CMPT276 Group Project: Phase Two

TAs vs Students

Thomas Kingsman, 301424023
Owen Liu, 301369997
Gurnoor Chahal, 301427771
Yoobin Nam, 301538106

Overall approach to game implementation:
Our approach to implementing the game started by being able to draw objects on a standalone game panel. This was done using Java's Java Swing library. Once we could render objects, the rest of the game was all about drawing in objects and making them move, spawn, and despawn as needed. We decided to approach it as an OOD process allowing for a base class to be able to create multiple related objects and add on any other functionality if needed. This led to a fairly procedural build, where each feature was either a new addition or an improvement to a past class.
We held fairly informal daily scrums to discuss progress and delegate upcoming tasks. This worked well for our small team doing fairly freehand development. It also allowed us to ensure that our resources were always dedicated to the most pressing tasks at hand and prioritize foundational tasks first. Our style of planning and delegation of tasks ensured that our project was completed in a timely manner and allowed for flexibility for members to work on their own time.

Adjustments to initial design:
Compared to our initial plan, we had to make several changes to our design.
One of the biggest changes came with our choice of graphics library, Java Swing. This library required that we add classes such as our gamePanel class on which the game itself is displayed. This was not accounted for in our initial design, as we hadn't committed to a graphics library yet.
Another notable change was the moving of MobileEnemy to be a subclass of Entity rather than being a subclass of a "Character" class. Our original vision was for both mobile units to inherit movement abilities from a superclass. However, we found that the two ended up being distinct enough that it made more sense to refactor our design. For example, player movement with the Controller class was done completely differently from the MobileEnemy's moveToPlayer() method and also relied on different checks than the player's movement.
Finally, we removed several unnecessary classes. These include the PauseMenu class and the dedicated HUD class, the latter of which was worked directly into the gamePanel class.
In addition, we made a few changes to the user interface. We added a description about how to play the game into the start screen and changed the layout of the actual game. For example, our initial UI mockup had a different maze design and wall blocks. We decided to change this design, as we wanted our mobile enemies to always follow the player. The initial design would be more difficult to play with, as it contained tight spaces and would allow the player to easily get caught.

There were several changes to the overall design of the game as well.
Firstly, rather than using a coarse grid, we used a finer pixel-based system for player and enemy spawns and movement. We felt that this not only technically satisfied the requirements for a general grid structure, but also went above and beyond by allowing finer movements. Going for a more free movement-based game instead of a fixed grid system came with challenges such as different hit detection and collision between objects and enemies, but we were easily able to implement it using a hit bot design. Our game logic also reflects this enhanced level of complication, while also maintaining a clean interface and high-quality code.
We also added several features not mentioned in the Phase 1 outline. For example, we added background audio to enhance the player's experience. Audio is an important part of gaming for many users, and we wanted to reflect that in our design. This not only added difficulty to our project but also made the final design more appealing to our target users as sound sets the tone of the game and environment. Our changes were generally aimed at improving the player experience without compromising on the complexity expected of a project at our level. Ultimately, they also gave us a chance to express our individual strengths as developing programmers in a creative environment.

Management and division of labour:
No one group member took on a strict leadership role; instead, we all worked together to complete our game. We tacitly agreed that the best way to reach our goal of a completed game was to start with the foundational tasks, then build up from there. We also mutually agreed that daily informal scrum meets were the ideal method for communication and division of labour. Pinning deadlines, tasks, and resources helped with organization, as well as setting informal deadlines of when we want certain functions or applications of the game to be finished was core to keeping ourselves accountable and on schedule.
Work was split among all of us. Each of us took turns to do each step of the development process, whether it be creating new classes, expanding upon old classes, or implementing new logic to the game. Our changes were generally done on branches, reviewed by other group members, then appropriately revised and merged.

External libraries:
The most notable library we used was the Java Swing library. This library formed the backbone of our GUI and featured in a significant portion of our game logic. This library was chosen due to being a very popular library that includes many foundational features that are not necessarily included in native Java. It is also a

fairly lightweight choice of library for graphics. This makes our code suitable for many different varieties of devices without dependence on the exact operating system used. In addition, some of our group members had prior experience with Java Swing, which would be an asset towards implementing the game.

Measures taken to promote code quality:

When writing code, we made sure to consider the tenets of good code studied in class.

For example, we made efforts to ensure that our code was closed to change, but open to extension. An example of where we applied this was in our enemy classes. Although we have a concrete class of MobileEnemies, should another programmer want to add a subclass of MobileEnemy that only chases the player while they are within a certain radius, it is fairly easy to do so by extending the existing MobileEnemy class.

Another key consideration we took was to reduce the number of hard-coded values. This way, our functions maintained the DRY coding principle by reducing repetitive code. For example, both players and MobileEnemies need to check if their hypothetical movement would be valid; they cannot stack nor collide with walls. Thus, we made one function to check for wall detection. Rather than hard-coding in the player sprite's dimensions in our overlap check though, we allowed the user to specify a Rectangle of the unit to check's location and dimensions. This made our code more reusable.

We also made sure to maximize our usage of the Law of Demeter. No function of ours exceeds the single dot limit in our classes' methods. This also indicates reduced coupling and enhanced levels of encapsulation within our design. These both promote good object-oriented principles.

Furthermore, adding Javadoc comments to all our functions helps reduce confusion about functions and variables as well as a way to help us keep track of what the purpose of each function is in the game. It allowed for a quick understanding of function parameter inputs and expected returns.

Overall, our code aims to be clean, understandable, and implement many of the design principles seen in lectures.

Challenges:

Some of the biggest challenges we faced included just getting off the ground. To render anything with our program, we had to have multiple facets of foundational logic all working at once. Thus, we had to have not just a single working class, but multiple classes working to start to see results. Although it took a bit of effort, we

were able to debug our issues and hit the ground running once the panel was rendering correctly.
Challenges such as making sure each interaction between each object was what it was expected to be also proved to be difficult for a while, as each object would need to be implemented perfectly before we would be able to test extensively how each would interact with the player and other objects around it. This would require multiple runs of playing the game through to make sure each function would not hinder another and seamlessly fit into the game.
One of the hardest challenges within the game was to have the ability for enemies to follow the player and appropriately detect collisions. We had to employ some mathematics to determine which direction the mobile enemies would move, as well as to detect when a player hits an obstacle or enemy. The implementation required a lot of communication between our team members, as well as online research on ways to better approach this with Java.
Another challenge faced, but which was to be expected, was debugging the code we wrote. Any complex program will inevitably have bugs, and debugging took a significant portion of our development time. One bug that came up several times was related to using an array to pass x and y coordinates. Accidentally putting a 0 instead of a 1 to access a certain element created strange and difficult to explain behaviour. Bugs like these took time to fix, but their removal created solid code that was difficult to break.

Closing:
Overall, we had an educational and informative experience working to create our game. We got to practice the techniques for software design we learned in class, and received hands-on experience using Java, Maven, and Git; all of these are useful skills in the workplace. In addition, getting to work in a close-knit group-oriented environment helped facilitate our communication, social, and organizational skills. We hope you enjoy our game and look forward to the next phase of the project.