Drake Rutherford

OOAD Final Report

Roguelike engine

Final State of System

The file-reading function is in place and allows the user to define the enemies and items that can spawn. The customizability is small; all enemies behave the same, the only differences between them being their ASCII representation and their stats. There are three types of items—healing items, weapons, and armor.

The basics of roguelike gameplay are almost in place. The player moves their character and interacts with the environment through the keyboard. Text is displayed on the top of the screen to inform the player of important events. The player can attack enemies and use their inventory. Missing features include:

- A leveling system
- Randomized floors
- Fail and win conditions

At the moment, the "game" is a square room with randomly placed items and enemies that does nothing more than demonstrate the basic features implemented. Due to time and workload constraints this is all that could be completed.

Third-Party Code vs Original

The system uses the PDCurses library to display characters on the terminal window. PDCurses is a public-domain Windows port of Curses, a library for developing text user interface applications on Unix-like systems. Curses was used in the development of the original *Rogue*.

PDCurses can be downloaded at https://pdcurses.org/

OOAD Process Thoughts

1. Pre-code planning. I found the activity diagram to be the most helpful in developing this project. The class diagram did little for the project; while it's easy to define objects and interfaces on paper, there are always more dependencies needed for the interfaces to actually function than initially assumed, and objects end up more tightly coupled than I anticipated or would like.
2. Teamwork. I wish I had worked on a team rather than alone. When working solo on a project like a game (something you only expect people to play rather than use the code from) OO principles can start to feel arbitrary and limiting—why make a variable private when *I* know exactly how it ought to be used, and no one else will ever see, much less use, the code? In a team I believe I would've developed a better appreciation for the benefits of OOP.
3. Refactoring. I developed a greater appreciation for refactoring through this project. It does for me what drawing up a class diagram didn't, because once I have some code down I have a better idea of the dependencies that exist and how to dispel them.