Clément Marescaux
08th of December 2017

PG5200-1 17H Tools programmering

Game Editor - Post-Mortem final

Project status

The editor now includes and starts on a Campaign Editor, which allows for management of campaigns. A Campaign is a container for all the game's data (maps, characters, etc), i.e. the data for a roleplaying game. The user has the possibility to load and save campaigns from and to JSON file format. The campaign maps can be edited by selecting the map first, and then switching to the Map Editor tab.

Unfortunately, while this version of the Game Editor will not crash upon loading a JSON file, the editing map functionality is not implemented… which I will detail below.

New structure

Taking into account the feedback received and the material presented in class, I decided to redesign the program's backend by changing the Tile and Map models, and integrating the MVVMLight framework.

New models

With the model structure I had for the Map model and the Tile hierarchy, I found myself locked and unable to go further in development. So I decided to opt for the dumb-and-straightforward approach of having a Map being a literal representation of its interpretation: a 2D array of Tiles, each of which contains direct information about what to display and what is located on it. While this is very inefficient as for memory space and speed, it allowed me to at least serialise and deserialise properly, and simplify updates to the Map model. The goal was to get a working model that could be serialized and deserialized, then refactor into something lighter afterwards.
However, it also meant that I had to change completely how the Views should work with the data. In addition, I really wanted to add the Campaign editor instead of just having standalone maps, so as to start giving some semblance of linkage between map instances.

MVVM

Adapting the solution for the MVVM pattern and integrating the MVVMlight framework proved both to be a blessing and a curse.
It's been a blessing in the sense that it did help me with the separation of data and functionality (and their representation). Also, it was my first attempt at event-based programming by using the *RelayCommand* and the *Messenger*, and it made things *so* much easier to make ViewModels communicate independently. As a new programming experience I really found it enjoyable, and when I understood how to work with it I managed to get results remarkably quickly (in comparison to the rest of the development…).

But MVVM has also been a curse, because doing “proper MVVM” requires to know WPF’s UserControls and the *Binding* hierarchy rather in depth, as the Views are only meant to consume the data they’re bound to. And since most of the data is delegated to the ModelViews, it felt like using a View’s code-behind would break the pattern, or seemed hacky and bug prone. So I ended up being stuck again when I had to refactor the grid editor for maps, because I did not manage to find a XAML structure that would properly bind back-and-forth with the map data and be interactive for the user at the same time.
The *TileView (*incl. *TileViewModel)* is **not implemented** and is only what remains of my attempt at creating dynamic UserControls - the reason the grid map of the Map Editor shows Tiles is because I hardcoded the body of *TileView* as a DataTemplate in the MapEditor XAML.

Conclusion

Since the editor is far from finished and not functional, saying that I’m disappointed would be an understatement. I feel like I have not managed to understand how WPF works, and therefore ended up with a limited set of options to get satisfactory results.

Features

Campaign editor:

- The editor loads and save campaigns from and to JSON with the respective buttons.
- The name of the campaign can be edited in the corresponding “Name” textbox.
- Maps added to the campaign are listed by name in a list box.
- A map can be removed by selecting it in the list and clicking “Remove map”, and a map can be added by clicking on “Add map”.
- Selecting a map in the list makes it active for editing in the Map editor. Changes during map editing persist when switching to another map.
- “Print campaign content” button is for debugging purposes, and will print both the map list of the “currentCampaign” DTO and the cached “CampaignMaps” list, to ensure synchronicity whenever updates occur.

Map editor:

- The name of the current map selected is automatically shown in the “Name” textbox, and updates the name of the map dynamically.
- A map can be imported or exported to JSON.
- The “Save” button overwrites the map data if the map already existed in the loaded campaign’s list of maps, otherwise adds it as a new map. Disabled when no campaign is loaded.
- While not yet fully implemented, the “Tile” active tile brush will update according to which tile sprite is selected in the “Terrain sprites” list.

- When selecting a tile in the grid, one can edit whether that tile is walkable, a spawn point for an NPC, and a transition to another map in the “Tile properties” box.
- **Not implemented:**
    - “Paint” editing mode (and “Select” mode is enabled by default, not by design)
    - NPC selection for Spawn tiles
    - Destination map selection for Transition tiles, although the dropdown ComboBox shows a list of the available destination maps.
- **Known bugs:**
    - If the map has been created this session, tile updates will propagate over all tiles. However, this is not the case if the map has been created by clicking on “Create debug map” ¯_(ツ)_/¯
    - Sometimes a tile won’t respond to selection. To allow selection again, try clicking in another tile on the same row.

Character Editor:

I have not had time to improve the character editor other than fix some serialization / deserialization bugs, and haven’t linked it to the other ViewModels.

- A character can be given various attributes by using the respective provided controls.
- The Healthpoints textbox will react to invalid data (requires integers)
- Characters can be saved to and loaded from JSON file format. The editor will automatically create a “characters” folder upon loading
- Although not implemented, the right-side pane will list the character’s “Inventory” (a list of strings).

Clément Marescaux
08th of December 2017

Game Editor - Post-Mortem v.1

Project

The editor is meant for the creation of a world which could be used for an adventure or role-playing game. The world is divided into maps, each of which consists of 8x8 tiles which all have at least one graphical sprite asset (no empty tiles). The editor should contain:

- A world editor **//To do**
- A map editor **// WIP**
- A character / creature editor **// WIP**

Structure

The editor is currently divided into tabs (one tab per tool), which will probably be subject to review before long as it is not very user friendly. Each tool is separated in its own XAML and is designed according to the MVVM design pattern through WPF and C#. For now, a lot of the view is very much directed by the code-behind, which is against how WPF is supposed to be used. The only reason for this is because I'm not yet sure how to make efficient UserControl / ItemsControl templates and components to encapsulate the data-to-view flow **all the while** keeping control of data flow between components. But my aim is to have the code-behind only handle events in the view, and link to the backend (asset loaders, DB / file managers, etc…).
I focused most of my efforts on the Map editor and tried to have a hierarchy of Tiles, each of which have a dedicated role. I have tried to design the hierarchy with separation of concerns in mind, but I am worried that this might lead to code rigidity because of an immovable inheritance tree of Tiles. I have thought about using the Decorator pattern, where each tile could have several roles at the same time (eg: a tile that can trigger both an enemy to spawn and a door to close).
I might change how the tiles are stored. At the moment, I feel that there is too high coupling between the Map editor and the Tile manager, and way too much responsibility given to the MapEditorControl.
In the next versions I would like to allow for characters / entities created within the editor to be directly available for placing in the Map, but that will be once I overcome the issues with Tile loading, and after I have created a basic World editor so that maps can link together.

Challenges - feedback request

- I would love some pointers on how to manage scope for design steps, because I feel like I'm drowning in a feature-creep nightmare while trying to learn WPF at the same time. I feel like the solution is an event-based architecture which would keep code separated in tight small classes, but I don't have much experience with custom-made event design patterns.

- I would also like to know how I can go about designing custom behaviour for Tiles in a serialisable fashion - by which I mean being able to save a tile’s type, coordinates, arguments and action inside the JSON file.
- Any feedback on anything particularly horrible or great in the present solution would be awesome as well, and what could (should?) be removed or added.