**Sprint 3 Reflection**

**Team Number: 4**

Hunter Figgs, Jared Perkins, Sergei Fedulov, Patrick Haughn, Jeffrey Gaydos

CSE 3902, Professor Boggus

26 October 2020

Summary:

For sprint 3 the team got off to a bad start. We were all very busy for the start of the sprint, and no one managed to get a solid amount of work done. We managed to meet and agree on how each branch should communicate and what design patterns we should use. Near the middle of the sprint, we really began to tackle the work and get through everything, even as new issues arose. Our biggest issue for this sprint was definitely time, which played against us as other courses littered us with labs, projects, and midterms. In the end, we still pulled the sprint through to a good place and completed most of the required features for the sprint, along with tackling the bugs that came with them.

Planning and Documentation:

Burndown chart, task board picture needed

Our burndown chart had a rough start, this was a result of creating large tasks at the beginning and not splitting them up into smaller tasks. As a result, it took us a decent amount of time to finish the initial tasks. When we did get around to splitting them up into smaller tasks we ended up with a large spike in remaining work and were in a bit of a rush to complete them. Our late start due to work and responsibilities we had in our lives did not set us up well to keep up with the ideal trend given in the graph. Bugs and issues that popped up also made this difficult to maintain as they added additional tasks to our task board.

Implementation:

Code Quality:

Moving past Sprint 2 and into Sprint 3, we made some changes that would positively impact the overall code quality of the project. Some larger classes, such as the Game1 class, had sections of code broken up and delegated to different classes or entirely new classes. Part of this came with creating the screen and room classes, which were naturally needed as the complexity of the game increased (e.g. single room to multiple rooms each with their own set of enemies, items, etc.). This overall change of breaking up classes was done in order to shorten and lessen the responsibility of larger classes. In the long term, this will effectively lead to higher cohesion and a more maintainable code base.

General code quality was also improved for new code by utilizing paired programming, where two team members sat down together to tackle harder problems. By having two programmers contributing to the same code simultaneously, improvements like more readable naming emerge as well as utilization of helpful/more efficient C# syntax. This especially came in handy when writing the “All Collision Handler” class. Figuring out the best “C#” way to take in two object types and a side character and determine the proper command to execute was difficult as we are relatively new to the language. These kinds of challenges are much easier and more beneficial when working as a team.