3815ICT – Software Engineering

Minesweeper

Milestone 1 - Requirements

Student: Zak Barker

Student#: S5085150

Subject: 3815ICT Software Engineering

Email: Zak.Barker@Griffithuni.edu.au

1. Functional Requirements

1. Title Page and Navigation

Application will require a title page on which the user will land once the application has been loaded. Navigation will be required on the title page depending on how the user decides he/she want to play the game. There will be three variations on the minesweeper game including:

- Standard Minesweeper
- Hex Minesweeper
- Colour Minesweeper

The user will have the option to play either of those game states. Furthermore, the user will require access to information about the game including the rules and instructions for playing the game. As such, the game will require five pages; being:

- Title Page
- Standard Minesweeper Gameplay
- Hex Minesweeper Gameplay
- Colour Minesweeper Gameplay
- Instructions Page

The user will require an interface which will provide navigability between these five pages. There will also be a requirement for the user to return to the title page in the event that they want to play a different mode.

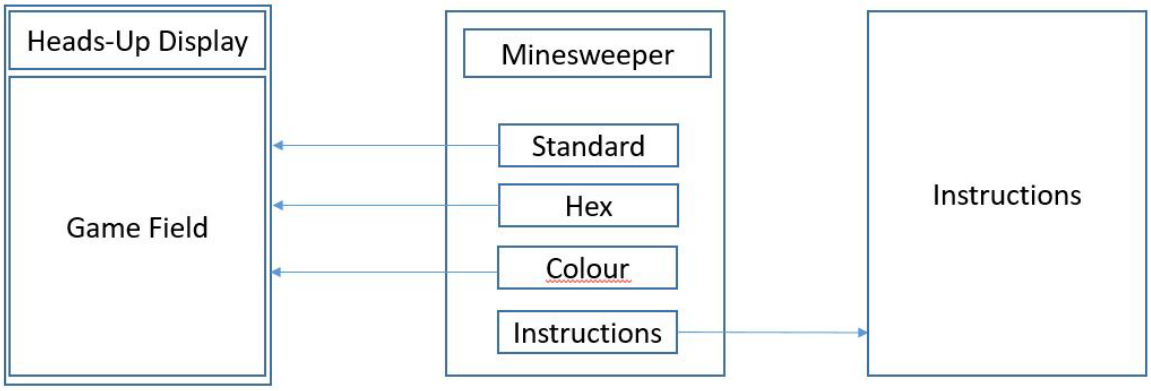

Figure 1 – Title Page & Navigation

2. Gameplay – Grid

The main gameplay component of the implementation of minesweeper will be the grid. This grid will be made up of intelligent cells which will hold data about their current state, their location within the grid and the state of each of their neighbouring cells. The grid size should be dynamically created based on user input. Each cell within the grid

should be individually clickable and a click event should be added to each cell based on the state of that cell.

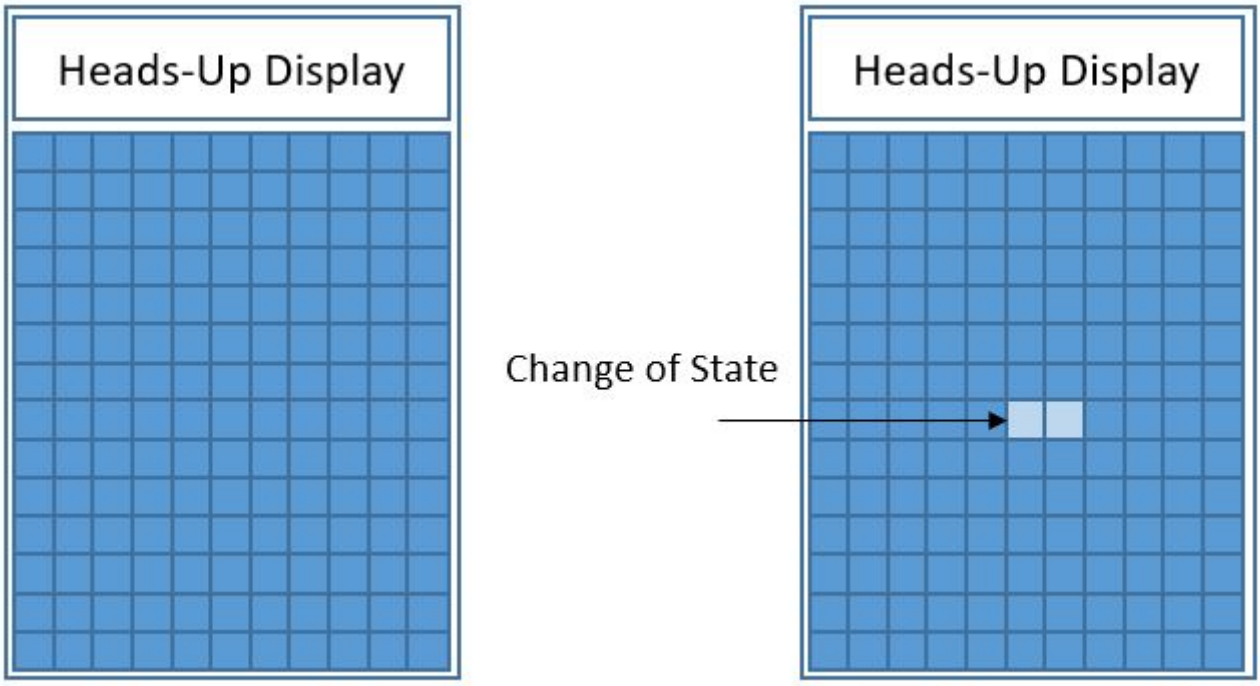

Figure 1b – The grid

Each cell can have one of two states:

- Not Revealed
- Revealed

The game begins with all cells being Not Revealed and as the user clicks on a cell, that cell is revealed. This change of state will be represented on the grid. Grid will be stored as a matrix.

3. Gameplay – Cells

Each Revealed cell can have one of three states:

- Bomb
- Number
- Empty

Figure 1c – Cell States

Each cell must store data on its own state in two separate ways being:

- Revealed or Not Revealed
- Bomb, Number or Empty

Each cell must act differently based on what data is stored therein.

- Bomb - A cell storing a bomb will end the game upon being revealed
- Number – A number must always be located in a cell neighbouring a bomb. This cell must check each of its neighbouring cells to determine if there is a bomb. The number displayed indicates the number of bombs which neighbour the cell.
- Empty – An empty cell will never neighbour a bomb. An empty cell will reveal itself and all of its neighbouring unrevealed cells. If another empty cell is revealed, that cell will also reveal all of the surrounding cells in a flow-on effect.

4. Gameplay – End Game

A timer must be implemented which starts when the player reveals their first cell and stops when either Victory or Loss conditions are met.

Victory Conditions are as follows:

- Player reveals all empty and numbered cells without revealing any bombs.

On Victory, the timer stops and this will be used as a score for the player. Player will have the option to restart.

Loss Conditions are as follows:

- Player reveals a bomb.

On Loss, all bombs on grid will be revealed to the player and timer will be stopped. Player will have the option to restart.

5. Gameplay – Flags

The player will have the option to ‘flag’ any unrevealed cell. This will create a second state for cells which are Not Revealed. Cell may be:

- Not Revealed
- Flagged

Player may flag or remove flag from any unrevealed cell. A flagged cell cannot be revealed until the flag has been removed.

Figure 1d – Flagged Cell

2. Use-Case – Winning a game of Minesweeper

1.0 Revision History

26/07/2019, Issue 1, Creation of Use Case, Zak Barker

1.1 Description

Using this use case, a game of Minesweeper may be played through until completion. This use case follows the events of an actor playing the game from running the implementation through till victory.

1.2 Actors

Actor is a casual gamer

2.0 Flow of Events

2.1 Basic Flow

Basic flow of events will see an actor(player) boot up the application and play the game through to completion.

1. Use case begins with the player booting up the application, Minesweeper opens at the title screen and the user decides to review the instructions by pressing the corresponding button.
2. Instructions Page appears with information on the rules, player presses return button and returns to the Title page before pressing ‘Standard’.
3. Standard Minesweeper board opens up with a 20x20 grid and 30 bombs are placed at random by system.
4. Player clicks on a random cell to be revealed.
5. Application begins timer and reveals the cell to be a number
6. Player tries another cell
7. Cell is revealed to be an empty cell, and this triggers a flow on until all adjacent empty cells are revealed along with their adjacent empty & numbered cells.
8. Player continues revealing numbered and empty cells until only 30 cells remain.
9. Timer stops, victory message is display and user is prompted to play again.
10. Use Case Ends.

2.2 Alternative Flow

2.2.1 Player Chooses a Different Game Mode

Player chooses to play Hex mode instead of Standard Minesweeper, flow is essentially the same as above with a different game mode

2.2.2 Player Loses and Replays

Player plays game as in basic flow but reveals a mine half-way

through the game. System reveals all of the bombs on the board and timer stops. User is prompted to restart game.

3.0 Special Requirements

3.1 No Special Requirements

4.0 Pre-Conditions

4.1 Actor must download and install program and script to run Minesweeper

5.0 Post-Conditions

5.1 N/A – User may exit application at any time

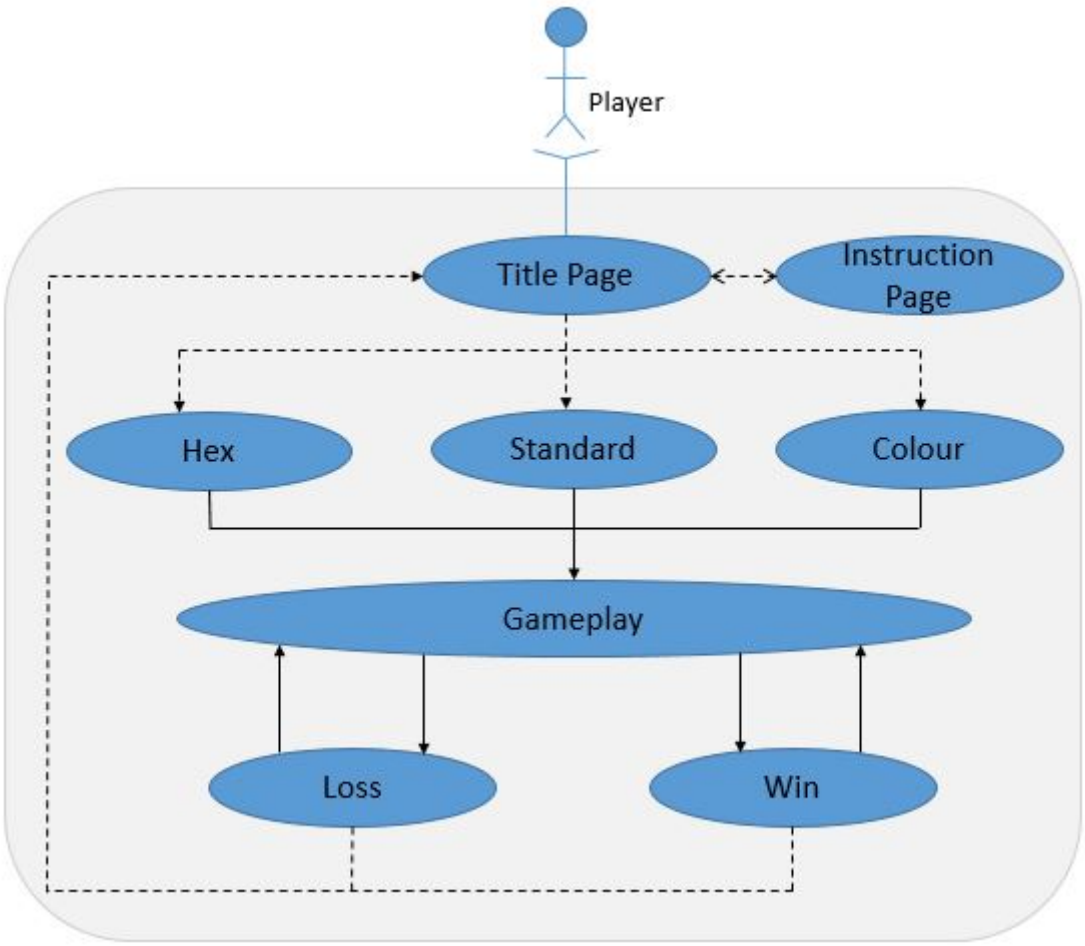

Figure 1e – Use Case Diagram

3. Non-Functional Requirements

1. **GUI Variation on Minesweeper**

   Instead of creating a standard GUI with mines on a grey screen, a variation on the minesweeper theme should be utilized with crabs used instead of bombs and palm trees used instead of flags. Colour scheme should reflect a beachy or ocean aesthetic with blues, greens, and yellows. This requirement is purely cosmetic.

2. **Languages**

   It is possible that people who do not speak English will be using this Application. It is important to provide these people instructions in their own language. An option to choose languages from a pre-defined set should be implemented and the user should be prompted upon booting up the application.

3. **Playable Via the Internet**

   The Application is relatively small. It is written using JavaScript and drawing heavily from the p5 library. This can easily run in a web browser and accessed via the internet to remove the necessity for the user to download the script along with an IDE in which to run the script. This will make it much more accessible to people who don't know how to run scripts or interact with IDEs.

4. **Instantaneous Reveal**

   The Cell's revelation must be instantaneous once the player has issued a click event. Game must operate without downtime or lag between cell states. This is due to the player being scored on a timer and from a general playability standpoint of the game. Application must be able to support this.

5. **No installation**

   Users who have downloaded the script should be able to play the game without downloading or installing anything else – besides an IDE through which they can run the script. Script is reliant on IDE to perform and will be split into several files. This should not impede the user's ability to launch the Application.

4. Constraints

1. **P5 Library**

   The entire Application is to be written in JavaScript and will be wholly reliant on the JavaScript P5 library framework for implementation. A suitable IDE will need to be utilized to adequately create, develop and test the application and live testing will be required. An online JavaScript IDE can be utilised, however, there are many all-purpose IDEs which are more suitable for use and available online.

2. **Time**

   Time will be a constraint on the project as a whole. 10 hours per week is the required amount of time dedicated to Software Engineering as a subject, lectures and workshops will require at least 5 hours per week. This leaves 5 hours per week to work on the project unless additional hours are given from developer's own time.

3. **Fault-Tolerance**

   A failure in one component of the finished application should not affect the operation of the rest of the application. Ie, when the Standard\Hex\Colour Minesweeper modes are implemented, a fault in one should not affect the functionality of another. The system should be resistant to faults in this manner and low coupling should be exhibited where possible.