Software Task 3

Contents

[Requirements Definition 1](#_Toc202395967)

[Problem Definition 1](#_Toc202395968)

[Legal and Ethical Considerations 2](#_Toc202395969)

[Requirements 2](#_Toc202395970)

[Design Specifications 3](#_Toc202395971)

[Storyboard 3](#_Toc202395972)

[Context Diagram 4](#_Toc202395973)

[Data Flow Diagram 5](#_Toc202395974)

[Structure Chart 6](#_Toc202395975)

[Algorithms 7](#_Toc202395976)

[Gantt Chart 8](#_Toc202395977)

[Development 9](#_Toc202395978)

[Development Log 9](#_Toc202395979)

[Testing and Debugging 15](#_Toc202395980)

[GitHub Repository 17](#_Toc202395981)

## Requirements Definition

### Problem Definition

Many people find it difficult to access maths games when bored. To fix this, minesweeper. To be unique, hexagons. This is intended for those that are bored, offline, looking for something for their brain to do or anyone looking for entertainment. This would include your average teenager and young adult, likely male as they are less likely to use social media to fill this gap.

This project will include minesweeper with either a square or hexagonal grid. Minesweeper includes mine distribution, neighbour checking, fail states, safe start and zero-spread (auto clicking if an adjacent tile had 0 mines as neighbours). A simple GUI will be made so that you can navigate the game. A text file will save some statistics, such as lowest time to beat, number of games played, win ratio and more. Sound effects will help the immersion yet can be muted. It will not ensure that each puzzle is solvable (like most minesweeper games), nor will it provide hints as to the solution.

The project must be completed in python, using the pygame module for the user interface. This means that I must learn the pygame module‘s functions.

### Legal and Ethical Considerations

The intellectual property of ‘Minesweeper’ as a name and game is not copyrighted in Australia, and is therefore fair game. The mechanics are not trademarked, nor are the generally used graphics. Minesweeper is only trademarked by Microsoft, although this is only for the subsection of minesweeper games named ‘Microsoft Minesweeper’, so this project is in the clear. The history of minesweeper is itself almost a legal mess. The creator, Curt Johnson, has admitted to borrowing the game’s design from another game, however unlike predictions, the original was not *Mined-Out*. To this day no one but Curt knows the mystery inspiration.

The game will be accessible, using point and click functionality with visible visual indicators. Common place accessibility features in minesweeper games is differently coloured cell numbers depending on the number. For example, green is often red, while one may be blue, some green, and so on. Although this will probably never happen, some have rarer numbers such as 6, 7 and 8 as rare colours, such as cyan, bright yellow or black. Spooky. Every button will have an outline, and as to not confuse people, only buttons will have an outline. These will always be bold, obvious outlines. No button will be too small to click, for those deficient in the fine motor skills required for mouse positioning. So that right clicking is easier, a ctrl + click will have the same effect.

### Requirements

Game functionality:

* Random mine placement
* Flagging
* Finding neighbouring mines
* Zero spread
* Fail state
* Timer
* Safe start

UI Functionality:

* Menu navigation
* Easily recognisable interactable features

Stats page functionality:

* Win ratio
* Total wins
* Total Losses
* Average time
* Total flags
* Total tiles revealed

## Design Specifications

### Storyboard

A diagram of a diagram

AI-generated content may be incorrect.

### Context Diagram

A diagram of a software system

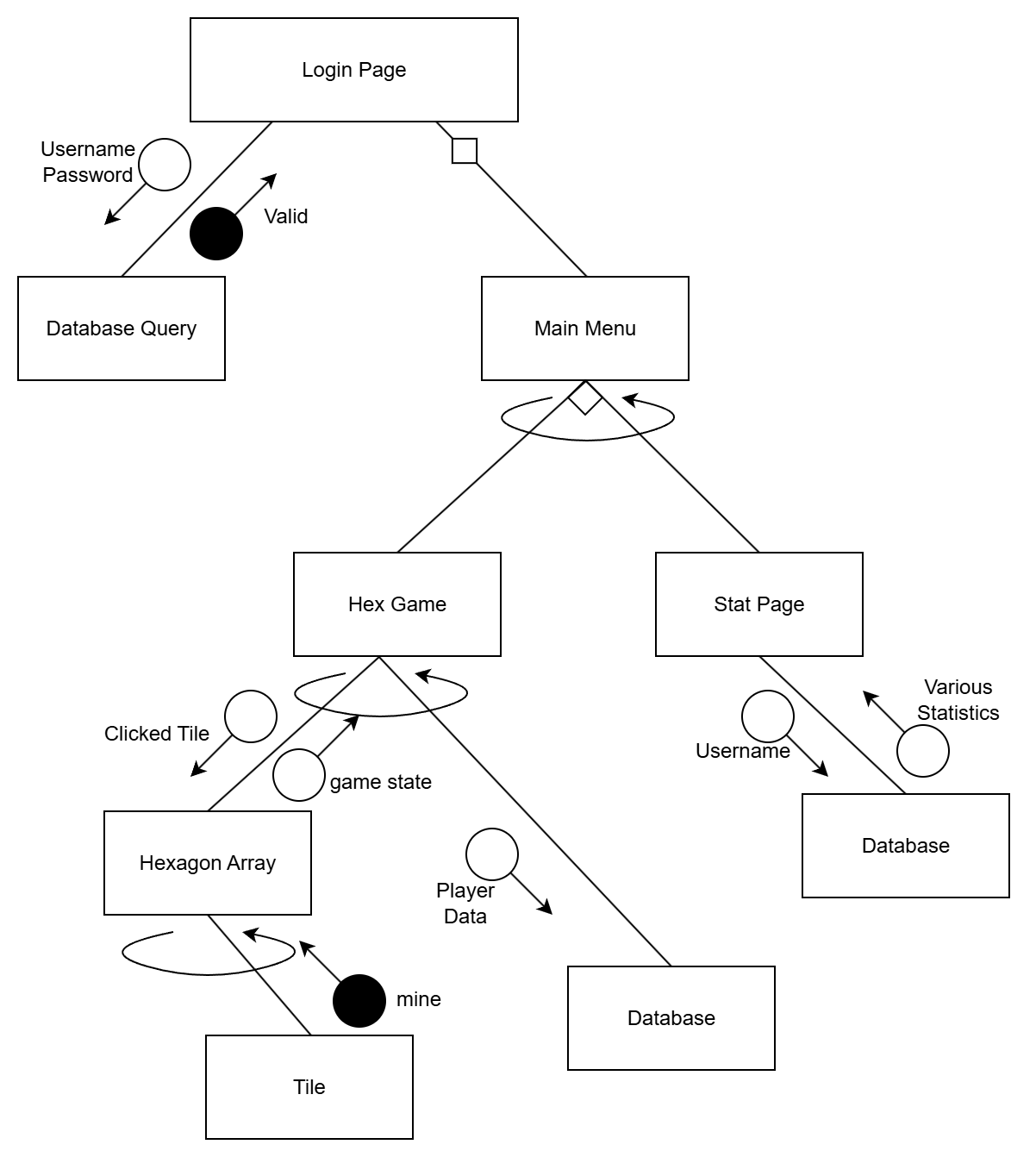
AI-generated content may be incorrect.

### Data Flow Diagram

A diagram of a user

AI-generated content may be incorrect.

### Structure Chart



### Algorithms

A diagram of a diagram

AI-generated content may be incorrect.

### Gantt Chart

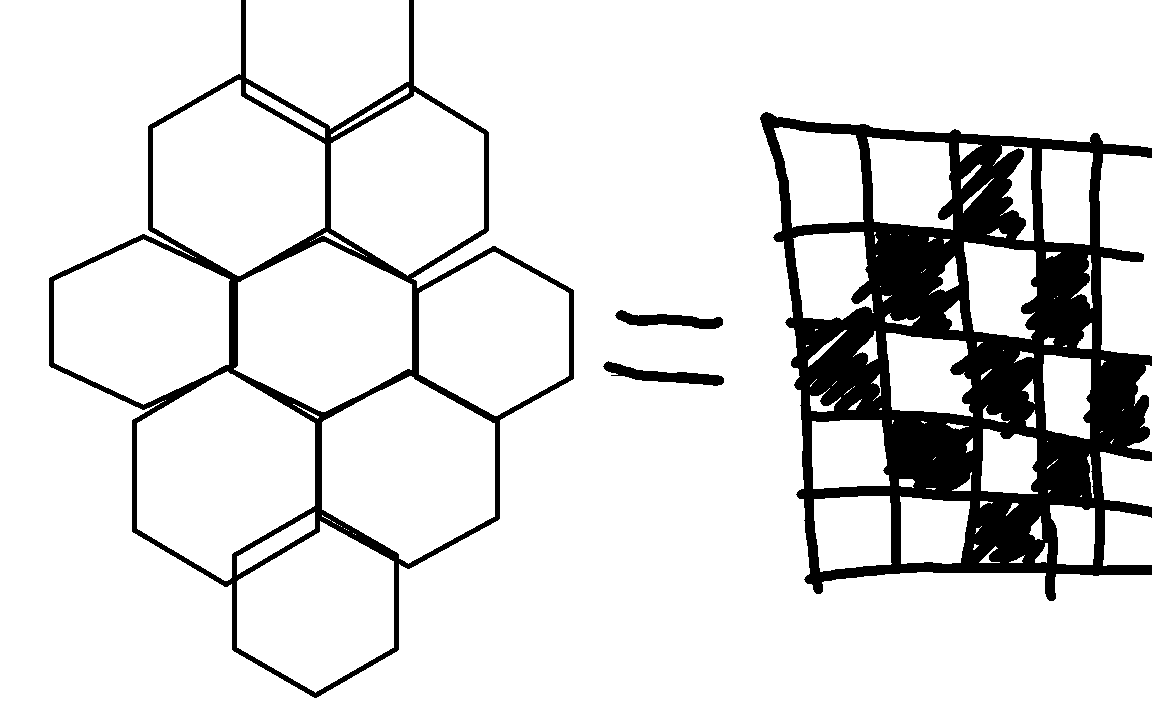
A graph with blue and purple squares

AI-generated content may be incorrect.

## Development

### Development Log

|  |  |
| --- | --- |
| Dev log Entry 1 | |
| Date | 12/5/25 |
| Week Number | 1 |
| Summary of Work | Started working on the codebase.  Created the hexagon class, which can draw itself.  Began working on the hex array. |
| Challenges and Solutions | Learning pygame took a while, but since I only have to learn it piece by piece it is manageable.  At first things were weirdly misaligned when using pygame.draw.polygon(), but after rounding every value before entering it as a parameter everything worked well.  When working on the hexagon array it was difficult to store a hexagonal grid in a 2D array. I eventually came up with the system as shown in the image below. |
| Milestones Achieved | Hexagon tiles created  First steps achieved |



|  |  |
| --- | --- |
| Dev log Entry 2 | |
| Date | 14/5/25 |
| Week Number | 1 |
| Summary of Work | Hex array finished |
| Challenges and Solutions | At first it was weird to tell each hexagon in the array where it would be drawn, so instead I saved their coordinates as an attribute at instantiation.  So that not every hexagon must do costly sqrts, I passed it as a parameter for maximum accuracy and minimum calculations.  Initially everything was misaligned. The problem was that I forgot to double some lengths. Now everything is aligned, and because of that it looks like a grid, rather than a lot of individual hexagons.  No challenge, but I drew the hexagons recursively. Put one in the centre, then add the six around that, then the six around each hexagon, and so on.  I accidentally misunderstood my own way of storing the array, and therefore drew it incorrectly. I drew it with the hexagon points on the y axis, while I did calculations regarding the array as if they were drawn with points on the x axis. Realised, and fixed. |
| Milestones Achieved | Hex array finished. |

|  |  |
| --- | --- |
| Dev log Entry 3 | |
| Date | 15/5/25 |
| Week Number | 1 |
| Summary of Work | Added mine distribution and cell numbers |
| Challenges and Solutions | Clicking was difficult, so I created the function findClosestHex(), which took the position of a click and found the hexagon that was clicked, by finding the closest tile and checking against a maximum distance away.  To add safe start, I distribute the mines after the first click.  When distributing mine counts to cells the function goes through all hexagons, retrieves the coordinates, checks the adjacent mines and then gives the count to each hexagon.  When implementing safe start, it didn’t help that much, so I changed it so that the adjacent tiles were also safe. This caused a lot of errors at the edges where it wasn’t exactly 7 safe cells, so I made it check each coord for a hexagon, so all now works. |
| Milestones Achieved | Mine distribution, clicking, cell numbers |

|  |  |
| --- | --- |
| Dev log Entry 4 | |
| Date | 19/5/25 |
| Week Number | 2 |
| Summary of Work | Zero spread, flagging, repository, button class |
| Challenges and Solutions | When clicking a zero, it should reveal all tiles around it. To do this, I added a return value to the reveal function. If it is 0, return true, and then reveal all tiles around itself. If there are more zeroes, it will reveal all of them too.  It was very simple to add flagging. Just toggle a spot, and you can’t click a flagged tile to reveal it, only to unflag.  I added a feature I saw in cardgames.io/minesweeper, where clicking a revealed tile with equal flags around it to mines around would reveal the other tiles adjacent. This caused a forever loop where it would reveal the tile, then the tile next to it, which - already filled with mines and revealed - would reveal all adjacent, which would in turn reveal all adjacent, and so on. An endless cycle would start, so I made it so that the reveal function had a parameter called ‘doSpread’, so that only the initial click could spread (as intended).  Making the repository was painless.  So that I could access a main menu or to restart the game, I needed buttons. Since registering clicking was near identical to that required for hexagons, I made the superclass Button.  RevealTile() stopped finding its own hexagon, and is now passed a coordinate and object reference. |
| Milestones Achieved | Zero spread, flagging, repository, button class |

|  |  |
| --- | --- |
| Dev log Entry 5 | |
| Date | 20/5/25 |
| Week Number | 2 |
| Summary of Work | Buttons now draw themselves |
| Challenges and Solutions | Very simple. Buttons now draw themselves. The hex grid still draws itself separately for modularity purposes + if it ain’t broke don’t fix it. |
| Milestones Achieved | Buttons now draw themselves |

|  |  |
| --- | --- |
| Dev log Entry 6 | |
| Date | 21/5/25 |
| Week Number | 2 |
| Summary of Work | Main menu, mine count |
| Challenges and Solutions | I made the game start at the main menu with navigation to the game optional.  By collecting references to each tile and retrieving their ‘mine attribute’ I can count the number of mines. Updating text was weird, but simply required some more checks before rendering the text. |
| Milestones Achieved | Game functionality basically complete |

|  |  |
| --- | --- |
| Dev log Entry 7 | |
| Date | 22/5/25 |
| Week Number | 2 |
| Summary of Work | Altered input system |
| Challenges and Solutions | When both clicking a button and entering a mouse click in the same game tick one would cancel out. To fix this, I separated the input gathering and management systems. Each input type would have its own Boolean, reset every frame and made true if its event comes up. After every event is found, then you find the current menu and manage the expected inputs.  TLDR: Input system has been refactored to reduce race-condition like failures. |
| Milestones Achieved | Finalised input system |

|  |  |
| --- | --- |
| Dev log Entry 8 | |
| Date | 23/5/25 |
| Week Number | 2 |
| Summary of Work | Fail state has been added, along with a new bomb texture and failed flag texture. Win and lose text. |
| Challenges and Solutions | At first, upon failure the game crashed. This was because of an infinite loop. When clicking a mine, it would reveal itself and all other mines. Each other mine would think it was the first mine and reveal all others again, forever. A simple parameter solved the issue. |
| Milestones Achieved | Fail state complete |

|  |  |
| --- | --- |
| Dev log Entry 9 | |
| Date | 30/5/25 |
| Week Number | 3 |
| Summary of Work | Data base added, login page almost functional, textboxes are now a thing |
| Challenges and Solutions | When making the textboxes, focusing and unfocusing on each was weird, but was quickly solved by declaring the variables that kept track of the current state everywhere to reduce confusion.  The textboxes themselves are a weird combination of button and text, so I implemented them but creating buttons that are then deleted and recreated with different text every time the text entered changes.  So that the textboxes would be aligned despite the variable sizes, I added a parameter to drawSelf() called leftMargin, which changes the draw function to align on the left. |
| Milestones Achieved | Login system 99% done |

|  |  |
| --- | --- |
| Dev log Entry 10 | |
| Date | 10/6/25 |
| Week Number | 5 |
| Summary of Work | Added login and signup functions |
| Challenges and Solutions | This was a small commit (33 lines added, 13 deleted), so thankfully no errors were encountered. |
| Milestones Achieved | Login page complete |

|  |  |
| --- | --- |
| Dev log Entry 11 | |
| Date | 11/6/25 |
| Week Number | 5 |
| Summary of Work | Tab to switch textboxes, match case syntax has replaced a lot of elifs, playing the game now adds data to the database. |
| Challenges and Solutions | Refactoring simply for the purpose of code looking nicer produced many temporary bugs, but they were quickly solved since they were silly mistakes. |
| Milestones Achieved | Database is now updated due to playing |

|  |  |
| --- | --- |
| Dev log Entry 12 | |
| Date | 24/6/25 |
| Week Number | 7 |
| Summary of Work | Text class with margin alignment is now existent to help with the stats page, found a sneaky bug in safe start code and another one in the data recording. |
| Challenges and Solutions | In the code for safe start, it would create bugs in particular edge cases (its paradoxical, and yet, it works) which were happening because declaring a variable as another variable which happens to be a list makes them both pointers to the same spot in memory, rather than separate iterables. With some list comprehension all is fixed.  When recording wins, I accidentally sent losses, so the stats were completely wrong. Changed the variable name and all is right in the world. |
| Milestones Achieved | Basically done |

## Testing and Debugging

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test ID | Category | Test Case Description | Input to Provide | Expected Output | Actual Output | Pass/Fail | Action Taken |
| 1 | Safe start | Top edge | Click top edge | Cell + neighbouring 4 cells are revealed to be safe | Error. No range from 1-1 for randint. | Fail | Check if the coordinates of neighbours have tiles, then adjust the total safe tile count accordingly. |
| 2 | Safe Start | Right Edge | Click right edge | Cell + neighbouring 4 cells are revealed to be safe | Error. No range from 1-1 for randint. | Fail | When checking which tiles are safe, there is a new and old list. The new list is a pointer to the old list, rather than a copy. Made it a copy. |
| 3 | Click Spread | Testing click spread | Click revealed tile surrounded by correct number of flags | Surrounding non-flag tiles are revealed | Frozen game. Forever loop | Fail | Add a parameter to revealTile() so that when something is revealed by click spread, it can not in turn click spread, stopping the loop. |
| 4 | Zero spread | Testing zero spread | Click on a tile with no surrounding mines | All surrounding tiles are revealed, and if it reveals a zero then recursive. | All surrounding tiles are revealed, along with more zeroes | Pass | N/A |
| 5 | Fail state | Click a mine | Click a tile with a mine on it | All mines are revealed | All mines and flags are revealed | Fail | Check if tiles are flagged before revealing them |
| 6 | Fail state | Fail with an incorrect flag | Click mine while a flag is on a safe square | All mines are revealed, flag is replaced with a cross | All mines are revealed, flag is replaced with a cross | Pass | N/A |
| 7 | Create Hex Array | Initial test | Size = 8 | Hexagonal grid resembling a hexagon with a side length of 8 hexagons. | Hexagonal grid resembling a squashed hexagon, too short and too long | Fail | Change the coordinates of neighbours in the recursive tile placement algorithm |
| 8 | Mine placement | Initial test | 30% mines | A random 30% of tiles are mines | Less than 30% of tiles are mines | Fail | Altered the counting of total hexagons, adding the collectHexReferences function for modularity. |
| 9 | Text input | No input | Empty String | “Please enter a username” | “Please enter a username” | Pass | N/A |
| 10 | Text input | Long | 20+ character long input | Inability to type more. Alert text | Inability to type more. Alert text | Pass | N/A |
| 11 | Textbox updating | Back to default | Remove all text | “Password” | “” | Fail | Check the current typed text for an empty string. If so, update the button’s text to the placeholder. |
| 12 | Textbox focus | Switch focused textbox | Press tab | Change typed text to the new textbox’s text. New input goes to the new textbox | Updates the new textbox with the text of the old one | Fail | Update the focus textbox functions to change the typed text upon changing |

## GitHub Repository

[Here it is](https://github.com/SamWamBamHam/Task3)

I think its pretty fire