Dylan Young, Kaung Si Thu, Maisha Supritee Chowdhury, Sajandeep Toor

CMPT 276 Phase 2 - Group 19

Game Name : Maze of the Dead
Implementation:

Our team split off individually to do more research on building 2D Games in Java. From there, we started off by creating directories, files and stubbed methods following our UML structure. Next, we added Maven to the project for build automation. We tackled major classes such as character, cells, and maze, before moving on to smaller classes such as rewards, traps, leaderboard, etc.

We utilized creational design patterns such as singletons for the Maze, Leaderboard and GameController classes.

We also created a utilities package to store all constants used in the project, as well as common functions which might be called by different classes. Our method of communication was our Discord group where we continuously brainstormed, tested each other's codes and made suggestions for improvements.

Once we had the backbone of the game code done, we moved on to the GUI portion. First we had to research and learn more about the external library Java Swing which we chose to use for the game. We took on the task of creating different game screens according to various use case scenarios, such as TitleScreen, GameWonScreen, LeaderboardScreen, etc. Then we handled the main GamePlayScreen where the actual game would take place. Next we created images for the player character, enemies, rewards, traps, etc and developed sprite sheets. Additionally, the creation of a components package helped us ensure all of our screens had the same style of labels and buttons, as well as the same functions for the same elements used.

Finally, we did some bug fixes for all of the code, checked that style and indentation is maintained, and appropriate javadoc comments have been added.

Adjustments:

Initially our UML structure had one massive package for the modules, but we decided to break it down to make it easier to build as well as clearer to navigate through. Another reason was if we needed to have protected methods, this would allow us to have methods only visible by the package. For example, we have a utilities package with classes Constants, Position, Movement and Functions. This wasn't in our initial UML mockup, but we realized it will make everything more organized during our coding process. Methods were also changed. For some the visibility changed. We also created a lot of new methods such as helper private methods for example isPathHelper() in the Maze class. A lot of these things we just didn't foresee during the planning phase and realized we needed during the implementation stage. However, the classes themselves remained largely the same as the planning stage.

We made the most changes in the UI portion though, adding multiple packages such as GameFonts, components, and images, as well as many classes such as GamePlayScreen, GameWonScreen, UIUtils, SpriteIcons, etc. This helped us tackle every specific UI component such as different screens, images for all game components, background images, etc in a divide and conquer method. We tried to test as much as possible to ensure things were working correctly during the development phase.

Management: For the division of coding responsibilities, it was based on their experience first and foremost, then their interest and availability. Through our Discord group, we each volunteered for roles we were most confident in and divided the coding in this process.

- Dylan : Cell, Maze, GameFonts, GameUI, GamePlayScreen, InstructionScreen, TitleScreen, KeyboardListener, UIUtils, UIConstants, SpriteIcons, as well as some bug fixes in classes
- Kaung Si Thu: Character, Reward, GameOverScreen, NewHighScoreScreen, LeaderboardScreen, UI components and PanelWithBackgroundImage
- Maisha : PlayerScore, Leaderboard, PauseScreen, GameWonScreen, and Report
- Sajan : Maven, GameController, Images, Utilities, and added to Cell, Maze, Reward and BonusReward

Branches were used by each member and once a feature was implemented or a bug was fixed the individual would submit a merge request. From there we discussed changes in the implementation, issues with the code quality (formatting, variable names, etc.), and once each member was happy with the proposed changes, it would be approved and merged to the master branch.

External Libraries: The only external library we used for this project is Java Swing, for doing the graphics and user interface of the game. We decided to choose Java Swing instead of other libraries, because it's the easiest to learn and develop for people who haven't had much experience with UI before.

Challenges:

- Java - Since Java is a new programming language for some of us, the syntax, although similar to C++, was unfamiliar and required some getting used to.
- Maven - This was very new territory for most of us, so it was a bit challenging to understand how to build and manage the project through this.
- Git - Almost every member had issues relating to git, it's a new platform for some of us and was pretty difficult to navigate through. Some of us had to delete the project or branches multiple times, figuring out how to squash commits, rebase, rename commits, rename branches, etc. was confusing too.
- Javadoc - Javadoc comment style was very new to a lot of us, so it took us some trial and error to get the hang of it.
- Game Controller is one big class, which has threads for game loop, enemy loop and time loop. So, it was hard to keep track of all of the little things to be done in this class and

had to be refactored multiple times. There were a lot of methods so one fix was using Java regions to separate the code into regions which could be easily closed and opened in most modern IDEs.
- There was an issue with pausing the game, as it took too long to do so after pressing the ESC button.
- Leaderboard - This class was very complicated to code and produced multiple complications, and had to be refactored. The addPlayerScore method went under a lot of changes and fixes. The readFromFile was also very confusing to write and using the string split method proved to be the solution.
- User Input complicacies such as having to tab in and out of the game to be able to enter any user input - user input was working, but the computer was not focusing on the program.
- The UI was very difficult to learn, create and implement in general. Even after multiple attempts with many different implementations, we were unable to scale the windows of our screens. Adding background images to the screens proved to be a bit difficult as well, and took a couple of tries to figure out.
- Enemy movement implementation was somewhat difficult, there were a lot of ways to do it, using different search algorithms. One of the simplest methods would be by using A* search and calculate the absolute distance from the player using pythagorean theorem and looking at all possible moves the zombie could make and picking the move with the lowest absolute distance using a heap. However as there were often a lot of deadends in the maze, this was not an effective solution as the zombie would end up getting stuck in the maze.
- Generation of entities in the maze was another difficulty. Entities are rewards, bonus rewards, traps and enemies. As the maze was generated randomly, the entities had to be generated randomly as well. One of the issues we noticed was sometimes the game would be unwinnable as rewards were not accessible as traps were in front of them or traps would be in front of the exit or start of the maze.
    - To overcome this difficulty, we couldn't choose locations for rewards to be purely random, instead we had to validate that the maze is solvable if a trap is placed here or a reward is placed here.
    - This is where the isPath method comes in, it checks if there is a path from the player to the reward, i.e. could the player reach the reward without going through walls, traps, etc, if there is the a path, then the reward can be generated there, if there isn't then we must generate a new random location for it.
    - Instead of calculating absolute distance, we repurposed the isPath method from maze to find the number of steps required for the zombie to reach the player, this would ignore cases where the zombie ends up dead ends and would ensure progress to the player. As the isPath method is a recursive BFS search, this is

more computationally expensive; however, we optimized it enough for it to work without any slow downs or lag.

Quality of Code

The quality of the code was maintained by regular refactoring. One issue we had with quality was code formatting, we decided to use the Oracle's Java Code conventions for the formatting of the code, this was easy to access in many text editors such as Visual Studio Code or Intellij. One of the issues we had with the Oracle's conventions were how switch statements were handled, as a team we decided to format it so everything inside the switch block was indented as this made it more readable. One way we made sure our code was consistent was having everyone approve during merge requests, here we could point out issues with variable names, method names, code formatting, etc. We followed the DRY method of Don't Repeat Yourself, if we noticed code was repeated multiple times we would refactor it to a clearer solution.