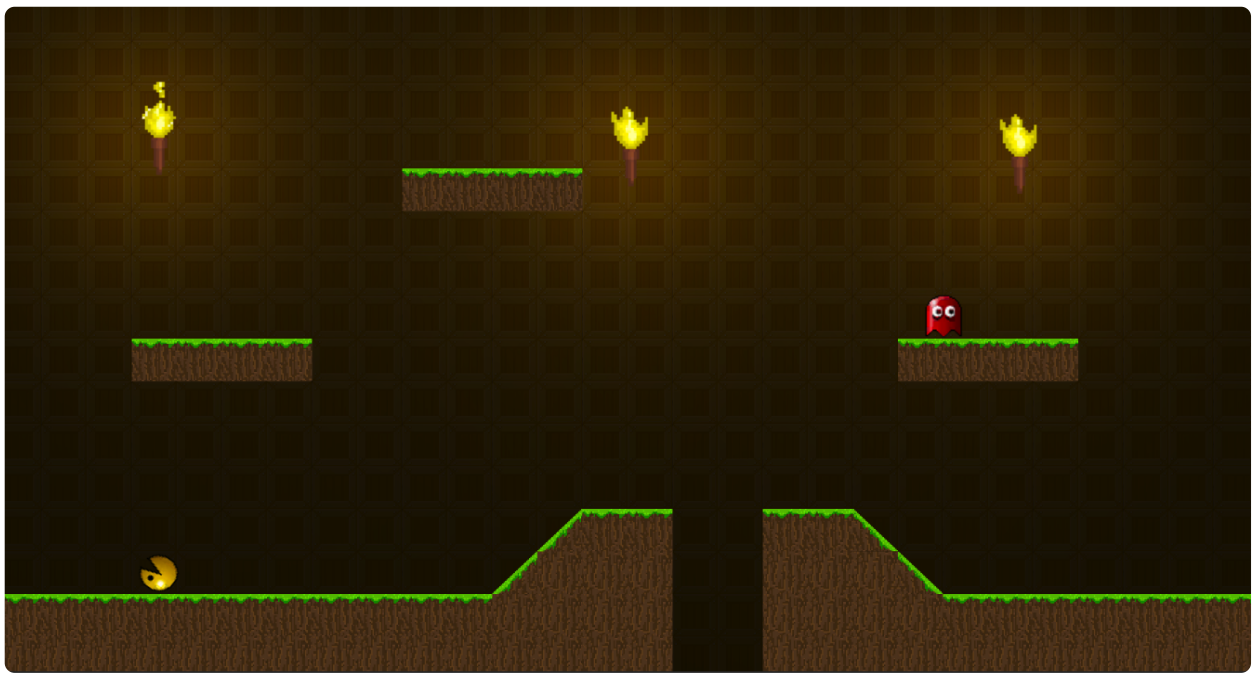
Lab - Level Design



Goals

- To learn how use Tilemaps in the Unity Editor.

Resources

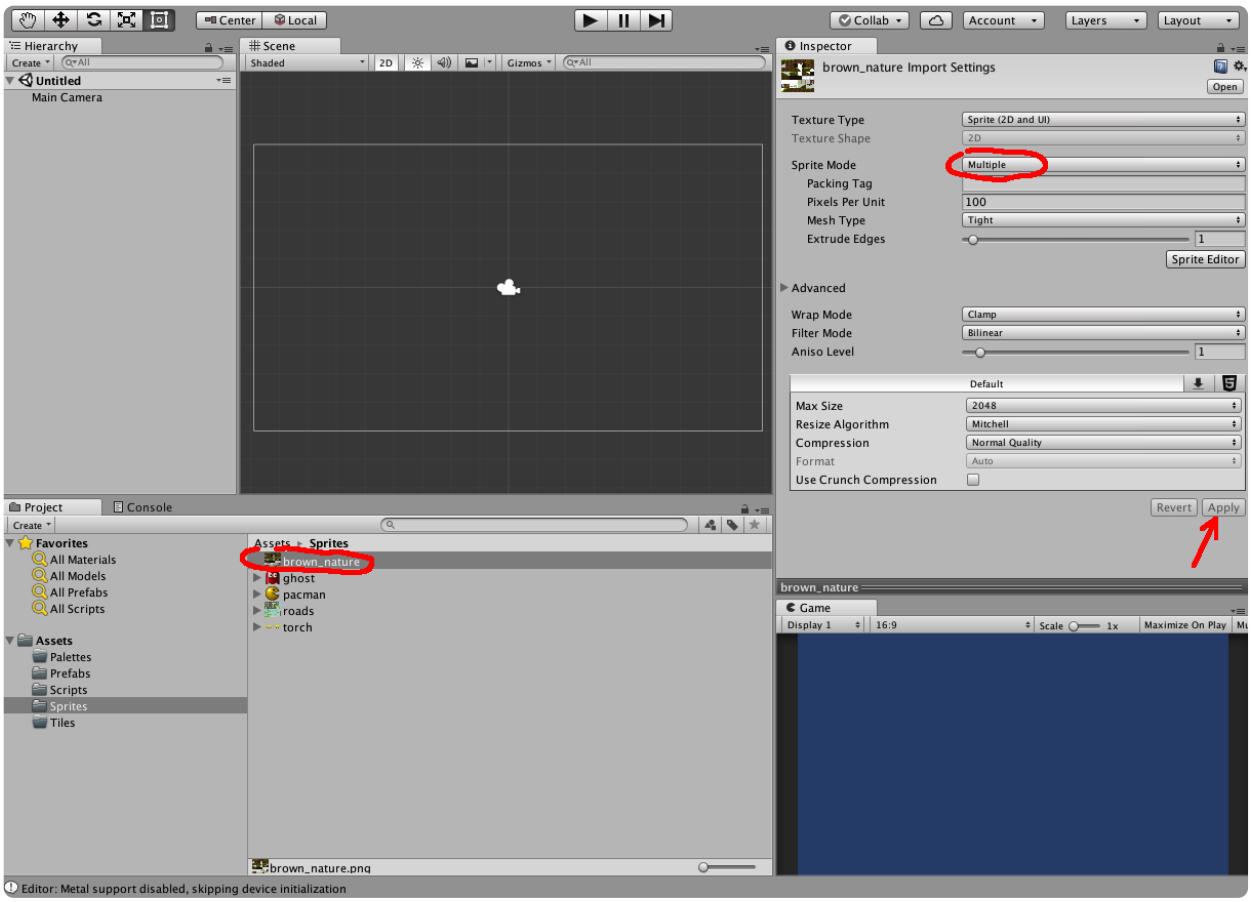
- Tilemap (Unity Manual) (https://docs.unity3d.com/Manual/class-Tilemap.html)
- Collection of tilesets (http://opengameart.org/content/best-orthogonal-rectangular-tilesets-for-tilemaps)

Basic Challenge

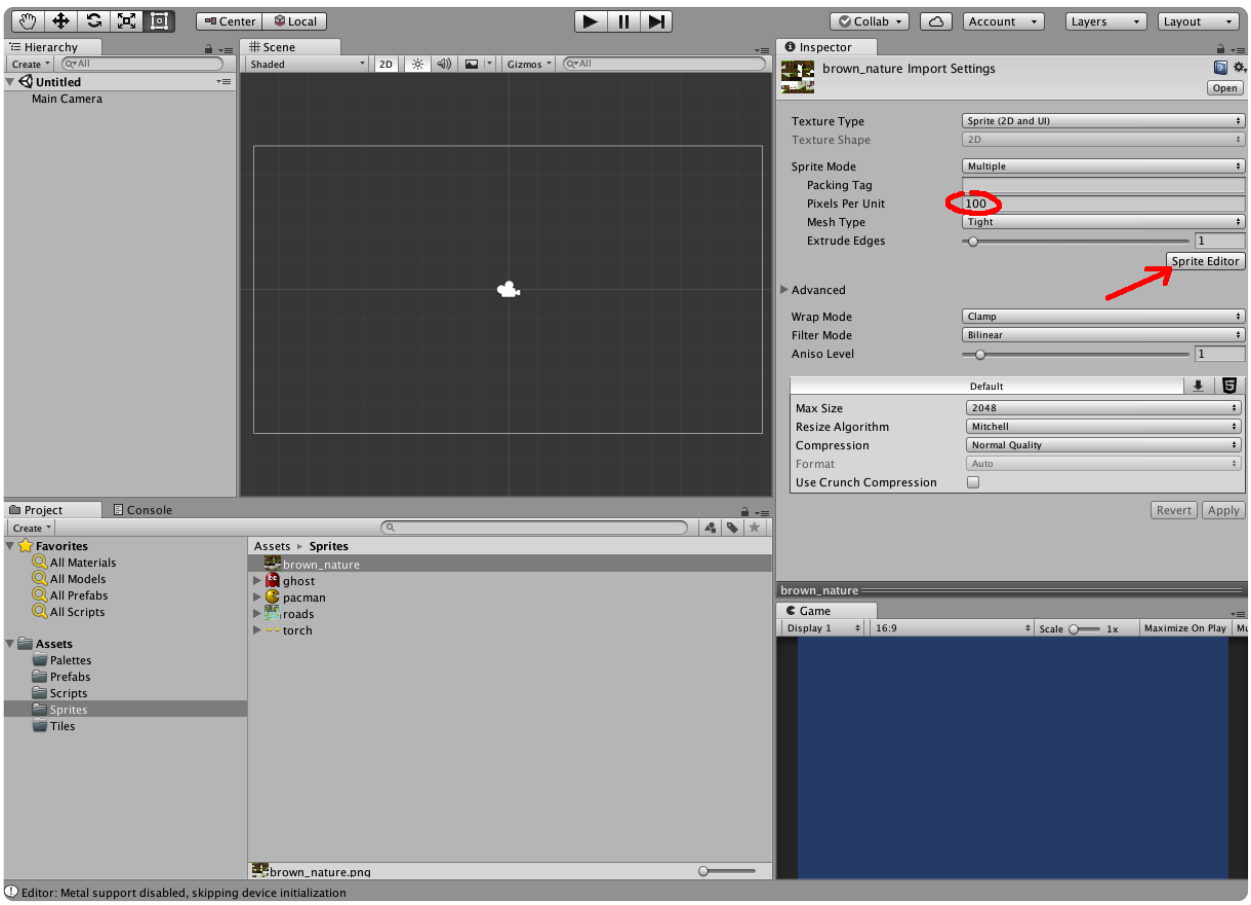
The basic challenge of this lab is not directly about level design, but rather about the tools for building levels in the Unity Editor. A good designer should always keep in mind the constraints posed by the used technology, therefore it's a good idea for a designer to get familiar with the tools. Unity version 2017 expanded it's 2D engine capabilities by adding tile mapping functionality to its set of tools. It works with tile sets, which are just images that contain a set of rectangular *tiles* that you can use to make up your game levels. Using tile maps you can drag and drop tiles to create a level. It's a great way to build levels.

Creating a palette

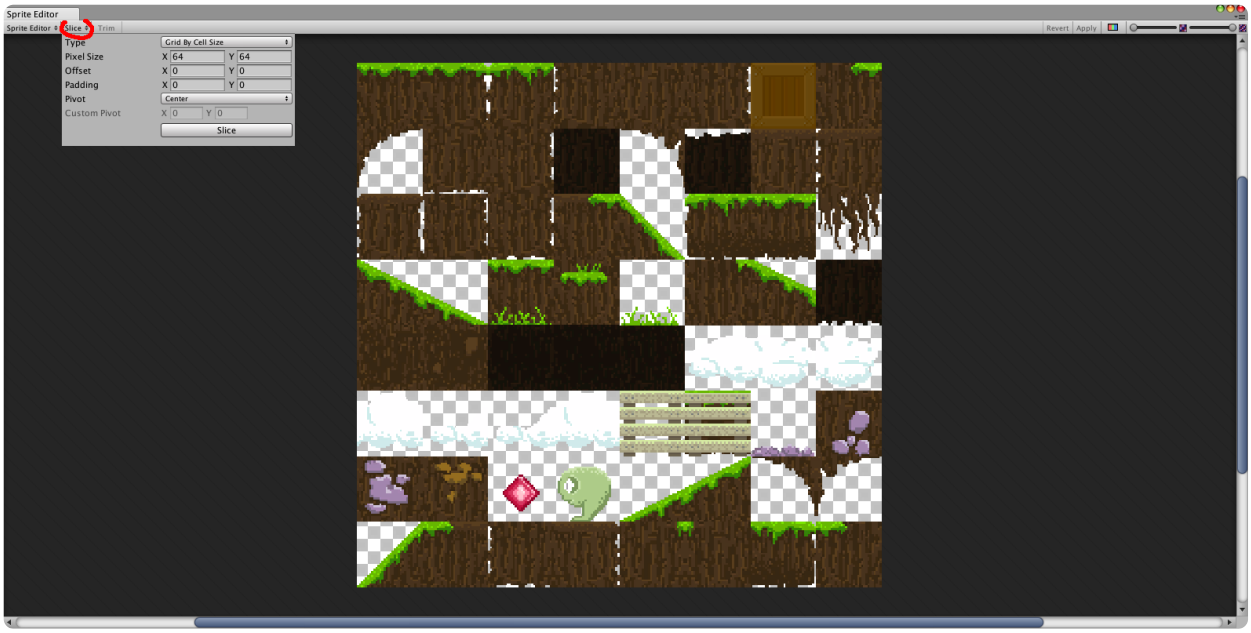
1. Fork the Tilemaps project (https://altitude.otago.ac.nz/cosc360/Tilemaps) on GitLab, then clone your fork to your hard-drive.
2. Open the *Tilemaps* project, the directory of the cloned repository, in Unity.
3. In the *Assets/Sprites* you'll find "brown_nature.png" spritesheet - it contains images of various tiles, some of which you'll be using to build a level with. You need to tell Unity that this image contains multiple sprites. Select the image in the **Project panel** and change the *Sprite Mode* in the *Inspector panel* to "Multiple". Press the *Apply* button.



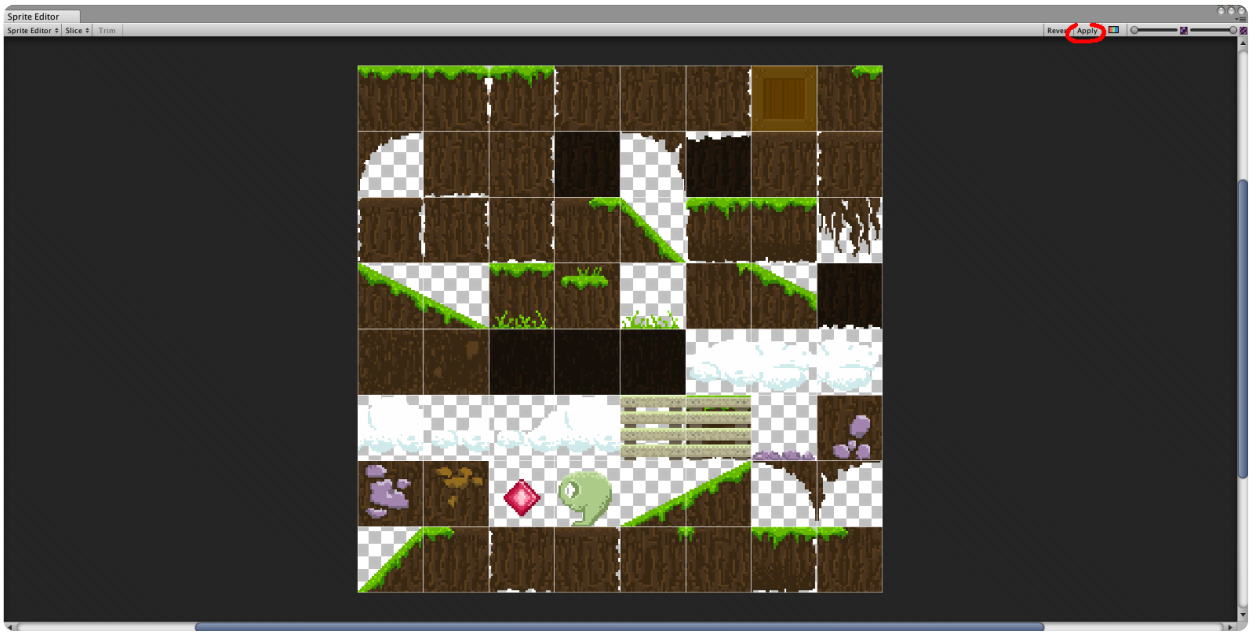
. Make sure that the "Pixels Per Unit" option is set to 100. Press the *Sprite Editor* button to open the *Sprite Editor*.



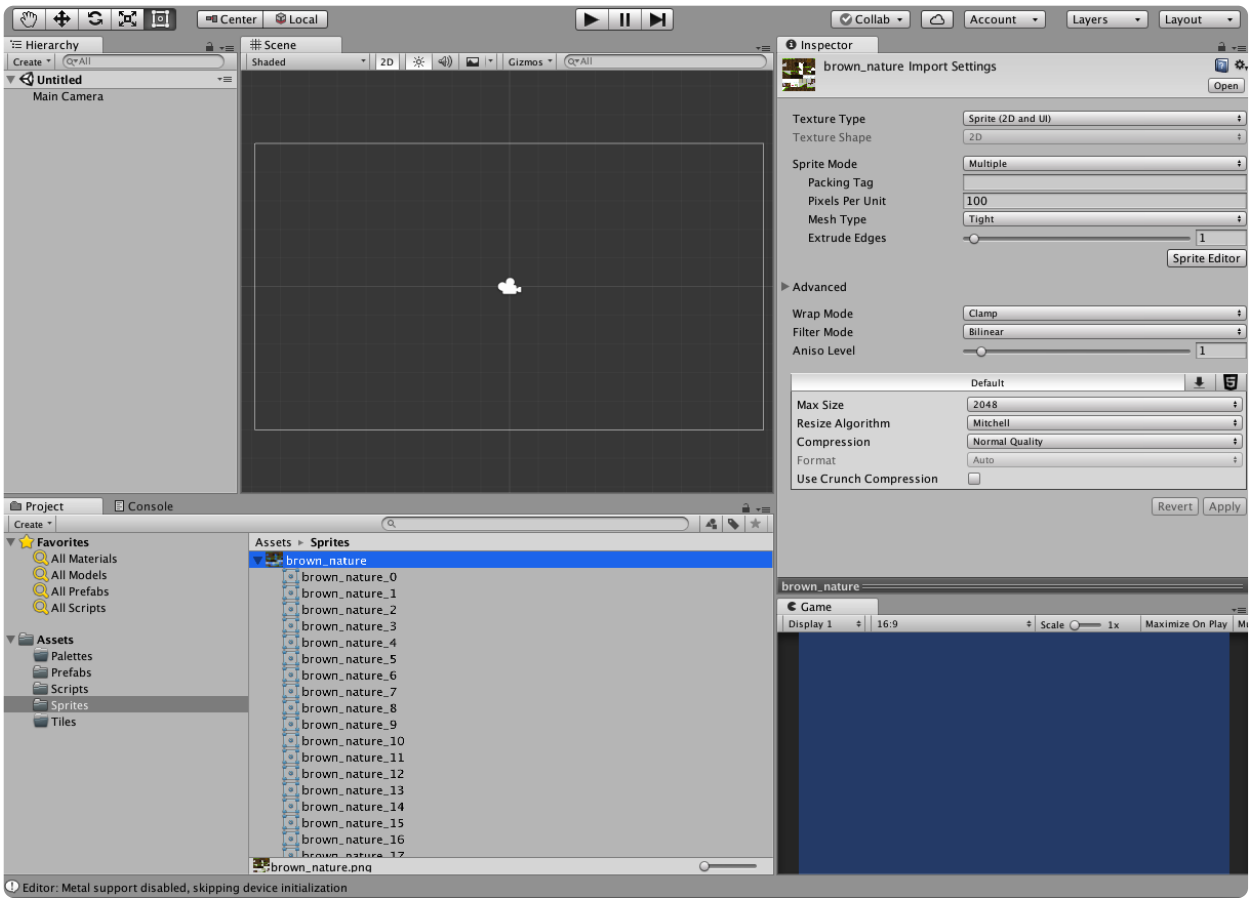
. The *Sprite Editor* allows you to instruct Unity how to extract individual sprites from a spritesheet. This particular sprite sheet is a grid of 64x64 pixel sprites. Press on the *Slice* button and select "Grid By Cell Size" for the *Type*. Make sure that *Pixel Size* is set to 64 by 64 and other setting are 0. Click on the *Slice* button at the bottom-right corner of the pop-up window.



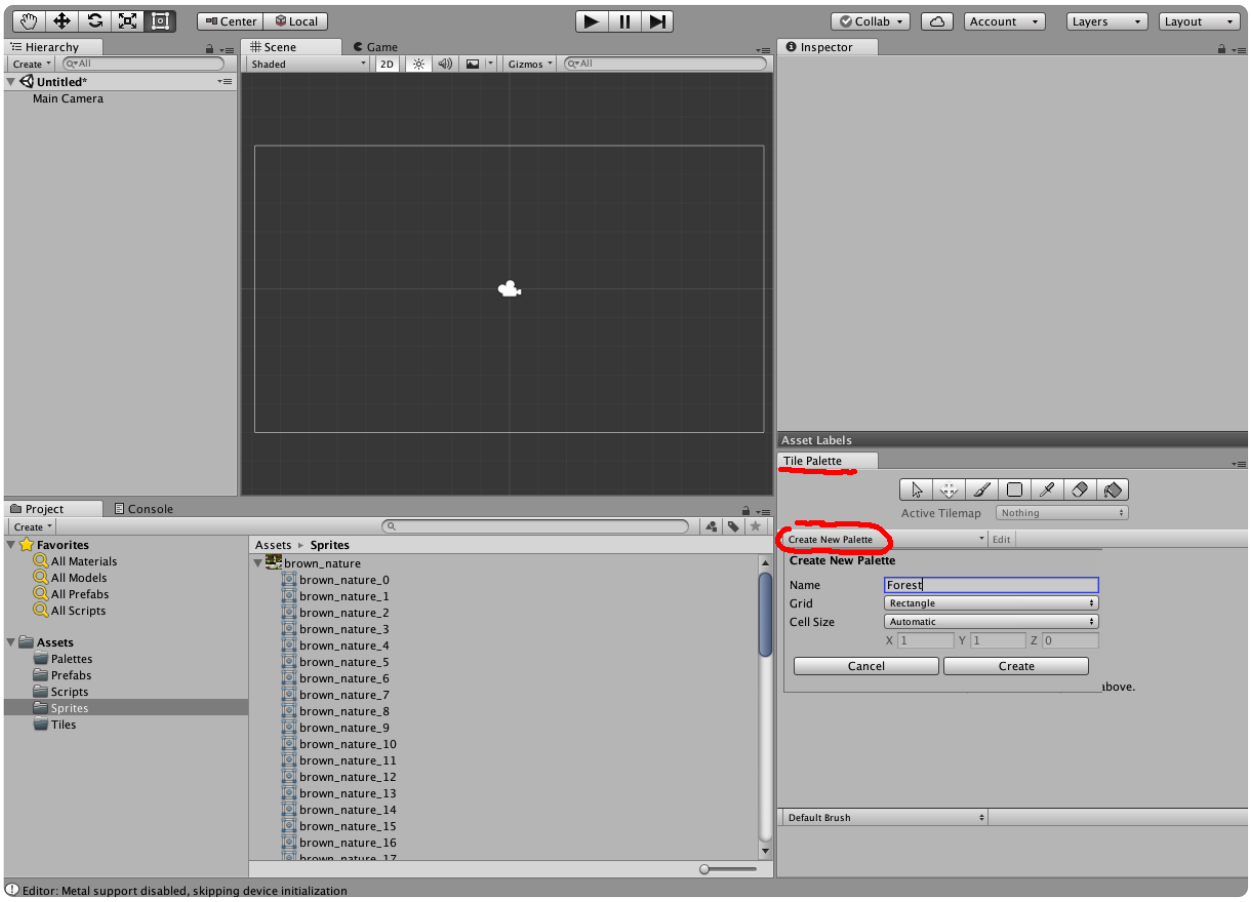
. Now the *Sprite Editor* should show you the grid over the spritesheet. Press the *Apply* button to slice the spritesheet.



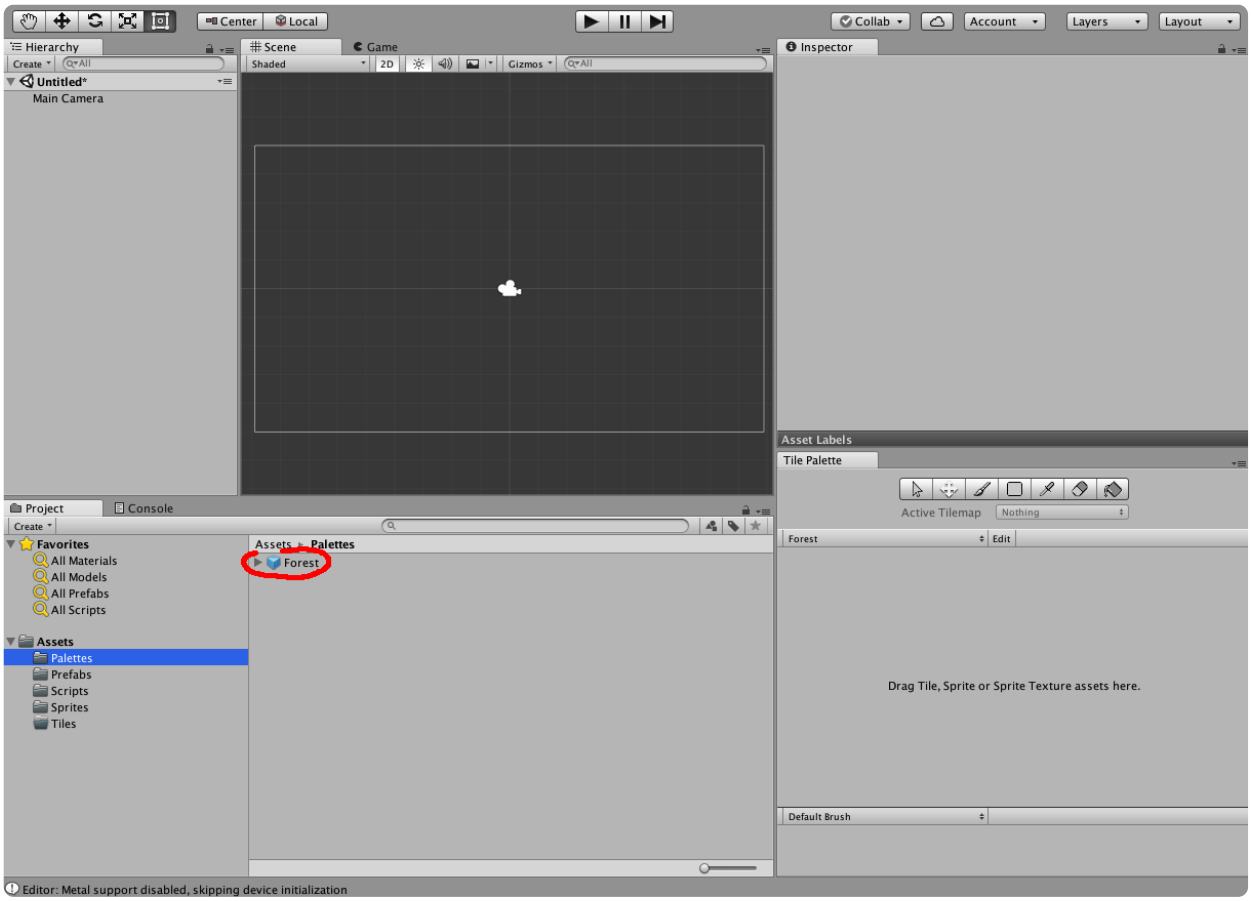
. You can close the *Sprite Editor* now. Take a look at "brown_nature.png" in *Assets/Sprites* - you can expand it now to see individual sprites index from 0 to 63.



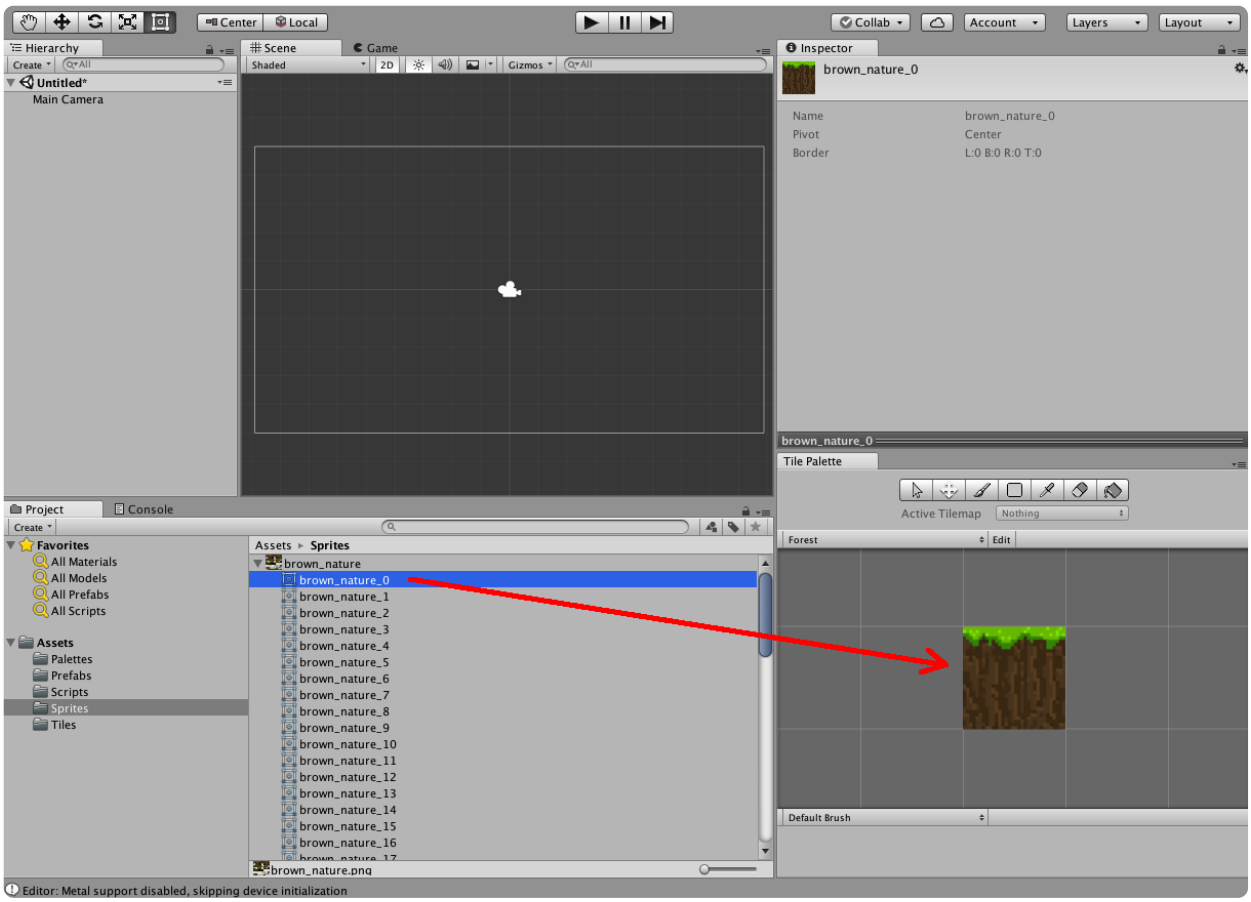
. From the main menu select *Window -> 2D -> Tile Palette*. A new panel will appear called **Tile Palette**. You can doc it anywhere you want in your Unity Editor - I chose to doc it underneath the **Inspector panel**, so that I can see both the *Inspector* and the *Palette*.



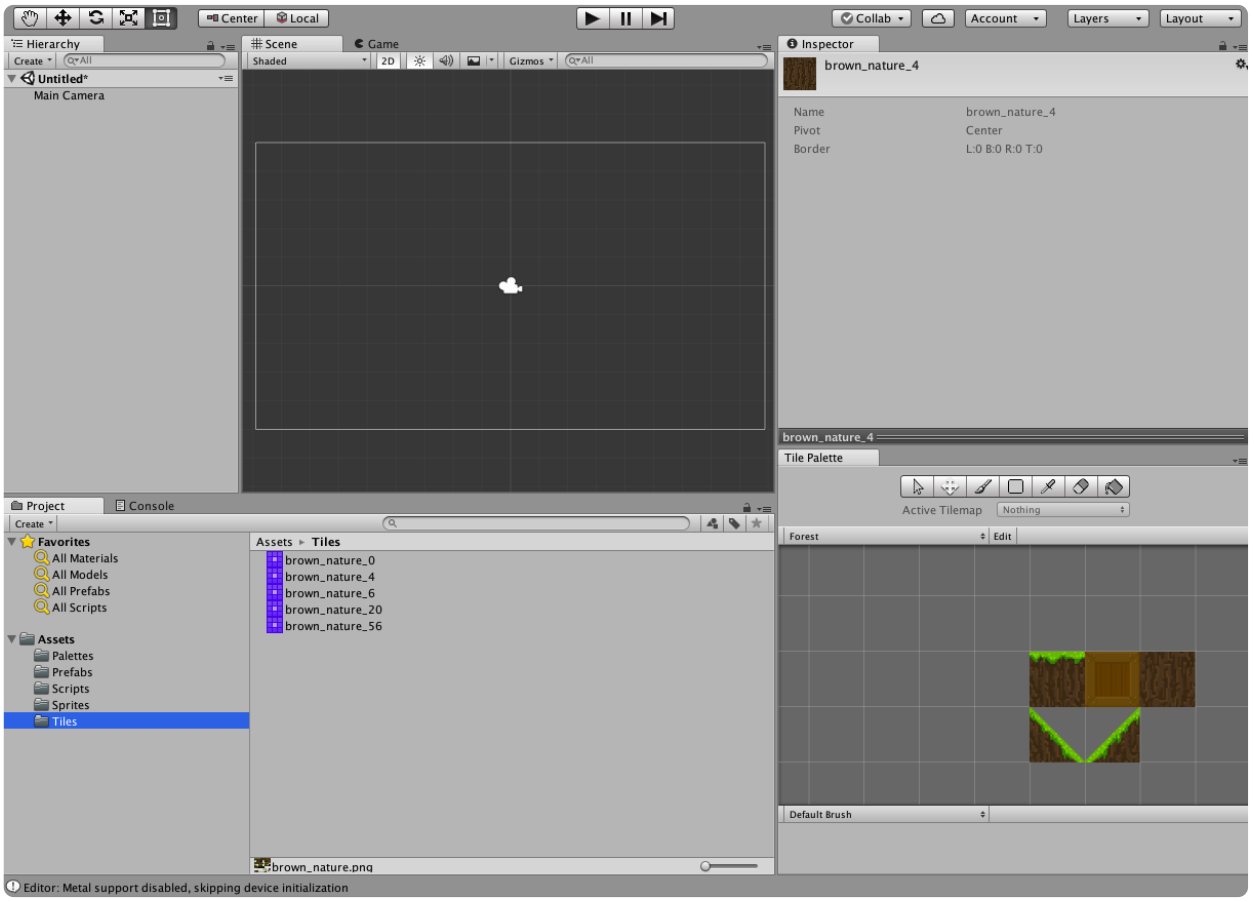
. In the **Tile Palette panel** click the *Create New Palette* button. Name the new palette "Forest" and save it to *Assets/Palettes*. Once you're done you should see the "Forest" asset in *Assets/Palettes*.



). Next, you'll need to add some tiles to your palette. Drag "brown_nature_0" sprite from *Assets/Sprites* in the **Project panel** to an empty tile in the **Tile Palette panel**. You'll be prompted to save the tile. Save it to *Assets/Tiles* - the default name, which is the name of the sprite, is fine.



1. Repeat the previous step of dragging the sprite to the palette and saving it as a tile for sprites index 4, 6, 20 and 56 from the "brown_nature" spritesheet. I am assuming your indexing will be the same as mine (if not, just take a look at the tiles shown below and select the same ones).

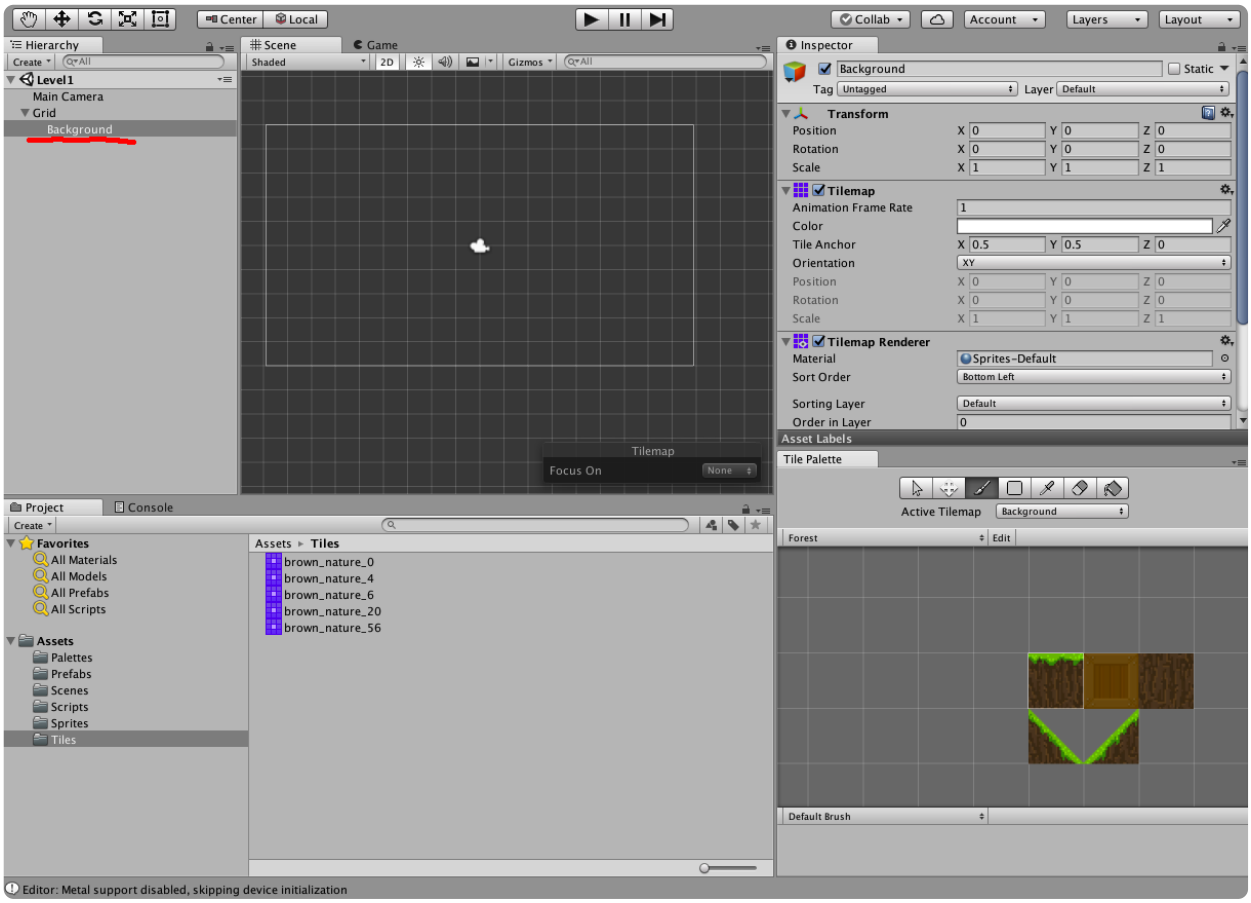


2. You have created a tile palette. Next you'll be using it to build tilemaps.

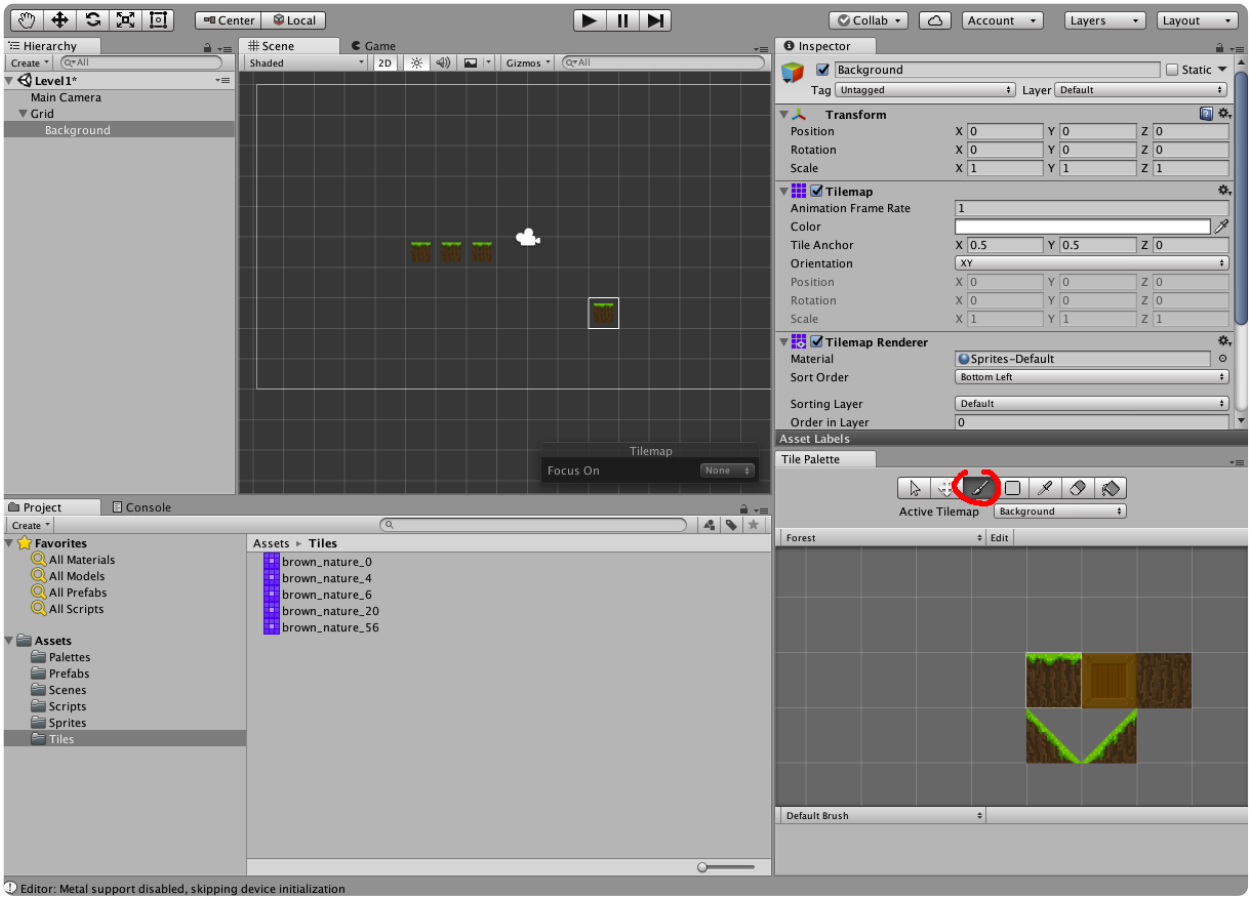
Creating a tilemap

3. Let's start by saving the scene, name it "Level1".

4. To create a new tilemap right-click anywhere in the empty space of the **Hierarchy panel** and select *2D Object/Tilemap* option. A "Tilemap" game object will appear with a "Grid" parent object. Rename the child to "Background". Notice the grid overlaying the entire scene.



5. From the toolbar near the top of the **Tile Palette panel** select the "Paint with active brush" tool. Then select any tile from the palette and start clicking on the grid squares in the **Scene** to lay down tiles.



3. Are your tiles smaller than the squares in the Tilemap grid? If you select the "Grid" object in the **Hierarchy panel** you should see the "Cell Size" in its "Grid" component in the **Inspector panel** is set to (1,1,0). These number are in scene units. Recall the "Pixels Per Unit" setting for the "brown_nature" sprite sheet - it was set to 100. That means, one scene unit is equivalent to 100 pixels...and since the sprite sheet contains tiles that are 64x64 pixels, they don't span an entire 1x1 unit square. Change the "Cell Size" of the "Grid" to (0.64,0.64,0). Now the tiles should fill in the entire square of the grid.

7. From the toolbar near the top of the **Tile Palette panel** select the "Paint a fill box with active brush" tool. Then select the box-like looking tile from the palette. In the *Scene* click on the left-top corner of the screen and drag the mouse over to the right-bottom corner. You will trace out a square, which will be filled with the box-like tiles. Take a note of other tools in the **Tile Palette panel**. You can delete, fill an area, etc...essentially paint with tiles. The tilemap we create so far is pretty boring, but hopefully you see how painting with different tiles has the potential to make it interesting with relative ease.

3. Save the scene and commit the project to the git repository. Don't forget to add the palette as well as tile files.

Colliding tilemap

1. The tilemap created in the previous section is meant to serve as the background. In this section you'll be creating a map that interacts with other game object in the scene.

10. In the **Hierarchy panel** right-click on the "Grid" object and select *2D Object/Tilemap* to create a second tilemap object. Name it "Platforms".

11. The tiles from the "Platforms" tilemap are meant to be rendered in front of the tiles from the "Background" tilemap. You can achieve this by changing the "Order in Layer" attribute of the "Platforms" object to 1, while making sure that "Background"'s "Order in Layer" is set to 0. The engine will render game objects in the order from the lowest to the highest "Order in Layer". Hence, objects with higher "Order in Layer" appear in front of objects with the lower "Order in Layer".

12. To instruct the **Tile Palette** that you want to add tiles now to the new tilemap, change its "Active Tilemap" setting to "Platforms". Then, with the tiles you currently have in your palette, recreate the platforms as shown in the image below.

13. Time to add another game object to the scene. Find the "PlatformPacman" prefab in **Assets | Prefabs** and drag it over to the scene somewhere on the map (just above the ground). It's a prefab that consists of a pacman sprite, with rigidbody subject to gravity, a circle collider 2D and a script component that puts the game object under player's control. Set it's Scale attribute to (2,2,1).

14. If the pacman sprite is not rendered in front of the map, you need to set its "Order in Layer" (in the *Sprite Renderer* component) to 2.

15. If you play the game now, pacman will just fall through the map, because you haven't set up any colliders for the platforms yet.

16. Select the "Platforms" game object in the **Hierarchy panel** and add *Tilemap Collider 2D* component to it. Every tile in this tilemap will get a collider around it.

17. If you play the game now, pacman should collide with the platform and remain on it. You can move the pacman and jump up using arrow keys. Note that if you jump up towards a platform you can't pass through. That's expected - after all they have a collider around. However, a common mechanic for many platformers is the ability of the character to jump through the platform on the way up and collide with it on the way down. It probably wouldn't be all that difficult to implement this functionality in a script...but Unity give you a component that delivers this functionality.

18. Add the *Platform Effector 2D* component to the "Platforms" tilemap. Make sure its "Use One Way" option is selected as well as the "Used By Effector" option in the *Tilemap Collider 2D* component.

19. Now if you jump up, pacman should go the platform on the way up and stay on it.

20. Time to add another game object to the scene. Drag the "PlatformGhost" prefab from *Assets | Prefab* onto a platform (or just above it). This prefab creates a game object with a rigid-body and a 2D collider. If you can't see the ghost sprite, set its "Order in Layer" attribute (in the *Sprite Renderer* component) to 2.

1. The ghost prefab contains a script component that will move the sprite left and right along a platform without falling off as long as the game object associated with the collider that the ghost is colliding with (or its parent) is tagged with the string "Platform". Hence, in order to get the ghost to move you need to tag the "Platforms" game object with a "Platform" tag.

2. Play the game, the ghost should move back and forth on its platform.

3. Save the scene and commit the project to git repository.

Adding lighting

4. Well, this is about it for the basic introduction to tilemaps. In this section you'll be putting some extra finishing touches, just to see how a bit of lighting can change the look and feel of the level.

5. Select the "Torch" prefab from *Assets/Prefabs* and drag it into the scene. This is a game object with a script that animates over 9 sprites to give an appearance of a burning torch. Set it's "Order in Layer" to 1 - rendered in front of the "Background" but behind the pacman and ghost game object. Play the scene to see the torch animation.

   That looks not too bad, but it would be nicer if the torch was actually lighting things.

6. Right-click on the "Torch" game object in the **Hierarchy panel** and select *Light/Point Light* option. This will add a "Point Light" game object as a child of the "Torch object".

7. Unselect the *2D* option from the "Scene" tab to see the scene layout in 3D. By default the light is superimposed over the torch. you probably want it a bit in front, so that it shines towards the wall. If you set "Point light"'s Z position to -2, that should put it a bit in front of the tilemap.

8. Still can't see any difference in the lighting. That's because "Sprites-Default", which is the default material used for all sprites (including the tiles) maintains a constant level of maximum illumination. You need to change the material of the "Background" tilemap to something that reacts to light. Select the "Background" object and change the "Material" property of its "Tilemap Renderer" component to "Default-Material" (which should be one of the options if you click the circle button on the right hand-side of the property). Now the tiles in the tilemap should get a bit darker (since they are not automatically illuminated to maximum) and will react to the illumination of the point light in front of the torches.

9. Select the "Torch" game object. Note its "Prefab" attribute in the **Inspector panel**, where you find the "Override" option. Select it and click on the "Apply to All" button to apply all the changes you made to the prefab. The prefab should now also contain a point light object. Drag two more "Torch" prefabs into the scene and position them wherever you like.

9. Save the scene and commit the project to git repository.

Assessment

Show your work to the demonstrator for assessment.

Intermediate Challenge

There are many components of a complete and polished game, and it might not be obvious what those things are until one has a chance to take a look at the final product. Hence, in this challenge you are going to take a closer look at one of the games from the previous year and analyse it. It's a game developed under the same conditions (in terms of resources and timeline constraints) as the one you are developing now.

Play and evaluate

Choose one of the games developed by your peers in the previous year - the games can be found on the course website (http://www.cs.otago.ac.nz/cosc360/game-projects.php). Play through and make notes on the design. In your notes, consider the following:

- What do you like?
- What didn't work so well?
- What was the experience provided by the game?
- Think about the game in terms of the Elemental Tetrad:
    - Mechanics
    - Story
    - Aesthetics
    - Technology
- What aspects of the game are novel? What aspects are more standard?
- How would you describe the game in a single sentence?
- How does the game guide you, and teach you its rules / mechanics?
- These games have all been built by 3-4 people in 5 weeks. Getting a complete game to a finished state in this time requires some compromises. Can you see where they deliberately limited the scope of the game to keep it manageable?
- Also, feel free to flick through the lab copy of The Art of Game Design (or the accompanying card deck), and pick a lens at random. Think about the game from the point of view of the lens.

Assessment

Show your detailed analysis (it must be written down) to the demonstrator in the lab. Be prepared to explain and argue your points if a discussion ensues.

Master Challenge

- Design few levels for a physics based or a racing game. Don't worry about the technology and complexity - design without constraints. Some aspect of the game should be novel - unusual mechanics, AI, or something like that. Write a mini-game design document focusing on level design - definitely need drawings and explanations here. In essence it's a detailed proposal for a game...which you won't have to implement.

  or

- Prepare a case study that analyses the design of an existing game of your choice. This should involve some research on the making of the game and a 20min presentation of the case study during class time. Arrange with Lech potential time slot for the presentation.

  or

- Devise and complete your own Level Design Challenge - just check with the lecturer or the demonstrator first, whether the scope of the work will be sufficient for the awarded skill points.

Assessment

Show either your design, or case study to the demonstrator for assessment.