***Preliminary Specification:* Cave Adventure**

1. *Purpose*

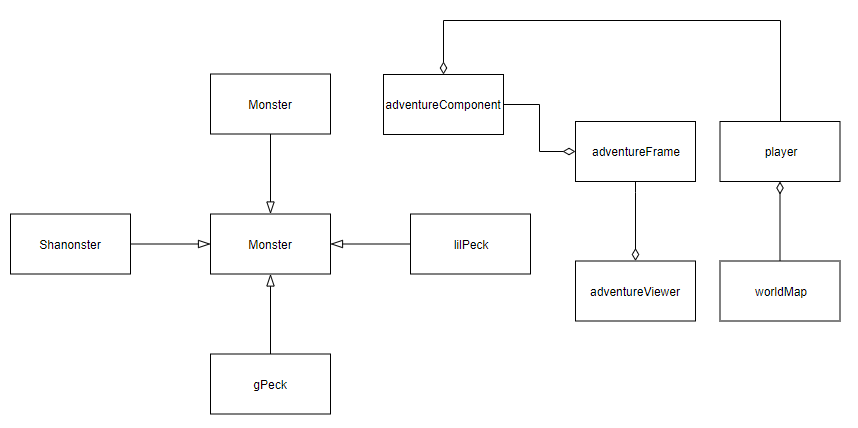
In this project we will make a 2D adventure game called Cave Adventure. Through random generation of upgrades and monsters, *Cave Adventure* creates a new experience with each playthrough. With our *Cave Adventure* game, the possibilities are endless.

*II. Functional Overview*

The game starts with the player, represented by the ‘@’, spawning in the center of “the cave” -- a 31 by 31 square grid. The player can be moved with the WASD keys along each square of the grid. As the player traverses the map they may come upon monsters and other items such as health or damage upgrades and traps. The player must reach the exit which is not visible. With everything being invisible until you step on it the player is essentially blind. If the player steps on a square of the grid with a monster, it will enter a battle. A player has a certain health and a damage level that it inflict on others. In the battle, the monster drains the player’s health in constant increments. The player can fight back by clicking a button to inflict damage on the monster. The first opponent to reach 0 health loses. If the player loses, they are transported back to their starting point (the center of the grid) with full health. Otherwise, the player continues exploring the playing field.

The player character, monsters, and playing field are represented through simple symbols. For example, the player character is represented by an ampersand (@). Grid points are represented by periods (.) . Before monsters are encountered, their grid plot is a period; afterwards, it becomes an exclamation point (!).

1. Structural Design
   1. The map of the cave (WorldMap) has a 2D array of squares (WorldMapSquare) as the grid the player can step on.
   2. AdventureFrame takes in user input and handles the logic. It looks for WASD key presses and clicks during a battle. In response, it calls the corresponding methods in AdventureComponent to draw the map.
   3. AdventureComponent draws the window’s contents. It has different methods to draw different parts of the window(the map, the sidebar).
   4. AdventureViewer opens the JFrame window and attaches AdventureFrame which calls the AdventureComponent to draw the window’s contents.
2. Object-Oriented Design
   1. WorldMap represents the map of the cave. Each WorldMapSquare represents one square of the grid, the location the player steps on.
   2. The Player is what the user of the program controls. It has its own health. damage, and it uses a WorldMap to play.
   3. Monster is an enemy with a health, damage, and rate of attack. It isn’t attached to any WorldMap, but will appear in a Battle. There are several types of Monsters, that are subclasses of the Monster superclass.
   4. A Battle takes a Player and a Monster and reduces their health based on the Player and Monster’s damage.



**Figure 1. Inheritance Diagram for “Cave Adventure”**

1. Detailed Design
   1. The detailed specs for the CaveAdventure classes have been generated from the Javadoc comments in the source files. Open index.html to see the documentation. ( When completed )
2. Testing
   1. Create a WorldMap of varying row and column numbers. Make sure the AdventureViewer opens a new window and the AdventureComponent draws the correct WorldMap 2D array grid properly.
   2. Create a Player and give them different (and sometimes out of bounds) row/column combinations to start on in their WorldMap.
   3. Open the window and test the KeyListener in AdventureFrame by pressing keys. Make sure WASD allows the player to “move”, by watching their row/column numbers on the sidebar in the window change and watching their ‘@’ avatar move around the grid. Control the players to the edge of the grid to make sure there aren’t any out of bound issues.
   4. Make a Monster with different health and damage levels (try negative numbers too). Create a Battle between your player and the monster. Check to see the player’s health is diminishing over time. Let the monster win the battle and check to see if the player’s row/column position resets and their health returns to full. Let the player win and make sure the player’s health is correct (it should have lost some from the battle).
   5. When a Player enters a Battle, the display and KeyListener should prevent the Player from moving around the map until the Battle is resolved. Check to see the player’s row/column position stays the same during a Battle even when WASD is pressed and that it can change afterward. A new window with buttons should appear when a Battle is invoked and should disappear afterwards. Test all buttons in the window and check if they have the correct effect on Player/Monster health. Then try closing the window in the middle of a Battle. The Battle should stop, but it doesn’t count as a victory for the player (no health increase, no inventory drops). Repeat with Monster subclasses.

.