

Lab 1 – Objects, States and Behaviours

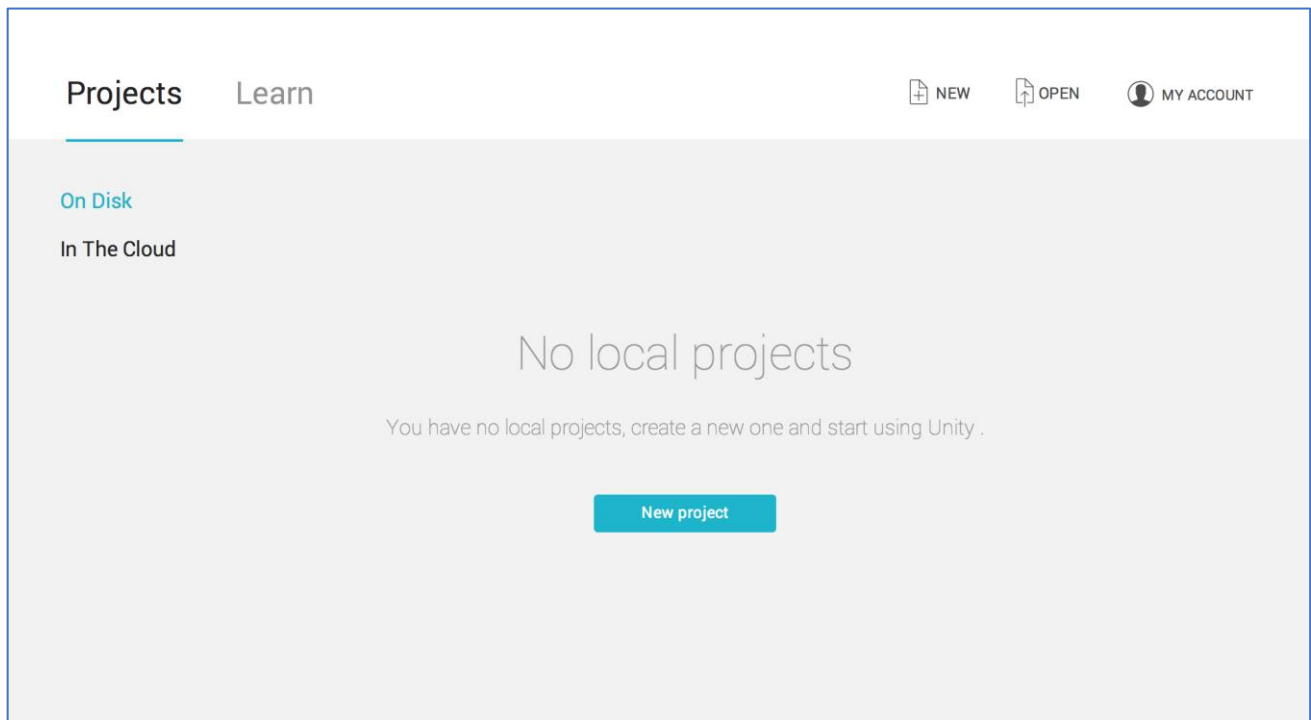
(updated 01/10/2018)

In this lab:

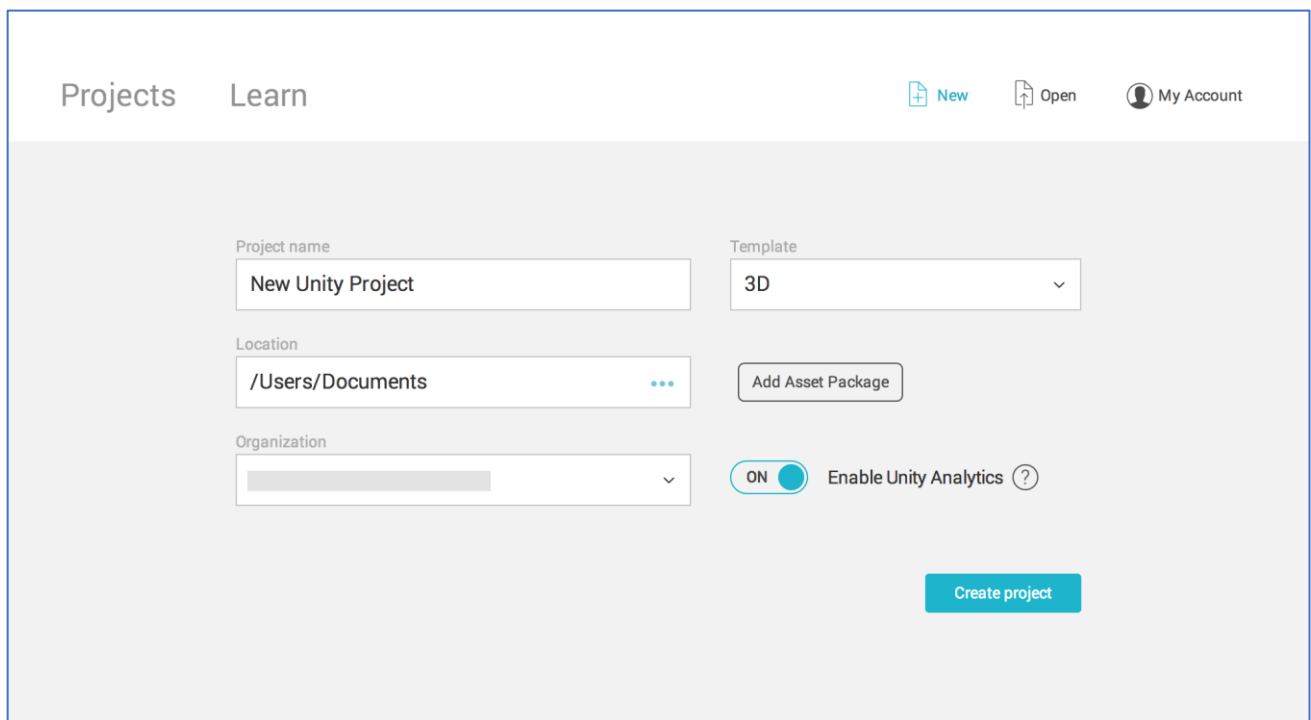
Task 1: Getting started.....	2
Task 2: Importing assets.....	3
Task 3: Adding game objects.....	5
Task 4: Saving the scene.....	10
Task 5: Creating scripts	11
Task 6: Creating functions	13
Task 7: Attaching scripts.....	14
Task 8: Creating variables	15
Task 9: Changing variable values	17
Task 10: Setting variable visibility	18
End of Lab	19

Task 1: Getting started

Look for Unity on the desktop or start menu and run it. If asked to sign in, follow the on-screen instructions to create an account and sign in.



Once signed in, create a new project by clicking **New**.



Enter a suitable name for the project, such as **Lab 1**, and choose where to save the project by clicking the three dots in the location field. Choose the **2D** template (choosing the correct template is important as it can be tricky to change it later) and click **Create project**.

If Unity prompts you to install an update, click **Skip new version**.

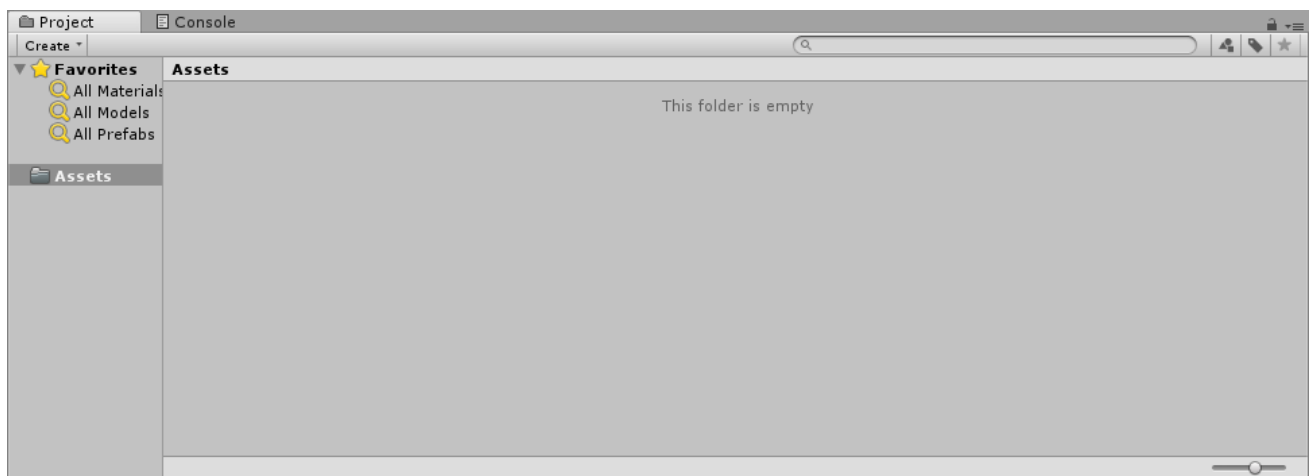
Task 2: Importing assets

Download the **Lab 1 - Assets.zip** file from GCU Learn. Open it up and copy the image files to somewhere convenient, such as the desktop (it is important to open the zip file and copy the images out of it, because Unity can't import images directly from a zip file).

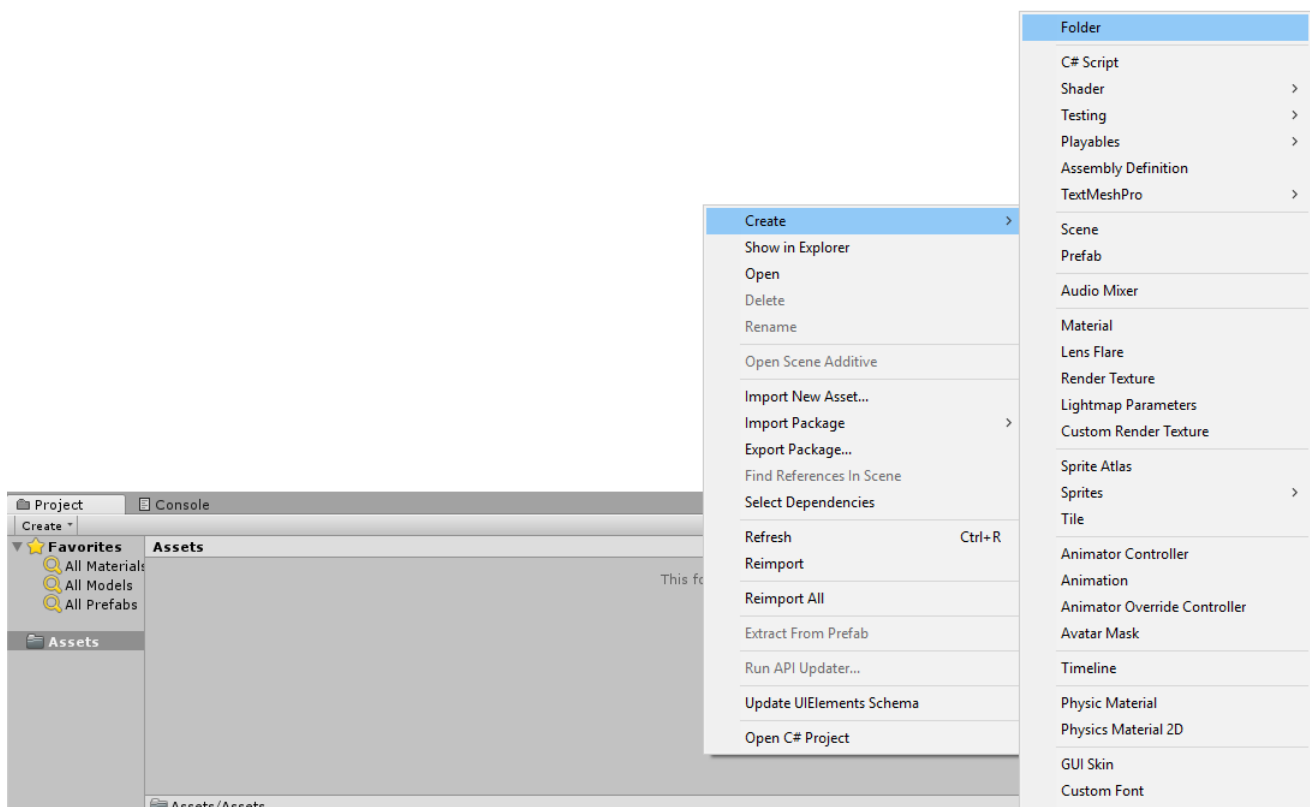
In Unity, an asset is any item that can be used in your project. An asset may come from a file created outside of Unity, such as a 3D model, an audio file, an image, or any of the other types of file that Unity supports. There are also some types of assets which can be created within Unity.

<https://docs.unity3d.com/Manual/AssetWorkflow.html>

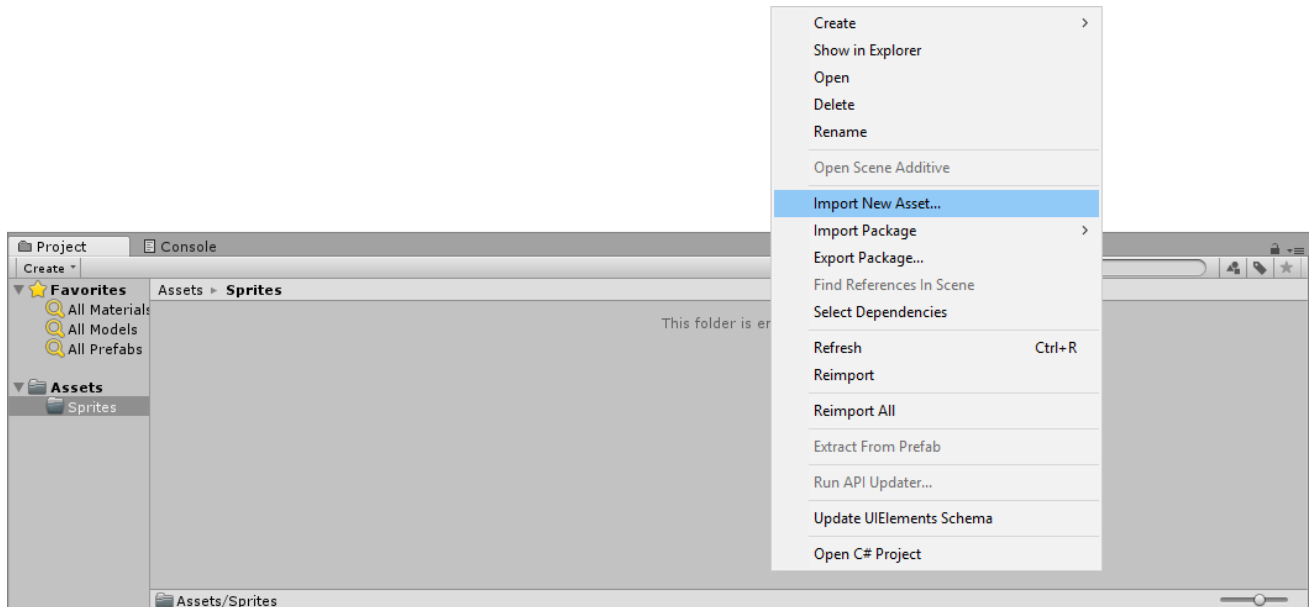
Look at the **Project Panel** at the bottom of the Unity window. It displays all your assets, available to use in your project. It is empty for now, but when you create or import assets into your project, they will appear here.



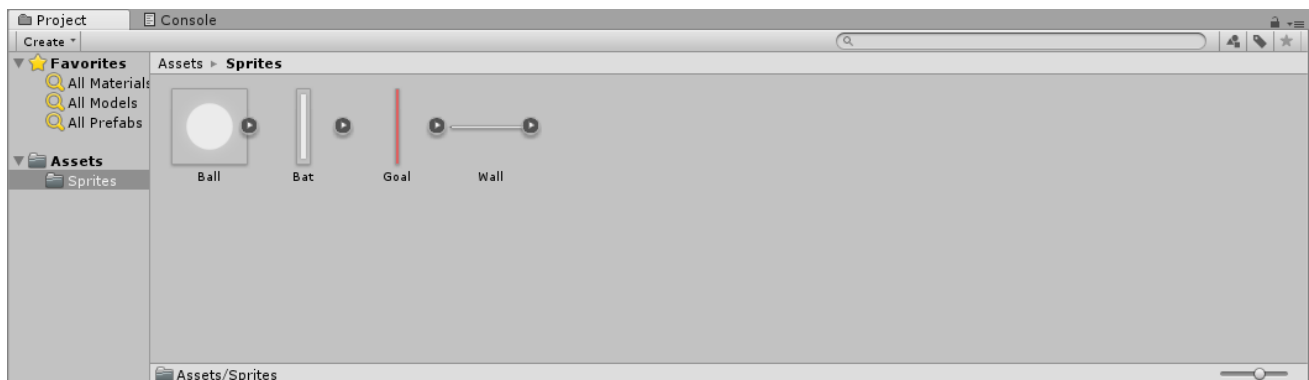
Right-click in the large empty space in the project panel and navigate to **Create** and then **Folder**. Name the new folder **Sprites** (images are generally referred to as sprites in 2D projects).



Double-click on the new Sprites folder to open it, then right-click in the empty space and select **Import New Asset**.



Select all the images you downloaded earlier and click **Import** to copy them into your project.



Task 3: Adding game objects

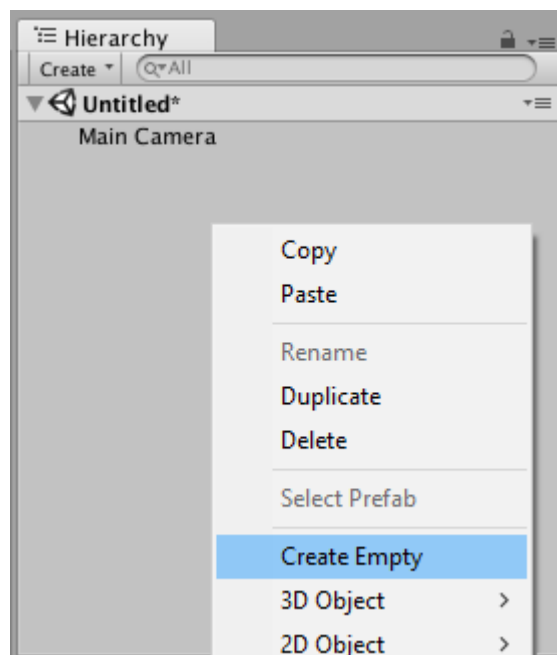
Unity projects are made up of **Scenes**. You will usually create a different scene for each level or environment in your game. Each scene is made up of many **Game Objects**. You can create a game object to represent literally anything in your game – a character, a light, a tree, a sound, or whatever else you'd like it to be.

<https://docs.unity3d.com/Manual/CreatingScenes.html>

Look at the **Hierarchy Panel** on the left of the Unity window. It contains a list of every game object in the current scene. For now, there is only a **Main Camera** (the camera determines which part of the scene the player can see, so every scene must have at least one camera).



Add a new game object to the scene by right-clicking in the large empty space in the hierarchy panel and selecting **Create Empty**.

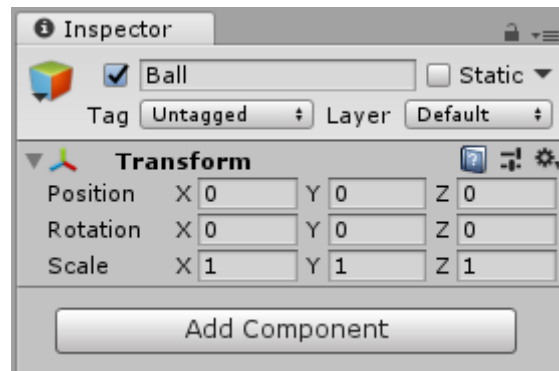


Change the name of the new game object to something like **Ball** by right-clicking on it in the hierarchy and selecting **Rename**.

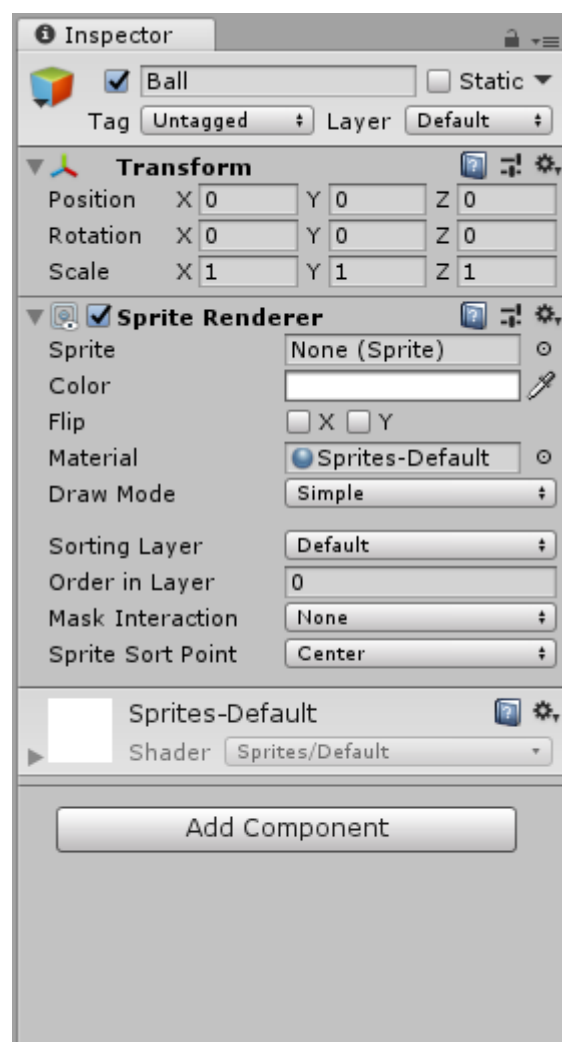
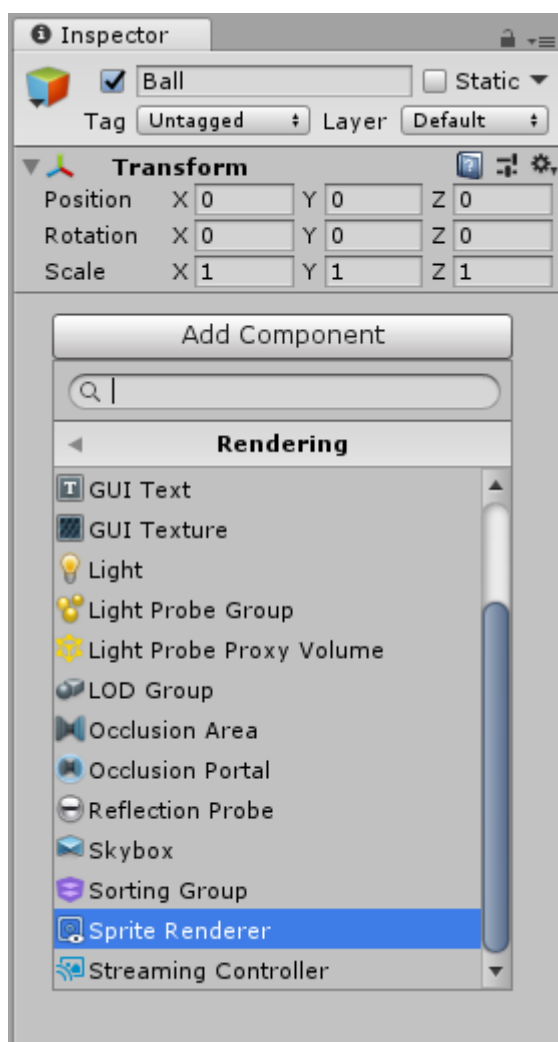
A game object can't do anything on its own, it is simply a container for **Components**, which allow you to add functionality to your game object. Depending on what kind of object you want to create, you add different combinations of components to a game object. Unity has many built-in component types, and you can also make your own components using scripts.

<https://docs.unity3d.com/Manual/GameObjects.html>

Select the Ball game object by clicking on it in the hierarchy. Then look at the **Inspector Panel** on the right. It allows you to modify and add components to your game object. Notice that by default it already has a **Transform** component (the transform component has several **variables** which store the position, rotation and scale of the object).



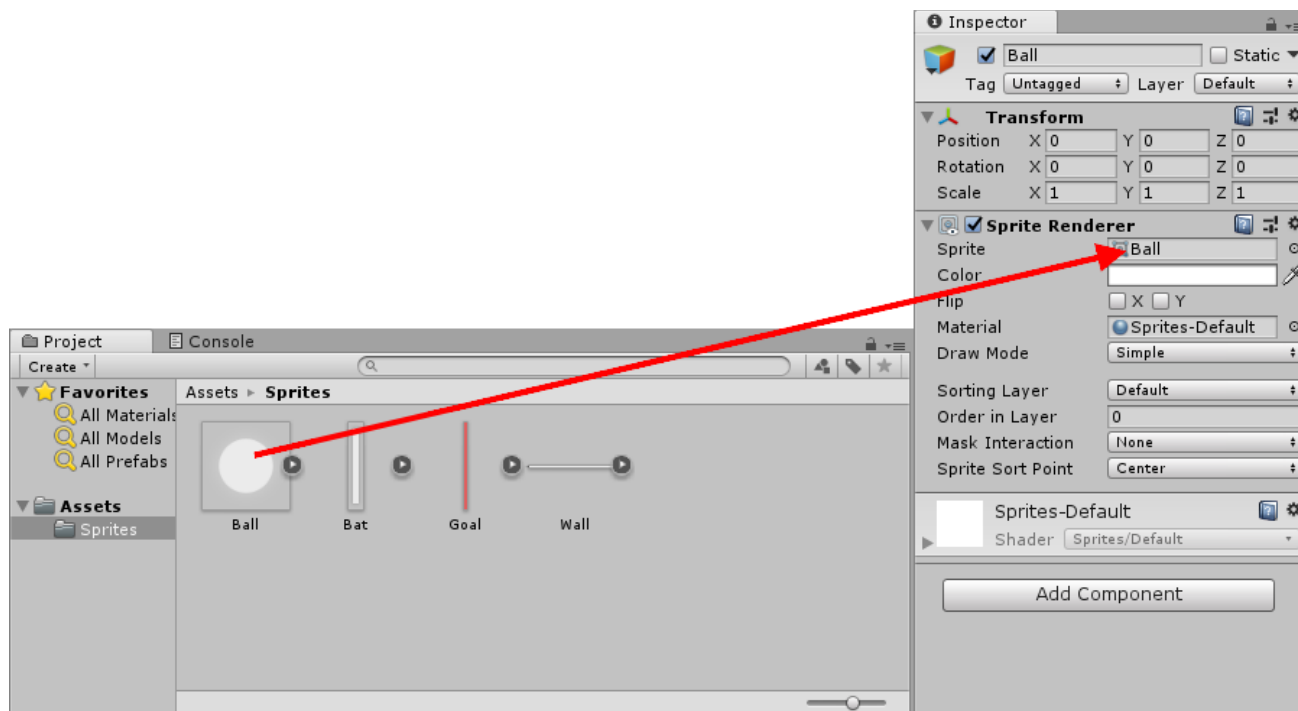
With the Ball game object selected in the hierarchy, click the **Add Component** button in the inspector and select the **Rendering** category then the **Sprite Renderer** component (the sprite renderer component allows your game object to display an image on the screen in your game).



The sprite renderer component has several **variables**, which are values you can change, that affect how the image is displayed. For example, the **Sprite** variable lets you set which image the sprite renderer will display.

<https://docs.unity3d.com/Manual/class-SpriteRenderer.html>

With the Ball game object selected, click and drag the **Ball** image from the project panel onto the 'Sprite' field in the inspector as shown in the image below:

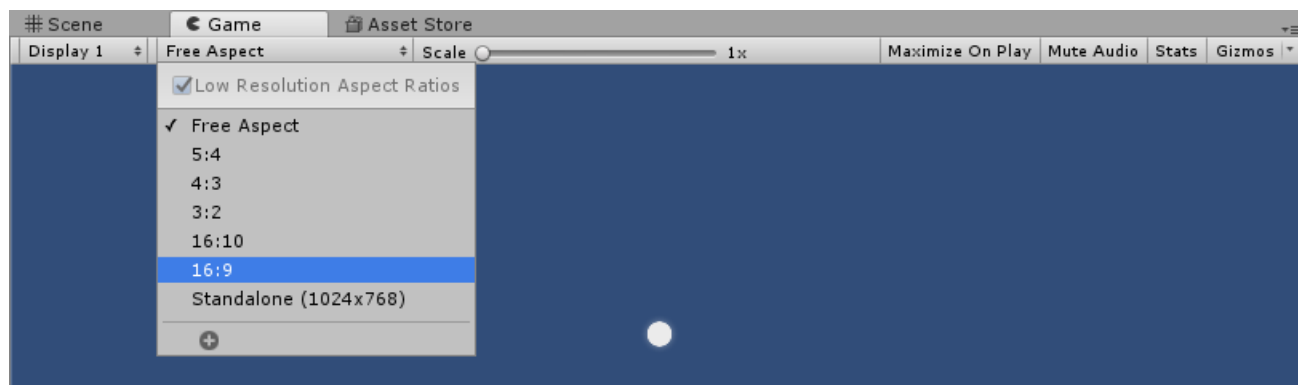


Look at the **Scene View** in the centre of the Unity window. It allows you to visually navigate and edit your scene. The tabs at the top allow you to switch between the scene view and the **Game View**. The game view shows what the scene will look like when the game is run. Switch to game view now and notice that the ball sprite is being rendered in the middle of the view.

Before positioning objects in the scene, it is usually a good idea to set a target aspect ratio, which affects the camera and determines how much of the scene will be visible. Most desktop monitors have a 16:9 aspect ratio, which makes it a good target aspect ratio to use.

<https://docs.unity3d.com/Manual/GameView.html>

At the top of the game view, click on the second drop-down menu from the left (which by default will probably say **Free Aspect**) and set it to **16:9**.



Switch back to the scene view and notice that the light grey box, which shows the limits of what the player can see, has changed shape to reflect the new 16:9 aspect ratio (you may need to zoom out with the scroll wheel to see the light grey box).

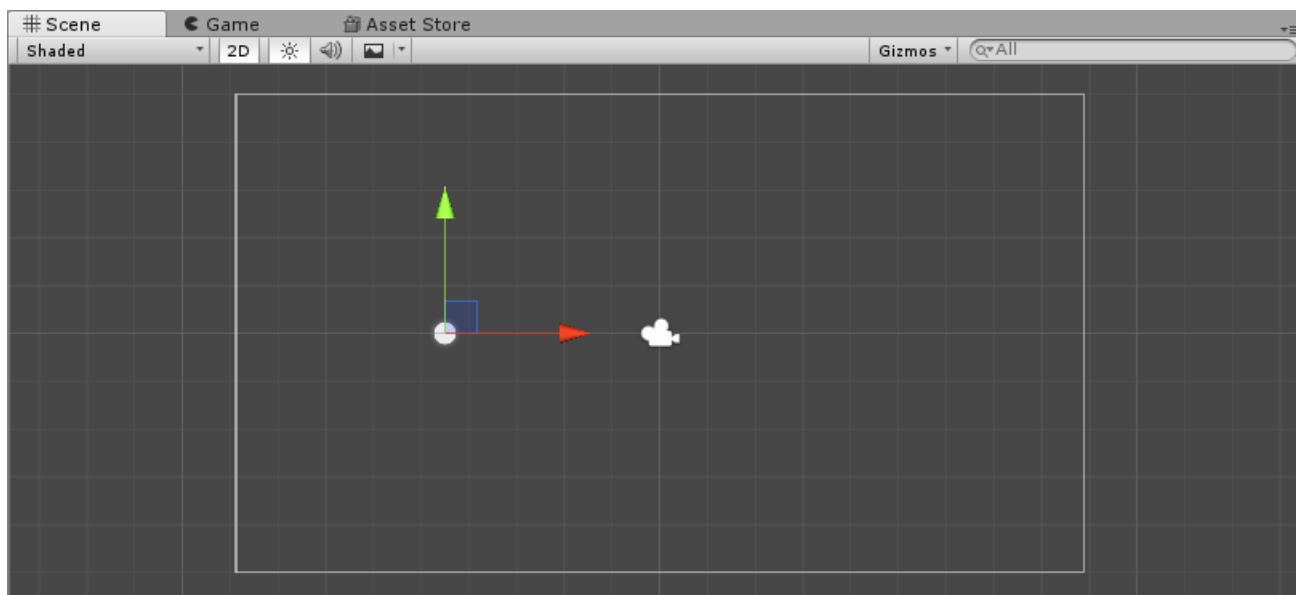
Look at the **Toolbar** at the top of the window. On the left are the basic tools for manipulating the scene view and the objects within it. Click on the **Move Tool**, the second button from the left.



With a game object selected, it can be moved up/down by clicking and dragging on the green arrow or left/right by clicking and dragging on the red arrow. Alternatively, it can be moved around freely by clicking and dragging on the small blue square.

<https://docs.unity3d.com/Manual/PositioningGameObjects.html>

With the Ball game object selected, use the move tool to position the ball in the middle of the left half of the visible scene by clicking and dragging on the red arrow.



TO DO:

You should repeat the steps above (from the start of **Task 3**) to add the following new game objects to the scene:

- Left Bat (use the Bat sprite)
- Right Bat (use the Bat sprite)
- Left Goal (use the Goal sprite)
- Right Goal (use the Goal sprite)
- Top Wall (use the Wall sprite)
- Bottom Wall (use the Wall sprite)

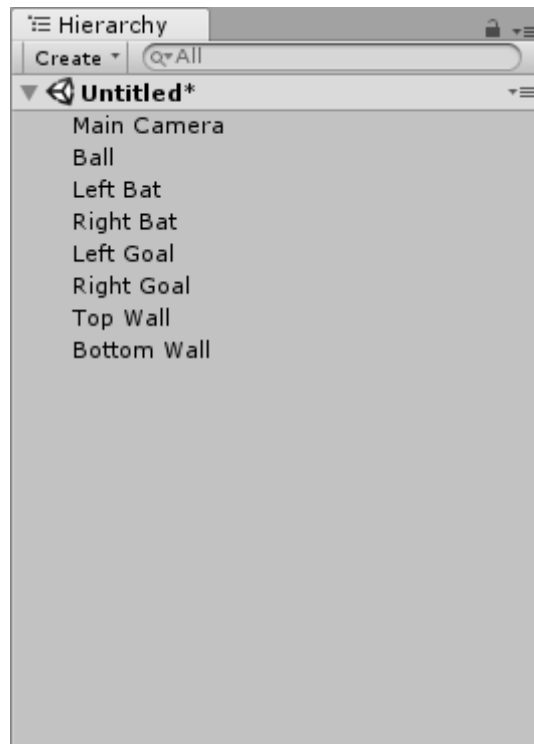
Tip: Instead of creating a new empty game object by right-clicking in the hierarchy, selecting **Create Empty** and then adding a sprite renderer component afterwards, you can instead right-click in the hierarchy and choose **2D Object** then **Sprite** to create a new game object with a sprite renderer already added.

Once that's done, arrange the game objects in the scene to look something like this:



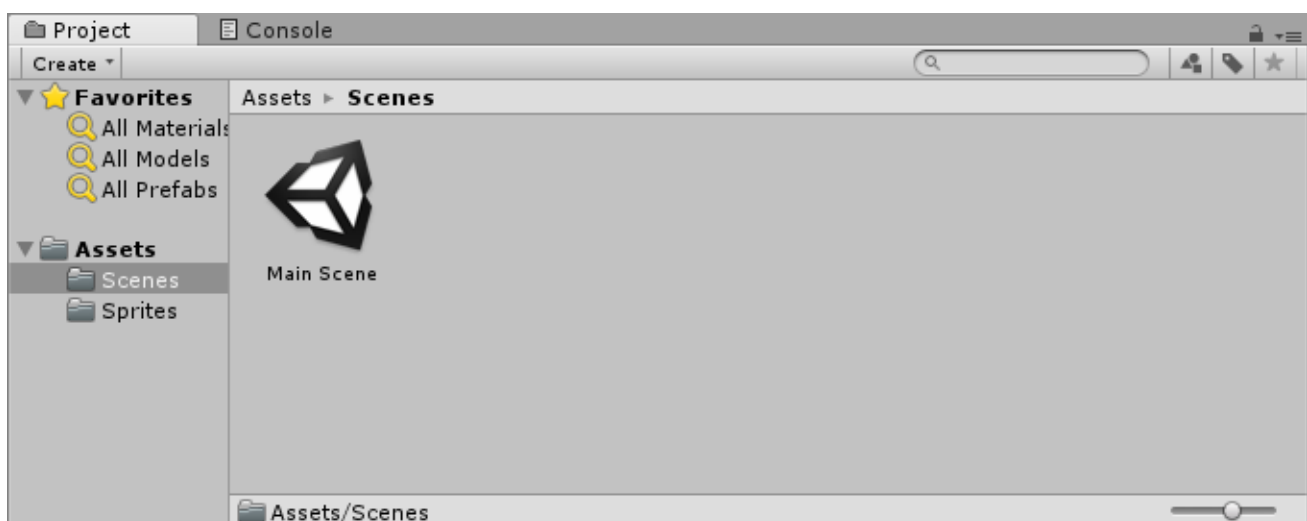
Task 4: Saving the scene

Look at the top of the hierarchy panel. Above the game objects is the name of the scene. For now, it says **Untitled** because the scene has not yet been saved (some versions of Unity might show the name as **SampleScene** instead).



Scenes in Unity are assets as well. In some versions of Unity, there may already be a folder called 'Scenes' within the main 'Assets' folder in the project panel. If not, then return to the main 'Assets' folder in the project panel and create a new folder alongside 'Sprites' called **Scenes**.

To save the scene, open the application menu at the very top of the Unity window, select **File** and **Save Scene As**. When asked where to save the scene, save it in the Scenes folder with a suitable name such as **Main Scene**. In future, you can simply use the **File, Save Scene** option (or use the keyboard shortcut **Ctrl+S**) to save the scene.

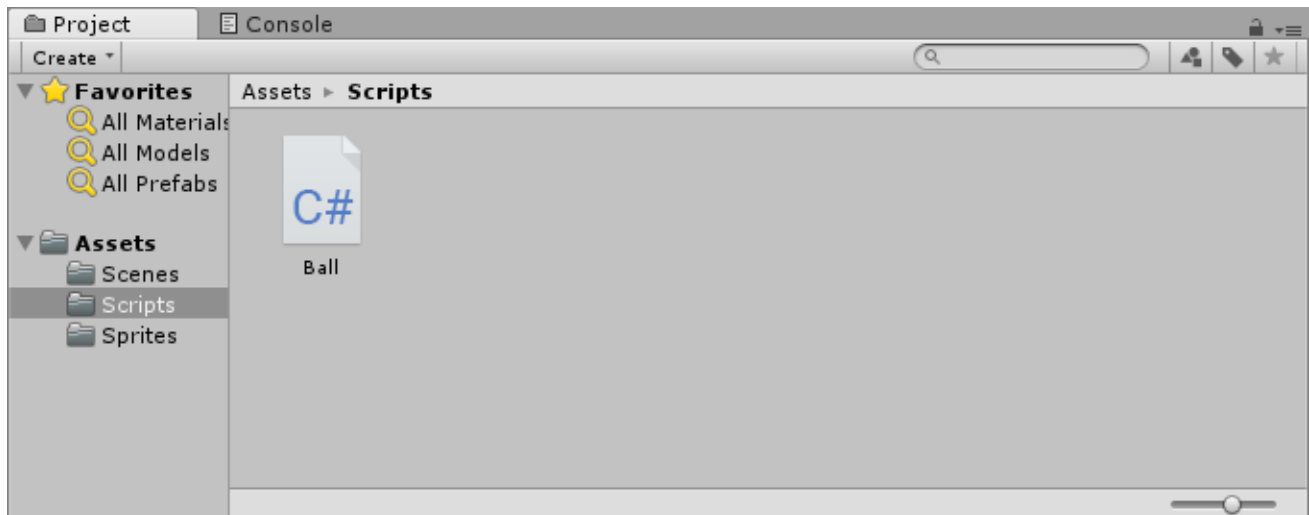


Task 5: Creating scripts

The behaviour of game objects is controlled by the components that are attached to them. Although Unity's built-in components can be very versatile, to implement your own gameplay features you will need to go beyond what they can provide. Unity allows you to create your own custom components using **C# Scripts**.

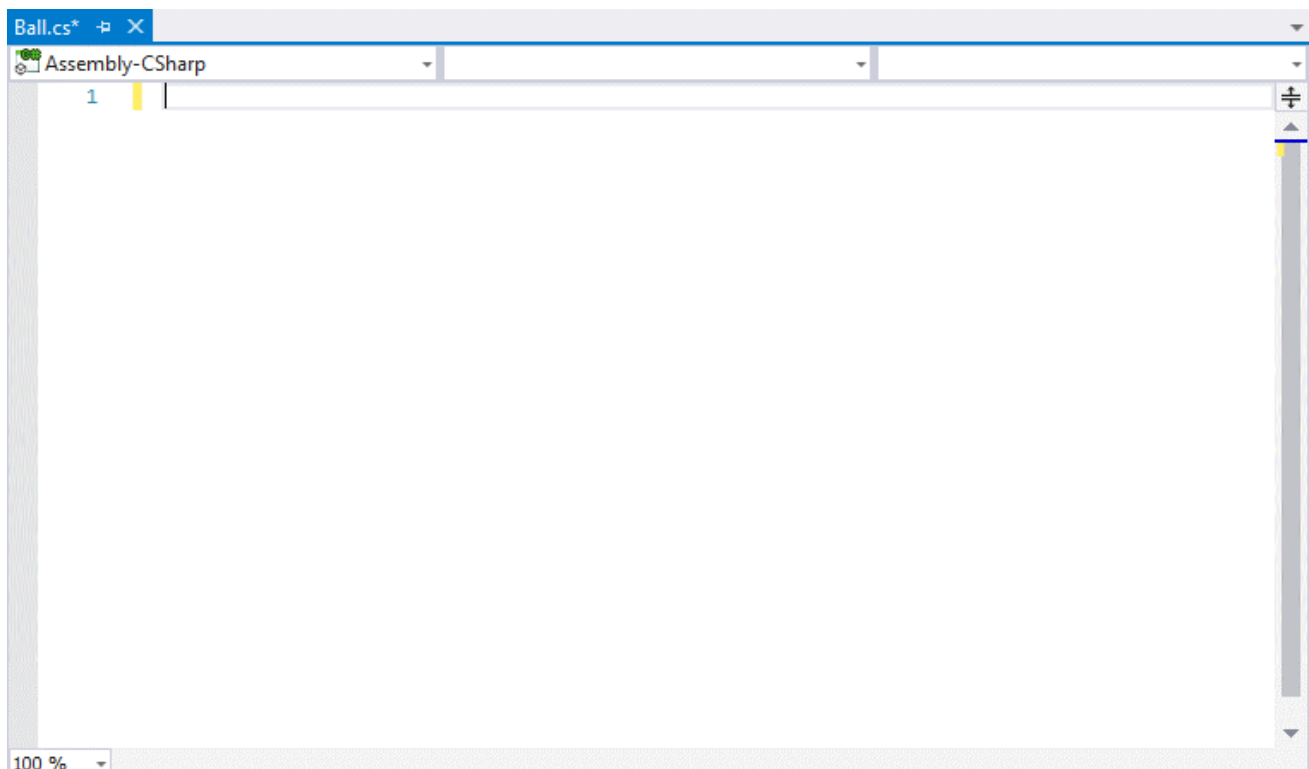
<https://docs.unity3d.com/Manual/CreatingAndUsingScripts.html>

In the project panel, return to the main 'Assets' folder and create a new folder called **Scripts**. Open the Scripts folder, right-click and select **Create** then **C# Script**. Name the new script **Ball**.



Double-click on the Ball script and wait for the code editor to launch. If you are prompted to sign in, you should be able to use your university login details (e.g. <username>@caledonian.ac.uk and your university password).

Once the script has opened in the code editor, remove the default generated code (select all text in the script and delete it).



C# scripts must contain a **class**, which is a collection of related code. To create a class, you use the **class** keyword followed by the **name** of your class, which must exactly match the name of the script file in Unity. This must be followed by a set of **curly brackets**, which mark the start and end of the class.

Create a class with the same name as the script file by typing (or copying and pasting) the following code into the script:

```
class Ball
{
}
```

It is important to note that C# code is case-sensitive, which means that capitalisation is very important. In other words, **ball** is **not** the same as **Ball**. Make sure that you use the same capitalisation as shown in this document.

Your class needs some additional functionality to allow it to be used in Unity. One way to include functionality in a class is to **inherit** the functionality from another class. This topic will be covered in more detail in a future lab, so for now it is fine to just copy and paste the code below without fully understanding it.

To be usable in Unity, the class needs to inherit functionality from a special class which exists somewhere in the Unity Engine code. To do this, change the script to make it look like this:

```
using UnityEngine;

class Ball : MonoBehaviour
{
}
```

Task 6: Creating functions

You can add your own custom functionality to a class by creating **functions**. A function is a list of instructions which will be performed by the computer whenever the function is run. To create a basic function, use the **void** keyword followed by the **name** of the function and a set of normal **brackets**. This must then be followed by a set of **curly brackets** to mark the start and end of the function. Functions are sometimes called methods in C#.

<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/methods>

Create a basic function called **Start** inside the Ball class.

```
class Ball : MonoBehaviour
{
    void Start()
    {
    }
}
```

You can name functions anything you like. However, Unity looks for functions with specific names and will run those functions at certain times. For example, Unity will look for a function called **Start** and run it once, when the game is first started.

<https://docs.unity3d.com/Manual/ExecutionOrder.html>

A function can contain a list of instructions, called **statements** in C#. A statement generally consists of a single line of code that ends in a semicolon. This will be covered in more detail in a future lab, so for now, just copy the code below which adds a **print** statement which will output a message to the Unity message log when the game is started.

```
void Start()
{
    print("Hello World");
}
```

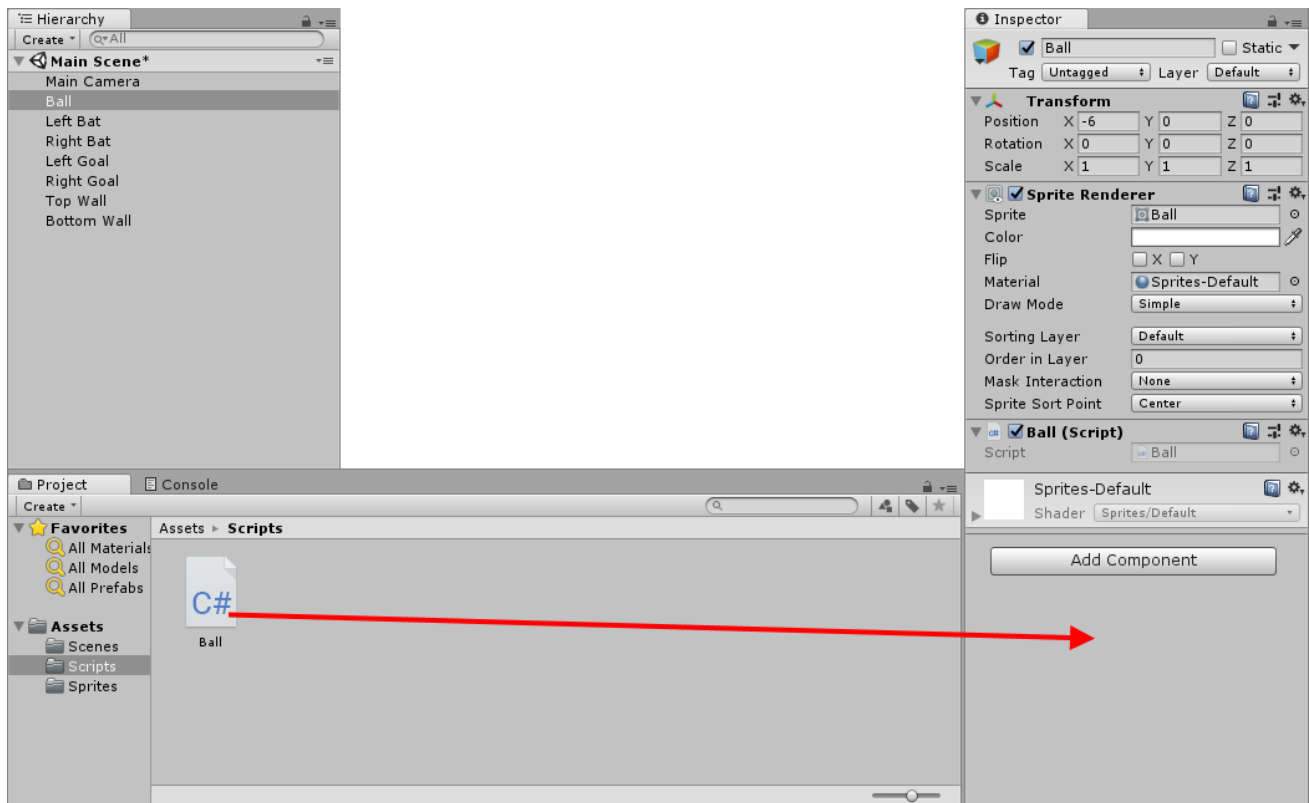
The complete script should now look like this:



Save the script (**File/Save** or **Ctrl+S**) and switch back to the main Unity window (Unity may freeze for a moment while it reloads the script).

Task 7: Attaching scripts

A script is just an asset, it will not do anything unless it exists in the scene as part of a game object, so select the Ball game object in the hierarchy and click and drag the script from the project panel to the empty space in the inspector (you could instead use the Add Component button and locate the script under **Scripts** then **Ball**).

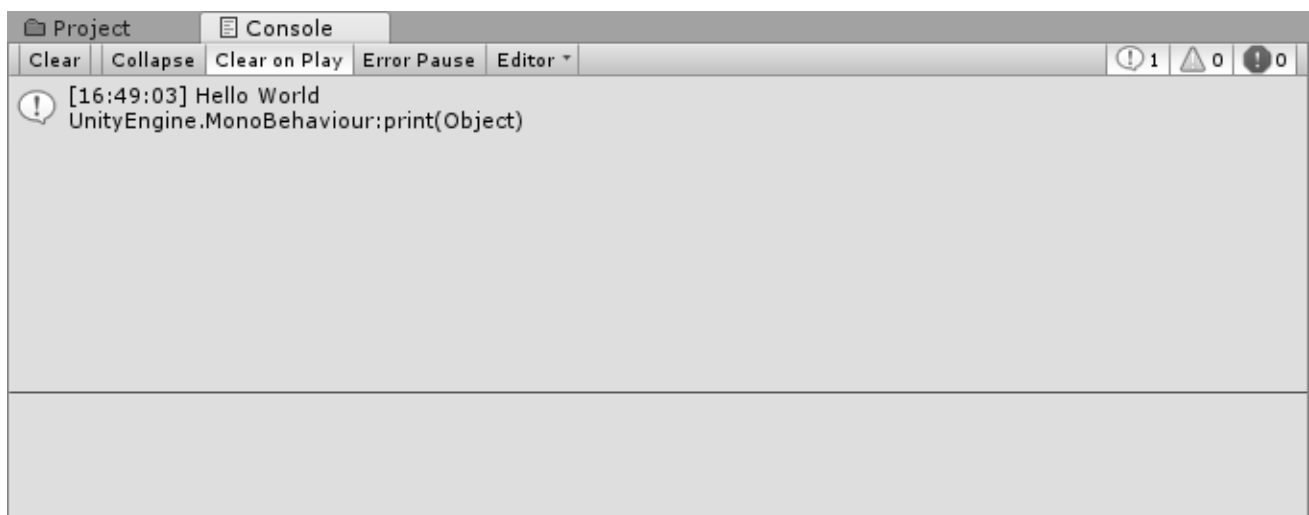


Click on the Console tab (next to the Project tab at the top of the project panel) to switch to the **Console Panel**, which shows errors, warnings and other messages generated by Unity. It also shows messages generated by scripts when the game is run.

Have another look at the Toolbar at the top of the window. In the centre are the **Play**, **Pause** and **Step** buttons.



With the console panel visible, click the Play button to enter **Play Mode** and you should see a message saying "Hello World" in the console panel. Click the Play button again to exit Play Mode.



Task 8: Creating variables

A class is like a reusable blueprint which you can use to create many different **instances** of the class. However, not all instances of a class will be 100% identical (for example, each sprite renderer displays a different image).

If you had multiple balls in your scene, they could have different sizes, perhaps they could move at different speeds or award the player different scores when they hit the goal. **Variables** can be used to store this kind of information. They can store the differences (or variations) between instances of your class.

To create a variable, you need to specify the **type** of data it should store followed by a unique **name** that can be used to identify the variable. Some common basic data types include:

- `int` positive or negative integer (whole number, -11 or 0 or 9 or 237)
- `float` positive or negative floating-point number (decimal, -5.74f or 7.0f)
- `char` single character (letter, number or symbol, 'e' or 'E' or '5' or '?')
- `string` series of characters ("John Smith" or "Hello, world!")
- `bool` yes/no value (`true` or `false`)

<https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/types-and-variables>

Looking at the list of data types above, what type of variable would be suitable to store the size of the ball? An `int` variable would work, but a `float` variable would offer greater precision since you could use decimal numbers.

Switch back to the Ball script and create a floating-point variable in the Ball class called **size**. Then, change the print statement to output the value of the size variable, like this:

```
class Ball : MonoBehaviour
{
    float size;

    void Start()
    {
        print(size);
    }
}
```

Save the script and switch back to Unity. Give Unity a second to reload the script, then use the play button to run the game. Look at the console panel and you should see that the script has output the **value** of the size variable, which is 0 by default.

Variables have a default value which they hold: for number types such as `int` or `float` the default value is 0 (zero). To provide a different initial value, when creating a variable, you should add an equals sign after the name followed by the value that you want the variable to store. This is called **initialising** the variable.

<https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/default-values-table>

Click the play button again to leave play mode. Then, switch back to the script. If you look up at the list of data types above, you'll see that when writing floating-point numbers in C# you should add a lowercase letter `f` to the end of the number, to indicate that it is a floating-point number. With that in mind, change the line of code where you created the size variable to initialise it to the value 1.

```
float size = 1.0f;
```

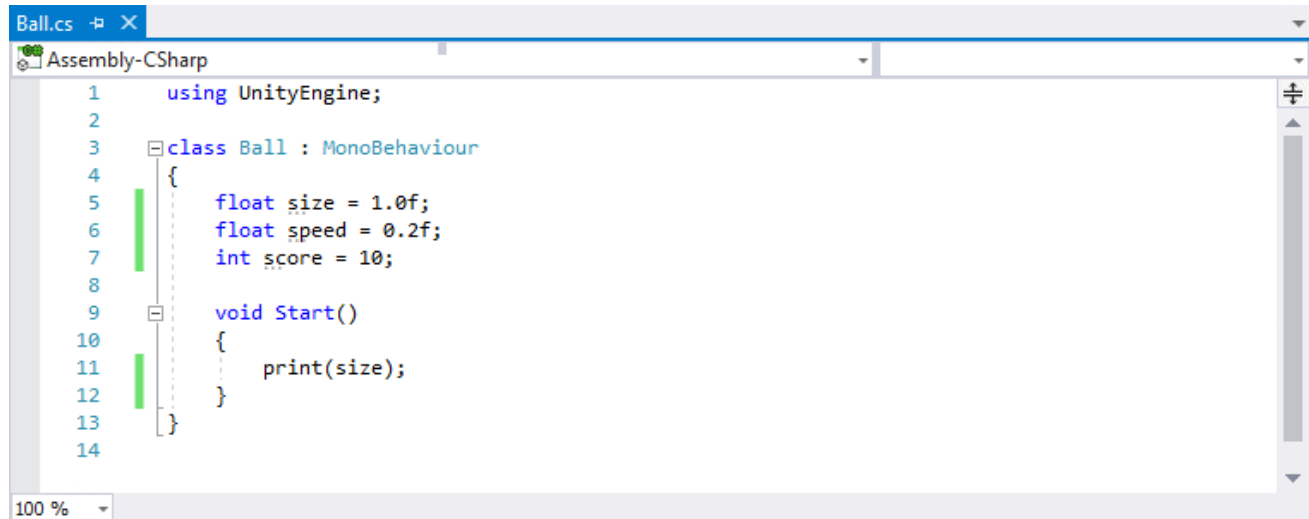
Save the script, return to Unity, use the play button and you should see that the script now outputs the number 1. Leave play mode and return to the script.

TO DO:

You should repeat the steps above to create the following variables in the Ball class:

- speed (a `float` with initial value `0.2f`)
- score (an `int` with initial value `10`)

The complete script should now look like this:



```
1  using UnityEngine;
2
3  class Ball : MonoBehaviour
4  {
5      float size = 1.0f;
6      float speed = 0.2f;
7      int score = 10;
8
9      void Start()
10     {
11         print(size);
12     }
13 }
14
```


Task 9: Changing variable values

Initialising a variable sets its initial value, but you can also change the value of a variable afterwards within a function. This is called **assignment**. To do so, you write the name of the variable followed by an equals sign and the value that you want to store.

<https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/assignment-operator>

In the Start function, after printing the initial size, assign a different value to size (such as 2.75f) and then print it a second time, like this:

```
void Start()
{
    print(size);

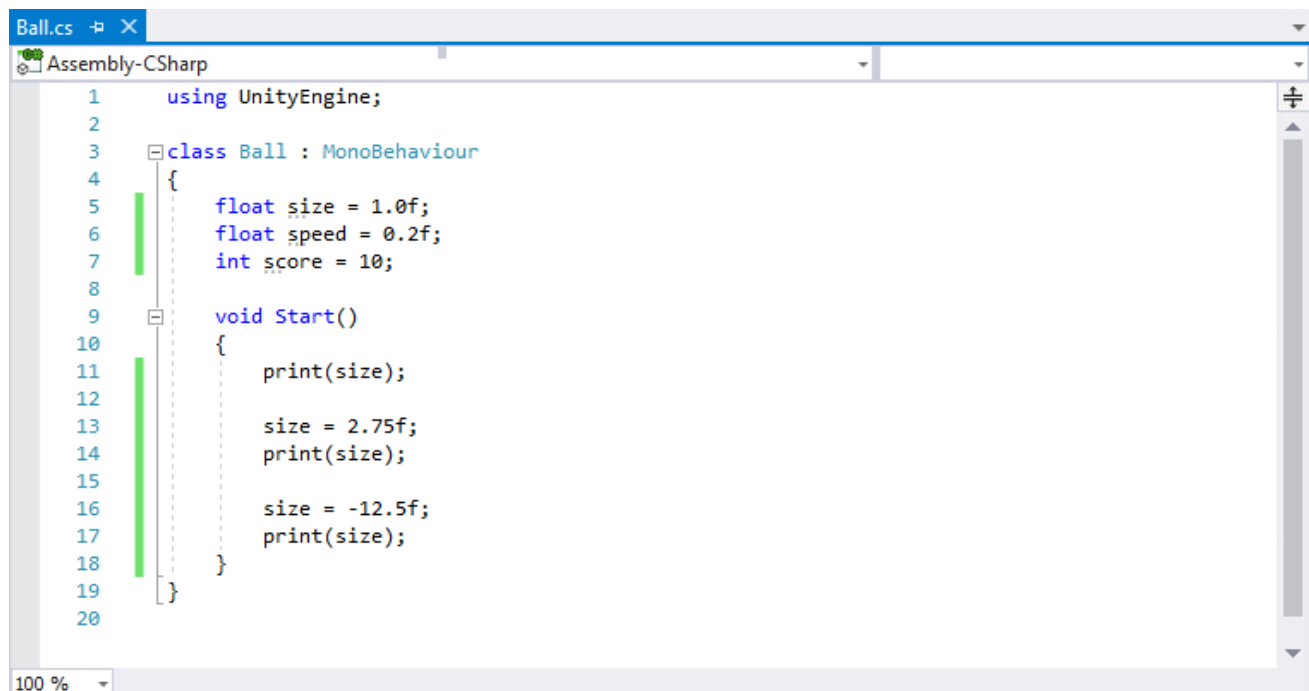
    size = 2.75f;
    print(size);
}
```

Save the script, switch back to Unity, use the play button and you should see that the script has output two different numbers to the console panel. First, it outputs the initial size and then it outputs the new size. Exit play mode and return to the script.

TO DO:

In the Start function, after printing the new value, assign yet another value to size (like -12.5f) and print the new value. Save the script and test your changes in Unity.

The complete script should now look something like this:



Task 10: Setting variable visibility

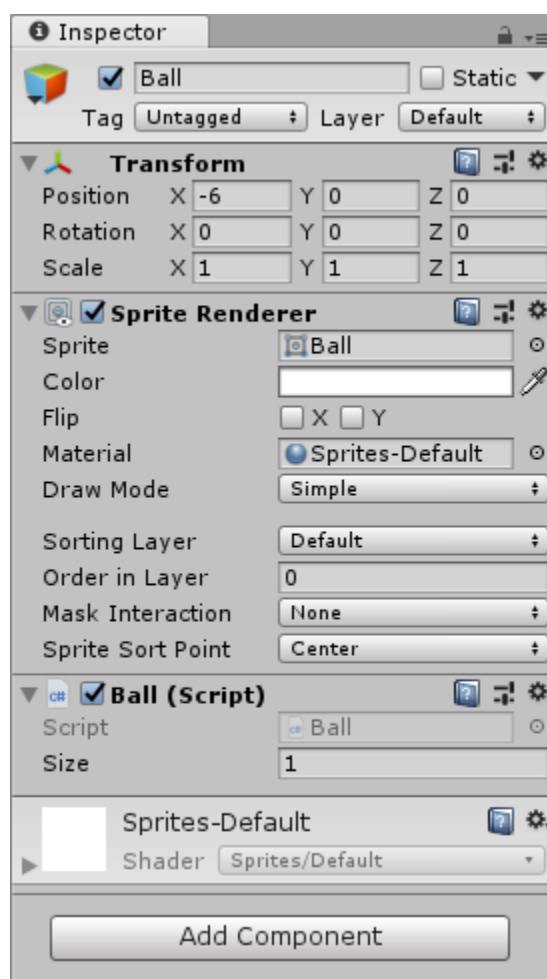
By default, variables in your class are **private**, which means that they can only be accessed from inside the class which contains them. If you want to be able to see and modify them in the Unity inspector, then you need to set their **visibility** to **public**. To do this, you must add the **public** keyword before the type.

<https://docs.unity3d.com/Manual/VariablesAndTheInspector.html>

Change the line of code where you created the size variable to give it public visibility:

```
public float size = 1.0f;
```

Now save the script and switch back to Unity. Give Unity a second to reload your script, then select the Ball game object in the hierarchy and look at the inspector. You should see the size variable in the inspector with the initial value of 1.



Try changing the value of the size variable in the inspector and then using the play button and looking at the values written to the console. You should see that the value you set in the inspector overrides the initial value set in the script. However, assigning a value in a function then overrides the value set in the inspector.

TO DO:

You should make the speed and score variables public and try editing them in the inspector. Try setting the score variable in the inspector to a decimal value – you're not able to because score is defined as an integer variable in the script.

End of Lab

Make sure that you save the script in Visual Studio before closing it **and** save the scene in Unity before closing it.

Although not essential, it would be useful to copy your project onto a USB drive or upload it to a cloud storage account – OneDrive cloud storage is available by signing in to office.com using your university login details (such as <username>@caledonian.ac.uk and your university password).