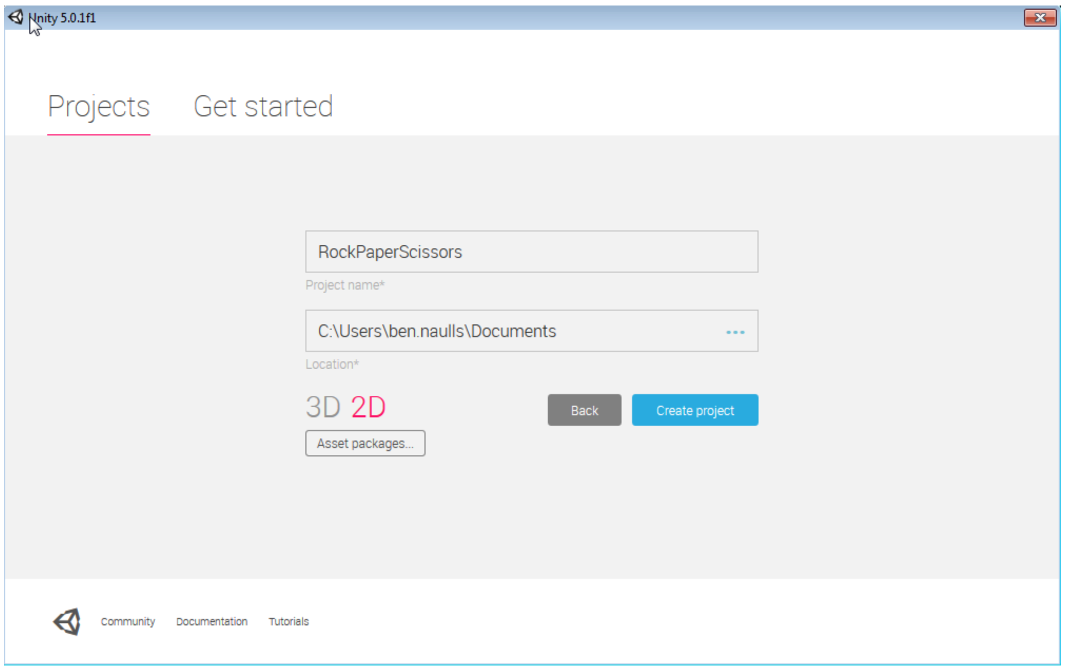
Rock Paper Scissors
A Unity UI Tutorial

Setting up the project

In Unity, create a new project.

Switch the default layout to **2D** (this only changes a few default setting in Unity and can be changed later).



Project name*: RockPaperScissors

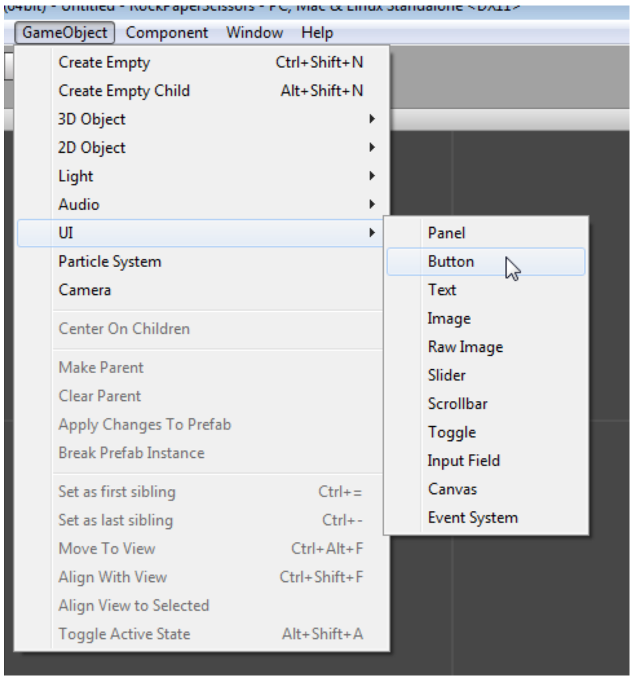
Location*: C:\Users\ben.naulls\Documents

3D 2D

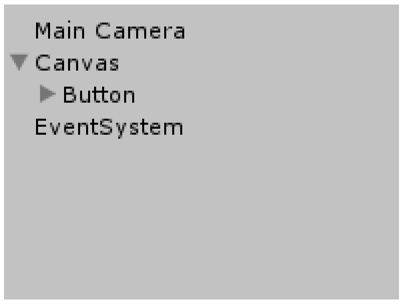
Go to File > Save Scene. Save it as Main.scene in a new folder called Scene (in your project's Assets folder). And let's start!

Creating the UI layout

Create a Button in your scene…



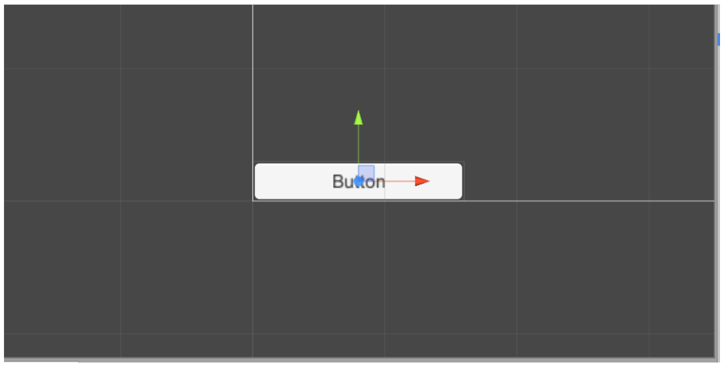
You will see that Unity has created several other objects along with your button to help manage your UI…



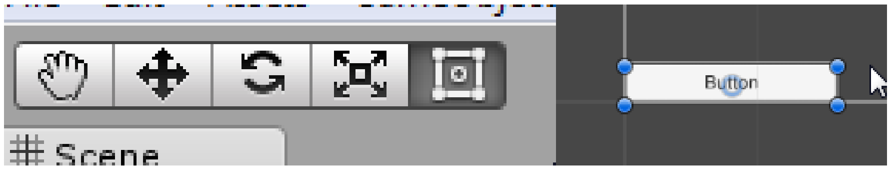
The **Canvas** is the object that holds all your UI objects. by default it is stretched over the entire screen on top of your camera.

The **EventSystem** is an object that handles the UI interface. It manages things such as button clicks and moving sliders.

Double click on the button in the Hierarchy to center the Scene on your new button.

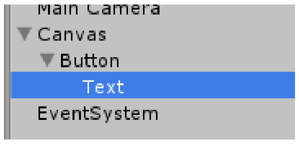


You can use the **UI Object manipulation tool to** resize and move the UI objects.

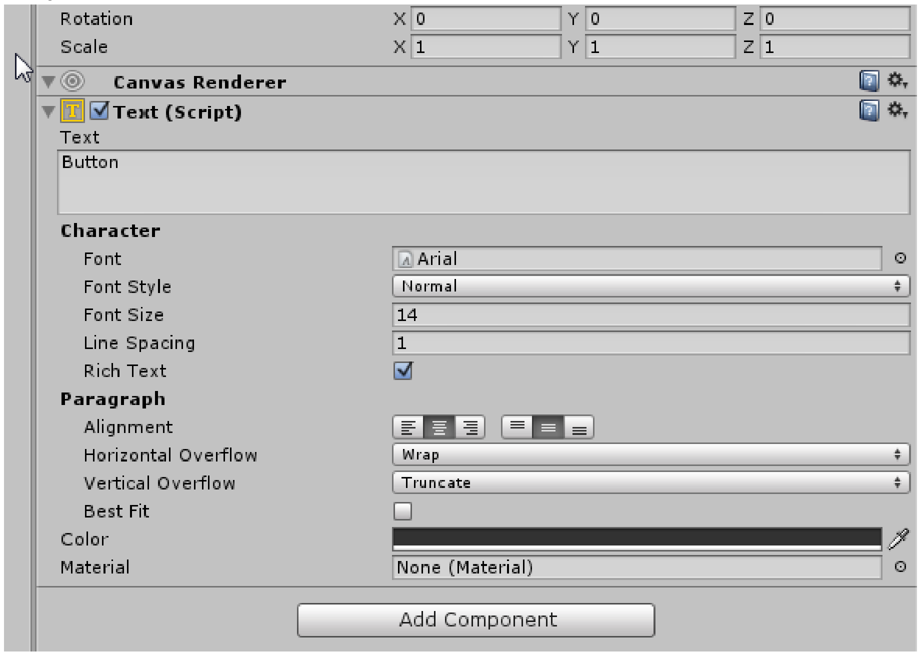


Now rename the **Button text** to say “**Rock**”.

To do this expand the button in the Hierarchy to show it’s **Text** object.



In the Inspector you should now be able to edit the text as well as format, font and size.



Rotation	X 0	Y 0	Z 0
Scale	X 1	Y 1	Z 1

Canvas Renderer

Text (Script)

Text

Button

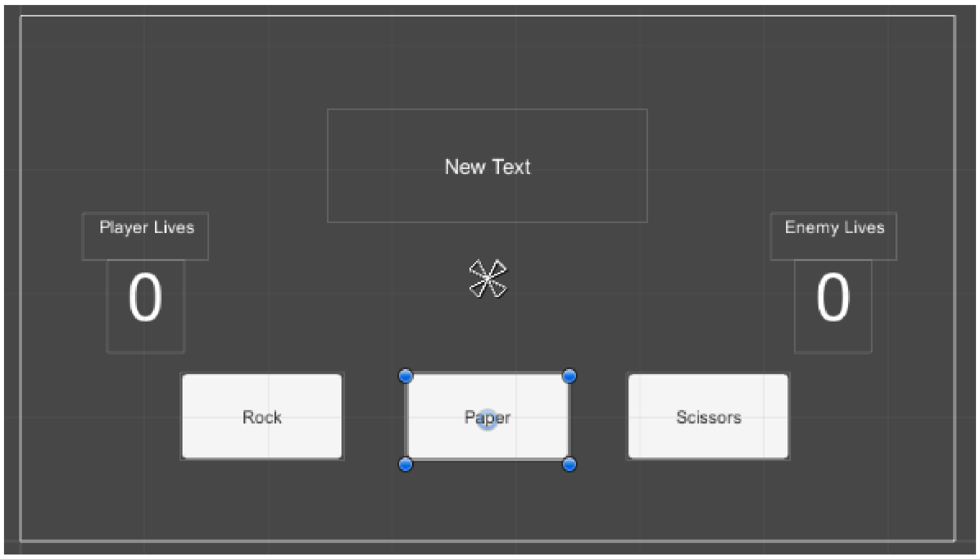
Character

Font	Arial
Font Style	Normal
Font Size	14
Line Spacing	1
Rich Text	✓

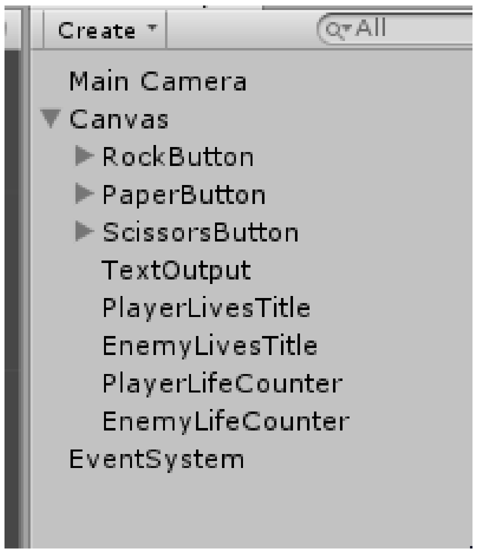
Paragraph

Alignment	
Horizontal Overflow	Wrap
Vertical Overflow	Truncate
Best Fit	
Color	■
Material	None (Material)

Using what you have learnt so far, setup your scene in the way shown below using three **buttons** and several **text** objects.



Note that all the items should be named correctly in the Hierarchy.

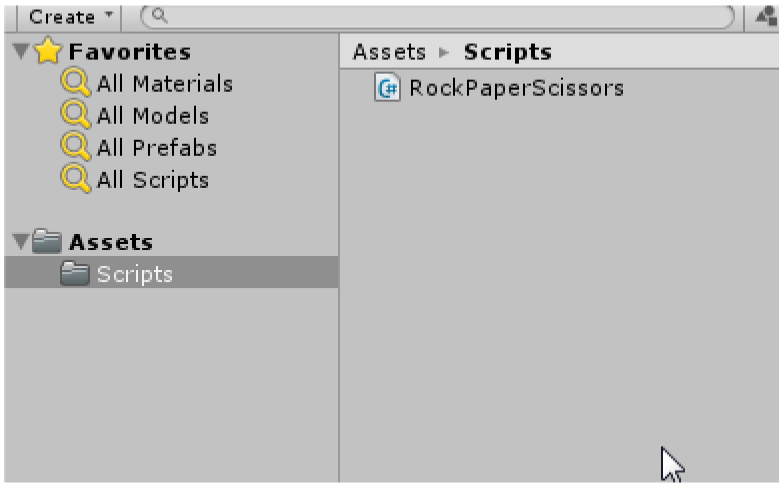


```
Main Camera
▼ Canvas
  ▶ RockButton
  ▶ PaperButton
  ▶ ScissorsButton
    TextOutput
    PlayerLivesTitle
    EnemyLivesTitle
    PlayerLifeCounter
    EnemyLifeCounter
EventSystem
```

Now that we have our scene set up, we need to program the functionality!

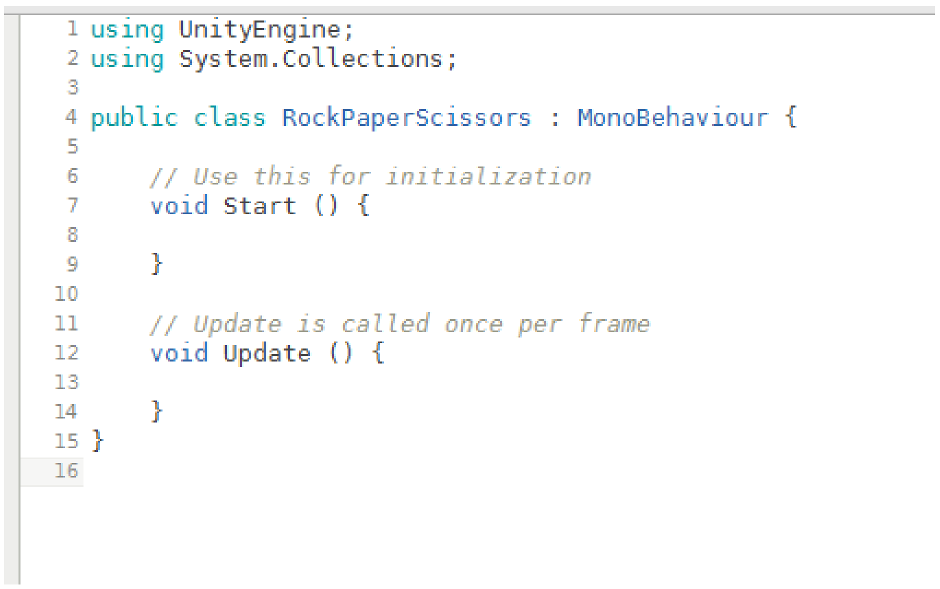
Programming the UI

Create a new folder in your Project called **Scripts** and inside that folder create a new **C# Script** called **RockPaperScissor**s …



Double click on this script to open it in MonoDevelop…

This is the default script setup you get with any new script you create…



```
using UnityEngine;
using System.Collections;

public class RockPaperScissors : MonoBehaviour {

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {

    }
}

```

It has three main areas…

Using: At the very top of the script, you can tell the compiler what ***code libraries*** you are going to be using. Most of the time the two defaults will be enough, but there will be others… for example, in this exercise we will be using **UnityEngine.UI** which has all the commands that deal with the User interface system.

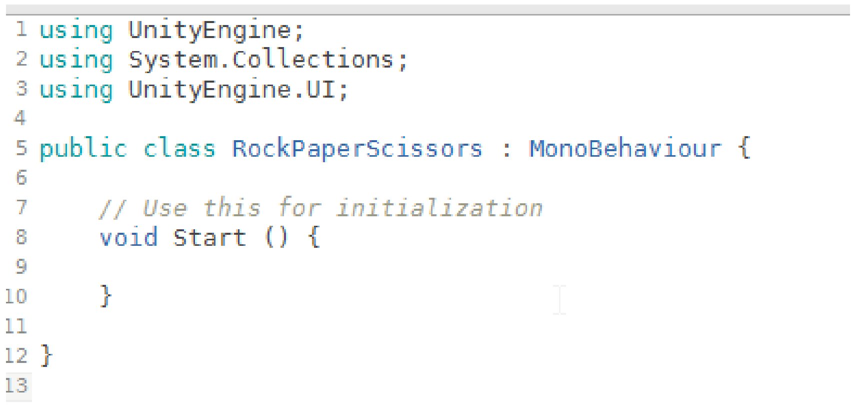
Start: This function gets called once, when the script first gets created. It is used to initialise values and create things needed for the script to work.

Update: This ***update*** section gets run once for every frame of your game. For this project we do not need it as we are only updating our game when the player presses a button.

Add the line at the top…
using UnityEngine.UI;

And remove the ***Update*** section entirely including the { }.

You should be left with the following script…



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

public class RockPaperScissors : MonoBehaviour {

    // Use this for initialization
    void Start () {

    }

}

```

Note: *We will later be attaching this script to the Camera, where it will act as a sort of Game Controller. Don't worry about this for now.*

Declaring some variables

Now we need to set up our **variables**.
Variable are values that can be referenced in this script, or elsewhere by the game.

There are two main ways to declare a variable, **Public** or **Private**.

A **Private** variable can only be accessed within this script only.

A **Public** variable can be accessed from other scripts. It also exposes the variable in the Unity Editor so we can assign object references to them (we will do this later).

Note: Public vs Private

*So why don't we make them all **Public**? Well there are a lot of reasons. But think about a game that had a thousand scripts and one hundred of them have a variable called '**count**' at some point you may get lost on which count variable belongs to which script.*

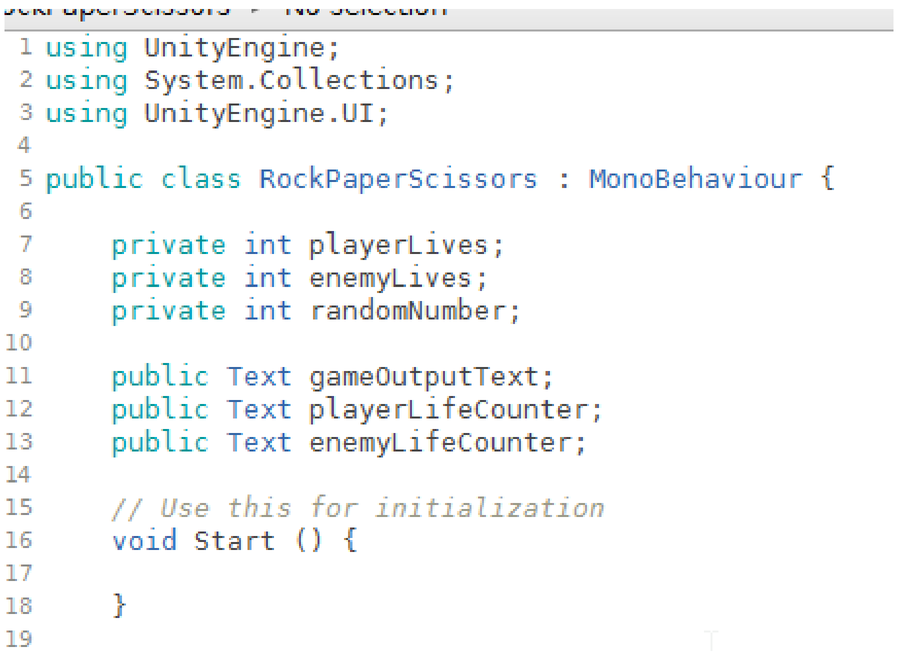
As a rule you only make variables public if you want them to be accessible in the editor, or accessible by other scripts in your game.

The variables we will be using are…

- Player Lives (defaulted to 3)
- Enemy Lives (defaulted to 3)
- A Random Number
- A reference to the UI textbox
- A reference to the player UI life counter
- A reference to the enemy UI life counter

We declare our variables at the top, above the Start function.

All three of the first variables are **integers** (whole numbers) so we declare them as **int** (note the use of camel case (lower case first letter, Uppercase first letter of every other word).



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

public class RockPaperScissors : MonoBehaviour {

    private int playerLives;
    private int enemyLives;
    private int randomNumber;

    public Text gameOutputText;
    public Text playerLifeCounter;
    public Text enemyLifeCounter;

    // Use this for initialization
    void Start () {

    }

```

The next three are reference objects of the type **Text**.

We are telling the computer that we will be using three Text objects in our code, and we are going to be calling them **gameOutputText, playerLifeCounter,** and **enemyLifeCounter.**

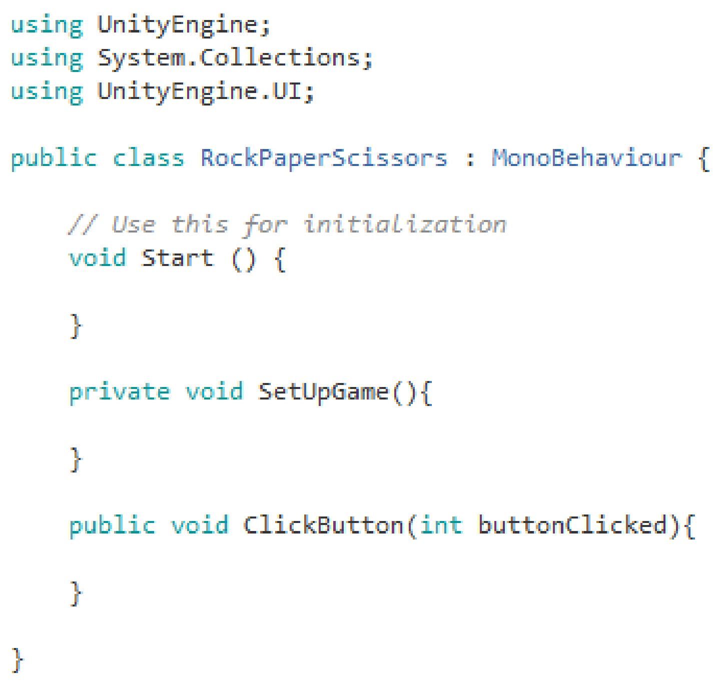
We will link them to the **Text** objects in our scene later.

Writing some functions

Now we need a function to set up our game, and a function to do something when you click a button…

> *Functions can also be made* ***private*** *or* ***public****. A public function (like a variable) can be accessed outside the script while a private function cannot.*

Let's add two functions to do that now…



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

public class RockPaperScissors : MonoBehaviour {

    // Use this for initialization
    void Start () {

    }

    private void SetUpGame(){

    }

    public void ClickButton(int buttonClicked){

    }

}
```

We have made the **SetUpGame** ***private*** as it will only be accessed from other functions within this script. We made the **ClickButton** function ***public*** as it will be accessed by other objects (the game buttons).

> ***Note: Naming conventions for variables and functions***
> *Note the capital letters in the front of the function names, unlike the lowercase letters we use for the variable names, this is a naming convention that helps to tell functions and variables apart.*

The **SetUpGame** function will change the text of a text box using this code…

```
private void SetUpGame(){
    //Change the text of the textbox
    gameOutputText.text = "Choose a move to make: Rock, Paper, Scissors";
}
```

In **SetUpGame**, we also need to initialise the life counter text, so add the two lines below to achieve that…

```
private void SetUpGame(){
    // change the text of the textbox
    gameOutputText.text = "Choose a move to make: Rock, Paper, Scissors?";
    playerLivesCounter.text = playerLives.ToString();
    enemyLivesCounter.text = enemyLives.ToString();
}
```

Now we need to declare our starting lives and call the **SetUpGame** function in the **Start** function…

```
// Use this for initialization
void Start () {
    playerLives = 3;
    enemyLives = 3;
    SetUpGame();
}
```

The ClickButton function

Passing values to a function

For our **ClickButton** function you will notice that in the brackets we have ***(int buttonClicked)***.

What this means is that when this function is called, you have to give it an integer (whole number), this integer gets stored in the variable called **buttonClicked**. All variables made in this way are destroyed once the function has finished.

For this integer we will have each button send a different integer to this function when it is clicked, so the program knows which button you pressed.

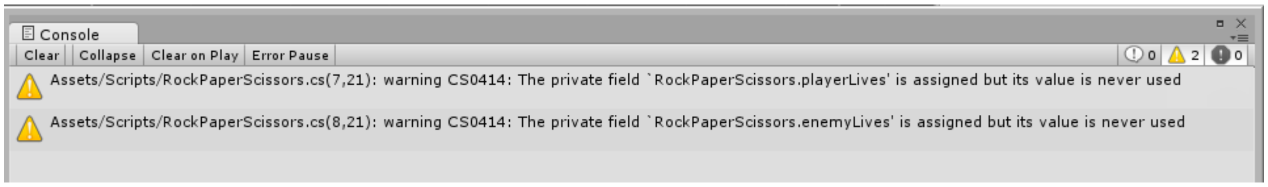
Therefore, we need an ***if statement*** to work out what was pressed and display this in the textbox…

```
public void ClickButton(int buttonClicked){

    if (buttonClicked == 1) {
        gameOutputText.text = "You chose Rock";

    } else if (buttonClicked == 2){
        gameOutputText.text = "You chose Paper";

    } else if (buttonClicked == 3){
        gameOutputText.text = "You chose Scissors";
    }

}
```

Save your file **(Ctrl+S)** and head back to Unity.

Check your console window to see if you have made any errors.
If the Console window is not visible in Unity, you can go to the menu **Window > Console**.

So far we should only have two warnings telling us we are declaring variables and not using them…



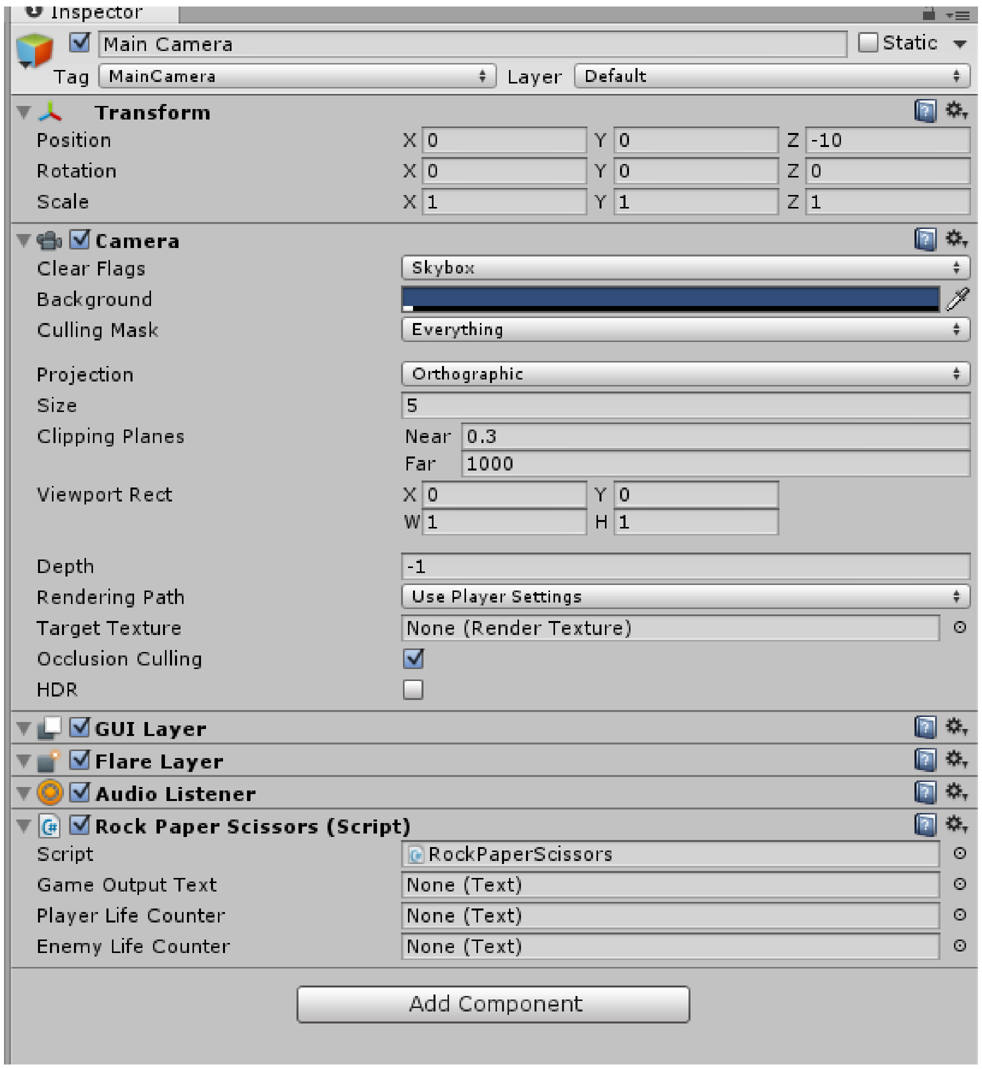
```
Assets/Scripts/RockPaperScissors.cs(7,21): warning CS0414: The private field `RockPaperScissors.playerLives' is assigned but its value is never used
Assets/Scripts/RockPaperScissors.cs(8,21): warning CS0414: The private field `RockPaperScissors.enemyLives' is assigned but its value is never used
```

Attaching the script to an object

For Unity to use this script, we need to attach it to something in the scene. In this game we will attach it to the **Main Camera**. This will act as a sort of Game Controller object for our simple game.

To do this just drag the **RockPaperScissors** script from the ***Project window*** to the **Main Camera** object in the ***Hierarchy window***…

If you have done this correctly your camera should look like this when you click on it…



Main Camera ☐ Static
Tag MainCamera Layer Default

Transform

Position	X 0	Y 0	Z -10
Rotation	X 0	Y 0	Z 0
Scale	X 1	Y 1	Z 1

Camera

Clear Flags	Skybox
Background	
Culling Mask	Everything
Projection	Orthographic
Size	5
Clipping Planes	Near 0.3
	Far 1000
Viewport Rect	X 0 Y 0
	W 1 H 1
Depth	-1
Rendering Path	Use Player Settings
Target Texture	None (Render Texture)
Occlusion Culling	✓
HDR	☐

GUI Layer

Flare Layer

Audio Listener

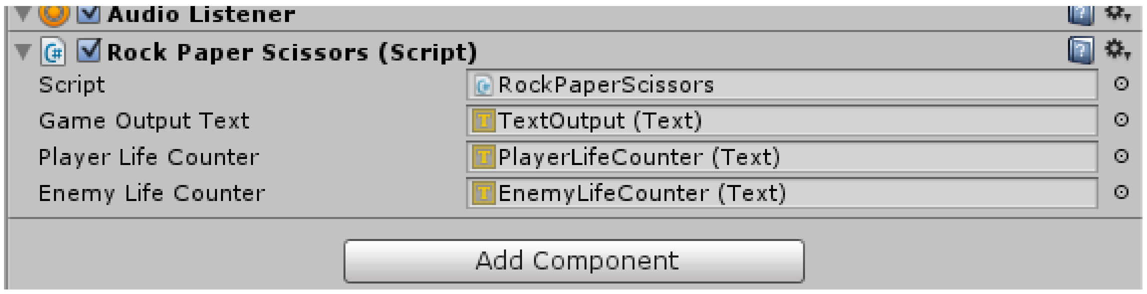
Rock Paper Scissors (Script)

Script	RockPaperScissors
Game Output Text	None (Text)
Player Life Counter	None (Text)
Enemy Life Counter	None (Text)

Notice that all three **variables** that we have made public in our script can be seen in the **Inspector** and that they are currently empty.

This allows us to drag **GameObjects** from the game into those slots to assign them to these variables. We are going to do now…

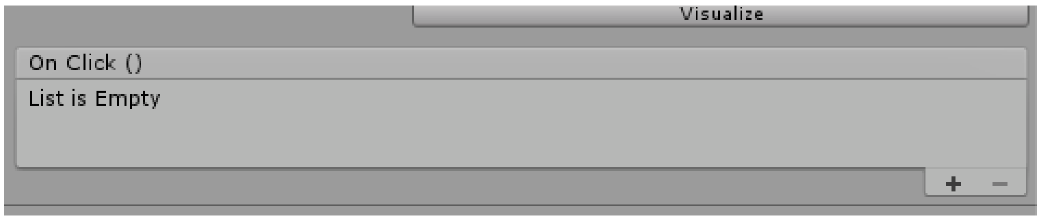
Drag each of the **PlayerLifeCounter**, **EnemyLifeCounter**, and **TextOutput** from the ***hierarchy window*** to each of the fields in the script on the camera…



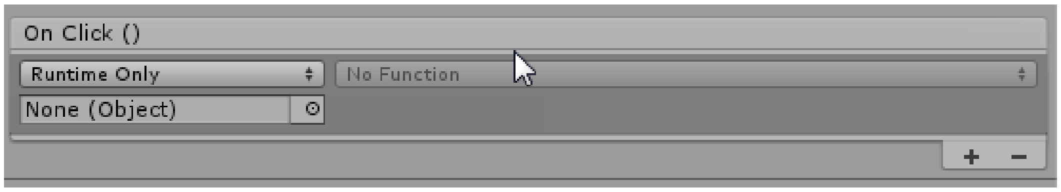
Script: RockPaperScissors
Game Output Text: TextOutput (Text)
Player Life Counter: PlayerLifeCounter (Text)
Enemy Life Counter: EnemyLifeCounter (Text)

Using the Events System

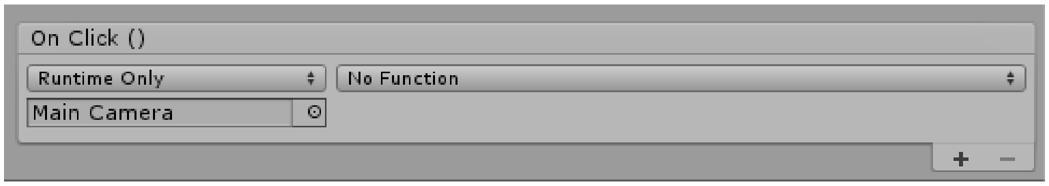
To set up the buttons we need to use its event system.
Select one of your buttons and find its **On Click ()** Event system in the Inspector view…



This allows us to tell the game what to do when this button is pressed.
Click on the **+** to add a blank event…

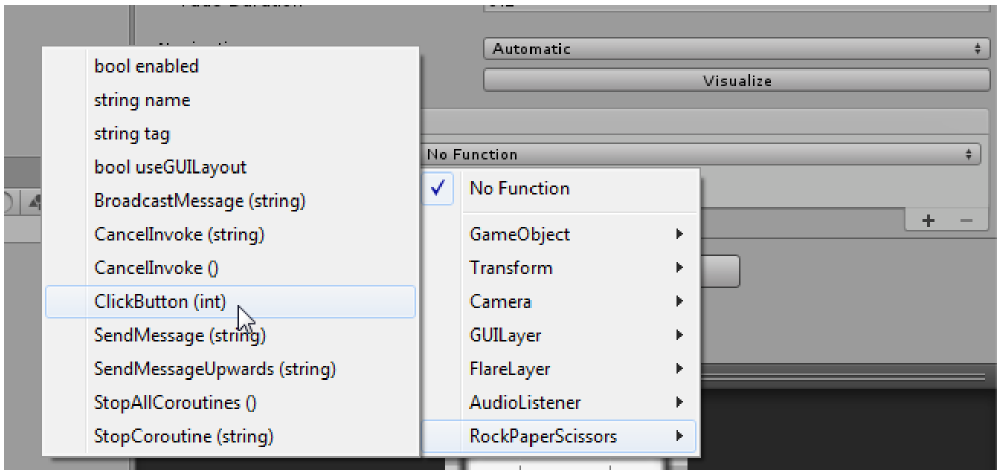


Our aim is to send a message to our script that is currently living on the **Game Camera**.
So drag the **Game Camera** from the Hierarchy view to the empty field in the Button Event…



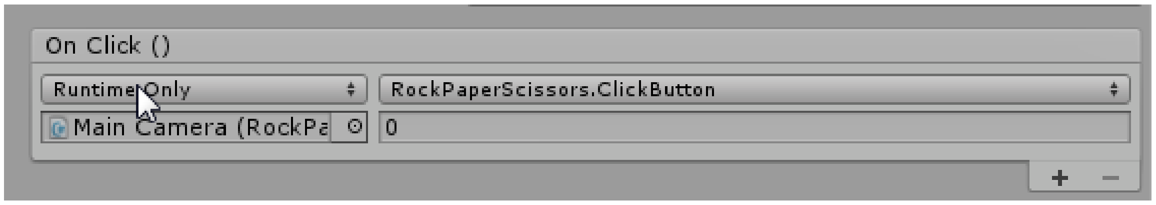
Now we have to say what to do with this object.

Change ***No Function*** to our script **RockPaperScissors**, and then our function **ClickButton(int)**.



*Note that the **int** that is listed here is telling us that we told the script to request an **integer** when this function is called.*

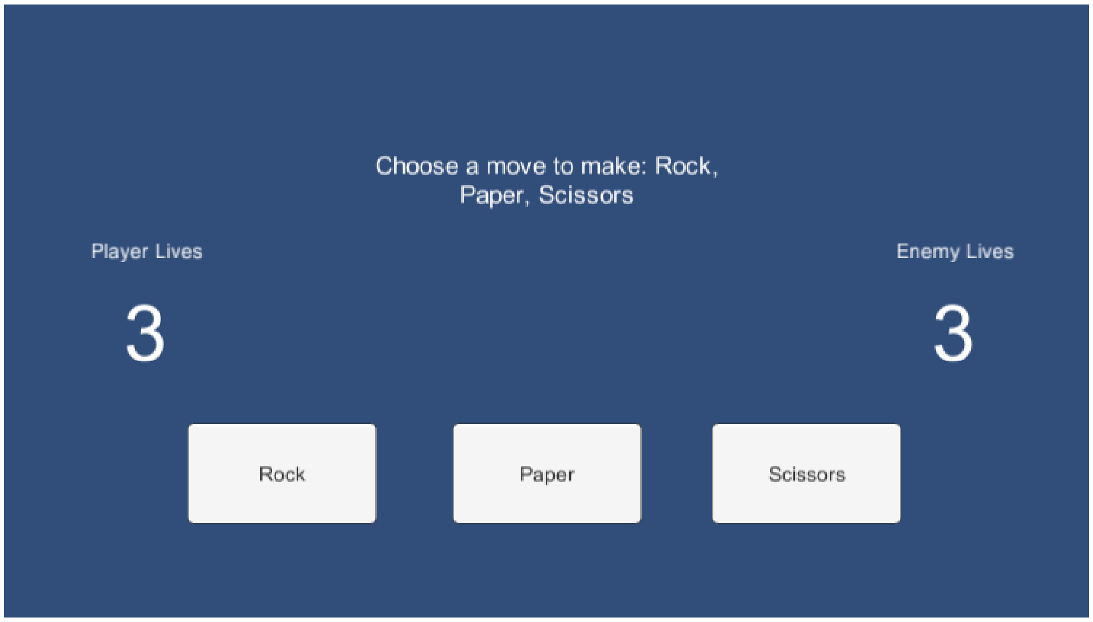
Once selected, you will see a new field appear, this is the integer that will be passed to the script…



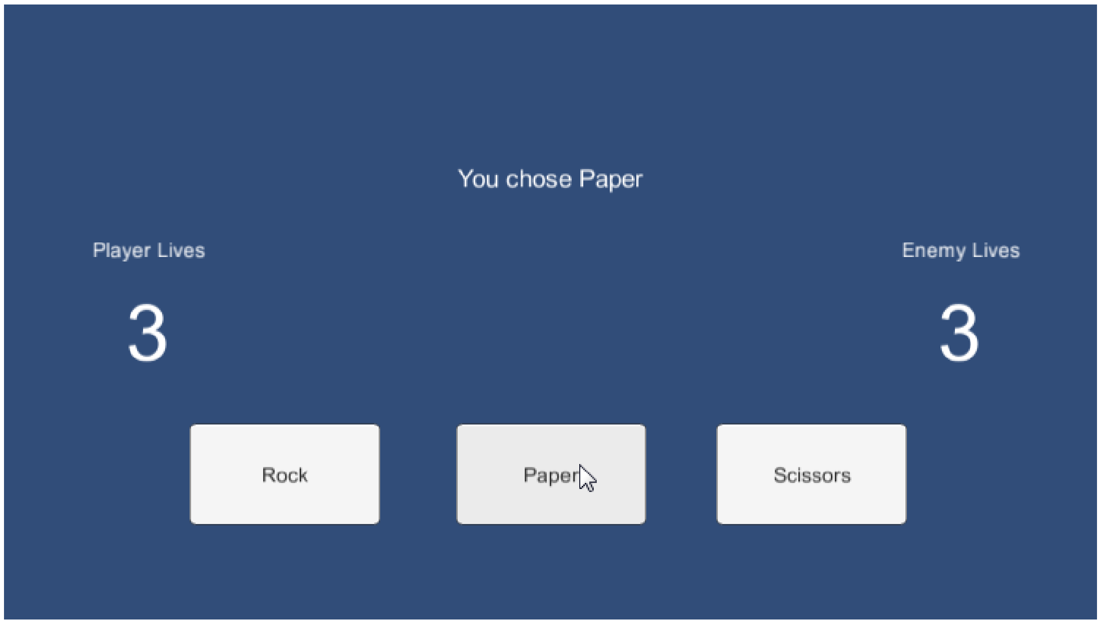
Set this up for each of the three buttons and set the integer to 1 for the Rock button, 2 for Paper and 3 for Scissors.

We should be able to press play and see if we have set up the game correctly.

You should see the life counters default to 3 each and the output text change…

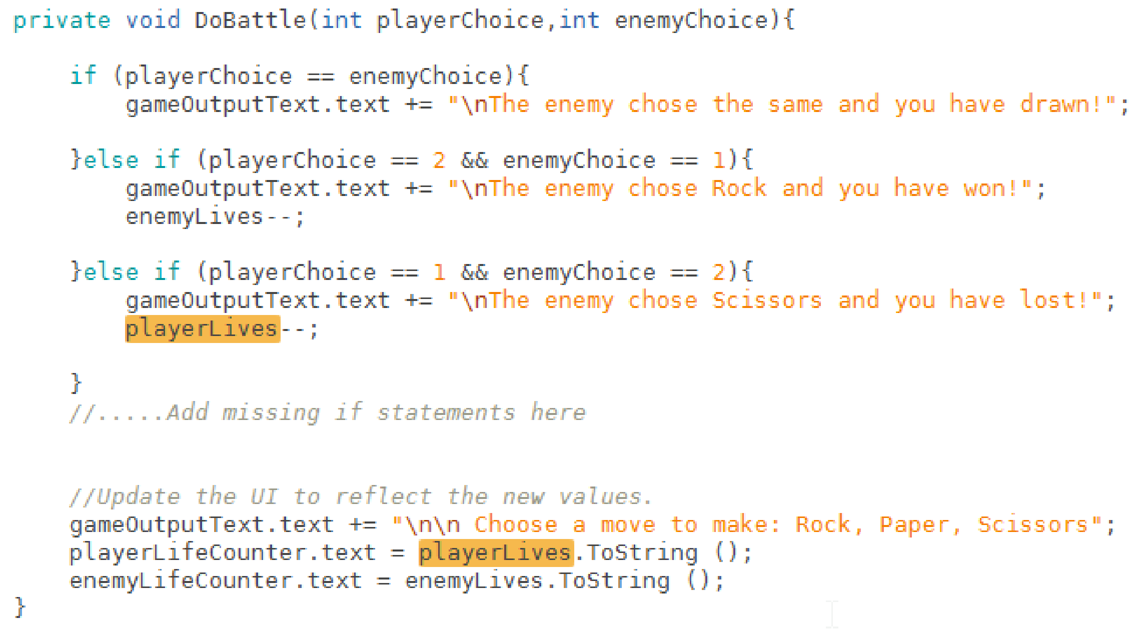


When you click each of the buttons the text box will change to tell you what button you pressed…



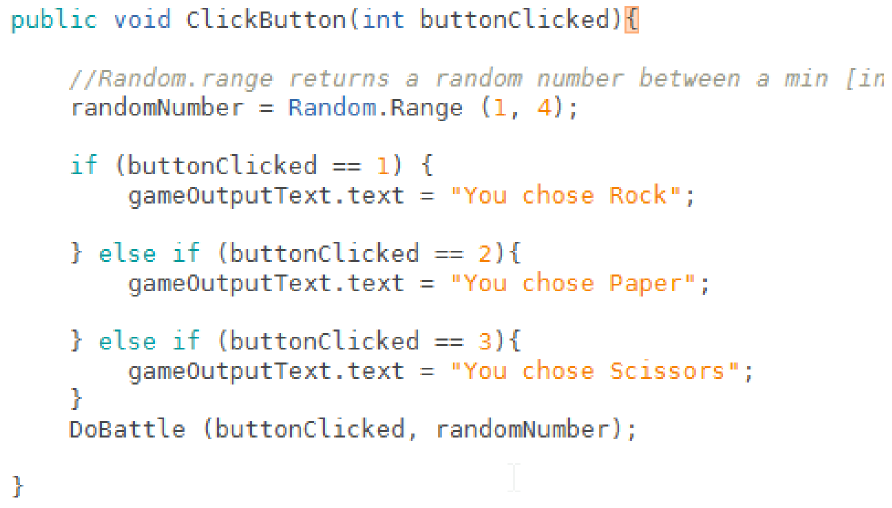
Now we have to add code to make the computer choose a number, and then compare your number with theirs. We need to write a new function to do the comparing and then adjust the life counters accordingly.

Here we have done a couple of comparisons for you. See if you can do the rest…



```
private void DoBattle(int playerChoice,int enemyChoice){

    if (playerChoice == enemyChoice){
        gameOutputText.text += "\nThe enemy chose the same and you have drawn!";

    }else if (playerChoice == 2 && enemyChoice == 1){
        gameOutputText.text += "\nThe enemy chose Rock and you have won!";
        enemyLives--;

    }else if (playerChoice == 1 && enemyChoice == 2){
        gameOutputText.text += "\nThe enemy chose Scissors and you have lost!";
        playerLives--;

    }
    //.....Add missing if statements here


    //Update the UI to reflect the new values.
    gameOutputText.text += "\n\n Choose a move to make: Rock, Paper, Scissors";
    playerLifeCounter.text = playerLives.ToString ();
    enemyLifeCounter.text = enemyLives.ToString ();
}
```

The last things we need to do is to add a **randomNumber** generator, and then call the **DoBattle** function from the **ClickButton** function…



```
public void ClickButton(int buttonClicked){

    //Random.range returns a random number between a min [in
    randomNumber = Random.Range (1, 4);

    if (buttonClicked == 1) {
        gameOutputText.text = "You chose Rock";

    } else if (buttonClicked == 2){
        gameOutputText.text = "You chose Paper";

    } else if (buttonClicked == 3){
        gameOutputText.text = "You chose Scissors";
    }
    DoBattle (buttonClicked, randomNumber);

}
```

You can see we are passing the **buttonClicked** and the **randomNumber** to this function for it to use.

And you're done!

Note: *You can note that if you resize the Game window, it will resize the shape and size of the canvas! You can select the Canvas object in the Hierarchy window and in the Canvas Scaler component, you can set the* ***Screen Match Mode*** *to "Match Width or Height" to address that issue. You can experiment more with features like that to suit your UI needs.*