Software Requirement Specification

**CONTENTS**

1. Introduction…………………………………………….. 2
2. Rules and Basics……………………………………….. 2
3. System requirements……………………………………. 3
4. Graphic library………………………………………….. 3
5. Basic algorithm…………………………………………. 4

**Team members (Group 13)**

1. Pratyarth Rao (140100019) (Team Leader)
2. Shubham Bansal (140100059)
3. Sunny Soni (140100054)
4. Soham Shanbhag (140100014)

**1. Introduction**

**Objective:**

The objective is to develop the classic game of minesweeper using C++ as the basic programming language and a suitable graphics library.

**About The Game:**

Minesweeper is a single player puzzle-type video game. The objective of the game is to clear the “field”(gameboard) of “mines” without detonating any of them. (i.e clicking on them).

The player is first presented with a grid containing a number of indistinguishable squares containing a number of pre-specified mines.

The number of mines and the size of the grid is decided by the user by selecting a particular difficulty level, according to which a grid of specified size and number of mines is displayed(the program developer generally prespecifies the grid size for each difficulty level).

To play the game, the player has to click on a square in the grid. The square maybe:

1. Blank: has no mine adjacent to it

2. Containing a number: Hints at the number of mines adjacent to it

3. Containing a mine: If the player clicks on this, the game is over.

The player can place flags on squares in which he thinks

To win the game, the player has to correctly “flag” the positions of all the mines on the board.

**2. Rules and Basics**

The objective of the game is to uncover all the spaces not containing mines. The faster the board is cleared, higher is the score.

The game consists of three difficulty levels

1. Easy

2. Medium

3. Hard

**Rules Of The Game**

## The rules are simple

* A square having a number shows the number of mines in the eight squares around it.
* Uncovering a mine ends the game.
* When all squares except those containing mines are uncovered, the user wins.

## How To Play

1. Click on the box to uncover it.

2. If you have a doubt regarding a square to be containing a mine, flag it.

## Hints

• An 8 indicates that all the squares around it are mined.

• Notice patterns. 2-3-2 implies 3 consecutive squares are mined.

• When in doubt, use flags to mark the cells.

• When no clue is found, try an unexpected region

**3. System Requirements**

The game has minimal system requirements and can be played even on computers with the most basic configuration.

* The game runs on multiple platforms such as Linux/Windows(XP and above)/Mac OS.
* A C++ compiler (such as Codeblocks, DevC++) with Allegro graphics library is required to run the game.
* The game does not have any specific processor or hardware requirements and can run even on primitive systems.

**4. Graphic Library**

We have decided to use the Allegro graphics library for C++ to aid in integrating graphics with C++ code.

Allegro is a library which was designed specifically to aid in game development and is a very versatile library in terms of different commands and functions available, hence it was the preferred choice in developing the game of minesweeper.

**5. Functions used:**

We have included the functions in the main program as header files:

Main file: Minesweeper.cpp

1. Credits.h

int credits: Opens up the screen which displays the credits. Initiated by clicking on the credits option in the main screen.

1. Endgame.h

int win: Displays the winning screen. It is called when the condition for winning is satisfied.

int lose: Displays the “game over” screen. It is called when a mine is clicked.

1. Highscores.h

int highscore: Displays the screen for high score. Allows the user to input his name and records the time taken to complete the game and assigns a corresponding score. Writes the name taken from the user to a text file (separate for the three different modes of the game) and the time taken in a different text file (separate for the three different modes of the game)

1. Timerec.h

void rectimeeasy: Is called when the game is completed in easy mode. It checks the current score with the high scores for easy mode. If a new high score is created it will update the new highscore with the corresponding recorded time.

void rectimemedium: (Similar functions as rectimeeasy, called when game is played in medium mode)

void rectimedifficult: (Similar functions as rectimeeasy, called when game is played in medium mode)

1. Workingeasy.h

int gridcreateeasy: Calls the other functions to initialise grid creation.(easy mode)

void allocate\_grid: Arranges mines in the grid and sets values of the other elements of array with respect to position of mines.

int analyse\_click: Checks the data in the box which is clicked and performs corresponding action, by calling function . Possible data values: empty box, box with a number, box with a mine

int blank\_box\_click:

Is called when the clicked box is blank and performs corresponding action.

int win\_check:

Checks whether the current state of the game can be considered as a winning condition.

int draweasy:

Draws a 8\*8 grid on which the game will be played. Also includes the gameloop for easy mode. Also displays all the elements on the screen.

6. workingmedium.h:

7. workingdifficult.h

(These files have functions corresponding to the medium and difficult mode of the game which are similar to the functions in the header file workingeasy.h)

**6. Basic Algorithm**

* Create a grid by initialising a 2d array which will contain information about the position of mines.
* Specified number of mines are assigned random positions in the 2d array.
* The 2d array is searched for the mines and value of nearest cells (excluding mines) are increased by one.
* This will create the final grid and now the game can be commenced.

**Game mode**

* Mouse input from the user is analysed to get the position of the box which is clicked.
* If the selected box contains mines, the game is terminated.
* If the selected box contains no., it becomes visible.
* If the selected box is blank, all adjacent (in horizontal and vertical direction) blank boxes become visible along with that box. Further this process is repeated for every blank box made visible till a box containing number is encountered and the box containing number is also made visible.
* All doubtful cells can be flagged.