Tutorial – GUI Design

In this tutorial we will cover everything you will need for your game in terms of GUI.

This includes a main menu, displaying the player's current score during the game, and adding a high scores screen.

There are a lot of components to this tutorial, and a lot of steps to get this working. We'll break it down into manageable pieces so this doesn't get too overwhelming.

Creating all the Scenes:

We'll start with the Main Menu first, since that will be the easiest for us to get working, and it's also easy to test.

Before we get started on the main menu though, we need to create all the scenes in our game.

At the stage you probably already have a game. It doesn't really matter what state your game is, as long as you have a scene set up we can use it. This scene will be our game scene. I recommend you call your scene "Game", but if you want to call it something else just make the appropriate changes when the time comes (I'll let you know).

We'll also need 2 other scenes – one for the main menu and one for displaying the high scorers.

From the **File** menu select **New Scene**. Save this scene with the name "Menu". This will be the scene we'll use for the Main Menu.

Create another new scene and save it with the name "Highscores". We'll use this screen to display a list of the high scores.

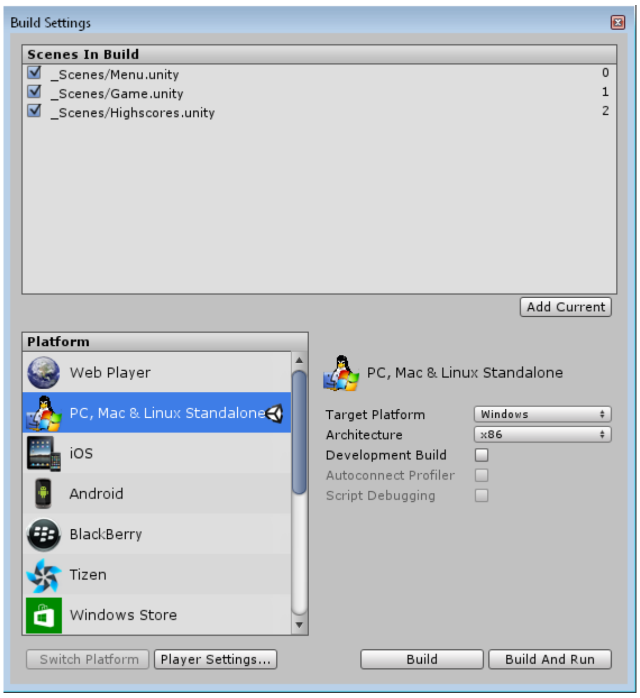
Now that we've created all the scenes we'll need in our game, there's just one more step – adding all the scenes to our build.

From the **File** menu select **Build Settings**. The build settings dialog will open.

From the **Project** window drag all of your scenes into the **Build Settings** window (into the section that says **Scenes in Build**).

We need to add the scenes in order, so make sure the Menu scene is first, followed by the Game scene and then the Highscores scene.

The window should look like this when you've finished:


Build Settings

Scenes In Build
- ✓ _Scenes/Menu.unity 0
- ✓ _Scenes/Game.unity 1
- ✓ _Scenes/Highscores.unity 2

Platform: PC, Mac & Linux Standalone

Target Platform	Windows
Architecture	x86
Development Build	☐
Autoconnect Profiler	☐
Script Debugging	☐

When you have set the build settings, close the window.

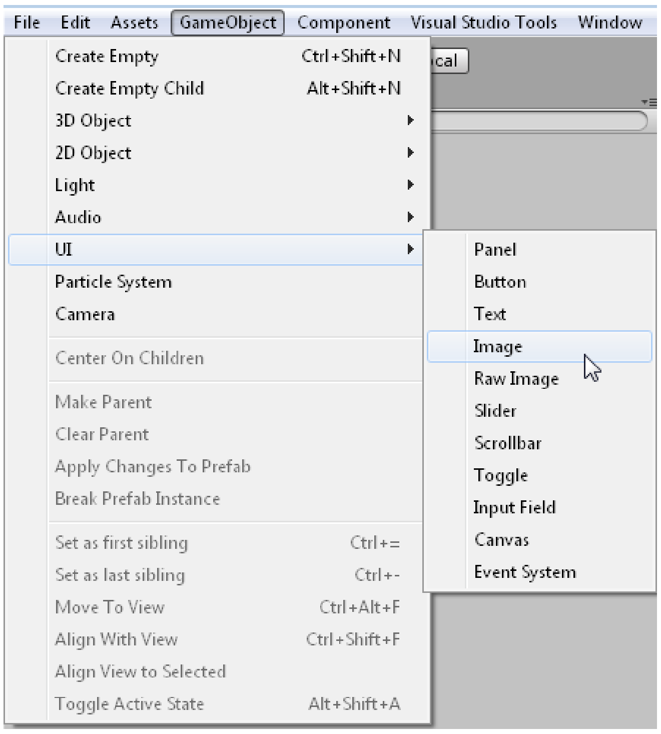
Creating the Main Menu:

Adding a Background:

Open the Menu scene. It’s time to set up our Main Menu.

If you created an empty scene and didn’t change anything, then the only objects you should see in the Hierarchy are the Camera and a Directional Light. You can leave these as is.

From the **Game Object** menu, select **UI** and then **Image**.

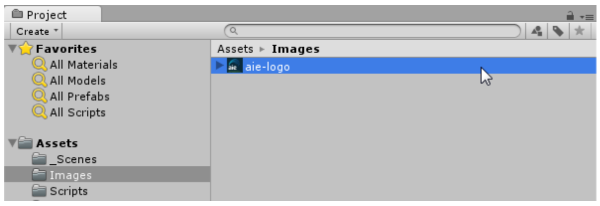


This will add an Image object to the scene. Because we haven’t created a Canvas object yet, it will also create the Canvas object and add it to the scene.

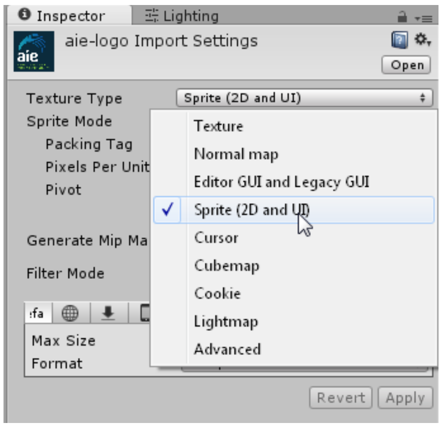
I renamed my Image object to “Image Background”

Now we need an image to use for the background. Find an image on the Internet (or create one) and add it to your project (PNG images work best).

Select the Image in the **Project** window.

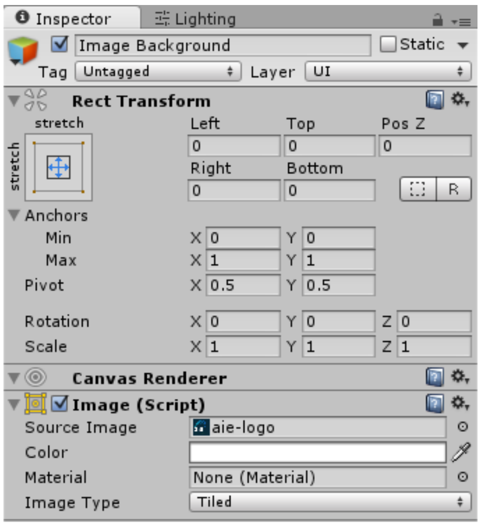


From the **Inspector** window, change the **Texture Type** to "Sprite (2D and UI)"



Now that we've set up our image as a Sprite, we can use it in our Image object in the UI.

Select the Image object in the **Hierarchy**, and in the **Inspector** window set your newly created Sprite as the **Source Image**.

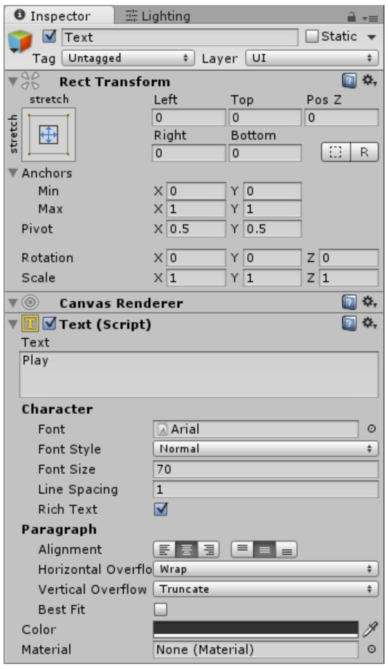


Adding Buttons:

Select **Game Object** -> **UI** -> **Button**. This will add a new Button object to the Canvas.

If the Button isn't already positioned below the Image in the **Hierarchy**, then do that now (you can drag it). We need to do this because the order UI elements are drawn in is the same as their order in the **Hierarchy**. Because we want the background image in the background (below all other elements), it needs to be the first element in this hierarchy.

Expand the Button element in the hierarchy to display its child Text element. From the **Inspector** window, change the text of this element to “Play”.



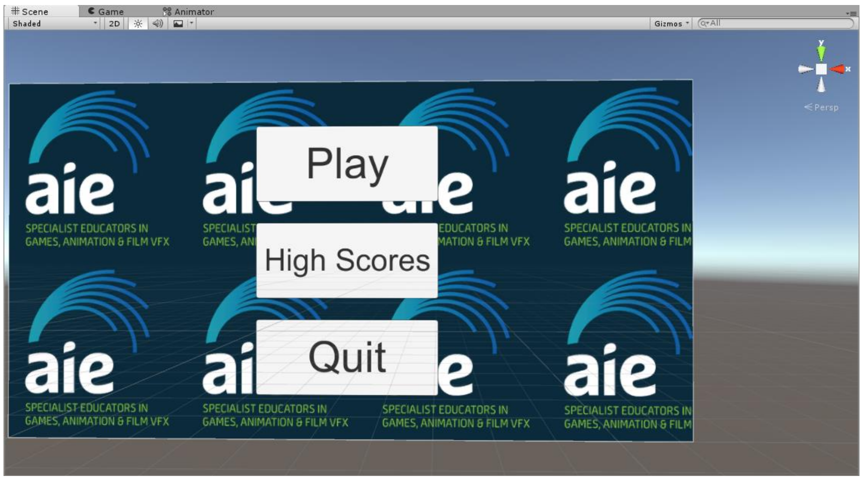
Text | Static
Tag Untagged | Layer UI

Rect Transform
stretch
Left 0 | Top 0 | Pos Z 0
Right 0 | Bottom 0
Anchors
Min X 0 Y 0
Max X 1 Y 1
Pivot X 0.5 Y 0.5
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

Canvas Renderer

Text (Script)
Text
Play
Character
Font Arial
Font Style Normal
Font Size 70
Line Spacing 1
Rich Text ✓
Paragraph
Alignment
Horizontal Overflo Wrap
Vertical Overflow Truncate
Best Fit
Color
Material None (Material)

Add two more buttons to the scene – “High Scores” and “Quit”. Arrange the buttons until you’re happy with the way they are formatted.



Adding the Scripts:

We’ve done all that we need to in the Editor for the moment. It’s time to write the code that will control what happens when these buttons are pressed.

Add two new C# scripts:

- ButtonLoadScene, and
- ButtonQuitApplication

The code for ButtonLoadScene is as follows:

```
using UnityEngine;
using System.Collections;

public class ButtonLoadScene : MonoBehaviour {

    public string _sceneName = string.Empty;

    public void OnButtonPressed()
    {
        Application.LoadLevel(_sceneName);
    }
}
```

And the code for ButtonQuitApplication is:

```
using UnityEngine;
using System.Collections;

public class ButtonQuitApplication : MonoBehaviour {

    public void OnButtonQuit()
    {
        Application.Quit();
    }
}
```

The code for these two scripts is similar. We are adding a new function to each script that we will call when the button is pressed.

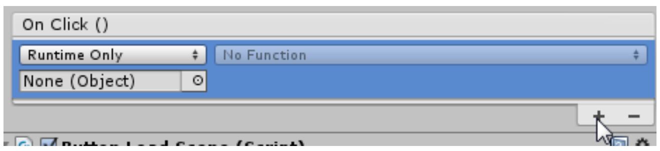
In ButtonLoadScene, the OnButtonPressed function will load whatever scene we've specified in the _sceneName field. This means we can use the same script anywhere in our program where we have a button that should open another scene.

The ButtonQuitApplication script will handle what happens when the 'Quit' button is pressed. In the OnButtonQuit function we call Application.Quit() to close the program. Note that when we run in the Unity Editor this function will do nothing. But when we build our final stand-alone executable, then this function will close the game.

Go back to the Unity Editor and lets hook up our scripts.

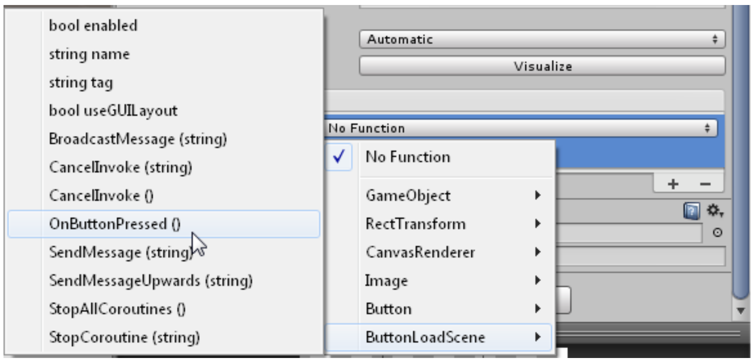
Select the Play button and add the ButtonLoadScene script as a component.

We need to specify which function to execute when the user presses the Play button, so in the **Inspector**, press the **+** icon where it says **OnClick()**

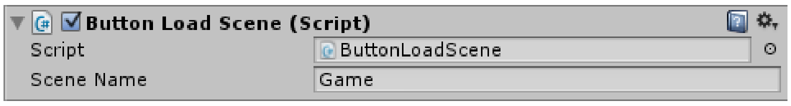


Drag the Play button from the **Hierarchy** into the field that currently says “None (Object)”. The name of your Play button object should now be displayed in this field.

From the **Function** drop down list, select the **ButtonLoadScene** script, and then the **OnButtonPressed ()** function.



The last step is to tell the **ButtonLoadScene** script which scene to load. With the Play button selected in the **Hierarchy**, go to the **Button Load Scene (Script)** options and in the **Scene Name** field enter the name of your game scene. (For me this is “Game”. If you called your scene something different then enter that instead.)



Button Load Scene (Script)

Script	ButtonLoadScene
Scene Name	Game

That completes the setup for the Play button.

The High Scores button will be almost exactly the same (we add the same ButtonLoadScene script), but the **Scene Name** will be “Highscores” instead.

The final button – “Quit” – is a little different. We want to add the **ButtonQuitApplication** script to this button. For the **OnClick ()** settings we want to add **ButtonQuitApplication** -> **OnButtonQuit ()**. There is no **Scene Name** field in this script, so that completes the setup.

Run your game now. You should be able to click the Play button to load your game scene, and click the High Scores button to load the Highscores scene. Clicking the Quit button won’t do anything – we’ll need to build and run it as a stand-alone executable to test this.

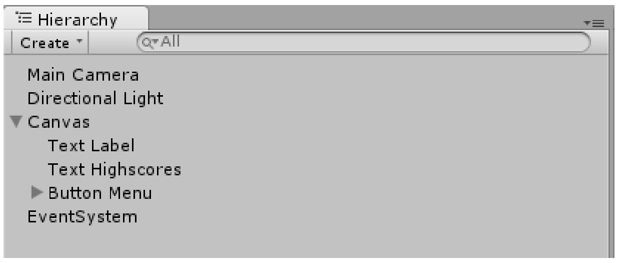
That concludes setting up the Main Menu. Next, we’ll set up the Highscores scene.

Creating the High Scores Scene:

Open your Highscores scene.

This scene will have one Button object (to get us back to the main menu), and two Text objects (one we'll use as a label, and the other we'll use to display a list of high scores).

Add two Text objects and one Button object to your scene.



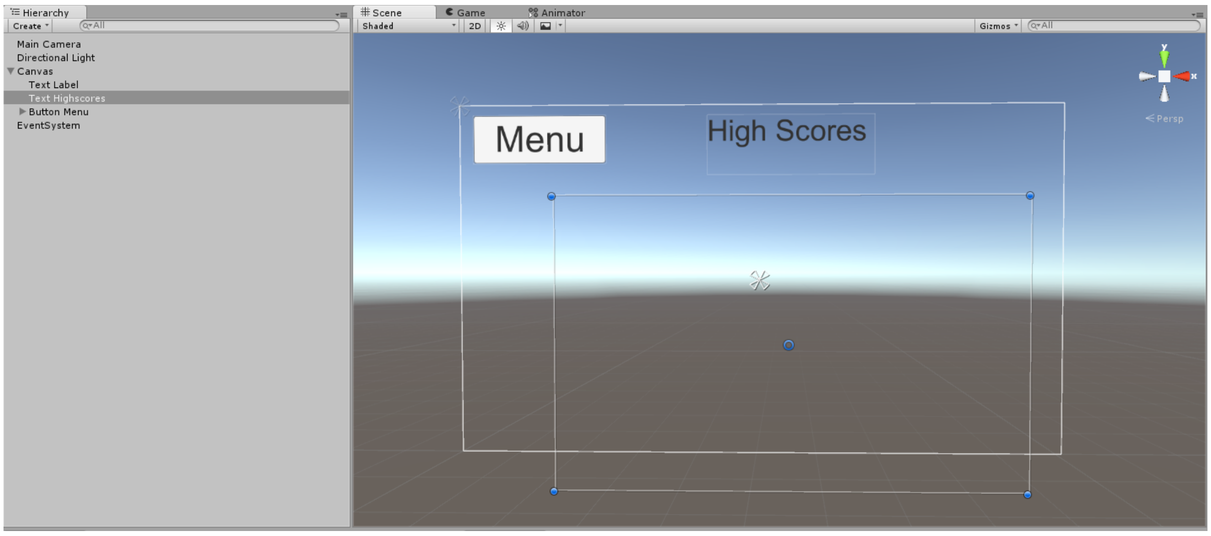
```
Main Camera
Directional Light
▼ Canvas
    Text Label
    Text Highscores
  ▶ Button Menu
EventSystem
```

Select one of the Text objects and set its text to "High Scores". Position it in the centre of the screen, somewhere close to the top. You may also want to rename this object to something like "Text Label".

Select the other Text object and delete all its text. Resize this object so that it is large enough to hold 10 lines. A with of 800 and a height of 500 should be enough. Call this Text object "Text Highscores".

You may want to increase the font size of each Text object. A Font Size of 36 is good.

Finally set the text of your Button object to "Menu", and arrange the objects on your screen until you are happy.



Add the **ButtonLoadScene** script to your Button object, and set the **Scene Name** to "Menu". Don't forget to set the **On Click()** property of the button.

Displaying the High Scores:

With our scene set up we need to add some code that will add the list of high scores to the Text Highscores text box when the scene is loaded.

Add a new empty game object to the scene (**Game Object** -> **Create Empty**). Rename this object to "Highscores".

On Portal there is a script called **HighScoresNewComplete** (under session 5). These script contains all the code you will need to save and load a list of high scores to and from a text file. Download this file, add it to your project, and then add this script to the Highscores object we just created.

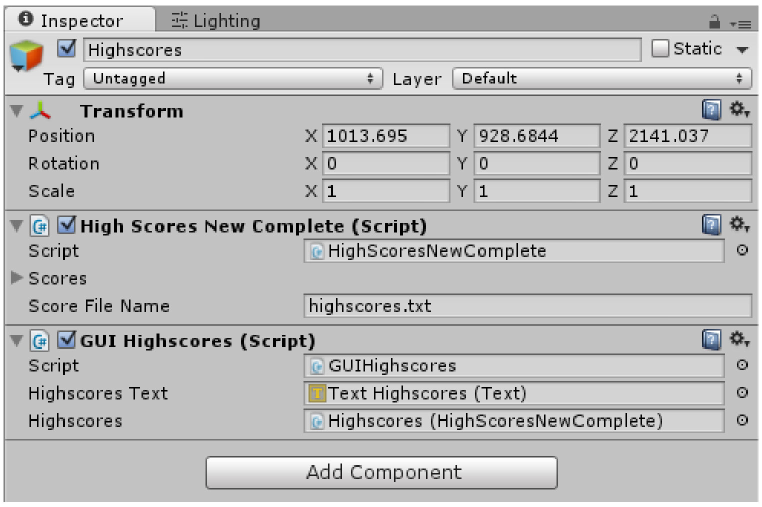
We also need to create a new script that will coordinate the process of loading the scores from the file and copying them into the Text object. Create a new C# script and call it **GUIHighscores**. Add this script to the Highscores object too.

Open the GUIHighscores script in monodevelop and update it as follows:

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;

public class GUIHighscores : MonoBehaviour {

    public Text _highscoresText = null;
    public HighScoresNewComplete _highscores = null;

    private bool _loaded = false;

       // Use this for initialization
       void Start () {
       }

       // Update is called once per frame
       void Update () {
        if (_loaded == false)
        {
            _loaded = true;
            _highscores.LoadScoresFromFile();

            foreach (int score in _highscores.scores)
            {
                _highscoresText.text += score.ToString() + "\n";
            }
        }
       }
}
```

When this script executes, if the _loaded variable is false, then the scores will be loaded from the saved file using the HighScoresNewComplete script. Each score is then copied into a new line in the Text object.

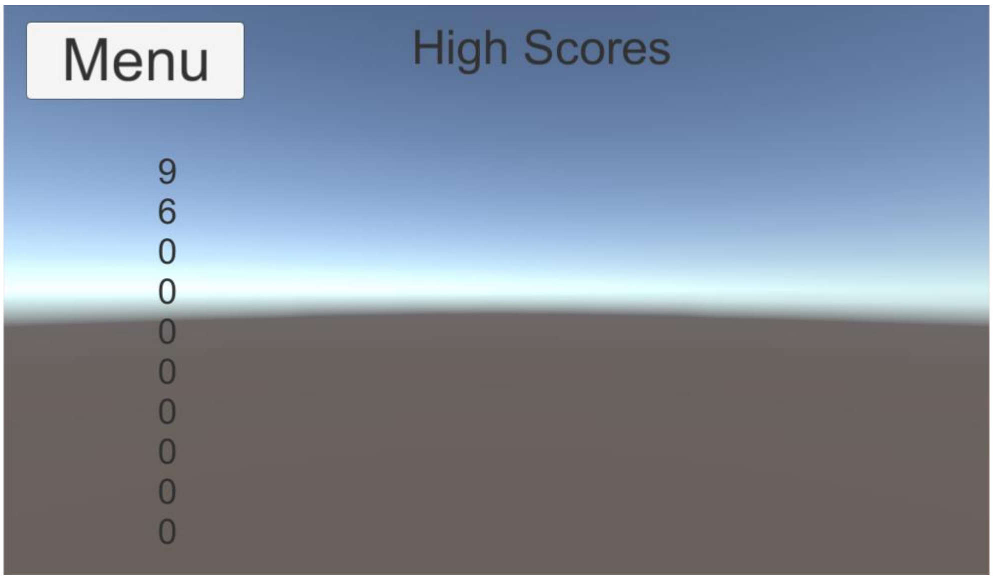
Back in the Unity Editor, with the **Highscores** object selected, drag the Text object you use to display your high scores (Text Highscores) into the **Highscores Text** field. Then drag the **Highscores** object into the **Highscores** field.



```
Inspector | Lighting
[✓] Highscores                                   [ ] Static
Tag  Untagged            Layer  Default
Transform
  Position   X 1013.695   Y 928.6844   Z 2141.037
  Rotation   X 0          Y 0          Z 0
  Scale      X 1          Y 1          Z 1
[✓] High Scores New Complete (Script)
  Script            HighScoresNewComplete
  Scores
  Score File Name   highscores.txt
[✓] GUI Highscores (Script)
  Script            GUIHighscores
  Highscores Text   Text Highscores (Text)
  Highscores        Highscores (HighScoresNewComplete)
Add Component
```

You should now have a functioning High Scores screen.

Play your game to test it. You should see your list of high scores displayed in the Text box. If you see 10 “0”s, then try editing the highscores.txt file by hand.



Adding a HUD to the Game Scene:

We need to add a Heads Up Display (HUD) to the game scene. A HUD typically contains information we want the player to know about – like their health, their current score, or maybe even a mini-map.

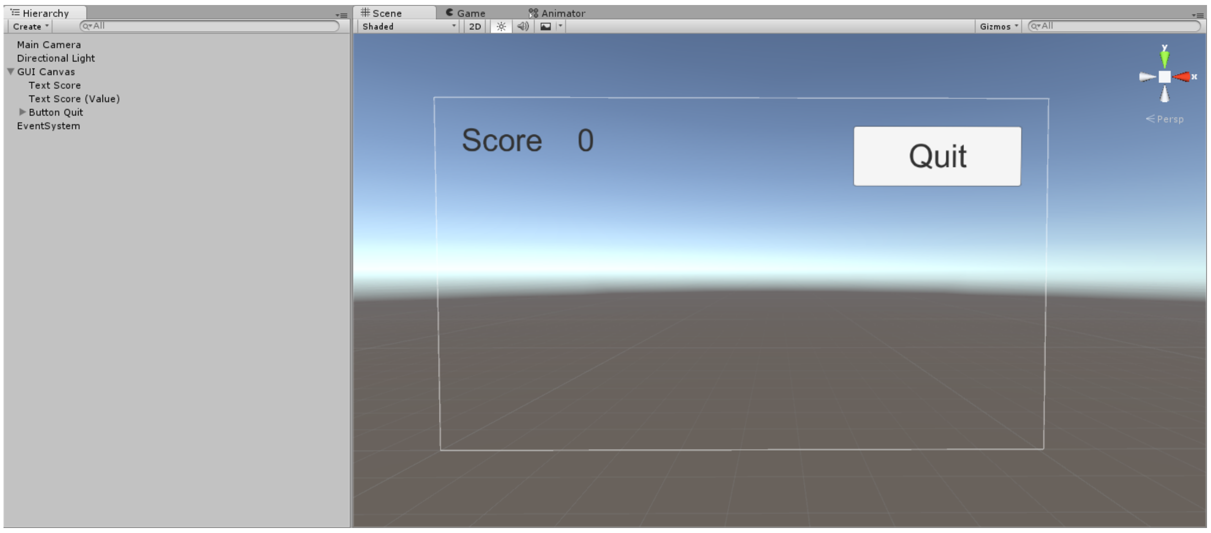
For our game our HUD will contain the player's current score, and a button to return to the main menu.

Add a Text object to your scene, rename it to "Text Score", and set the text to "Score".

Add another Text object, call it "Text Score (Value)", and set the text to "0".

Finally, add a Button object to the scene, call it "Button Quit" and set its text to "Quit".

Position the objects on your screen so that you are happy with them.



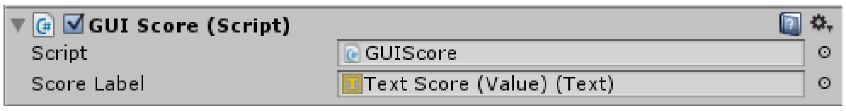
Now that we have a Text object that will display a score, we need some way to set the score. We'll use a script for this.

We need to attach a script to the **Text Score (Value)** object that will allow any other object in our game to get the current score or to increase the score.

Add a new C# script called **GUIScore** and set the code as follows:

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;

public class GUIScore : MonoBehaviour {

    public Text _scoreLabel = null;
    private int _score = 0;

    public int GetScore()
    {
        return _score;
    }

    public void AddScore(int value)
    {
        _score += value;
        _scoreLabel.text = _score.ToString();
    }
}
```

Add this script to the **Text Score (Value)** object. With this object selected in the **Hierarchy**, drag the **Text Score (Value)** object into the **Score Label** field.


GUI Score (Script)
Script: GUIScore
Score Label: Text Score (Value) (Text)

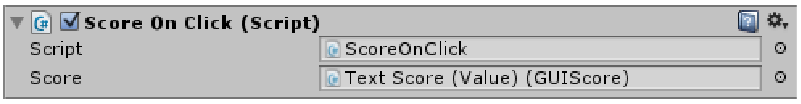
Updating the Player's Score:

Because I don't have a game set up, I want something simple that will modify the score. I've created a simple script that will increase the score every time the player presses the left mouse button.

To add this script to your game, add an empty game object and set its name to "ScoreOnClick". Create the following C# script, called **ScoreOnClick** and add it to the ScoreOnClick object:

```
using UnityEngine;
using System.Collections;

public class ScoreOnClick : MonoBehaviour {
    public GUIScore _score = null;

       // Update is called once per frame
    void Update () {
        if (Input.GetMouseButtonDown(0) == true)
        {
            _score.AddScore(1);
        }
    }
}
```

Back in the Editor, with the **ScoreOnClick** selected in the **Hierarchy**, drag the **Text Score (Value)** object into the **Score** field.



Script	ScoreOnClick
Score	Text Score (Value) (GUIScore)

This gives our **ScoreOnClick** script access to the game object that keeps track of and displays the score. So now we have a way of modifying the player's score, and a way to display it to the screen. We're almost done!

Saving the Score on Exit:

We need to add the **HighScoresNewComplete** script to our game so that we can save our score to the list of high scores.

Create a new empty game object and call it "Highscores". Add the **HighScoresNewComplete** script to this object.

The final step is to make sure this script is called before we exit so that we can add our player's score to the list of high scores and save the list to the highscores.txt file.

Create a new C# script called "ButtonQuit". Edit the code as follows:

```
using UnityEngine;
using System.Collections;

public class ButtonQuit : MonoBehaviour {

    public string _sceneToLoad = string.Empty;
    public GUIScore _score = null;
    public HighScoresNewComplete _highscores = null;

    public void OnButtonQuit()
    {
        int score = _score.GetScore();
        if (_highscores != null)
        {
            _highscores.AddScore(score);
            _highscores.SaveScoresToFile();

            Application.LoadLevel(_sceneToLoad);
        }
    }
}
```

Add this script to your **Button Quit** object. With the object selected in the **Hierarchy**, set the **Scene To Load** field to "Menu.

Drag the **Text Score (Value)** object into the **Score** field.

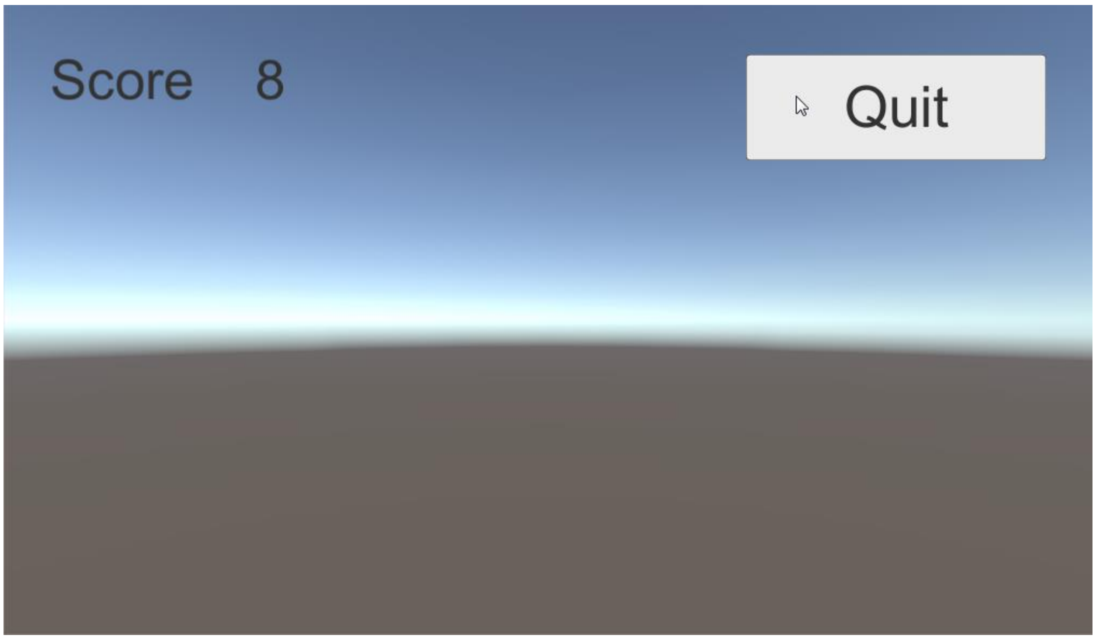
Finally, drag the **Highscores** object into the **Highscores** field.

Don't forget to set the **On Click ()** property of the button to **ButtonQuit.OnButtonQuit**.

When the user presses the Quit button, the OnButtonQuit () function will execute. This will get the current score from the Text Score (Value) object and add it to the list of highscores. Then the

SaveScoresToFile function is executed in the HighSocresNewComplete (which is attached to the Highscores game object) and everything is saved to the file. Finally the Menu scene is loaded and the player will be returned to the main menu.

Run your game new. You should be able to increase the player's score and see the GUI text update. When you press the Quit button, the game should save the high score list to the highscores.txt file and then load and display the main menu.



This has been a long tutorial, but if you made it this far you should have all the GUI components you need to complete your game.