SDV503 Assessment 1

Project 1 – Minesweeper Game Design - Caleb Eason

# Task 1

## Outline

Minesweeper is a simple, 2D, logic puzzle game. The object of the game is to identify the location of a number of mines within grid without setting any off. The player can left click on a cell within the grid to uncover it. Uncovering the cell will reveal how many mines are present within the surrounding 8 cells. If the user thinks they know the location of a mine, they can right click to place a marker. The user wins the game by uncovering every cell that does not contain a mine. If the user uncovers a cell that contains a mine, the mine will detonate, and the user will lose the game. The user’s time for each game will be displayed, starting from when the first cell is uncovered and ending when the game is finished (Wikipedia)

## Project Goals

The goal of this project is to design a beginner level game flow for the game Minesweeper. This will be achieved by creating three player scenarios to define, demonstrate and test the behaviour of the game flow.

Milestones:

1. Define player scenarios.

Each player scenarios will need to be comprehensively detailed. A description of each scenario will need to be developed that clearly describes each user action and the program’s reaction. The underlying logic of each scenario will need to be defined in a sequence of If/them conditions. This will show the specifications of the program though input/output sequences. The game architectures and data structure requirements will also need to be defined.

Tasks:

* Define scenario 1
* Define scenario 2
* Define scenario 3
* Scenario 1 logic specifications
* Scenario 2 logic specifications
* Scenario 3 logic specifications
* Document architecture and data structures

1. Design logic gate diagrams.

The logic specifications of each scenario will be developed further into a logic gate diagram. This diagram will show the player scenarios expressed as a sequential flow chart of logic gates, detailing the possible outcomes of each player input. Each diagram should be regularly tested during the design process to ensure the game flow is robust, efficient and accurate with logic gate truth tables (Wikipedia, 2023, minesweeper).

Tasks:

* Scenario 1 logic diagram
* Scenario 2 logic diagram
* Scenario 3 logic diagram

1. Test logic gate diagrams.

A detailed testing plan should be developed to comprehensively test each logic gate diagram, accounting for all possible user inputs. If any bugs or unwanted behaviours are found, the diagrams should be updated to rectify the issues.

Tasks:

* Create testing plan
* Test scenario 1 logic diagram
* Test scenario 2 logic diagram
* Test scenario 3 logic diagram

1. Create supporting documentation.

Once the game is developed, a user manual and developer guide will be created. These will explain the games rules, features and controls.

Tasks:

* Create user manual
* Create developer guide

## Software Development Life Cycle Method

This project has a relatively small scope. The project requirements are well defined and are unlikely to change during development. The project has a very minimal risk factor, as most project tasks/requirements can be identified and defined before development begins. For these reasons I have decided to select the waterfall method for this project.

The waterfall model is a linear SDLC that is suited for low-risk projects, making it a good fit. While the waterfall model’s linear progression can make midway redesigns impractical, this shouldn’t be an issue for this project as the project requirements are unlikely to change.

## First two phases of the Waterfall Method

Gathering Requirements

Phase 1 of the waterfall method involves gathering requirements. In this phase all necessary program requirements must be identified and defined. These will include but are not limited to functional, non-functional and UX requirements. All relevant requirements for the project should be collected and collated into a software requirements specification (SRS) document. This document will communicate all the requirements of the project to any project participants and stakeholders. (Wikipedia, 2023, waterfall model).

The requirements for this project are detailed in the outline section and the project goals section. Any additional requirements can be found in the assessment rubric which contains the support notes for the Minesweeper game and in the user manual and developer guide under section 6.

Design

Phase 2 of the waterfall method involves creating a design for the program based of the system requirements. In this phase, a detailed design is created for the program functionality and any UI elements of the project. These designs should encompass and meet all program requirements defined in the SRS document (Wikipedia, 2023, waterfall model).

The project design will be detailed under sections 3 and 4. Section 3 will detail the functionality of the program through a series of user scenarios. It will also include the design for the game architecture and data structure requirements. Section 4 will expand on the designs with a series of logic gate diagrams.

## References

Google. (n.d.). *Google minesweeper*.

Retrieved March 10, 2023,

from <https://g.co/kgs/4Cmoyo>

Wikipedia. (2023, February 17). *Minesweeper (video game)*. Wikipedia, the free encyclopedia. Retrieved March 10, 2023, from <https://en.wikipedia.org/wiki/Minesweeper_(video_game)>

Wikipedia. (2023, January 11). Waterfall model. Wikipedia, the free encyclopedia.

Retrieved March 20, 2023,

from <https://en.wikipedia.org/wiki/Waterfall_model>

Academo. (n.d.). Logic gate simulator. Academo.org - Free, interactive, education.

Retrieved March 20, 2023,

from <https://academo.org/demos/logic-gate-simulator/>

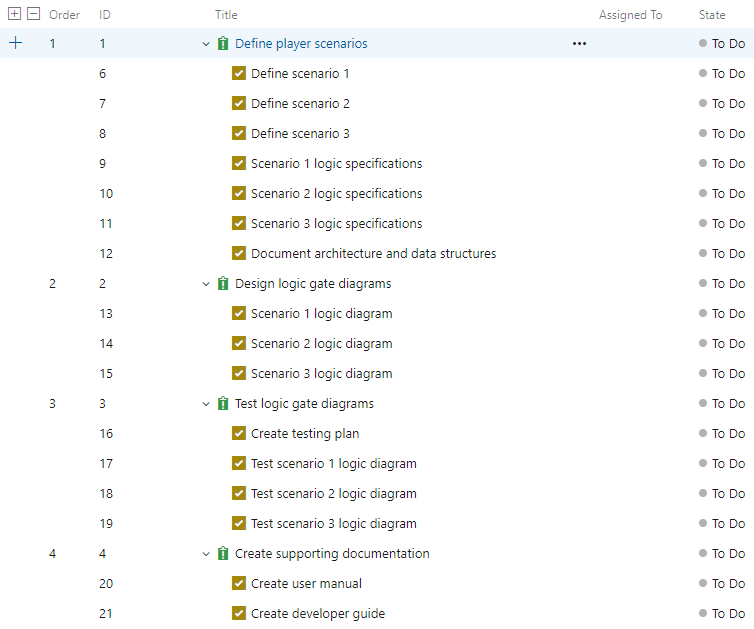
# Task 2

## Timeline

DevOps space

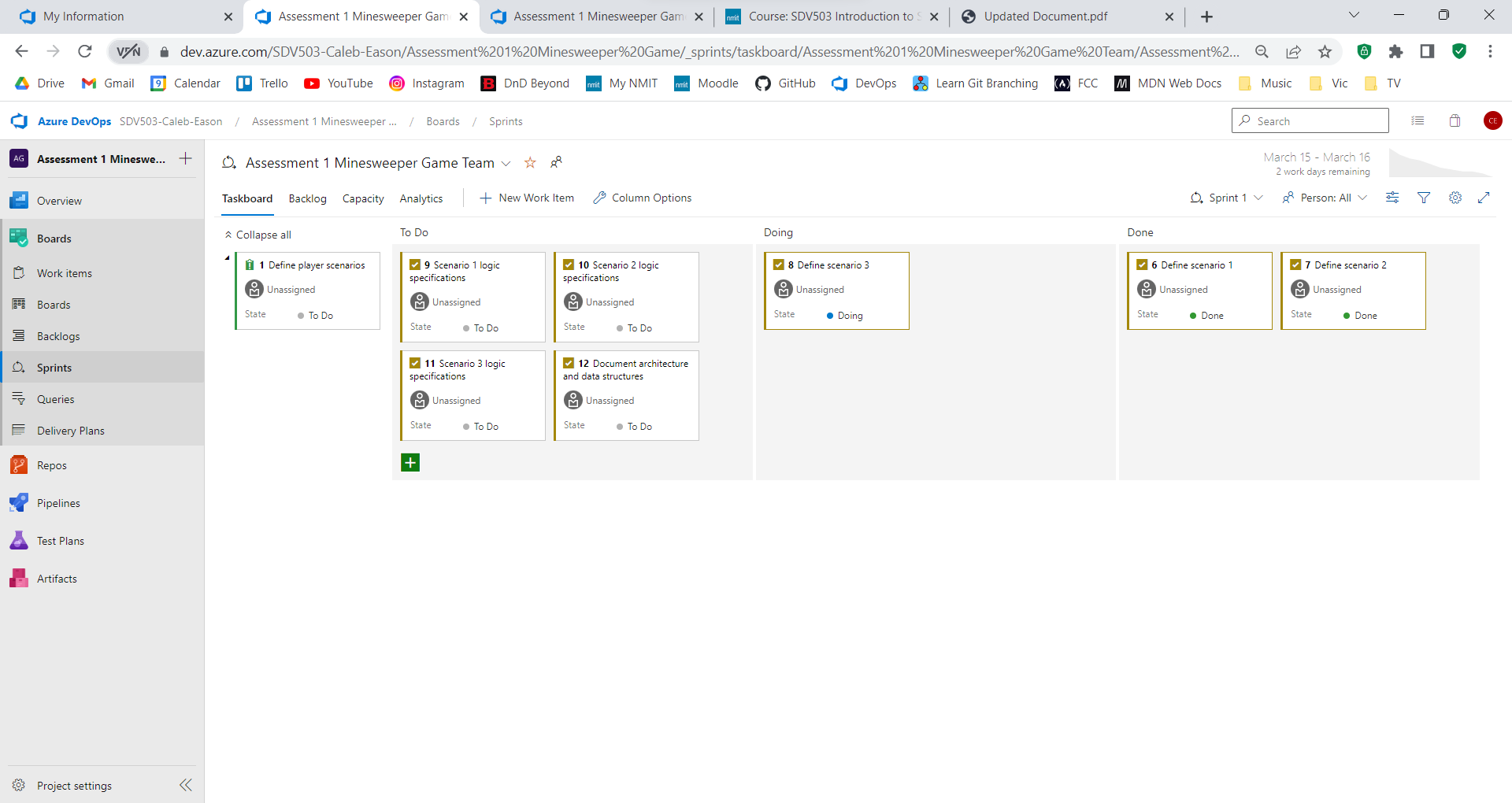
<https://dev.azure.com/SDV503-Caleb-Eason/Assessment%201%20Minesweeper%20Game>

Below is a screenshot of all my milestones and tasks translated into DevOps. Each milestone Is listed as an issue with the accompanying tasks attached.

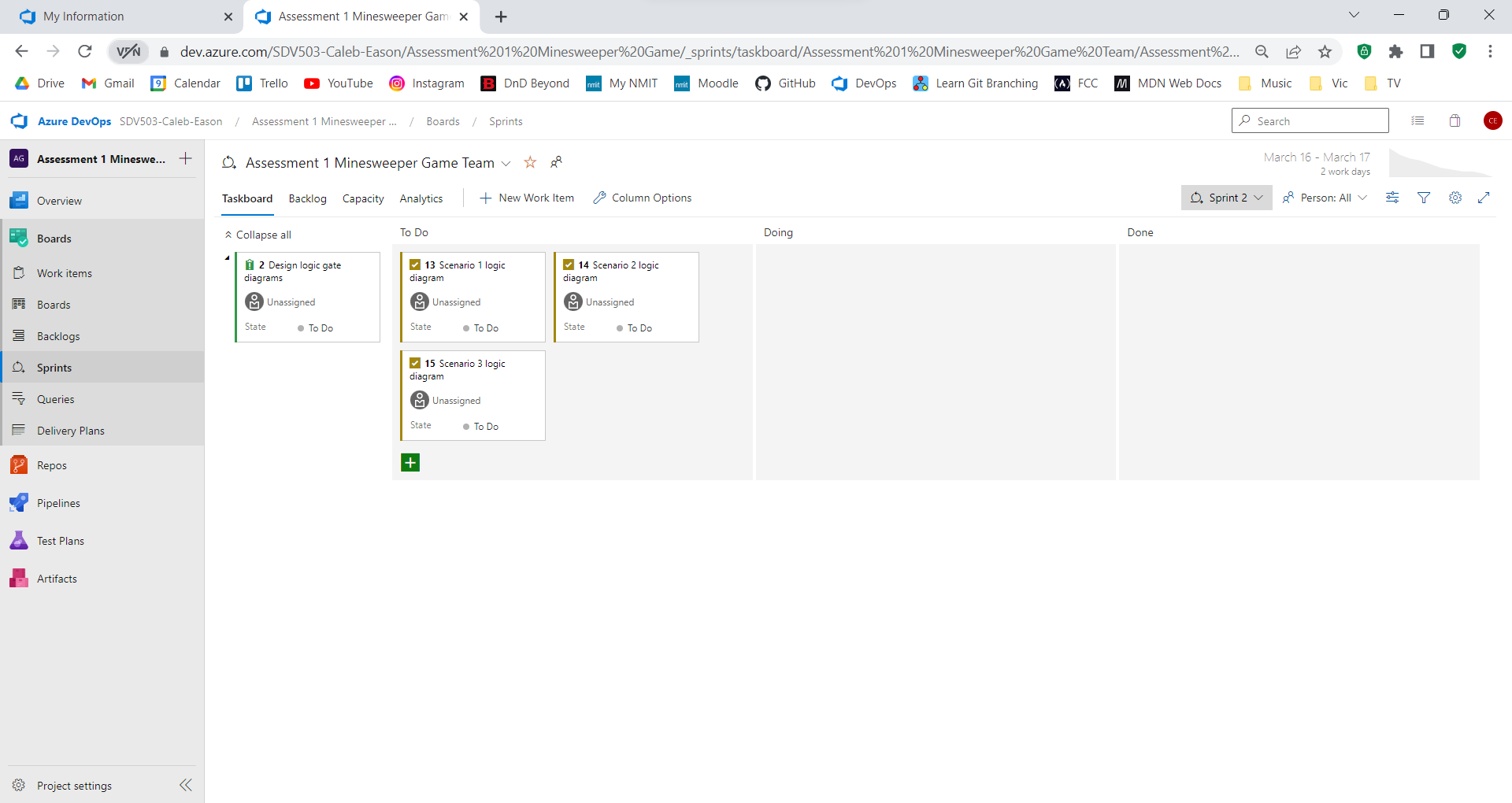


I assigned each milestone to its own sprint, shown below.

Sprint 1, 15/03 – 16/03:



Sprint 2, 16/03 – 17/03:



Sprint 3, 17/03 – 18/03:

Graphical user interface, application

Description automatically generated

Sprint 4, 18/03 – 19/03:

Graphical user interface, application, Word

Description automatically generated

## Required Resources

The program will be developed in Visual Studio Code using JavaScript coding language and will be designed for windows 10 and 11 operating systems.

# Task 3

Cells within the game grid will be referred to as shown in the table below.

|  |  |  |
| --- | --- | --- |
| A1 | A2 | A3 |
| B1 | B2 | B3 |
| C1 | C2 | C3 |

Scenarios generated with google minesweeper (2023).

## Player Scenario 1

Description

1. Graphical user interface

   Description automatically generatedThe player begins the game by left clicking on cell A1, revealing the number 1.
2. Graphical user interface

   Description automatically generatedThe player then left clicks on cell A3, revealing another 1.
3. A picture containing text, game

   Description automatically generatedThe player left clicks on cell B2, revealing a mine and ending the game; the player loses the game.

If statement logic

1. **Initialisation**

When the user begins the game a random number of mines (limitations to be determined) will be randomly distributed across the game board. Each cell will be assigned a value of either 1 or 0. A value of 1 means the cell contains a mine while a value of 0 means it does not.

In this case cell, B2 is assigned a value of 1, while every other cell is assigned a value of 0.

1. **When the player left clicks on cell A1.**

The program will check if the cell contains a mine.

If user left click == 1 AND cell contains mine == 1 then the mine will detonate, and the user will lose. If the User Left Click == 1 AND cell contains mine == 0 then the cell will be uncovered.

In this case, cell A1 does not contain a mine so the cell will be uncovered.

The program will then check if there are any mines in the surrounding area.

If contains mine == 1 for cell A2 OR cell B1 OR cell B2 then the program will count how many mines are in the surrounding tiles and display the result on cell A1.

In this case, cell B2 contains a mine so the number 1 will be displayed on cell A1.

1. **When the player left clicks on cell A3.**

If the User Left Click == 1 AND cell contains mine == 0 then the tile will be uncovered.

Cell A3 does not contain a mine so it will be uncovered.

If contains mine == 1 in cell A2 OR cell B2 OR cell B3 then the program will count how many mines are in the surrounding tiles and display the result on cell A3.

Cell contains mine == 1 for cell B2 so the number 1 will be displayed on cell A3.

1. **When the player left clicks on cell B2**

If user left click == 1 AND cell contains mine == 1 then the mine will detonate, and the user will lose.

In this case, cell contains mine == 1 for cell B2. When the user left clicks on B2 they will detonate the mine and the game will end in a loss.

## Player Scenario 2

Description

1. A picture containing graphical user interface

   Description automatically generatedThe player begins the game by left clicking on cell C3, revealing a 1.
2. Graphical user interface, application

   Description automatically generatedThe player left clicks on cell C1, revealing an empty tile. This causes cells B1, B2 and C2 to be automatically uncovered, revealing the numbers 1, 2 and 1 respectively.
3. The player right clicks on cell B3 to place a flag.
4. Graphical user interface

   Description automatically generated with medium confidenceThe player left clicks on cell A3, revealing a 1.

Graphical user interface

Description automatically generated

1. The player then left clicks on cell A2, revealing a 2. The player has now successfully uncovered every cell that does not contain a mine, causing the game to end. The player has won the game.

If statement logic

1. **Initialisation**

When the user begins the game a random number of mines (limitations to be determined) will be randomly distributed across the game board. Each cell will be assigned a value of either 1 or 0. A value of 1 means the cell contains a mine while a value of 0 means it does not.

In this case cells A1 and B3 are assigned a value of 1, all other cells are assigned a value of 0.

1. **When the player left clicks on cell C3.**

The program will check if the cell contains a mine.

If user left click == 1 AND cell contains mine == 1 then the mine will detonate, and the user will lose. If the User Left Click == 1 AND cell contains mine == 0 then the cell will be uncovered.

In this case, cell C3 does not contain a mine so the cell will be uncovered.

The program will then check if there are any mines in the surrounding area.

If contains mine == 1 for cell C2 OR cell B2 OR cell B3 then the program will count how many mines are in the surrounding tiles and display the result on cell C3.

In this case, cell B3 contains a mine so the number 1 will be displayed on cell C3.

1. **When the player left clicks on cell C1.**

If the User Left Click == 1 AND cell contains mine == 0 then the tile will be uncovered.

Cell C1 does not contain a mine so it will be uncovered.

If contains mine == 1 in cell C2 OR cell B1 OR cell B2 then the program will count how many mines are in the surrounding tiles and display the result on cell C1.

In this case, cell contains mine == 0 for all the surrounding cells, so C1 will remain empty and cells C2, B1 and B2 will be automatically uncovered.

For each automatically uncovered cell, the check for any nearby mines will be performed.

Cells C2 and B1 both have 1 mine nearby so a number 1 will be displayed in each. Cell B2 has 2 mines nearby so it will display the number 2.

1. **When the player right clicks on cell B3**

The program will check if the tile contains a flag or a question mark.

If user right click == 1 AND cell contains flag == 0 AND cell contains question mark == 0 then place a flag on the cell.

Cell B3 does not already contain a flag or a question mark so a flag will be placed on the cell.

1. **When the player left clicks on cell A3**

If the User Left Click == 1 AND cell contains mine == 0 then the tile will be uncovered.

Cell A3 does not contain a mine so it will be uncovered.

If contains mine == 1 in cell A2 OR cell B2 OR cell B3 then the program will count how many mines are in the surrounding tiles and display the result on cell A3.

Cell contains mine == 1 for cell B3 so the number 1 will be displayed on cell A3.

1. **When the player left clicks on cell A2**

If the User Left Click == 1 AND cell contains mine == 0 then the tile will be uncovered.

Cell A2 does not contain a mine so it will be uncovered.

If contains mine == 1 in cell A1 OR cell A3 OR cell B1 OR cell B2 OR cell B3 then the program will count how many mines are in the surrounding tiles and display the result on cell A2.

Cell contains mine == 1 for cell B3 and cell A1 so the number 2 will be displayed on cell A2.

The program will then check if every cell that does not contain a mine has been uncovered.

If, for every cell in the grid, (Cell contains mine == 0 AND Cell has been uncovered == 1) XOR cell contains mine == 1, then the game will end in a win.

In this case, every cell without a mine has been uncovered so the game will end in a win.

## Player Scenario 3

Description

1. Graphical user interface

   Description automatically generated with medium confidenceThe player begins the game by left clicking on cell C1, revealing an empty tile. This causes cells B1, B2 and C2 to be automatically uncovered, revealing the numbers 1, 2 and 1 respectively.
2. The player right clicks on cells A1, A2, B3 and C3, placing a flag on each.
3. Graphical user interface, icon

   Description automatically generatedThe player then right clicks again on cells, A1, A2, B3 and C3, replacing the flags with question marks.
4. Icon

   Description automatically generated with medium confidenceThe player left clicks on cell A3, revealing a 2.
5. Table

   Description automatically generatedThe player right clicks on cells A1, A2, B3 and C3 to remove the question marks, then right clicks again on cells A2 and B3 to place a flag on each.
6. The player middle clicks on cell B2. This causes cells A2 and C3 to be uncovered. Both cells reveal a 1. The player has now successfully uncovered every cell that does not contain a mine, causing the game to end. The player has won the game.

If statement logic

1. **Initialisation**

When the user begins the game a random number of mines (limitations to be determined) will be randomly distributed across the game board. Each cell will be assigned a value of either 1 or 0. A value of 1 means the cell contains a mine while a value of 0 means it does not.

In this case cells A2 and B3 are assigned a value of 1, all other cells are assigned a value of 0.

1. **When the player left clicks on cell C1.**

The program will check if the cell contains a mine.

If user left click == 1 AND cell contains mine == 1 then the mine will detonate, and the user will lose. If the User Left Click == 1 AND cell contains mine == 0 then the cell will be uncovered.

Cell C1 does not contain a mine so it will be uncovered.

The program will then check if there are any mines in the surrounding area.

If contains mine == 1 in cell C2 OR cell B1 OR cell B2 then the program will count how many mines are in the surrounding tiles and display the result on cell C1.

In this case, cell contains mine == 0 for all the surrounding cells, so C1 will remain empty and cells C2, B1 and B2 will be automatically uncovered.

For each automatically uncovered cell, the check for any nearby mines will be performed.

Cells C2 and B1 both have 1 mine nearby so a number 1 will be displayed in each. Cell B2 has 2 mines nearby so it will display the number 2.

1. **When the player right clicks on Cells A1, A2, B3, C3**

For each cell, the program will check if they already contain a flag or a question mark.

If user right click == 1 AND cell contains flag == 0 AND cell contains question mark == 0 then place a flag on the cell.

None of these cells (A1, A2, B3, C3) already contain a flag or a question mark, so a flag will be placed on each.

1. **When the player right clicks again on Cells A1, A2, B3, C3**

For each cell:

If user right click == 1 AND cell contains flag == 0 AND cell contains question mark == 0 then place a flag on the cell. If user right click == 1 AND cell contains flag == 1 then replace flag with question mark.

In this case, all these cells (A1, A2, B3, C3) already contain a flag, so the flag will be replaced with a question mark for each.

1. **When the player left clicks on cell A3**

If the User Left Click == 1 AND cell contains mine == 0 then the tile will be uncovered.

Cell A3 does not contain a mine so it will be uncovered.

If contains mine == 1 in cell A2 OR cell B2 OR cell B3 then the program will count how many mines are in the surrounding tiles and display the result on cell A3.

Cell contains mine == 1 for cells A2 and B3 so the number 2 will be displayed on cell A3.

1. **When the player right clicks a third time on Cells A1, A2, B3, C3**

For each cell:

If user right click == 1 AND cell contains flag == 0 AND cell contains question mark == 0 then place a flag on the cell. If user right click == 1 AND cell contains flag == 1 then replace flag with question mark. If user right click == 1 AND cell contains question mark == 1 then remove the question mark.

In this case, all these cells (A1, A2, B3, C3) contain a question mark so the question mark will be removed for each.

When the user right clicks for a fourth time on cells A2 and B3:

If user right click == 1 AND cell contains flag == 0 AND cell contains question mark == 0 then place a flag on the cell.

Neither cells A2 or B3 already contain a flag or a question mark, so a flag will be placed on both.

1. **When the player middle clicks on cell B2.**

The program will count the number of surrounding mines and flags/question marks.

(If the User Middle Click == 1 OR User Left + Right Click == 1) AND number of surrounding flags >= number of surrounding mines. Then all nearby tiles without flags or question marks will be uncovered.

In this case, the number of flags around cell B2 is equal to the number of mines so all the nearby cells will be automatically uncovered.

In this case, cells A1 and C3 do not contain a flag so they will be uncovered, both cells display a 1.

The program will then check if every cell that does not contain a mine has been uncovered.

If, for every cell in the grid, (Cell contains mine == 0 AND Cell has been uncovered == 1) XOR cell contains mine == 1, then the game will end in a win.

In this case, every cell without a mine has been uncovered so the game will end in a win.

## Data structure Requirements

Cell Data

Cell data will be stored in an Array of Tuples. Each tuple will represent a cell within the grid and will contain three integer items and two Boolean items. There will be one tuple for each cell, a total of 9.

* The first two items in each tuple will represent the cells position within the grid (column and row) in integers. These items will be constant so there is one tuple for each cell in the grid.
* The third item will represent in Boolean whether the cell contains a mine. A value of 1/True means the cell does contains a mine while a value of 0/False means it does not. This value will be variable and will have a default value of 0/False.
* The fourth item will represent in Boolean whether the cell has been uncovered. A value of 1/True means the cell has been uncovered while a value of 0/False means it has not. This will be variable and will have a default value of 0/False.
* The fifth item represents in integers how many mines are in the surrounding 8 tiles. This will be variable and will have no default value (null/undefined).

Array format:

[(*column, row, contains mine, uncovered, nearby mines*),   
(*column, row, contains mine, uncovered, nearby mines*),   
etc… ]

Example tuple:

(2,3,1,0,NULL) – This represents the cell in the 2nd column, 3rd row. This cell has not been uncovered and contains a mine. The value for nearby mines has not been assigned yet.

(1,1,0,1,3) – This represents the cell in the 1st column, 1st row. This cell has been uncovered and does not contain a mine. There are three mines near this cell.

## Game Architecture

Initialisation

Generate a random number of mines to be placed in the grid (limitations to be determined).

Randomly distribute the mines across the grid.

Begin main gameplay loop.

Main gameplay loop

Loop:

Wait for input.

**When the player left clicks on a cell:**

If this is the first time the player has left clicked:

Remove any flags/question marks that have been placed prior to the first left click.

If the first cell the player clicks contains a mine:

Remove the mine from the clicked cell.

Loop:

Select a random cell.

If the cell does not contain a mine:

Place a mine in the selected cell.

Break

Else:

Nothing will happen.

Else:

Uncover the left clicked cell.

Start the game timer.

Else

Run function leftClick, parsing the tuple for which cell was clicked.

**When the player right clicks on a cell:**

Run function rightClick, parsing the tuple for which cell was clicked.

**When the player middle clicks on a cell, or simultaneously left and right clicks on a cell:**

Run function middleClick, parsing the tuple for which cell was clicked.

**After player input**

Check if all cells that do not contain a mine have been uncovered.

If all the cells without a mine have been uncovered:

Stop game timer.

Break main gameplay loop.

Else:

Do nothing.

Functions

**leftClick:**

Check if the cell has already been uncovered.

If the cell has not already been uncovered:

Check if the cell contains a mine.

If the cell contains a mine:

Reveal the mine and end the game in a loss.

If it does not contain a mine:

Run function uncoverCell.

Else (If the cell has already been uncovered):

Nothing will happen.

**rightClick:**

Check if the cell has already been uncovered.

If the cell has not already been uncovered:

Check if the cell has been marked with a flag or question mark.

If the cell contains a flag:

Replace the flag with a question mark.

Else if the cell contains a question mark:

Remove the question mark.

Else:

Place a flag.

Else (If the cell has already been uncovered):

nothing will happen.

**middleClick:**

Check if the cell has already been uncovered.

If the cell has not already been uncovered:  
 Nothing will happen.

Else:

Check if the cell has at least 1 mine nearby.

If there is at least 1 mine nearby:

Count how many flags/question marks are in the surrounding area.

If the amount of flags/question marks is greater than or equal to the number of mines nearby:

Uncover all surrounding tiles that have not already been uncovered.

Else:

Nothing will happen.

**uncoverCell:**

Check if there are any mines in the surrounding cells.

If there are mines in the surrounding cells:

Count the number of surrounding mines and display that number on the centre cell.

Else:

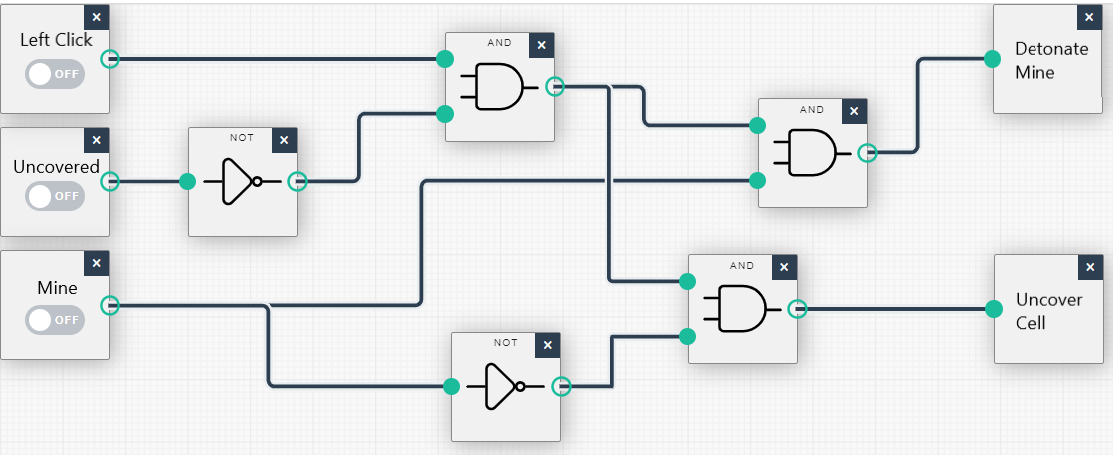
Leave the uncovered cell blank and automatically rerun this function for all surrounding cells that aren’t already uncovered.

# Task 4

Logic gate diagrams generate with Acedemo Logic Gate Simulator (retrieved 2023).

# Scenario 1 Logic Gate Diagram

Version 1



This is the first version of the logic diagram for scenario 1. It contains 3 inputs:

* Cell has been left clicked (A)
* Cell is uncovered (B)
* Cell contains mine (C)

And 2 outputs:

* Detonate the mine and end the game (Z)
* Uncover the cell (Y)

Inputs abbreviated to A for ‘Left Click’, B for ‘Uncovered’ and C for ‘Mine’. Outputs will be abbreviated to Z for ‘Detonate Mine’ and Y for ‘Uncover Cell’.

The equations for this diagram are:

* Detonate Mine (Z) = (A AND NOT B) AND C
* Uncover Cell (Y) = (A AND NOT B) AND NOT C

Truth Table:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | B | C | B’ | C’ | AB’ | (AB’)C | (AB’)C’ | Z | Y |
| 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Version 1.1

This diagram can be slightly refactored with a triple AND gate.

Diagram, schematic

Description automatically generated

New equations:

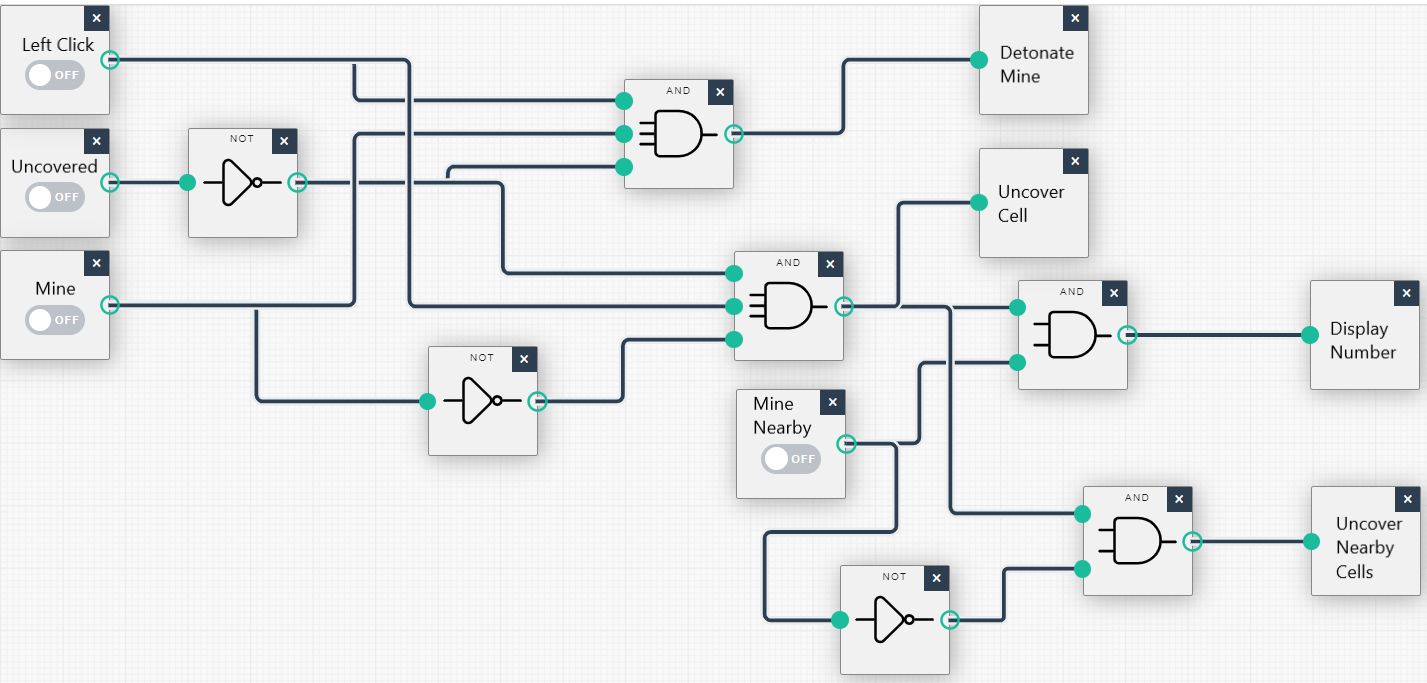
* Detonate Mine (Z) = A AND NOT B AND C
* Uncover Cell (Y) = A AND NOT B AND NOT C

Truth Table:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | B | C | B’ | C’ | AB’C | AB’C’ | Z | Y |
| 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |

The next step is to add the logic for displaying a number on the uncovered cell.

Version 2



This version contains 1 additional input:

* Mines in the surrounding cells (D)

And 2 additional outputs:

* Display the number of nearby mines on the uncovered cell (X)
* Uncover all surrounding cells (W)

The ‘Nearby Mines’ input will be abbreviated to D. The ‘Display Number’ output will be abbreviated to X. The ‘Uncover Nearby Cells’ output will be abbreviated to W.

The equations for this diagram are:

* Detonate Mine (Z) = A AND NOT B AND C
* Uncover Cell (Y) = A AND NOT B AND NOT C
* Display Number (X) = (A AND NOT B AND NOT C) AND D
* Uncover Nearby Cells (W) = (A AND NOT B AND NOT C) AND NOT D

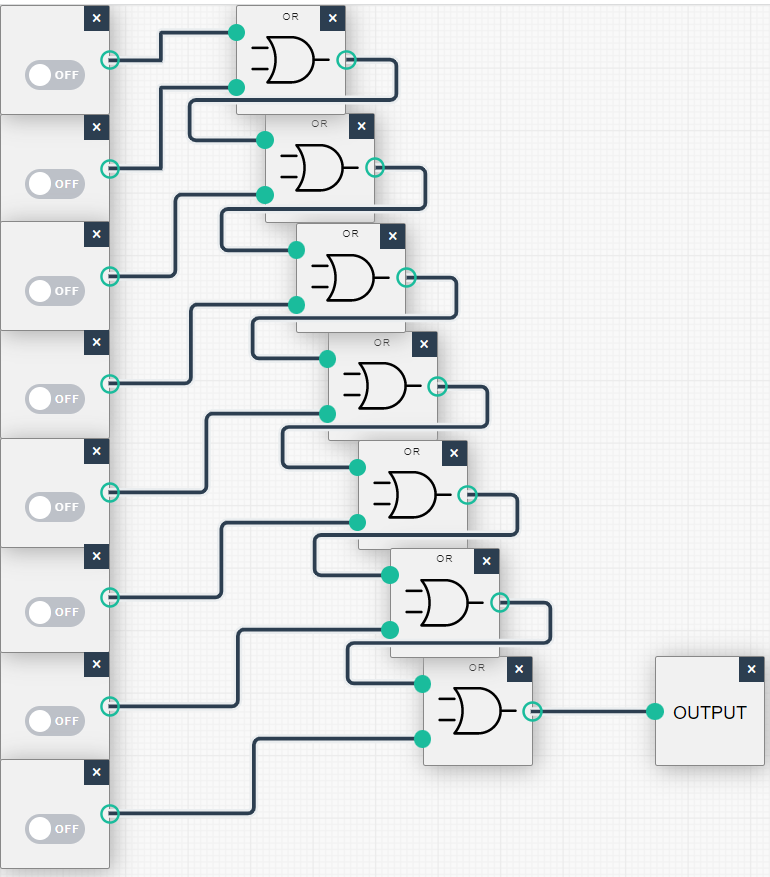
Truth Table:

The logic to determine Y and Z has not changed so it can be excluded from this table.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| AB’C’ | D | D’ | (AB’C’)D | (AB’C’)D’ | X | W |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 | 1 | 0 |

Mine Nearby logic diagram

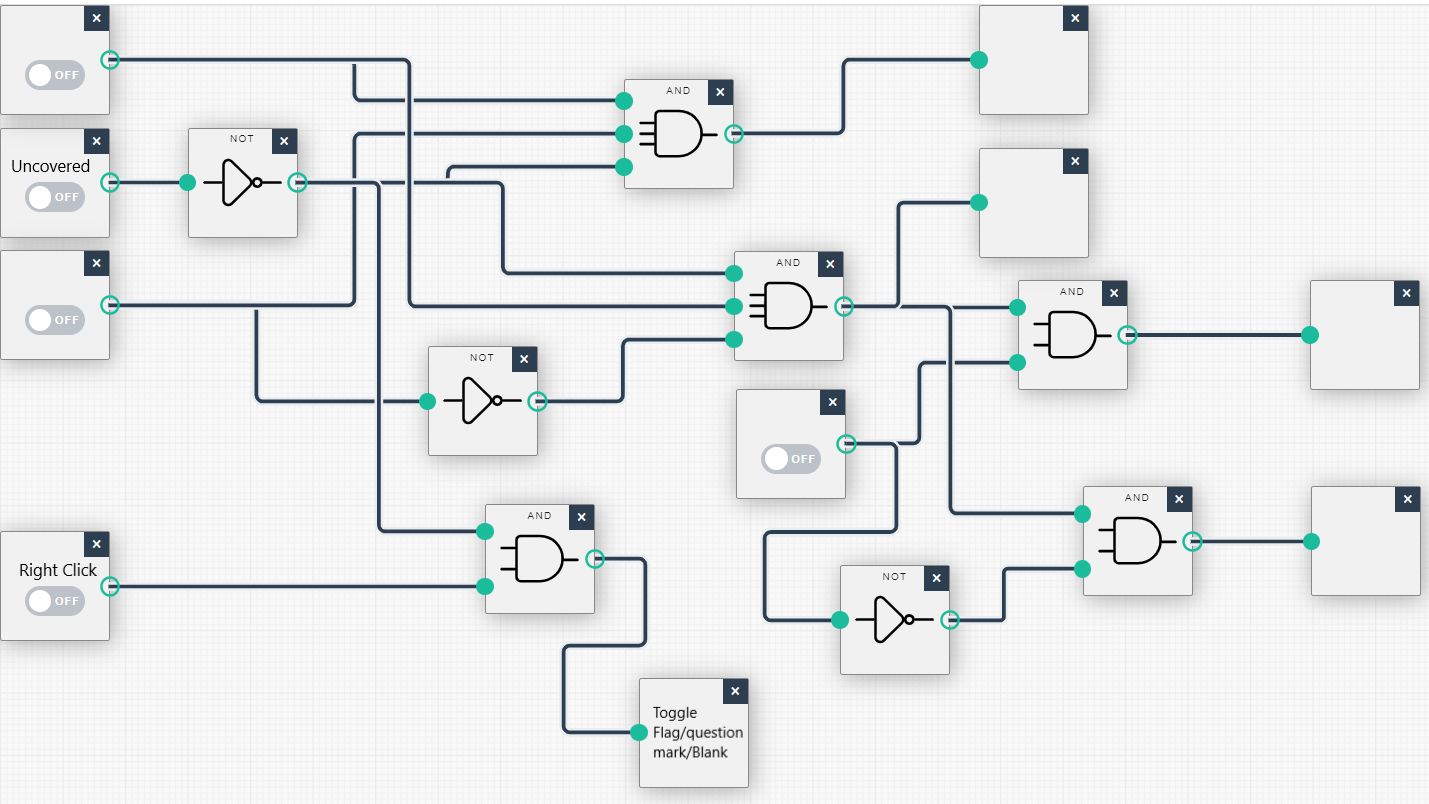
The ‘Mine Nearby’ input is an abbreviation of the below diagram.



Each input represents the presence of a mine in one of the surrounding 8 cells. If 1 or more inputs are equal to 1, the value of ‘Mine Nearby’ (D) in the version 2 diagram will be 1. If all inputs are 0, the value of ‘Mine Nearby’ will be 0.

# Scenario 2 Logic Gate Diagram

Version 1



This is the first version of the logic diagram for scenario 2. It builds off the diagram from scenario 1, adding 1 new input for when the user right clicks (E) and 1 new output which toggles between a blank cell, placing a flag and placing a question mark (V).

The ‘Right Click’ input will be abbreviated to E and the ‘Toggle flag/question mark/blank’ output will be abbreviated to V. B is an abbreviation for the ‘Uncovered’ input as stated in scenario 1.

The equation for the new output is: V = E AND NOT B

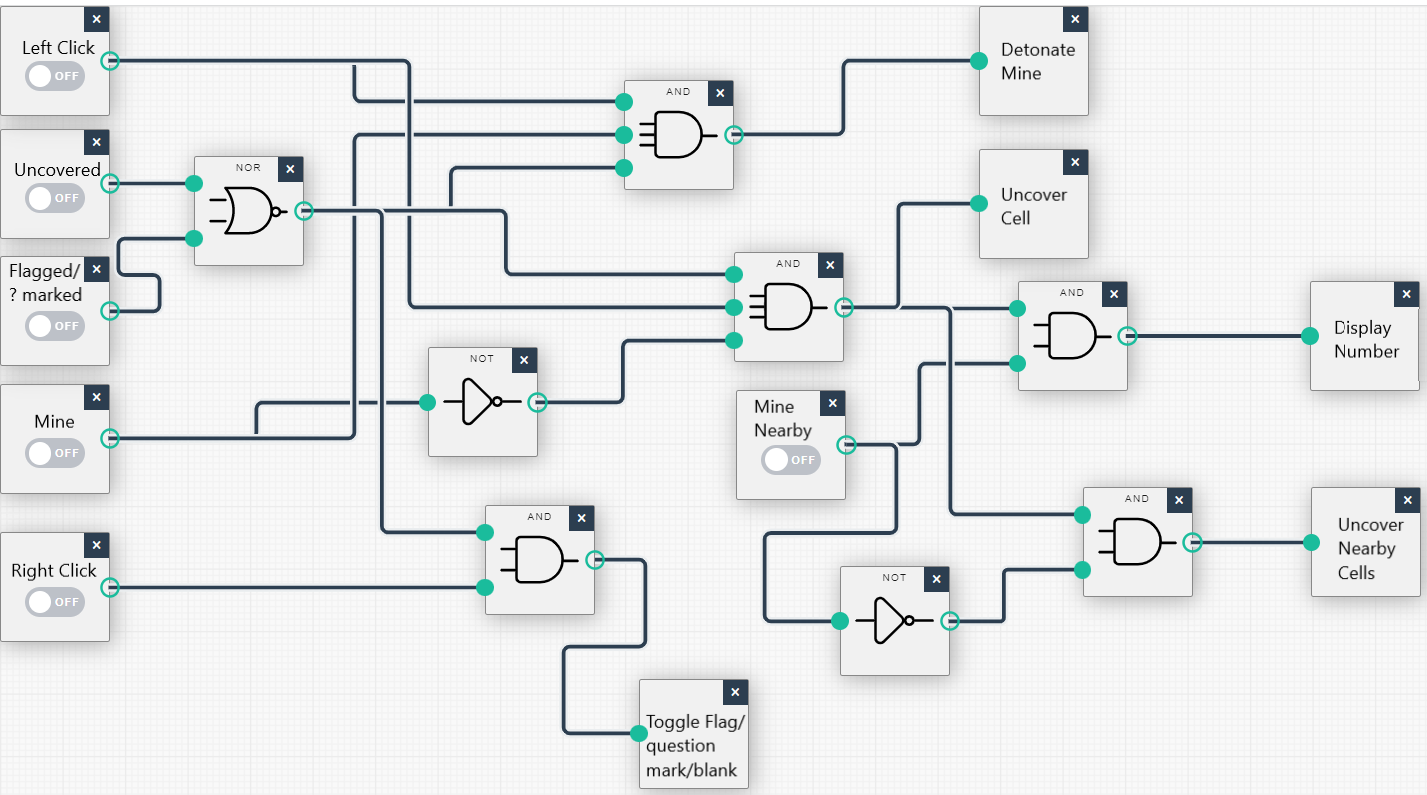
Truth table:

This truth table will only focus on the circuits for the new output as the rest of the diagram has not changed.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| E | B | B’ | EB’ | V |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 |

Version 2

Amended after testing in Task 5.



The version adds one additional input, whether the cell contains a Flag or Question mark (F), and a NOR gate. This adds the functionality to prevent cells containing flags/question marks from being revealed if the user left clicks on them. The NOR gate means cells that contain a flag/question mark OR have already been uncovered can’t be uncovered.

‘Flagged/? Marked’ will be Abbreviated to F

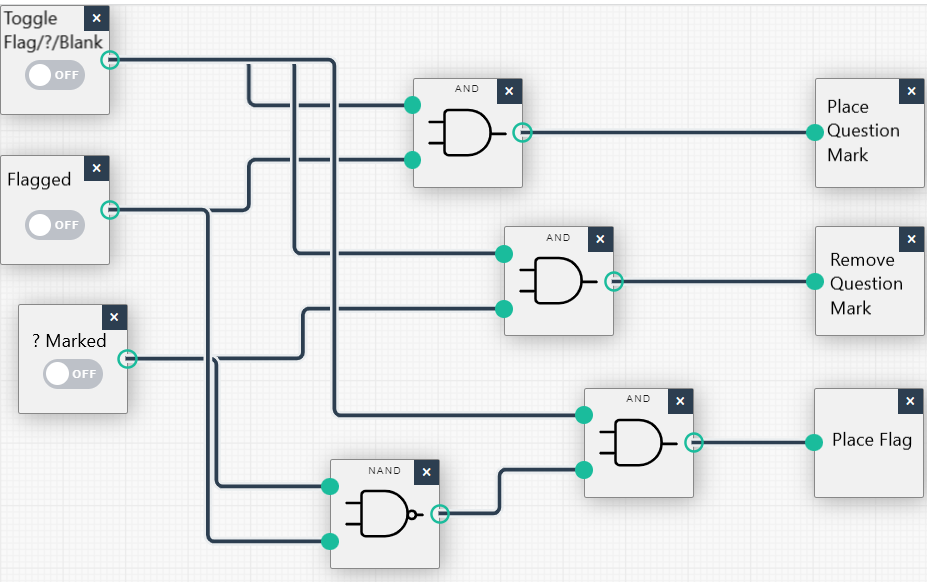
Equation for NOR gate output: NOR gate output = NOT (B OR F)

Truth table for NOR gate output

|  |  |  |
| --- | --- | --- |
| B | F | (B + F)’ |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

Toggle flag/question mark/blank logic Version 1

When the ‘Toggle flag/question mark/blank’ output is triggered, some additional logic will be needed to determine what will happen.



This is the first version of the logic that will be trigged when the ‘Toggle flag/question mark/blank’ output is activated in the main diagram. I had to separate this from the main diagram to avoid running out of space.

This diagram has three inputs:

* Toggle flag/question mark/blank output has been triggered (1A)
* The Cell contains a flag (1B)
* The cell contains a question mark (1C)

And three outputs:

* Place a question mark (replaces a flag) (1X)
* Remove question mark (1Y)
* Place Flag (1Z)

Inputs abbreviated to 1A, 1B and 1C respectively. Outputs abbreviated to 1X, 1Y and 1Z respectively.

Equations:

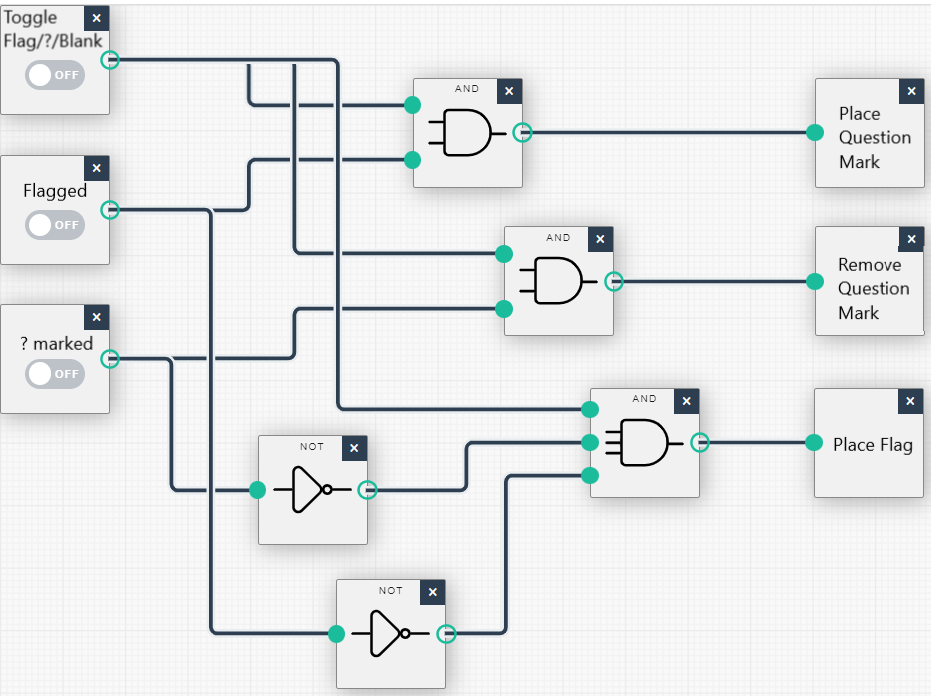
* Place question mark (1X) = 1A AND 1B
* Remove question mark (1Y) = 1A AND 1C
* Place flag (1Z) = 1A AND NOT (1B AND 1C)

Truth Table:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1A | 1B | 1C | 1A.1B | 1A.1C | (1B.1C)’ | 1A.(1B.1C)’ | 1X | 1Y | 1Z |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |

This truth table shows that the logic for the ‘Place Flag’ (1Z) output is not functioning correctly. There should only be one combination of inputs that produce the ‘Place Flag’ output, but this truth table shows there are three. Investigation revealed this was caused by the NAND gate. Placing the inverter after the AND comparison is what causes the issue. Instead, both inputs should be inverted before being fed into the AND gate.

Toggle flag/question mark/blank logic Version 2



This version of the diagram fixes the issue in the previous version by inverting the 1B and 1C inputs before they are compared with input 1A to determine output 1Z. This version also uses a triple AND gate instead of two double AND gates for output 1Z.

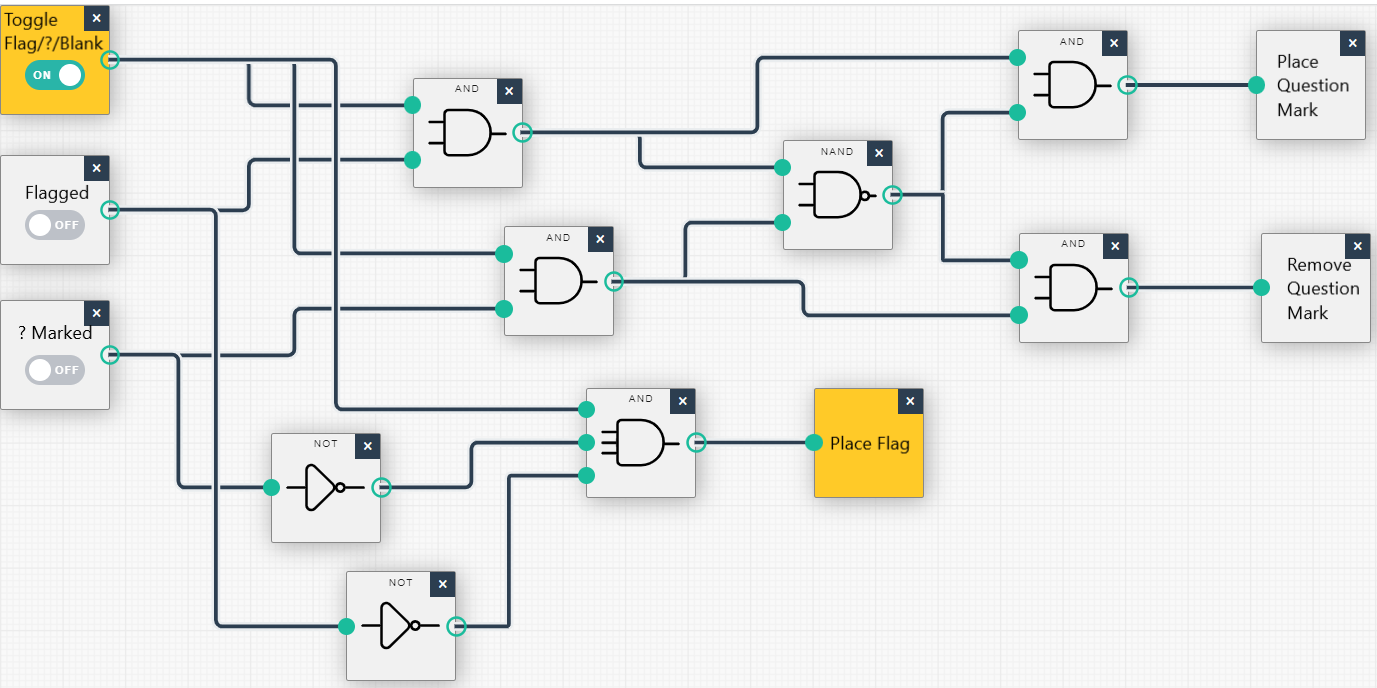
New Equation for output 1Z: Place Flag (1Z)= 1A AND NOT 1B AND NOT 1C

Truth Table:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1A | 1B | 1C | 1B’ | 1C’ | 1A.1B | 1A.1C | 1A.1B’.1C’ | 1X | 1Y | 1Z |
| 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 |

This truth table shows this issue has been resolved as there is now only one combination of inputs that will trigger the ‘Place Flag’ (1Z) output. However, it also shows there is a combination of inputs that will cause the program to attempt to place and remove a question mark at the same time. While this input combination should be impossible to achieve (as it would require a cell to already contain both a flag and a question mark) I want my program to be robust so I will attempt to remove this possibility.

Toggle flag/question mark/blank logic Version 3



This version adds the logic to prevent ‘Place Question Mark’ and ‘Remove Question Mark’ from being triggered simultaneously.

Equations:

* Place question mark (1X) =

1A AND 1B AND ((1A AND 1B) AND NOT ((1A AND 1B) AND (1A AND 1C)))

* Remove question mark (1Y) =

1A AND 1C AND ((1A AND 1C) AND NOT ((1A AND 1B) AND (1A AND C1)))

* Place Flag (1Z) = 1A AND NOT 1B AND NOT 1C

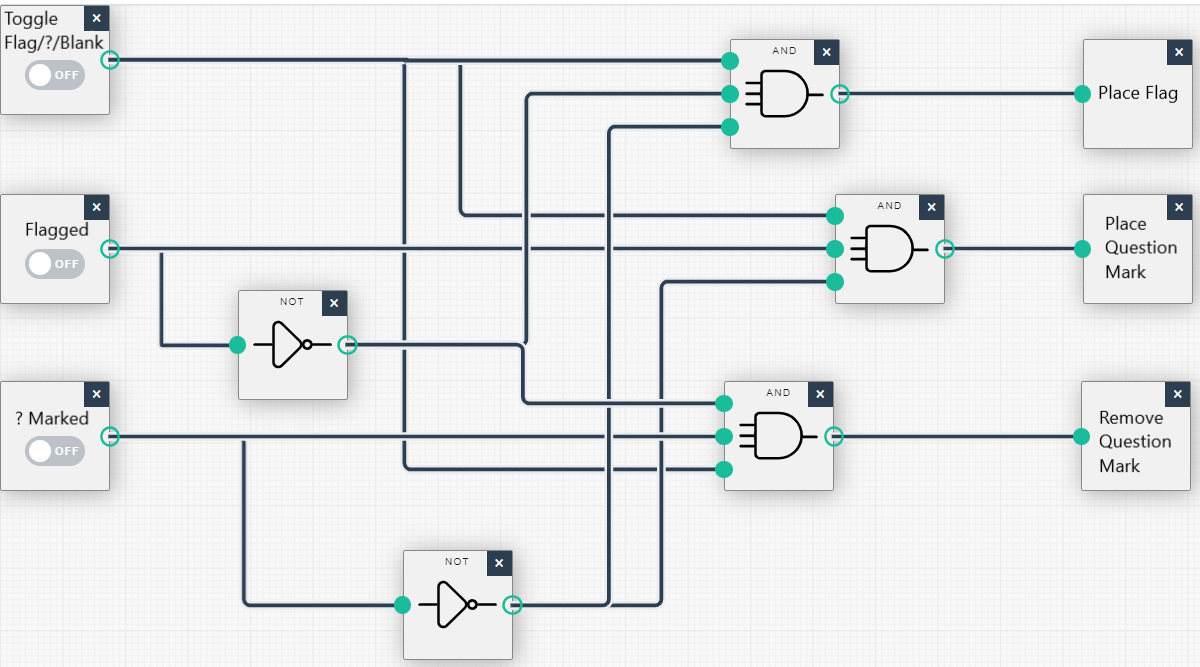
Truth Table:

The logic for output 1Z has not been altered so it will be excluded from this truth table to save space.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1A | 1B | 1C | 1A.1B | 1A.1C | ((1A.1B)(1A.1C))’ | 1A.1B.  ((1A.1B)(1A.1C))’ | 1A.1C  ((1A.1B)(1A.1C))’ | 1X | 1Y |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |

This new truth table shows there is no longer any combination of inputs which will result in outputs 1X and 1Z being triggered simultaneously. However, the equations for outputs 1X and 1Y are very complicated and can likely be refactored.

Toggle flag/question mark/blank logic Version 4



This version is a refactor of the previous version. It makes use of triple AND gates to reduce the number of gates by three.

N.B. The position of the outputs has changed to make the diagram more readable. Outputs will still be referred to with the same abbreviations.

New Equations:

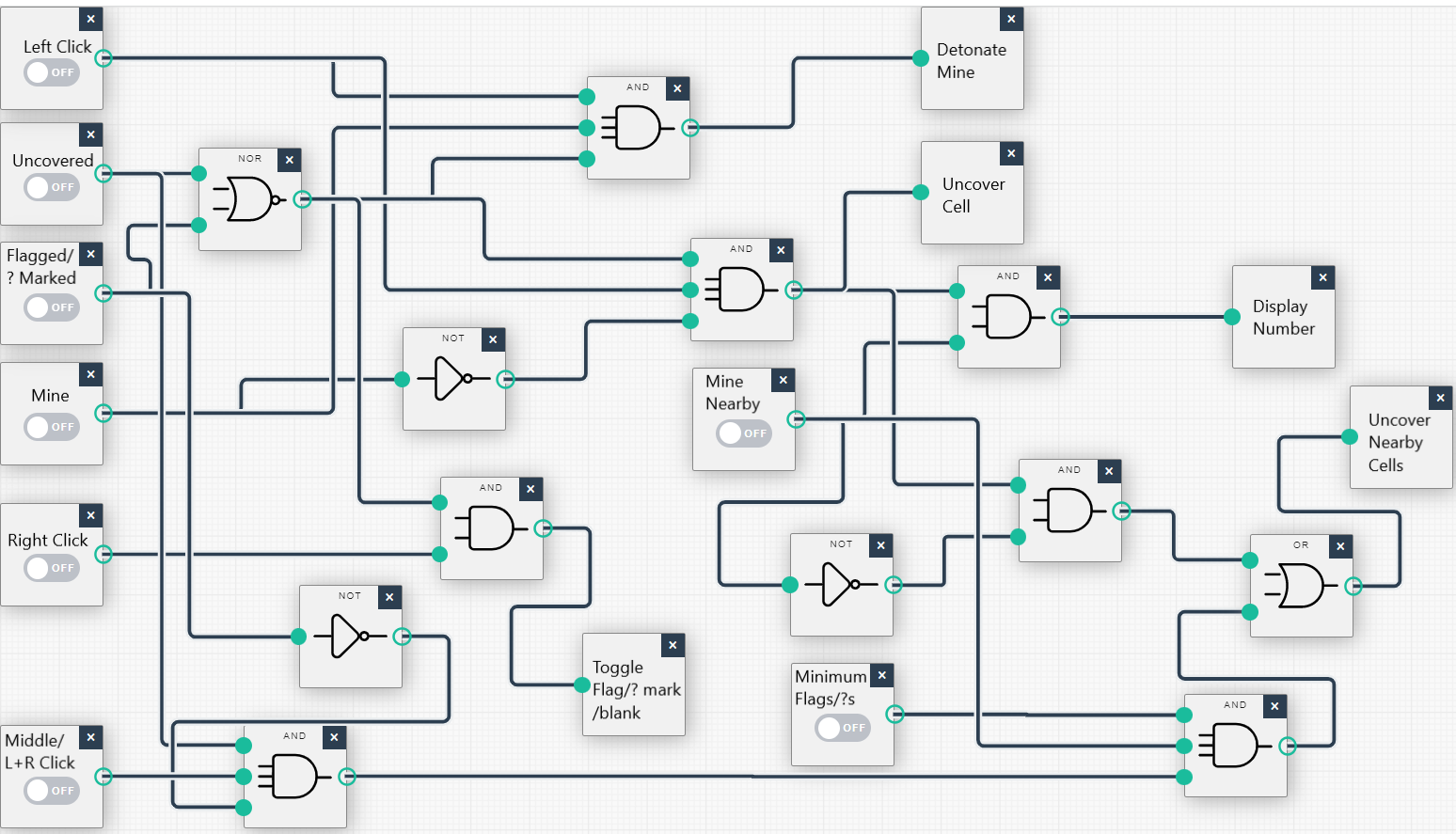
* Place Question Mark (1X) = Place Flag = 1A AND 1B AND NOT 1C
* Remove Question Mark (1Y) = 1A AND 1C AND NOT 1B
* Place Flag = 1A AND NOT 1B AND NOT 1C

Truth Table:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1A | 1B | 1C | 1B’ | 1C’ | 1A.1B.1C’ | 1A.1B’.1C | 1A.1B’.1C’ | 1X | 1Y | 1Z |
| 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

# Scenario 3 Logic Gate Diagram

Version 1



This diagram adds the logic circuit for the middle click/Left + Right click functionality.

This diagram adds two additional inputs to the diagram from scenario 2:

* Middle Click/Left + Right Click (G)
* Minimum number of flags present (H)

The new inputs shall be abbreviated to G and H respectively.

It Also reuses the ‘Uncover Nearby Cells’ output which now has an OR gate connecting it to the two possible activation circuits. The input ‘Minimum Number of Flags’, checks if the cell the user middle/L+R clicked is surrounded by a greater or equal number of flags/question marks than surrounding mines.

The new equation for this method of triggering ‘Uncover Nearby Cells’:

Uncover Nearby Cells (W) = (G AND B AND NOT F AND D AND H) OR ((A AND NOT B AND NOT C) AND NOT D

Truth Table:

To avoid creating a 256-input combination truth table, I will only deal with the new logic circuits, and I will be ignoring the other side of the OR gate to activate output W. Nothing else in the diagram has changed so including it would be unnecessary.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| G | B | F | D | H | F’ | GBF’DH | W |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |

# Task 5

See Caleb\_Eason\_SDV503\_Assessment\_1\_Testing.xlsx for testing tables.

## Scenario 1

All inputs produced the expected output. There were no apparent user experience weaknesses.

## Scenario 2

Step 4, user left clicks on cell B3 did not produce the desired output. The flag on the cell should have prevented the program from uncovering the tile and detonating the mine underneath. However, this was not the case as the program produced the detonate mine output. To fix this, another input needs to be added to the circuit diagram to test if a cell contains a flag/question mark. If at any time this input is true, the program should not be able to produce any output.

See Scenario 2, Version 2 for implementation of this new condition.

Upon retesting the updated logic diagram, all the user inputs produced the correct output.

## Scenario 3

All inputs produced the expected output. There were no apparent user experience weaknesses.

# Task 6

## User Manual

Minesweeper is a simple, 2D, logic puzzle game. The object of the game is to identify the location of a number of mines within a grid without setting any off. The player can left click on a cell within the grid to uncover it. Uncovering the cell will reveal how many mines are present within the surrounding 8 cells. For example, if a user clicks in the middle of a 3 by 3 grid and reveals the number two, this indicates there are at least 2 mines within the 8 surrounding cells. If the user uncovers a cell that has no mines nearby, the program will reveal all nearby cells in a ripple effect until a cell containing a number is revealed.

If the user thinks they know the location of a mine, they can right click to place a flag. Flags server as a visual indicator that there is a mine in the marked cell. Placing a flag will not confirm to user if a cell contains a mine and does not uncover the cell. The user can right click again to remove a flag. Some versions of minesweeper allow the user to place a question mark if they are unsure about the presence of a mine. In these versions of minesweeper, right clicking on a flag will change it into a question mark instead of removing it. The question mark can then be removed by right clicking again.

If the user middle clicks or simultaneously left and right clicks on an uncovered cell containing a number, all surrounding cells will be revealed, but only if the user has placed the same or a greater number of flags in the surrounding cells as the centre number. For example. If the user has uncovered a cell with a number 3 and has placed 3 or more flags in the surrounding area, they can use this feature to instantly reveal all other surrounding cells. However, if the user has only placed 2 or less flags in the surrounding area, this feature will not be usable. It is worth noting that if the user has not placed the flags in the correct position, this action can uncover a mine, causing the user to lose the game.

The user wins the game by uncovering every cell that does not contain a mine. If the user uncovers a cell that contains a mine, the mine will detonate, and the user will lose the game. When the user loses, all the cells containing mines will be revealed and any incorrectly placed flags/question marks will be removed. The user does not need to place a flag on every mine to win the game. The user’s time for each game will be displayed, starting from when the first cell is uncovered and ending when the game is finished.

## Developer Guide

This version of minesweeper takes place in a 3 by 3 grid. There are 3 possible user inputs and 6 program determined inputs for each possible action. Each user action has 8 possible outputs including no output.

User input:

* Left click.
* Right click.
* Middle click or left + right click.

Program determined inputs:

* Cell is uncovered.
* Cell contains mine.
* Cell contains flag.
* Cell contains question mark.
* Cell has mines nearby.
* The number of flags surrounding a cell is greater than or equal to the number of mines.

Possible outputs:

* Detonate mine and end game in a loss.
* Uncover cell.
* Display number of surrounding mines in uncovered cell.
* Uncover surrounding cells.
* Place a flag.
* Place a question mark.
* Remove a question mark.
* No output.

There is also 1 extra input and output that is not linked to the main logic diagram of the program. This extra input will activate once the user has uncovered every cell that does not contain a mine. When this activates, it will directly trigger the output to end the game in a win.

The program architecture is detailed in pseudo code under section 3. This code is written as plain text with indentation to show different structures. The game architecture is split into three sections:

* The initialisation which the program will perform each time it is run.
* The main loop which will last for the duration of the game.
* Functions which can be called from the main loop during gameplay.

The program functionality is shown in logic gate diagrams under section 4 along with truth tables to display the produced outputs from all possible inputs. Note that scenario 4, version 1 shows the final design for the logic gate diagrams. There are also 2 additional diagram that are not incorporated into the main logic diagram. See Toggle flag/question mark/blank logic Version 4 under Scenario 2 and Mine Nearby logic diagram under Scenario 1. Note that the input and output for ending the game are not displayed in the logic diagram this condition does not require any gates. The input is routed directly to the output.