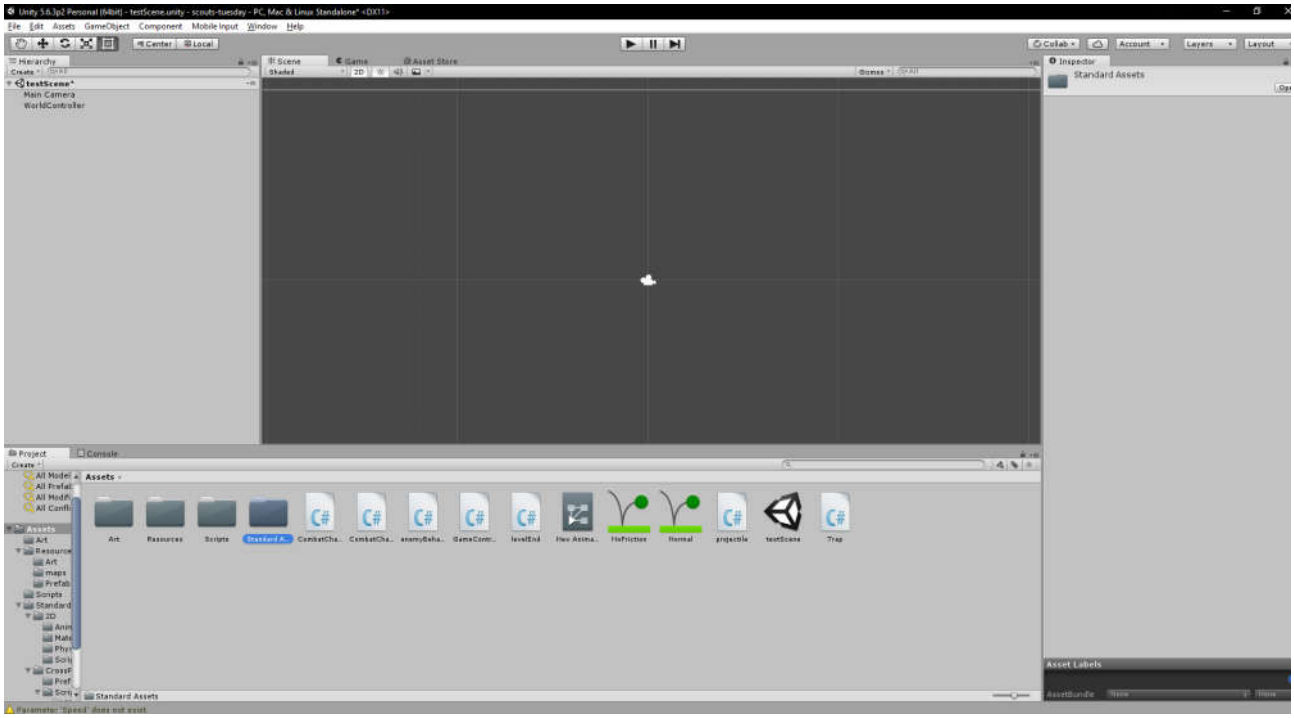
Section 4: Unity

You can download Unity here. It's not strictly necessary for most of the jobs, but it makes most of them a *lot* easier. Once you've downloaded and installed it, made an account, and logged in, you'll get an option to create a new project or open a project. Choose to open a project, go to whichever folder you pointed GitHub at (that is: you want to highlight the folder that contains the folders Art, Resources, Scripts, and Standard Assets), and hit "select folder"). It'll sit there opening up for a minute, and you'll be left with something like this:

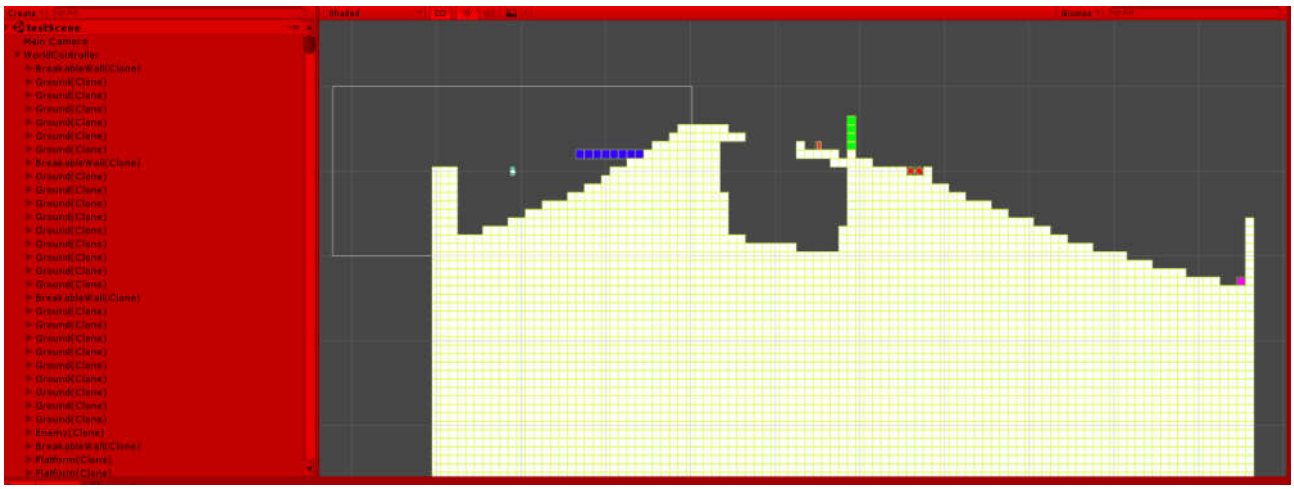


(I think that's the default layout, but yours might be slightly different if I'm wrong).

Starting with the easy things: if you press the "play" button at the top-centre, it'll start the game in that middle bit. At the moment, this is exactly the same as a not-full-screen version of levelTest.exe, but it'll get more and more different over time. You can press play again to go back to where we started.

If you click over to "console", you'll see (hopefully) an empty box. If there are any error messages here, they're probably important (though we can also use this for testing – there's a function in the code for just putting text here while it's running, which is useful for checking what's going on behind the scenes).

While it's running, hit pause. There'll now be a version of the level that you can look around freely in the middle. If you expand out the WorldController in the left bar, you'll get an utterly ridiculous list of items:

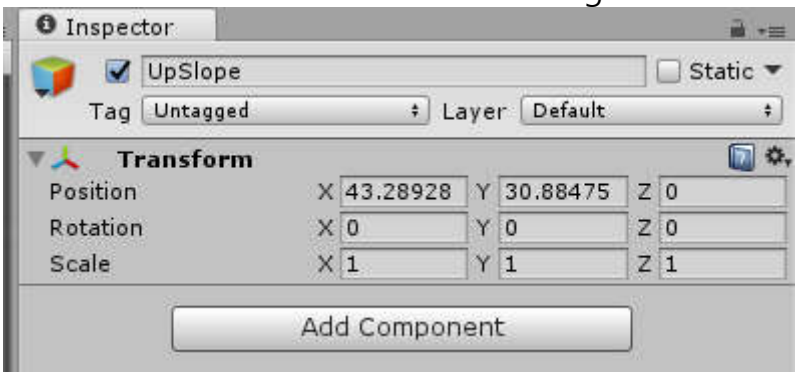


(Yours probably won't be red – I've set mine to do that so that it's really obvious when I've got the game running and when I haven't. You might not have the boxes around each tile either – that's another thing I've changed).
There's nothing really to do here right now, but notice how everything has "(Clone)" at the end of the name? That's going to be important later. Make sure you stop the game running before you do anything important – stuff isn't saved while it's running.

Making new tile types
So, the level design people have told you that they want a new tile (or maybe you are one of those people and you're doing it for yourself). Let's say they want some slope tiles.

We're going to make what's called a *Prefab*. These are what those objects that we saw in the list on the left are clones of – the MapLoader script attached to the WorldController clones a different prefab depending on what's in each pixel of the map image – the map.txt file tells it which colours match to which Prefab name. To do this, right click in the box on the left, and click "Create Empty"). This should give you a new item underneath Main Camera and WorldController called "GameObject". Rename it to something sensible for what you're doing (either using the box on the right, or press F2).

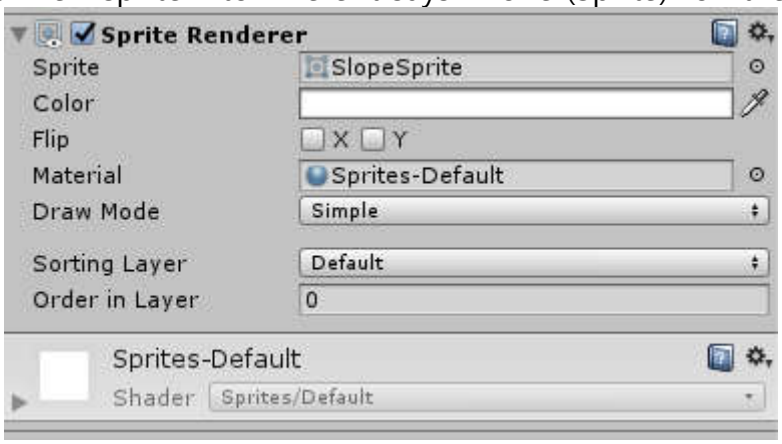
Select it and look on the left. There'll be a box something like this:



Change the numbers on the "position" row to all be 0. This is important, and will make all sorts of weird bugs if you don't do it. Also, click on the "Layer" dropdown and pick something suitable (or make something new if nothing is suitable. This is part of the ground, so I'll pick Ground.

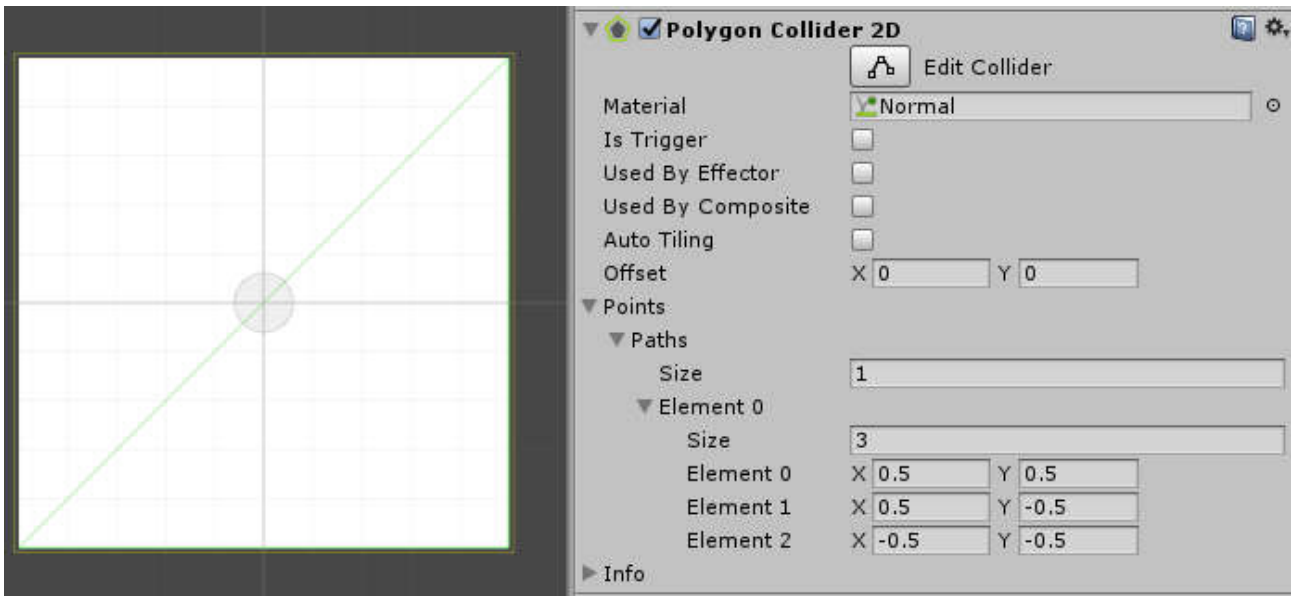
Now, we need to put in all of the things that will make this something other than just an empty game object. First up, some art: right click on the name on the left, and click "Create Empty" again. This should give you a new game object contained in the original one:

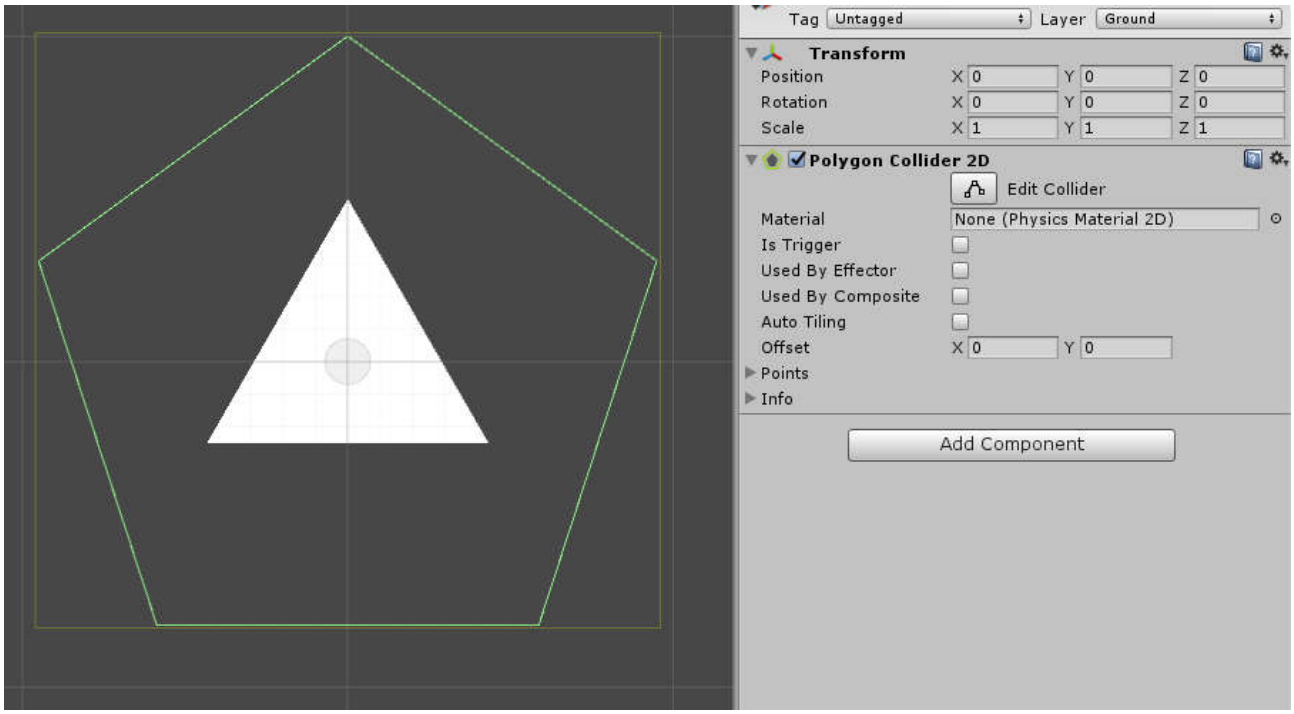
Rename it to "Art", and select it. Click "Add Component" on the right, and type in "sprite" and hit enter. We don't actually have any art for this yet, so we'll make a placeholder. In the project menu at the bottom, go to Assets/Art. Right click on the background in there, and click Create-Sprite-whichever seems most suitable. I'll use squares, just because it makes it more obvious what's going on in the next step (this will look ridiculous, but never mind). Call it something suitable – the others are all "[tilename]Sprite", so we'll copy that, and call this SlopeSprite. You can create a real image in there instead if you prefer, but this will do for now. Click on your Art on the left, and drag your new sprite into where it says "None (Sprite)" on the right.



You can fiddle with the colours here, but I wouldn't recommend it – it's much more effective for that to be done in the art instead. You can do it temporarily, though if it'll help – that's how my placeholders are coloured. I'll leave this one white. There should be a white shape in the middle screen now (if not, look around for (0,0) and it'll be there).

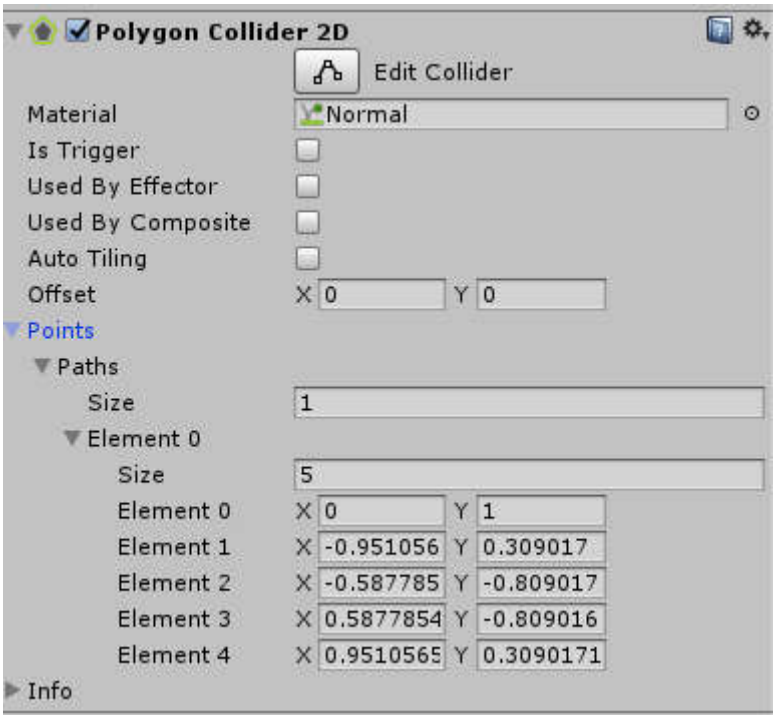
Now, go back to the top-level game object (the one you I called UpSlope). Click "Add Component" again, and type in "Collider 2D", but don't press enter. This will give you a bunch of options for Colliders – Box, Capsule, Circle, Composite, Edge, Polygon. If your tile is something where the shape doesn't matter or a rectangle, use Box Collider 2D. If it's circular, use Circle Collider 2D. For pretty much anything else, use Polygon Collider 2D. In this case, a slope wants to be triangular, which is neither round nor square, so we'll use Polygon Collider 2D. You'll get something like this:





Drag "Normal" from the Assets folder at the bottom into the "None (Physics Material 2D)" box. If this tile is something you want the player to be able to walk through (the points-giving tile does this, for example), tick the "Is Trigger" box. Otherwise, leave it as it is.

Click on "Points" near the bottom of that.

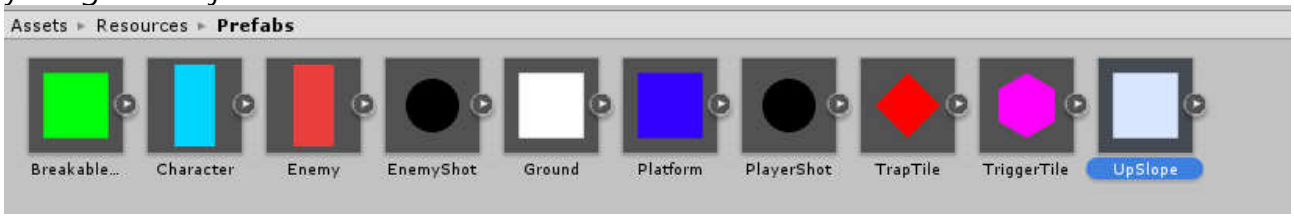


Now, we need to edit the points so that they match the shape – change to the right number of points with the second "Size" box, then fiddle with the rest of the numbers until they look right. In this case, it's fairly easy: we want 3 points, at (0.5,0.5) (top-right), (0.5,-0.5) (bottom-right), and (-0.5,-0.5) (bottom-left).

That green triangle is the bit that will stop us, so the bit that we can walk on.

This is a very simple tile, so that's all that we need – if we're doing anything fancier, we'll need to add more components. Key ones: anything that's moving around needs a Rigidbody 2D, anything that can shoot or be damaged needs a Combat Character, and anything that wants to do something when the player walks into it needs a Trigger Script. There are loads of other things, but those will come up more in the coding section. Some of those have settings, which should hopefully be mostly self-explanatory.

When you're done, open the Assets/Resources/Prefabs folder at the bottom, and drag your game object from the left section down to there.



To test that your new tile works, press play then immediately press pause. Select your object on the left (you might want to un-expand WorldController so that you can find it easily). Move the tile to somewhere you can reach it easily in game (changing the

Position on the right to 14,37,0 works with the current default map, or just drag it around using the move tool (four arrows in the top-left corner). Unpause, and make sure that it does what you're expecting when you run up it.

In this case, it turns out that this slope is too steep to run up, so we probably want to make a wider, flatter slope at some point, but it's otherwise fine, and this might be something useful anyway.

In this case, since we made a slope going one way, we should really make a slope going the other way too. We're going to be massively lazy here: we'll copy/paste the UpSlope game object on the left, rename the new one to "DownSlope", and change the "X" option on the scale row of its transform (top right) to -1. This just mirrors the whole thing horizontally. Drag this new one down into the Assets/Resources/Prefabs folder, and you're done.

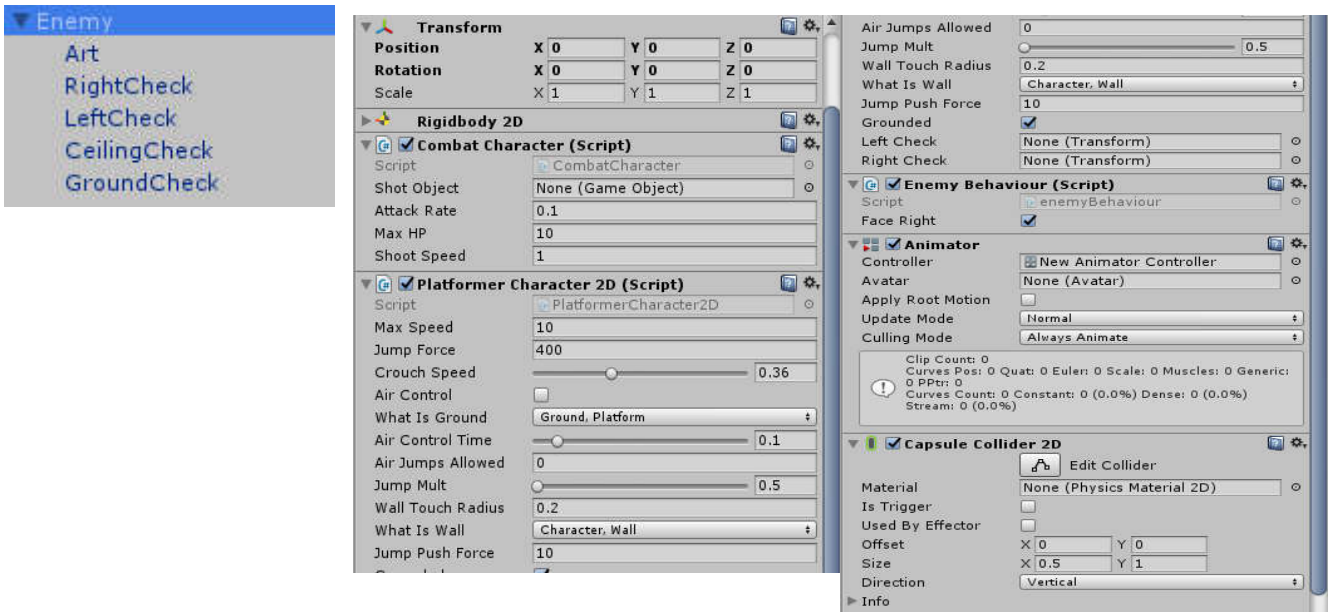
You probably want to add your new tiles to the map.txt file, with some colour (anything that hasn't already been used is fine), or you can just leave that for the level designers to do.

One last thing: try not to leave things sitting around in the scene. Notice how, at the start, there's only a Main Camera and a WorldController in there? Try to keep it that way. Delete your new tiles after you've dragged them into the prefab folder (they don't leave the prefab folder, so you don't need the one in the scene any more).

Save everything, do the GitHub thing, and move on to the next job.

Making Enemies

Enemies are a bit more complicated than tiles, so rather than making them from scratch, we'll just edit copies of the one that I made to start with. Drag the "Enemy" prefab into the scene. Set the position to (0,0,0) if it isn't already, and expand it out on the left.



There's a whole lot of stuff here. Firstly, note that there's an "Art" on the left, just like for our other tiles. This is exactly the same as there – it's got the Sprite Renderer in, which pulls in the sprite. Change the sprite out for whichever new one you're using for this enemy, or a new placeholder.

There's also four other child game objects – Left/Right/Ceiling/Ground Check. These just specify points on the outside that tell the code where the edges of the enemy are, so it can turn around when it hits walls, and tell if there's something above or below it. You should move them so that they match the actual size of the art on your enemy when you've changed that.

Important: LeftCheck and RightCheck should have their position set to be the same, but with the signs flipped. That's just the same as saying that the middle of your enemy should actually be in the middle. If that isn't the case, *weird* things will happen. Other than that (and moving CeilingCheck and GroundCheck up and down to match), you don't need to do anything with the children.

So, let's select the main Enemy game object and look at all of that mess on the right.

First off, we've got a Transform – like before, make sure that your numbers match the ones in my image if they don't already.

Right down at the bottom, we've got a Capsule Collider 2D: this is like the other colliders, except it's a slightly odd shape that turns out to work really well for humans. Swap it out for something that matches the shape of your enemy (keep it simple – rectangles are totally fine).

Next, we've got a Rigidbody 2D, which handles physics stuff. Don't touch anything in there (which is why it's minimised).

There's also an Animator – right now, that's doing literally nothing, and it's going to stay that way for a while. Animations are going to be a problem for later (or never).

The other three things are where the work is happening – Combat Character deals with health, damage, and shooting (though this enemy doesn't actually have the ability to shoot, since we haven't set a bullet – I've fixed that now, but it still won't shoot, for reasons that we'll get to later). PlatfomerCharacter2D deals with moving around – feel free to play with the numbers and see what they do (the enemy won't jump yet, either). Don't change "What is Ground" or "What is Wall", or bad things will happen. Left Check and Right Check don't actually need to be set to LeftCheck and RightCheck – the code sets them right when it loads.

Lastly, we've got Enemy Behaviour, which controls what the enemy actually does. At the moment, that's very little – it runs right until it hits a wall, then turns around and runs left until it hits a wall, and repeats. Writing a proper version of this so that enemies can actually do things is going to be one of the first jobs for the coding team.

Once you've done fiddling, rename your enemy to something sensible, then drag it down into the Prefabs folder. Save, do the Github thing, etc.

Other Useful Components

Attach a **Projectile** component to anything you want to be shot – it does the damage stuff. If "damage scaling" is ticked, it will do more damage if it hits faster, otherwise it's a flat amount.

Effector 2D components do some classic platforming things – Platform Effector 2D lets you go through one way, but not the other. Surface Effector 2D makes conveyor belts, Point Effector 2D does points that drag you towards/away from them (magnets, gravity, whatever), Buoyancy Effector 2D gives water that you can float in, and Area Effector 2D lets you add force to anything that goes into the area.

Trap components just deal fixed damage to anything that touches them.

The **TriggerScript** component can do a whole bunch of things – it can give (or remove) points, end the level, and will be able to trigger conversations, change the map, and let you pick up items, when the coding team has finished writing those things.

Joint 2D components attach game objects together in various ways. Unlikely to be useful for us, but it might be.

Constant Force 2D makes something constantly accelerate in a certain way – use this for things like rockets.