Minesweeper

Software Design Specification

(Session 2018 - 2022)

Guided By:

Mr. Manoj Pawaiya Sir

Submitted By:

Hritik Sahu (18C7120)
Muskan Sharma (18C7130)
Vikalp Rusia (18C7163)

Department of Computer Engineering

Institute of Engineering & Technology

Devi Ahilya Vishwavidyalaya, Indore (M.P.)

(www.iet.dauniv.ac.in)

October 2021

Table of Contents

Chapter-1 Introduction

1.1 Overview

This document is the software design specification for the Minesweeper project. The team is composed of undergraduate students majoring in Computer Science at Institute of Engineering and Technology, DAVV. The team members are Hritik Sahu, Muskan Sharma and Vikalp Rusia. Successful delivery of the software product will fulfil the minor project requirement for the student team members.

1.2 Problem Definition

Minesweeper is a simple one−player computer game commonly found on machines with popular operating systems such as Linux or Microsoft Windows. The game consists of a 2-dimensional rectangular playing field (or board) where some known numbers of mines have been hidden. Initially, all the squares on the board are "covered up" and no information is given to indicate the location of the mines. The player's job is to either deduce or guess which board squares are clear of mines and step there to reveal a number. If successful, this number will indicate the number of mines to be found in the squares adjacent to the square with the number. Obviously, the first move of the game must be a guess because no information has been provided. Since the board is a rectangular grid, each interior square has exactly 8 neighbouring squares, edge squares have 5 neighbours, and corner squares have 3 neighbours. Therefore, the number found under any given square will be in the range of 0 to 8 (inclusive). Game play continues until the player has uncovered (or "stepped" on) each square that does not hide a mine, while avoiding all the mines. If the player can do this, they are considered to have won the game. However, if at any point the player attempts to uncover a square that contains a mine, the game immediately ends, and the player is said to have lost.

1.3 Proposed Solution

To improve upon the game, there are several features that could be added. For example, adjusting the program to ensure that a bomb could not be uncovered on the first tap. This could be done by generating the random row and column coordinates of the bomb locations in the gameboard array after the first touch had been recorded on the touch screen. An if statement could be used in the code to prevent a bomb from being randomly placed in the location that corresponds to a location that was already touched by the user on the screen. By generating the

location of the bombs after the first touch has been made by the user, there is no way that the user could touch a bomb on the first touch. Then, the remaining logic of the code could still be applied. Another recommendation for the game is that flags could be added to the game. this would involve adding a button board that could be used to select locations on the board that the user suspects to have a bomb. Then, the user could tap these locations and a shape could be printed to that desired location. An if statement could be used to allow for the user to tap the space again to remove the shape. These changes could be implemented with more time.

Chapter-2 Literature Survey

Early algorithms developed to solve Minesweeper focus on the deterministic deductions needed to uncover safe moves. Adamatzky modeled Minesweeper as a cellular automaton or CA [1]. The cell state is given two components. The first component indicates whether the cell is either covered, uncovered, or a mine. The second component, which is only available to the system if the cell is uncovered, is the number of mines adjacent to that cell. One limitation to the CA model is in the transition function which only accounts for two basic deductions in Minesweeper. However, its main weakness is its tendency to become stuck on configurations. Since the CA solver is deterministic, it never makes guesses or random moves, an occasionally necessary step in completing a game especially at harder difficulties. Adamatzky expressed the need for stochastic features necessary for future algorithms.

A single point algorithm was mentioned by Kasper Pederson [2], although it was known beforehand, specifically by the author of Programmer's Minesweeper or PGMS1. Although not mentioned here, section 5.2 will analyze the single point algorithm in PGMS in more detail. Nevertheless, Kasper's single point method determines safe moves deterministically by looking at individual squares, like CA. However, if no safe moves are discovered, a square is probed uniformly at random. This prevents the algorithm from becoming stuck. Nevertheless, this approach is still too naive which is underscored by the poor performance on harder difficulties where guessing becomes more prominent.

Kasper Pederson offers an alternative strategy called limited search. This method utilizes depth-first search and backtracking on a small zone of interest around uncertain squares, an idea suggested by Pe˜na and Wrobel [3]. Since deductions often involve constraints placed on squares that are not direct neighbors, this technique can deduce more safe moves than the single point counterpart. Furthermore, Pederson added probability estimations that allowed for "smart" guessing. By collecting solutions from the limited search, estimates could be made about the probability of mine locations. Squares determined to have a low probability of being a mine would then be chosen as smart guesses, an improvement from simply probing uniformly at random.

Chapter-3 Analysis

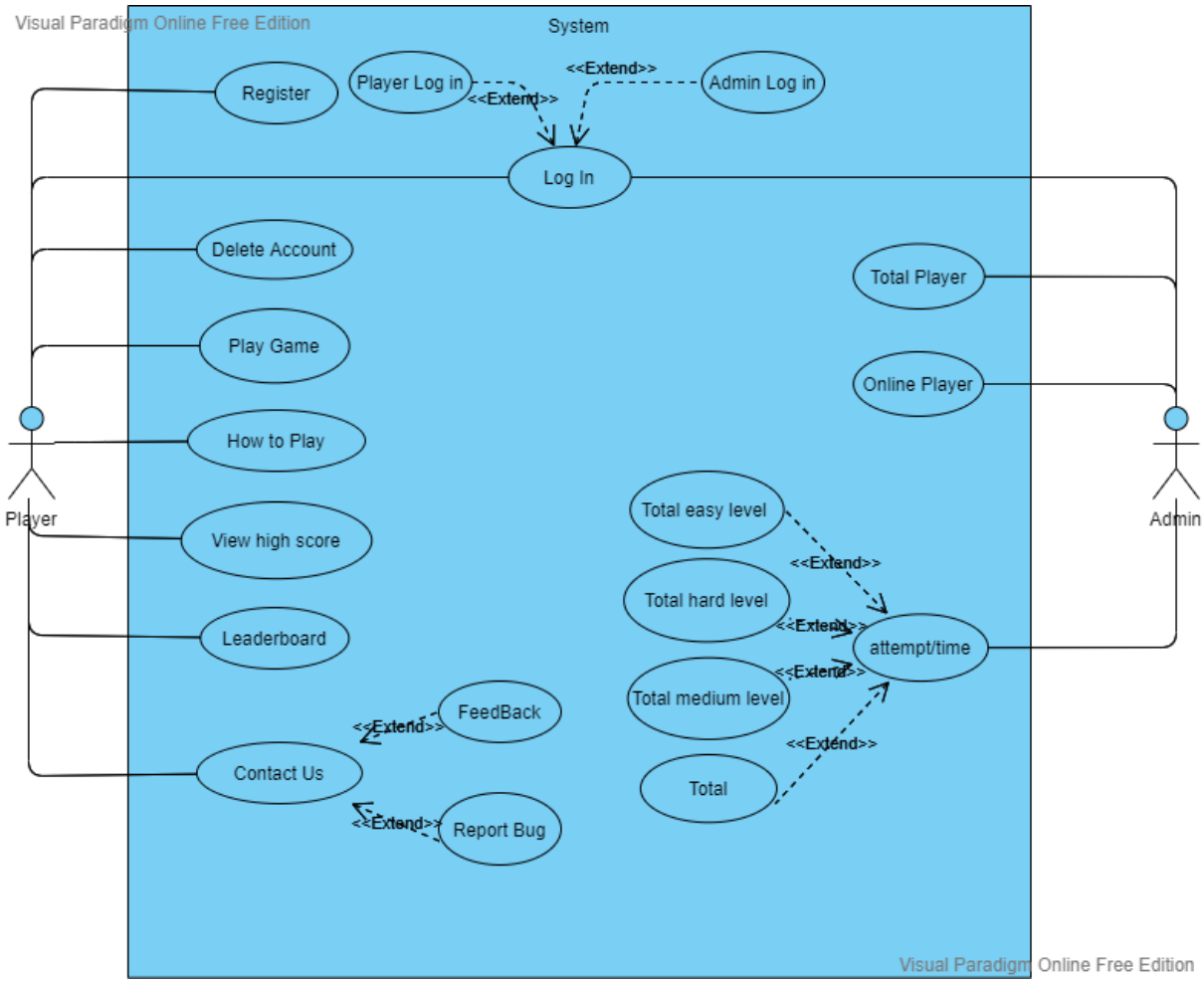
3.1 Software Requirements

Tool/Technologies	Description/Justification Operating
Operating System	Any modern OS can be chosen for development of Minesweeper Game.
MySQL	MySQL is to be used as a database to save the login credentials of a user and their respective data.
JDBC	JDBC makes it possible to establish a connection with a data source and extract data from a database.
Hibernate	Primary feature of hibernate is mapping from Java classes to database tables, and mapping from Java data types to SQL data types.
HTML, CSS, JavaScript	Front-End of application is to be developed using HTML, CSS and JavaScript.
Java	To implement the project, Java was chosen as a platform for its more interactive community and lots of libraries.
IntelliJ Idea	IntelliJ IDE is to be used as the main development tool to develop this project.
Web Browser	Minesweeper Game can be developed and tested with any of the modern web browsers.

3.2 Hardware Requirements -

Hardware	Justification
Computer	For running web browser
Server	For deploying web app

3.3 Use Case Model



3.4 Use Case Description

Use case description of each use case in the use case model must be there and it must be properly numbered and according to the format. In a particular use case description don't simply write the Preconditions, Post conditions and Special Requirements as none before analysing the particular use case in detail.

1. Register -

   Precondition - Server should be running and the browser can connect properly.

   Postcondition - Browser can redirect users.

   It registers users.

2. Log In -

   Precondition - User should be registered for Log In.

   Postcondition - Browser can redirect users.

   It logs-in users.

3. Delete account -

   Precondition - User should be Logged In.

   Postcondition - Browser can store a token.

   It deletes users.

4. Play Game -

   Precondition - User should be Logged In.

   Postcondition - Browser can redirect users.

   Users can play games having 3 levels - EASY, MEDIUM, HARD.

5. How to play-

   Precondition - User should be Logged In.

   Post condition - None

   It teaches users to play minesweeper.

6. View High Score -

   Precondition - User should be Logged In.

   Postcondition - Browser can render HTML.

   It displays high Score.

7. Leader board -

   Precondition - User should be Logged In.

   Postcondition - Browser can redirect users.

   It displays user High score.

8. Contact us -

Precondition - User should be Logged In.

Postcondition - Browser can redirect users.

It allows users to communicate with us.

9. Total player -

Precondition - User should be Logged In as admin.

Postcondition - None.

It displays all users.

10. Online player -

Precondition - User should be Logged In as admin.

Postcondition - None.

It displays currently active users.

11. Attempt/Time -

Precondition - User should be logged In as admin

Post condition - None

It displays all time and attempt time.

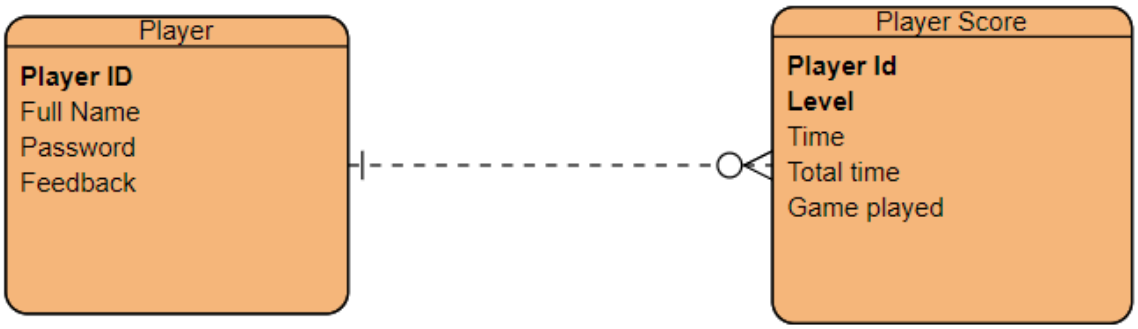
Chapter-4 Design

4.1 Technology Selection

- Java- To implement the project, Java was chosen as a platform for its more interactive community and lots of libraries.
- MySQL- It is to be used as a database to save the login credentials of a user and their respective data. JDBC makes it possible to establish a connection with a data source and extract data from a database.
- Hibernate- The primary feature of hibernate is mapping from Java classes to database tables, and mapping from Java data types to SQL data types.
- HTML, CSS, JavaScript- Front-End of application is to be developed using HTML, CSS and JavaScript.

4.2 Packaging

4.3 Database Design



Chapter – 5 Conclusion

During the process of creating the game, there were several obstacles that were encountered. For example, there was difficulty in developing a manner in which all of the surrounding locations of a selected location could be checked for mines. This problem was solved by making an if-else selection structure that could check the values of the surrounding locations in the array that represented the game board. This was done by incrementing the indices of the array by different values depending on the location on the screen. For example, the corner locations each had a different set of locations to check. The top right corner has to check the location below, below to the left, and the location to the left.

Another difficulty that was encountered was developing a method to win. This was done by changing the values in the game board array that correspond to touched locations to equal two. Then, after each touch, the components of the array are summed and compared to the value of the array if all the non-bomb locations had been touched, changed to a value of two, and summed with the value of the bomb locations. Once the sum is reached, the user has cleared the minefield and won the game. This solution took planning and trial by error.

The final difficulty that was encountered was difficulty in testing the game because the mines were randomly generated. To solve this problem, the randomly generated mines were commented out and bombs were manually placed in certain locations. This allowed for the testing of the game. In terms of things that worked well, the for loops used to draw the boards worked well and made symmetric boards. Additionally, once the method for checking the surrounding locations of a selected location was developed it worked well in printing the number of surrounding bombs.

Bibliography

[1] Andrew Adamatzky. How cellular automaton plays minesweeper. Applied Mathematics and Computation, 85(2–3):127–137, 1997.

[2] Kasper Pedersen. The complexity of minesweeper and strategies for game playing. Project report, univ. Warwick, 2004.

[3] Lourdes Pe˜na Castillo and Stefan Wrobel. Learning minesweeper with multi relational learning. In IJCAI, pages 533–538, August 2003.