Phase 2 Report

CMPT 276
Group 9

Overall Approach to Game Implementation:

The very first step of implementation was to set up an in-person meeting to discuss a general plan for how to divide the project among each group member. During the first meeting, we decided to allocate the project by classes because the UML diagram was the only resource we had at the time. However, we soon realized that our UML diagram was not good enough to finish the project. For example, some of us had inappropriate access modifiers and names so that we were spending too much time understanding each other's work. Therefore, we decided to stop the project and start from scratch and we agreed to make decisions on specific details before we divided the project into parts. For example, we had to make sure that we used the same libraries for creating window applications such as Swing. Also, we had to make decisions on the naming convention for uniformity. We had to determine the names for certain things such as our Maven project and classes because we knew that we would be required to use different packages and call different functions. At this point, we felt the necessity to start the project together, as opposed to working individually and remotely. Although the initial plan from the first meeting was to divide the work and to create branches on the Git server for each person, our group agreed that working on the project physically together could eliminate the problems that occurred during the past meetings on zoom for Phase 1.

One of the disadvantages of physical meetings was setting up a venue. Online meetings were easier to set up and they could save a lot of commuting time while in-person meetings required a space and more time. Therefore, we decided to meet for a longer time for a shorter overall period. We used one of the study rooms at the Student Union Building for all our meetings and each of them was at least three hours long.

Having little knowledge on Java and Maven, we were not sure which libraries we needed to begin our project. We spent more time doing research and finding out the functions of the libraries than writing the code. Also, each of us had different levels of knowledge, so we tried to continuously check on each other as we progressed.

Although we deviated from the UML diagram, we tried to follow it as much as we could. The UML diagram still helped us a lot because it reminded us of the features of the game that we had to implement without referring back to the project description again. Also, our Use Cases helped us set a goal for each meeting session. We tried to finish at least one Use Case per meeting since we were meeting in-person and thus did not want to spend a longer time on the

project. Like the UML diagram, we were not able to follow the Use Cases identically, but the Use Cases showed us the details that we had to be aware of which were not present in the UML diagram.

Overall, the experience of this process enabled us to learn more about game development and team collaboration. Despite any individual adversity faced, all group members were available to help with both technical and moral support. Again, the varying levels of knowledge and speed of learning made us spend lots of time helping each other. As a result, we found most of our time being spent coordinating the team. Moreover, we learned how crucial a detailed and well-thought-out UML diagram could be when it comes to acting as a blueprint. We found out that spending more time on the UML diagram could have helped save hours of stumbling around. Overall, this phase presented us with a great opportunity to learn the entire process of teamwork and game development.

Adjustments and Modifications to the Initial Design:

Referring to the UML Diagram, we found that some major changes and slight modifications were needed to properly execute our plan. The biggest change was regarding the display of our game to the user. In our UML diagram, we never thought about the packages and methods required to graphically display our game. Thus, we had to create an additional class which implemented the jFrame class from the Swing package. This class was then decided to hold the instantiation of our different classes, and served as our game window.

Another adjustment we had to make was regarding our map class. Previously we stated how we wanted our map class to store map coordinates, the different number of objects we had on the board, and hold functions to generate new objects to place in our game. However, with the new implementation of our game window class, many of these parts were now unneeded. Instead, we converted our map class to only storing 2D arrays which contained where the objects of our game were on the screen. These 2D arrays were split into one containing the player, rewards, punishments, and walls, with the other one holding the player and enemy. These changes allowed us to efficiently implement collision systems inside our player class and saved us from having to separately implement collisions between a variety of different classes.

Moving on, some of the minor adjustments made in our code when compared to the UML diagram would be some of the packages that we utilized. Formerly, we had thought to use java.util.Scanner to receive user input. It was only during the implementation that we realized that java.awt.event.KeyListener would be more suited for handling user input. This change of implementation from java.util.Scanner to java.awt.event.KeyListener meant that an additional class would have to be added that was not previously described in our UML diagram.

In regards to the changes made to our stated use cases, only minor changes were made. In our use case for interaction with enemies, we stated how an exception would be if the user pressed the escape key to quit the game. However, after a group discussion, we decided it would be better to only have movement keys available during our game. Finally, we wrapped up all of our open issues. We decided keyboard shortcuts for anything other than movement was unnecessary since it could trigger unwanted actions. We also decided that there was no reason why the user would have a limited amount of time to select from the main menu. This resulted in us leaving the main menu running until the user either started the game, or closed the application of their own regard.

The Management Process and Division of Roles and Responsibilities:

After having a zoom meeting to discuss the way we want to follow when we start this phase, we decided to do the coding part of Phase 2 together on campus so that no one will be left behind during the whole process. After that, we would finish the report of Phase 2 online using Google Docs.

Among our group members, Milo had some experience with Java so his role was a mentor for the entire process of the group. He gave a lot of ideas when other teammates were lost during the phase. He tried to help other members who were having errors or issues in their codes so that we could quickly move on to the next task. Milo's technical responsibilities (class implementation) were: App, handleInput, Moving, and Player.

The role of Ian (Gihwan) throughout the whole process was a coordinator. He hosted group meetings and helped a lot when we were having trouble with communication. At first, every group member had different levels of understanding of Java during this phase, he made a lot of effort to help the whole group communicate with each other more and to make sure that no one fell behind in progress. Ian's technical responsibilities were: endScreen, gameMap, gameWindow, Punishment, and PunishmentList.

The role of Parmida was a supporter, she gave confidence and courage to every member of the group. When the deadline of Phase 2 was getting closer and everyone was exhausted during the group meetings, the positive behaviors she exhibited helped us get things right on track. She also gave a lot of thoughts and expressed new ideas to improve the overall project and speed up the process. Parmida's technical responsibilities were: regularReward, reward, rewardManager, and specialReward.

The role of UCheng Hong was to search information and to provide ideas. She expressed her ideas and made a lot of suggestions to the group during Phase 2. Also, she gave a lot of

feedback and threw questions at various situations to the group which established good communication among the group members. Alice's responsibilities were: endingCell, Enemy, Entity, wallImage, and wallManager.

List of External Libraries Used and Justification of their Usage:

- Swing: As the game needed to be graphically displayed to the user, Swing was chosen as our graphical interface library due to its extensive collection of packages that would allow us to easily place interactable buttons on the screen, and paint all of our different sprites onto our game window. Moreover, Swing also came with its own timer, which would allow for us to schedule in the spawning of random rewards.

- Keylistener: We used the Keylistener library for receiving the keyboard events (up, down, left, and right keys) from users.

- Util: We used the Random from the java.util.Random package to generate random integers for generating coordinates used in spawning special reward and randomizing where the random reward would spawn from.

- Awt: To create the user interfaces of our game (e.g. setting the background colour of the game window, setting the size of the screen and so on). For example, we used ActionListner and actionPerformed to spawn the special reward with a timer.

- BufferedImage: To store our image files of Entity objects, which included player, enemy, rewards, and walls.

Measures Taken to Enhance the Quality of the Code:

As collaboration and teamwork were critical in ensuring a quality final product, the measures that we took to ensure quality code included: writing comments for ambiguous lines of code, competing javadocs for each function after completion, and working in person to ensure ease of communication for anything vague. Moreover, we tried to follow some of the naming conventions such as camel casing for classes.

Since everyone worked on separate implementations, we tried to declare variables and classes in a way such that they would intuitively make sense. Also we tried to be as critical as possible towards each other's code in order to make improvements.

After implementation, we proofread our work to get rid of redundant lines code. We tried to unsparingly use indentation and line-spacing for easier reading. Lastly, we imported the specific packages we needed instead of using the asterisk to clearly show what each class was doing.

The Biggest Challenges During Phase 2:

The biggest challenge that we faced during this phase was dividing responsibilities among the group members especially because some of the members were totally new to Java and needed more time with understanding the codes. Also, even after we divided the tasks and assigned each group member to do a part of the code, we still needed to fully understand each other's code since these classes were dependent on one another. There was a point where we decided to start over the project from scratch since each of us had completely different names and structures and we were spending too much time understanding each other's work. Therefore, we decided to start over and implement our classes more carefully.

Another challenge was the Git with which we had different levels of knowledge. Some of us struggled with even relatively simple tasks such as cloning, pulling and pushing. However, we tried to solve these issues by arranging in-person meetings as well as zoom meetings so that we could communicate better with one another and help each other out in order to speed up the process.