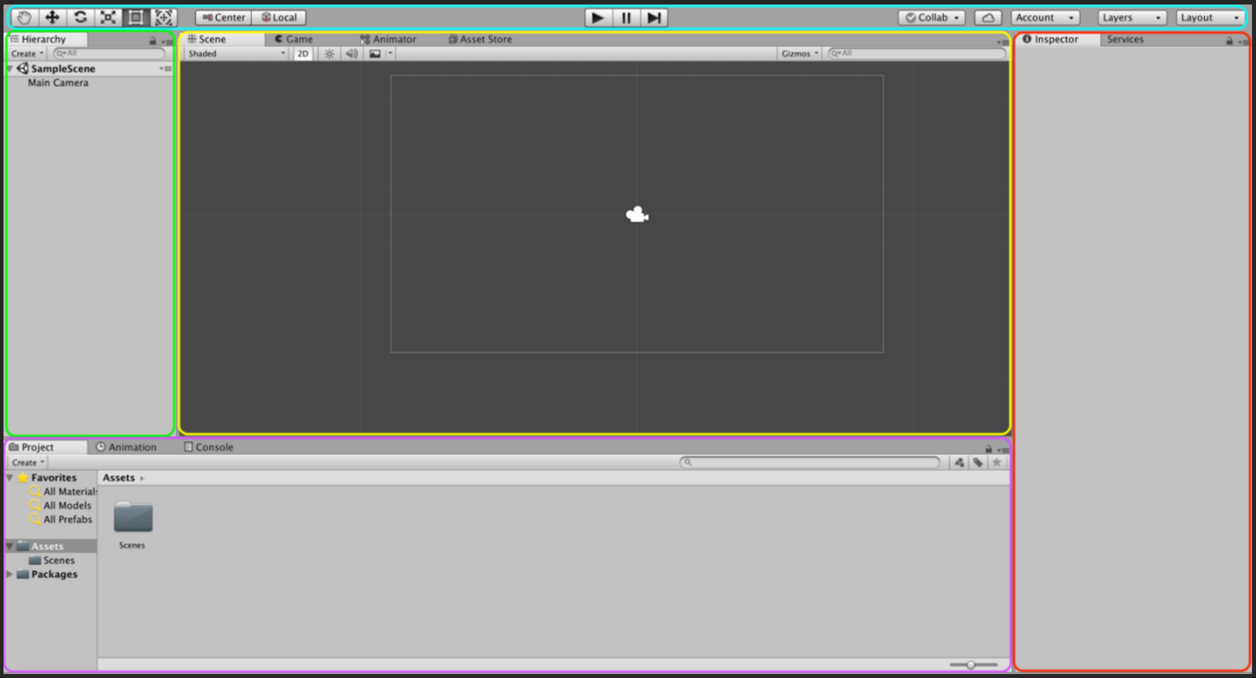
Making A Mario Game in Unity!

We begin with a newly created empty project in Unity. Here's what it looks like:



To the right we see the Inspector, which holds all the components or properties of any object in the game. A component or a property is simply a behavior attached to the object, which the object is expected to perform or follow.

The big section in the middle is the Viewport. This is a view of our entire 2D game universe. Everything placed onto the viewport exists in our game world.

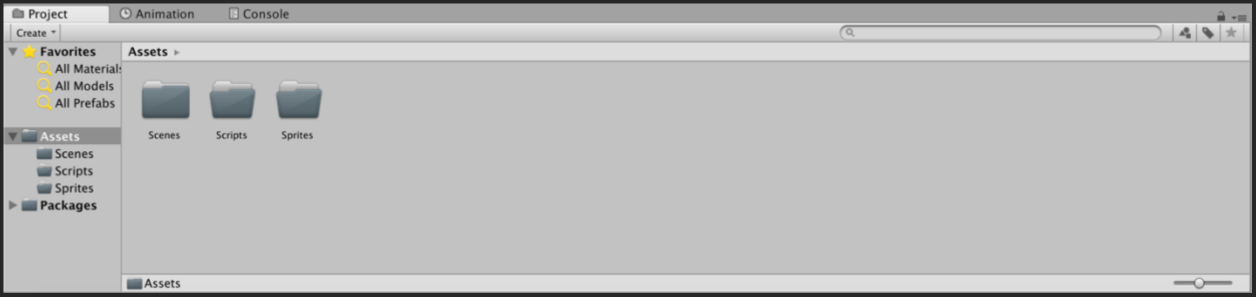
To the left is the Hierarchy, which is simply a list of all existing objects in our world. We see that an empty project starts with just a 'Main Camera' object. If we look over to our viewport, we can see a white rectangle, centered with a white Camera icon. This is our game camera. The rectangle is merely the frame of our game. We as viewers see whatever is inside that rectangle frame.

On top of these 3 sections, is the Tool Bar. We see multiple buttons in this section. The most important ones are to the left, which our transform buttons. We have the move, rotate and scale tool which we would use frequently. In the middle of this section, we see the Play, Pause and Step button. We use the Play and Pause button to test our game inside the Viewport.

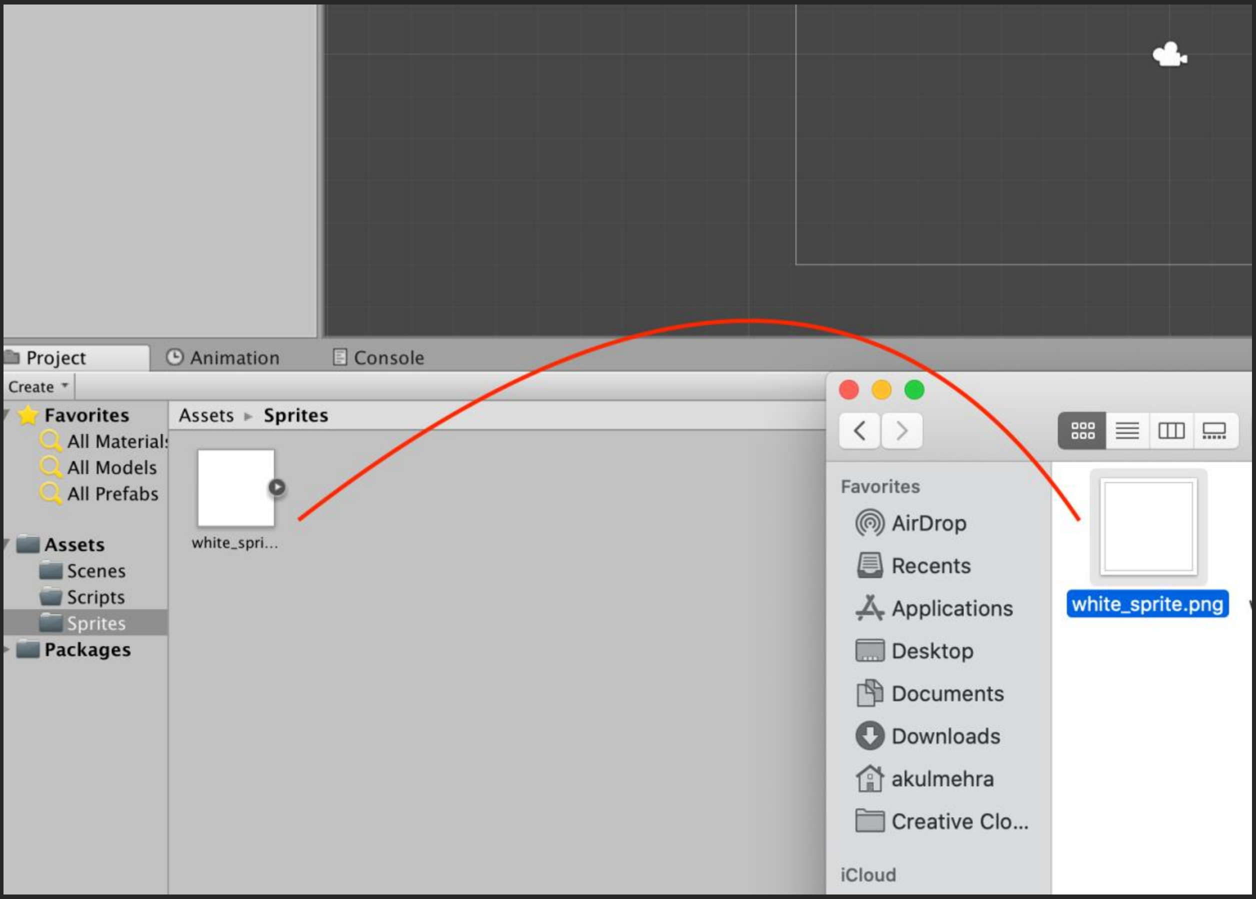
Lastly, we have the Project View at the bottom. This is Unity's way of displaying the root directory of our project. It is simply a list of all folders and files associated with our project.

The first step is to create our main player character. In this project, we're only going to use a plain white square image as artwork. If you are on Windows, I recommend simply saving an empty white image using MS Paint. If you are on Mac, I recommend going into a folder with a lot of white space and taking a square screenshot of just the white space. The picture need not be a perfect square.

To make sure our project stays organized, we segregate our files. Right click in the project view, Create → Folder. Create 2 folders apart from the already made Scenes folder, name them '**Scripts**' and '**Sprites**'. Our project view should look like this.



Now, open the Sprites folder, and from outside Unity, drag and drop the white square image into your project view.



As this image is now in our project view, it is a file existing in our project, which can be used anywhere in our game. We'll use this white square for to create every object in our game.

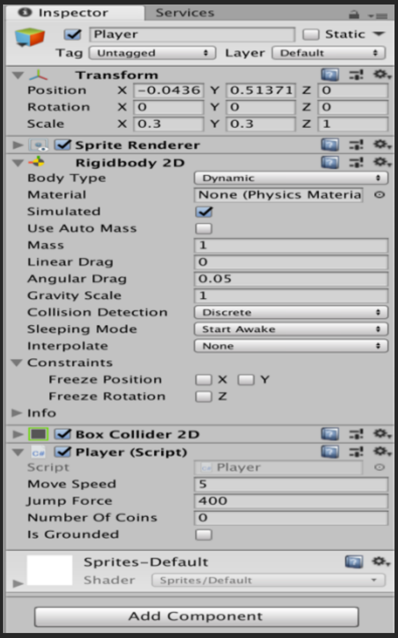
From the project view, drag and drop the white square onto either the Viewport or the Hierarchy. This adds an instance of that square into our project.

Having selected the newly added square, if we look over to the Inspector, we see it has 2 properties, Transform and Sprite Renderer.

-In the Hierarchy, right click on the white square and click rename, and rename it to "Player".
-In the Inspector, move over to the Transform property. Change the 'Scale' value to 0.3 for X and 0.3 for Y
-In the Inspector, click the Add Component button, search for "Rigidbody2D" and click Add.
-Inside Rigidbody2D, change the Collision Detection to 'Continuous'. Drop down the Constraints menu, and check Freeze Rotation Z.
-Click the Add Component button again, search for "Box Collider 2D" and click Add.

At this point, go back to the Github repository, and download all the Scripts provided there. If possible, download them straight into your 'Scripts' folder in Unity. Otherwise, move each script into the Scripts folder by dragging and dropping.

Once you have all the scripts, add the 'Player' script to the Player object by either dragging and dropping it onto the Player in the Hierarchy or Viewport, or the Player's Inspector. Now a new component of type 'Player (Script)' would be visible in the Player's Inspector. Change the Move Speed to 5 and Jump Force to 400. These values can be altered later. Now our Player's Inspector should look like this.

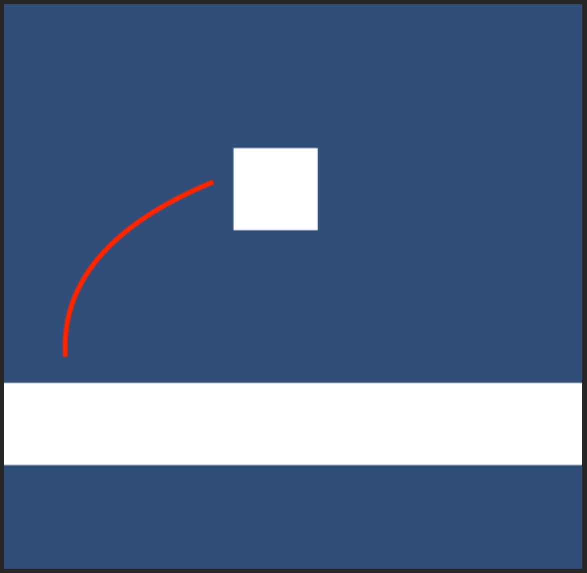


```
Inspector    Services
[✓] Player                          [ ] Static
Tag  Untagged          Layer  Default
Transform
  Position   X -0.0436   Y 0.51371   Z 0
  Rotation   X 0         Y 0         Z 0
  Scale      X 0.3       Y 0.3       Z 1
[✓] Sprite Renderer
Rigidbody 2D
  Body Type              Dynamic
  Material               None (Physics Materia
  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
[✓] Box Collider 2D
[✓] Player (Script)
  Script                 Player
  Move Speed             5
  Jump Force             400
  Number Of Coins        0
  Is Grounded            [ ]
Sprites-Default
  Shader  Sprites/Default
Add Component
```

To test out the movement of the Player, we need a platform for the Player to move on. Simply drag and drop another instance of the white square from the Sprites folder onto the Viewport. Rename this new instance to "Ground".

Having selected the Ground object:
-In the Inspector, click Add Component and search for "Box Collider 2D", and click Add.
-Move over to the Transform, and change the Scale values to 10 for X and 0.3 for Y.
-Finally, select the Ground object in the Viewport, hit the W key to use the Move tool, or select the move tool from the top corner in the Tool Bar. And move the Ground to somewhere below the Player by dragging the vertical arrow.

Hit the Play button in the Tool Bar and test out the game.

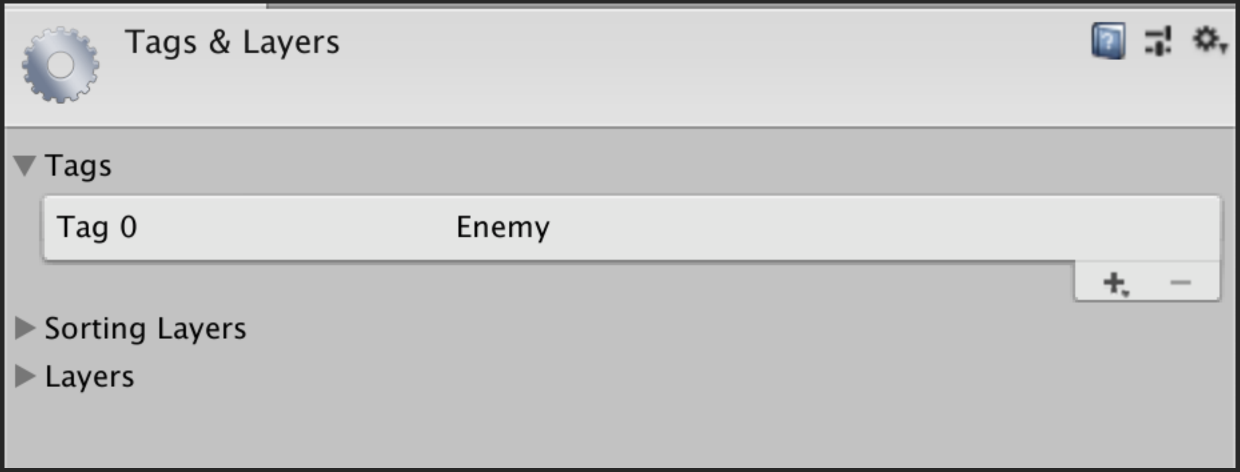


Now let's build and Enemy for this Player.

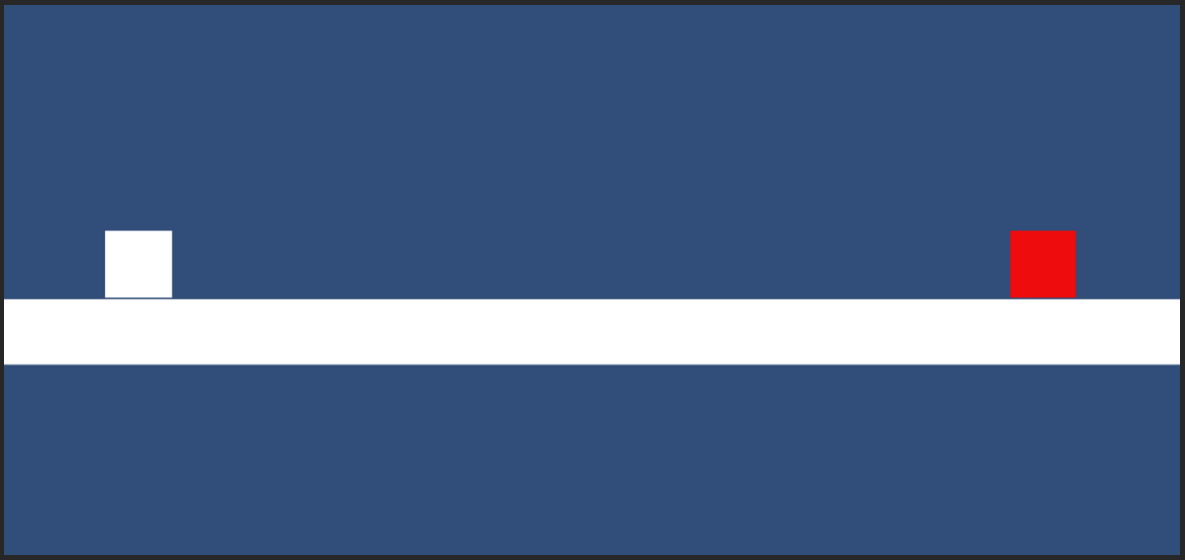
Drag and drop another instance of the white square and rename it to "Enemy".
At this point, we need to add a 'Tag' to our Player, as this is required in the EnemyAI script. Select the Player, and in the Inspector, drop down the 'Tag' under the object name. In the dropdown menu, there are some predefined tags that are frequently used in projects. Select the "Player" tag. Make sure the text for the tag is exactly the same as "Player".

Now select the Enemy:
-In the Inspector, click Add Component and search for “Box Collider 2D”, and click Add.
-In the Inspector, click the Add Component button, search for “Rigidbody2D” and click Add.
-Inside Rigidbody2D, change the Collision Detection to ‘Continuous’. Drop down the Constraints menu, and check Freeze Rotation Z.
-From the project view, add the EnemyAI script onto the Enemy’s Inspector.
-In the EnemyAI script, change the Move Speed to **4** and Min Distance to **7.**
-Move to the Sprite Renderer in Enemy’s Inspector, and change the Sprite Color to a bright red.

Just as we added a “Player” tag to our Player, select the Enemy object and click the Tag dropdown menu. As there is no predefined “Enemy” tag, click Add Tag.



In the Tags & Layers menu, click the ‘+’ icon. Name the tag “Enemy” (make sure it is exactly the same) and hit Save.
Now reselect Enemy and add the Enemy tag from the dropdown menu. Our scene now looks something like this.



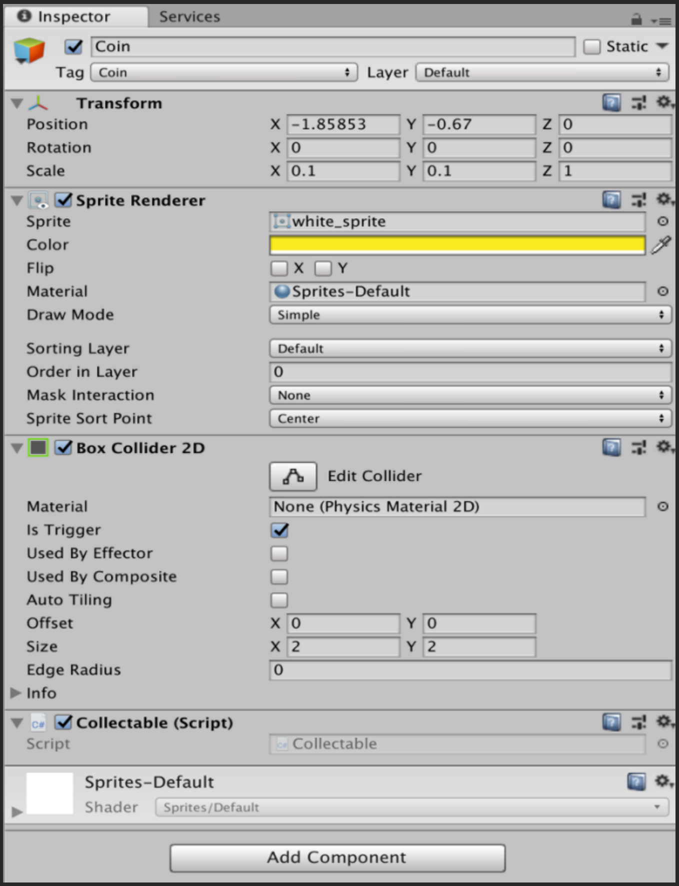
The Enemy should now be charging towards left when Player is within minimum distance of the Enemy and also be killing the Player on collision. The Player should be able to kill the Enemy by jumping on top.

Next we're going to work on our Collectables. For this project, we'll only create collectable Coins.

For our coin, drag and drop another instance of the white square into the Viewport. Rename it to "Coin".

Having selected the Coin:
-Add a "Coin" tag to the Coin's Inspector. You'll have to create a new tag.
-In the Transform property, change the Scale values to 0.1 for X and 0.1 for Y.
-In the Sprite Renderer, change the Sprite Color to a Golden Yellow.
-Using the Add Component button, add a Box Collider 2D to the Coin.
-Check the 'Is Trigger' box inside the Box Collider 2D component.
-Finally, add the 'Collectable' script from the downloaded scripts by dragging and dropping it into the Inspector.

The Inspector for Coin should now look like this.



✓ Coin
Tag Coin   Layer Default

Transform

Position	X -1.85853	Y -0.67	Z 0
Rotation	X 0	Y 0	Z 0
Scale	X 0.1	Y 0.1	Z 1

✓ Sprite Renderer

Sprite	white_sprite
Color	🟨
Flip	☐ X ☐ Y
Material	Sprites-Default
Draw Mode	Simple
Sorting Layer	Default
Order in Layer	0
Mask Interaction	None
Sprite Sort Point	Center

✓ Box Collider 2D

Material	None (Physics Material 2D)
Is Trigger	✓
Used By Effector	☐
Used By Composite	☐
Auto Tiling	☐
Offset	X 0 Y 0
Size	X 2 Y 2
Edge Radius	0

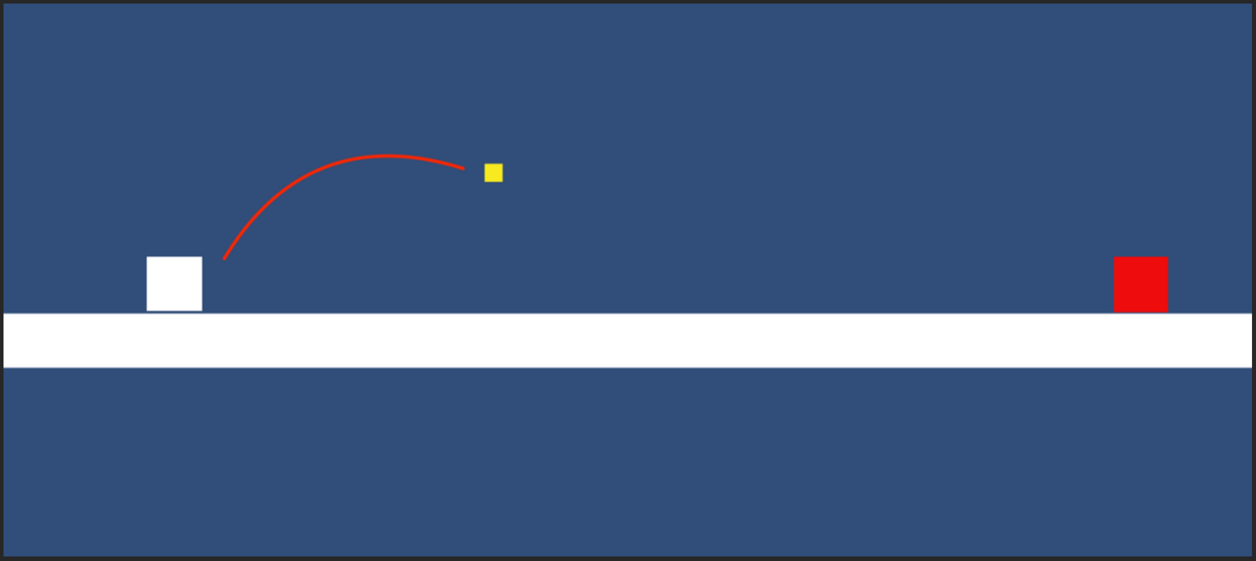
Info

✓ Collectable (Script)

Script Collectable

Sprites-Default
Shader Sprites/Default

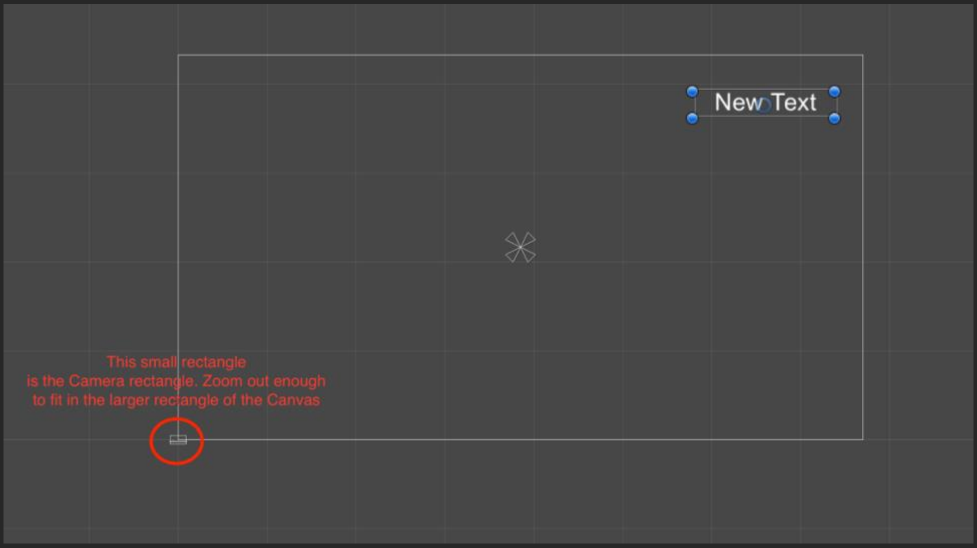
And our updated scene looks like this.



Next we are going to build a simple User Interface to display our score.

Go over to the Hierarchy, and right click on empty grey space. Go down to UI → Text.
This will create a Canvas object, and inside the Canvas object it will create our Text object that we need. Select that Text object inside the Canvas (If only Canvas is visible, click on the down arrow next to Canvas), and rename it to "ScoreText".

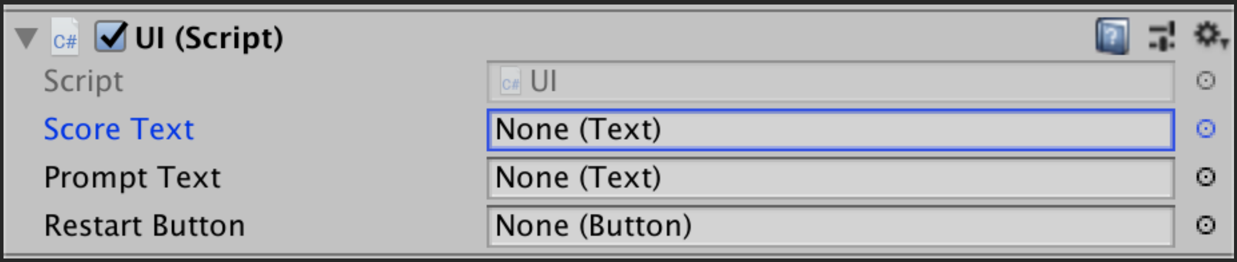
Having the ScoreText selected:
-Zoom out in the Viewport so that the entire Canvas is visible to you. It will be much larger that the Camera frame. Once zoomed out, select the ScoreText and move it to the top right corner of the Canvas rectangle. Like this.


New Text
This small rectangle
is the Camera rectangle. Zoom out enough
to fit in the larger rectangle of the Canvas

-Keeping ScoreText selected, move over to the Text property. Change the font size to 12. Under 'Paragraph' center the **Alignment** both horizontally and vertically. Change the Color to **White.** Finally, check the box for 'Best Fit'.

Next, right click in the Hierarchy and select Create Empty. This creates an Empty game object without any behaviors attached to it. The only property attached to it is the Transform. Rename this empty game object to "GameManager".

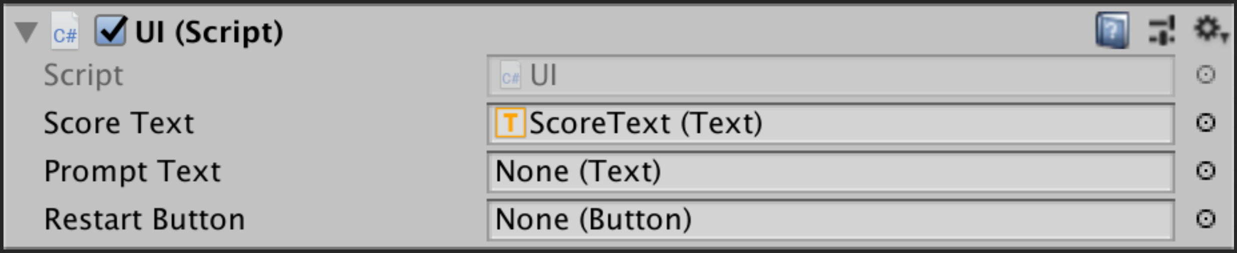
Having selected this GameManager, add the UI script to it. The UI script on the GameManager looks like this.



UI (Script)

Script	UI
Score Text	None (Text)
Prompt Text	None (Text)
Restart Button	None (Button)

Keep the GameManager selected and drag and drop our ScoreText object onto the Score Text field in the UI script on GameManager's Inspector. Changing it to something like this.



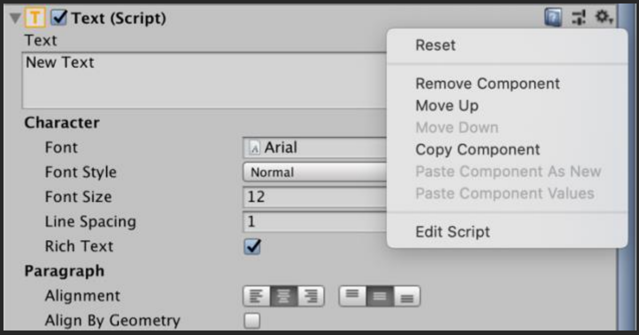
UI (Script)

Script	UI
Score Text	ScoreText (Text)
Prompt Text	None (Text)
Restart Button	None (Button)

Next we will build a simple Prompt Text mechanism, that displays appropriate prompts as and when required.

This is already programmed into our UI script. There is even a slot for a PrompText. So simply right click on the Canvas and create another Text. Make sure to right click on the Canvas and not on empty grey space in the Hierarchy, as that will create another Canvas object which could cause bugs. Rename this new text object "PromptText". We require the same fonts and properties as the ScoreText we created, hence go over to the ScoreText.

On the Text component, click the gear icon on the top right, and hit copy component.



Text
New Text
Character
Font Arial
Font Style Normal
Font Size 12
Line Spacing 1
Rich Text ✓
Paragraph
Alignment
Align By Geometry

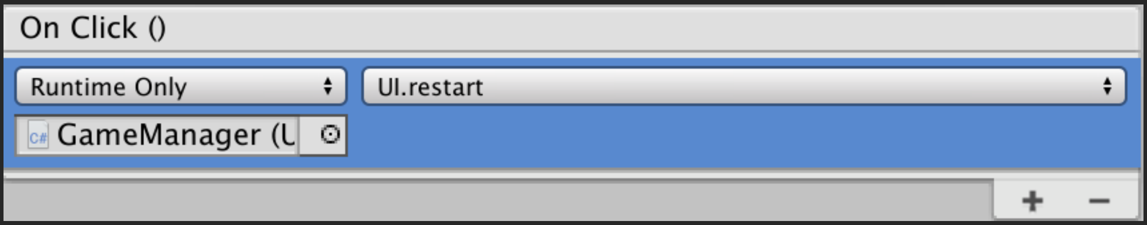
Now select the PromptText. Click the same gear icon and hit ‘Paste Component Values’. Place this PromptText object somewhere in the middle of the Canvas, you’ll have to zoom out to view the Canvas entirely.

The last UI component we need in this scene is a **restart button.**

We follow a similar process for this. Right click on the Canvas object in the Hierarchy, and UI→Button. Rename the Button object to “RestartBtn”. Place this button right below the PromptText object. Now, click on the down arrow next to RestartBtn in the Hierarchy, and select the Text object inside it. Moving over to the Inspector, change its Text value in the Text component. In order to do that, tap into the text box, and type **“Restart?”**.

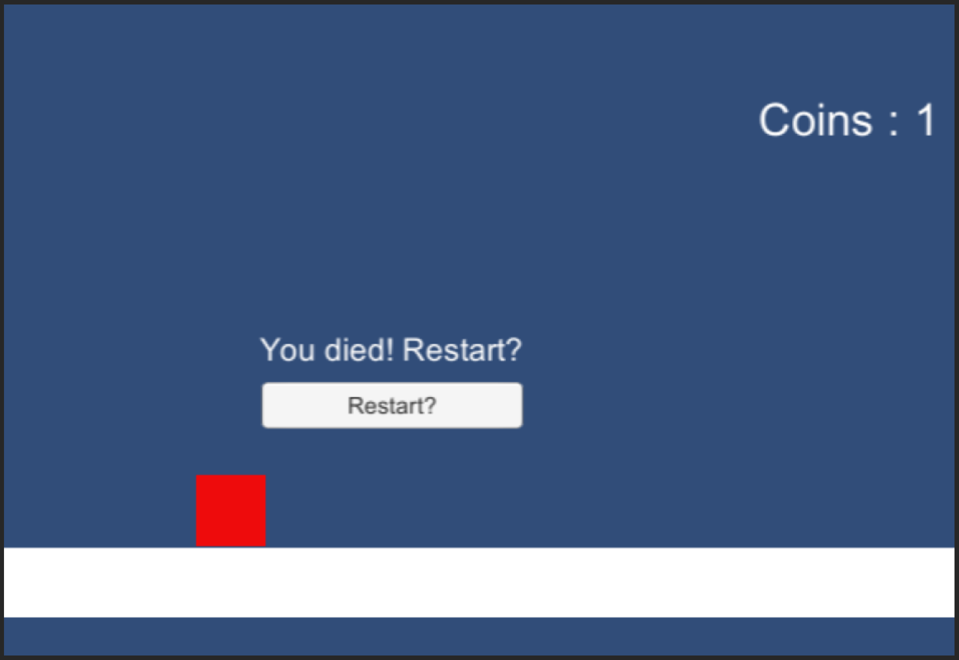
Lastly, select the GameManager object, move over to the Inspector and drag and drop the RestartBtn onto the Restart Button field in the UI script component.

Our displaying mechanism for all UI elements should work now. However, our Restart Button is still ineffective. To make it work, select the RestartBtn object. Go to the Inspector and move over to the On Click() section in the Button script component. Click on the ‘+’ icon. In the GameObject field drag and drop the GameManger. Open the dropdown menu next to it where it says, ‘No function’. Go to UI→restart(). The section should look something like this.



On Click ()
Runtime Only
UI.restart
GameManager (U

Now our UI system is working flawlessly. Here is what our scene looks like.



An important aspect of any game is the goal of the game. Our game is yet to have that aspect to it. Let's work on a simple location-based goal, which means that in order to complete the level our Player would have to physically reach that point.

Drag and drop another instance of the white square. Rename it to "LevelComplete".

Having selected LevelComplete:
-In the Inspector, click Add Component and search for "Box Collider 2D" and click Add. Once added, check the 'Is Trigger' box for this collider.
-In the Transform property, change the Scale to 3 for Y, keeping 1 for X.
-In the Sprite Renderer, click on the Color field to open up the palette. Pick a bright green color. Now, turn down the 'A' value below the R, G and B values, till the square becomes fairly translucent. 'A' stands for Alpha value of the color, and refers to the level of transparency.
-Lastly, drag and drop the 'completeLevel' script onto the LevelComplete object's Inspector. Drag and drop the PromptText, RestartBtn and the Player object each into their respective available fields.

Our game now has a goal and the mechanism for completing the level works well.

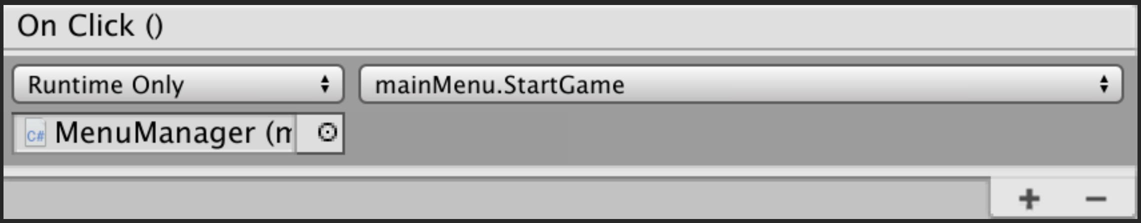
A game also requires a Main Menu in order for the player to promptly find the right part, and not get lost.

A main menu in our project would be a separate scene, from the scene we were working on right now. Firstly, hit File→Save Scenes. Now hit File→New Scene. This will create an empty scene with an empty Hierarchy.

For this scene, we only require 3 UI elements.
Right click on the Hierarchy to add in a Text object. Rename this to "Title" and change the Text value in the Text component to "Mario Prototype", or something more creative. Next, right click on the Canvas object to add two buttons. Name them, "PlayBtn" and "QuitBtn". Change the Text property of each of PlayBtn and QuitBtn's attached Text objects to ''Play Game" and "Quit Game".

Now, just as we had a GameManager script to overlook general game logistics, we need a MenuManager to handle menu events. Create an empty game object. Rename it "MenuManager" and add the 'mainMenu' script to it from the project view.

The mainMenu script has all the functions we need. We simply need to assign them on the correct events. Select the PlayBtn, move over to the On Click() section of the Button component in the Inspector. Click the '+' button to add on On Click() event. Drag and drop the MenuManager in place of the required game object. In the dropdown menu beside it, select mainMenu→StartGame(). This is what it should look like.

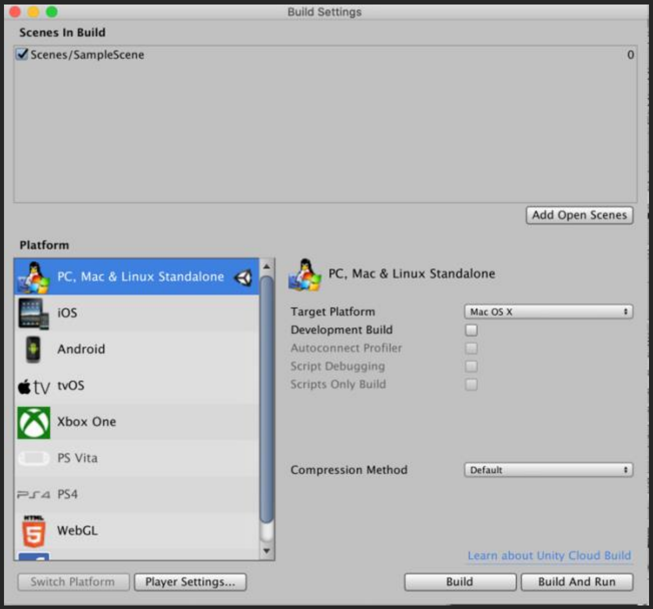


Similarly, select QuitBtn, move over to the On Click() section. Drag and drop the MenuManager, and select the QuitGame() function.

Our programming and logistics are all set up. We need to do one last thing in order for the main menu to work.

As of now, the mainMenu script is attempting to launch the next scene after this scene, but it has no clue which scene is the next scene. There is a list in Unity that describes the order in which the scenes in your game are built. Open File→Build Settings.

A window like this should appear.

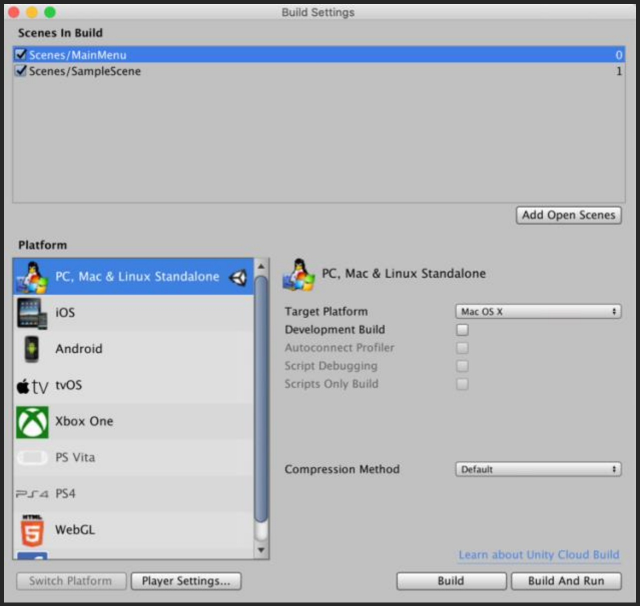


Scenes/SampleScene 0

PC, Mac & Linux Standalone
Target Platform: Mac OS X
Development Build
Autoconnect Profiler
Script Debugging
Scripts Only Build
Compression Method: Default

The list we just talked about is the 'Scenes In Build' list. Right now, it only has our main game scene which is called 'SampleScene' in our list. To add the current MainMenu scene, click the 'Add Open Scenes' button. The MainMenu scene would be added on top of the SampleScene, which means at index 1 instead of index 0 (the index of the scene is the number to its right on the list).

We want our MainMenu to load before any other scene, hence we require it to be at index 0. Simply drag the MainMenu scene over the SampleScene. Our Build Settings now look like this.



Scenes/MainMenu 0
Scenes/SampleScene 1

PC, Mac & Linux Standalone
Target Platform: Mac OS X
Development Build
Autoconnect Profiler
Script Debugging
Scripts Only Build
Compression Method: Default

And that’s a wrap for development. Our game is now fully ready for production. In fact, we’re just a click away from building it. In the Build Settings window, select ‘PC, Mac & Linux Standalone’ under Platform. On the right side, select your required Target Platform. Note that in case of Mac OS X, it will build a .dmg file and in case of Windows it will build an executable .exe file. Finally, hit ‘Build and Run’.