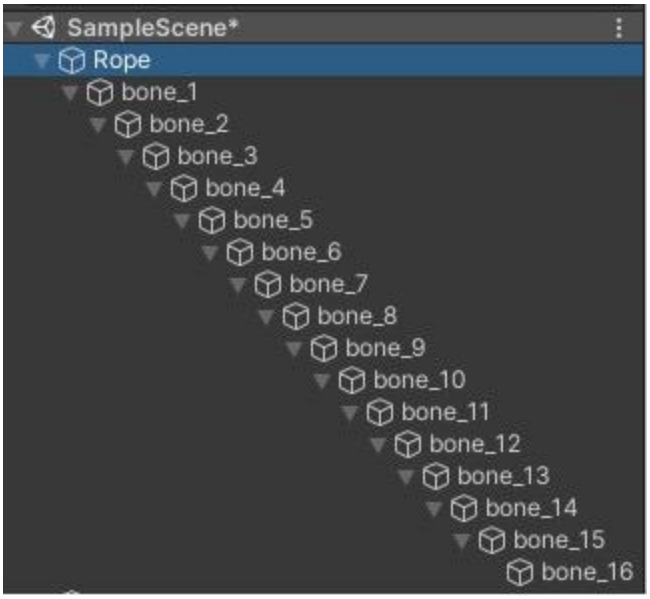


13. Now after attaching the “Sprite Skin '' Script, under that script click on create bones so that Unity can apply the bone system we created by using Unity`s “Sprite Editor”.

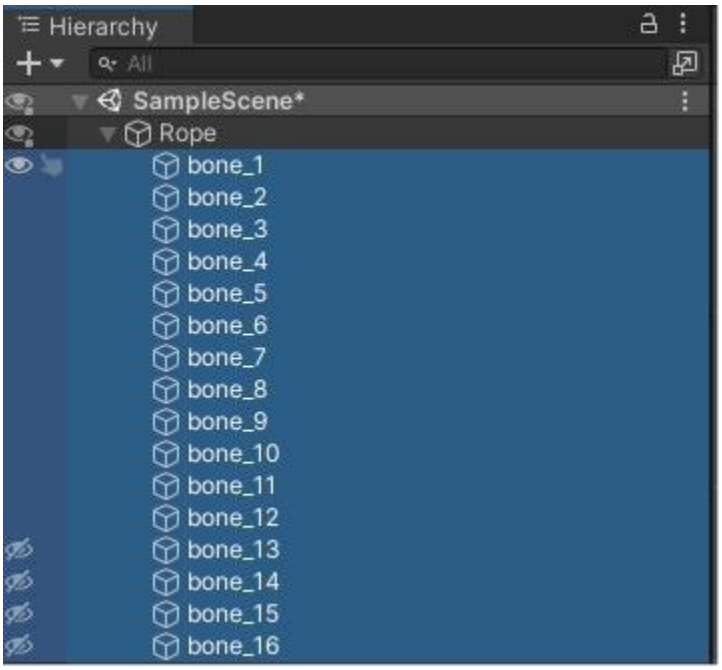

Sprite Skin
Always Update
Auto Rebind
Root Bone
None (Transform)
Bones
List is Empty
Create Bones
Reset Bind Pose
Install Burst and Collection package to enable deformation batching.
Enable batching
Root Bone not set
Sprites-Default (Material)
Shader
Sprites/Default
Edit...
Add Component



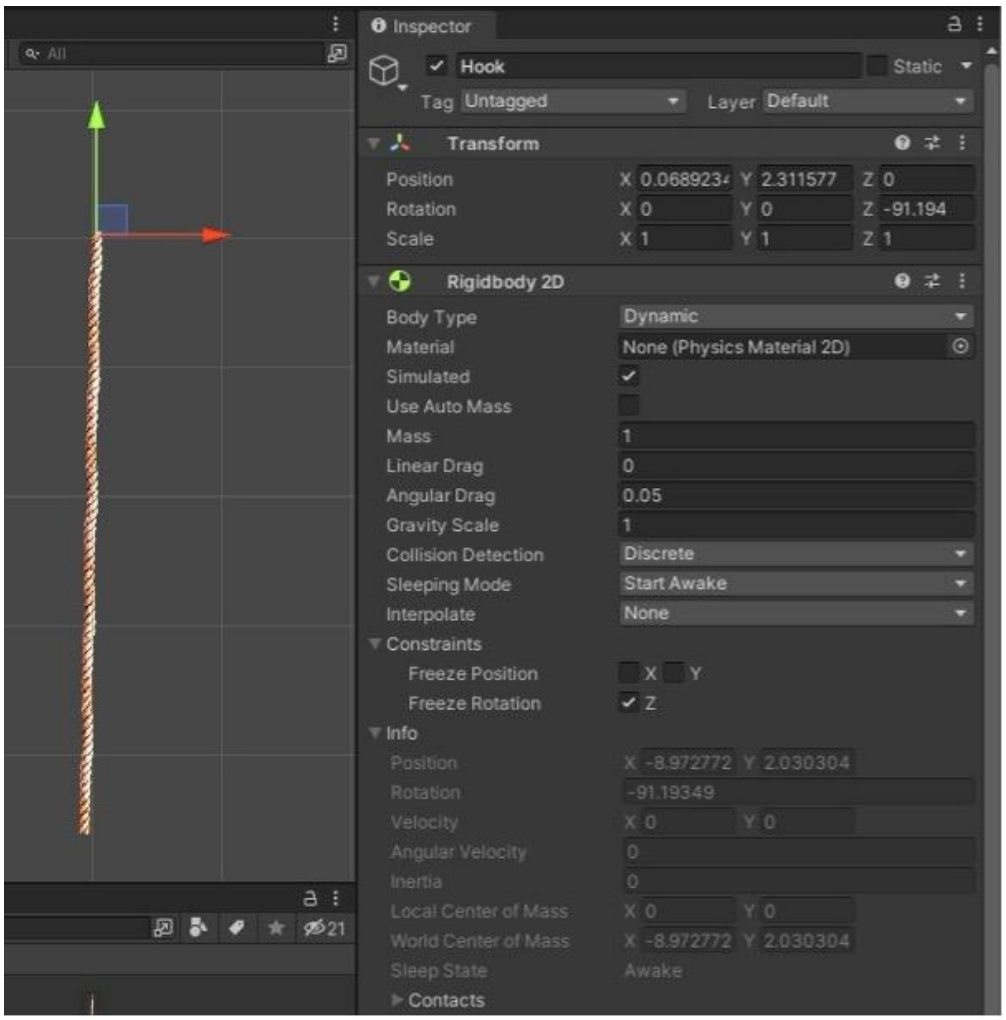
14. Now you will see that many child objects will be created under the parent object of the rope.



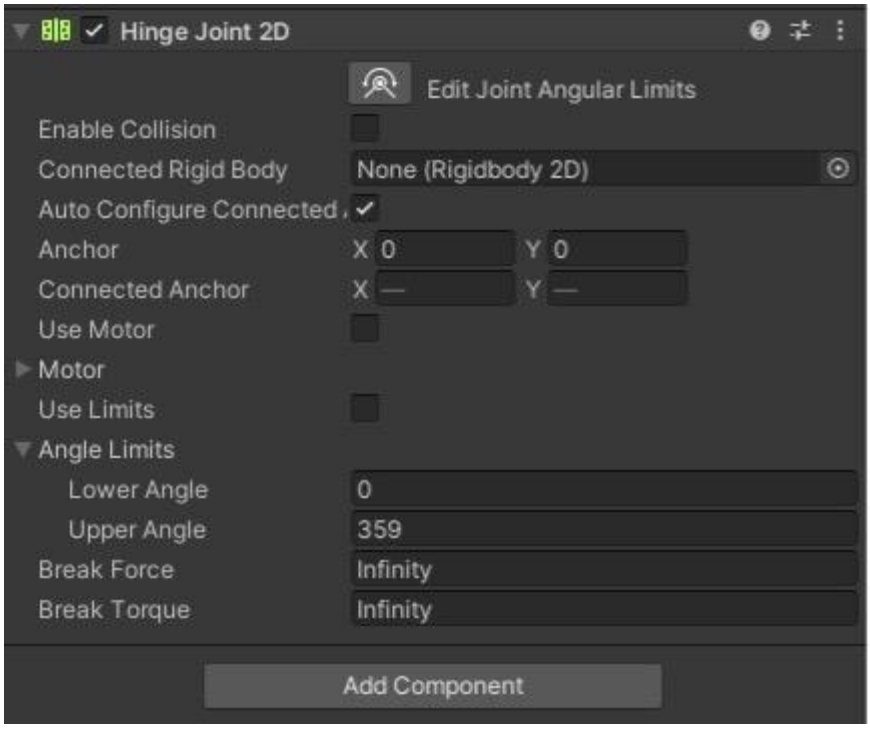
15. Now select and drag all the child objects into the Main parent object so that everything will be under one Parent object “Rope”.



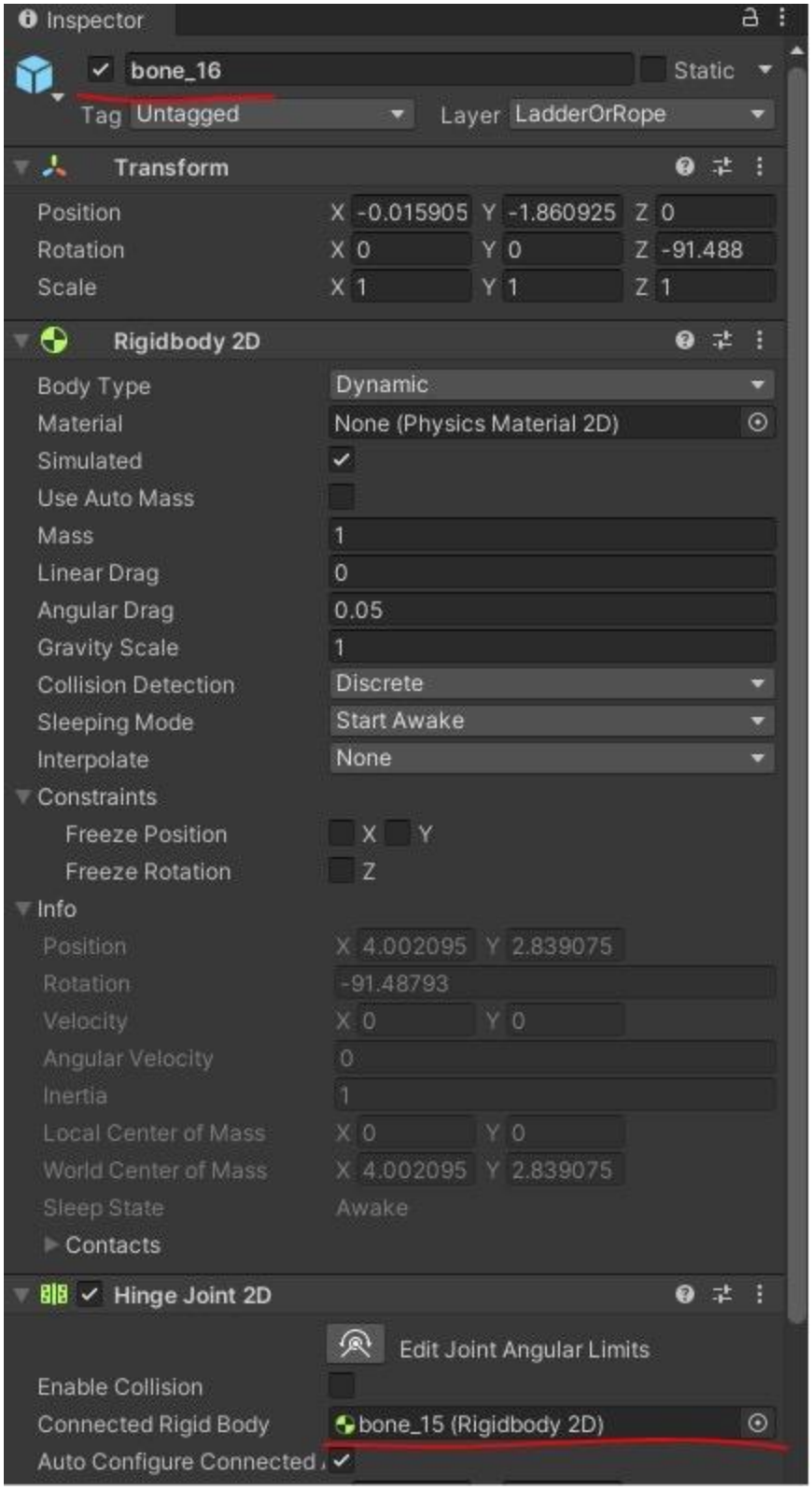
16. Now, create an empty child object under the Parent Object and rename it “Hook” and position it on the same as “bone_1” position and apply “Rigidbody 2D” component and under “Rigidbody 2D”, check the “Freeze Rotation” Constraints of Z-axis , because it will work as a hook for swinging the Rope left and right.



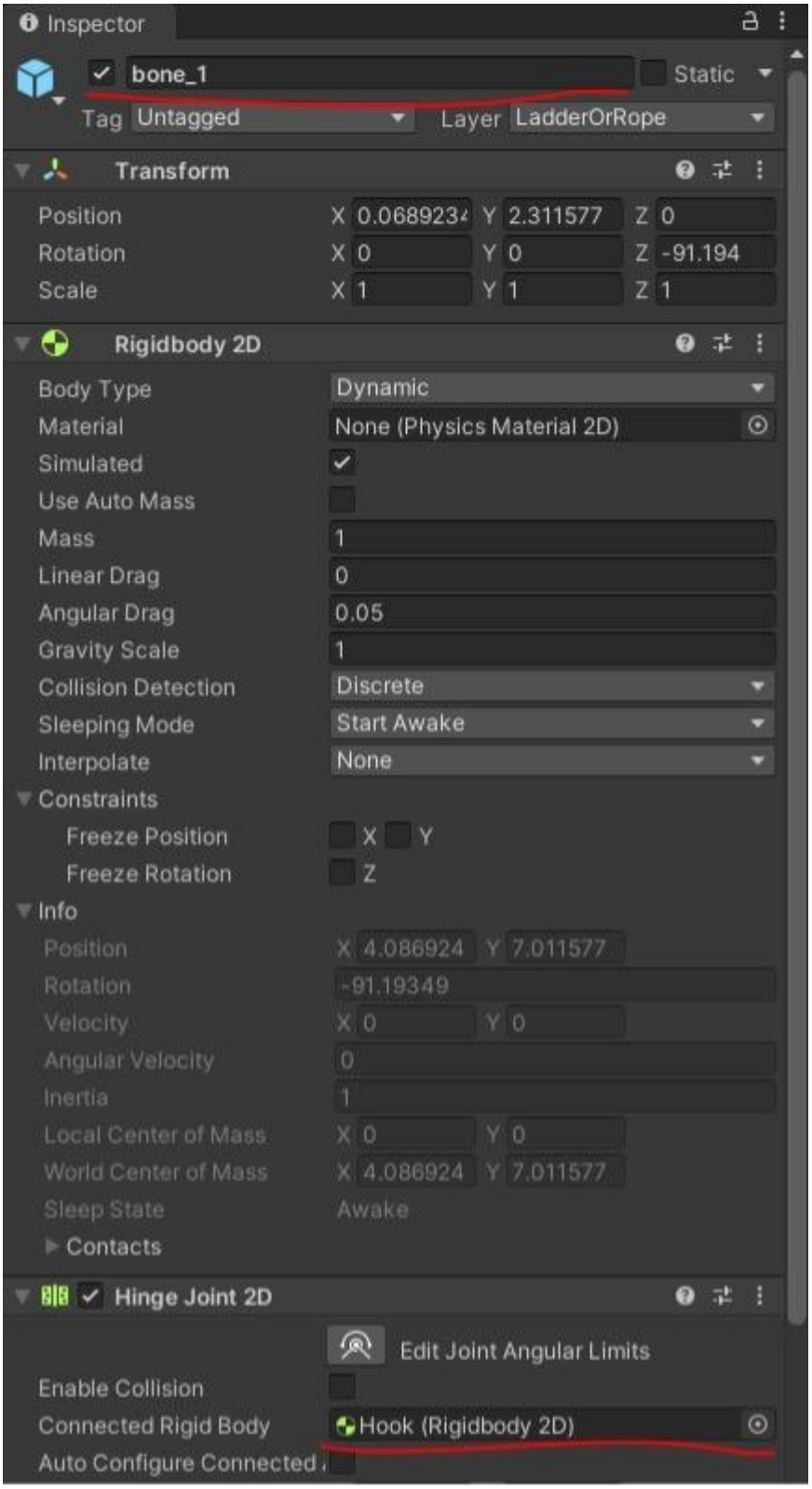
17. Now select all the child objects of the parent and add “Hinge Joint 2D” component.



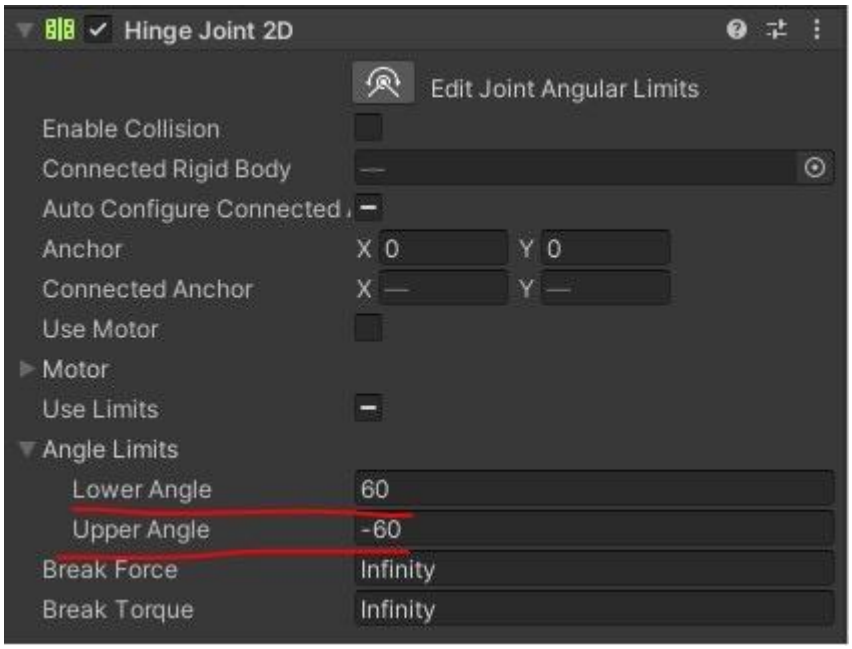
18. Now since every bone game object under the “Rope” has a hinge joint 2d component under , we will drag the preceding bone bone object to the hinge joint 2d of the next bone. For example bone_16 will have bone_15 as the “connected rigid body” under the Hinge joint 2d component.


Inspector
bone_16
Static
Tag Untagged
Layer LadderOrRope
Transform
Position X -0.015905 Y -1.860925 Z 0
Rotation X 0 Y 0 Z -91.488
Scale X 1 Y 1 Z 1
Rigidbody 2D
Body Type Dynamic
Material None (Physics Material 2D)
Simulated
Use Auto Mass
Mass 1
Linear Drag 0
Angular Drag 0.05
Gravity Scale 1
Collision Detection Discrete
Sleeping Mode Start Awake
Interpolate None
Constraints
Freeze Position X Y
Freeze Rotation Z
Info
Position X 4.002095 Y 2.839075
Rotation -91.48793
Velocity X 0 Y 0
Angular Velocity 0
Inertia 1
Local Center of Mass X 0 Y 0
World Center of Mass X 4.002095 Y 2.839075
Sleep State Awake
Contacts
Hinge Joint 2D
Edit Joint Angular Limits
Enable Collision
Connected Rigid Body bone_15 (Rigidbody 2D)
Auto Configure Connected

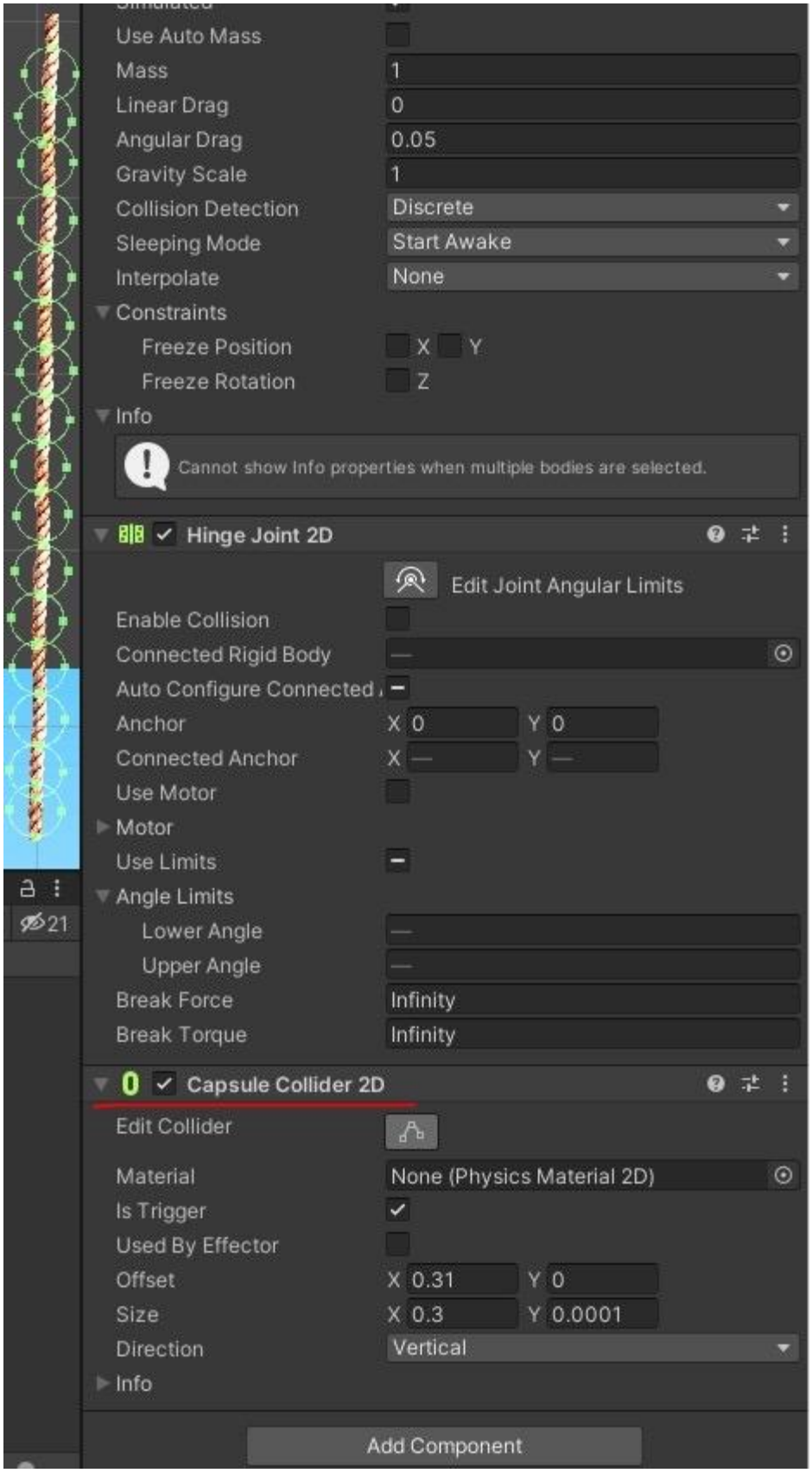
19. Similarly the “Hook” gameobject created earlier will be connected to the bone_1’s , “Hinge Joint 2D”. The person has to drag the “Hook” gameobject to the “Connected rigid body”.


Inspector
bone_1
Static
Tag Untagged
Layer LadderOrRope
Transform
Position X 0.0689234 Y 2.311577 Z 0
Rotation X 0 Y 0 Z -91.194
Scale X 1 Y 1 Z 1
Rigidbody 2D
Body Type Dynamic
Material None (Physics Material 2D)
Simulated
Use Auto Mass
Mass 1
Linear Drag 0
Angular Drag 0.05
Gravity Scale 1
Collision Detection Discrete
Sleeping Mode Start Awake
Interpolate None
Constraints
Freeze Position X Y
Freeze Rotation Z
Info
Position X 4.086924 Y 7.011577
Rotation -91.19349
Velocity X 0 Y 0
Angular Velocity 0
Inertia 1
Local Center of Mass X 0 Y 0
World Center of Mass X 4.086924 Y 7.011577
Sleep State Awake
Contacts
Hinge Joint 2D
Edit Joint Angular Limits
Enable Collision
Connected Rigid Body Hook (Rigidbody 2D)
Auto Configure Connected

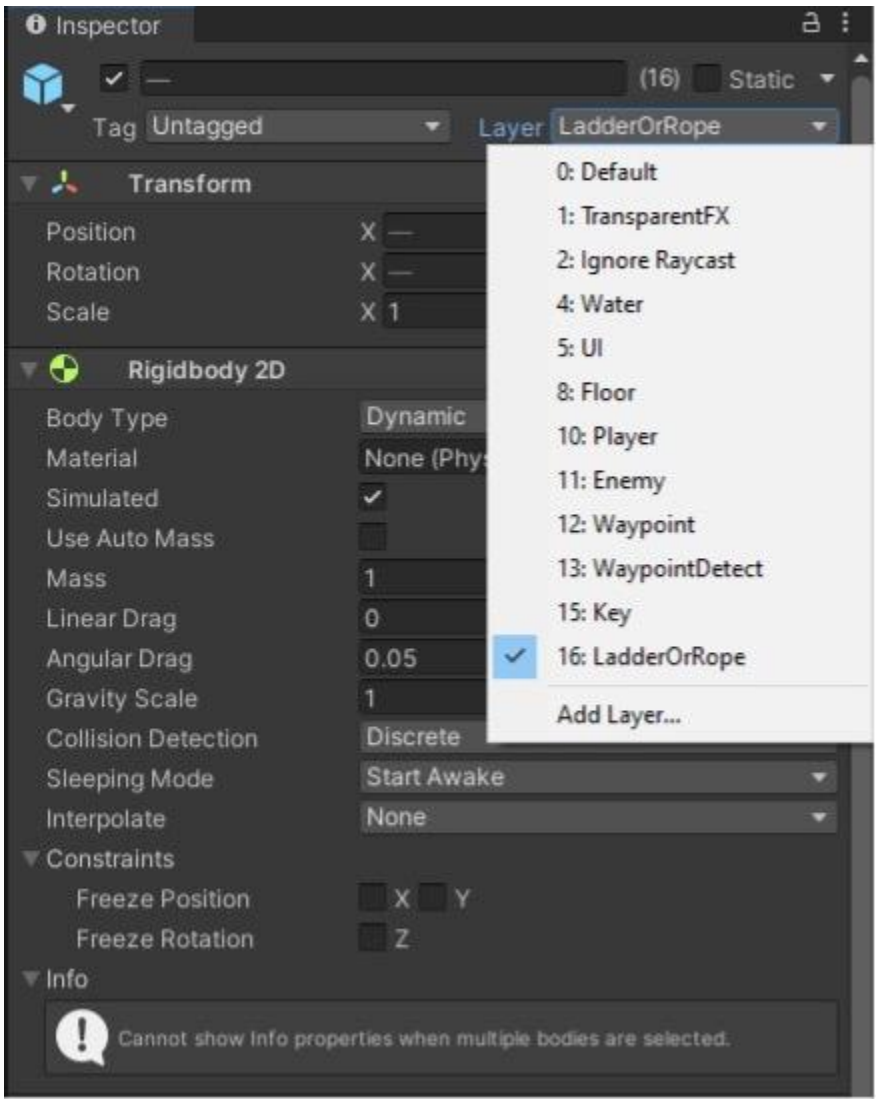
20. Now, select all the bone objects and set their Angle limits. Set lower angle to “60” and upper angle to “-60” except the last bone object. Set its lower angle to “30” and upper angle to “-30”.



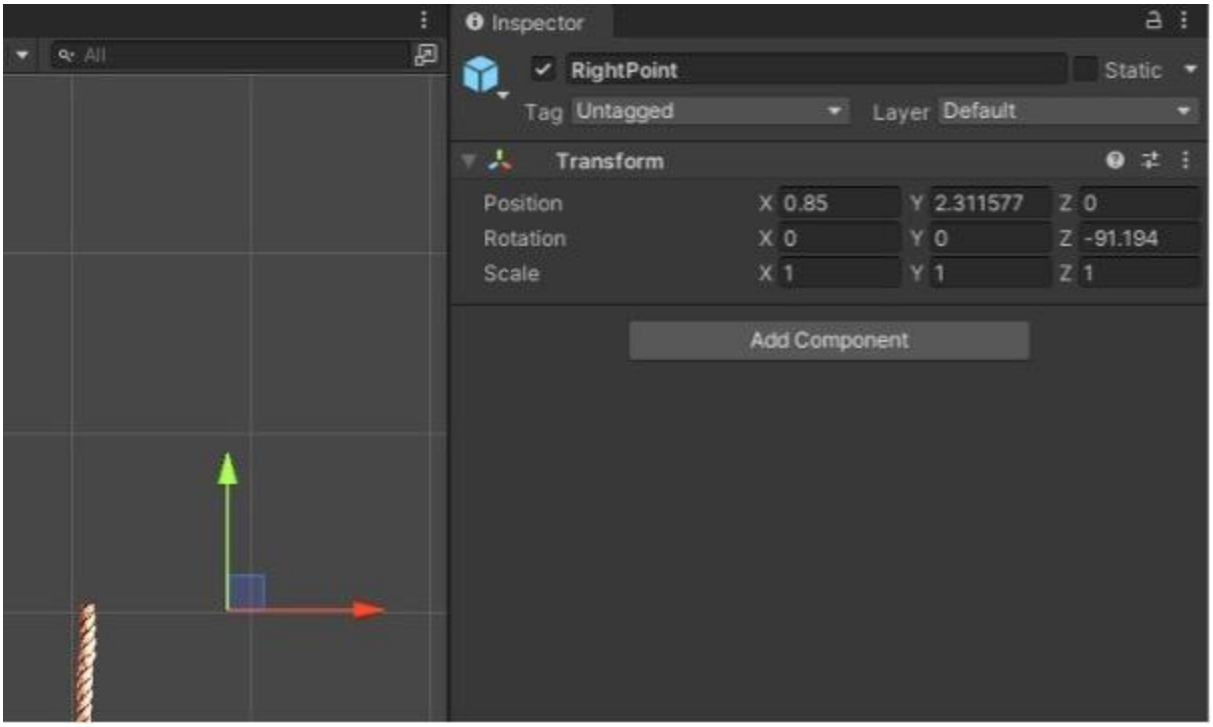
21. keeping all the bones objects selected ,apply “Capsule Collider 2D” on them and check “Is Trigger” so that the player can detect the collision to perform a specific function.

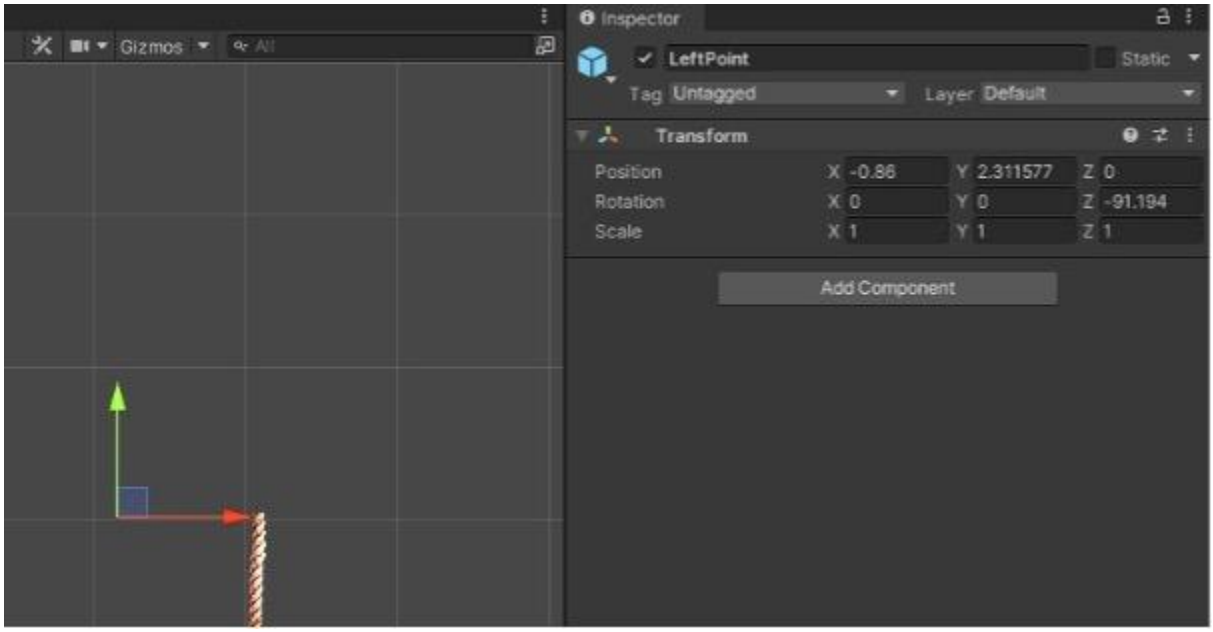

Use Auto Mass
Mass
1
Linear Drag
0
Angular Drag
0.05
Gravity Scale
1
Collision Detection
Discrete
Sleeping Mode
Start Awake
Interpolate
None
Constraints
Freeze Position
X
Y
Freeze Rotation
Z
Info
Cannot show Info properties when multiple bodies are selected.
Hinge Joint 2D
Edit Joint Angular Limits
Enable Collision
Connected Rigid Body
Auto Configure Connected
Anchor
X 0
Y 0
Connected Anchor
Use Motor
Motor
Use Limits
Angle Limits
Lower Angle
Upper Angle
Break Force
Infinity
Break Torque
Infinity
Capsule Collider 2D
Edit Collider
Material
None (Physics Material 2D)
Is Trigger
Used By Effector
Offset
X 0.31
Y 0
Size
X 0.3
Y 0.0001
Direction
Vertical
Info
Add Component

22. Now change the Layer of the object from “Default” to “LadderOrRope”.

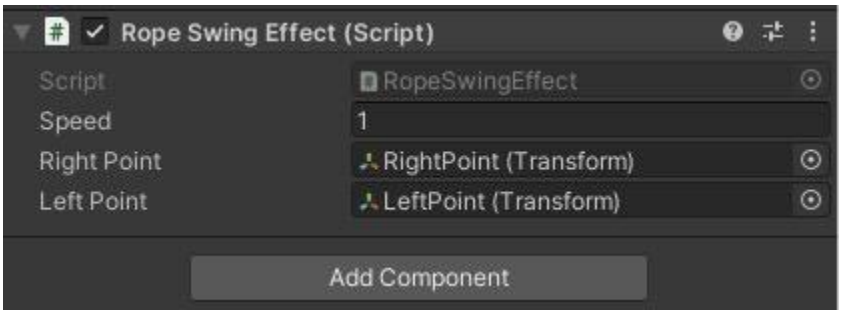


23. Now, create two child objects under Parent object and rename them to “RightPoint” and “LeftPoint” as we will use them to swing the rope left and right and move “RightPoint” right of the rope and move “LeftPoint” left of the rope.


Inspector
All
RightPoint
Static
Tag Untagged
Layer Default
Transform
Position X 0.85 Y 2.311577 Z 0
Rotation X 0 Y 0 Z -91.194
Scale X 1 Y 1 Z 1
Add Component


Gizmos
All
Inspector
LeftPoint
Static
Tag Untagged
Layer Default
Transform
Position X -0.86 Y 2.311577 Z 0
Rotation X 0 Y 0 Z -91.194
Scale X 1 Y 1 Z 1
Add Component

24. Now, select the “Hook” object and attach the “Rope Swing Effect” script to it. Set the swinging speed and drag and drop the “RightPoint” object and “LeftPoint” object to it.

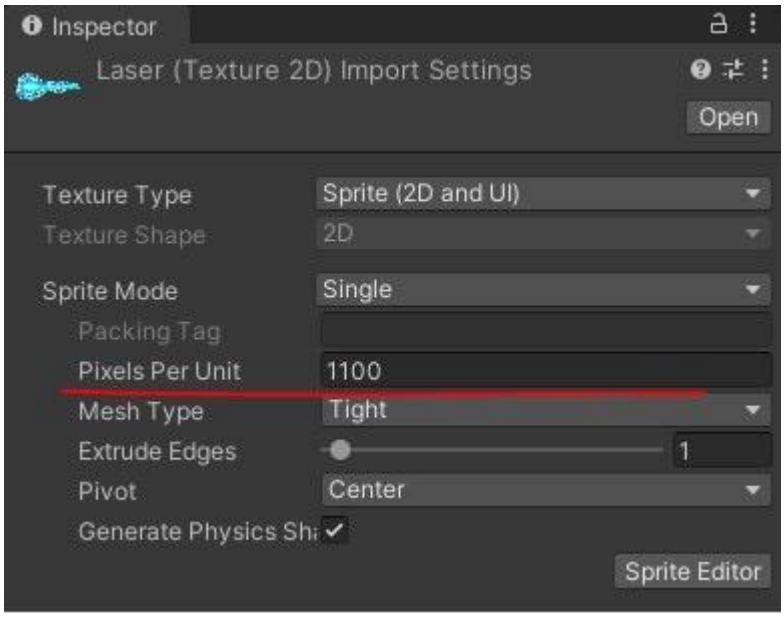


Creating Projectile System:

1. Download an image that suits the laser.

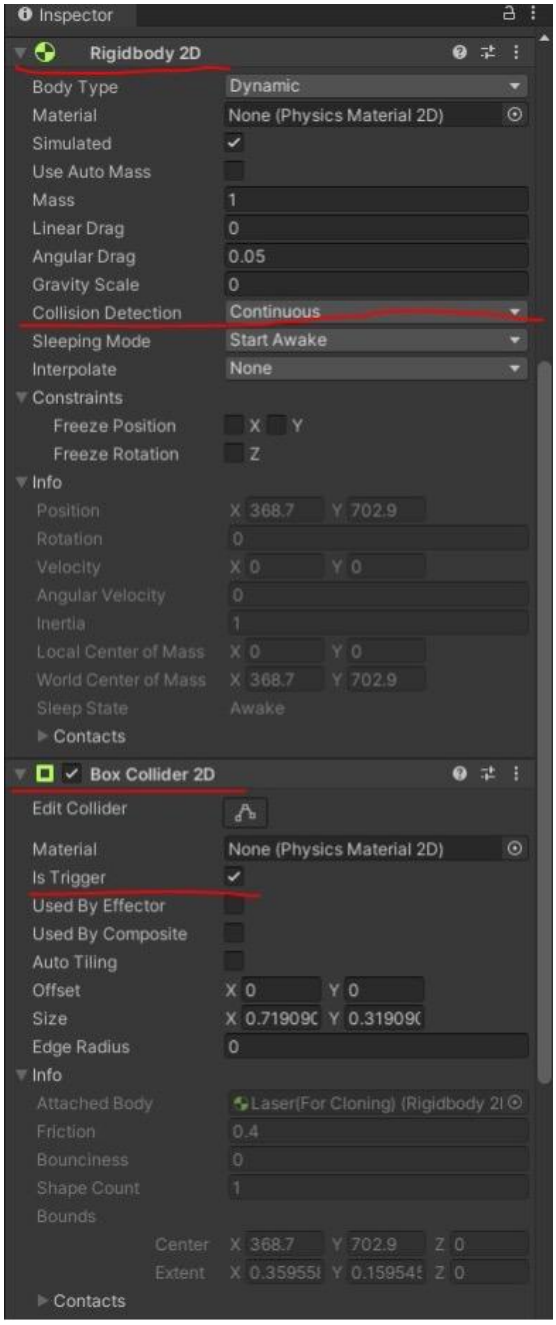
2. Drag and drop that image in the sprites folder of the project.

3. Rename the file name to “Laser”.
4. Adjust its size according to the scene by changing the “Pixels Per Unit” value.



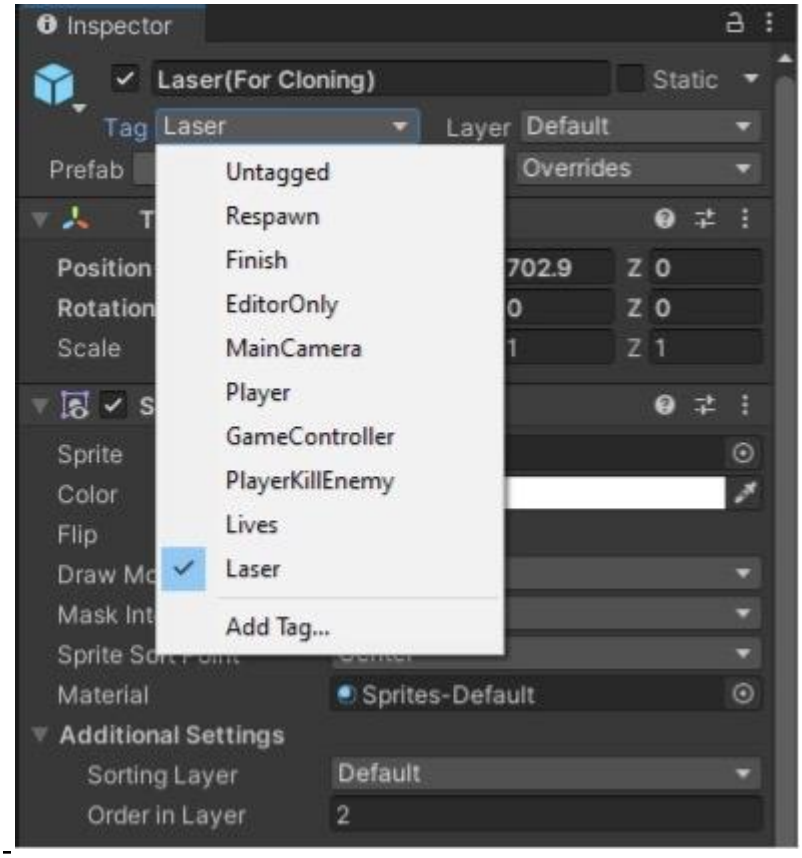
5. Now, drag and drop the created sprite into the scene so that it can be converted into a scene object.

6. Apply “Rigidbody 2D” component to the object and set it`s collision detection to “Continuous” and “Box Collider 2D” component to the object and adjust the collider size according to the sprite and check “Is Trigger” so that player can detect it as a trigger to perform a specific function.

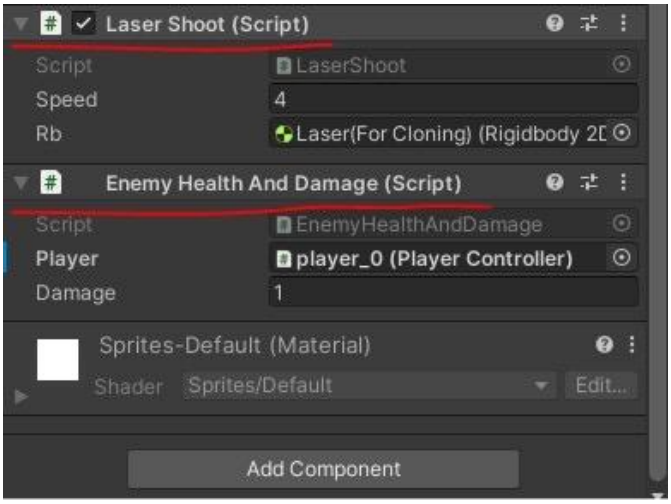


-

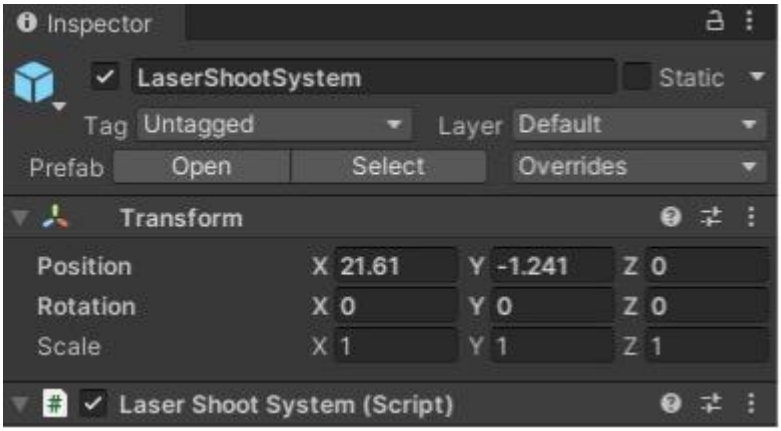
7. Now change the Tag of the object from “Untagged” to “Laser”.



8. Now, attach the “Laser Shoot” script and set the Speed and Rb (Rigidbody 2D) component to it and also attach the “Enemy Health And Damage” script and drag “player_0” object to player and set the Damage value.

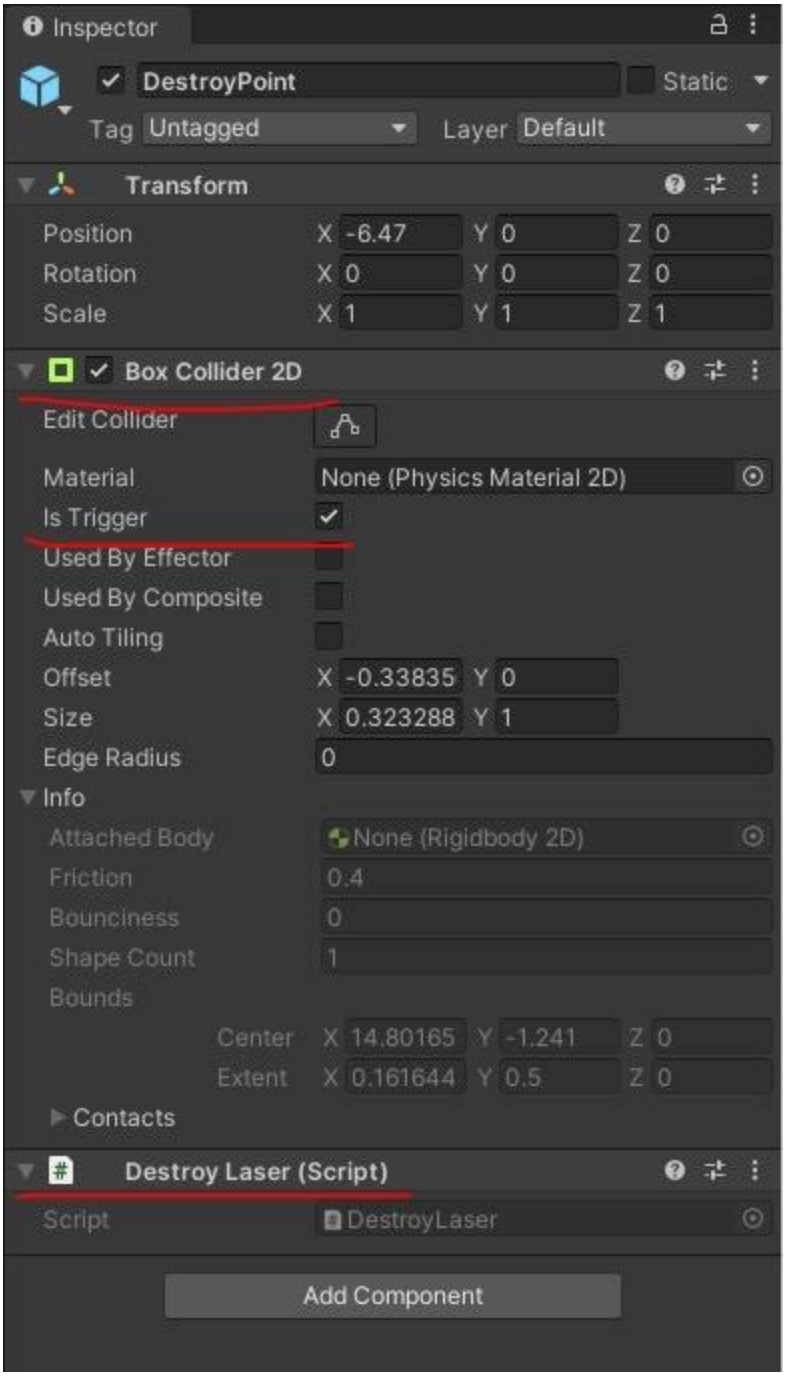


9. Now, create an empty object in the scene and rename it to "LaserShootSystem" and attach "Laser Shoot System" script to it.



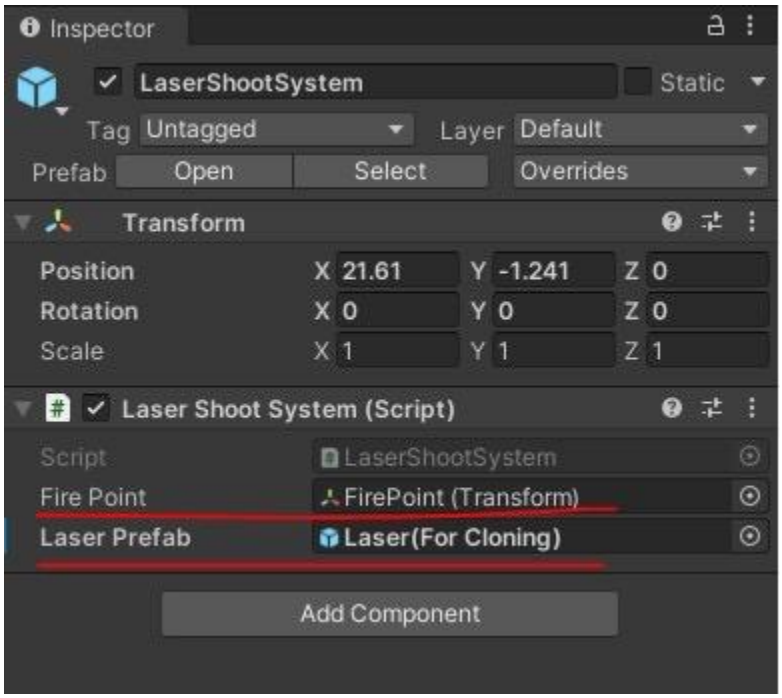
10. Now, create two empty objects under the Parent object "LaserShootSystem" and rename them "FirePoint" and "DestroyPoint". "FirePoint" will be the point from where the laser will be shot, so place it in the scene for your choice of projectile origination point. The laser and"DestroyPoint" will act as the point where the projectile ends.

11. Now, apply “Box Collider 2D” to the “DestroyPoint” object and adjust the collider size and check “Is Trigger” so that player can detect that from the thing that he is colliding is using as trigger to perform a specific function and also attach the “Destroy Laser” script to it.



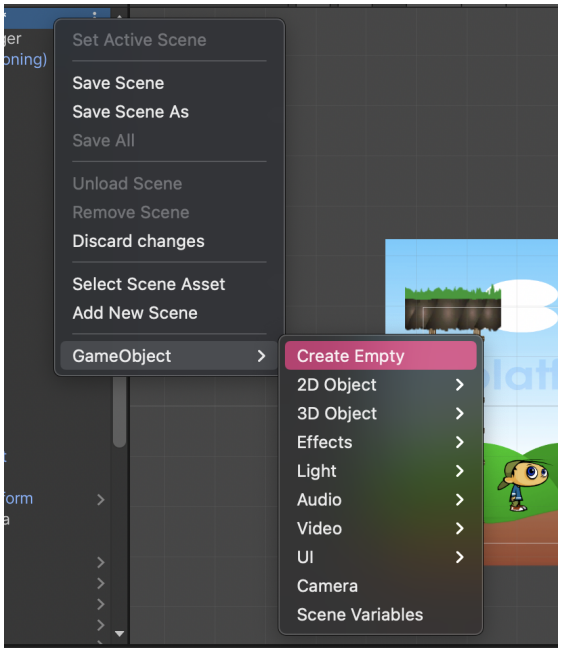
12. As we have attached the “Laser Shoot System” script to “LaserShootSystem” object, now drag and drop the “FirePoint” object to

Fire Point and “Laser” object that we have created at start to “Laser Prefab”.

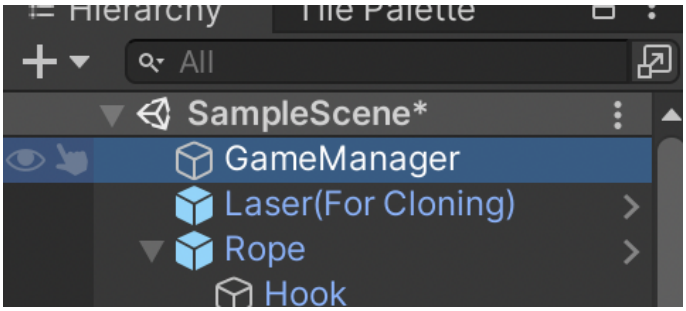


Creating GameManager For Controlling Lives System And Checkpoint Reset System:

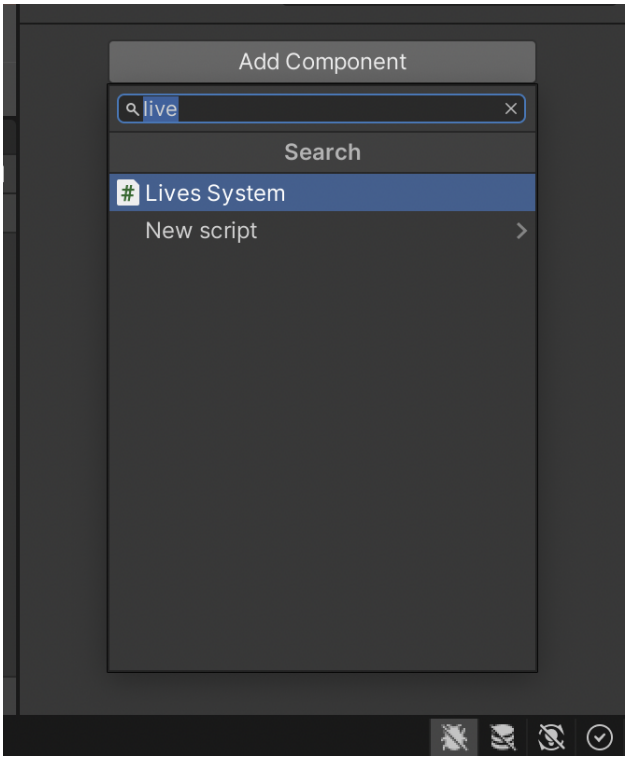
1. Create an empty GameObject in the scene.



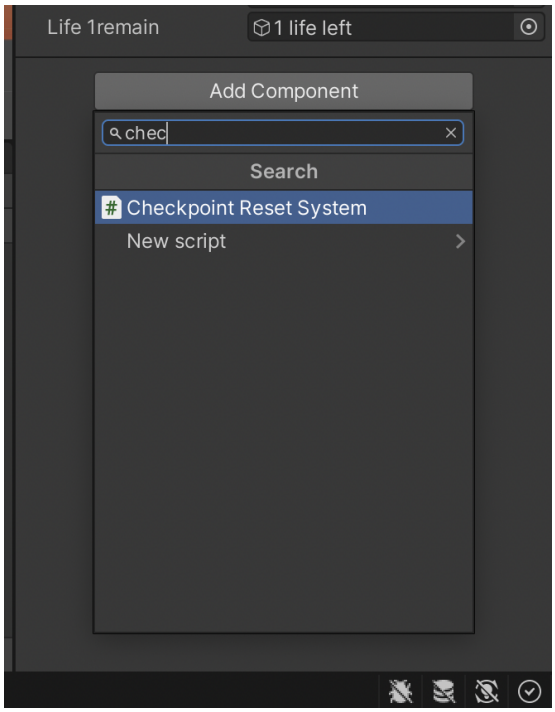
2. Select the created GameObject and rename it to GameManager.



3. Now, select the GameManager Object and click on Add Component and search for the Lives System Script and attach it to the GameManager Object.

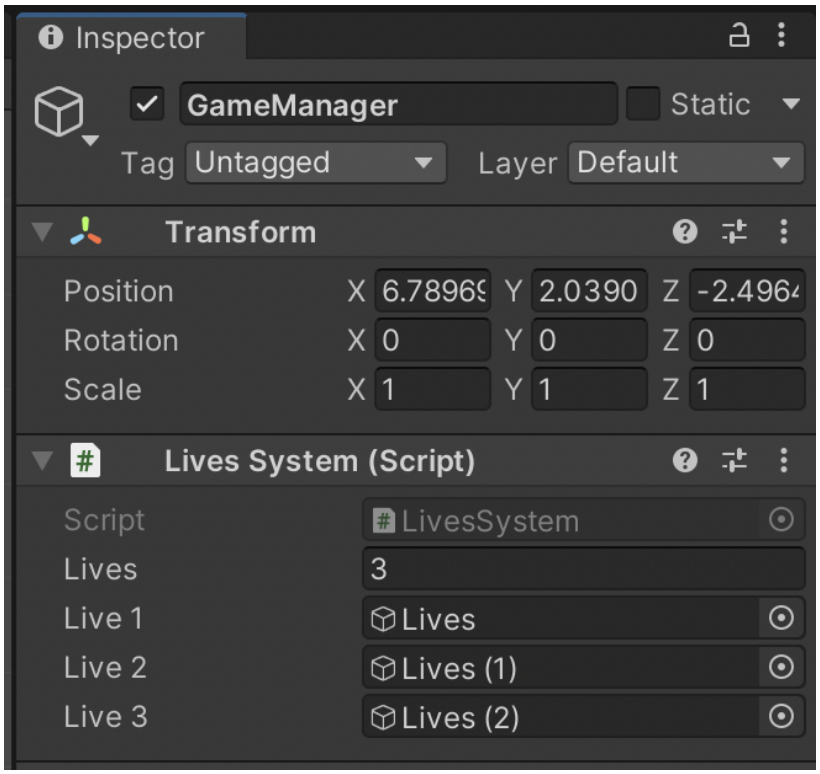


4. Now again select the GameManager Object and click on Add Component and search for the Checkpoint Reset System and attach it to the GameManager Object.



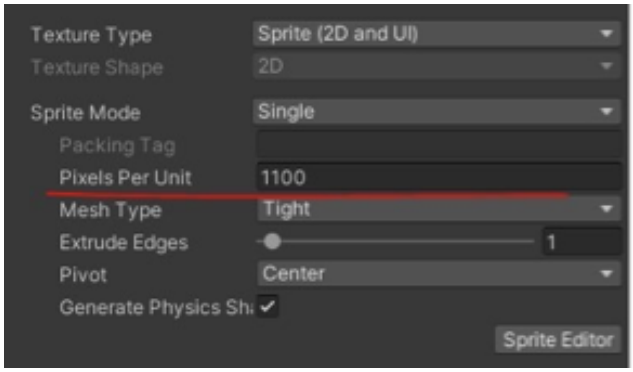
Creating Lives System:

1. As we have already attached our Lives System script to the GameManager Object, we will see that the “Lives” component contains “3” ,which means our Lives System (Whole Level) has 3 Lives in total.

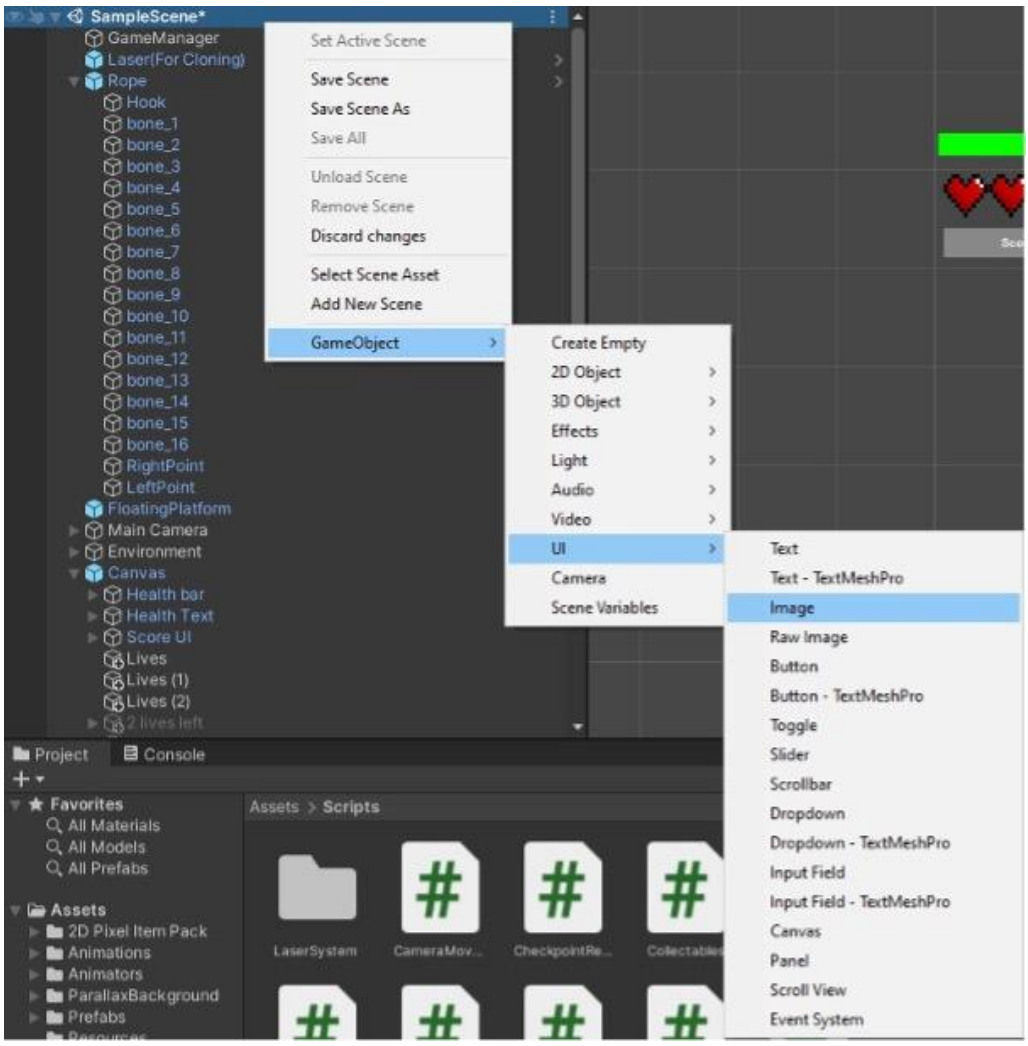


2. Download an image that suits for the Heart (Lives).
3. Drag and drop that image in the sprites folder of the project for convenience.

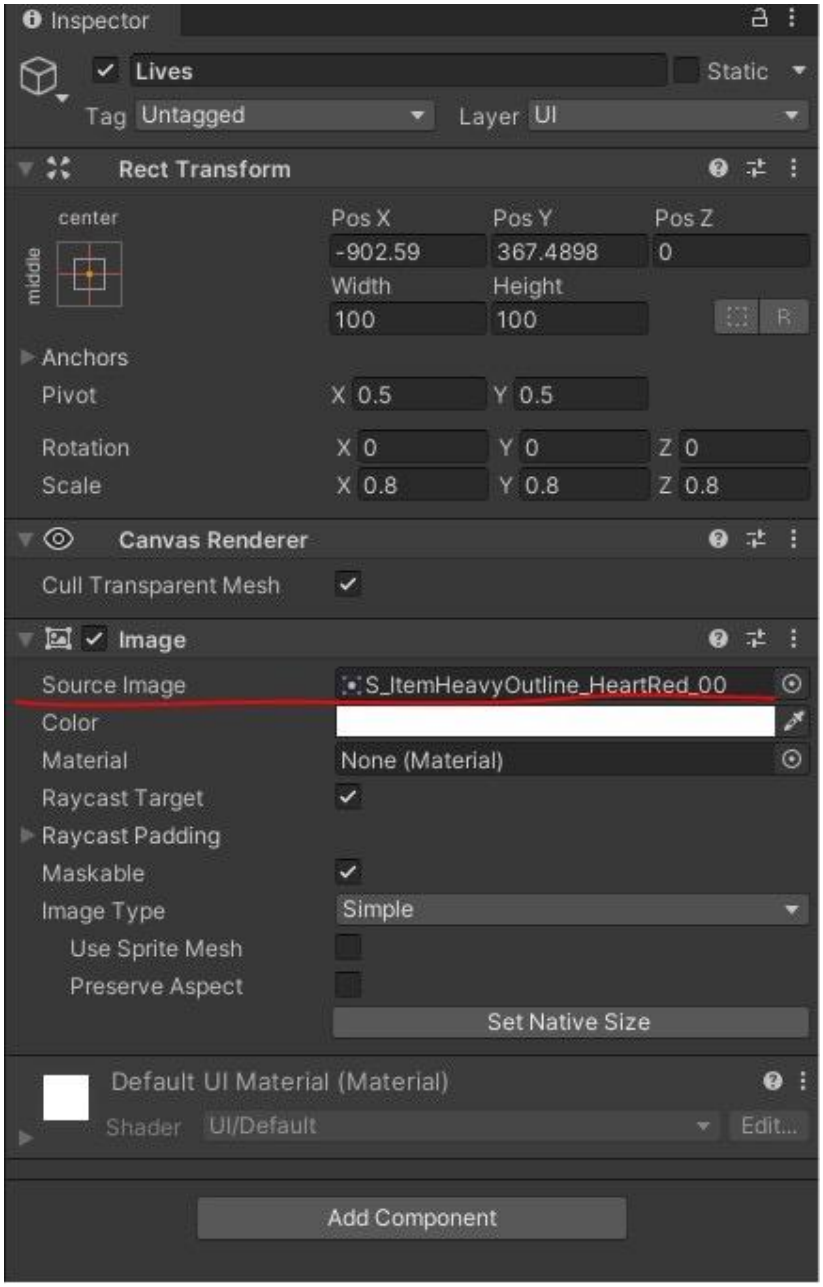
4. Rename the file name to “Heart”.
5. Adjust its size according to the scene by changing the “Pixels Per Unit” value.



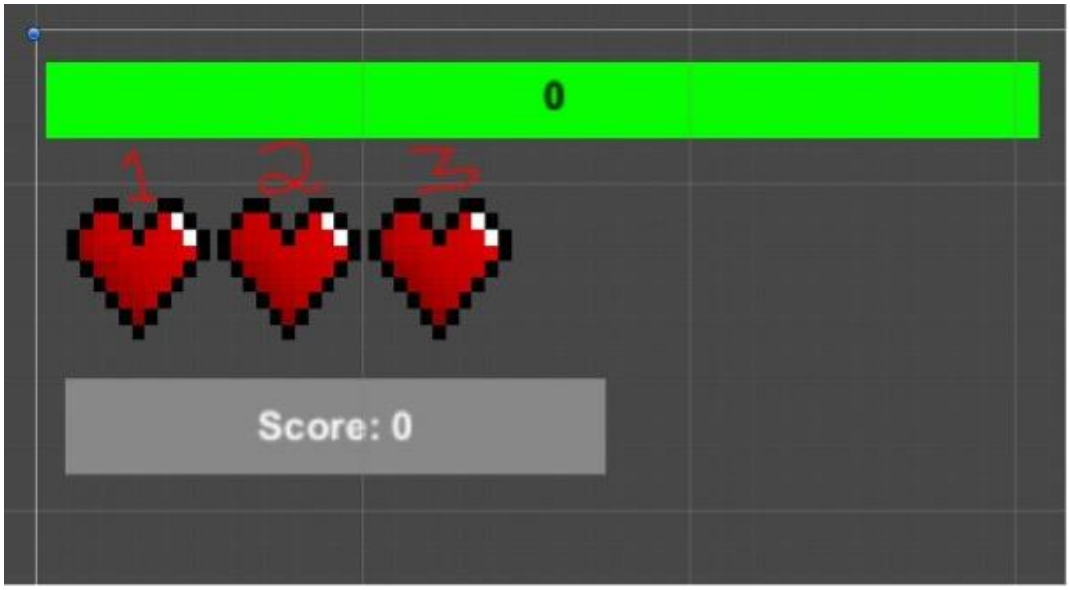
6. Now, in the scene right click in the Hierarchy and select GameObject -> UI Image. This will create an Image UI under the Canvas of the scene.



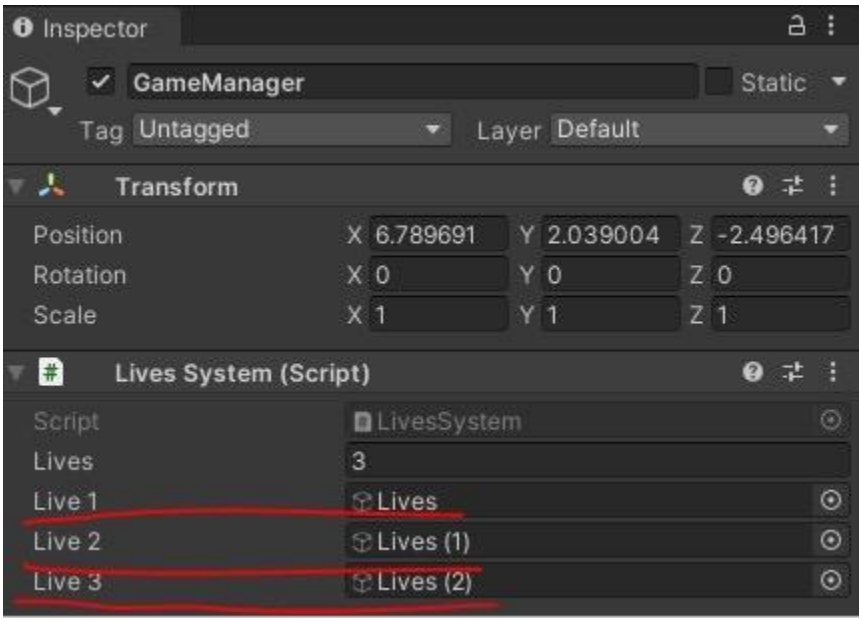
7. Now, select the created Image UI and rename it to “Lives” and under the Image Component, select the Source Image function and drag and drop the Heart image in it. You can find the heart sprite in the “heats folder” under sprites.



8. As we have “3” Lives in total, so we will duplicate the created Image UI “Lives” and will put them at some distance from one another.



9. Now, after creating the Image UI for Lives, we will drag n drop these 3 objects (Image UI) under the Lives System script of the GameManager Object that we have already created at start.

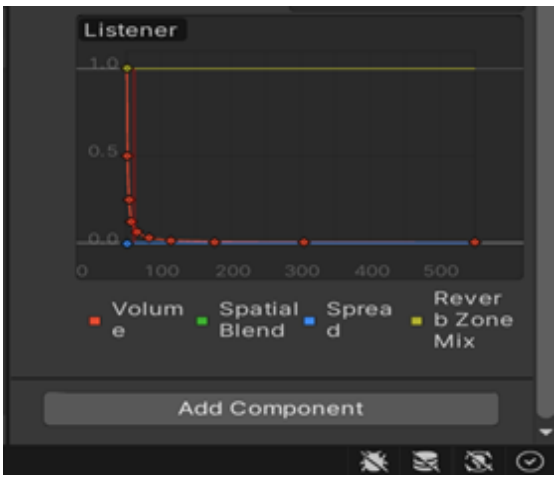


10. We have created these Image UI objects to show how many lives are left for the Player to complete the Level. As the player will die, the lives will keep reducing and the Image UI Objects will also Deactivate one by one to the lives left. The lives are lost from bottom to top , meaning that the live 3 sprite will be the first to deactivate , then 2 and then 1.

Game Audio

Audio Source

An Audio Clip is played back by the Audio Source in the scene. It is attached to a Game-Object for playing back sounds in a 2D environment. The clip can be played to an audio listener or through an audio mixer. The audio source can play any type of Audio Clip and can be configured to play these as 2D.



Audio Clip

The Audio-Clip is the actual sound file that will be played back. The Audio Source is a controller for starting and stopping playback of that clip and modifying other audio properties. Audio Sources need to be attached as a component to a Game-Object. The unity user needs to be aware that the Game-object does not do anything on their own. The Audio Source needs to be added as a component to a Game-Object to make this object become an Audio feature.

There are three types of Audio's involved in the entire game for the 3 main scenes: -

1. Main Menu Audio
2. Game Screen Audio
3. Win Screen Audio

Resources

All the game audio clips required and played on loop on each of the scenes are stored under the 'Resources' folder.

The audio effects for the main character of the game is coded on the 'Sound Manager Script' under the components of the game-object 'SoundManager'.

How Game Audio is Implemented:

- The audio files are imported into the Unity 2D templateProject. These are now AudioClips.
- Under GameObject an empty component been created and named 'SoundManager'
- You can search for Audio Source and select this. An Audio Source will be attached to the GameObject in the inspector.
- As shown above all the AudioClip to the Audio Source.

Please find the names of each audio currently used in the entire game: PlayerDeath - Played when the player dies

- Jump - Played each time the player jumps
- Click - Plays every time the player collects a collectable
- enemykill - Plays when the player kills the enemy
- Win - When the player wins they will be directed to the win screen with this audio played
- Loose - The player will be directed to the audio screen with this audio being played.
- Theme - The main theme of the song changes with each screen. This audio is the main background music that is played throughout the song.

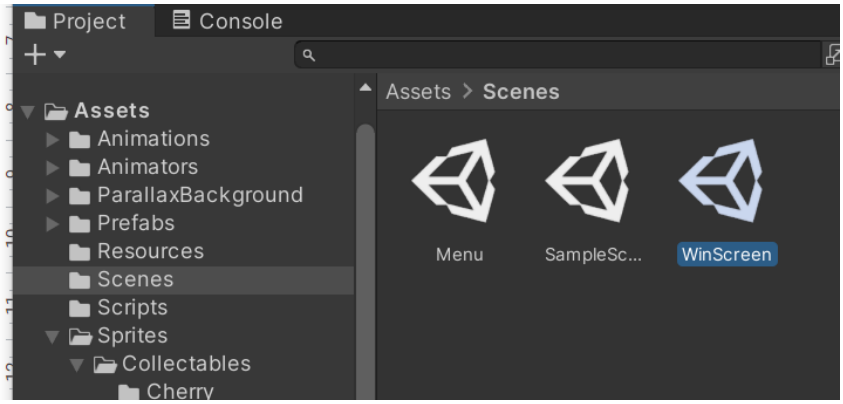
01. Adding a new audio

I. Please note: In order to create new background music, you will need sounds to be downloaded. For this game we have input sounds from the www.freesound.org website. You can either use the same website or any other website where you can download your matching music. You may also add any of your own music.

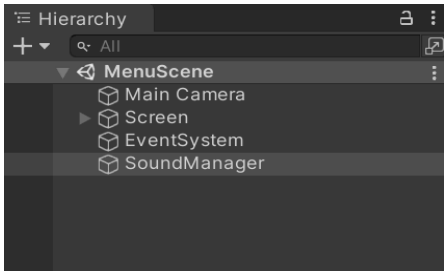
II. After downloading all the required sounds, rename each of the clips as it is easy to identify.

III. Once done, Drag and drop your audio into the **Resources** folder under 'Assets' on the project window.

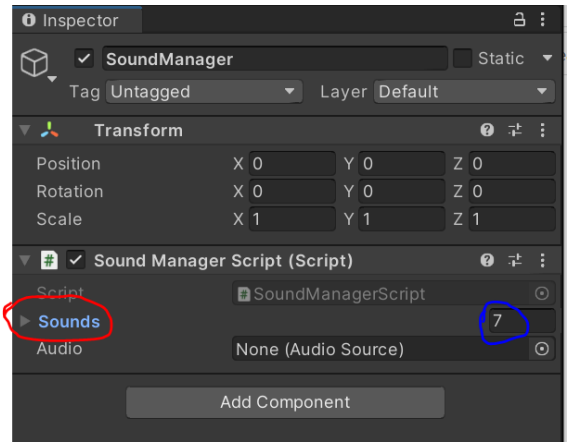
IV. Select the screen you wish to add sound to:



V. Click SoundManager under Hierarchy.

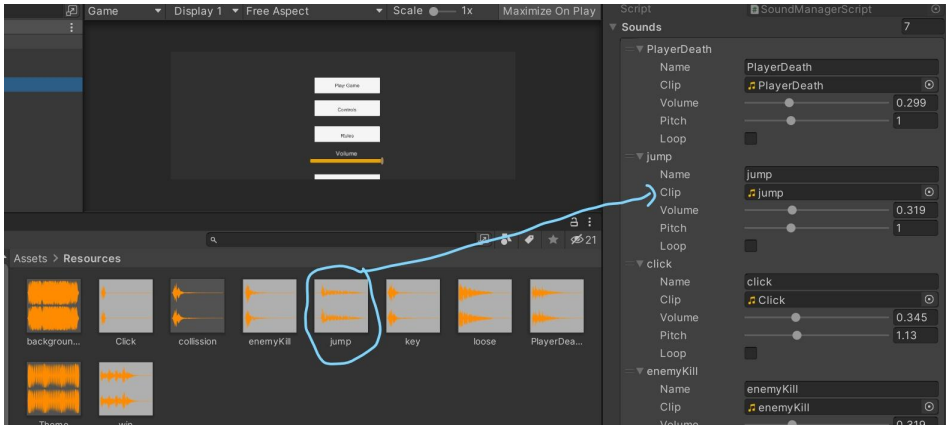


VI. On the ‘Inspector’ (Right Hand side) you will see the below screen.Please click on “Sounds”. Already you can see seven sounds being implemented.



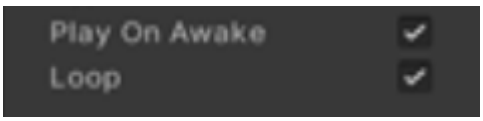
VII. The following are the seven added audio:

VIII. To change the audio, all you need to do is drag and drop your new audio clip under ‘Clip’ on the audio manager.



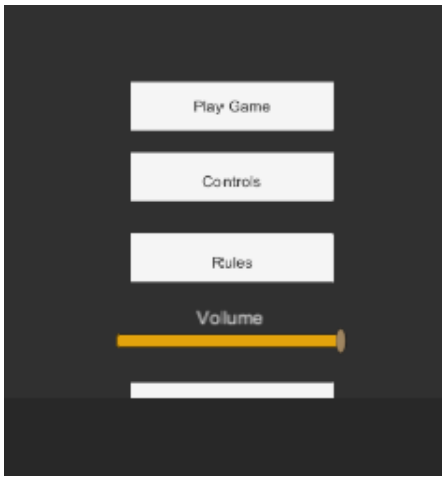
IX. For an audio, where you want it to repeat, for example the ‘Theme’ audio, you can click the **Loop** button the sound manager.

You can also click the ‘Play on Awake’ to Play on Awake will start the Audio Source as soon as the object is enabled.



02. Control Volume

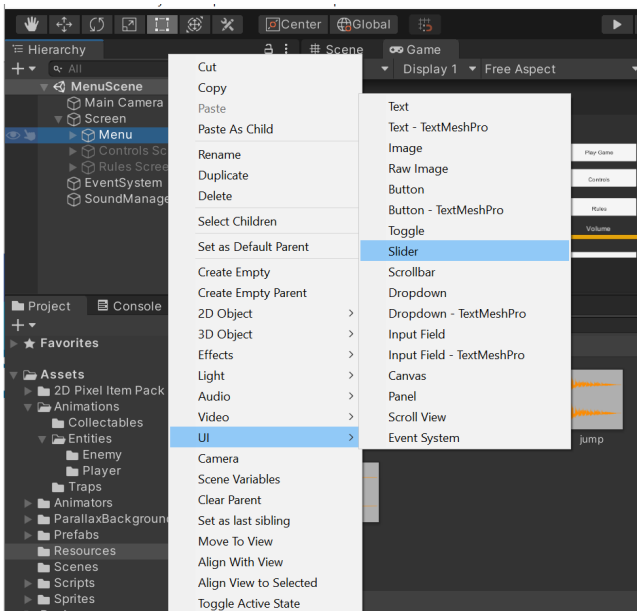
The entire game audio sound is controlled by a volume slider shown in the main menu screen. The volume is set between 0-1, with 1 being the loudest. When the player hits play, all audio sound is set to 1 and they can then adjust the sound to their preference.



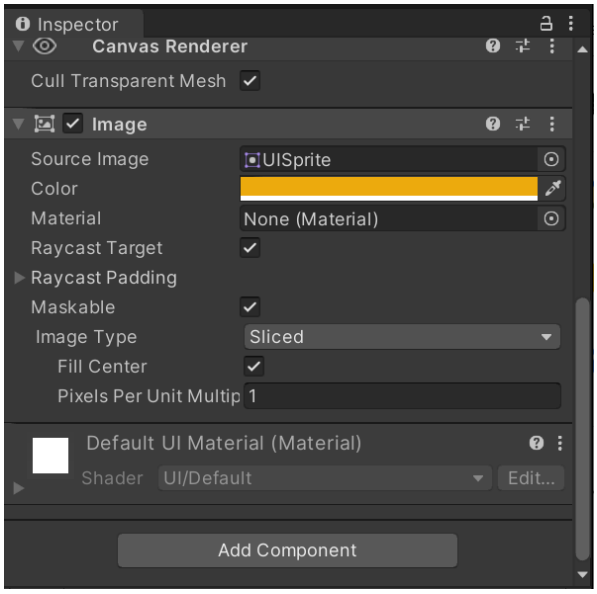
03. Creating a volume Slider

In order to create or design the volume slider, you will need to have already downloaded and added the music to the game, or you can use the music already input. Please follow the following steps:

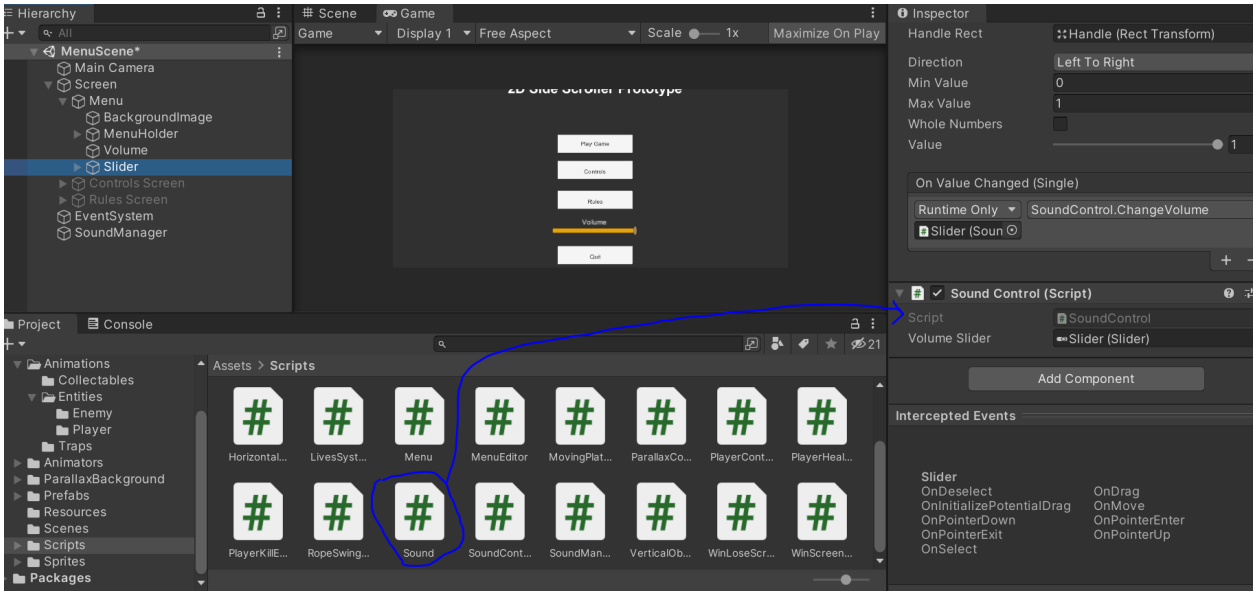
1. Create a new slider under ‘Menu’ and you can name it as anything. For now we will use the name ‘Slider’.



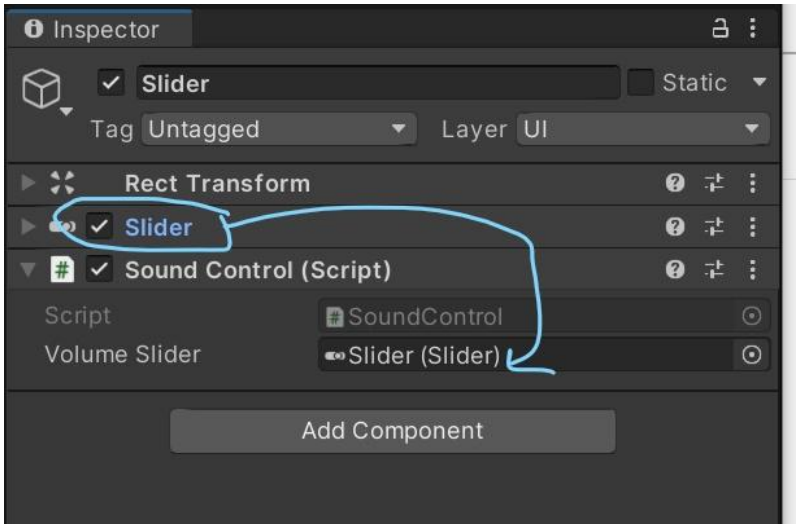
2. You can change the colour of the slider background, handle area and the fill area on the inspector.



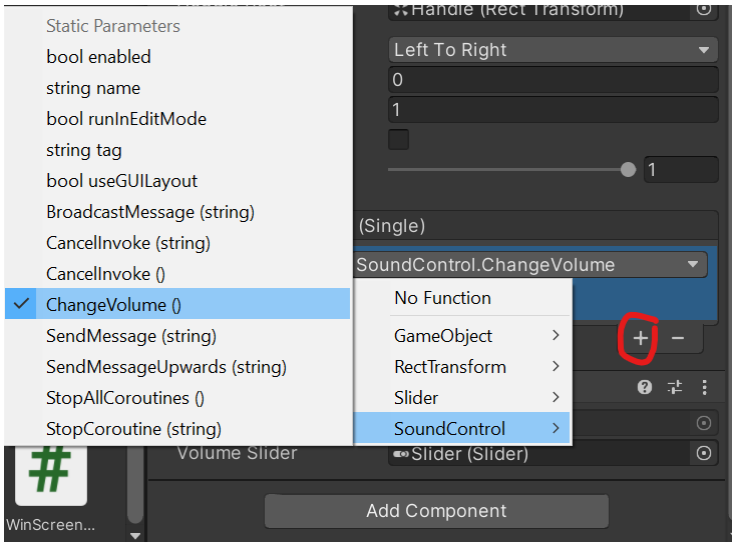
3. You will then need to attach the ‘Sound’ Script to the ‘Slider’ game object.

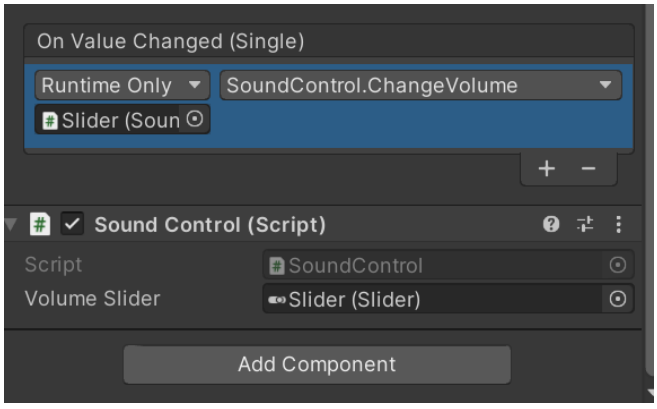


4. Then drag and drop the Slider component to it.



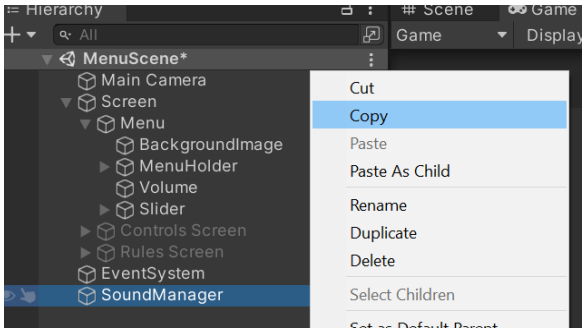
5. Under ‘On value change option’ click on the plus sign (+), then drag and drop the ‘Sound control Script’ to the ‘none (object)’.
6. Lastly, we want to call the ‘ChangeVolume’ function.
’



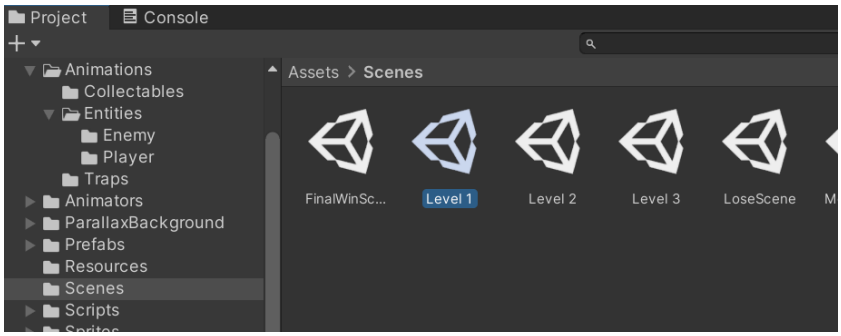


04. Adding a new sound Manager to the game

1. Select any of the original input screens. For example, you can click on the menu screen.
2. In the Hierarchy section, right click and copy the ‘SoundManager’.



3. Select the screen you wish to add the new sound manager to. To demonstrate, I will be using the ‘Level 1’.



4. Go to the selected screen hierarchy and paste the soundmanager on the Hierarchy.

Win Condition

Overview

For this simple 2D side scroller, the win conditions will be a key and a chest. The player will be required to collect the key in order to open the chest and win.

Key **Chest**

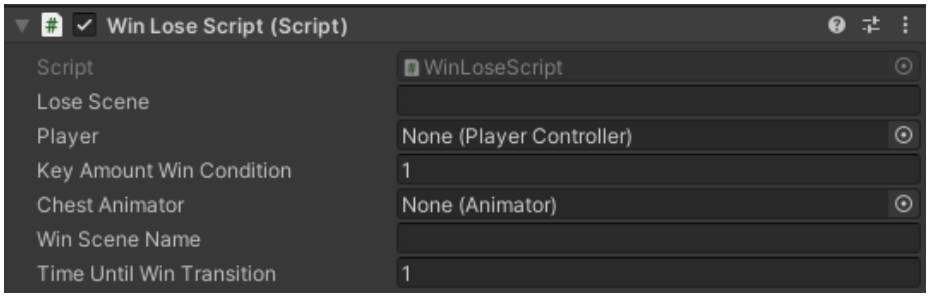
Required Scripts:

- WinLoseScript
- Collectables
- ScreenTransitionScript

Required Objects:

- Key Collectable

***NOTE - Refer to the Collectables section in this document above to learn how to add and implement a Collectable.**



Variables to focus on:

- **Player** - Player game object
- **Key Amount Win Condition** - the number of keys the player needs to open the chest and win the game
- **Chest Animator** - the chest game object in the **Hierarchy**'s animator object
- **Win Scene Name** - The name of the **'win'** scene file
- **Time Until Win Transition** - how long does the player need to wait from the time he touches the chest to the time the screen transitions to the '**win**' scene.
    - **If 0, then the screen changes instantly, if 1 then the player waits 1 second before the screen changes.**

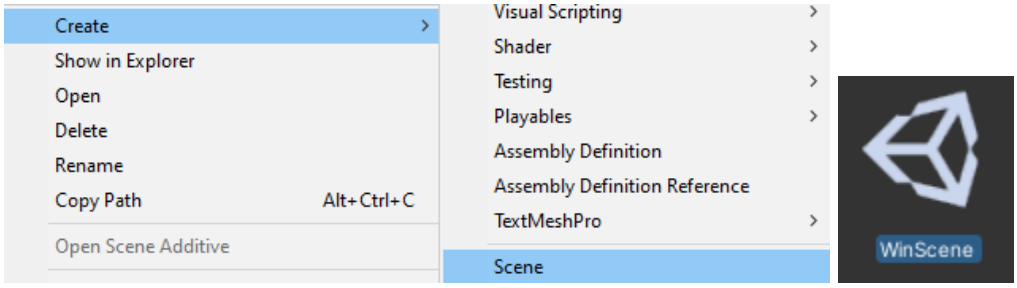
ScreenTransitionScript

Overview

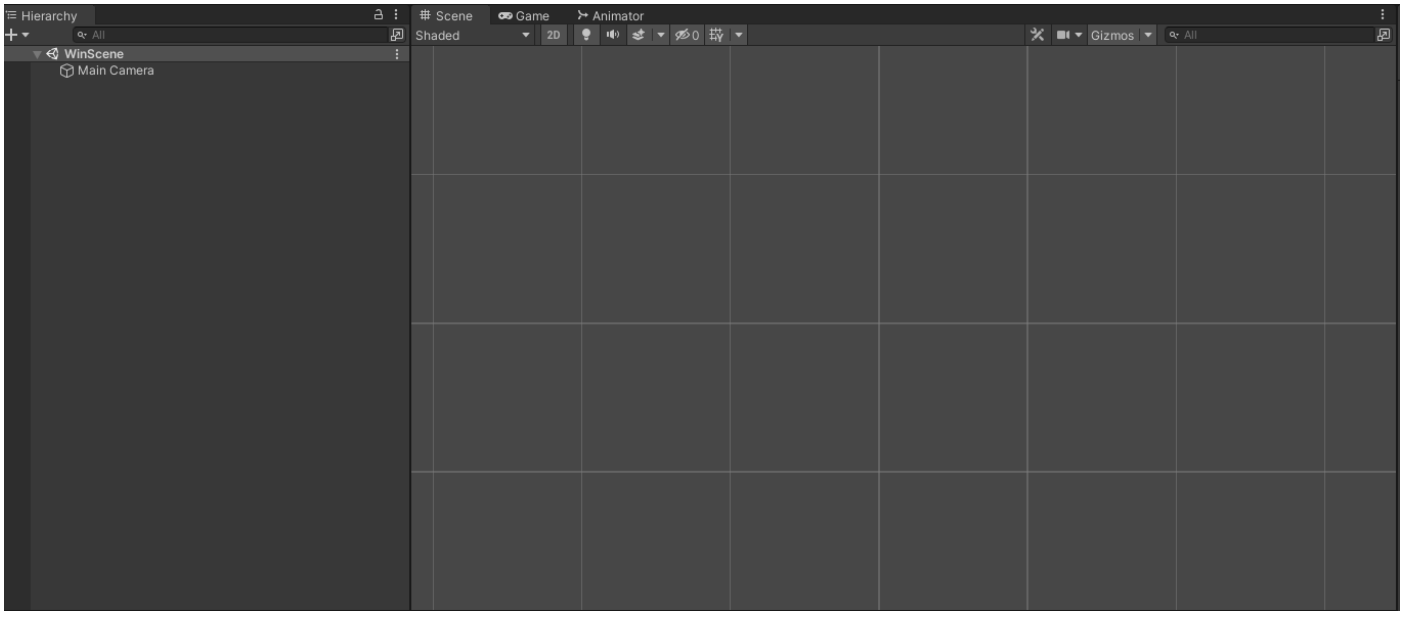
This script allows the player to change from the **‘win’** scene to another scene of their choice by clicking a button. You can have as many win screens as you want.

Creating a New Scene - Win Scene

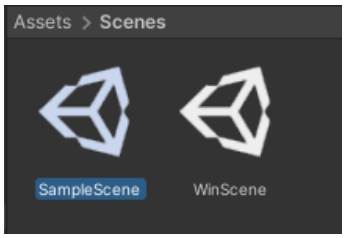
1. Right-click the **Scenes** folder in your Project file and create a new Scene. **Create > Scene**. Give it a name.



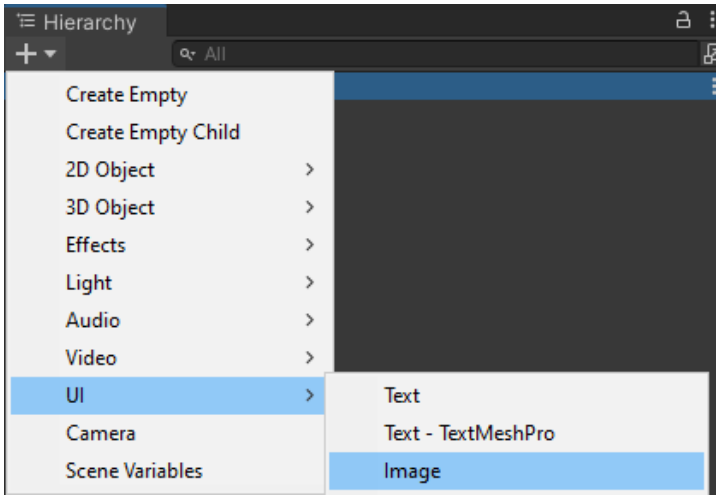
To edit the Scene, double click it and a new empty scene will open.



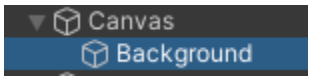
To go back, go into your Scenes folder and double click a scene.



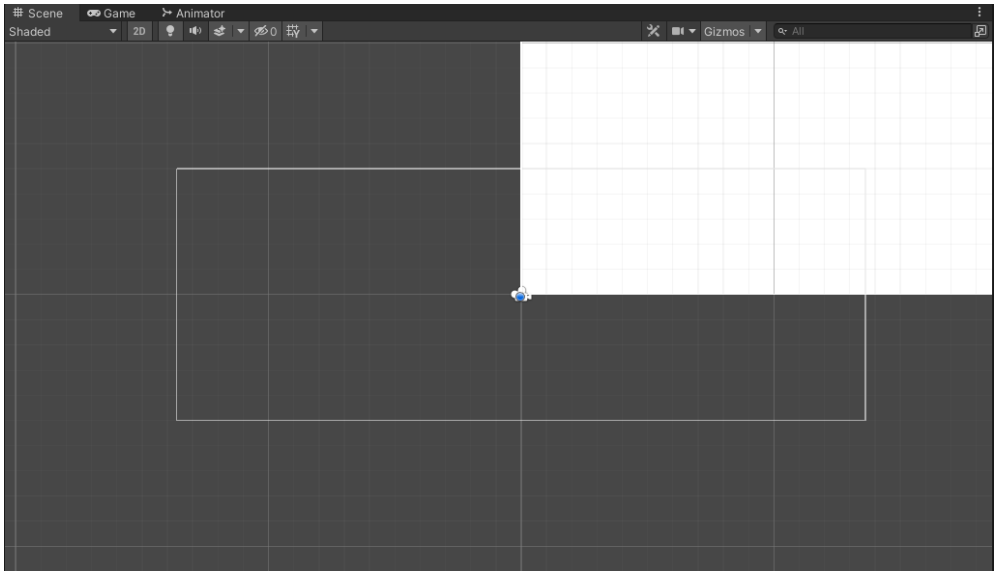
2. Create a new **Image** game object. **+ > UI > Image**

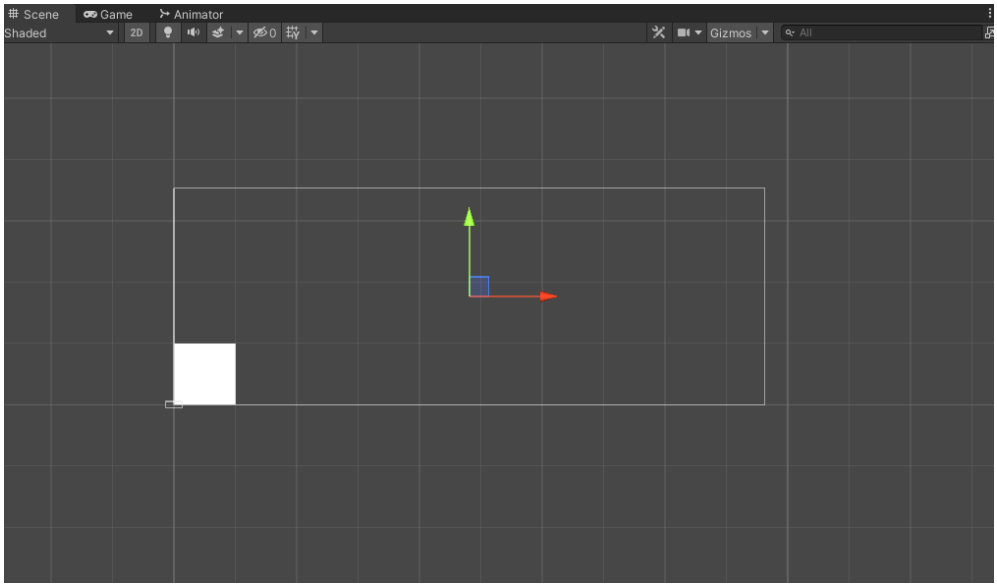


3. Name it **Background**.



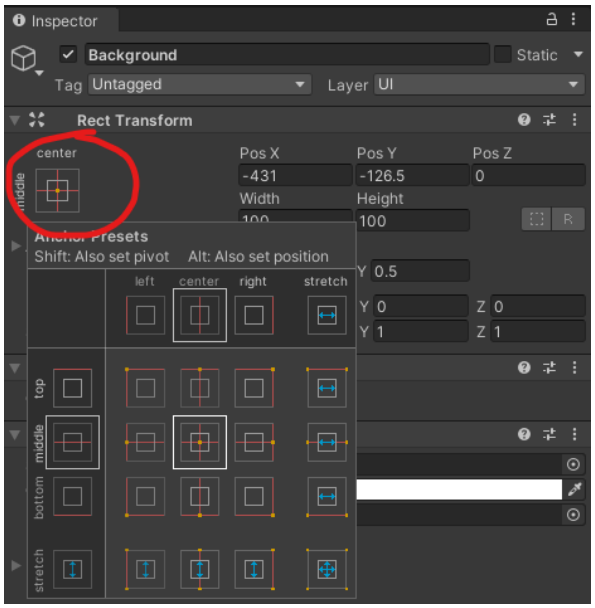
4. Click on the **Canvas** game object, hover your mouse over the **Scene** window and press **F** on your keyboard. **This will quickly zoom in and focus on the object in the Scene**.



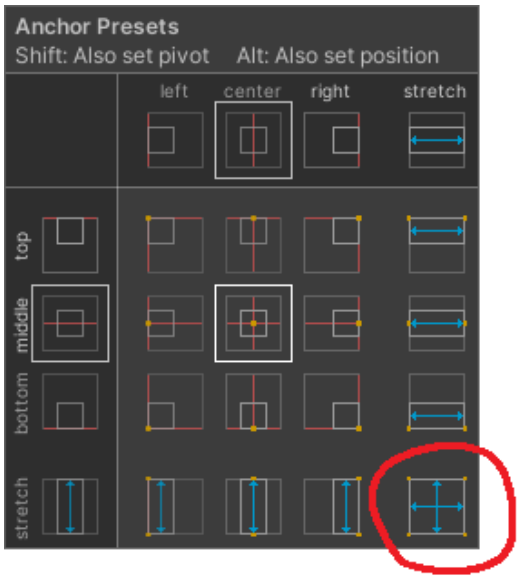


5. Click the **Background** object in the **Hierarchy**.

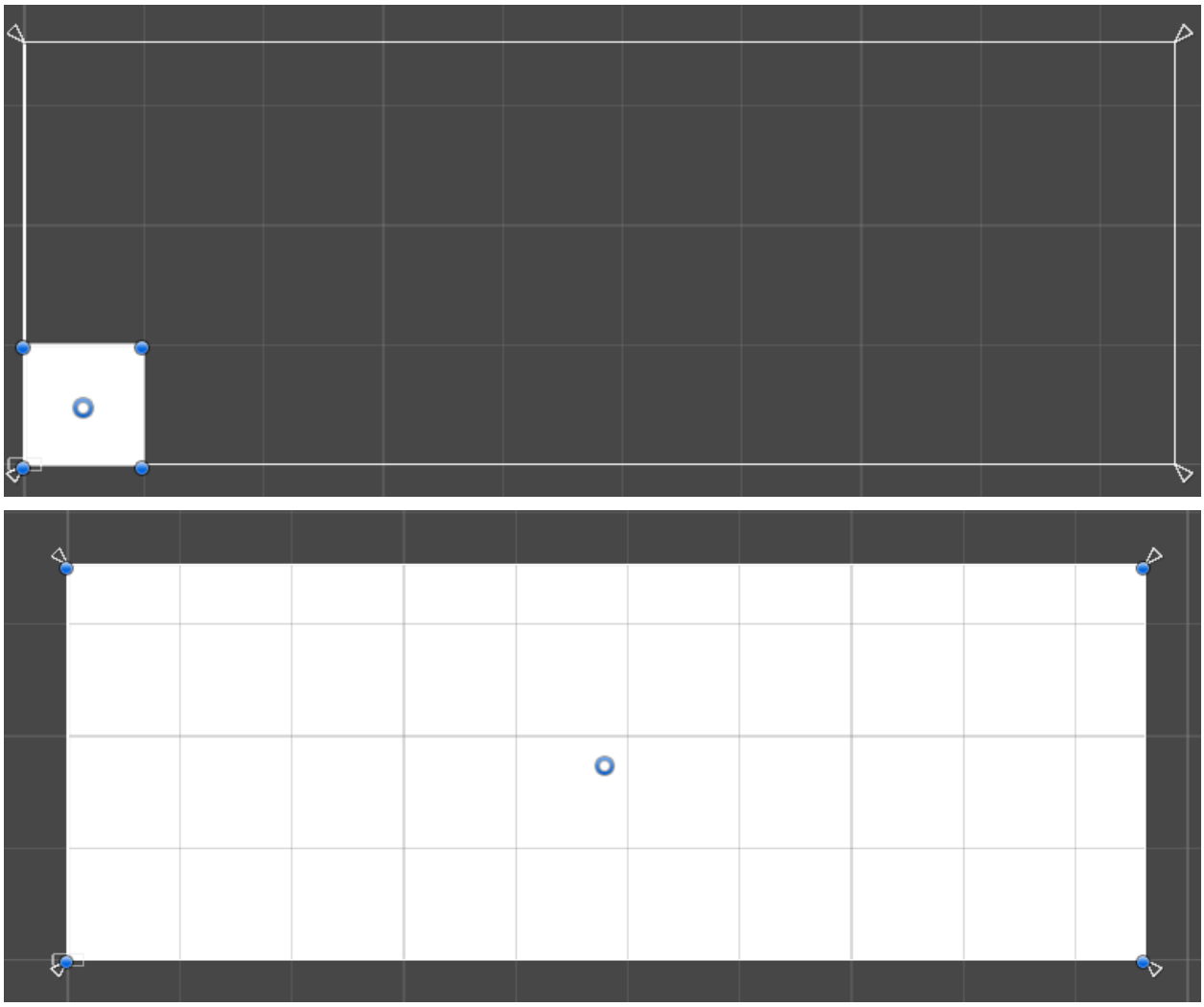
6. In the Inspector, click the **Anchor Presets** image to change the Rect size of the game object.



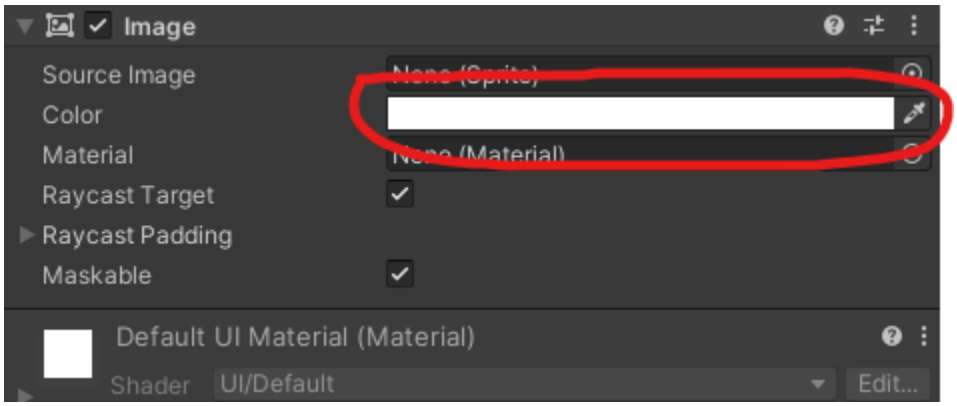
7. Hold **Alt** on your keyboard and click the bottom right icon in the **Anchor Presets** dropdown window.

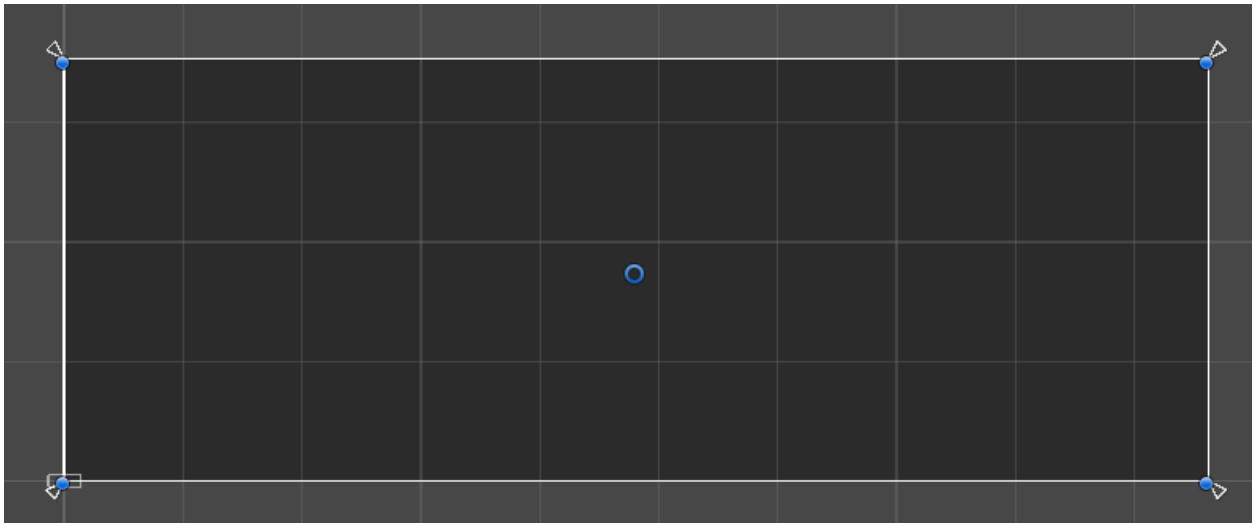


This will increase the size of the **Background** game object to the size of its parent object. In this case it will increase the size to be the same as **Canvas**.



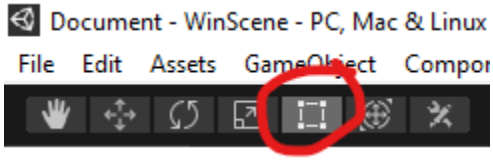
8. Change the colour to one that suits your liking.

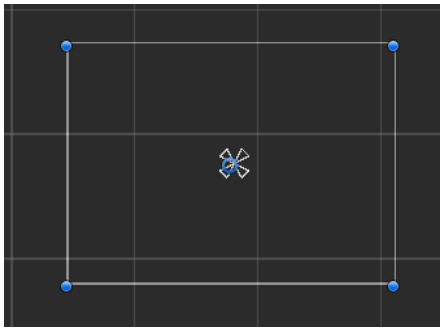




9. Create a new empty game object as a child of **Canvas** and name it **WinMessage**. Right-click **Canvas > Create Empty**.

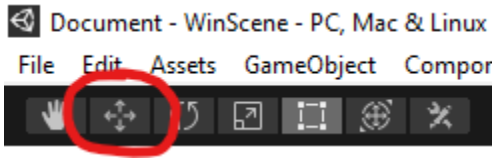
10. Resize using the **Rect tool**.

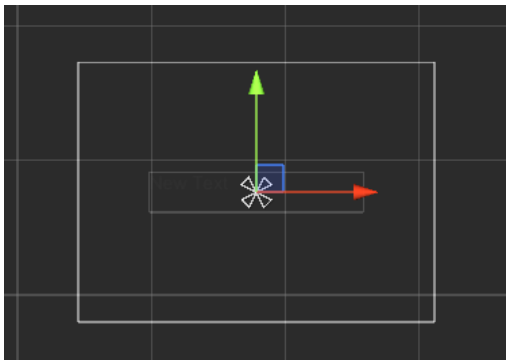


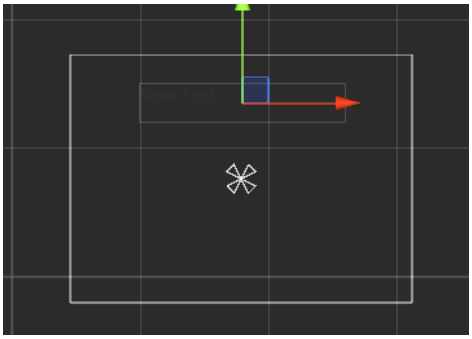


11. Create a **Text** object as a child of **WinMessage**. Right-click **WinMessage > UI > Text**. Give it a name. E.g. **MessageText**.

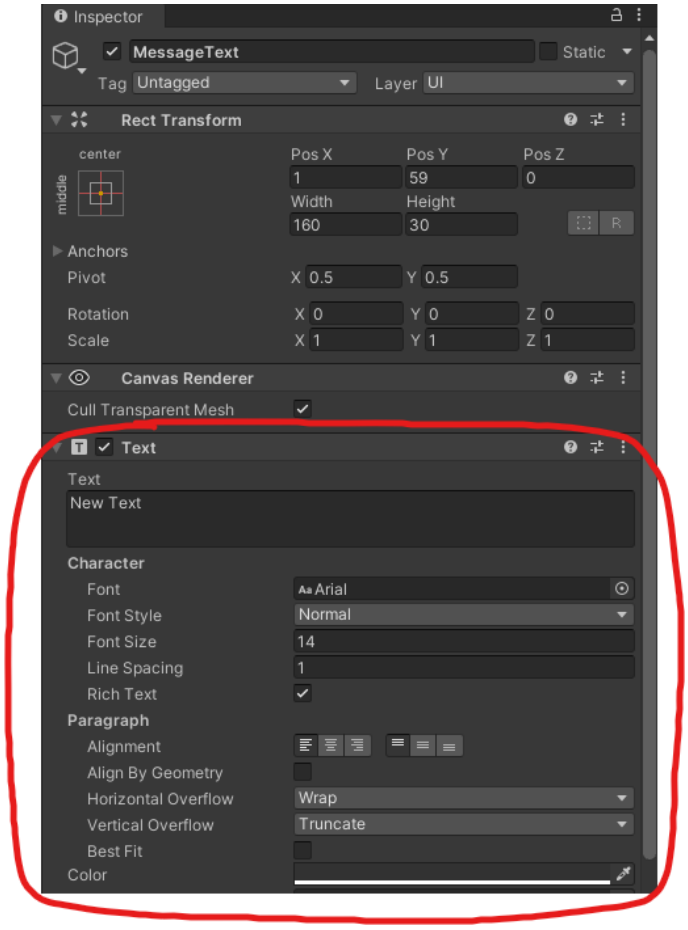
12. Click on the **Move tool** to adjust the position of the game object.

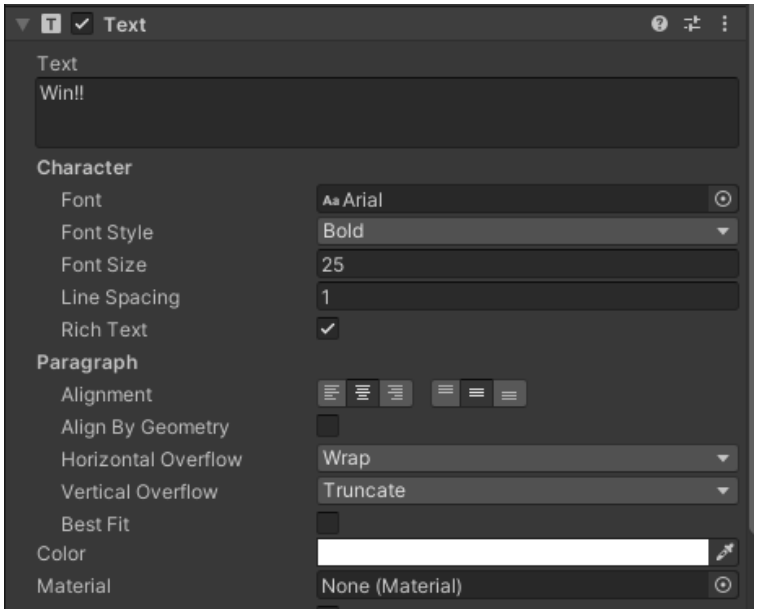




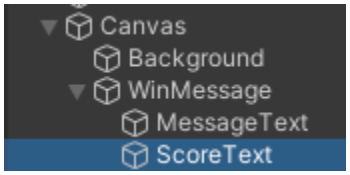


13. In the **Inspector** - **Text** component, customise the **Text** to your liking.

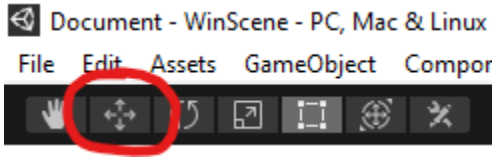


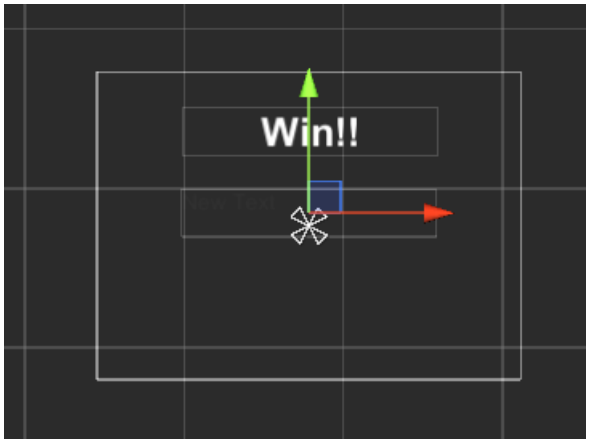


14. Create a new **Text** object as a child of **WinMessage**. Right-click **WinMessage > UI > Text**. Give it a name.

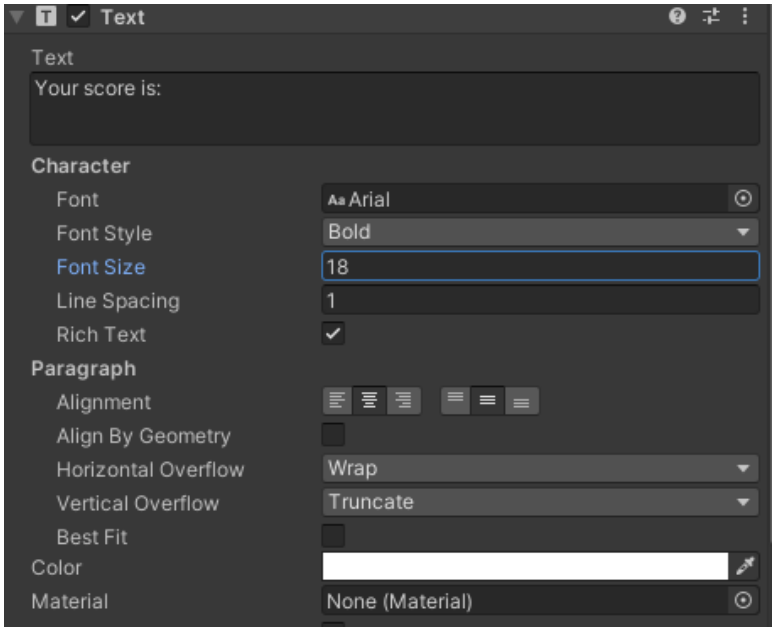


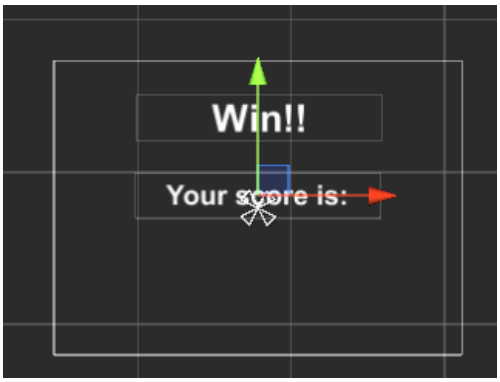
15. Adjust the position with the **Move tool**.



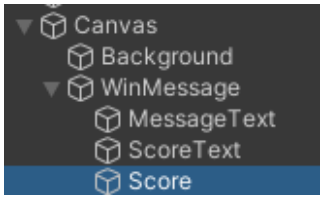


16. Change the **Text** component settings.

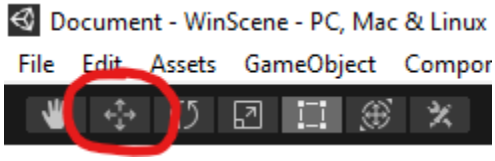


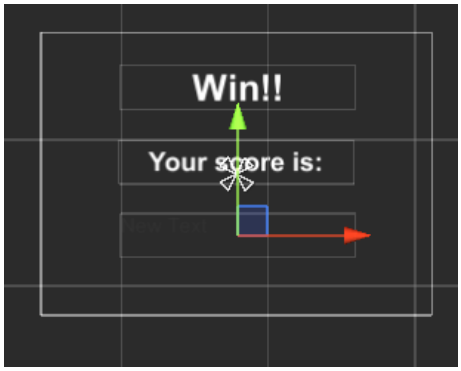


17. Create a new **Text** object as a child of **WinMessage**. Right-click **WinMessage > UI > Text**. Name it **Score**.

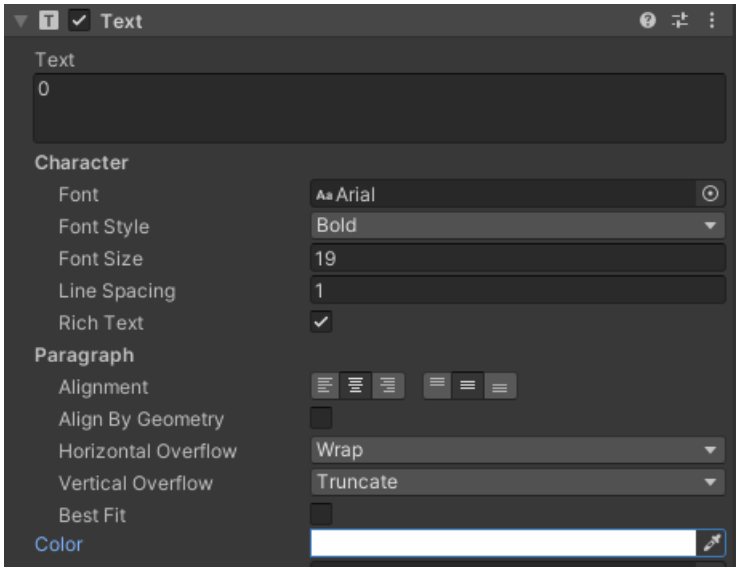


18. Adjust the position with the **Move tool**.

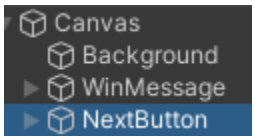




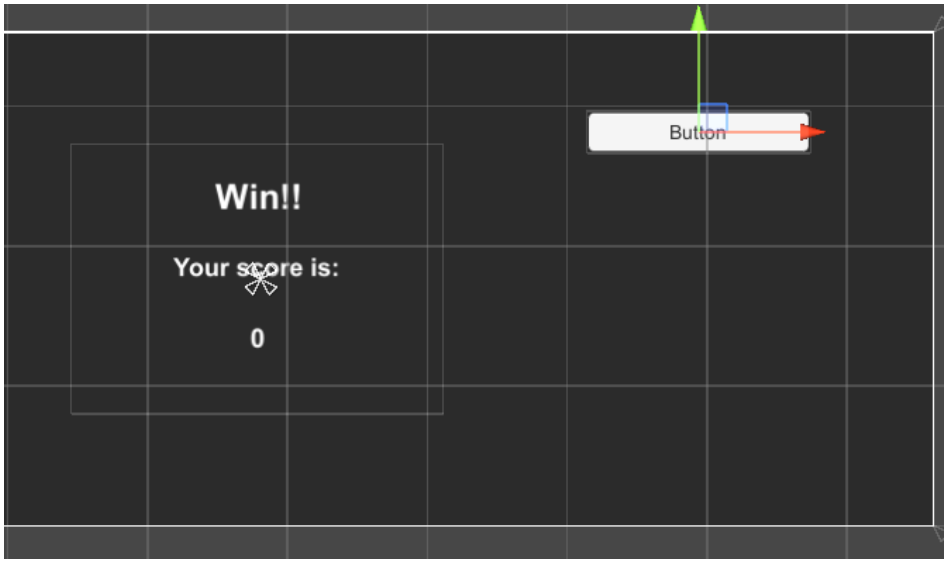
19. Customise the **Score** text component.



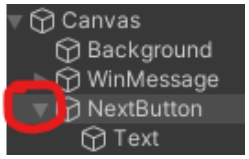
20. Create a **Button** object as a child of **Canvas**. Right-click **Canvas > UI > Button**. Give it a name.



21. Adjust the position of the **Button** object you just created.

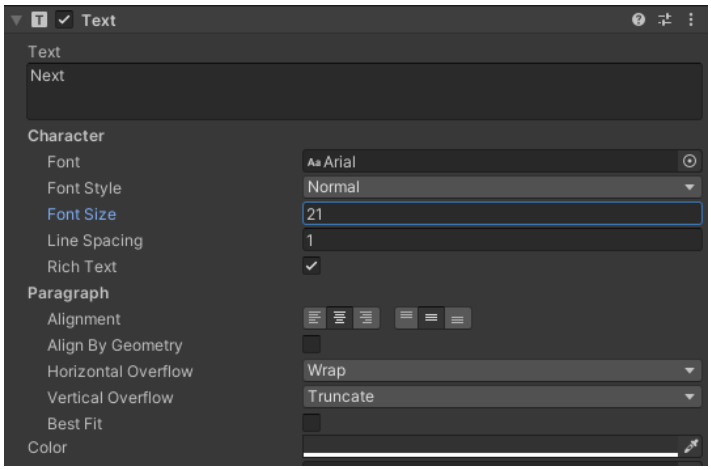


22. Expand the **Button** game object in the **Hierarchy**.

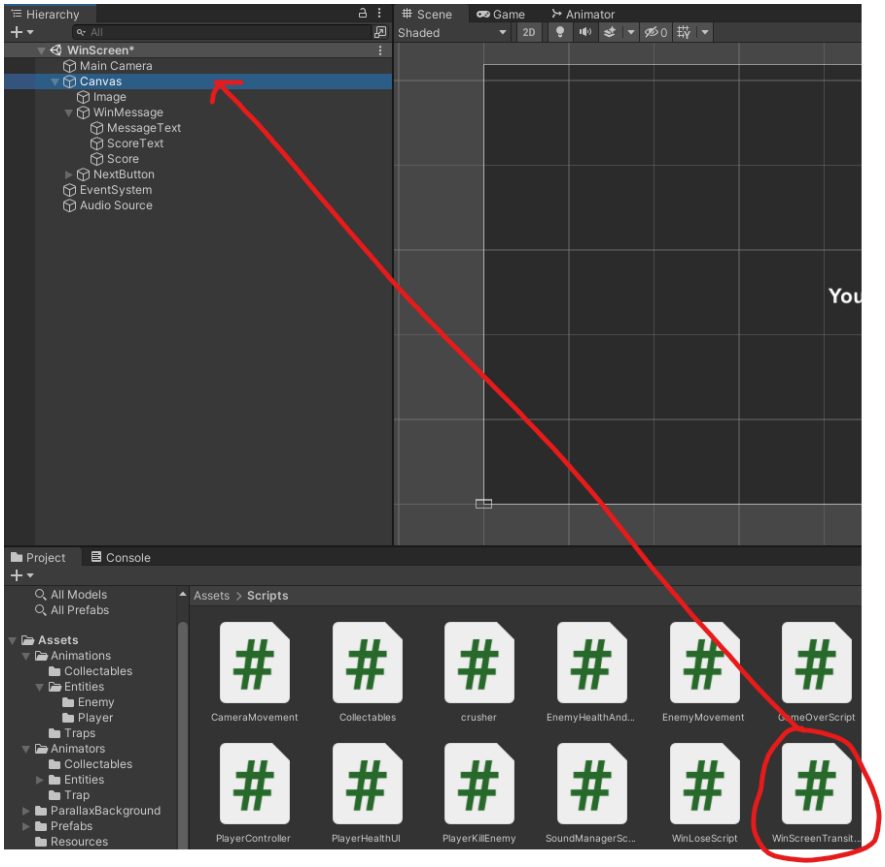


23. Click the **Text** game object.

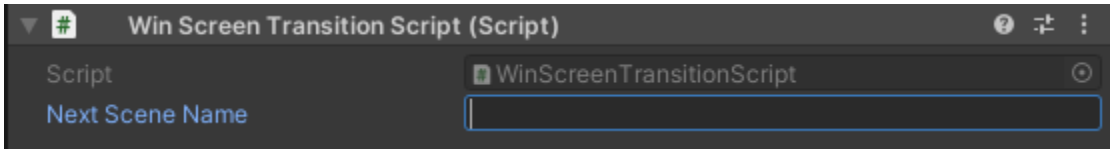
24. In the **Inspector** customise the text to your liking.



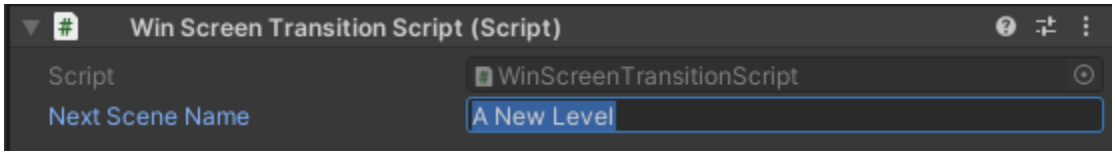
25. Drag the **ScreenTransitionScript** script into the **Canvas** game object.



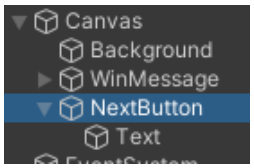
26. Click on the **Canvas** game object. In the **Inspector,** the component will show up as:



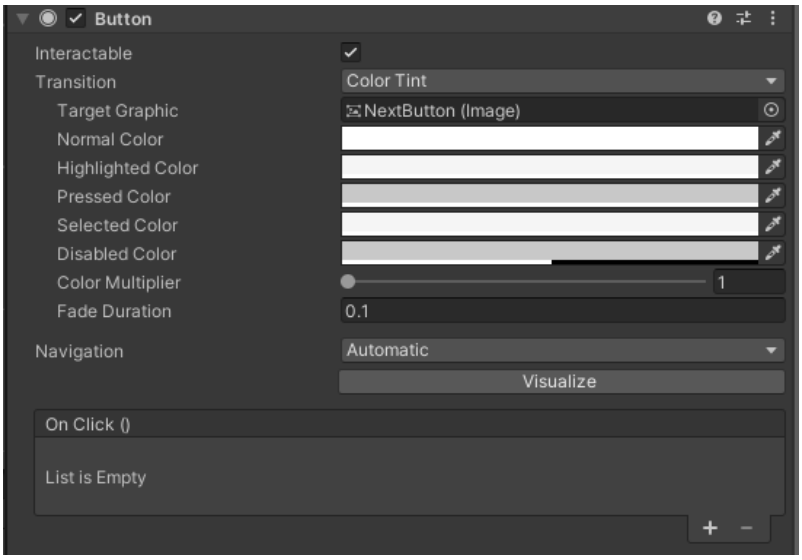
27. Create a new scene of your choice or use an existing scene. Type in the scene name into the **'Next Scene Name**' variable section.
    a. **It must have exactly the same name with spaces, numbers and symbols.**



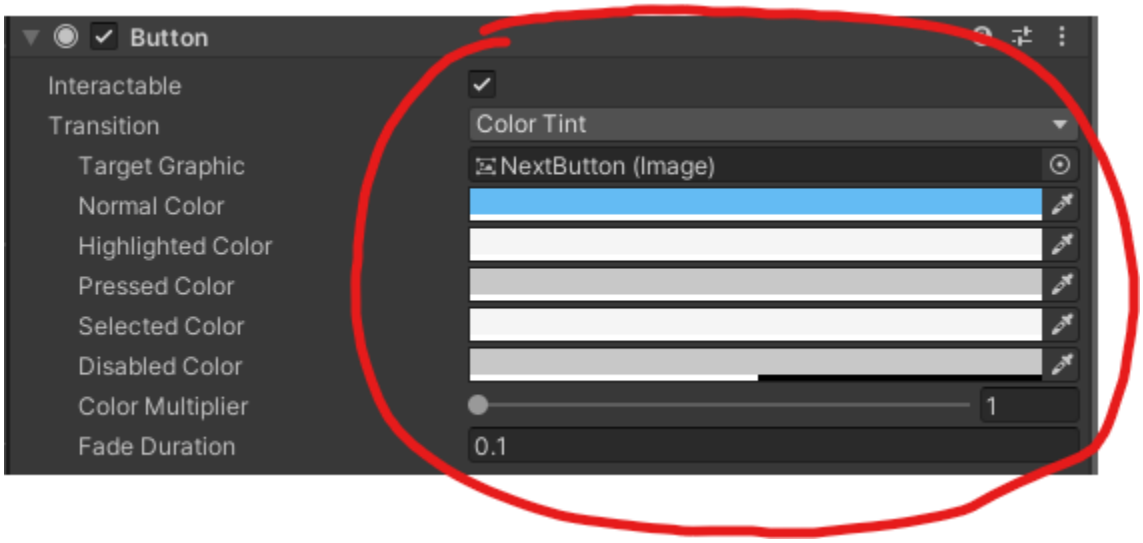
28. Click on the **Button** game object again.



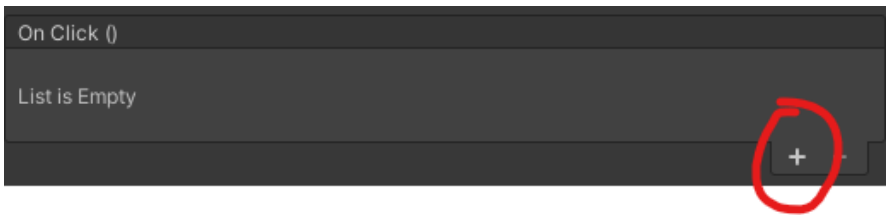
29. In the **Inspector**, scroll down to the **Button** component.

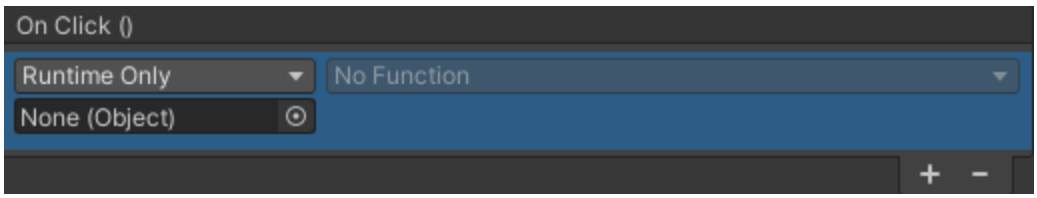


30. You can customise the button's aesthetics to your liking.

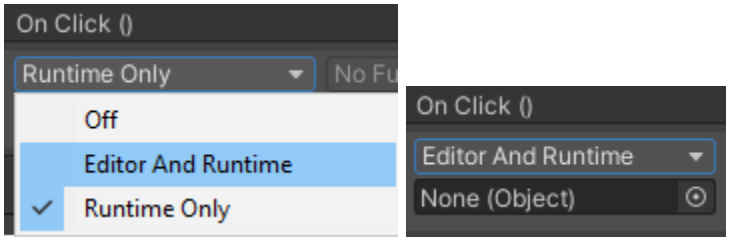


31. Below it in the '**On Click()'** section, click the **+** to add a function to the button it is clicked.

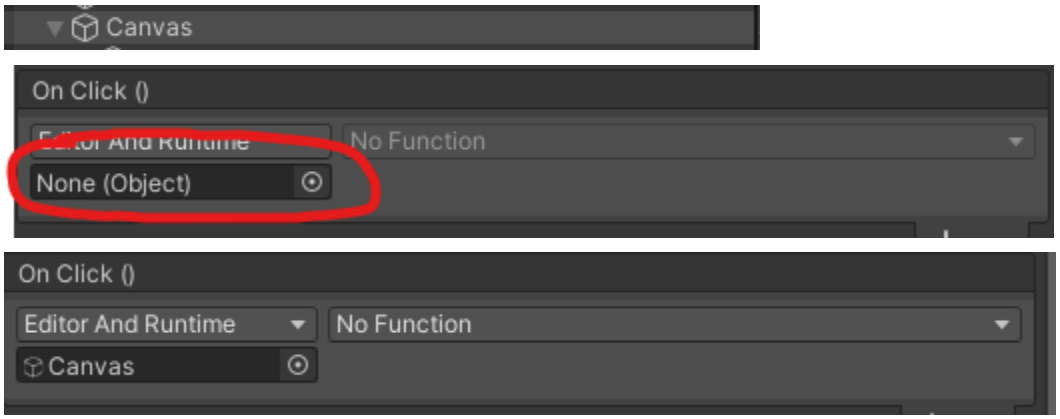




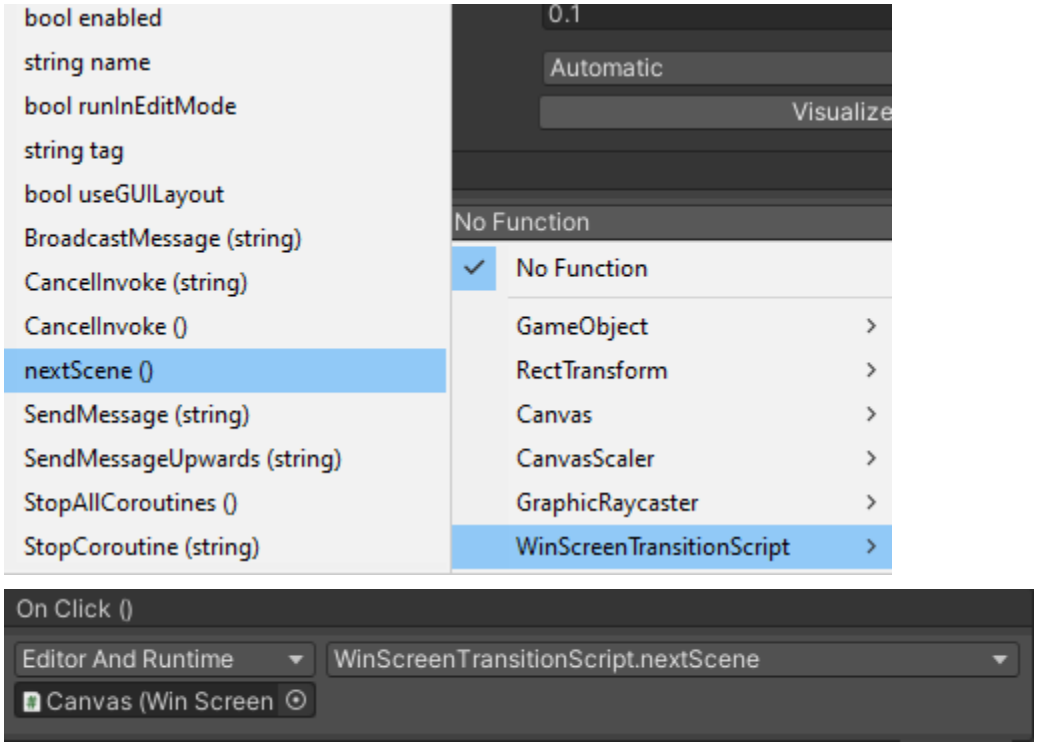
32. Click on **Runtime Only** and change it to **Editor and Runtime** to enable testing without needing to **Build and Run** the game.



33. In the **Hierarchy**, drag the **Canvas** object into the **OnClick()** variable **“None (Object)”**.



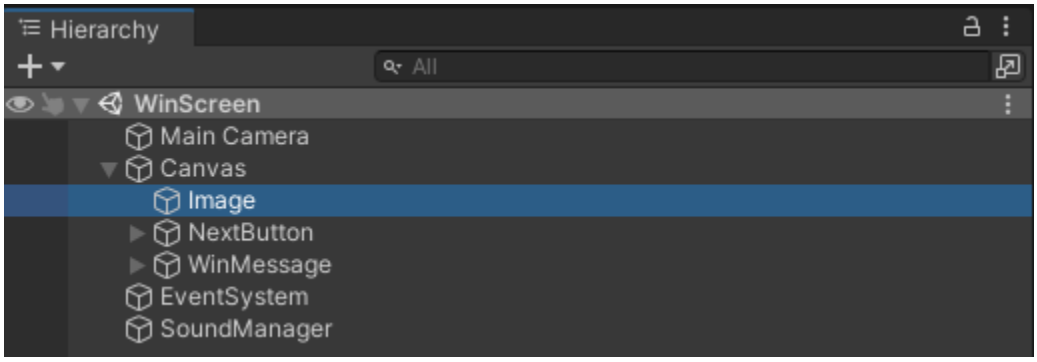
34. Click on the “**No Function**” dropdown. Click **ScreenTransitionScript > nextScene()**



This adds the “nextScene()” function to the button which when pressed, will change screens to the scene the player stated in the ScreenTransitionScript variable Next Scene Name.
In the example it will transition to the “A New Level” scene.

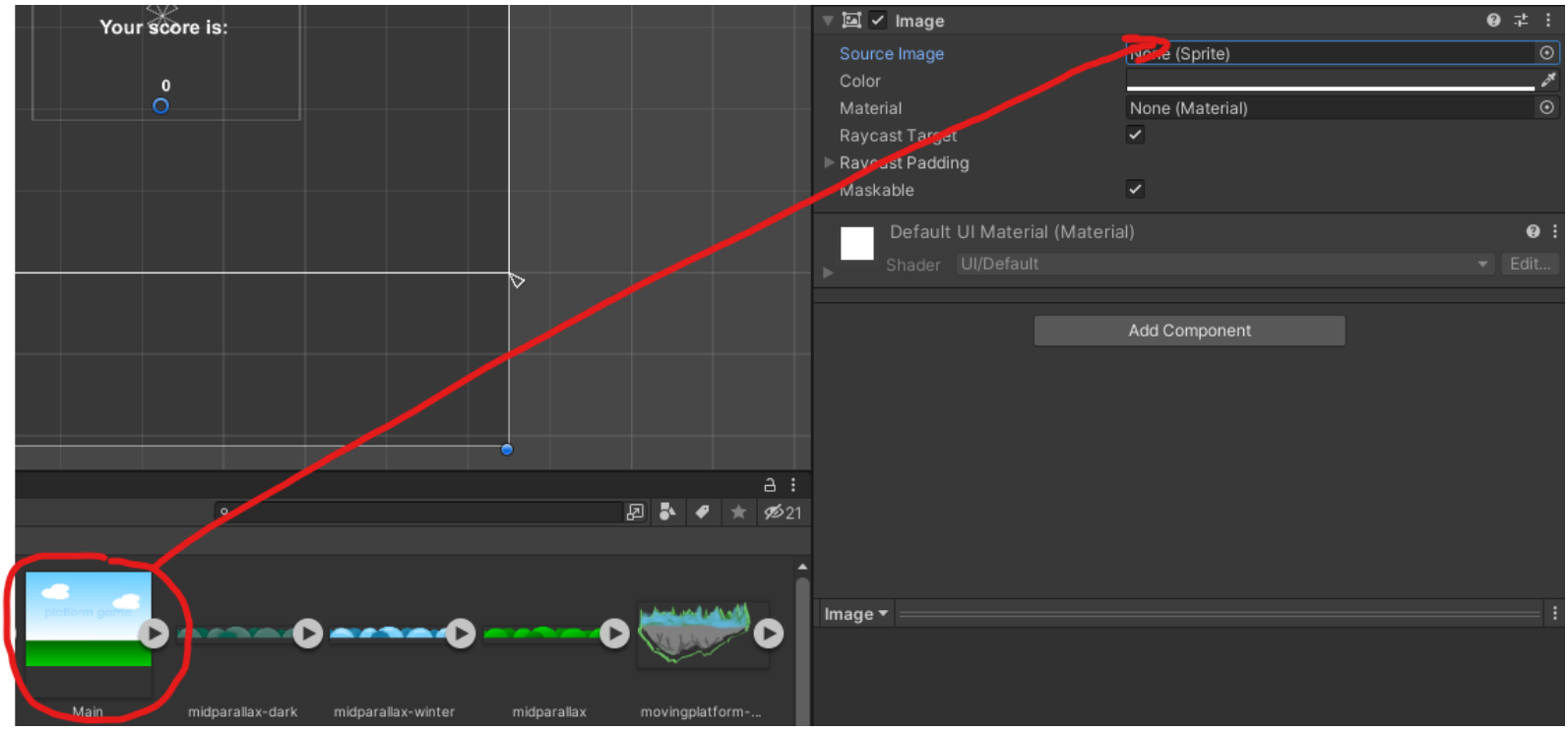
Adding an Image Background to Win Screen

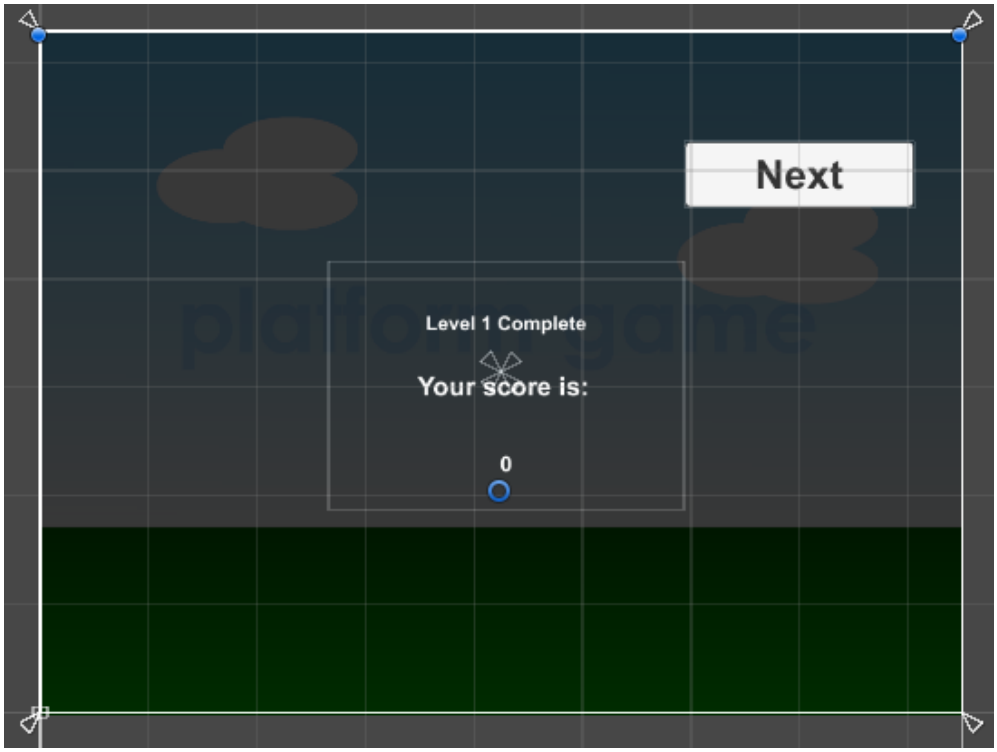
Select the background **Image** game object in your **Hierarchy**.



Drag and drop the desired image sprite from your **Project** window into your image's **Inspector.**

Image Component > Source Image.



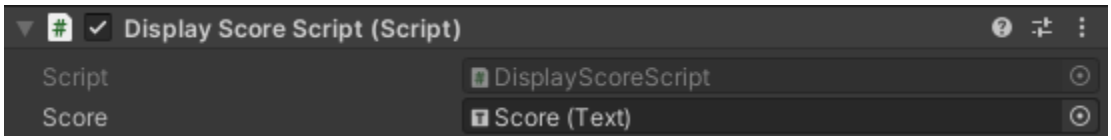


Win Screen Example Background in Canvas

DisplayScoreScript

Overview

Shows the player score on the specified **Text** game object.



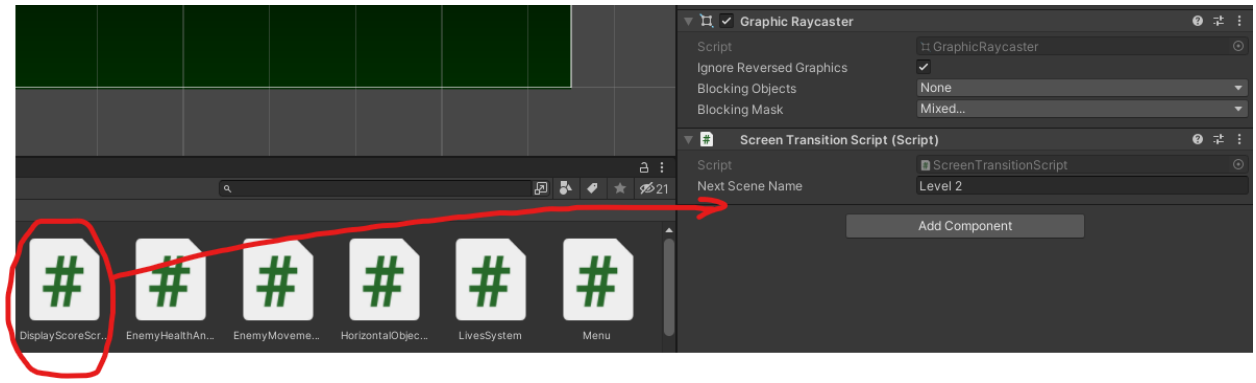
Variables:

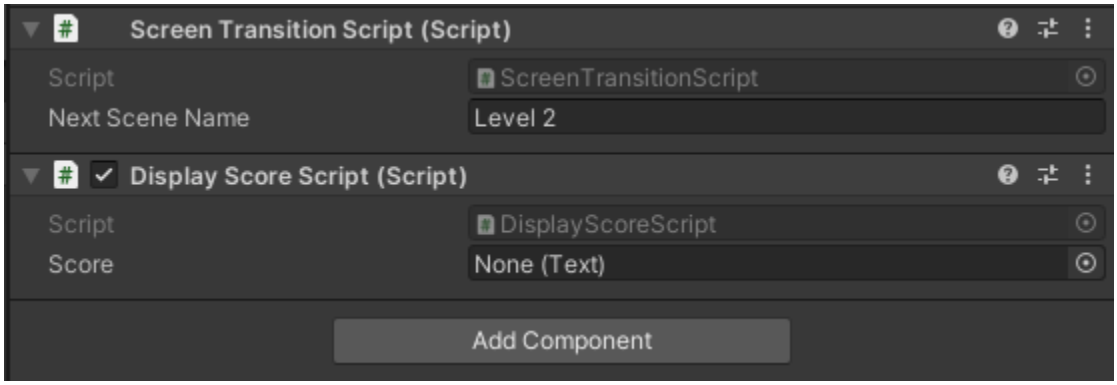
- **Score** - the **Text** game object that will display the score.

Adding and Implementing the Script

In your **Hierarchy**, click on your **Canvas** game object.

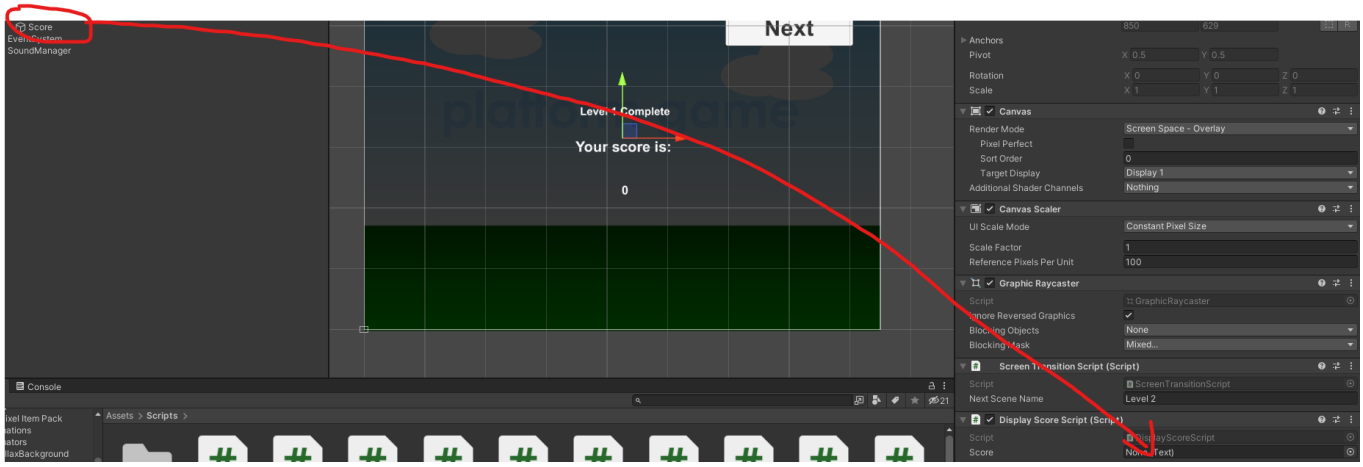
Drag the **DisplayScoreScript** script from your **Project** window into the **Canvas** game object's **Inspector**.





In your **Hierarchy**, find the **Text** game object that will display the score amount.

Drag it into the **Canvas** game object's **Inspector** with the script component **DisplayScoreScript**.



WinLoseScript - Win Condition

Overview

For the **Win Condition** portion of this script, it will get the number of keys the player currently has and compares it with the number of keys required to open the chest.

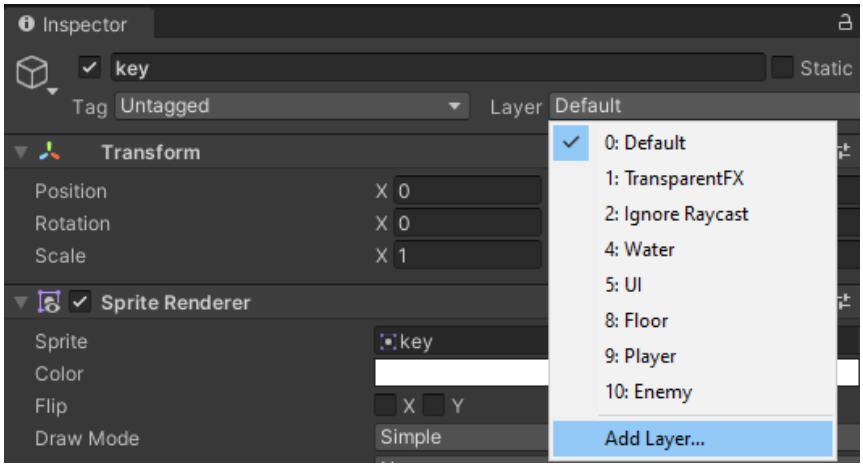
- If the number of keys is less than the amount required, do nothing.
- If the number of keys is equal to the amount required, when the player walks into the chest collider, the chest will open and transition to the '**Win**' scene

Setting the Key - Layer

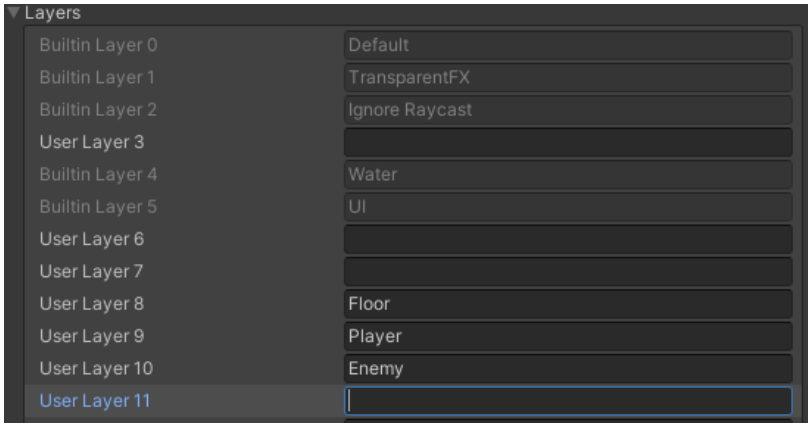
***NOTE - Refer to the Collectables section in this document above if you have not created a key collectable.**

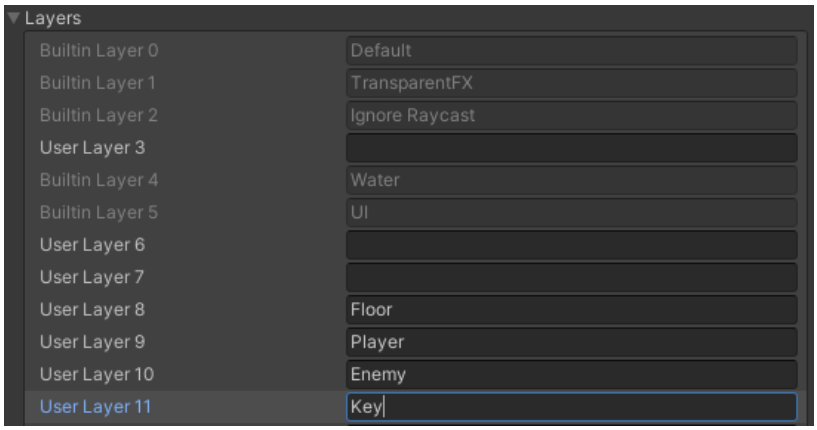
1. Click on the key object in the **Hierarchy**

2. Click the **Layer** dropdown and '**Add Layer…**'.

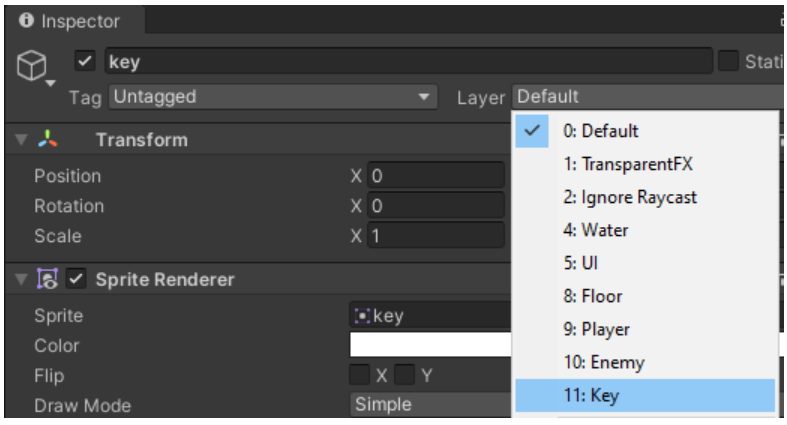


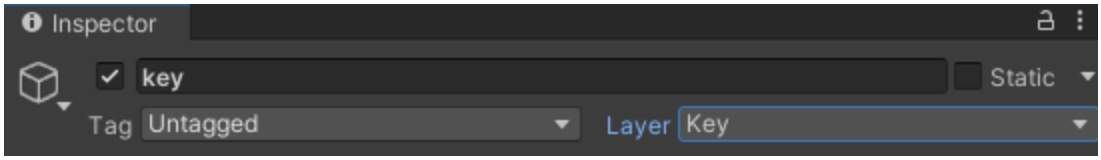
3. In a “**User Layer”** section, choose a number that is 8 or bigger. Type the name **“Key”**.





4. Go back to the **Inspector** for the key game object. Click the **Layer** dropdown and choose the newly made Layer **Key**.



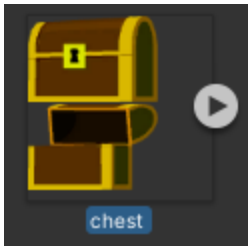


This will allow the functions dedicated to the Layer “Key” in the Collectables script to be used.

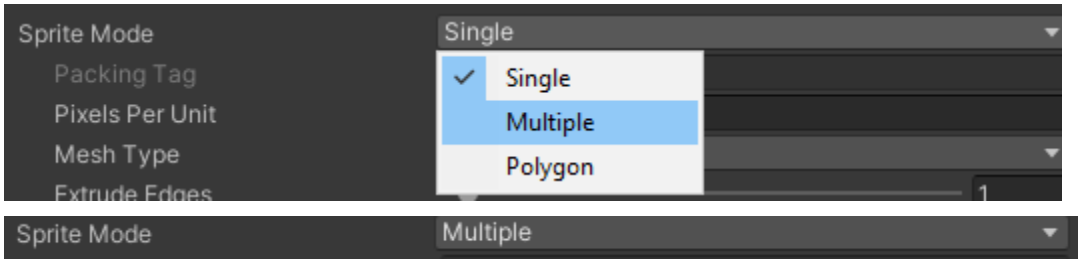
Chest

Sprites

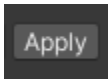
1. Click on the chest sprite in your **Project** window.



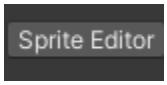
2. Set **Sprite Mode** from **Single** to **Multiple** to allow for sprite Slicing.

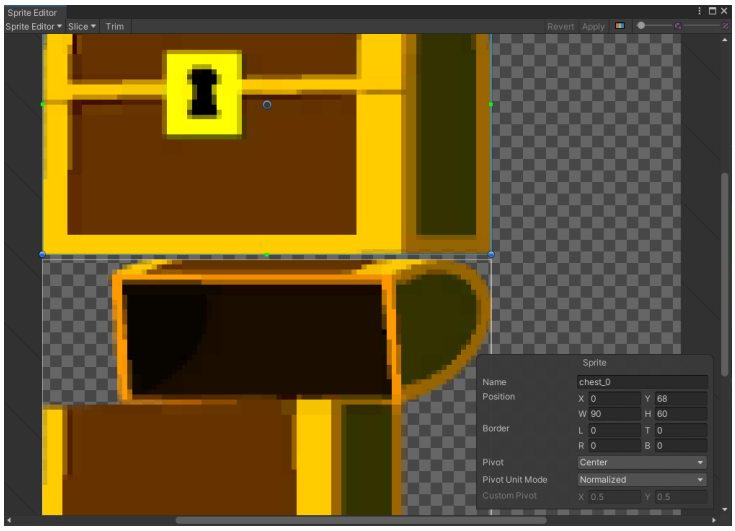


3. Click **Apply**

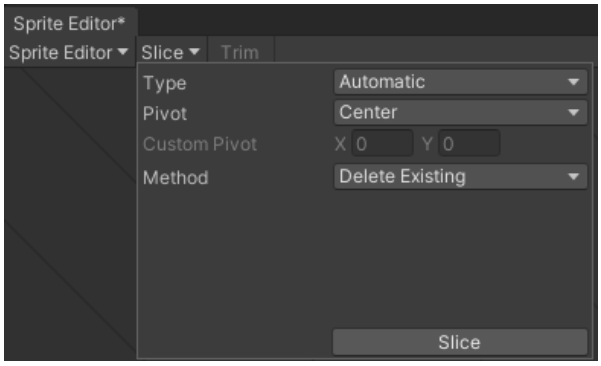


4. Click **Sprite Editor** and a new window called **Sprite Editor** will appear.

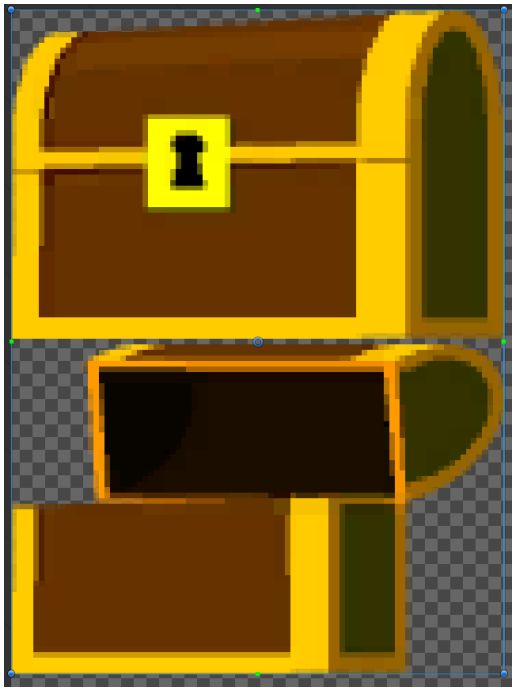




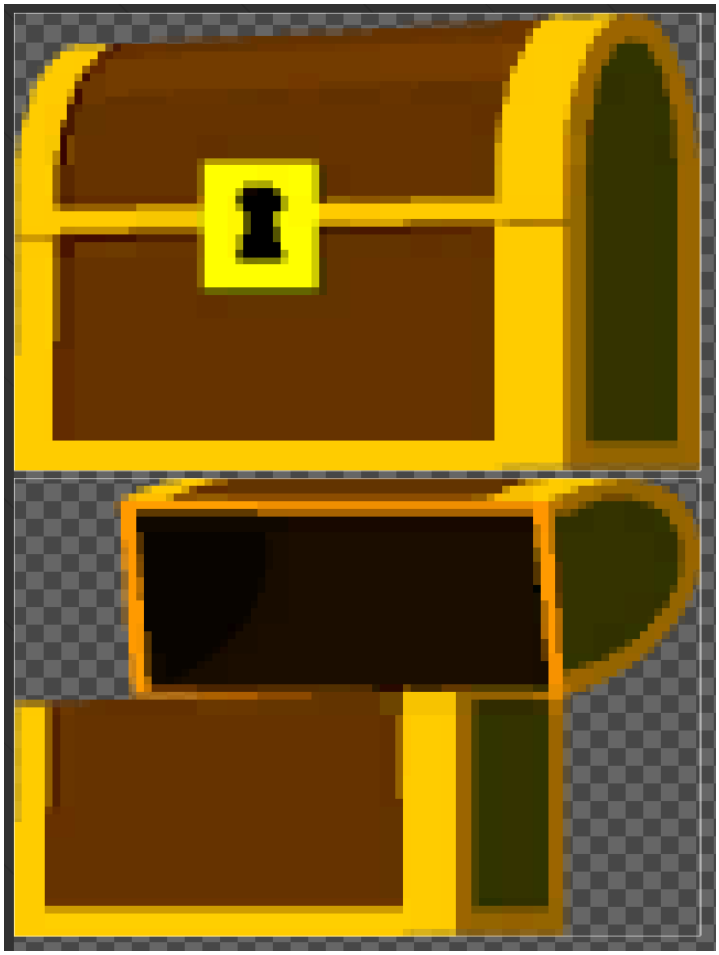
5. Click **Slice** dropdown **> Slice**



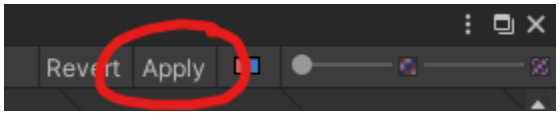
If it does not Slice properly and joins 2 sprites as one, as shown below:



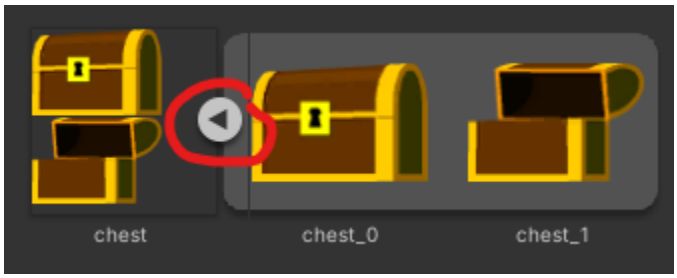
Click the box and delete it. Then manually drag a box around the sprites to draw.



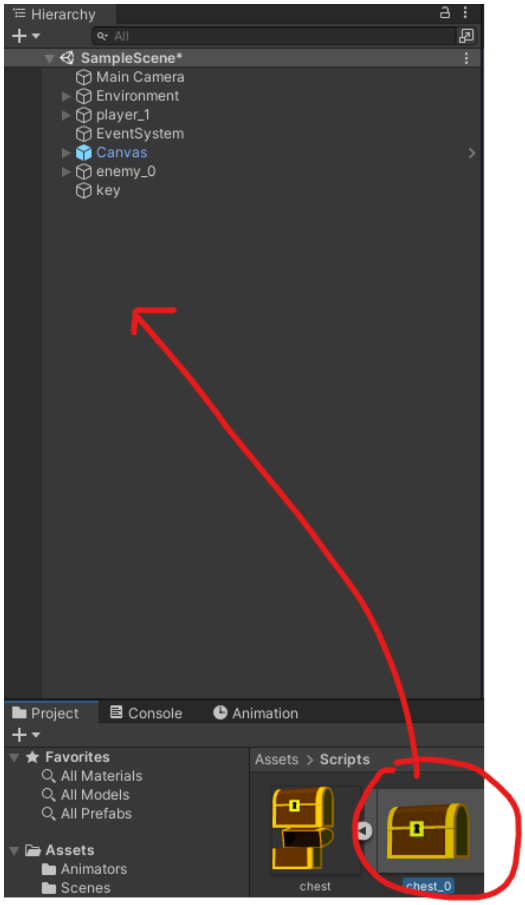
6. Click **Apply**

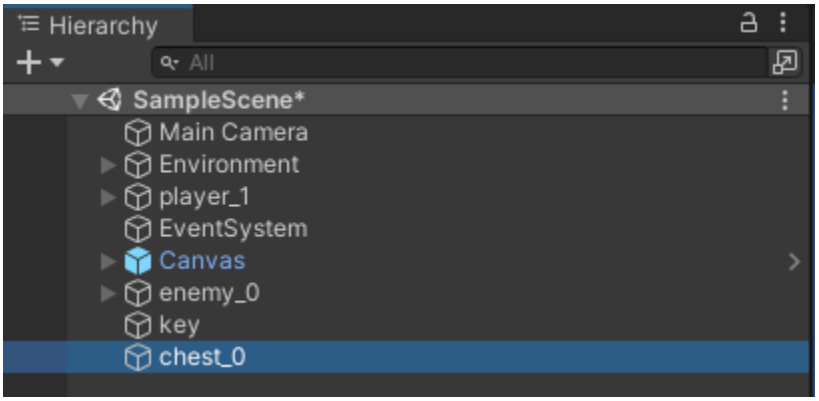


7. In your **Project** folder, click the arrow to expand your sprite and see your **Slices**.

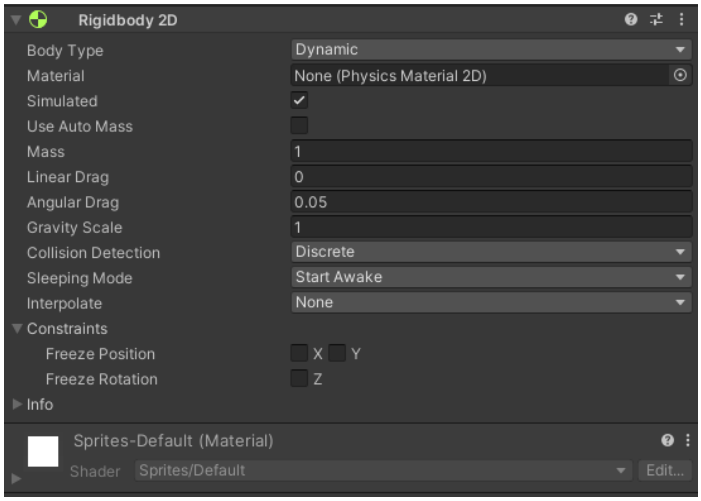


8. Select a **Slice** and drag it into the **Hierarchy**.

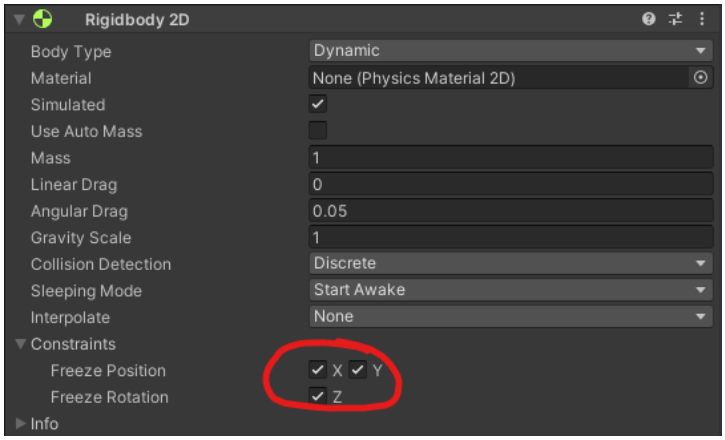




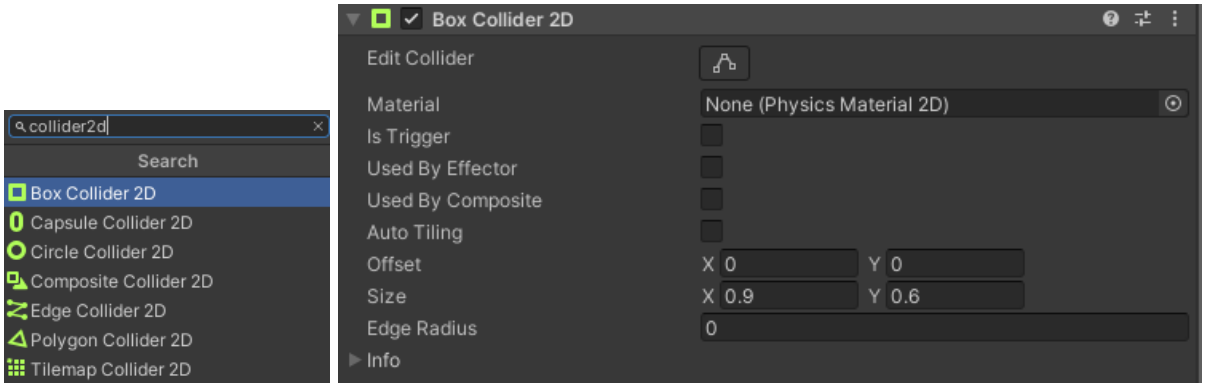
9. Add a **Rigidbody2D** component. **Add Component > Rigidbody2D**.



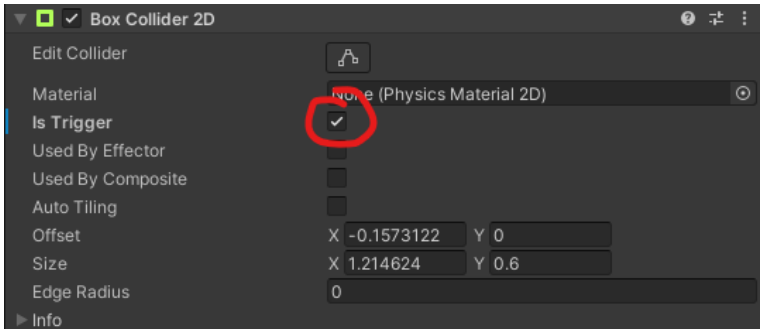
10. **Freeze Position** and **Freeze Rotation** for **X Y** and **Z**.



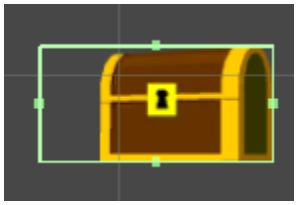
11. Add a **Collider2D** component**. Add Component >** any collider2D of your choice.
    a. **Here we chose a BoxCollider2D**



12. Tick **'Is Trigger**'



13. Click **Edit Collider** to edit the size if needed.

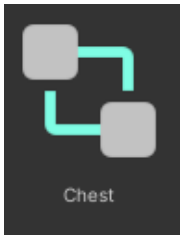


Animation

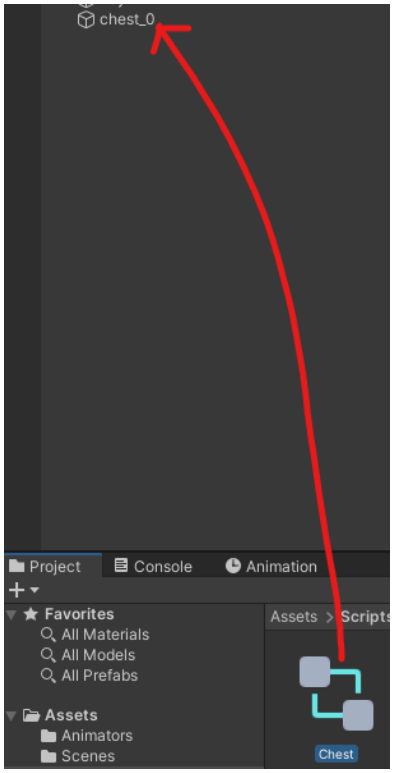
Animator

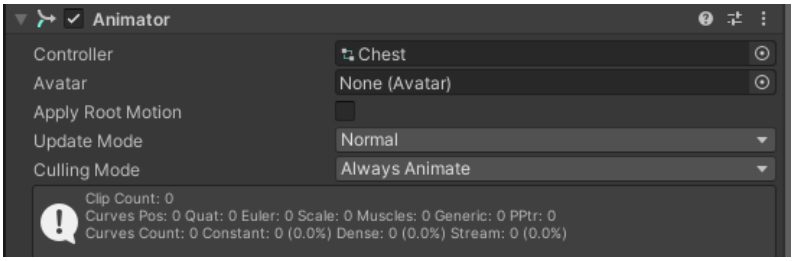
1. Make a new folder and give it a name.

2. In that folder make a new **Animator Controller** and name it the same name as the object it is for. Right-click the folder **Create > Animator Controller**.



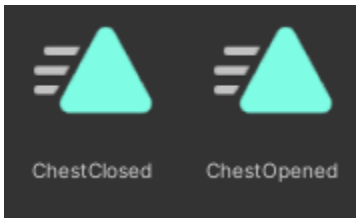
3. Drag the **Animator Controller** into your game object.
    a. **It will appear like this in your Inspector:**



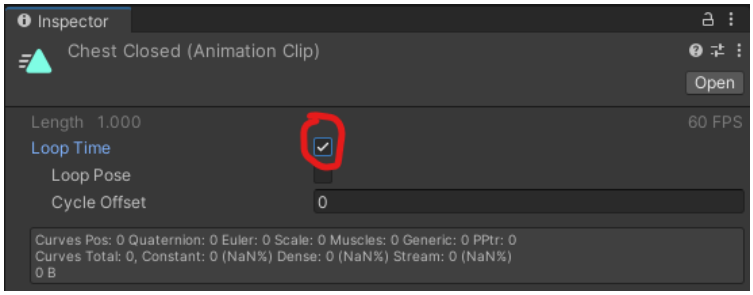


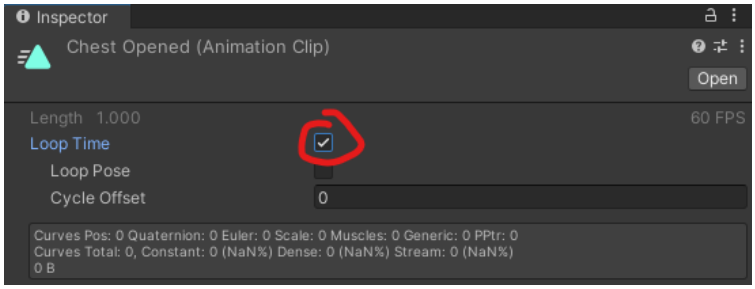
Animations

1. Create 2 **Animation** files; **ChestClosed** and **ChestOpened.**
    a. Right-click the folder **Create > Animation**.

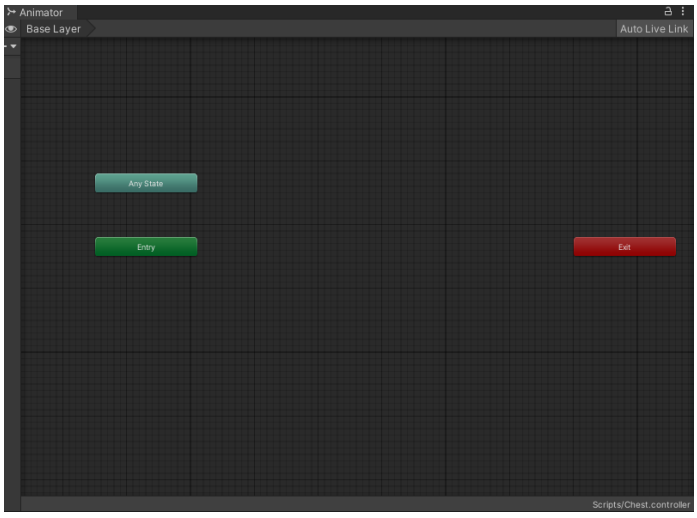


2. Click on each file and tick '**Loop Time**' in the **Inspector**.

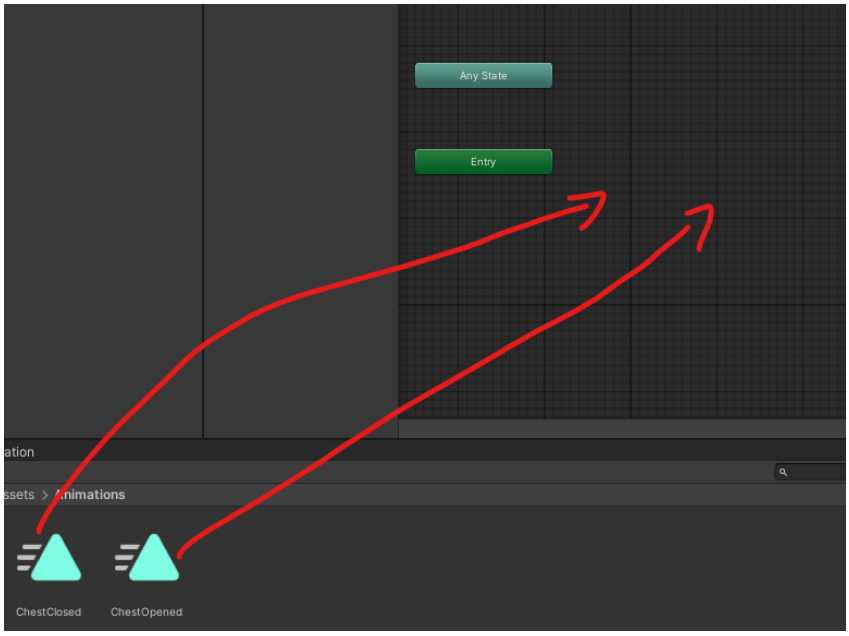


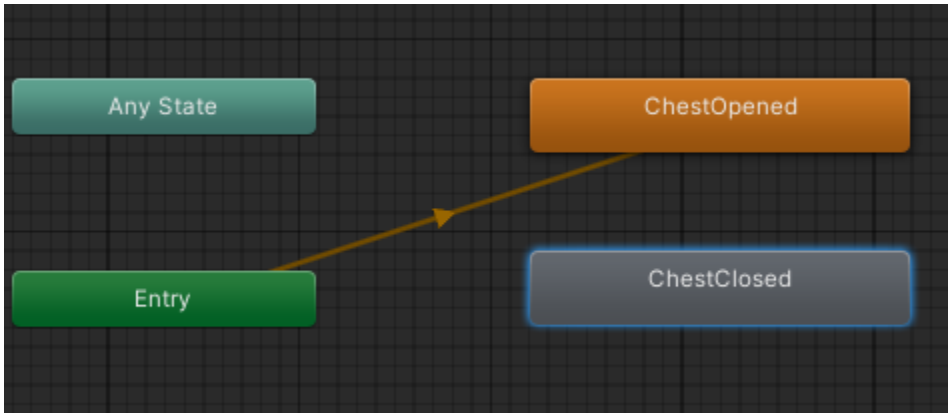


3. Go back to where you created the **Animator Controller** file, double click the **Animator Controller** file and it will open up an **Animator** window.

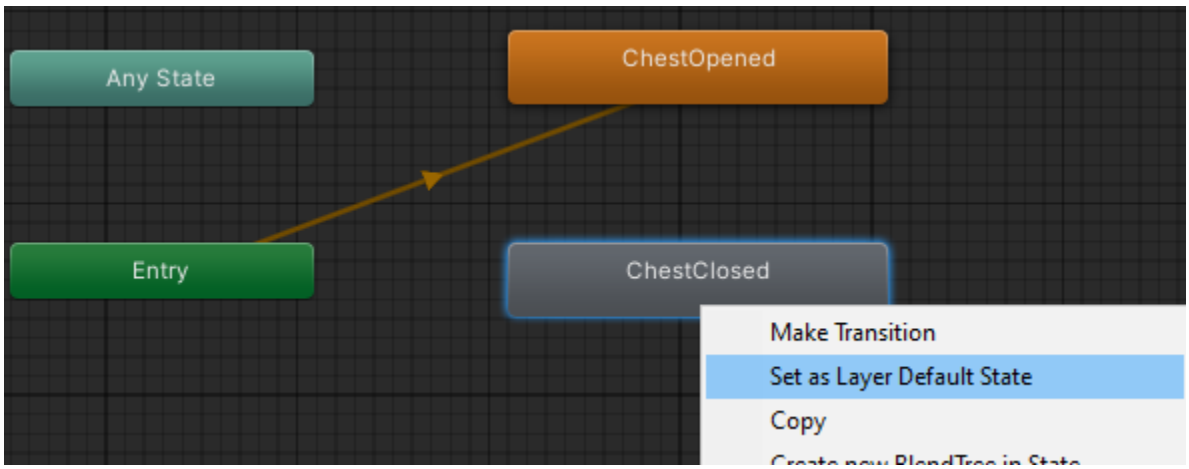


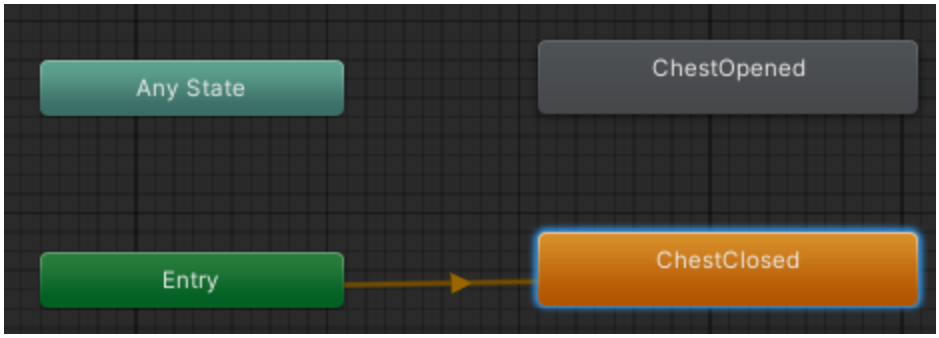
4. Drag each of the **Animation** files into the **Animator** window.



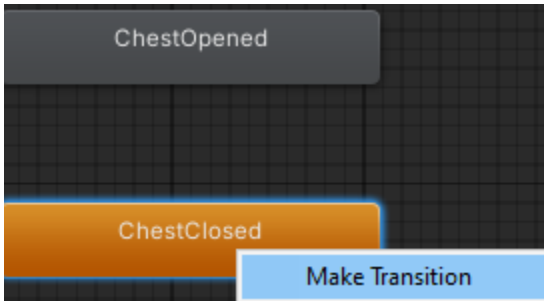


5. **Since we want the chest to be closed at the start and only open when we want it to**. Right-click **ChestClosed** and click '**Set as Layer Default State**'
   If ChestClosed is already the default state, skip this step.

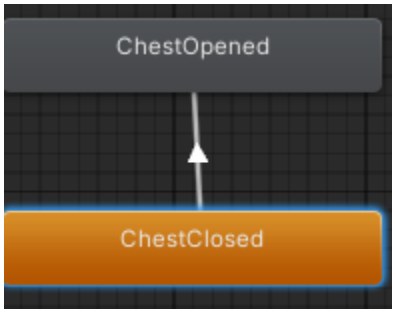




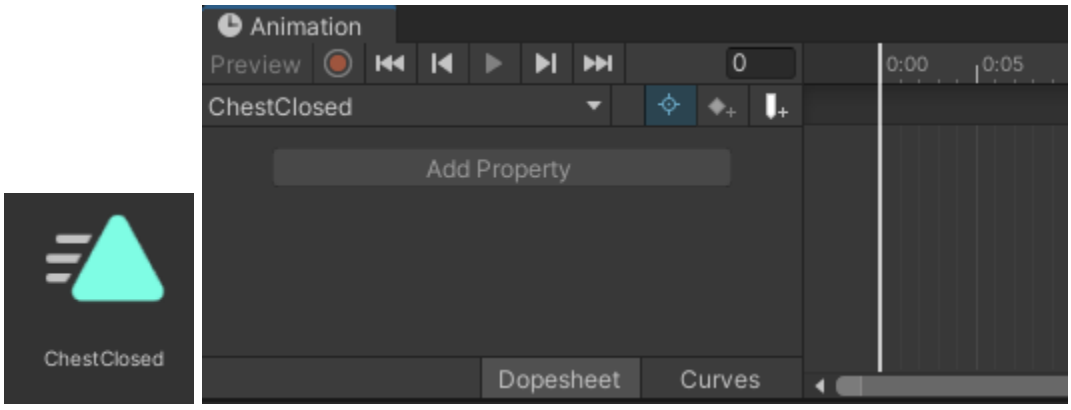
6. Right-click **ChestClosed** and click '**Make Transition**'



7. Click on **ChestOpened**

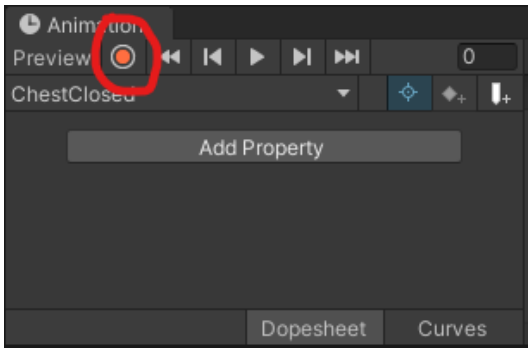


8. Go to your **Project** folder and double click the **ChestClosed** animation file. This will open an **Animation** window to make your animations.

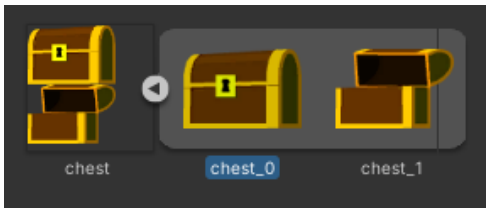


9. Click on your chest object in the **Hierarchy**.

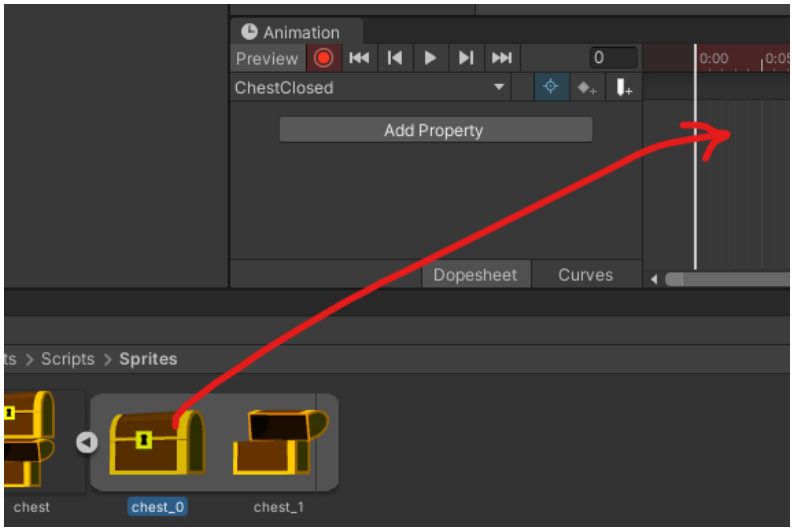
10. Click on the **record** button in the **Animation** window.

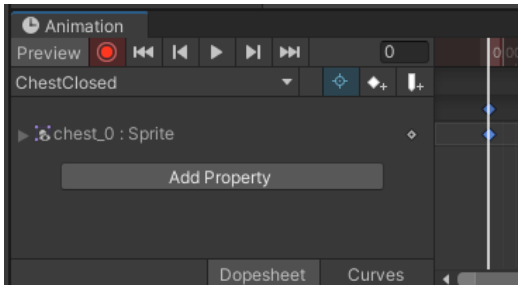


11. Go back to the folder with your chest sprites. Expand the sprite if haven't already.

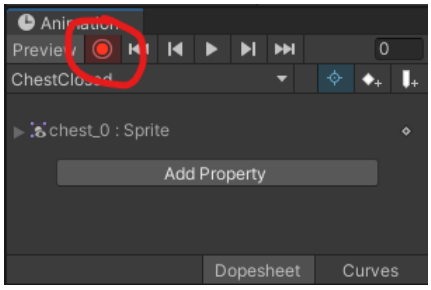


12. Click and drag the desired sprites into the **Animation** window.

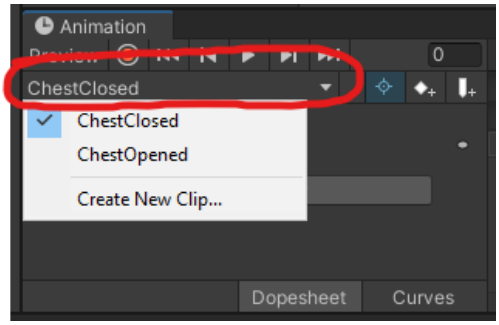


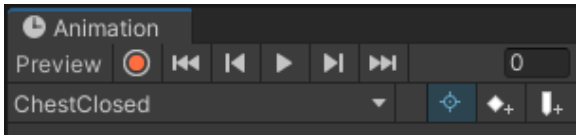


13. Click the **record** button again once you are happy with your animation.



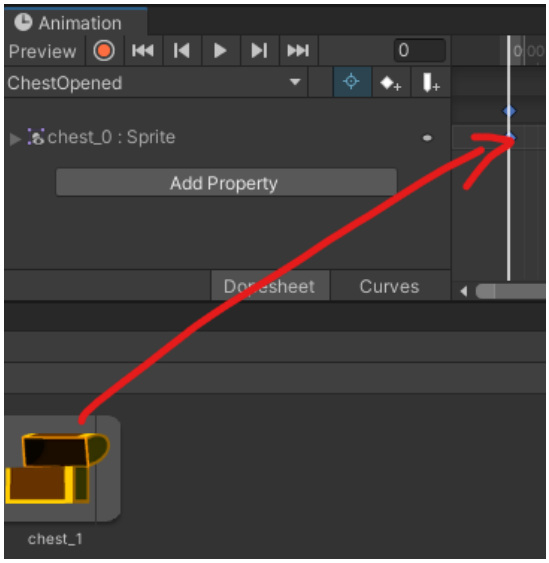
14. Click the dropbox and click **ChestOpened**.





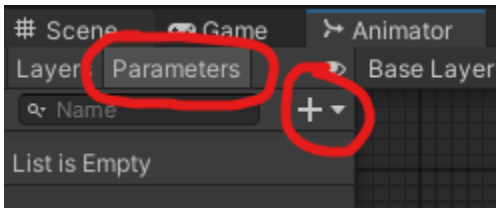
This will allow you to now edit the animations for ChestOpened for the Chest game object.

15. Repeat steps **10** to **13** for the **ChestOpened** animation file.

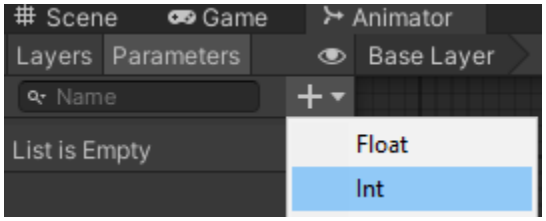


Transition Conditions - Chest

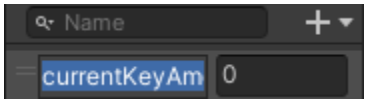
1. In the **Animator** window, click **Parameters > +,** to add a new condition.



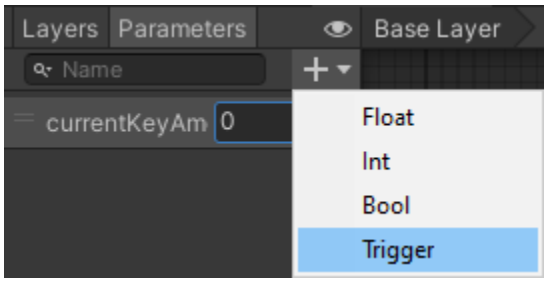
2. Click **Int**. This will make a condition based on an “integer” which is a whole number

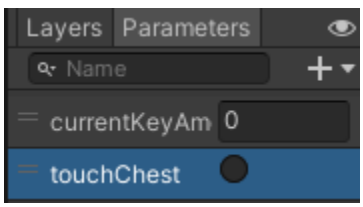


3. Name it **“currentKeyAmount”**.
    a. **This condition will be for the transition from ChestClosed to ChestOpened.**
    b. **This checks the number of keys the player has currently collected to see whether to transition between the animation files or not.**
    c. **The number will be 0 as the player will not have any keys at the start of the game**.

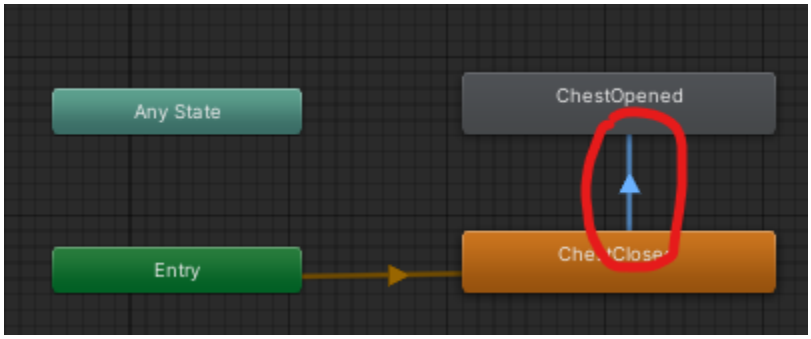


4. Make a new parameter based on triggers called **touchChest**. **+ > Trigger**.
    a. **This condition only activates if a collider “triggers” or activates it.**
        i. **In this case, we only want the animation to play when the player walks into/touches the chest.**

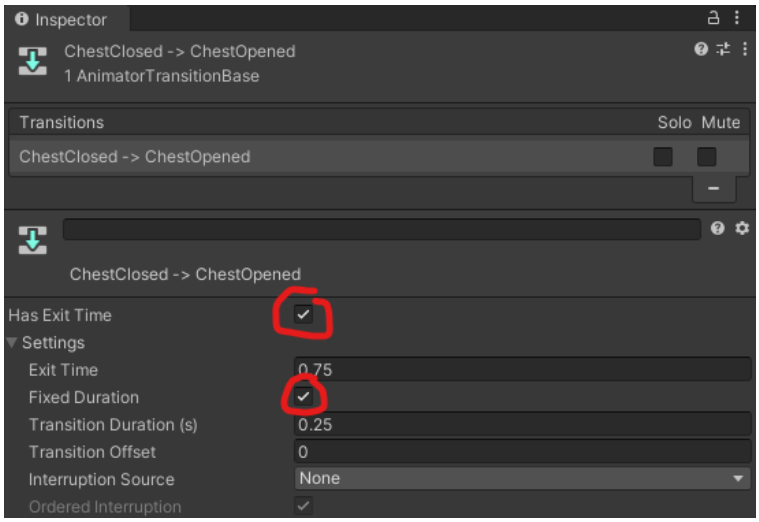


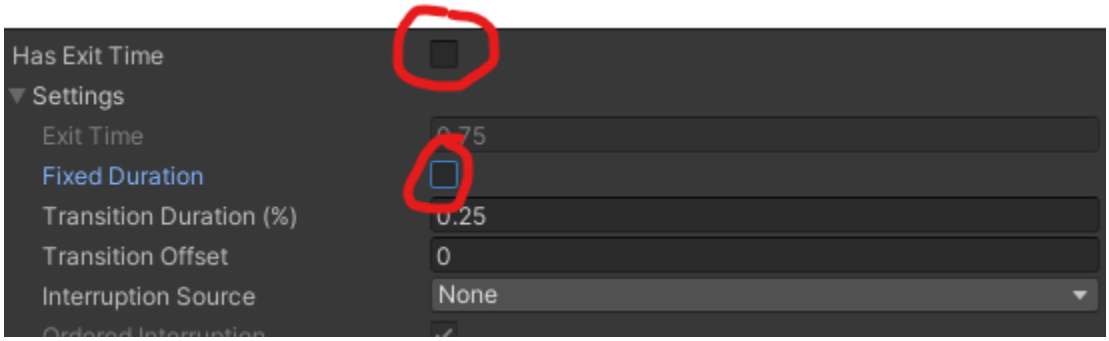


5. Click on the transition between **ChestClosed** and **ChestOpened**.

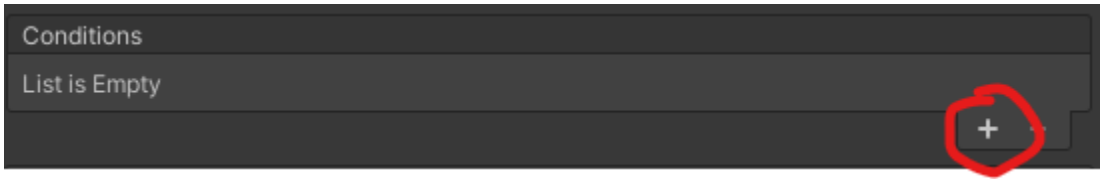


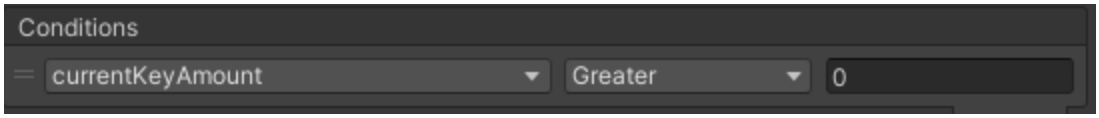
6. In the **Inspector**, expand **Settings** if not already expanded, and untick '**Has Exit Time'** and '**Fixed Duration**'.



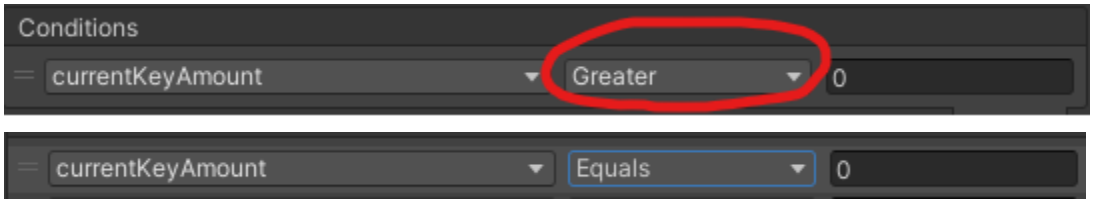


7. Below that, click **+** to add the condition we created to the transition.

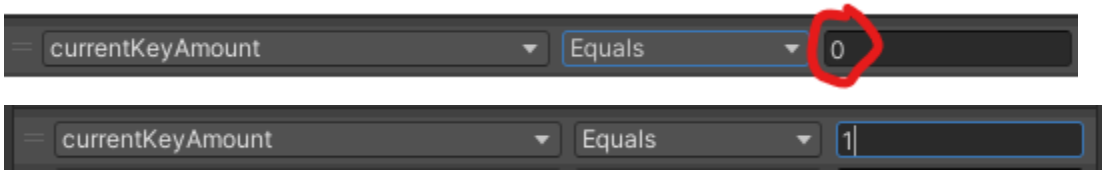




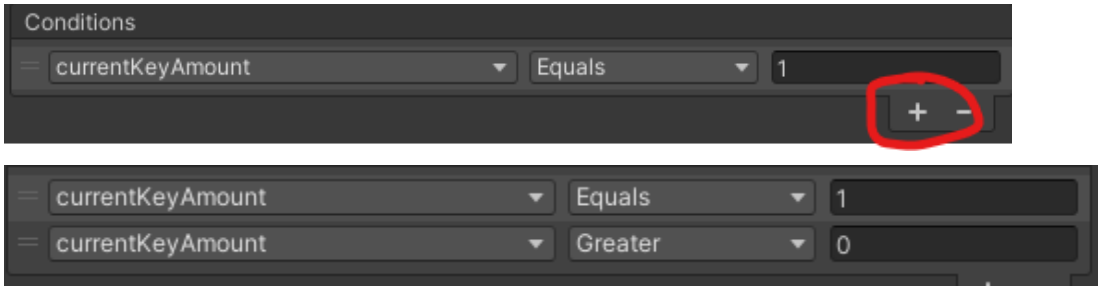
8. Change the **Greater** dropbox to **Equals**



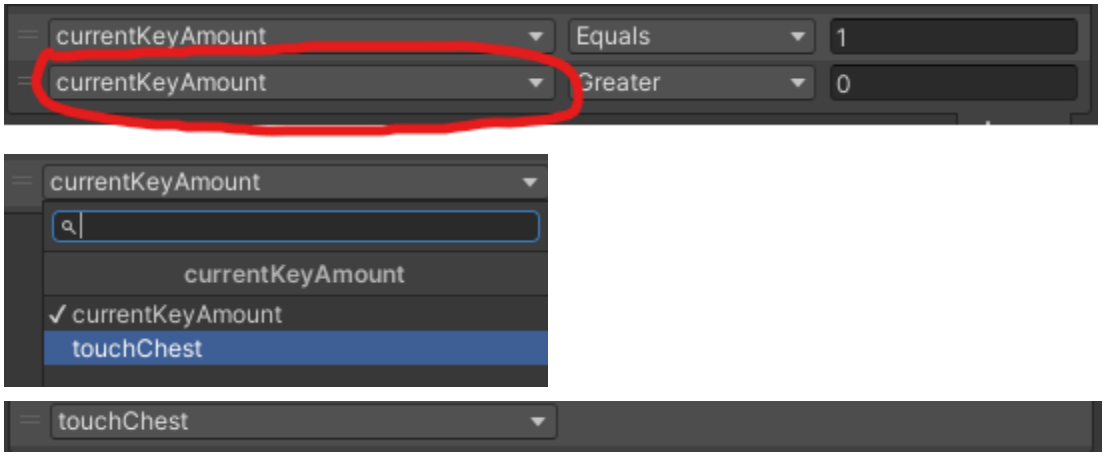
9. Change the value from **0** to the number of keys the player needs to collect in order for the chest to open.
    a. **In this case, the player will only need to collect 1 key for the chest animation to change from ChestClosed to ChestOpened**
    b. **If the current amount of keys the player has is less than 1 then the chest animation will not open.**
        i. **Feel free to change the value of the number of keys the player needs.**



10. Click **+** to add another condition.

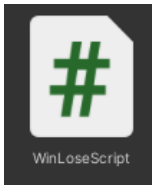


11. Change the first dropbox to the other condition we create, **touchChest**.

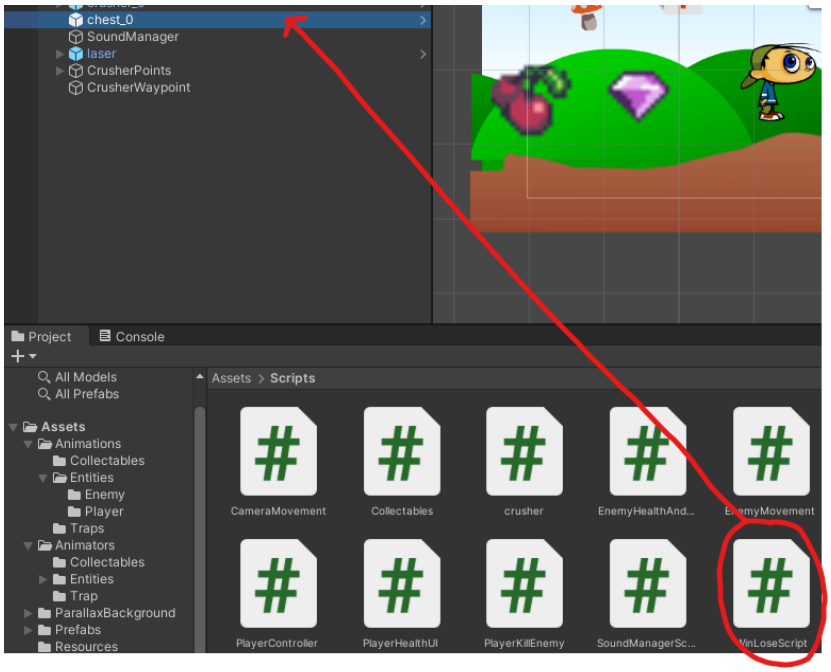


Attaching the Script

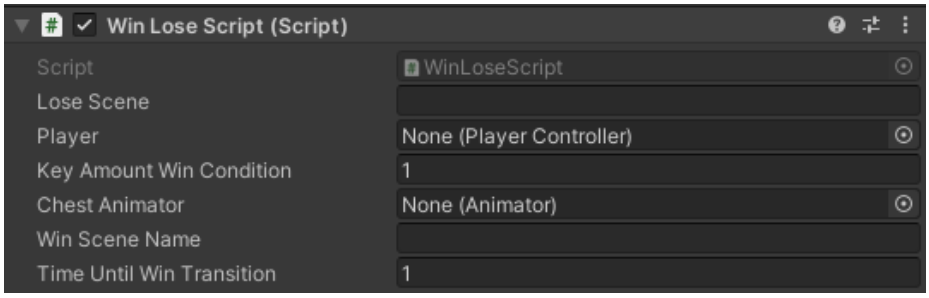
1. Go to where the **WinLoseScript** is located.



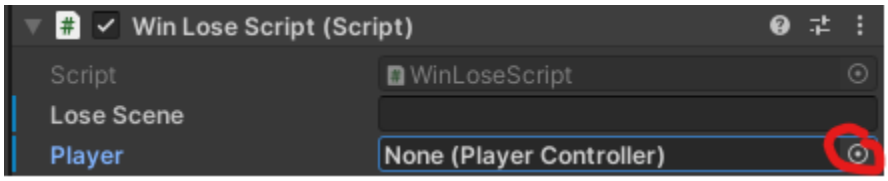
2. Drag and the **WinLoseScript** into your chest object.

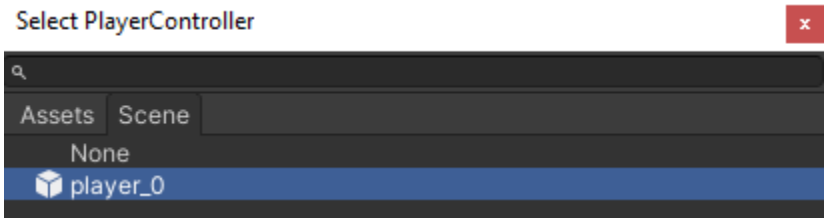


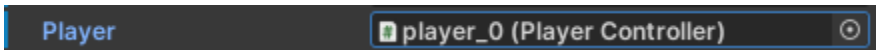
In the Inspector, it will look like this.



3. Click on the circle and choose the player game object.

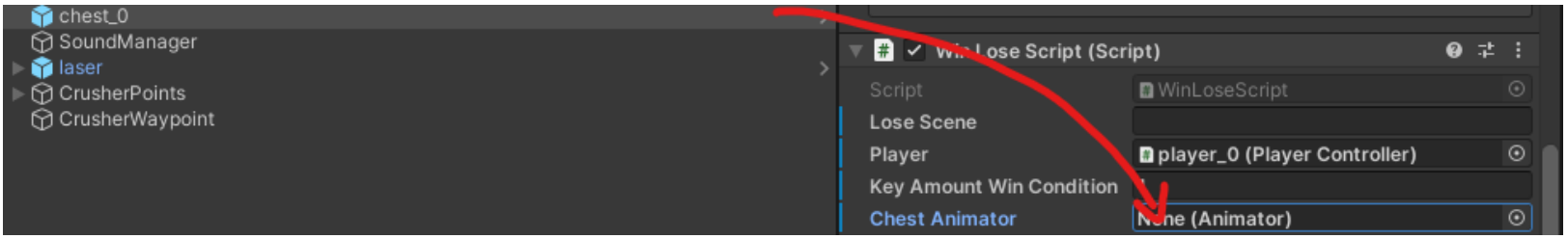




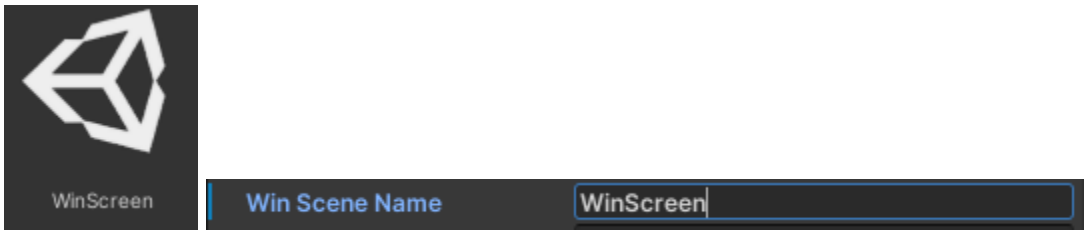


4. Type in the number of keys the player are required to collect in order for the chest to be opened

5. Drag the **chest** game object from the **Hierarchy** into the '**Chest Animator'** variable section.



6. Type in the name of the '**win**' scene file you want to transition to when the player has enough keys and opens the chest.



7. Type in the number of seconds the player needs to wait before the screen changes.
    a. **This is useful for chests that have longer animations.**

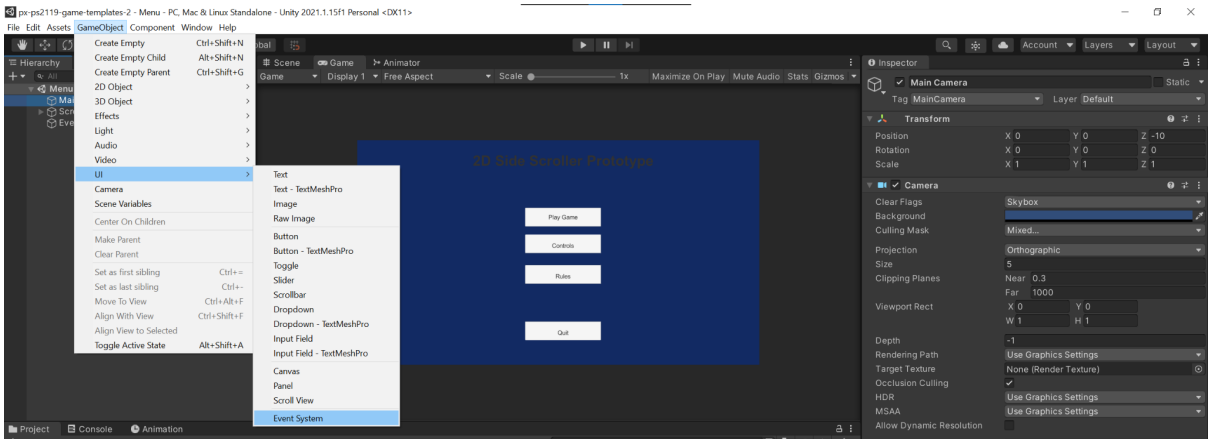
Menu Systems

Creating a Scene:

Right-clicking ‘Scenes’ found in workspace window Project under ‘Assets’ folder, there is the option ‘create’ upon expanding that, click ‘Scene’.

Enter a title for that scene.

The Scene will have an initial item called “Main Camera” no modifications are required.

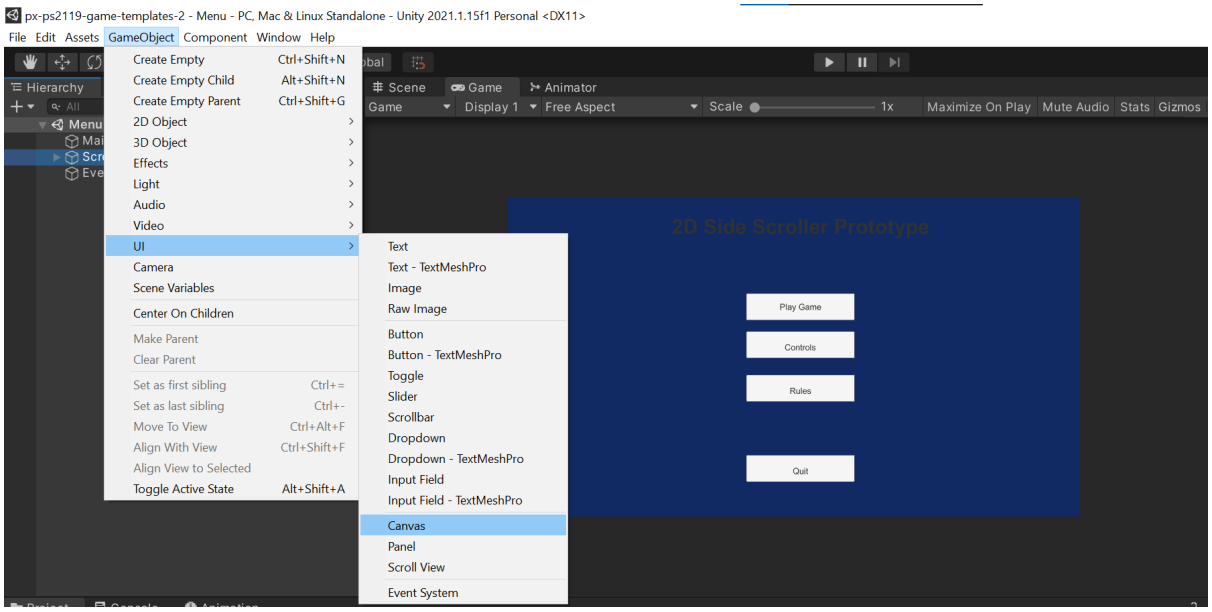


Adding Event System if its not in the scene already

EventSystem allows the processing of inputs via keyboard or mouse. Without it, clicking on a button will result in nothing happening.

Creating a Basic Menu:

Now that the scene and its EventManager a Menu can be created.

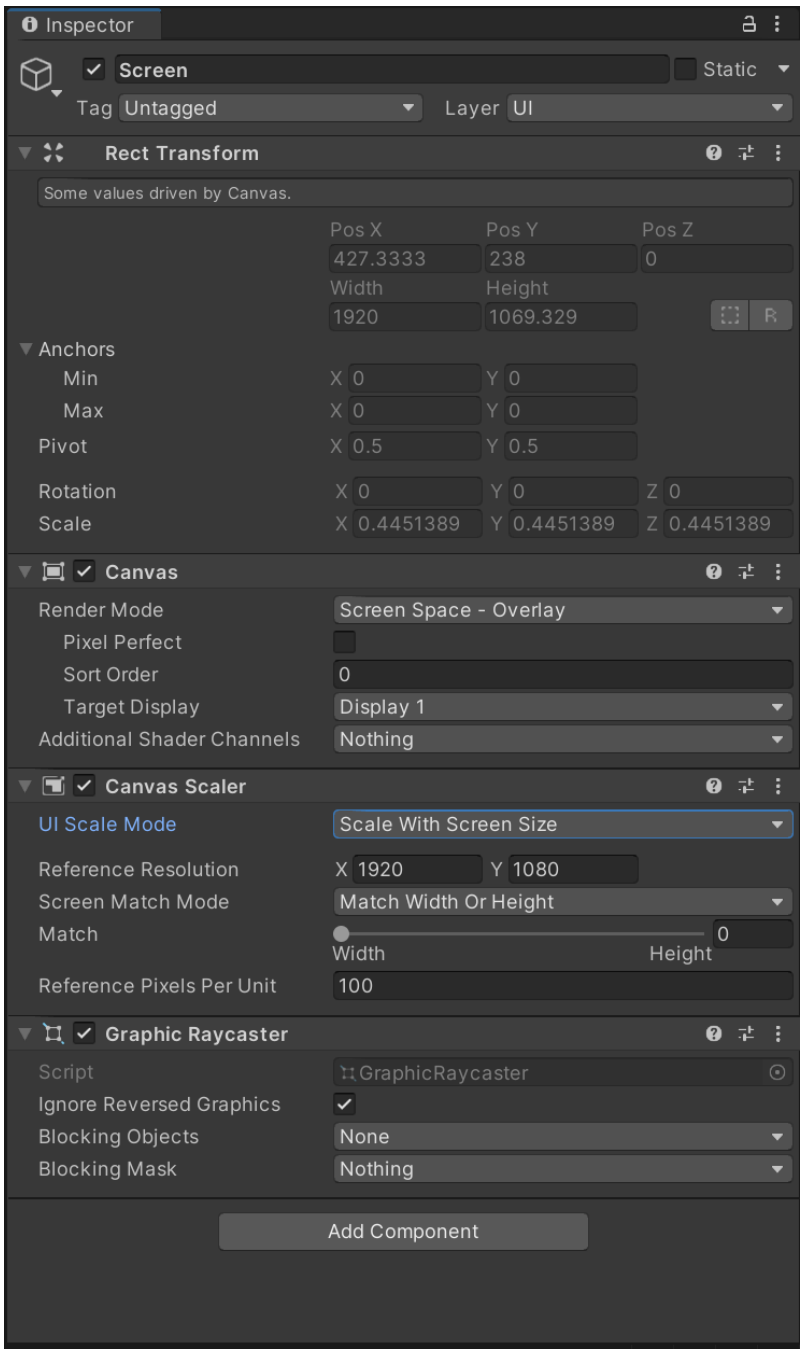


Adding a Canvas

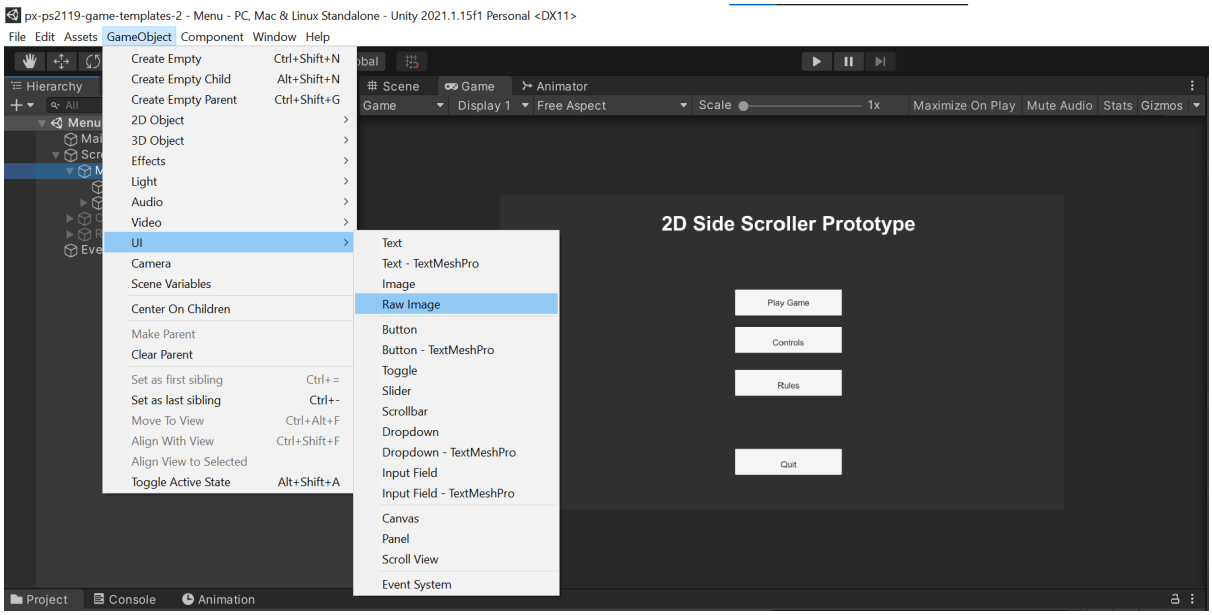
Canvas is used as a holder to make the design and adjustments of assets easier. In simpler terms it is a folder in terms of Adobe editing software.

Self-scaling Screens:

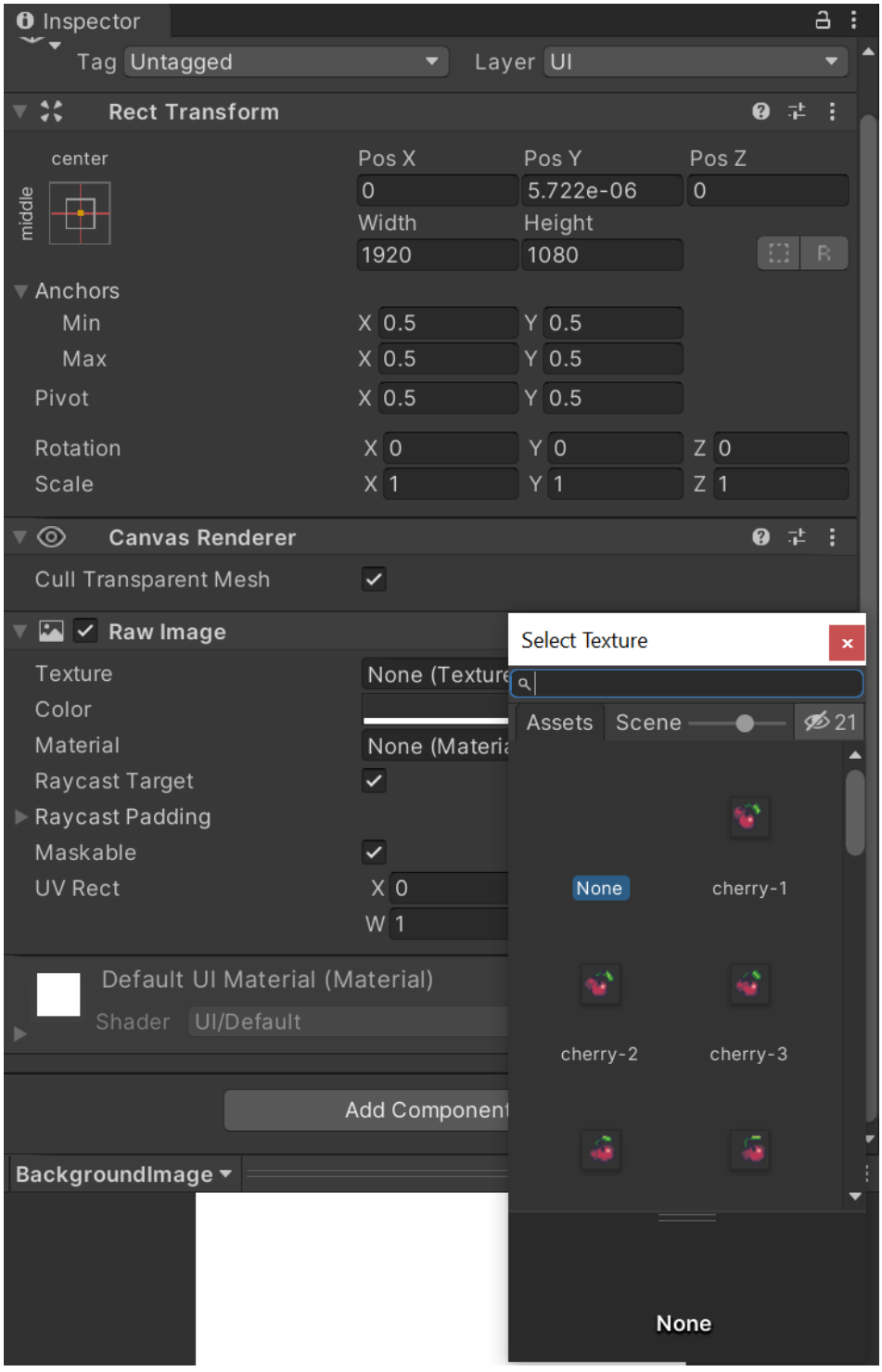
Once our canvas is added, in order to allow the game to run clearly on different screen sizes we will set the “UI Scale Mode” in “Canvas Scaler” from Constant Pixel Size to Scale With Screen Size with the Reference Resolution set to X:1920 and Y: 1080.



Adding the background:



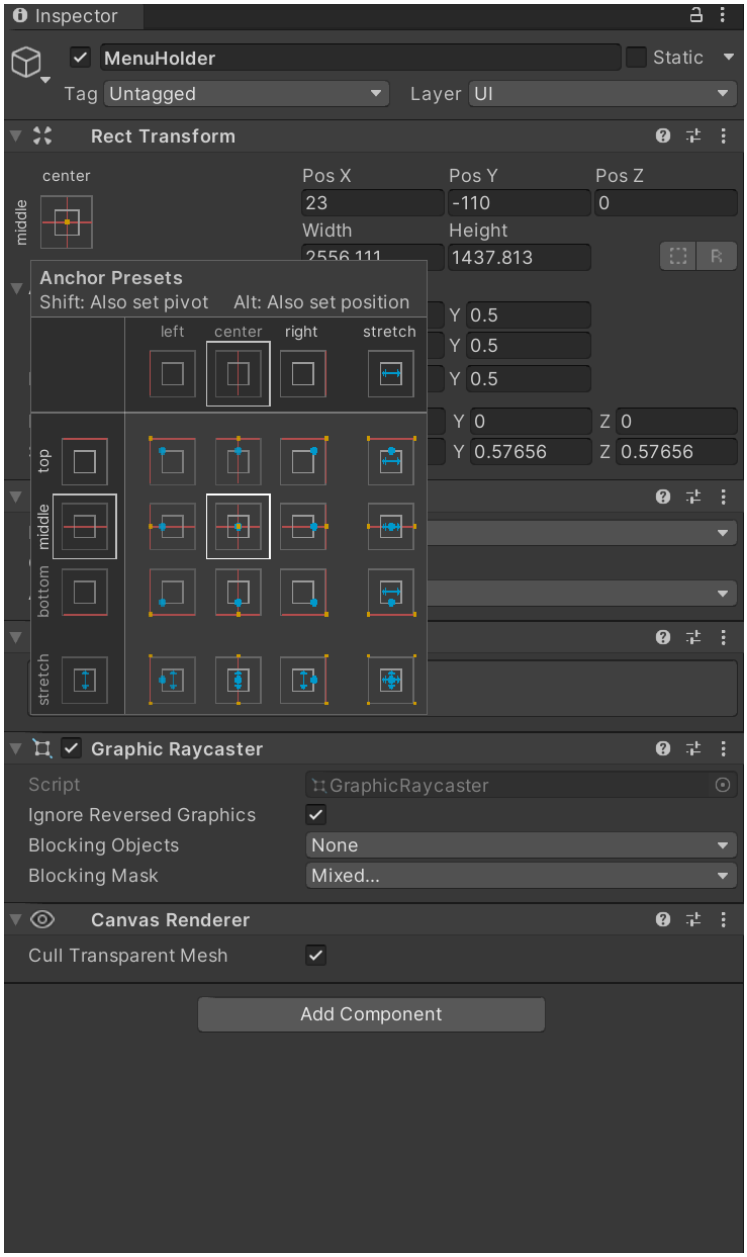
Using the same method as the previous this time select Raw Image instead of Event System. The difference between Raw Image and Image is rendering speeds thus any can be used. Name the Image/Raw Image and to change the image



Texture is used. The images imported in the project can only be used therefore, dragging and dropping the desired image first is necessary.

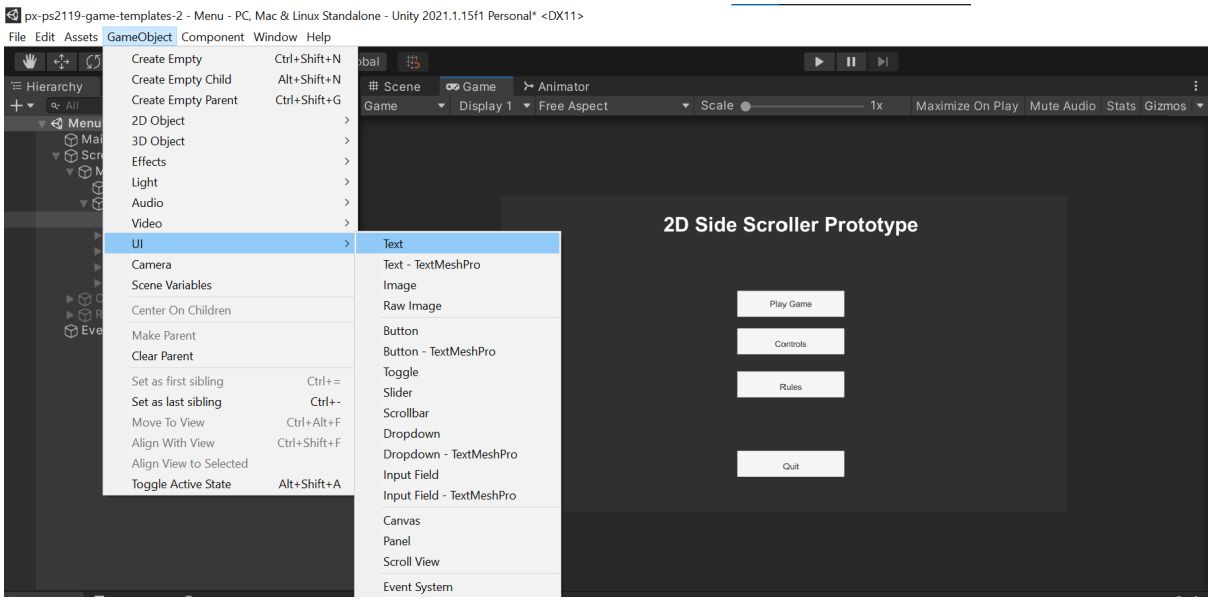
Secondary Canvas:

Another canvas is created, this is for the title and buttons that the menu will have. Once the canvas is added make sure the transformation is centered based.



This allows for when the screen is scaled it will align it by the center.

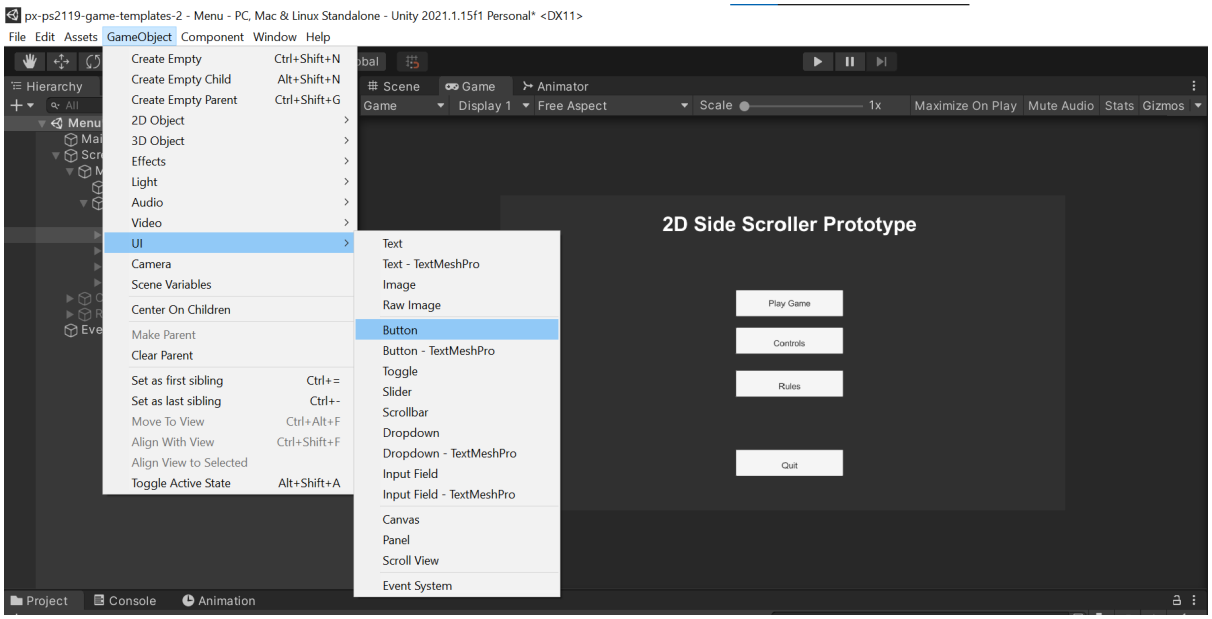
Game Title:



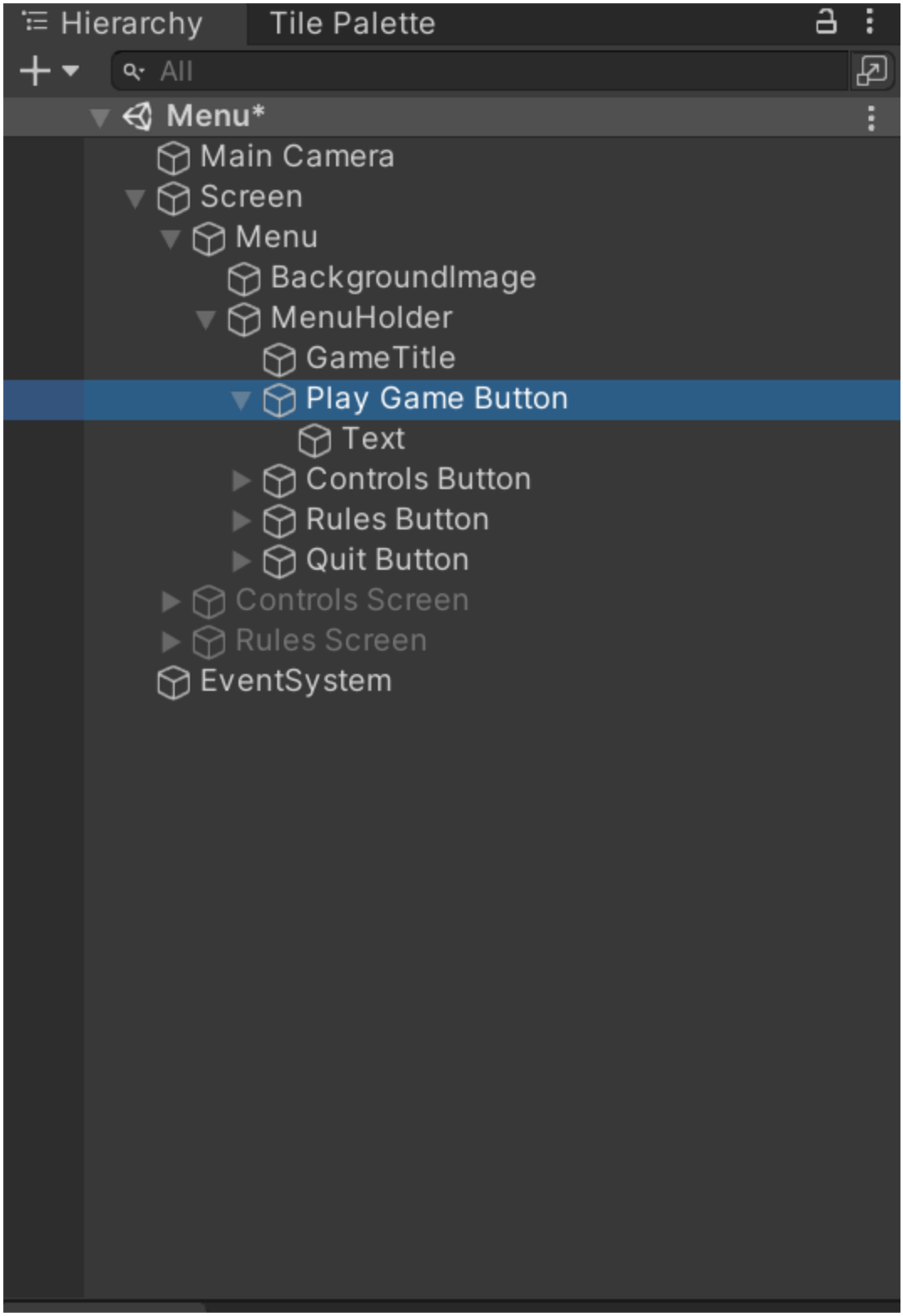
Next we create a Text GameObject called Title.
To edit the text format and words navigate to Text in the Inspector Window.

Note that the Title is also top aligned.

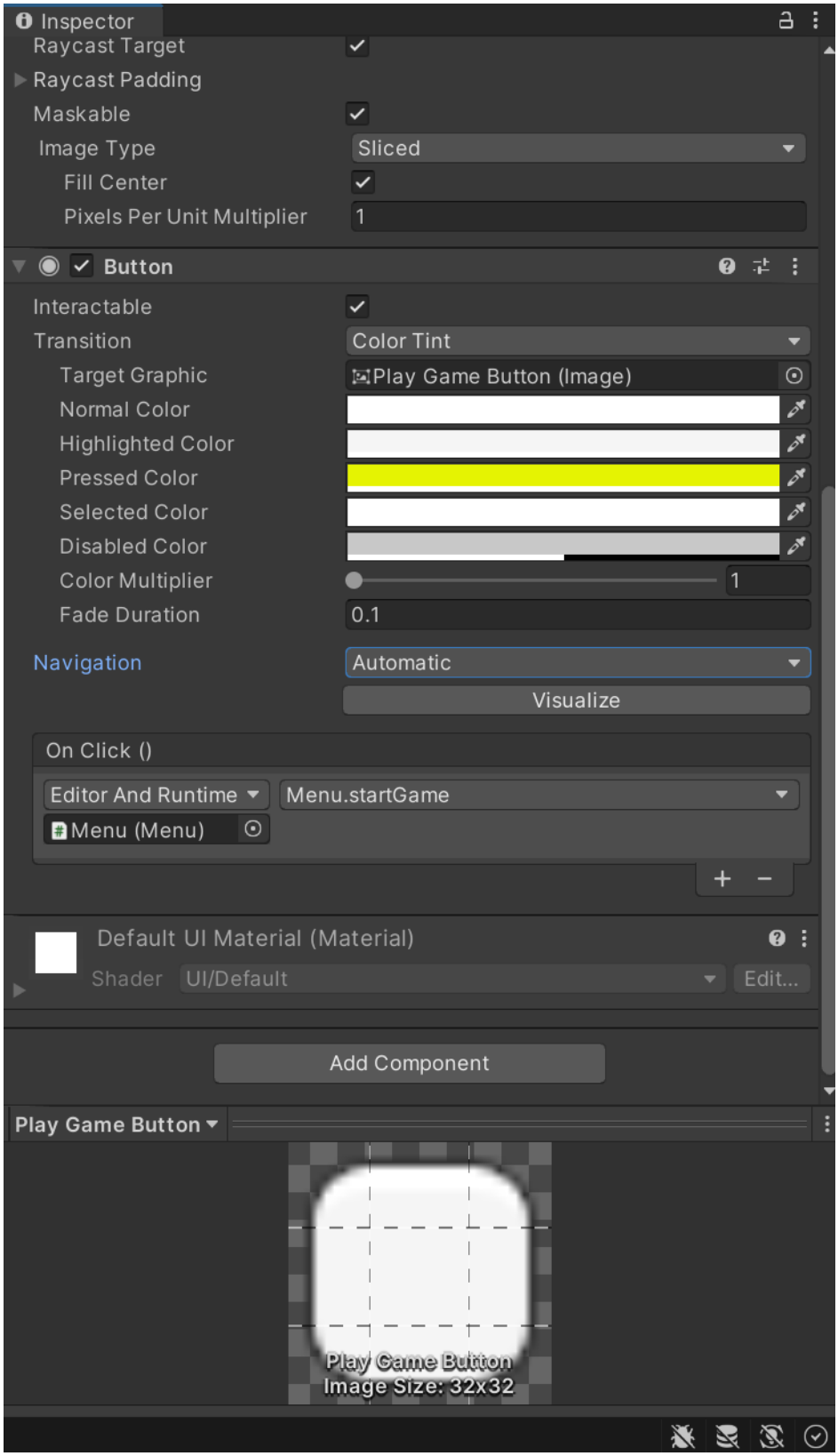
Adding Buttons:



To add a button simply follow as above in the screenshot. When a button is added there will an arrow next to it named Text by default.



Editing the text of the button is much like the title. Make sure to click on Text and not the button.



The Button subheading in inspector allows the option to change the colors.
OnClick is the handler when the button is clicked on, this will be expanded later on.

Repeat this step for how many buttons are needed.

Menu:

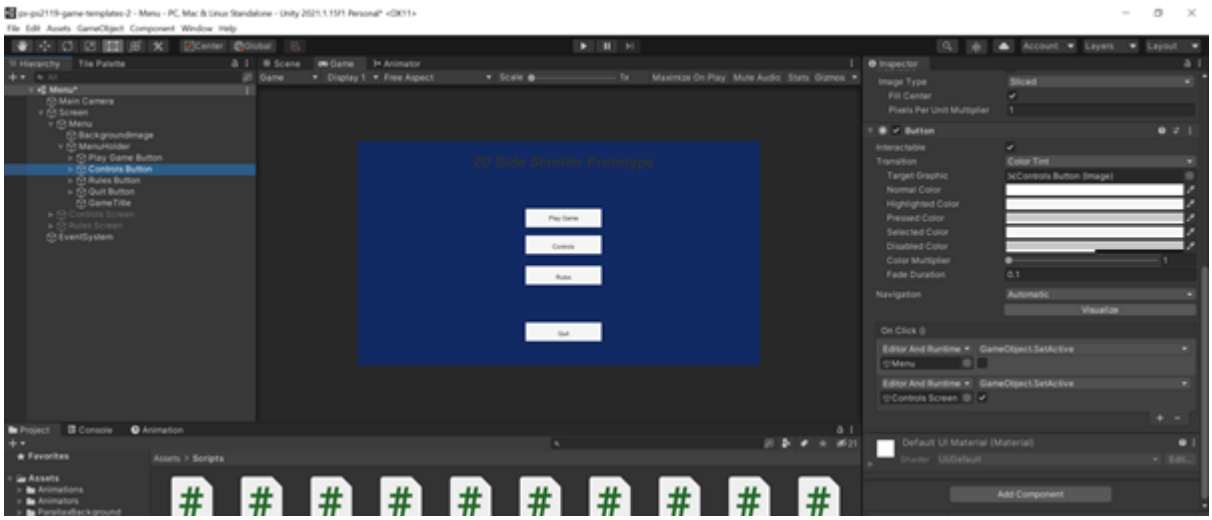
Scripts involved:

- Menu.cs

Transitioning to another ‘Canvas’

To transition from one canvas to another, requires **no code** whatsoever. Because this is achieved by using GameObject.SetActive(), alternatively this can also be used as code.

No code Option:

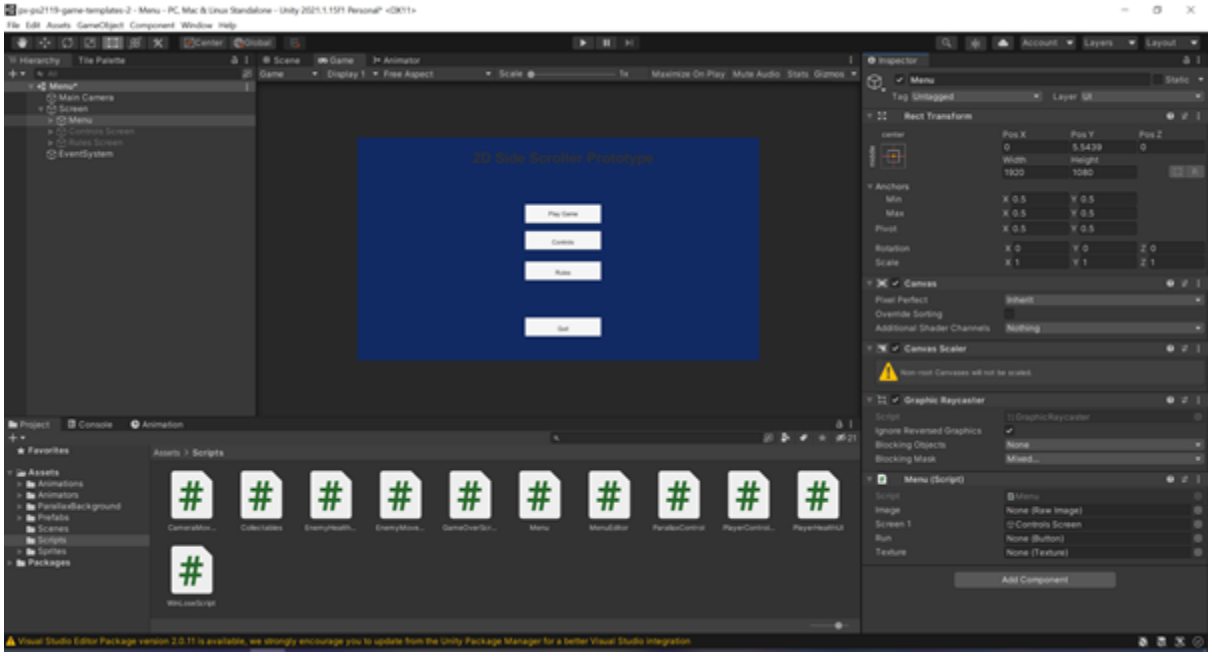


Code Option:

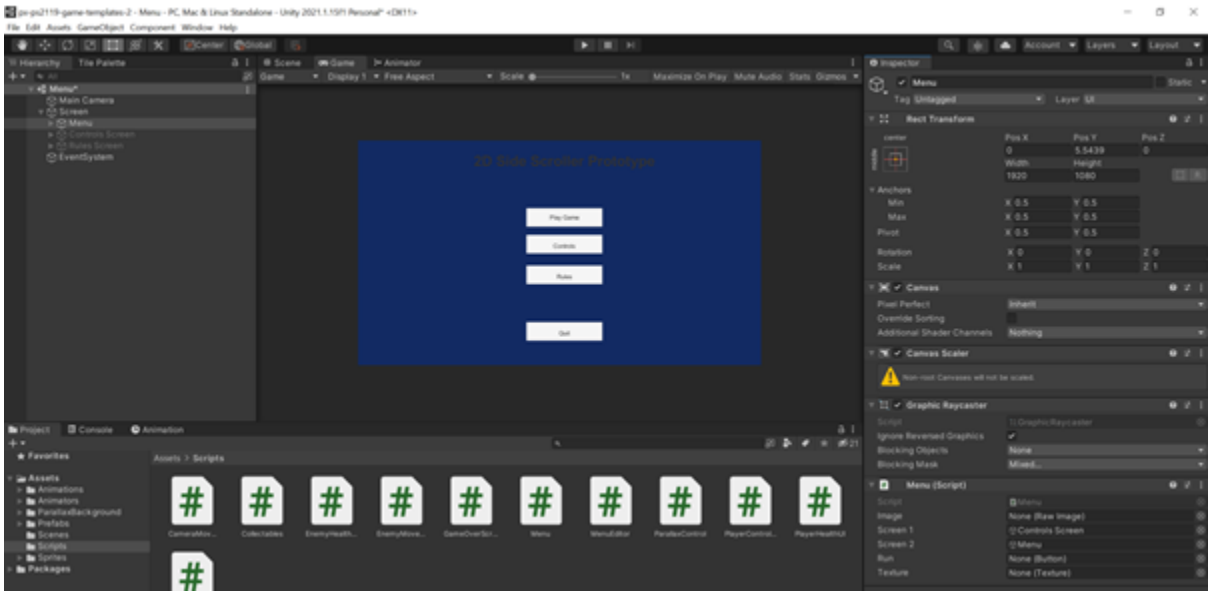
When referencing the file Menu.cs (found in scripts folder) and then adding the code:

```
[SerializeField] public GameObject screen1;
```

Then dragging the desired screen into that asset:



Another is created to have the current screen:



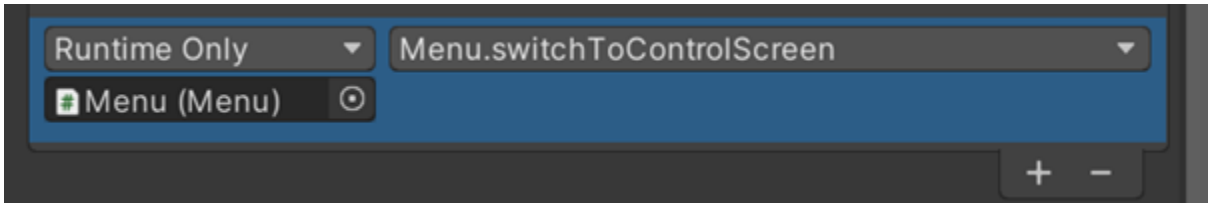
Following that a function will need to created i.e.,:

```
public void switchToControlScreen()
{
   screen2.SetActive(false);
   screen1.SetActive(true);
}
```

The idea of SetActive is visibility, by turning one's visibility off and on it allows a transition. Note this only works when they are in the same scene. Another method is required when using different scenes. (See **Start Game**).

Then we assign that script to the root canvas.

By clicking "Add component" Script->Menu to Menu then on the desired button OnClick click on the plus icon and drag Menu under the option Runtime Only, after that click on the option right to Runtime only. Navigate to menu and find the function switchToControlScreen.



Note when assigning function to buttons this part is always used.

Starting Game:

In Menu.cs there is a function called start game:

```
public void startGame()
 {
        SceneManager.LoadScene("SampleScene");
        }
```

The idea behind this is that it loads a Scene whose name matches the one in quotation marks. In this case “SampleScene”. Further in the document there will be another way to allow better user experience.

Quitting game:

Is very self explanatory.

```
public void quitGame()

  {

        Application.Quit();
        }
```

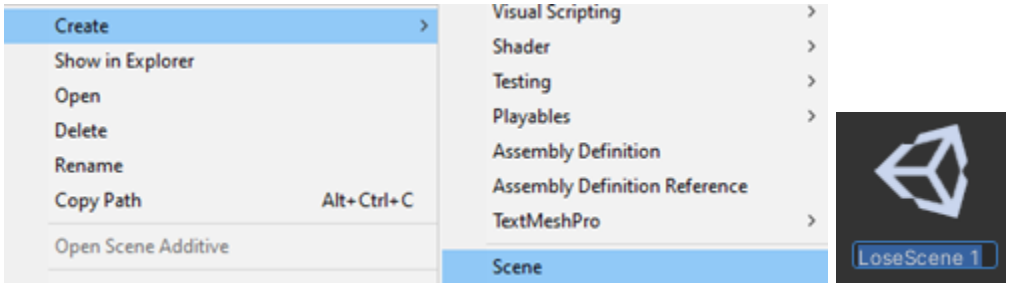
Lose Screen

Overview

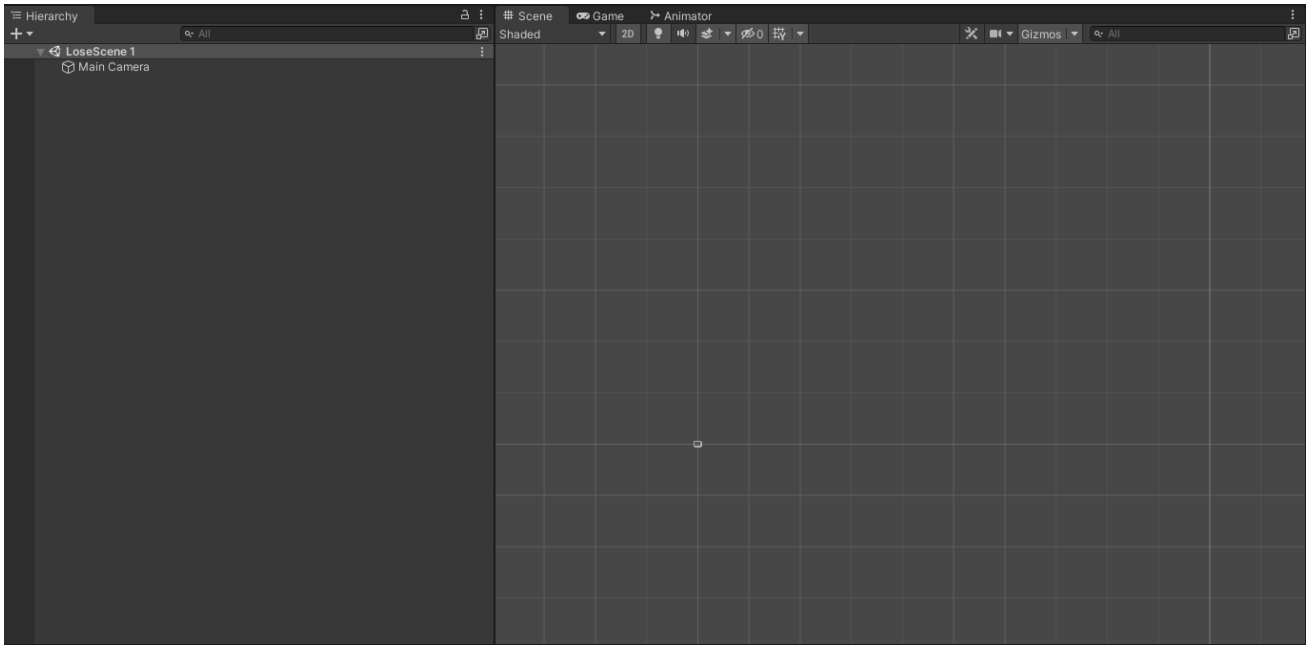
The scene that is shown the player loses all their lives and loses the game. The level screen will transition to a lose screen. You can have as many lose screens as you want.

Creating a New Scene - Lose Scene

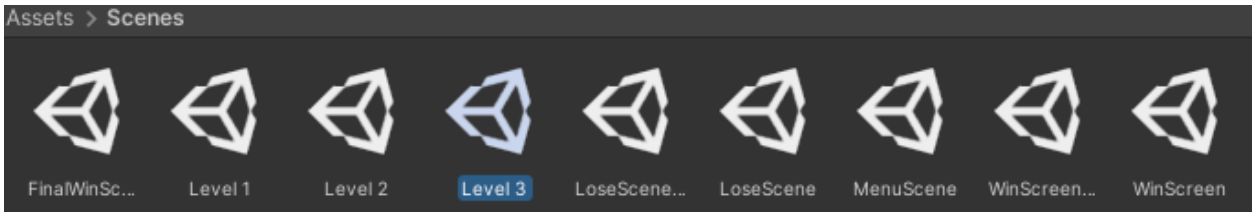
Right-click the **Scenes** folder in your Project file and create a new Scene. **Create > Scene**. Give it a name



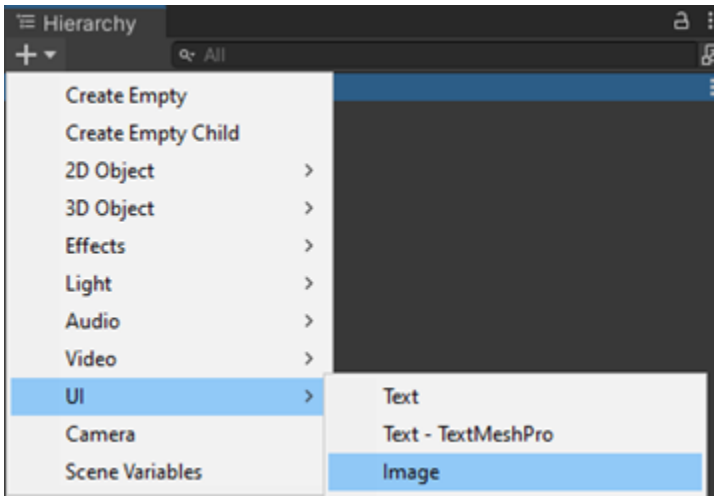
To edit the Scene, double click it and a new empty scene will open.



To go back, go into your Scenes folder and double click a scene.



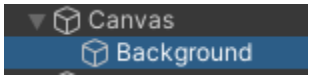
Create a new **Image** game object. **+ > UI > Image**



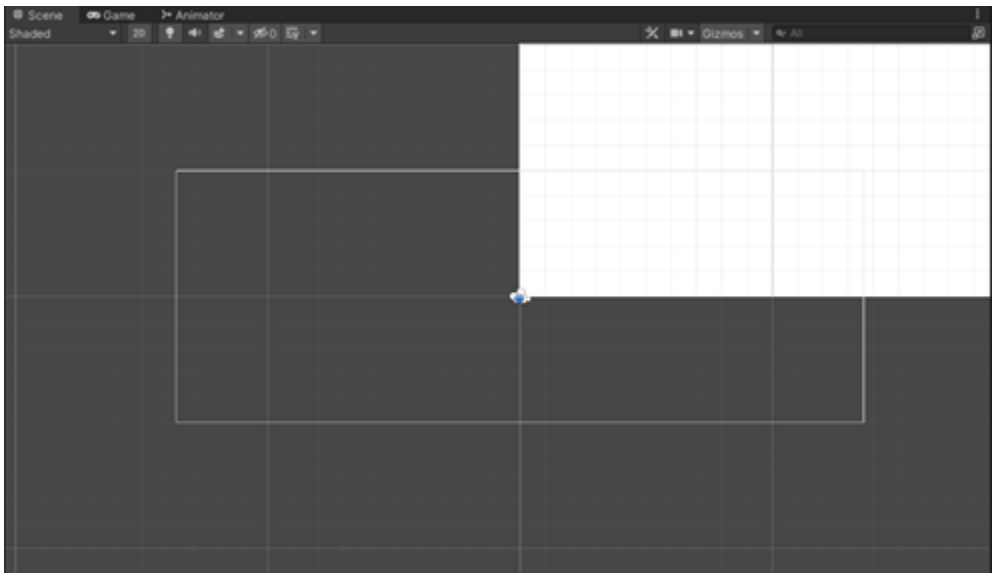
Name it **Background**.

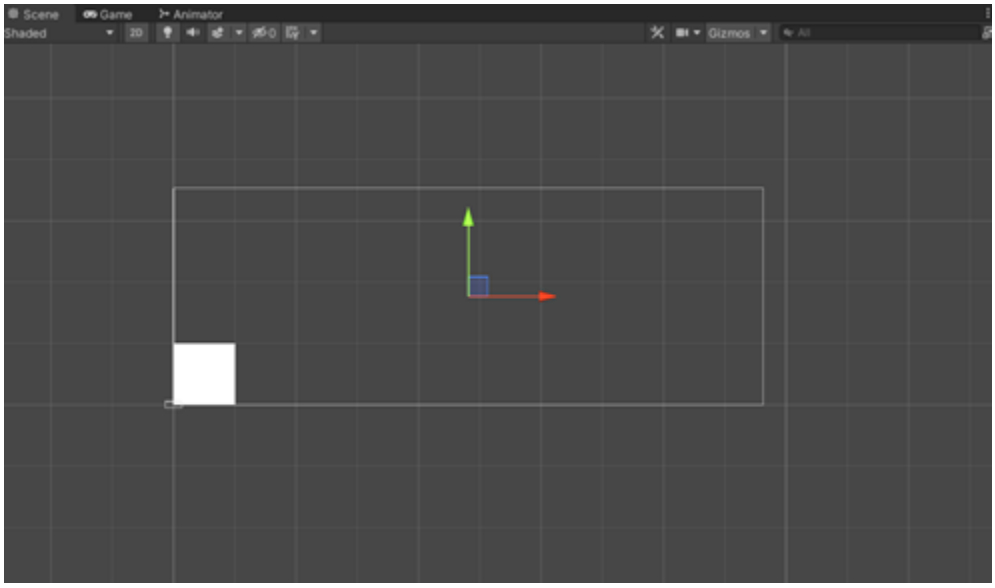
Note - Background in **LoseScene** is not related to any other game objects with the same name in other scenes

E.g. Background in '**Level 1'** is not the same as and will not inflict with **Background** in '**Level 2**'



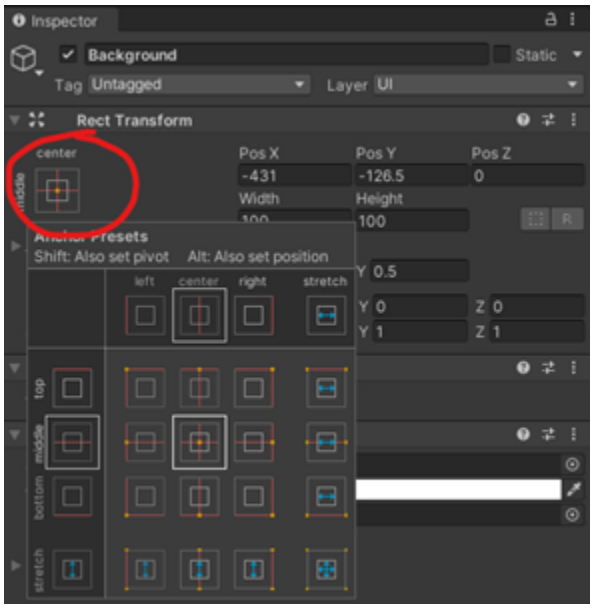
Click on the **Canvas** game object, hover your mouse over the **Scene** window and press **F** on your keyboard. **This will quickly zoom in and focus on the object in the Scene**





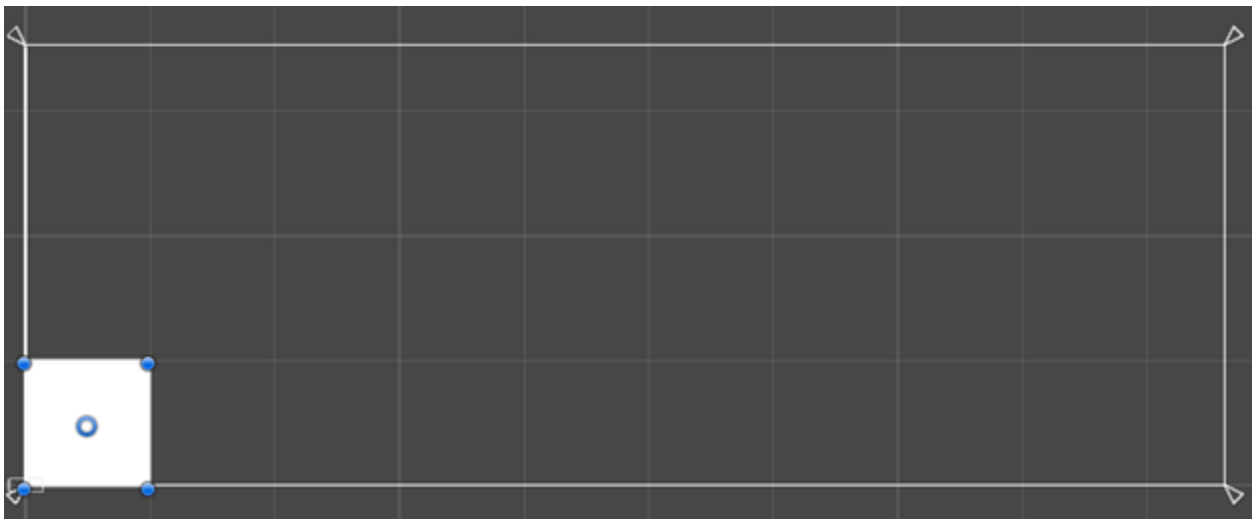
Click the **Background** object in the **Hierarchy**

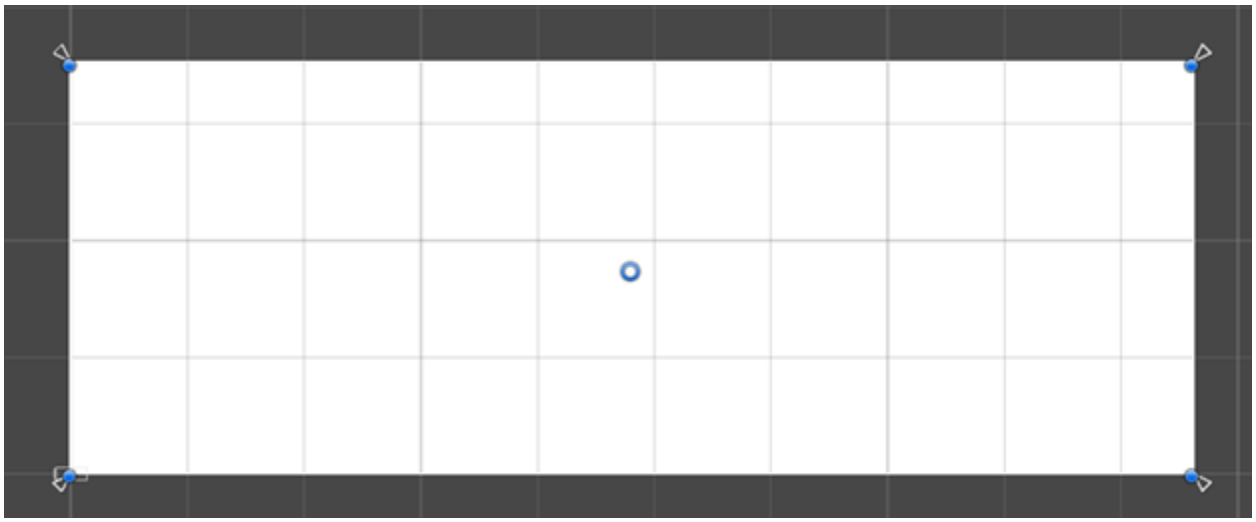
In the Inspector, click the **Anchor Presets** image to change the Rect size of the game object



Hold **Alt** on your keyboard and click the bottom right icon in the **Anchor Presets** dropdown window

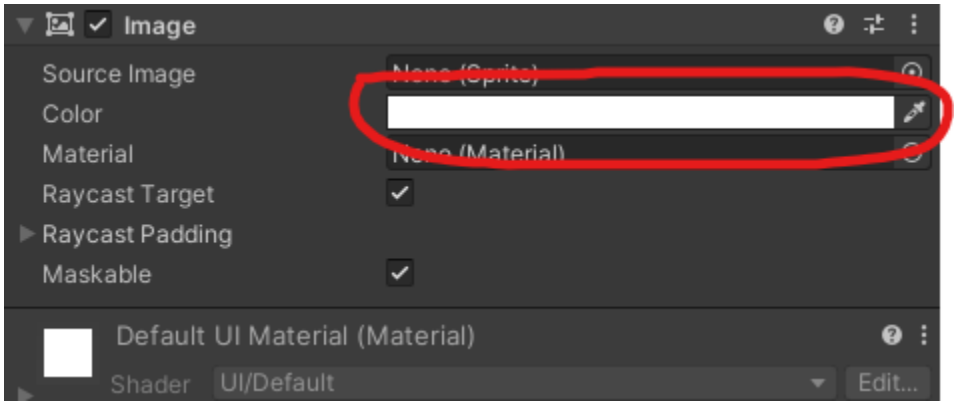
This will increase the size of the Background game object to the size of its parent object. In this case it will increase the size to be the same as Canvas

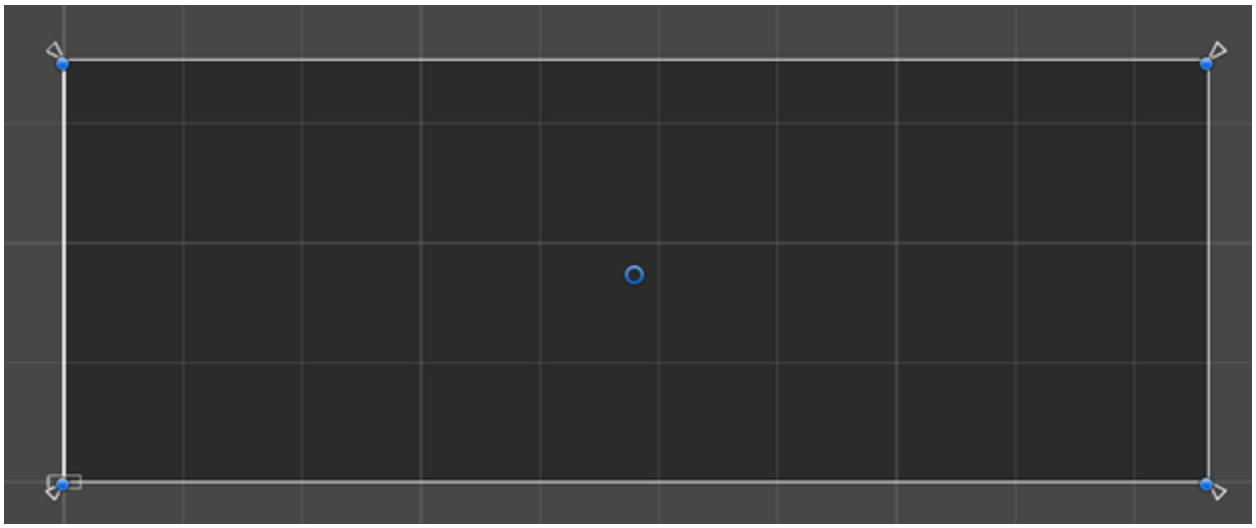




Change the colour to one that suits your liking **OR** place an image.

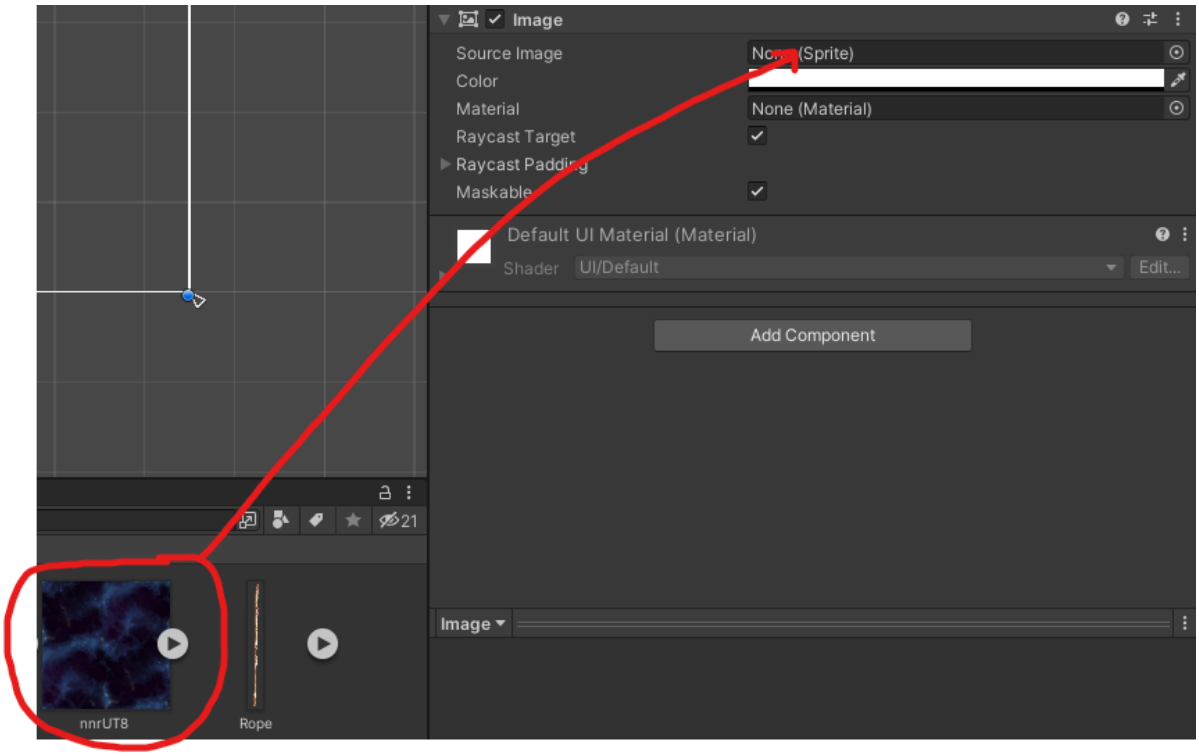
Change the Colour

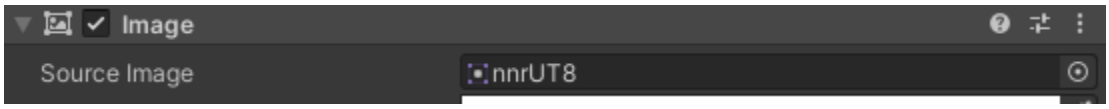


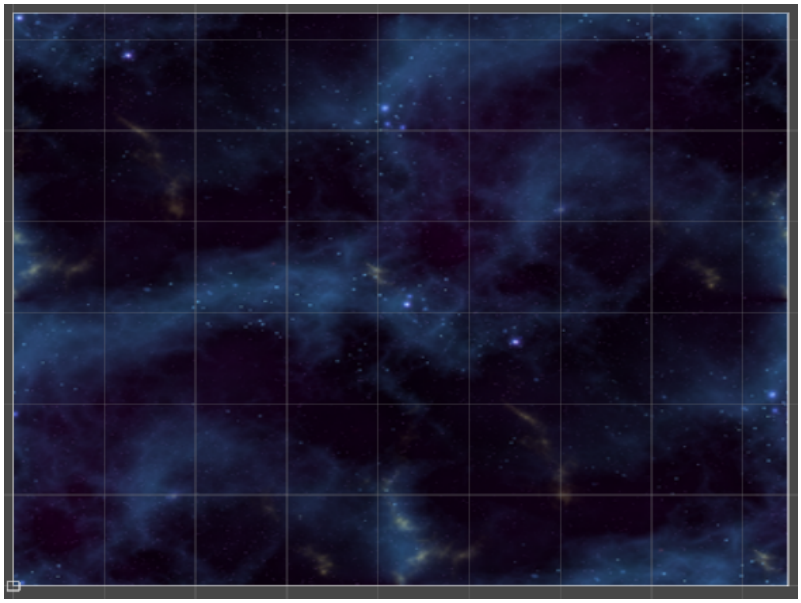


Change the Image

Drag the desired image sprite from your **Project** window into **Source Image** of the **Background** game object **Inspector**.



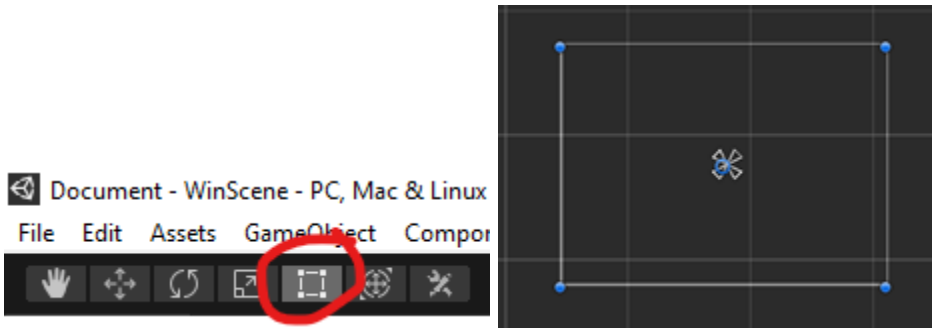




Adding the Lose Text Objects

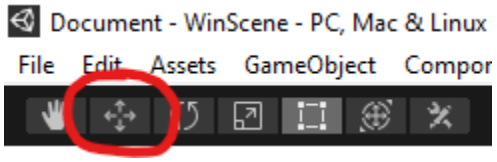
Create a new empty game object as a child of **Canvas** and name it **LoseMessage**. Right-click **Canvas > Create Empty**.

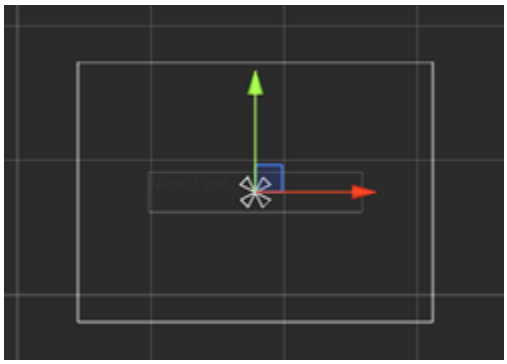
Resize using the **Rect tool**

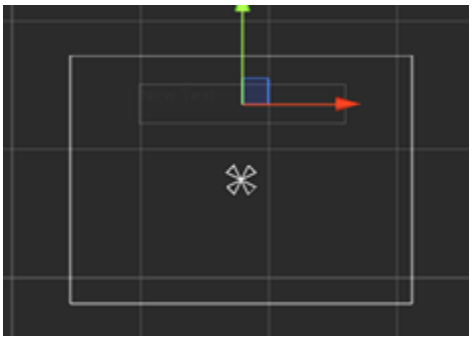


Create a **Text** object as a child of **LoseMessage**. Right-click **LoseMessage > UI > Text**. Give it a name. E.g. **Score.**

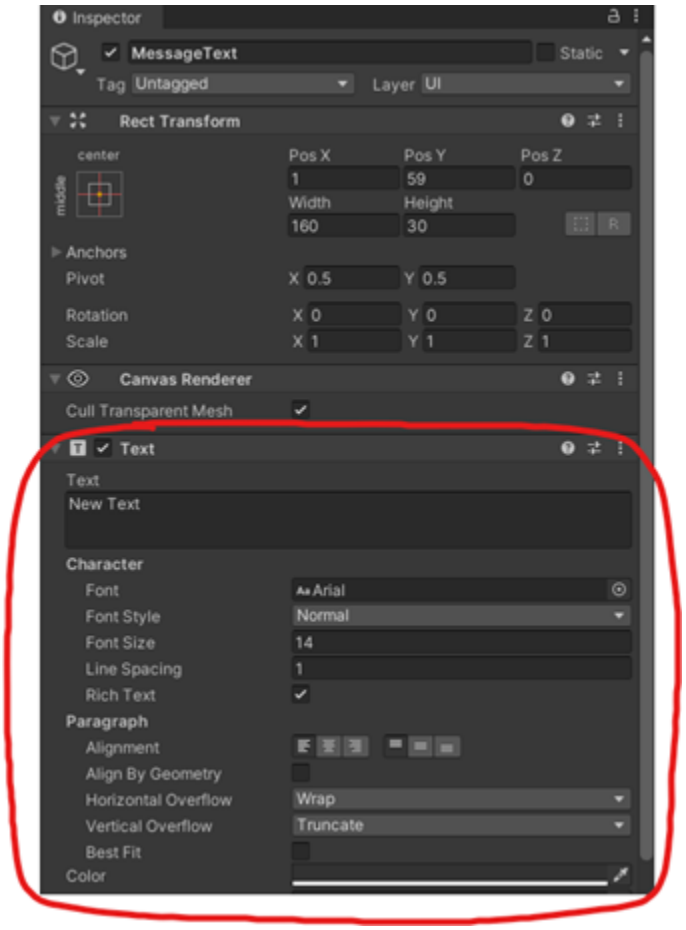
Click on the **Move tool** to adjust the position of the game object

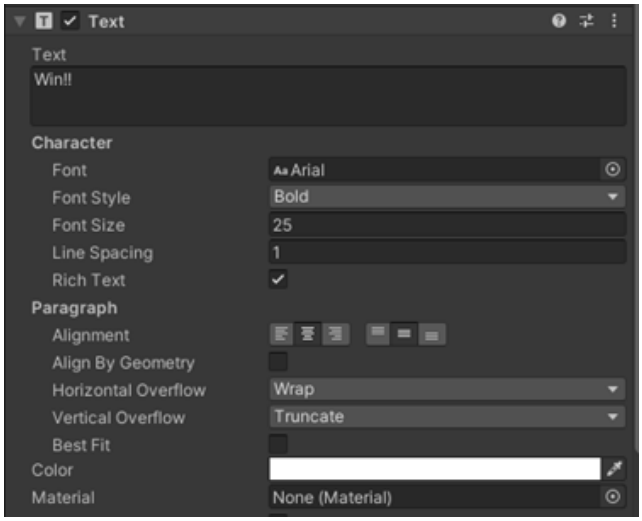


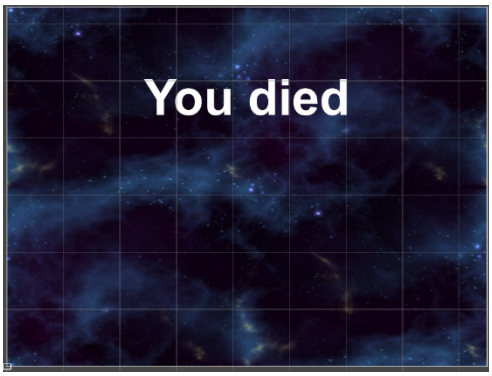




In the **Inspector** - **Text** component, customise the **Text** to your liking



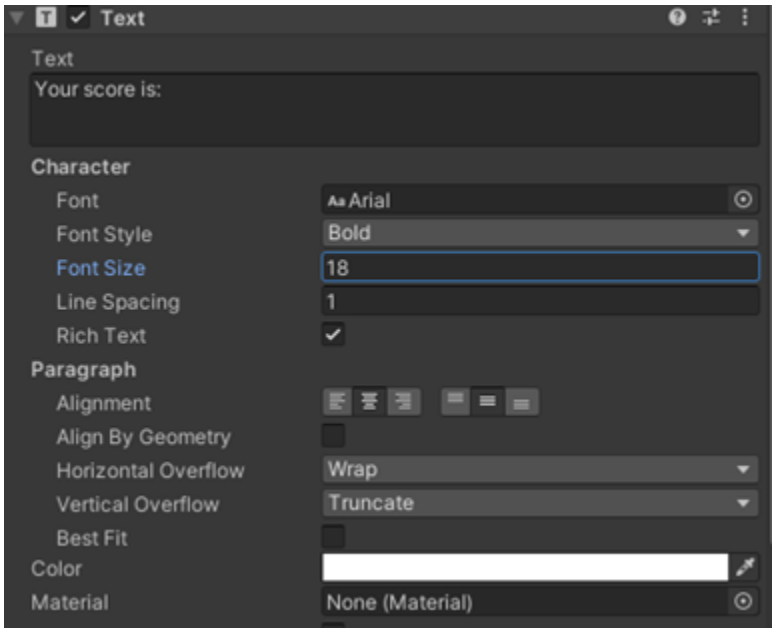


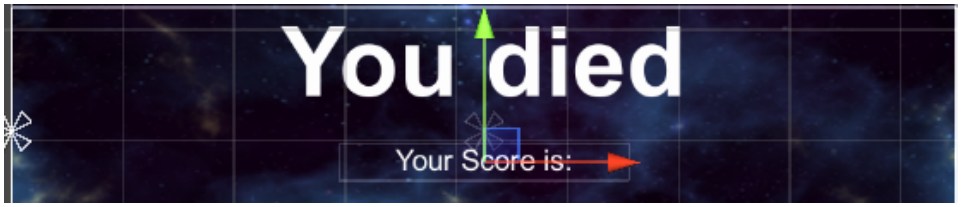


Create a new **Text** object as a child of **WinMessage**. Right-click **WinMessage > UI > Text**. Give it a name

Adjust the position with the **Move tool**

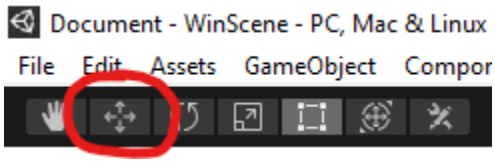
Change the **Text** component settings

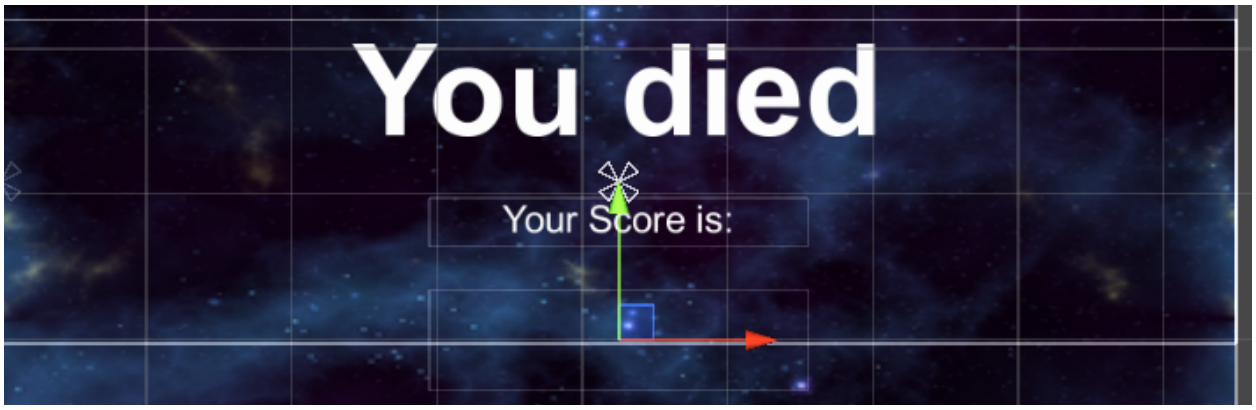




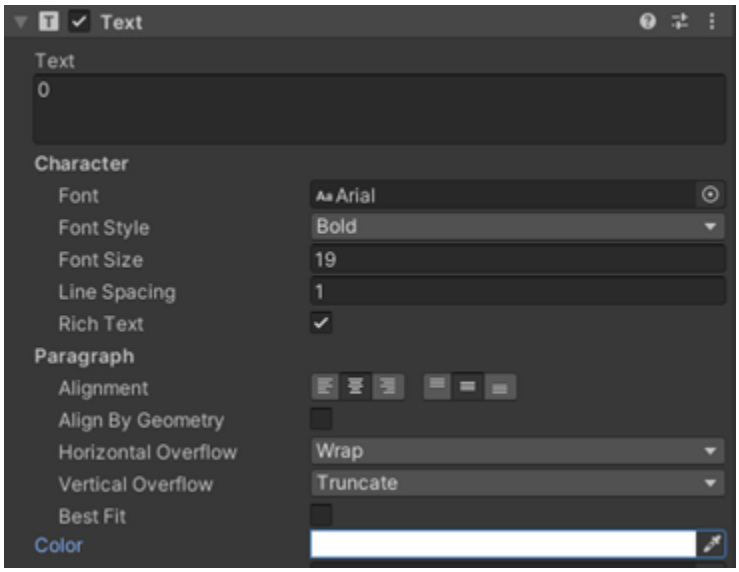
Create a new **Text** object as a child of **WinMessage**. Right-click **WinMessage > UI > Text**. Name it **Score.**

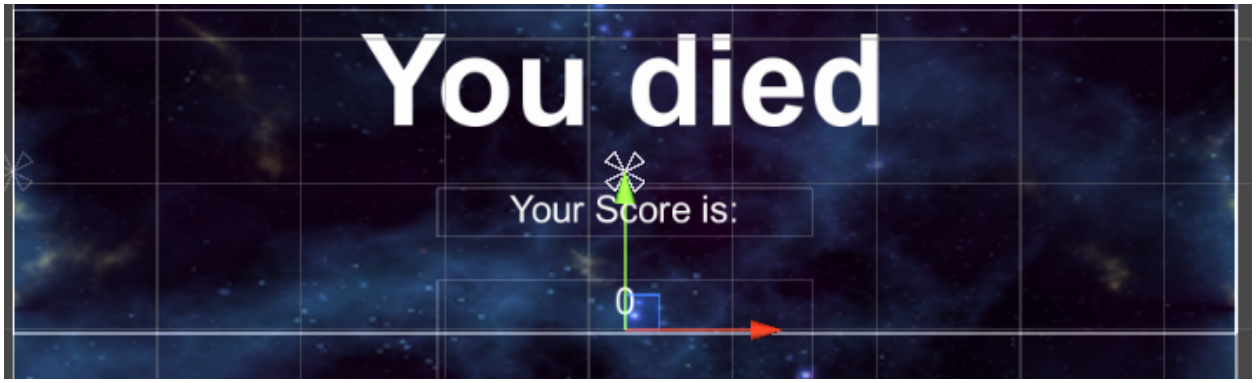
Adjust the position with the **Move tool**





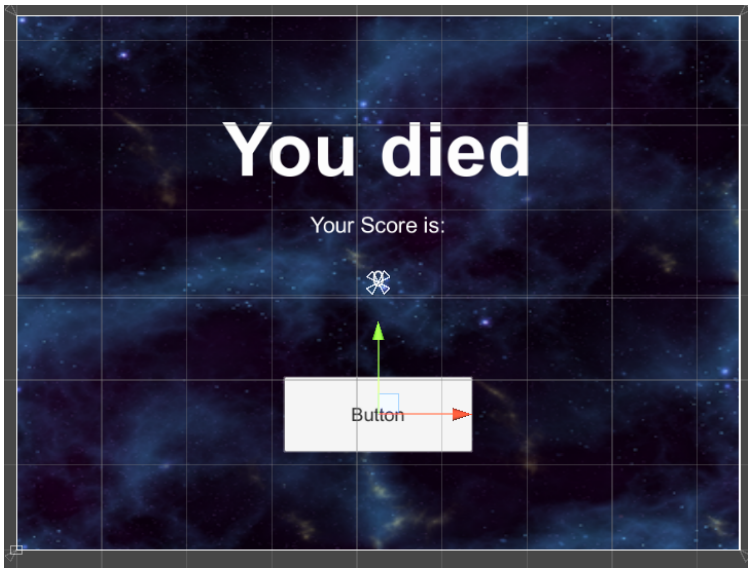
Customise the **Score** text component



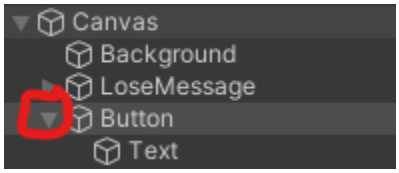


Create a **Button** object as a child of **Canvas**. Right-click **Canvas > UI > Button**. Give it a name.

Adjust the position of the **Button** object you just created

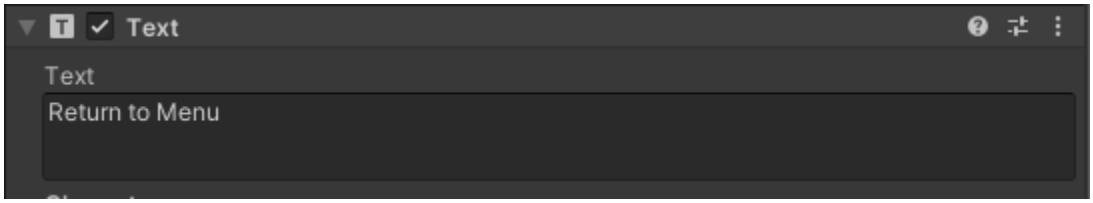


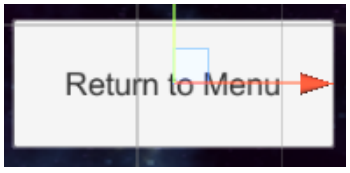
Expand the **Button** game object in the **Hierarchy**



Click the **Text** game object

In the **Inspector** customise the text to your liking





ScreenTransitionScript

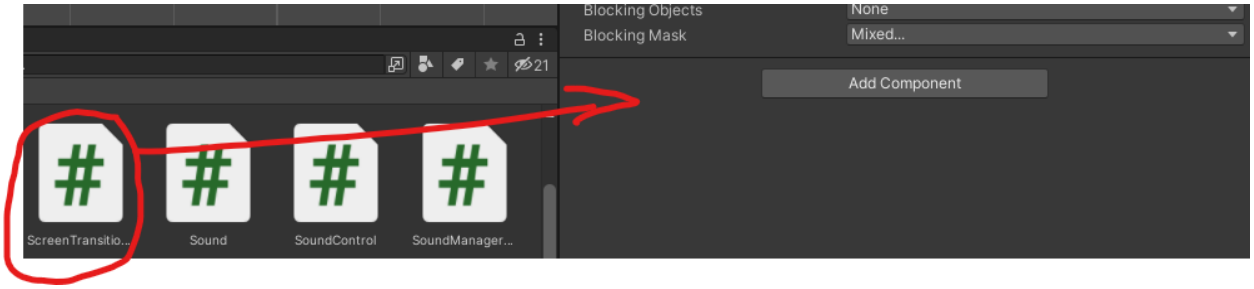
Overview

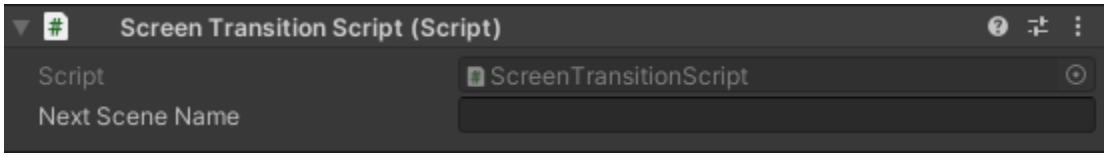
This script allows the player to change from the **lose** scene to another scene of their choice by clicking a button. You can have as many lose screens as you want.

Implementing the ScreenTransitionScript Script

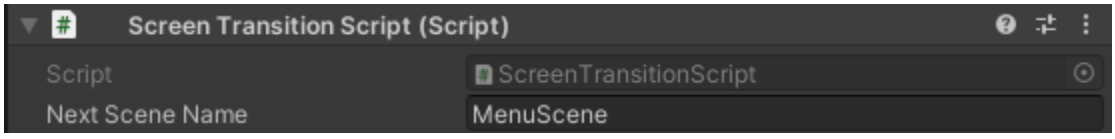
In your **Hierarchy**, click on your **Canvas** game object.

Drag the **ScreenTransitionScript** from your **Project** window into the **Canvas** game object's **Inspector**.



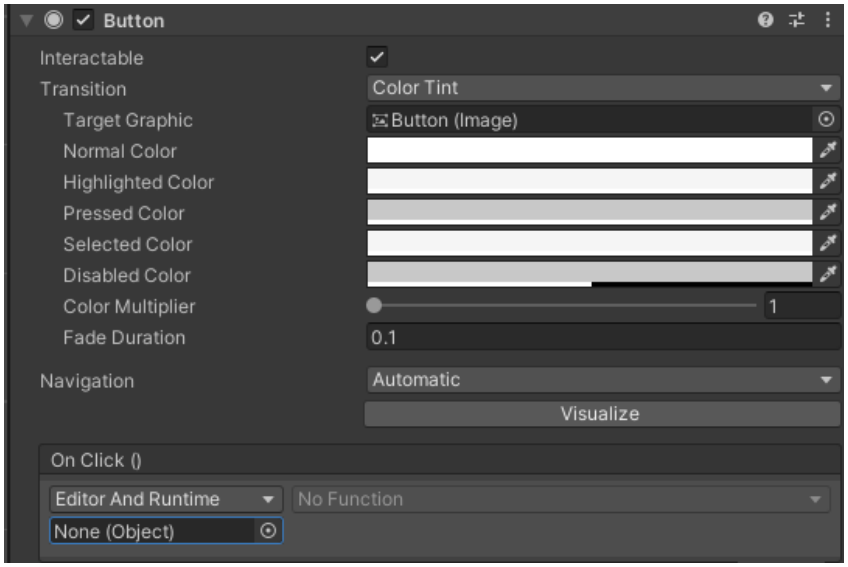


Type in the name of the scene you want to transition to. In this case, we want the player to return to the main menu so the name would be **MenuScene**.
However, you can choose for them to replay the game by typing in the level name, i.e. **Level 1**.

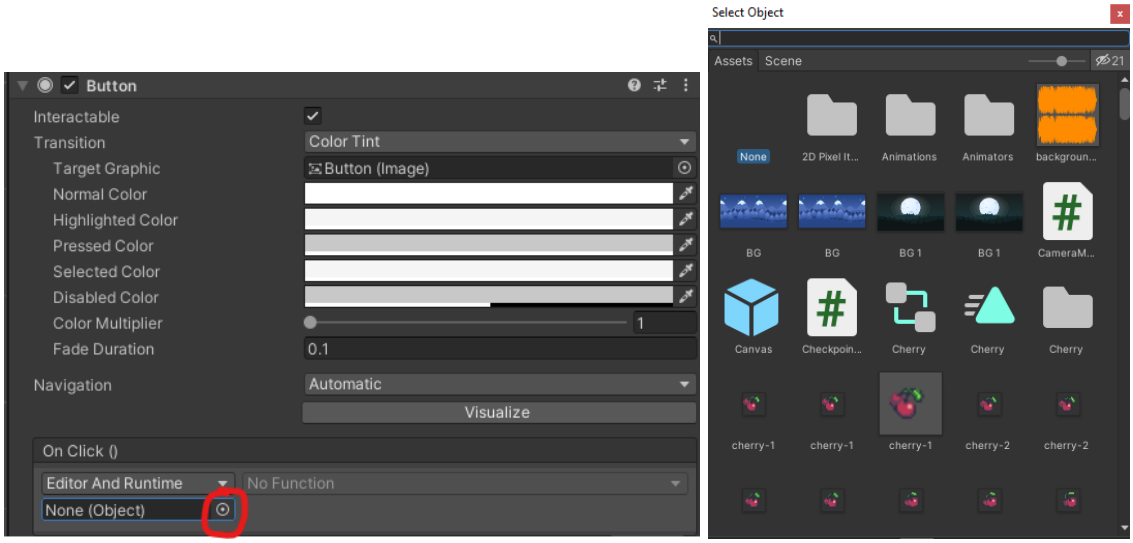


In you **Hierarchy**, click on the **Button** game object.

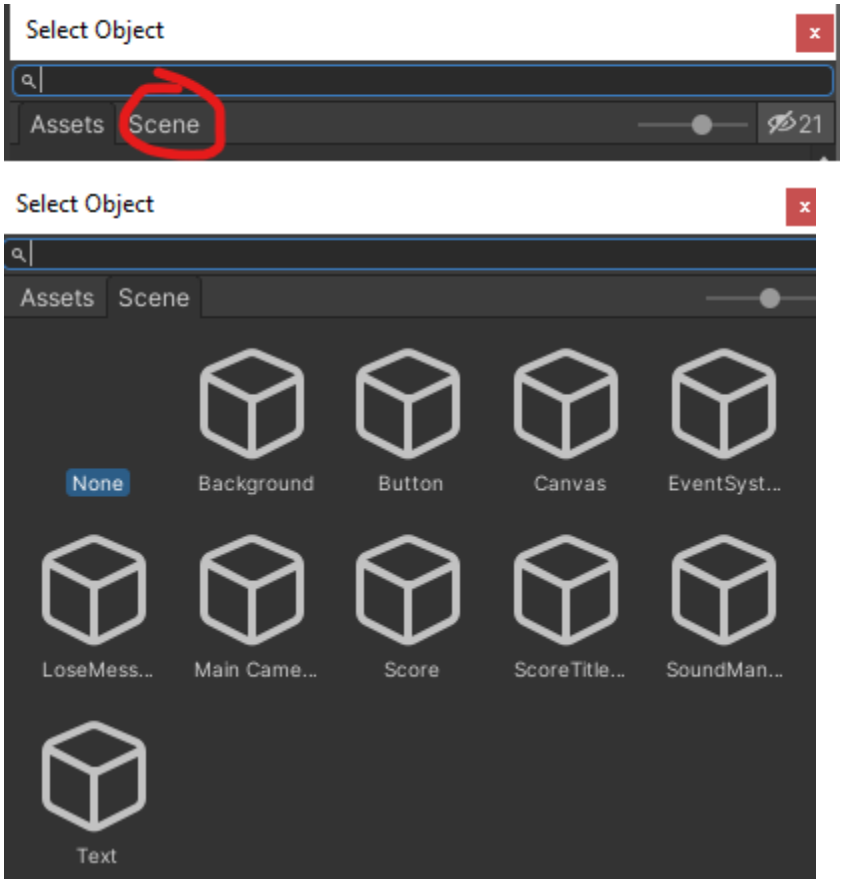
Scroll down to the **Button component** in the **Inspector**.



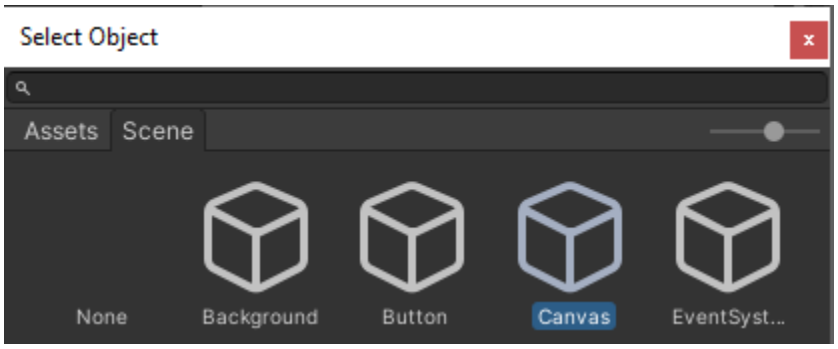
Click on the circle next to the **None (Object)** component section. A pop-up will appear to select a game object.



Click on **Scene** in that new window.

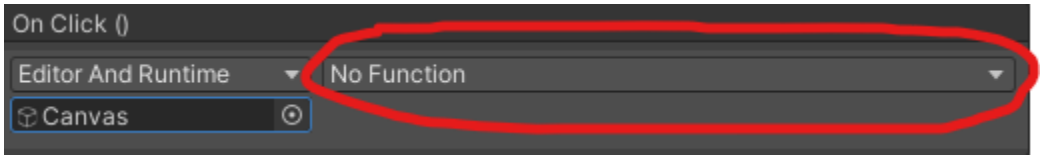


Click on **Canvas**. This will use that object and any scripts it has when the button is pressed.

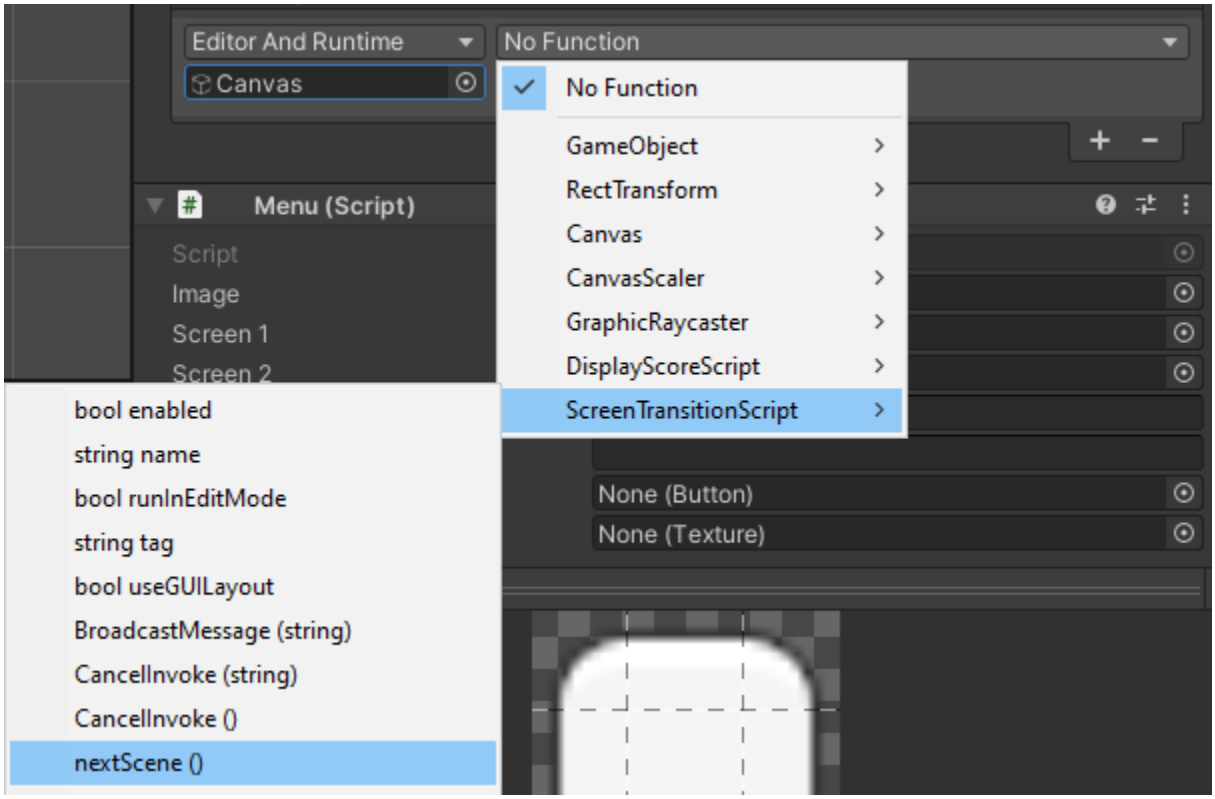


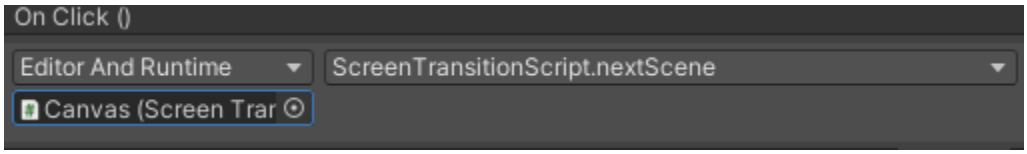
Close the window.

Click the dropdown with **No Function**.



Click on **ScreenTransitionScript > nextScene()**.
This will use the **nextScene** function found in the **ScreenTransitionScript** script and activate it when the button is pressed.

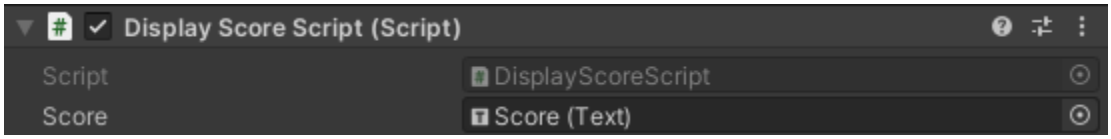




DisplayScoreScript

Overview

Shows the player score on the specified **Text** game object.



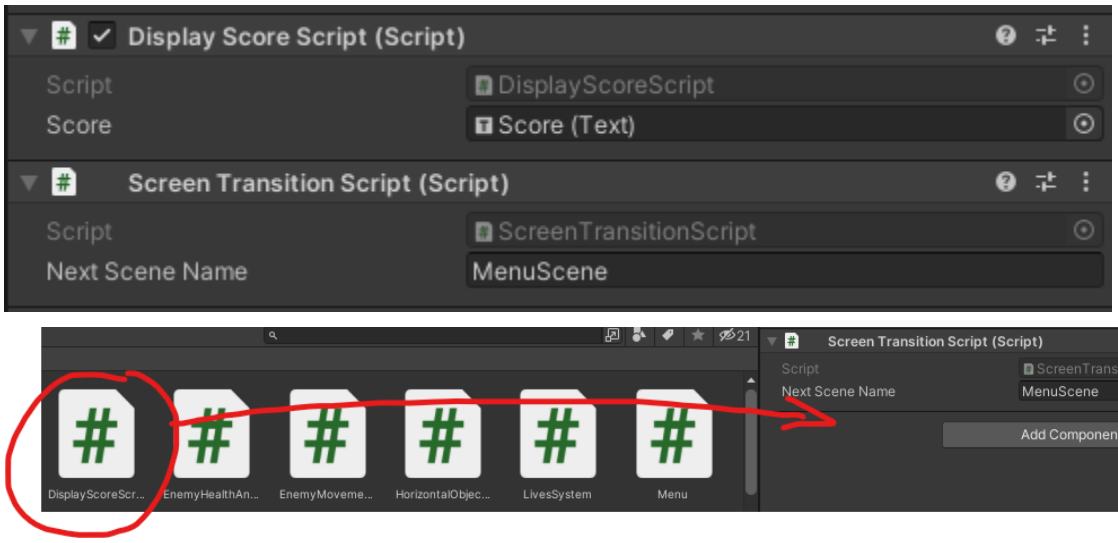
Variables:

- **Score** - the **Text** game object that will display the score.

Adding and Implementing the Script

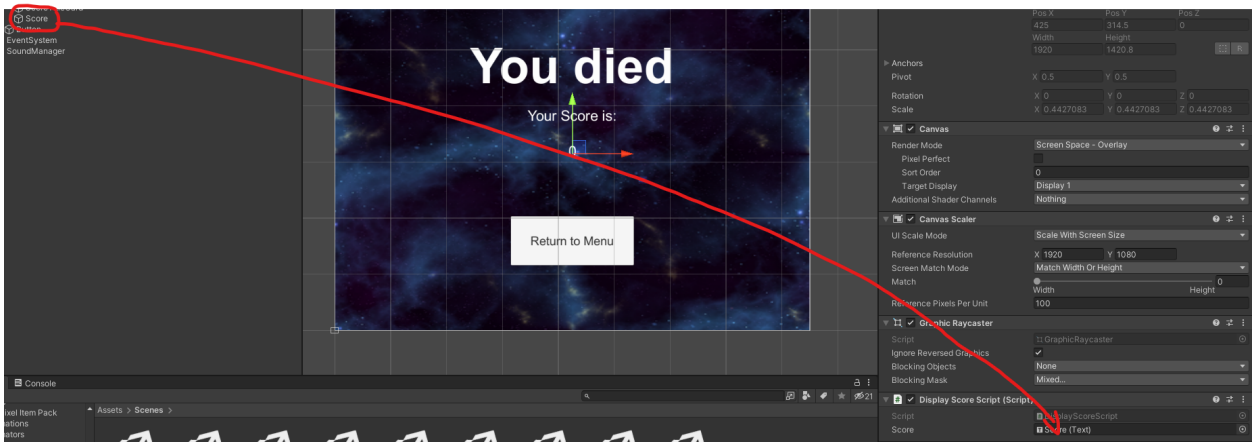
In your **Hierarchy**, click on your **Canvas** game object.

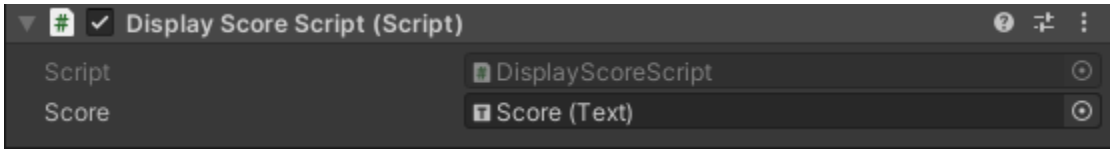
Drag the **DisplayScoreScript** script from your **Project** window into the **Canvas** game object's **Inspector**.



In your **Hierarchy**, find the **Text** game object that will display the score amount.

Drag it into the **Canvas** game object's **Inspector** with the script component **DisplayScoreScript**.





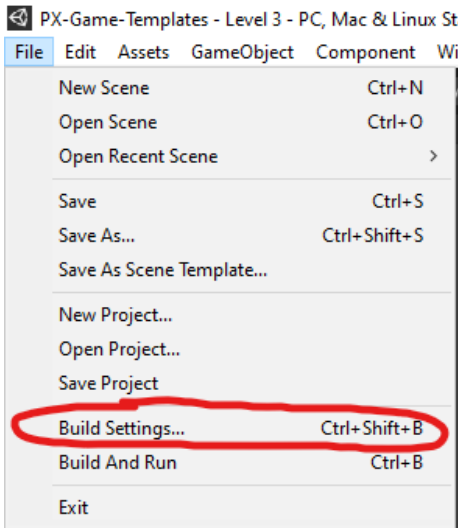
Build and Run the Game

Overview

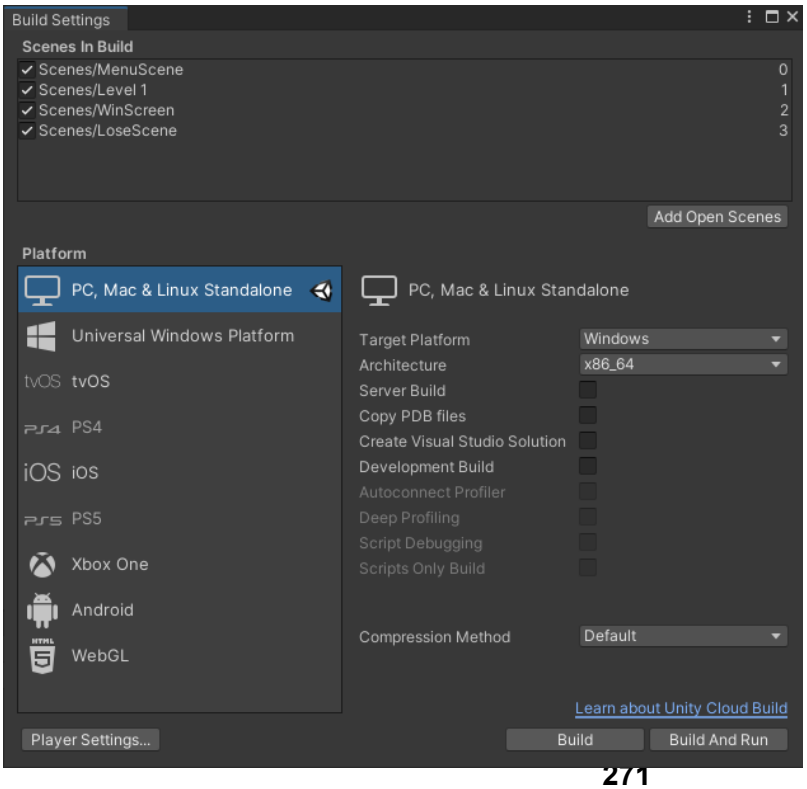
Building the game will collect all the resources, game objects and scenes you have created and create an **exe** file so that it is playable.

Building the Game

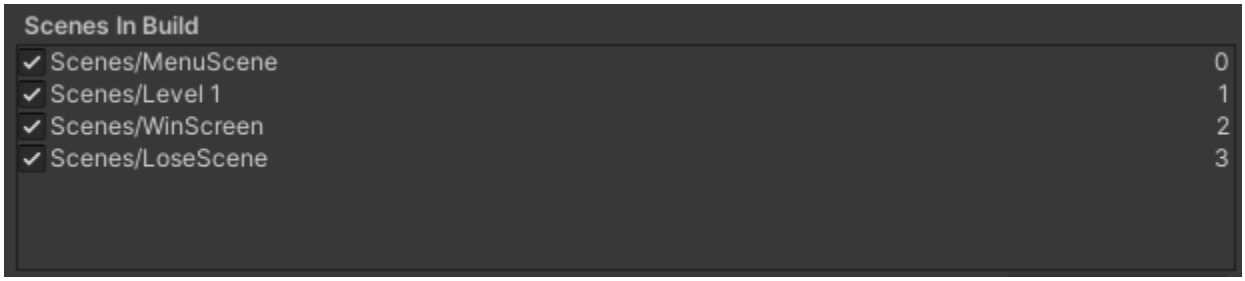
Go to **File** > **Build Settings**...



This will open a new window called **Build Settings** where you will compile everything and make it into a runnable **exe** file to play.



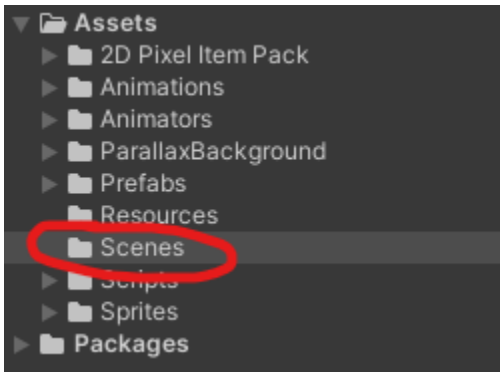
Scenes In Build are the **Scene** files that will be compiled and made playable.



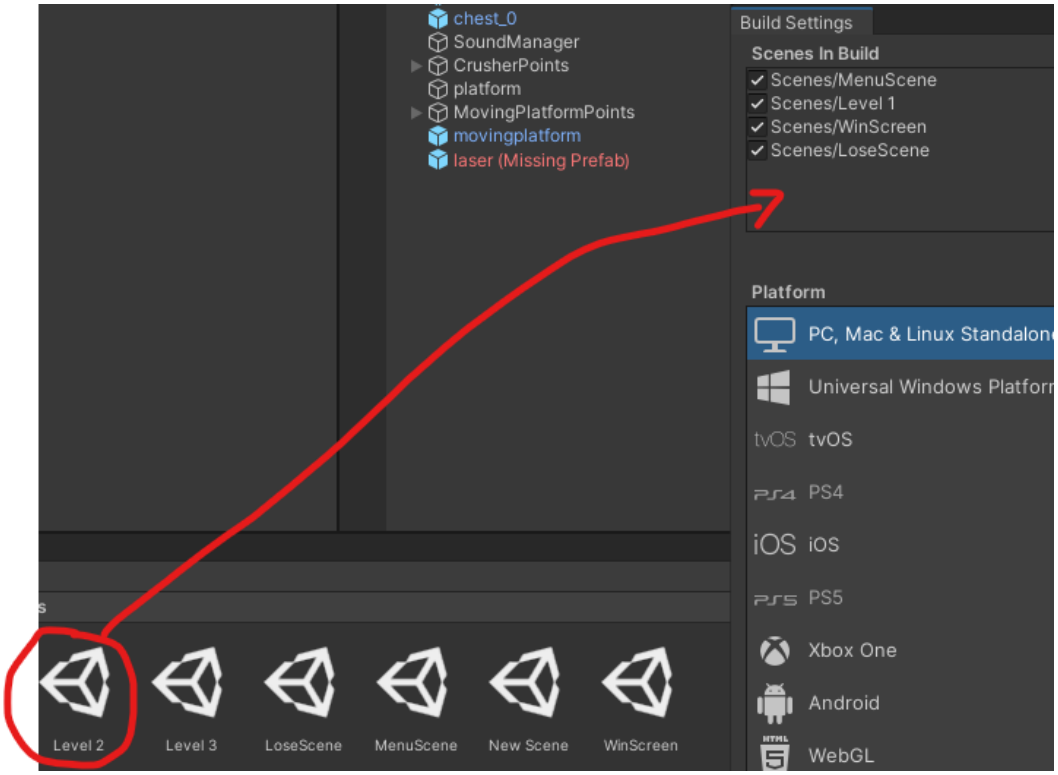
If there aren't scenes in '**Scenes in Build',** the Unity Editor will compile and make them playable when opening the **exe** game file after building the game.

Adding Scenes to 'Scenes In Build'

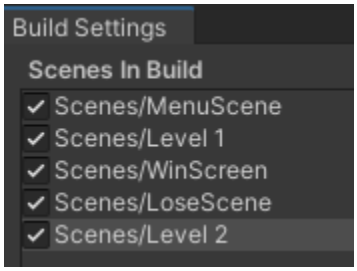
Find the location of the scene files.



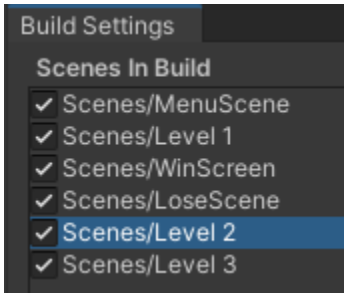
Drag the file into **Scenes In Build**.



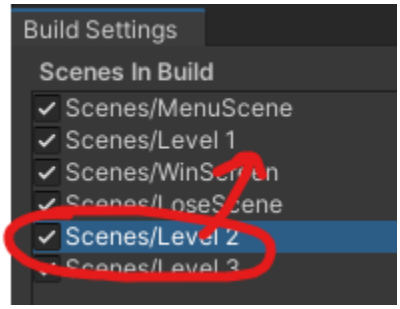
The added scene will appear at the bottom of the list.

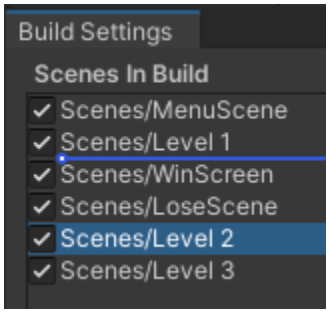


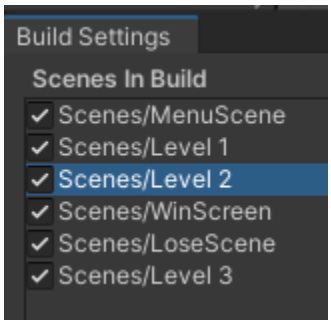
Repeat for any scenes you would like to add.

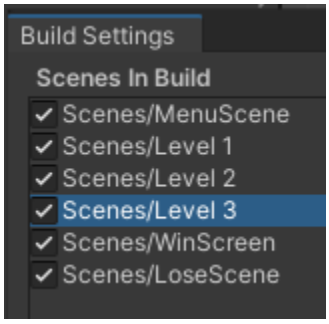


Organise the scenes by dragging it into the desired position in **Scene In Build**.

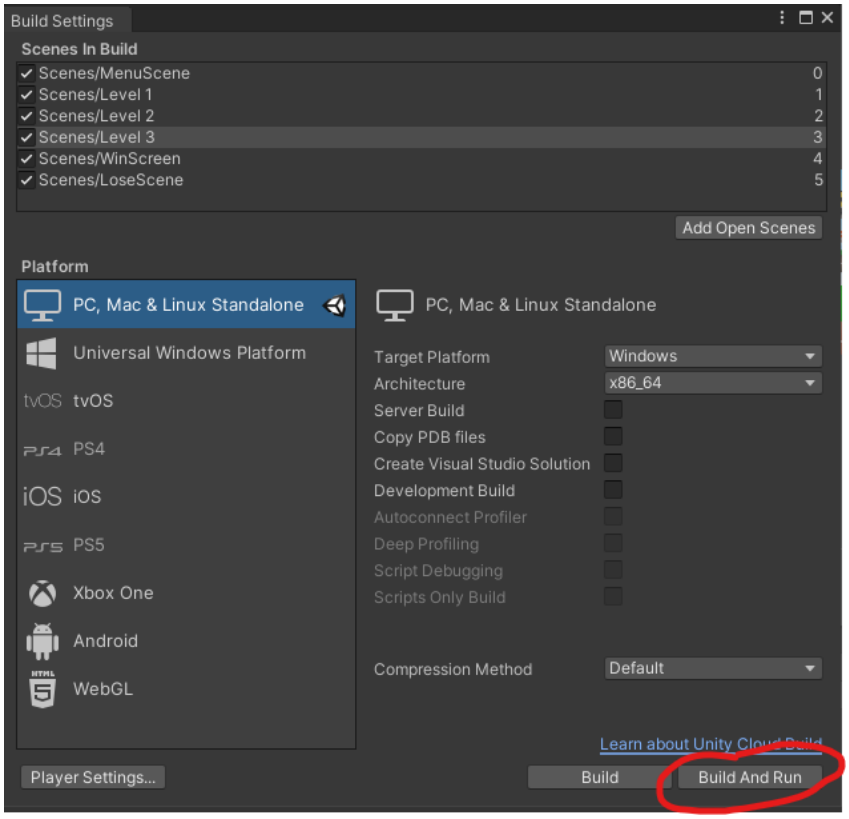


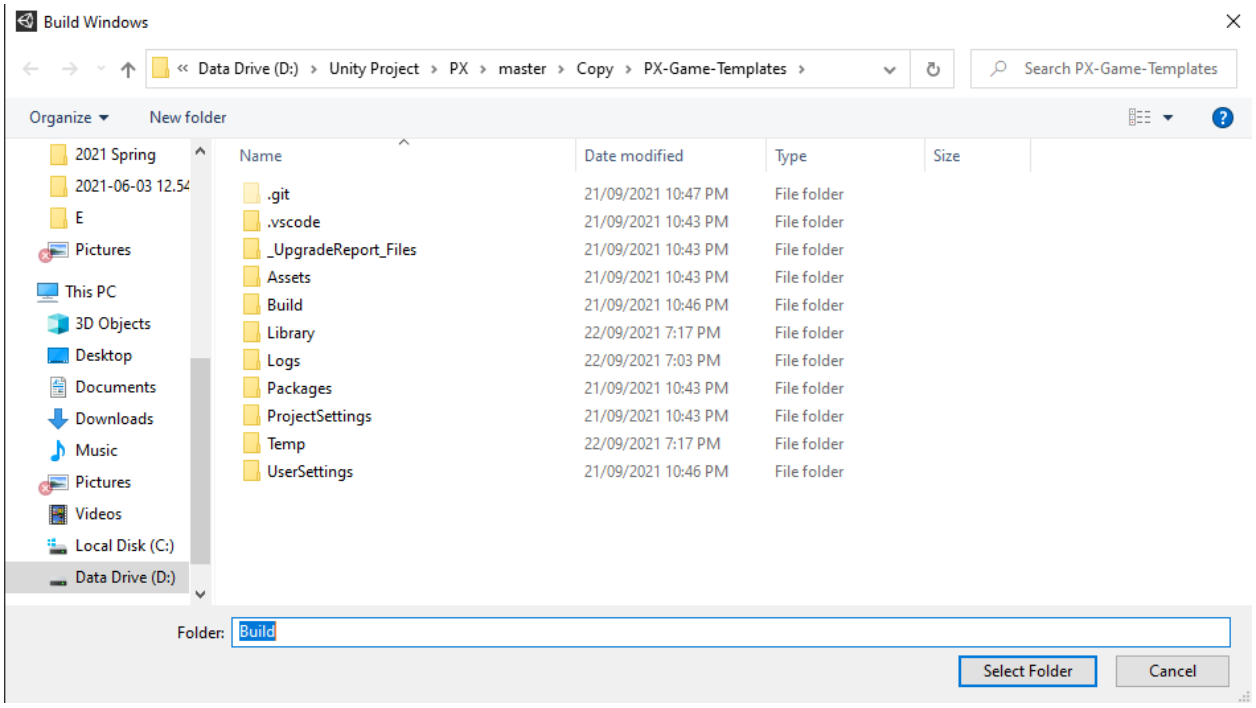




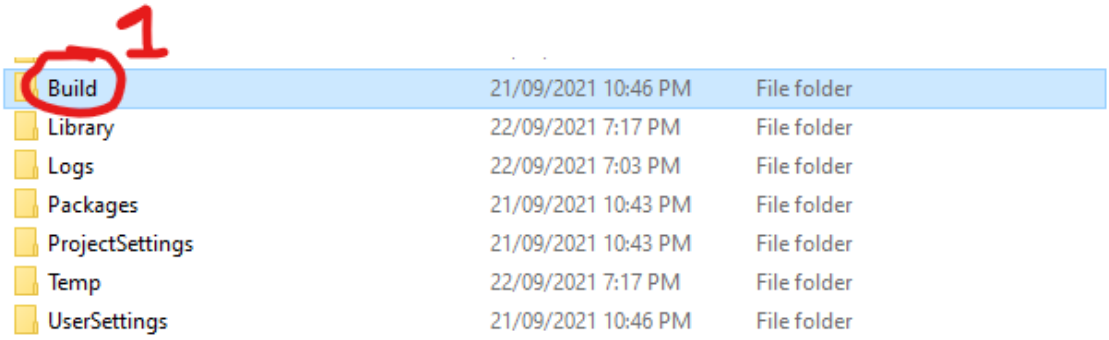


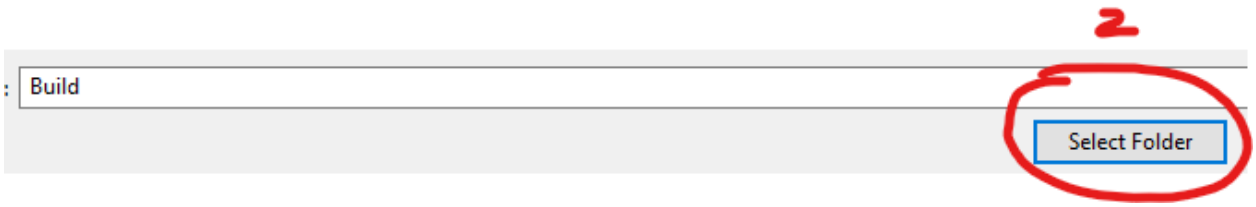
Finally click **Build and Run**.



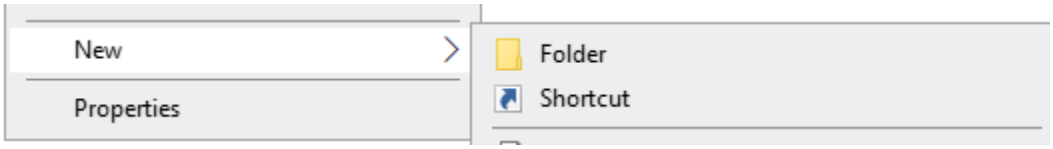


Click on the folder named **Build** and click **Select Folder**.





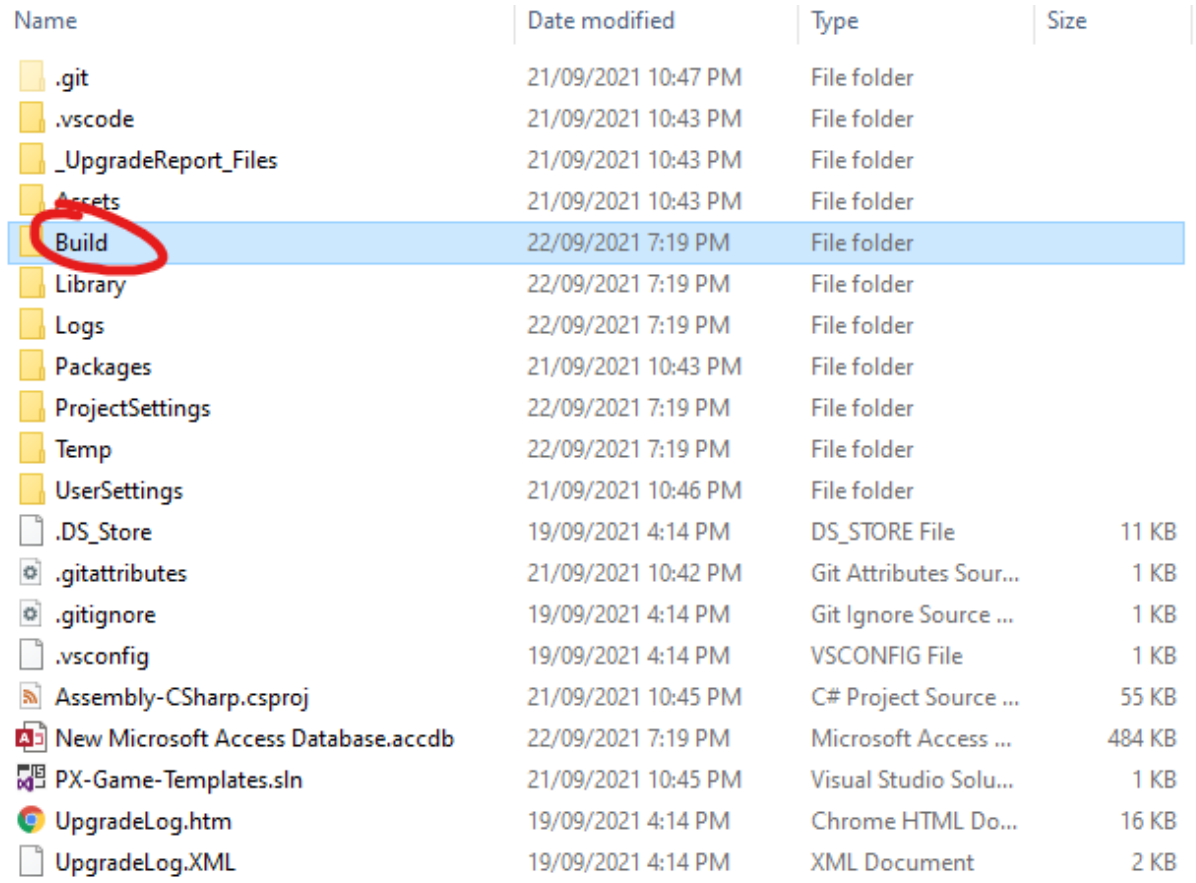
If you do not have a folder named Build, you can create one.
Right click an empty space, click New, then Folder.



After you click **Select Folder**, wait until it finishes compiling and it will open the game automatically.

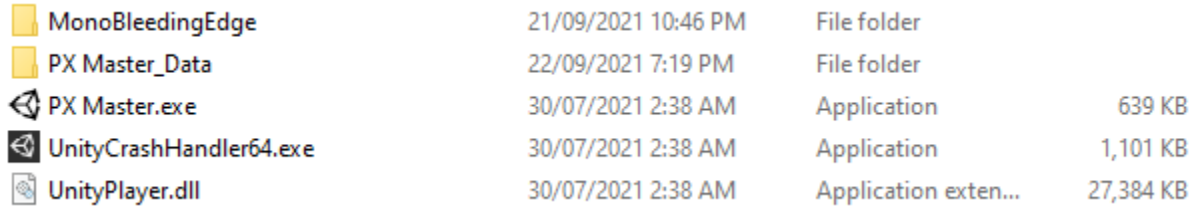
Manually run and play the game

Navigate to your project in your file explorer and open **Build**.



Name	Date modified	Type	Size
.git	21/09/2021 10:47 PM	File folder	
.vscode	21/09/2021 10:43 PM	File folder	
_UpgradeReport_Files	21/09/2021 10:43 PM	File folder	
Assets	21/09/2021 10:43 PM	File folder	
Build	22/09/2021 7:19 PM	File folder	
Library	22/09/2021 7:19 PM	File folder	
Logs	22/09/2021 7:19 PM	File folder	
Packages	21/09/2021 10:43 PM	File folder	
ProjectSettings	22/09/2021 7:19 PM	File folder	
Temp	22/09/2021 7:19 PM	File folder	
UserSettings	21/09/2021 10:46 PM	File folder	
.DS_Store	19/09/2021 4:14 PM	DS_STORE File	11 KB
.gitattributes	21/09/2021 10:42 PM	Git Attributes Sour...	1 KB
.gitignore	19/09/2021 4:14 PM	Git Ignore Source ...	1 KB
.vsconfig	19/09/2021 4:14 PM	VSCONFIG File	1 KB
Assembly-CSharp.csproj	21/09/2021 10:45 PM	C# Project Source ...	55 KB
New Microsoft Access Database.accdb	22/09/2021 7:19 PM	Microsoft Access ...	484 KB
PX-Game-Templates.sln	21/09/2021 10:45 PM	Visual Studio Solu...	1 KB
UpgradeLog.htm	19/09/2021 4:14 PM	Chrome HTML Do...	16 KB
UpgradeLog.XML	19/09/2021 4:14 PM	XML Document	2 KB

It should look like this.



MonoBleedingEdge	21/09/2021 10:46 PM	File folder	
PX Master_Data	22/09/2021 7:19 PM	File folder	
PX Master.exe	30/07/2021 2:38 AM	Application	639 KB
UnityCrashHandler64.exe	30/07/2021 2:38 AM	Application	1,101 KB
UnityPlayer.dll	30/07/2021 2:38 AM	Application exten...	27,384 KB

Double click the **exe** file that has the same name as the project.