

# Introduction to Unity Workshop: Walkthrough Notes

By Peter Mitchell

## Contents

Introduction to Unity Workshop: Walkthrough Notes .....	1
1 Foreword .....	2
2 Project Creation .....	2
3 The Unity Interface .....	2
4 Creating Game Objects .....	3
4.1 Creating Primitives .....	3
4.2 Organising folders .....	5
4.3 Creating a Material .....	6
4.4 Applying a Material .....	7
4.5 Moving GameObjects .....	8
4.6 Creating a Second Material .....	8
5 Prefab Creation .....	9
5.1 Prefabs Introduction .....	9
5.2 Prefab Variants .....	10
6 Unity Controls .....	12
7 Scripting .....	12
7.1 Changing your Script Editor .....	12
7.2 C# Scripting Introduction .....	12
7.3 Running the Application .....	14
7.4 Spawning GameObjects from Code .....	14
7.5 Destroying objects with a script .....	19
8 Extra Content .....	21

# 1 Foreword

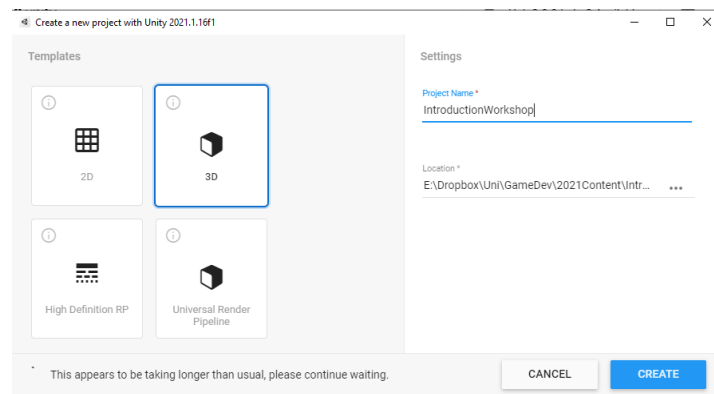
This document in combination with the practical demonstration during the workshop and provided materials along with the material for following weeks are intended to introduce the basics of the Unity Game Engine. You can find the full project files on GitHub at:

<https://github.com/Squirrelbear/IntroductionWorkshop>

Unity Version Used: 2021.1.16f1

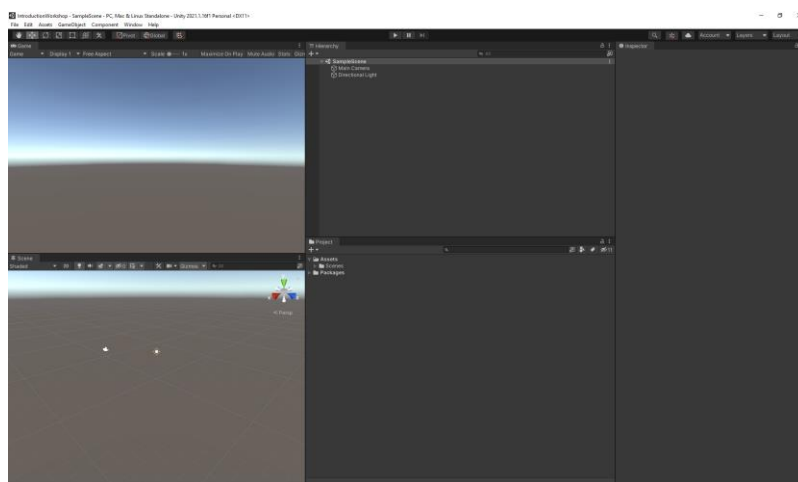
## 2 Project Creation

1. When creating projects, you will normally want to just select 2D or 3D for initially learning. For this example, select the 3D, give it a name like IntroductionWorkshop, and choose somewhere for it to go.

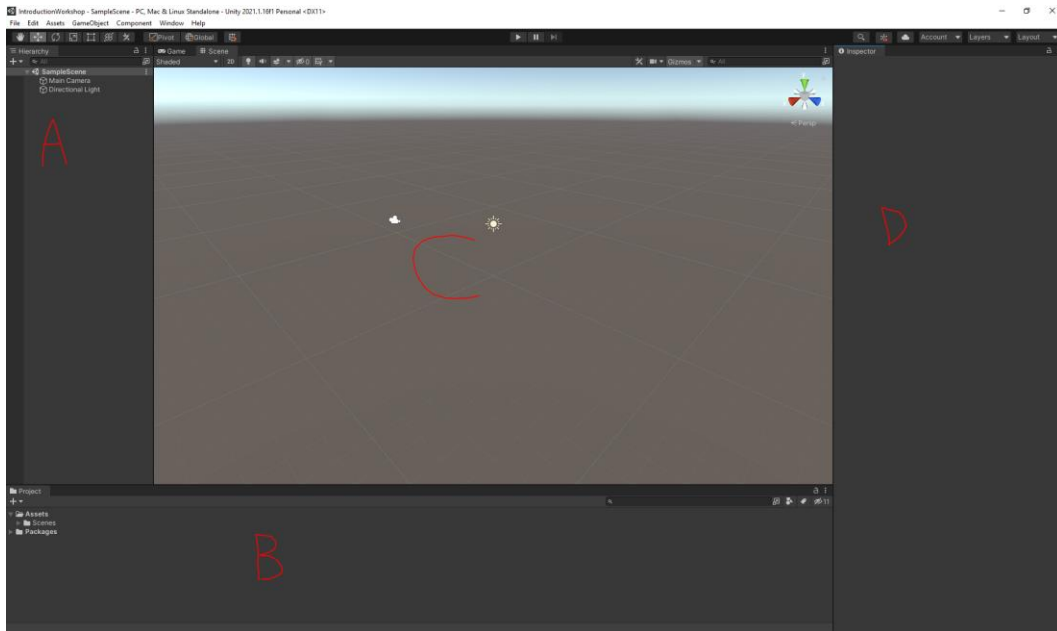


## 3 The Unity Interface

2. Once the project opens you may get an interface that looks similar to this layout. You can move the panels around if you choose.



3. Personally I have mine reorganised as seen below, but you can place them however you like.

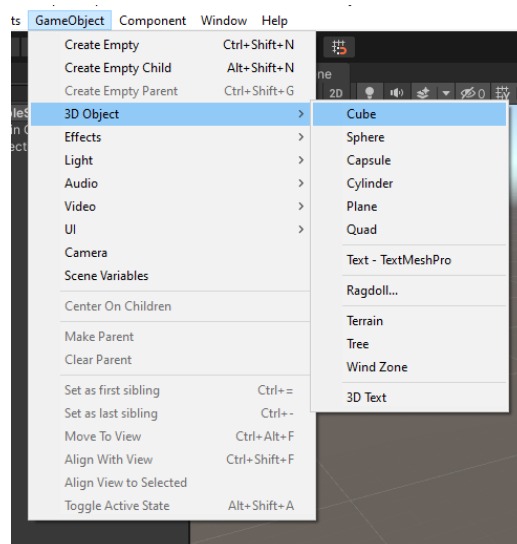


- A: Hierarchy Tab: Shows all the objects in your game scene.
- B: Project Tab: Shows all the files available in your project.
- C: Scene/Game Tab: Shows the current view within your scene or running game.
- D: Inspector Tab: Shows the properties of a currently selected game object.

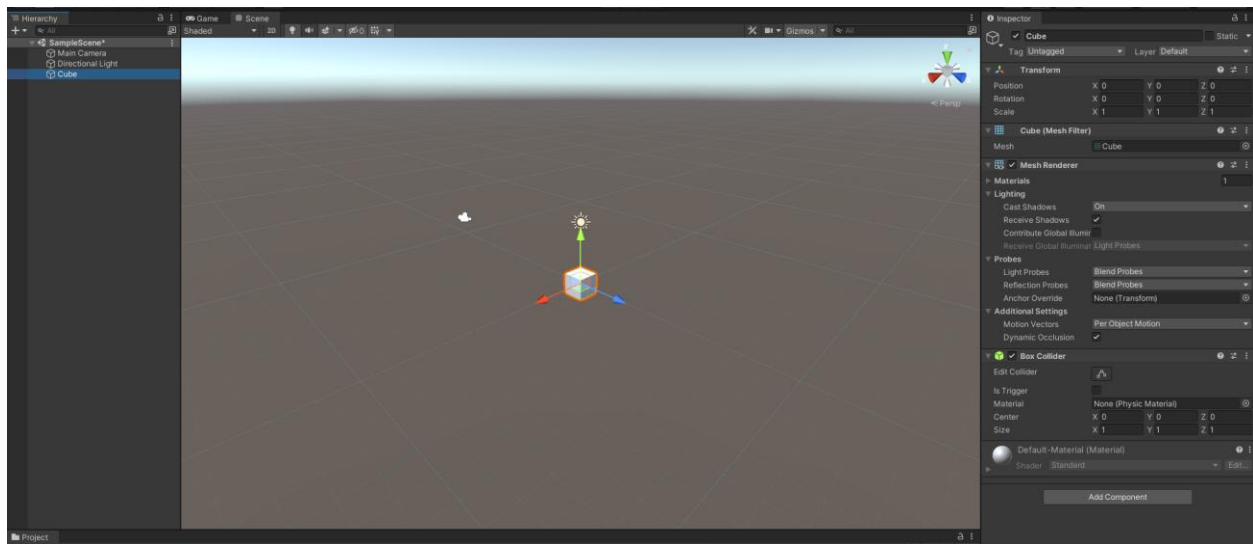
## 4 Creating Game Objects

### 4.1 Creating Primitives

4. You can create basic 3D primitives via the GameObject menu. For example, to create a Cube object you can go to GameObject -> 3D Object -> Cube.



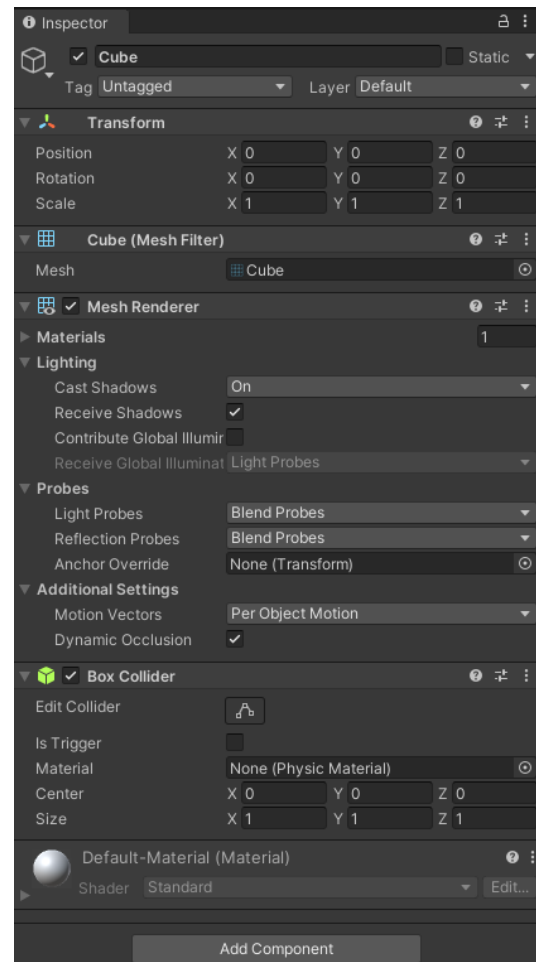
5. After creating the cube, you should see it appear as seen below.



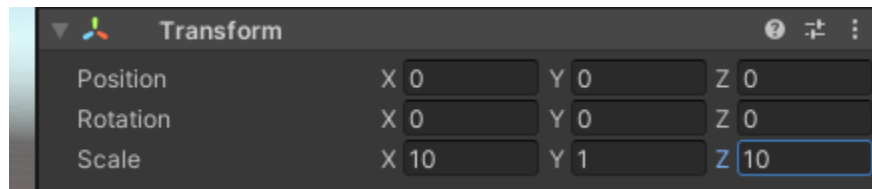
6. The Inspector with the Cube selected is seen to the right.

The inspector contains all the information about the properties of a single object. It is made up of multiple different components listed below.

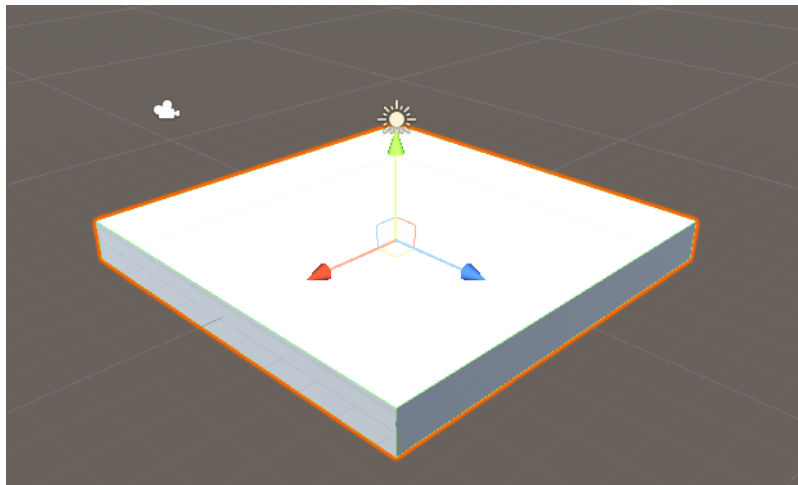
- Transform
  - Position
  - Rotation
  - Scale
- Cube (Mesh Filter): The 3d model of the cube.
- Mesh Renderer: Manages the rendering of the mesh.
- Box Collider: Allows for detection of collisions with other objects. Note that this one is only for 3D. The 2D one has 2D in the title of the component.
- Material: Automatically shows Default-Material. This is the component controlling colouring and texturing.



7. Set the X and Z scale to 10.

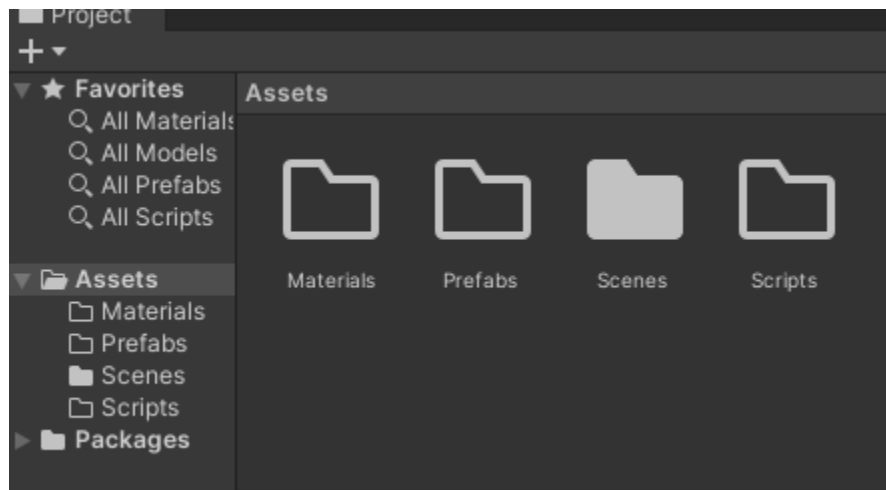


You should see something similar to the below.



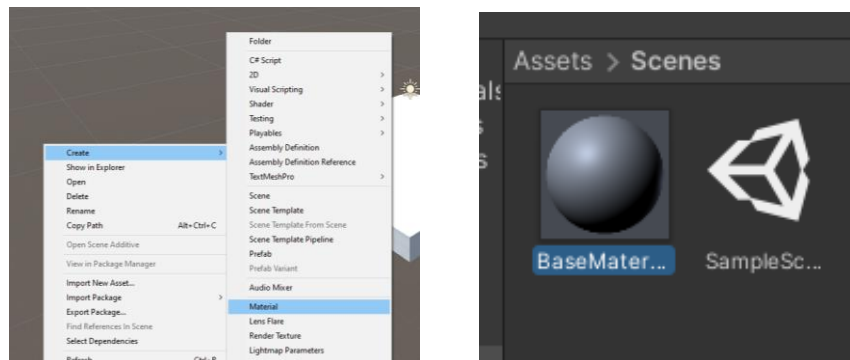
## 4.2 Organising folders

8. When managing your project, it will likely contain many different types of assets. You should try to split files up into different folders depending on their types and perhaps even sub-folders for categories within those folders. For example, see below.



## 4.3 Creating a Material

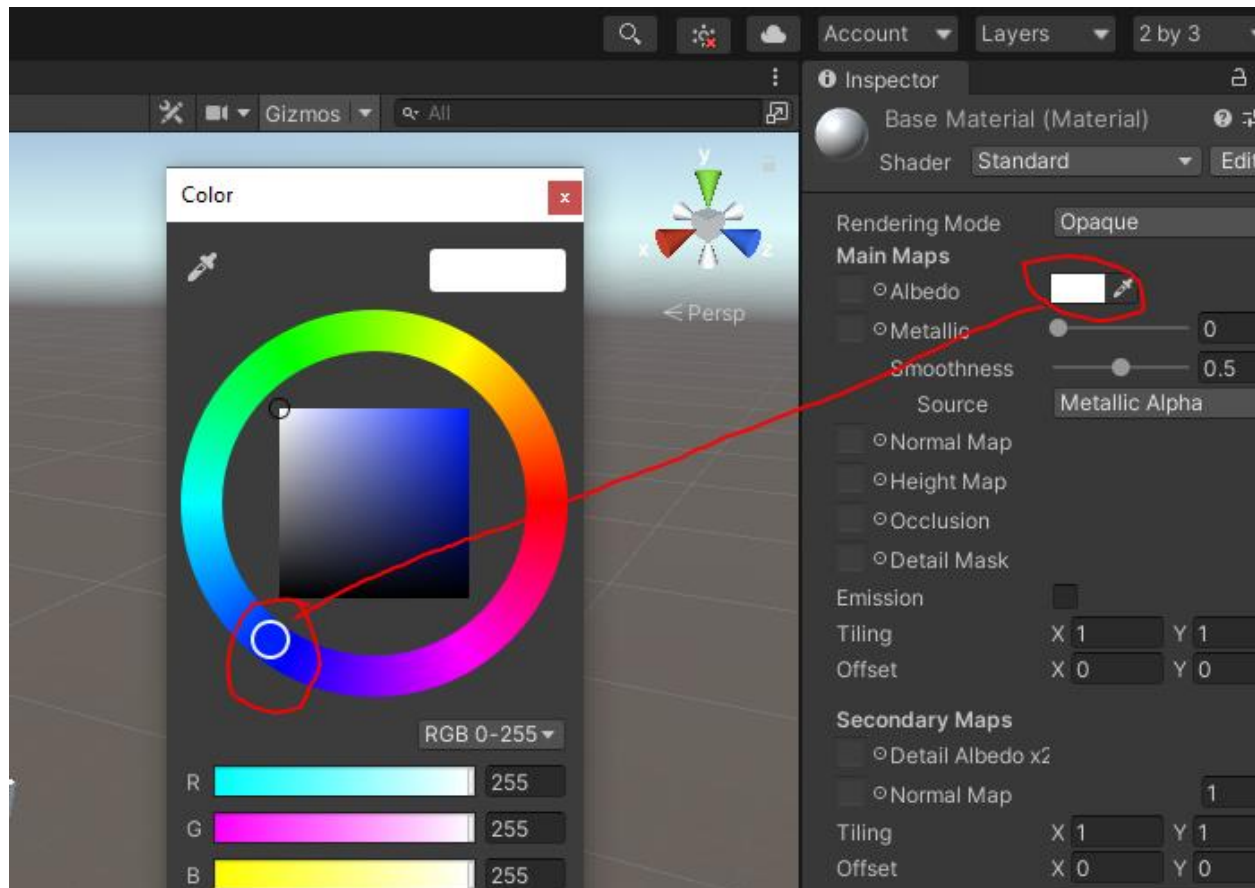
9. Right click in scenes folder and create a material and call it BaseMaterial.



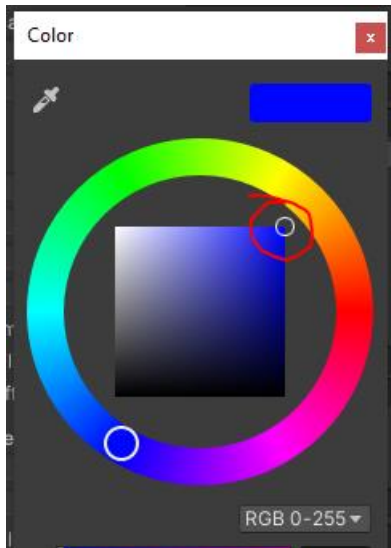
10. Editing materials have many properties. For this workshop only colours are going to be used. You can look into what the other options are for. The following two are the important attributes for this.

- Rendering mode: Opaque = flat colour, Transparent = allows for see through materials.
- Albedo: Colour for the material

Set the Albedo to Blue.

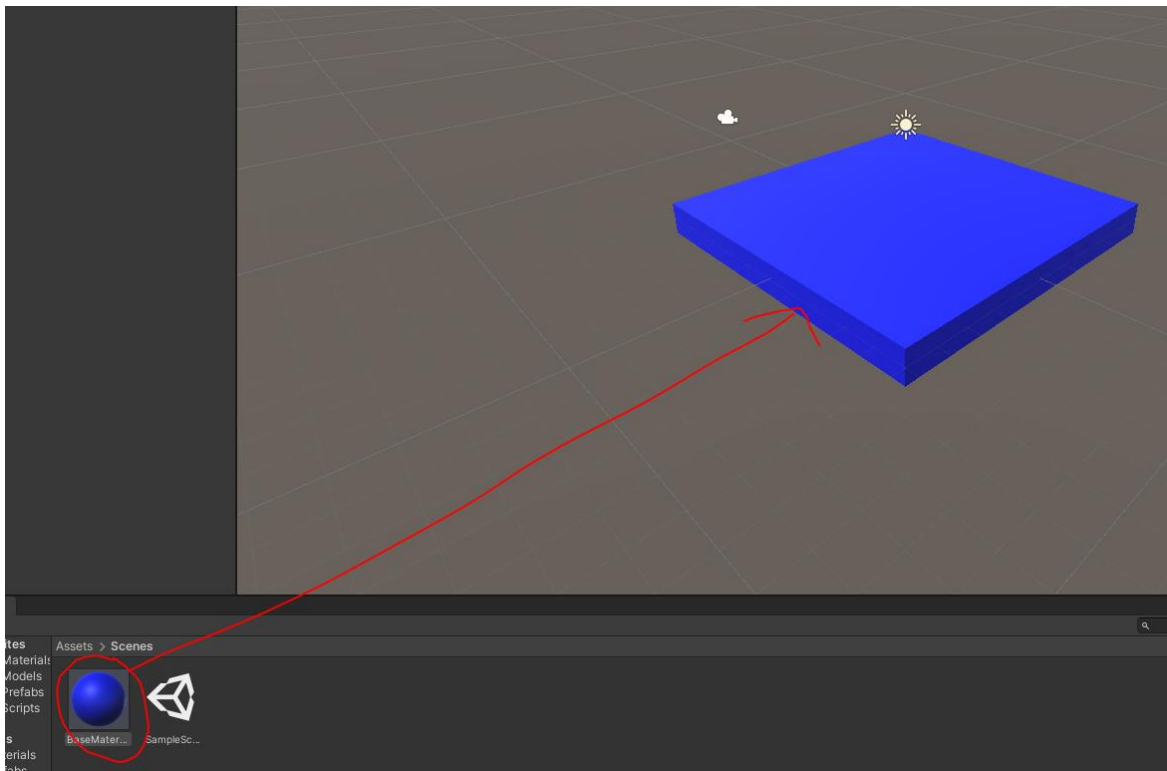


11. You will need to make sure you move the little circle on both the ring and in the square to change the colour.



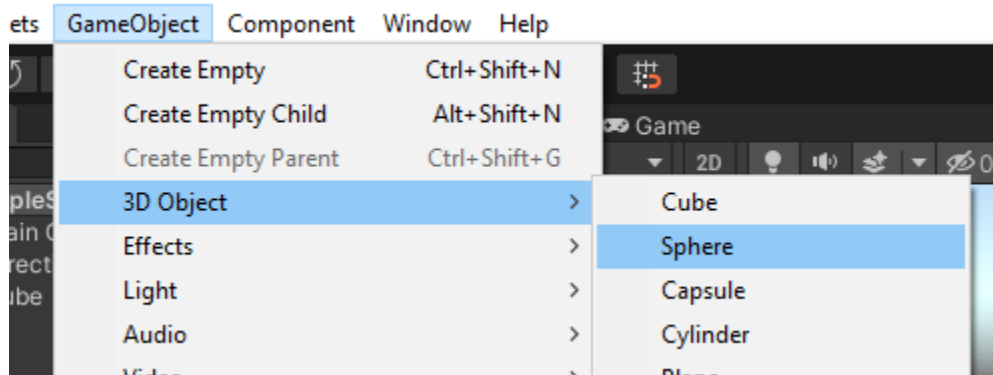
## 4.4 Applying a Material

12. Drag and drop the material onto the object.



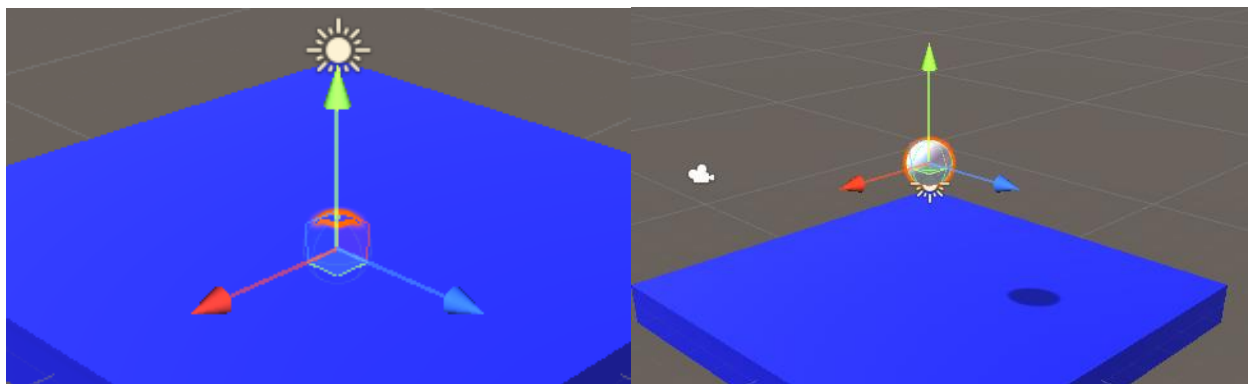
You can also add this by dropping it the Cube object in the hierarchy.

13. Add a Sphere.



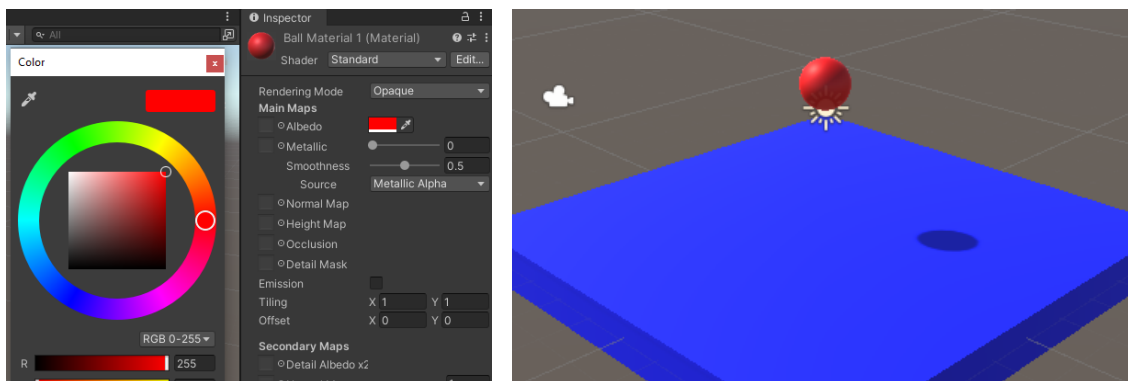
## 4.5 Moving GameObjects

14. You will notice after the sphere is created, or any other object are selected the three coloured arrows appear. You can use these to move the object in the scene. For example, select the green arrow and drag upward. You should see the result as seen on the right.



## 4.6 Creating a Second Material

15. Create a second material called BallMaterial1 and set the Albedo to Red. Then drag the material onto the sphere to apply it.



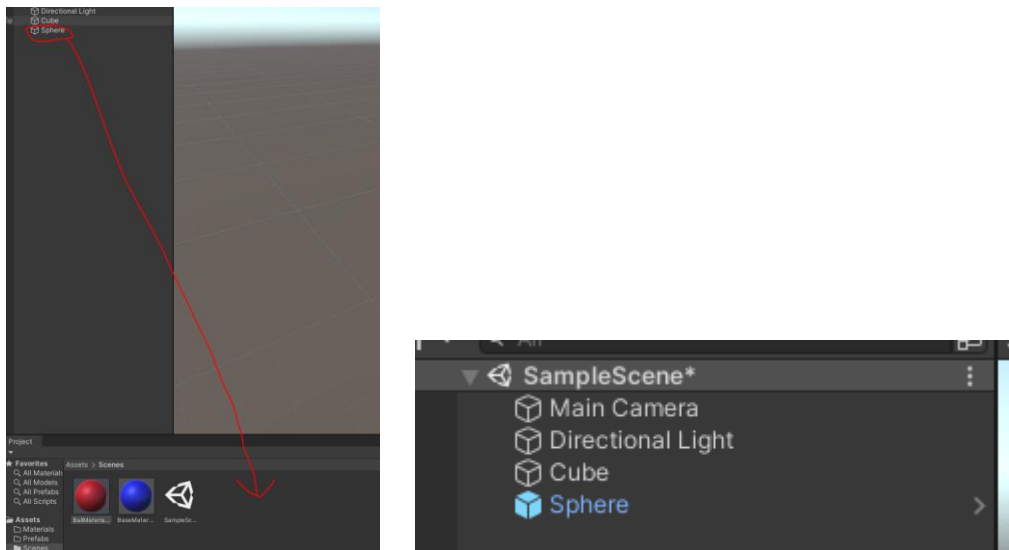


## 5 Prefab Creation

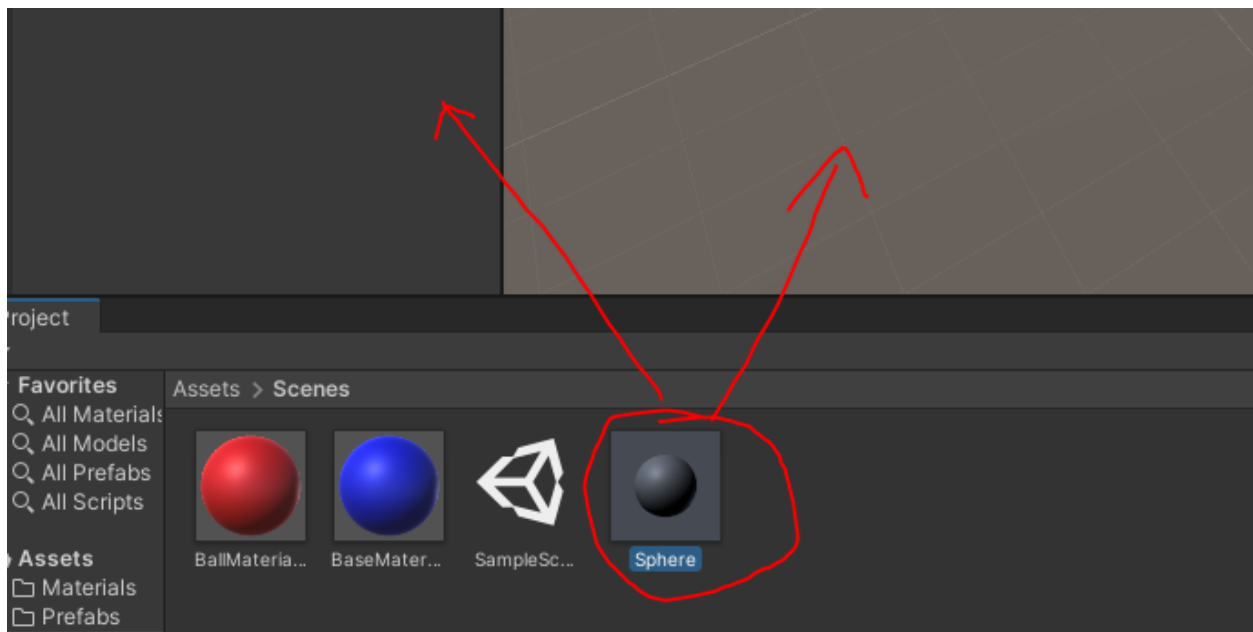
Prefabs allow you to create objects that are templates for objects, or already complete objects that are ready to drop straight into a scene with components all set up.

### 5.1 Prefabs Introduction

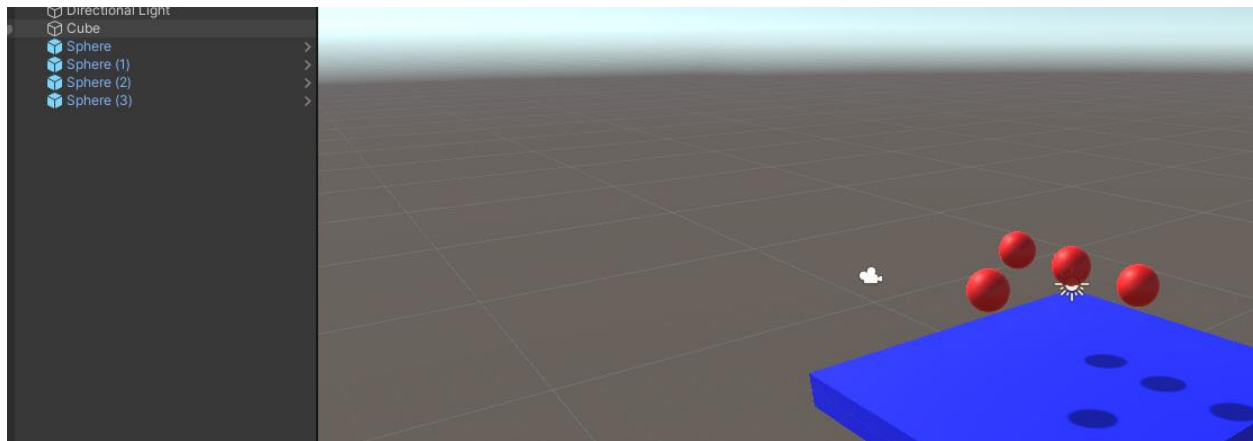
16. Drag Sphere GameObject into Project Window folder. Note the visual change after you do this in the hierarchy that shows the object is linked to a prefab. Making changes to the prefab would also change objects in the scene.



17. Can now drag and drop multiple spheres into the scene. You can also duplicate the sphere in the scene with Ctrl+D.



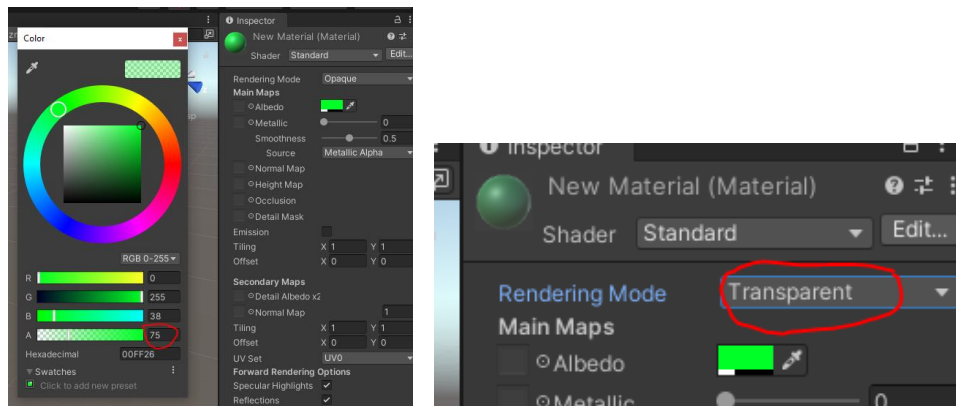
18. For example, you may have something like below. Where the spheres have been added four times and then moved to be spread apart.



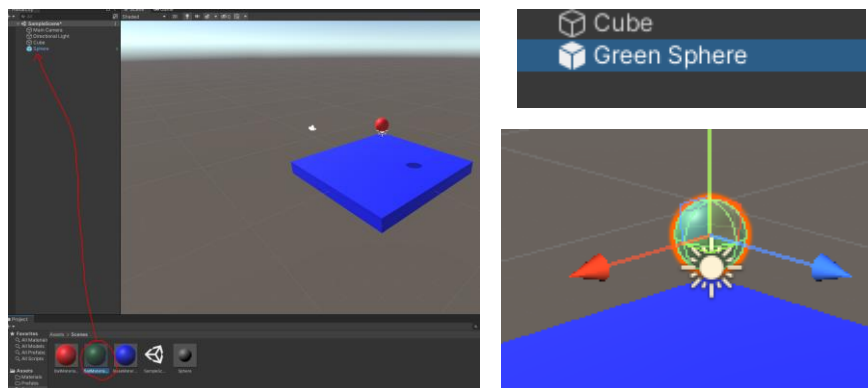
19. Delete the extra spheres. You only need one in the scene at the moment.

## 5.2 Prefab Variants

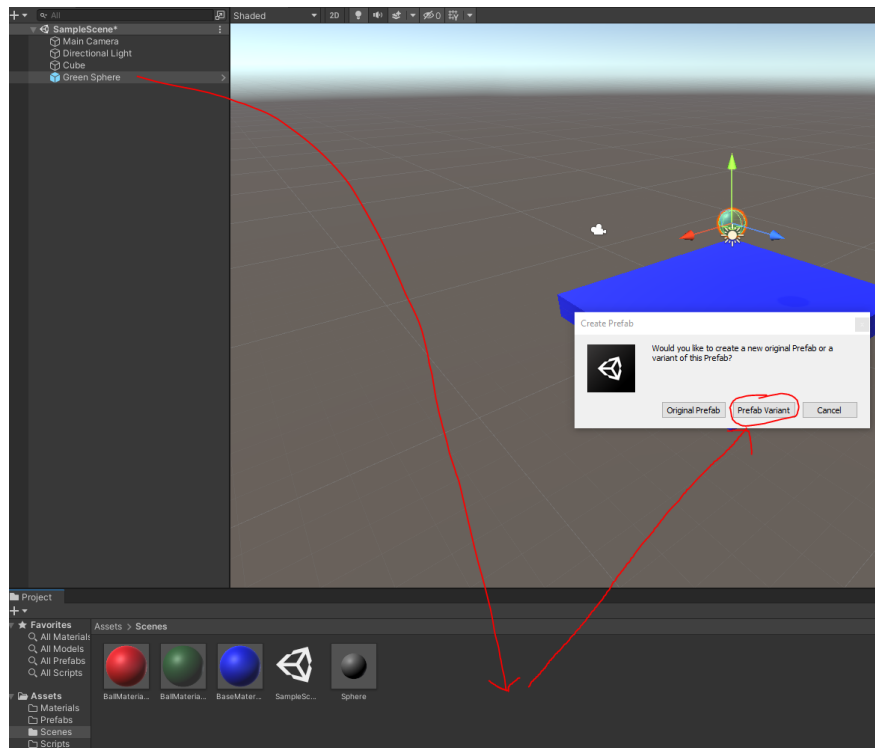
20. Create second material for ball called BallMaterial2: Albedo green and set alpha to 75. Change rendering mode to transparent.



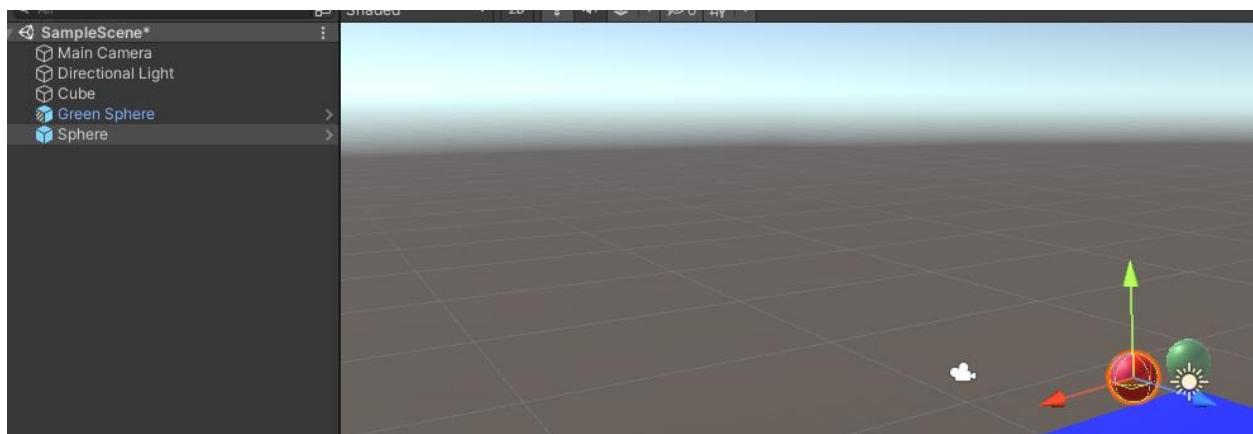
21. Drag second material onto a prefab object in hierarchy and name the object Green Sphere.



22. Drag the object into the project folder again. It will ask whether to make it a Prefab or Prefab Variant. Prefab Variant means that if you change the original Prefab those changes will be also made to the Prefab Variant as well.



23. Drag the original Sphere prefab into the scene again and move it to not be inside the green sphere. You should be able to see two spheres as seen below.



## 6 Unity Controls

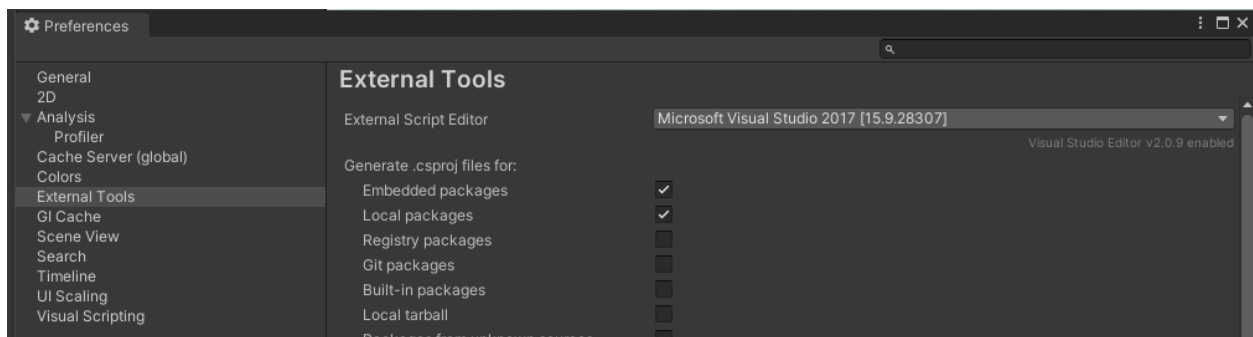
24. Some useful controls to know about for getting around.

- Camera Controls
  - Zoom: Scroll mousewheel
  - Look Up/down/left right: Hold right mouse button
  - Move in direction: Hold right mouse button and use WASD
  - Pan with current rotation: Hold middle mouse button and move around.
- Duplicate object: Ctrl + D

## 7 Scripting

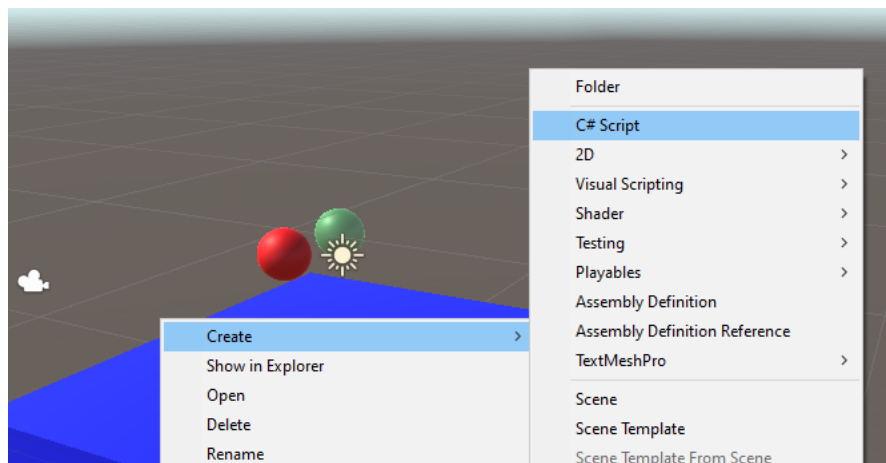
### 7.1 Changing your Script Editor

25. If you have a different preferred script editor to what it has you using, you can change that. Go to Edit->Preferences And then select External Tools and at the top is the External Script Editor option. Personally, I would recommend using some version of Visual Studio if you can.



### 7.2 C# Scripting Introduction

26. Right click and create a new C# Script and call it BallSpawner.cs.



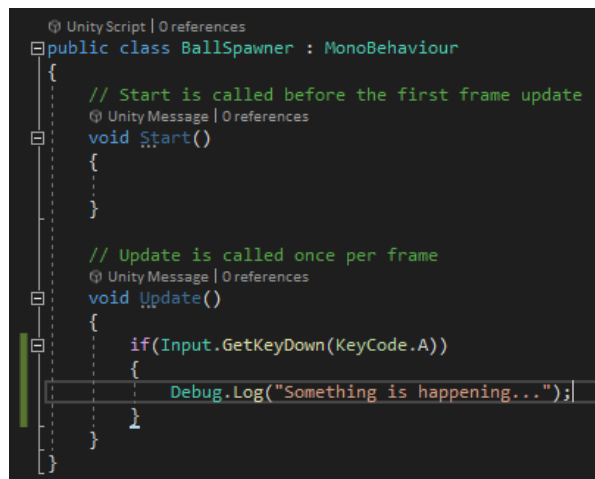
27. The BallSpawner class automatically extends from MonoBehaviour. This is required for any object that exists as a component on a GameObject. You do not need to inherit from MonoBehaviour for scripts that don't appear on objects.

Contains a Start() function and Update() function. Start is called when the object is created, and Update is called at regular intervals to allow changes to be made to the object before the next pass of rendering.

In the Update() method add the code:

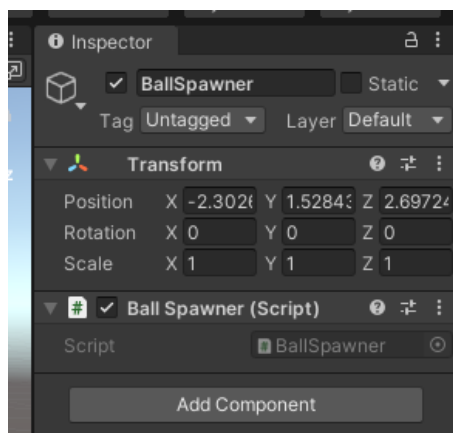
```
if(Input.GetKeyDown(KeyCode.A)) {  
    Debug.Log("Something is happening...");  
}
```

It should look like this.



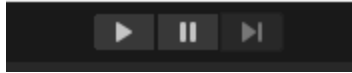
```
UnityScript | 0 references  
public class BallSpawner : MonoBehaviour  
{  
    // Start is called before the first frame update  
    Unity Message | 0 references  
    void Start()  
    {  
    }  
  
    // Update is called once per frame  
    Unity Message | 0 references  
    void Update()  
    {  
        if(Input.GetKeyDown(KeyCode.A))  
        {  
            Debug.Log("Something is happening...");  
        }  
    }  
}
```

28. Create an empty GameObject via the GameObject menu. Rename the object BallSpawner.
29. Drag the BallSpawner script onto the BallSpawner object.  
(You can also use the Add Component button to search for "BallSpawner" and click to add it.)



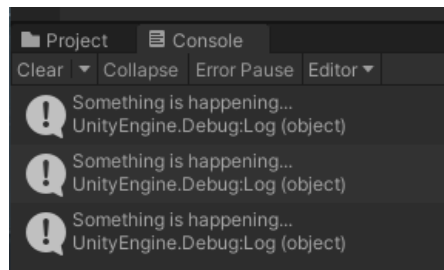
## 7.3 Running the Application

30. Press the Play button at the top.



31. The Project/Console tab will swap to the Console tab. You may wish to click on “Collapse” to make it stop collapsing debug messages that are the same into a single message. You can look at the counter on the right going up to show that it has triggered multiple times though.

32. Press A and you should see debug messages appear either in multiples, or just the one, with a count of how many times it has been shown on the right.



## 7.4 Spawning GameObjects from Code

33. Open the BallSpawner.cs file again and comment out the debug line.

34. Declare a public GameObject instance variable called ballPrefab. By making this variable public it will be visible in the inspector within Unity and we can set it to a prefab reference for cloning.

35. Just after the commented out Debug.Log line write the following.

Instantiate(ballPrefab, transform.position, Quaternion.Identity);

This should so far look like the following.

```
UnityScript (1 asset reference) | 0 references
public class BallSpawner : MonoBehaviour
{
    // Reference to an object that can be set inside the Inspector
    public GameObject ballPrefab;

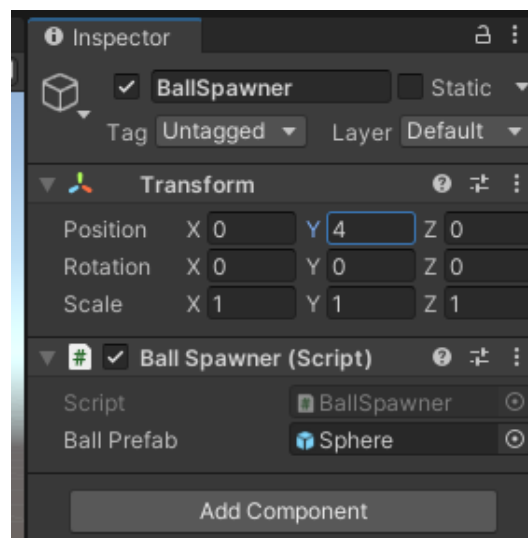
    // Start is called before the first frame update
    @ UnityMessage | 0 references
    void Start()
    {
    }

    // Update is called once per frame
    @ UnityMessage | 0 references
    void Update()
    {
        // When the A key is Pressed Down
        if(Input.GetKeyDown(KeyCode.A))
        {
            // Write to the Console
            //Debug.Log("Something is happening...");

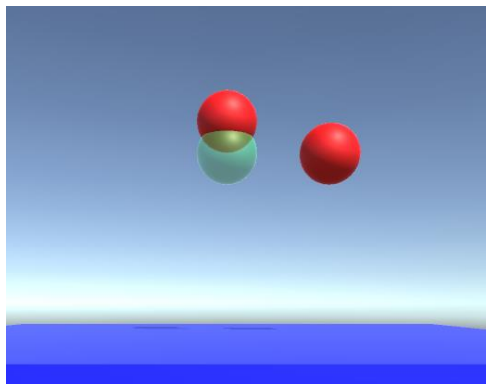
            // Create an object cloning the ballPrefab object spawning it at the position
            // of the current object this script is attached to with no change to rotation.
            Instantiate(ballPrefab, transform.position, Quaternion.identity);
        }
    }
}
```

The Instantiate command takes three parameters. The first is a reference to a GameObject. This object will be cloned when instantiating you can either use a Prefab from your project files or an object from your scene. The second parameter is a Vector3 position, this sets the position in 3D space. The value being passed to this variable is a reference to the current object's transform getting the position of that transform. This comes from inheriting from MonoBehaviour. It accesses the transform of the object the script is attached to. The last parameter is a rotation represented by a Quaternion. Quaternion.identity is essentially saying 0 rotation.

36. Save the script and return to Unity. Make sure to stop Unity if it is running before the following steps. Making changes while the application is running will result in the changes being lost after running ends.
37. Select the BallSpawner object. Set the Position to (0, 4, 0). And Drag the Sphere Prefab from the Project tab into the BallPrefab variable on the Inspector. It should look like the below.



38. Run the program and test that pressing A now spawns a red sphere. It should appear something like the below.



39. Open the BallSpawner.cs file again.
40. Add another public instance variable of type List<GameObject> called balls. A List is an array that automatically resizes itself as objects are added.

41. Duplicate the Instantiate line you wrote. Comment out the original so you can still see what it looked like and modify the duplicate to be as follows.
- ```
Instantiate(balls[Random.Range(0,balls.Count)], transform.position, Quaternion.identity);
```

This will take the list of balls and select any randomly from the list. You should see code that appears as follows.

```
UnityScript(1 asset reference) | 0 references
public class BallSpawner : MonoBehaviour
{
    // Reference to an object that can be set inside the Inspector
    public GameObject ballPrefab;

    // List of multiple balls to spawn randomly from
    public List<GameObject> balls;

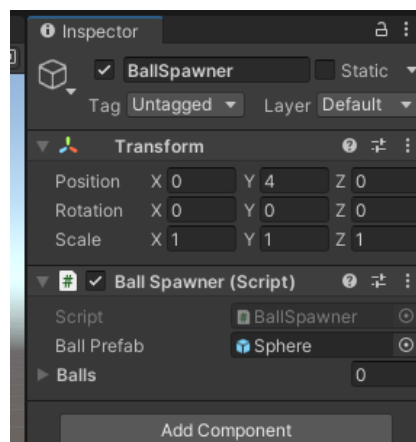
    // Start is called before the first frame update
    @ Unity Message | 0 references
    void Start()
    {
    }

    // Update is called once per frame
    @ Unity Message | 0 references
    void Update()
    {
        // When the A key is Pressed Down
        if(Input.GetKeyDown(KeyCode.A))
        {
            // Write to the Console
            //Debug.Log("Something is happening...");

            // Create an object cloning the ballPrefab object spawning it at the position
            // of the current object this script is attached to with no change to rotation.
            //Instantiate(ballPrefab, transform.position, Quaternion.identity);

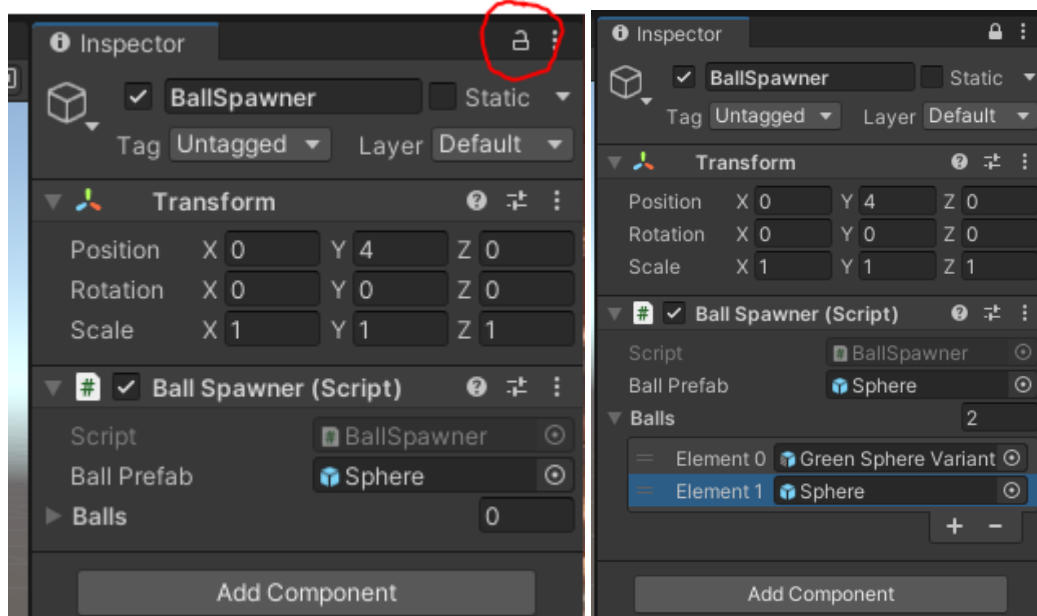
            // Spawn a random ball from the balls array.
            Instantiate(balls[Random.Range(0,balls.Count)], transform.position, Quaternion.identity);
        }
    }
}
```

42. Open Unity again. You should now see on the BallSpawner GameObject's BallSpawner script component a Balls element as seen below.

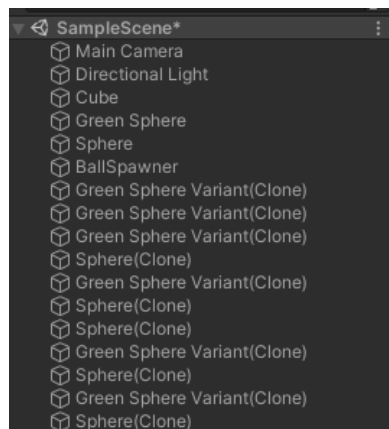




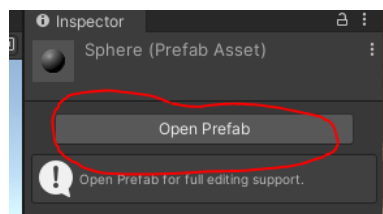
43. Click the lock icon in the top right corner of the inspector with the BallSpawner object selected. Select the Sphere and Green Sphere Variant objects from the Project tab. Drag them into the Balls List in the Inspector.



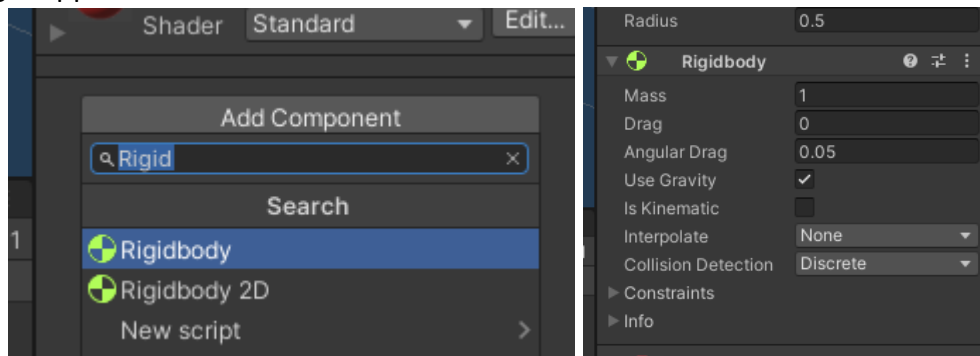
44. Make sure to click the lock icon again to unlock the inspector.
45. Run the program and test that multiple random elements are spawning. Press A multiple times and you should see something similar to below. Note that they will all appear inside each other for now.



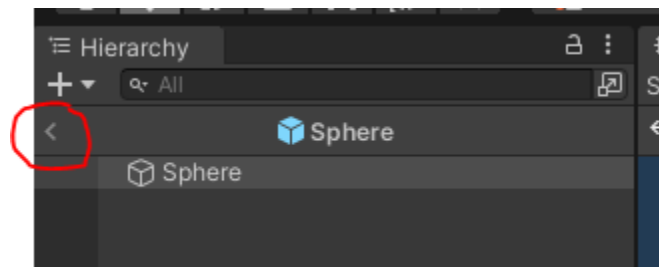
46. Stop the program running. You can now delete the two spheres from the Hierarchy tab.
47. Select the Sphere Prefab in your Project tab and then click the Open Prefab button seen below.



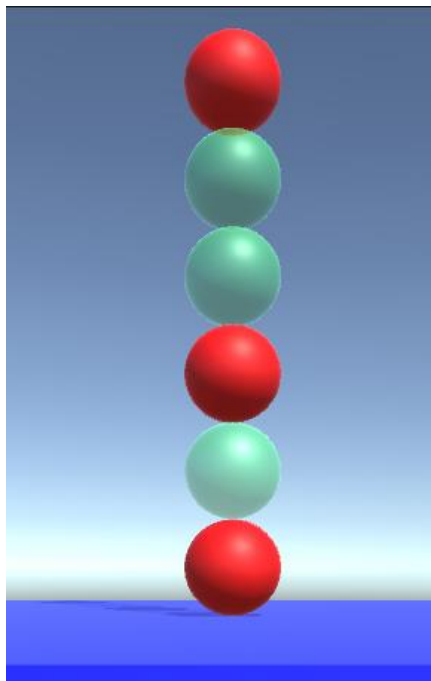
48. Click Add Component and start to type Rigidbody. You should see it appear. Make sure you select Rigidbody and not Rigidbody 2D. And you should see the component seen on the right appear.



49. To return out of editing the Prefab you can click the arrow circled below in the Hierarchy tab.



50. Run the game again and now when you press A the spheres will spawn and begin to drop. They should stack somewhat like seen below.



## 7.5 Destroying objects with a script

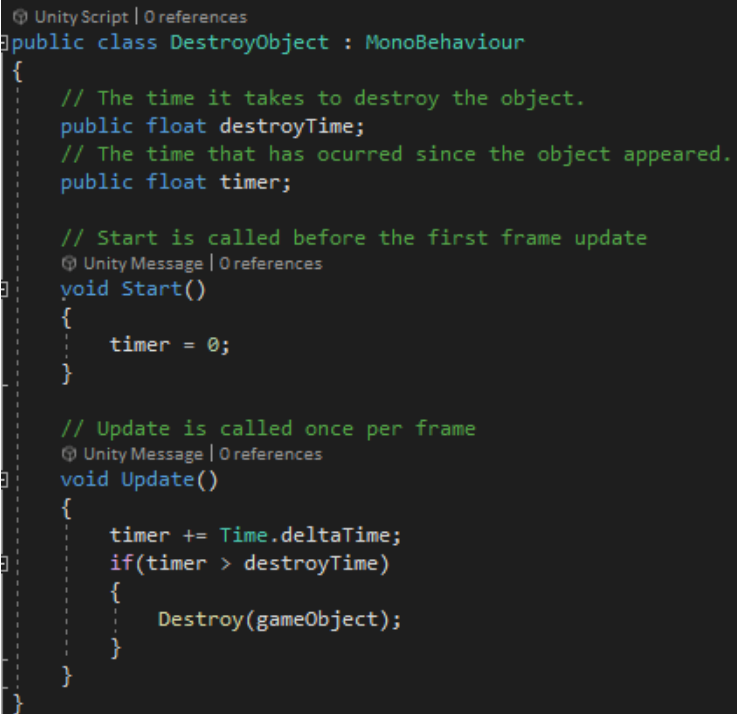
Spawning an infinite number of GameObjects is nice, but what if we want them to disappear after a short time.

51. Create a new C# Script called DestroyObject.
52. Declare a public instance variable of type float called destroyTime. This will be the time it takes to destroy the object that you will set in the inspector.
53. Declare a public instance variable of type float called timer. This is a timer that updates until the destruction happens.
54. In the Start() method set the value of timer to be 0.
55. In the Update() method add the following.

```
timer += Time.deltaTime;
if(timer > destroyTime) {
    Destroy(gameObject);
}
```

This code will update the timer variable by increasing it by the amount of time that has passed since the last update. Then it checks if the timer is now long enough to reach the destroyTime and if it is the object is destroyed. “gameObject” is a reference via the MonoBehaviour inheritance to the object the script is attached to.

Your code should appear as follows.



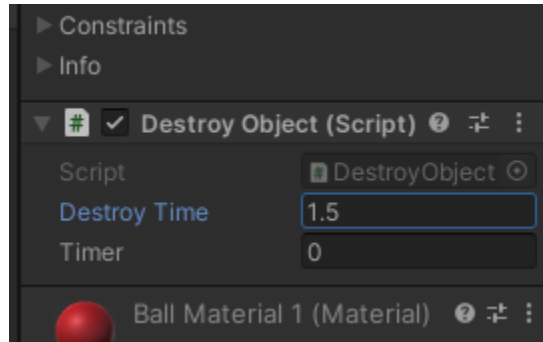
```

@ Unity Script | 0 references
public class DestroyObject : MonoBehaviour
{
    // The time it takes to destroy the object.
    public float destroyTime;
    // The time that has occurred since the object appeared.
    public float timer;

    // Start is called before the first frame update
    @ Unity Message | 0 references
    void Start()
    {
        timer = 0;
    }

    // Update is called once per frame
    @ Unity Message | 0 references
    void Update()
    {
        timer += Time.deltaTime;
        if(timer > destroyTime)
        {
            Destroy(gameObject);
        }
    }
}
```

56. Save your code and return to unity. Make sure the game is not running as well.
57. Open the Sphere Prefab again and use Add Component to add the DestroyObject script. Then set the Destroy Time to 1.5. This represents 1.5 seconds. You will notice that because you changed the property it will highlight that property name in blue to show you it is modified from any default. Note that this will have also added the script to the Green Sphere Variant object as well.



58. Run the game again and observe that the balls now appear and then disappear after 1.5 seconds.

## 8 Extra Content

You will find an additional extension to the example in the ExtraContent folder. This modifies the content to turn it into a minigame. The aim of this game is simply to spawn spheres with A, remove the green ones by clicking on them, and once you have 10 red spheres you win.

The ExtraContentScene contains:

- Modified BallSpawner object using a new “Limited Ball Spawner” script where most of the work is being done.
- Two simple cubes stretched out to act as boundaries, so the spheres don’t fall off.
- A WinText object that is some 3D text set to not active (hidden) by default.

The folder for the ExtraContent also contains:

- Ball Prefab and Ball Variant Prefab. These are both constructed almost the same as the Sphere and Green Sphere Variant were. They do NOT have the DestroyObject script on them. Instead, they have a ClickToDestroy script. They also have a Tag set to “Ball” that can be seen at the top of the inspector.
- ClickToDestroy Script: This script simply checks for when the mouse presses on the attached object and then adds the DestroyObject script set to 1 second destroy time.
- LimitedBallSpawner Script: Adds a lot of functionality to restrict the number of elements spawned, move positions where the spawning is happening, searches for objects with a tag and checks their material names to verify they are all the same material. Then checks for the victory condition and reveals the object if the game is won.

The code for each of these two scripts follow.

```
public class ClickToDestroy : MonoBehaviour
{
    // Start is called before the first frame update
    @ Unity Message | 0 references
    void Start()
    {
        ...
    }

    // Update is called once per frame
    @ Unity Message | 0 references
    void Update()
    {
        ...
    }

    // This method is called when the sphere object this is attached to is clicked.
    @ Unity Message | 0 references
    void OnMouseDown()
    {
        // Create a DestroyObject object, attach it to the current game object and use the reference to the newly
        // created script to then set the time until the object is destroyed using the other script.
        DestroyObject scriptRef = gameObject.AddComponent<DestroyObject>();
        scriptRef.destroyTime = 1;
    }
}
```

```

@ Unity Script (1 asset reference) | 0 references
public class LimitedBallSpawner : MonoBehaviour
{
    // List of multiple balls to spawn randomly from
    public List<GameObject> balls;
    // Position where the spawner first started
    public Vector3 startPos;
    // The offset on the X axis from the start position
    public float offset;
    // Reference to the object to enable when the game has been won
    public GameObject winObject;
    // The material that has to be on all objects to win
    public Material winningMaterial;
    // The maximum number of balls that can be spawned.
    public int spawnLimit;

    // Start is called before the first frame update
    @ Unity Message | 0 references
    void Start()
    {
        startPos = transform.position;
        offset = 0;
    }

    // Update is called once per frame
    @ Unity Message | 0 references
    void Update()
    {
        // When the A key is Pressed Down
        if (Input.GetKeyDown(KeyCode.A))
        {
            // Search for all the objects with a specific tag and count the number of them to limit spawns.
            if (GameObject.FindGameObjectsWithTag("Ball").Length >= spawnLimit)
            {
                return;
            }

            // Spawn a random ball from the balls array.
            Instantiate(balls[Random.Range(0, balls.Count)], transform.position, Quaternion.identity);

            // Move the spawner along on the x-axis.
            offset += 1.5f;
            if (offset > 3)
            {
                offset = -1;
            }
            transform.position = new Vector3(startPos.x + offset, startPos.y + startPos.z);

            // Check if all the balls contain the winning material.
            int countUsingMaterial = 0;
            GameObject[] allBalls = GameObject.FindGameObjectsWithTag("Ball");
            foreach (GameObject ball in allBalls)
            {
                // Necessary to use "StartsWith" because the prefab spawned elements have (instance) at the end.
                if (ball.GetComponent<MeshRenderer>().material.name.StartsWith(winningMaterial.name))
                {
                    countUsingMaterial++;
                }
            }

            // If the max limit are all that colour show the win message.
            if (countUsingMaterial == spawnLimit)
            {
                winObject.SetActive(true);
            }
        }
    }
}

```