Phase 2 Report

CMPT 276

Group 23

Nov. 6, 2022

Overall Approach

Our overall approach to the development of the game was very similar to the Unified Process SDLC. Using the UML diagram and use cases we developed in Phase 1, our communication and planning stages were already done and we could move straight into modeling our game.

During the modeling phase, we set up our skeleton code and focused on the architecture of classes that did not rely on object input such as Score, Timer, and Position. We also informed ourselves on the implementation of our libraries that we included in our UML diagram and built separate classes for UserInput, GameTimer, and Window. Our Game class required us to use threads and states to reflect the player's progress and this class became the backbone for the rest of our game logic. The calls to our Graphics methods were included in various draw() functions in our Game and Map classes and our character images were included in the MovingEntity and StaticEntity sub-classes constructors.

The combination of our skeleton code and library method use allowed for a smoother transition into our Construction phase to build the first version of our game. This consisted of the backend development towards the behavior of the characters and the frontend development towards the image renderings of our map and entities. Once we got our GUI to properly display and work with the entity behavior, we began conducting tests to make sure we were meeting the requirements detailed in Phase 1.

Finally, we reached our deployment stage where we wrote formal documentation using JAVADOCS and made note of known issues to fix in the next Phase.

Adjustments from Phase 1

From the methods we described in Phase 1, we were able to implement all of them but found we needed to extend access to certain variables or create separate classes for methods that were becoming too large. Below is a list of our major changes:

- Added getter and setter methods to Map, Entity, Cheese, Crumb, Mouse. The purpose of these was to be able to get positions/objects from the map that were located in arrays to use in other classes and set positions on the map to emulate movement.

- User input for mouse movement was changed into its own class. Here, a flag was also added to trigger the cat's movement on the first key press from the user. The purpose for this was to avoid creating a method inside <insertClass> that would have been too long and was better suited as its own class.
- Created a Menu class to store and draw the menu. The reason behind the class is similar to the point above.
- Created a Window class to generate a separate window to create the GUI
- Created a tick() method in Map class to keep of when a ticked has passed, so we can then update character movements
- Created a bunch of generate methods in Map class to create objects that spawn at the start of game
- Made CheeseExist() in Map generate and remove cheese, so timer value and countdown method weren't needed in Cheese Class

Project Management

Every member of the team was able to choose which classes they wanted to implement at the start. The code was developed on individual branches and we merged with each increment. As the game progressed and we relied on other parts, we made the necessary adjustments in classes we didn't work on or helped our other teammates with their code development. We met at least once a week either in person or online through Discord to facilitate peer programming and messaged the group chat with any questions or updates. Here is the breakdown of our work:

- Thimira: Map, Cheese, GameTimer
- Robert: Position, Entity, MovingEntity, Mouse
- Ethan: Game, Window
- Karina: Score, Cat, UserInput

For the JAVADOCS comments, everyone took four classes to write the documentation in and the report was contributed to by all of us. The list below outlines who wrote the JAVADOC comments for each class:

- Thimira: Map, Crumb, MouseTrap, StaticEntity
- Robert: MovingEntity, Mouse, Entity, Position
- Ethan: UserInput, Window, Game, Menu
- Karina: Cat, Cheese, GameTimer, Score

External Libraries

The external libraries we used were java.util.*, java.awt.*, and java.io.*
Java.awt was used to create the user interfaces allowing us to use shapes and dimensions for the menu and also using awt.event for navigating the menu.
Java.io was used for the images and graphics we need from our resources folder.
Java.util was used for storing entities in an ArrayList so we can keep track of all the entities in the game.

Code Quality

Our top priority for enhancing code quality was making sure we stuck to OOP principles such as Inheritance and Polymorphism. We attempted to generalize methods that were used in multiple classes by having them inherited from the base classes and have descriptive names for our variables.

Challenges

Throughout the development of the project, we made sure to merge other people's work with our own branches to get the most updated code and not leave it for the last few days. This would sometimes cause compilation problems that we had to fix immediately and could take a few hours to figure out.

Our biggest challenge came with figuring out the cat's best next move where we attempted to implement a breadth-first search algorithm with a depth of 2. Although the majority of the code is done, we decided to leave it for Phase 3 as one of the areas we could improve and instead, went with a simpler version of the movement.