Mobi3002

Project #1

Unity #1

Mike LeBlanc

# So, you want to make a game?

The world of game development is enticing, especially for those of us with a programming background already. We’d take one look at the systems in place and have a generally good idea of what’s going on. That’s fine and dandy, until you realize (if you’re like myself,) “Dang, I can write the logic of this, but what’s a pencil, and how do I get the pretty pictures on the screen?” or, “Why is it so silent? My kids must be into something. Do I have to write the music for this too?”

Great news ladies and gents! I have all your needs covered for the “Hello, world!” of making a video game.

# ASSETS!

Here are a bunch of links, screenshot and save this for side projects if you wish.

<https://develop.games/> - This is one of THE most important links.

<https://itch.io/> - for all your indie game fixes and asset needs. 2d/3d/audio/sfx

<https://sfx.productioncrate.com/> - Lots of free sound effects and background music.

<https://opengameart.org> – Another great asset location

Also, lots of great, free assets on the Unity/Unreal/Godot asset stores.  
(Unreal gives $250 - $400 in free assets away EVERY month. Mostly 3D assets).

Name of the game (heh) here in the early days is free assets, and what you can create. After you’ve gathered your assets, then its time to learn how to get it into the hands of the people.

# Getting started (background)

(We’re skipping this, Jeager will probably cover this, but I’ll add it here.)

Okay, in Unity, after we’ve downloaded the program, we’ll select new Project and Name it something, doesn’t matter the name because we’d likely not publish anything with this project.

A screenshot of a computer

Description automatically generated

Once we’re loaded in, we get a quite intimidating screen.

A screenshot of a computer

Description automatically generated

However, need not be discouraged, this will be easier to see once we get some stuff imported. Let’s start easy by creating some folders to get some organization for our project. In the middle bottom of the screen, we create some new folders. Let’s have a folder for our assets.

A screenshot of a computer

Description automatically generated

See? We’re already at home. Let’s pull our background asset in and put it in the folder.

Once they’re in the folder, we can pull them up and into the scene.

A screenshot of a video game

Description automatically generated

We have the background in, it’s a little small, but it doesn’t seem to be in the right order. Thankfully, we have a setting on the right side of the program specifically for this.

A screenshot of a computer

Description automatically generated

So, we’ll put the sky as -3, the clouds as -2, and the grass as -1. In this order. And after that with a little bit of moving, we have a picturesque background!

A screenshot of a video game

Description automatically generated

Now we just need to scale our background to be bigger than the white lines for the camera. Not hard, just select the 3 backgrounds from our scene view on the left side of the program, and scale them up.

A screenshot of a computer

Description automatically generated

# Tiles and the floor

Now, as we’ve established, I am no artist. Of any kind. At all.

But thanks to the power of the internet, I don’t have to be. Let’s grab our “Tileset” which contains the floor we’re going to use and drag it into its own folder that we’ll call ‘floor’. Very intuitive.

Now, lets create something that’s called a Tilemap. This is important because we’ll be able to essentially paint our floor tiles onto our scene and in front of our background.

A screenshot of a computer

Description automatically generated

We’ll call this asset appropriately ‘floor’.

Next is extremely important, and is a small learning curve, and unless you knew to look for it, will be difficult to just fumble into.

Look at this next picture carefully because there is a lot of info.

A screenshot of a video game

Description automatically generated

After selecting the tile set in our project folder, we have some settings in the inspector that we need to change. First, we must ensure that the Texture Type at the top is set to Sprite, Sprite Mode must be set to ‘Multiple’ as we have more than 1 sprite in the tile set. Most importantly, Pixel per Unit must be set identically to the squares that appeared where our background is.

By default, these squares are 16 by 16 pixels each, so our Pixels per Unit must also be set to 16.

Now we can hit Sprite Editor, and a new window will pop up. It’s important to change the Type at the top from ‘Automatic’ to ‘Grid by Cell Size’. And, as you may have guess, just below we are going to set our grid size to 16 x 16 to match up with our cells in our background.

Once we hit ‘Slice’ at the bottom of the popup window, it will prompt us to use a folder to save all our new assets. Go ahead and make a new folder called ‘FloorTiles’ and save them in there for safe keeping.

To effectively paint our tiles into our Scene (where our background is currently) we need to open a tile palette, this is easy and quick to do, I’ll show you!

A screenshot of a computer

Description automatically generated

Once we open this up, lets make a new palette, and then drag the tile set we just spliced into the Tile Palette and get ready to paint.

A screenshot of a computer

Description automatically generated

When you get that done, your Tile Palette should look something like this:

A screenshot of a video game

Description automatically generated

This is lengthy, so let’s wrap this up. Shall we?

At the top of the Tile Palette, there is a brush icon. Select that, and now select the tile underneath that you want to paint with, and then paint away in your scene!

A screenshot of a computer

Description automatically generated

We have just completed our first floor! Congratulations!

(If Jeager explains this, we won’t. Hopefully we won’t because this is way over 10 minutes!)

# Player Sprite and collision detection!

I really am hoping this PDF isn’t too long winded and Jeager has finished by now. But let’s continue.

We can follow the same steps as before for importing our player sprite. Find a good-looking character asset from one of the stores above, and create a new folder called ‘PlayerSprite’ or something to that effect. Then select the asset, make sure the settings are correct, and then slice them up appropriately. Usually, your first player sprite will be 1 tile (unlike the floor, where we can paint a bunch of them).

Here’s what we look like after that: (Also, don’t forget to put the Order to 1 in the Sprite Renderer!!)

A screenshot of a video game

Description automatically generated

Now, we’re going to work with this player sprite and the ground to create some collision and movement scripts. We’ll take it one step at a time, so there’s no need to rush. It’s like learning to fly. (There’s a song from yesteryear.

A screenshot of a computer

Description automatically generated

First, we’ll want to make a new layer, this isn’t important now, but an easy thing to forget.

A screenshot of a computer

Description automatically generated

Now we go back and select ‘Ground’ as the layer. Done!

A screenshot of a computer

Description automatically generated

Now, still working with the ground, we must add a new component to it. We’ll want a 2D Box Collider.

A screenshot of a computer program

Description automatically generated

Great! Now we have a weird green box around our camera. We’ll want to hit the ‘Edit Collider’ button and adjust the green square to the place on the floor that we’d like to collide with.



Kind of hard to see, but its there. Notice I put the collision box a little from the top of the grass, this is just preference, you can honestly put it wherever you’d like your player to collide with or not.

Before we go colliding with the floor, we must set up 4 items on the player. These items are a 2D Box Collider, a RigidBody2D, a player movement script we’ll write ourselves, and a material so we don’t stick to the floor.

Starting with what we know, go ahead, and set up another Box Collider and move the green box to around our little character we brought in.

2nd component is also easy, Let’s add a new component called RigidBody 2D, and set the following settings:

A screenshot of a computer

Description automatically generated

The mass in our situation doesn’t apply, but the one thing to note is in the constraints. We do not want to lock our position on the X or Y axis, meaning we can move both up and down. Secondly, we don’t want this sprite to rotate on the Z axis, so we can lock that.

The next easiest thing to do is we’re going to make a new folder under our root workspace we’ll aptly name ‘Materials’. Now, as you may have guessed, we’ll make a new material, specifically a new Physics Material 2D. And name this one ‘NoFriction’, the intent as you may guess, is to have no friction to the floor.

A screenshot of a computer

Description automatically generated

Now, just set Friction to 0. Easy enough.

A screenshot of a computer

Description automatically generated

And finally, drag our ‘NoFriction’ material onto the Box Collider 2D where it says Material. Like this:

A screenshot of a computer

Description automatically generated

Great! I lied when I said like 4 things. It takes a little bit to get the ‘Hello World’ of game dev. But I promise we are almost there! (1500 words later)

One last thing before we get into the script. On the left, lets create a new empty object, and attach it to our player object in the Hierarchy (left hand side).

A screenshot of a computer

Description automatically generated

Let’s name that object ‘GroundCheck’ and it will be crucial that we either now, or after the completion of the script, to move it to the bottom of our Sprites feet. Our script will check for this object to see if we collide with the ground or not.

A screenshot of a video game

Description automatically generated

Okay. Now for the bit where we know what is happening and can read. We need a new script for player movement. So hit that Add component button while our player sprite is selected and let’s choose a script. Name this ‘Player Movement’ to let us know what’s up. A screenshot of a computer program

Description automatically generated

Here is a snippet of all the code we’ll need. Notice the [SerialzeField] in square brackets. What this does is allow us to kind of ‘plug in’ components into the script like we did with the ‘NoFriction’ material. Very neat! We can also get this effect by using ‘public’ variables, but that’s not good coding standards.

It’s important to note that the FixedUpdate function is called by unity every 0.02 seconds. This is the length of time of the frame. So currently we are getting about 50 frames per second.

Its important to note that Unity has a lot of movement already set up. WASD to move the cardinal directions and space to jump. Not entirely sure of the others yet.

From top of the script to the bottom, we have 4 private variables that we’ll initate and use and we have 3 fields to plug into this script for 3 different components that we need, and you’ll remember setting these up already. RigidBody 2d, our Ground Layer that we made, and the empty Ground Check object that is attached to the player Sprite.

Every 0.02 seconds (frame) we have to check to see if the space bar was pressed or if we turned. If space is pressed and we are on the ground, we need to add some vertical like our boy King James! So, we need a new Vector2 (makes changes to the x and y axis only (2D Space)) that takes the current value of our x axis and adds our jump power to create a nice jump effect! If we want to turn to go the other way, we want our Sprite to flip and look the other way as well, no walking backward here!

So as long as we are horizontal, we want to flip our boolean to be not what it was before, then we want a vector3 (3D space) to transform the x axis of the sprite image from 1 (looking right) to -1 (looking left) and multiplying our x value by -1 will always achieve this transformation.

In the FixedUpdate function, we are just moving our sprite based on the directional button. Taking the horizontal value, multiplying our speed, then updating.

The boolean checking to see if we are grounded refers to the empty object we attached to our player sprite. This create a small transluscent circle that returns true if it collides with the box collider we set up earlier, thus, disallows our Sprite from just falling through our floor.

Save, but don’t close this file. IF there are any errors, nothing will compile and you can’t test your little game if you don’t have everything just right. And this is where our knowledge of programming REALLY drives us forward.

Okay, that was a lot, and maybe a lot of frustration. Now the easy part.

A screenshot of a video game

Description automatically generated

Drag your RigidBody2D component to its spot, and the GroundCheck object to its spot. For the Ground Layer, use the drop down to select our layer we created earlier.

Okay. Moment of truth. The MONEY SHOT! Hit the play button at the top middle of Unity. And, if everything is just right, YOU CAN PLAY YOUR LITTLE GAME!!!!

A screenshot of a video game

Description automatically generated

YES! THIS IS SO GOOD! Now, its not much yet, but the world is your oyster with this. This is a big feat if you haven’t done this sort of thing before. I’m proud of you, because it’s the small victories that count (all victories count, so give yourself some credit!).

# Camera Movement

I love everything about this and what you’ve been able to do. To kick things up into high gear and add some depth to our level here, lets add the ability for our camera to be able to follow the sprite left and right and allow us to see a little further than what we can now.

A screen shot of a computer

Description automatically generated

29 lines is all it takes. Let’s go over this script. From top to bottom, we have another [SerializeField] for our player (sprite), an offset and offset smoothing float, as well as the vector3 for the player position.

We set our player position vector 3 to the current x, y, and z values. Then, we set the camera to the x position plus our or minus the offset depending on which way we are facing.

Finally, we transform the position of the camera to a new vector3 using its Lerp function to the new position, and so the camera doesn’t snap (and snap our necks with whiplash too) we add some smoothing. The offset smoothing is multiplied by the difference in time. Higher smoothing means slower time to get to the new position.

So, like before, we can create a new script on the Main Camera object this time, (don’t forget to make a spot for your custom scripts to live.)

A screenshot of a video game

Description automatically generated

And drag our Player object down and put it in the Camera Controller script. Easy!

Now, this is what our camera will look like when we run our game and test it:



Notice our camera is no longer center and will follow us around the map! Go you!

I love that we have movement and camera control now. This is awesome! You know what isn’t awesome? When you run off the sides of your map and now you must reset the game. Let’s fix this! And since we know that the floor prevents us from falling down, lets make some walls so our character cant walk through them.

Make a new empty object in the Hierarchy and add a 2d collider to it.

A screenshot of a video game

Description automatically generated

And set the boundaries to higher than the scene, and that’s pretty much it. Easy fix for us!