Project report: Pinn Booranamaitree

Outline at a high level what your project is designed to accomplish.

This should be understandable by a layperson (a grandparent for example). Try to avoid using any especially technical words.

# Part 1: Planning

## 1.1 Planning tools & techniques

Discuss the tools and techniques you used. Why did you choose them? Include things like a Gantt chart if you did one. Show you know what tools are available and why they are useful.

Git hub

Trello

Kanban

Burndown chart

Manage the time my task … mvp. prioritize the time. It is more important to finish the game and the writing

### 1.1.1 Actual use of the planning tools

Give examples of how you used the tools and techniques you selected. What was the impact of using them? How did they contribute to the success (or otherwise) of your project?

Show that you not only know **how** to plan a project but that you actually **did** plan a project. If you wrote out a project plan, include that.

It is fine to write about things that didn’t work as well as things that did.

### 1.1.2 Version management

Discuss version management and version control tools. Which techniques or tools did you choose to use? Why? How did your tool impact on your project for good or bad?

## 1.2 Project components

Go into more detail about what the project was about. You can be a little more technical here, but try to avoid too much detail.

### 1.2.1 Problem decomposition

How did you break it up into manageable chunks? How did you break them up further? What was the rationale behind the way you broke the project up into smaller pieces? When you talk about the various components you ended up with you can go into detail if you think it helpful. The person reading this may not necessarily be a coder though. They are interested in why you broke things up into parts, not how the parts work internally.

Trello/comment on the program

What parts are necessary to make your program work? How did you come to the conclusion about the makeup? Remember any problem can be solved in an almost infinite variety of ways, so why did you pick your way? Tell a story that helps the reader to understand your decision-making process.

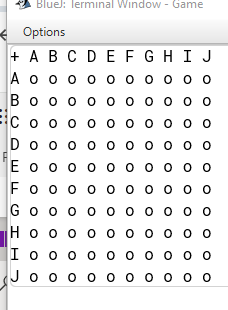
Trialing Document

|  |  |
| --- | --- |
| Date | What is the thing I am trialing |
| 14/05/24 | GUI of the game. |

*Copy and paste the table above before filling it out*

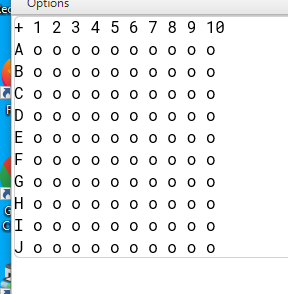
### Version A

### Coordinate x and y have the same label which is hard to understand.



### Version B

X and y coordinates have different types of label (alphabet and number).



Person: Mr. Fairhall (my teacher)

Date: 14/05/24

Comment: From version A if the user puts the coordinate of e.g. [D][C] the user might not know which grid they interacted with (because there are two possibilities DC or CD)

### Evaluation and next steps

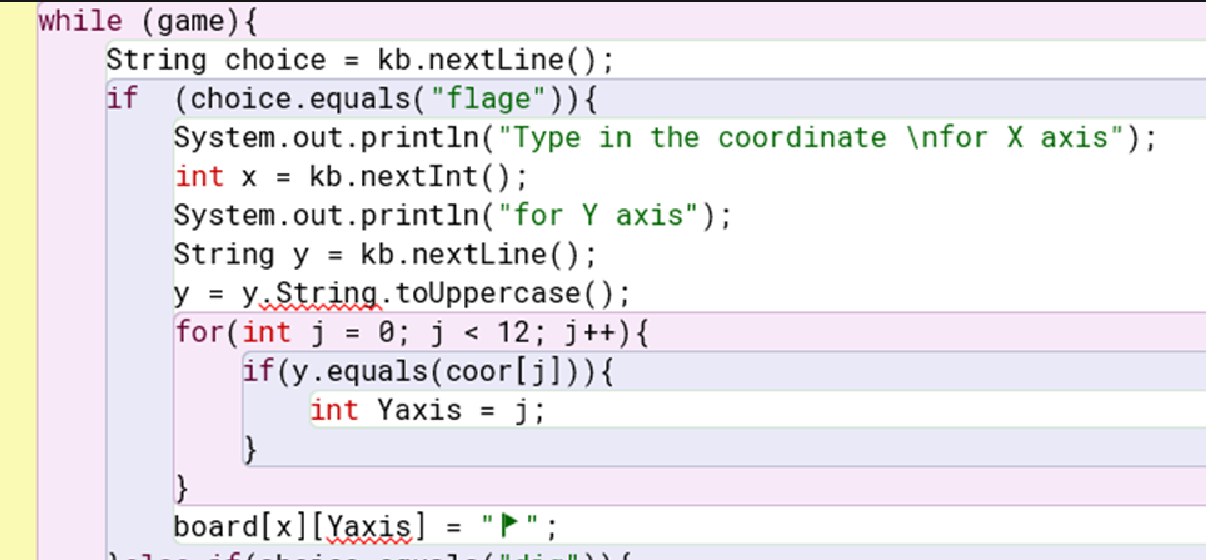
Based on this trialing and the feedback, I have decided to use version B because its easier to understand and use.

|  |  |
| --- | --- |
| Date | What is the thing I am trialing |
| 21/05/24 | Find out the Y coordinate as an integer for the arrays (because our y coordinate is alphabet) |

*Copy and paste the table above before filling it out*

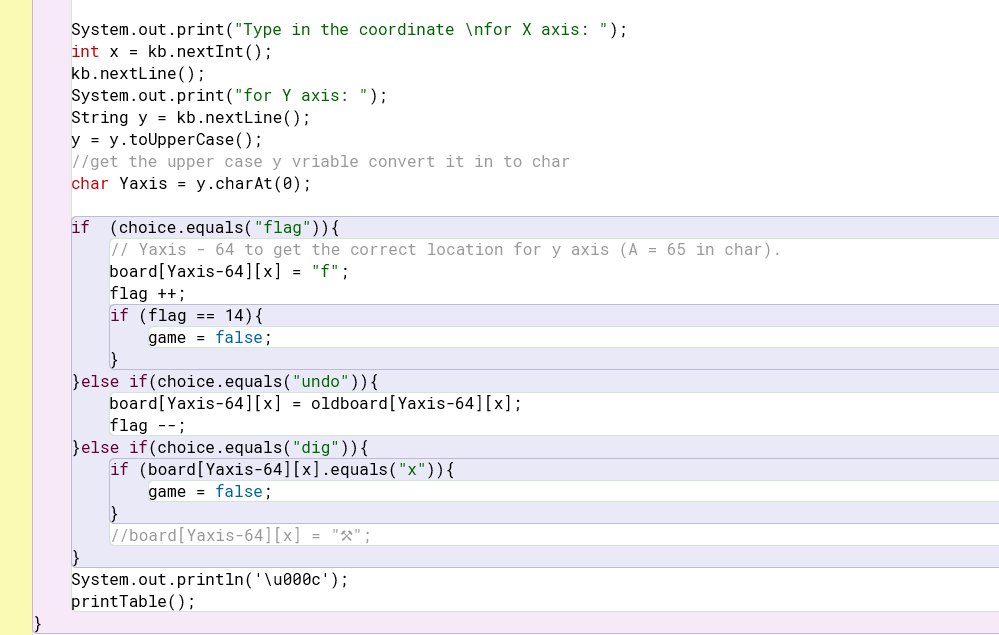
### Version A

To find the y coordinate I use a for loop to loop through the arrays of y coordinate [a, b, c, d…] and keep track of how many times we loop through [ j ] then set it to equal to the y axis coordinate which over complicate it. The better way to do it is to use char.



### Version B

Because char is an ASCII it has a value in int as well (A = 65) so I converted a string of y axis to char then minus it with 64 to get the y coordinate. Now it works fine with minimal code (compared with loop through the coordinate as string).



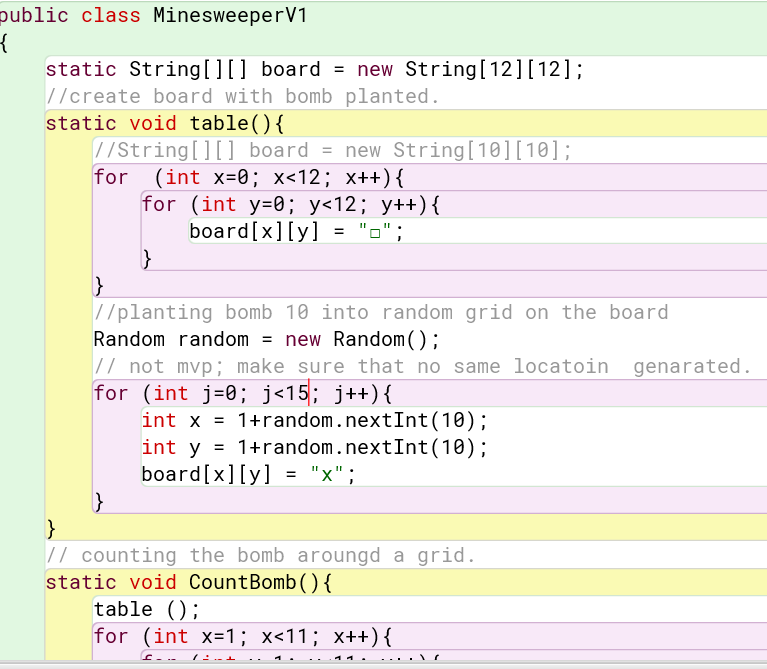
Evaluation and next steps

Based on this trial I decided to use version B because version B is faster, more efficient, and requires less code for the same output.

|  |  |
| --- | --- |
| Date | What is the thing I am trialing |
| 5/06/24 | Different ways of counting a bomb around each cell. |

### Version A

To count how many bombs are around a cell I created a method that takes the input of x and y coordinates then I ran a check on the cell around it ( [x-1, y-1] [x, y-1] [x+1, y+1]… ) if it contains a bomb or not but this will not work if the cell that we are checking is on the corner or at the rim of the board so what I do is create a bigger board (12x12) and only use the middle (10x10) part of it so if the cell that is being checked is at the edge of 10x10 board it will still works fine because the board is 12x12 so there will be no index out of range error.



### Version B

I used a normal-size board (10x10) and passed the surrounding coordinates ( [x-1, y-1] [x, y-1] [x+1, y+1]… ) to another method called countbomb() which takes x and y coordinates as an argument. If the x and y coordinates are out of range the method will return 0 but if that cell is not out of range and contains a bomb the method will return 1 to the main Checkbomb method which will keep track of the amount of the bomb.



Evaluation and next steps

Based on this trial, I have decided to use version B because the board size of the board is being used throughout the entire game. If I use a size 12x12 board when I print the board out or generate a bomb inside the board the coordinates have to be in a range of 1 – 10 and the number of board lengths is not consistent throughout the game which made the game quite hard to deal with. Also, I think version B is easier to understand for everyone in case anyone wants to develop this project.

|  |  |
| --- | --- |
| Date | What is the thing I am trialing |
| 11/06/24 | User try the game 1 |

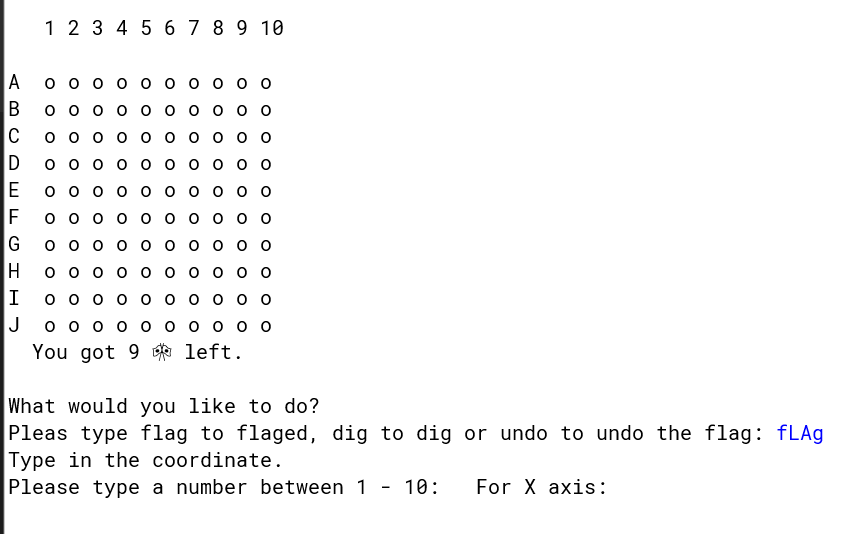
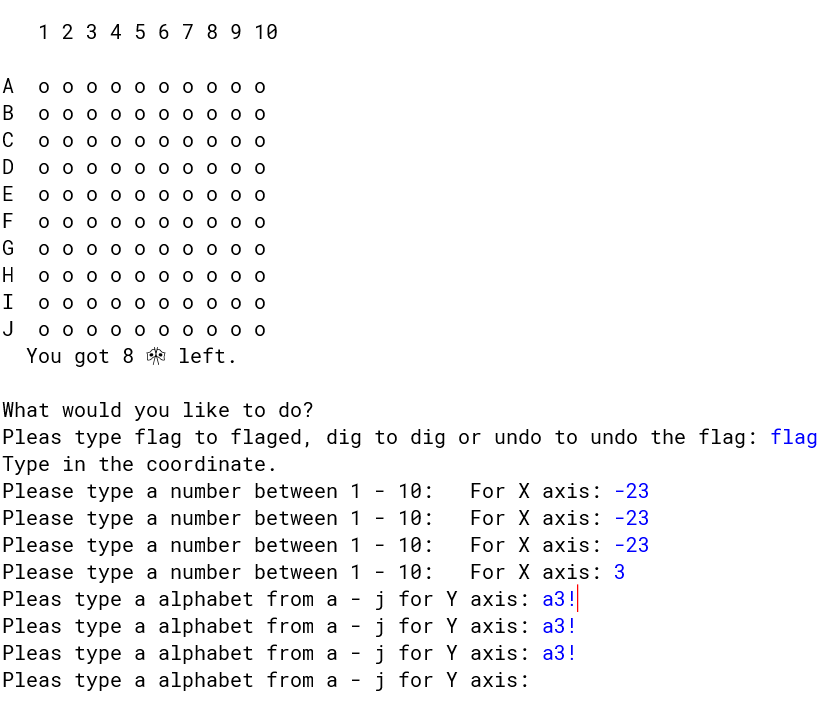
### Feedback

Person: Huy (my classmate)

Date: 11/06/24

Comment: According to Huy, the game is really hard to play because if the player types something wrong, the game will end with an error immediately.

I put error checking into my game by passing every input into a method and will loop through until I get the valid input. if the player types something wrong the program will loop through until it gets the valid input. This will also work when the user inputs a wrong case e.g. [fLaG] the program will still work fine.



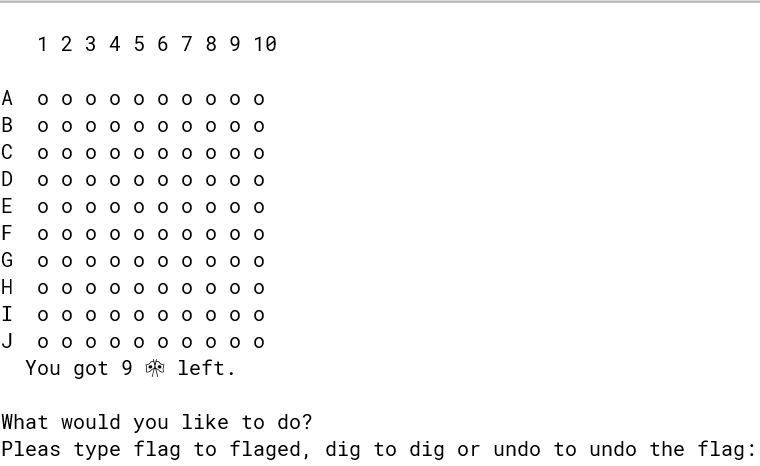
|  |  |
| --- | --- |
| Date | What is the thing I am trialing |
| 13/06/24 | Users try the game 2 |

Person: Aksel (my friend that try the game)

Date: 3/05/24

Comment: According to Aksel he suggests that I put an undo flag option in the game because if the player accidentally places a flag at a wrong spot or places it wrong then finds out later they cannot undo it.

I have decided to add an undo option to my game, now the player can undo the flag if they make any mistake.



## 1.3 Relevant implications

Your project has an impact on people. These are encapsulated in the relevant implications.

### 1.3.1 Relevant implication #1 Usability

*Describe the relevant implication in general terms*

Usability refers to the user's ability to interact with the product or the system easily and should be able to achieve their goal efficiently and effectively. There are many aspects when talking about usability such as simplicity, intuitiveness, and user satisfaction. In software or game development usability ensures that the product is accessible, engaging, and straightforward, minimizing any potential error or confusion that could reduce positive user experience.

*Explain how it impacts on your project.*

Usability is a crucial key to a successful game. A game that is difficult to understand and navigate or play can lead to user frustration and negative experience and might cause the player to not continue playing the game. Ensuring high usability means that the game should have clear and easy-to-follow instructions for the player. The player should be enjoying the game experience and having fun, and the game should also be accessible to a diverse group of players with varying levels of prior gaming experience.

*Discuss what you have done to address this implication.*

Clear instructions: I have designed very straightforward and easy-to-follow instructions in the game which guide the player step by step throughout the game. The program will also tell the player if they did anything wrong e.g. “Please type a number between 1-10” or “Please remove the flag be for dig the cell”.

User interface design: I did a lot of trailing on this and came up with a very easy-to-navigate user interface with labels on the x and y-axis. The program refreshes every time the player finishes in that round and prints a new update display.

Error handling: I have implemented robust error handling in my program. If the player types some invalid input the program will tell the player and loop through until it gets a valid input. I also added a feature so that if the player uses capitalized letters randomly the game will still work fine. I also ran a lot of tests and minimized the bugs that could happen during a game.

### 1.3.2 Relevant implication #2 (give the name)

Describe the relevant implication in general terms

Explain how it impacts on your project.

Discuss what you have done to address this implication.

### 1.3.3 Relevant implication #3 (give the name)

Describe the relevant implication in general terms

Explain how it impacts on your project.

Discuss what you have done to address this implication.

|  |  |
| --- | --- |
| **Implication** | **Why is it relevant?** |
| usability | The game should be able to play with minor errors and be easy to understand with clear instructions. Also, it should be easy to use for everyone (e.g. color blind). |
| sustainability and future proofing | A programmed that easy to maintain and update overtime, program that can be scale and extend in the future time. Programs with enough information on the code good structure that easy to understand for future development. |
| end-user considerations | The end-user should have a positive experience with the game. The game should run smoothly without any bugs or errors. An easy to use and understandable game user interface and game control e.g. how players interact with the game (keyboard, mouse). How the game communicates with the user e.g. wrong input, win or lose. |

# Part 2: Testing

Test subject: Input taking/interacting with the board

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Test | Expected | Actual | Date | Notes |
| Expected cases | Inputting coordinates for a particular cell.  [1, a] | The player can interact with cell [1, a] (in this case flag the cell) |  | 15/05/24 | Coordinate for the y-axis is a string but it has to be an integer to locate a particular cell e.g. board[3][3] |
| Inputting coordinates for a particular cell.  [1, a] | *The player can interact with cell [1, a] (in this case flag the cell)* |  | 15/05/24 | Change the Y coordinate from string to char because char can refer to a number. |
| Inputting different options  [Dig]  [6, e] | After selecting the cell, that cell should be revealed. |  | 15/05/24 |  |
| Inputting different options  [flag]  [9, f] |  |  | 15/05/24 |  |
| Inputting different options  [undo]  [9, f] | The cell that is [F]  Should go back to “o”. | ➡️ | 15/05/24 |  |
|  |  |  |  |  |
| Invalid cases | Inputting coordinates for a particular cell. But using invalid input  [-23, a3!] | Loop through until get a valid input. |  | 11/06/24 |  |
| Inputting an Invalid option  [ fl4, fl%, di, di#, un!, und@ ] | Loop through until get a valid input. |  | 11/06/24 |  |
| null | It should also loop through until it is valid. |  | 11/06/24 |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| Boundary cases | Inputting  Coordinate at the edge of the table.  [Flag]  [1, a], [10, a]  [1, j], [10, j] | Should be able to work fine. |  | 11/06/24 |  |
| Coordinate of y-axis with capital letters.    [flag]  [5, G] | Should be able to work fine. |  | 11/06/24 |  |
| Inputting different options with random capitalized letters.  [fLAg] | Should be able to work fine. |  | 11/06/24 |  |
| Undo a cell that have no flag  [undo]  [5, h] | The program should tell the player that they can’t undo this cell. |  | 11/06/24 |  |
| Flag the cell that has already been open  [flag]  [1, g] | The program should tell the player they can’t place a flag here. |  | 11/06/24 |  |

Test subject: Revealed a number

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Test | Expected | Actual | Date | Notes |
| Expected cases | When opening an empty cell, it should expand out. |  | \*\*stack overflow error. | *2/06/24* | The recursive function that checks the cell around one cell checks each other back and forth causing an infinite loop. |
| When opening an empty cell, it should expand out. |  |  | 2/06/24 | If the cell already checks and is empty change that cell to “0” Then if the cell is “0” return because it has already been checked. |
| When opening a cell that contains a number  [flag]  [7, g] | It should reveal the number. |  | 2/06/24 |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| Invalid cases |  |  | Invalid cases such as out-of-range coordinates or wrong input for the y-axis (not a-j) when digging are already tested in the input taking part. |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| Boundary cases | When digging a cell that has already been flagged  [dig]  [8, a] | Should not reveal the cell. |  | 7/06/24 | It works fine but it should also tell the player that there is a flag there so the player knows. |
| When digging a cell that has already been flagged  [dig]  [8, a] | The program should also tell the player that they have to remove the flag first. |  | 7/06/24 |  |
| When digging a cell that has already been flagged and that cell contain a bomb  [dig]  [9,a] | The player should not lose because there is a flag and they dig. Even though the bomb is there but there is also a flag so the program should tell the player first. |  | 7/06/24 | It should work fine but there is some error in the code. |
| When digging a cell that has already been flagged and that cell contain a bomb  [dig]  [8,d] | The player should not lose because there is a flag and they dig. Even though the bomb is there but there is also a flag so the program should tell the player first. |  | 7/06/24 | The error is because I check if there is a bomb in that cell before if there is a flag so if there is a bomb the player will lose straightway.    I fix it by changing the odder of if/if else statement so the program checks if there is a flag first then check if there a bomb.    It works fine now. |

Test subject: End of game/win/lose

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Test | Expected | Actual | Date | Notes |
| Expected cases | When all the flags are used and the flag is at the correct location. | Tell the player that they won. |  | 10/06/24 |  |
| When all the flags are used and not all the flag is in the correct location | Tell the player that they lost and say how many bombs they missed. |  | 10/06/24 | The flag is at the wrong place but it still says the player wins. There is something wrong |
| When all the flags are used and not all the flag is in the correct location | Tell the player that they lost and say how many bombs they missed. |  | 11/06/24 | Before fixing the error    After fixing the error    The error is because the if statement for checking the flag will run if display[x][y] = “F”. But I use “f” instead the program did not run the check at all. |
| If the player digs a bomb  [dig]  [4, f] | Tell the player that they dug a bomb and they lose. |  | 10/06/24 |  |
|  |  |  |  |  |
|  |  | - |  |  |
| Invalid cases |  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| Boundary cases |  |  | - |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

# 

## 2.4 Changes resulting from trailing and testing

Did you make any major changes as a result of testing? If so, what, and why?

One of the major changes in my program was when I did a trial on different ways of counting bombs around each cell. The first way was creating a bigger board of 12x12 and using the inner part of the board to prevent index out of range and the second way was to create another method that takes in the surrounding x and y coordinate as an argument and checks if that cell contains a bomb or not and it will return 1 if there is a bomb or 0 if there is no bomb or the index is out of range. Based on this trial I decided to go with the second version because as I said it is way easier to keep the board size the same all the time because we use this variable a lot throughout the whole game which will make the code less complicated and easier to understand to everybody in case anyone wants to develop it further.

The other change that I made after I did a trial with the user was to make my program more robust and easier to play. In my first version, I did not do any error checking at all so when my friend tried to play it and type something thing wrong the whole game just stopped and an error text was shown on the screen and they had to start all over again. So based on this trial, I decided to add error checking into my program, by passing every input into a method and looping through until the program gets a valid input. I also programmed that if the user use wrong capitalize the program will still work. On another trial with another user, they suggested I put an undo option in the game so that if the player found out later that they flagged a wrong cell the undo option would come in handy.

I make a lot of changes in my program according to my testing results. For example, I did a lot of testing on revealing numbers on a cell after the player dug it. The test shows that when I try the boundary case for example digging the cell that has already flagged nothing should happen and the program should tell the player that they cannot dig here and have to remove the flag first but instead, the game ends, and the player is lost. This should not happen, so after I found the error in the code I went back to my program and fixed it. I would not have found this out if I had not done the testing so testing helps me to improve my game.

In my opinion, trialing is helping me to get a better outcome for my program because after I get feedback from my friends and after I run the trial myself, I can see a different way of doing things and different problems. This helps me see the bigger picture of my program and helps me decide which solution is the best. Testing also plays a crucial part in my program. It helps me identify the problem, keep track of it, and address it correctly which lead to higher quality outcomes.

# 3. Summary

All the planning, trialing, and testing has led to a great final program. Talk about that, summarise how each of these (both individually and in combination) have contributed to that excellent final product that you have produced.

Provide evidence from your project to back up your discussion.