COMP1150/MMCC1011 Week 3 Prac

Topics covered:

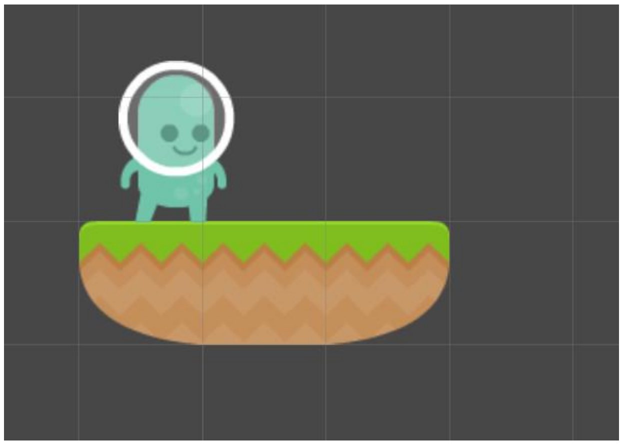
- Adding a playable character
- Rigidbody2D
- Colliders
- Layers
- Triggers
- Prefabs

For the next few weeks, we will be transforming the platformer scene that you started to make in the first two pracs into a small game. Feel free to make small adjustments to your scene in-between classes (such as adding new decorations, platforms, etc).

The outcome of today's prac will be a scene featuring a playable character that can move around on platforms and coins that the player can collect.

Adding a Playable Character

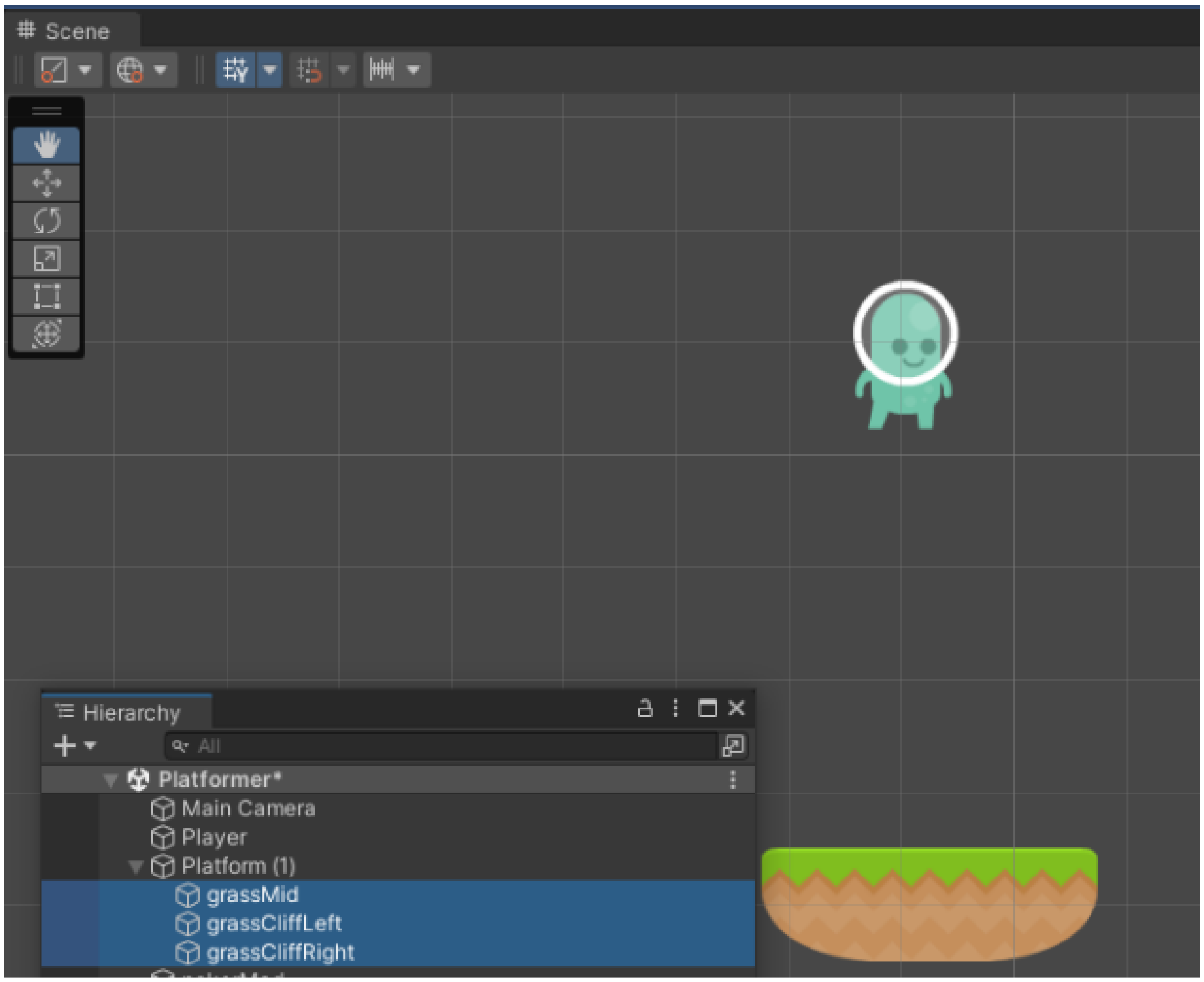
Add one of the player sprites to your scene from the platformerGraphicsDeluxe folder if you have not done so already. I'm using the green one:



Before we make the player move and jump around, it's important to understand the basic physics needed to make this work in Unity. To help demonstrate/verify an object has physics implemented, it's sometimes worth doing a simple test. In this case, we're going to verify the player has physics implemented by testing its gravity. If implemented properly, the affected object (in this case our player character) will:

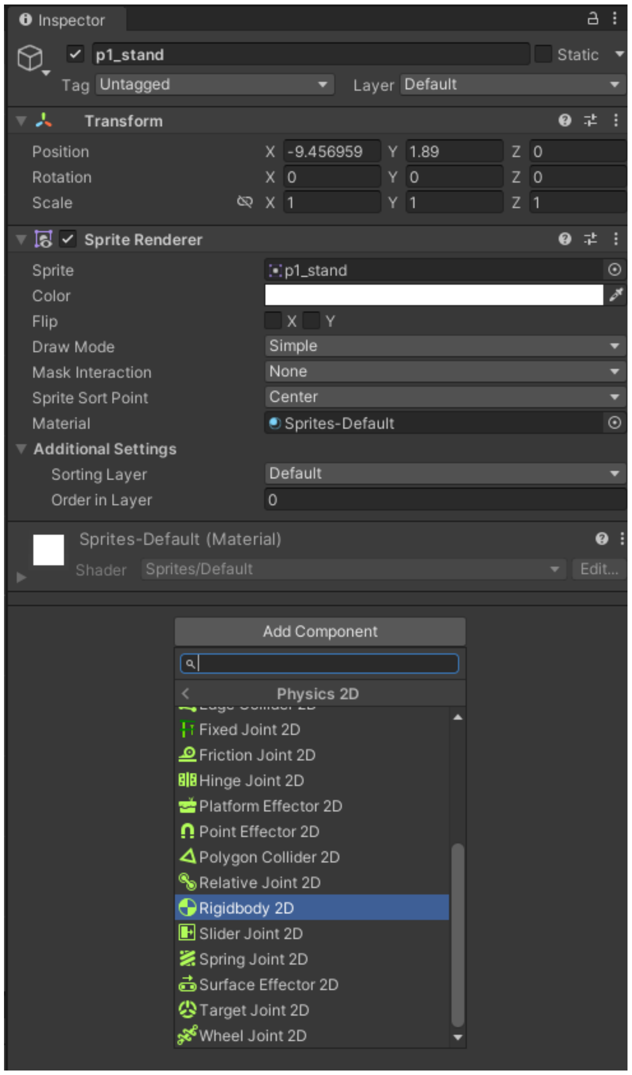
1. Fall out of the sky, towards the bottom of the screen/in the direction gravity is pulling
2. Collide with other objects in the scene (in this case platforms) and stop moving.

Let’s tackle the first part of this test. To the side of your level, make a new platform. Make sure you group the three platform sprites into a single platform object in the Hierarchy using an empty object as the parent (as explained in the week 1 prac). Move your player character sprite above the platform as shown, and your Main Camera so that it is centred on this part of the scene.



To allow the player to be affected by physics (e.g. gravity), we will need to give it a ‘rigidbody’. There are two kinds: **Rigidbody2D** for 2D games using sprites, and **Rigidbody** for 3D games using models. We’ll be using the 2D version.

Select the player and press the **Add Component** button in the Inspector panel. Select **Physics 2D > Rigidbody 2D**.



p1_stand Static
Tag Untagged Layer Default

Transform

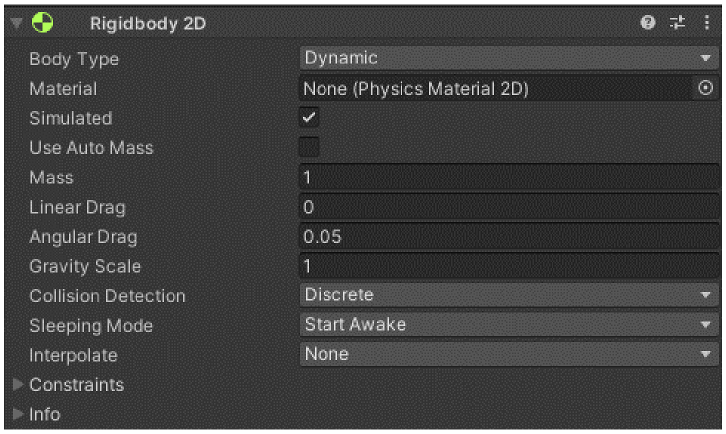
Position	X -9.456959	Y 1.89	Z 0
Rotation	X 0	Y 0	Z 0
Scale	X 1	Y 1	Z 1

Sprite Renderer

Sprite	p1_stand
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...

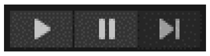
A new component should appear in the Player object, called a **Rigidbody 2D**. Leave the settings as their default for now.



Rigidbody 2D

Body Type	Dynamic
Material	None (Physics Material 2D)
Simulated	✓
Use Auto Mass	
Mass	1
Linear Drag	0
Angular Drag	0.05
Gravity Scale	1
Collision Detection	Discrete
Sleeping Mode	Start Awake
Interpolate	None
Constraints	
Info	

To see what this rigidbody does, press the **Play button** on the toolbar (located at the top middle of the editor window).



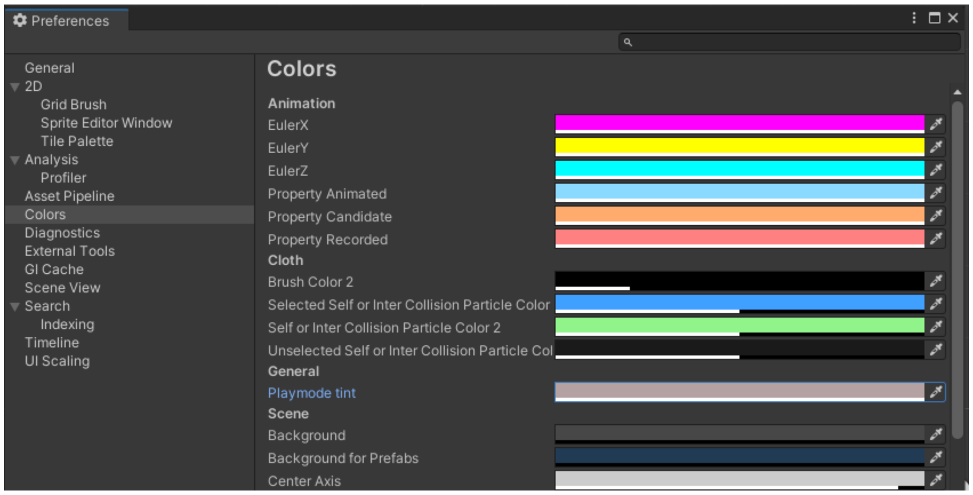
This allows you to run your game within the editor. You should see the player quickly fall off screen, controlled by gravity. However, it will fall straight through the platform because we haven't told Unity that the Player and the platform are solid objects, yet.

It is also worth pointing out that you can examine objects in the Scene and Inspector views while the game is running. Select the Player from the Hierarchy view and look at its Transform value in the Inspector. You should see the Y value decreasing, as the Player keeps falling.

You can stop the game by pressing the Play button again.

Important: All changes you make while 'Playmode' is active will be lost once it is stopped. You should always stop Playmode after you finish testing your Scene.

Additionally, you can alter the Unity settings to change the colour of the Unity window when the Scene is being Played, just to make it easier to determine if you are in Play mode or not. To do this press **Edit > Preferences** (on PC, or Unity > Preferences on Mac) then in the popup window go to the **Colors** tab and alter the colour of the **Playmode Tint**.



Preferences

General
2D
Grid Brush
Sprite Editor Window
Tile Palette
Analysis
Profiler
Asset Pipeline
Colors
Diagnostics
External Tools
GI Cache
Scene View
Search
Indexing
Timeline
UI Scaling

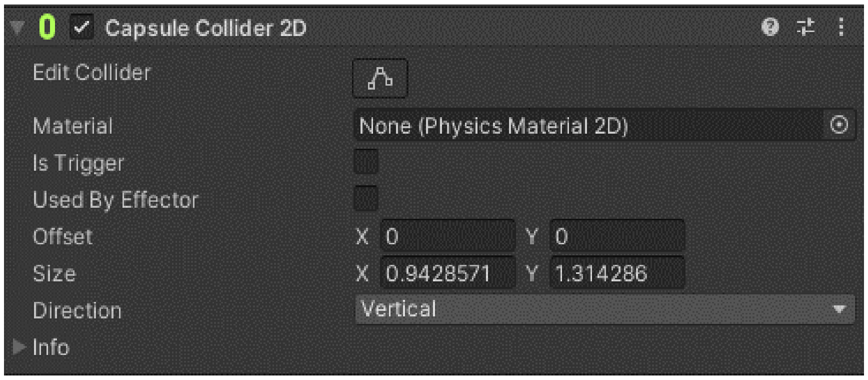
Colors

Animation
EulerX
EulerY
EulerZ
Property Animated
Property Candidate
Property Recorded
Cloth
Brush Color 2
Selected Self or Inter Collision Particle Color
Self or Inter Collision Particle Color 2
Unselected Self or Inter Collision Particle Col
General
Playmode tint
Scene
Background
Background for Prefabs
Center Axis

Making Things Collide

To make objects collide with each other (step two of our test) we need to add components to them called **Colliders**. There are various kinds of colliders available depending on the shape of your object. We will look at three: 'CapsuleColliders', 'BoxColliders', and 'PolygonColliders'.

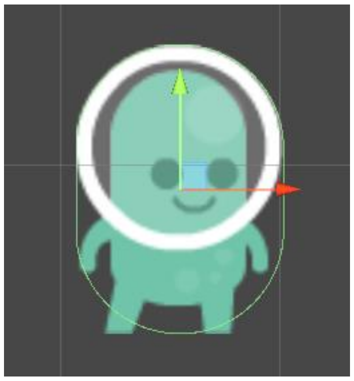
Select the Player and press the **Add Component** button. Select **Capsule Collider 2D** from the 'Physics 2D' menu. You should see a new component as shown below.



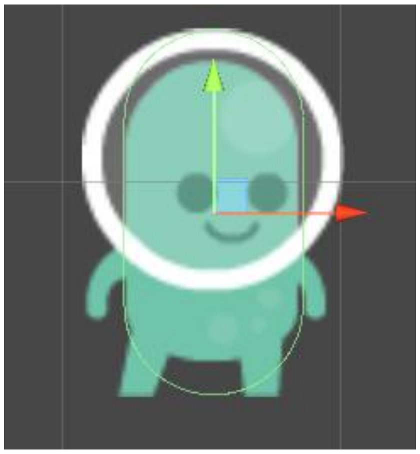
Capsule Collider 2D

Edit Collider	
Material	None (Physics Material 2D)
Is Trigger	☐
Used By Effector	☐
Offset	X 0 Y 0
Size	X 0.9428571 Y 1.314286
Direction	Vertical
Info	

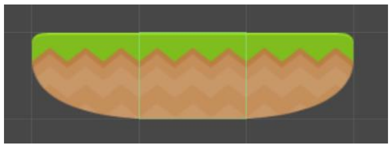
In the Scene view, the player will now have a green circle around it, indicating the shape of the collider. Initially, it will be bigger than the size of the player.



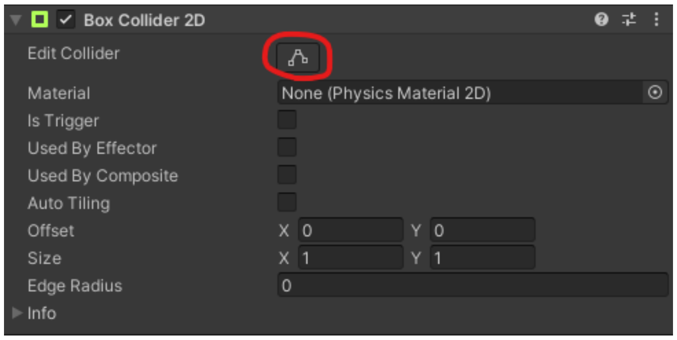
Change the **Size** of the 'CapsuleColider2D' until fits snugly around the body of the player as shown below. We still want it to cover the players feet (so it can touch the ground), but we want to ignore the arms and the helmet (for reference, the size for the collider shown below are X: 0.65 Y: 1.3). It does not need to be perfect though!



Now that we’ve set the bounds for the collider attached to our player character, we need to give it something to collide with. Select the platform object (the parent GameObject, **not** the sprites) and add a **Box Collider 2D** component from the Physics2D list. ‘Box Collider 2D’ is a simple collider in the shape of a box. The collider should appear inside the platform as shown.

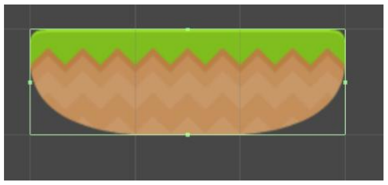


By default, the box will likely be too small. You can make it bigger either by changing the **Size** field, or by pressing the **Edit Collider** button and stretching the box manually.

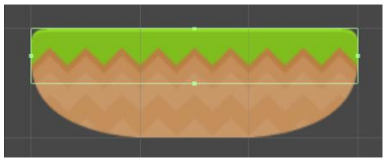


Box Collider 2D	
Edit Collider	
Material	None (Physics Material 2D)
Is Trigger	
Used By Effector	
Used By Composite	
Auto Tiling	
Offset	X 0 Y 0
Size	X 1 Y 1
Edge Radius	0
Info	

The next question is how big to make the collider. If we make it too big, then objects will collide with the blank space outside the sprite.



If we make it too small, then objects will be able to overlap with the parts of the platform that are not covered by the collider.



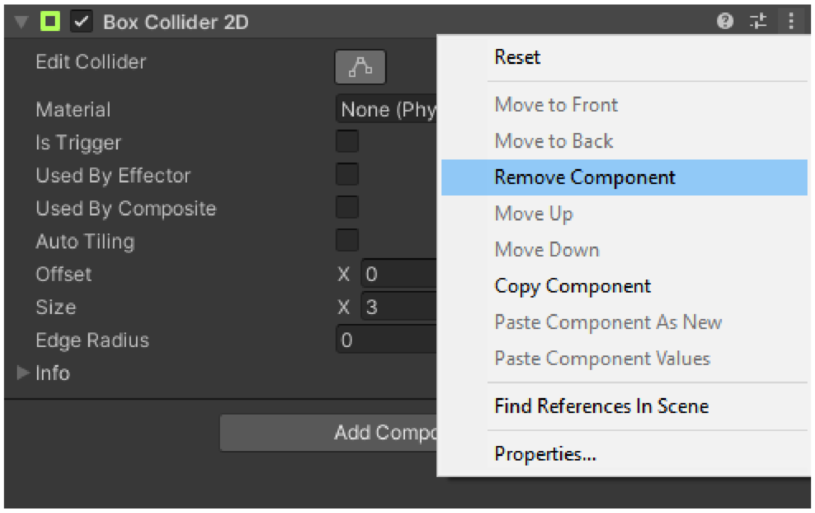
Ultimately this depends on what role the object would play in the game. If you play the scene and the let the player fall on the box collider you will see it is functional. That being said, it’s worth keeping in mind that physics simulations can never be perfect if the collider is not aligned with the sprite.

Advice: In general, it is a good idea to make colliders the player wants to hit **big** and colliders they don’t want to hit **small**. Otherwise the player will complain that the game is being unfair.

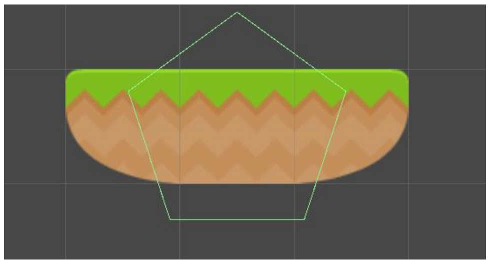
To make our collider better match the platform we could instead use a **Polygon Collider 2D**, which provides more control over its shape.

Advice: It is generally a good idea to use the simplest collider you can get away with for your purpose in the game, since more complex colliders will be more computationally heavy, potentially affecting performance. Collisions often don't need to be pixel-perfect. If it fits the shape of its matching sprite well enough, then a box or capsule collider that is slightly bigger or smaller is better for performance than a polygon collider which fits exactly.

There are three vertical dots in the top-right corner of the Box Collider 2D component in the Inspector. Click on this to get a drop-down menu, and select **Remove Component** to remove the Box Collider.

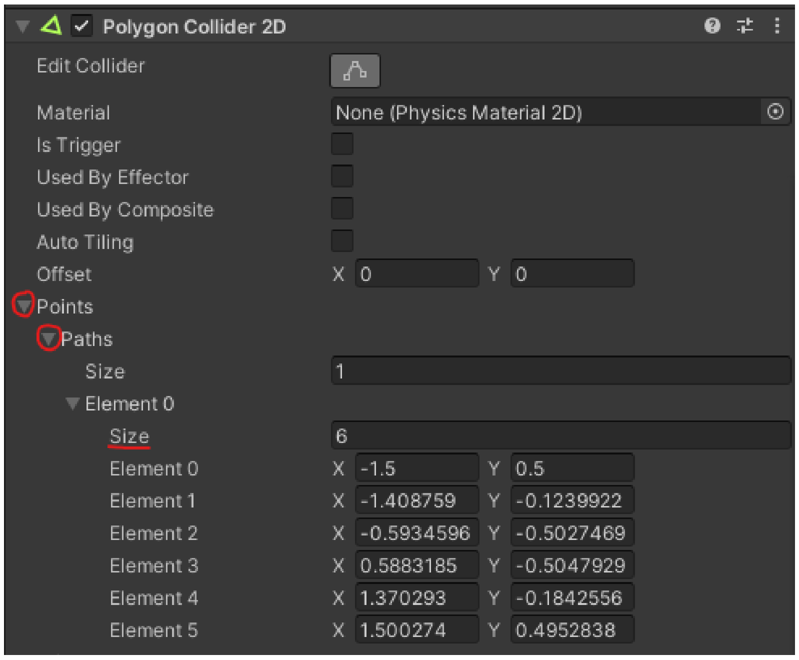

Box Collider 2D
Edit Collider
Material None (Phy
Is Trigger
Used By Effector
Used By Composite
Auto Tiling
Offset X 0
Size X 3
Edge Radius 0
Info

Now add a **Polygon Collider 2D** in its place, which will appear as a pentagon in your scene view.



Press the **Edit Collider** button to edit the points in the collider polygon so it fits closely around the platform. Add extra points by clicking in the middle of an edge. You should be able to make a tight fit with just six points.

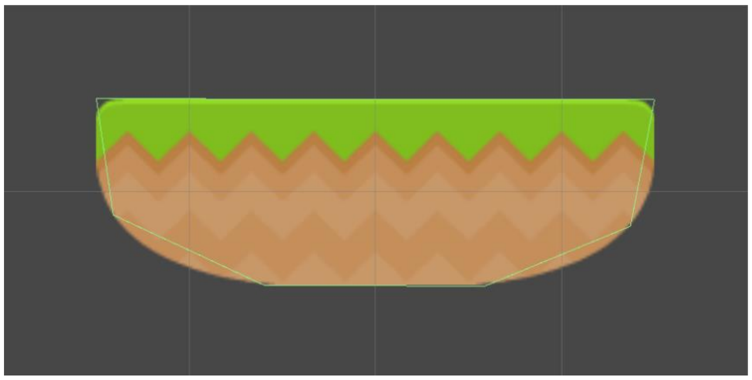
You can check how many points you have by clicking **Points**, then **Paths**, as seen below – In this context, ‘Size’ refers to the amount of points the collider has.



Polygon Collider 2D

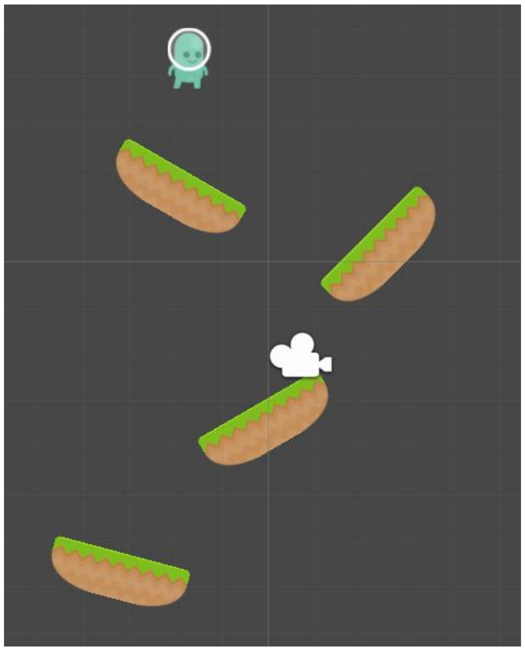
Edit Collider	
Material	None (Physics Material 2D)
Is Trigger	☐
Used By Effector	☐
Used By Composite	☐
Auto Tiling	☐
Offset	X 0 Y 0
Points	
Paths	
Size	1
Element 0	
Size	6
Element 0	X -1.5 Y 0.5
Element 1	X -1.408759 Y -0.1239922
Element 2	X -0.5934596 Y -0.5027469
Element 3	X 0.5883185 Y -0.5047929
Element 4	X 1.370293 Y -0.1842556
Element 5	X 1.500274 Y 0.4952838

This is a good collider. It fits the shape well without too many points.



Now that you’ve added colliders to both the player character and the platform, press **Play** again. The player character should fall and land on the platform. Try rotating the platform and making a few duplicates. Notice that the collider rotates with the platform object and is duplicated when you copy it.

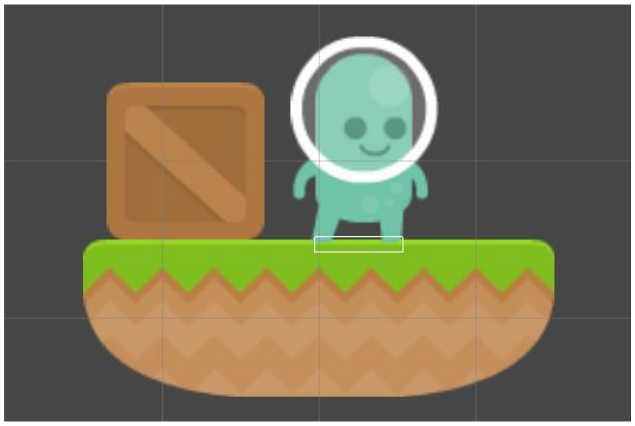
Create an obstacle course for our player to slide around on, like the one pictured below, and play it to see what happens.



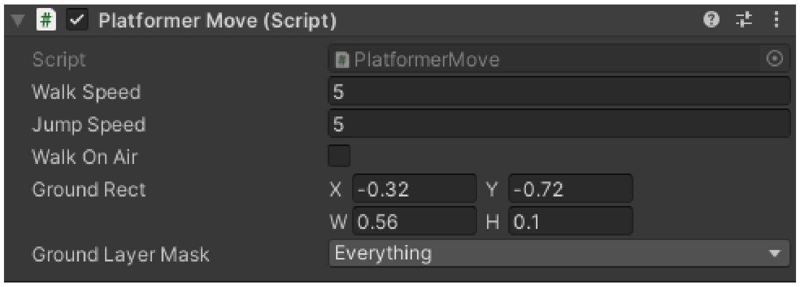
Try adding a box to your scene that will interact with your obstacle course when it falls. You'll need to configure it with a rigidbody and a collider. What components do you need? What is the best collider to use?

Controlling the Player

In the **Scripts** folder of your project, find a script called **PlatformerMove.cs**. This is a little script we've written to implement basic platformer-style controls for running and jumping. Add this script to your playable character object (**Add Component > Scripts > PlatformerMove**) You should see a little white box-gizmo appear in the scene view at the feet of your player sprite:

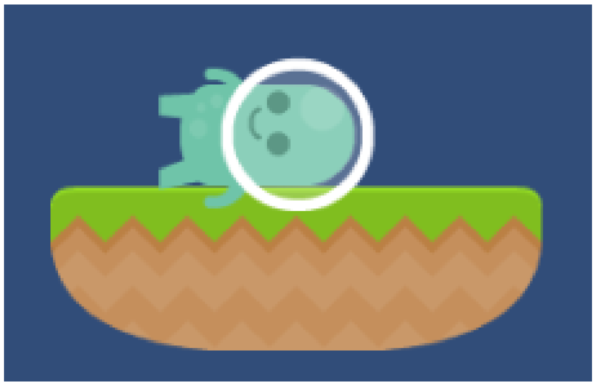


This is the **Ground Rect** - a box the script uses to detect if there is anything under the player's feet. You can adjust the size and position of this rectangle in the Inspector by changing the **Ground Rect** values in the **PlatformerMove** component. The default position should be fine, but make sure it lines up with your sprite's feet and sticks out a little bit below the sprite, as shown above.

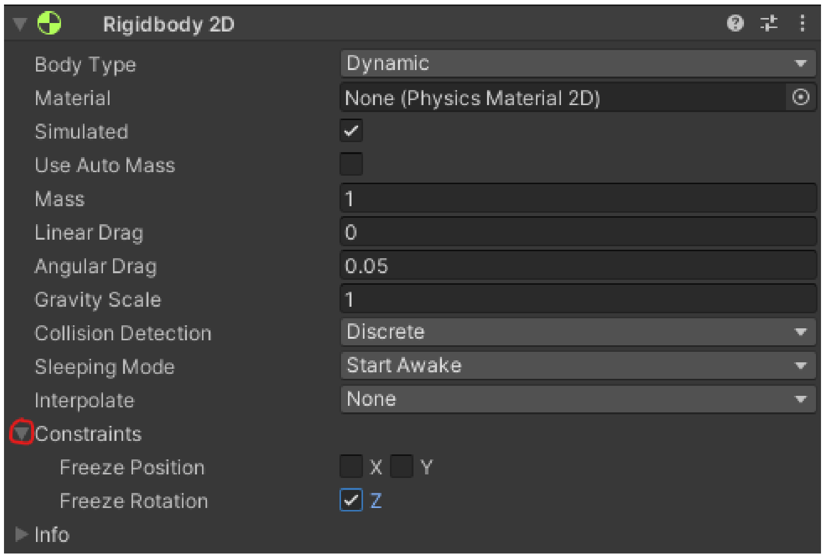


Platformer Move (Script)	
Script	PlatformerMove
Walk Speed	5
Jump Speed	5
Walk On Air	☐
Ground Rect	X -0.32 Y -0.72
	W 0.56 H 0.1
Ground Layer Mask	Everything

If you press play now, you should be able to make the sprite move using the left and right arrow keys (or A and D). However, you will find that the sprite will prefer to lie down instead of moving around.



Unlike other physical objects, we don't want the player character to rotate. We can prevent this by adding a constraint to its 'Rigidbody2D'. In the Inspector, open the **Constraints** tab at the bottom of the player character's 'Rigidbody2D' and tick the **Freeze Rotation Z** checkbox:



Rigidbody 2D	
Body Type	Dynamic
Material	None (Physics Material 2D)
Simulated	✓
Use Auto Mass	☐
Mass	1
Linear Drag	0
Angular Drag	0.05
Gravity Scale	1
Collision Detection	Discrete
Sleeping Mode	Start Awake
Interpolate	None
Constraints	
Freeze Position	☐ X ☐ Y
Freeze Rotation	✓ Z
Info	

Play the game again. You should find that the player character no longer rotates.

Reminder: Save, Commit and Push to GitHub

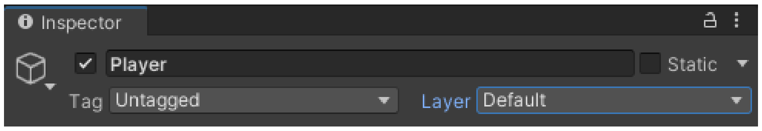
Layers

While playing the game, press **Spacebar** to jump. You may find that the player character can jump even if they are already in the air. If you look at the player character in the Scene view, you will see that the 'Ground Rect' box has turned red. This is supposed to indicate that the player character is on the ground, but it seems to be red all the time.

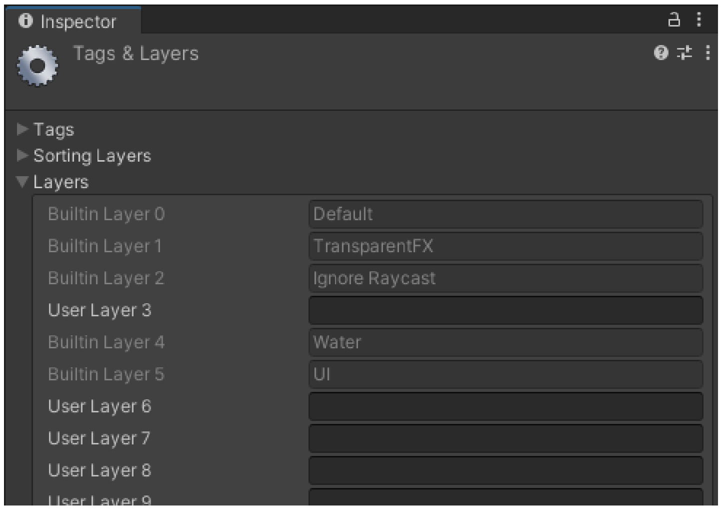
The reason for this is that the 'Ground Rect' is intersecting with the player's own collider, and the script is not smart enough to recognise that the player cannot stand on itself. We could solve this by making sure the 'Ground Rect' and the collider don't overlap, but a more general solution is to simply move the player to a separate layer.

'Layers' are used to sort game objects into groups. We can then create rules which prevent objects in one layer from colliding with objects in another layer. In this case, we want to move the player to its own layer and tell the 'PlatformerMove' component to ignore objects in the player's layer.

Select the player character in the Scene view. At the top of the Inspector you should see two drop-down lists marked **Tag** and **Layer** which should be set to Untagged and Default respectively.

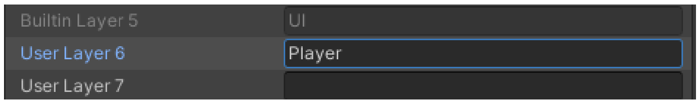


Click the Layer drop-down and select **Add Layer…** You will see a list of layers:


Inspector
Tags & Layers
Tags
Sorting Layers
Layers

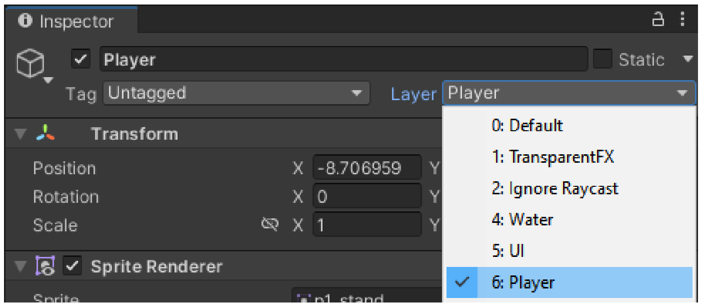
Builtin Layer 0	Default
Builtin Layer 1	TransparentFX
Builtin Layer 2	Ignore Raycast
User Layer 3	
Builtin Layer 4	Water
Builtin Layer 5	UI
User Layer 6	
User Layer 7	
User Layer 8	
User Layer 9	

We want to create a custom layer for the player sprite. In the box next to **User Layer 6** type **Player**.



Builtin Layer 5	UI
User Layer 6	Player
User Layer 7	

Now return to the player character in the Inspector, click on the Layer box again and select the new **Player** layer.



Inspector

Player — Static

Tag: Untagged — Layer: Player

- 0: Default
- 1: TransparentFX
- 2: Ignore Raycast
- 4: Water
- 5: UI
- ✓ 6: Player

Transform

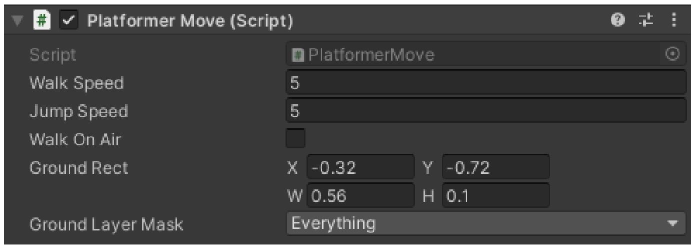
Position X -8.706959

Rotation X 0

Scale X 1

Sprite Renderer

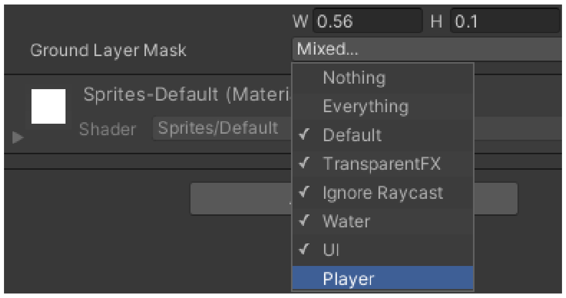
The player character is now in a separate layer to the other objects in the scene. By default, objects in all layers are set to interact with each other. To fix the jumping bug, we need to tell the ‘PlatformerMove’ component to exclude objects in the ‘Player’ layer. In the ‘PlatformerMove’ component, you should see a field marked **Ground Layer Mask** which is set to ‘Everything’.



Platformer Move (Script)

Script	PlatformerMove
Walk Speed	5
Jump Speed	5
Walk On Air	☐
Ground Rect	X -0.32 Y -0.72 W 0.56 H 0.1
Ground Layer Mask	Everything

Click on this box and deselect Player from the list of layers. The ‘Ground Layer Mask’ will now change to ‘Mixed…’, as shown below.



W 0.56 H 0.1

Ground Layer Mask: Mixed...

- Nothing
- Everything
- ✓ Default
- ✓ TransparentFX
- ✓ Ignore Raycast
- ✓ Water
- ✓ UI
- Player

The script will now know to ignore the player character when determining if the player character can jump. Play the game and test this out.

Other Parameters

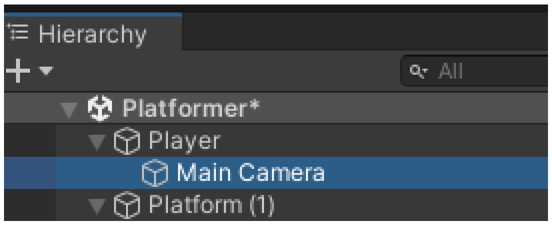
Now that the player is moving and jumping around, try adjusting some of the other parameters on the ‘PlatformerMove’ component:

- **Walk Speed** controls how fast the player moves
- **Jump Speed** controls how fast the player leaves the ground when they jump
- **Walk On Air** if checked, allows the player to change direction (left or right) in mid-air.

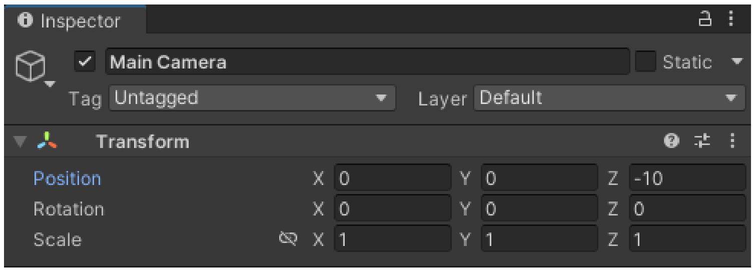
Follow Camera

Just as you can collectively manipulate sprites by making them children of an empty 'GameObject', you can also pair other objects together. One common use of this technique is to make the camera track a particular object.

In the Hierarchy panel, drag the 'Main Camera' onto your player character so that it becomes its child.



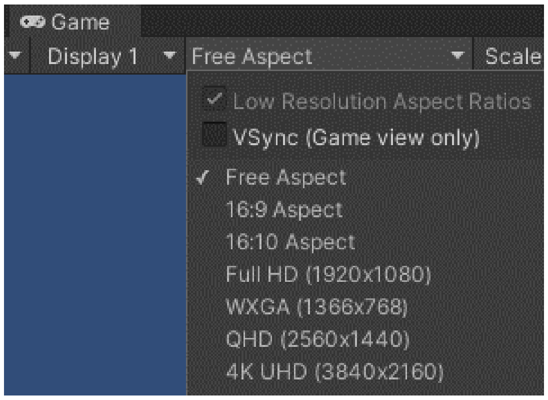
Now select the Main Camera and edit its Transform so that the X and Y position are both zero as shown below (the Z position should remain at -10). This means that the player is in the centre of the camera's view.



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

Now play the game. Because the camera is attached to the player, you should find that they both move around together, which causes the camera to follow the player around the world. You can use this to allow the player to explore worlds which are bigger than a single screen.

As a reminder, you may need to adjust the aspect ratio setting at the top of the Game panel. This is the ratio of screen width to height. By default this is set to **Free Aspect**, which means the aspect ratio always matches to the shape of the 'Game' view. You can change it to a different fixed value by clicking on this box and selecting from the drop-down list.

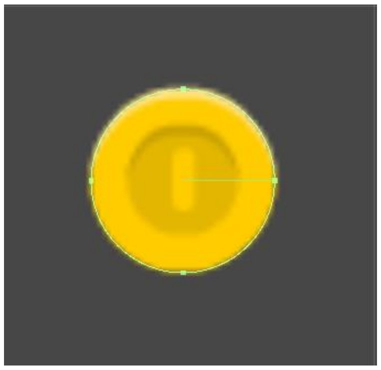


The best ratio will depend on your target platform for your game. If you want to make a standalone full-screen game, the **16:9** option is a good default.

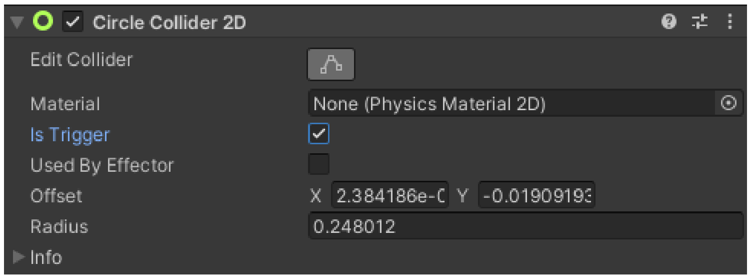
Reminder: Save, Commit and Push to GitHub

Making Collectibles: Triggers

Let's add some collectible coins to the game. Add a coin sprite to your world and attach a **Circle Collider 2D** to it. You may need to adjust the collider to fit it to the sprite (remember, you can do this via the **Edit Collider** button in the inspector).



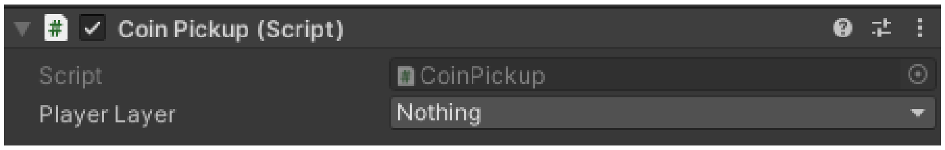
In this case, we don't need the coin to be affected by physics, so it won't need a Rigidbody2D. We also don't want it to be solid like the platforms. Instead, we want objects to pass through the coin but also to detect when the player character touches it. We can achieve this using a 'trigger'. Triggers are a special kind of collider which detects collisions but doesn't react to them physically. Change the 'CircleCollider2D' into a trigger by ticking the **Is Trigger** checkbox in the Inspector.



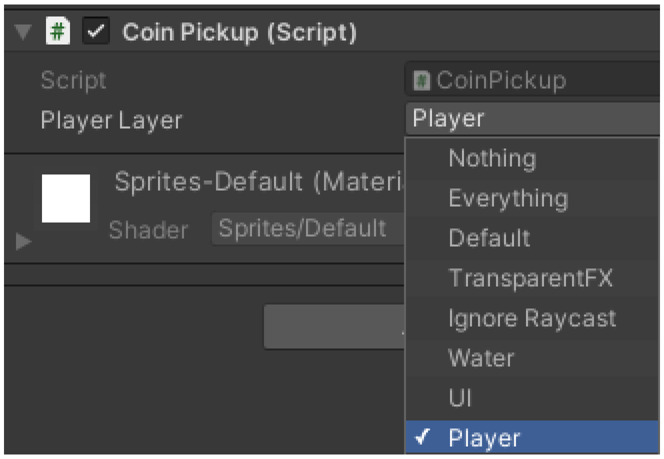
Circle Collider 2D

Edit Collider	
Material	None (Physics Material 2D)
Is Trigger	✓
Used By Effector	
Offset	X 2.384186e-(Y -0.01909193
Radius	0.248012
Info	

If you play the scene now, you will find that the player character (and other objects) can now pass straight through the coin. To make the trigger have an effect we need to add the **CoinPickup** script, which you can find in the Scripts folder. Add this script as a component to the coin object you've just created.


Coin Pickup (Script)
Script CoinPickup
Player Layer Nothing

The script needs to know what layer the player object is in, so that it only responds to the player touching it (and not other objects). Set the **Player Layer** field on the Coin Pickup component to the Player layer you created earlier.


Coin Pickup (Script)
Script CoinPickup
Player Layer Player

Now if you play the game, the coin should disappear when the player touches it.

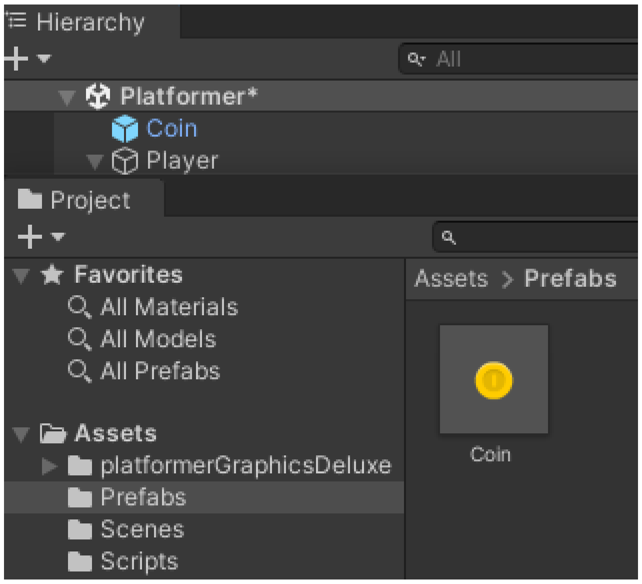
Prefabs

The average platformer contains a lot of coins. It would be very tiresome if we had to create each one using the process just outlined. The good thing is since we've now got one that works, we can just duplicate it many times. We could do this by using Copy/Paste or Duplicate, but then if we wanted to change the settings on all our coins we'd have to go back and change them all one by one.

A better approach is to turn our object into a **prefab**. A prefab is a standard pattern for an object, which can be used to create multiple instances with the same parameters. You can create a prefab from an existing object by simply dragging it from the Hierarchy panel to the Assets folder in the Project panel. It is generally good practice to give your prefabs an exclusive folder. Make a new folder in your Project panel, and call it 'Prefabs'.

Rename your coin object to 'Coin', if you haven't already, and drag your Coin GameObject from the Hierarchy, to your new 'Prefabs' folder. This will make the coin a prefab.

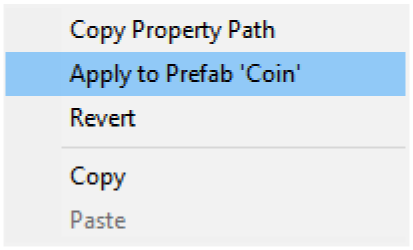
The original coin object in your Hierarchy should change to a blue box to reflect that it is now an instance of the Coin prefab.



You can now drag and drop the prefab into your scene multiple times to create many copies of the coin with the same components and settings as the original. Alternatively, you can Copy/Paste or Duplicate the object in your Scene or Hierarchy panels.

Select the Coin prefab you created in the Project panel. The Inspector will show its components, as with a normal object. However, any changes you make to these settings will affect **all** the instances of the prefab in the scene.

Changes that you make to the instances of your prefabs (e.g. the Coins in your Scene or Hierarchy view) will only affect that instance. To apply changes made to the instance of a prefab to the main prefab itself, you will need to right click on the changed property in the inspector and select **Apply to Prefab <name>**. We will cover this in more detail next week.



Prefabs are very useful when you have a number of identical objects in your game and you want to be able to configure them all at once. We'll be further modifying our Coin prefab next week so that collecting it produces a sound.

Reminder: Save, Commit and Push to GitHub

Working on Your Scene

If you have time, apply the techniques you've learned today to the rest of the objects in your platformer scene. You may also like to continue to develop your scene to make it larger, now that you know how to make the camera follow the player.

Some other things to consider when designing your scene:

- How does the player interact with the physics?
- Can coins be placed to give the player goals?
- Are there any common objects that you could turn into prefabs (e.g. platforms)?

Important: As your scene becomes more complex, with more and more objects, so will your Hierarchy panel. To help keep your scene manageable (and follow professional practice) you should use empty GameObjects to group relevant sets of objects together (e.g. platforms and other decorative elements, collectables like coins, etc.). Once you've finished updating the design of your scene, take some time to clean up your hierarchy.

Reminder: Save, Commit and Push to GitHub

Show your Demonstrator

To demonstrate your understanding of this week's content to your prac demonstrator you might show:

- Your player character followed by the camera
- Your player character with customised player movement physics
- Your player character collecting coins, and that your coins are prefabs
- Additional objects in your scene using rigidbodies and colliders
- Additional objects in your scene set up using prefabs
- That your Project and Hierarchy panels are tidy and well organised
- Your updated GitHub repository