Phase 2 Report

Group 18: Haruka Mibuchi, Gurinder Bhogal, Asifiwe Julio Patrick

Overall Approach

In order to implement the game our team had to do a lot of research on how to get our code to be displayed on the screen. We began by creating a maven project and pushed it onto GitLab so anyone could get started once they were assigned their tasks. We began by looking over our UML diagram and assessing the things we needed for sure and the things we weren't so sure about. Since most of our doubts were related to displaying the code on a screen and reading input, we began by focusing on a way to make a simple version of the game first. After our research on different ways to make a UI, we decided to use the Java Swing library in order to draw our game onto the screen. We started by drawing a box on an empty screen and passing input to it in order to be able to move it around. This built our confidence and helped us better understand how the game might look. We realized we would need a game frame which would handle the different panels of the game and the state of the game. We then slowly, piece by piece, added improvements to that game such as a map for that box to walk through, rewards on the screen, collision into the map and the rewards, and more. The group met using discord and in person often so we could constantly give updates on our parts of the code, and we could handle any conflicts that came with combining our code.

Modifications

Since we weren't very sure how we were going to handle the UI in phase 1, the UML diagram is completely lacking references to any UI or input handling classes. We decided to add all the UI-related classes to a completely new package that handles the main frame and many panels. Our EntitieManager and GameManager essentially became our GameFrame and GamePanel classes. We noticed that our GamePanel class was essentially doing the same thing we had planned for the EntitiesManager class which was constantly updating entities. The GameManager class had very similar functionality to the GameFrame class since they both handle the logic of the game like starting, pausing, ending, score, and more. We also did not need a StaticEntity class which we were going to use to represent the rewards and traps classes. It was not needed because we decided to implement the map using a text file, which is essentially a grid of different numbers representing different types of StaticEntities and CellTypes (walls, barriers, etc.). We combined the StaticEntite and CellType from our UML diagram to a single Cell class. These were all the things that would stay stationary on

the map whenever they were present. This made it so we could easily check the type of cell the player is walking into and take the appropriate steps afterwards.

Management Process

During the beginning the group did general research on how to implement the game and then we gave everyone tasks using the UML diagram so we could get started. We communicated daily using discord messages and calls and met in person every other day in order to ensure everyone was getting their work done and that we could help each other when we needed help.

Tasks:

- **Gurinder**: MoveableEntities, Collision, Map, Cells, GameFrame, GamePanel, Input Handling, EnemyMovement, Rewards
- **Haruka**: PausePanel, StartPanel, WinPanel, LosePanel, and designing the visuals of the game
- **Julio**: Score, Point, EnemyMovement, Rewards

External Libraries:

We used Java Swing to handle the visuals of our game. We found that this library is relatively simple compared to many other libraries we saw. Since we did not have much experience with making UI's we decided to go with a simpler library which still got the job done.

Quality Assurance:

In order to increase the quality of our code we constantly looked over our code to see where we could make some improvements to the overall design. Some improvements we made were, using method chaining as little as possible to make sure the code had better readability and consistency, used polymorphism and private/protected variables as much as possible to ensure our code was used exactly how we wanted it to be used, and we separated different classes into packages in order to organize the code of the game in a way that makes sense.

Biggest Challenges:

- The biggest challenge we faced during this project was having a group member drop the course during the busiest part of the project. It was unfortunate as they did not let us know in advance, so we had already assigned them tasks, which we expected them to complete. This meant the rest of the group had to work extra hard in order to ensure we are carrying the extra weight.

- Learning Java Swing was also a very big challenge as it was completely new to the entire group. We had to look for many online resources to learn the different ways we can use different tools.

- Enemy movement was also very challenging as it involved a path finding algorithm to implement which none of us have learnt yet. It took multiple days of bugs and testing to make sure the enemies were correctly following the player.

- Adding a restart button at the end of our game was very challenging as we had to make sure to completely clear anything which could carry one to another game. We were experiencing many bugs while we were trying to implement this which resulted in it taking a while

- The GameFrame contains a lot of the classes inside of it since it handles most of the higher-level logic in the game. This meant that initializing objects in an incorrect order could result in an error. We had to be very careful to make sure no objects were conflicting with each other when created.

- None of us have ever used maven or any other build tool. It took us more time than we had initially planned for to add our dependencies as we would keep on getting errors