Software Requirement Specification

1. Introduction:

This document is the software requirements 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.

2. System Overview:

When the program is launched, a board is randomly generated, and a timer is initialized. Then a window is created containing that minefield and that timer as well as a button that allows the player to reset the board. The reset button re-randomizes the board, reset the timer, and hides the values of the board's squares.

To understand the gameplay, please refer to Wikipedia's description of the mechanics of Minesweeper as linked to above.

This implementation of the Minesweeper game also contains high score functionality, where at the end of the game the player is prompted for their name, and it gets added to a list of the fastest completion of the game.

3. System Architecture:

Architectural Design-
In this program, most of the functions and data encapsulation are held within the monolithic MinesweeperGame class. It contains almost all the window GUI functions and the functions that initialize the HighScores and Board class as described below.

The actual board mechanics of the Minesweeper game is held in the board class, separating it entirely from the GUI components held within the MinesweeperGame class. The Board class generates a two-dimensional grid of values that either contain a value corresponding to a mine or an empty value. These values are assigned randomly, with only 10 of the grid values being assigned mine values.

The HighScores class contains a list of the times at the end of each game held within the game's timer, listed from lowest time to highest time. The HighScores class, being mostly modular and independent from the mechanics of the Minesweeper game, creates its own GUI components to prompt the user for their name and to display the values held in the list. The HighScores class is initialized with the launch of the program, and from that point on there exists a window containing the top ten times but is kept hidden until prompted to show itself by the menus of the MinesweeperGame class. Scores are also held in a database so that the values stay persistent across occasions that the program is run.

Data Decomposition-

MinesweeperGame contains several JavaFX GUI elements, brought together by a grid layout. In fact, MinesweeperGame itself is an extension of the JavaFX Stage object. There is an instance of the Board class that corresponds to the game's minefield. Separate from the Board class, there is a two-dimensional array of integers that contains the number of mines adjacent to each cell in the Board class. It also contains a Timer object for keeping time. In addition, it has a BST that caches the values of the HighScores instance contained within MinesweeperGame and after closing the game it moves to Database.

The Board class is fairly simple, only containing a two-dimensional array of integers in which 0 corresponds to a square without a mine and 1-4 corresponds to a square with a mine.

Functional Decomposition-

- **MinesweeperGame**:
  - There are a series of public methods that establish behaviour of each button, the behaviour varying depending on the mouse button used.
  - A public flood fill method that sets the icon for a space depending on the number of adjacent mines or if the space has a mine.
  - A public method that stops the timer and shows all the mines at the end of a game.
  - A public method that checks to see if the player has won the game. (Ninety squares are revealed and none of them are mines.)
- **Board**:
  - A public method that fills the board with zeroes.
  - Randomly sets ten of the spaces to have mines.
  - A public method setting the number of mines around a space.
  - A public method that returns the two-dimensional array that holds the board's grid.
- **HighScores**:
  - A public method that writes to the file holding the persistent list of scores.
  - A public method that displays the top 10 scores in a separate JavaFX instance.
  - A public method that makes the window displaying the top 10 scores visible.
  - A public method that traverses the score list data and creates a list from a binary search tree.

4. Literature Survey [2]-

- **Using Games to Teach Cognitive Skills** The use of games to attempt to teach cognitive skills goes back to ancient times. It is thought that the game now known as "Go," is possibly one of the first games that were used to teach mental ability. Chess also has a history of being used to teach thinking, one legend suggests that the philosopher Phallometer created the game in the late Middle

Ages to teach the king to live a virtuous life (Adams, 2006). While Chess likely existed before the time of this legend, the legends about Chess and Go both show that people have believed that these games had the ability to help a person learn things beyond the game itself. Several studies with chess suggest that learning chess can help with math ability, although a study by Thompson (2003), which controlled for IQ showed no scholastic improvement. Further, research by Horgan and Morgan (1988), suggests that spatial abilities are more correlated to chess playing abilities than logical abilities. The game Minesweeper, and its variants, in contrast to many of the other games studied for cognitive effects, is recognized to utilize direct logical thinking.

- **The Value of Logical Hypothetical Thinking** -Wilson and Conyers suggest that Hypothetical Thinking is a "Cognitive Asset," which they define as skills that are related to thinking which are of extraordinary value. Wilson and Conyers have defined 26 such assets that they believe are important for students to gain executive intelligence. Wilson and Conyers define the cognitive asset of Making Inferences/Hypothetical Thinking as "The ability to solve 9 problems and create new information by making inferences based on the information given. Hypothetical reasoning is a central topic in cognitive science and is a major component of what is known in Dual Processing Theory as Type 2 Processing done by our brains. Type 2 processing is what allows us to generally analyse possibilities and make rational decisions, compared to what is usually our default thinking, which is called Type 1 processing. Type 1 processing is fast and automatic, but often leads to wrong conclusions. Research is showing that instrumental rationality, defined as behaving in the world so you get what you want given your resources, is reliant upon Type 2 processing, and having good logic skills combined with specific types of thinking dispositions.

4. Data Design:

1. The game's board data is held in a two-dimensional array of integers.
2. The high score data is contained in a binary search tree but maybe exported to an Array List and database as needed.

The MinesweeperGame class is the container for both classes that contain these two data structures and manipulates them accordingly.

5. Component Design:

- **MinesweeperGame**:

The function that determines whether the game has been won determines how many squares have been revealed. If 90 squares have been revealed and not a single mine has been uncovered, then the game assumes that the player has successfully completed the board.

- **Board**:

The mines are set using a brute force function that uses a random number generator to generate to integers in the range of 0-8 and checks to see if a space has been filled with those coordinates. If it hasn't been designated a mined space, it is assigned a mine. If it already has a mine, the function tries again with a new set of coordinates. This is done until there are 10 spaces designated to have mines.
The function that cycles through the game board and determines the number of adjacent squares containing mines simply go through each using two linear loops corresponding to each dimension of the array. Then, for each space, the function takes care to not overrun the bounds of the arrays and circles around each space to determine the number of mines adjacent to the present working space, incrementing as it goes and assigning that incremented integer when it is done.

- **HighScores**:

The top 10 list is derived from a binary search tree, getting the 10 leftmost elements in the tree retrieves the 10 highest scores.

6. Human Interface Design:

Overview of User Interface-
The user can interact with the grid of spaces by either left-clicking or right-clicking the spaces, with different responses for each of those inputs.

There is also a button to reset the game and start a new one.

There are no prompts for the user other than an pop-up box that allows the player to enter their name if their score was sufficient.

The menu bar at the top of the window allows the player to reset the game, close the game, read the top ten completion times, read about the creators of the program, or read about how to play the game.

Screen Objects and Actions-
The window is made visible as soon as the program is loaded. There is a grid of buttons 10 columns by 10 rows. There is also a timer, a counter for the remaining mines, and a button that allows the game to be reset. The icons for the buttons can either be blank, have a left-clicked mine, a right-clicked mine, a question mark, or the number of adjacent mines depending on the context.

The menu bar at the top of the window has options to close the window and end the program, an option to view the top ten players, an option to reset the game and start a new one, an option to view information about who created the game, and an option to view how to play the game.

The high scores menu has a column for each of the top players' names and a corresponding column for their scores.

7. Requirements Matrix:

Requirement	**Fulfilment**
Implements Minesweeper in Java.	Java was indeed used to create a 10x10 game of Minesweeper.
Allows for the user to click with the left and right buttons.	The program detects the mouse input and has the appropriate distinct functionality for both.
The window for the game must have a reset button, a field displaying how many mines are left to be found by the player, and a field displaying the seconds elapsed since the game started.	There is a functional JavaFX for resetting the game, a Label displaying how many mines are left, and a Label with the time elapsed as determined by a Java Timer object.
A menu bar with two drop down menus, Game and Help. The Game drop down menu having a Reset item, a Top Ten item, and an Exit item. And the Help drop down menu having a Help item and an About item.	Menu Bars, Menus, and Menu Items were used to implement this requirement with the appropriate functionality.
Displays the top ten lowest times in a window when prompted by the user.	Retrieves values from the HighScores class and displays them in a window.
Persistently stores all the time that the player has completed the game in.	The HighScores class reads and writes to a text file in order to keep the scores persistent.

8. References:

- https://en.wikipedia.org/wiki/Minesweeper_(video_game)
- Minesweeper as a Constraint Satisfaction Problem
- Minesweeper and Hypothetical Thinking Action Research & Pilot Study (ed.gov)
- Wikipedia
- Playing Minesweeper
- Learn Playing Minesweeper