Team: Tyler Behm, John Chamberlin, Josh Killinger

Title: 2D Roguelike Game

Description: A 2D top down game based on the 1980 game Rogue but with sprite graphics. Players will guide a main character to navigate a dungeon that contains enemies and items.

Platform/Environment: Unity and C#

Programming Languages:

Programming Language	Tyler Behm	John Chamberlin	Josh Killinger
Unity	Beginner	Beginner	Intermediate
C#	Beginner	Beginner	Guru

Functionality:

- Player can move main character
- Enemies attack player
- Dungeon is generated
- 2D sprites visualize the game actions

[OPTIONAL] Stretch Functionality:

- Add more enemies
- Add more items

For our group project, we propose to implement a 2D Roguelike game. In the following paragraphs of our proposal, we will explain what a 2D Roguelike game is, how we intend on working on it, and why it makes a good object-oriented project.

A Rougelike game is modeled after the 1980 game Rogue. A Roguelike has the player taking control of their main character. The player guides their character through a dungeon as seen through top down 2D perspective. The character may encounter enemies and other obstacles. They may use power ups or other items available in the dungeon. Most commonly sprite graphics are used to depict the character, enemies, and everything else in the game. This helps to capture the retro feel and user experience.

For our project, we will use Unity and C# to implement the game from the Unity intermediate-level tutorial exactly so we will have a working product. https://unity3d.com/learn/tutorials/projects/2d-roguelike-tutorial

Then we add ghost enemies (they possess the character and forcibly move them away wasting player's turns) and invincibility power up (10 turn immunity to enemies) objects. This will be an extension to add extra objects. Then we refactor the game away from level progression towards screen based generated map exploration. This should be a refactor because we need to allow the player to return to the previous map square they just exited. Then we work from there time permitting.

This is a good object-oriented project because the tutorial takes advantage of many object-oriented programming paradigms. They create an abstract class MovingObject which is the class of the main character and enemy classes. This is an example of the Dependency Inversion Principle. Our extensions to the tutorial game will make use of the Open-Close Principle. Finally, our refactor towards the screen based generated map exploration will rely on good coupling to minimize the number of classes necessary to change.