Team 22 : Memoji

Sprint 3 Retrospective

Team Members

Scrum Master - Jonathan Poholarz (jpoholar@purdue.edu)
Maxwell Jones (jone1268@purdue.edu)
Manoj Polisetti (mpoliset@purdue.edu)
Andrew Ring (amring@purdue.edu)
Delun Shi (shi272@purdue.edu)

What Went Well

Our final sprint aimed to tie up loose ends in our application and get it to a playable state as well as refactor some of our code. We successfully completed all of our stories and were able to incorporate a new visual theme to the application. This allowed us to execute a clean, straight-forward demo presentation where we could show off all of our hard work. No real issues arose during the demo.

Code refactoring went pretty well for our UI screen code. Overall, we were able to significantly reduce the number of lines of code as well as make it more readable using newly defined coding standards from Sprint 2. As this front-end refactor concluded, we were able to add theme files into our program which can flexibly be altered to change the style and design of our program as needed.

The rest of the stories did not cause too much trouble either as we, as a team, had figured out a workflow that seemed to work well for us.

What Did Not Go Well

We did run into a few minor problems throughout the sprint. Firstly, we discovered a few more networking details that caused problems. Specifically, we encountered a bug with the Godot Engine version 3.0 where networking sockets would not update if a connection was dropped. Upon further investigation, we learned that this was a software problem fixed in the next major release of Godot (3.1) released in March. However, because this version is not backwards-compatible, we were not able to fix the small networking bug in this sprint. Upgrading versions would have caused many additional problems we were not prepared to handle in our allotted time frame for development. In the future, we would plan to upgrade to Godot 3.1 and rewrite our networking code to use the newly improved networking library instead of hacking together our own solution.

Similar to previous sprints, testing was, again, rather difficult. It typically becomes a team effort as running 4 clients on one's own computer is time consuming and difficult to debug. We found that resorting to a team effort in testing the application would go more smoothly. When debugging frequent crashes late in the program, it takes a significant amount of time to restart everyone's applications each and every crash.

Although we did refactor much of the screen code this sprint, we did not refactor some of our main game logic code which is in dire need of it. Files such as our GameStateManager script have become significantly bloated and confusing to work with. Some variables and functions have gone unused but were not marked as such, making it difficult to determine which code exactly is responsible for a bug. In some cases, the same information would be defined in multiple scripts, making solving bugs harder than it needed to be.

Also carried over from previous sprints, our overall game logic had to be tweaked to incorporate new features. Due to conflicting understanding of which code belonged to which scripts, we ran into difficulty planning adequately. There were times where team members did not know who was responsible for which code. In order to combat this problem, we needed to devote time to rewriting the core game logic which put us behind schedule.

How To Improve

In the future, we have determined a number of ways in which we would be able to improve our workflow and our project. One of the main ways we could improve would be to incorporate better testing protocols and work toward test-driven development. Writing test functions for each of our scripts would be useful to determine if changes have broken previous features as well as make the code more stable before manual testing kicks in.

It would also be a good idea to apply clean code principles such as checking the negatives of conditions and returning from functions instead of nesting 4 or 5 levels of if statements. In addition, committing to our coding style to make screens more flexible would be a good idea. We would build UI screens with scalability and theme application in mind so as to reduce the refactoring that would be needed later.

Another good idea would be to comment and document our code more frequently with descriptions for functions and their execution flow. This would make different parts of the codebase accessible to team members other than just the author of a given script. In addition to removing unused code, this would reduce confusion attempting to make sense of someone else's code. We should also communicate with each other about the details of implementation so that two different team members are able to write code for interlocking systems without needed to rewrite one side of the logic to accommodate the other side.