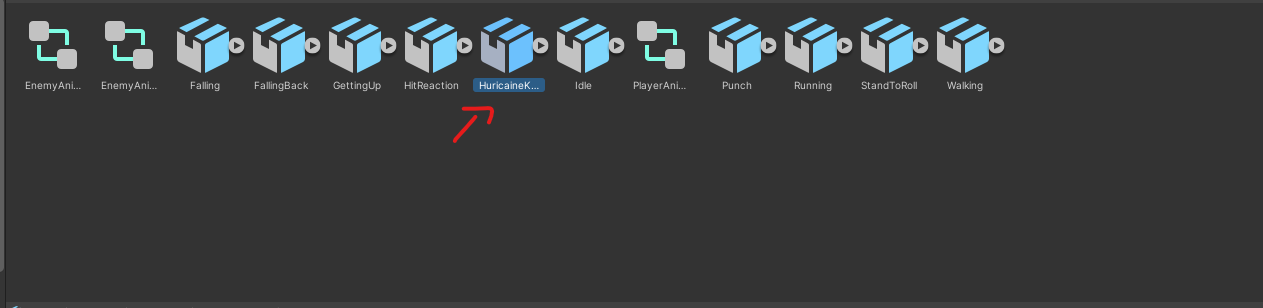
Lab 6 : UI, VFX, sound effect and scene management

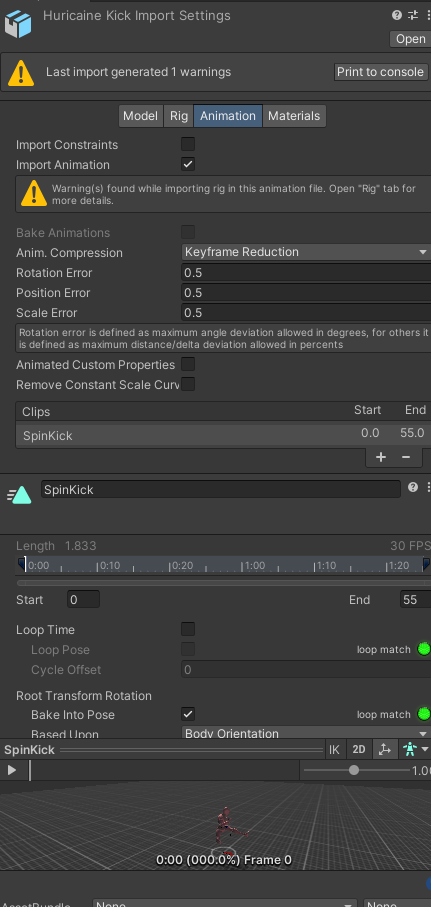
Open the assignment :   
<https://classroom.github.com/a/mCi9lSZx>

### **Add a jump attack to our character**

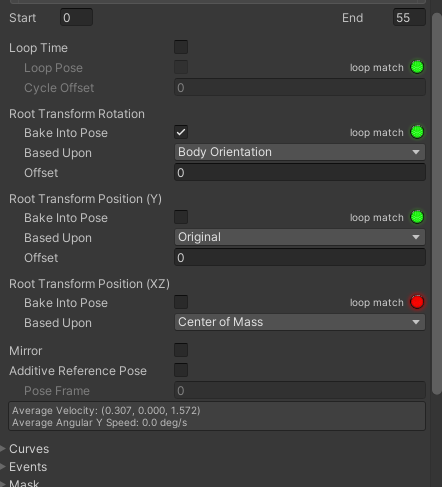
Open the Huricain Kick Settings by clicking on the



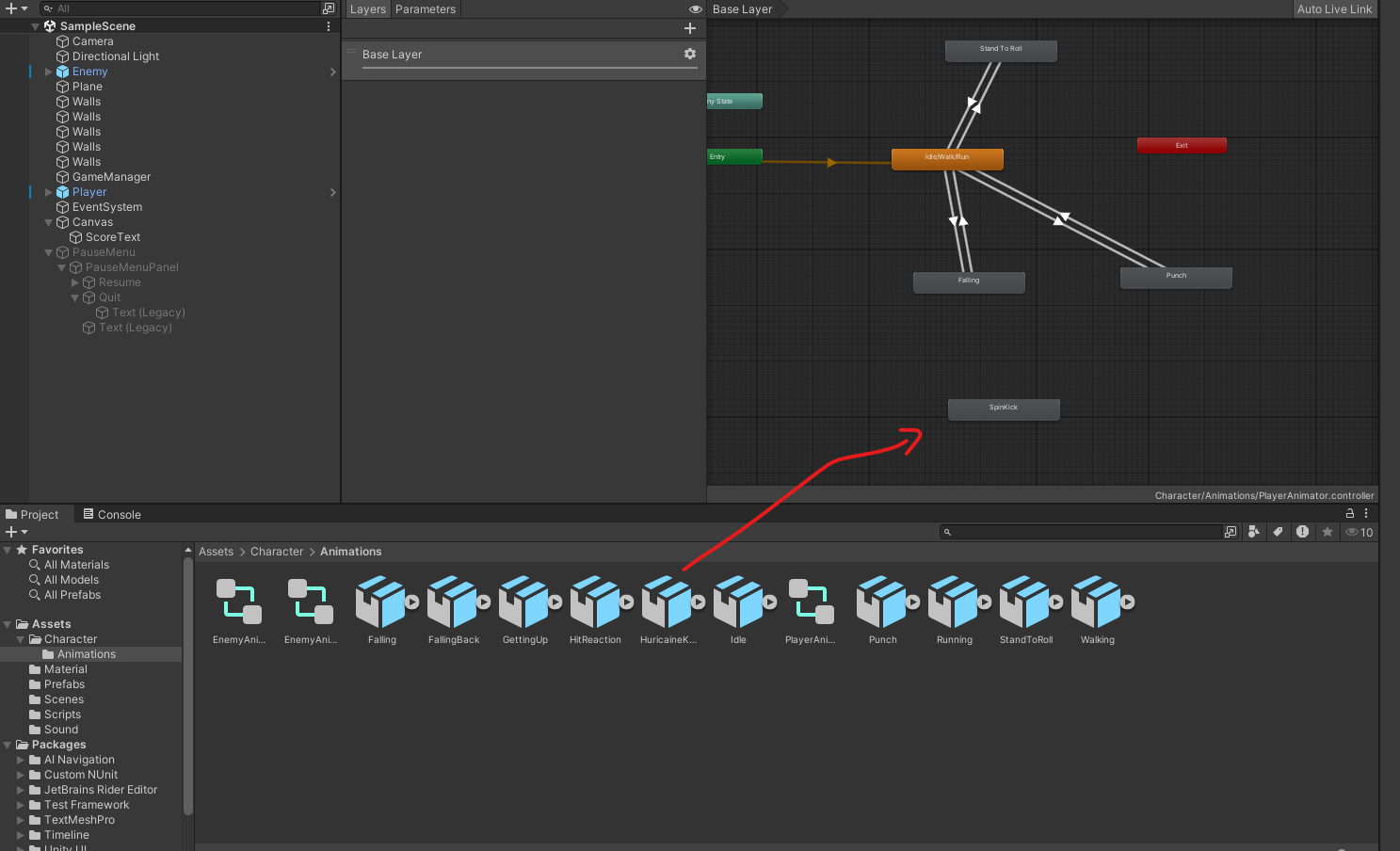
To Setup the Huricaine Kick animation click on the Animation tab



Scroll down in the animation settings and make sure the Root Transform Rotation is Baked into the pose.

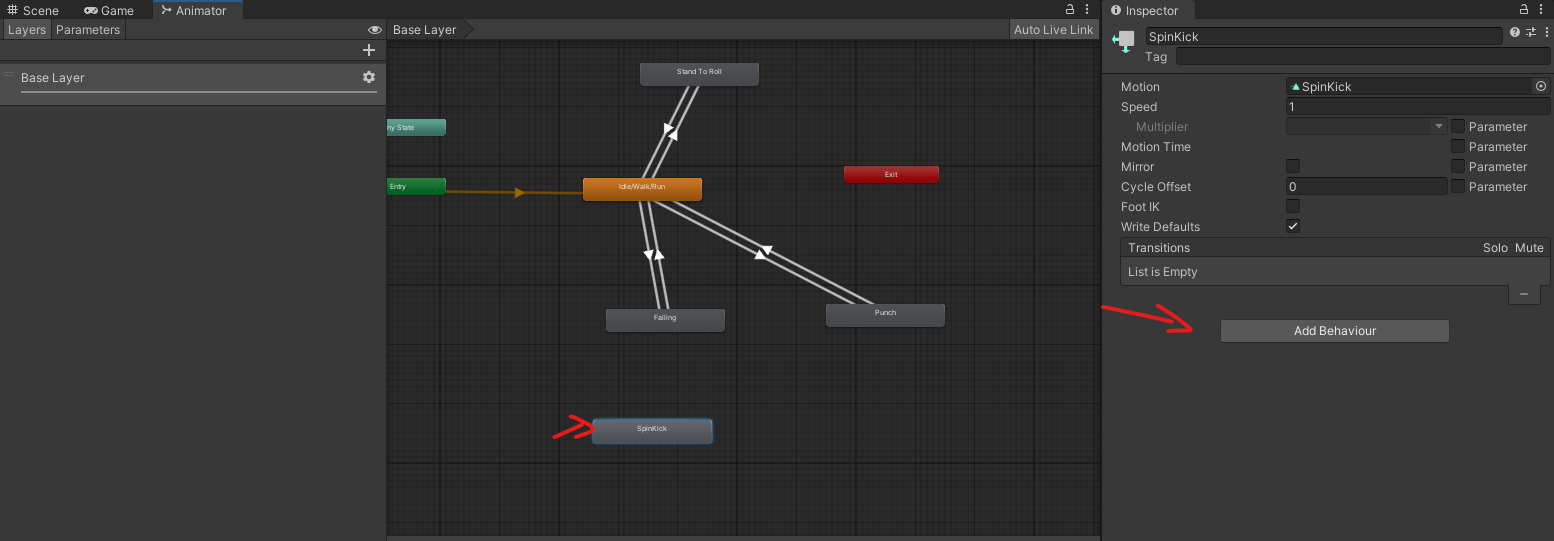


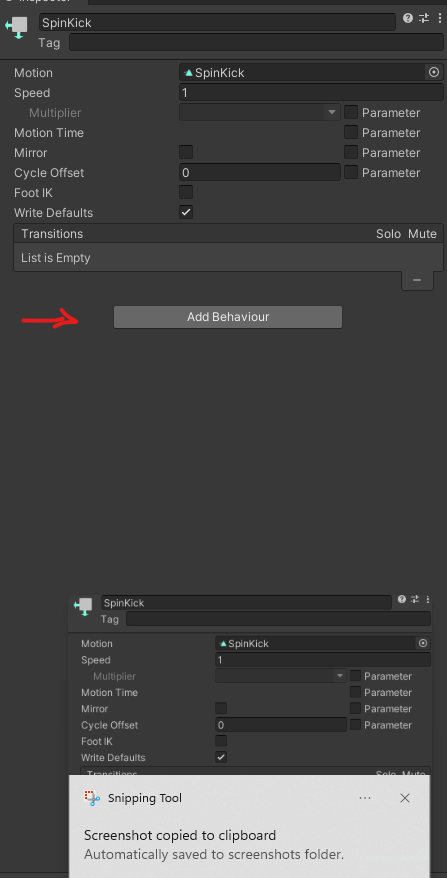
Next we want to add the animation to the Player Animator



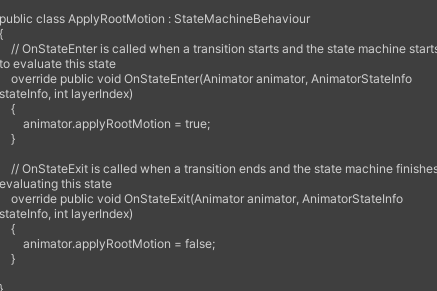
Next, we will want to add a script to our animation to use the root motion of the animation to move our character

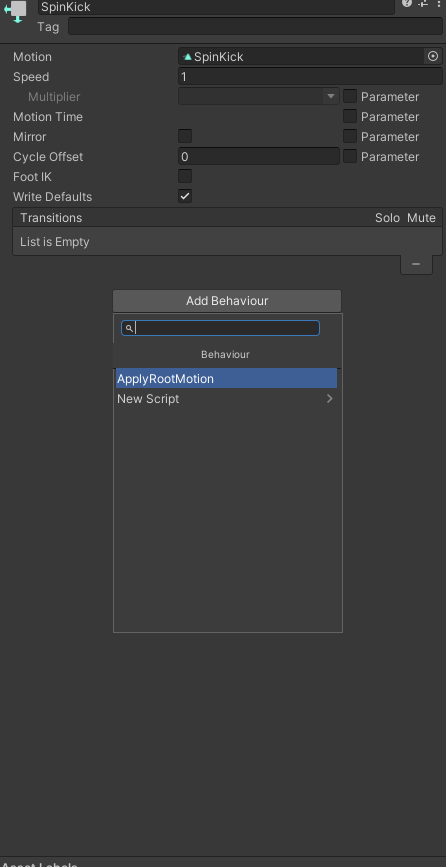
Click on the Animation you just added to your animator

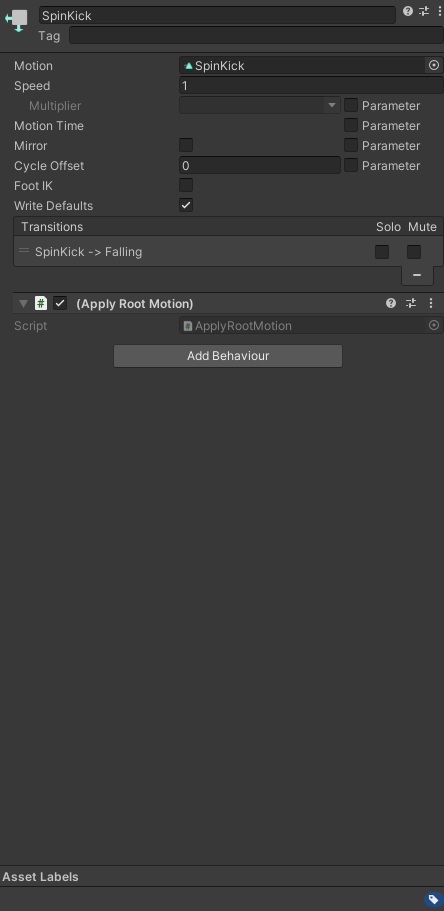




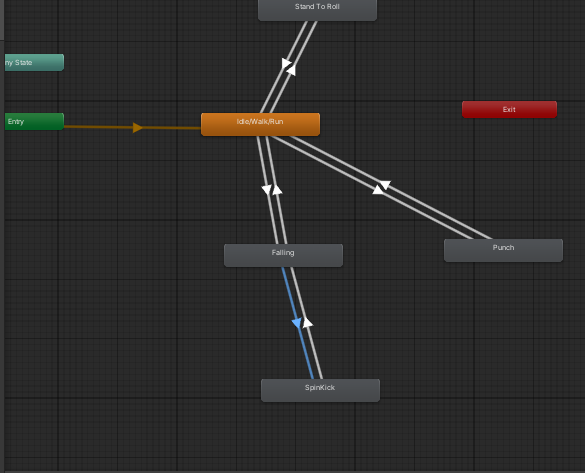
Select the ApplyRootMotion script ( It is a simple script that enables and disables the root motion of the animator when the animation starts and ends )



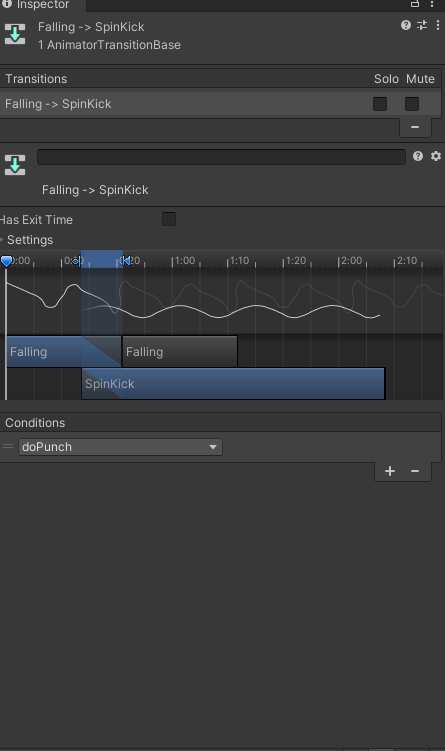




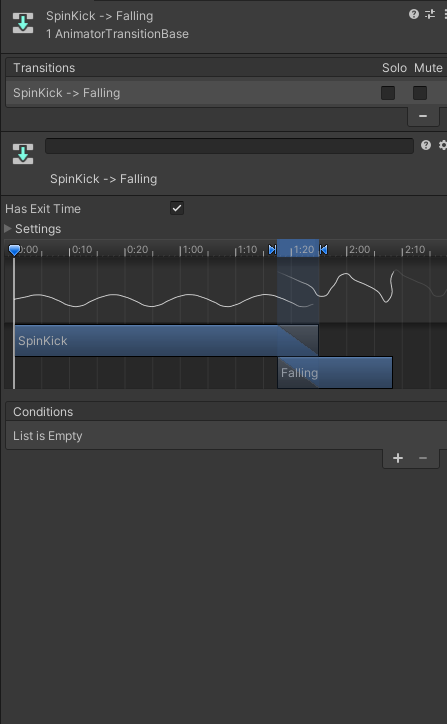
Setup the transition like so :



The falling to spin kick transition should be setup like so



The spin kick to falling transition should be setup like so :



### 

### 

### **Add a Particle System Component to the Enemy GameObject**

**Select Enemy GameObject:**

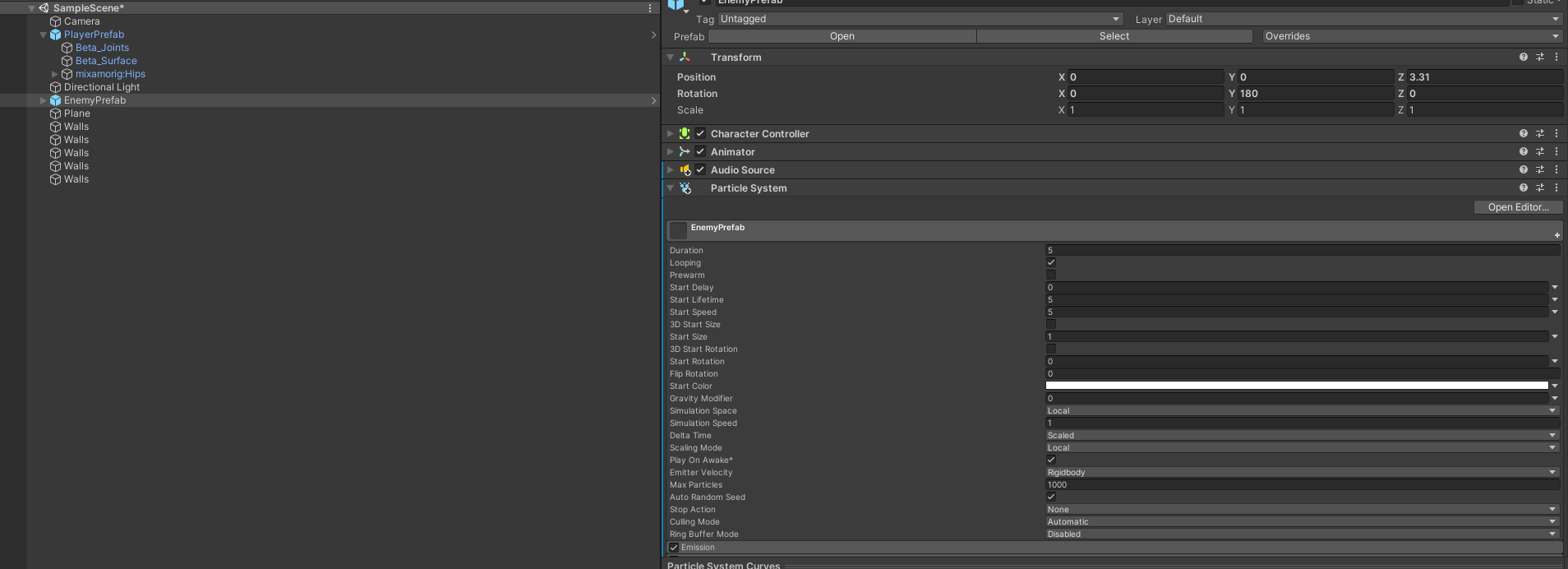
In the **Hierarchy**, select your enemy GameObject.

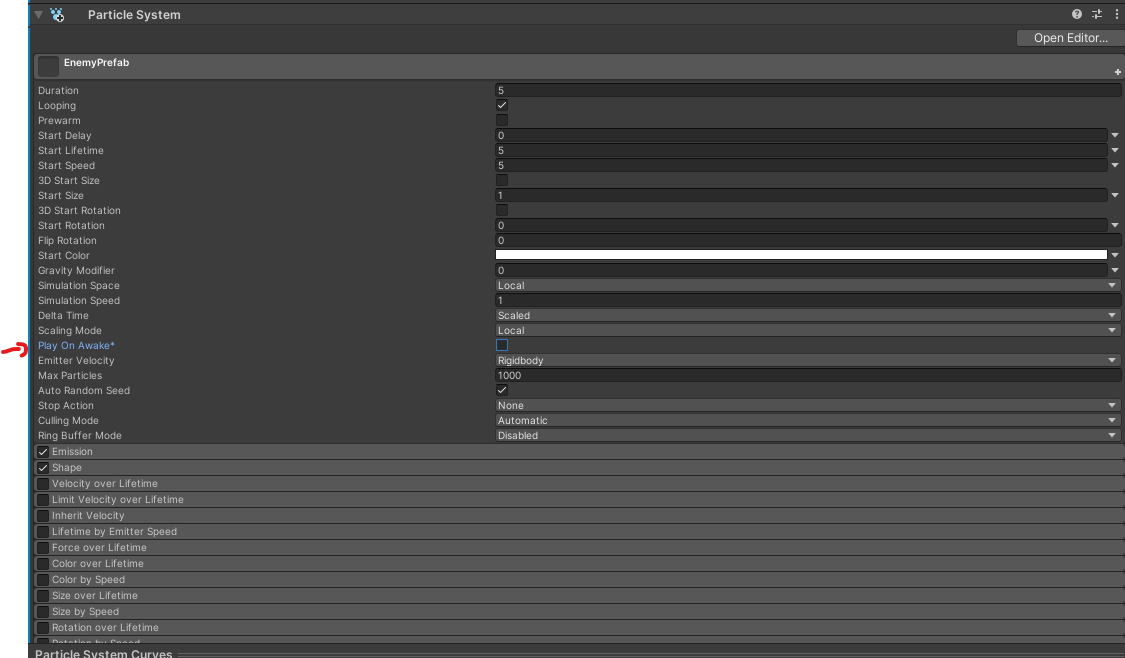
**Add Particle System:**

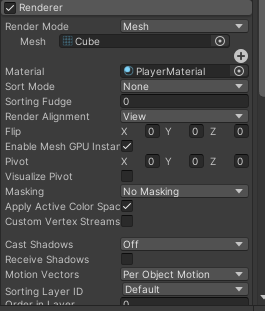
In the **Inspector**, click **Add Component**.

Choose **Particle System**.

**Configure Particle System:**

Adjust the particle system settings as desired (e.g., shape, emission rate, lifetime).

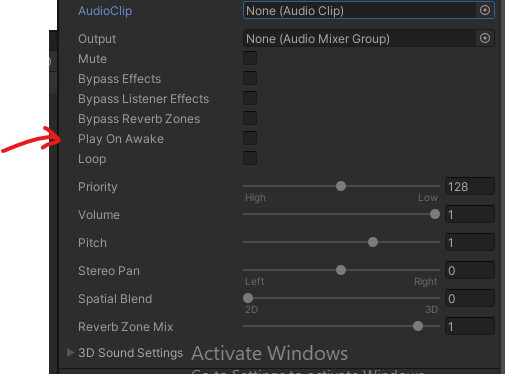
Uncheck **Play On Awake** to prevent it from playing automatically.

Set the renderer to cubes and assign it a material  
\

**Add Audio Source to Enemy:**

Select the enemy GameObject.

Click **Add Component** and choose **Audio Source**.



Uncheck **Play On Awake**.

**Create Enemy Script:**

In the **Project** window, create a new C# script named EnemyController.

**Edit the Enemy Script:**

using UnityEngine;

public class EnemyController : MonoBehaviour

{

private Animator animator;

private ParticleSystem hitParticles;

private AudioSource audioSource;

void Start()

{

animator = GetComponent<Animator>();

hitParticles = GetComponent<ParticleSystem>();

audioSource = GetComponent<AudioSource>();

}

public void GotHit()

{

animator.SetTrigger("GotHit");

hitParticles.Play();

audioSource.Play();

if (GameManager.Instance != null)

{

GameManager.Instance.AddScore(1); // Add 1 point per hit

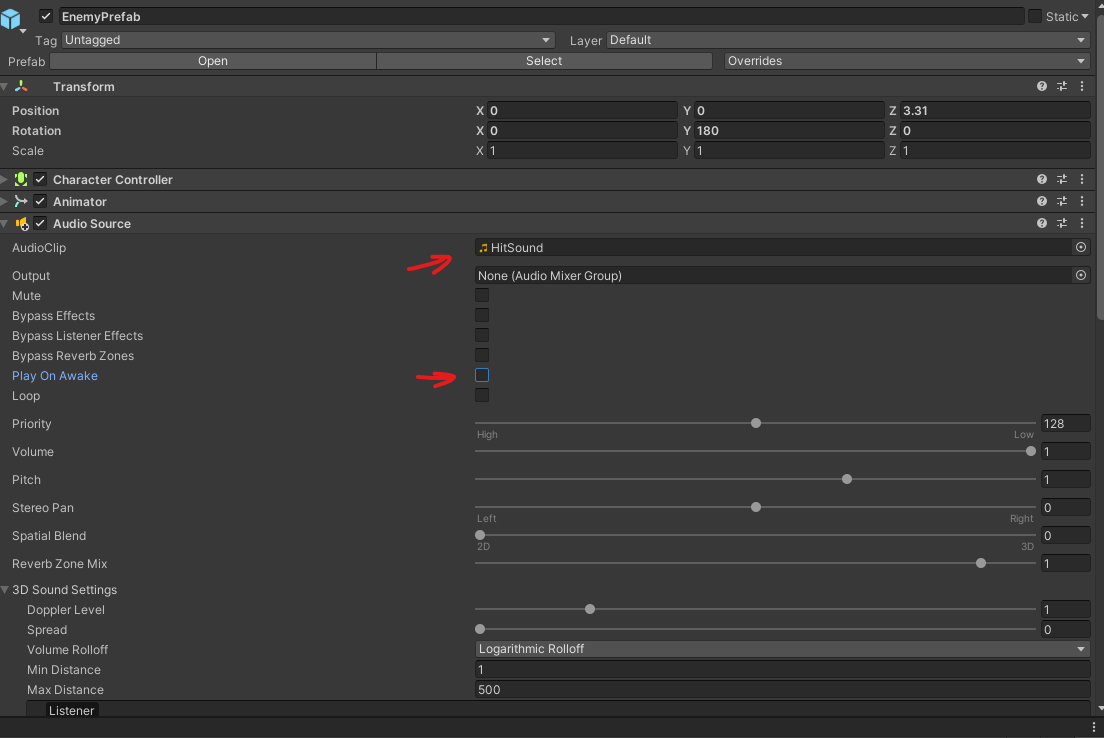
}

}

}

**Assign Hit Sound:**

In the **Inspector**, assign the hit sound to the Audio Source component.



**Set Up Animator Trigger:**

Ensure the **GotHit** trigger is set up in the enemy's Animator Controller.

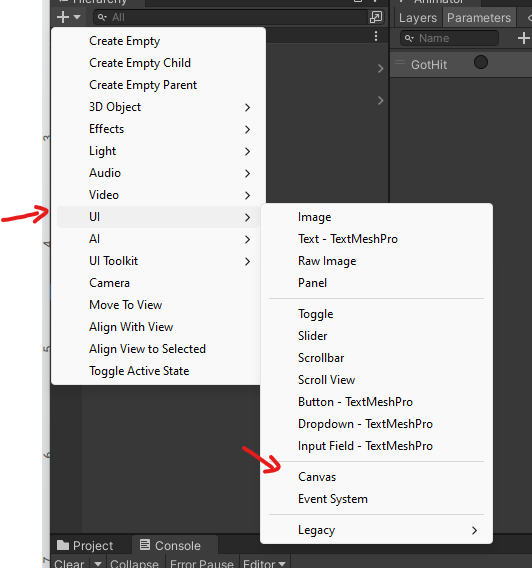
## **Add a Health Bar UI and Health Component to the Enemy**

**Objective**: Attach a health bar UI element to the enemy and create a Health component script to handle the health logic.

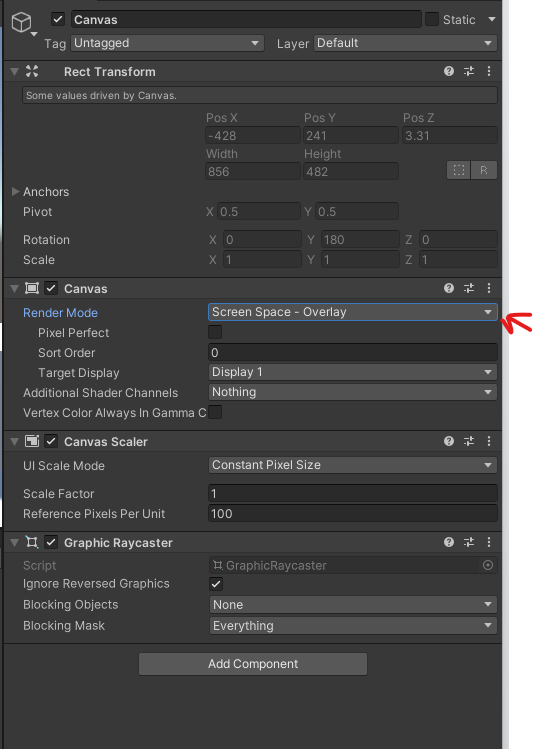
### **Steps:**

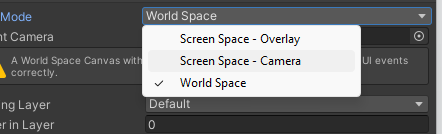
**Create a Health Bar UI**:

Right click on the EnemyPrefab then select : UI > Canvas.

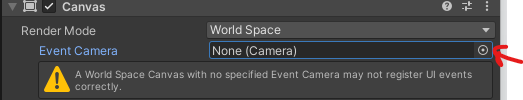


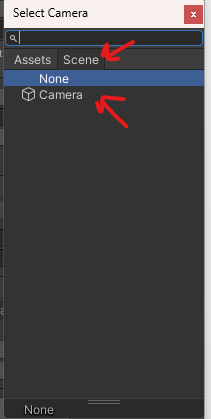
Set the Canvas Render Mode to World Space.





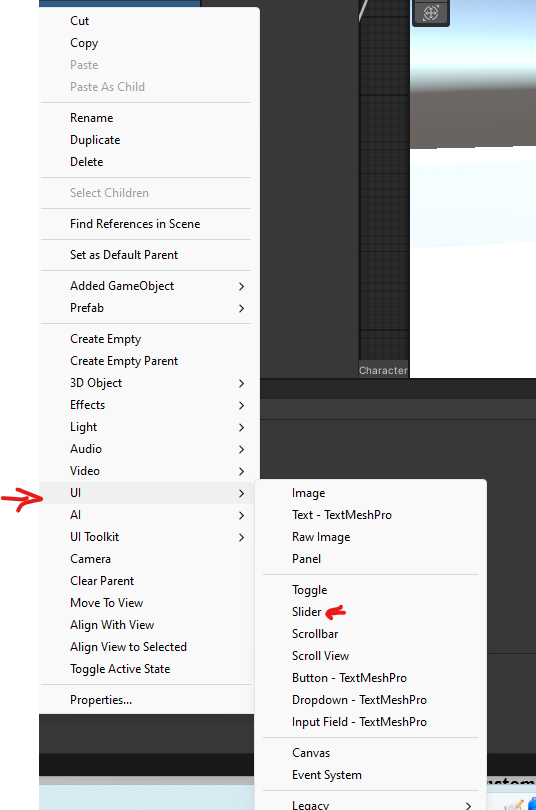
Then select the camera for the Option called Event Camera





Inside the **Canvas**, create a new Slider UI element (Right-click on the Canvas, then UI > Slider).

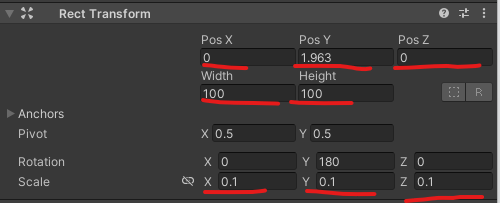




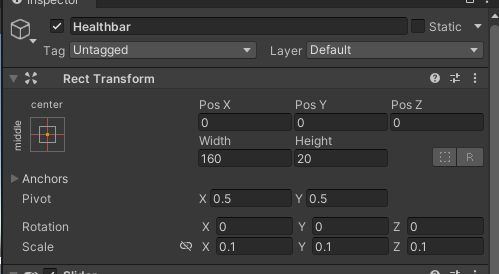
Call it health bar

Position the health bar Slider over the enemy’s head (you will need to adjust the RectTransform of the Canvas and Slider).

Use these values for the Canvas RecTransform

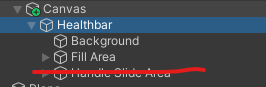


And these for the Healthbar Slider

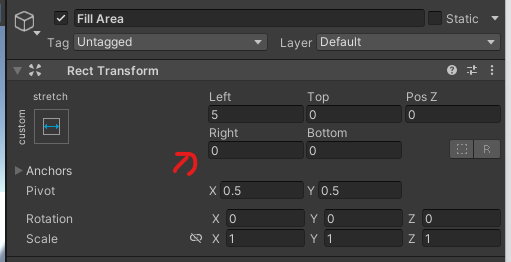


**Customize the Health Bar**:

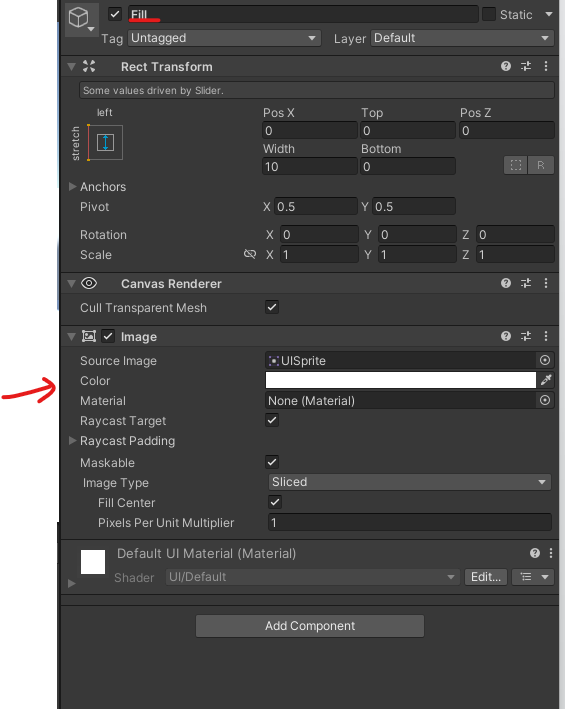
Remove unnecessary UI elements from the Slider, like the Handle

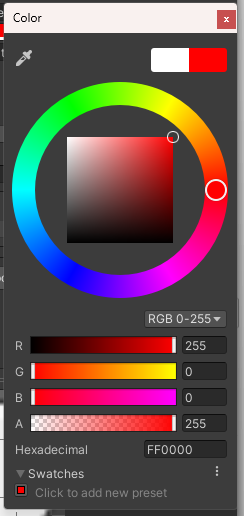


Change the Fill Area to represent the health amount (use a red color for health).

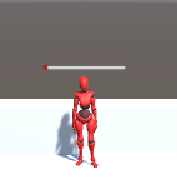






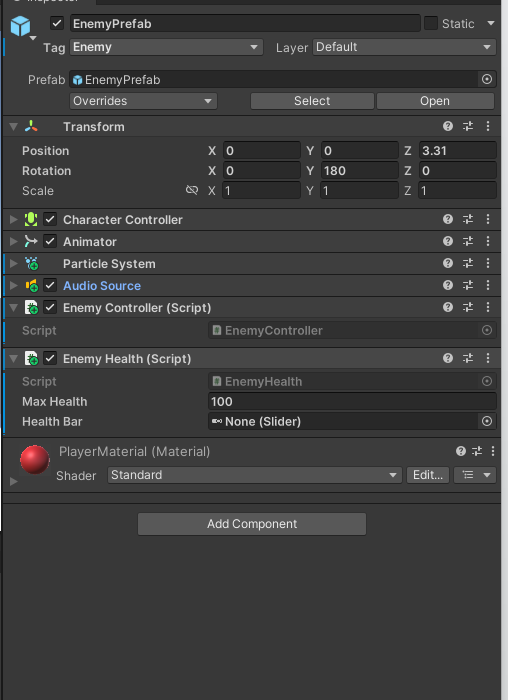


The healthbar in the game should look like this



**Create a Health Script**:

Create a new script named EnemyHealth.cs and attach it to the **EnemyPrefab** GameObject.



Implement the following logic in the script:

Make sure it has UnityEngine.UI

using UnityEngine;

using UnityEngine.UI;

public class EnemyHealth : MonoBehaviour

{

public int maxHealth = 100;

private int currentHealth;

public Slider healthBar;

private void Start()

{

currentHealth = maxHealth;

healthBar.maxValue = maxHealth;

healthBar.value = currentHealth;

}

public void TakeDamage(int damage)

{

currentHealth -= damage;

healthBar.value = currentHealth;

if (currentHealth <= 0)

{

Die();

}

}

void Die()

{

Debug.Log("The enemy has died");

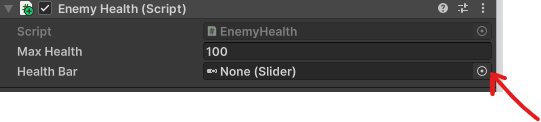
}

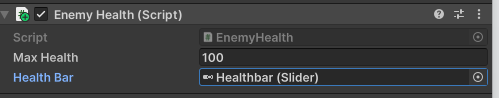
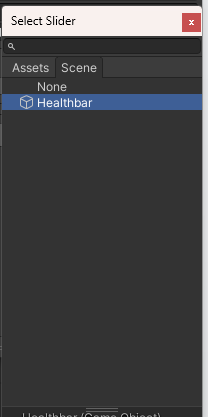
}

**Link the Health Bar to the Script**:

Select the EnemyPrefab GameObject in the Hierarchy.

In the EnemyHealth script component, link the healthBar field to the Slider object in the Canvas.





Now lets make the punch game mechanic do damage to our Enemy.

In the EnemyController add a call to TakeDamage in the GotHit() method

And remove the GameManager Increment score call.

We will move that to the Die function.

using UnityEngine;

public class EnemyController : MonoBehaviour

{

private float health = 0;

private EnemyHealth enemyHealth;

private Animator animator;

private ParticleSystem hitParticles;

private AudioSource audioSource;

void Start()

{

animator = GetComponent<Animator>();

hitParticles = GetComponent<ParticleSystem>();

audioSource = GetComponent<AudioSource>();

enemyHealth = GetComponent<EnemyHealth>();

}

public void GotHit()

{

if (enemyHealth != null)

{

enemyHealth.TakeDamage(10);

animator.SetTrigger("GotHit");

hitParticles.Play();

audioSource.Play();

if (GameManager.Instance != null)

{

GameManager.Instance.AddScore(1); // Add 1 point per hit

}

}

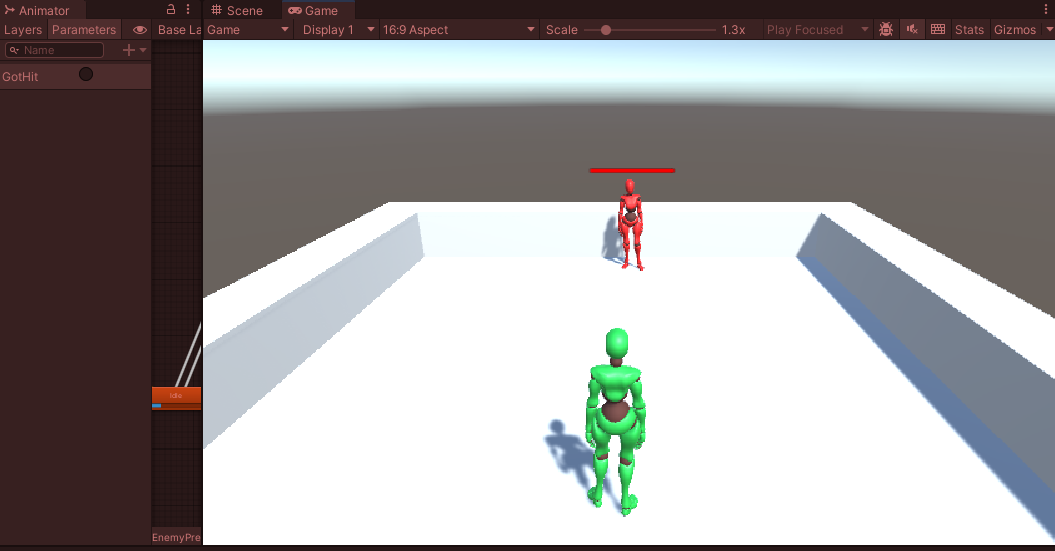
}

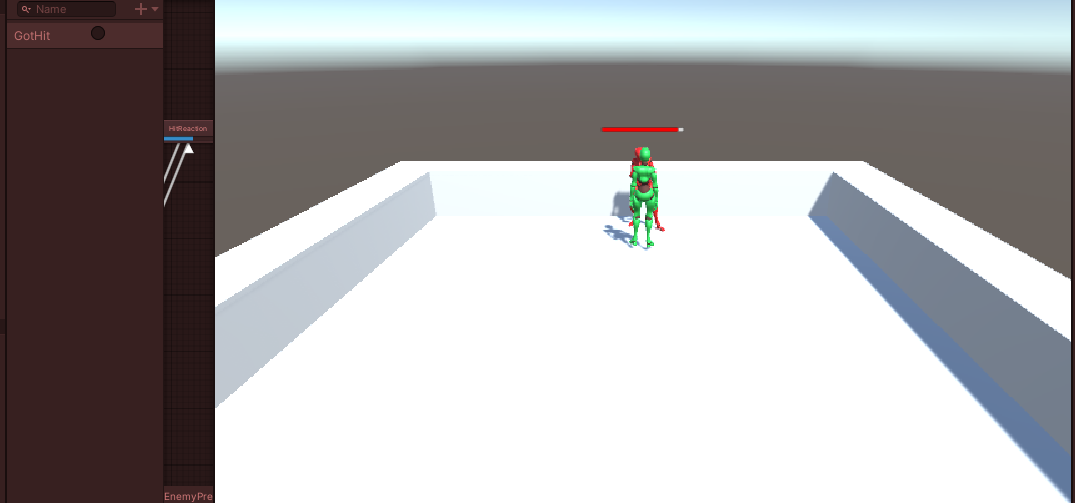
}

**Test the Health Bar**:

Run the game and check that the health bar updates correctly.

You should see the enemy with a full healthbar and when we punch it slightly decreases





## **Load the next scene when the Enemy is dead.**

Now lets modify the GameManager so that it isnt a score that changes the scene but an method that we can call from our EnemyHealth scripts :

Dont forget the **Using UnityEngine.SceneManagement**;

using UnityEngine;

using UnityEngine.SceneManagement; // Add this namespace

public class GameManager : MonoBehaviour

{

public static GameManager Instance;

private int score = 0;

public int targetScore = 4; // Score to reach before changing scenes

void Awake()

{

// Singleton pattern

if (Instance == null)

{

Instance = this;

DontDestroyOnLoad(gameObject); // Optional

}

else

{

Destroy(gameObject);

}

}

public void AddScore(int amount)

{

score += amount;

UpdateUI();

}

private void UpdateUI()

{

if (scoreText != null)

{

scoreText.text = $"Score: {score}";

}

}

public void LoadNextScene()

{

SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1 );

}

}

Note that we changed a bit

public void LoadNextScene()

{

SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1 );

// Use your new scene's name

}

Will allow us to load the next scene regardless of it’s name if it exist in the build config.

Now lets modify the EnemyHealth so that when we die we increment and move on to the next level and increments your score.

And modify the EnemyHeath script to call those function :

void Die()

{

if (GameManager.Instance != null)

{

GameManager.Instance.IncrementScore();

GameManager.Instance.LoadNextScene();

}

}

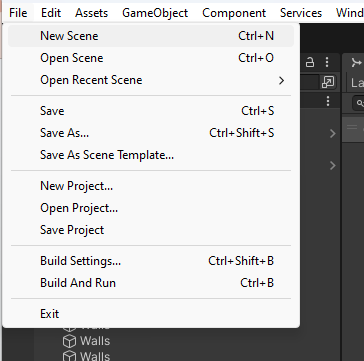
## **Add a Menu to Load the Sample Scene**

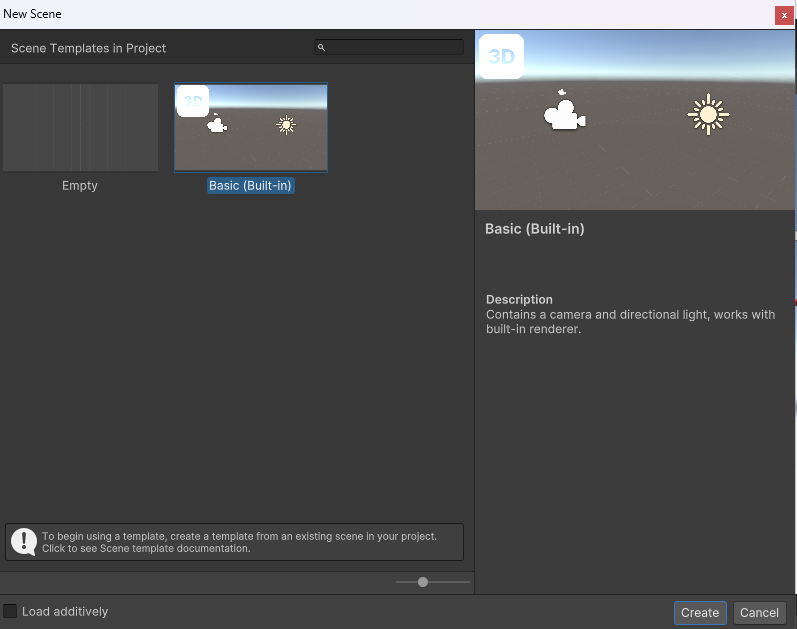
**Objective**: Create a main menu UI that allows the player to load the main game scene.

### **Steps:**

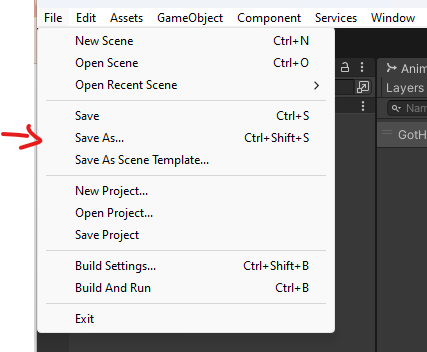
**Create a New Scene**:

Go to File > New Scene and name it MainMenu.

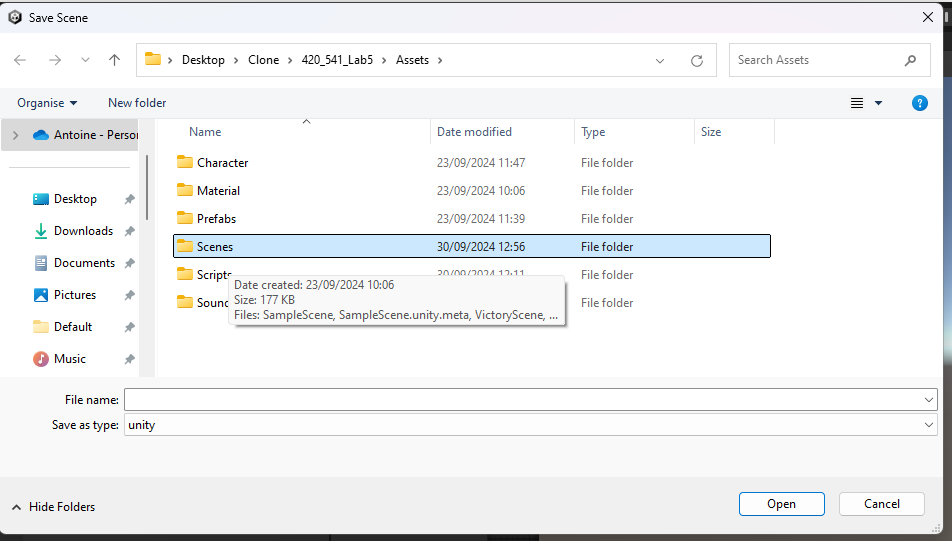


Choose Basic Built in 

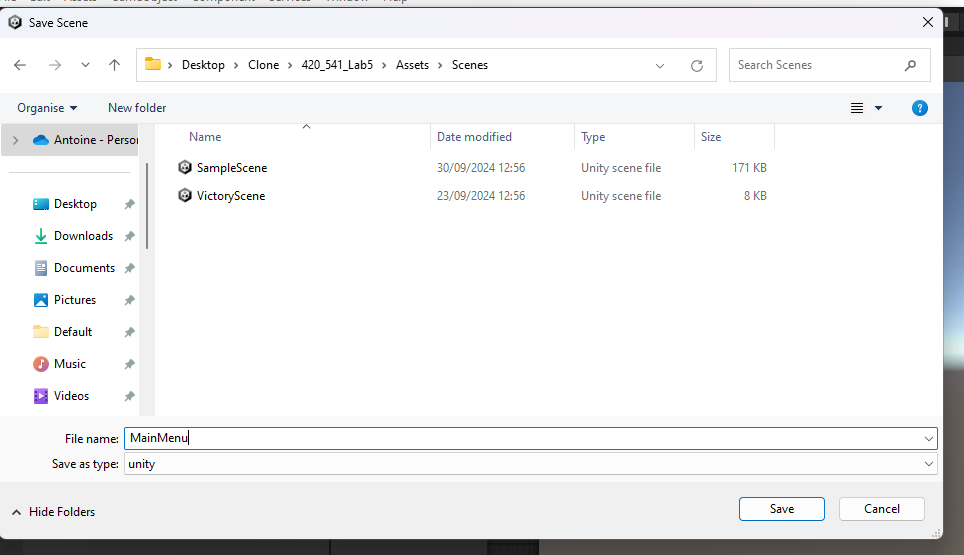
**Save it to save your new Scene**

****

**Double Click on the scene folder**

****

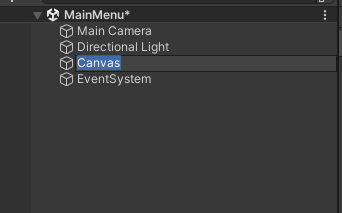
**Call your scene MainMenu**

****

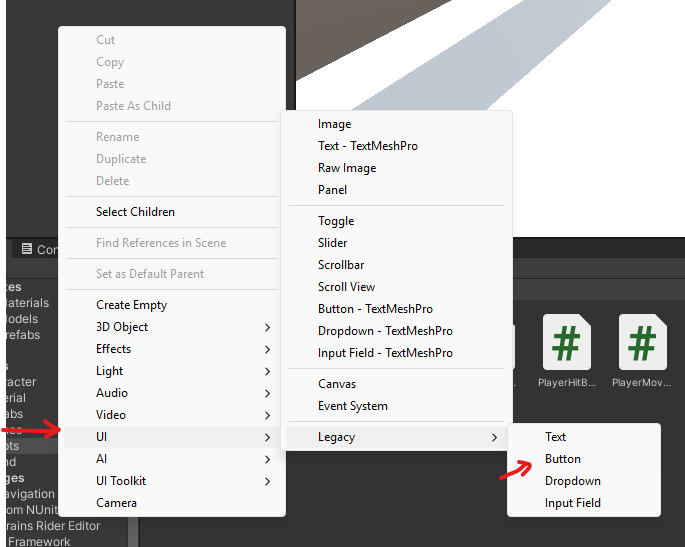
**Then click save**

**Create the UI Layout**:

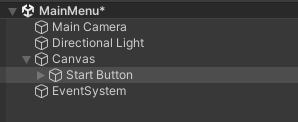
Add a Canvas (Right-click in the Hierarchy > UI > Canvas).

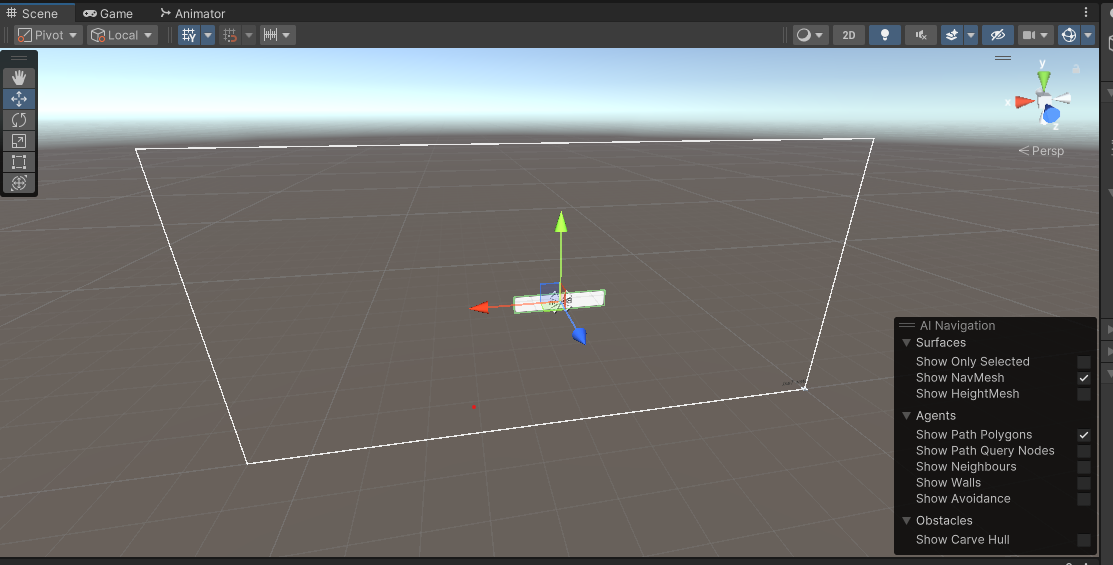


Create a Button (Right-click on Canvas > UI > Legacy >Button).

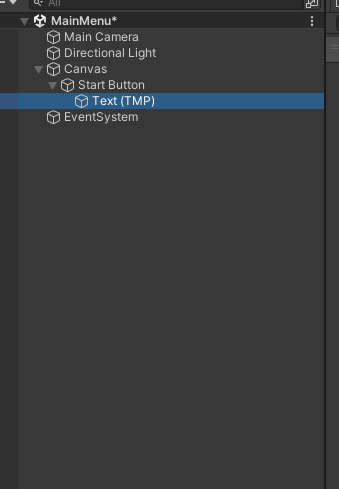


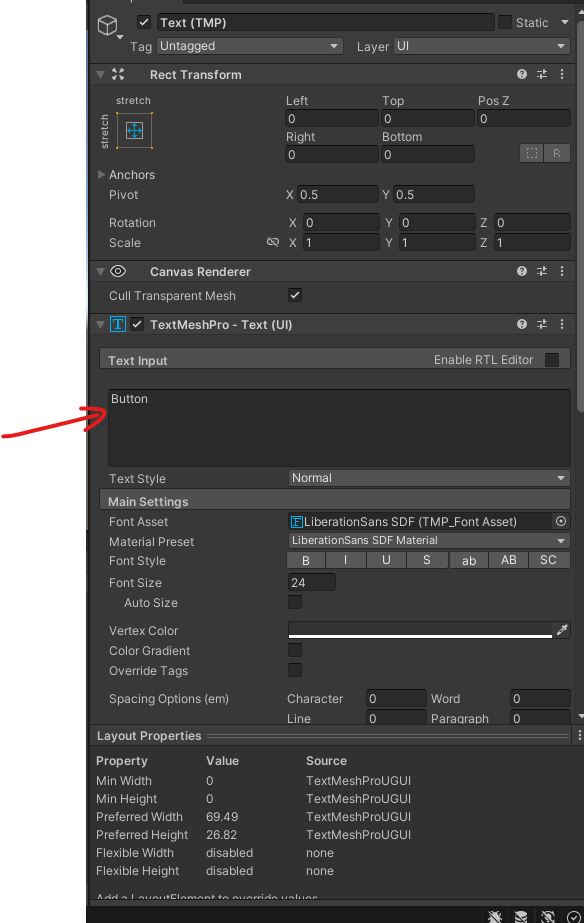
Name the Button StartButton.

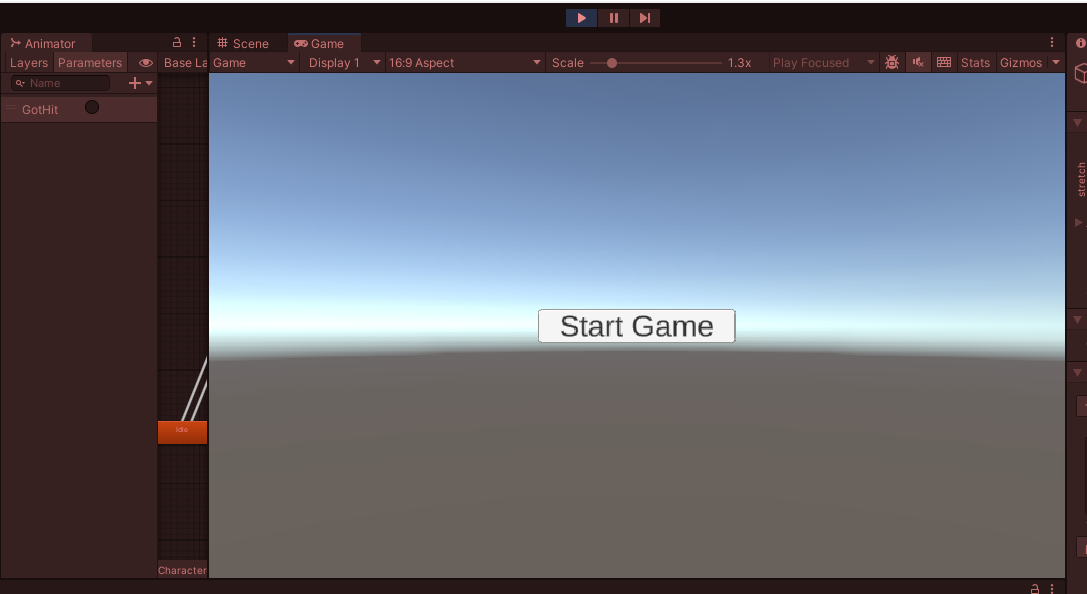


You can move your button around using the transform tool in the scene view. The white outline represent your screen.

Set the text on the button to Start Game by modifying the Text child object.

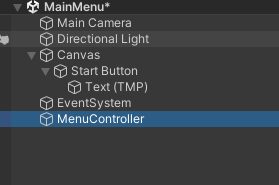




If you test your scene you should see this   


**Create a Script to Handle Scene Loading**:

Create a new script named MenuController.cs and attach it to an **empty GameObject** in the scene.



Implement the following code to load the Sample Scene:

using UnityEngine;

using UnityEngine.SceneManagement;

public class MenuController : MonoBehaviour

{

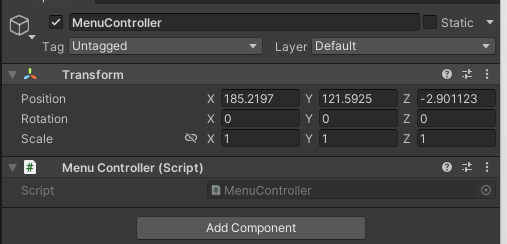
public void StartGame()

{

SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1 ); // Replace with your scene name

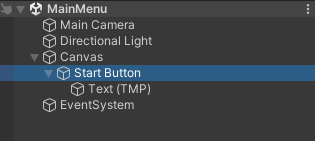
}

}

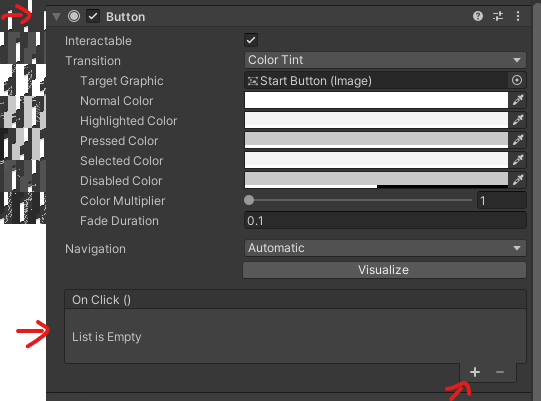


**Link the Button to the Script**:

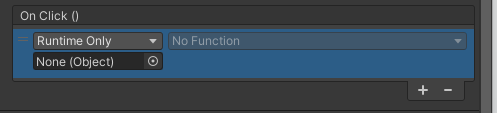
Select the StartButton object in the Hierarchy.

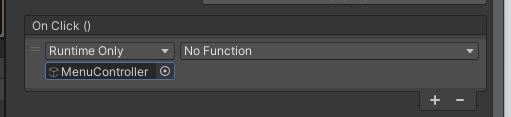


In the Button component, go to the On Click() section.

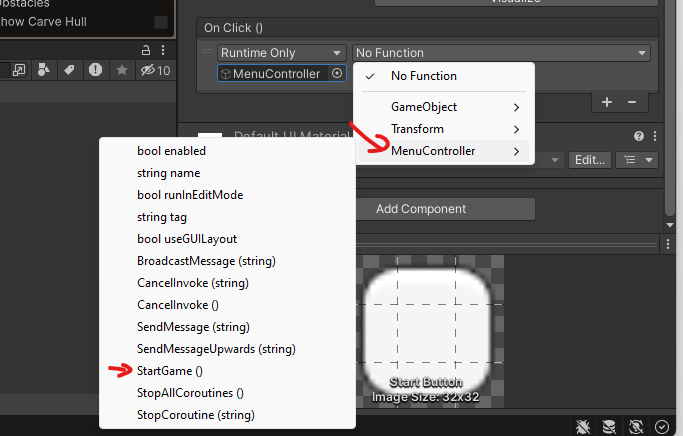


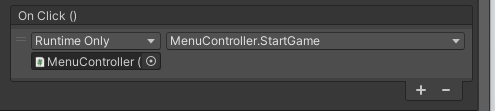
Click the + button, and drag the GameObject with the MenuController script to the field.





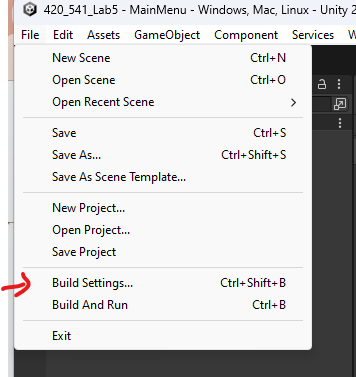
From the dropdown, select MenuController > StartGame().



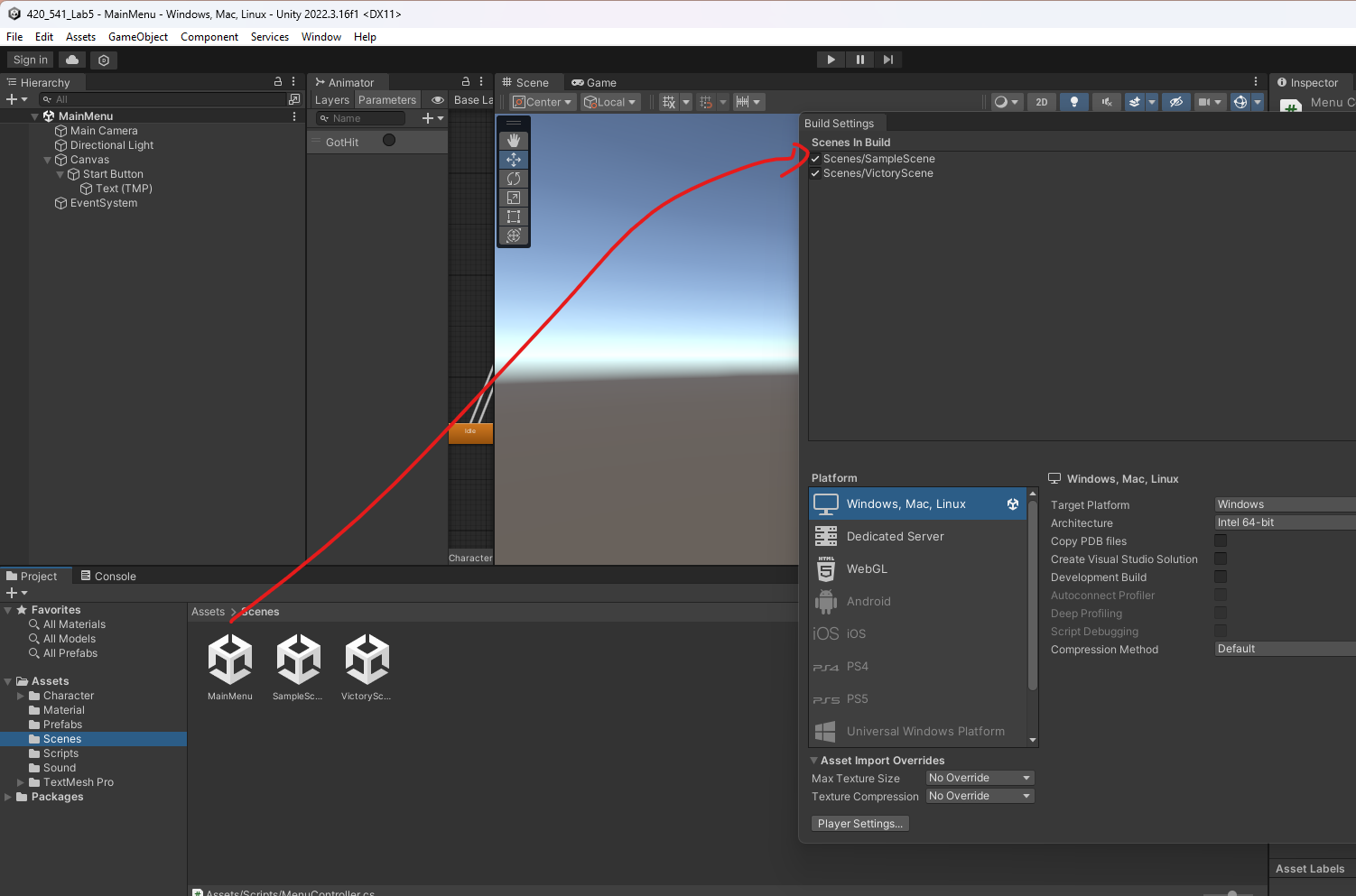
****

**Set the Scene in Build Settings**:

Go to File > Build Settings.



Add the MainMenu to the build by dragging the MainMenuScene from the Project Hierchy to the Build Settings



Make sure its the first on in the list.



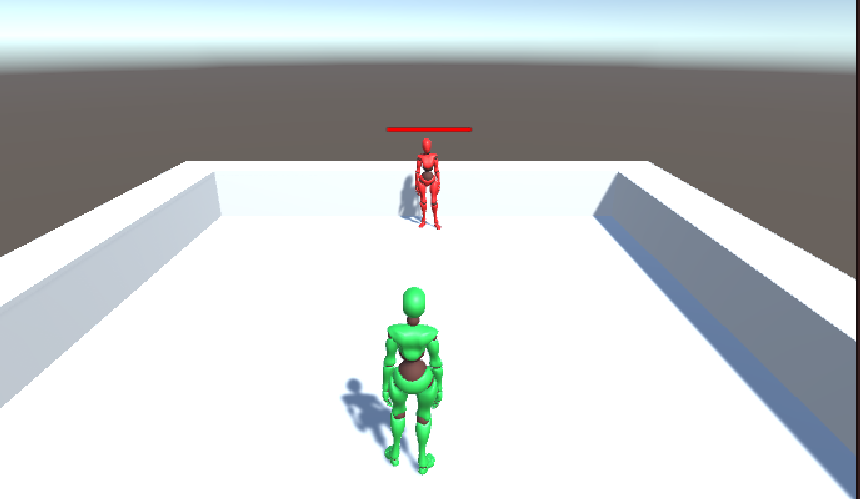
**Test the Menu**:

Save your scene

Play the MainMenu scene.

Click the Start Game button and verify it loads the SampleScene.





## 

## 

## 

## 

## 

## 

## **Add a Game Over UI with the Score from the GameManager**

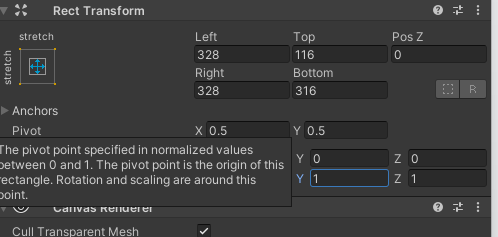
### **Steps:**

**Create the Game Over UI**:

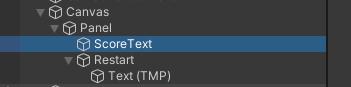
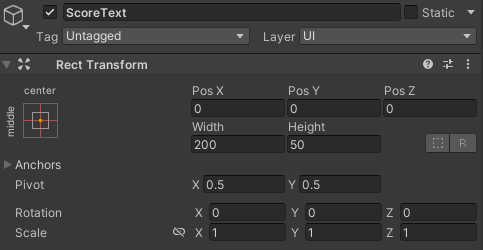
In the Victory scene, add a Canvas (Right-click in Hierarchy > UI > Canvas).

Add a Panel to the Canvas (Right-click > UI > Panel) and name it GameOverPanel.

Set the Panel coordinates to this



Add Text element and name it ScoreText. Set its position to these coordinates leave the text as Score: 0 for now.



Add a Button to the Panel and set the text to Restart.

**Create a Script to Manage the Game Over UI**:

Create a new script named GameOverController.cs.

using UnityEngine;

using UnityEngine.SceneManagement;

using TMPro; // Make sure to include the TextMeshPro namespace

public class GameOverController : MonoBehaviour

{

public TextMeshProUGUI scoreText;

public GameObject gameOverPanel;

public void Start()

{

gameOverPanel.SetActive(true);

if (GameManager.Instance)

{

scoreText.text = "Score: " + GameManager.Instance.score.ToString();

}

}

public void RestartGame()

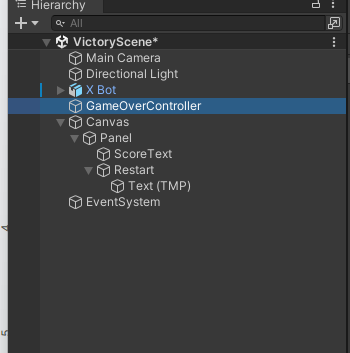
{

SceneManager.LoadScene(0);

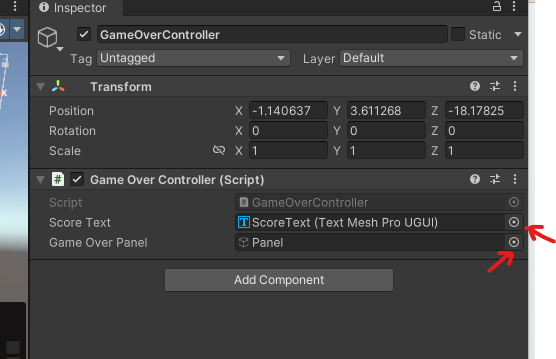
}

}

And add it to a empty game object called GameOverController

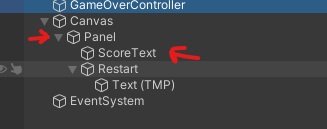


**Link the UI Elements**:

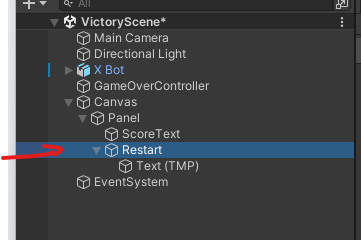


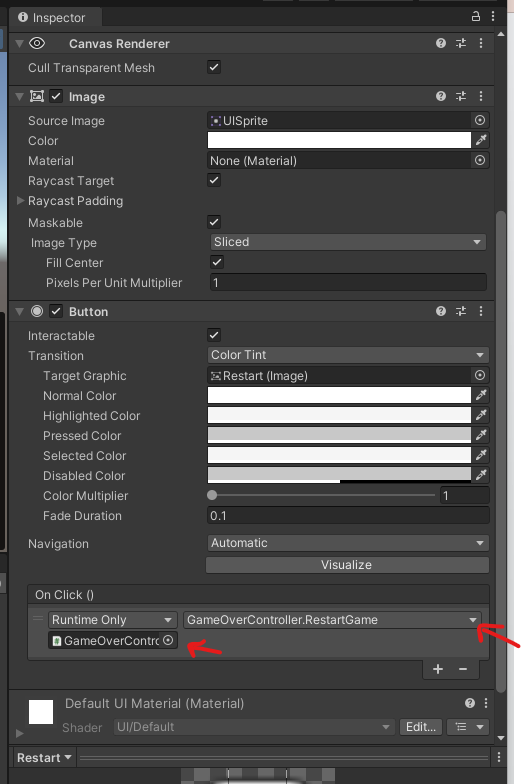
Drag the corresponding value from the Hierarchy :

In the GameOverControllercomponent, link the ScoreText and the GameOverControllerto their respective fields.



Now setup the On click for the Restart Button :





**Test the Game Over UI**:

Run the game from the First scene , trigger a game-end condition, and verify that the Game Over screen shows with the correct score.

## **Creating a pause menu**

In the **Hierarchy** window, **Right-click** → **UI** → **Canvas**. Call your new canvas Pause Menu

This creates a Canvas GameObject (if one does not exist already) and an EventSystem GameObject automatically.



## **3. Create the Pause Menu Panel**

In the **Hierarchy**, **Right-click** on the newly created **Canvas** → **UI** → **Panel**.

Rename this new Panel to something like "PauseMenuPanel".

In the **Inspector**, set up the Panel’s properties:

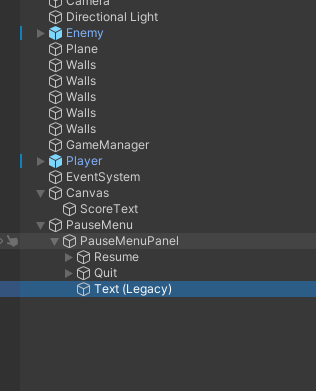
**Rect Transform**: Make sure it stretches to the entire screen (the default panel usually does).

Inside "PauseMenuPanel", you can add UI elements:

**Text** For a “Paused” label at the top.

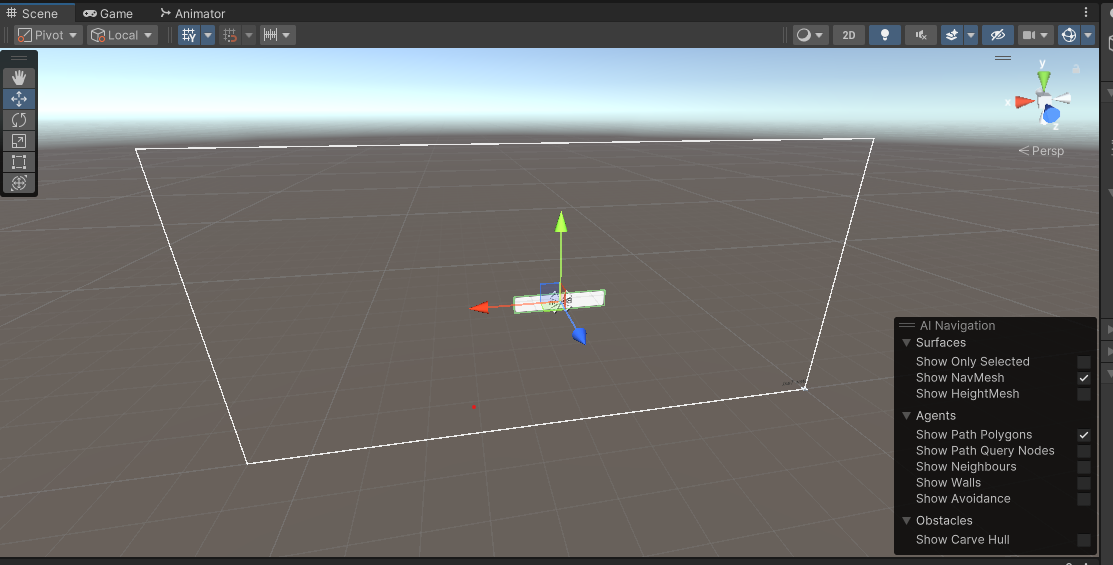
**Buttons**: For “Resume”, “Quit”, or other options.

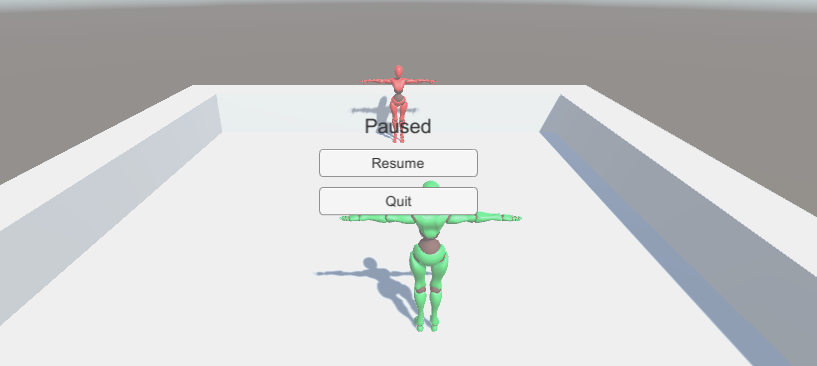
**PauseMenuPanel** (Panel)  
└ **TitleText** (Text )  
└ **ResumeButton** (Button)  
└ **QuitButton** (Button)



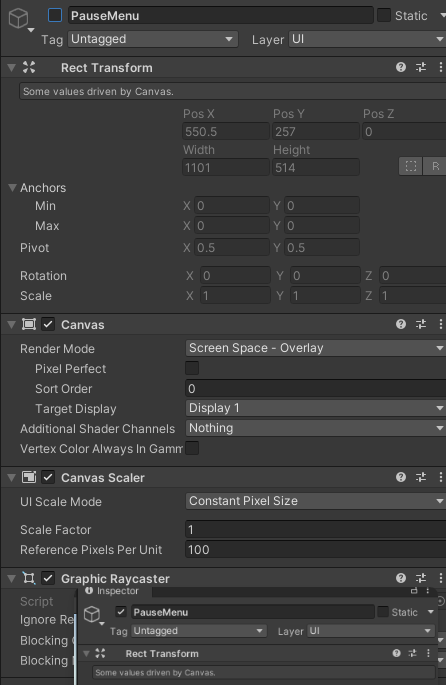
Using the transform tool in your scene view. Adjust the layout of these UI elements to your liking.

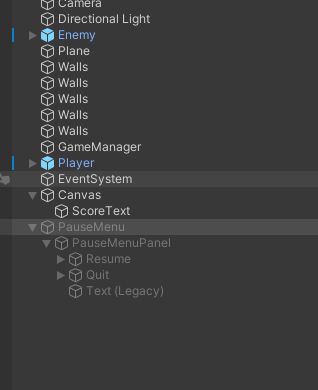
You can move the UI element to your liking using the Transform tool the white outline represent your screen.





**Deactivate** the entire "PauseMenuPanel" object in the Hierarchy initially so it doesn’t appear at the start of the game. (Uncheck the box in the Inspector next to the GameObject name.)





## **4. Create a PauseManager Script**

You need a script that can:

Detect when the player wants to pause/unpause (often the Escape key or a dedicated button).

Show or hide the pause menu UI.

Stop or resume the game’s timescale.

In your **Project** window, **Right-click** → **Create** → **C# Script** and name it "PauseManager".

Attach this script to GameManager gameobject

Open "PauseManager.cs" and add the following code:

using UnityEngine;

public class PauseManager : MonoBehaviour

{

[SerializeField] private GameObject pauseMenuPanel;

private bool isPaused = false;

void Update()

{

// Check if the player presses the "Escape" key (or any key you choose).

if (Input.GetKeyDown(KeyCode.Escape))

{

if (isPaused)

{

ResumeGame();

}

else

{

PauseGame();

}

}

}

// This method pauses the game.

public void PauseGame()

{

// Show Pause Menu UI

pauseMenuPanel.SetActive(true);

// Freeze game time

Time.timeScale = 0f;

// (Optional) Freeze audio

// AudioListener.pause = true;

isPaused = true;

}

// This method resumes the game.

public void ResumeGame()

{

// Hide Pause Menu UI

pauseMenuPanel.SetActive(false);

// Unfreeze game time

Time.timeScale = 1f;

// (Optional) Unfreeze audio

// AudioListener.pause = false;

isPaused = false;

}

// Optional: a method for quitting the game or returning to the main menu.

public void QuitGame()

{

// If you're in the editor, this won't fully work,

// but in a built application, this will quit the game.

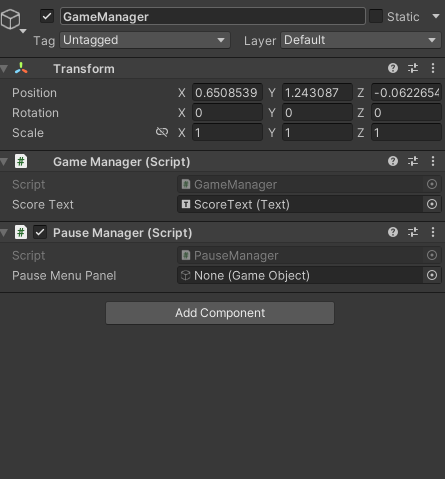
Application.Quit();

// If you have a Main Menu scene, you might do:

// SceneManager.LoadScene("MainMenu");

}

}



### **Explanation of Key Lines**

Time.timeScale = 0f; **freezes** the game’s physics, animations, and anything else that relies on Time.deltaTime.

pauseMenuPanel.SetActive(true); **shows** the pause menu panel by enabling the GameObject.

pauseMenuPanel.SetActive(false); **hides** the pause menu panel by disabling the GameObject.

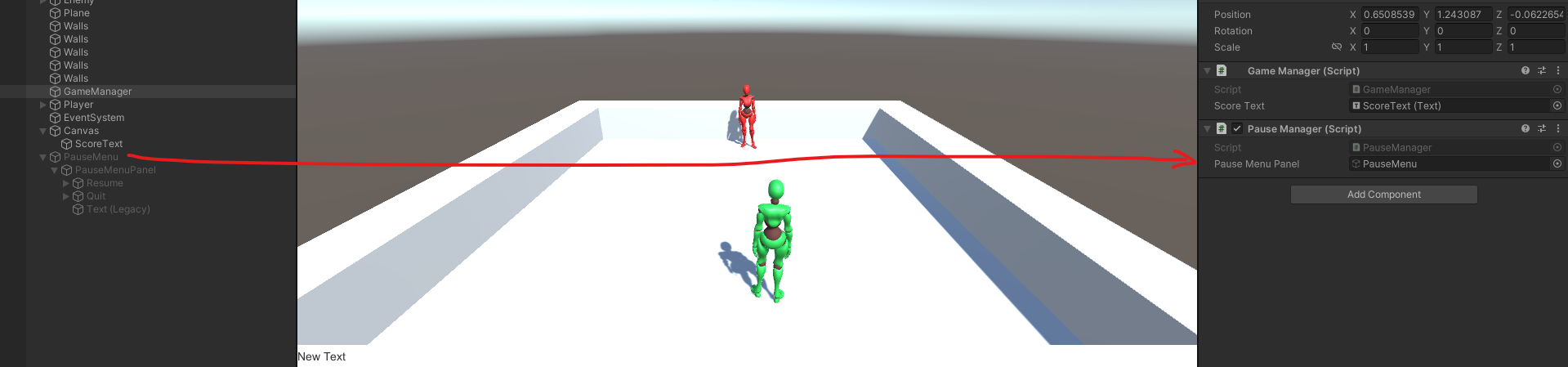
We use a bool isPaused to keep track of whether the game is currently paused.

## **5. Link the PauseMenu Canvas to the Script**

Select the **GameObject** that has the "PauseManager" script attached (e.g., "GameManager").

In the **Inspector**, under "PauseManager", you’ll see a field named "Pause Menu Panel".

**Drag and drop** the "PauseMenu" GameObject from the **Hierarchy** into that field.



This links the Canvas to the script so that the script can enable/disable it.

## **6. Configure the Resume and Quit Buttons**

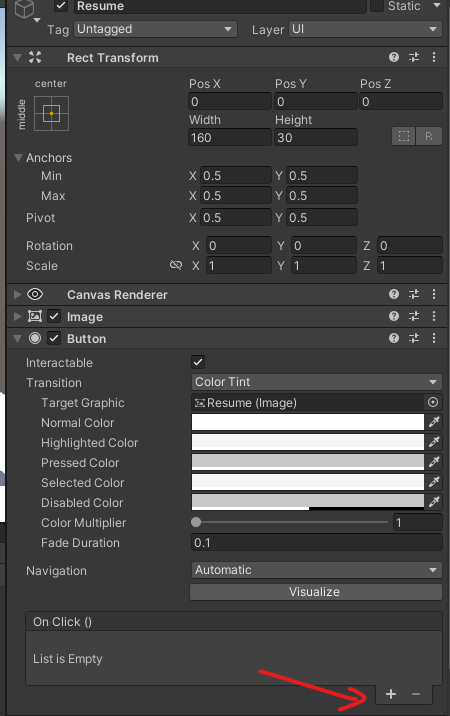
Now you have buttons in your Panel. We want them to call the "ResumeGame()" or "QuitGame()" methods in the script.

**Select** the **ResumeButton** in the Hierarchy.

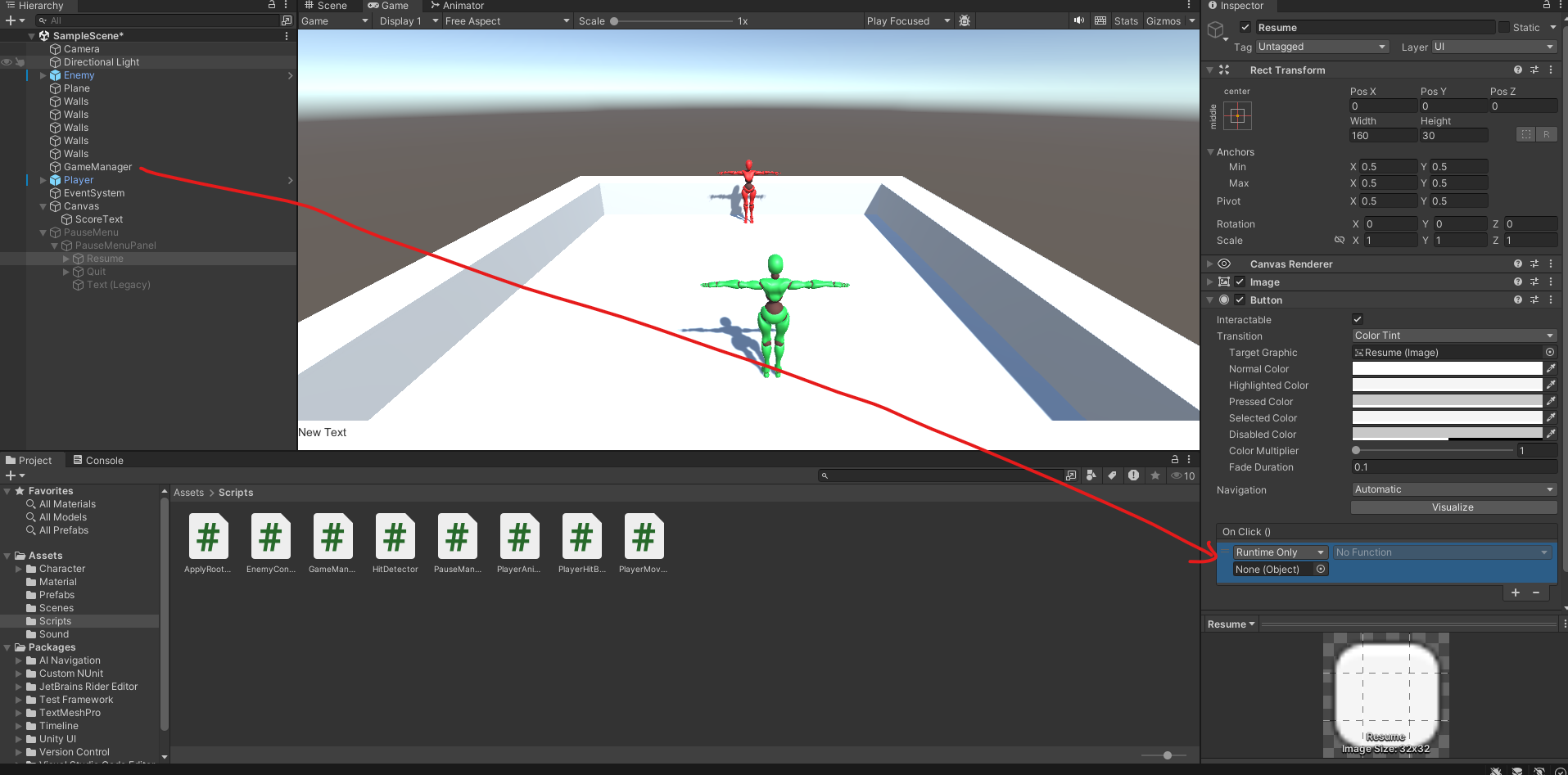


In the **Inspector**, go to the **Button** component. You’ll see an area labeled **On Click ()**.

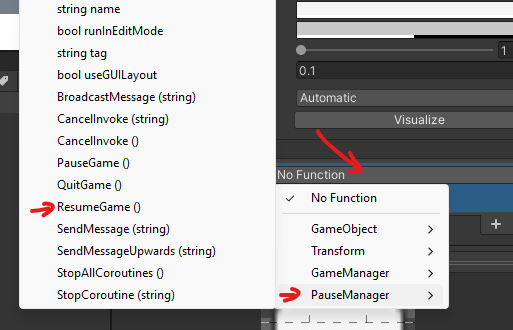
**Click the “+” icon** to add a new event.



**Drag the GameObject** with the PauseManager script into the event’s **object** field.



From the **dropdown**, select "PauseManager" → "ResumeGame()".

This sets the button so that when it is clicked, it calls the ResumeGame() method.

Repeat the steps for the **QuitButton**, but choose "PauseManager" → "QuitGame()".

## **7. Test the Pause Menu**

**Enter Play Mode** in Unity.

Press the **Escape** key (or whichever key you chose in the script).

The game should freeze (Time.timeScale = 0), and the pause menu should appear.

Click **Resume**.

The game should unfreeze (Time.timeScale = 1), and the pause menu should disappear.

(Optional) Click **Quit**.

In the Unity Editor, nothing might happen (or you’ll see a message “Application.Quit() was called”), but in a built application, the game will close.

**Submission:**

Submit your assignment on github classroom.