***Functional 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 *Cave Adventure*, 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 grid is initially blank -- composed entirely of space (‘ ’) characters, so the player is initially blind to the layout of the map. The player can be moved with the WASD keys along each square of the grid. With each movement, a 3 by 3 area around the player’s location in the grid is revealed.. As the player traverses the map they may come upon monsters that randomly spawn with each step. As the map is revealed, they can reach locations (shown as letter characters) that affects the player’s health, upgrades damage and block, or has other effects. The player must search around and reveal the map to find the exit, shown as a green ‘E’. Stepping on the ‘E’ wins the game and shows the end screen. If the player steps on a square of the grid that happens to spawn a monster, it will enter a battle with that monster. A player has a certain health and a damage level that it inflict on others. The monster likewise has a health and damage to inflict on the player. A new window for the battle opens, and the player and monster take turns inflicting damage on each other. The player can fight by clicking a button to inflict damage on the monster. After the player’s turn, the monster will hurt the player’s health. The first opponent to reach 0 health loses and the window closes as the battle ends. If the player wins, they get an extra 5 health and can continue exploring the map. If the player loses, the end screen is displayed and they need to close the game window.

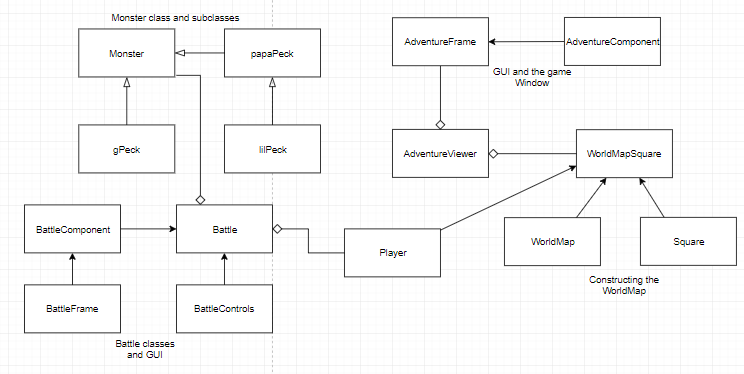
The player character, monsters, and playing field are represented through simple symbols. For example, the player character is represented by an ampersand (‘@’). Grid points are represented by periods (‘.’). Locations are represented by letters (‘P’, ‘H’, ‘D’, ‘B’).

*III. 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. It also opens the BattleFrame when a Battle is triggered.
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.
5. BattleFrame opens a JFrame with a BattleComponent and BattleControls GUI.
6. BattleComponent draws the health of the Player and Monster in response to the current state of the Battle.
7. BattleControls handles the logic and progression of the Battle. Pressing the button of the GUI is the player’s attack. It closes BattleFrame when the Battle is over.

*IV. 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 (gPeck, lilPeck, papaPeck).
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”**

*V. Detailed Design*

The detailed specs for the classes have been generated from the Javadoc comments in the source files. Open index.html to see the documentation.

*VI. 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. Check if each step reveals a 3 by 3 diamond of grid around the Player.
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. Repeat with Monster subclasses.
6. Visit the different locations and check to see if they have the intended effect on the Player (check Javadoc for the effects).
7. Check the endings of the game. Make the Player reach 0 health after a battle. The game screen should display the “Game Over” red screen and the KeyListener should be deactivated. Make the Player step on the exit square. The game screen should display “You Win” green screen and the KeyListener should be deactivated.

.