Unity Engine Video Game Fundamentals

Introduction to the basic Unit Engine UI and game creation through the creation of a simple 2D game.

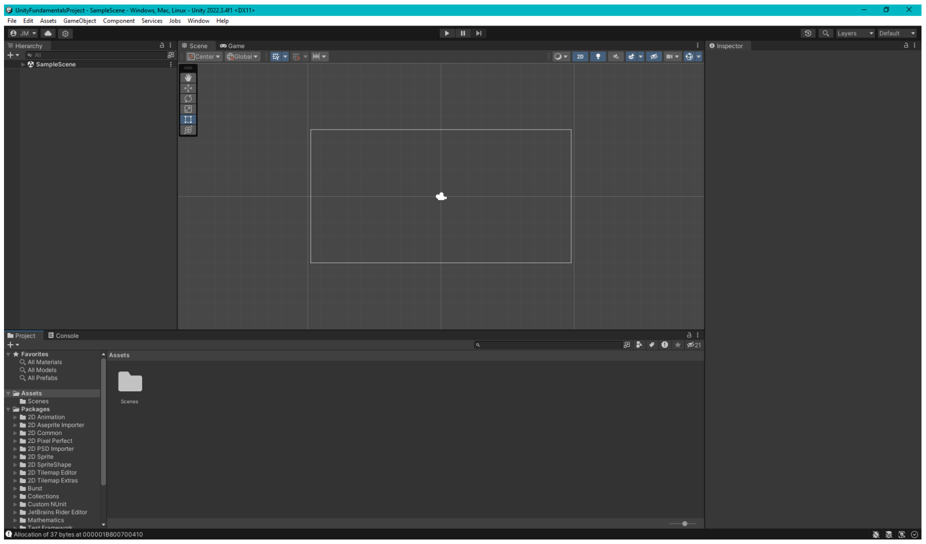
Jack Morgan
Technical Officer

Faculty of Science and Engineering
Manchester Metropolitan University
31st July 2024

Contents

1 Introduction

Welcome to the Unity Engine Video Game Fundamentals session! In this session I will be going over the basics of the Unity Engine without teaching any programming.



Allocation of 37 bytes at 0000018800700410

Figure 1: The default layout of the Unity editor.

This will involve an introduction to the basic Unit Engine UI and game creation through the creation of a simple 2D game. The activity will focus on the UI and using the editor itself, rather than teaching programming concepts. I have provided scripts and resources to everyone to allow for a “Lego Bricks” style experience where they won’t be required to create any code or find image resources.

2 Tasks

This document has written information about the different parts of the Unity Engine, but just reading isn't going to get you anywhere so here are some tasks for you to complete! Without further ado here are the tasks:

1. Add an object into the scene to work as a player,
2. Add a second object into the scene to work as a platform,
3. Add collider and rigidbody components onto these objects, how will they interact in the game world now?
4. Turn an object into a player using the PlayerController script, what else appears when you add it?
5. Allow the player object to move, jump, and dash (AD, Spacebar and J are the default keybinds!),
6. Have a look at the names of the other scripts, see if you can figure out what they do and how to use them within your game,
7. Using the assets provided, create a better looking player and platform,
8. Add in a background. How does changing the z value in the transform component affect what you can see when you run the game?
9. Figure out where to place the background and camera follow scripts, these will allow your game to escape the confines of a static camera!
10. Go wild! Add more platforms, add in enemies. Create a level and see if you can beat it fast, and let other people around you have a go!

You will more than likely need to read parts of this document (and ask us for help!) to understand what to do at points but the most important thing in game design is to experiment and find things that work for you. If you're not sure how something works, mess with it and break it!

3 User Interface Overview

3.1 Hierarchy Window

In Figure 2 you can see the hierarchy window. This window is used to show the objects in the current loaded scene. It's through this window that you will order game objects into hierarchies for formatting and to create parent objects. We'll go over why this is useful later when we create our first objects.

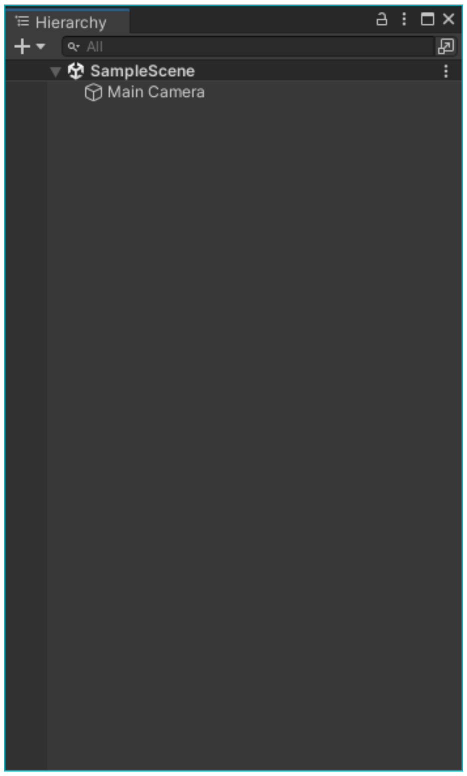


Figure 2: The hierarchy window.

3.2 Project Window

The project window, shown in Figure 3, is the window used to show all of the assets used within the creation of the current game. This can include each of the scenes, prefabs, materials, and more used for creating a video game. Keeping on top of the organisation of your project files is a huge help to maintaining quicker development of games!

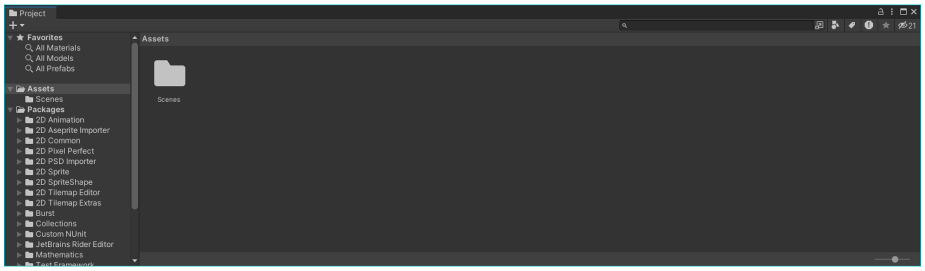


Figure 3: The project window.

3.3 Scene and Game Window

The scene window is where the "magic" happens. This is the window that the developer will create scenes (hence the name) and place down assets. The biggest use of this window is to facilitate level design and therefore this is where most of your time in this session will be spent. This window also has a lot more visual information on it than the game window does, this is to allow you to see exactly what your objects are doing while the game is running. In addition, you use this window to create the user interface of your game which will be explained later.

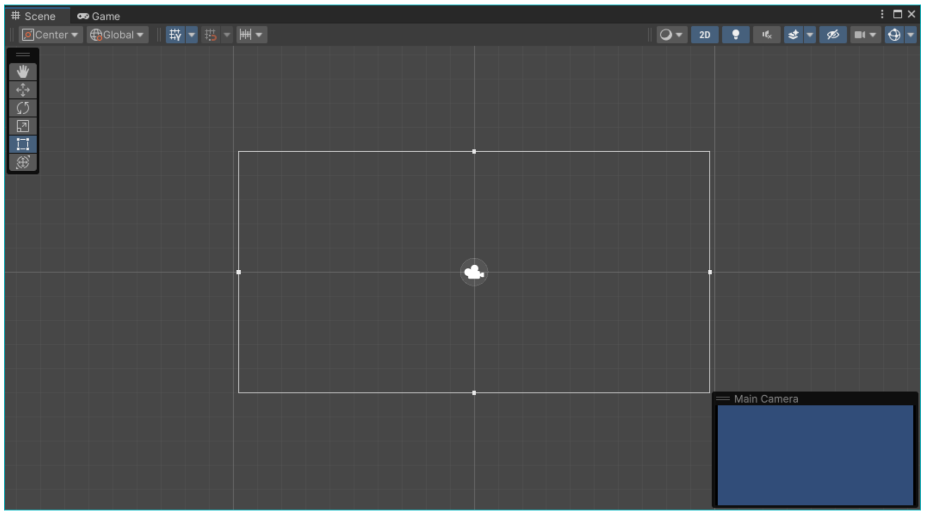


Figure 4: The scene window.

The game window is what allows you to see what your current product looks like in play mode. By pressing the play button (which can be seen in Figure 1) this runs the current iteration of your game within this window without having to build the game every single time. This is extremely useful for creating a game as it allows you to rapidly iterate different design ideas. While we wont be doing any scripting in this session, the game window is invaluable in allowing you to check that the functionality of your programming is correct as they made not always have visual cues.

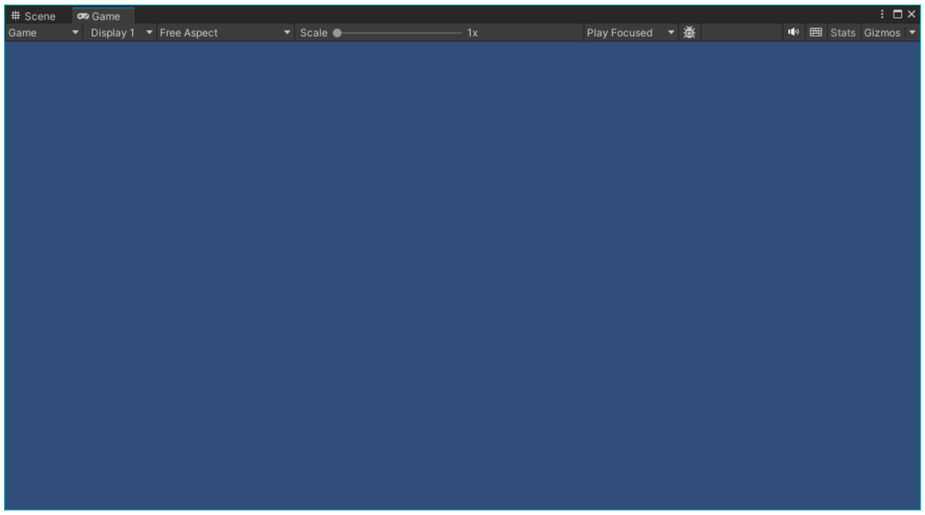
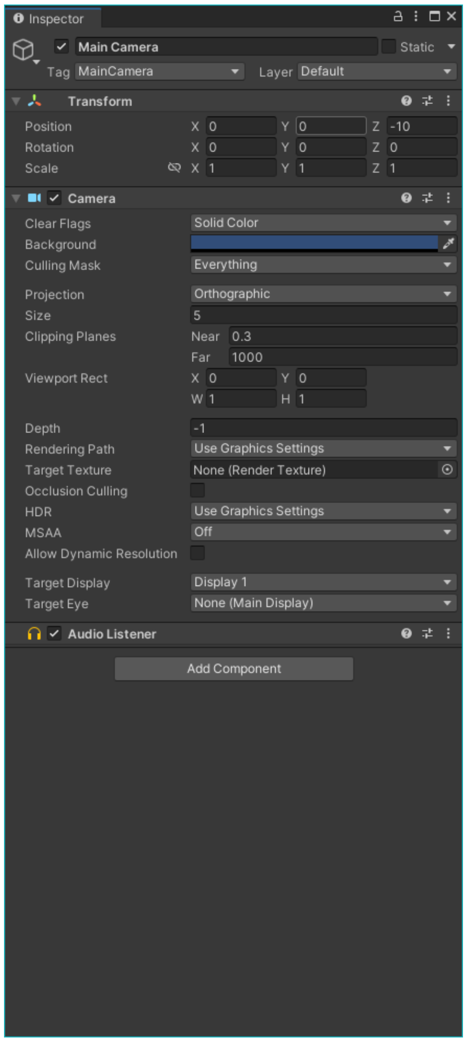


Figure 5: The game window.

These windows together allow you to rapidly change the level and UI design of your game through the scene window and then see how they interact during the actual game through the game window. As a quick note, you can change values and objects while play mode is running (i.e. the game window is running). Any of these changes, however, will be removed once play mode finishes so you should make a note of the changes you make before ending play mode if you want to keep them.

3.4 The Inspector

The inspector is what can be used to see what components are linked to an object. Each component is attached to an object to do a job. In Figure 6 you can see three components, a Transform component, a Camera component, and the Audio Listener component.



Inspector

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	Solid Color
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 Graphics Settings
Target Texture	None (Render Texture)
Occlusion Culling	
HDR	Use Graphics Settings
MSAA	Off
Allow Dynamic Resolution	
Target Display	Display 1
Target Eye	None (Main Display)

Audio Listener

Add Component

Figure 6: The inspector window.

Each of these components handle different tasks for the object and their sum comes together to become the object within the scene. This is where we can change values within scripts and inbuilt Unity components to make the object do the job that is desired.

3.5 The Console

The console window is shown in Figure 7. This window wont be used very much during this session as it mostly shows information from the code base (such as errors, warnings, and debug statements). Some information from the editor is shown however, so it's useful to keep an eye on it. You will sometimes get warnings around having multiple audio listeners or other

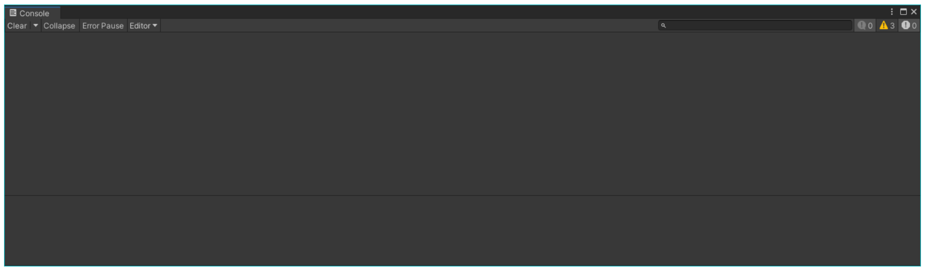


Figure 7: The console window.

4 Game Objects

4.1 Adding an Object

Adding an object in Unity is simple. First head to the hierarchy, then either click the plus in the top left hand corner (as shown in the figure below) or right click anywhere within the hierarchy. For this example it is easiest to choose a 2D square as we are creating a 2D platformer.

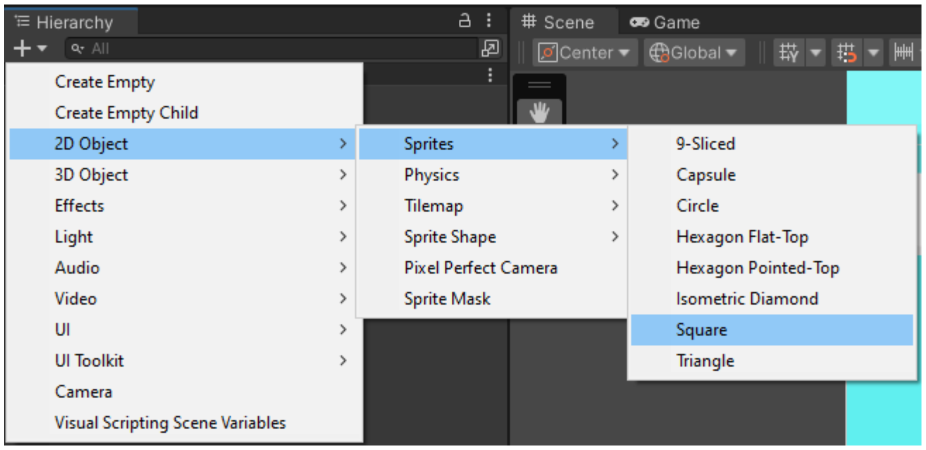
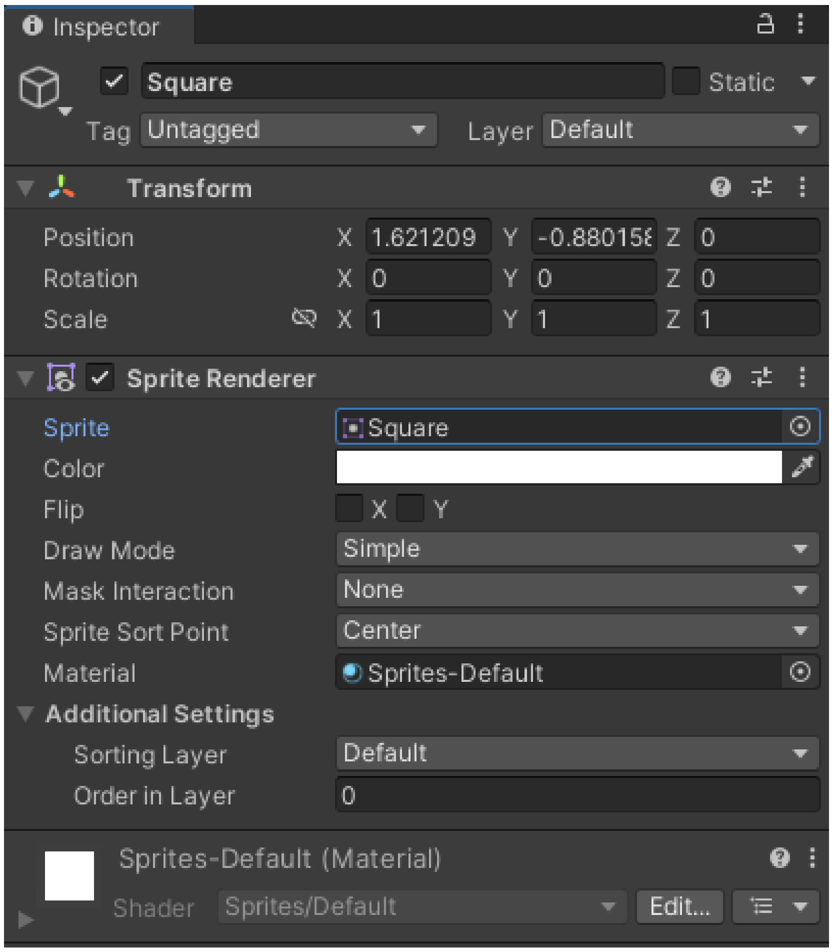


Figure 8: An example of how to add an object within the Unity editor.

4.2 Objects in the Inspector

In Unity2D objects have two basic components when created as above. The transform component handles the position of the object within space. In 2D this means using an X and Y value for left/right and up/down, along with a Z value for positioning towards the camera. This means that a higher Z value positions the objects further from the camera, and a lower Z value is closer. This allows you to control what objects appear in front of other objects.


Inspector

Square — Static

Tag: Untagged — Layer: Default

Transform

	X	Y	Z
Position	1.621209	-0.88015ε	0
Rotation	0	0	0
Scale	1	1	1

Sprite Renderer

Sprite	Square
Color	
Flip	X Y
Draw Mode	Simple
Mask Interaction	None
Sprite Sort Point	Center
Material	Sprites-Default

Additional Settings

Sorting Layer	Default
Order in Layer	0

Sprites-Default (Material)

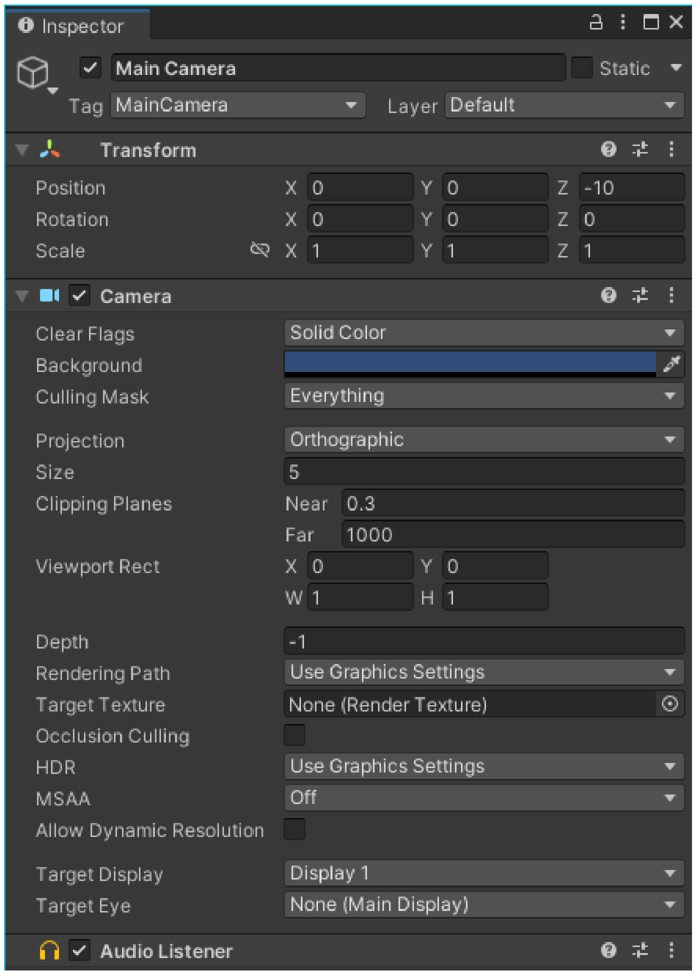
Shader: Sprites/Default — Edit...

Figure 9: A basic square object inspector window.

The renderer component (in this case a Sprite renderer) handles how the object looks within the game. In the case of a basic square the sprite is an inbuilt Unity sprite of a square. This sprite is normally a picture file with transparency such as a PNG. These components can be seen in the figure above.

5 The Camera Eye

In your empty project you should be able to see an object called Main Camera. The camera is the object that controls what your player can see during the game. In this game the camera will be quite simple, but in more complicated games you can require multiple cameras.



✓ 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	Solid Color
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 Graphics Settings
Target Texture	None (Render Texture)
Occlusion Culling	☐
HDR	Use Graphics Settings
MSAA	Off
Allow Dynamic Resolution	☐
Target Display	Display 1
Target Eye	None (Main Display)

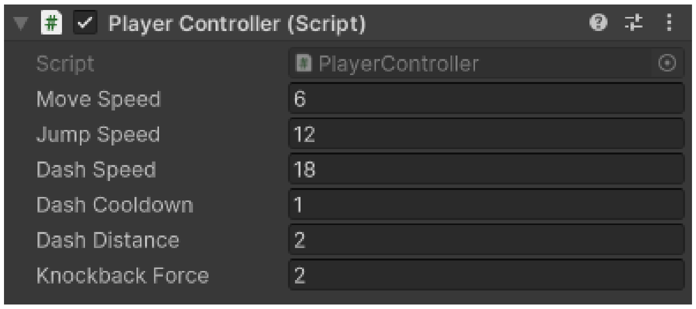
✓ **Audio Listener**

Figure 10: The default Main Camera components .

When you click the camera object within the scene hierarchy, you'll notice that inspector has, in addition to the usual transform, a Camera component. This component is what turns an object into a Unity camera. For our purposes we'll keep the default settings as they work very well for a basic 2D platformer.

6 Scripts

Scripts are Unity's way of implementing programming. Generally done in the programming language C# they can be used to create custom interactions within the game. For our purposes today, I have created 5 basic C# scripts.



Script	PlayerController
Move Speed	6
Jump Speed	12
Dash Speed	18
Dash Cooldown	1
Dash Distance	2
Knockback Force	2

Figure 11: The player controller script.

The script above named PlayerController handles... the player! This is the script that will be placed onto the player object and contains variables visible to the inspector that allow you to change how the character interacts. The other four scripts I will leave for you to figure out how to place.

7 Collision and Physics

As a game engine, Unity has inbuilt methods of dealing with collisions between objects and physics. The most common way of dealing with these concepts is using their inbuilt Collider and RigidBody components. These components can be accessed through selecting Add Component and searching for them as shown in the figure below. As we are creating a 2D game, you will want to select a 2D collider that is relevant for the object. The collider does not have to be exact to the shape, but at least a good approximation of it.

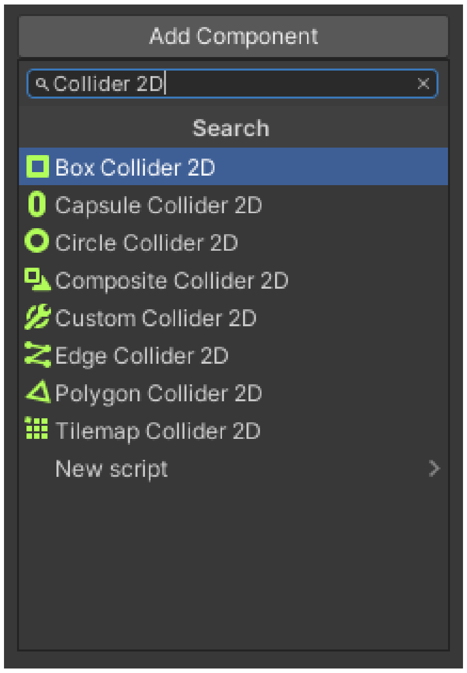


Figure 12: Searching for a collider 2D component

The rigidbody component is what allows the player to be affected by physics. The most basic example of this is once the rigidbody is added to an object it will be affected by gravity! This component is also what allows an object to detect collisions. Without this component objects will pass through each other.

List of Figures