Part C: Design Reflection

The design had witnessed quite a few changes throughout different milestone. The most important changes made are those centered around software objects, how they relate to the domain model, each of their responsibility, and how that can give rise to the design choices of fields and methods.

In milestone A, many design choices are made based on incorrect assumptions of the game. For instance, monasteries do not consist of segments like roads and cities. The relationships between segments, tiles and features are also unclear. This had caused errors, complications and inconsistencies in the object model and the interaction diagrams. Another issue is the disparity between the digital version of the game and the cardboard version. In the cardboard version, scoring and validation are done by the players while these functions are performed automatically by the game. I was conflating the two during the design in milestone A, leading to many misconceptions.

In milestone B, the responsibility and functionality of each object were much clarified. For instance, each features should know the tiles they are consisted of; each segment should know if they link to features; each tile should know all of its segments and the features on the tile. With better understanding of the gameplay dynamics, I have gained a better picture of how to bound the responsibility of each object (only talk to their “neighbors” and no more) as well as exposing only adequate amount of information in its API. This allows for much faster code development, as we don’t have to wait for every object to be figured out to start writing code. In addition, I have also employed design patterns such as the strategy pattern, design template method and decorator patterns to address commonly encountered design issues. For instance, cities and roads are both extendible features, therefore if they inherent from the same abstract class, it will allow for code reuse. On the other hand, the strategy pattern is great for different feature types to substitute their scoring algorithm on the fly. The city scoring only differ from road scoring by the number of shields, thus a decorator can add additional functionality without repeating code. There are many challenges encountered, such as merging various features and each participating object’s responsibility.

In milestone C, the focus is on developing a GUI that can operate without interfering with the core of the game. Using the observer pattern, there’s a clear API for both the game and the GUI where the controllers are defined in between how these two API communicate with one another. This had enable flexibility as neither the GUI nor the game depend on one another’s implementation, or as limited as possible. Finally, the GUI also helps expose many hidden bugs that are had to discover relying only on unit testing in milestone B.