PROJECT PHASE 2: REPORT

Implementation Approach

As a team, we had an in-person meeting to set up Apache Maven for our project, go through the Phase 2 requirements, and break down the implementation into four steps. We also set goals and timelines for the completion of each step. What seemed like a daunting task was made significantly easier when breaking down the implementation into simpler/smaller tasks. All subsequent meetings were done remotely and held after the completion of each task. In these meetings, we would go through the code to ensure everyone was up to date and able to understand the new implementations. Additionally, we would give feedback and make any necessary revisions. Upon the completion of the final step, we had a final team meeting to test the game, clean up the code, as well as write JavaDocs and the report.

Modifications to Initial Design

We made the following modifications to our initial design:

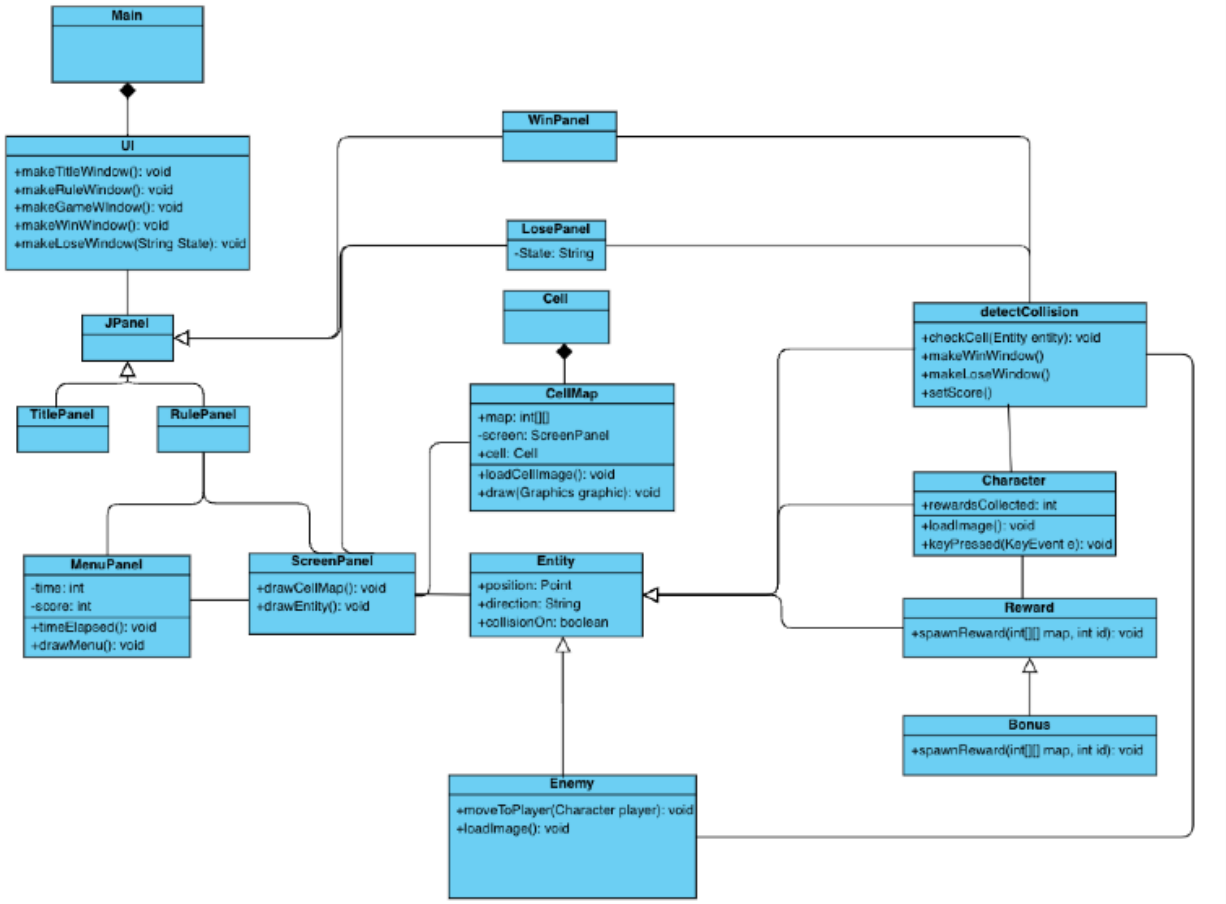
- Updates to our UML diagram
- Distribution of punishments between enemies and obstacles

UML:

When we started implementing the game, we noticed that the Phase 1 UML diagram was inadequate in that it did not include game logic such as collision and win/lose conditions. Thus, we concluded that the entire initial design had to be revised. The UML diagram below follows the structure of our implementation and is a revised version that abstracts away the implementation details and just focuses on the game logic.

The important additions and modifications we made compared to our initial design include:

- Main class only has access to UI class
- UI class generates panels for each state of the game
- When exiting the RulePanel, the MenuPanel and ScreenPanel are created and populated with entities (character, enemies, and rewards)
- detectCollision class includes collision logic (enemies, rewards, barriers) and logic for determining if the game is won/lost; detectCollision has access to the win and lose panel
- Win and lose panels connect back to the screen panel, allowing the user to replay the game



Enemies/Obstacles:

After reviewing our requirements and use cases from Phase 1, we decided to alter the impact on the main character when walking into punishment cells versus enemies. Originally, walking into enemies would decrease the player's score by a certain number of points (depending on the type of enemy) and walking into lava would result in the game being over. We later changed this so that walking into punishment cells decreased points and walking into enemies resulted in the game being over. After a few run-throughs of playing our game, we decided that this modification would make the game more challenging and fun for the user. In addition to this change, we added more punishment cells on the board (lava pits and spider webs) and decreased the number of moving enemies.

Project Management

Management Process:

We wanted to ensure we were communicating effectively at every step of the implementation of our game. To do this, we held scrum meetings 3-4 times per week. In these stand-ups, we discussed the work we had done, brainstormed ideas, and offered suggestions to improve the quality of our code. We also communicated in an ad hoc manner via a group chat to discuss bugs and provide updates. Throughout this phase, we maintained a backlog of tasks that included to whom they were assigned

and when they were to be completed. Keeping this backlog ensured we remained on track with our deadlines.

Division of Roles:

In order to divide roles and responsibilities, we first split Phase 2 into four logical steps. These steps were:

1. Create the UI (map and scoreboard) and develop the main character's movement
2. Build collision detection logic when walking into barriers and punishments
3. Display rewards and bonuses and develop logic when the main character walks into them
4. Display moving enemies to track the main character if it comes within a certain distance

We then delegated these steps among the four of us. Preet completed step 1, Jimmy step 2, Jeffrey step 3, and Jessy step 4. While we divided the roles in this manner, each step involved a great degree of collaboration so that we could help one another. This collaboration was done as described above via scrum meetings and ad hoc messaging.

External Libraries/Packages Used

Below is a list of the external libraries and packages we used for the implementation of our game and our rationale for choosing them:

Library/Package	Purpose
Java.awt	• Aid in the creation of our GUI • Primarily used for Event handlers and listeners to read user input and render the system accordingly • Also used for Container positioning
Javax.swing	• Provide extra GUI capabilities in addition to AWT to allow us to make a more robust and visually appealing interface • Used to create JFrames, JPanels, and JButtons • Timer used for character movement delay, on-screen timer, and time between bonus appearances
Java.text	• Used for generating and formatting text to display on screen
Java.io.File	• Used for file/directory paths
Javax.sound.sampled	• Utilized for allowing the playback of audio during gameplay • AudioSystem class used for storing audio clip • Clip interface used for continuous loop of background music

Measures to Enhance the Quality of Code

In order to enhance the quality of the code, we deleted repetitive code and created methods that could be called and utilized in various classes.

Initially, the ScreenPanel and UI classes were instantiated multiple times throughout different classes. This caused multiple problems because there was only one screen being shown. Thus, we used the Singleton design pattern to ensure that only one instance of the ScreenPanel and UI were being created. This ensured that all classes instantiating the ScreenPanel and UI shared the same attributes so that time and score would not conflict in different classes.

In addition, we also practiced modular design by separating independent classes and methods into different folders and files. For example, in the beginning, we did not have the UI class so the makeTitlePanel(), makeRulePanel(), and makeGamePanel() methods were all in separate classes that had no connection to each other. Therefore, we created a UI class to store all the panel creations into a single file which allowed for cleaner and more coherent code.

Challenges Faced

This section will discuss the biggest challenges faced separated into each step in the project as outlined in "Management Process and Division of Roles."

Step 1:

The primary challenge faced in this step was image rendering. As a group, we are new to UI design, and so learning how to load and draw images was a learning curb. More specifically, challenges were faced when mapping out the cells and drawing images into each of them. Then some challenges were faced with rendering the image of the main character to lie on top of the cells' images, and not override them. These challenges were overcome with a lot of debugging and researching Java's GUI packages and their methods through resources such as documentation and discussion threads.

Step 2:

The biggest challenge with the detectCollision logic was determining which cell the character was in and if that cell contained a barrier or punishment. The initial algorithm read the character's position as x and y coordinates with the top-left corner being (0,0). However, because we used a 2D array to map each entity to a cell, reading the character's position could not be done with x and y coordinates, but rather using the [row][column] notation. For example, (0,1) in terms of x and y coordinates would just be one unit down from the initial (0,0) position, however, [0][1] in terms of row and column would be the 0^{th} row and 1^{st} column. We overcame this challenge by realizing that we had to abandon the conventional x and y coordinates, and instead use the [row][column] notation.

In addition to this, the detectCollision logic did not handle the moving enemies as it was initially created to focus on the elements that were stationary such as rocks and lava. In order to handle the

collision detection for enemies, we had to create a new method that kept track of the enemy's position and check if any obstacle was present before moving.

Step 3:

The main challenge faced in this section was implementing our UML diagram into code. Mainly there were issues in keeping track of the score between the Character, MenuPanel, and ScreenPanel classes. Additionally, when implementing the disappearing behaviour of the bonuses, we had to keep track of the time between detectCollision, MenuPanel, and ScreenPanel. After encountering these difficulties, the problems with our original UML diagram were noticeable, and as such, we decided to revise it.

A minor challenge we faced in this section was working with code from previous steps and maintaining a consistent and clean coding style. This was only a minor issue as we held meetings after every step. By keeping everyone involved and up to date, it was a lot easier to understand, use, and add to pre-existing code. As for our coding style, this step would have been trivial by hard-coding rewards and bonuses into CellMap. Instead, we decided to follow the principles of object-oriented programming to maintain a clear, clean, and adaptable coding style.

Step 4:

The primary challenge faced during the implementation of the Enemy class was figuring out where other enemies were located. This is because the enemies were not stationary and therefore could not be hardcoded as an obstacle at a specific coordinate. To counteract this, we had to keep track of all enemy (object) movements and constantly update their coordinates in order to check if the next available position contains an enemy or not before moving. One way we solved this problem was by using an ArrayList that holds the object's x and y coordinates and is constantly updated with the new position when the enemy moves. Later we check the new position the enemy wants to move to with the ArrayList to check if there is no enemy present at that location.