Software Engineering Methods Project Reflection

Team 24

During our project we have learned a lot of things in practice. First of all, we did not expect to be able to develop a game like this in two weeks and using correct coding practices turned out to have quite steep learning curve, but we've gained a lot of experience. The biggest change in our way of developing is that the ideas and processes that we learned in the lectures have become part of the process instead of being an afterthought.

We progressed a lot in our use of code review and pull request development. For the first version, most functionality was merged from the console and not requested via a pull request. After the first version, we started to use pull requests, but these requests were usually accepted and merged right away. This changed from assignment 3. Now every part of the code is reviewed twice before it is merged and we actually find improvements to the code. We did not use pull requests before, because we thought it would slow down development too much. Now that it is part of our routine, it only positively affects development.

Another area that we progressed in is tooling. For the first version, we had not set up Travis or any of the tools and, as with code review, it became part of our routine around assignment 2. We had agreed to use Maven site every time we submitted a pull request and this made us much more aware of the reports the tools generated. We improved our use of tooling to eliminate most warnings and all errors and Travis is functional for all branches. Travis has actually helped eliminate some errors in the game that were missed before pushing.

When we look at the code of the first version compared to the code we have now, we can see there is a lot more structure in the packages and the code. To illustrate our progress with regards to design patterns and structure, we will discuss a specific pattern: MVC. We have tried to implement a separation of the model, view and controller throughout the project, but this mostly resulted in some static classes that turned into God or data classes. As we learned more about the MVC design pattern in the lectures, we tried to improve the use of this design pattern, but as it required big refactoring, it ended up not working. One issue that complicated the use of this design pattern is the library Slick2D. It forced us into a certain code paradigm that made separating the model, view and controller harder to do. Another issue is the fact that we did not know the full extent of the design pattern when trying to implement it, which resulted in a half-baked MVC pattern without the right underlying patterns. In the final version, we have implemented a design pattern that - to our knowledge - follows the MVC pattern and has the correct structure necessary for easy extension and maintainability. Looking back at our progress, it is understandable that we were not able to implement the right MVC pattern right away, but for a new project we will know how to begin and set our project correctly.

In the lectures we learned about coding principles, like programming against an interface, and we have slowly incorporated that in the code too. Some team members were more 'perfectionistic' in pursuing these coding principles than others, what resulted in parts of the code following the right principles and others still feeling a little like it was 'hacked' together. Around the end of the project, the team members were more on par about design principles, so we all decided to commit to a refactor in order to improve code quality and the use of coding principles.

The last area that we will reflect on in regards to code quality and design is that of UML and Responsibility driven design. As with all the other areas, we did not have a lot of experience with these 'techniques' and did not use them to design our product yet, this resulted in 'hacked' code the same way our inexperience with coding principles resulted in 'hacked' code. A function was thought up and someone started coding right away. Now that we use UML and think about the responsibilities a class has, we actually take time to think through our decisions. This results in cleaner code and code that has better distribution of responsibilities and data.

We did not only grow in regards to code quality, tooling and design patterns, but also in group process. The first couple of weeks, we had a daily standup every day. For assignment 2 and 3, this declined and we did not communicate about the project enough. We picked it up again for the last couple assignments, as we found out communication was key. We also progressed in planning for sprints and knowing what tasks we could do and wanted to do. At the beginning, task assignment was quite vague and usually multiple people were assigned to one task. This did not work, since it requires two people to work together closely to finish that one task. A task needs one person responsible for it, so they can work on it even when they are not around the other people. We learned how to do this more as the project progressed and now our tasks are more specific and assigned to one person, so they feel responsible for completing that task.

As we wrap up this project there are still areas we would like to grow in, like testing, use of design patterns and communication during the project. We will try to improve this even in the last weeks, but take these with us as challenges for next projects. Concluding, we can see a lot of progress in the way we work, we have included valuable processes and methods in our routine and understand the way we have to approach a project more. After this project, we are confident that we can take on any project of this size.