PROJECT TITLE: Minesweeper

PROJECT MEMBERS: Meeti Baliga and Krithika Damodar Bhat

FINAL STATE OF SYSTEM

When the user runs the .jar file, he/she will be able to play a simple version of the minesweeper game. The user will first be presented with a GUI interface to play the game. During the game, if the user steps on a mine, he/she will lose the game. If the user manages to flag all the mines and open all the cells, without stepping on a mine, he/she will win the game.

The following features were implemented in the minesweeper game:

1. A GUI is displayed to the user with a timer, number of mines and a new game button.
2. If the user clicks on the new game button, a new game is spun up.
3. If the user performs a left click on a cell and there is a mine underneath, the user loses. The GUI displays a 'game over' message at the bottom of the panel and the number of mines left. The timer stops and displays the elapsed time.
4. If the user performs a left click on a cell and there is no mine underneath, nor are there mines surrounding it, the surrounding areas around the cell with no mines are opened simultaneously.
5. If the user performs a left click on a cell and there is no mine underneath, and there is/are mines around it, the cell is opened displaying a number corresponding to the number of mines adjacent to that cell.
6. If the user performs a right click on an unopened and unflagged cell, a flag (represented by '?') is placed on the cell. The user cannot open a flagged cell.
7. If the user performs a right click on unopened and flagged cell, the flag is removed from the cell. This means the user will be able to open it.
8. If the user flags all the cells with mines underneath them, the mine counter is 0 and all cells (without mines underneath them) are opened, the user wins the game.
9. If the user wins/loses, a dialog box appears and displays game statistics.

The following features were planned but not implemented:

1. We had planned to use singleton pattern to implement the MineField Class but that did not work, as in one session, the user can play multiple games, which would then require the MineField Class to be reinitialized with a different minefield. We instead implemented the singleton pattern for the Controller class.
2. We had planned to have three different difficulty levels (EASY, MEDIUM and HARD) that the user could set before playing the game. It could not be done due to lack of time. Hence the values for making the size of the grid, the number of cells and number of mines have been hardcoded in the program.

FINAL CLASS DIAGRAM AND COMPARISON

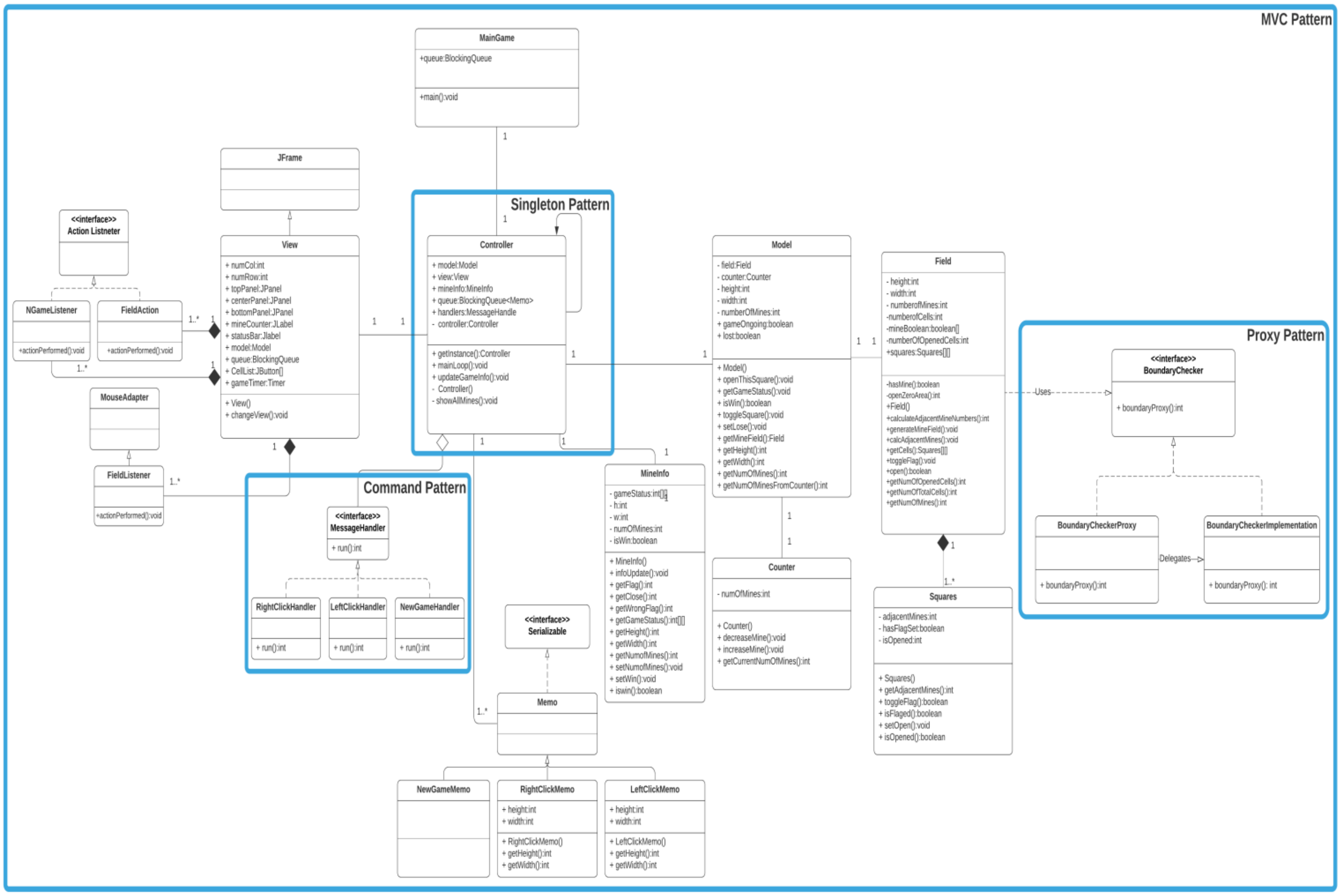

Fig 1.1 Final UML Diagram

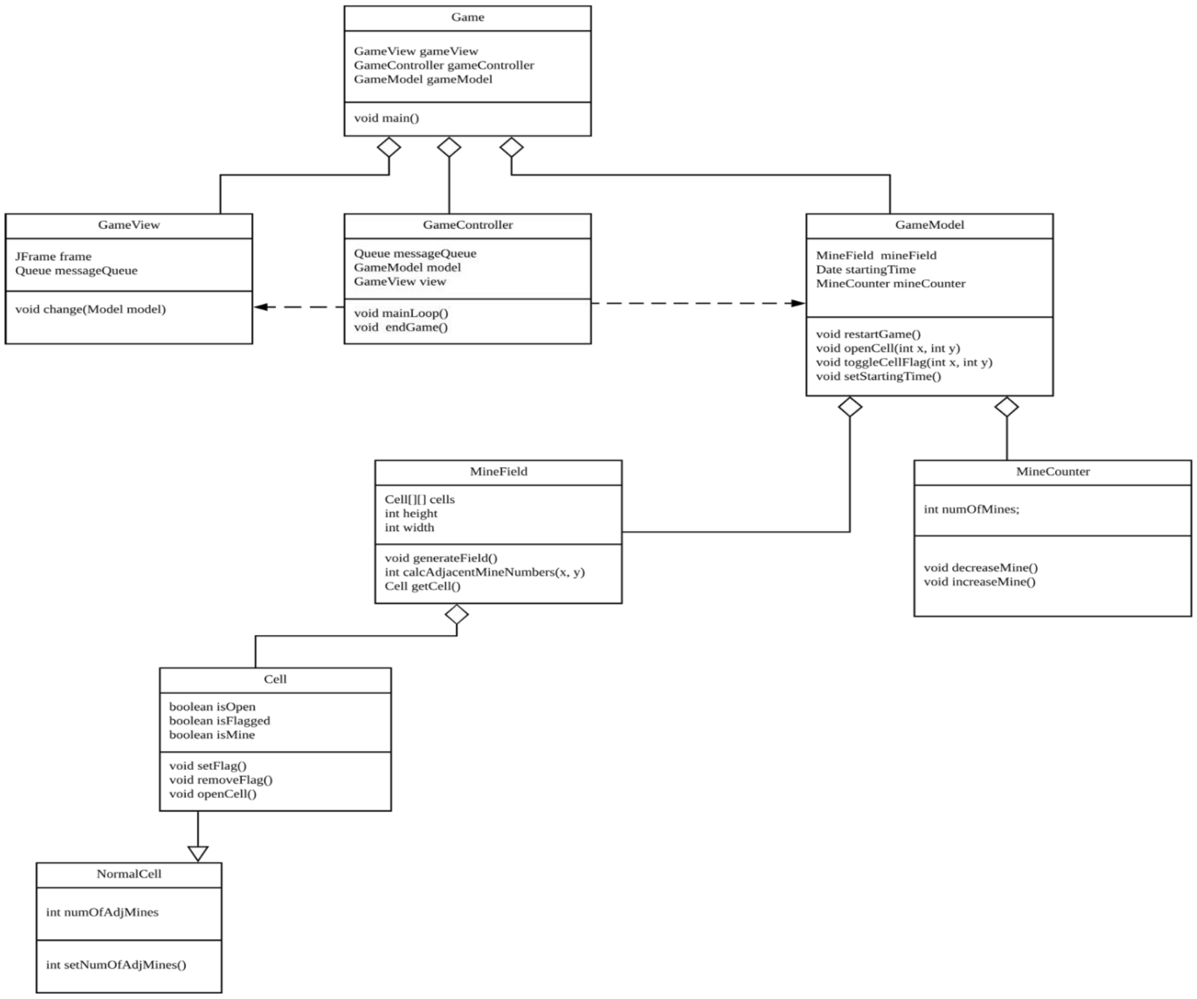

Fig 1.2 Project 4 UML Diagram

The following were the differences between the UML diagrams of project 4 and the final UML diagram:

1. We were able to incorporate the proxy pattern in the final UML diagram as opposed to the absence of the proxy pattern in the UML diagram for project 4.
2. We moved the Singleton Pattern from the MineField class to the Controller class as we saw that a particular game session could contains multiple instances of MineField whereas the Controller has only one object.
3. We were able to implement an extra design pattern in our project as we progressed which was the Command Pattern. We identified the concrete classes NewGameHandler, RightClickHandler and LeftClickHandler, and the interface MessageHandler in our code that aligned with this particular design pattern.
4. We initially planned on using MySQL as a datastore for the Controller to access MVC messages. We now have used Java’s Serializable interface to implement messaging queues as shown in the UML diagram.
5. We used a MYSQL database for storing statistics on the games played by the user.
6. Various listeners have been added as helpers to the View object which we did not think of in the first iteration.
7. We created only a single controller object in MainGame.java as opposed to the first UML diagram which created 3 objects, Model, View and Controller.

ORIGICAL CODE VS THIRD-PARTY CODE

We referred to the following sources to get some code logic for the minesweeper game:

- http://zetcode.com/tutorials/javagamestutorial/minesweeper/
- https://www.geeksforgeeks.org/cpp-implementation-minesweeper-game/

Some parts of view.java have been borrowed from the above 2 links. The code to implement boundary checker proxy pattern was replicated to fit the project's needs. The rest of the code is original.
We referred to the following tutorials:

- https://www.baeldung.com/java-proxy-pattern
- https://medium.com/@ssaurel/learn-to-make-a-mvc-application-with-swing-and-java-8-3cd24cf7cb10
- https://docs.oracle.com/javase/tutorial/uiswing/index.html
- https://www.youtube.com/watch?v=RFpJp62ZoY8

ANALYSIS OF THE OOAD PROJECT

The following were some of the challenges we faced during the whole process:

- The first issue which we encountered was to identify relevant design patterns that could fit into our project requirements to make it more efficient. We went back and forth from trying to implement strategy pattern, iterator, and then finally decided to implement the singleton and the proxy pattern.
- Since the game does not necessarily require a datastore, we had to explore Java's inbuilt messaging services/app preferences to store any stateful information that was generated by the game.
- Due to a smaller number of team members, we could not implement a few features that was initially planned i.e. in the initial planning phase, we overestimated on how many features we could incorporate into the game.

The project helped us understand software development process in more detail. More importantly, we learnt how design patterns can be incorporated to deliver a more efficient software system.