



Merchant Venturers School of Engineering
Outreach Programme

Unity2D

Building Your Own Platformer in Unity2D

Created by

Will Mustoe

Organised by

Caroline.Higgins@bristol.ac.uk

Published on February 23, 2018

Notes to Teachers & Helpers

- This workshop is intended to last $2\frac{1}{2}$ to 3 hours.
- This workshop is intended for ages 10⁺ (years 5⁺).
- The content is intended to allow the student to work at their own pace with this worksheet as a guide.
- The learning platform is Unity2D a popular cross-platform game engine.
- There are a number of versions of Unity not all of which are compatible with this workshop:
 - Unity 2017.x for Windows or Mac** This version **is compatible**.
This is the latest version available from the unity website.
 - Unity 5.6.x for Windows or Mac** This version **is compatible**.
This is a slightly outdated version but is still compatible with this workshop.
 - Unity 5.5.x and below for Windows or Mac** This version **is not compatible**.
This is an older version of unity and does not contain some of the features this workshop relies on.
- Students should already be comfortable using Unity.
This means they should be able to navigate the Unity scene view and create game objects. They should be familiar with the principles of working with Unity. A good place to start before this would be the Introduction to Unity Workshop or the Roll-a-Ball tutorial on the Unity website.
- This workshop teaches the following skills:
 - Basic Programming (Variables, Methods, etc)
 - 2D Animation with Unity
 - Basic Game Design and Level Design

1 Introduction

Hi! In this short workshop we're going to try to introduce some of the concepts that Game Developers and Software Engineers use every day to design everything from your mobile games to AAA releases.

Let's get started. Each section is made up of four parts:

Actions Stuff for you to do. They are highlighted in blue.

Notes Notes about important stuff you need to be aware of (and possibly remember!). They are highlighted in red.

Questions Questions you should try to answer. Sometimes you'll need to write things down; other times you'll need to build something in the game. They are highlighted in yellow.

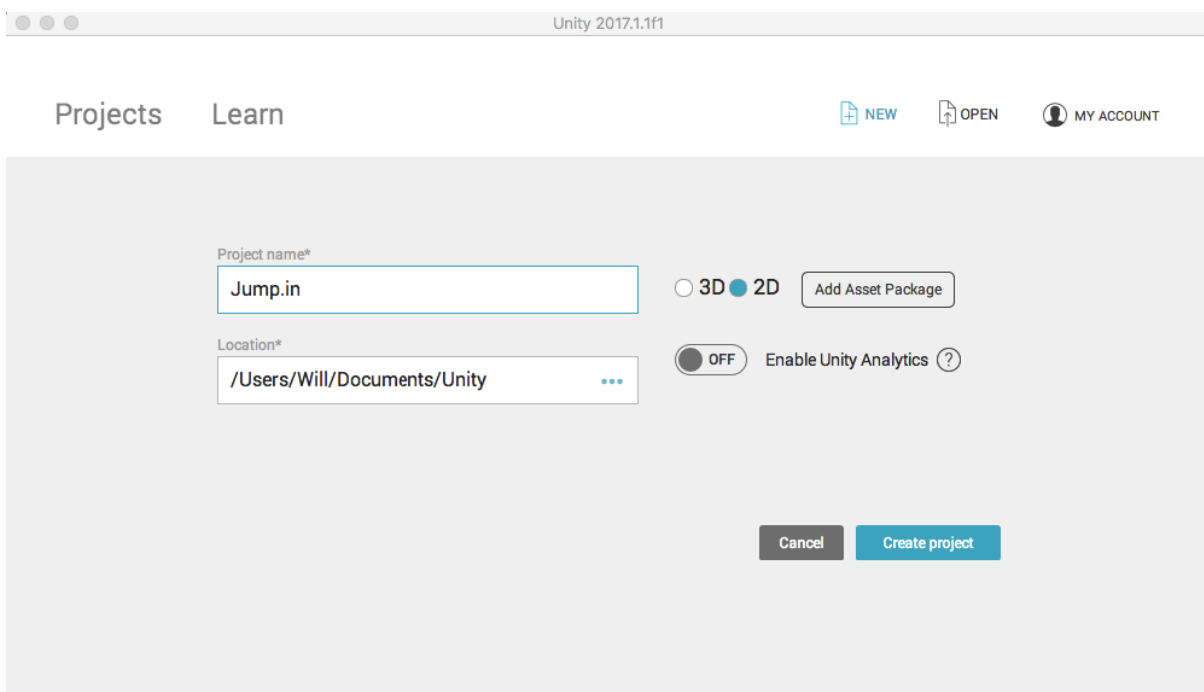
Ask a helper or the teacher to check your answers.

Goals Stuff you should have completed at the end of each section. They are highlighted in green.

We'll also write some information between parts and include plenty of screenshots to help you out.

Actions

1. Open Unity
2. Click New
3. Select 2D
4. Set the location to somewhere you will remember
5. Give your project a name



Unity's new project screen

Actions

6. Go to assets -> Reveal in Finder/Explorer (This is in the top menu)
7. Copy and paste the provided folder "Sources" into this folder

Notes

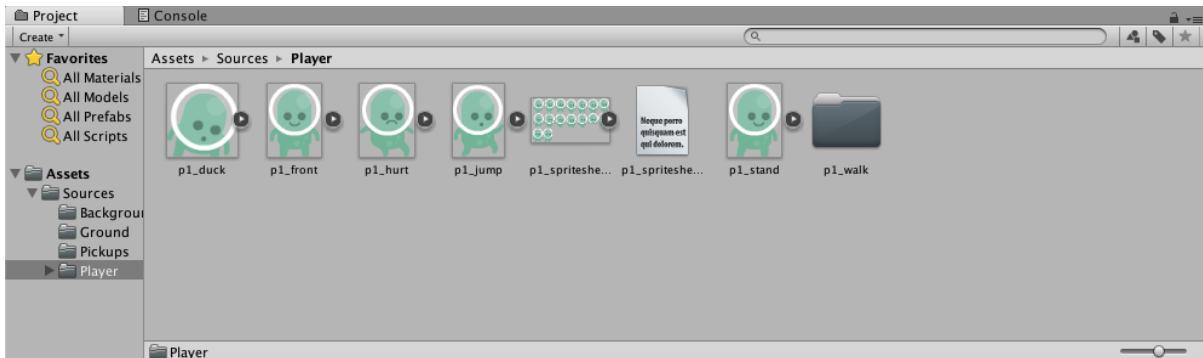
Unity will then analyse and import the sources, this may take a few seconds, when it is finished you should see a folder named "Sources" in the project window.

2 Creating Our Character

A good place to start when building a 2D game for the first time in Unity 2D is with the character we want to play as. You may already be used to importing a 3D model in Unity3D, or building one out of different 3D objects. Unity2D is slightly different though, rather than having different types of object, everything in the game is a sprite object. Lets start by opening the provided sources folder, here we have provided everything you need to build a similar looking platformer to ours, but feel free to use your own assets.

Actions

1. In the Project Window at the bottom, navigate to Assets -> Sources -> Player



The Provided Player Assets

Actions

2. Open the folder "p1_walk/PNG"
3. Select all of the files and click and drag them to the scene window
4. Save the new animation as "PlayerWalk" in a new folder called animation next to our sources folder
5. Rename the player game object in the hierarchy window to "Player"

Goals

Hit the play button at the top of the screen, and your character should start walking, although they aren't going anywhere yet!

So our character looks like they are walking, but they do look a little jerky, maybe we can polish this up and take a look at the animator window.

Actions

6. Open up the animation window, by going to Window -> Animation
7. Dock the window somewhere accessible by clicking and dragging



Animation Window

With our player object selected, your animation should look like the above. The play, pause and tracking buttons allow us to test our animation without running the game.

The drop-down menu with "PlayerWalk" currently selected allows us to view different animations in the scene.

Next to that is the samples setting, this is how many frames the animator will show a second.

Actions

8. In the top left of the animation window set the sample rate to 18

Goals

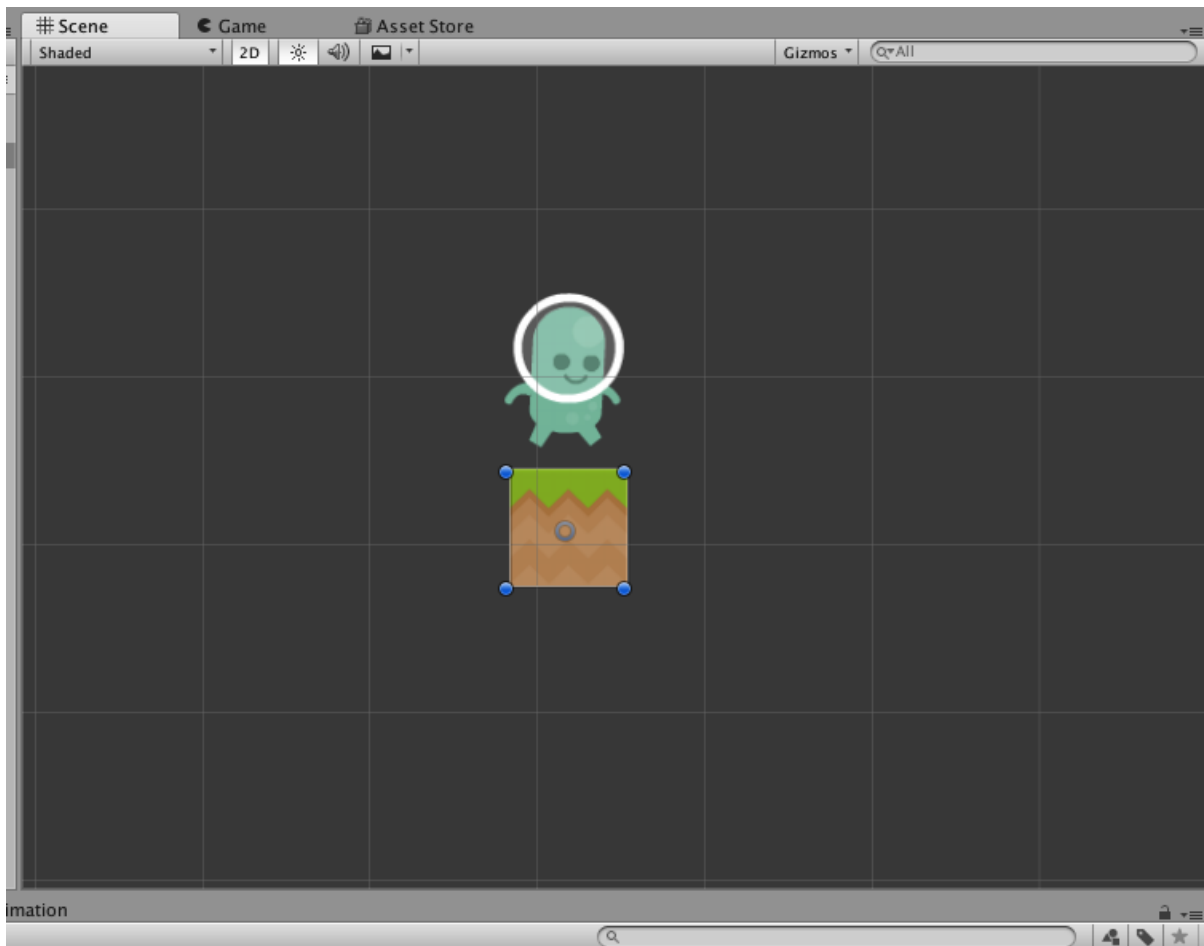
If we now hit the play button, we can see our character now walks a little more smoothly.

3 Let's Build a World

So we have a character, but they don't have a world to move about in yet. So let's address that!

Actions

1. Go back to the "Sources" folder in the the Project window, then click on ground.
2. Click on "grassMid"
3. In the Inspector window, where it says "Mesh Type" change this to "Full Rect" (you'll see why later)
4. Drag the grassMid file into the scene, and place it just underneath our character



Character and Ground

If we hit play now, nothing happens, our player just hovers in mid air walking. We need to give them some physics, so lets do that.

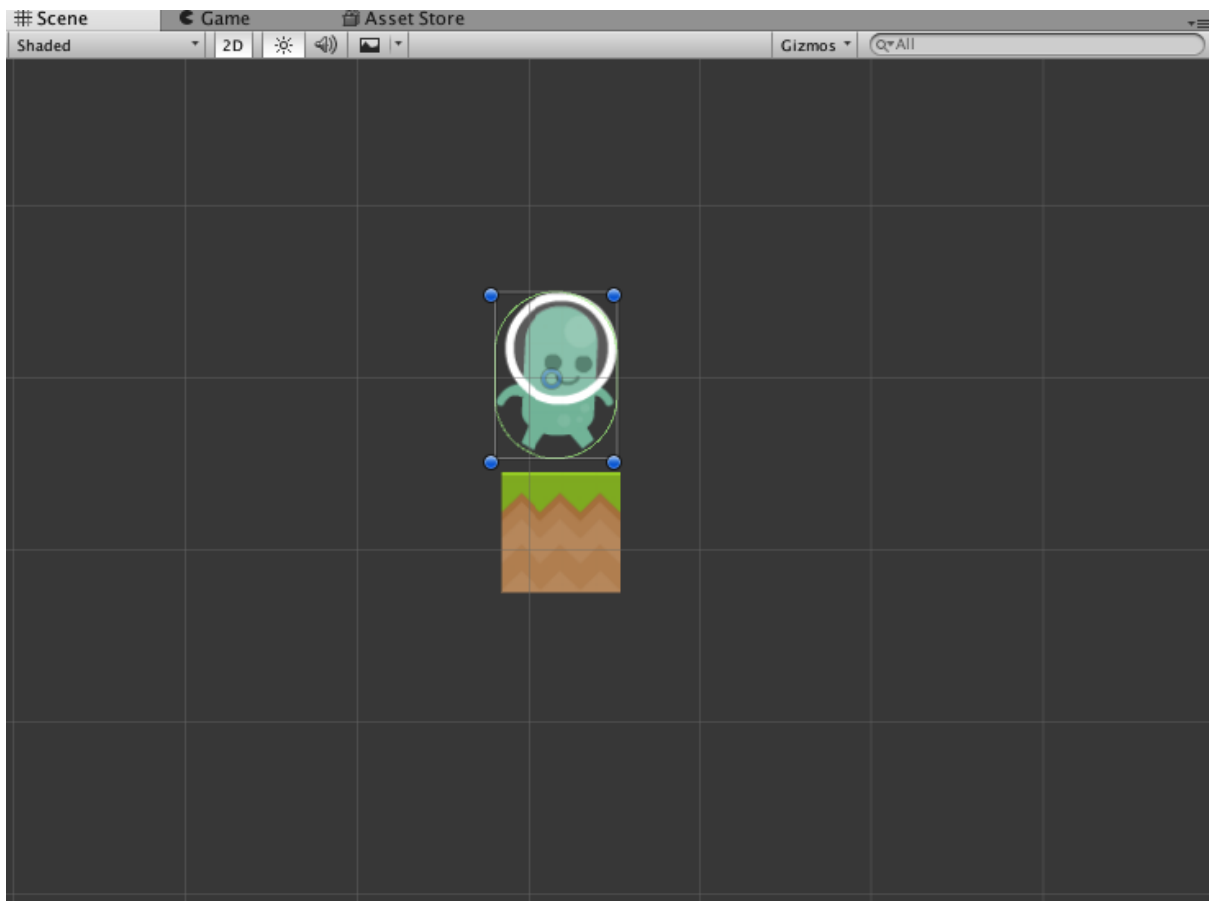
Actions

5. Select the Player game object
6. Click the add component button in the Inspector window
7. Select Rigidbody2D which is under Physics2D

What happens now if we press play? Our player falls through the world! This is because our ground and our player objects don't have Colliders, Unity uses Collider components to calculate when two objects hit each other and what should happen.

Actions

8. Select the ground object, and add a "BoxCollider2D" component
9. Select the player object, and add a "CapsuleCollider2D" component



A Capsule Collider

Now when we hit play, our player falls and lands on the ground. Perfect, but we haven't got much of a level to play through yet. Lets expand the platform a bit, and maybe add some other platforms to jump to.

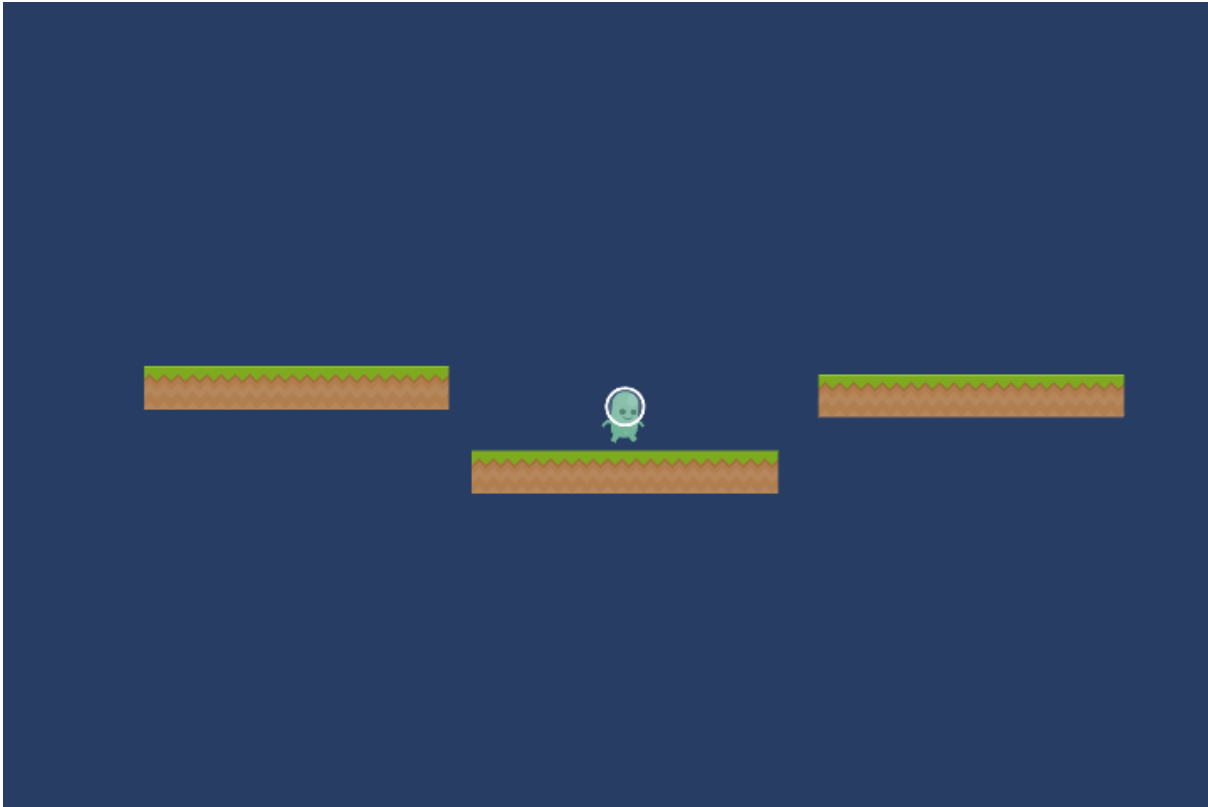
Actions

10. Select the ground object
11. In the Sprite Renderer component, set the draw mode to "Tiled"
12. Set the width to 5
13. In the Box Collider 2D component, tick the box labelled "Auto Tiling"

So what did we just do? Remember how when we imported the ground object we had to set it to be imported as a "Full Rect", this is a new feature in Unity 2D that allows to increase the size of an object without stretching the texture. Then we set the collider component to follow this size, so we don't just fall through the ground.

Actions

14. Duplicate the ground object a few times & place them under a parent object called "Ground"
15. Place the new ground objects around the scene and design a level for our character



Our Completed Level

Goals

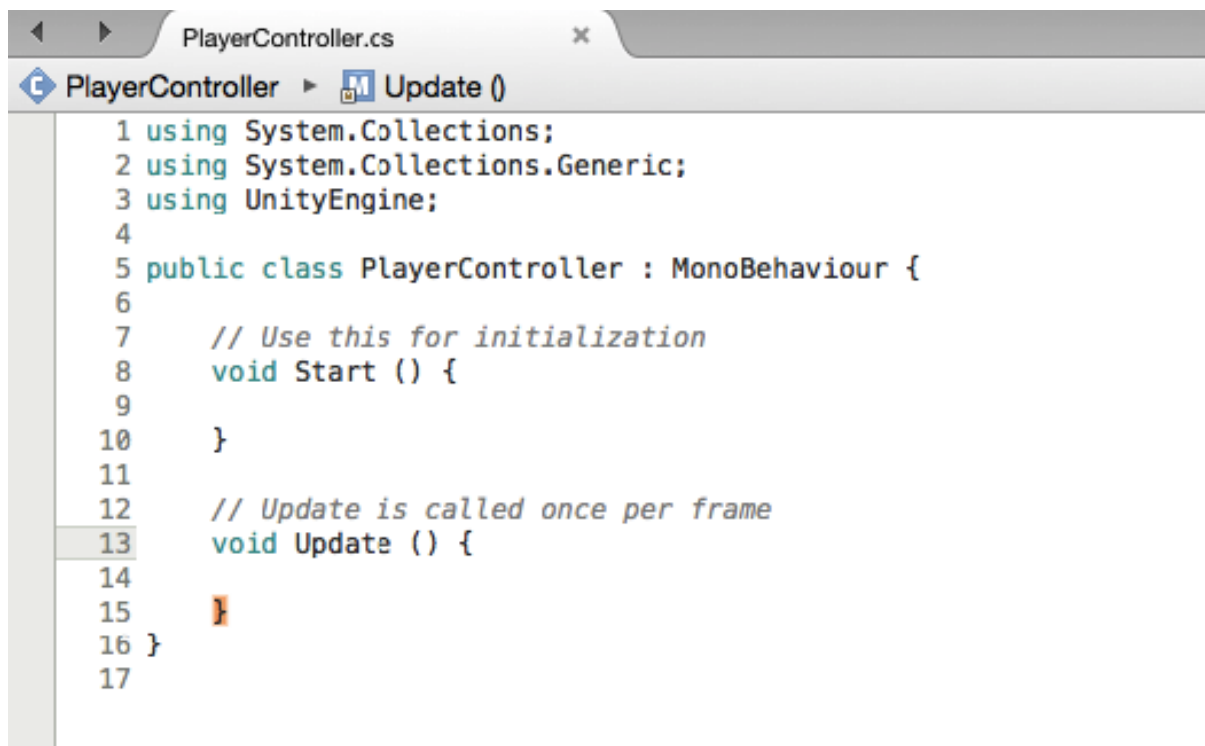
Our character should now have an interesting level to explore

4 Let's Move

We now have a character and a world, but the player can't move about in the world. To fix this we need to write some scripts to attach to the player object make them react to button presses. First lets add a blank script to our player.

Actions

1. Select the Player game object
2. Click Add Component, and scroll down to new script
3. Call the new script "PlayerController" and make sure the language is set to "C Sharp"
4. Click Create and Add
5. Open up the script by double clicking on it in the Inspector Window



```
PlayerController.cs
PlayerController ▶ Update ()
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class PlayerController : MonoBehaviour {
6
7     // Use this for initialization
8     void Start () {
9
10    }
11
12    // Update is called once per frame
13    void Update () {
14
15    }
16 }
17
```

The Bank Script Template

So let's take a look at programming in Unity, it's might be a bit different from what you are used to. When we open up our PlayerController script we see two functions already prepared for us. Both of these functions are run by Unity during our game, so we don't need to call them ourselves.

Start(): Unity runs this bit of code whenever the object is "initialised" i.e. when it's made, since we won't do anything where we need to create more players, this will just be at the start of the game.

Update(): Unity runs this code every frame update, i.e. when Unity draws a new "picture" to the screen, this can vary depending on the system and the number of object on screen. The rate at which this is run is known as the frame rate.

Fixed Update(): Unity runs this code at a fixed rate every second, we will need to add this to our script. This is where we should do all of our physics/rigid-body work. If we used Update() for this then our character would go faster or slower depending on what computer it was run on!

Actions

6. Add a function called FixedUpdate() which returns void
7. Add two new public float fields to the PlayerController script, called moveForce and maxSpeed
8. Set the new fields to a default value
9. Add a new private field of type Rigidbody2D called rb2d;
10. Add the code on the next page to Start()
11. Add the code on the next page to FixedUpdate()

```
public class PlayerController : MonoBehaviour {

    public float moveForce = 365f;
    public float maxSpeed = 5f;

    private Rigidbody2D rb2d;

    // Use this for initialization
    void Start () {
        rb2d = GetComponent<Rigidbody2D>();
    }

    // Update is called once per frame
    void Update () {

    }

    void FixedUpdate () {
        float h = Input.GetAxis("Horizontal");

        if (h * rb2d.velocity.x < maxSpeed)
            rb2d.AddForce(Vector2.right * h * moveForce);

        if (Mathf.Abs(rb2d.velocity.x) > maxSpeed)
            rb2d.velocity = new
                Vector2(Mathf.Sign(rb2d.velocity.x) * maxSpeed,
                    rb2d.velocity.y);
    }
}
```

Move Code

This looks very complex, but it breaks down nicely and is very nice way of moving characters in Unity. Let's see if we can understand what we have just done:

Start(): Here we are getting the rigidbody (Physics bit) of the game object the script is attached to. We need this to be able to apply forces to our player.

FixedUpdate(): So first we are setting a float (number with decimals) "h" equal to the current horizontal axis, this is essentially the direction the player wants us to move in. Then we need to do one of two things, if we are below our maxSpeed then we just push our character in that direction, and if we are above our maxSpeed we need to slow our character down to the maxSpeed. This is what the two if statements we added do.

Actions

12. Go back to Unity and try your game!

Notice anything wrong? Our character is rolling around all over the place. This isn't what we want, we want them to stay upright.

Actions

13. Select the player game object, and go to the Rigidbody2D component.
14. Click the dropdown labelled "Constraints", and then select "Freeze Rotation"

If we try to run the game again, our character now moves about as we would expect. Next we would like to be able to give our player the ability to jump, otherwise we can't reach those other platforms we made. To do this, we are going to record when the player presses the space bar, and then on the next FixedUpdate(), we are going to give our character a push in the upwards direction. We do have to check our character is on the ground though, otherwise we'd be giving our character the ability to fly!

Actions

15. Add a new public float called jump force, and set it to a default value
16. Add a two new private variable called jump and grounded, they will be of type bool
17. Add a new public variable called groundCheck, this will be of type Transform
18. Add a new public variable called jumpForce, this will be of type float
19. Add the below code to Update()
20. Add the below code to FixedUpdate()

```
public float moveForce = 365f;
public float maxSpeed = 5f;
public float jumpForce = 250f;
public Transform groundCheck;

private bool jump;
private bool grounded = false;

void Update () {

    grounded = Physics2D.Linecast(transform.position ,
        groundCheck.position , 1 <<
        LayerMask.NameToLayer("Ground"));

    if (Input.GetButtonDown("Jump") && grounded)
    {
        jump = true;
    }

}

void FixedUpdate (){

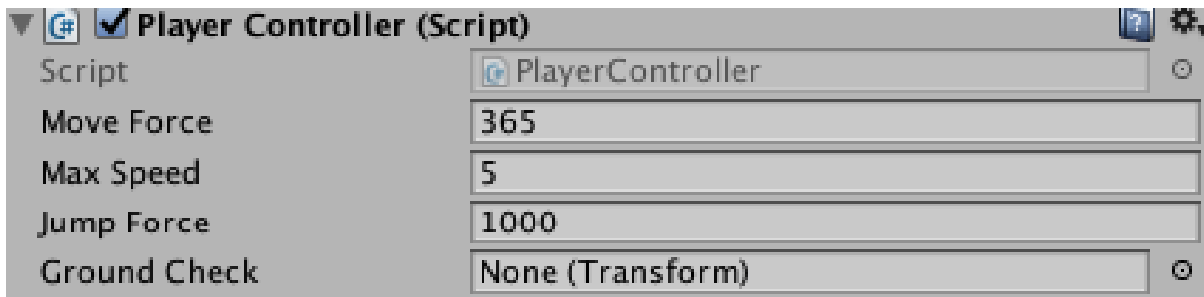
    if (jump)
    {
        rb2d.AddForce(new Vector2(0f, jumpForce));
        jump = false;
    }

}

}
```

Jump Code

If we go back to Unity, and press play we notice we get an "Unassigned Reference Exception" this is telling us that we defined a public field for our script but didn't assign anything to it.



Inspector Window

In the inspector window we can see all the public fields we defined in our script, and the default values we gave to them. We also see the field Ground Check which is expecting something of type "Transform" in Unity a Transform is just a location/rotation in space. This transform will allow us to check if our character is on the ground, so let's have a look at creating it.

Actions

21. Click on the player object, and click on Create -> Create Empty Child
22. Name this new object GroundCheck
23. Using the Inspector set the position values to X: 0, Y:0.7, Z:0
24. Click on the player object, and drag the GroundCheck object to the Ground check transform field of the Player Controller Script

We need to do one more thing, in the code when we set the variable "grounded" we check to see if our transform is in a Sprite with the layer "Ground". We haven't set up that layer yet, so let's do that next.

Actions

25. Select your Ground parent object
26. In the inspector click on the layer drop down in the top right
27. Click add layer, and add a new layer called "Ground"
28. Go back to the Ground parent object, and click the drop down again then select "Ground"

Goals

Hit play, and you character should be able to jump and move around in our level

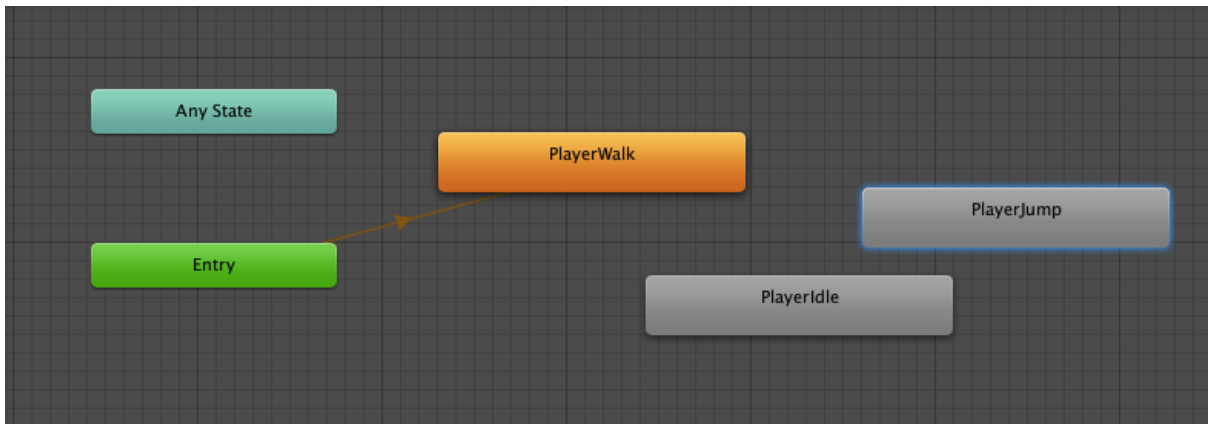
5 Fixing our Animation

So hopefully by this point your game is looking pretty good! You should be able to move around and explore the world you have created, but there are a few things that are off, for one our character never stops running! Even when they are stood still or flying through the air, so we need to make a few changes to our animator to fix that. For this we need one more tool, called the Animator which allows us to change the animation of our character based on some values we can assign.

Actions

1. Click on the Player object, and then go to Window -> Animator, and drag the window to dock it somewhere handy
2. In the Animation window, click on the drop down labelled "PlayerWalk" and create a new clip called "PlayerIdle"
3. From the project window, drag "p1_stand" from the Sources -> Player folder, into the Animation window.
4. In the Animation window, click on the drop down labelled "PlayerIdle" and create a new clip called "PlayerJump"
5. From the project window, drag "p1_jump" from the Sources -> Player folder, into the Animation window

Ok so that was a lot to take in, but essentially all we have done is create two new "states" for our character to be in, one for when they are standing still and one for when they are in the air. Now we just need to tell the animator when the character changes state, we do this with the diagram in the animator window which should look like this.

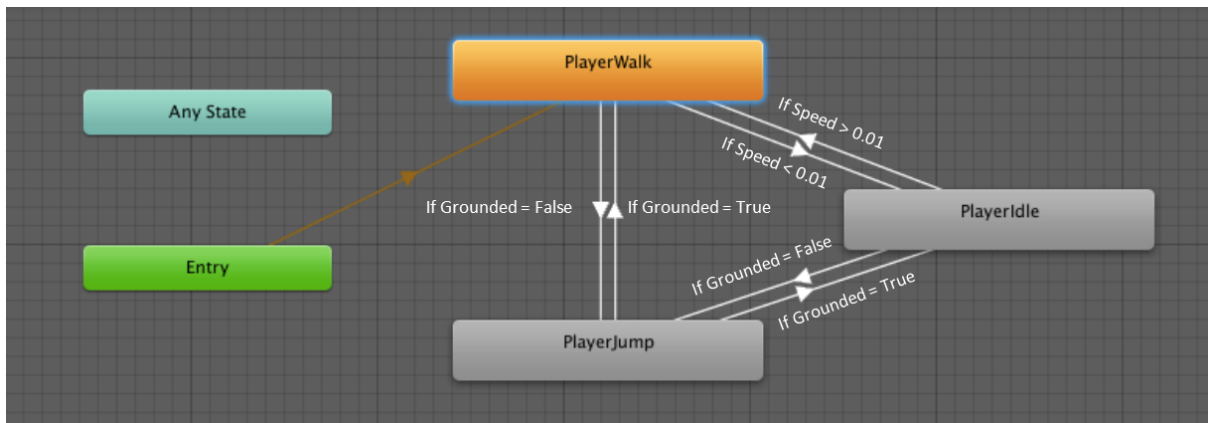


Animator Window

Now we need to add some transitions between these states, but first we need a few values to transition on, then we can add the conditions on the transitions.

Actions

6. In the animator window, click the button labelled "Parameters" in the top left
7. Next click the plus button, and add one float called "Speed" and one bool called "Grounded"
8. For each arrow below, right click on the starting block and select "Make Transition"
9. Then click on the Transition and in the inspector window add the conditions listed in the diagram



Animator Window

Finally we need to update the values we are transitioning on from our code, so lets add a few lines to do that. First we need access to the animator from our code, and then we need to update the values we defined as we move and jump.

Actions

10. Add a new private variable called anim of type Animator;
11. In our start function set anim equal to "GetComponent<Animator>();" ;"
12. Finally update both our Update code, and Fixed Update code so they look like the below

```
void Update () {  
    grounded = Physics2D.Linecast(transform.position ,  
        groundCheck.position,1<<  
        LayerMask.NameToLayer("Ground"));  
    anim.SetBool ("Grounded" , grounded);  
    ...  
}  
  
void FixedUpdate () {  
    float h = Input.GetAxis("Horizontal");  
    anim.SetFloat ("Speed" , Mathf.Abs (h));  
    ...  
}
```

Animator Code

Cool, now our character should move between states smoothly, there is one more thing we need to correct to make our character look a bit more natural, they only face one way! So to avoid our character moon walking, lets add a flip function.

Actions

13. Add a new private bool called facingRight which is initially equal to false, near where we added bools for jump and grounded
14. Next we need to detect when we need to flip our character, so add the code below to our FixedUpdate function
15. Finally lets add the flip function as below

```
if (h > 0 && !facingRight) {  
    flip();  
}  
else if (h < 0 && facingRight) {  
    flip();  
}
```

FixedUpdate

```
void flip(){  
    facingRight = !facingRight;  
    Vector3 scale = transform.localScale;  
    scale.x *= -1;  
    transform.localScale = scale;  
}
```

Flip

Goals

Now hit play, and our character should look a lot more natural moving around the level!

6 Winning and Losing

So our game is finally coming together, we are just missing one more thing. Gameplay! At the moment we can't win or lose, so there's not much point in playing, so let's fix that. How about we add some coins to collect, and maybe some spikes to avoid?

Actions

1. From the Sources -> Ground folder, click and drag the spikes somewhere into your level.
2. Select the spikes object, then in the Inspector select the Tag drop down, and add a new tag called "Death"
3. Select the spikes object again, and assign it the tag "Death"
4. Finally add a new Component to the spikes called "Polygon Collider 2D" under Physics 2D
5. Then tick the "Is Trigger" box on the new Collider

Now you can duplicated the spikes around your level as much as you like, now for the coins.

Actions

6. From the Sources -> Pickups folder, click and drag the coin somewhere into your level.
7. Select the coin object, then in the Inspector select the Tag drop down, and add a new tag called "PickUp"
8. Select the coin object again, and assign it the tag "PickUp"
9. Finally add a new Component to the coin called "Polygon Collider 2D" under Physics 2D
10. Then tick the "Is Trigger" box on the new Collider

Ok so now we need to add some code to make our player react to the new items we've added to the level. Let's deal with the spikes first, if the player hits them we want to reset the level, so they can try again. To do this we need two new functions.

Actions

11. Add the following import to the top of your code file:
"using UnityEngine.SceneManagement;"
12. Add the OnTriggerEnter2D function to your code as below
13. Add the reload function to your code as below

```
void OnTriggerEnter2D(Collider2D other){  
    if(other.gameObject.CompareTag("Death")){  
        reload();  
    }  
}  
private void reload(){  
    Scene scene = SceneManager.GetActiveScene();  
    SceneManager.LoadScene(scene.name);  
}
```

Lets make sure we understand what Unity is doing here, the OnTriggerEnter2D function, is called when our player bumps into something. So here we are checking to see if the thing we bumped into has the tag "Death" which we assigned to the spikes early, if it does we want to reload the current scene. The reload function is what we call to do that.

Goals

Test your new code, by running the game and walking the character into the spikes, you should be teleported back to the start

One last thing to add, and we have completed our basic game. Lets add some code to count the coins we have, and check if we have all of them so we can win the game.

Actions

14. Add the following import to the top of your code file:
"using UnityEngine.UI;"
15. Add a new private variable called count, of type int;
16. Add a new private variable called totalCount, of type int;
17. Add a new public variable called winText, of type Text;
18. Add the following code to your start function

```
void Start () {  
    ...  
    count = 0;  
    totalCount =  
        GameObject.FindGameObjectsWithTag("PickUp").Length;  
    winText.text = "";  
}
```

Next we are going to add code to our `onTriggerEnter2D` function to check for when our player bumps into a coin.

Actions

19. Add the following code to `onTriggerEnter2D`
20. Add a new function called `checkWon` as below

```
void OnTriggerEnter2D(Collider2D other){
    ...
    if(other.gameObject.CompareTag("PickUp")){
        other.gameObject.SetActive (false);
        count++;
        checkWon();
    }
}

private void checkWon(){
    if (count == totalCount) {
        Time.timeScale = 0;
        winText.text = "You Won!";
    }
}
```

Finally we need to add a Text object, that will display "You Won!" when the player wins.

Actions

21. In the Hierarchy, click create and under UI click Text
22. Select the new Text object, which will be under the object called Canvas
23. Set its position to (0, 0, 0), and it's width and height to 200
24. Set the font size to 32
25. Select the player object, and click and drag the new Text object into the field labelled `winText` under the `PlayerController` heading

Goals

Test your new code, by running the game and walking the character into the coins, once you have collected them all the game should pause and display the text "You've Won!"

7 Finished

Congratulations! If you have reached this far, then you have completed a 2D platformer game with Unity2D. This means you should now have an understanding of all of the core concepts and features needed to build games in Unity2D. From here you can start your own new game, or build from what you've already created, below we've listed some helpful resources to guide you further.

Unity 2D Manual:

<https://docs.unity3d.com/Manual/Unity2D.html>

Unity 2D Game Creation Tutorial:

<https://unity3d.com/learn/tutorials/s/2d-game-creation>

Unity 2D Roguelike Tutorial:

<https://unity3d.com/learn/tutorials/s/2d-roguelike-tutorial>