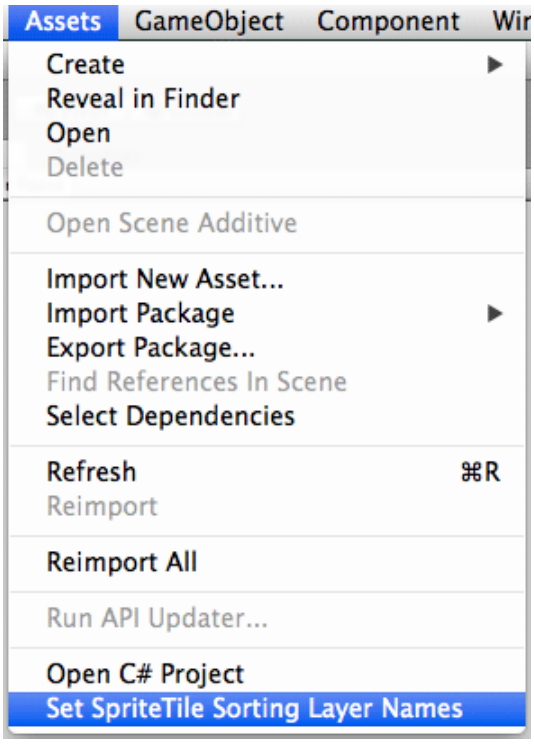
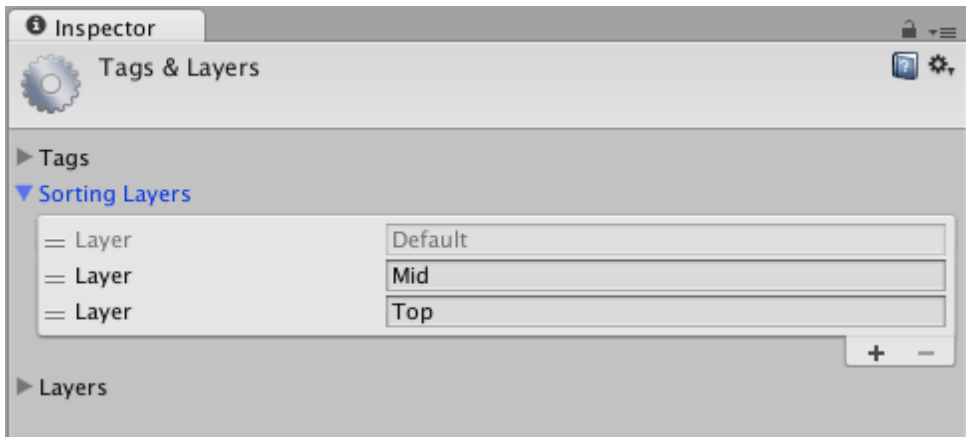
SpriteTile QUICK START

Welcome to SpriteTile! There's a lot to learn when just starting out, but let's jump in and cover the basics so you can get going quickly.

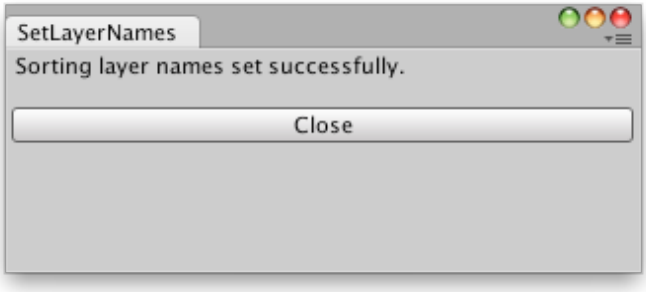
To start with, make sure you have the SpriteTile dlls or source code imported into your project. You'll need to do a bit of prep work if you're planning on having more than one layer in your levels. In Unity, go to the **Tags & Layers** editor (Edit -> Project Settings -> Tags and Layers), and add as many sorting layers as you expect you'll be using. You can always add more later if needed.

When you're done, select **Set SpriteTile Sorting Layer Names** from the **Assets** menu. This tells SpriteTile about the sorting layers, so it can use them properly.

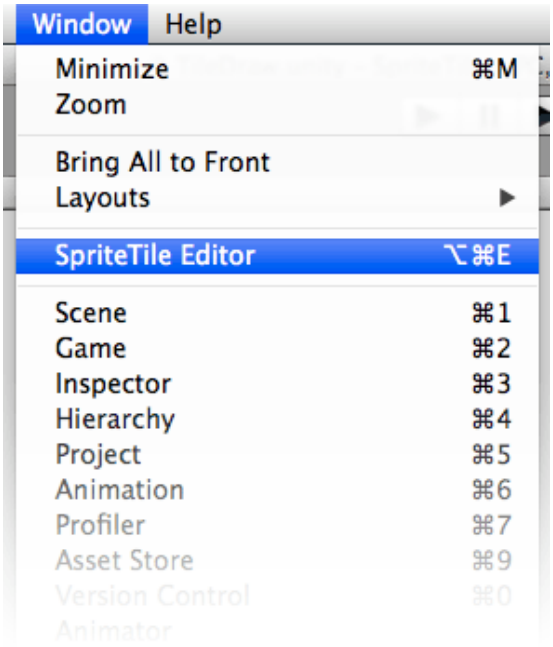




When that's done, you'll get a confirmation message:

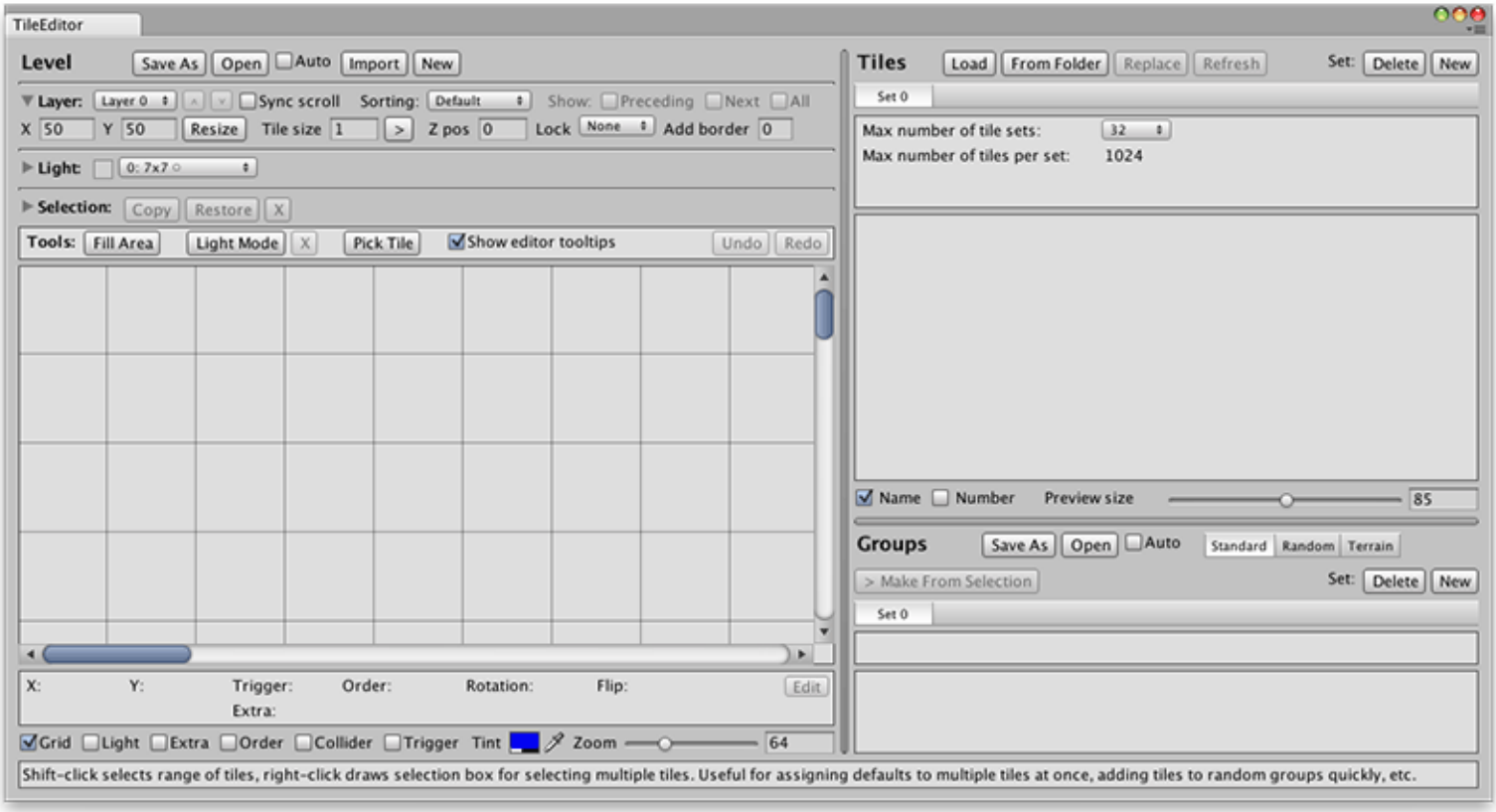


If you add more sorting layers later, or change existing layers, just run the Set SpriteTile Sorting Layer Names menu item again.

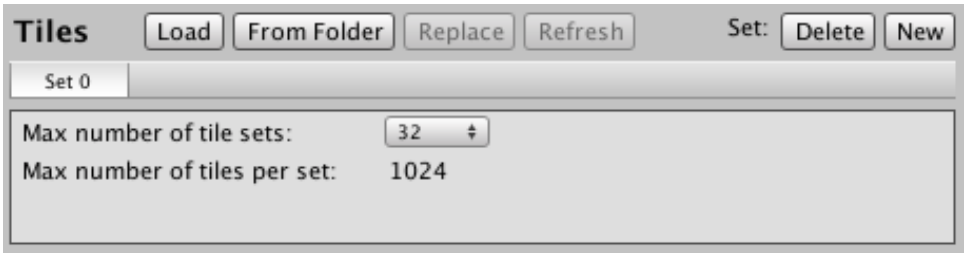


Now we'll bring up the editor. Go to the **Window** menu and choose **SpriteTile Editor**. This opens the SpriteTile editor, which has sections for editing the level, managing tiles, and managing groups. You can resize the editor window as desired, and drag the horizontal and vertical dividers between sections to adjust the amount of space you want for each section.

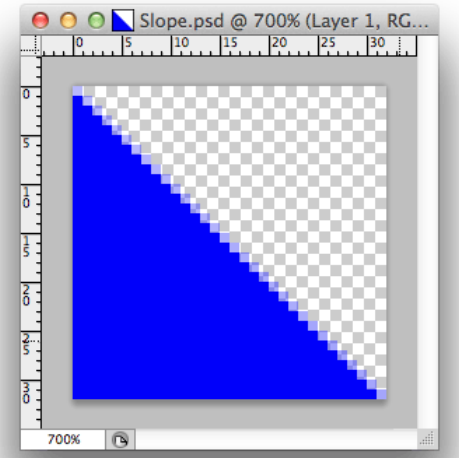
The first thing you'll need to do is decide how many tiles will be in each set. SpriteTile supports 32,768 different tiles, by default divided into 32 sets of 1024 tiles each. Aside from organization, this allows you to easily do things like switch all the tiles in a level from one set to another—for example, think alternate day/night tilesets. (See the SetMapTileset function in the SpriteTile Reference guide for more details about this.)

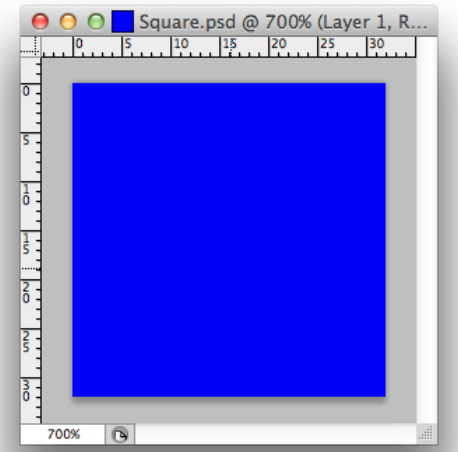


If you'd rather have a different number of tiles in each set, choose an appropriate number from the pop-up menu.

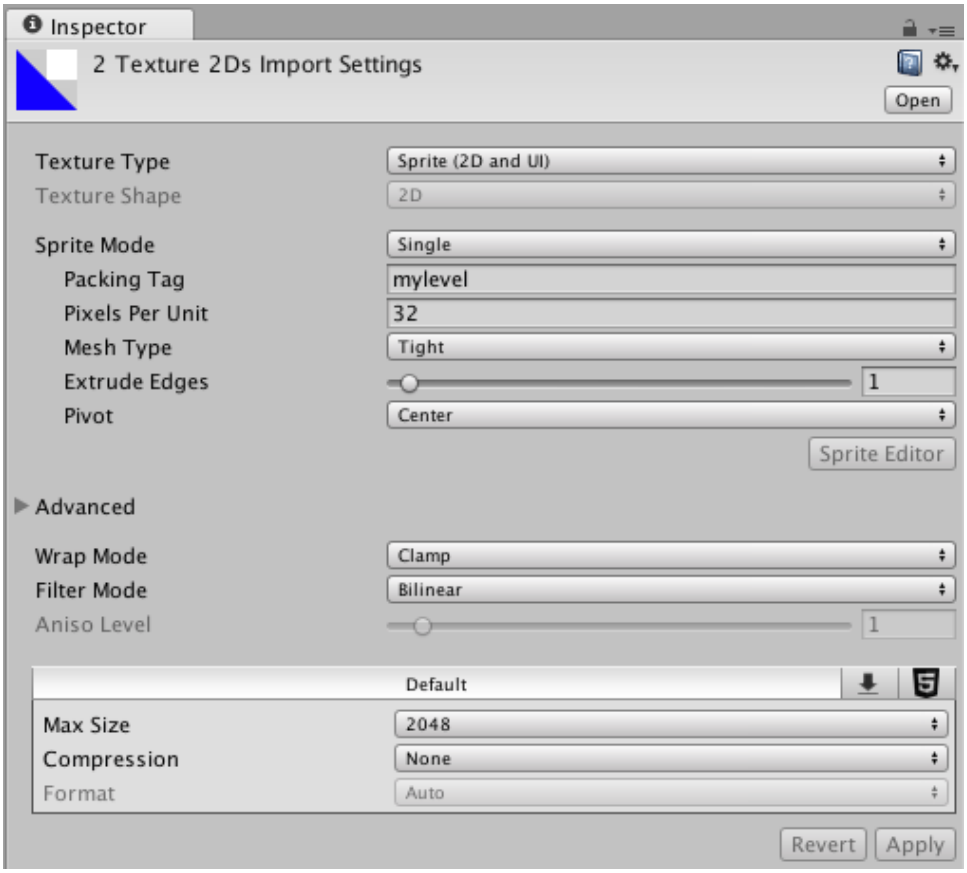


Now let's add a couple of tiles. You can make tiles in any sort of paint program. You can have them be part of a texture atlas, or save them individually. Unity will automatically create atlases from individual sprites as long as you give them a packing tag.



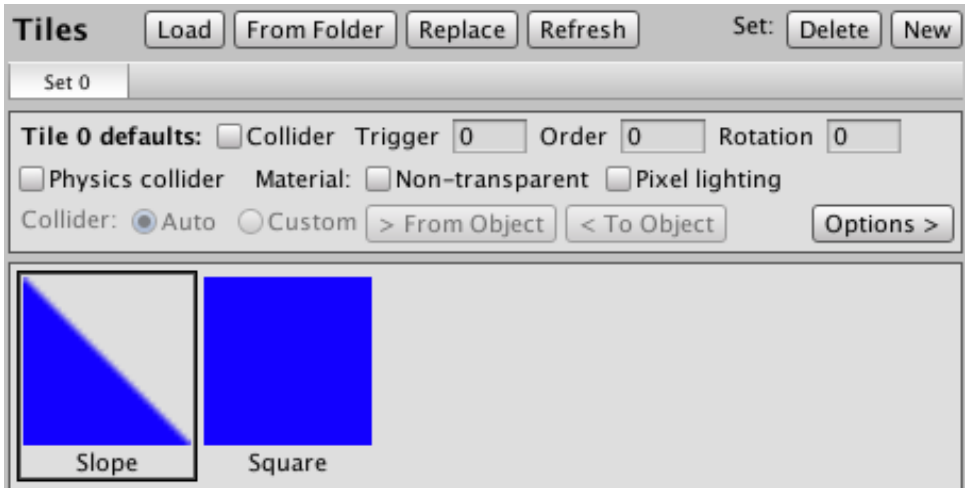


We'll do something really simple here...a slope tile (with a transparent background), and a square tile. We'll make them 32x32 pixels in size. Save them in your project. In Unity, make sure the tiles are set as the Sprite texture type.



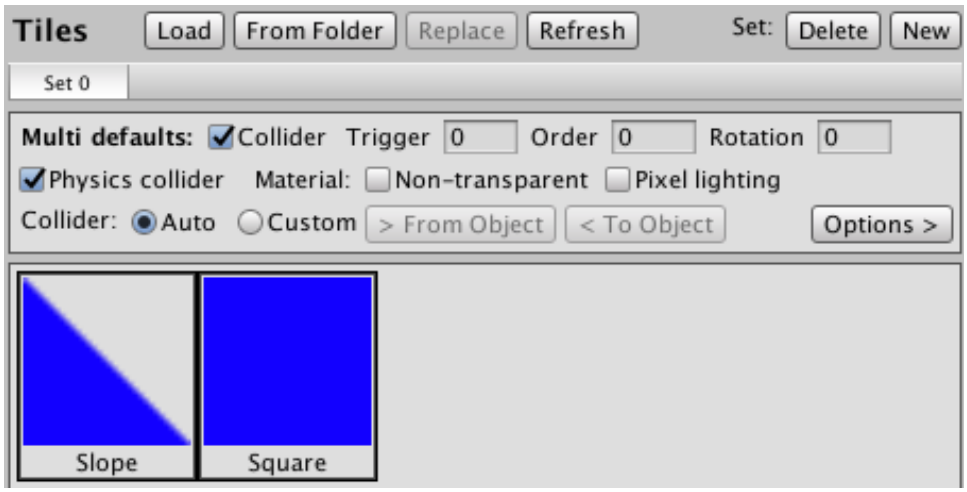
Since these have been saved as single sprites, we've added a packing tag called "mylevel". Set the compression type as desired; we're not going to use any compression for these. We've set the Pixels Per Unity value as 32, the same size as the tile. This makes the tiles 1 unit big, which is usually a good value to go for, though you can use whatever you prefer.

Now let's switch back to the TileEditor. You can load tiles individually with the **Load** button, or if you saved them into their own folder, use the **From Folder** button to automatically load all sprites in the selected folder as tiles.

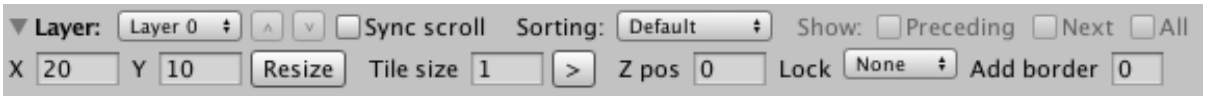


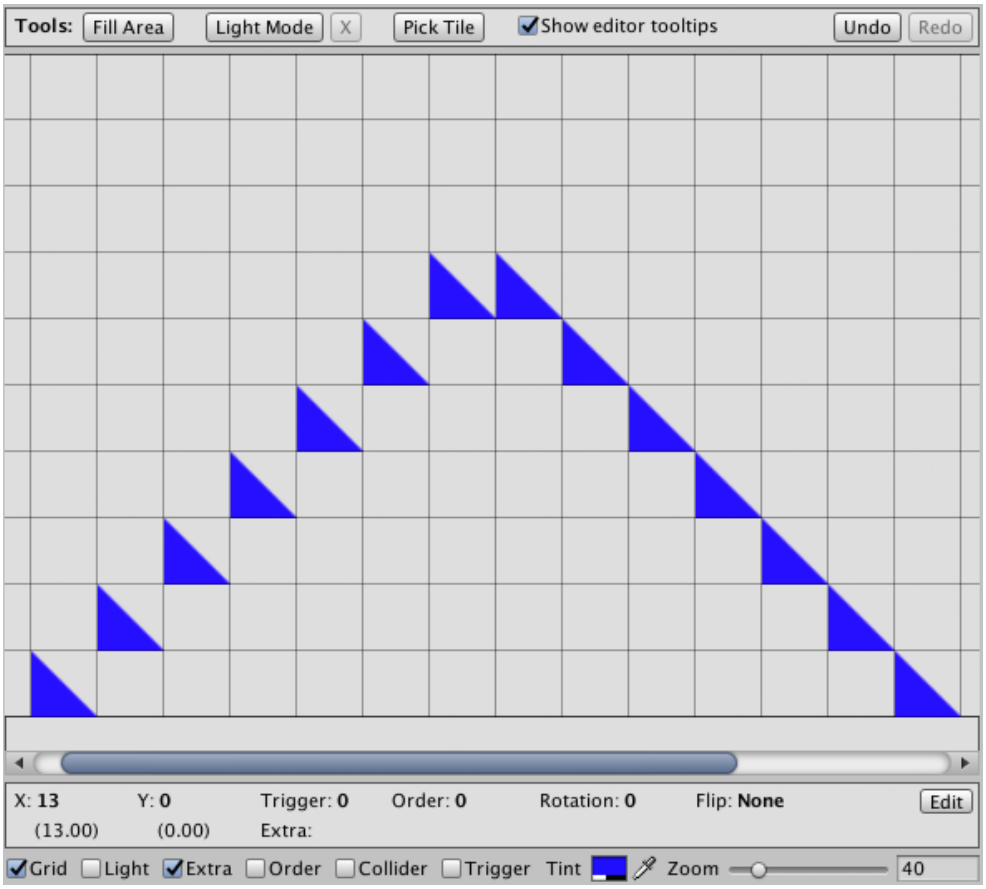
If you have tiles saved as an atlas, use the Load button to load the atlas, and all the tiles in it will be automatically loaded. (As long as you've set up the atlas properly in Unity by using the Multiple sprite type, and the appropriate slicing. See the Unity docs for more details about this if you're unsure.)

We're going to use these tiles as physics colliders, so we'll click on the **Physics collider** checkbox in the tile defaults, which also automatically selects the **Collider** checkbox. You can do this one tile at a time, or else select both tiles, in which case any default settings you pick will be applied to all selected tiles. Multi-selection can be done either by clicking on the first tile and shift-clicking on the next (this would also select all tiles in between if you had more than two tiles), or else by drawing a selection box around the tiles with the right mouse button. (Mac users can also use Command-Click for drawing a selection box, in case right-clicking isn't convenient.) We'll leave the collider generation set as Auto, so Unity will automatically create the appropriate collider shapes.



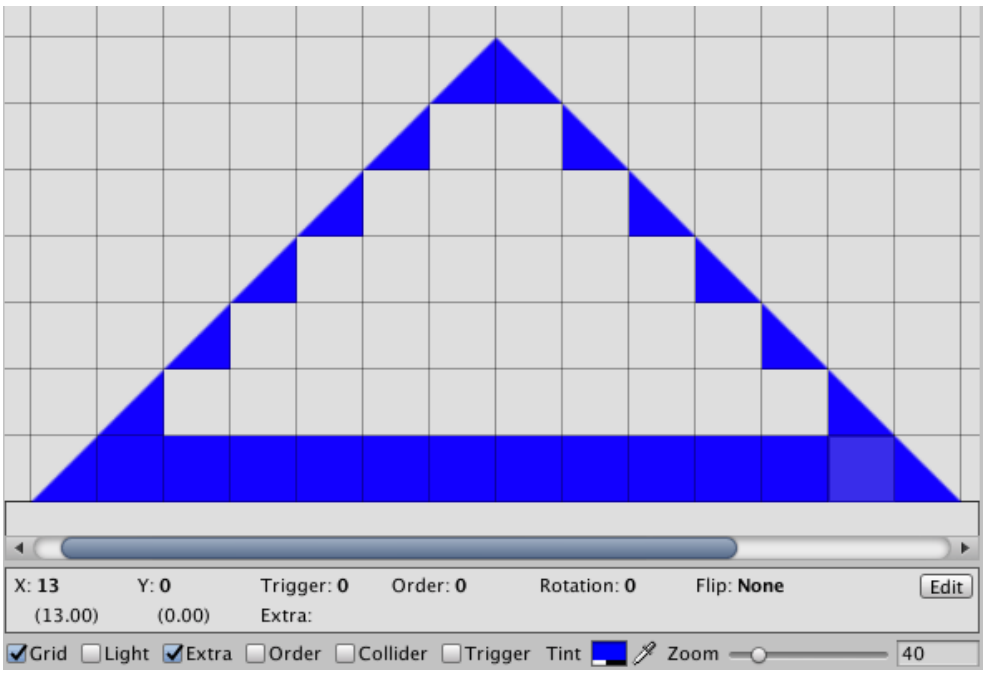
Let's start making a level. We'll change the layer size to 20 x 10, since this will be a small level. In the Layer section, change **X** to 20 and **Y** to 10, then click the **Resize** button. The tile size should be set to 1, since earlier we set the Pixels Per Unit for the sprites to 32, which makes them 1 unit big. Note that you can click the triangle next to the Layer, Light, and Selection controls to expand or collapse them as needed.



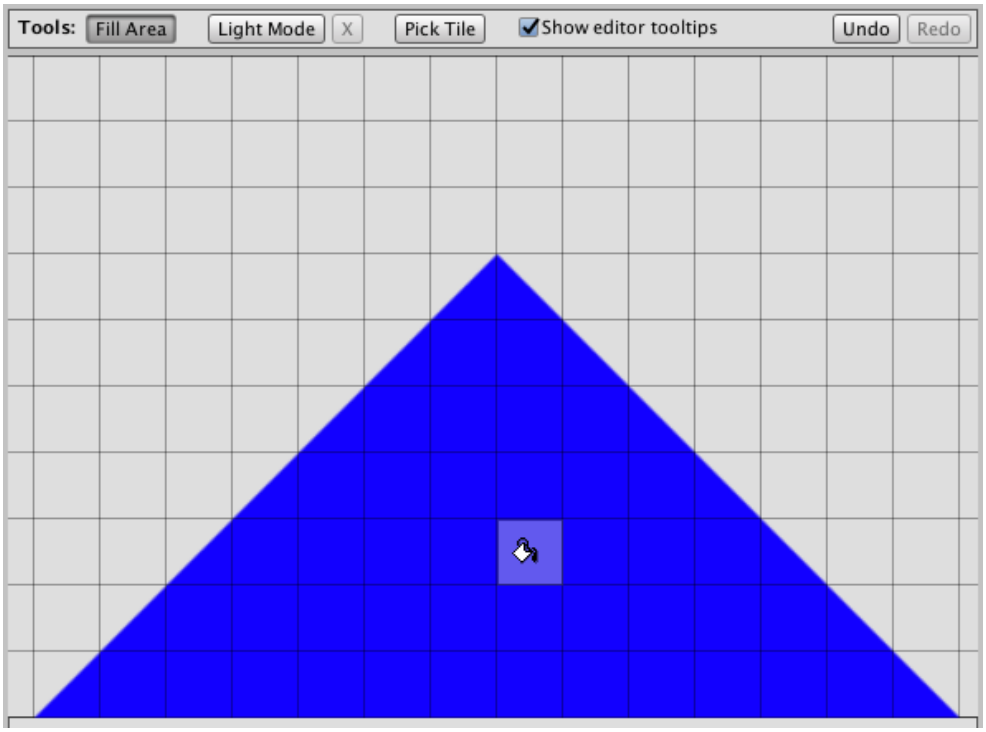


With the Slope tile selected in the Tiles section, draw tiles in the map by left-clicking in the map window.

We'll make the left slope look correct by flipping the tiles. (You could also make a separate tile for the left slope if you prefer.) You can alt-click a tile to set its properties, including tile flipping, but a faster way is to hover the mouse over the tile you want to flip and press the **X** key to flip it on the X axis. Naturally, pressing the Y key would flip it on the Y axis.



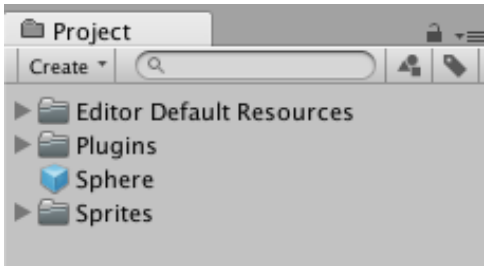
When that's done, select the Square tile and draw the tiles along the bottom. You could proceed to fill in the rest of the tiles this way, or you can use the Fill Area tool, as shown below.



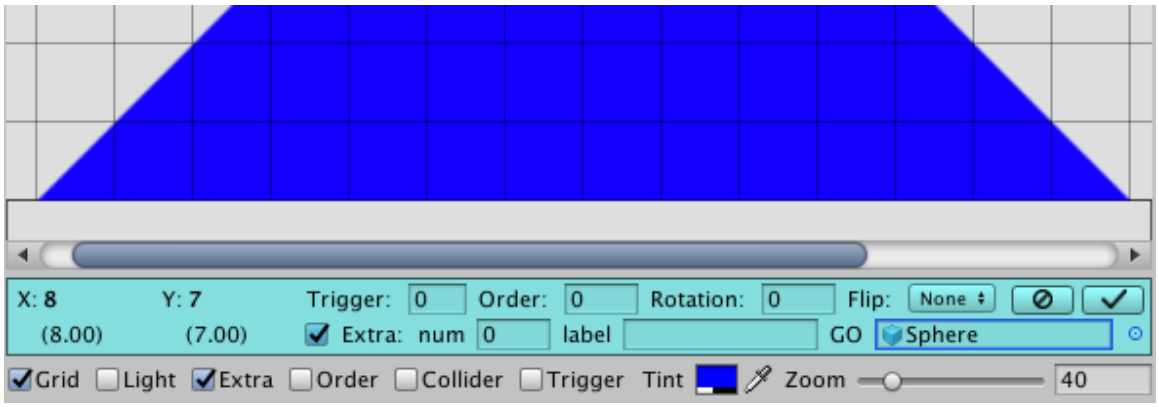
Click the **Fill Area** button in the **Tools** box. This turns on Fill Area mode. The cursor changes to a paint bucket, so as long as Fill Area mode is active, clicking in the map will fill contiguous areas with the currently-selected tile. So click in the empty area, and now our little hill is complete. Click the Fill Area button again to turn it off.

If you prefer the keyboard, **Shift-F** will also toggle this mode. Almost all controls have a keyboard shortcut. You can see them if you have the **Show editor tooltips** checkbox on and have the mouse hovering over a control. Also, check out the **TileEditor Keyboard Shortcuts** document for a list of them all.

Now let's have some fun and add GameObjects directly to the map. Going back to Unity, make a sphere by selecting the **GameObject** -> **3D Object** -> **Sphere** menu item. Delete the sphere collider component, then add Rigidbody 2D and Circle Collider 2D components from the Component -> Physics 2D menu. Drag the sphere from the hierarchy pane to the project pane to quickly make a prefab. Delete the sphere from the hierarchy.

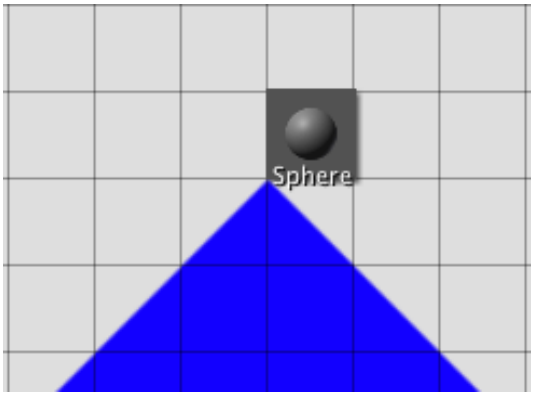


Switch back to the Tile Editor. We want to place a Sphere object at the peak of the hill, so alt-click on a tile near the peak to edit properties for that tile.



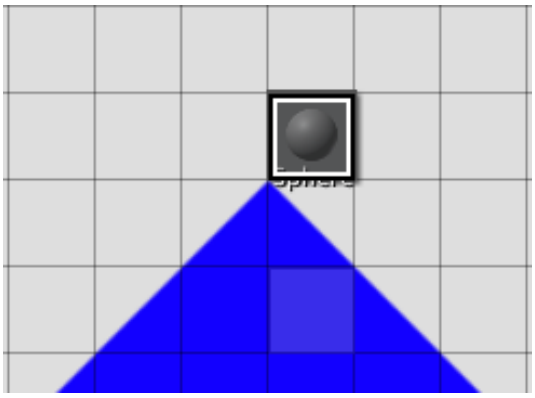
Click on the **Extra** checkbox in the colored box (below where it says Trigger) to set extra properties for the tile. In this case we want a GameObject, so click on the little target icon next to the GO box, and select the Sphere prefab from your project. Click the checkmark button to accept the change.

There are a number of different overlays you can use to see different tile properties. In this case we want to see extra properties for tiles, so click on the **Extra** checkbox at the bottom of the level section, next to the Grid and Light overlay checkboxes.

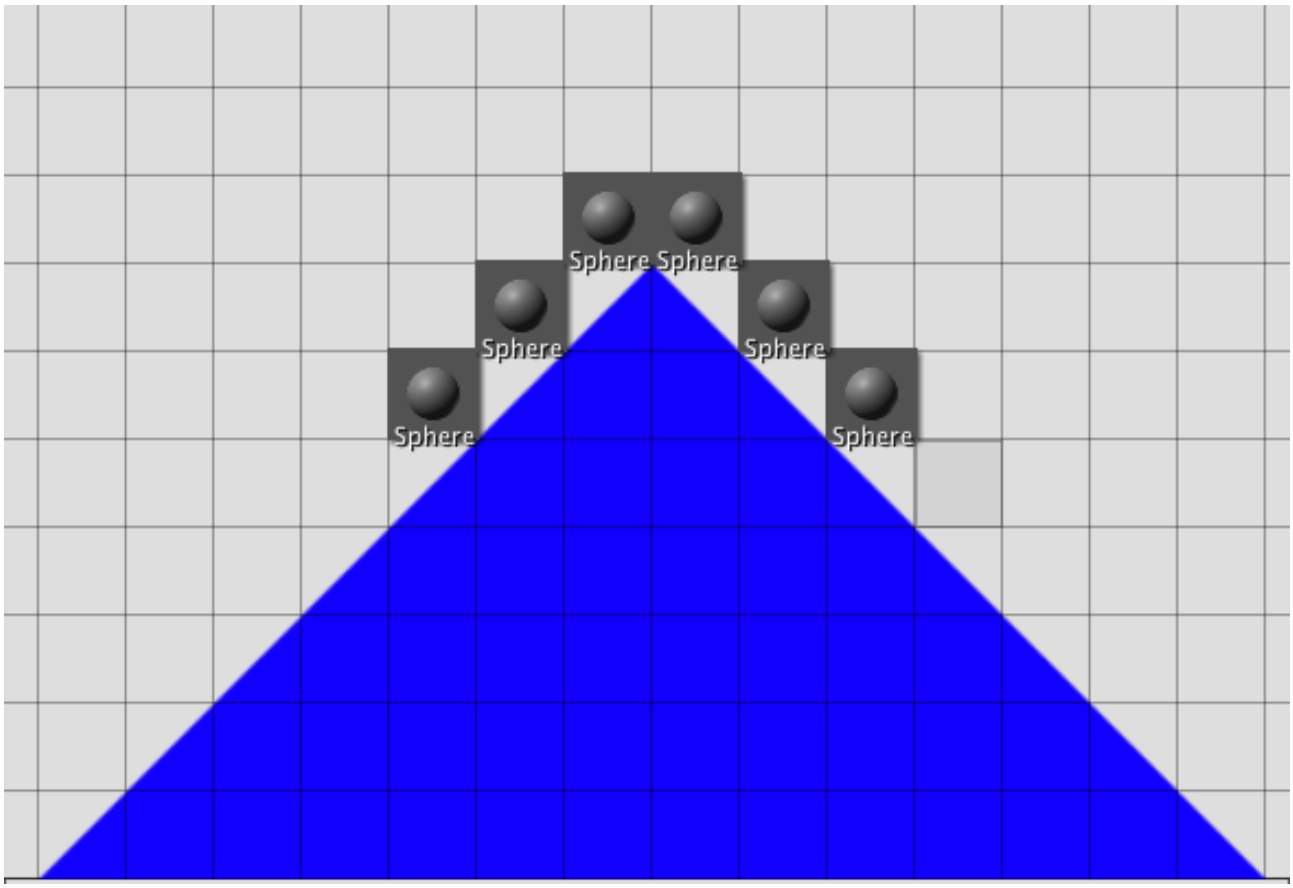


There, now we can see the Sphere prefab in the appropriate tile. Let's add some more—you can repeat the process to edit the GameObject property for more tiles, but it would be faster to copy and paste.

To do this, draw a selection box around the Sphere tile by right-clicking and dragging. You can now use the **Copy** button in the **Selection** section, or use the **Alt-C** keyboard shortcut.



Now you have the selected tile in the copy buffer, so click on each tile where you want to paste the Sphere tile. When done, de-select the copy buffer by clicking the **X** button in the Selection section, or else press the **Esc** key.



Hey, our level is done! Let's save it by clicking the **Save As** button, and save it somewhere in the project. We'll just call it MyLevel.

So, now we need to load the level. For this, make a script file, either Unityscript (Javascript) or C#. Name it LoadLevel. The script should look like this:

```
// Unityscript
#pragma strict
import SpriteTile;

var level : TextAsset;

function Start () {
     Tile.SetCamera();
     Tile.LoadLevel (level);
}
```

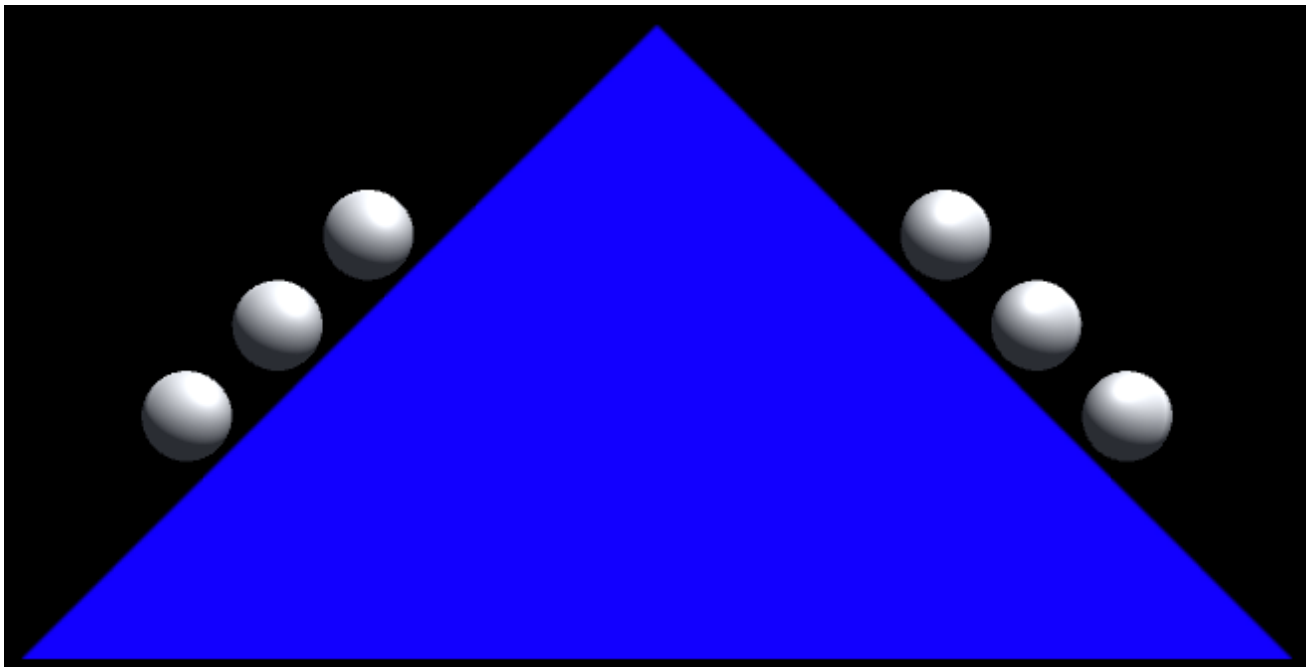
```
// C#
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using SpriteTile;

public class LoadLevel : MonoBehaviour {

     public TextAsset level;

     void Start () {
          Tile.SetCamera();
          Tile.LoadLevel (level);
     }
}
```

Make a new scene, and attach the script to the camera. Add a directional light, and set the camera background color to black. Position the camera somewhere around 7 units on the X axis and 4 units on the Y axis. Click on the camera, and drag the MyLevel file from your project onto the Level slot on the script attached to the camera. Now press Play in Unity, and watch the show—the sphere objects are instantiated at the appropriate tile positions when the level is loaded, and they'll roll down the hill, since the tiles were set to have physics colliders.



Of course, this is just the beginning. There's lots of stuff you can do with SpriteTile, so read the **SpriteTile Documentation** file for complete details. When coding, refer to the **SpriteTile Reference Guide** for a list of all functions and how to use them. It's also a good idea to import the **SpriteTileDemos** package, and play around with the different demo scenes. It's best to start a new project before importing the demos, so you don't overwrite your existing tile setup with the demo setup.