CMPT 276: Project

Phase 2: Implementation
Group 24

Armin Hatami
Jacob Yee
Reid Lockhart
Ryan Marwaha

March 11th, 2025

Management Process & Division of roles and responsibilities
We initially all met up together as a group at the start of Phase 2 to whiteboard out the structure of the code, determine what tasks actually had to be done, and who would do what task.

To manage communications, a Discord server was set up. The overall distribution of work was relatively even, and spillover in responsibilities was frequent. The table below is what we originally decided on during our first meeting, although in practice, everyone worked on a little bit of everything when needed in addition to what was decided.

Person	**Main Tasks**
Jacob Yee	Game Logic Main Loop Game Rendering General Bug fixes Report
Armin Hatami	Enemy Pathfinding Algorithm Maze Generation Art and Level Creation Bug fixes Report
Reid Lockhart	Art and Level Creation Win and loss conditions Game Logic Bug fixes Report Images and tutorial
Ryan Marwaha	Initial Planning Javadocs Java Swing Integration Menu and UI Bug fixes Report

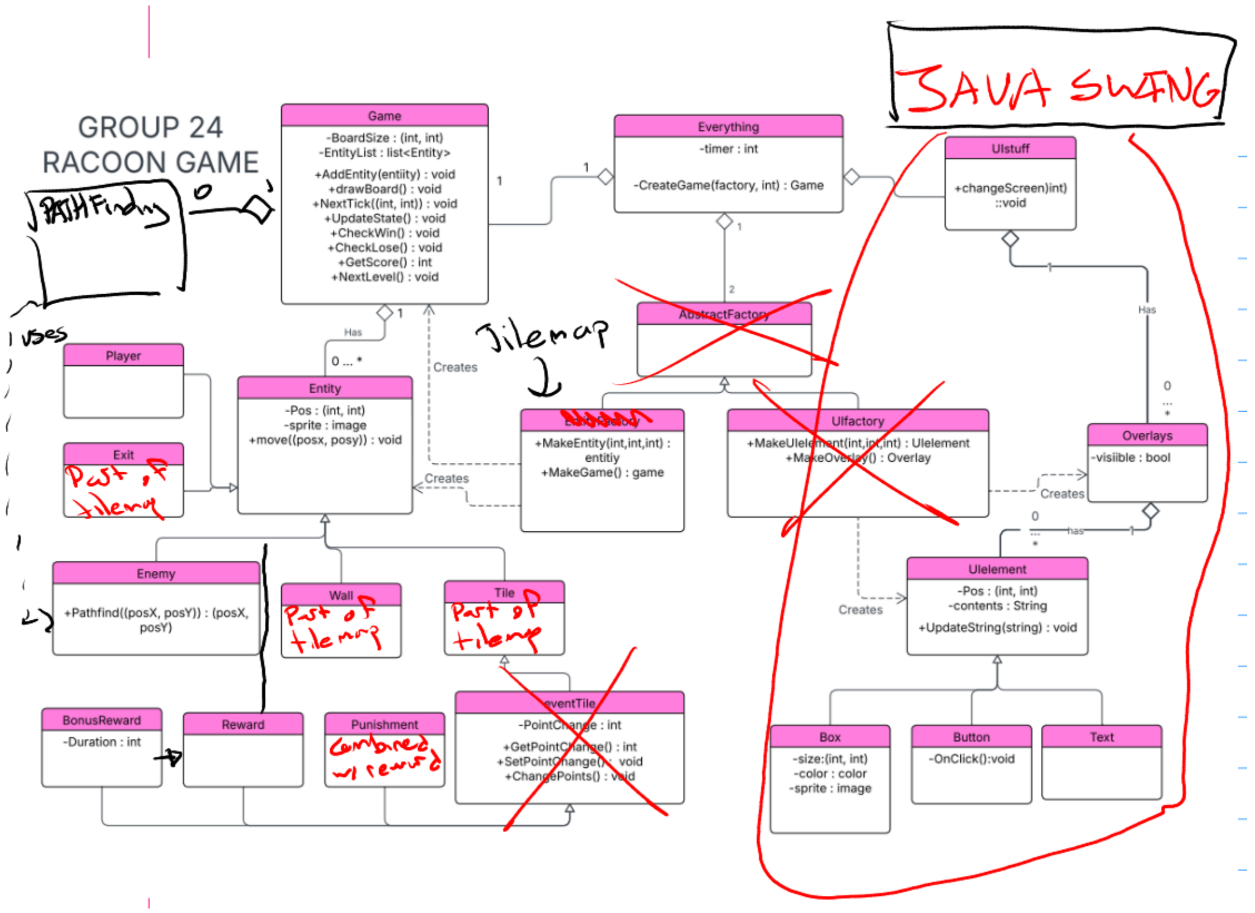
Overall Approach
We divided the project into two parts to separate the core parts of the game and simplify the overall process. Part A consisted of creating the core parts of a maze game, including the engine and world generation. Then in Part B, we added the intricacies and features that transformed a simple maze game into something more dynamic and challenging. This included adding NPCs, their pathing functionalities, and a scoring system. The halfway deadline was determined to be when UML templates were implemented.

Project Timeline

1. Part 0 - Setting Up UML Templates (Halfway deadline)
2. Part A - Basic Maze Explorer Game
    a. Create the main loop to control the engine/tick rate
    b. Implementing a maze-generating algorithm with start/end
    c. Displaying the main character on the board
    d. Displaying objects
    e. Displaying tiles
    f. Ability to control the main character
3. Part B - Converting into Treasure Hunting Game
    a. Creating other characters
    b. Creating an NPC movement system
    c. Creating scoring system
    d. Creating a winning/losing system

Adjustments and Modifications

The most significant changes from our initial plan are the removal of all of the classes related to UIElements and Tiles. For the UIElements, we realized that it would be both easier and technically better to use the preexisting Java Swing Library to manage graphics than create our own solutions. We will elaborate further on this reasoning in the external library section down below. As to the tiles, we simply realized that there was little distinction made between tiles and entities, so many of the objects that we had previously classified as tiles (Rewards, Punishments, etc) were changed to simply extend Entity instead. Other minor changes to the



classes include combining the Reward class and the Punishment class - a punishment is just a negative reward after all - and adding needed functions that we had not previously thought of.

The updated UML diagram of the project with changes highlighted.

For map design, we initially wanted to create a set of levels for the player to progress through. However, we determined that designing distinctive levels would require significant manpower. Instead, we opted to modify a maze generation algorithm to produce mazes with loops while ensuring a uniform distribution of guards and rewards.

External Libraries
First of all, as mentioned above, Java Swing was used for the GUI elements of this project. It was chosen for various reasons - most prominently the ease of implementation - but also because of its default inclusion with Java. We knew since the start that we weren't going to be doing anything intensely complicated with the UI or the game, so our main focus when deciding was simplicity and the number of external teaching resources available. Java Swing fit our criteria, so we picked it quickly to come to a decision rather than being paralyzed with choice. Other ancillary libraries that we used were more mundane (timer, random) and were picked because they were easy to implement.

Measures taken to increase code quality
Right from the start we knew the general shape of the functions and classes that we had to create because of the UML diagram from phase 1. This was a great help with keeping functions isolated and preventing spillover of unneeded information between functions. In the times that the plan was changed, we informed our group members of the change and how it would affect what they needed to do, if anything. This sort of collaboration also forced us to reduce function coupling by making us be clear with what information we needed from functions that other people were writing.

For complex mechanisms such as the Bread-first search method in the pathfinder class used to control the NPC movement, we implemented unit tests to quickly identify and fix bugs.

Biggest Challenges
The biggest challenge was balancing cohesion, complexity, and functionality. Our final design was a simplified version of the initial plan after we realized that we were simply adding complexity to our project for complexity's sake. This increased cohesion greatly but also significantly reduced coupling. Getting everything to work was a bit of a struggle without a clear idea of the final product.

The second biggest challenge was managing the git history. Although multiple branches were used and we had constant communication, there were a couple of incidents with merge conflicts that ended with rollbacks and tears. We were learning how to use git while using git, and it came with some incredibly painful growing pains. It ended up working out though, and git itself was helpful for undoing whatever we broke using git.

Another more general challenge that we encountered was the group's lack of experience with Java. This led to difficulties with implementing specific features, most noticeably with anything related to Java Swing.

Figuring out what to put in the pom.xml took way longer than expected, and our inexperience ended up making some simple tasks take ten times longer than they should have.

Overall this phase was a great learning experience for collaborating on software development in an iterative manner.