INFO-I 311
Group 4: Matt Allen, Mason Ashment, & Nina Dorenbos
Minesweeper: Phase II Documentation

NOTE: Documentation in italics is from Group 1's Phase I Documentation

UML Class Diagram:

General Information:
Minesweeper is a Java application that utilizes the JavaFX graphic library for its GUI. It can easily be run using the IDE Eclipse.

Game Description:
Minesweeper is a game that presents the user with a grid of cells. Depending on the difficulty selected, a percentage of the cells are hiding mines and the user's goal is to select all the cells that are not mines. The user is aided in this process with the number of adjacent mines revealed to the user when a cell is selected. The user may also mark a cell it expects to be a mine.

Game Flow:
When the game is run Main.java is called and loads the BoardDisplay.fxml GUI. The `initialize()` method is called which set the default controls for a game.

When the 'Start' button is pressed previous boards are cleared and a new one with the specified difficulty and size is created by calling `buildBoard()`. It is then displayed, and the timer starts.

When a cell button is selected `btnPressed()` is called. It determines if a flag must be set or unset or calls the `checkReveal()` method. Game win/loss is checked, and the timer stops if the game has ended.

Architectural Decisions:
We decided to forego the class `CellButton` specified in the Phase 2 Project Description. This decision was based on the previously designed architecture in addition to the required functionality satisfied without it.

Classes:
Main.java, Cell.java, Board.java, BoardDisplayController.java, resultDisplayController.java

-----Main.java-----
The class launches the application by loading the fxml code.

-----Cell.java-----
The cell class is used to create new cells which are then stored in the board's ArrayList to be used in the game. This class includes methods for checking if a cell is a mine, is flagged, or has been revealed which then determines how it is represented in the board and a method to show the count of nearby mines, as well as other methods used to set the status of the cell used in the board. Stores the variables for use in the logic of how the board is displayed as well as what happens.

-----Board.java-----
The board class is used to create a list to store Cells in and plays a large part in the making of our game. This class includes some complex methods for generating the board itself and storing cells inside of it and randomizing the placement of the mines as well as methods for revealing the spaces on the board recursively if a blank cell has been clicked. This class also contains the methods to check if a game has been won or lost. Acts as a communicator between the Cell class and the GUI, storing all of the individual cells as well as where they are located on the board and telling the GUI the status of the given Cell when clicked.

-----BoardDisplayController.java-----
This is the controller for the main window that is launched when the program is run, which includes the board that displays to the user. As the UML class object above shows, BoardDisplayController has four variables along with 15 fxml variables. There is also an event handler `btnHandler` that is linked to every button on the board.

`-startTime` is time a game starts
`-timer` is a `Timer` object
`-game` is a `Board` object
`-mineTotal` is the number of mines in the board determined by the difficulty selected

`void initialize()` is a method that when the fxml code is loaded, sets the default controls for the game: difficulty to easy and board size to 5x5.

`void startPressed()` is an action method called when the start button is select. It clears the board of any past cells and then creates a board specified by the users input in difficulty and size. If easy, medium, or hard is selected 10%, 15%, and 20% of the cells are marked as mines respectively. A `GridPane` is created with the size constraints specified by the user and then `game` is set to a new `Board` with the size and mine count and `buildBoard` is called. Lastly, the timer is set and begins its tally.

`void buildBoard()` is a method that fills the grid with buttons to make a board. For each cell in `game` a button is created with a label behind it. For each cell if it has surrounding mines the number is set to the label with a corresponding color. If the cell has a mine the mine image is set to the label.

`void btnPressed()` is an action method that is called when a button in the `GridPane` is selected. First the method determines the location of the selected button to determine which cell it is in the board. If the flag checkbox is selected the button displays a flag or removes a flag, if the flag is displayed the cell cannot be revealed. If the flag checkbox is not selected `checkReveal()` is called. Then, there is a determination if the game was won or lost and if so, the timer stops.

`void checkReveal()` is a method that reveals the buttons covering labels of cells.

`void clearBoard()` is a method that clears the board

`void clearBtn()` is an action method that clears the cells on the board if the ‘Surrender’ button is selected.

-----resultDisplayController.java-----
This is a controller for the resultDisplay.fxml window that displays the game history to the user. It controls the text for the labels on the window. The labels show: the win/loss frequency at each difficulty and the best time for each difficulty played. All the methods are setter methods for the text of labels beside the one action method `wipeHistory()`

`void wipeHistory()` is an action method that is called when the ‘Reset Data’ button is selected. This method sets all of the labels to zero.