

How To Implement Unit Tests

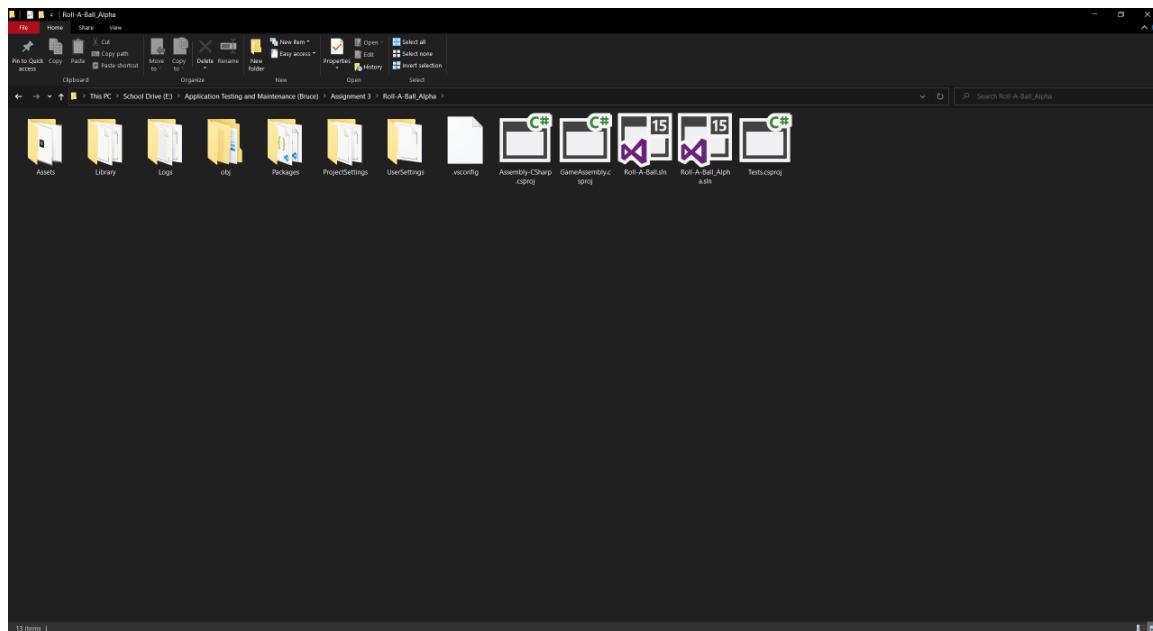
Table of Contents

Setup Project for Testing.....	2
Setup Unit Tests.....	15
Running The Unit Tests.....	30

Setup Project for Testing

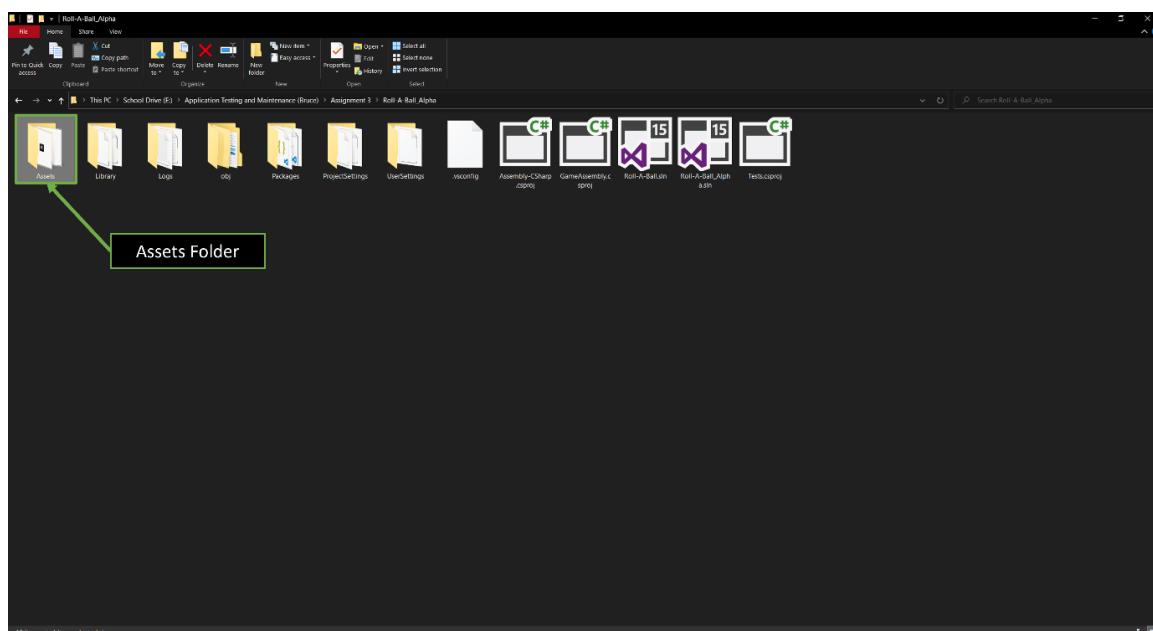
Step 1

- Navigate to the “Roll-A-Ball_Alpha” project directory using file explorer



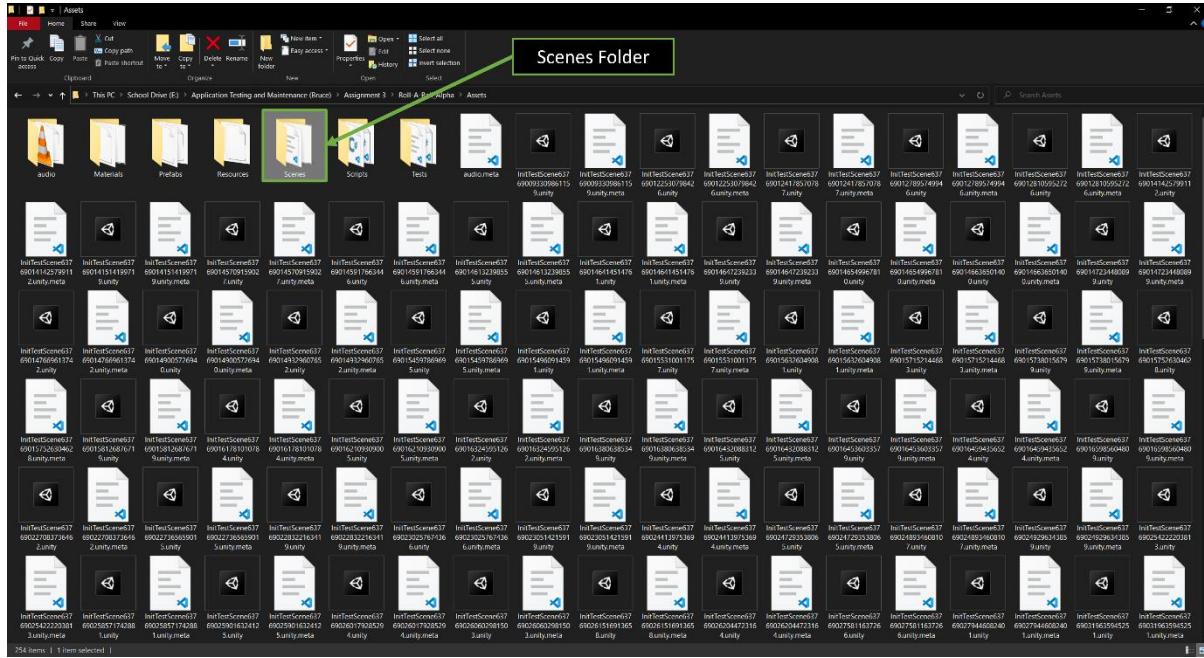
Step 2

- Left click twice on the “Assets” folder to open it



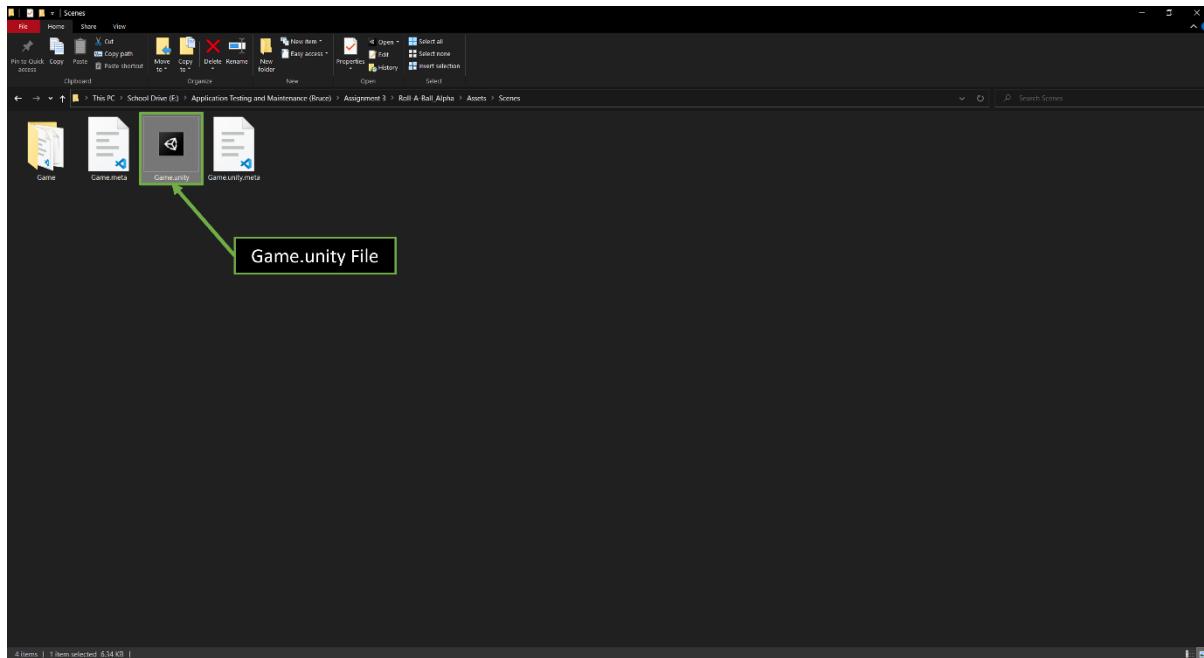
Step 3

- Left click twice on the “Scenes” folder to open it



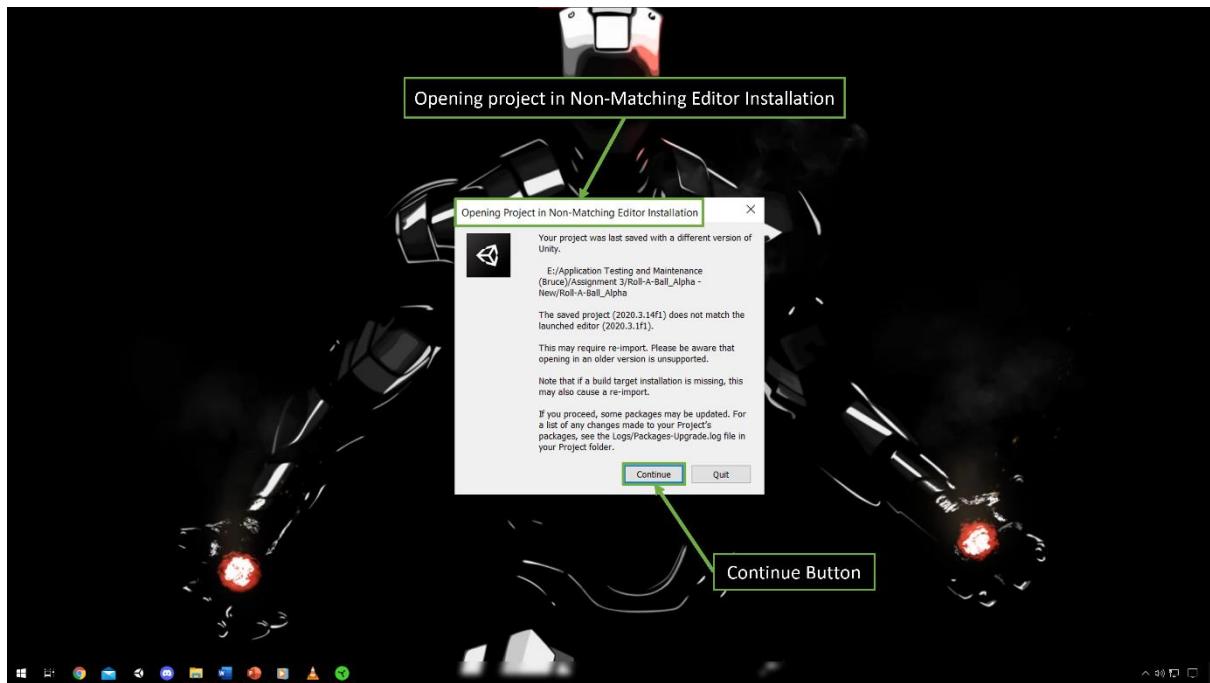
Step 4

- Left click twice on the “Game.unity” file to open the “Roll-A-Ball_Alpha” project



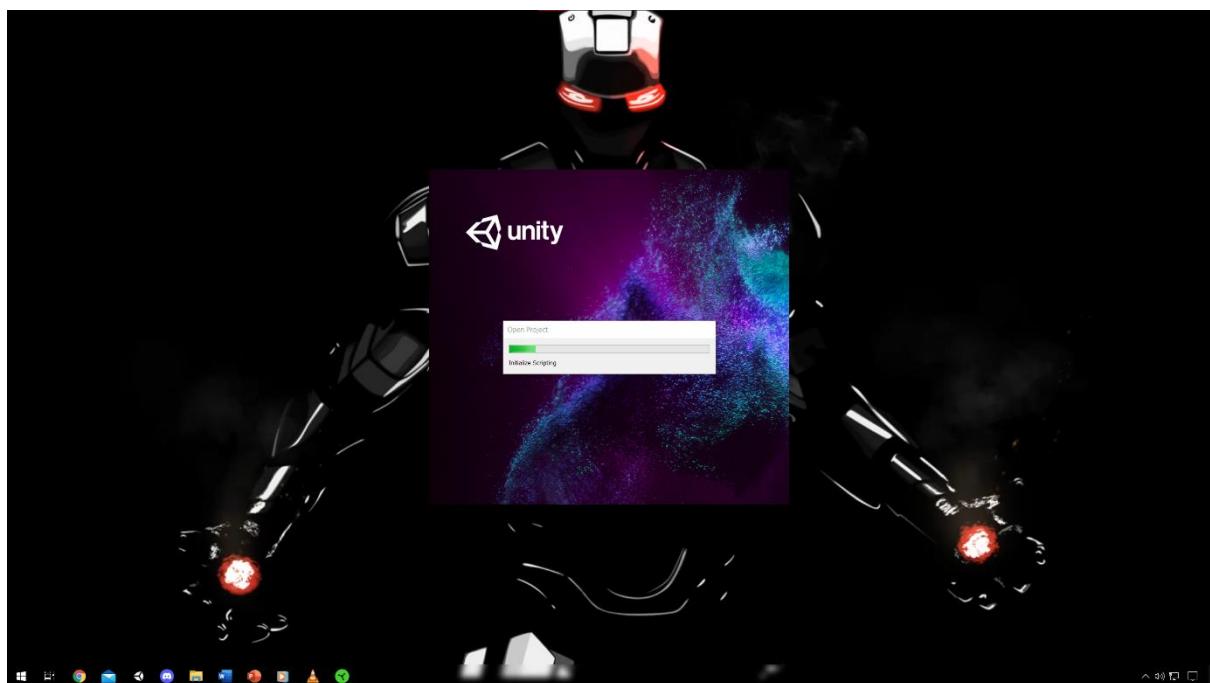
Step 5

- If a window appears titled “Opening Project in Non-Matching Editor Installation”, left click once on the “Continue” button



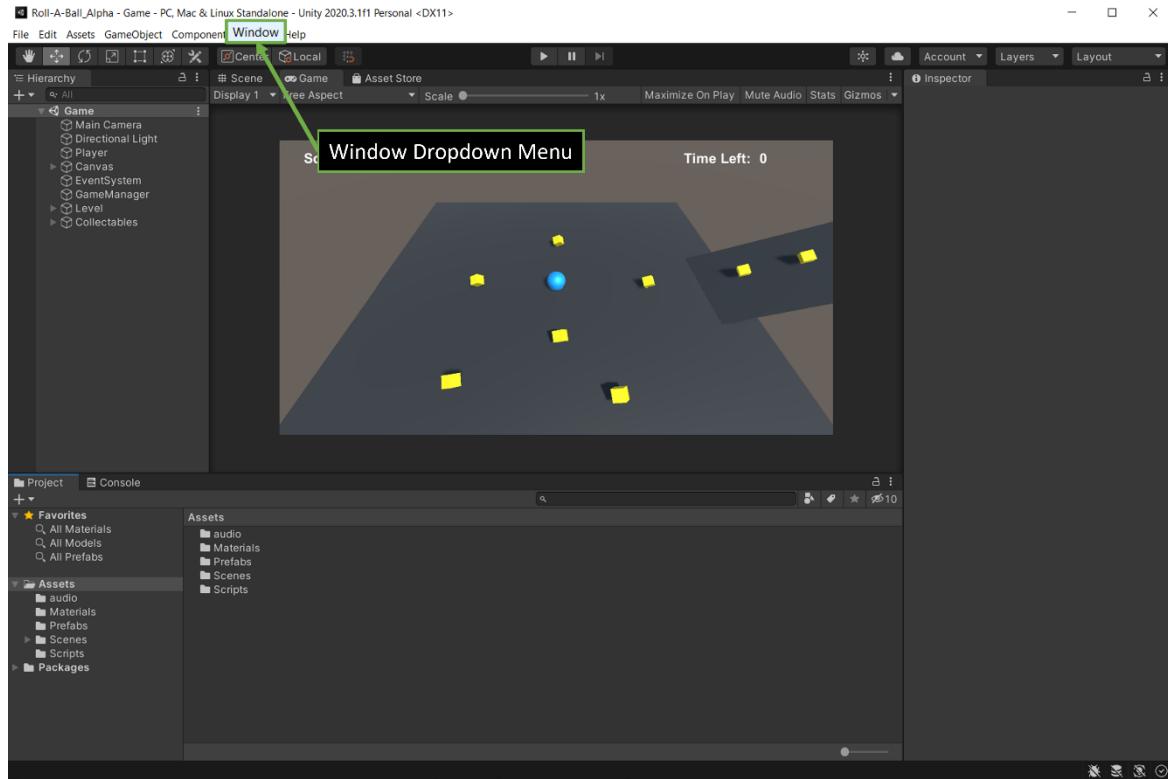
Step 6

- Wait for the unity project to open



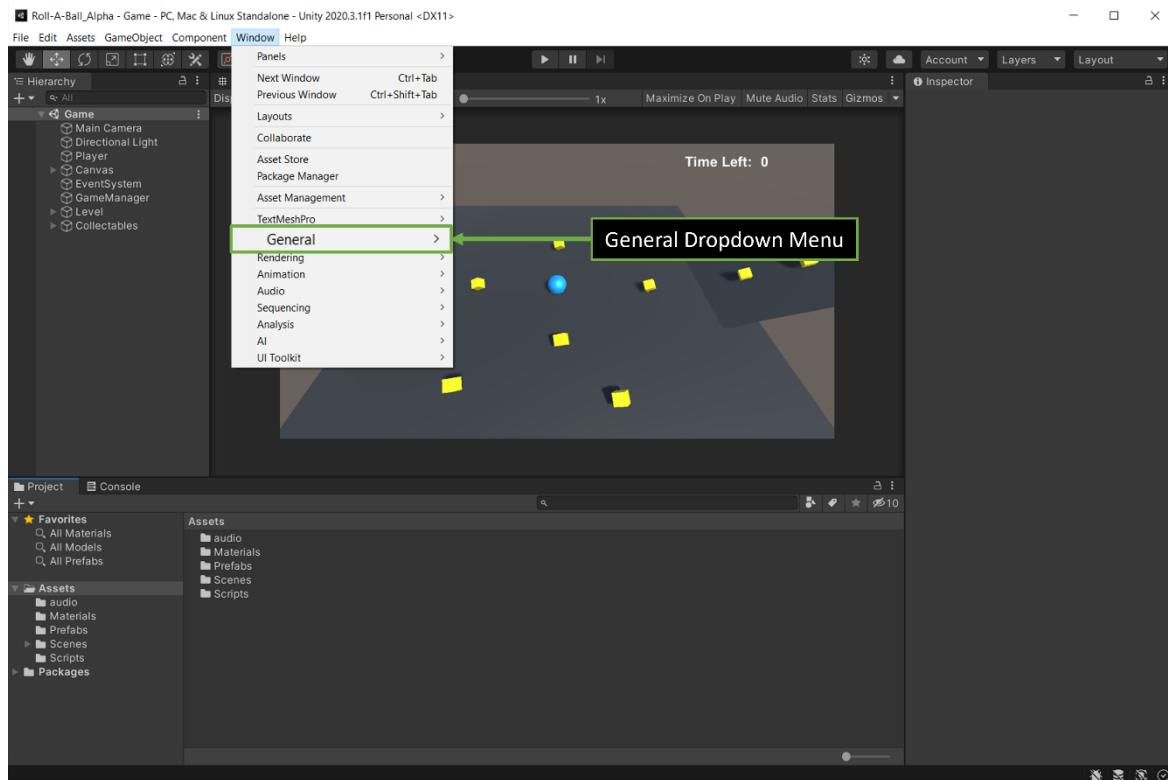
Step 7

- Once the unity project has opened, left click once on the “Window” dropdown menu at the top of the unity editor



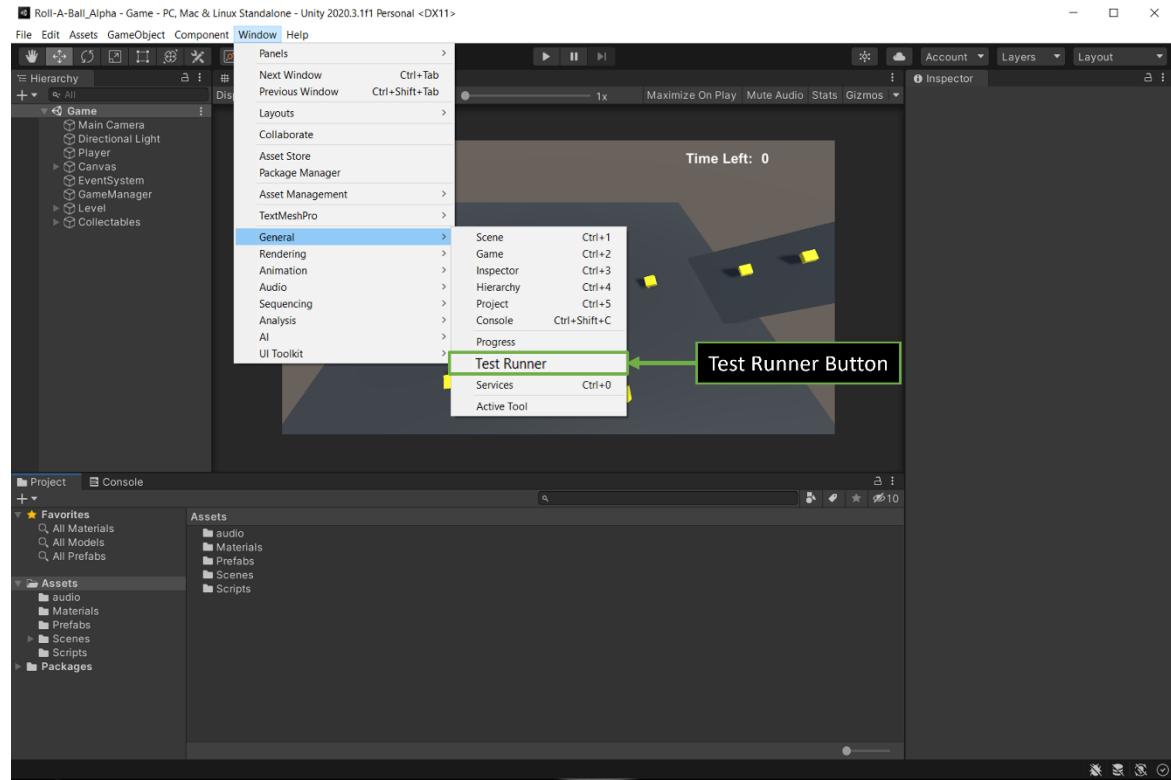
Step 8

- Hover over the “General” dropdown menu within the “Window” dropdown menu



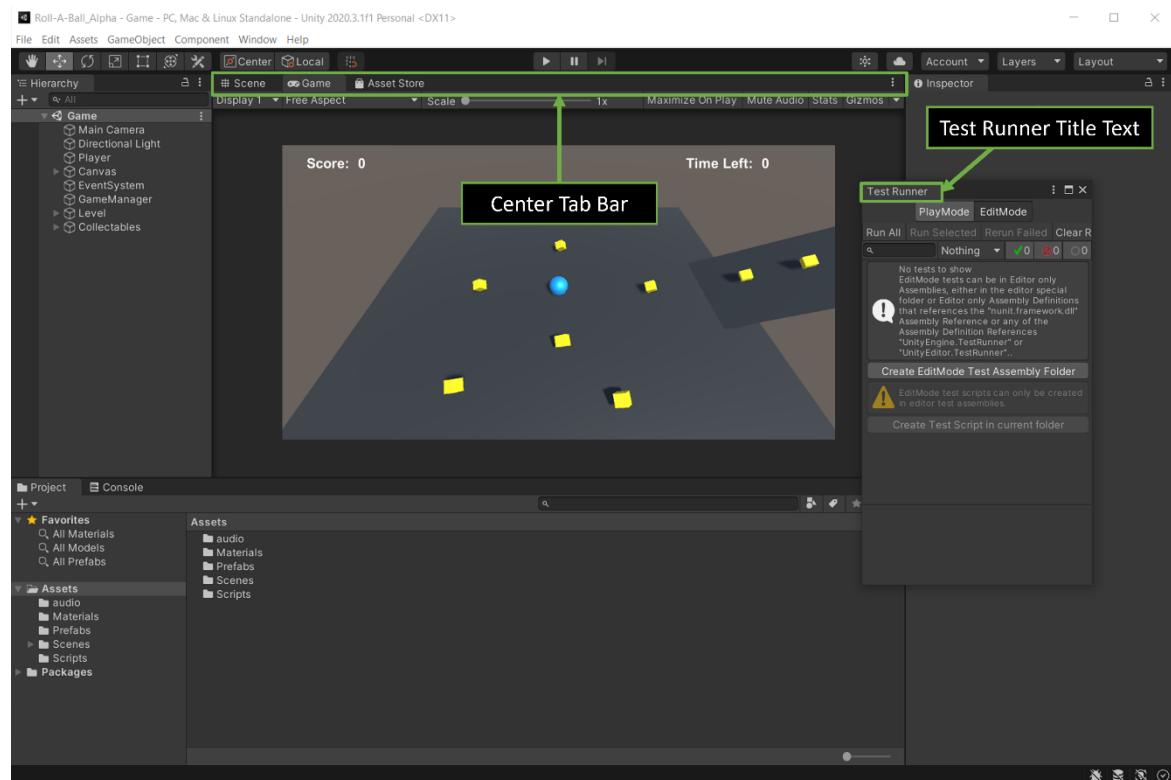
Step 9

- Left click once on the “Test Runner” button within the “General” dropdown menu



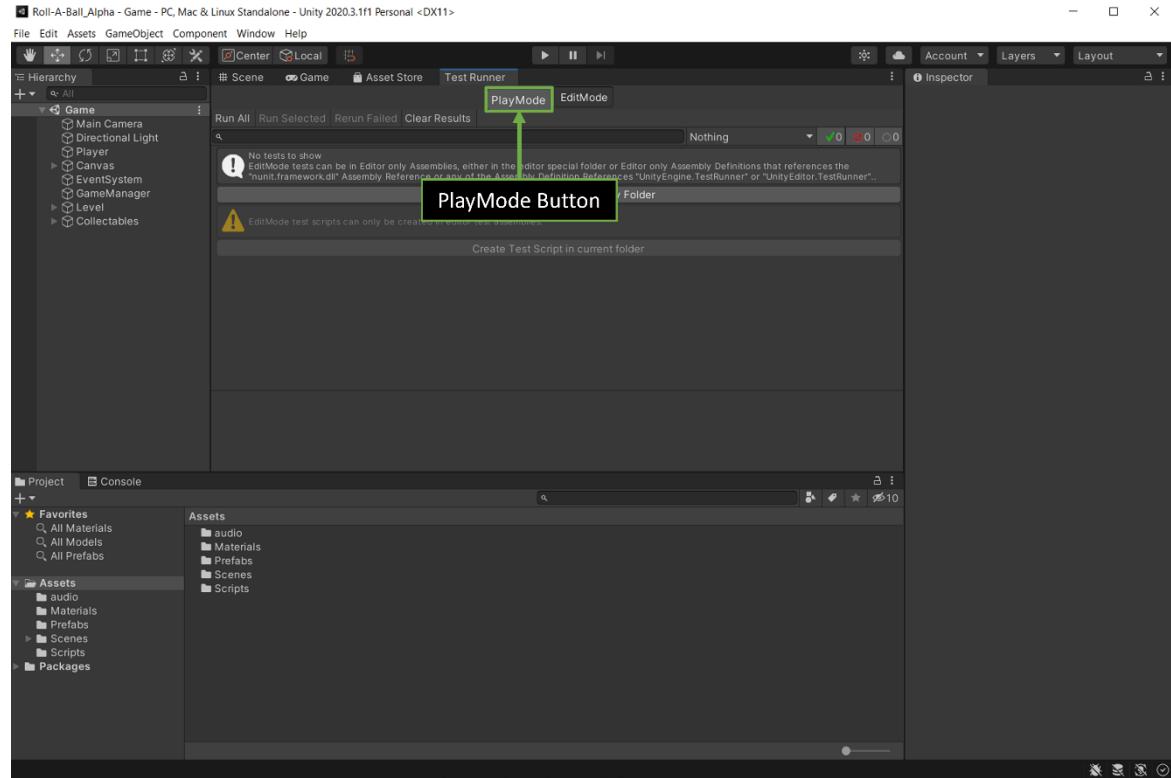
Step 10

- Once the “Test Runner” window appears, left press once on the “Test Runner” title text at the top of the window and drag it on to the center tab bar



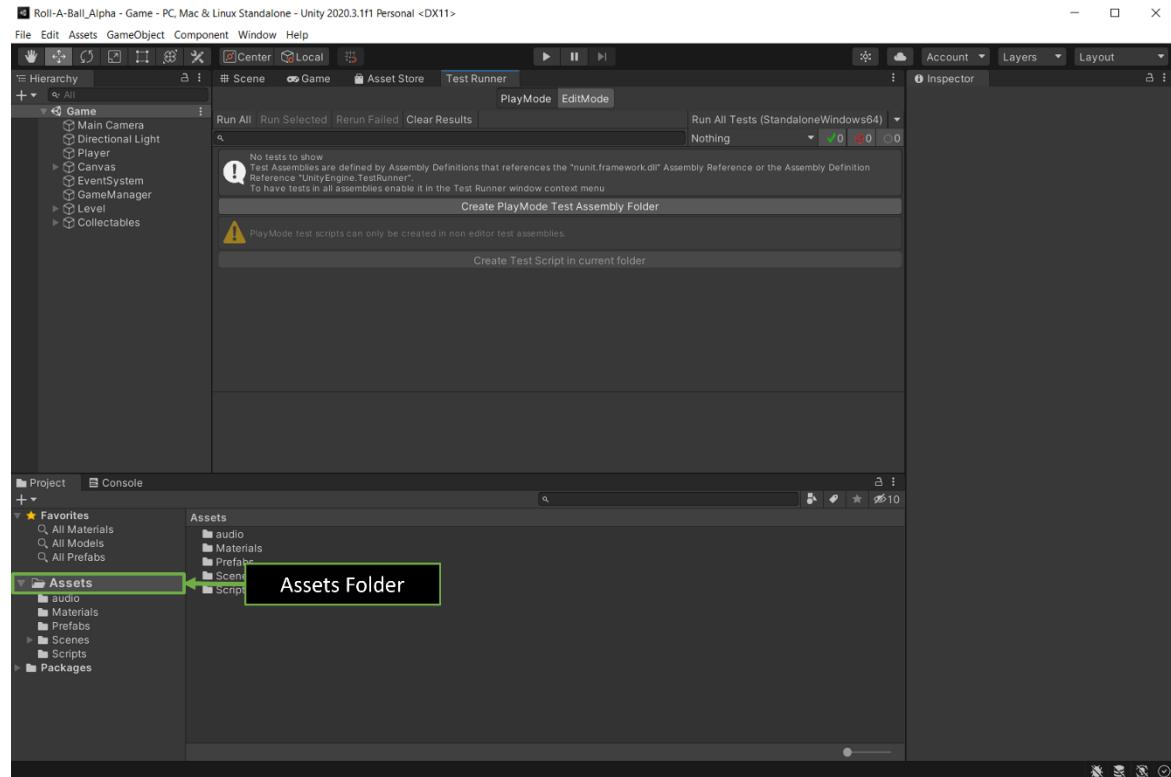
Step 11

- Left click once on the “PlayMode” button within the “Test Runner” window



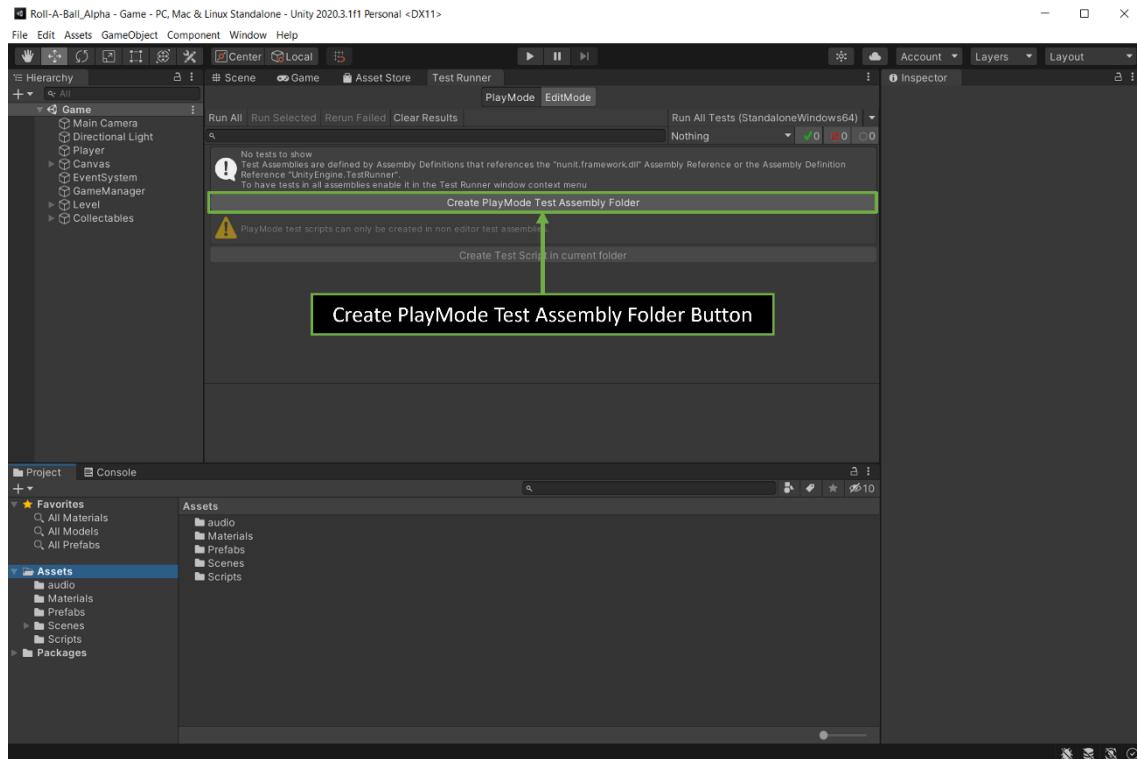
Step 12

- Left click once on the “Assets” folder within the “Project” window to select it



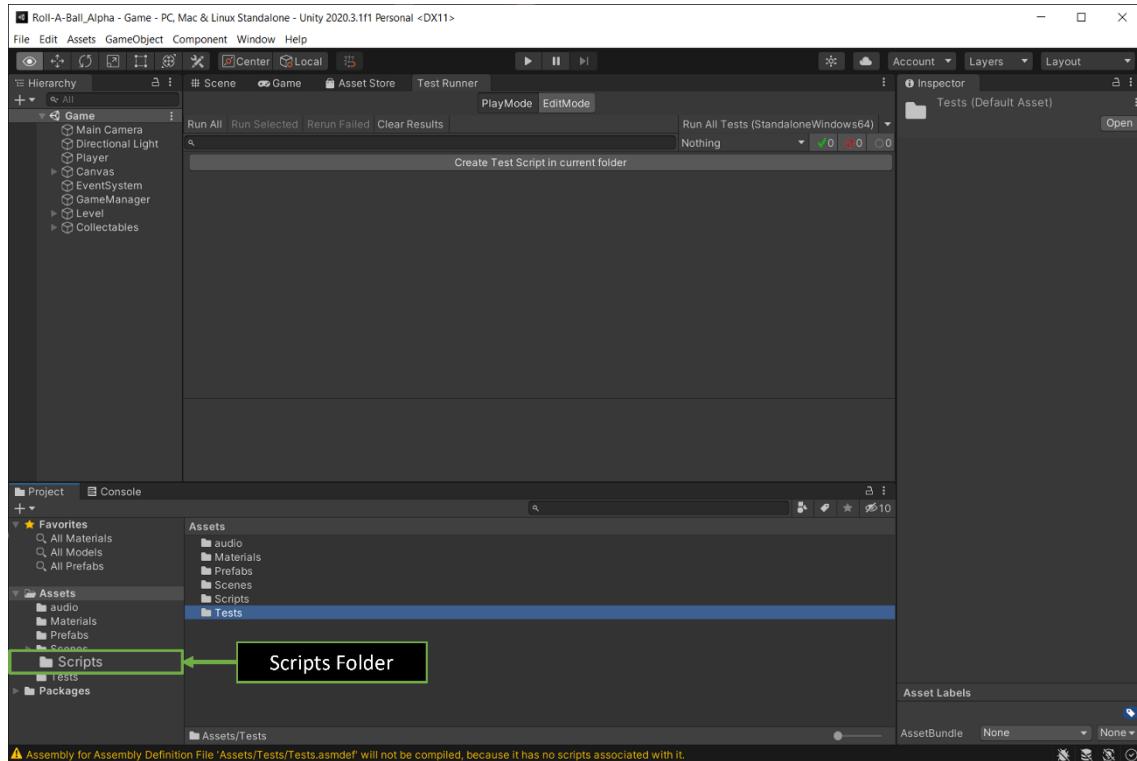
Step 13

- Left click once on the “Create PlayMode Test Assembly Folder” button within the “Test Runner” window
- Press enter once



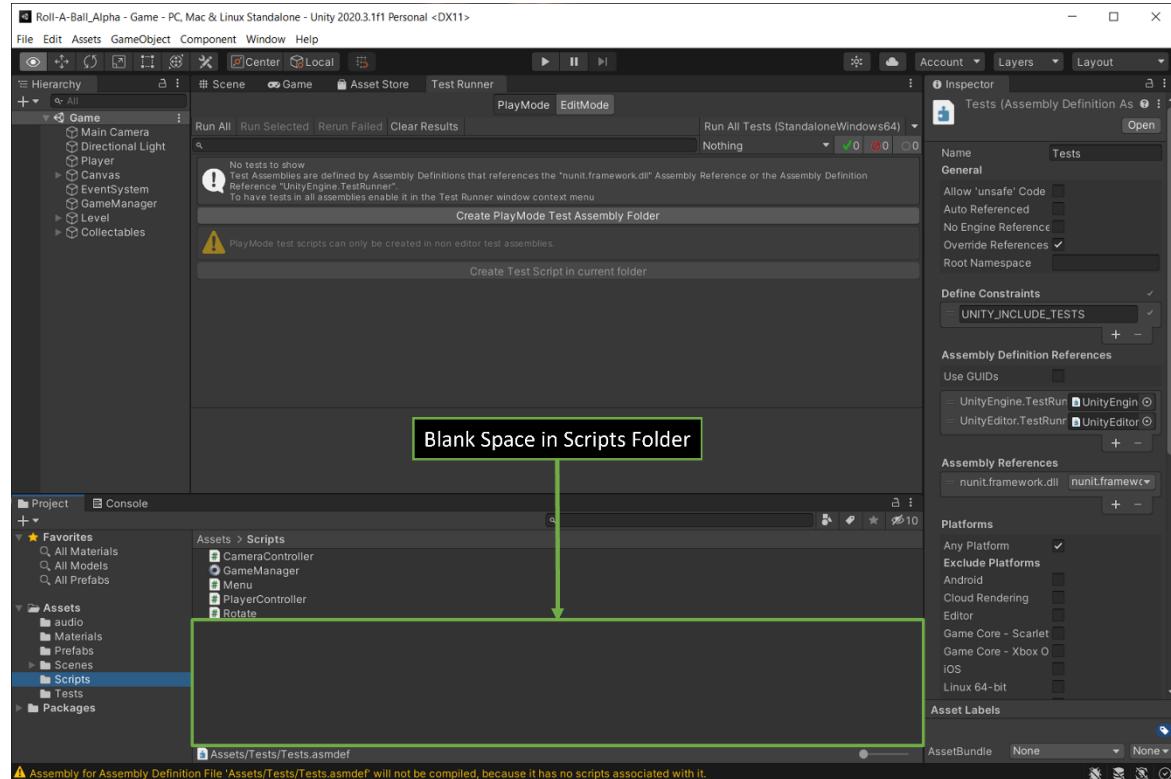
Step 14

- Left click once on the “Scripts” folder within the “Project” window



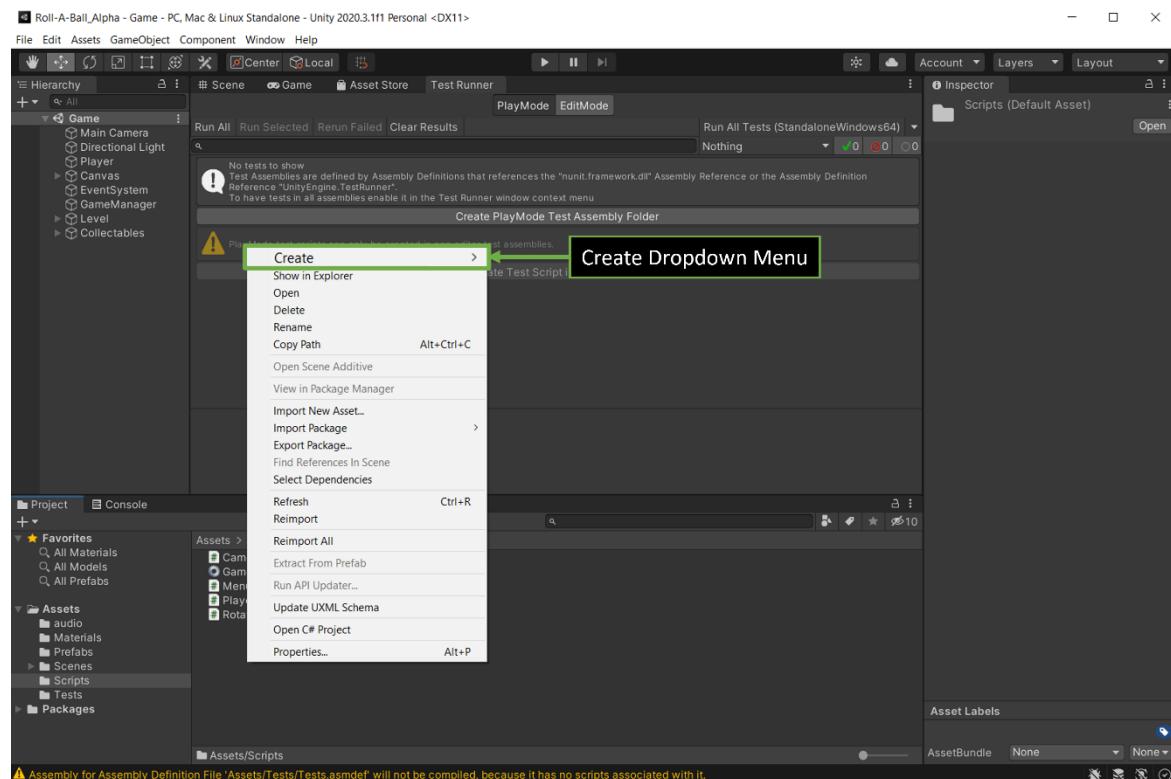
Step 15

- Right click once on the blank space within the scripts folder directory



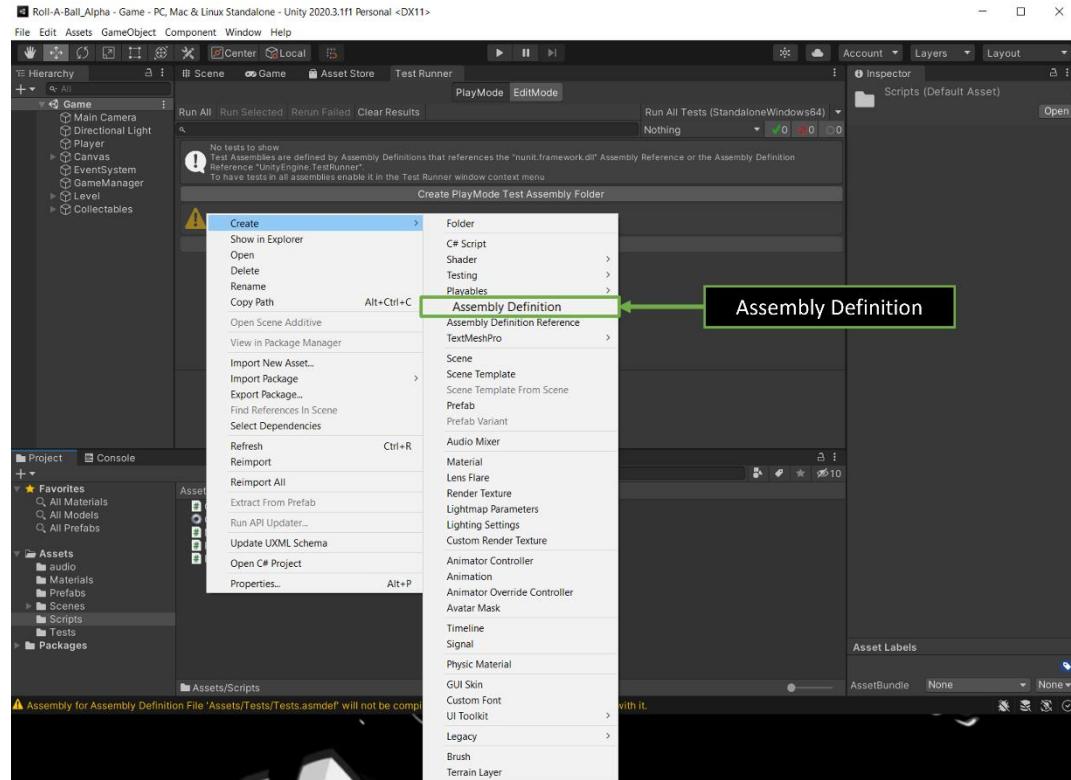
Step 16

- On the menu that appears, hover over the “Create” dropdown menu



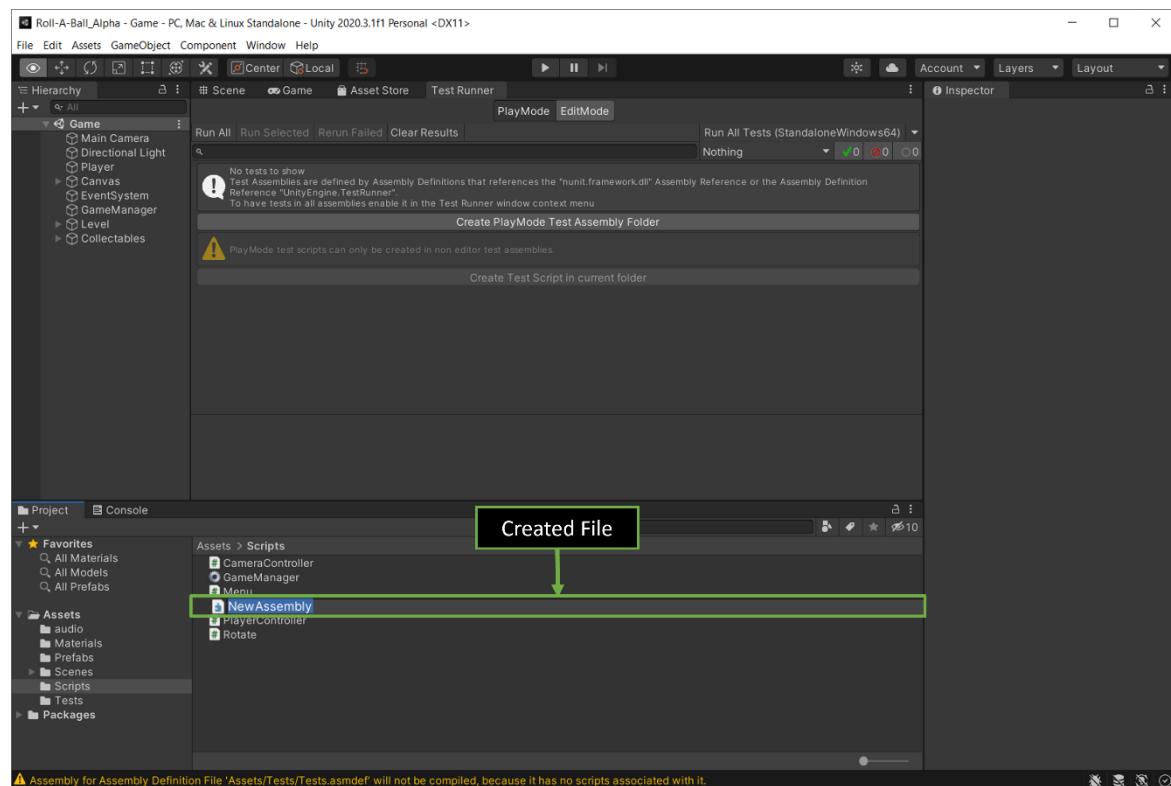
Step 17

- Left click once on “Assembly Definition” within the “Create” dropdown menu



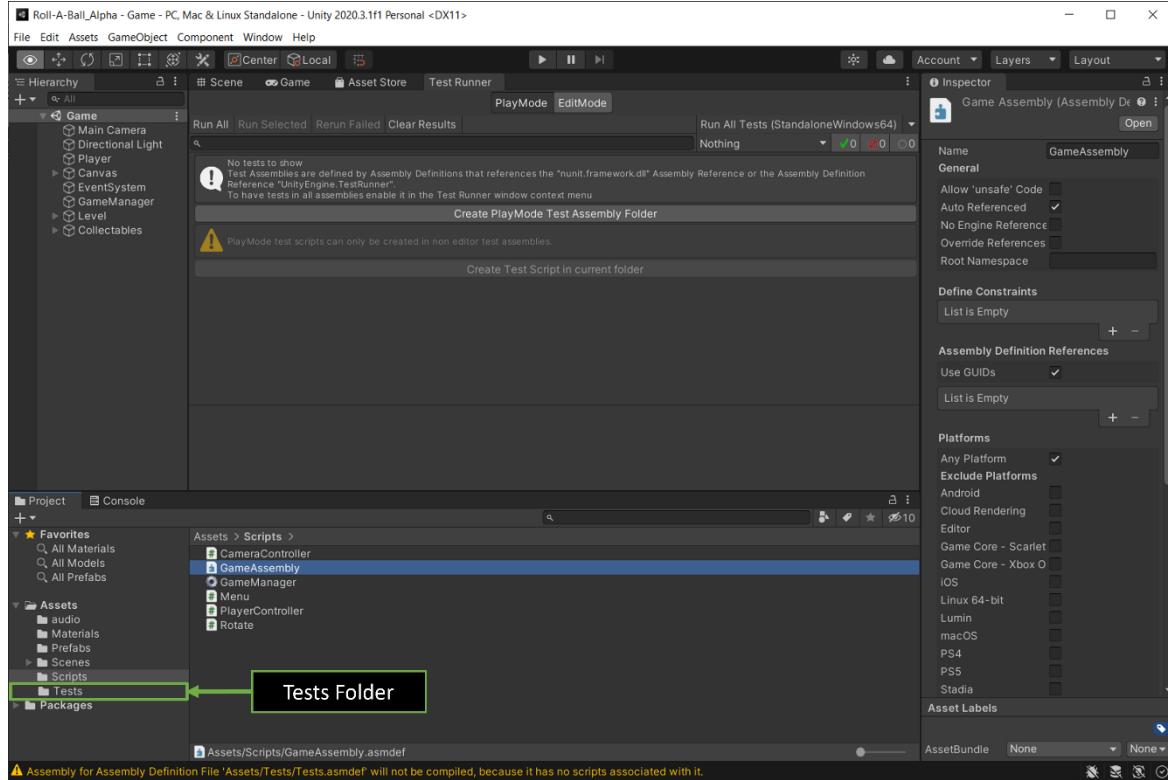
Step 18

- Rename the created file within the “Scripts” folder by typing in “GameAssembly”
➤ Press enter once



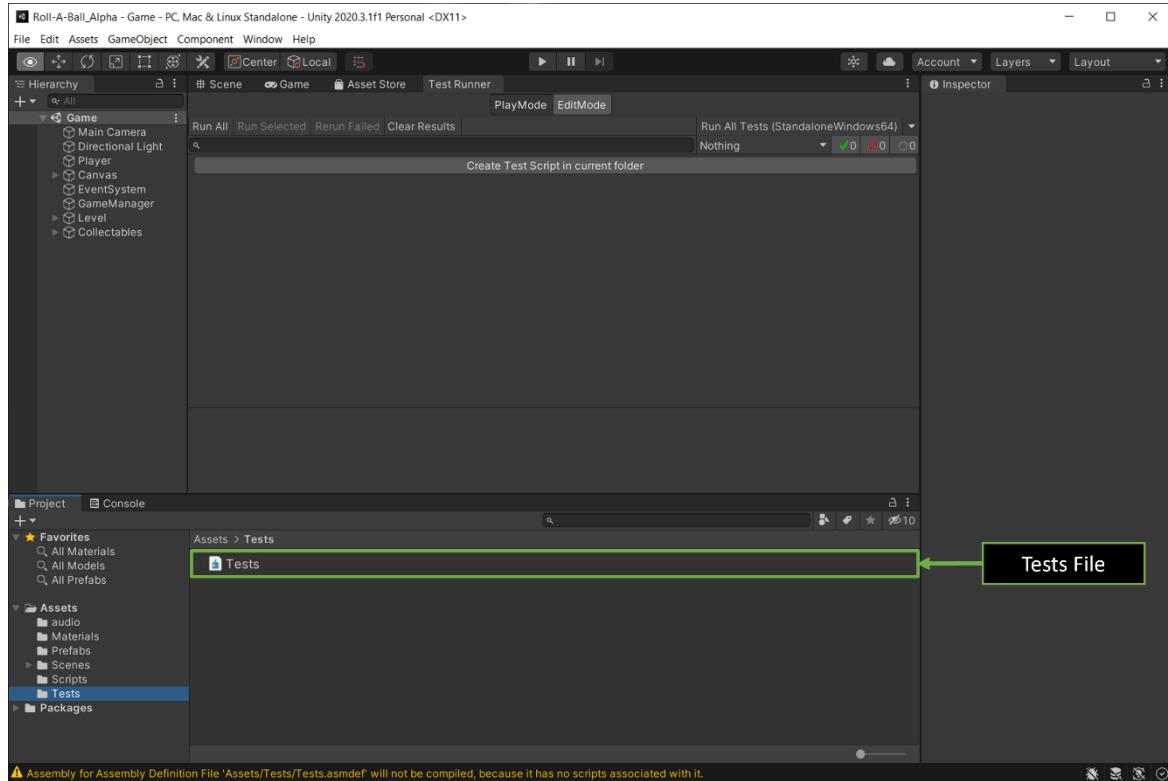
Step 19

- Left click once on the “Tests” folder within the “Project” window



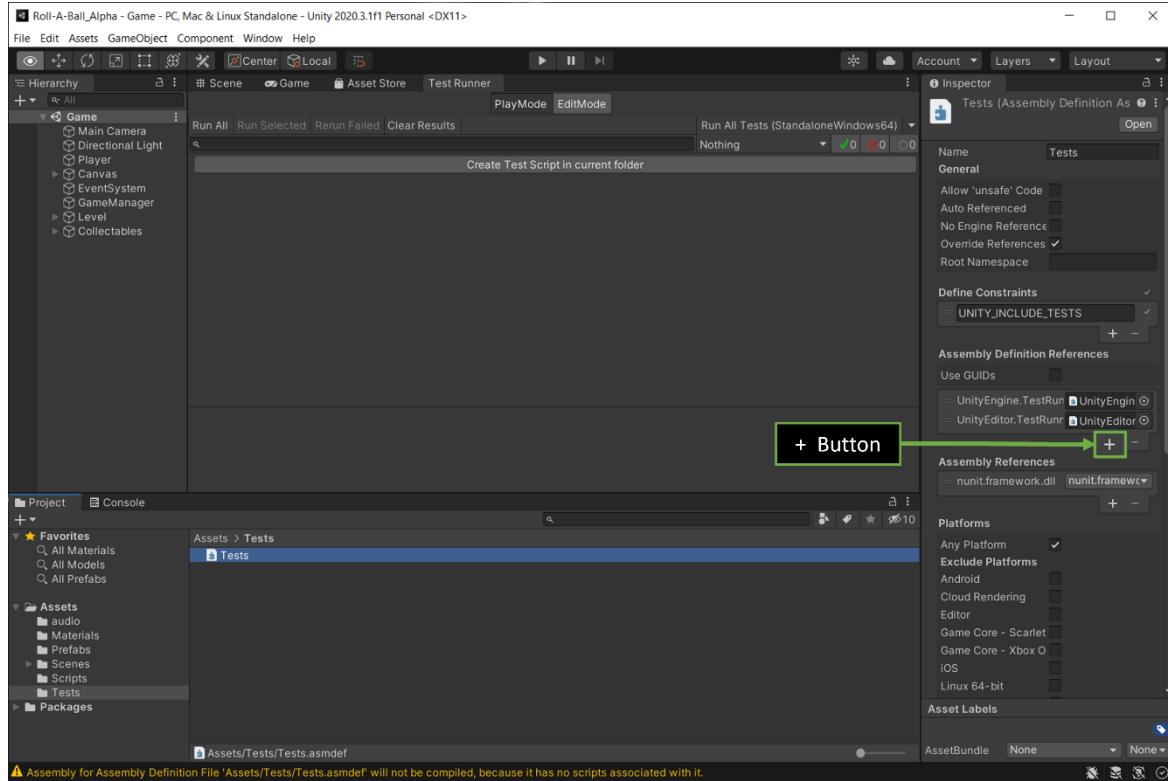
Step 20

- Left click once on the “Tests” file within the “Tests” folder to select it



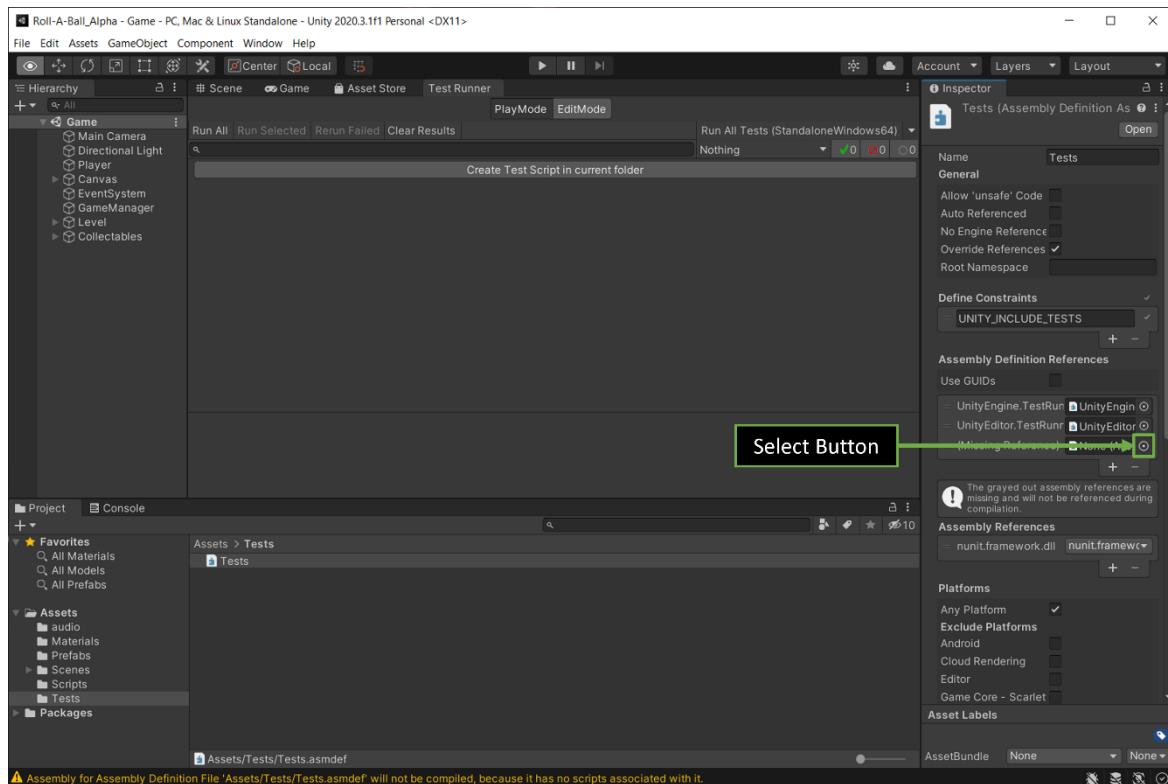
Step 21

- Left click once on the “+” button in the Assembly Definition References within the “Inspector” window



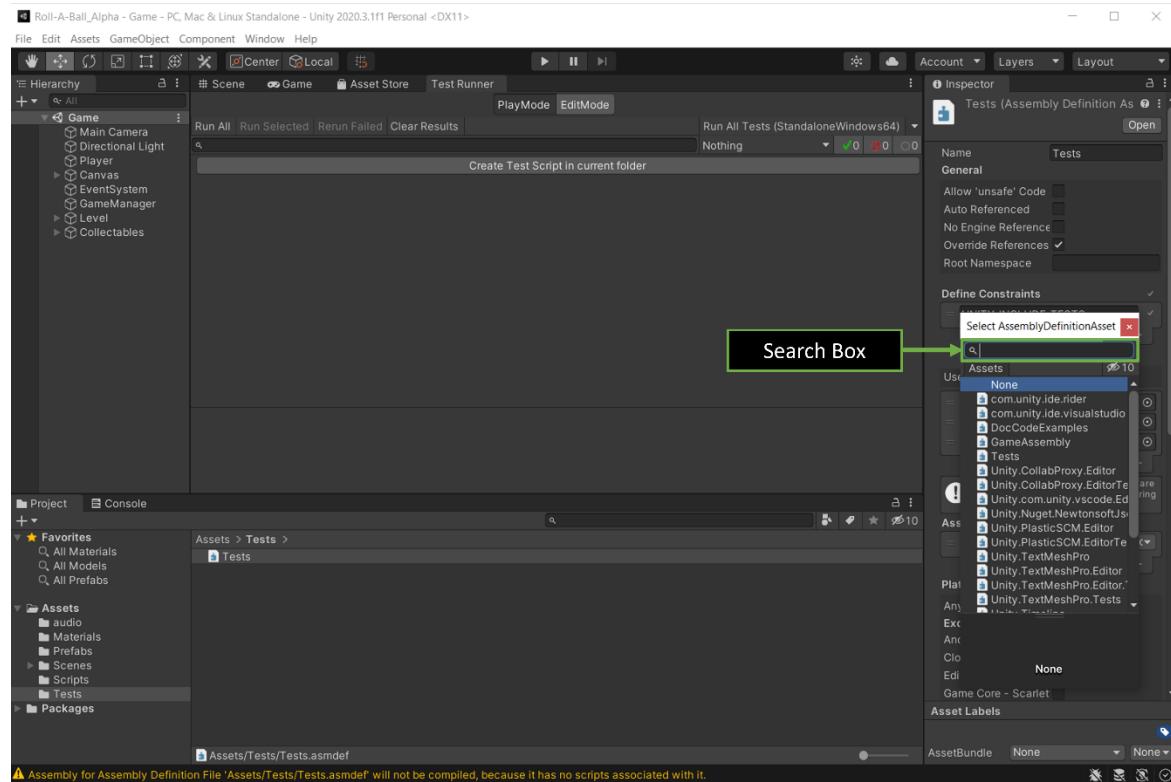
Step 22

- On the assembly definition reference that is created, left click on the select button



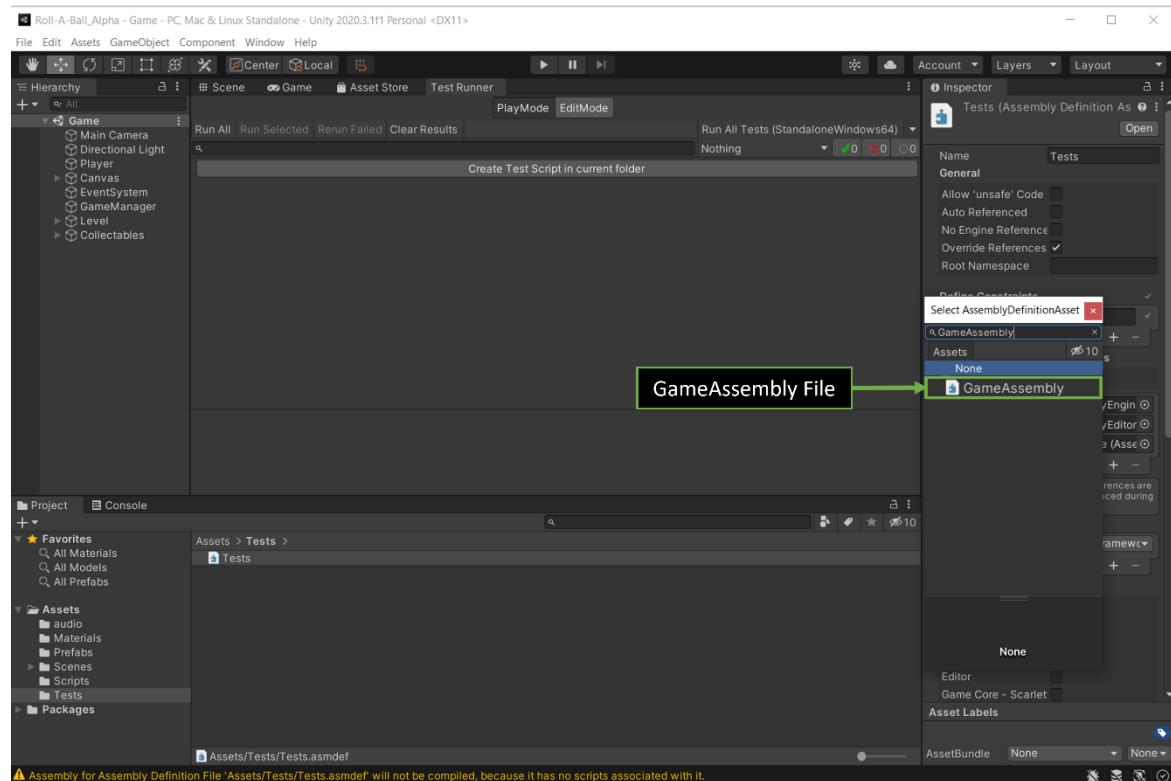
Step 23

- Once the select window appears, type in “GameAssembly” within the select search box



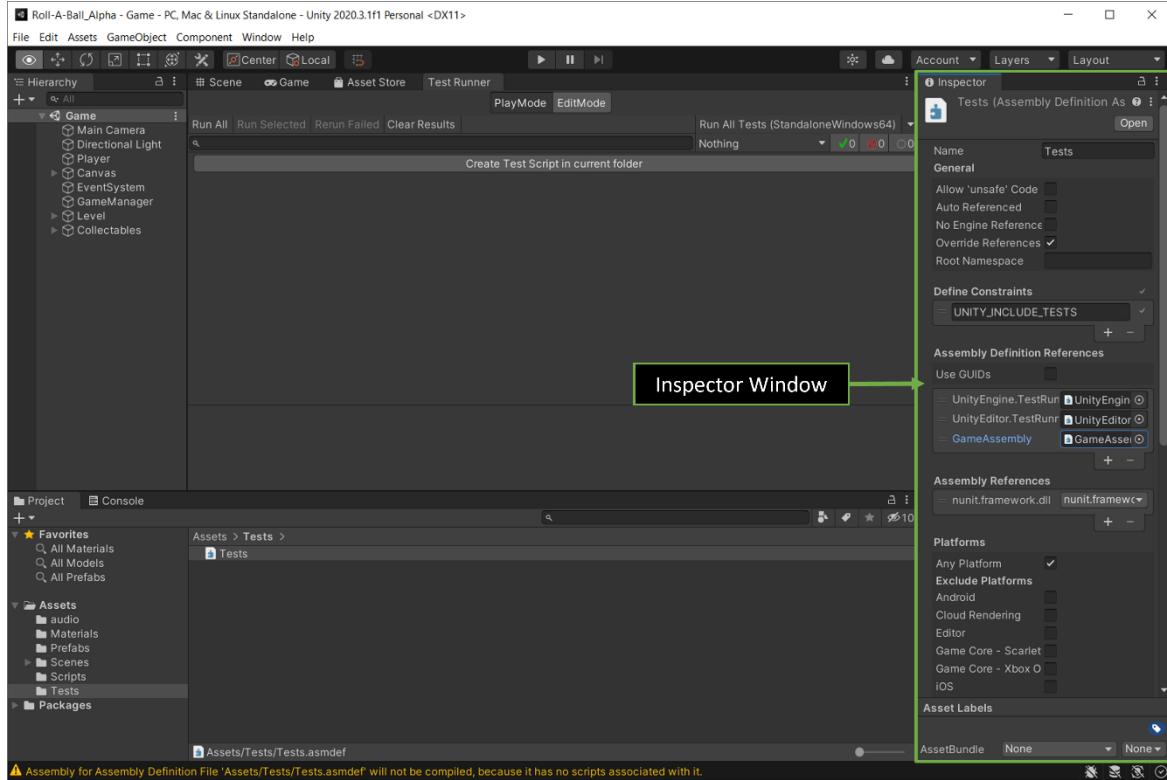
Step 24

- Left click twice on the “GameAssembly” file



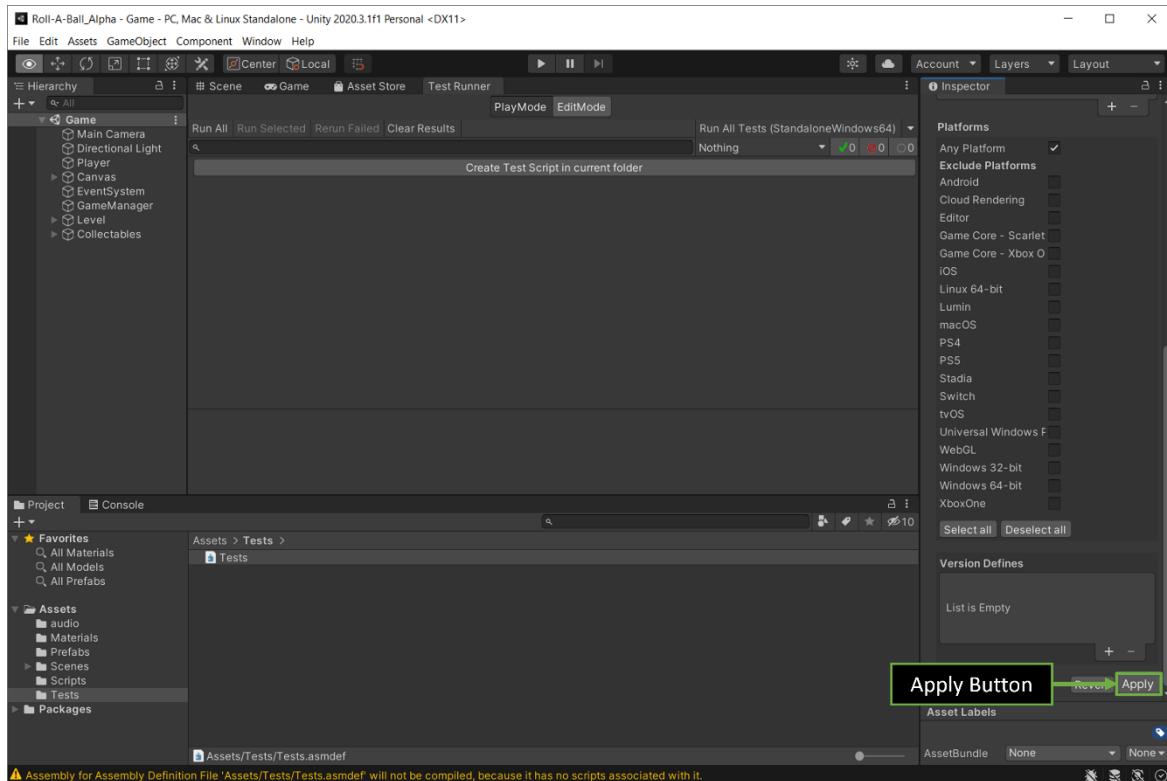
Step 25

- Scroll down to the bottom of the “Inspector” window



Step 26

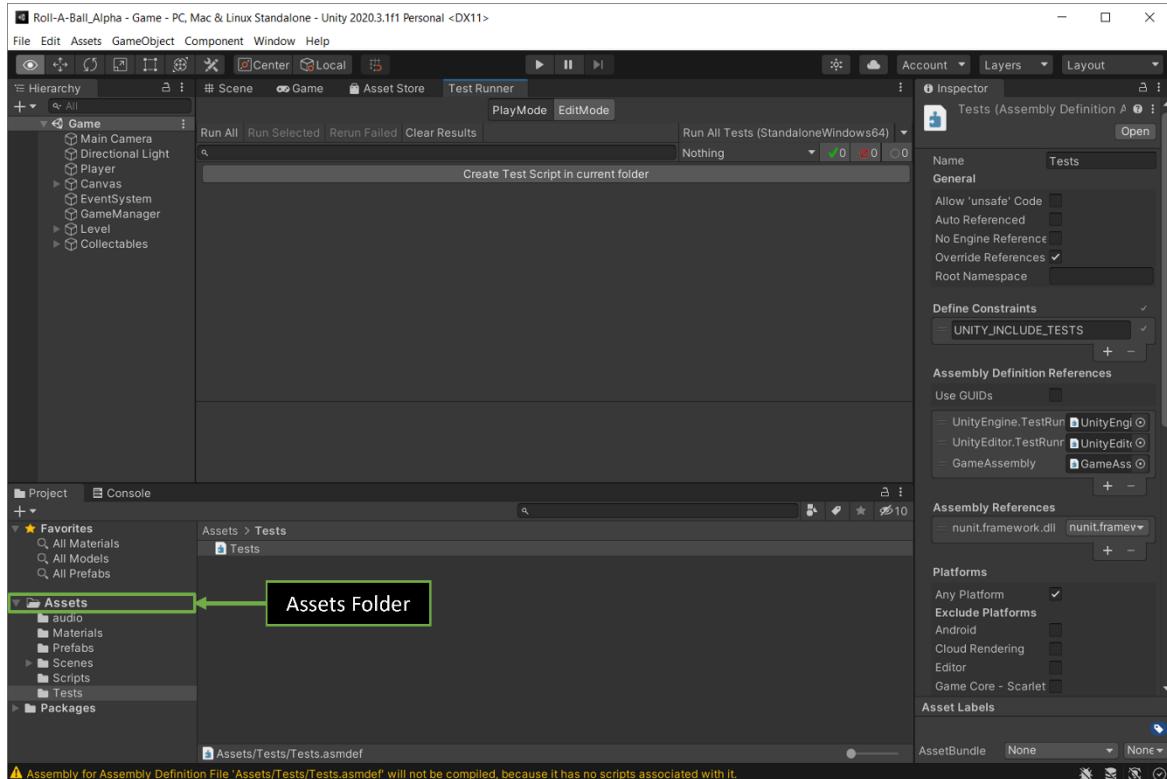
- Left click once on the “Apply” button



Setup Unit Tests

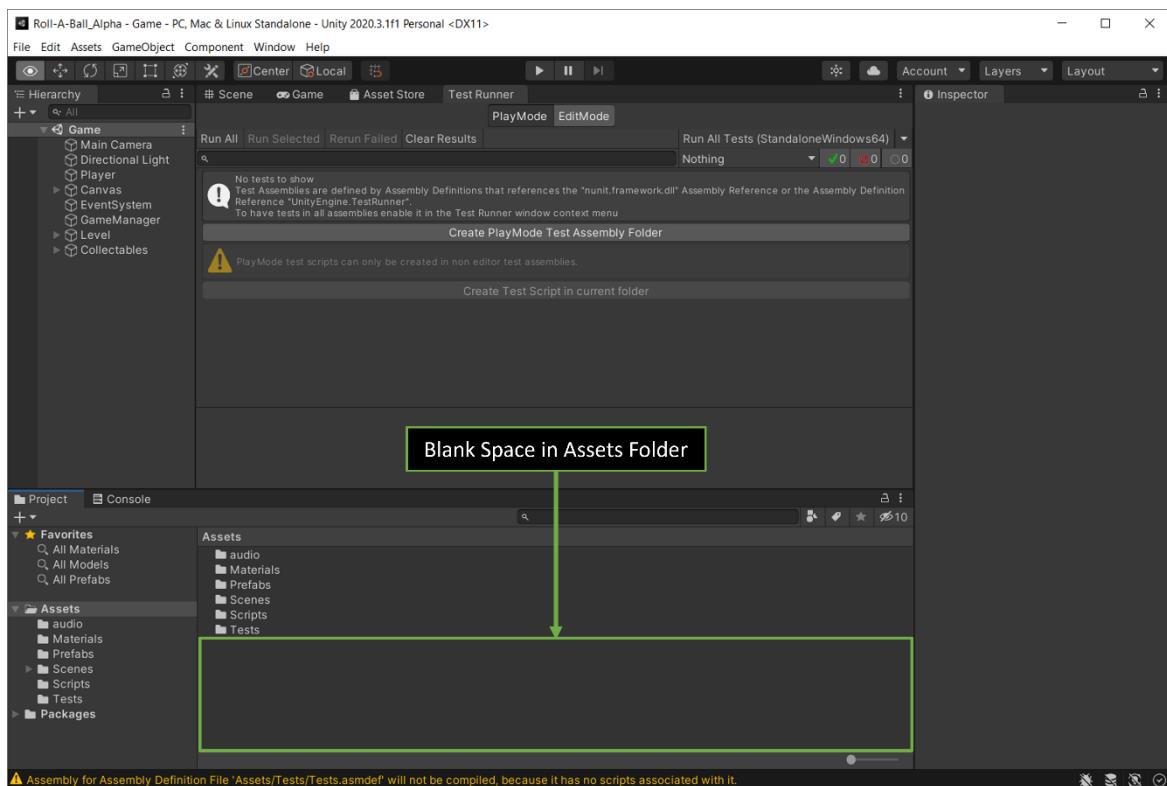
Step 1

- Left click once on the “Assets” folder within the project window hierarchy



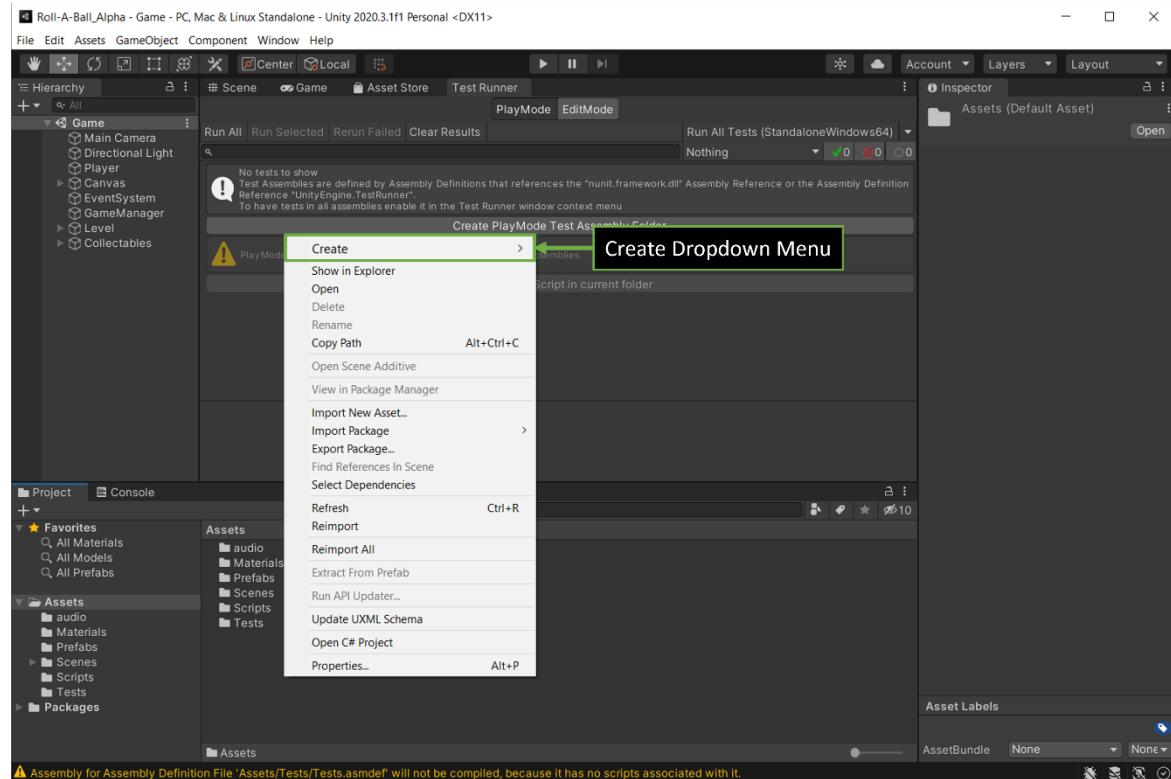
Step 2

- Right click once on the blank space within the assets folder directory



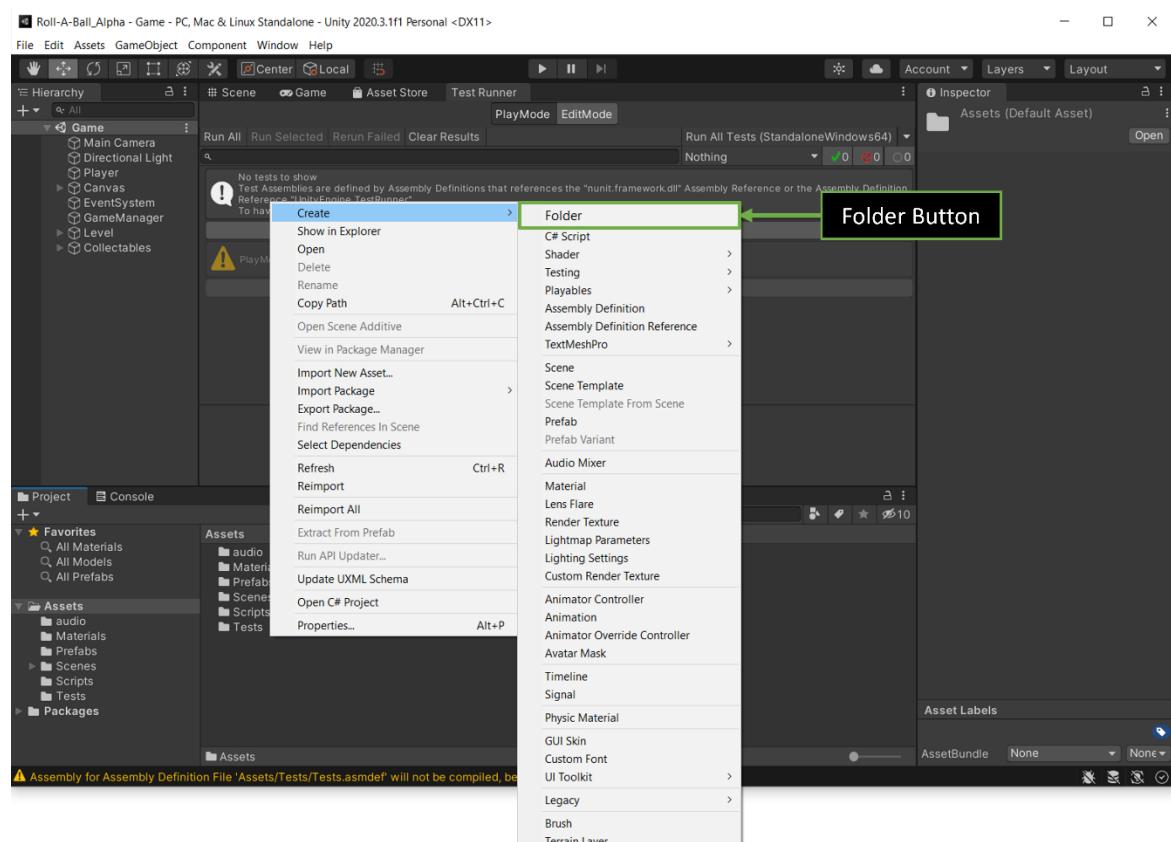
Step 3

- Once the menu appears, hover over the “Create” dropdown menu



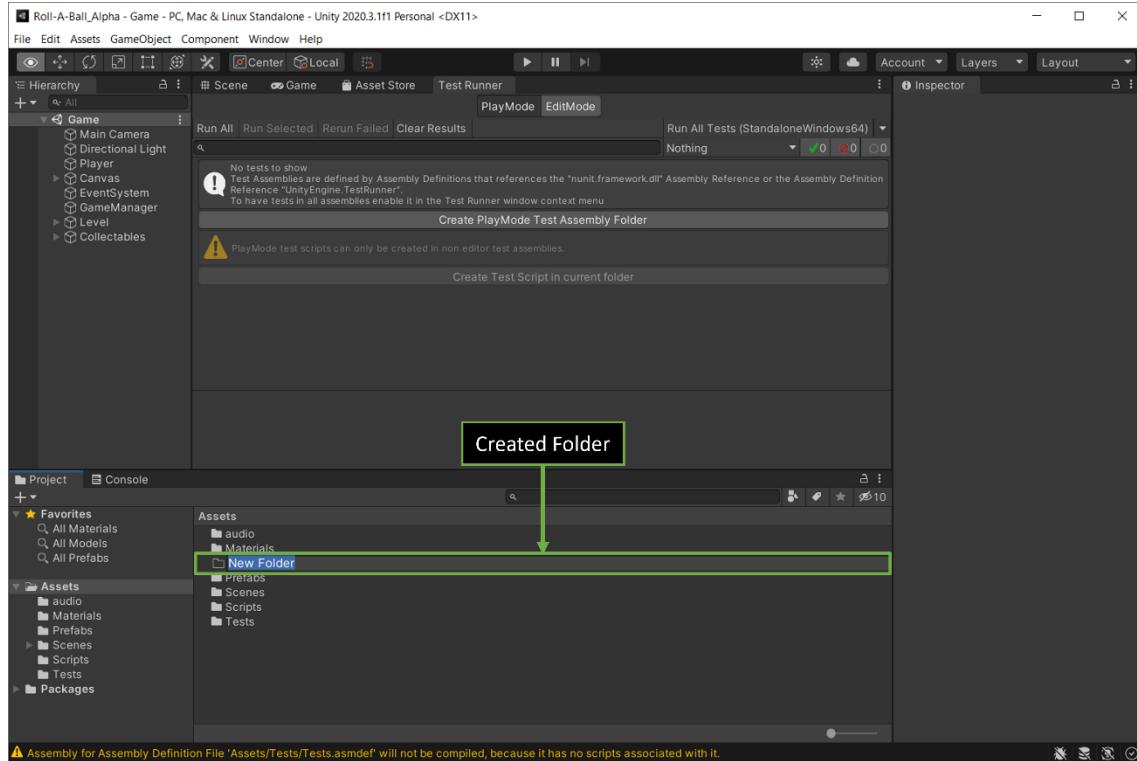
Step 4

- Left click once on the “Folder” button



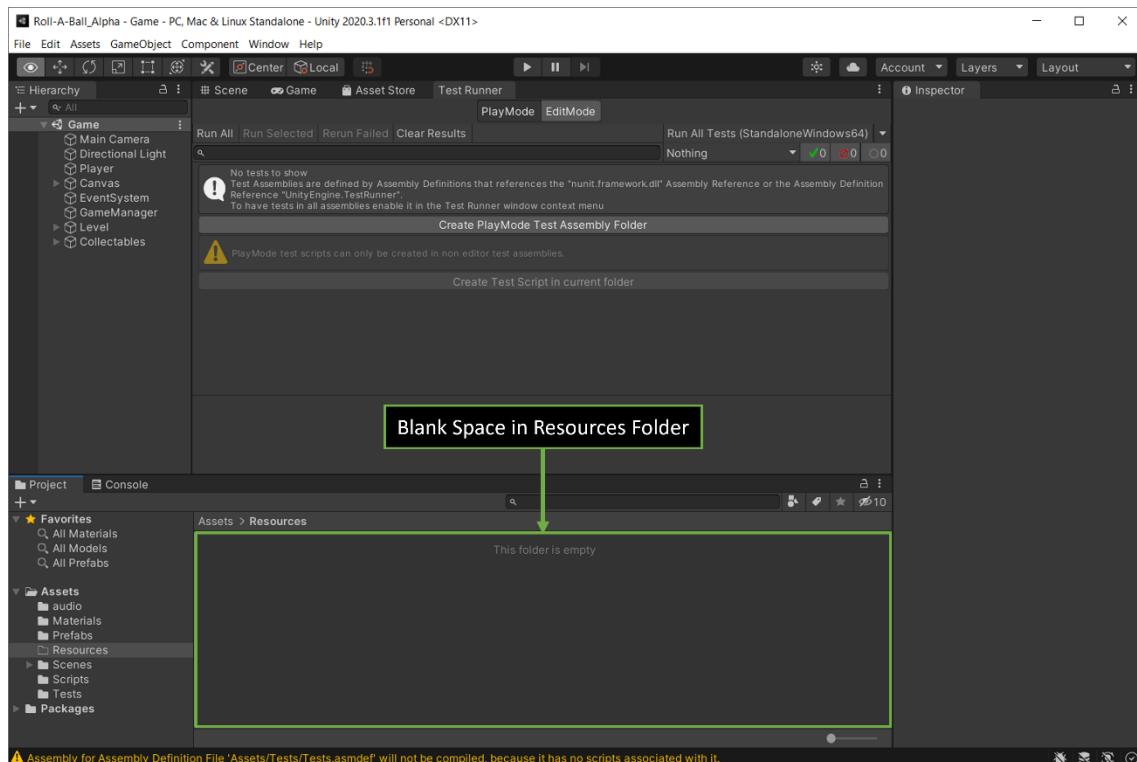
Step 5

- Rename the created folder within the “Assets” folder by typing in “Resources”
- Press enter twice to rename the folder and then open it



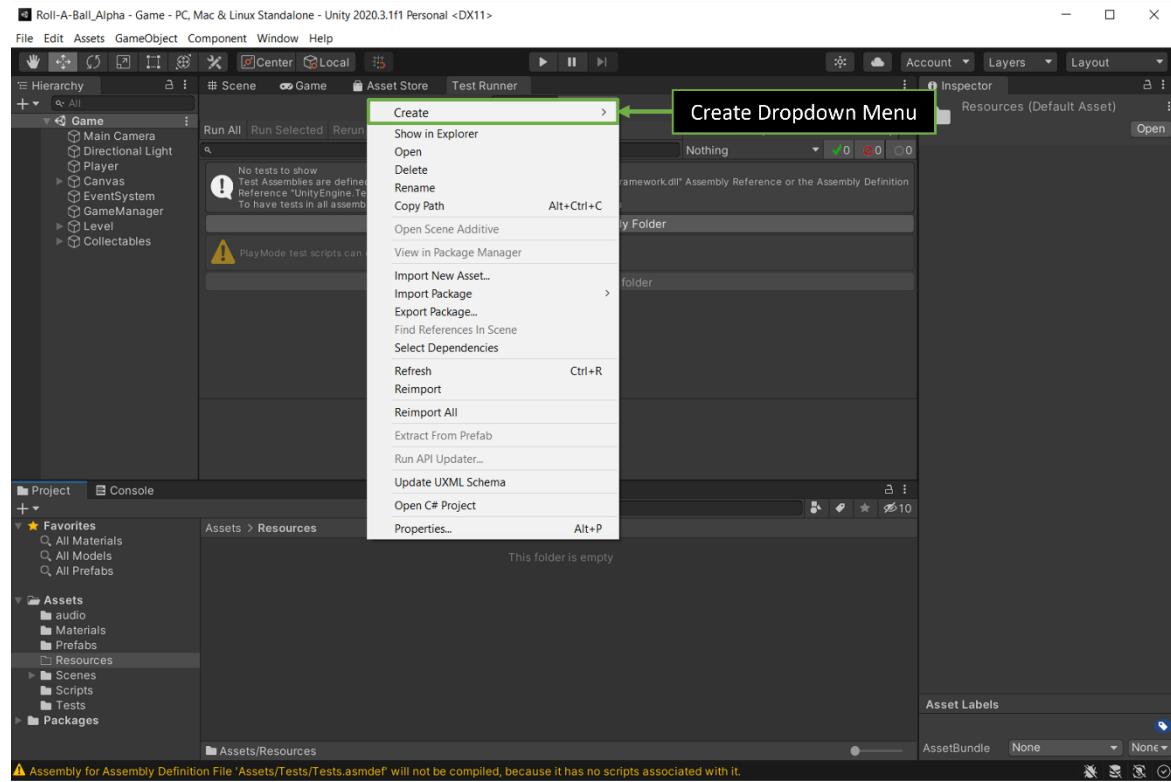
Step 6

- Once the Resources folder is open, right click once on the Blank Space within the Directory



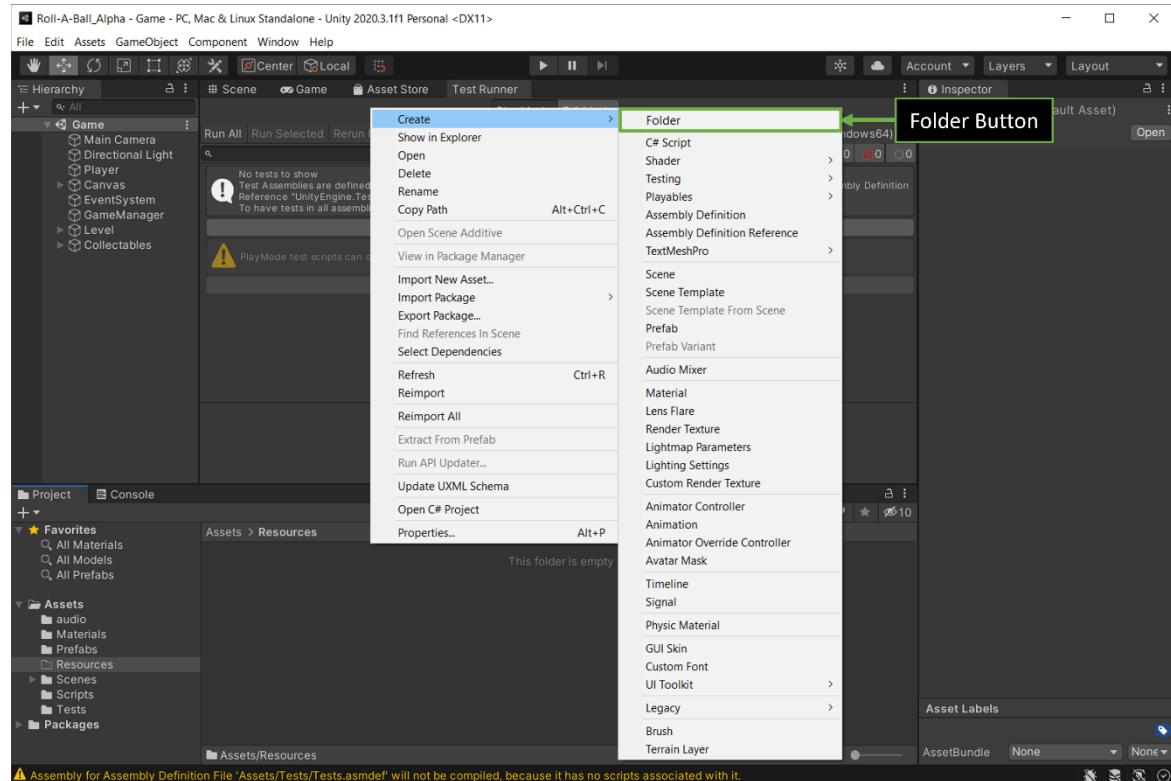
Step 7

- Once the menu appears, hover over the “Create” dropdown menu



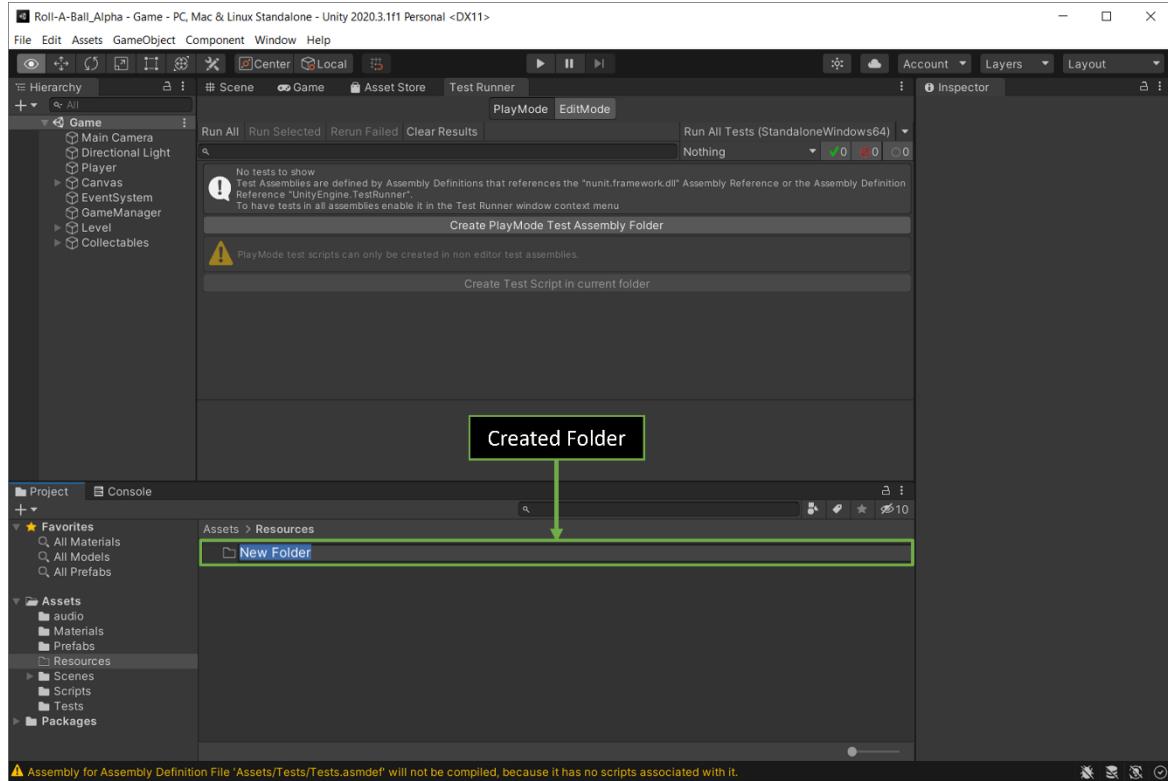
Step 8

- Left click once on the “Folder” button



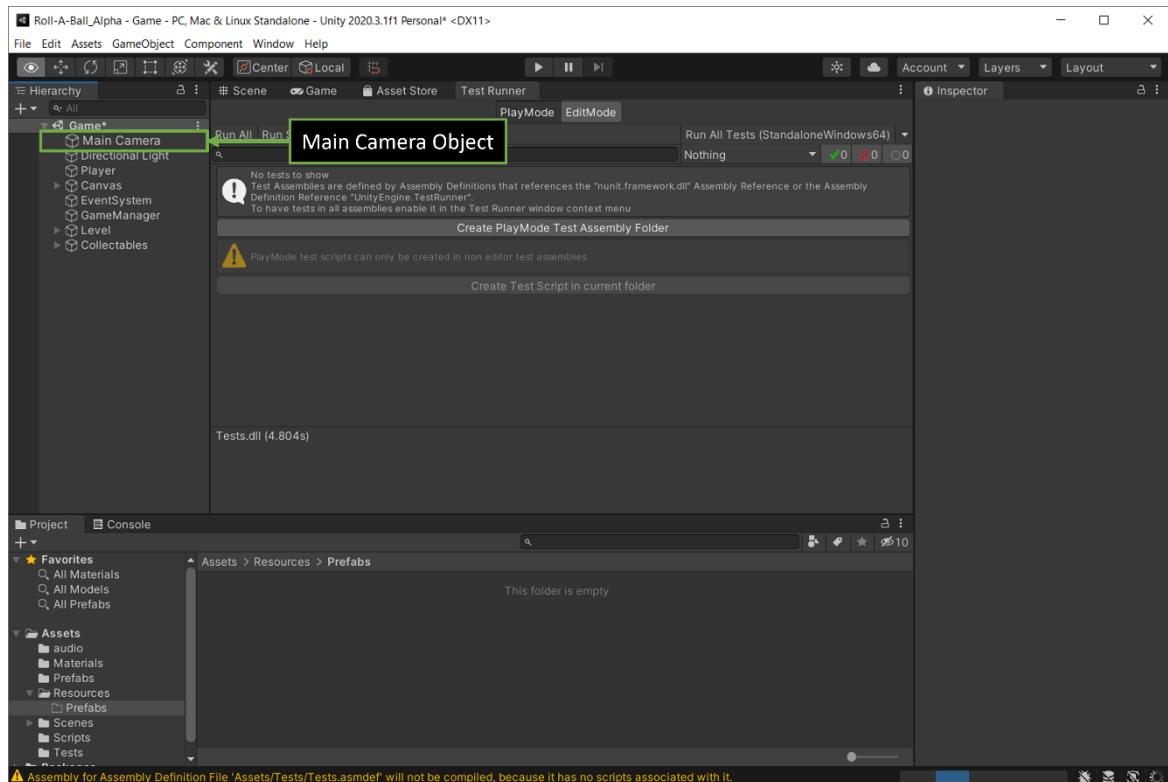
Step 9

- Rename the created folder within the “Assets” folder by typing in “Prefabs”
- Press enter twice to rename the folder and then open it



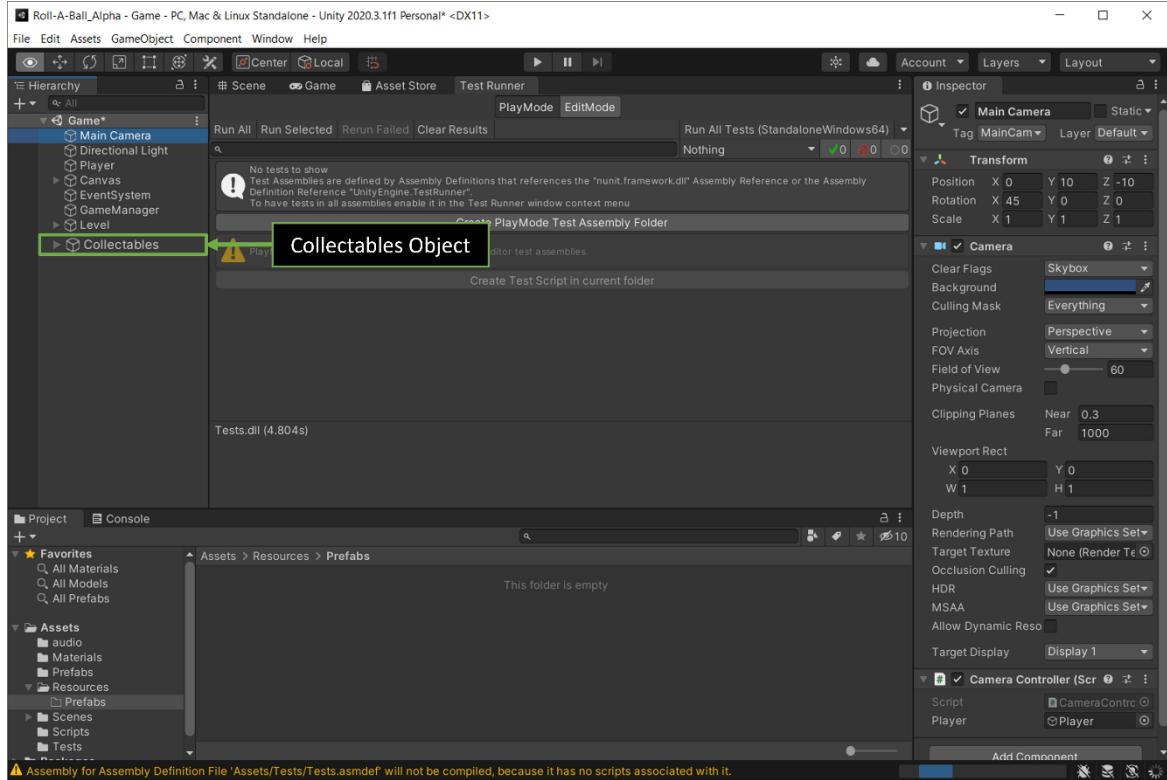
Step 10

- Left click once on the “Main Camera” object



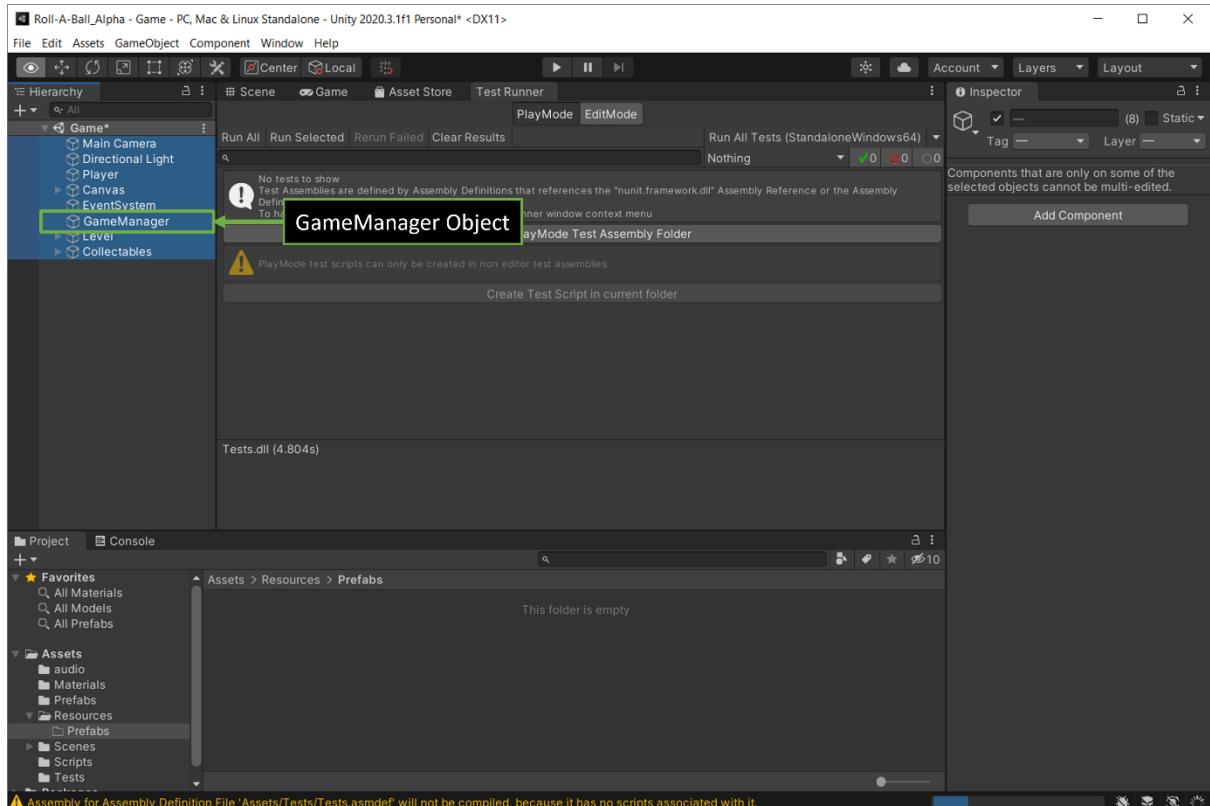
Step 11

- Press shift and left click once on the “Collectables” object



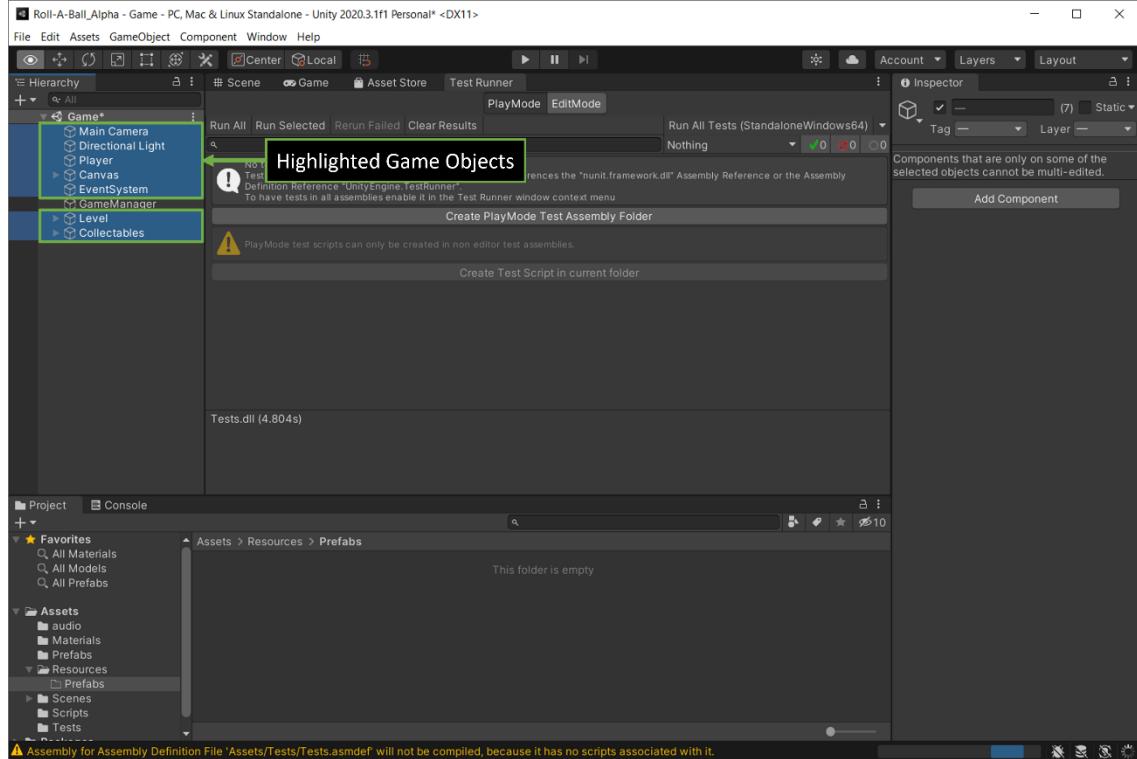
Step 12

- Press control (CTRL) and left click once on the “GameManager” object



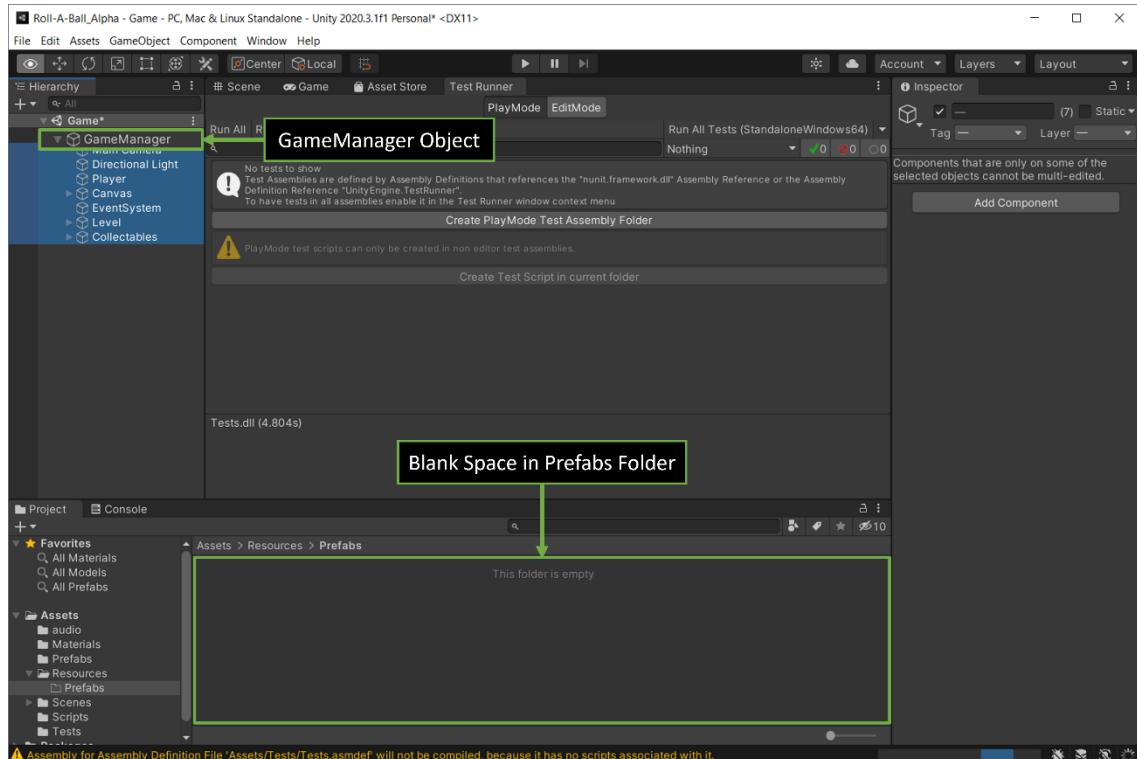
Step 13

- Left press on the highlighted game objects and drag them onto the “GameManager” object



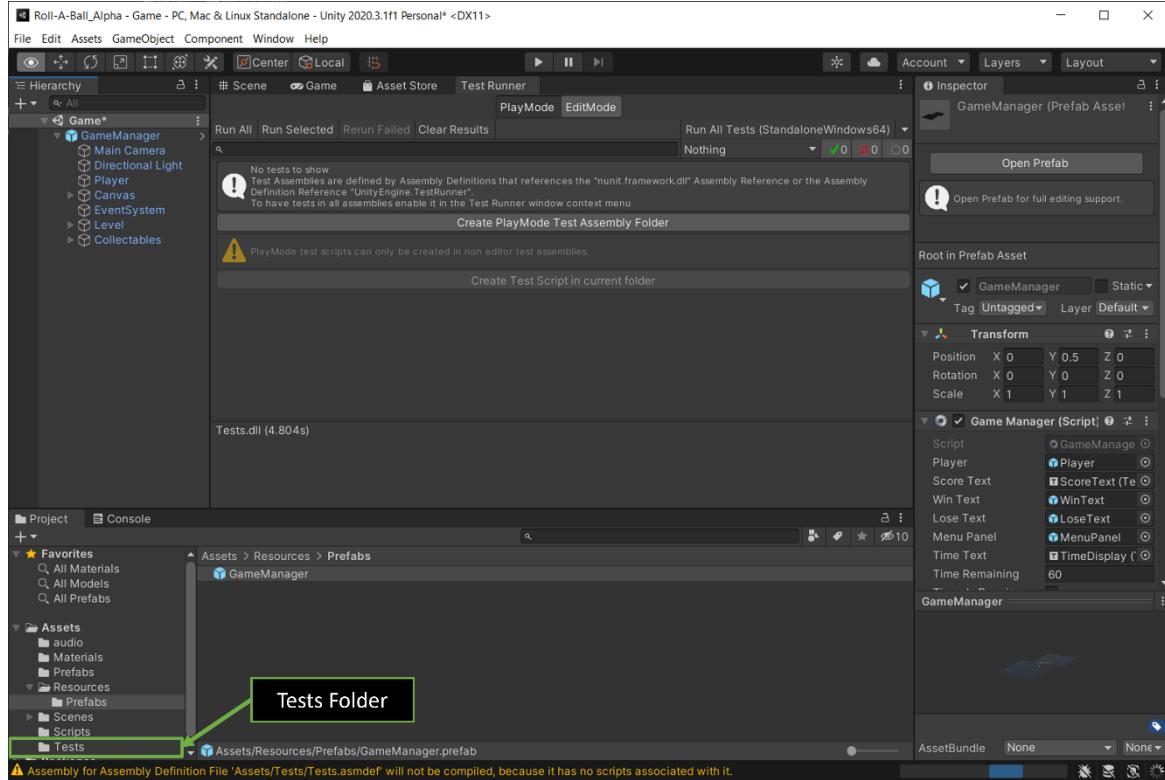
Step 14

- Left press on the “GameManager” object and drag it on to the blank space within “Prefabs” directory



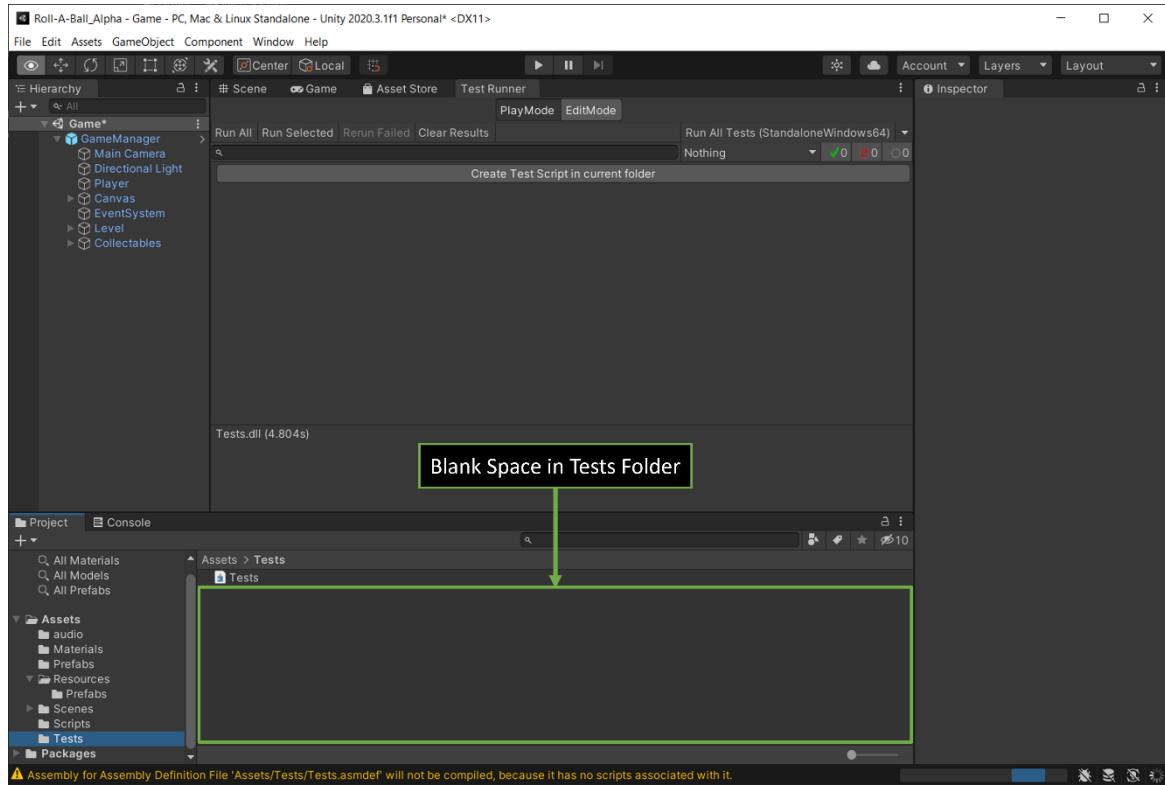
Step 15

- Left click once on the “Tests” folder within the project window hierarchy



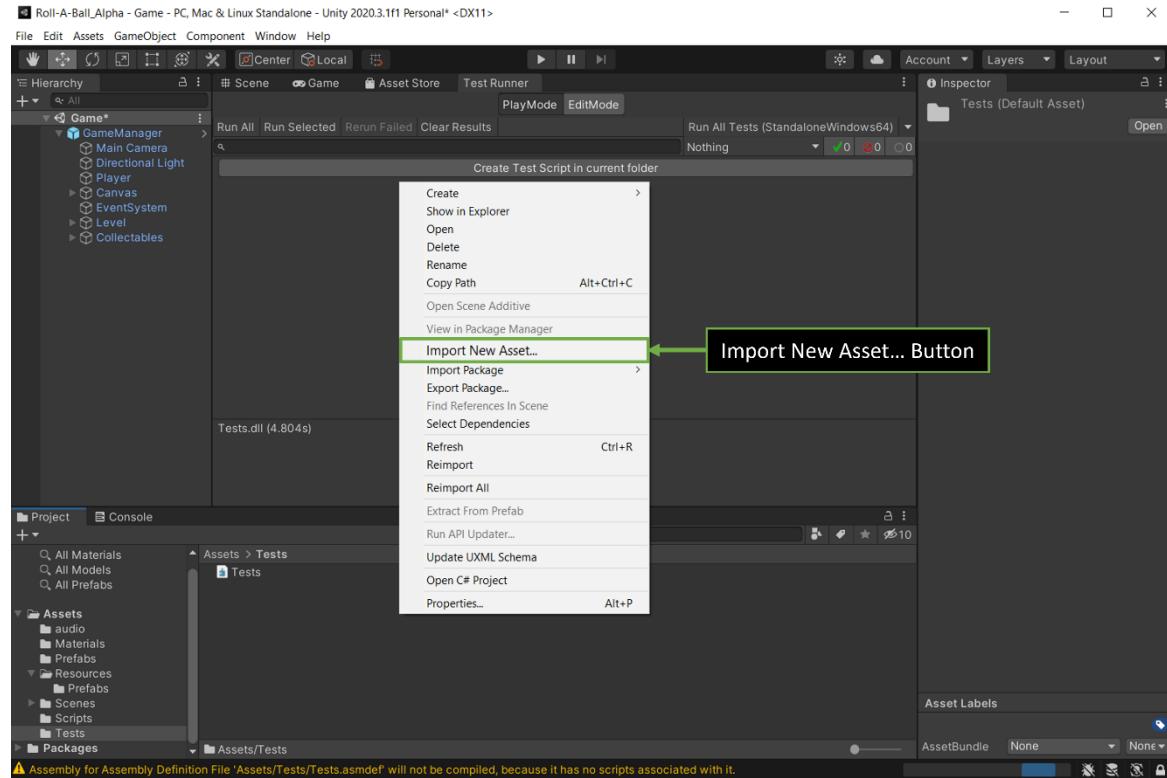
Step 16

- Right click once on the blank space within the Tests folder directory



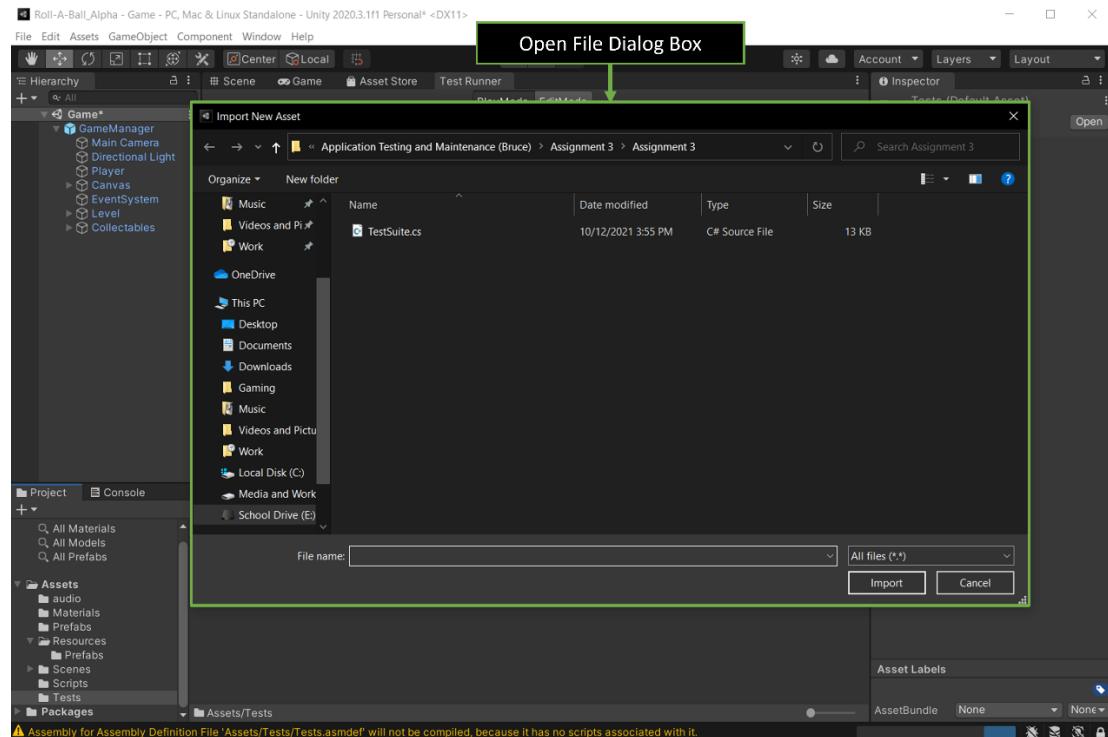
Step 17

- On the menu that appears, left click once on the “Import New Asset...” button



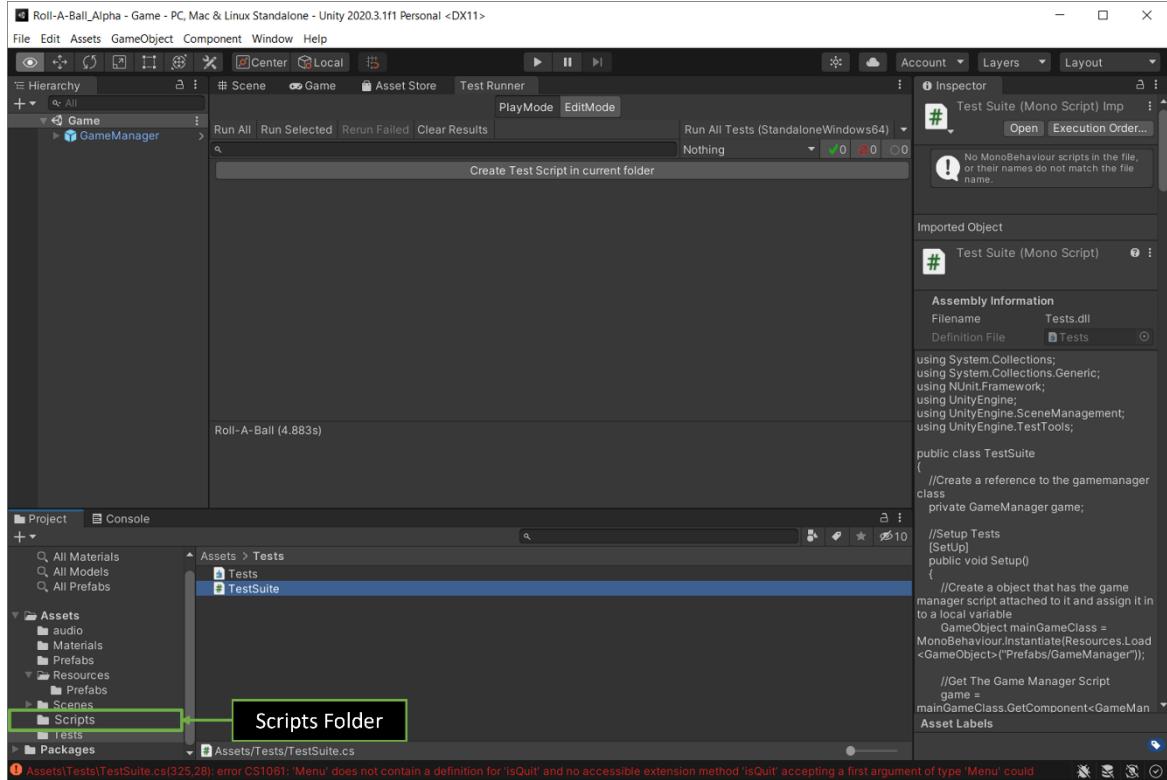
Step 18

- Once the open file dialog box appears, navigate to the TestSuite.cs file included in the assignment folder
➤ Left click twice on the file to import it into unity



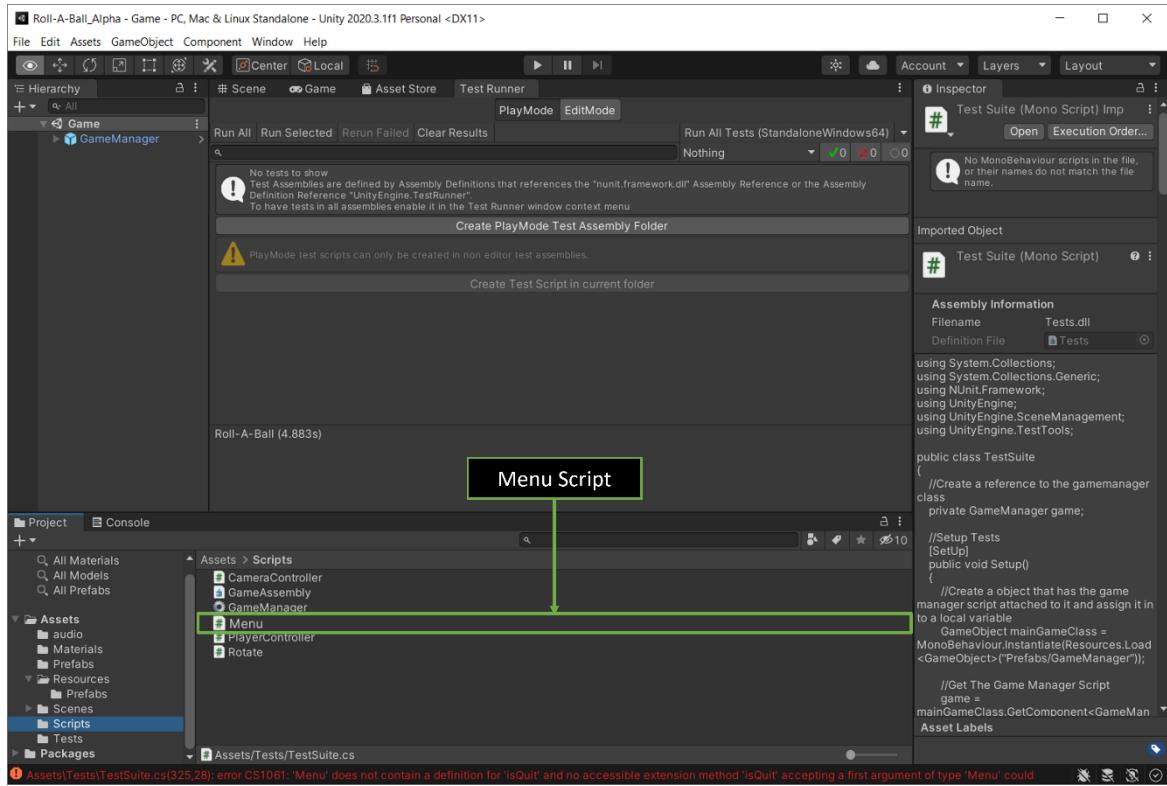
Step 19

- Left click once on the “Scripts” folder within the project window hierarchy



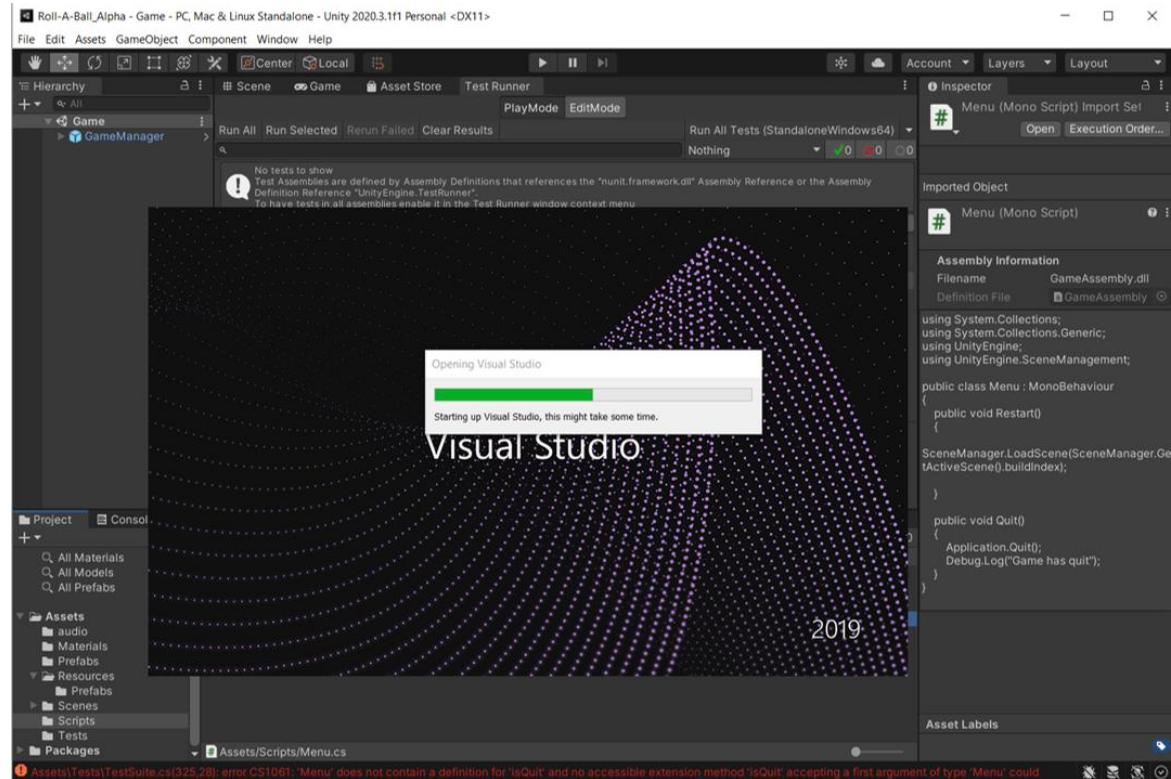
Step 20

- Left click twice on the “Menu” script to open it



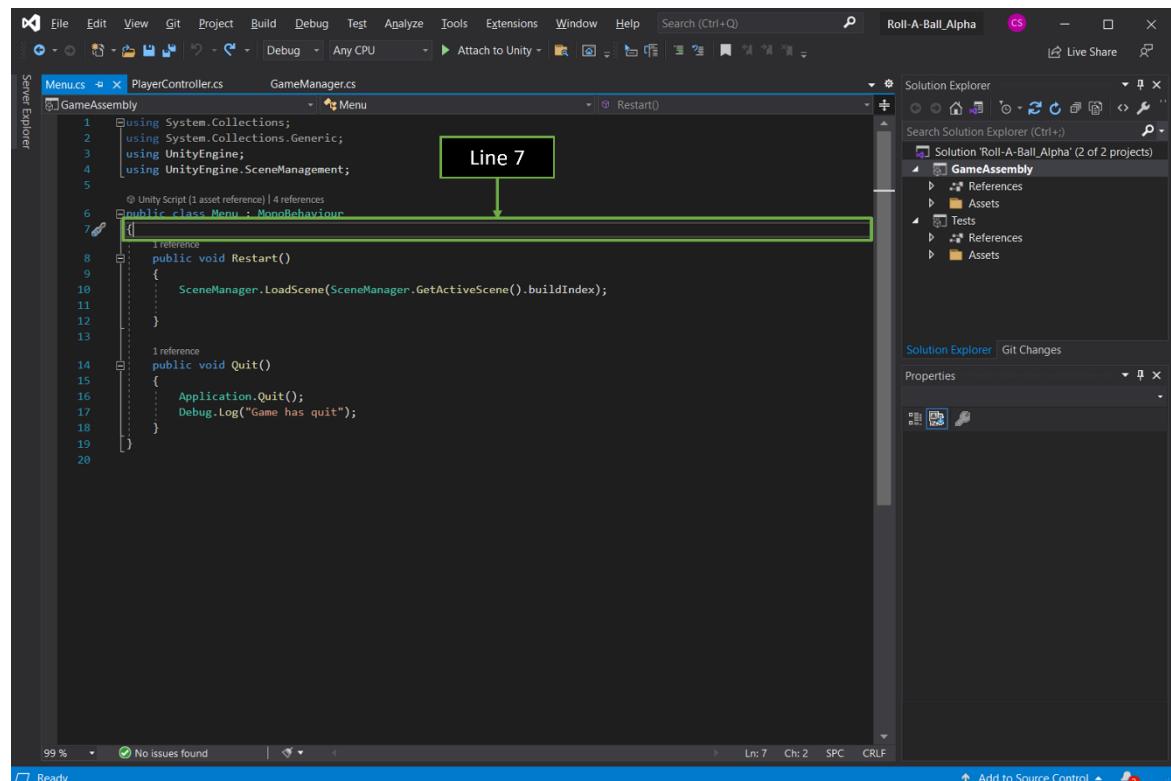
Step 21

- Wait for the menu script to open



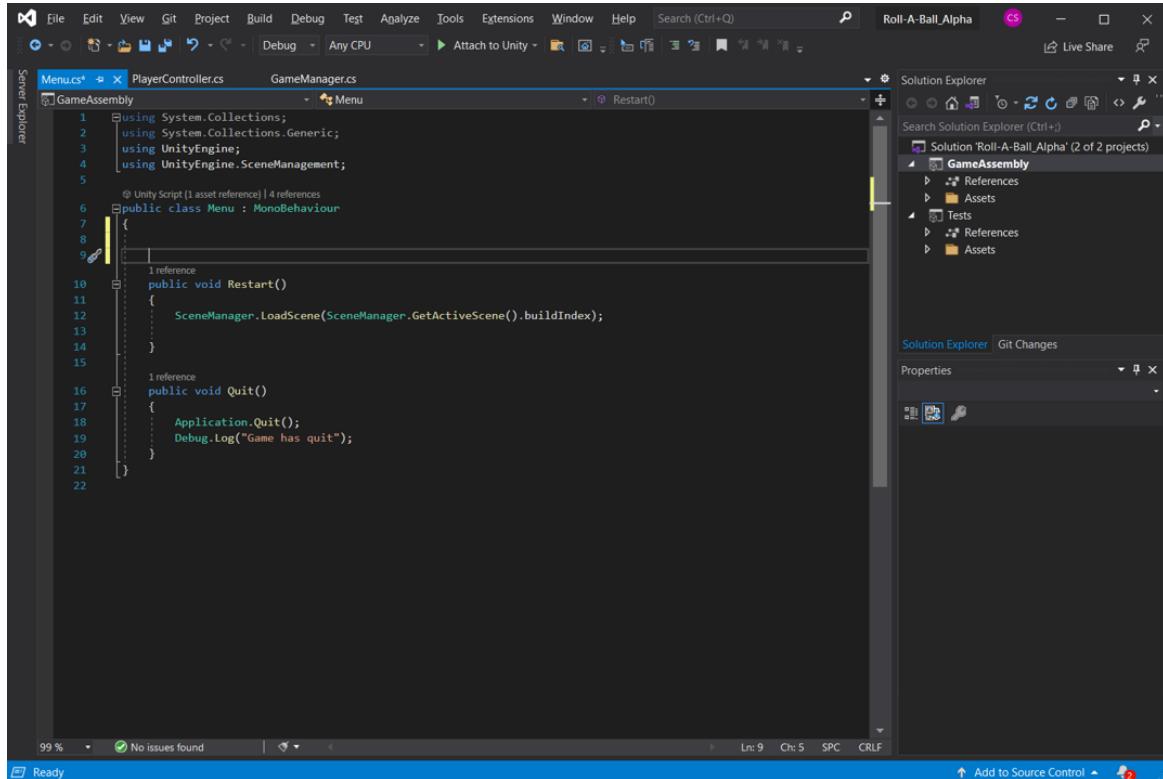
Step 22

- Once the menu script opens, left click once on line 7 ensuring that your mouse cursor is placed after the curl bracket



Step 23

➤ Press enter twice



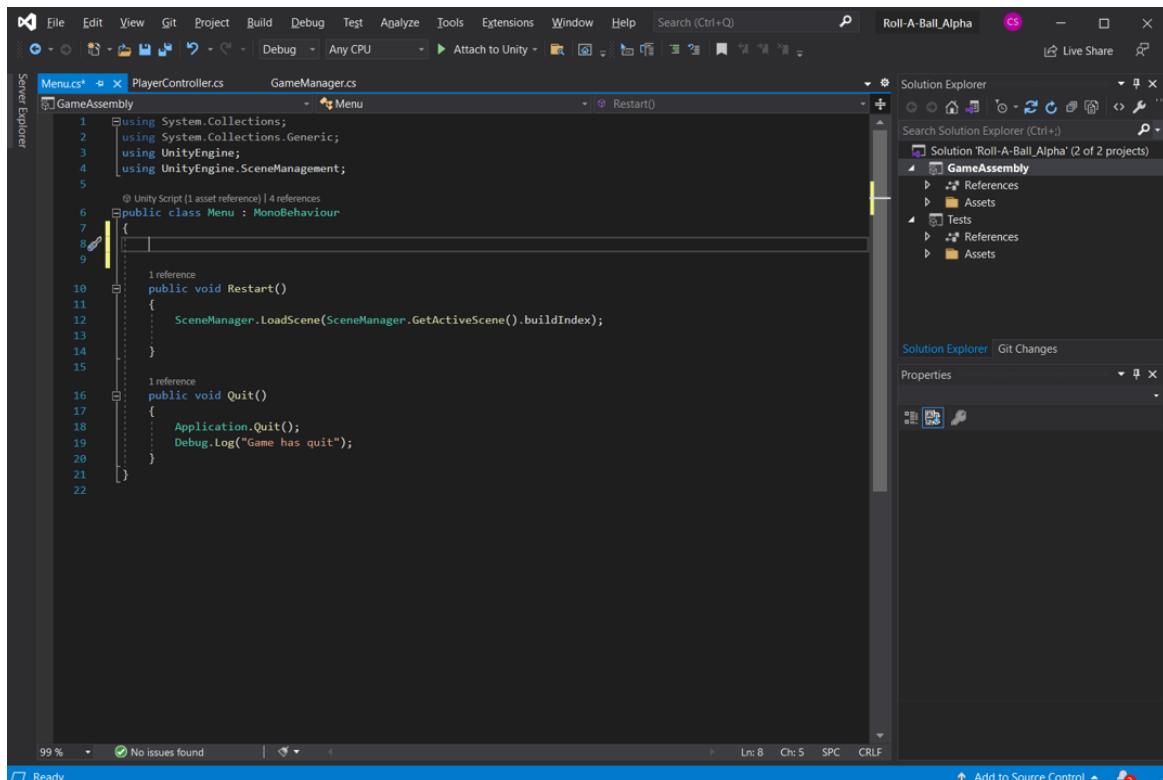
The screenshot shows the Visual Studio IDE with the code editor open. The file is named 'PlayerController.cs' and contains the following C# code:

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.SceneManagement;
5
6  @Unity Script (1 asset reference) | 4 references
7  public class Menu : MonoBehaviour
8  {
9      public void Restart()
10     {
11         SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
12     }
13
14     public void Quit()
15     {
16         Application.Quit();
17         Debug.Log("Game has quit");
18     }
19 }
20
21 }
```

The cursor is positioned at the end of the line '10 }'. The Solution Explorer and Properties windows are visible on the right side of the interface.

Step 24

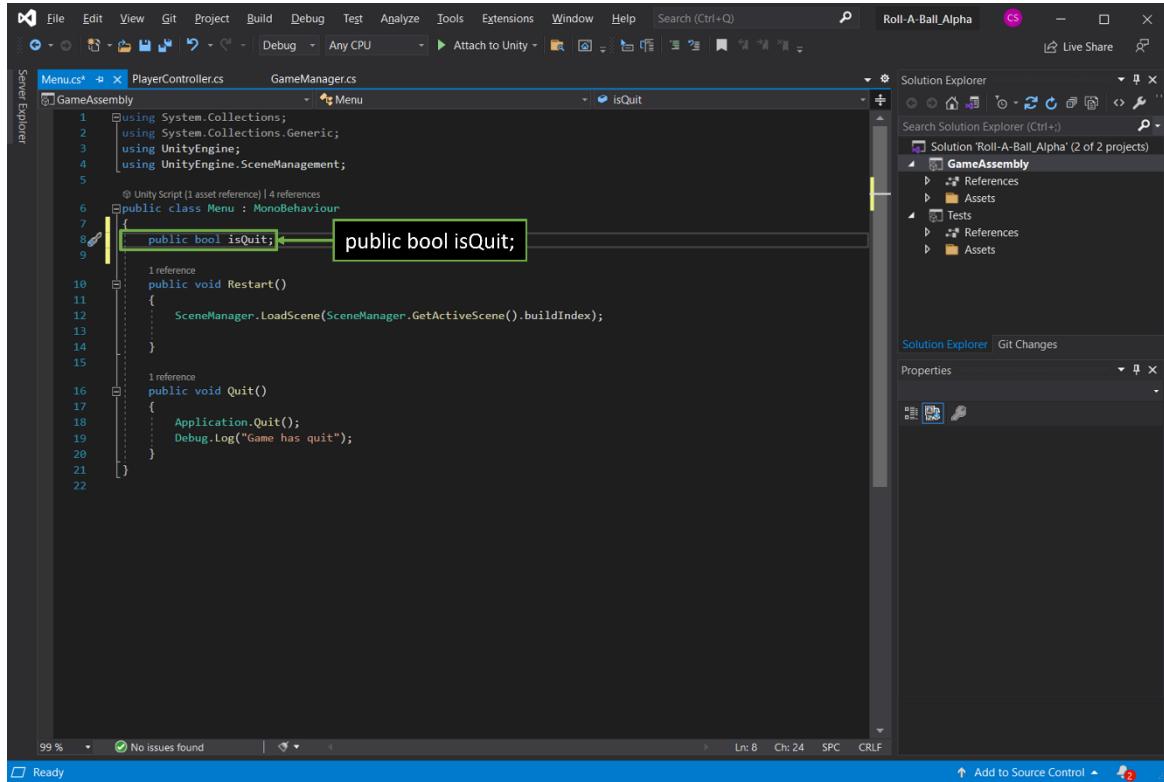
➤ Press the up arrow once



The screenshot shows the Visual Studio IDE with the code editor open. The file is named 'PlayerController.cs' and contains the same C# code as Step 23. The cursor is now positioned at the end of the line '10 }', which is now highlighted in yellow. The Solution Explorer and Properties windows are visible on the right side of the interface.

Step 25

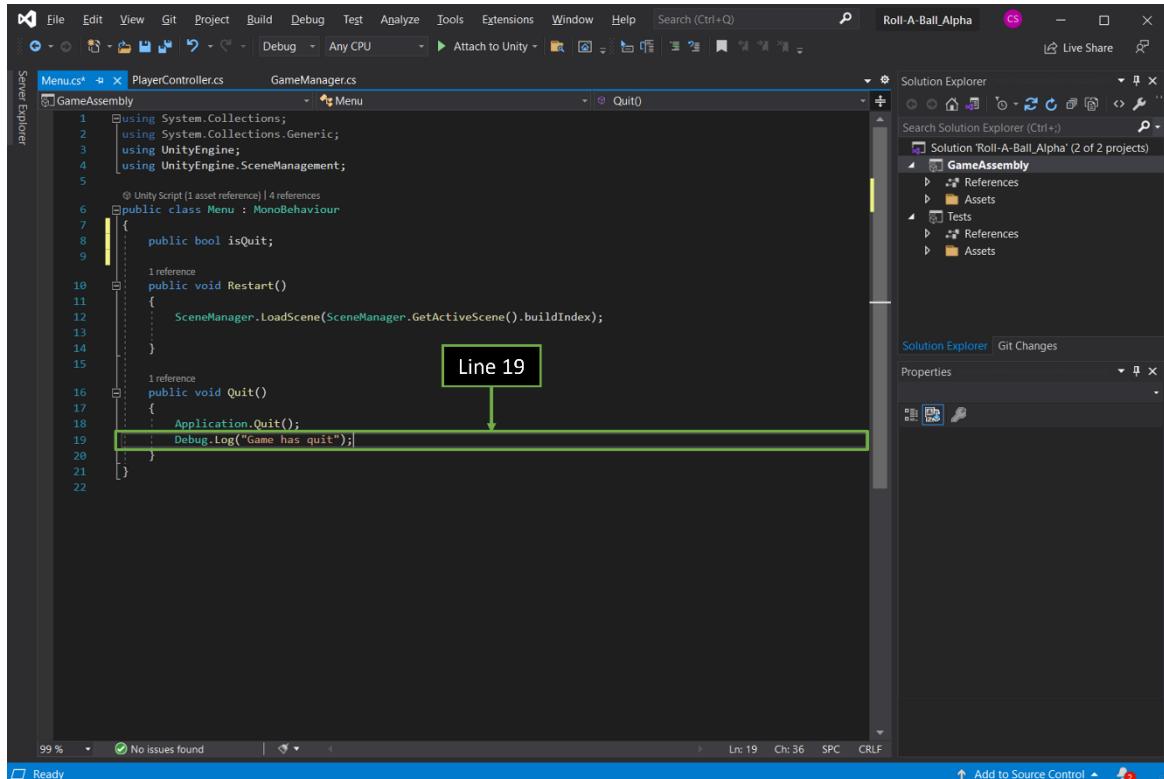
- Type in “public bool isQuit;” without the quotation marks



```
Menu.cs
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.SceneManagement;
5
6  @Unity Script (1 asset reference) | 4 references
7  public class Menu : MonoBehaviour
8  {
9      public bool isQuit;
10
11     public void Restart()
12     {
13         SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
14     }
15
16     public void Quit()
17     {
18         Application.Quit();
19         Debug.Log("Game has quit");
20     }
21 }
22
```

Step 26

- Left click once on line 19, ensuring that your mouse cursor is placed after the semicolon



```
Menu.cs
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.SceneManagement;
5
6  @Unity Script (1 asset reference) | 4 references
7  public class Menu : MonoBehaviour
8  {
9      public bool isQuit;
10
11     public void Restart()
12     {
13         SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
14     }
15
16     public void Quit()
17     {
18         Application.Quit();
19         Debug.Log("Game has quit");
20     }
21 }
22
```

Step 27

Press enter twice

The screenshot shows the Visual Studio interface with the 'PlayerController.cs' file open in the code editor. The code defines a class 'Menu' that implements the 'MonoBehaviour' interface. It contains two methods: 'Restart()' and 'Quit()'. The 'Quit()' method calls 'Application.Quit()' and logs a message. The cursor is positioned at the end of the 'Quit()' method, just before the closing brace. The Solution Explorer on the right shows the 'GameAssembly' project structure.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.SceneManagement;
5
6  @Unity Script (1 asset reference) | 4 references
7  public class Menu : MonoBehaviour
8  {
9      public bool isQuit;
10
11     1 reference
12     public void Restart()
13     {
14         SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
15     }
16
17     1 reference
18     public void Quit()
19     {
20         Application.Quit();
21         Debug.Log("Game has quit");
22     }
23 }
```

Step 28

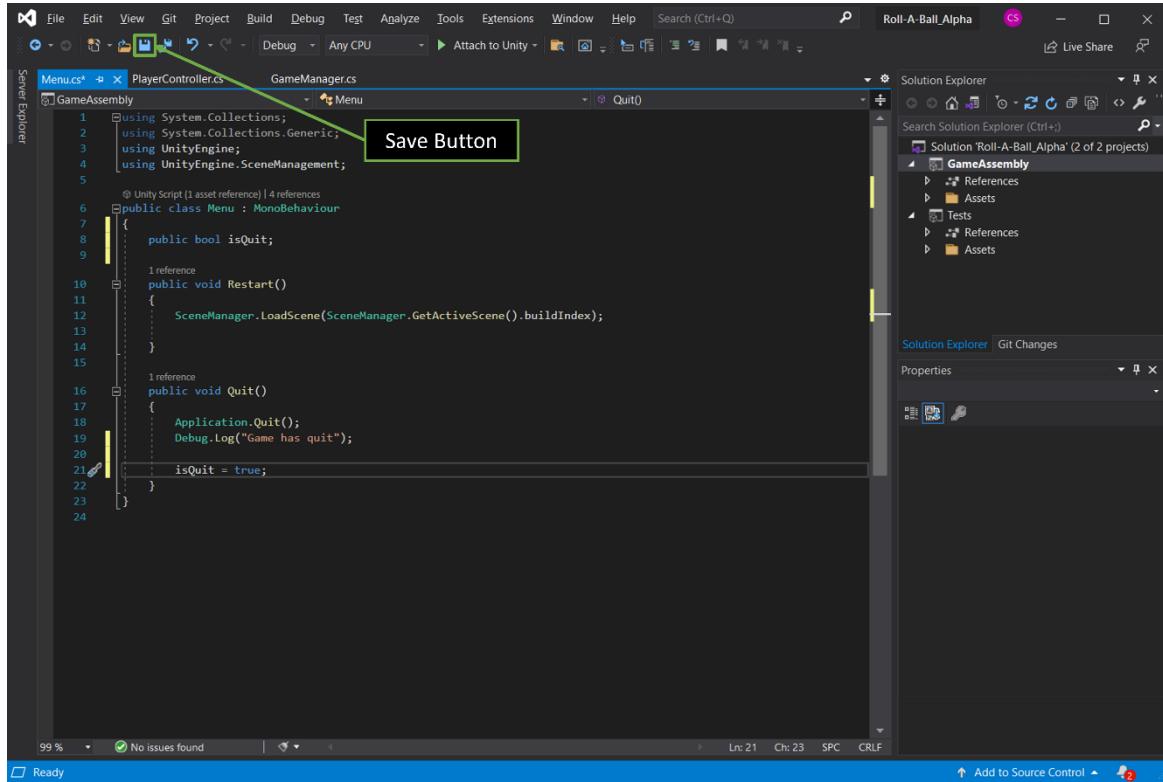
➤ Type in “isQuit = true;” without the quotation marks

The screenshot shows the Visual Studio interface with the 'PlayerController.cs' file open. The 'Quit()' method is visible. A green rectangular box highlights the line 'isQuit = true;'. The cursor is positioned at the start of this line. The Solution Explorer on the right shows the 'GameAssembly' project structure.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.SceneManagement;
5
6  @Unity Script (1 asset reference) | 4 references
7  public class Menu : MonoBehaviour
8  {
9      public bool isQuit;
10
11     1 reference
12     public void Restart()
13     {
14         SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
15     }
16
17     1 reference
18     public void Quit()
19     {
20         Application.Quit();
21         Debug.Log("Game has quit");
22         isQuit = true; // This line is highlighted with a green box
23     }
24 }
```

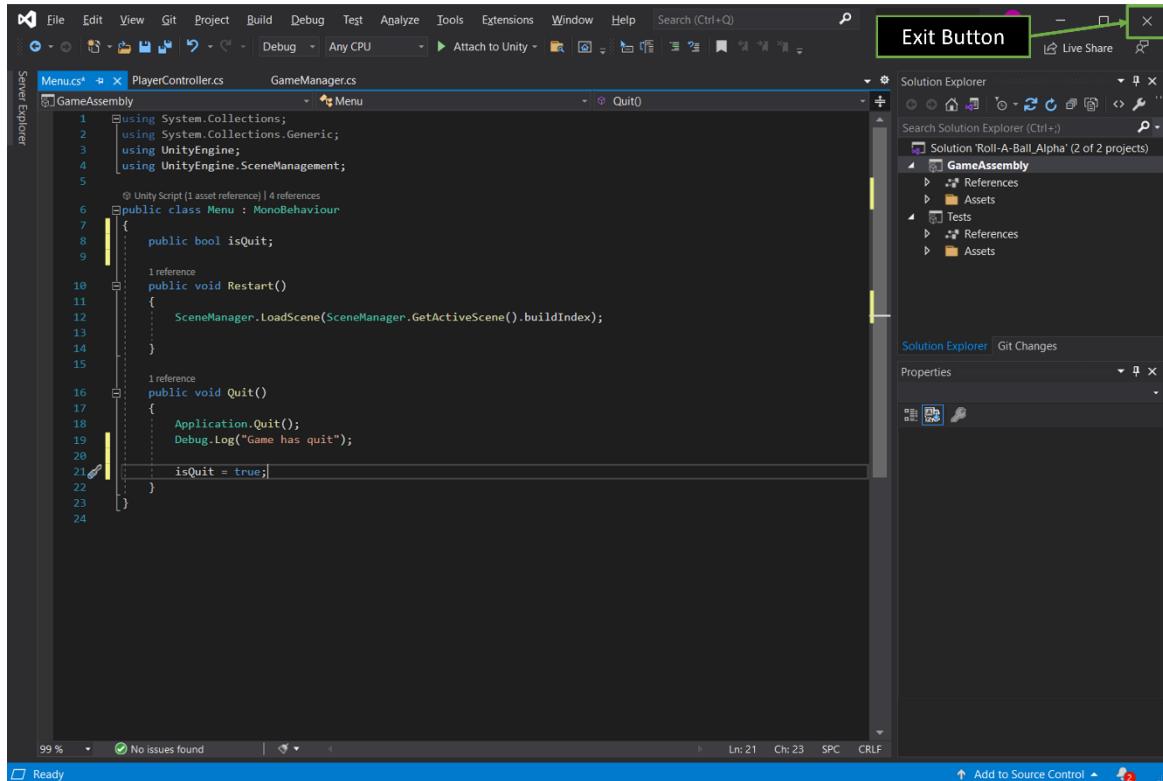
Step 29

- Left click the save button once



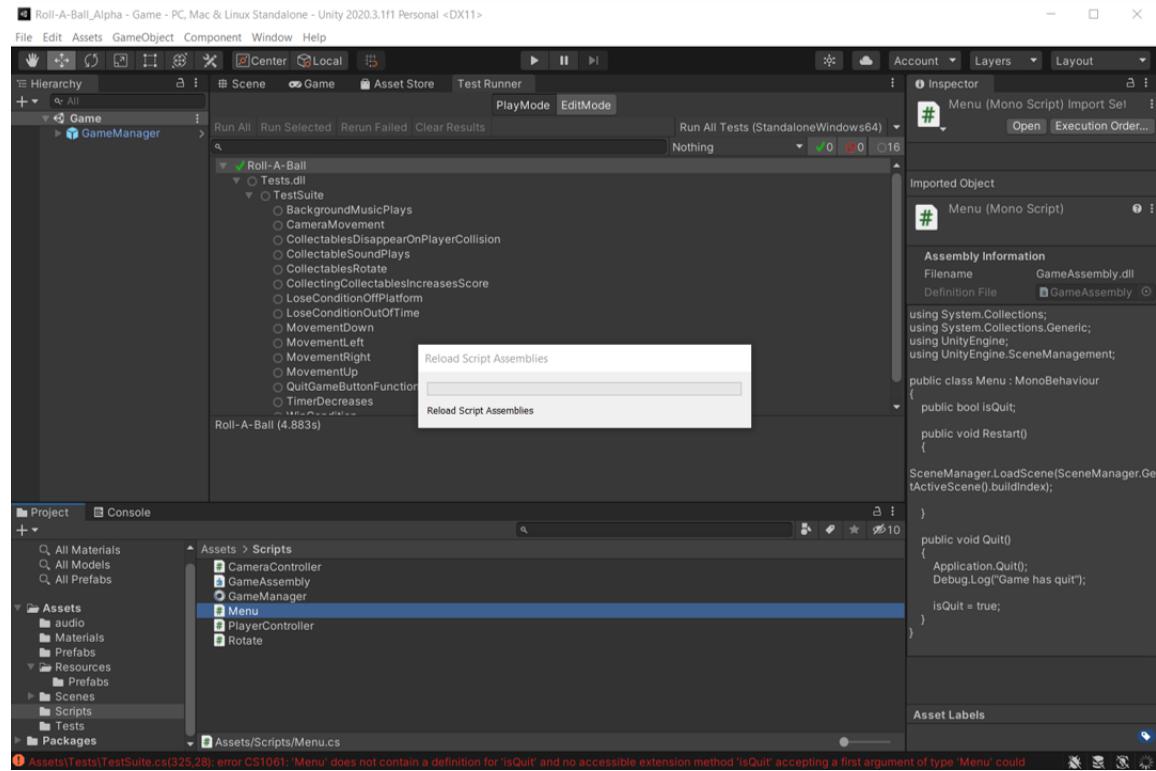
Step 30

- Left click once on the exit button on the top right of the window



Step 32

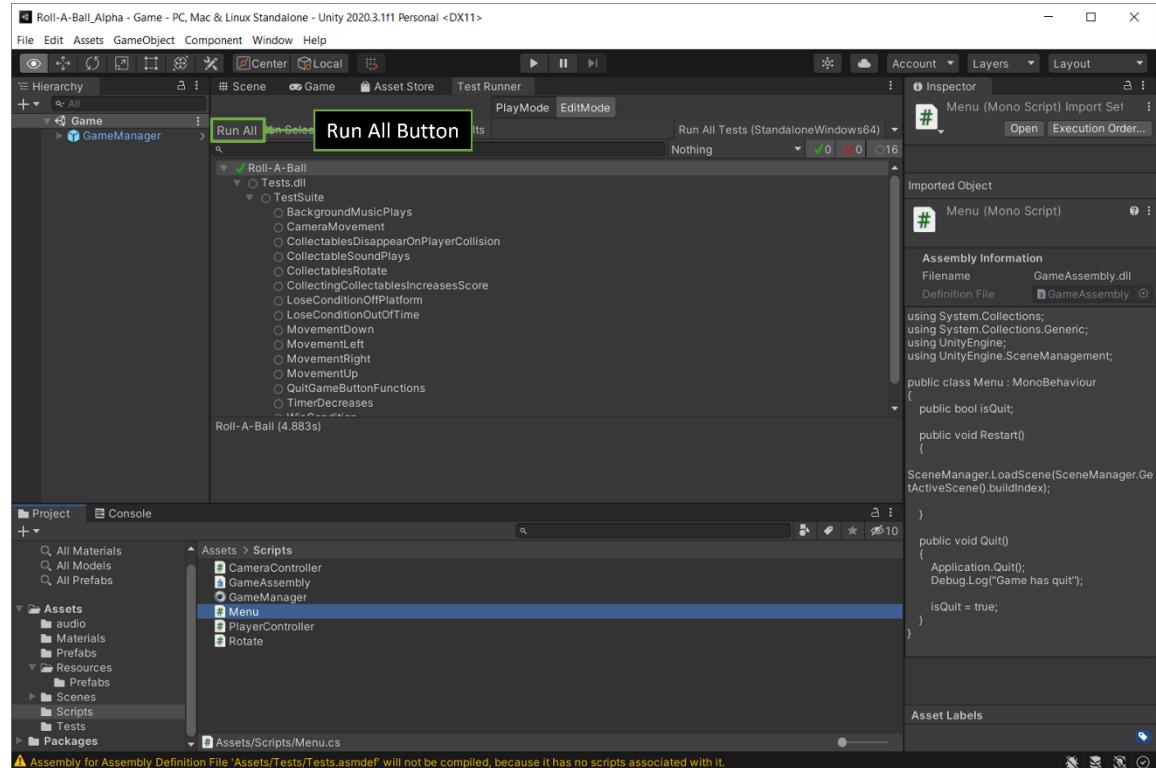
- Wait for the script assemblies to reload



Running The Unit Tests

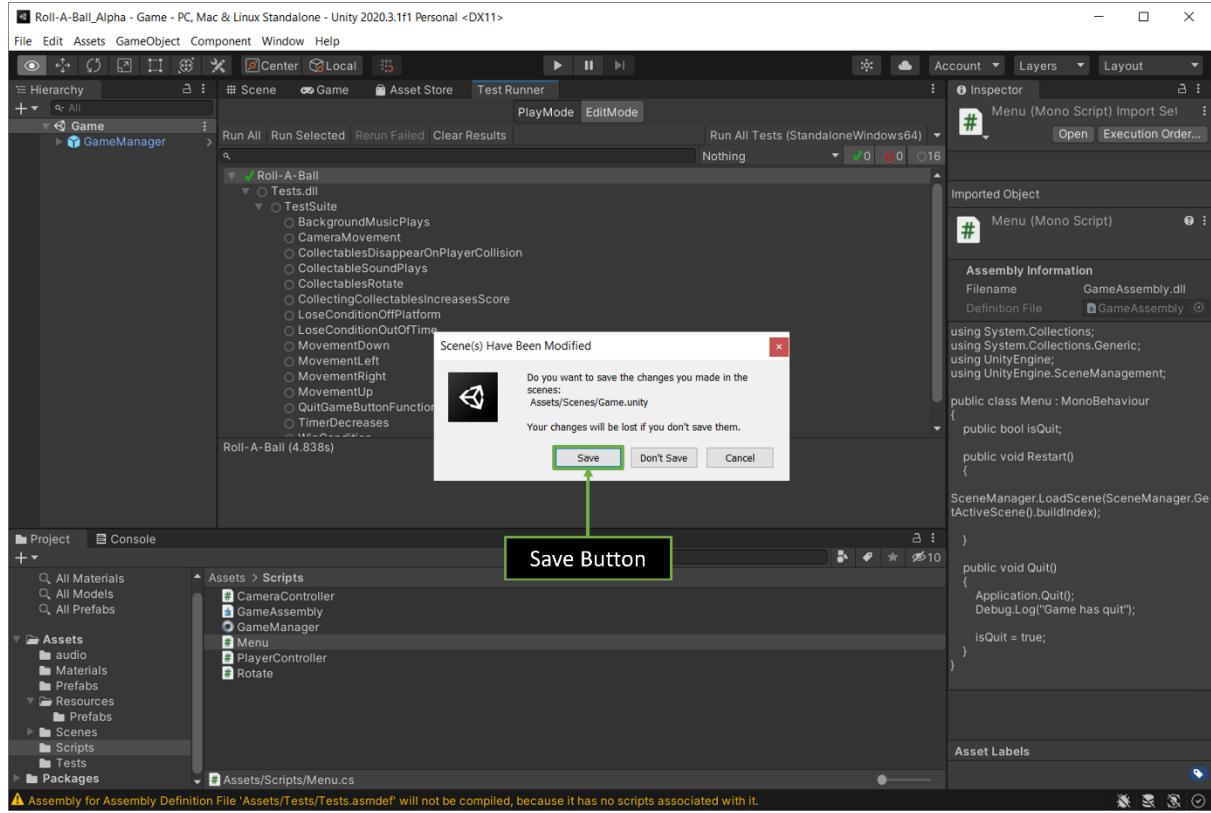
Step 1

- Left click once on the “Run All” button within the test runner window



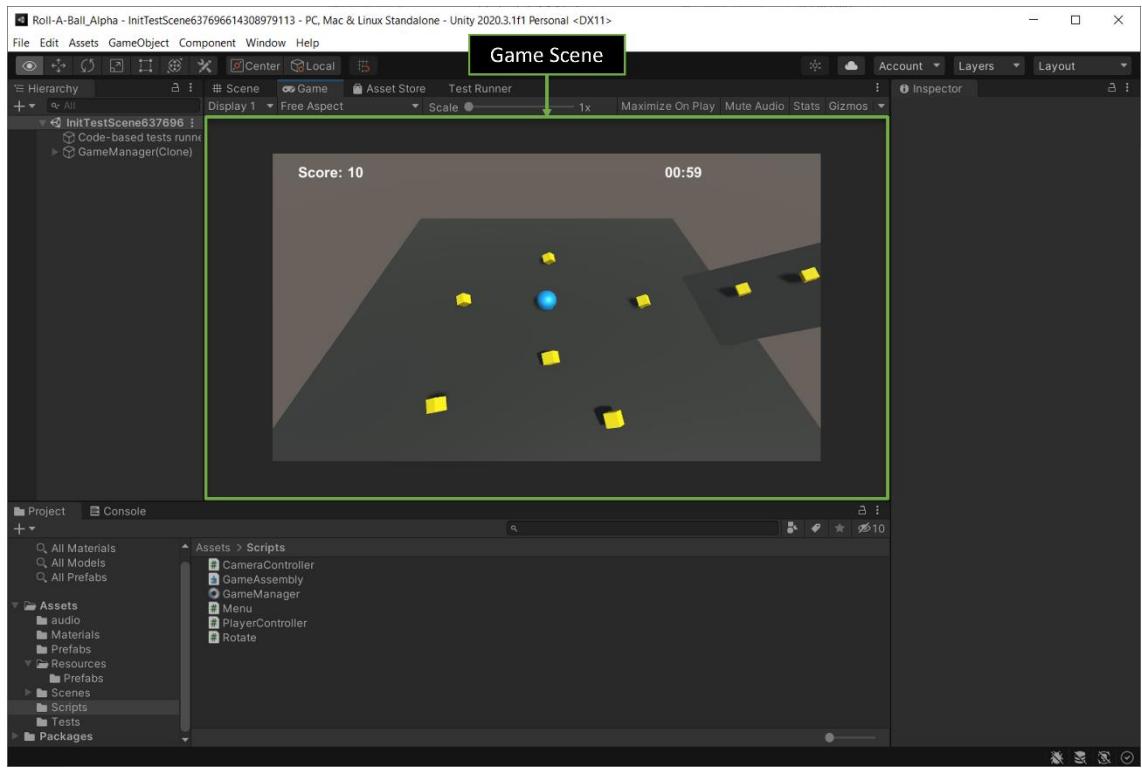
Step 2

- Once the Scene(s) Have Been Modified window appears, left click once on the “Save” button



Step 3

- Once you have clicked the save button, the center tab will change to the “Game” scene and run the unit tests



Step 4

- Once the unit tests have ran, left click once on the “Test Runner” tab to see the unit test results

