Tilemaps Lab

Table of Contents

Overview

A Tilemap is a Unity asset that allows you to quickly create new maps and levels for games that use 2D assets, such as 2D platforming games. Tilemaps is still relatively new, so while certain assets, such as Tilemaps, are built into Unity, other assets, like Rule Tiles, need to be downloaded separately from Github.

In this lab, you will learn how to create and implement a Tilemap.

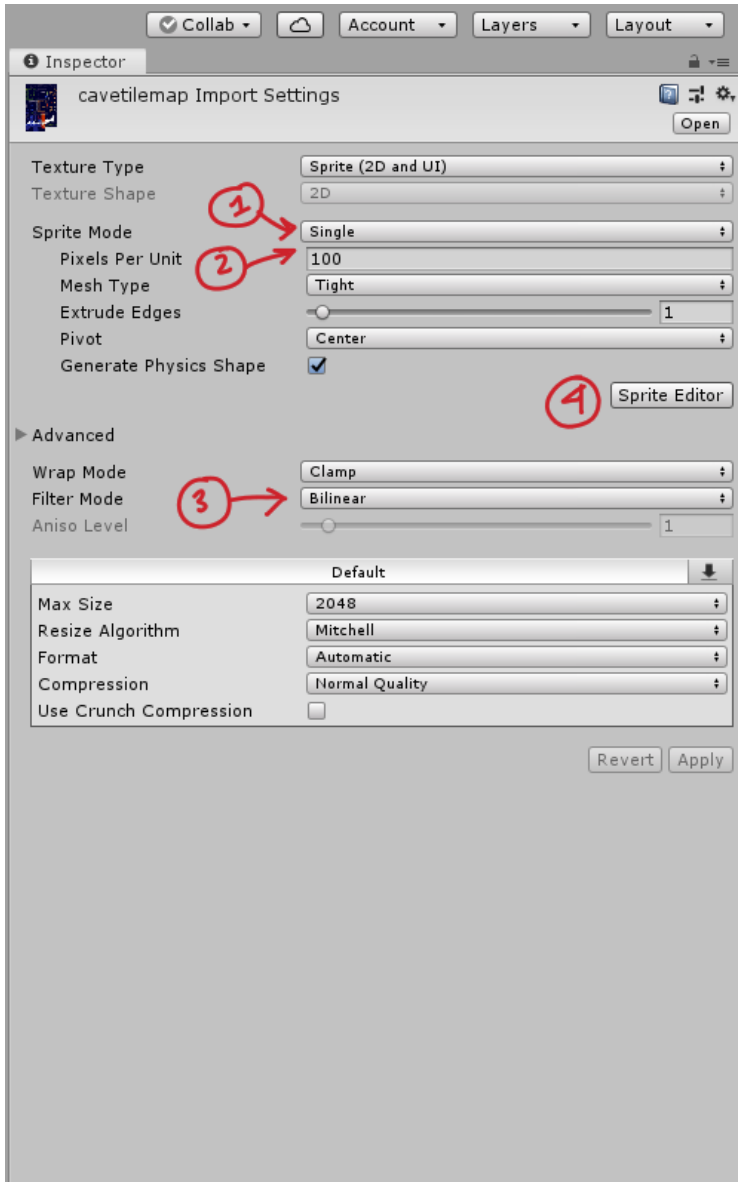
Basic Tiles and Tilemaps

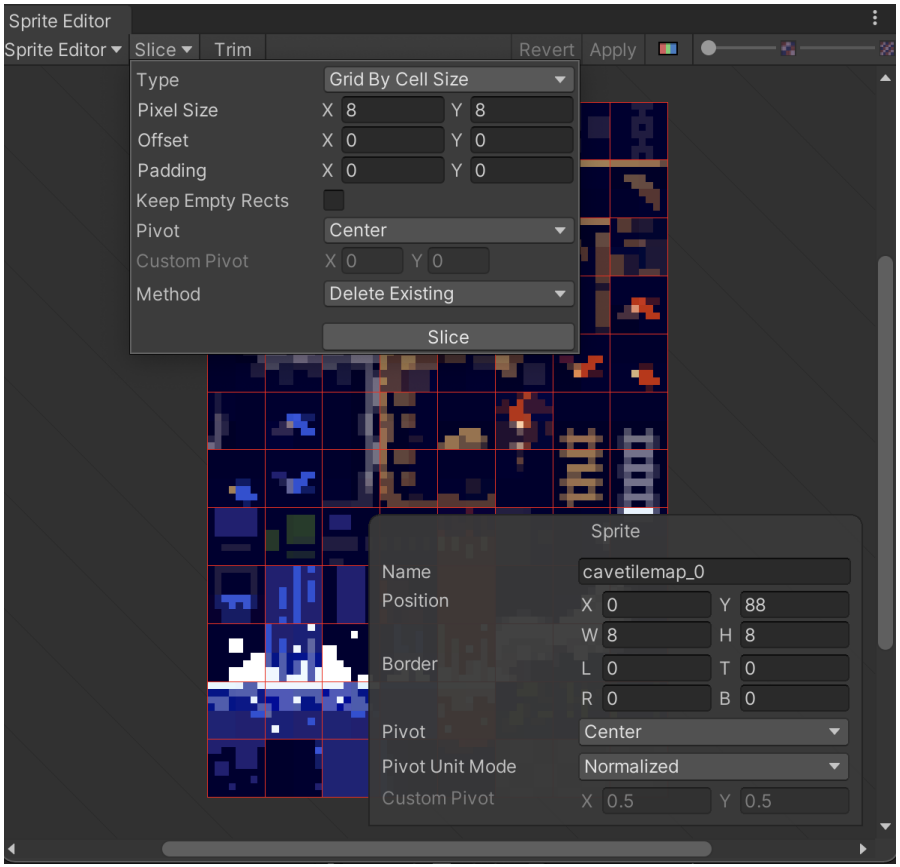
To start, since Tilemaps require art assets, we are going to import some Tilemaps to work with. Inside the zip folder, there are several sprite sheets. We are going to import all of them into the Sprites folder and split them up so they can be usable assets in later steps. To use images as sprite sheets for the Tilemap, we have to do a few extra steps to split up the images.

For the first part of the lab, we will be using the cavetilemap.png, click on it to see this in the Inspector.

First, we need to change **Sprite Mode (1) to Multiple** so that we can split up the image into multiple Tiles. Since this image is a bit small, we need to change the **Pixels Per Unit (2) to 8**. To maintain the crispness of the pixel art, set **Filter mode (3) to Point (no filter)**. Click on apply to apply the import settings to the image.

Now, to actually split up the Tiles. Click on **(4) to access the Sprite Editor**. It will open the Sprite Editor which we can use to slice the image.





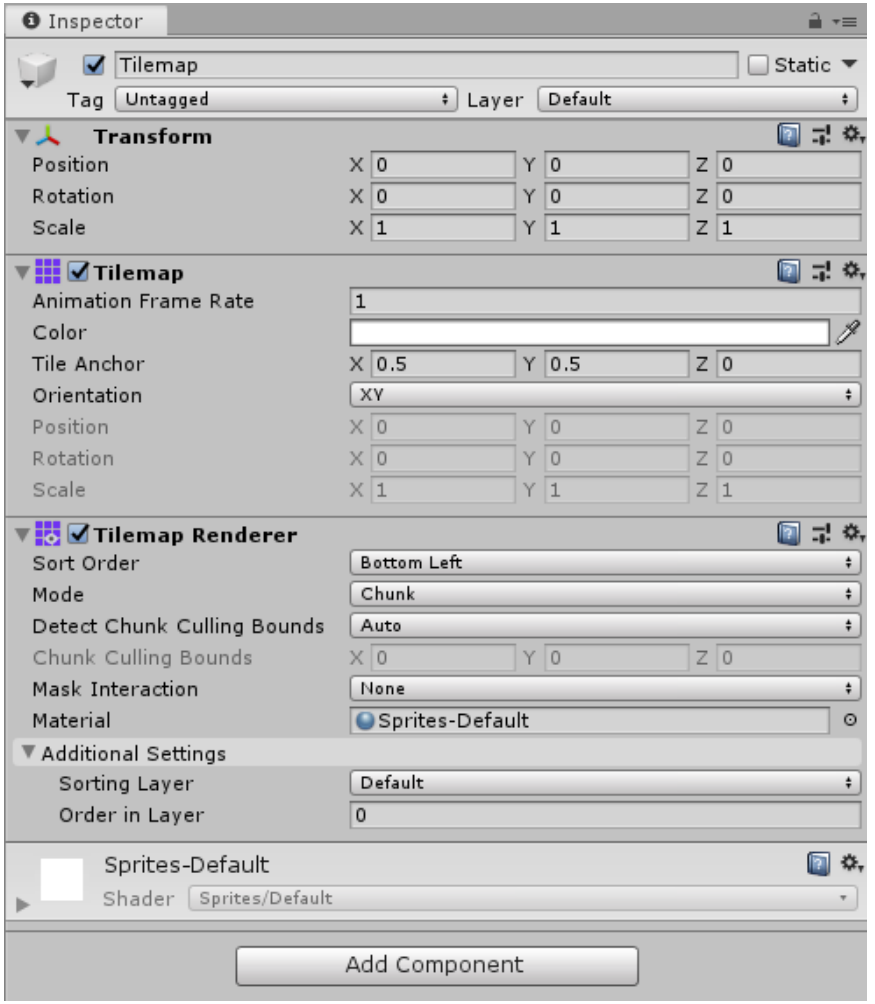
In the top left-hand corner, there is the slice button that will slice Spritesheets. **Slicing** is when you cut up a large Spritesheet comprised of many sprites into smaller sprites. In this case, our sprite sheet has things like ladders, floors, waterfalls, etc., so we want to "slice" them into their own individual elements to use. Usually, keeping the slice setting to automatic is fine, but for this sprite sheet, we need it a bit smaller. Set the type from **"Automatic" to "Grid by Cell Size"** and set the pixel size to 8x8 and click **"Slice,"** then press the **"Apply"** button to apply your change.

Now we can create our Tilemap. In the hierarchy, right click and navigate to *2D Object > Tilemap > Rectangular*. It will create a Grid that will layout the format for our Tiles. Inside the Grid GameObject, there is a child called Tilemap. That is going to be the Tilemap that we will be working with. For now, we can rename the Tilemap to be Ground, since that will be the first thing that we will be making.

The Grid is essentially a UI grid, where it stores objects in each cell. The Tilemap is made up of 2 different components: **Tilemap** and **Tilemap Renderer**.

On the Tilemap component, there are variables to control how it appears. The tile anchor determines how tiles are positioned relative to the grid spaces. The orientation determines the x and y direction. The Tilemap Renderer is similar to normal Sprite Renderers.

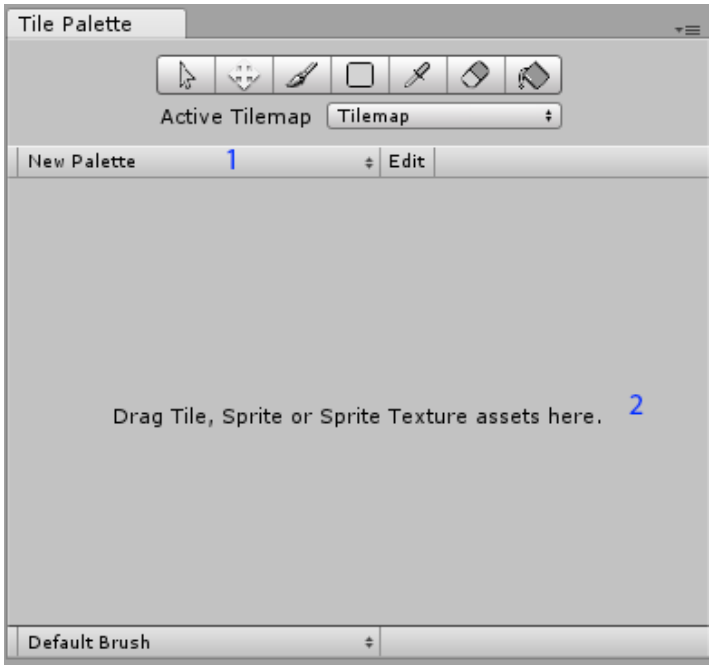
To actually work with the Tilemap, we need a new window. On the top bar next to "Help," navigate to *Window > 2D > Tile Palette*



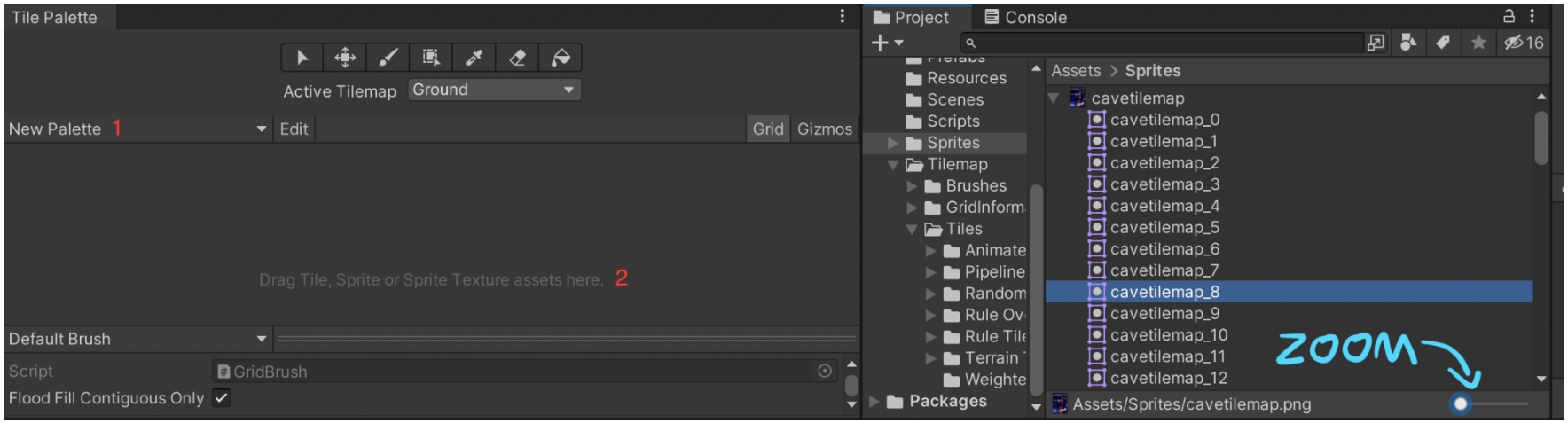
Tile Palettes are a collection of tiles, in the same way an artist's palette is a collection of paints. This is where your tiles will be kept and largely for organizational purposes. Tiles are objects that make up the Tilemap. They hold the sprite that will be shown, as well as the positioning of that sprite. This is similar to a sprite renderer of a game object. A Tile Palette can be used on multiple Tilemaps and a Tilemaps can be made up of multiple Tile Palettes.

Create a new Tile Palette by pressing on (1) and name the palette Grass. There is an area where you can place new Tiles and select them to put them into the scene. The Tile Palette has several options as to how you can paint the tiles into the scene. On the toolbar, there are several options. From left to right:

1. **The mouse button:** inspects the Tile and allows you to edit tile properties
2. **The selection button:** can select and move Tiles
3. **Paint brush button:** the paintbrush allows you to paint onto the scene with the Tiles that you want to be placed into the scene
4. **Paint filled box:** will fill the area within a box you define with the selected tile
5. **Dropper:** will select the tile of whatever is selected in the scene
6. **Eraser:** erases the Tiles painted into the scene
7. **Paint bucket (flood fill):** will fill in the inside of a box that has a boundary



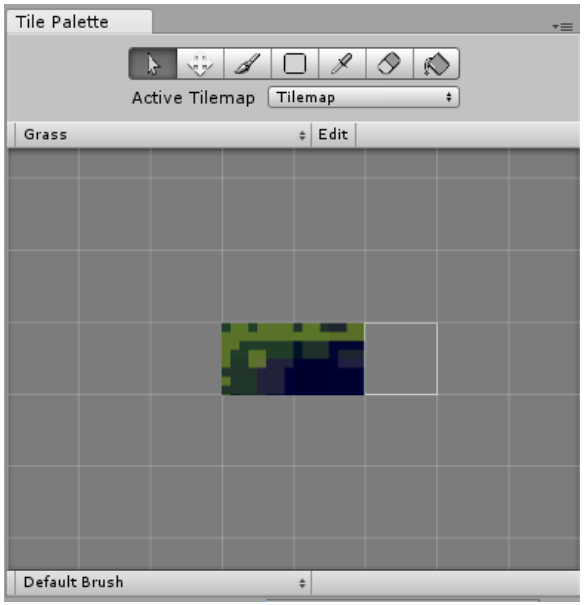
We are going to set up a grass Tile Palette. The image is a bit small, but using some naming conventions, hopefully it will be a little easier. There is also an image that is much larger in the zip folder if you'd like to use that. In the sprite sheet we just split up, we are going to use cavetilemap image 8, which is a corner Tile. To make it easy to find, zoom out in the Project view and expand the Spritesheet you sliced and find cavetilemap_8.



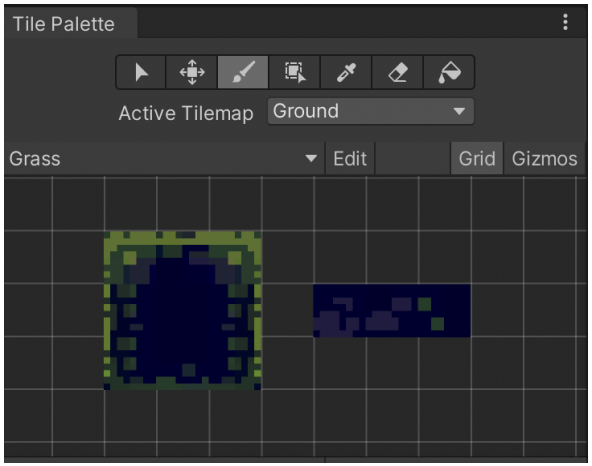
Drag the sprite from the Project window into the blank space in the Tile Palette (2 in the previous picture). This will automatically create a new tile object using our sprite. We can create this into a new folder called Tiles. Name it ”8” so that we can access it easily later.

Now we can see that we have our corner tile in the Tile Palette. If we click on the paintbrush and click and drag in the scene, we can see all of the corner pieces fill the tiles we drag by. Clicking the eraser and doing the same thing will remove it. If your paintbrush and eraser aren’t working, make sure you have the Edit toggle on.

Let’s put a few more tiles out so that we can complete the set of grass Tiles. The next tile that is going to be placed is cavetilemap_9. For organization, we are going to put it next to tile 8 so that the Tile Palette looks like this:



You can name this tile just by 9 too, just so that it is easier to get to. The next tile is going to the opposite corner tile. If you look at the actual sprite sheet, it doesn't actually have all the pieces to create the full square Tilemap as follows:



So, we are going to reuse tile 8. Now, since “8” has already been made, we can just drag that into place to the right of 9. However, this one is flipped in the wrong direction. So, we click on the tile while the mouse button is selected and then move to the Inspector window. Here we see the position, rotation and scale. Similar to other sprite renderers, we can use this to flip the image around. Under scale, change the x value to -1 and the sprite will flip. That finishes off this row quite nicely. Now, fill in the rest of the grass palette so that it looks like the picture above. The numbers for each tile is:

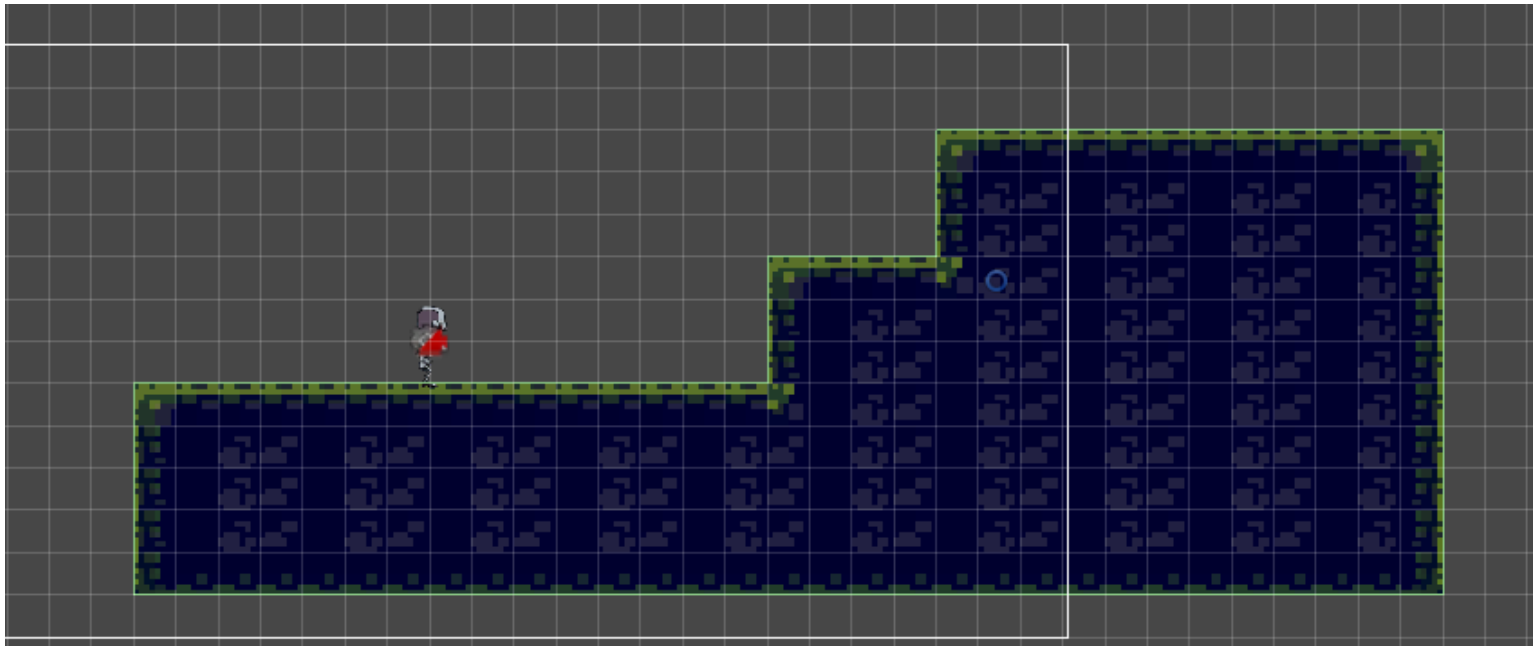
8	9	8’
16	0	16’
24	25	24’

’ refers to the flipped sprite.

Off on the side, there are additional sprites that we will use to fill the center. Those are: 1 2 17.

Now, we can paint in the grass for the level. Select the paintbrush, click on 8, and place it into a box on the screen. Click 9, click and drag along a straight line. Click 8’ and place at the end of the tiles. Click on 16 and then draw a couple of squares down under the tile for 8. Do the same for 16’ on the other side. Click on 24 and make the corner. Do the bottom boundary for 25 and fill to the end. Fill last corner with 24’. Click on the paint bucket button and select 0, then hover over the middle and fill it in with the tiles.

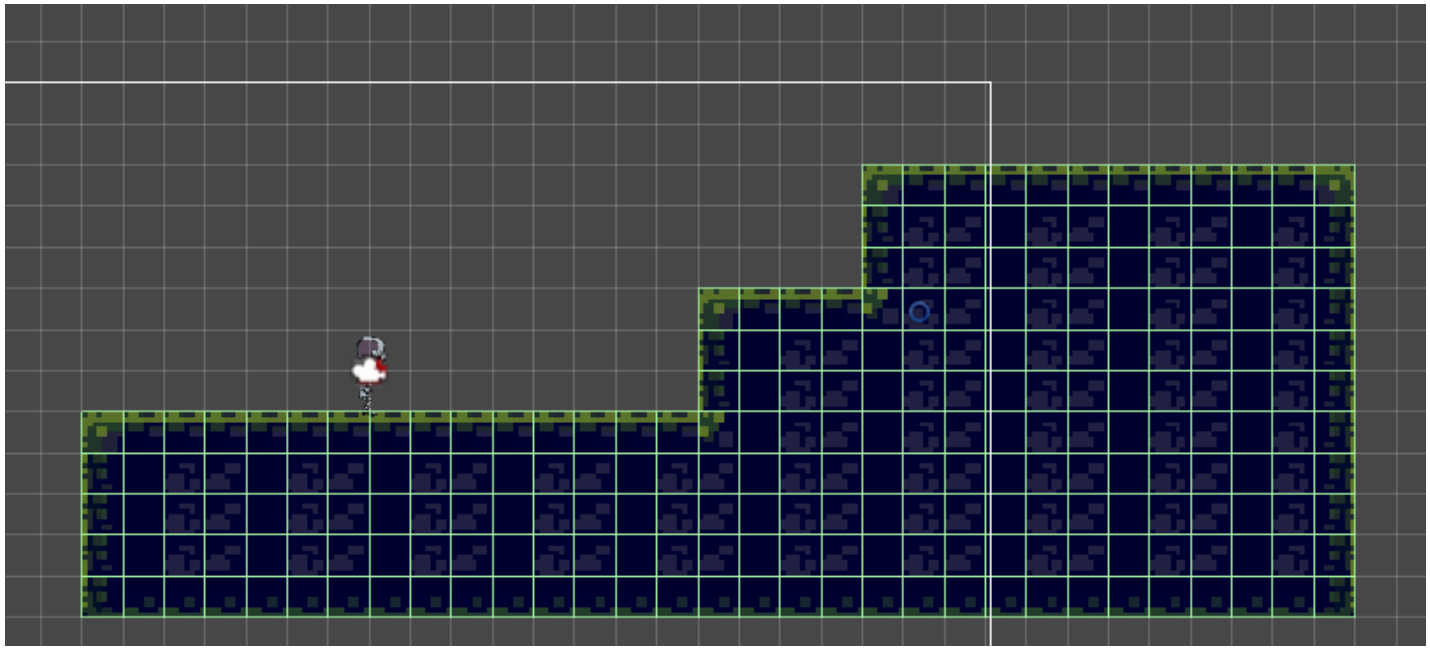
You don’t have to match this, but this is a reference if you want something to go off of for the level:



You can also use the box fill button to fill up the space so that we don’t have to fill in the whole box by hand.

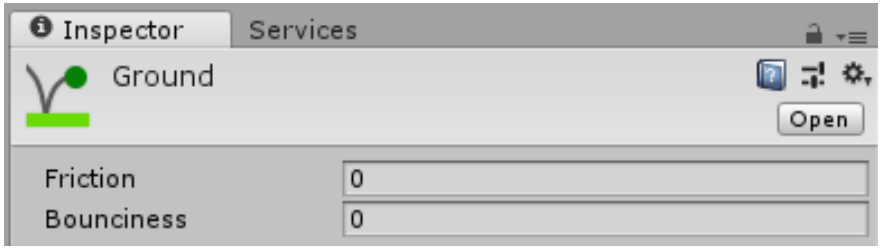
We can also select multiple tiles to put into the scene at once. If we select both tiles: 1 and 2 inside of the Tile Palette with the selection tool we can ”stamp” them onto the scene using the brush tool.

Drag CharacterRobotBoy from the Prefabs folder into the hierarchy and hit play. You’ll see that the player falls right through the ground. That is because there aren’t any colliders on the ground right now. To fix that, add a **Tilemap Collider 2D** component on the Tilemap (which is named Ground) GameObject.



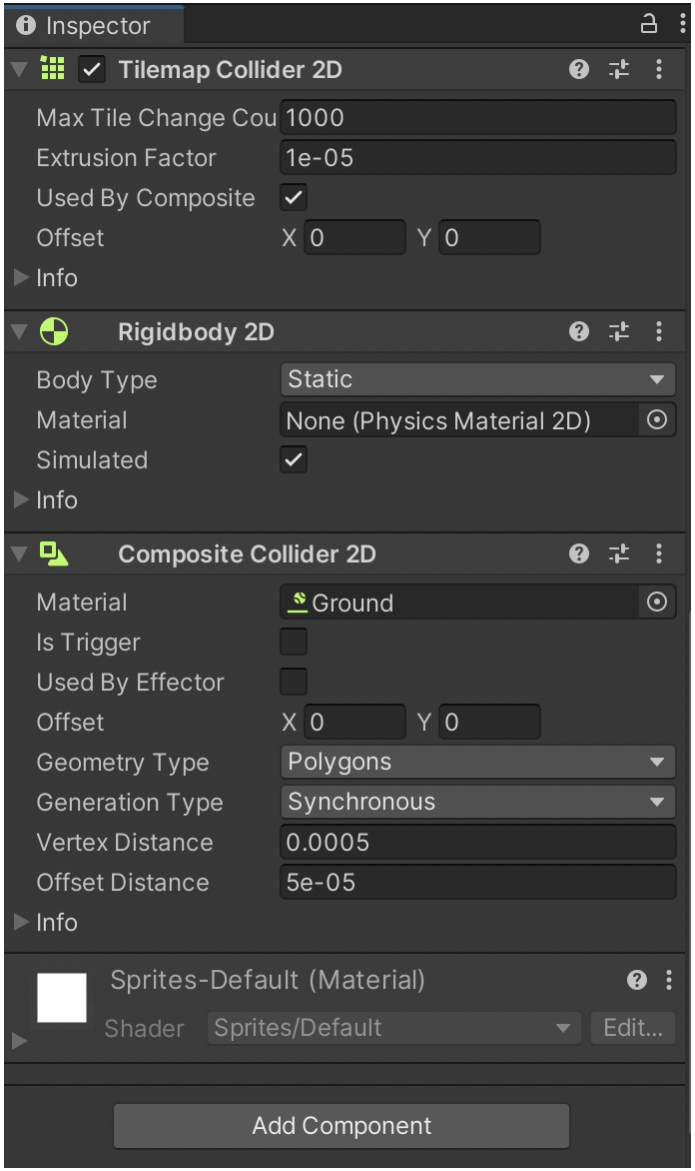
Looking at all of the objects, each individual tile has its own collider. We can reduce all of this to just the outline of the large block by also adding a **Composite Collider 2D** onto the Ground GameObject. Be sure to click “Used by composite” under the Tilemap Collider component, or else it won’t work. By adding this, the colliders will only be on the outside of each large block. Depending on the type of game you’re making, it may help with collision detection to change “Geometry Type” from “Outline” to “Polygon.” This will make the tiles act as solid blocks rather than hollow boxes. Adding a Composite Collider also adds a **RigidBody2D** onto the GameObject. On the RigidBody2D, change the body type to be static. We want our tiles to float indefinitely, unaffected by gravity.

Play it now and the player can run along and jump on the platforms! However, the player will stick to the walls if you jump into a wall. To get rid of this, we need to create a custom material for the Composite Collider 2D. In the project window, go to the base folder, right click and create a new Materials folder, and then *Create > 2D > Physics Material 2D* in the folder and name it Ground.



In the material, there are 2 options. **Friction** determines how much the player will stick to an object (like the player sticking to the wall). **Bounciness** determines how bouncy the collider will be. To stop the player from sticking to the wall, set the friction to 0. Then add the material onto the composite Collider 2D component on the Tilemap GameObject.

We’re done! The Tilemap GameObject components that we just added should look like this:

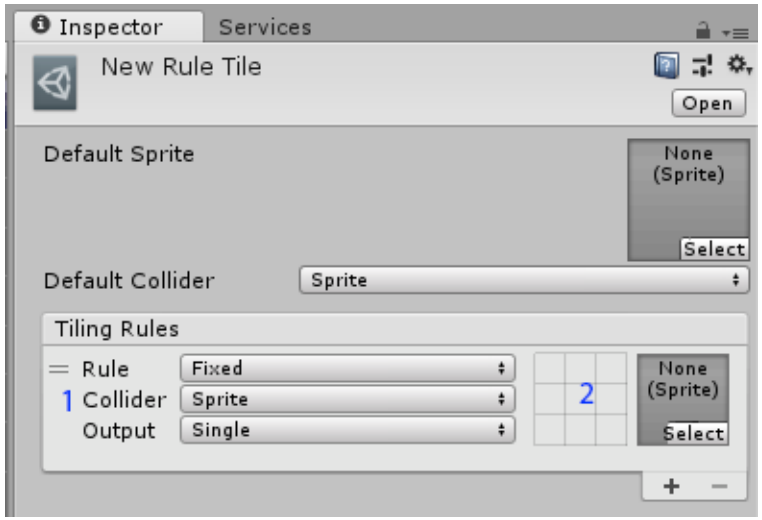


Rule Tiles

Rule tiles are essentially a tile that have a set of conditions so that Unity will automatically place tiles into the scene based on their proximity to other tiles in the scene. They are very powerful and make level creation super fast and easy, but takes a bit of set-up.

We are going to need to import some assets from a Github repo. For this lab, they are already imported into the project, but if you want to use them later on, they can be found here: https://github.com/Unity-Technologies/2d-extras. Inside the Tilemap folder, there are a bunch of assets that help us create Rule Tiles. There are also a lot of other assets within this package that we won't cover in the lab, but are definitely worth taking a look at later.

Inside the Tiles folder, we are going to right click and navigate to *Create* > *Rule Tile*, and we can name it Stone because we will be using the stone tiles. With the Stone Rule Tile selected, view the Inspector. Notice that we have a default sprite and a list of conditions that will determine the sprites that will be used.



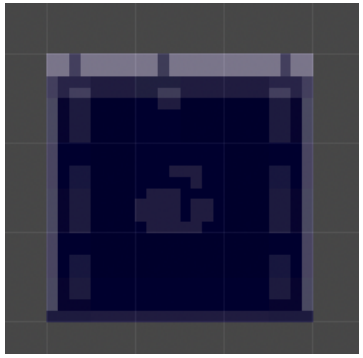
When using Rule Tiles, you don't actually need to create new tiles. We need only add existing sprites onto each set of rules. For the default sprite, let's use image 0 from cavetilemap. Find it in your Project and drag it into the Default Sprite slot.

Now we can start creating rules. Click the plus, and you will get the option to create a new rule. There are a few properties that we can use. (located at 1)

1. Rule allows us to set instances for which we want that particular rule applied. For example, we can make it so that a rule can be applied when also turned 90 degrees or flipped on a particular axis. If that seems confusing, I'll go over it with an actual example in a bit.
2. Collider is just the type of collider that should be attached to it. For our purposes, we just leave it set to sprite.
3. Output is how we want to be painting that specific rule onto the scene. Single gives you the image as is, Animation cycles through multiple sprites to create an animation, and Random picks one random tile from a set. We will be going through all three output options in this lab.

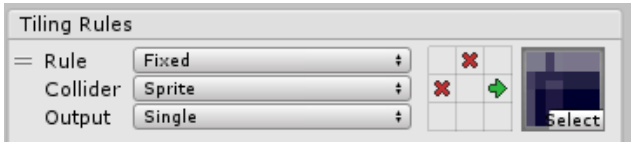
In the grid (2), all boxes outside of the middle one can be set with a condition that applies to the tile spot relative to the tile. If you click it, it will set it to a different setting. When it is an **X**, that spot must be empty. When it has an **arrow**, that spot must have another tile from *this* Rule Tile. It is important to note that tiles from other Tilesets will not be counted when seeing if there is a tile present in the spot adjacent to a tile. When it is blank, it does not matter if there is something there or not. For example, if the top left box has an X, then this specific tile will be placed only if the top left corner next to this tile does not have a tile on it.

So, let's make our first set of rules. We want it to look like our grass tiles in the style of the tile layout, so it should look like this:



The sprites we will need are: 11 for the upper corners, 12 for the upper middle, 19 for the middle walls, 0, 1, and 2 for the middle, 27 for the bottom corners and 28 for the bottom middle.

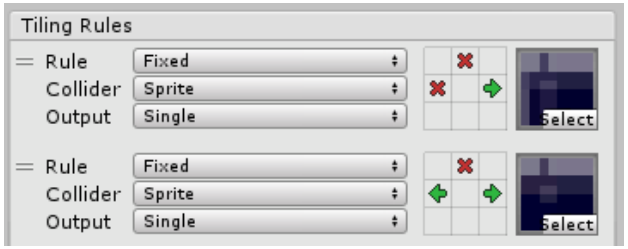
For the first corner tile, select cavetilemap_11 as the sprite for the first rule. Then, we can start with the conditions. Since we want this tile to show up when there is nothing on the left and top, mark those with Xs. You can change the icon in the grid by clicking on that specific part of the grid. We also want to make sure that there are blocks to the right of the corner block at least. In theory, we also want to require there to be a tile underneath it, but if we have a platform of just one layer of blocks, we want the top layer to be the layer that is shown. It should look like this:



Now, let's add another rule. Click the plus button and add a new rule. A thing to note is that rules are processed from the top down. If a tile fits into a category, it will be chosen even if it also fits into a later category from the list.

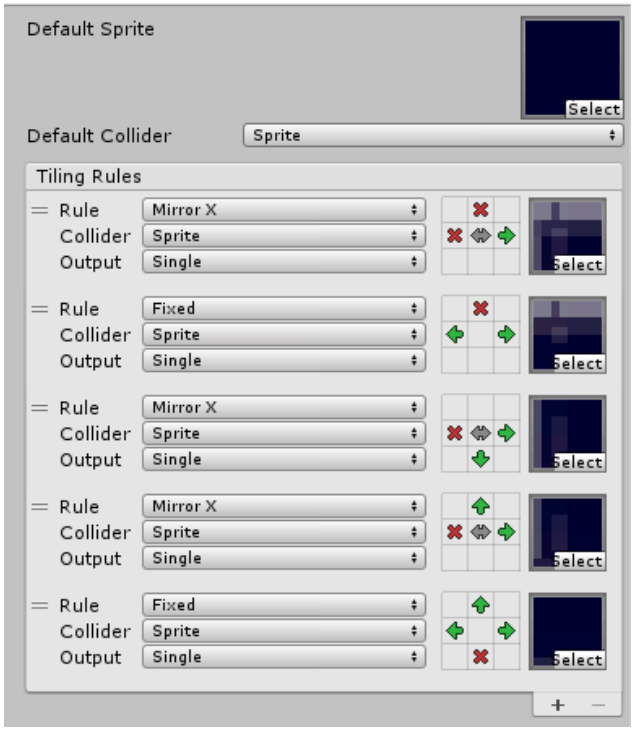
For tile 12, we want to make sure that it shows up when there is a tile to the left and right of the tile, as it is a center tile piece.

We want to make sure the top is empty. So now the rules should look like this:



Now for the opposite corner. Here, our work is cut out for us. Previously I mentioned the Rule section determines how the rule can be applied. The top right corner functions the same as the top left corner in terms of sprites and conditions, just flipped on the x axis. So we change the Rule of that tile so that when the conditions for the top left corner are satisfied, but for the top right corner (flipped on the x axis), it will flip the sprite along the x axis as well. To do this, click on Rule and change it to **Mirror X**. Or click the center of the grid until you see a double headed arrow pointing left and right.

Now, if you try to paint something, you can see that it will automatically determine what type of tile should be placed there. And that the top right corners are being put in place just by mirroring the rule for the top left corner. Great! Now fill in the Rules for 19, 27, and 28 in that order so that it looks as follows:



To draw with the Rule Tile, open up the Tile Palette and create a new palette called “stone”. Drag the stone Rule Tile you just created into the Tile Palette, and select the brush to paint. Notice how the tiles will automatically correct to fulfill the rules you just set.

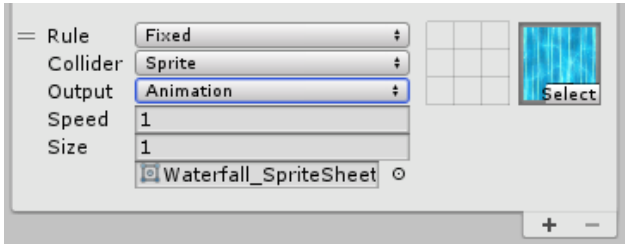
This is much more efficient! But the middle seems a bit plain with only the blank default tile. So, let’s change it up with random output Rule Tiles. We can add a new tile onto this ruleset. For this one, we want to change the output to Random instead of Single. This will create some new parameters: Noise, Shuffle and Size. Noise determines how random it is, shuffle is how it can change the sprites (like flipping on an axis), and size is how many random sprites you are giving it.

Let’s set size to 3. The 3 sprites we will use are 0, 1 and 2, so drag those into the 3 slots. We can set shuffle to Mirror X for a bit more variation. You might notice that these changes are happening as you are making settings, so that is definitely very helpful to see how changes work out. Now, you can slide the slider for the noise and see what you think looks good.

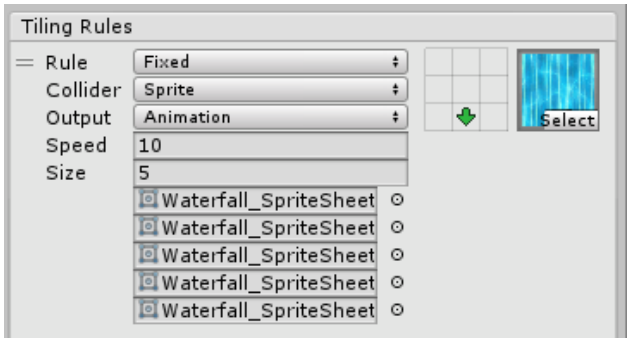
Animation Rule Tiles

Great, now we've been able to make levels easily, let's make something a bit more pretty. We can also make animation tiles using Rule Tiles. For this though, we need a new tile set that was made specifically to be a waterfall animation. If Waterfall_SpriteSheet.png hasn't been imported yet, import that. Again, we will have to change some settings for it. Set Sprite Mode to Multiple and Filter Mode to Point (No Filter) as before, for now, **set pixels per unit to 128**, and when you slice it, the size of the cell is 128x128. This Sprite Sheet should produce 10 tiles, 5 for the body and 5 for the bottom.

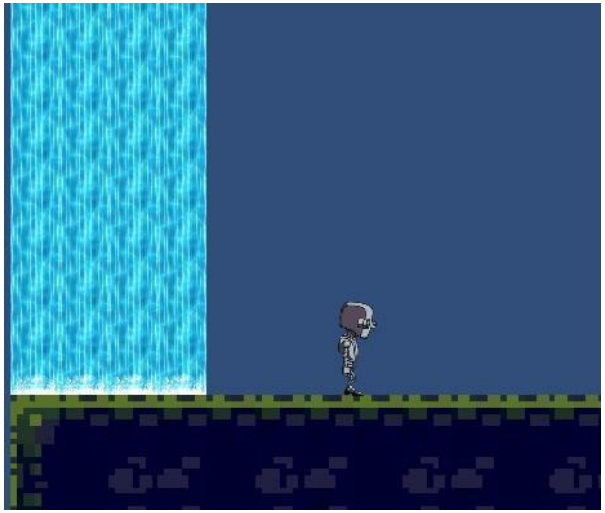
Now, let's make a new Tilemap because we don't want colliders on this layer. Make a new Tilemap called Scenery. Create a new Palette and a new Rule Tile for the waterfall. The conditions are simpler for this one. We just need to check if the tile is the bottom-most tile, we'll place the tiles with the splashes there. The first condition is making sure that there is a tile directly underneath it. Then, change Output to Animation. Then, that will create Speed and Size variables.



Speed is just how fast it cycles through and size is how many images there are in the animation. So, set Size to 5 and place all of the images in numerical order into the slots under size. This rule should look like this:



If you place them onto the map and press play, you can see that they will cycle through the images, but a bit slow. Change the speed to 10 and it is a lot better. We need the bottom of the waterfall as well. Add a new condition for the bottom of the waterfall. Now, the tiles that were originally painted are already automatically updated to reflect the new rule. Since pdf technology hasn't advanced to hold animations, I can't show you the waterfall in motion, but here's what a frame looks like:



Supplementary Lecture

A few notes about Spritesheets in general, usually sprites will require some tweaking to be able to fit it into the grids of the Tilemaps, to have it look good or line up with other tiles if you have smaller details you want to add in using Tilemaps. It typically is just tweaking the pixels per unit or changing the position, rotation or scale of the tile. It will take some time and needs finicking so don't worry if something doesn't just fit right away.

This supplementary lecture provides a peek into Unity's official 2d-techdemos. Take the time to follow along and explore some of the advanced Tilemap technology that 2d-techdemos has to offer. There is plenty more that can be done. Rules can be scriptable so that more conditions (other than just if a tile is present or not) can be applied. Paintbrushes can even be scriptable to create prefabs or create other properties. We'll demonstrate all these things with the help of 2d-techdemos' sample assets.

The lecture is linked here: https://youtu.be/zIEkINKfm_w

Checkoff

1. Create a level! Use platforms made of grass and stones. The level must contain Rule Tiles with animations and randomness.
2. Place some torches (45) and animate some birds (31, 38, 39, 41, 48, 49), all in the background layer.

Challenges

1. Find a new Spritesheet, import it and create a level. You could use Unity Asset Store as shown in the mini-lecture, or other sites like https://itch.io/game-assets/free/tag-Tilemap or https://opengameart.org/content/best-orthogonal-rectangular-tilesets-for-Tilemaps
2. Make a Terrain Tile, which is like a Rule Tile but has predefined scripting that is really useful for making quick and easy terrain.
3. Create some scripted rules and create some new Tilemaps using more specialized rules