Nazareth Phase 2 Report
Group 19
CMPT 276

Overall approach of implementing the game

For our project, we made a game titled Nazareth: Search for the Lost One. The main approach we took however was to follow the UML but to trust ourselves with making changes because our past selves were making these documents without any real experience in making a game with Java.

The core logic of the implementation of the game is alternating between different states based on the user's actions.

We made use of the following number mappings to indicate 6 possible states of our the game:

0- Player is on the Main Menu.

1- Player is on the Game Screen.

2- Player is in the Lore Menu.

3- Player is in the Settings Menu.

4- Player is in the Pause Menu.

5- Player is in the Victory Menu.

6- Player is in the Failure Menu.

Each of the states has one or more preconditions and some states can only be accessed from a particular state.

The whole game resides in one window container and switches to different states depending on the user's input.

Models of the states are designed using a series of image drawings, each layer drawn on top of another. The images are taken from free and non-copyrighted resources on the Internet.

Adjustments and modification made to the initial design

In this release, we were not able to implement the Lore, Settings menus and StartGame specified in the design phase of the project. In order to meet the deadline, our group has decided to postpone the implementation of these modules for later releases.

Changes made to the UML diagram:

1) Tick class now holds 2 private fields. One of the variables is holding the tick count and our game logic resets the tick variable to 0 every 4 ticks to update the moving enemy's position on the board. The second tick variable holds the tick amount starting from the initial launch of the Game Screen in order to ease the process of spawning bonus rewards on the board.
2) The Moving Characters Skull and Player class are now dependent on the Game Screen. The reason for this decision is that the Moving Character class is switched to an interface.
3) The Grid class is split up into two dependent subclasses: Tile and TheGrid. The tile class specifies the type of tile. A tile corresponds to a number mapping we chose to indicate the type of an object at a specific coordinate. TheGrid holds an array of type Tile and keeps track of the tiles of the board.
4) Tick class now is not dependent on the GameScreen but runs on a separate thread. We found it easier to make a singleton Tick class keep track of the passed time.
5) We ended up scrapping the interfaces Reward and Start Game. The Reward interface was dropped and we ended up making 2 classes Reward and BonusReward, but we realised BonusReward can be reworked into an extension of Reward.
6) We were only able to make one level so we did not need an interface to help us make multiple levels.

Most of the classes have different functionalities. One of the biggest reasons is that at the Design stage of our project we did not account for GUI external libraries. Most of the logic has been changed to accomodate the GUI's usage.

Discarded classes:

OuterWall
Entrance
Exit
Settings
Lore
Reward

Management process and division of roles

To help keep our group on track, we chose to meet on Discord. These meetings happened almost every night, and sometimes a few nights in a row. We would discuss any changes that we had made during the day and also any issues that we were able to finally overcome. During this time we would also take the time to share our screens and explain how we implemented our work. This helped the rest of the group understand the meaning of the code, which was useful for when others needed to interact with the code.Whenever any of us were having trouble , we would use this time to troubleshoot with everyone together. More likely than not the problem would be solved by the next day. The group also chose to do our merging during these times as well. We were able to explain the changes we made and could solve any merge conflicts together. Once the night was over, we would look over our UML from phase 1 and divide out tasks accordingly. Some items took longer, so they would take multiple days to complete. No member was assigned responsibilities, but instead chose tasks that they were confident in completing. Each member chose a section that they felt most comfortable with, such as GUI or implementing our character's functions.

External Libraries

The external libraries our group chose to work with were Slick2D and LWJGL. When looking online we found lots of positive posts about LWJGL and found that Slick2D was great for beginners. We learned that Slick2D was based around LWJGL and decided to implement both. Slick2D worked great for us, because we were able to create different game states that allowed users to switch between menus and the game. Slick2D handled all of our graphical needs, while LWJGL focused on our inputs.

The measures we took to enhance the quality of code.

1) Naming convention for our class method to ease the accessing and understanding of the functionalities.
2) We used the singleton design pattern for classes that were operating on one instance.
3) Each object drawn on the screen is defined in a separate class to make it easier to manipulate related classes.
4) We followed the modularity principle so each class is less dependent on another.
5) Information hiding.
6) Abstraction. We made use of interfaces for classes that share the same functionality.
7) Followed the single dot rule.

Our biggest challenges

We faced small issues here and there while working through this phase, but some issues seemed a bit too overbearing. 2 of our members lived in the western hemisphere, while the other 2 lived in the eastern hemisphere. The group decided to meet around 8:30pm PST, so 2 of us would be heading to bed and the other 2 would just be waking up. The time difference meant it was harder for us to ask questions or get information from other members without having to wait hours. Due to the time difference, the 2 members in the eastern hemisphere had difficulties attending office hours and they instead had to ask a third party to ask questions for them. Git worked most of the time, but merging seemed to have brought some difficulties. Screen Sharing while doing these merges became a useful tool and helped quicken the process. Lastly, Maven was our biggest challenge to work with. The group did not have a full understanding of how Maven operated and what it required. This caused our game to inexplicably stop working and we would be forced to spend time troubleshooting. It wasn't difficult to get LWJGL as a dependency in Maven but since Slick2D did not have a version, we were forced to try different versions until the code was able to run properly.