Software Requirements Specification

for

Minesweeper

Version 1.0 approved

Prepared by Dominic Burke,

Joseph Hansen,

Julie Johannes-Frohliger,

and Abdinasir Muhumed

23512 Team 1

April 15, 2019

Table of Contents

Table of Contents ii

Revision History iii

1. Introduction 1

1.1 Purpose 1

1.2 References 1

2. Overall Description 1

2.1 User Classes and Characteristics 1

2.2 Operating Environment 1

2.3 Design and Implementation Constraints 1

2.4 Assumptions and Dependencies 1

3. External Interface Requirements 1

3.1 User Interfaces 1

3.2 Hardware Interfaces 2

3.3 Software Interfaces 2

3.4 Communications Interfaces 2

4. System Use Cases 2

4.1 Use case name and identifier 2

4.2 Withdraw money from ATM (U2) 3

4.3 Deposit money into ATM (U3) 4

5. Other Nonfunctional Requirements 4

5.1 Performance Requirements 4

5.2 Safety Requirements 4

5.3 Security Requirements 4

5.4 Software Quality Attributes 4

6. Other Requirements 4

7. System Requirements Chart 4

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Reason For Changes** | **Version** |
| Nasir | Feb 26, 2019 | Added the dynamic grid code | 1.0 |
| Julie | Mar 3, 2019 | Added the class structure | 1.1 |
| Joe | Mar 3, 2019 | Added the PopulateCells() methods | 1.2 |
| Joe | Mar 3, 2019 | Added the code to create buttons on the grid | 1.3 |
| Joe | Mar 4, 2019 | Added the CreateValue() method that calculates the value of each cell in relation to the mine locations | 1.4 |
| Nasir | Mar 6, 2019 | Added the dynamic click event handler | 1.5 |
| Julie | Mar 9, 2019 | Added game logic framework for when a button is clicked: I.E. parse the name to get the cell array location, if/else statements depending on the cellValue property, returns the cell displayValue. Currently located in ‘GameLogic’ class | 1.6 |
| Julie | Mar 9, 2019 | Added logic to parse button name for cell array coordinates, returns cell display value | 1.7 |
| Dominic | Mar 10, 2019 | Added a GameLogic constructor and validation for user input | 1.8 |
| Julie | Mar 10, 2019 | Added call to ButtonLeftClicked method in GameLogic class | 1.9 |
| Joe | Mar 11, 2019 | Added spread function and changed x and y in the array to more accurately reflect the display | 2.0 |
| Julie | Mar 17, 2019 | Changed button naming convention to btn\_x\_y to allow for double digit location values. Added underscores to the string list for returning buttons to differentiate the values | 2.0.1 |
| Nasir | Mar 25, 2019 | Added the animation to disabled cells | 2.1 |
| Julie | Mar 30, 2019 | Added windowGrid, gameboard grid to windowGrid, and status barGrid with flag counter label. Added right-click button, disable button events on left click, and tagged property to a cell object rather than changing the cell value to -1 | 2.2 |
| Julie | Mar 30, 2019 | Debug fixes | 2.2.1 |
| Julie | Mar 30, 2019 | Trouble shot flag counter label, changed background to green | 2.2.2 |
| Julie | Mar 30, 2019 | Added Timer and Thread to update the timer every second | 2.3 |
| Joe | Mar 31, 2019 | Edited the button enabling status on the left click event; added all effected cells from the spread function to the returned list; changed window title | 2.4 |
| Nasir | Mar 31, 2019 | Changed the window so it resizes itself to the number of cells that exist | 2.5 |
| Dominic | Apr 1, 2019 | Added win game functionality and navigation menu | 2.6 |
| Joe | Apr 1, 2019 | Added icon to the window’s header, minor clean up of code changes | 2.7 |
| Dominic | Apr 1, 2019 | Added post game menu, check score button, sound folder and cleaned visuals | 2.8 |
| Joe | Apr 7, 2019 | Added flag.png to icon folder and code to change the background of flagged cells | 2.9 |
| Dominic | Apr 8, 2019 | Added beginning of navigation bar | 3.0 |
| Joe | Apr 8, 2019 | Corrected the path to the flag image for the right click, changed the ButtonRightClicked method to return a Boolean and took out the display of the letter ‘F’ for flagged cells | 3.1 |
| Joe | Apr 8, 2019 | Cleaned up code and syntax and removed and added comments in certain places | 3.1.1 |
| Dominic | Apr 14, 2019 | Added navigation buttons below game board, created click events | 3.2 |
| Joe | Apr 15, 2019 | Changed the background color of grid so when a button is flagged it is the same color as the other buttons | 3.3 |
| Joe | Apr 21, 2019 | Added an EndGame() method to the MainWindow that consists of a Reveal() method and added a shell for the CalculateScore() method, cleaned up unused code, implemented a way to determine the game is over when the user clicks a mine | 3.4 |
| Joe | Apr 22, 2019 | Added bomb.png | 3.4.1 |
| Julie | Apr 27, 2016 | Added empty btn\_unlick method to re-assign as a button lick method after a button is ‘disabled’ | 3.5 |
| Dominic | Apr 27, 2019 | Added navbar, credits, rules, restart and grid size limit | 3.6 |
| Nasir | Apr 28, 2019 | Remove btn\_rightclick function at ‘disable’ and added the empty method | 3.7 |
| Joe | Apr 29, 2019 | Added a disabledButton() method used to simulate the disabling of a button where applicable, code to change the visuals to simulate disabled buttons, the bomb image to unflagged bombs at the end of the game, colored borders to cells that were flagged correctly, incorrectly or missed, calculates and displays score | 3.8 |
| Julie | May 1, 2019 | Added DispatcherTimer clock.Stop() function in 3 places to stop the timer at the end of the game | 3.9 |

# Introduction

## Purpose

Minesweeper is a computer game where the user chooses a number of columns across, rows down and number of mines. The user is then able to select cells either by left clicking to reveal the cell value or left clicking to flag as a mine. The goal of the game is to reveal all the cells that are not mines and flag all the mines without revealing a mine. If a mine is left-clicked the user loses the game. Cell values are based on how many mines touch the cell.

## References

None.

# Overall Description

## User Classes and Characteristics

None.

## Operating Environment

Windows XP or newer.

## Design and Implementation Constraints

None.

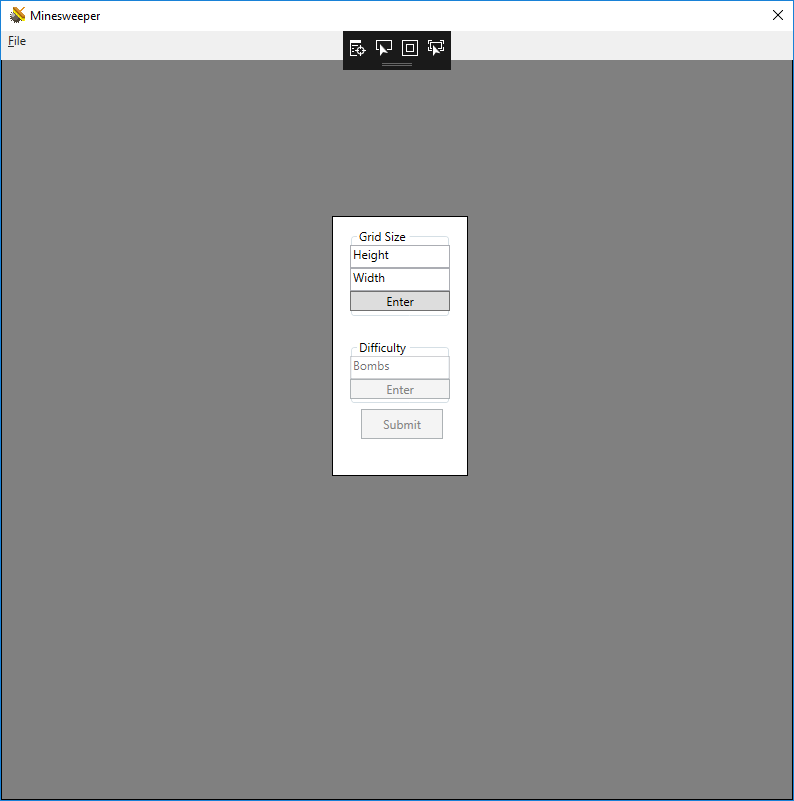
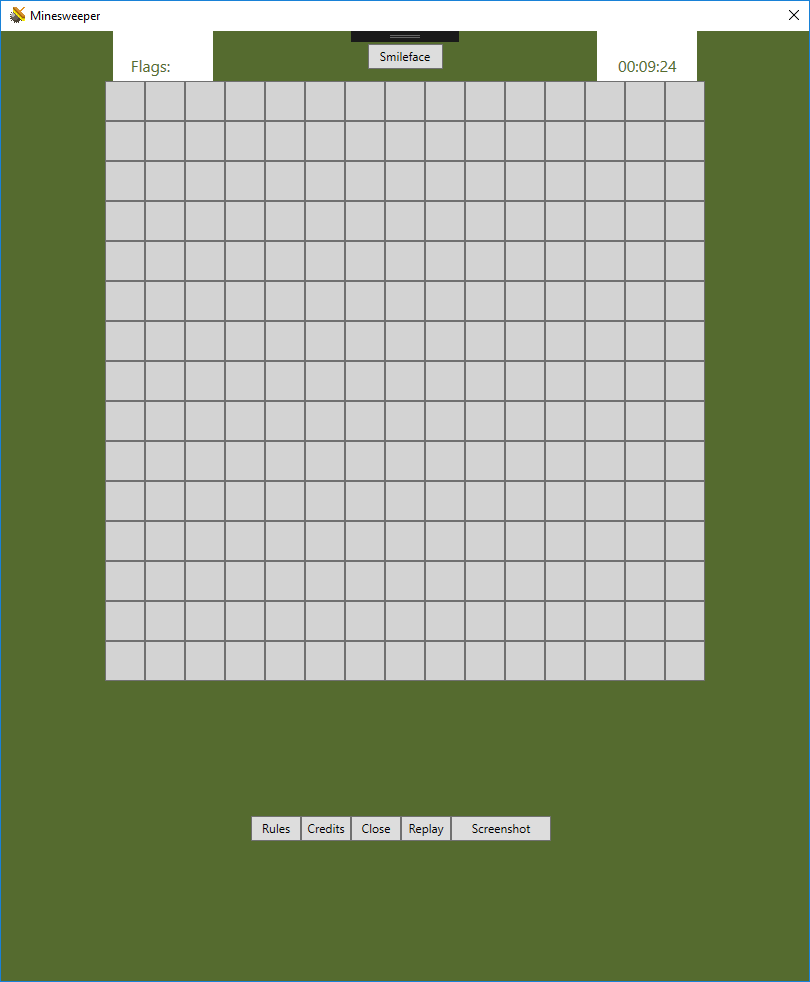
## Assumptions and Dependencies

None.

# External Interface Requirements

## User Interfaces

all clickable buttons will be grey with the default border

* all fonts will be Arial
* buttons “disabled” will be yellow
* mines have an image of a mine
* flagged buttons have an image of a flag
* Game Options user input form: includes number of columns across, rows down and number of mines
* Status Bar: includes the number of flags placed, the timer and the smiley face to check the board (ending the game)
* Game board Grid: includes the mine field with a button on each space. Reveals number of mines touching the space, the flag, if user flags it or the mine if applicable.
* Menu Bar: includes a button to see the rules, the credits, ability to Close the game, Replay the game or take a screenshot of the current board status.

## Hardware Interfaces

None.

## Software Interfaces

Operating system of Windows XP or greater.

## Communications Interfaces

None.

# System Use Cases

The overall use case diagram should be here.

The text description of each use case should follow.

## Use case name and identifier

1. **Unique Identifier**
2. **Objective** - What is the ultimate objective of the use-case. What is it trying to achieve? What was the source of the use-case requirement?
3. **Priority** – The overall priority of this use-case (Low, Medium, High)
4. **Source** – Who is the main source of this use case. Who cares most about this functionality? This should be the one person you would ask if there is a question about this use-case. (Make up a name and cite their: John Smith (End-user) here.)
5. **Actors** - Who is involved in the use-case? Which actors/stakeholders?
6. **Flow of Events** 
   1. **Basic Flow** - flow of events normally executed in the use-case
   2. **Alternative Flow(s)** - a secondary flow of events due to infrequent conditions
   3. **Exception Flow(s)** - Exceptions that may happen during the execution of the use case
7. **Includes** - other use case IDs that are referenced in steps in the flow of events.
8. **Preconditions** - Any condition that must be satisfied before the use case begins. If the condition is “User is logged in”, then the first step of the use case is NOT “User logs in”. They are already logged in if that is a pre-condition!
9. **Post conditions** - The conditions that will be satisfied after the use case successfully completes
10. **Notes/Issues** - Any relevant notes or issues that need to be resolved

## Withdraw money from ATM (U2)

1. **U2**
2. **Objective** – The customer is withdrawing money from the ATM and the system will debit the customer’s account.
3. **Priority** – High
4. **Source** – Carl Gnome (marketing)
5. **Actors** – Customer, central bank computer
6. **Flow of Events** 
   1. **Basic Flow**
      1. Customer chooses the checking option on the ATM
      2. Customer chooses the amount of money needed
      3. Customer confirms the choice
      4. System validates the amount
      5. System asks central bank computer to debit the customer’s account
      6. System issues money to the user
   2. **Alternative Flow 1** – At step 5.1.4 the amount is not a multiple of $20
      1. An error message is displayed telling the customer they must use multiple of $20.
      2. Return to step 5.1.2
   3. **Alternative Flow 2** – At any step the user presses “cancel”
      1. System returns to the main menu
   4. **Alternative Flow 3** - At step 5.1.5 bank computer returns a failed status, “insufficient funds”
      1. An error message is shown to the user
      2. Return to step 5.1.2
   5. **Exception Flow 1** –
      1. Database is locked due to backup in progress. System executes use case U5
7. **Includes**
   1. U5 – Exception occurs
8. **Preconditions** – User is logged in
9. **Post conditions** – Money has been returned to the user and their account balance has been updated.
10. **Notes/Issues** - None

## Deposit money into ATM (U3)

….

# Other Nonfunctional Requirements

## Performance Requirements

**In this section, just say “See section 7 requirements 23-27”. And I’ll assume those requirements are Performance related.**

## Safety Requirements

**In this section, just say “See section 7 requirements 25-32”. And I’ll assume those requirements are Safety related.**

## Security Requirements

**In this section, just say “See section 7 requirements 35-42”. And I’ll assume those requirements are Security related.**

## Software Quality Attributes

**In this section, just say “See section 7 requirements 55-62”. And I’ll assume those requirements are Software Quality related.**

# Other Requirements

None.

# System Requirements Chart

*< Include a* ***table*** *in this section with the following columns:*

***ID*** *– Unique requirement ID*

***Priority*** *– Priority of this requirement*

***Type*** *– Functional(F) or Non-functional(NF)*

***Source*** *– Who is most interested in this requirement (John Smith – Customer). For this project you can make it up, in reality you’ll want to capture this as you capture the requirements.*

***Contained in Use Case(s****) – Which use cases reference this requirement or which use cases when executed will perform this requirement. There may be a few functional requirements without a use-case and the non-functional requirements generally will NOT be part of a use-case (so put N/A).*

***Description*** *– The description of the requirement. “The system shall …. “*

*>*

These requirements should match up with your use case diagrams.