Level Editor

This document discusses the development of the Level Editor project.

JSON export format

The editor exports tile data in a JSON list with each object in the list containing `X` and `Y` coordinates (in tiles, not pixels), and the name of the tile type as a string. It's simple and naive, perhaps, but as a first iteration it is functional.

Challenges

A lack of time

Much more than with the Card Creator project, this project was developed on a very tight deadline (again: due to how I chose to prioritize other subjects), so there are a number of things I simply did not have the time to implement properly. The rushed nature of the project did, however, provide a nice counterbalance to the Card Creator's very intricate and interface-heavy architecture. This project too uses interfaces, but not as rigorously.

Canvas and image slicing

Probably the most daunting task before really starting the project was using the `Canvas` user control and finding a tile set and slicing it, i.e. how you deal with image data in WPF. In the end it turned out to not be as difficult as one might expect and some trial and error got me there in the end. The most confusing thing is still resource handling.

Non-updating components and messages

I had an issue where I had to pass on a command from a button to the view model, which then had to pass that down to all the tiles individually. This not only seemed unnecessarily complicated, but was also hard to implement. Then I discovered that messaging in WPF was super easy and that the `MVVM Light` toolkit has a service already available on the `ViewModelBase` class. This allowed me to simplify my button handling logic to something as simple as sending off a message, which each tile would then be able to intercept. Later I had an issue where the tile map wouldn't update and display the loaded map properly when loading from a file. This too, it turned out, was best solved by using messages.

And while the solution for this might not be ideal as it stands, it's fine at the current scale of the application. As the saying goes: 'everything is fast for small `n`'.

Highlights

This project saw me familiarizing myself further with WPF, and I found out about things like the `StackPanel` and `DataTemplates`, which let me dynamically generate UI elements based on code.

Apart from that: this project is much simpler, less 'engineered' and more to-the-point, which did provide a nice little break from the Card Creator project.

Unfinished business

The most notable items I did not get around to implementing / fixing

Unit tests

One of the requirements for the assessment is 30% unit test coverage. This sounds quite attainable and yet I was not able to get there. I did start writing tests for the `TileProvider` class, but due to the resource URI not working correctly when running the tests, the tests would always fail as the class couldn't initialize. Apart from this, though, there really aren't a lot of public methods to test in the project, save for the serialization logic in the `TileViewModel` where we want to convert pixels to tile coordinates. Most of the classes are POCOs (Plain Old CLR Object) or DTOs (Dumb Transfer Object) and don't really have any testable surface. Apart from the IO in `TileProvider`, the next best thing might be to test things like LINQ functions, and there really isn't much good to be had from that. I'm usually very adamant about testing, but in this case, I'm hard pressed to find something that feels like it would be worth testing.

Tile interaction

Having the tile sprites automatically adjust based on the neighboring tiles and create edges automatically would have been amazing, and was something I was really looking forward to working on, but due to time constraints I did not have time to implement this. Relating back to the unit testing point above, this would be a prime candidate for unit testing as it has a lot of application logic that needs to be validated.

Hard-coded values

Going back to the testing point above, the `TileProvider` class uses an image resource to create tiles and needs this to start up. Ideally this would be a parameter for a constructor along with tile width and height, so that the same tile provider can be instantiated with multiple images. In that case, though, it would not be a singleton, but that was a decision that made sense for this little MVP.

Layers / scenery / dynamic sizing

Moving on from tile interaction, it would also be great to have layers to the map, so that one could place scenery such as trees, plants, rocks etc on top of the first layer of tiles. Also: various height levels and a map that you can change the dimensions of are quite high on the list.

Unit placement

To get more game logic brought into the editor, after implementing the above features, it would be fun if you could mark tiles as 'unit starting positions' and mark them with factions. This would allow for a lot of work to be done with just this tool.