SFU

SIMON FRASER UNIVERSITY

ENGAGING THE WORLD

CMPT 276: Phase 2

Nov.6, 2022

Our overall approach to implementing the game:

We did our best to modularize everything into their respective packages. We first started with implementing the models of our game which are described in our UML class diagram. After that, with the interface of the Board class we defined, some of the group members went ahead and started implementing the UI components of our game using the Java Swing library, while other members worked on the logic of the game.

We separated the game logic and UI so that the game logic can be used in any type of UI. With this structure, we were able to work in parallel with each other, as the board logic is abstracted from the client (in this case the UI).

Once everything was implemented, we pieced together the UI client and the logic class together to create the game.

State and justify the adjustments and modifications to the initial design of the project (shown in class diagrams and use cases from Phase 1):

There were some holes in our initial UML diagram design. We did not have an abstract creational pattern for our game entities. Although we were using Lombok. Builder to create builder methods for each class, there were some class objects that were children of another. To abstract the complexity of our animate and inanimate entity creation, we created factories for their respective categories. This way we were also able to define defaults that were specific to our game theme.

There were also redundancies in our initial design, the Space entity was only holding a Position object, which can be directly put into our Board class since we don’t need another class to wrap the object model. With this change, we are able to utilize a Set to keep track of barriers, and checking for valid movement is much more efficient since we no longer need to iterate through the whole list of Space entities.

With the current design, we can create many different themed Boards and create many different themed UIs by making very small changes, and extending the current functionality that has already been implemented.

Explain the management process of this phase and the division of roles and responsibilities:

During this phase, we mainly communicated through discord, and held meetings in person and sometimes through discord chat, but we mostly just managed ourselves, and give ourselves tasks to do. Whenever a problem came up relating to another person's part, we would tell them and they would fix it themselves or we would just collaborate on a certain issue to fix it.

The roles were mainly split into 3 sections, the sections board logic, board factory, and UI. Ryan was mainly responsible for the board logic section, Parth the board factory, and Leon and Lawrence for UI. Even though the roles were split, we all collaborated and contributed to all the sections such as debugging, and coding during the in-person or virtual meetings.

List of external libraries we used, for instance for the GUI and briefly justify the reason(s) for choosing the libraries:

The libraries we used were Lombok, JSwing, abstract window kit (AWT).

We used Lombok to reduce the clutter in our code by getting rid of needing to write getters and setters and to mainly utilize the builder pattern that Lombok generates.

We used JSwing and AWT which were used for the UI component such as JPanel, and JFrame. Lastly, the KeyListener is used to listen for user input to then change the state of the board.

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

The first measure we took to enhance the quality of our code was using Lombok, so we can just use Lombok builder to reduce the clutter in our code for each class.

We also used a code style formatter defined as a team to have the same code format throughout the project. This was enforced so that everyone would have the same style of code, we are able to read the code easier, and we are able to automatically remove any unused imports.

We also reviewed each other’s code to give suggestions for writing cleaner code. Ideally, our code was written so that no comments were required, and that the code was self-documenting. We wrote brief javadocs to describe the classes that had specific uses in the project.

We also took into account how we would unit test the classes and their respective methods to prevent any potential code smells. i.e. If a behaviour or result was not the classes’s responsibility, we should be able to mock it in our unit test cases.

The biggest challenges we faced during this phase:

The biggest challenge we faced was resolving merge conflicts when they happened. Some group members are not as experienced with version control, but we managed to resolve them through communication.

We were also very busy and have very different schedules. It was a challenge to set up a time when we can meet and discuss the project, but we had Discord and remote meetings as an alternative to get group members together when there is a problem.