Unity Tutorial : Platform Game

1. Get a player moving across the screen

Open the scene “Level1”. To each of the sprites in the scene add a ‘BoxCollider 2D’ component. To the Player object, add a ‘Rigidbody 2D’ component.

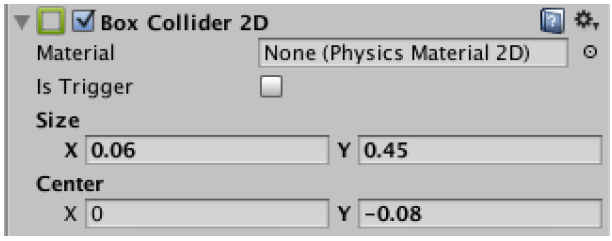
Also to the Player object add a ‘PlayerController’ script.

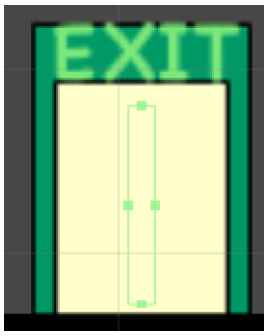
This script is responsible for moving the player left and right across the screen. It does this by checking the horizontal input axis and applying the value to the Players Rigidbodys velocity. The script has one parameter ‘Move Speed’ which affects how fast the player will move.

To the Exit Door object add the similarly named ‘ExitDoor’ script. This script loads a new scene when it is collided with and takes one parameter ‘Scene Name’ For starters set this to “Level2”.

You will most likely want to modify the size of the colliders attached to the player and the exit so that the player sits comfortably on the floor and so that the collision area of the door is small enough so that the next scene is not triggered until the player is midway across the sprite.

Use the size and center options of the Collider component or hold shift and drag the little green squares in the scene view:



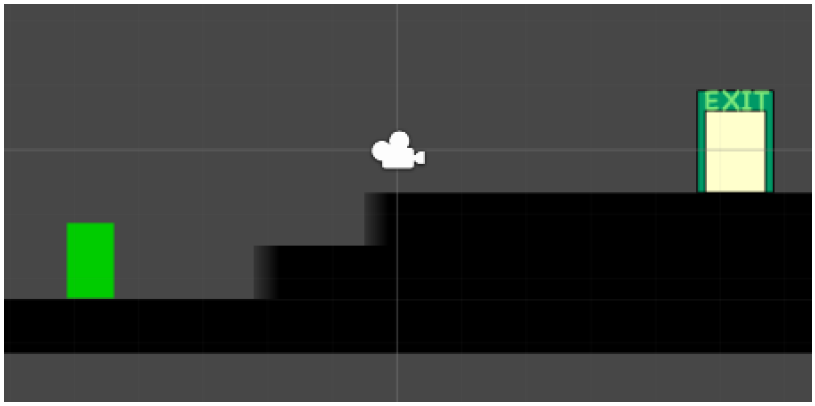


Each of the new features of this tutorial will be separated into different levels/scenes and so we will be relying on prefabs to share the features between the levels. So make sure to select both the exit door and the player object in the scene and click ‘Apply’ in the inspector to save the changes you made to them.

2. Add the ability to jump

Make sure to check the Fixed Angle property of the Players Rigidbody to prevent the character from spinning when we are jumping.

Open up the scene “Level2”. You will see that this level will require the player to jump up some steps in order to get to the exit. To accomplish this, open up the ‘PlayerController’ script and uncomment the rest of the code.



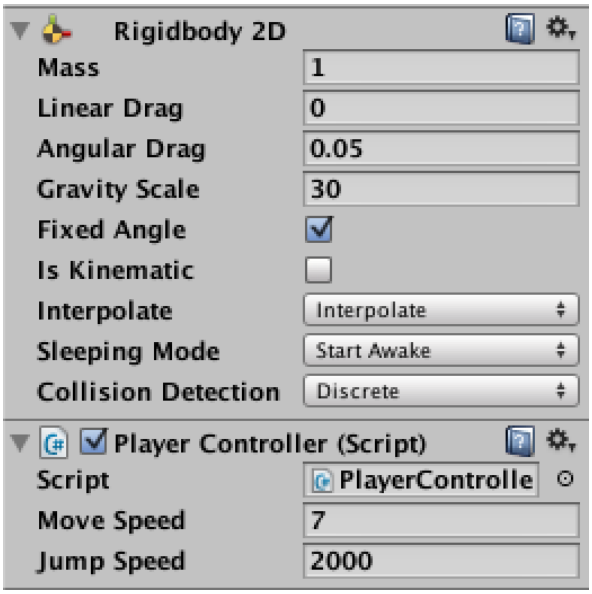
The code to make the player jump checks to see if the space button has been pressed and then applies a force to the Rigidbodys vertical position sending the character upwards. It also sets a flag to indicate that the player is jumping and only resets that flag when the y velocity returns to 0, this prevents the player from constantly jumping.

We have added code to get the player moving up but we also need to tweak the gravity value of the players Rigidbody to get the player falling down at the jumps peak. It’s a delicate balance to tweak how the player jumps and we are just using 2 variables to control this:

The Rigidbodys gravity value is used to control how fast the player falls down when airborne, both jumping and falling off a ledge. Low gravity values will make the player float down slowly, so to achieve a faster falling rate increase this.

The ‘Jump Speed’ parameter of the PlayerController script will control both how high the player jumps and how fast they jump too. One problem with this is that it’s actually linked directly to gravity, so the higher the gravity value, the higher that this value will need to be to push against it.

For now let us use the values of 30 for Gravity and 2000 for Jump Speed. Also check the Rigidbodys ‘FixedAngle’ option to prevent the player from falling over and set Interpolation to Interpolate which gives us a smoother result when moving and jumping.



Again remember to Apply these changes to the Player prefab so that they take effect in all of the other levels.

3. Get the camera following the player.

Open the scene ‘Level3’. You will see that the exit door is far to the right of the screen out of the cameras view. We will make the camera follow the player as they move around the screen.

Drag the camera onto the Player object so that it becomes nested underneath it. Now wherever the player moves the camera follows. Also reposition the camera so the player is more in the centre, that way you can easily react to the environment.

Apply this change to the prefab, this will cause a few problems in the previous 2 levels as now they will both contain 2 cameras, which by default will freak Unity out and it will give you a warning about 2 audio listeners being present which is a bad thing. It is best practice however to choose one of the cameras and remove the other one for each level.

4. Add a pitfall type object that kills the player.

Open ‘Level4’. This level has a small red pit in the middle of the floor (think dangerous Lava!) and we want to make this so that if the player lands in it, they die and the level needs to be restarted. Select the ‘Lava’ object and add to it a ‘BoxCollider 2D’ and the script

‘RestartOnCollision’.

The main purpose of this script is to reload the current level when the object is hit. To make this less sudden though, we use the code to animate the player disappearing and then pausing before the level reloads.

5. Add a moving platform.

Open ‘Level5’ You will notice that there is a big lava pit in the middle of this level that we can’t simply jump over! We’re gonna have to add in a moving platform to help transport the Player across the gap!

There is already a platform in this scene, it just needs setting up so that it can firstly move and secondly transport the player.

Start by creating 3 empty Game Objects and naming them “Platform Group”, “Start” and “End”. Drag the existing Platform object and both the Start and End objects onto the Group object so that they are all children of it.

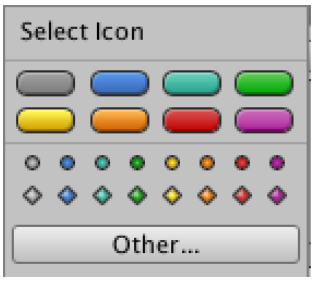
To the Groups Platform Object (The one with the sprite Renderer on it) add A BoxCollider 2D , A Rigidbody 2D and the script ‘PlatformController’ to it.

This script takes 3 parameters: A ‘Start’ and ‘End’ transform and a ‘Speed’ value. Set the Start and End to the corresponding Game Objects that you just created and leave the speed to it’s default value for now. The code will make the platform yo-yo between those beginning and end

points at the given speed.

Position the Start and End objects so that they are at either end of the lava gap, It doesn't matter which way around they are, only that when the game first starts, it will start moving the platform towards the End position first.

To make it easier to see where the Start and End objects are in your scene you can give objects icons that only appear in the scene view and not in the final game. Select the Start object and click the following cube icon in the Inspector panel, and then select a colour to use:



Then do the same for the End object and your scene should look similar to this:

In order for the script to work correctly we need to set up the parameters of the Platforms Rigidbody component. Set the settings as follows:

Gravity to 0 so it doesn't fall.
Fixed Angle to true so the platform doesn't rotate.
IsKinematic to true so that forces and collisions don't move the object, only transformations.
Interpolate to Interpolate, otherwise the object will shake as it moves.

The reason we need to use the Rigidbody here with these settings is that we are relying on the physics system to carry the player across on the platform. If we moved the platform without a Rigidbody using transform.Translate, the platform would move but the player would just stay at the same spot and fall off of it.

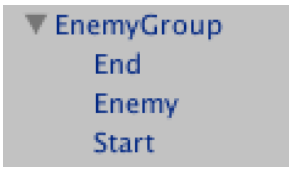
Now when you test the level, you will see that the player sticks to the platform when they jump on it and unsticks when they leave it.

6. Add some enemies

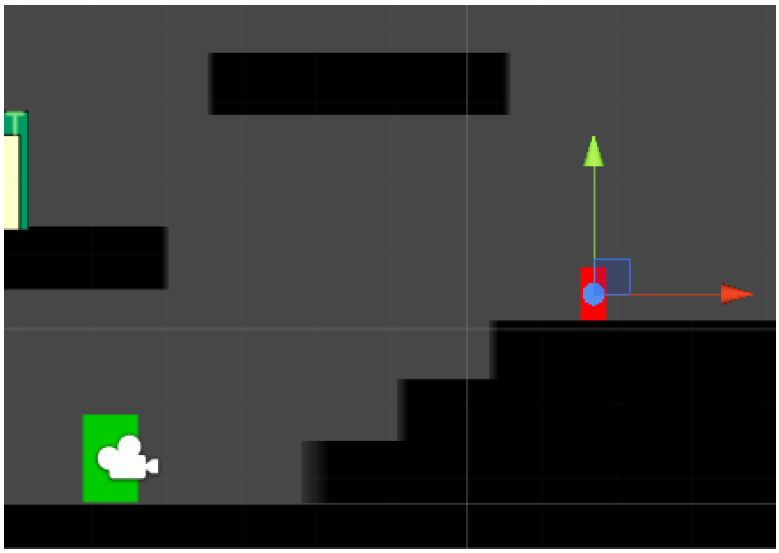
Open ‘Level6’. For this level we are going to add some enemies. The enemies will actually behave similarly to the platforms in that they will move back and forth between 2 points. We are going to make the enemies behave in classic Sonic or Mario style where you have to jump on their heads to defeat them. To achieve this we are going to use 2 colliders, one that covers the body and one smaller one that looks like a hat above the enemy that if the players hit will destroy it.

The easiest way to recreate this is to clone the Platform prefab and change it’s settings as follows:

Rename the objects in the hierarchy to:



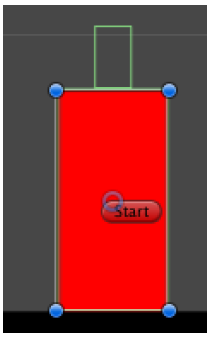
Select the Enemy object that has the sprite renderer and change the sprite to point to the red square and then use the Transform tool to scale and position the enemy, here’s mine:



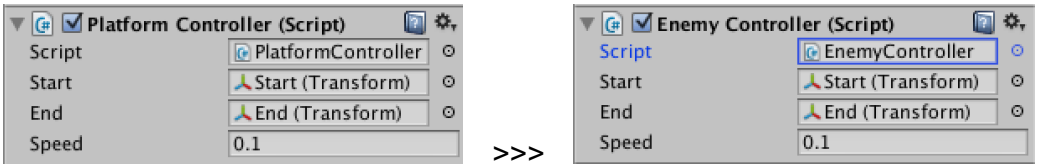
Now it’s named right and looks right we need to set the locations that it’s gonna walk to, so just like the platform I have position the Start and End objects in the level to specify a nice straight line of movement for this particular enemy:

I changed the Icon colors too so that I can easily distinguish enemy way-points from platform ones.

The next step is to modify the colliders attached to the enemy objects. The topmost enemy collider needs to be the trigger collider and the bottom one the regular collider. The top one should be repositioned and sized so that it sits like a hat on the enemy and the bottom one should cover the shape, like so:



Now we need to change the scripts attached to the enemy. Replace the 'PlatformController' script attached to the first enemy object by clicking the circle to the right of it in the Inspector and choosing the 'EnemyController'. Luckily the replacement script has the exact same parameters so we don't have to re-assign them!



Taking a quick peek at this script you will see that the main difference from the Platform script is that if the Trigger clip gets hit, the object will get destroyed.

We now need to add another script to handle when the enemy bumps into the player and should kill the player. For this we can re-use the earlier 'RestartOnCollision' script, simply attach it to the Enemy object:

Now if we bump into the larger collision rectangle the player will fade away and the level will restart!

Now why not clone the enemy and position a few more around the level?

7. Add some collectibles.

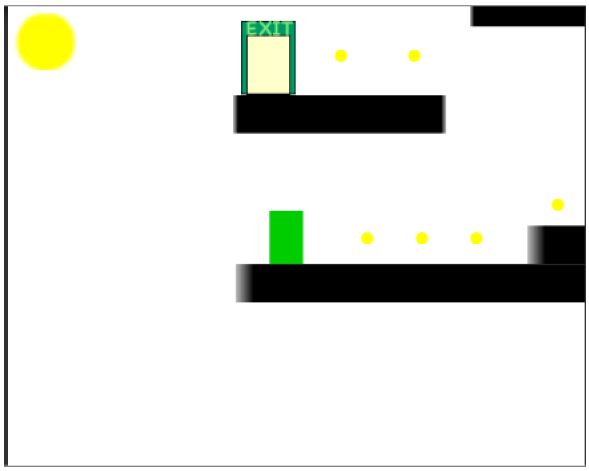
Open up ‘Level7’, you will see at the start of this level there are small yellow objects each named ‘Token’. We’re gonna treat these as our coin/ring collectibles for the game.

First select just one of the ‘Token’ objects in the Hierarchy and then add a Circle Collider 2D to it. Set the collider to be a trigger as we will want to pass through the object. Also add the ‘TokenController’ script to it.

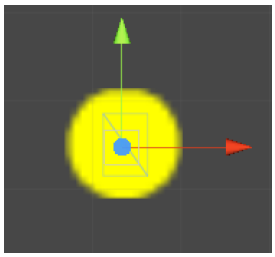
This script has the similar behaviour as the Restart On Collision script, in that it will make the token fade out when you collide with it. It will also animate the object upwards to give the impression it is being collected.

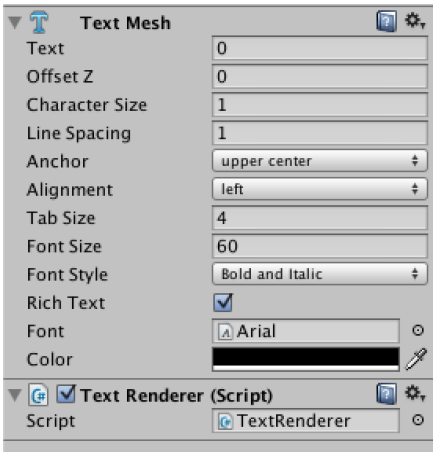
Hit Apply on the Token prefab so that the effect applies to all of the instances in the level.

Next up we want to display your current token count on screen while we play. Drag another Yellow Circle in from the Assets > Sprites folder and scale and position it so that it looks like a normal HUD icon, e.g:



Rename this object to be TokenUI and drag it onto the Main Camera, this way when the camera moves, so will this. Now create a 3D Text object, rename it to TokenText and also nest it under the Main Camera. You will want to setup and position the 3D text as follows:





Don't forget to add the TextRenderer script to it, so that it appears above our sprite.

Finally uncomment the remaining line in the 'TokenController' script.

This script has another unique way of referencing an object in the scene without having to set it in the Inspector. Because we attached the Text Mesh to the camera we can access anything attached to the games active camera using Camera.main. So we can use this here to grab a reference to the text field and increment it's current score. This is not the nicest solution, but I just wanted to show you that it is another possibility!

Now when you test the level, you will see that for each token you collect, the token counter will increase!

That's it for the basics of a platform game! As a challenge why not try to add some special power ups to make the player move faster, jump higher, and be immune to damage?