CMPT 276 D100

Phase 2 Report

The features in our design phase were:

- Main Character
- Board
- HUD
- Map(s)
- Guards
- Elites
- Lasers
- Shattered Glass
- Turrets
- Money
- Gold

We then added new features that were not in our design phase. We also adjusted and changed some features of the plan that did not fit our design direction.

Our overall approach to implementing the game was to first implement our basic features from the design phase. At this stage, the goal was to make a simple maze that the player could walk through, with no obstacles or rewards. During this beginning phase, we also tried to think of more ideas, as well critique our old ones, and figure out what could be implemented and what would be too difficult to work on. Establishing communication was also part of this beginning phase, as discussion and collaboration were very important in completing this project.

Throughout the weeks, tasks were split up so that each member could effectively work on their own. Each person communicated what they were working on and what they planned to do after finishing each feature. Enumerations, comments and javadocs were used to keep the code readable, and the names of classes and methods were written to state what their function was. By the end of the second week, we created a maze game that met the requirements of the project, although there were some minor errors in a few classes and methods.

During the final few days, as well as the extension period, we mainly focused on refining the game and ironing out bugs that hindered functionality. Optimised rendering and other features were added to keep the game running smoothly. Additionally, we added different settings that could make the game more entertaining, such as different difficulties, adjusting volume settings, and different kinds of enemies.

An overall roadmap of the implemented features looks like this:

- Review phase one design and establish communication
- Come up with different ideas and scrap unrealistic ones
- Create a game loop with a thread to run it
- Create a window and interactive panel (JFrame and JPanel)
- Display simple character
- Move character
- Display background tiles
- Collision detection with background tiles
- Create and display objects (Stationary entities)
- Collision detection and interaction with objects
- Sound effects and music
- Score system and timer
- Displaying score system and timer
- Basic UI for winning and losing
- Create and display enemies
- Collision detection and interaction with enemies
- Basic enemy movement
- Improve UI by adding title screen and readjusting elements
- Make enemy follow player via the A* Search algorithm
- Create bonus reward
- Set random time and location for bonus reward to spawn
- Custom sprites
- Options in pause screen
- Improved performance and rendering

The adjustments/modifications we made from the initial design of the project were

- Different Difficulties: we felt adding difficulty levels to our game would improve on our game's design.
- Added helper classes for game logic (AssetSetter, CollisionChecker, KeyHandler, TileManager, Pathfinder : Helper classes make the game's logic more encapsulated and more readable.
- Added helper classes for audio and image input/output
- Added custom sprites for tiles, objects, and entities : We added custom sprites so the game objects will look like what we describe in our plan.
- Enemies were revised to Guards and Guard dogs : we made this change to better represent the theme of the game as a bank robber trying to collect money.
- Changed bonus rewards to diamonds and made gold and cash interchangeable : Stylistic change
- Scrapped shattered glass, turrets (for now) : we did not have time to implement these and they are not required in the specifications of the project.

During the implementation of our game, we consistently revised the components of our system to ensure they met our current needs. As we evaluated our system, we divided some of the functionality of our GamePanel class into several helper classes to improve the encapsulation of our game logic and make the code more readable. These helper classes are used to handle asset setting, collision, user input, and enemy pathfinding. Additional classes were required to assign sprites and sound effects to objects. Custom sprites were made in Piskel for UI components, tiles, stationary objects, enemies, and the main character, while the sound effects were downloaded from the internet. Some stylistic changes we made were to change enemies to guards and guard dogs, which was more appropriate for the bank heist theme of our game. Gold was initially supposed to act as the bonus reward, but it is now interchangeable with money, while diamonds act as the bonus reward. Due to time constraints, we were not able to implement the extra functionality of transparent windows, shattered glass, switch-activated doors, or state changes between pacing and chasing for enemies. Finally, while map loading and difficulty settings are functional, they are only used in the easy difficulty because we devoted more time in fixing other bugs, rather than making more game levels.

The management process of this phase was decided throughout the second week, as the group slowly became familiar with their roles naturally in the first week. Jaime focused on implementation and creating the game logic, ensuring parts moved well and functioned together, as well as fixing any bugs that were easy to spot. George also focused on writing code, enhancing readability and performance, and catching any bugs that were initially missed. Brayden primarily worked on sprites and art, continuously enhancing the readability of the UI, making the screen functions intuitive, and integrating the sprites into the game menus. Linden primarily worked on the report of the project. Group meetings were scheduled in advance for our members to discuss their responsibilities and tasks.

We did not use any external libraries for this project. Our GUI was made with Java's Abstract Window Toolkit and Java Swing. Map data was read and processed by Java Input and Output. The Java Util package was necessary to create certain game elements, such as the spawn and despawn timers. ArrayList was also used to store entities and nodes, for drawing and pathfinding respectively. Other packages helped implement sounds and image processing.

The package we mainly used for the GUI is Java's Abstract Window Toolkit. We chose this package because it supports graphics and images and isn't hard to work with. Other packages we used were:

- ArrayList
- Random
- ImageIO
- Swing
- Sound

We chose to use the ArrayList package because we needed a data structure to store elements in and ArrayList is easy to work with since it is dynamic, has no size limit, and we could add/remove elements anytime. In our game, ArrayLists are used to store Entities that need to be drawn to the board, as well as Nodes, which are representations of cells on our board that are used for pathfinding. We chose to use the Random package because we needed to randomise spawn times and locations of bonus reward objects, as well as randomise the sprite used for base rewards. We chose to use the ImageIO package because we needed to read and write the images for our game. We chose to use the Swing package for parts of the GUI because it has more flexibility and customization than AWT. We chose to use the Sound package because we needed a way to play the sound files for our game.

The measures we took to enhance the quality of our code were:

- Writing modular code
- Utilising enums instead of ints
- Using meaningful and descriptive names for variables, functions and classes
- Documenting the code using comments
- Reviewing the code
- Testing the code
- Using consistent formatting of the code

Modular code helped because it made code easier to read and test, and increased the maintainability. It was also safer and decreased the change of the code conflicting with itself. Meaningful and descriptive names for variables, functions, and classes made the code easier to read and edit for other group members. Documenting the code using comments helped us understand each other's code. Reviewing the code helped us spot any errors and ensure the quality of the game stayed high. Testing the code also helped us find and correct errors. Lastly, using consistent formatting of the code improved the quality by making it more readable and understandable.

The biggest challenges we faced during this phase were:

- Implementing the pathfinder algorithm (A* Search) alongside collision detection
- Optimising rendering process
- Enhancing code readability
- Debugging
- Setting up UI
- Coordination with group members
- Communication with group members
- Time management
- Setting up the game to be full screen

Implementing collision detection between the player and non-moving objects was quite easy, as we just had to make rectangles that would interact with the player if they intersect each other. However, moving enemies, along with collision detection was much harder to create, as we needed to learn about pathfinding, as well as adjust the path if there were tiles in the way. The A* Search algorithm is a popular path search algorithm that would make the enemy find the player in the shortest possible path. The implementation of this was quite hard and used the Node and Pathfinder classes, but we eventually created a pathfinding algorithm.

Setting up the UI took a very long time, not necessarily due to difficulty, but because of how tedious it was to create. The sprites and images had to be placed in various spots that required a lot of trial and error, as well as calculations of width and height by pixels.

Debugging and enhancing code readability took a lot of time too. With so many features, there were bound to be errors in the code, as well as confusion with the functions with classes and methods. Fixing, renaming, and enumerating these classes, methods, and variables took as much, if not more time, than the actual implementation.

Coordination and communication with the group was solved by the creation of a discord server, where we could message each other about updates regarding the project. Additionally, we could easily schedule meetings and hold voice calls whenever we were able to work at the same time.