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

CMPT276 Phase 4 - Group 19

Game Name : Maze of the Dead

The Game :

This is a 2D post apocalyptic dungeon crawler, where the player's goal is to survive the zombie apocalypse. They need medical supplies and weapons to survive. The player navigates through a maze trying to collect these supplies, while avoiding zombie attacks and traps around the maze. The player must escape from the maze alive. Our overall theme as well as the picture we had in mind for this game, hasn't changed from our original plan. Similarly, for UI we have not deviated from our original design.

The UML structure for our game has changed however, as there were several flaws in our original design. 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. This would also enable us to have methods only visible by the package if we needed to have any protected methods. Such as the utilities package, which wasn't present in our initial UML mockup, containing the classes Constants, Position, Movement and Functions, helped make our coding much more organized.

There were also a lot of changes in the UI portion, compared to our initial plan. We created multiple packages like components, Frames, Screens, etc with a lot of different classes like Buttons, Elements, GameFrame, GamePlayScreen, etc to handle each specific item related to the total UI for the game.

Similarly, we distributed the code for GameController into multiple classes like Entities, EntitiesGenerator, Time, etc. In our first UML structure, we were still unaware of how much this one class would have to handle, which we later on discovered during our coding process. Dividing that one huge class into multiple classes helped with the clarity of the code, made it easier to navigate, and made it easier to test. Moreover, it removed a "bad smell" in our code. Each class was cohesive, only having methods and data related to the class itself and was a better practice.

Another change we made was adding an enemy class, before enemies were in the CharacterModel class, however it made sense to move the enemy movement logic to the enemy class itself as previously enemy movement logic was in GameController. It made more sense than moving it to GameLogic since that was more general and responded to the player's input. It also made it more clear in the code that it was an Enemy. Enemy still extended the CharacterModel.

Besides these, we didn't make too many changes from our original UML design. For some classes new private, public methods or a constructor with different parameters were added but overall, we stayed faithful to our original UML design.

We did not have a lot of changes for classes such as Leaderboard and PlayerScore. But we had quite a challenge to fix the error with the leaderboard because the leaderboard is only

supposed to store the top five player scores in order. There were small changes in many classes where we had to remove the unused methods or variables.

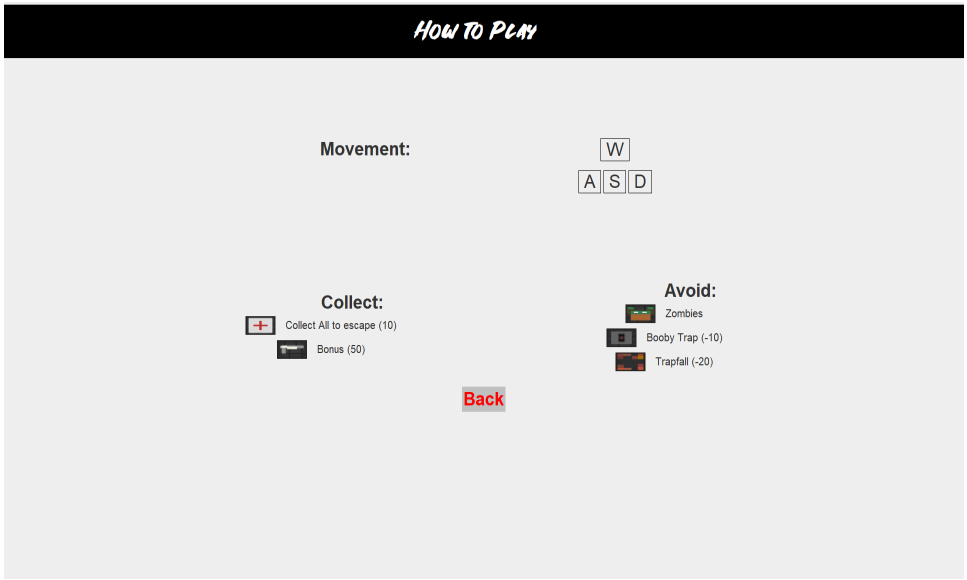
Some important lessons we learned included spending more time on planning. This would prevent us from having to make changes when we are implementing. We wouldn't need to refactor large classes or methods and would save us time in the long run. Also it would prevent issues from building up and technical debt that would require refactoring at a later date. It would also prevent others from writing new methods for classes since they don't know if a certain method exists since it wasn't in the original UML design and once changes were merged, there would be 2 implementations of the same method. This happened a bit in our GameController class where there was a way to get an entity for a position and another member added a method to get an entity for an x y coordinate, they did basically the same thing but different parameters and was redundant and unnecessary to have both.

How to Play:

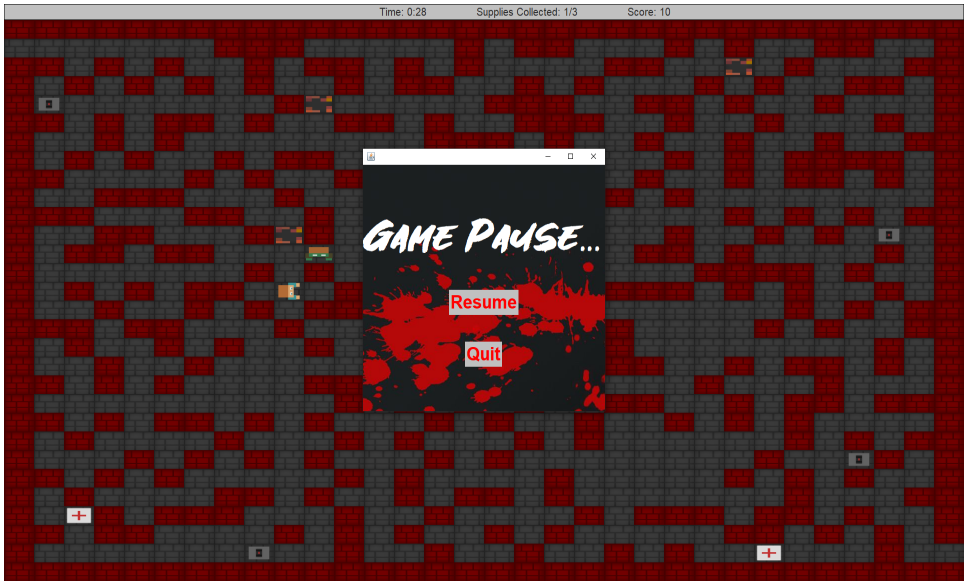
To play the game the player must create a JAR using maven and run it, this is detailed in our README.md. Once the game has started, the player is presented with a title screen showing the options “Start”, “Instructions”, “Exit Game”.

Start and Exit Game simply starts/closes the game respectively.

Instructions takes the player to the Instructions screen, where they can learn information about how to make their character move on the screen, what the rewards look like as well as the amount of points they give, and what they need to avoid such as zombies and traps. The player moves with the “W”, “A”, “S”, “D” keys for up, left, down, right respectively.



Once the player has started the game, they can pause the game by pressing the “Esc” button. This will take them to the Pause Screen, from where they can either resume playing by clicking the resume button or quit the game, by clicking the quit button which will take them back to the Title Screen. The pause screen stops the timer and gameplay, the player cannot interact with the gameplay itself anymore until hitting resume.



Interactions between entities:

- **Definition of reachability:** A position is reachable if there exists a path without going through walls or traps between the player and the position.
- The Zombie chases the player, finding the shortest path between its current location and the player’s location using the Breadth First Search algorithm. When the zombie goes in the same cell as the player, the player dies and loses.

- When the player collects a reward or steps on a trap, the player's score is updated and if the player's score goes below zero, the player loses, if not the trap or reward is removed from the maze and the player continues as normal. Rewards are all spawned at random "reachable" locations from the player.
- Bonus rewards have a 2% chance of appearing at any second, they can last from 10 seconds to 20 seconds, this value is random and appears at a random "reachable" location in the maze.

Game Features:

Leaderboard:

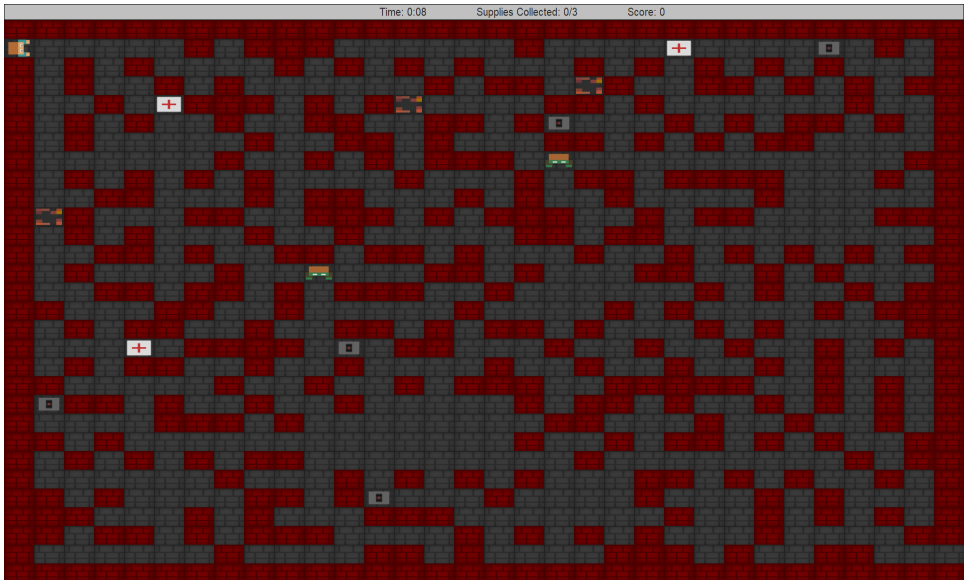
The leaderboard adds competition to the game. This allows players to try to fight for the top spot on leaderboard. Instead of rushing to collect all the rewards and quickly trying to get out of the maze, the player has to make a decision whether to stay in the maze and to try to get additional bonus rewards to improve their score.

Randomly Generated Mazes:

Every time the player plays a new game, a new maze will be randomly generated. These mazes are generated based on Prim's algorithm with a few additional features. One of the features of the maze are the rooms. These rooms allow the player to move more freely around the maze and creates a few additional paths to help the player avoid the enemies. The rooms are also randomly generated but each room cannot be generated from the same top left corner. Entities, such as zombies, rewards, and traps are also placed at random locations making each maze even more unique. We've ensured there's never a case where the maze is impossible.

The reason for randomly generated mazes is to improve the replayability of the game. This prevents the player from memorizing the game and how to easily win. This makes the game more fun as the player is presented with a brand new maze each time.

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



Examples of mazes not being the same.