Gift Mkwara - gmk49 Joseph Manifold - jma459

SENG201 Project Report

Our game was structured to be modular and expandable, to accommodate our feature set which included an on-the-fly inventory system, which allowed users to upgrade their monsters mid-battle. Special abilities, moods, and a rarity system.

We first partitioned our game into eight main packages based on the requirements. The GUI, main, monsters, battles, items, shop, player, and random. Each package contained related classes that would handle one aspect of our game. The GUIs used inheritance to modularise the various screens in our game. Each screen inherited an abstract screen class which provided functionality for opening, closing, and hiding the screens. These screens would communicate with our game environment UI class to keep the GUI code separate from our game environment class.

Monsters were generated by the facade class, monster factory. Which communicated with the monster creator class to generate any number of unique monsters. The generic monster class used decomposition to modularise related attributes such as health data and battle data.

The items also used inheritance by extending an abstract item class which provided abstract methods for activation that each item could implement according to its specification.
Both the monsters and the items implemented a purchasable interface. Which provided methods for retrieving the price information and the descriptions. Items and monsters were sold through the shop class which communicated with the game environment, monster factory, and item class to restock the shop with items and monsters based on the game completion.

Battles were generated pseudo-randomly by the battle selection class and were scaled based on the difficulty and game completion. The battle class would communicate with the battle selection class to retrieve the chosen battle and mark it as used. The battle class facilitated all classes related to the battle system. This includes the opponent logic class and the special abilities. Which used inheritance by extending an abstract special ability class to provide functionality for activation and deactivation. The AI opponent was represented by a generic player class. While the user inherited this class and was given additional functionality for storing items, gold, and XP. The transmission of gold, XP, monsters, and items was facilitated through the game environment class.

Our overall test coverage was 38% however, excluding our GUI classes we had an average of 80% test coverage across all our classes. Many of our classes depended upon random events which were difficult to test. But overall we ensured to cover the core functionality of our game with our test cases.

Thoughts, feedback, and retrospective

This project, despite its challenging and at times seemingly overwhelming requirements, was enjoyable and insightful. We learned a lot about what it's like to create, work on and improve a relatively larger project under a time constraint. Although the time constraint - even with the break included - was stressful and at times caused us to make sacrifices in other areas of our life and study, it was useful to understand what that could be like in the real world.
As we worked through the project, we also identified areas that took longer than we anticipated and things we did not foresee. This included the great amount of time we spent debugging our game. Although we expected errors to be numerous early on, it seemed as though the more errors we corrected, the more errors we identified. This ended up taking up an enormous portion of our time as we were linking up all the mechanics and functionality of our game.

In terms of feedback, we felt as though the time constraint was reasonable for the most part. We did, however, feel that we would be a lot happier working on the project if there weren't still time-consuming weekly labs happening when the project was running. One of these labs, in particular, the advanced Java lab, took rather long and was very challenging. This ate into the time we could have used on the project.

The resources we were provided were helpful, although we found ourselves referring to students who'd done the course in the past to get a lot of information about what the markers will look for, what we should include in our javadocs, ideas for code design and architecture, and how we should be going at any given point in the course of the project.

A reliable and skilled teammate was crucial. Having another person we could rely on helped us with stress management and meant we could explore what it's like to create a larger-scale java program using OO-programming in a team.
A big challenge that we handled well was delegating tasks and communicating our work and progress to one another regularly. We made sure to avoid stepping on one another's toes as best we could regularly, ensuring that we weren't wasting time doing work that had already been done. Another challenge was creative differences. We handled this by negotiating compromises and looking at the bigger picture so as not to get hung up on the finer details. Other things we noticed were how useful it was to modularise everything from the beginning and to keep scalability in mind when building our code.

Effort and contribution

Although we didn't keep track beyond the weekly reports, we likely spent a total of 260 hours on this altogether between the two of us. This includes a 65-hour week in the last week before the due date. Our effort was well distributed and both of us are satisfied with the other's work output and contribution which we determine to be 50% each. This amounts to 130 hours each over 8 weeks. Much of our time was spent working on game logic, hooking up the logic to the GUI screens, and testing and debugging our game. Javadoc and JUnit testing outside of debugging took only a small portion of our time. Overwhelmingly, debugging took the longest.