External Libraries Used

For the creation of this game, the library JavaFX was chosen as the graphics framework. This is because it allowed the graphical user interface (GUI) design to be simpler to implement and debug. Many functionalities of JavaFX were used, such as Scene, Button, ImageView, KeyEvent, Text, Label, etc. Java.util is another Java library that was included to time the user’s gameplay and reports it back to them at the end of the game. To clarify, the end of the game can be achieved upon the player either winning or losing. The Java. util library was also used for timing the appearance and disappearance of the bonus rewards in their positions.

Overall Approach

When the game first runs, an initial menu page is displayed with two buttons displaying “enter” and “help”. The “enter” button brings the player to the main page of the game where the game begins. Upon clicking the “help” button, there follows a series of pages (scenes) that can help the first-time players understand the rules of the game before leading them to another “start” button to begin playing. The first-page scene’s implementations were done in the JavaFX class, as all these features were to be displayed on the main game window. Each new window was implemented as a JavaFX scene with JavaFX buttons to switch between them.

The game begins and a maze appears with the main character at the start position, the enemy in the center of the map, and the rewards and punishments scattered throughout the maze. To create the maze, the map was typed into a “maze” text file, the 0’s representing the background (walkable tiles), 1’s as the position of the walls, 2’s being the regular rewards, 4’s as the punishments, 5’s where the bonus rewards will appear and disappear, and 6’s is the gate, which is initially closed. There are also 3’s and 7’s, but they are not initially placed in the text files. The 3’s are the bonus rewards and the 7’s appear when all the regular awards are collected, opening the gate for the player to pass through and win the game (in other words, 7’s take the place of the 6’s). In the JavaFX class, the text file is read and the image assigned to each number in the maze is printed in their designated tiles. On the right side of the game page, the timer and score are presented to the player. The timer increases as the game goes on and was implemented using Java.util’s timer to calculate the time passed and JavaFX’s Timeline for the animation of the time updating on the screen every second.

The main character is displayed in the tiles marked with 6's at the start of the game. The implemented CharacterEntity class and the MainCharacter class extend CharacterEntity are for setting the character's position. In the JavaFX class both the main character and enemy's appearance are randomized thus every time the game starts, a new emoji is picked to be the player and one to be the enemy. To move the character, JavaFx's KeyHandler is used such that when the arrow keys are pressed, the character moves in each of the four directions. Collision detection was implemented by checking if the tile the player is moving into is a valid tile (anything other than a 1, which is a wall). However, the enemy moves towards the main character based on an algorithm that checks which tiles were already visited by the player and follows the same path to reach it. The enemy initially waits for 1000 milliseconds and then starts following the player at a fixed speed using the timer and sleep functions of Java.util. If the user has not moved from the starting position, it just moves towards the start position. The area around the enemy also needs to be continuously updated for it to be displayed in its new position.

A score class is made to include the main methods and information about the reward system. This includes adding the value of each reward to the current score, subtracting points for stepping over a punishment, and the class ultimately decides when the user can win the game (after they have gathered all the regular rewards). The purpose of this class as opposed to coding all the information in the JavaFX class is to use information hiding to protect the code and allow future access to parts of the code separately without re-coding everything. However, the rewards are implemented in the JavaFX class as they have to be displayed on the game's main page. The regular rewards add 3 points, the bonus rewards add 5 points, and the punishments deduct 5 points from the score and each reward/punishment disappears from the map once the user steps through them. The bonus rewards are implemented to have two sets of positions and a Java.util timer was created to set a random amount of time to make the bonus rewards disappear from their current positions and reappear in the other set of positions.

Adjustments and Modifications from Phase 1

As the process of implementing the game began, a few adjustments needed to be made to the initial plan that the UML diagram described. Prior to Phase 2, it was not clear on how we would actually place each functionality of the game so as we got deeper into the phase, some things had to be altered. The map was implemented in a different format than in the UML diagram since, with using JavaFX, we found it unnecessary to create a separate class called board. In addition, the gate (start and end position) does not follow the UML diagram and does not have its own class; instead, it is also added as a part of the JavaFX class that changes status when the player has reached the

required final score. In JavaFX, everything can be displayed on the window while being in the same class which is later called in the main class. The character and main character classes are quite similar to what was planned and likewise, a class called moveEnemy was implemented as anticipated. For the rewards, however, instead of setting different classes for each type of reward, only one “Score” class was used to combine and include all the features. This was possible due to the flexibility of the way the map was implemented.

Division of Roles and Responsibilities

In this phase, the responsibilities were partially divided between everyone in the earlier stages. But regardless of the assigned sections, everyone helped each other with implementing the different ideas by adding to or editing the parts that needed further attention. The basic division of responsibilities was as follows:

Beatrice Chan - 301 357 564:

- Initial landing page implementation
- Instruction page implementation
- Score/timer display next to the game
- Final score and time on the end screen

Samuel Jen - 301 390 517:

- Logo, landing, instruction, game result page design (using Photoshop)
- Score system implementation
- Randomly timed appearance/disappearance of the bonus reward
- Interactions between the main character and the items on the map
- Randomized enemy appearance
- Gate display hint (orange to blue) when all rewards are collected

Matthew Wells - 301 396 858:

- Entire map board creation
- Enemy’s search algorithm
- Code optimization and peer assistance

Sepinoud Fasihimajd - 301 373 520:

- Main Character’s movement using key handlers
- Adding collision detection
- Randomizing the main character’s appearance
- Gathering and writing the final report

Measures to Enhance the Quality of Code

Some measures were taken to enhance the quality and performance of the game. First of all, the maze's map was not hard-coded into the program, instead, it was made as a text file imported by the program. Doing this allows the possibility to create different mazes for further improving the game. On the other hand, to avoid repeating the same parts of code for different purposes, classes and methods were created as much as possible to ensure that each functionality was implemented only once. One of the most important measures that were taken to optimize the code and help to decrease the lag was to change the way the map would update after each change or movement. At first, it was implemented such that after the main character or the enemy moved, the entire map (scene) would be updated to display the changes. After noticing this caused significant lag, this was changed so that only the tiles around each character would update.

Biggest Challenges

There were quite a few challenges that needed to be tackled in this assignment. First of all, working with a group and matching your way of coding with everyone else's can be a difficult task. This is because all the code is tightly interconnected and to be able to change one part, understanding the implementation of the rest of the program is needed. Any bug created by one section could affect the functionality of the entire program. Another challenge was needing to adapt the general functions and features of several library objects to fit the purposes of our program. For example, the timer display was mostly intended (in JavaFX's documentation) to be used as a standalone program so adapting it to a smaller scale was difficult. Furthermore, it was challenging to implement the randomly timed appearance and disappearance of the bonus points and the updating of the map accordingly. However, without a doubt, the most challenging part was figuring out the path-finding algorithm for the enemy to move closer to the main character. Many algorithms had come to mind for achieving this but they were either inefficient or incompatible with our situation.