Guide to an in-depth understanding of the Tile Builder Package.

Please read this part

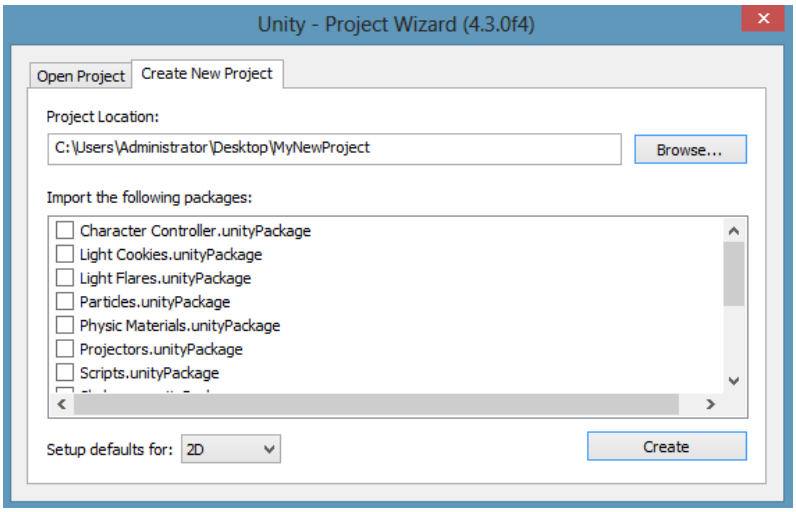
First, I would like to thank you for purchasing the Tile Builder Package. In this guide I will be documenting a clear vision of how to use this level editor tool. If something remains unclear or not well enough explained, please do contact me at wouterversloot@gmail.com as I will be glad enough to explain it to you.

I will be explaining everything step-by-step with images so that every user, pro or beginner, can keep up in a solid way. The most important notes will be repeated at the bottom of this guide. I hope you enjoy using this package as I enjoyed creating it.

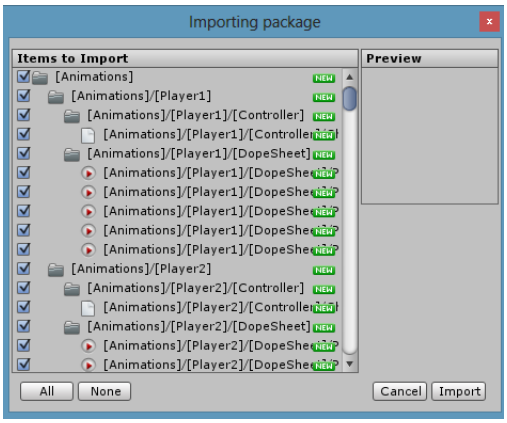
Please don't forget to leave a review on my asset store page!

So we downloaded the Package, what's next?

Create a new project, move it where ever you want and put its Setup defaults to: 2D. You don't have to import anything just yet! Hit the create button and wait for Unity to start.



Now once we are in Unity, go to the Asset Store window and Import the package. Make sure to hit the all button to make sure you import everything.



Upon importing, you should see that your Project window (Ctrl + 5) has changed. It has added some default, and important, new folders to locate its assets. At the time of writing, this will probably look a little different than what it will look like in the final product, because I haven't released this asset just yet.

Setting up the default play area

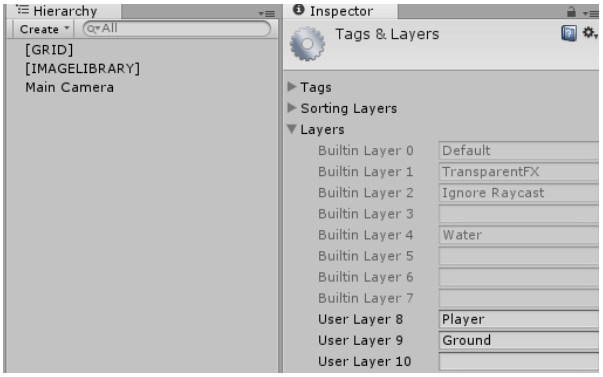
If you wish to jump right into action, there is a MainScene ready to be explored. If you want to take the full tour, this is where we move on.

Save your new scene, and add two empty GameObjects (**Ctrl + Shift + N**) into the scene. Name them whatever you like, but to keep things clear, I will name mine: "[GRID]" and "[IMAGELIBRARY]". The reason why I use brackets is because the hierarchy folder sorts assets by names. The ' [' character is one of the first characters to be sorted on top. This means that the Grid and ImageLibrary will always be visible at the top of your hierarchy folder. (Also position them at **0,0,0**).

Let's add some basic scripts to our scene.
Onto your **Main Camera**, drag the script: "LerpToPlayer.cs"
Onto your **[GRID]**, drag the script: "Grid.cs"
Onto your **[IMAGELIBRARY]**, drag the script: "ImageLibrary.cs"

Let's also add some interactivity to our scene. A scene with no playable character isn't really fun to play with right? Right!

First off, locate and choose one of the possible characters you wish to play with in your *"[Prefabs]/[Players]"* folder. Drag it onto the screen, but don't press play yet.
Then, add two simple Layers. Name one *"Player"* and the other *"Ground"*.



Now, on your Player prefab, set the layer to *"Player"*.
Next, locate the object the player will be walking on. This object is found in the *"Resources/[Prefabs]/"* folder. Select the TilePrefab and set its layer to *"Ground"*.

We made sure that, when we are done building our level, our player can walk over it whenever we hit play.

That wasn't too hard

By now you should've noticed the white lines running across your screen after adding the grid script. These lines represent the possible placement spots for your tiles. By selecting the "**[GRID]**" object, you are able to set its width and height to anything you like (Only integers, not floating numbers). I made it so that a grid width and grid height of 1 unit will be exactly the same size as your sprite images. Leave the width and height at 1 and feel free to play around with the color of the grid to whatever suits your eye.

Images!

Let's get on with the fun and interesting part. In order to spawn images into the grid slots, you will have to insert your own atlas sprite sheet. (Don't worry, I have also included a sprite sheet with over 500 different sprites for you to experiment with).

To create an atlas, I used, and recommend, *TexturePackerGUI* available here: http://www.codeandweb.com/texturepacker. Simply insert your own PNG tile textures into the programme, play with the settings and publish a fresh tile sheet texture. The tile sheet and settings I used can be found here: http://i.imgur.com/gJMYB18.png. Also select the option "*Clean transparent pixels*" at the bottom that just fell out of screen.
*Set the common divisor to the width and height of your tile, mine are 70 x 70.

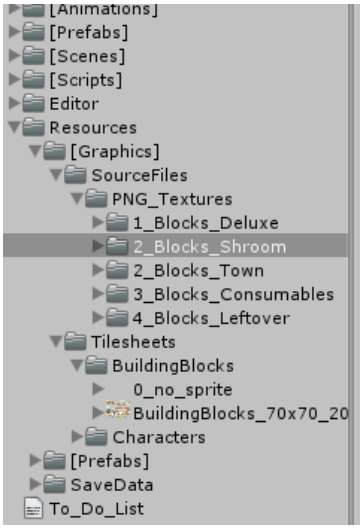
Once you have created your atlas, drag the tile sheet into Unity. The tile sheet has to have a specific path for the tile builder to be found. Either put it in:
"Resources\[Graphics]\Tilesheets\BuildingBlocks"

or change the path (in the ImageLibrary.cs file).

*"Object[] spriteImageFromDataPath = Resources.LoadAll**(@"[Graphics]/Tilesheets/BuildingBlocks**", typeof(Sprite));".*

Now, for every image that has been used in your atlas, you will also need each tile separate as a PNG image in your "*Resources\[Graphics]\SourceFiles\PNG_Textures"* folder or change its path to something different, but it has to be inside of the Resources folder. These PNG images are used as buttons in the editor window representing the sprites you will place inside the grid slots later on.

Your project folder should now look something like this:



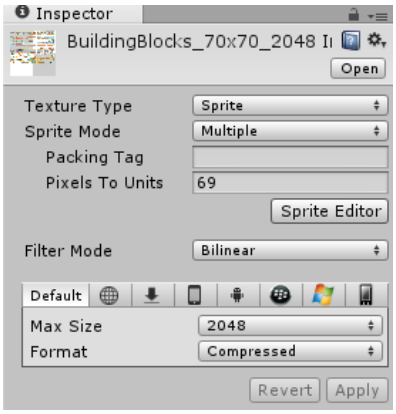
Inside the BuildingBlocks folder, I have added my tile sheet “BuildingBlocks_70x70_2048”. And inside my PNG_Textures folder I have added my PNG textures of all separate tiles. (These tiles are sorted in their own folder though).

Also notice the “*0_no_sprite*” texture. Make sure to have one fully transparent texture imported that isn’t visible to use as a ‘null’ space in case you ever need one. Make sure this texture is above all of your other tile sheets.

The reason why it has to be above all other tile sheets is because the way the sprites are imported into the editor tool. While importing the sprites (will get back to this later) into your level editor, it will load in all found sprites from top to bottom. Each sprite will receive its own unique tile integer. The 0_no_sprite sprite will receive the number 0 because it’s the first sprite being found by the editor tool. I use the integer 0 as a null check. Just leave it there, it won’t harm you, promise!

Next up: Importing

Click on your tile sheet, in my case “BuildingBlocks_70x70_2048” and change its settings to the following:



*Max size should be the size of your imported squared tile sheet.
*Pixels to Units should be the width/height of your PNG texture minus 1. This will reduce seams while placing tiles next to each other. My tiles are 70x70 so a Pixel to Unit ratio of 69 should do!

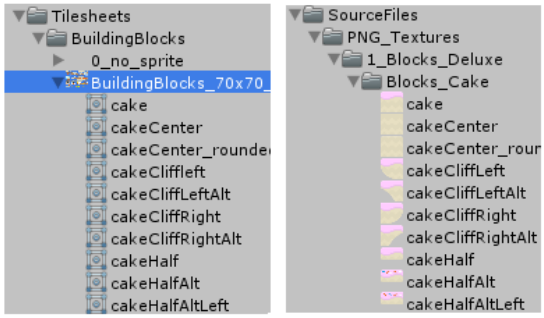
Next up, open the Sprite Editor and slice all your texture images. You can either do this automatically or by slicing a grid.

I chose to do this automatically because in the TexturePackerGUI toolkit, I have already separated each tile with a 4x4 offset. If you wish to do it by grid, then you will have to place every tile against each other. Both options have their own pro’s and con’s.

If you wish to do it automatically you will have to manually check each and every tile if its size is still 70x70 (the Sprite Editor cuts off alpha in textures).
If you wish to do this by grid-style, you are risking the chance of some textures to have an extra 1 pixel border on each side of the sprite. This 1 pixel border is taken from the texture space next to it which is rather sloppy.

Done slicing

When you are done slicing your images and are good to go, you have one more thing to worry about. Each and every sliced tile in your Sprite Editor has to have the same name as the PNG texture you use for it in the *SourceFiles/PNG_Textures* location. (Don't worry about capital letters, but I still recommend the use of capital letters). By default the tiles will be named to: "YourTileSheetName01", "YourTileSheetName02". After changing the names, it should look more like this:



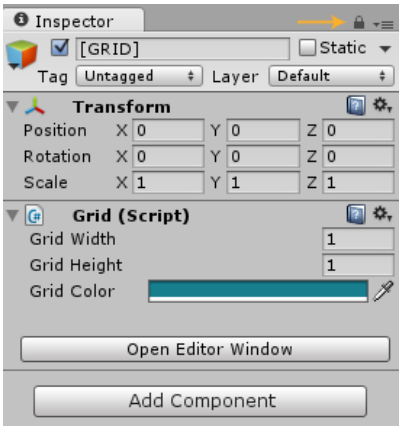
This looks and reads a lot better. Also make sure you do not have duplicated names for your tiles.

Almost there!

Now, if everything went by the books, Unity will be able to link the sprites (left hand side) to the texture images (right hand side). To clarify: The sprites are used In-game as tiles to spawn in the grid slots, and the textures are used In-editor as buttons. Once pressed on a tile button, unity will locate its sprite counterpart and use that sprite as the image to put on the TilePrefab.

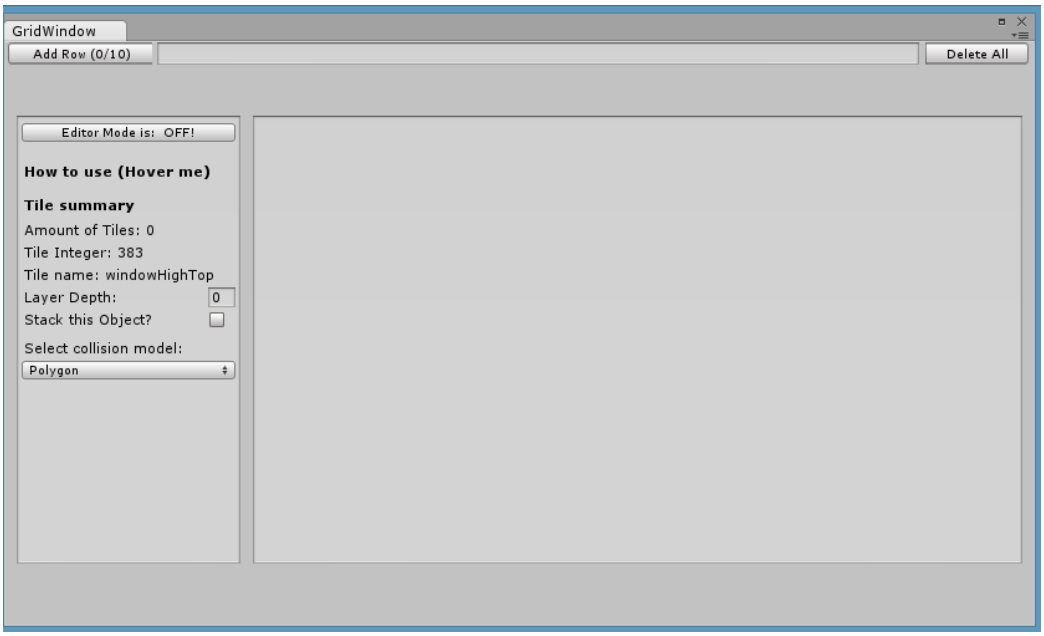
Editting mode

Next up, select your **[GRID]** GameObject and hit the "*Open Editor Window*" button. As you might have noticed, the little lock icon at the top of the inspector window (image below) has been locked. This happens automatically. Whenever you close the editor window, it will unlock the lock.



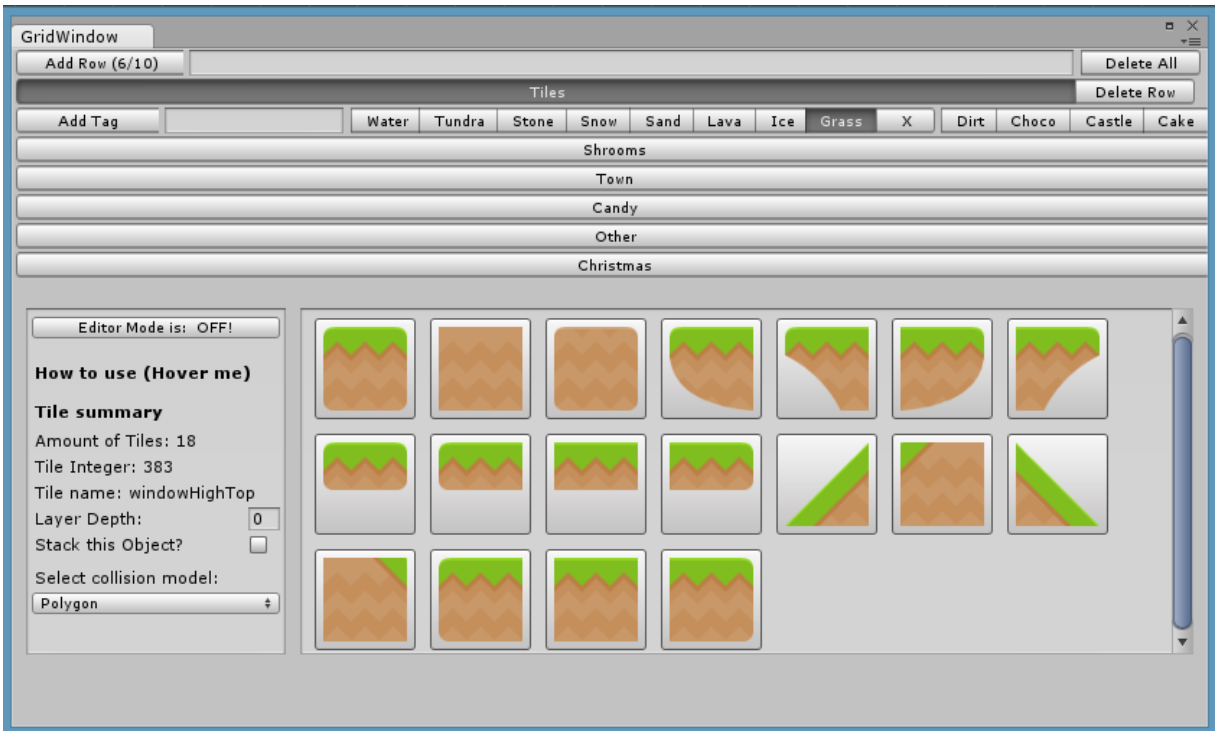
***Special note**; The inspector has to be locked on the **[GRID]** GameObject in order to let the editor tool know you want to work with it. Else it switches to the default unity behavior and the tool won't work properly.

A new window should show:



By default, its rather empty. But you are able to customize it to your own liking! By inserting a name and clicking "Add Row" you will insert a new category. Once pressed on that tab, a new dropdown follows. This category can hold all sorts of keywords you wish to add to find the tiles from your tile sheet. The way this system works is as the following: Whenever you insert a tag like "*Grass*", "*Candy*", "*Choco*", "*House*" or "*Cake*", it will find every sprite available with the matched name in it. It acts like a search bar that categorizes your tiles.
Note that Rows and Tags cannot start with a number, since numbers are used for internal code flow.

I have taken the liberty to insert a couple of generic tile names into the editor window for you to play around with:



Buttons, what do they do?

Add Row: Adds a new row that can hold new categories.
Delete All: This button deletes every row and tag you inserted.
Add Tag: This button adds a new tag, or category, into your row. These tags act as keywords. A tag named "*Grass*" will return every available sprite with the word "*Grass*" in it.
Editor Mode: Switch on to place, delete and copy and drag tiles. Switch off to stop editing.
X button: Next to the selected tag, this button will delete the selected tag.
Delete Row: Deletes the whole row including the tags accompanied with it.

Summary

Amount of tiles: The amount of tiles this tag holds.
Tile Integer: The unique number used in the code flow per tile texture.
Tile Name: Name of the PNG tile.
Layer Depth: Z-Depth of the tile.
Stack Object: Able to stack the selected tile onto another tile (Up to two tiles).
Collision Model: Different collision types, choose from: *Polygon*, *Box*, *Circle*, *Edge* or *None*.

More information can be found hovering upon the tile summary elements.

The good part

Now when you press on a tile button in the editor window, your current selected tile updates. Whenever you position your mouse over an empty grid space, press on the ‘a’ key to build your selected tile. Press the ‘d’ or Delete key to delete your tile. Press the ‘c’ key to copy a tile. Once copied, simply press ‘a’ to spawn the copied tile. Press Ctrl + Z to undo your action. You are also able to select tiles in your scene and drag them to other grid spaces while snapping!

Notes:
- The default keys of ‘a’, ‘d’ and ‘c’ can be changed in script. Go to the Grid.cs script and modify the following lines of code to whatever you like:

public char buildingKey = 'a';
public char deletingKey = 'd';
public char copyKey = 'c';

- Capital letters require the use of the shift-key. ‘A’ requires you to hold shift and press a.

- **Important note:** Edit the scripts with caution. Please do not edit scripts while in the process of building levels to prevent unwanted errors / behavior! Whenever you have edited the scripts, you will have to re-open your editor window for changes to affect.

Let’s Play!

Now that you have learned the basics, you are able to create all sorts of levels. This guide seems like a lot of information, but after a couple of tries you will get used to it quickly.

Go on and create your own level, press the play button, and let your character be free!
A quick little level: http://i.imgur.com/fe4onXV.png. Notice the low drawcalls, great for mobile!

Credits!

Thanks to Kenney Vleugels for his awesome sprites included in this package. More of his sprites can be found below at.
http://opengameart.org/content/platformer-art-complete-pack-often-updated

Imported summed up notes:

- Page 5: Notice the “0_no_sprite” texture
- Page 6: Pixels to Units should be the width/height of your PNG texture minus 1.
- Page 7: Each and every sliced tile in your Sprite Editor has to have the same name as the PNG texture you use for it in the SourceFiles/PNG_Textures location.
- Page 7: Make sure you do not have duplicated names for your tiles.
- Page 8: The inspector has to be locked on the **[GRID]** GameObject in order to let the editor tool know you want to work with it. (This happens automatically when you press the *‘’Open Editor Window’’* button)
- Page 10: Important note: Edit the scripts with caution. Please do not edit scripts while in the process of building levels to prevent unwanted errors / behavior! Whenever you have edited the scripts, you will have to re-open your editor window for changes to affect.