Bomb Sweeper
Game Design

Version 1.0
October 30, 2013

Table of Contents

1. Introduction

This document gives a precise detail of the program such as the game structure of BombSweeper. It also deals with how all of producers' knowledge about Ready to Program Java has been putted in to create the program.

1.1 Scope

This document has been created to be read by programmers, produces involved in the design, and players who are interested to know about the structure of BombSweeper.

1.2 Tasks

Tim Baek	Ioan Tataru
- Created Design Document - Algorithm - Debugging and adding extra fillers	- Coding - Algorithm - Debugging and Testing

2. References

1. **Zet Code Mine Sweepr**
   October 30, 2013 – IT Design Ltd
2. **StackOverflow Java**
   October 30, 2013 – IT Design Ltd
3. **MineSweeper**
   November 1, 2013 – IT Design Ltd

3. Development System

3.1 Software

Bombsweeper will be using Ready to Program Java.
The Windows compatibility will be XP/Vista/7

4. Specification

4.1 Concept

The goal of Bomb Sweeper is to allow the players to have fun and to think critically by locating where the bombs are with the given hints.

4.2 Objective

The objective of the program is to find and detonate all bombs with the given hints.

4.3 Levels

The program will consist of 3 levels: easy, medium, hard

4.3.1 Easy

Grid Size: 10 x 10
Points: 1000
# of Bombs: 10% of Grid Size (10 Bombs)

4.3.2 Medium

Grid Size: 15 x 15
Points: 2000
# of Bombs: 20% of Grid Size (45 Bombs)

4.3.3 Hard

Grid Size: 20 x 20
Points: 3000
# of Bombs: 30% of Grid Size (120 Bombs)

4.4 Grid Generation

In BombSweeper, there are two 2D arrays that are determined by the height and length of the grid. One will be used to display the grid for the players to see, and the other will be used for a hidden grid to know where the bombs are without the players knowing of it. Also, the grid size is determined by the difficulty of levels.

4.5 Players

BombSweeper will be a single player game that is against an AI or it can be played as co-op with assistance from a different player

5. Game Play

5.1 Interface

Menu: Play Game, Instruction, Scores with Names, and Quit Game.

5.2 Control

Players will have 2 options: they can either click on a square to check if there's a bomb or not, or they can flag the square if they think there's a bomb. When a player clicks on a square, it will output a number which tells the number of bombs nearby, and if there are no bombs, it will open nearby squares until the program finds the squares that are near the bombs and output the hint.

5.2 Timer

BombSweeper will have a timer for two purposes: one is to show how long the player is taking to complete the game, and the other is to use for scoring system.

5.4 Scores

Scoring will be determined by the amount of points and time. Players will begin with a certain amount of points which vary according to the difficulty of level, and 1 point will decrease per second. If the scores become negative, after the game finishes, it will output 0.

5.5 Saving File

During the game, if players desire to save the file, they can by going into a different interface which the program will ask the players whether to save the file or not. When the answer is yes, the program will create and write to the new file in a folder, which can be read or accessed when the players want to continue the game in the future.

6. Front End

6.1 Intro

A menu will pop up which will direct to sections where the players want

6.2 Name Saver

In the end, the game will ask the players to enter their name which will be saved into the program with their name and the scores they earned.

7. Algorithm

7.1 Game Setup and Screen

1. Declare a global integer variable height.
2. Declare a global integer variable length.
3. Declare an integer nummine.
4. Ask the user the level he want to play (e/m/h) using string variable called level.
5. Start a while loop (while the level input is wrong)
    a. Print invalid input.
    b. Ask for the level again
6. Determine the height and length of the board based on the level.
7. Determine the number of mines based on the level.
8. Initialize an integer based grid of size [height][length].
9. Initialize a string based grid of size [height][length].
10. Call a void method InitializeDisplayGrid which takes as input a grid.
    a. The method sets all the values of the string grid to "X " and outputs it using 2 for loops.
11. Call a void method InitializeMine which takes as input nummine and an integer based grid.
    a. The method has a counter=0.
    b. In a while loop (while counter<nummine)
        i. Generates a random integer in range for the height
        ii. Generates a random integer in range for the length
        iii. If (height, length) is not already -1
            1. It sets (height, length)=-1
            2. It adds 1 to the counter
12. Call a void method Countmine which takes as input an integer grid
    a. Start two sets of for loops to check each square of the grid
        i. If a square is=-1 (mine)
            1. Determine its place (center, one of 4 corners or one of 4 sides not including corners)
                a. If adjacent square is not =-1 (not a mine)
                    i. Add 1 to it (since that square is adjacent to a mine

7.2 Game Play and Interface

1. Declare a global Boolean integer finish
2. Declare integer variables, len, hei and displayed (global)
3. Declare a Boolean variable click
4. In a do loop (while finish==false and displayed<height*length-nummine)

a. Call a void method Click
    i. In a do loop (while the input is invalid)
        1. This method gets input on whether the user whishes to click or flag
    ii. returns true if user wants to click
    iii. returns false otherwise
b. If click==false (user want s to flag)
    i. Get the length coordinate by calling a method that validates the input to make sure it's in range
    ii. Get the height coordinate by calling a method that validates the input to make sure it's in range
    iii. Change that specific square to an "F " in the String grid
    iv. Display the String grid by calling in a method that displays a grid
c. Else
    i. Get the length coordinate by calling a method that validates the input to make sure it's in range
    ii. Get the height coordinate by calling a method that validates the input to make sure it's in range
    iii. If that coordinate==-1 (mine)
        1. Print you lose
        2. finish=true
    iv. Else if (the specified square ==0)
        1. Call a void method RecursiveEmpty which takes as input an integer grid, a string grid, hei and len
            a. The method turns the specified square to a 0 in the string grid
            b. Displayed++;
            c. Then check where is that square (9 cases)
                i. If an adjacent square is 0
                    1. The method turns the adjacent square to a 0 in the string grid
                    2. Displayed++;
                    3. The method is re-invoked recursively
                ii. Else
                    1. It displays puts the value from the integer grid to the string grid
                    2. Displayed++;
        2. The String grid is displayed
    v. Else (square is a number from 1 to 8)
        1. It displays puts the value from the integer grid to the string grid
        2. Displayed++;
        3. The String grid is displayed

7.3 File Handling

1. When saving to a file, use the variable level to write the letter of the level on the first line
2. Start a set of double for loops
    a. Store line by line, the integers from the integer based grid
3. Start a set of double for loops
    a. Store line by line, the strings from the strings based grid
4. When reading from the file (loading a game), first read the first line.
5. Sets of conditions to determine the size of the 2 grids based on the letter (e/m/h)
6. Start a set of double for loops
    a. Read line by line, the integers from the file and store it in an integer based grid
7. Start a set of double for loops
    a. Read line by line, the Strings from the file and store it in a string based grid

7.4 Solution

1. If Boolean lose==true
    a. Start a set of double for loops
        i. Check if what you are about to output is a -1
            1. Output a M (for mine instead)
        ii. Else
            1. Output the integer based grid
2. Else
    a. Ask the user if he want to store his result
    b. If yes
        i. Create a .txt document called results that stores its score

7.5 Testing

1. I will test each method separately first
2. To test if the grid generation is correct, output a couple and see if numbers make sense (represent the # of mines)
3. Then, play the game while outputting the integer grid and click on different numbers (0s, 1 to 8, and -1s) to test
4. Save a game a couple of times and reload it (win and lose after)
5. When winning, try saving results
6. When losing, see if the outputted solution makes sense (same as integer based grid)
7. Enter unexpected values (out of bounds, different type than expected, etc.)

8. Project Plan

8.1 Deliverables

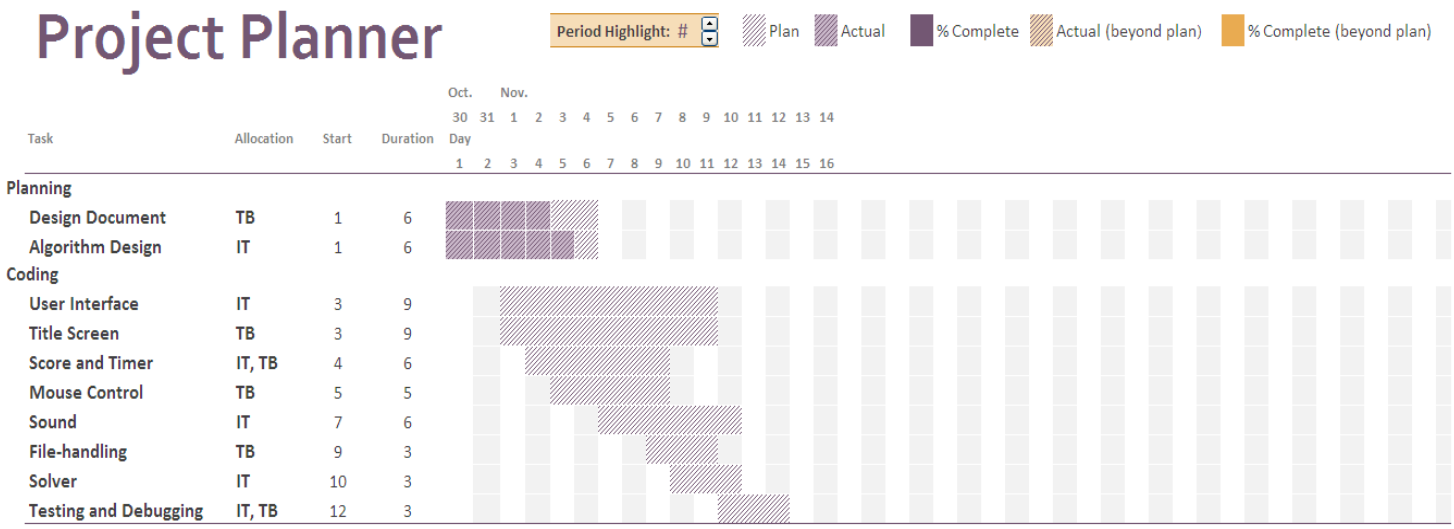
8.1.1 Tangible Deliverables

Tangible deliverables include project plans, the program BombSweeper, Game Structure, grid, and much more.

8.1.2 Intangible Deliverables

Intangible deliverables include the knowledge of programming, time, and sources of information about Mine Sweeper.

8.2 GANTT Chart



TB = Tim Baek, IT= Ioan Tataru

9. Team

Project Manger	: Tim Baek
Programming	: Ioan Tataru
Art	: Tim Baek
Design	: Ioan Tataru
Producer	: Tim Baek, Ioan Tataru

10. Time

Official Start Date:	October 30, 2013
Complete Game Design:	November 5 2013
Algorithm Completion:	November 6, 2013
Programming Completion:	November 14, 2013