**Data Structures And Algorithms**

**Project Report**

**Game Minesweeper**

*A game written in Java applying DSA principles*

*With Undo feature*

**Members:**

**Phan Hoàng Long – ITITRG18019**

**Nguyễn Kỳ – ITITIU18274**

**Demo Day: 7/6/2022**

* Section I: Introduction
* Section II: Github repository
* Section III: Rules of game
* Section IV: Function of version
* Section V: UML class diagram
* Section VI: Key design concepts
* Section VII: Future development

1. **Introduction**

In this report, we introduce our version about Game Minesweeper . This version is a effective game with undo function helping users to undo their turn in a well way. We developed this software with the help of the JSwing framework of Java.

1. **Github repository**

More details about the project through the source code:

https://github.com/minkate99/MineSweeperDSA.git

1. **Rules of game**

The goal is to sweep all the “mines” or bombs from a 16x16 mine field. There are 40 in total.

To obtain information on where the bomb is, left click to uncover the cells. A cell with a number reveals the number of neighboring cells (the 8 most direct ones surrounding it) containing bombs, although itself will not contain a bomb. A cell that does not contain a bomb in its direct neighbor cells (the 8 most direct ones surrounding it) is an empty cell, and when clicked on, will reveal the entire region of all empty cells until a cell with a number appears. Use this information plus guess work to avoid the bombs.

To mark a cell you think is a bomb, right-click on the cell and a flag will appear. You have 40 flags in total, one for each bomb. You will be notified when you have used up all your 40 flags with a count of how many flags you have left in the lower left corner.

The game is won when the user has successfully identified all the cells that contain bombs and the game is lost when the player clicks on a cell which contains a bomb.

The user can “un-flag” a cell by right clicking the cell again. The user can undo any number of moves for any type of move, which includes clicking on flagged cells, empty cells, and neighbor cells.

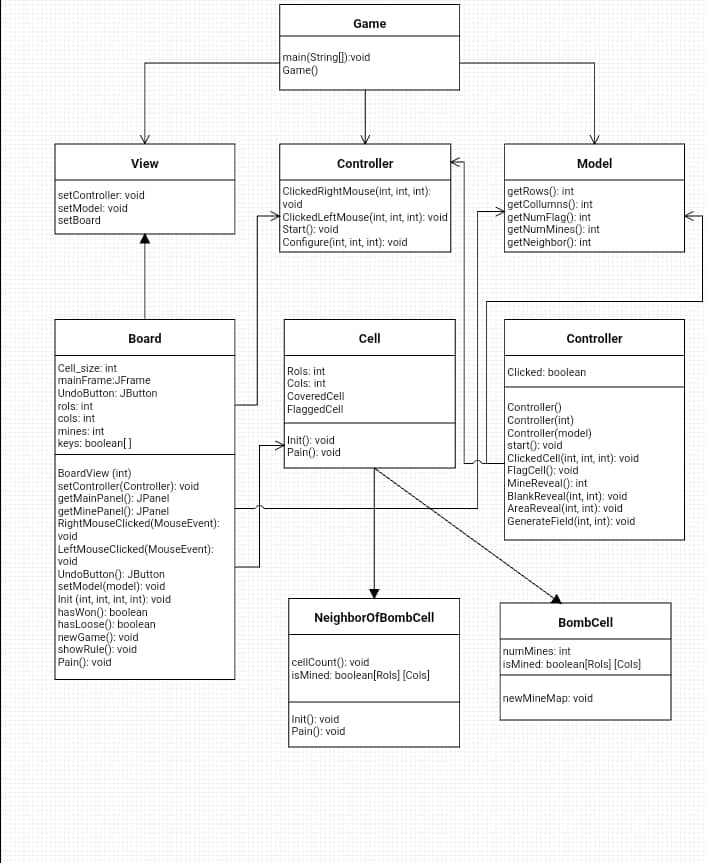
To start a new game, the user just clicks anywhere on the board. The user can stop the game at any point by exiting the game. The game will automatically be saved. When re-loaded, the user will have the option of starting a new game or starting from the most recent version of the game when exited.

1. **Function of this version**

This version is similar to traditional Minesweeper game except the user has the option to "save" game status and “undo” any number of moves.

* The user can only undo moves after he/she has played any moves, which is true even when the previous saved game status is reloaded, i.e, the user cannot undo moves he/she made in that previous round (If the game is reloaded with 2 flagged cells from the previous round, the user cannot undo those 2 flagged cells, but can undo the 3rd flagged cell, which would count as the user's first move in this new round).
* The game status is automatically saved when the user exits out the window, and is given the option of whether they want to restore previous status or start a new game when the game is reopened.

1. **UML Diagram**



1. **Key design concepts**

**2D Array**

Since Minesweeper takes place on a 16x16 grid, we used a 2-D array of “Cell” objects to represent the grid to store the state of each cell in the grid in terms of which cells contain bombs, if they are empty, or if they are neighbor of bomb cells (i.e “number’ cells). “Cell” was a parent class we created that stores the properties of each cell and handles basic cell behavior depending on its type. The 2-D array was used whenever a new game was started, in which the 2-D array would be randomly filled with bomb cells, number cells (cells that are neighbors to bomb cells indicating how many bombs are in the vicinity), or empty cells. We would traverse through a double for loop that represents the rows and columns of the 2-D array or call individual elements of the 2-D array to initialize Cell objects, handle different user actions on a specific cell, or obtain information about about a certain cell (to read from/write into a text file when saving and loading the game status). Our use of a 2-D array was extremely fundamental in organizing our overall structure of the game board and locating user actions/specifying desired locations to handle game logic.

**File I/O**

We used a File I/O to store and save 1) the player’s username and 2) the game state in a file when user wants to close the game and play again later. The state of the grid in terms was represented by the 2D array is stored in a file, which is loaded again when user wants to continue the game from before. The properties of each cell are saved in a text file, and when player wants to load the game, the file is read and parsed. In the text file, each line represents a the saved state of a cell (256 lines in total, representing the 256 total cells), with 4 pieces of information, converted to “String” type: 1) the cell type 2) whether the cell is marked 3) whether the cell is flagged 4) if applicable, what the number on the cell is (only for cells indicating the number of bombs nearby). The username and game state are separated by the “$” character, and each cell in the 2D array game state is separated by a whitespace hyphen whitespace “ - “.

**Collections:**

We decided to add a feature to our Minesweeper game in which a user can "undo" their moves when they hit the "undo" button, unless the user hits a bomb, which then the game ends. We implemented the undo function by storing the user moves in a Collections called Stack, which takes advantage of the "last in first out" property to undo the user's most recent step by "popping" it out of the Stack — removing the user move from the Stack and returning the values (the state of the cell depending on the user’s last click). We also use the "push" function to add steps to the Stack whenever the user left clicks a number or empty cell, and right clicks to flag/un-flag a cell, which represents the user’s most recent move. The user can undo any number of moves for any type of move, which includes clicking on flagged cells, empty cells, and neighbor cells, but for bomb cells, the Undo function is disabled (to ensure the game is over when a bomb is clicked) by “clearing” the Stack.

**Recursion**

We used recursion to reveal and clear a mine-free region of cells, specifically, when a user clicks on a cell that has no bombs with adjacent cells also bomb-free so all adjacent cells are automatically cleared and the region with bomb-free cells is revealed up until the cell that is connected to a bomb.

1. **Future development**

We will improve:

* + Save function to save more than 1 previous status.
  + More eye-catching design display.
  + “Redo” function.