CMPT 276: Phase 2 Report

Efe Erhan
Brianna Espena
Ruochun (Jasmine) Liu
Moon Wang

Implementation Approach

First, we discussed and decided how our game would work as well as details that weren't clarified in our Phase 1 description. We started by creating the different classes and methods specified in our UML diagram. Keeping the use cases in mind, our group modified the classes, by changing relationships, and adding, and editing methods. Our approach was mainly evolutional and we learned by doing. We implemented the game feature by feature, and restructuring when needed. Towards the end, we wanted to ensure connectedness of every element to enable ease of use. No external libraries were used to create our game.

Adjustments and Modifications

- The Tracker class was added as the parent class for scoreTracker and timeTracker. We decided to do it since some methods of scoreTracker and timeTracker are the same.
- The score class was removed because we found the getMark() and addMark() method can be done in the scoreTracker class.
- The Game class became the main class because it already acted as the main class.
- The Position class was removed since position is integrated in all JPanel/JLabels.

- The NonAnimatedEnemy and MovingEnemy became children of the RegularRewards class. NonAnimatedEnemy was given a value of -1, while MovingEnemy brings the player straight to the Game Over screen.
- A new class called collisionBox was created to handle player and element collision. Various methods and members changed as understanding of Swing's framework grew.
- The UserInterface class morphed into the JFrame AppWindow, which is the topmost container.
- The Direction enumeration class was removed to suit a previous implementation of directions and collision handling.

Management Process and Division of Roles

When Phase 2 was released we discussed which member would implement certain classes. We divided the work by classes. Each member was in charge of creating 3-5 classes as well as the methods and attributes in those classes. We distributed these assignments to our strengths. As a group, we set a deadline for completing the classes to ensure we had enough time to combine all of our work. When we just finished our class implementations, we shared them with each other. After we tested our implementations, we shared them again if they had been updated. So everyone can know each other's progress, and we could timely communicate any problem.

Enhancing Code Quality

To enhance the quality of our code, we individually tested our code and reviewed each other's. We all tested the methods and attributes of each class based on the possible

use cases. Then, we sent our code to each other and gave constructive feedback on how it could be improved. After making the modifications, we each tested and shared it with each other again.

Biggest Challenges

The biggest challenges we faced during this phase was making sure the code each of us wrote worked coherently with everyone else's. We prepared for this challenge by allotting time specifically for putting our code together and fixing any issues and inconsistencies.

Working with the JFrame libraries had a very steep learning curve, especially when having to fix past implementations based on new knowledge. But once we figured it out, it was very streamlined and intuitive.

Another challenge we faced was adapting to how to work with Maven. All of us haven't worked with Maven before. So firstly, we did research separately, then we shared some useful websites and tutorial videos to help each other understand Maven better. Then, we realized, we didn't really need its dependencies control, since we were only working with the included Java libraries.