CS 342: PROJECT 2
Minesweeper

Bryan Spahr
George Maratos

TABLE OF CONTENTS

1. INTRODUCTION **3**
- 1.1 Purpose 3
- 1.2 Scope 3
- 1.3 Reference Material 3

2. SYSTEM OVERVIEW **3**

3. SYSTEM ARCHITECTURE **3**
- 3.1 Architectural Design/Overview of Classes 3-4
- 3.2 Design Rationale 4

4. DATA DESIGN **4**
- 4.1 Data Description 4
- 4.2 Data Dictionary 4-7

5. COMPONENT DESIGN **8**

6. HUMAN INTERFACE DESIGN **8**
- 6.1 Overview of User Interface 8
- 6.2 Screenshots 9

7. REQUIREMENTS MATRIX **10**

1. INTRODUCTION

1.1. Purpose

This design document describes the architecture and system design of this project, which is Minesweeper

1.2. Scope

This project is a GUI interface game, written in Java, that contains four classes

1.3. Reference Material

Mainly the project handout; also various Java SWING web sources

2. SYSTEM OVERVIEW

The goal of this game is to uncover all squares on the grid without triggering a bomb, with standard Minesweeper game mechanics in place. The process of this program can best be described by these steps:

a) The Game begins with 100 buttons in grid formation, the goal being to uncover all squares that don't have a bombs.
b) Left clicking on any square will reveal what is under it.
c) The square will reveal whether or not it contains a bomb or how many bombs are adjacent. (in all directions including diagonals)
d) When the user is certain that a square has a bomb, they can right click on that square to flag it as containing a bomb. If (at the end of the game) they were right, the button will be marked with a defused mine icon. The user can also right click on a flagged button again to give it a temporary "?" mark, which will not count as flagged but can be used to label potential mine locations.
e) If the user clicks a square containing zero adjacent bombs, the game will proceed to clear out all adjacent squares that contain zero adjacent bombs as well.
f) The game ends when the user uncovers 90 squares none of which contain a bomb
g) When the user wins they can enter their name I the leader board, when they want to play again the user can access the pull down menu "Game" and click on "Reset"

3. SYSTEM ARCHITECTURE

3.1. Architectural Design / Overview of Classes

Button: The Button class is the basic unit of this game. It contains necessary info like its coordinate on the grid, whether or not it is a bomb, if the button can be clicked, and if it is flagged or not. Method functions include, toggles between states, getter functions for other classes to determine what button contains.

Score: The Score class is used for storing information into the Top ten leader board, which can be viewed from the game under "Game" tab. Score contains three variables which serve the purpose of holding a single user's name and time of completion.

Method's include getter functions for a formated name and a formated time. Also implements a Comparable override compareTo so it can be sorted by its time.

Game: The Game class is the heart of the program that contains most of the functionality of the program, ranging from displaying the formated Minesweeper board, randomizing bomb location, keeping track of game time, and performing all necessary Minesweeper game logic. Variables include JSWING elements (JFrame, JLabel, JPanel), an array of Button objects, size of board, integers to keep track of game time, and various integers to keep track of game state (for example bomb counter). Methods include functions necessary to initialize the table for example setting JPanel and JMenu to the JFrame, making sure bombs are distributed into unique coordinate locations, clearing adjacent squares containing no adjacent bombs, displaying updated board, updating scores to the leader board, reseting board, and keeping track of game time.

Main: The main class being necessary to run the program simply creates a game object and sets frame to say “Minesweeper”.

3.2. Design Rationale

The goal for the source code was to separate it into as many classes as possible, to make it more concise and easier to understand. The Main class was created to isolate the Main method that starts the program. The Button class was created to implement JButton, but contain additional functions and global variables to interact with the board. The Score class was created as an asset when reading score data from the data file. The Score class is also sortable by time. The Game class contains everything else in the game, including GUI elements and various Listeners.

4. DATA DESIGN

4.1. Data Description

All collections of objects in this codebase are stored in ArrayLists.

4.2. Data Dictionary

Button:

```
public Button(int iPos, int jPos)

private boolean flagged

public int getI()

public int getJ()

public ImageIcon getIcon()

public int getToggleState()
```

```
public int getVal()
private boolean hidden
private int I
private ImageIcon icon
private ImageIcon[] icons
private boolean isBomb
public boolean isBomb()
public boolean isDebunked()
public boolean isFlagged()
public boolean isHidden()
private int j
public void reset()
private static final long serialVersionUID
public void setBomb(boolean bomb)
public void setHidden(boolean b)
public void setToggleState(int toggleState)
public void setVal(int val)
public void toggleFlags()
private int toggleState
private int val
```

Main:

```
public static void main(String[] args)
Game game
```

Score:

```
public int compareTo(Score s)
String name
```

public Score(**String** line)

private static final int N

String time

public String toString()

int totalSecs

Game:

class aboutWindow

public void addScore(**int** score)

private JPanel bar

private JPanel board

private JLabel bLabel

private int bombCounter

private Button[][] buttonGrid

public void clearZeros(**int** i, **int** j)

private ClockListener cl

private class ClockListener

private JPanel content

public void createScoresFile()

public void create_menu()

private int currSecs

private int deadCounter

private int endCounter

public void findBombs()

public Game(**String** windowLabel)

class helpWindow

public void init()

private JFrame mainFrame

private static final int N

```
private final int numOfBombs

public void refreshButtons()

public void reset()

public void showBoard(boolean gameOver)

private final int size

public void startTimer()

public void stopTimer()

private Timer t

private JLabel tLabel

public void updateBombLabel()

public void updateEndCounter()
```

5. COMPONENT DESIGN

`public void init();`
This function's purpose is to initialize variables, create the buttons on the board, and set the random bomb locations.
First, an ArrayList of Points is randomly generated. As each point is generated the program checks to make sure the newly generate Point isn't a duplicate, so that the final list contains 10 unique points.
Second, the entire grid of Buttons is created and initialized to 0 (blank).
Finally, the list of Points is iterated through and the corresponding coordinates in the board are set to be bombs.

`public void findBombs();`
This function's purpose is to locate all the bombs and assign each button its appropriate value representing how many bombs there are in its surrounding 8 spaces. Basically it loops through the entire grid and counts the number of bombs in the surrounding 8 spaces, then assigns that value to the button. It also recognizes all four edge limits of the board in its loops to prevent exceptions.

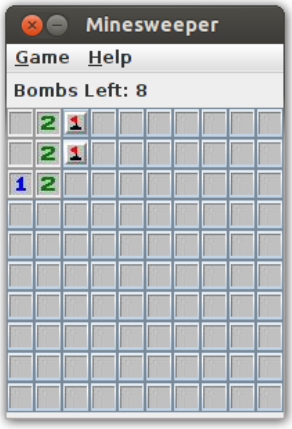
`public void clearZeros();`
This functions purpose is to clear out adjoining spaces with zeroes when one is clicked on. First, the current Button is disabled. Second, it checks the surrounding 8 spaces for more zeroes. If it finds another zero, it recursively calls itself with that button's coordinates. If it finds a number, it just unhides that button and disables it.

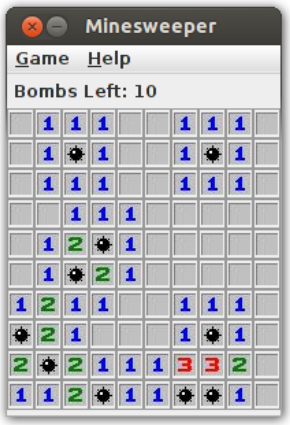
6. HUMAN INTERFACE DESIGN

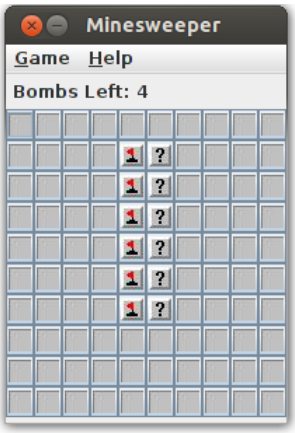
6.1. Overview of User Interface

Gameplay is as described in Section 2. The Program displays a 10x10 grid of buttons where the user has three options. They can left click on a square revealing whatever lies underneath, they can right click a single time marking the square as flagged, or they can right click a square they have previously right clicked marking that square with a question mark meaning that the user is unsure whether or not that square has a bomb. The user also have a few options in the pull menu above the grid. If they select the pull down menu "Game" they can select "eXit" which will close the program, they can select "Top Ten" which will display the Top ten previous scores ranked by time distinguished by name, or they can select "Reset" which will reset the game. If the other pull down menu "Help", then the user can select the option "help" (which is unique from the pull down menu option "Help") which will display a helpful message on how to play the game. The last option in the "Help" pull down menu is the "about" option which will display an important message about who programmed the game.

6.2. Screenshots







7.

Requirement	Satisfied By
Java Swing library	Game Button
10 randomly placed mines	Game
Toggle button from blank to flagged to ?	Game Button
Show button when it's clicked on	Game
Clear adjacents when a zero is clicked on	Game
End game when mine is clicked on	Game
End game when 90 buttons are revealed or flagged	Game
Save time and username when game is won	Game
Mine counter showing mines left	Game
Clock showing elapsed time	Game
Game menu and Help menu	Game
Top ten scores dialog	Game Score
Clear Scores	Game Score
Exit button	Game
Reset button	Game
Help button	Game
About button	Game
Optional button icons	Button