Department of Computer Science

BSc (Hons) Computer Science (Software Engineering)

Academic Year 2021 - 2022

Algorthm - An Educational Algorithms Visualizer

Majd Barakat

1945214

A report submitted in partial fulfilment of the requirements for the degree of Bachelor of Science

Brunel University London
Department of Computer Science
Uxbridge
Middlesex
UB8 3PH
United Kingdom
T: +44 1895 203397
F: +44 (0) 1895 251686

Abstract

Algorthm is an educational web-application created to visualize various algorithms with the aim of changing the traditional method of teaching algorithms and data-structures in the field of computer science. This report was written to document the issues behind the ways of teaching a core subject in computing as well as propose a possible solution.

Acknowledgements

Firstly, I would like to thank my supervisor Dr. Alina Miron for assisting me through every phase of this project's development. I am extremely grateful for her input and help.

Many thanks to my family who supported me throughout my university experience every day.

Finally, thank you to my fellow students that participated in my studies and aided me in completing this report.

I certify that the work presented in the dissertation is my own unless referenced.

Signature [illegible]

Date April 8th 2022

Total Words: 12382

Table of Contents

List of Tables

List of Figures

Chapter 1 Project Overview

1.1 Introduction

In computer science, the traditional method of teaching algorithms is through diagrams and pseudocode, which is not considered engaging or interactive to most students. As algorithms are complex, it is difficult for those with a limited understanding of the topic to visualise the pseudocode and imagine how the algorithm would function. Whilst there are learning resources available such as video examples of how the algorithms run, the learner does not have any degree of control: they cannot change the environment, experiment for themselves, or interact with what they are learning. The purpose of this project is to contest this flawed and lacklustre teaching method by providing an interactive and visual solution to understanding algorithms, offering the student a much larger degree of engaging learning opportunities.

Schools and universities worldwide teach algorithms above to their computer science students, but with the rise of online lectures/classes and the decline of interactive and visual teaching, students struggle to remain focused and immerse themselves in the material presented to them (UIS).

An algorithm is a pre-defined set of step-by-step instructions required to execute a variety of functions. This can be traced back as far as 300 BC, found inscribed on Babylonian clay tablets with instructions and formulas on tracking grain stock and cattle (Norman, 2021).
The term “Algorithm” can be dated back to 850 AD. It came to life in a time where “The father of Algebra” Abdullah Muhammad bin Musa al Khwarizmi the Persian mathematician and astronomer wrote a book in which his name was translated to Latin as “Algorithmi” (University of Klagenfurt).

Pathfinding algorithms are very important as they are used in many applications such as Google Maps, navigation systems, computer games, and routing packets over the internet. Some common examples of pathfinding algorithms are:

- Dijkstra’s
- A* Search
- DFS (Depth First Search)

1.2 Aims and Objectives

The main aim of this project is to provide a tool that can use a selection of different algorithms to generate mazes and find the shortest path solutions, all the while educating users about the individual algorithm's features. For this project to successfully achieve this goal, ten objectives must be fulfilled:

- **i.** Identify and evaluate applicable software engineering methodologies and apply their structure in this project's development journey.
- **ii.** Thoroughly research the chosen algorithms for maze generating and pathfinding and the technology to use when implementing the application.
- **iii.** Design a user interface that allows the users to switch between algorithms, set a start and an end, and change the size of the grid/maze.
- **iv.** Apply useability engineering principles throughout the entire process of designing the UI.
- **v.** Create a function that allows users to draw their own mazes/environments using their mouse.
- **vi.** Implement two ways to generate random mazes to provide the pathfinding algorithms different environments to work with.
- **vii.** Implement two algorithms to find the shortest path between two cells.
- **viii.** Test and evaluate the user experience through questionnaires before and after using the application.
- **ix.** Write a detailed report on the project going through all development stages and discoveries upon completing the report.
- **x.** Implement any significant changes suggested through testing feedback and discuss plans in concerns of the future of this application.

1.3 Project Approach

The primary method used to tackle this project would begin with background research into every used algorithm with distance calculations. The research will include usability engineering principles to assist the development of the application and help align itself with professional standards in UX (user experience).

A significant part of this project's development will involve a series of minor tests to ensure the algorithms work in the varied environments correctly. During the development phase, there will be a recurring cycle of testing and rapid development of fixes (should any problems arise) to ensure that the program can withstand user misinput or intentional attempts at breaking functionality.

This project will be developed using the Waterfall methodology. The Waterfall methodology requires a strict list of objectives to be achieved before the design and development stages take place. Every step forward using the method cannot begin without completing the previous step. It is an advantageous model to use as it is linear and specific. A project of this size benefits from this method greatly, mainly as it allows issues to be tackled in a chronological order, offering a steady progression of the project efficiently and succinctly.

1.4 Dissertation Outline

Chapter 1 – Error! Reference source not found.**:** This chapter introduces the problem and some of the background research briefly. Aims and objectives are set clearly in the chapter and a short section discussing the approach that will be taken to complete this project.

Chapter 2 – Research & Literature Review: This chapter defines the problem as well as propose a solution. Research is presented from currently available solutions to an in-depth analysis of every algorithm implemented.

Chapter 3 – Error! Reference source not found.**:** This chapter discusses the approach I took to the development of this project.

Chapter 4 – Design: This chapter displays all the designs and diagrams made before beginning the implementation phase.

Chapter 5 – Implementation: This chapter presents the process of implementing main and side features of the project and how they were all integrated into a designed user interface.

Chapter 6 – Error! Reference source not found.**:** This chapter presents and discusses results of series of tests that were carried out to identify bugs or inconveniences in the program. Minor revisions were done based on the results.

Chapter 7 – Error! Reference source not found.**:** This chapter concludes the report. It evaluates the accomplishment of the project by referring back to the original aims and objectives set in the Introduction. Limitations and future work are discussed in this chapter.

Chapter 2 Research & Literature Review

2.1 Introduction

This chapter will thoroughly review relevant research and literature and summarise the findings. The research will extend over a few topics, such as the seamless implementation of grid structures and user experience, to an extensive analysis of maze generating and pathfinding algorithms. Before addressing the technical research, briefly contextualising the background of this project is required to further explain what the project aims to achieve.

2.2 Background

2.2.1 The Problem

There is a widespread perception that computer science classes are difficult to engage with, particularly for those with little to no experience with the subject. This misconception is rooted in many factors. For example, studies found that introductory computing classes focus on enervating details (Jepson & Perl, 2002), potentially turning many students away from the topic early on. This problem may also be disproportionately affecting women entering the field. According to a study done by AAUW (1998), girls are significantly less likely to enrol in advanced computer science and graphics courses in comparison to data-entry or spreadsheet typing classes. This can partially be due to gender expectations; however, improving entry-level classes may help retain marginalised identities that were historically alienated from the subject.

While algorithms and programming are typically a person’s first exposure to computer science, it fails to excite many students. A report from Helsinki University of Technology (Kinnunen & Malmi, 2006) points out that the dropout rate for their introductory course in programming is between 30 to 50% a year, and almost half of the cases’ reasoning was difficulty and lack of motivation.

Programming in a conventional environment is considered stale and monotonous. The same thing can be said about learning algorithms without an interactive and beginner-friendly option, as opposed to reading diagrams and pseudocode. Developing interactive and visual tools in the ever-evolving online learning environment (Babson Survey Research Group, 2015) may help aid learners enjoy learning, increase engagement, and prevent dropping out (Moreillon, 2015). Everyone digests knowledge through a variety of mediums, but most people are visual learners, meaning that they process images, videos, or any visual stimuli much faster than other learning methods (Mayer & Massa, 2003).

2.2.2 The Solution

Algorithms and Data Structures play a crucial role in computing, yet many people disregard the topic (Upadhyay, 2022) due to their preconceived notion of its complexity and difficulty. This is expected, as traditional teaching methods include very minimal interactive tasks or fail to provide visual means of learning about the topic.

The visualisation of algorithms displays how algorithms work in a graphical manner. It helps simplify how complex the algorithms seem, as well as deepen the knowledge and understanding of their structure and operations. Pathfinding and sorting are the two most used categories of algorithms that many developers encounter, making them essential to understand to at least some degree.

Interactive learning is paramount, especially within a field that is often laborious and difficult to digest. Sadikan and Yassin (2012) found in a survey of students within The National University of Malaysia; there was a clear distinction in exam results between students using 'traditional learning methods' (such as the use of textbooks along with videographic publication on websites) against students utilising an interactive tool. Students in the former group yielded significantly higher results than the latter.

The project's focus is directed toward pathfinding algorithms due to the difficulty of visualising them without the help of interactive tools. Pathfinding algorithms have many applications that extend terrain analysis and road building to navigation systems (Krayzman, et al., n.d.).

2.3 Currently Available Solution (Visualgo)

Visualgo by Erin Ling (2015) is a web-based learning tool that aims to improve the methods of educating data structures and algorithms through visualisation and interaction. This tool offers a great variety of algorithms and data structure visualisation. It provides an extensive amount of information upon entering the application. Whilst it prides itself as a powerful tool to test your understanding and learn data structures, the website is overwhelming. To a person with limited knowledge on the topic or just starting out, it seems difficult to understand. The website also looks and feels dated, as seen below in Figure 2-1.

This tool's shortcoming is the inability to offer beginners an easy and straightforward visualisation of the algorithms. Visualgo is geared towards those with prior knowledge of the algorithms. While it has many algorithms and data structure options to visualise, it does not have a pathfinding and maze generating visualiser.

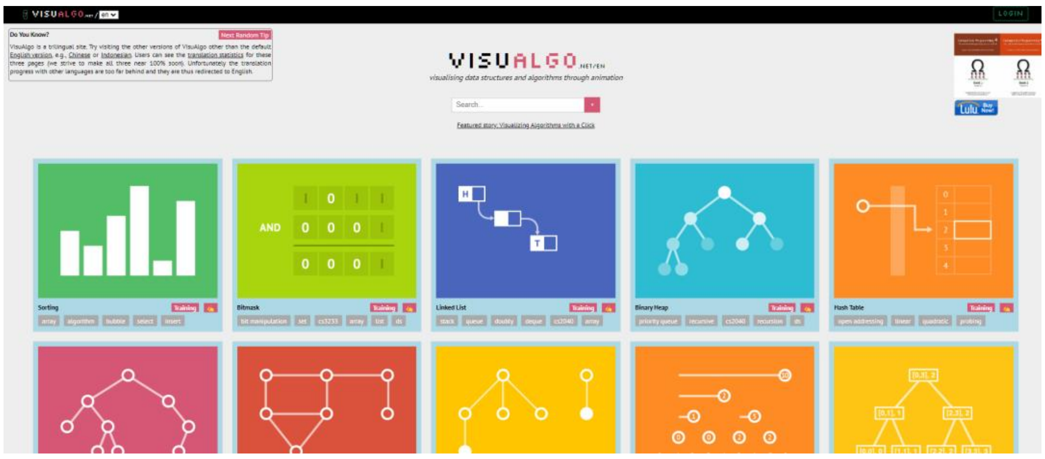


Figure 2-1 – Visualgo screenshot

2.4 Algorithms Environment

For users to further understand the operations and comprehend the differences between algorithms, a randomly generated environment is required. To compare the different outcomes of the same algorithm, the user would need to observe the pathfinding algorithms solving mazes of different shapes and paths. One way to achieve a unique environment is to allow the user to allocate the walls and passages with complete freedom. Another would be to provide a method of generating random mazes using a variety of algorithms.

There are two main approaches to creating a maze within a grid (Buck, 2015):

1. Nodes contain 4 sides where each could be allocated an attribute of being a wall or a path (Referred to as *linewise*).
2. Nodes themselves can be allocated an attribute of being a wall or a path (Referred to as *blockwise*).

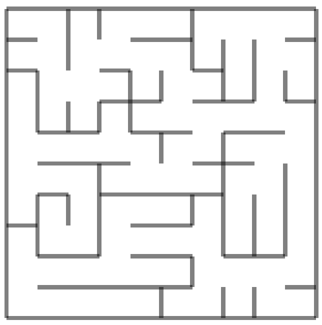


Figure 2-2 – "linewise" maze type (Buck, 2015)

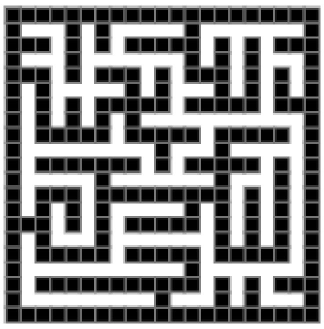


Figure 2-2 – "blockwise" maze type (Buck, 2015)

2.5 Distance Heuristics

There is a variety of distance heuristics that will assist the pathfinding algorithms to produce the shortest path solution. Each heuristic can be matched with a movement type, both within and outside of a grid (Yang, 2019) (Garcia, 2018):

Manhattan(L1) – The sum of absolute differences, used on a grid of squares that only allows **4 directions** of movement.

$$L_1 = |x_1 - x_2| + |y_1 - y_2|$$

Chebyshev/Diagonal(L∞) – The max value of the differences, used on a grid of squares that allows **8 directions** of movement.

$$L_\infty = max(x_2 - x_1 , y_2 - y_1)$$

Euclidean(L2) – The square root of the sum of differences squared, used on movements that are not necessarily confined within a grid but can move in **any direction** usually using.

$$L_2 = \sqrt{(x_2- x_1)^2 + (y_2 - y_1)^2}$$

2.6 Pathfinding Algorithms

There is a variety of pathfinding algorithms available to integrate. But the ones that will be researched and discussed are the most popular that are also very likely to be implemented in the final project.

2.6.1 Depth First Search (DFS)

DFS, also referred to as Recursive Backtracker, will find a solution between two nodes, but that does not necessarily mean that the solution found is the shortest one (Pullen, 2021). DFS was initially created to detect cycles in a graph as well as for performing topological sorting (Chakraborty, 2019).

Its core basics consist of creating a stack with a size equivalent to the maze storing all the visited cells. If the algorithm reaches a wall, it recursively checks the previous cells of the stack for a possible path until the end is reached.

Prerequisites:

1. Start node exists
2. Every node contains a Boolean attribute of visited/unvisited

Pseudocode:

```
1. Create a stack of visited nodes and another of unvisited nodes
```

```
2. Add start node to the visited nodes stack
3. While solution is not found, and unvisited stack is not empty:
    1. Check for unvisited neighbour nodes of the top node in the visited list
    2. If there are no available neighbour nodes remove the top node from the visited
       stack
    3. Else remove the neighbour node from the unvisited list and append the node to
       the visited list, change the nodes state to visited
```

2.6.2 Dijkstra's

Unlike Depth First Search, Dijkstra's Algorithm finds the shortest distance between the starting node and the end node. Dutch computer scientist Edsger Dijkstra proposed his algorithm to be applied to weighted graphs in 1959. Since then, Dijkstra's algorithm remains one of the most used pathfinding algorithms to this day (Buck, 2015).

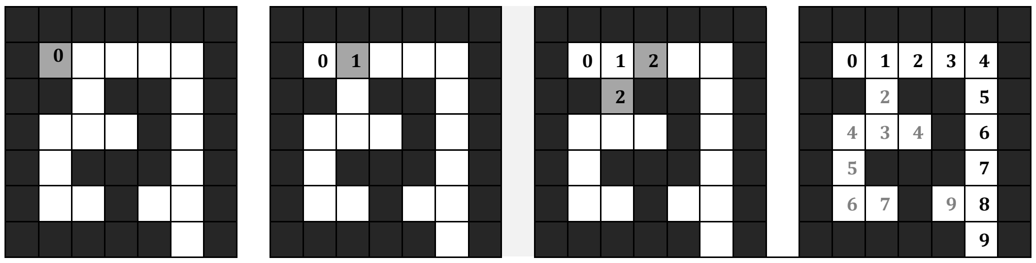


Table 1 – Step-by-step Dijkstra's Algorithms

Dijkstra's algorithm operates by marking the starting node with a distance of 0 and expands in levels where every level is marked with a distance based on the level (e.g., nodes marked on the 2nd level of expansion will be given a distance of 2) (see Table 1).

Dijkstra's Algorithm has one cost function, which is the real cost value/distance from the source that's applied to each node:

$$f(x) = g(x)$$

Prerequisites:

1. Start node exists
2. Every node contains an attribute of visited, unvisited and a distance value

Pseudocode:

```
1. Create a set containing all unvisited nodes
```

```
2. Set distance to 0
3. Create a stack storing all the nodes in the level and append the starting node to it
4. While solution is not found, and unvisited set is not empty:
    1. For every node in the stack:
        i. Set the distance attribute to the variable distance and set state to visited
        ii. Check for unvisited neighbour nodes and store them
    2. Replace the stacks nodes with the neighbours discovered
    3. Increment distance by 1
5. If the solution is found trace back to the start by drawing a path on nodes with
   decrements of 1.
```

2.6.3 A* Search

A* search algorithm is used to find the shortest path using the same principles as Dijkstra's. Though it provides a faster solution depending on the type of environment it is in, it is commonly used in most pathfinding applications such as navigation systems and games AI (Isaac Computer Science).

A* is considered to be a variation of Dijkstra's since it functions similarly. The key difference between the two is that A* has two cost functions (Patel, 2011):

$$f(x) = g(x) + h(x)$$

$g(x)$: Same as Dijkstra's, this calculates the real distance between the node and the source (start).

$h(x)$: An addition to Dijkstra's, this is a heuristic function that approximately calculates the cost form the current node to the final node. It is referred to as an admissible heuristic, meaning that the cost it calculates to reach the goal will never exceed the lowest possible cost from the current node to the end node.

Prerequisites:

1. Start node and End node exists
2. Every node contains an attribute of visited, unvisited, f-score, and g-score

Pseudocode:

```
1. Create a set containing all unvisited nodes each having a pre-set h-score and another
   with all visited nodes
2. Set distance/g-score of all unvisited nodes to $\infty$, which also means an f-score of $\infty$
3. Set start node's distance/g-score to 0 and an f-score based on the h-score it was given in
   the unvisited set
4. While solution is not found, and unvisited set is not empty:
    1. Set current node to the node in the unvisited set with the lowest f-score
    2. For each neighbour of the current node:
        i. If neighbour is not in the visited set:
```

        1. Calculate new g-score = weight of edge + g-score of current node
        2. If new g-score is less than neighbour's g-score in unvisited set:
            a. Update the neighbour's g-score with the new g-score and f-score = new g-score + h-score
            b. Update the neighbour's previous node to the current node
    3. Copy the values for the current node from the unvisited set to the visited set
    4. Remove the current node from the unvisited set
    5. Return the visited list
5. If the solution is found draw the path backwards from the end using the determined f-score of the nodes decremented by 1 with every node

2.7 Maze Generating Algorithms

By definition a maze is an intricate and complex network of paths, and with the help of algorithms, random and unique mazes can be generated to satisfy the need of distinctive environments. Different maze types need to be established and understood before covering the different types of mazes generating algorithms.

2.7.1 Types of Mazes

Mazes come in different shapes and forms, in this project there will be two main types of mazes that will be created by algorithms (see Table 2).

Perfect Maze – A maze without any loops, every node is within reach from any start point. It consists of a network of dead ends. There will always be exactly one solution to the maze.

Partially Braided Maze – A maze with a mixture of dead-ends and loops, every node is reachable from any start point, and there can be multiple solutions to the maze.

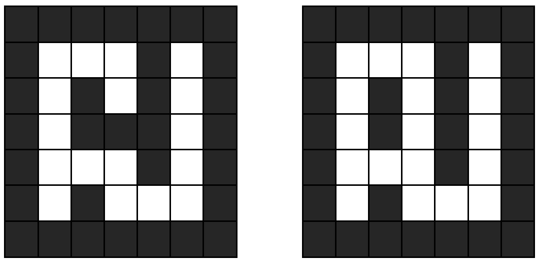


Table 2 – Perfect Maze vs Partially Braided Maze

2.7.2 Recursive Backtracking

Similar to Depth First Search, Recursive Backtracking uses the same principles to generate mazes. This algorithm generates a perfect maze. Recursive Backtracking is a widely used algorithm to generate mazes due to its speed (Buck, 2015). The main downside to it is that it takes up a large sum of space in the memory when generating larger mazes as it will store every single node in that maze.

Prerequisites:

1. Every node contains an attribute of wall/passage and position

Pseudocode:

```
1. Set all the nodes to state wall
2. Create a stack with all the passage nodes
3. Set node 1,1 to current node and change it's state to passage, add it to the stack
4. While stack is not empty:
    1. Check for available neighbours of the top node in the stack (if neighbours that
       are 2 nodes away are in state wall)
    2. If there are no available neighbour nodes remove the top node from the stack
    3. Else change the state of the neighbour and the node between to passage and add
       them to the stack
```

2.7.3 Random

This maze generating method is simply a random maze generator with a density value. The density value will change the chances of the node being a wall or a passage.

Prerequisites:

1. Every node contains a Boolean attribute of wall/passage

Pseudocode:

```
1. For every node, if random number is > density? Set state to wall.
2. Else state to passage.
```

2.7.4 Prim's

Prim's algorithm similar to Recursive Backtracking generates a perfect maze. This algorithm is used to produce a minimum spanning tree, it requires storage equal to the size of the maze therefore could be memory heavy depending on the size of the maze. The operations of Prim's algorithm pick a front node to link to the main passage similar to a branch to a tree. It repeats this until a minimum spanning tree is created.

Prerequisites:

1. Every node contains an attribute of wall/passage and position

Pseudocode:

```
1. Set all the nodes to state wall
2. Set a random node to state passage, and find its frontier node. A frontier node is one that is within the grid, is 2 nodes away and has the state wall
3. Create a list containing all frontier nodes
4. While list is not empty:
    1. Pick a random frontier node from the list
    2. Find neighbours of that frontier node that have the state passage, are 2 nodes away and within the grid.
    3. Pick a random neighbour to link to the frontier node by setting the neighbour to passage as well as the node between
    4. Find frontier nodes to the chosen neighbour, append them to the list and remove the chosen frontier from the list.
```

2.8 Architecture

The technology stack or software architecture is important to establish in the early stages of the software's development life cycle. The architecture dictates major decisions regarding the software based on that technology's capabilities and the experience the developers possess using the stack/language.

In the case of this project, it is apparent that there will be no need for a complex software stack for the application to run smoothly. This project does not deal with any databases or data storage and huge content sizes therefore no need for APIs or a Database. I considered using frameworks such as React.js, but the small size of this project did not justify that. My experience with JavaScript frameworks was not enough for me to implement the software to the quality standards I initially set out to achieve using them.

I will write the front end of the application with:

- **i.** HTML & CSS
- **ii.** jQuery

And the back end with:

- **i.** JavaScript

2.9 Conclusion

This project aims to implement a minimum of **two** algorithms from each type. The types of mazes that will be generated are **perfect** mazes and **partially braided** mazes. I chose to focus more on perfect mazes due to the vast amount of documentation and information available to me from the literature review section. Partially braided mazes will provide the user with various environments to observe the pathfinding algorithms.

2.9.1 Maze Generating Algorithms

I have chosen to focus on **Recursive Backtracking**, and **Prim's** algorithms for maze generating. I have chosen these two as both are frequently used when implementing a maze generator to systems such as computer games. Prim's is more complex than Recursive Backtracking, which makes it a better choice to begin with implementing RB over Prim's. If I find time to implement them, I will write a method to generate a random maze as well as a variant of recursive backtracking where the maze is partially braided.

2.9.2 Pathfinding Algorithms

The main two algorithms I will implement are **Dijkstra's** and **A* Search** algorithms.
I have chosen these two due to their relation to each other, where both follow the same core principles, but A* has an extra heuristic value to calculate. The additional implementation will be DFS (Depth First Search). All these algorithms have different behaviours and applications based on their environment. A* will be the most challenging of all three algorithms, but it is the most used algorithm in pathfinding applications. I will implement A* after Dijkstra's algorithm is working as Dijkstra's works as a foundation for A*.

Chapter 3 Methodology & Approach

3.1 Introduction

To begin the project, some prerequisite requirements needed to be completed and methodology choices that needed to be made. The project's planning begins in this chapter, all decisions made about algorithms to use, principles to apply, and more while using all the background research done in the previous chapter.

3.2 Software Development Life Cycle (SDLC)

There are many different SDLC models, where each have a key benefit and disadvantage over another. There are three main models commonly used in software development: Waterfall, Agile, and Iterative (Boyde, 2014).

I will employ the Waterfall model on this development life cycle. It accommodates the requirements of this project as all of them are structured in a linear manner. For the project to progress through a requirement, the previous one needs to be completed. This model allows me to tackle each issue in chronological order (Sommerville, 2016). Before moving between stages, a thorough evaluation of the previous stage will be done to ensure that it is completed to the best quality.

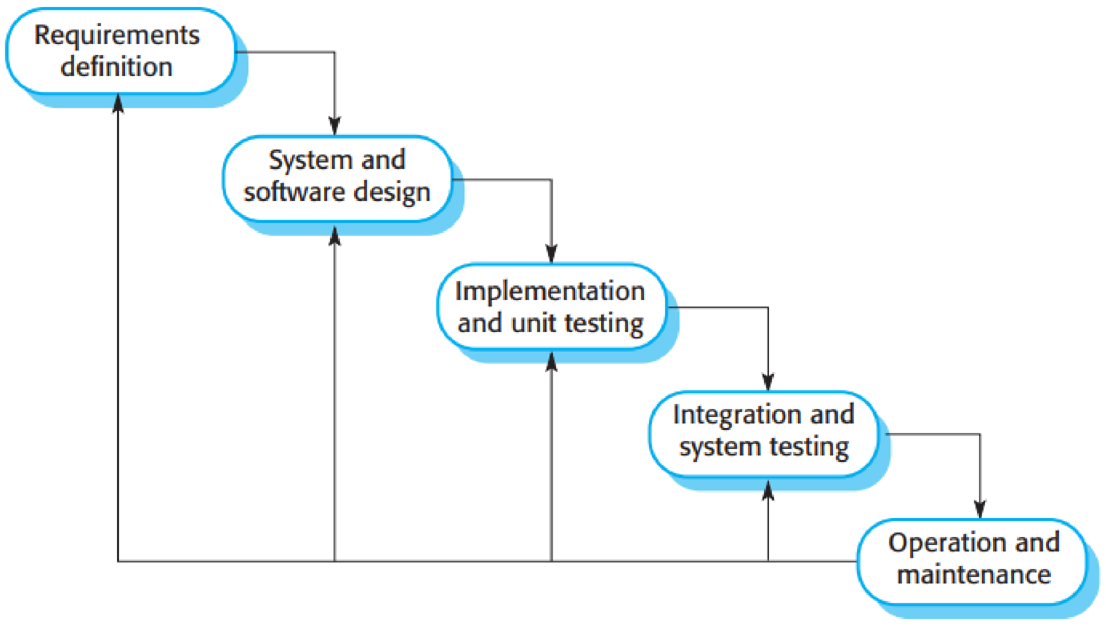

Requirements definition
System and software design
Implementation and unit testing
Integration and system testing
Operation and maintenance

Figure 3-1 – Waterfall Model (Sommerville, 2016)

The main limitation of using this model is that it is impossible to make any changes to previously completed stages unless the project is completed (Sommerville, 2016). This means that after integrating and testing the system, there will be a stage of evaluating test results and amending the software with features based on the findings and consultation of users.

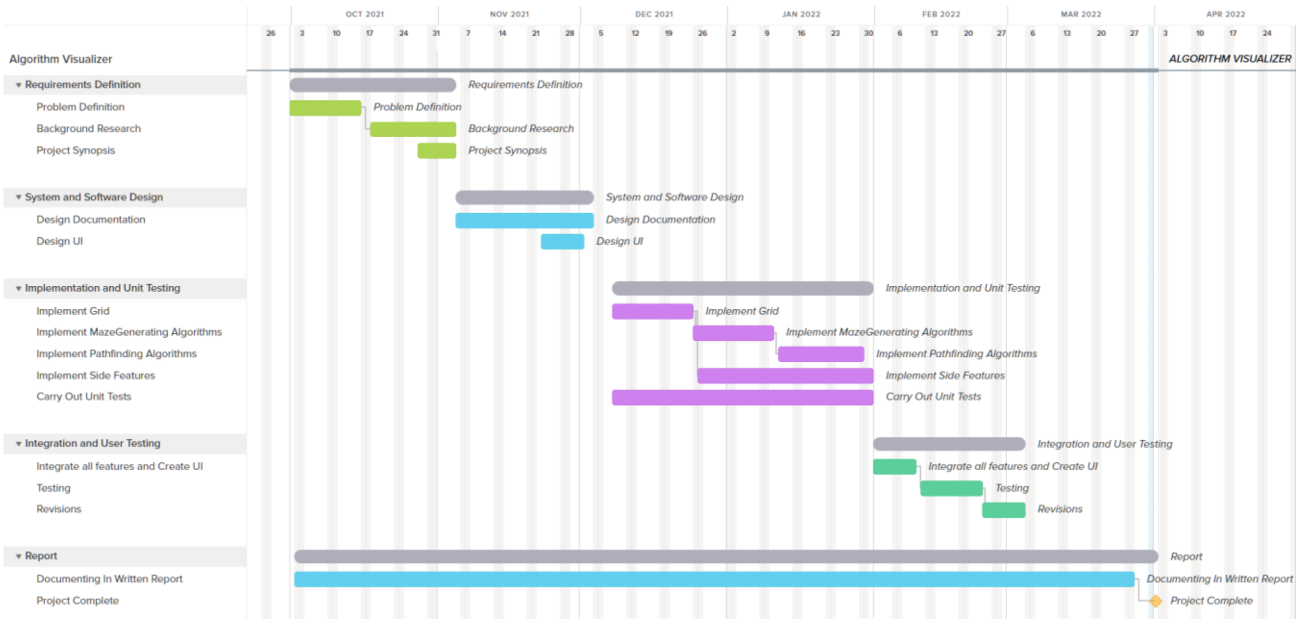


Figure 3-2 Application Gantt Chart

In Figure 3-2, I display my approach to The Waterfall model. I sorted all the tasks into groups based on the same model displayed in Figure 3-1 (Sommerville, 2016). Each step can be linked to a part of this report as such:

Requirements Definition – The requirements can be found in the *Aims and Objectives* sub-section. I identified the main aim of the project and listed the objectives that need to be completed to achieve the aim. Background Research was done to understand the problem in more depth. At the end of this period, a project synopsis was due.

i. **Problem Definition –** Located in *The Problem*
ii. **Background Research –** Located in *Research & Literature Review*

System and Software Design – This period was documented in the *Design* Chapter. During the design phase, I documented and designed various diagrams to assist me in the implementation phase. I also designed a UI for the application.

i. **Design Documentation –** Located in *Documentation & Diagrams*
ii. **Design UI –** Located in *Graphical User Interface*

Implementation and Unit Testing – This section can be found in the *Implementation* Chapter and the *Testing and Evaluation* Chapter. I implemented the features discussed in earlier chapters and tested them.

i. **Implement Grid –** Located in *Grid*
ii. **Implement Maze Generating Algorithms –** Located in *Maze Generation*
iii. **Implement Pathfinding Algorithms –** Located in *Path Finding*

iv. **Implement Side Features –** Located in *Implementing Other Functions*
v. **Carry Out Unit Tests –** Located in *Unit Tests Table*

Integration and User Testing – This section can be found in the *Implementation* Chapter and the *Testing and Evaluation* Chapter. I integrated all the features into a user interface and carried out a couple of questionnaires on participants. Revisions were made based on the results of the questionnaires and unit tests.

i. **Integrate all features and Create UI –** Located in *User Interface*
ii. **Testing –** Located in *Questionnaire Results*
iii. **Revisions –** Located in *Revisions and Fixes*

3.3 Usability Engineering

The main focus of the project is interactivity and an in-depth graphical display of the algorithms, this places functionality and interface quality. Due to the size of this project, it becomes difficult to apply most of useability engineering's principles, as the application consists of a very limited user interface and heavily focuses on what is occurring in the grid/maze. The principles that are will most likely be implemented in the final version of the project are as follow:

Clarity & Conservation of Attention

Clarity is considered the most important principle in UI/UX design (Porter), and for a great reason. It helps the user to conserve their attention to the main purpose of the application or page by not wasting time searching for features or buttons. It is paramount to provide a clear UI where the user knows exactly where everything is positioned, either instantly through predictions or after short use (Memon, 2019). This means clearing the UI of clutters using methods like:

1. **Whitespace –** This provides the user interface an element of minimalism. Whether it is vertical or horizontal, whitespace is very important to space out buttons, texts etc.
2. **Colours –** When considering a colour palette, it is important to keep the options simple. No more than three colours, not including neutral colours, i.e., black, grey, and white.
3. **Menus –** Categorizing buttons, navigation tabs, or side functions will immensely assist in de-cluttering the user interface. This should only be used on secondary objects while keeping the primary functions easily accessible to the user.
4. **Simple Language –** Simplicity has evolved to become the best practice in general design, but it is vital when choosing language for the application's interface. Avoiding technicalities and unnecessary annotations on buttons/features is important to convey the function to the user by relating the word with the user's thoughts (2022).

Accessibility

To ensure as many people benefit from this project, accessibility will be an important element to consider when designing the UI. The following are accessibility principles that are likely to be implemented:

1. **Clear Indication of Error/Warning –** It is important to deliver an error or a warning to the user while using the application for the user to begin rectifying the error quickly without confusion.
2. **Providing a Choice in Colour –** To accommodate for visual disabilities, it is a common practice to allow the users to change the colour of certain aspects of the application where colour is essential. This could be a simple option for high and low contrast colour palettes (Interaction Design Foundation).

3.4 Fault Prevention

To ensure a software is free of faults or bugs, a couple of practices could be applied to decrease or completely abolish faults in the system.

Write Clean Code – One of the most important rules in programming is to ensure that the code you write is clean. What is meant by that is the code can be read and amended to by a developer other than the original author. In his book Clean Code, Robert C. Martin (2008) establishes and discusses many rules that the reader should follow to consistently write clean code. All of which I intend on applying to this project, with the main ones being:

i. **Simplicity** over everything (e.g., small functions set out to do one thing)
ii. **Consistency** throughout the code
iii. **Descriptive naming** (following standard conventions)
iv. **Explain the code** with comments
v. **Eliminate Code smells** – i.e., needless repetition and complexity

Code Review & Refactoring – This is usually done in a team environment where group members would review your code and point out any faults or bad practices and suggest a way to refactor the code. But in this case, it is an individual project, that does not necessarily mean that this practice cannot apply. I will review my own code after having a long break from writing it, and that will lead me to either finding faults and correcting them or simply refactoring the code for increased efficiency.

Software Metrics – Another method of distinguishing faults in a software is by closely observing its metrics. Metrics can include anything from lines-of-code (LOC) to depth-of-inheritance (DIT). This will be difficult due to the small size of this project, but it is still a great practice to carry out nonetheless.

3.5 Source Control Management (SCM)

SCM is an essential tool to utilise, especially when working on large projects with a team of developers. It also offers benefits to individual developers, such as acting as a back-up if the files got corrupted/misplaced or lost, another benefit to using a SCM system is the ability to rollback an update if an unknown fault was to arise.

The SCM system of choice for this project will be GitHub. The main reason I have chosen GitHub is due to my 2 years' experience with Git. It is a great tool for not only storing your projects on but also allows the project to be worked on by multiple devices easily. This project will be uploaded to a public repository on my personal GitHub account (GitHub, 2022).

3.6 Testing

Testing is a crucial process to carry out before the deployment of any software. This will be one of the final steps in the software's development life cycle, the testing stage consists of two methods:

Unit Testing – This will be done throughout the implementation process to verify that every unit meets its specification. It will include a set of tests with an expected outcome and real outcomes, if the real outcome matches the expected one then the test passes. A table of all the results will be presented in the later chapters.

User Testing – This consists of a series of questionnaires asking the users questions regarding their experience using the software. There will be two types of user testing:

i. **User Experience/ Usability Testing** – The users will answer questions about how usable the application was. The questions will be regarding the interface as well as an assessment of the general usability.
ii. **Accomplishment of Aim –** These questions will assess how well the aim of the project was met. In this case the questions will ask about the knowledge of the algorithms before and after using the software and compare the results to evaluate the success of the project in achieving its goal.

The questionnaires are split into two parts, the first being a pre-test questionnaire, it includes questions about the subject's knowledge about algorithms. This is meant to be answered before testing the application. Then a post-test questionnaire including knowledge questions and usability questions. The questionnaires sent to the users are found in the appendix.

3.7 Ethics

Ethics approval was provided by the BREO and can be found in the appendix.

Chapter 4 Design

4.1 Introduction

This chapter aims to display the early stages of the application's development. With the assist of the knowledge gained from the background research on existing applications, the prototyping and implementation phase of the project will begin once the design is completed.

4.2 Documentation & Diagrams

4.2.1 Flowchart

In order to begin developing this application, a clear view of the bigger picture is essential. Flowcharts enable developers to identify all the crucial steps and processes directly through a clear visual representation.

The main functions and purpose of this application can be broken down to a few simple steps. As seen in the diagram, the main structure ensures that the user has met the prerequisites of generating a path or a maze. Then displaying the results accordingly with an offer of clearing the grid and starting over.

I have decided to exclude the extra functionalities when designing this flowchart as it will aid me in keeping focused on the main functions of the application. The other features will be discussed in detail in the implementation chapter.

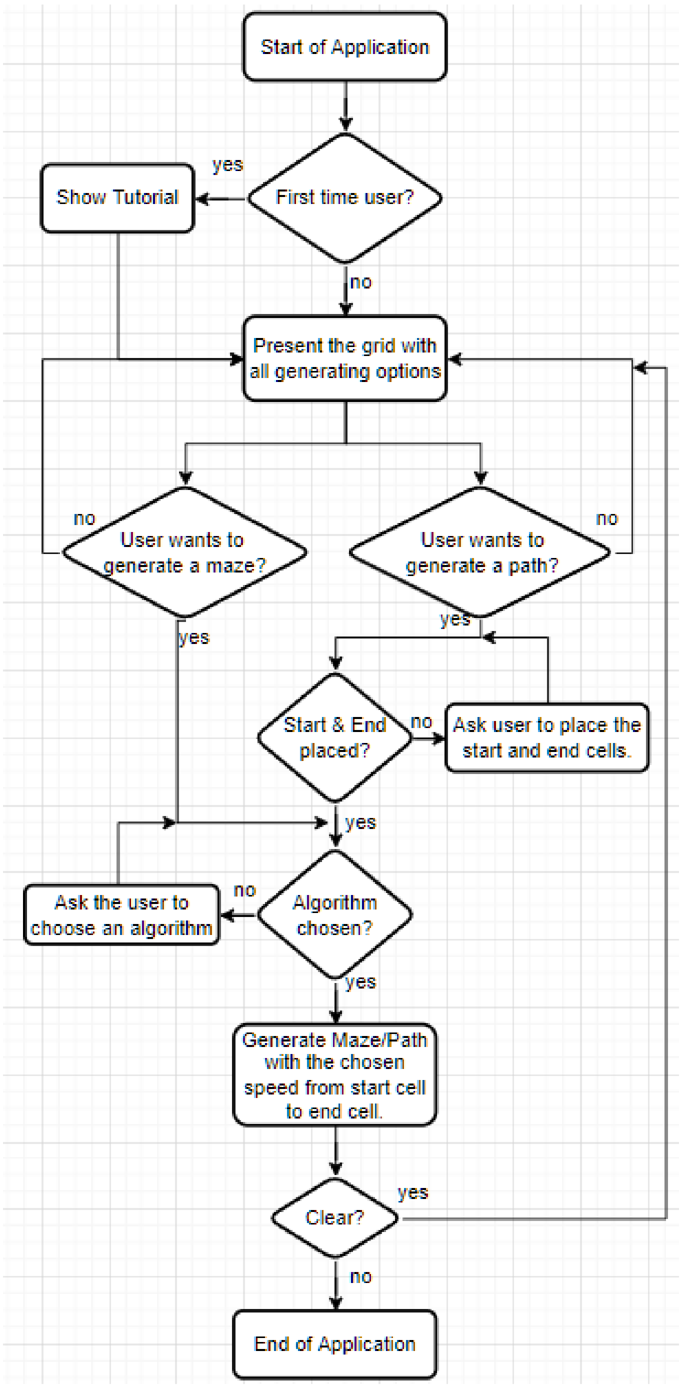

Start of Application
First time user?
yes
Show Tutorial
no
Present the grid with all generating options
no
User wants to generate a maze?
no
User wants to generate a path?
yes
yes
Start & End placed?
no
Ask user to place the start and end cells.
yes
Ask the user to choose an algorithm
no
Algorithm chosen?
yes
Generate Maze/Path with the chosen speed from start cell to end cell.
Clear?
yes
no
End of Application

Figure 4-1 – Application Flowchart

4.2.2 UML Sequence Diagram

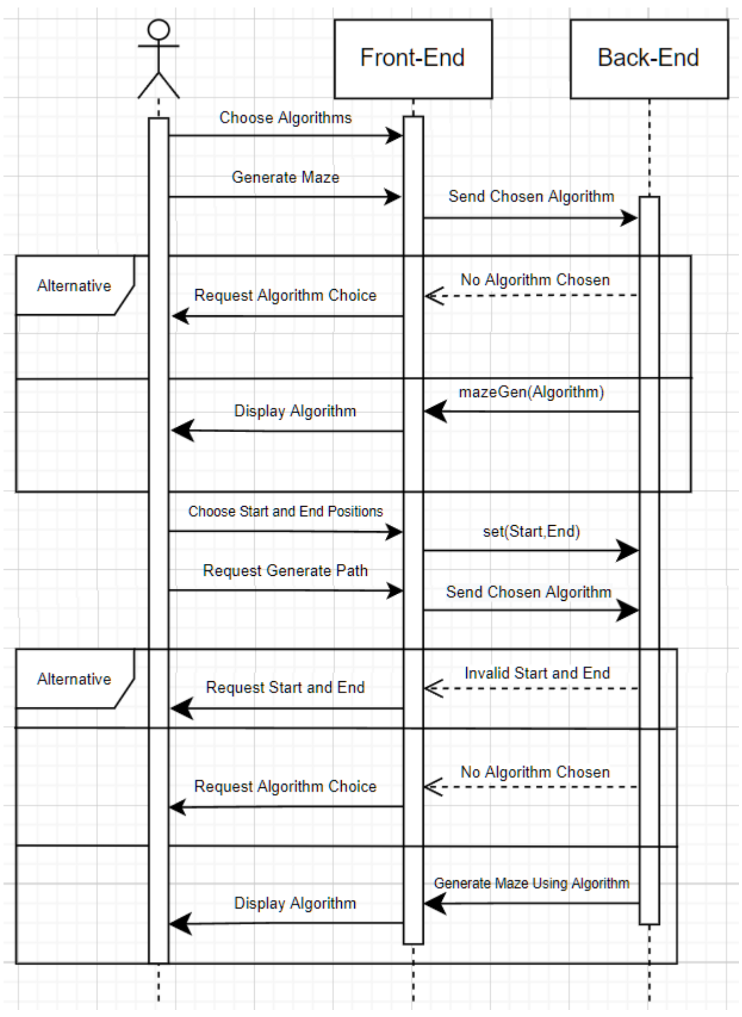


Figure 4-2 – Application Sequence Diagram

UML or The Unified Modelling Language is a universal modelling language mostly used in the field of software engineering.

UML Sequence Diagrams detail how operations are carried out when interacted by an object or in this case, the user.

This diagram is used to display the order of interactions between the user, the front end, and the back end. The user interactions trigger a sequence of operations and messages between platforms in order to carry out a function, which in this case are generating mazes and finding the shortest path from a node to another.

4.2.3 UML User Story Diagram

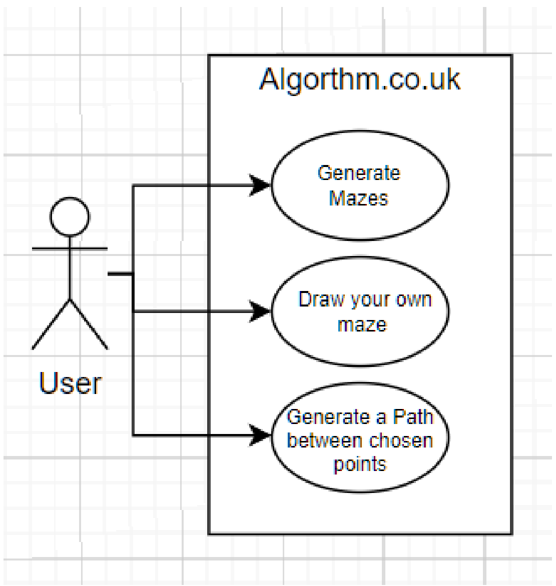


Figure 4-3 – Application User Stories Diagram

User Stories are short and simple descriptions of functions/processes that a certain user would like to execute in the software. In most cases there are multiple users to one software, but it is different in this case.

This project's demographic is quite narrow, where all the users have the same aim when using the application. This is subject to change with future upscaling plans and additional features.

4.2.4 UML Component Diagram

Component diagrams are used to model the interactions relating to the application's components and interfaces. It focuses on the physical aspects of object-oriented systems. Components could be anything from classes to databases, in this case, the components are JavaScript files and user interfaces. The main four files that operate the application are:

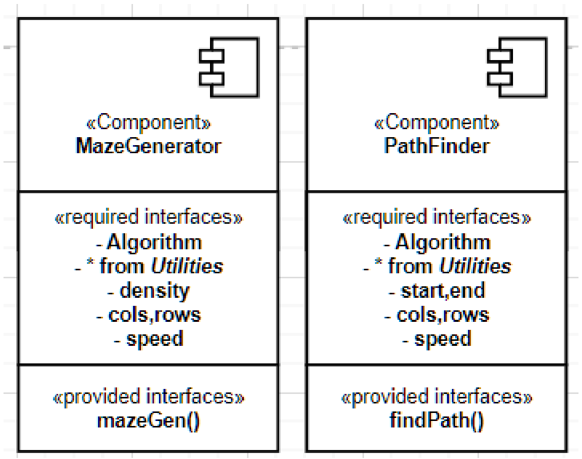


Figure 4-4 – Application SIMPLIFIED Component Diagram

PathFinder.js – which handles all the pathfinding algorithms and functionalities
MazeGenerator.js – which handles all the maze generating algorithms and functionalities
Utilities.js – which contains tools and utilities that assist the two files above in executing their algorithms, this file exports functions to the algorithm files to prevent repetition in code.
Main.js – This is a file that is not displayed in the diagram as it the controller of which operations to execute from the algorithm files and contains the side functions i.e., drawing custom mazes and changing grid size.

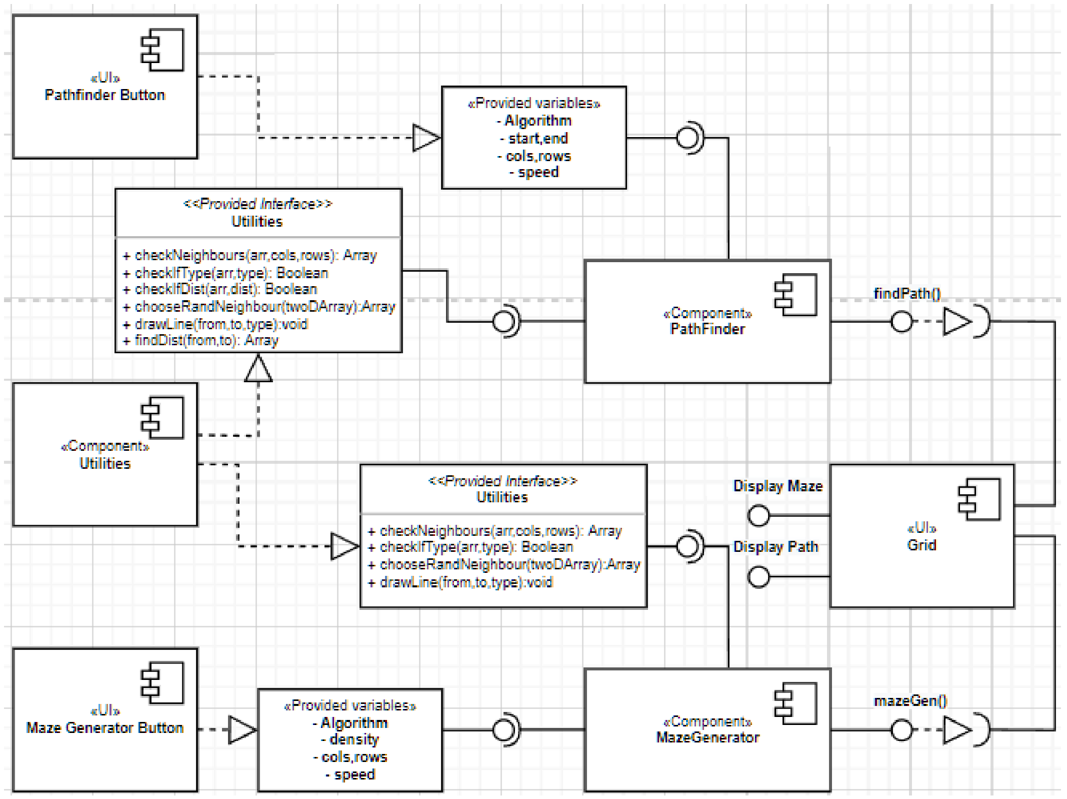


Figure 4-5 – Application Component Diagram

4.3 Graphical User Interface

After thorough research into useability engineering principles, I have gathered enough knowledge to begin designing the user interface of this application. I will apply accessibility options to increase the amount of people that can comfortably use this application. For designing the interface, I used Figma, a web-based user interface design and editing application.

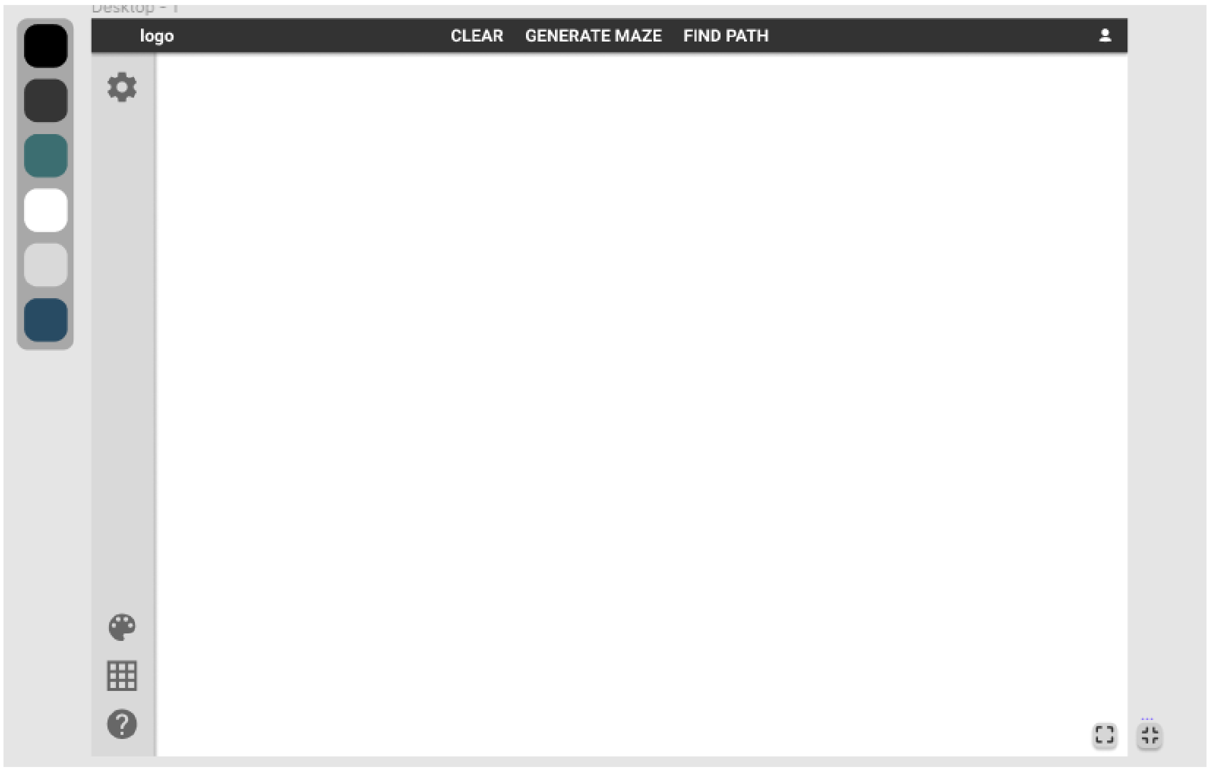


Figure 4-6 – GUI Preliminary Design

The design above focuses on preserving attention from everything outside of the grid area by choosing non-distracting colours. The grid will reside in the white empty area. The buttons on the left-side bar will provide the user with all the main and side functionalities.

⚙ **Cogwheel –** Contains the main setting functions such as picking algorithms, setting start and end node positions, and clearing the grid. It is placed in the top part of the side bar to ensure that the user opens this tab first.

🎨 **Palette –** Contains different colour palettes to provide the user with options that are more comfortable to use and observe.

▦ **Grid –** Contains the function that allows the user to change the grid size.

? **Help –** Contains a quick tutorial of the application's functionalities.

4.3.1 Colour Palettes

To accommodate for people with colour blindness, I have designed three colour palettes using David Nichols' tool (Nichols), the tool allowed me to visualize how different colour blindness see colours. In Figure 4-7, every node has its own allocated colour. The first palette is designed for all users whilst the second is for users with protanopia and deuteranopia, finally, the last palette is for users with tritanopia (Nichols).

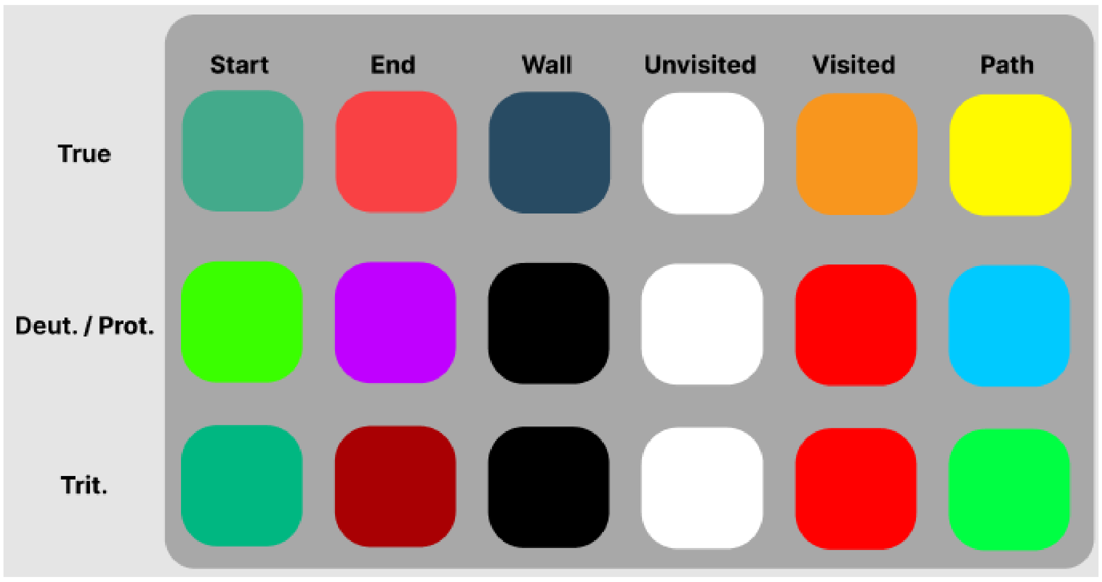


Figure 4-7 – Accessibility Colour Palettes

Chapter 5 Implementation

5.1 Introduction

This chapter aims to present the approach taken to developing every part of the application. The full source code of the project will be available in an open repository on GitHub (2022).

5.2 Overview

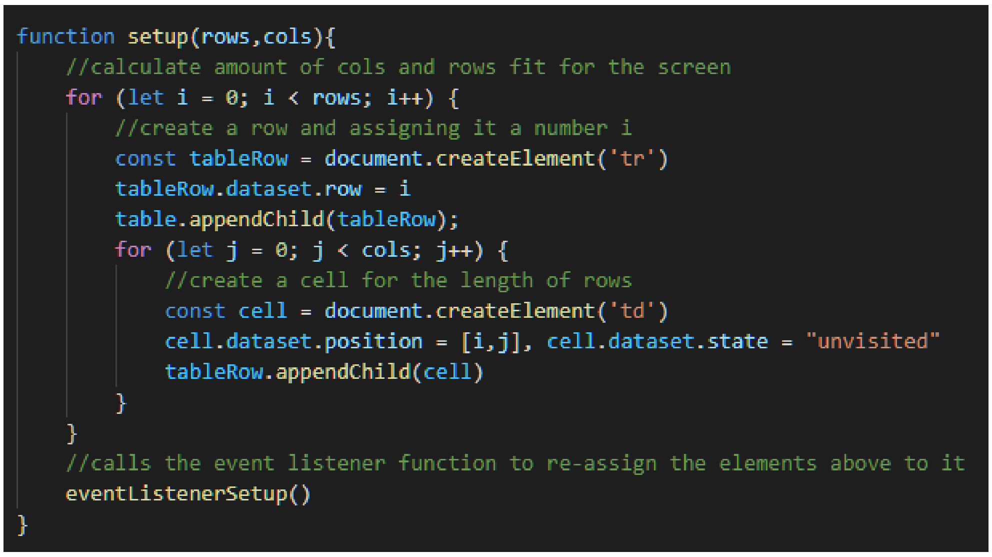
The software was developed using JavaScript (+jQuery), HTML and CSS on the Visual Studio Code IDE. I chose to use VS Code as my IDE since I am very familiar with its interface and extra features/extensions. Using the "Live Server" extension I was able to host the application locally and display any changes made automatically and fast. I used Firefox Developer Edition to view my locally hosted application as it offers many helpful utilities for web development. The application consists of four JavaScript files which were established in the design section, and two other files that contain front-end material, the html file where all the elements reside, and a styles file that gives the frontend the pleasing appearance that was designed earlier.

5.3 Grid

In order to implement any functionality, I had to have a canvas/grid implemented where every node has customizable states, styles, a defined position and accessible to the back end.
There were two main methods of implementing a grid in html:
HTML <canvas> – This is an element that is commonly used to draw graphics in a document using JavaScript. Anything can be drawn on it such as boxes, circles, text, or lines.
HTML <table> – This is an element that allows developers to arrange data into rows and columns.
When generally comparing <canvas> to <table>, it is without doubt that canvas' capabilities exceed those of the table. Many things can be done with canvas going as far as creating an HTML game contained within the canvas. When considering the aim of the grid and the project as a whole, the canvas could fill in all the requirements, but it comes with added complexity to the software as everything will be stored in the back end, while the table method would provide all the capabilities we need without any increased complexity. The table method is much more suitable for this task.
HTML Table has many tags available to use within the element but in this project, I will only focus on <tr> and <td>.
<tr> – Defines a table row.
<td> – Defines a cell in the table, it stands for table data.

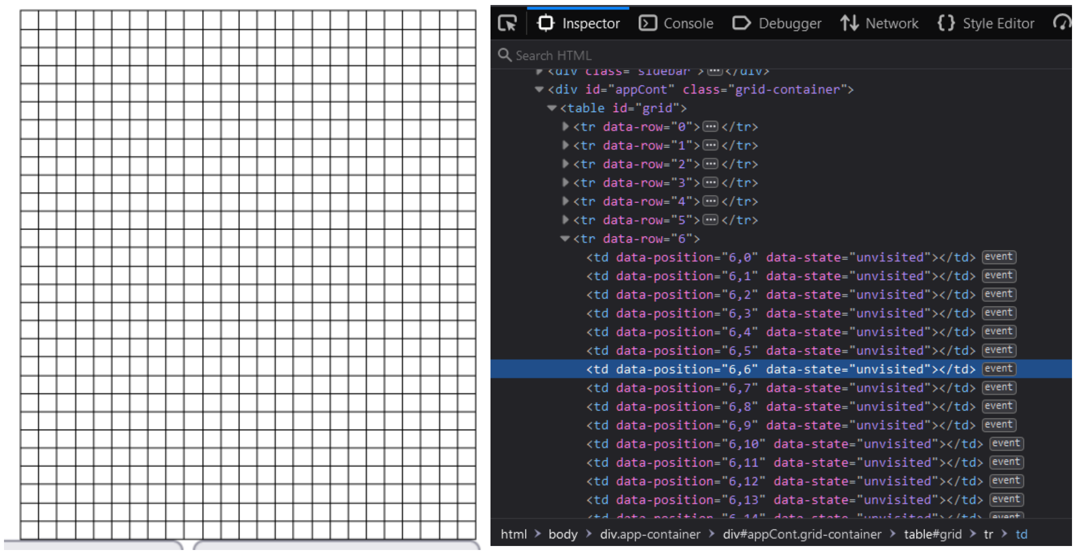
To set a state and a position to every cell/node, I utilised html element attributes. For the position of the cell, I set each <td> element two dataset attributes, position, and state. Position was a 2-integer long array containing the coordinates of that node, and state was a string value containing the state/type of node it is.

My approach was to create a table with defined table rows and within each row are a set of defined table data elements. Based on the row's given number (dataset-row) the first coordinate will be assigned, and each <td> in the said row will be given the second coordinate. To implement this, I had to create a nested for loop as shown in Figure 5-1.



```javascript
function setup(rows,cols){
    //calculate amount of cols and rows fit for the screen
    for (let i = 0; i < rows; i++) {
        //create a row and assigning it a number i
        const tableRow = document.createElement('tr')
        tableRow.dataset.row = i
        table.appendChild(tableRow);
        for (let j = 0; j < cols; j++) {
            //create a cell for the length of rows
            const cell = document.createElement('td')
            cell.dataset.position = [i,j], cell.dataset.state = "unvisited"
            tableRow.appendChild(cell)
        }
    }
    //calls the event listener function to re-assign the elements above to it
    eventListenerSetup()
}
```

Figure 5-1 – Nested for loop creating a grid (JavaScript)



```html
<div id="appCont" class="grid-container">
  <table id="grid">
    <tr data-row="0">…</tr>
    <tr data-row="1">…</tr>
    <tr data-row="2">…</tr>
    <tr data-row="3">…</tr>
    <tr data-row="4">…</tr>
    <tr data-row="5">…</tr>
    <tr data-row="6">
      <td data-position="6,0" data-state="unvisited"></td> event
      <td data-position="6,1" data-state="unvisited"></td> event
      <td data-position="6,2" data-state="unvisited"></td> event
      <td data-position="6,3" data-state="unvisited"></td> event
      <td data-position="6,4" data-state="unvisited"></td> event
      <td data-position="6,5" data-state="unvisited"></td> event
      <td data-position="6,6" data-state="unvisited"></td> event
      <td data-position="6,7" data-state="unvisited"></td> event
      <td data-position="6,8" data-state="unvisited"></td> event
      <td data-position="6,9" data-state="unvisited"></td> event
      <td data-position="6,10" data-state="unvisited"></td> event
      <td data-position="6,11" data-state="unvisited"></td> event
      <td data-position="6,12" data-state="unvisited"></td> event
      <td data-position="6,13" data-state="unvisited"></td> event
html › body › div.app-container › div#appCont.grid-container › table#grid › tr › td
```

Figure 5-2 – Result of setup() no style included (Firefox Dev tools)

The result of setup() shown in Figure 5-2 is exactly what was expected and required to complete this task. Every row has a number, and every node has a defined position and state ("unvisited") as displayed on the right side of the figure in the developer tools section. The setup() function is used again when changing the grid size.

5.4 Drawing

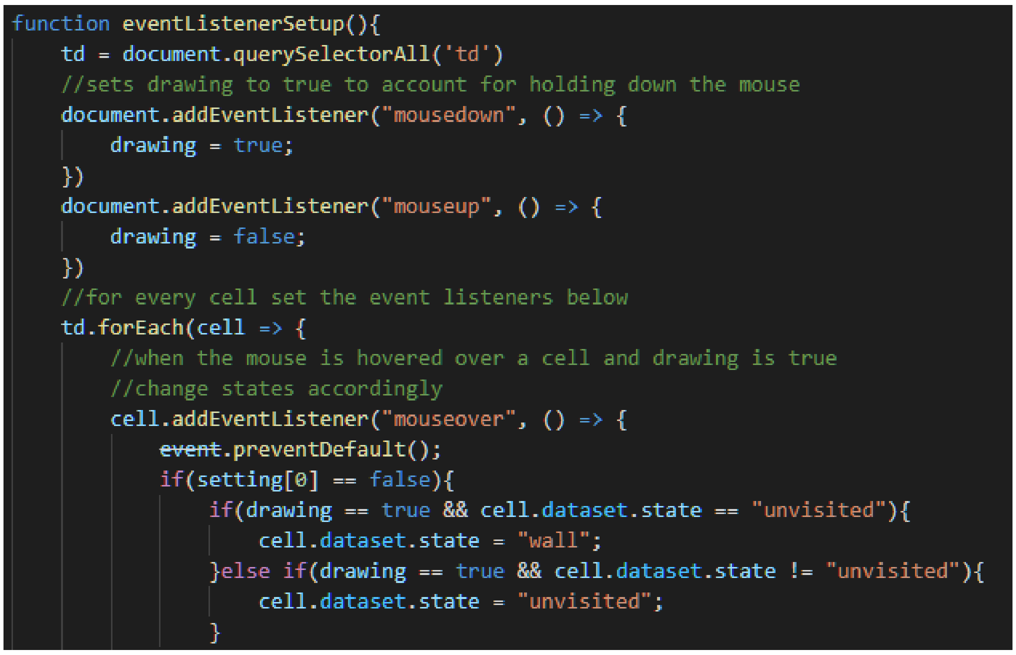
There are two ways to draw on a grid using a mouse, either by clicking on each node, or dragging while holding down the mouse button, both methods will be implemented to the drawing feature.

Using JavaScript's Event Listeners, these are methods that attach event handlers to defined elements. Event handlers will trigger based on the action it is listening for, in this case these actions will be:

mousedown – Triggers when the mouse button is clicked

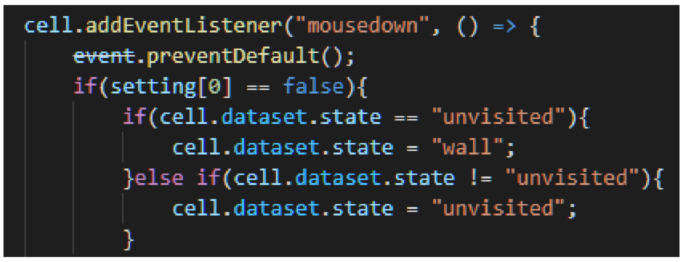
mouseup – Triggers when the mouse button is unclicked/lifted

mouseover – Triggers when the mouse hovers over the element



```javascript
function eventListenerSetup(){
    td = document.querySelectorAll('td')
    //sets drawing to true to account for holding down the mouse
    document.addEventListener("mousedown", () => {
        drawing = true;
    })
    document.addEventListener("mouseup", () => {
        drawing = false;
    })
    //for every cell set the event listeners below
    td.forEach(cell => {
        //when the mouse is hovered over a cell and drawing is true
        //change states accordingly
        cell.addEventListener("mouseover", () => {
            event.preventDefault();
            if(setting[0] == false){
                if(drawing == true && cell.dataset.state == "unvisited"){
                    cell.dataset.state = "wall";
                }else if(drawing == true && cell.dataset.state != "unvisited"){
                    cell.dataset.state = "unvisited";
                }
```

Figure 5-3 – Drawing using dragging (JavaScript)



```javascript
cell.addEventListener("mousedown", () => {
    event.preventDefault();
    if(setting[0] == false){
        if(cell.dataset.state == "unvisited"){
            cell.dataset.state = "wall";
        }else if(cell.dataset.state != "unvisited"){
            cell.dataset.state = "unvisited";
        }
```

Figure 5-4 – Drawing using clicking (JavaScript)

event.preventDefault() was used to prevent the default behaviour of browsers, which when dragging elements commonly results in the element being held and moved around by the mouse in case it contains a link.

5.5 Utility Functions

To prevent repetition in code, I have created a small library of utility functions that will be imported into other files to assist functions in executing processes without the need for extra code. Some of these key utility functions are:

checkNeighbours(arr, rows, cols, interval) – This function returns the neighbours of a defined cell position. It takes in 4 values, arr – which is an array containing the position of the targeted cell, number of rows, number of columns, and an integer interval of how far away the neighbours are.

checkIfType(arr,type) – This function checks if the state of the cell with position arr is equals to type.

checkIfDist(arr,dist) – Similar to checkIfType(), but with distance.

chooseRandomNeighbour(arr) – Chooses a random neighbour out of an array of neighbours arr defined using checkNeighbours()

drawPath(currentCell, nextCell,type) – Draws a path of type from currentCell to nextCell.

uniqueXDimensionalArr(arr) – Returns a unique array from the given arr, used to remove repeats.

findDistance(from,to) – Finds the distance between two points.

dijkstraCalcDist(startCell, rows,cols) – Allocates cell distances using Dijkstra's algorithm, used in A* Search.

Some other utility functions are placed within the files and exported to the front end as they are used by the algorithms and are also considered a side function used by the user from the front end. These include:

clearAll() – Clears all node types. This is used before running any maze generating algorithms to start with a clean grid.

clearPath() – Clears all path/visited nodes. This is used before running any pathfinding algorithms to start with a clean maze.

fillWalls() – Fills the grid with walls, this is used at the start of some maze generating algorithms such as Prim's

clearWalls() – Clears all wall nodes.

5.6 Maze Generation

Using the pseudocode created with the background research in the earlier chapter, I was able to implement 3 different maze generation and 1 variation of the recursive backtracking algorithm. The code of all the functions used to generate mazes will not be displayed but can still be found in the linked GitHub repository (2022).

5.6.1 Recursive Backtracking

This algorithm was altered to create a variation of it that instead of starting off with every cell set to state wall, it starts with a all edges as walls and draws the maze from inside with the same algorithm. Some functions from Util.js were used to aid the algorithm in executing it's functions without repetition in code such as **clearAll()**, **checkNeighbours**(), **checkIfType**(), and **chooseRandomNeighbour**().

The result of implementing and running the algorithms are displayed below:

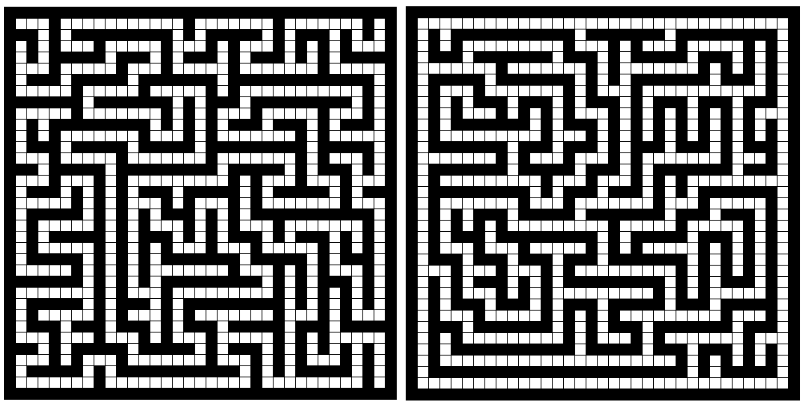


Figure 5-5 – Recursive Backtracking Implementation (left) and variant (right)

5.6.2 Randomized Prim's

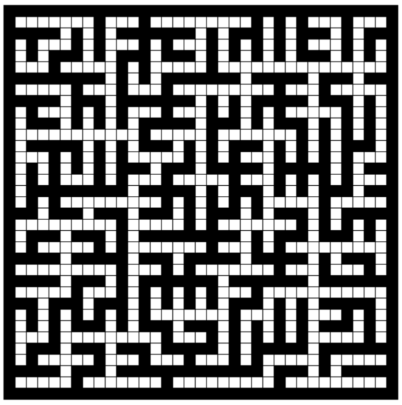


Figure 5-6 – Randomized Prim's Implementation

The implementation of this algorithm went smoothly due to the thorough research made in the earlier stages of this project. The pseudocode highlighted all the key processes that occurred in this algorithm, and made the process of integrating it into the program less difficult. The same utility functions were used with an addition of one, **uniqueXDimensionalArr**().

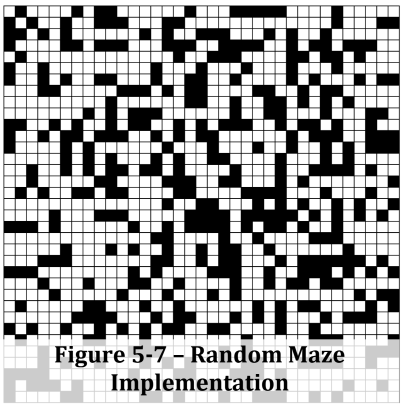


Figure 5-7 – Random Maze Implementation

5.6.3 Random

This was the simplest to implement. The result is displayed in the figure to the left. No utility functions were used in this algorithm.

5.7 Path Finding

With prior research, I was able to implement all the pathfinding algorithms discussed in the earlier chapter. In this part, I will display how all the algorithms behaved in a constant environment (partially braided maze).

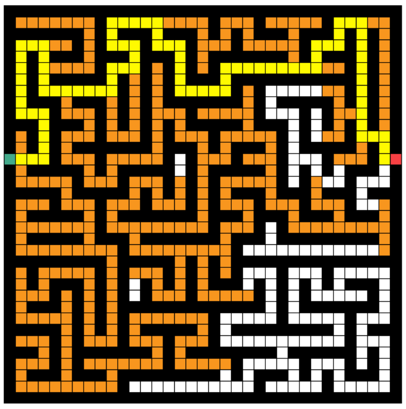


Figure 5-8 – Dijkstra's Demo

5.7.1 Dijkstra's

I have chosen to implement this first as it will help me implement A* Search after with ease, since they both follow the same principles. As shown in the screenshot on the left, Dijkstra's Algorithm found the shortest path to the end. This algorithm utilised some of the util.js functions such as: **clearPath**(), **checkIfType**(), **checkNeighbours**(),**chooseRandomNeighbour**(), and finally **uniqueXDimensionalArr**().

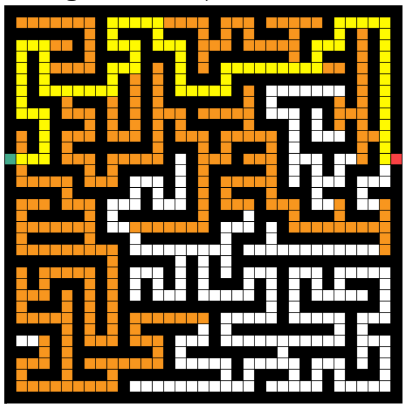


Figure 5-9 – A* Search Demo

5.7.2 A* Search

This was the most difficult algorithm to implement. With the use of it's heuristic, A* Search was able to find the shortest path whilst also visiting the least amount of cells. This algorithm used the same utility functions as Dijkstra's, but it also used **findDistance**(), **checkIfDist**(), and **dijkstraCalcDist**().

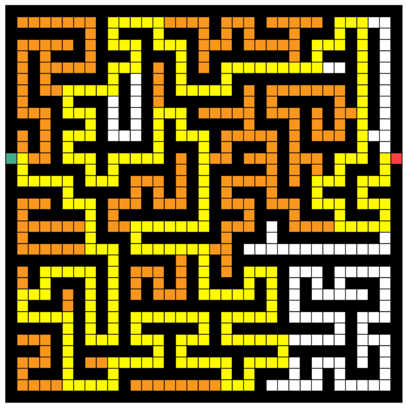


Figure 5-10 – DFS Demo

5.7.3 Depth First Search

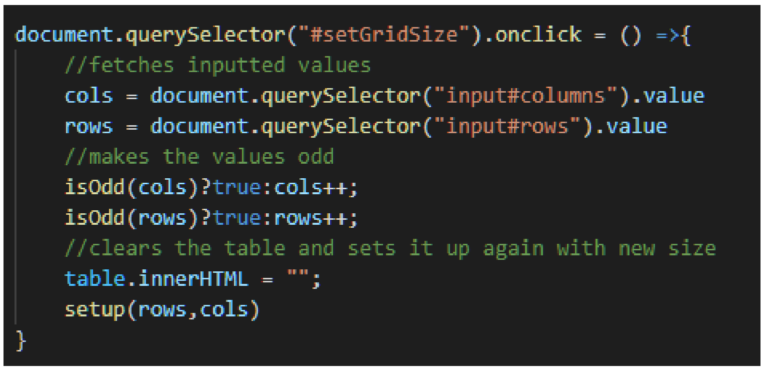
This was the least difficult pathfinding algorithm to implement due to its similarities with the recursive backtracking maze generating algorithm. This algorithm is best used in perfect mazes, as displayed in the demo to the left, it is unlikely that the algorithm finds the shortest path when there are multiple solutions.

5.8 Implementing Other Functions

This section will display all the additional functions that were implemented to increase accessibility and usability.

5.8.1 Changing Grid Size

This feature allows users to change the size of the grid by changing the number of columns and rows that grid contains. This is helpful because different screens have different sizes, and on some screens the cells within the grid may appear too wide or too narrow. It also has a use case of being convenient for users that want to experiment with simpler or more complex grid sizes. This function calls back to setup(), the first function the code executes to generate the grid on document load.

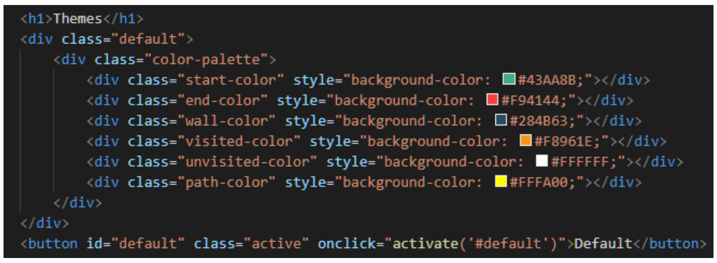


```
document.querySelector("#setGridSize").onclick = () =>{
    //fetches inputted values
    cols = document.querySelector("input#columns").value
    rows = document.querySelector("input#rows").value
    //makes the values odd
    isOdd(cols)?true:cols++;
    isOdd(rows)?true:rows++;
    //clears the table and sets it up again with new size
    table.innerHTML = "";
    setup(rows,cols)
}
```

Figure 5-11 – Grid Size Change Function (JavaScript)

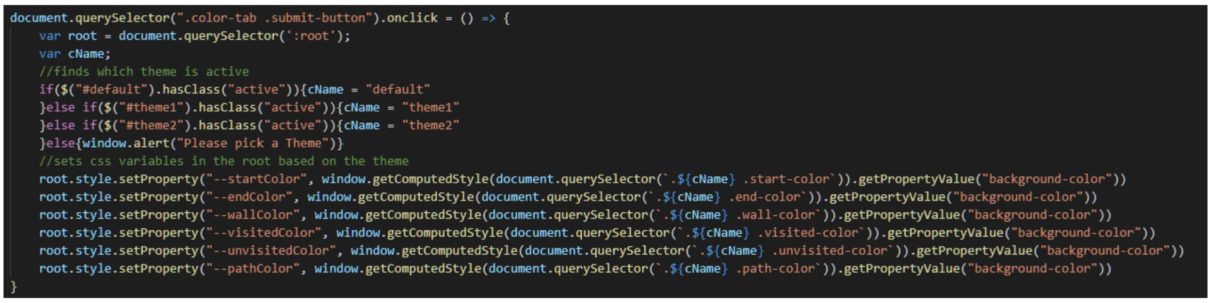
5.8.2 Changing Colour Palette

The approach I took to changing the colour palette is setting the colour palette colours in the key shown along the theme, then the JavaScript code fetches the colour based on which theme is active from the displayed key itself. This is shown below:



```
<h1>Themes</h1>
<div class="default">
    <div class="color-palette">
        <div class="start-color" style="background-color: 🟩#43AA8B;"></div>
        <div class="end-color" style="background-color: 🟥#F94144;"></div>
        <div class="wall-color" style="background-color: ■#284B63;"></div>
        <div class="visited-color" style="background-color: 🟧#F8961E;"></div>
        <div class="unvisited-color" style="background-color: ⬜#FFFFFF;"></div>
        <div class="path-color" style="background-color: 🟨#FFFA00;"></div>
    </div>
</div>
<button id="default" class="active" onclick="activate('#default')">Default</button>
```

Figure 5-12 – Colour Palette key (HTML)

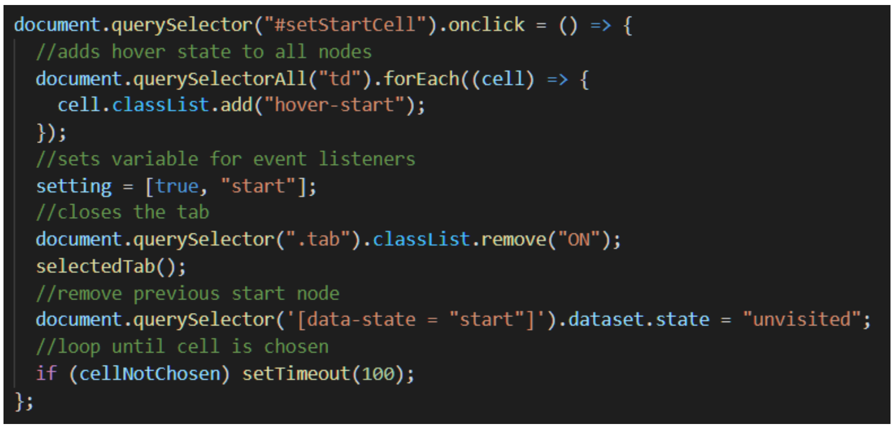


```
document.querySelector(".color-tab .submit-button").onclick = () => {
    var root = document.querySelector(':root');
    var cName;
    //finds which theme is active
    if($("#default").hasClass("active")){cName = "default"
    }else if($("#theme1").hasClass("active")){cName = "theme1"
    }else if($("#theme2").hasClass("active")){cName = "theme2"
    }else{window.alert("Please pick a Theme")}
    //sets css variables in the root based on the theme
    root.style.setProperty("--startColor", window.getComputedStyle(document.querySelector(`.${cName} .start-color`)).getPropertyValue("background-color"))
    root.style.setProperty("--endColor", window.getComputedStyle(document.querySelector(`.${cName} .end-color`)).getPropertyValue("background-color"))
    root.style.setProperty("--wallColor", window.getComputedStyle(document.querySelector(`.${cName} .wall-color`)).getPropertyValue("background-color"))
    root.style.setProperty("--visitedColor", window.getComputedStyle(document.querySelector(`.${cName} .visited-color`)).getPropertyValue("background-color"))
    root.style.setProperty("--unvisitedColor", window.getComputedStyle(document.querySelector(`.${cName} .unvisited-color`)).getPropertyValue("background-color"))
    root.style.setProperty("--pathColor", window.getComputedStyle(document.querySelector(`.${cName} .path-color`)).getPropertyValue("background-color"))
}
```

Figure 5-13 – Colour Palette Changing function (jQuery)

5.8.3 Setting Start and End Positions

This feature is essential as it allows the user to test out the pathfinding algorithms with a variety of start and end positions. I used even listeners to search for where the mouse is located, wherever it clicks it places the node type chosen. A CSS hover cue is applied to all the nodes, when hovering over a node the according colour is projected on it with a low opacity.

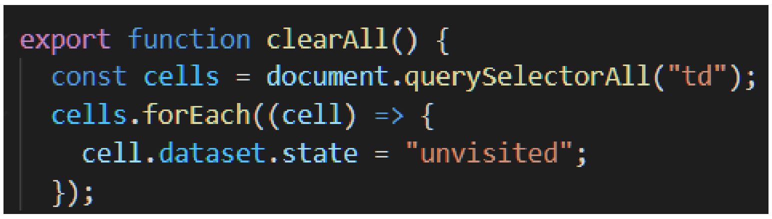


```
document.querySelector("#setStartCell").onclick = () => {
  //adds hover state to all nodes
  document.querySelectorAll("td").forEach((cell) => {
    cell.classList.add("hover-start");
  });
  //sets variable for event listeners
  setting = [true, "start"];
  //closes the tab
  document.querySelector(".tab").classList.remove("ON");
  selectedTab();
  //remove previous start node
  document.querySelector('[data-state = "start"]').dataset.state = "unvisited";
  //loop until cell is chosen
  if (cellNotChosen) setTimeout(100);
};
```

Figure 5-14 – Set Start Cell function (JavaScript)

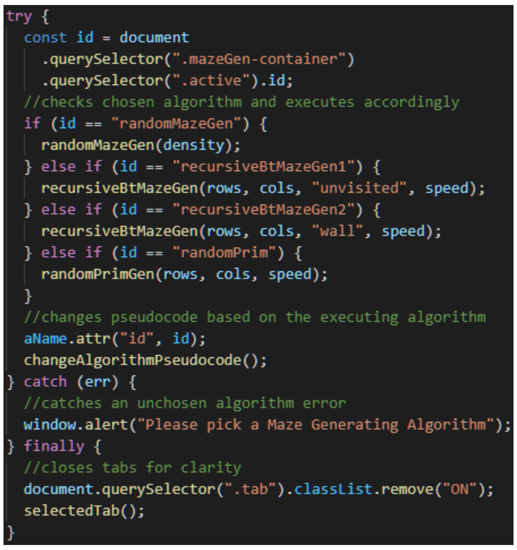
5.8.4 Clearing The Grid

There are three main ways of clearing the grid, clearing all nodes, and setting them to unvisited, clearing walls, and clearing paths. Clear all is used before running any maze generating algorithm, and clear paths is used before running any pathfinding algorithm. Clearing walls is a feature provided to the user for usability reasons.



```
export function clearAll() {
  const cells = document.querySelectorAll("td");
  cells.forEach((cell) => {
    cell.dataset.state = "unvisited";
  });
```

Figure 5-15 – Clear All function (JavaScript)



```
try {
  const id = document
    .querySelector(".mazeGen-container")
    .querySelector(".active").id;
  //checks chosen algorithm and executes accordingly
  if (id == "randomMazeGen") {
    randomMazeGen(density);
  } else if (id == "recursiveBtMazeGen1") {
    recursiveBtMazeGen(rows, cols, "unvisited", speed);
  } else if (id == "recursiveBtMazeGen2") {
    recursiveBtMazeGen(rows, cols, "wall", speed);
  } else if (id == "randomPrim") {
    randomPrimGen(rows, cols, speed);
  }
  //changes pseudocode based on the executing algorithm
  aName.attr("id", id);
  changeAlgorithmPseudocode();
} catch (err) {
  //catches an unchosen algorithm error
  window.alert("Please pick a Maze Generating Algorithm");
} finally {
  //closes tabs for clarity
  document.querySelector(".tab").classList.remove("ON");
  selectedTab();
}
```

Figure 5-16 – Displaying Pseudocode (JavaScript)

5.8.5 Displaying The Algorithm's Pseudocode

This feature was added though it was not mentioned in the requirements, that is because I did not consider a demographic of users that have never seen these algorithms before. With the help of this feature, users should be able to understand the algorithms further as they can read the pseudocode whilst watching the algorithm execute. Based on the algorithm that is executing, this feature will present the user with the according pseudocode in a collapsible tab.

5.9 User Interface

Integrating the UI was the final stage of development to having a fully functional and usable application. I began by creating the navigation bar, and side function bar. Then allocated the grid a container for it to reside in at all times covering the entire empty space.
Below is a comparison of what the application's UI looked like before CSS and after:

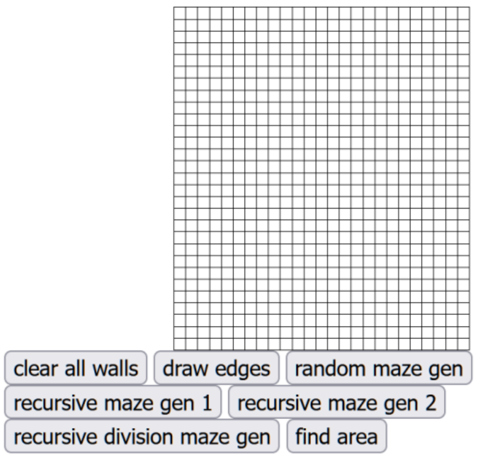


Figure 5-17 – Application's UI without CSS

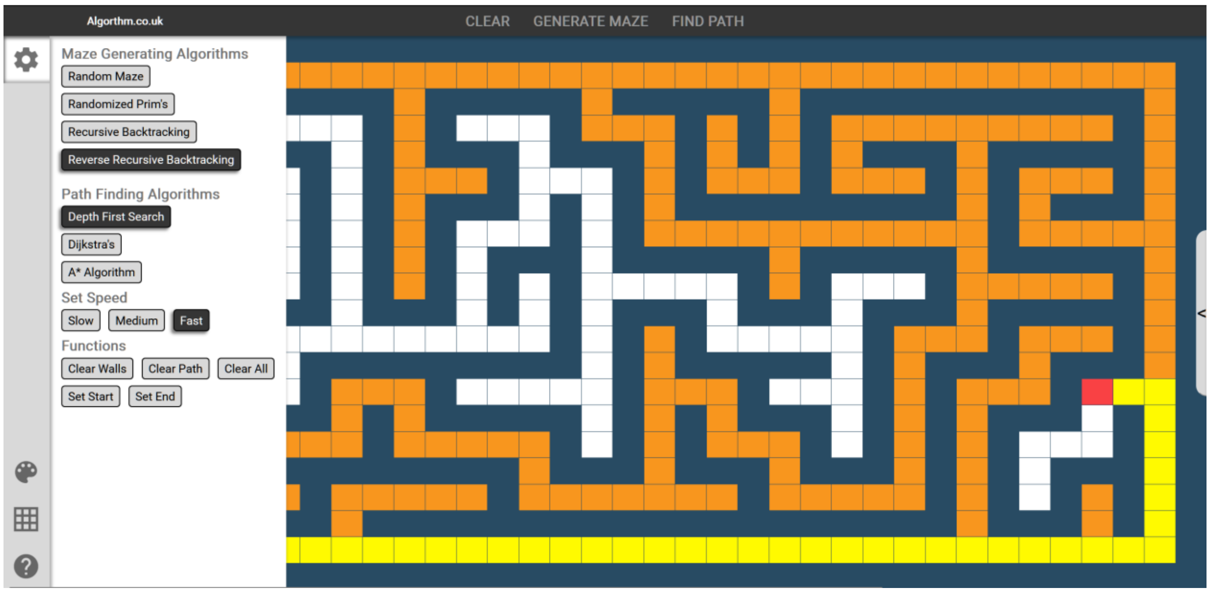
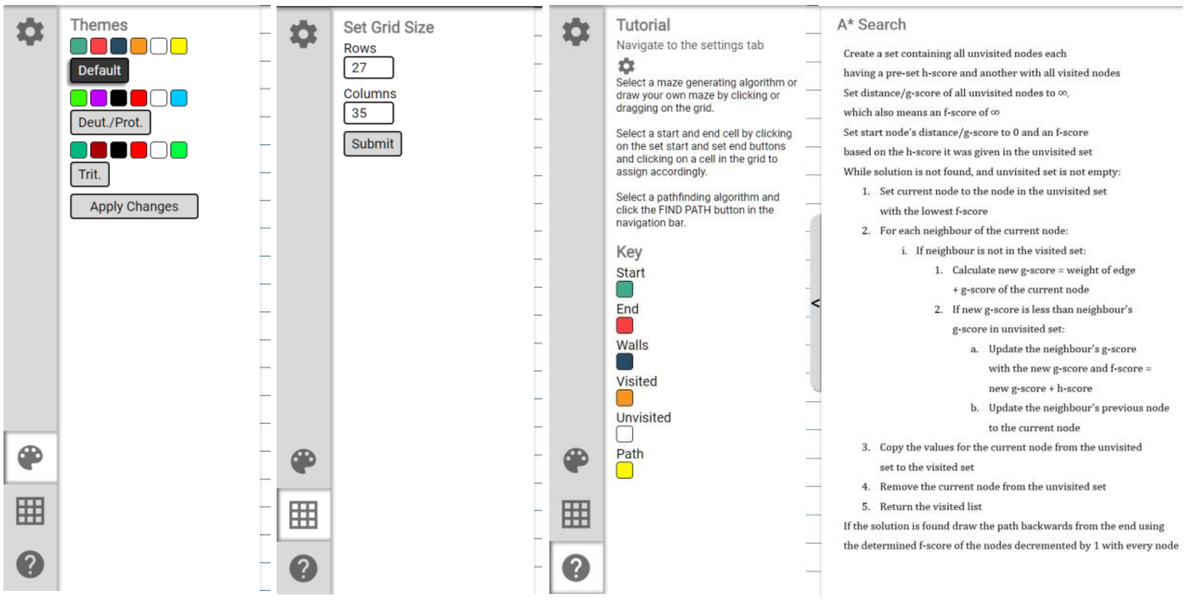


Figure 5-18 – Application's UI with CSS

I have integrated the UI without adding unnecessary stylization as the purpose of the UI is to be easy to use and not distracting. To ensure that the users know what options are selected I used box shadows and highlighting the button using a bold colour as shown in Figure 5-12 where the picked options are highlighted and stand out.
The additional tabs present all the extra features, such as displaying the pseudocode, changing the colour palette for accessibility, changing the grid size to match your screen size or preference and a tutorial tab briefly explaining the application's functionalities to the user.
All the tabs are displayed below:



Navigate to the settings tab

Select a maze generating algorithm or draw your own maze by clicking or dragging on the grid.

Select a start and end cell by clicking on the set start and set end buttons and clicking on a cell in the grid to assign accordingly.

Select a pathfinding algorithm and click the FIND PATH button in the navigation bar.

A* Search

```
Create a set containing all unvisited nodes each
having a pre-set h-score and another with all visited nodes
Set distance/g-score of all unvisited nodes to ∞,
which also means an f-score of ∞
Set start node's distance/g-score to 0 and an f-score
based on the h-score it was given in the unvisited set
While solution is not found, and unvisited set is not empty:
    1. Set current node to the node in the unvisited set
       with the lowest f-score
    2. For each neighbour of the current node:
          i. If neighbour is not in the visited set:
               1. Calculate new g-score = weight of edge
                  + g-score of the current node
               2. If new g-score is less than neighbour's
                  g-score in unvisited set:
                     a. Update the neighbour's g-score
                        with the new g-score and f-score =
                        new g-score + h-score
                     b. Update the neighbour's previous node
                        to the current node
    3. Copy the values for the current node from the unvisited
       set to the visited set
    4. Remove the current node from the unvisited set
    5. Return the visited list
If the solution is found draw the path backwards from the end using
the determined f-score of the nodes decremented by 1 with every node
```

Figure 5-19 – Other Functionality tabs

Accessibility options were implemented as discussed in the design section. Different colour palettes are presented to the user and if they would like to change the palette based on their colour blindness. It is very simple to use, as all they would have to do is select a palette and click "Apply Changes". The different palettes applied to the app are displayed below:

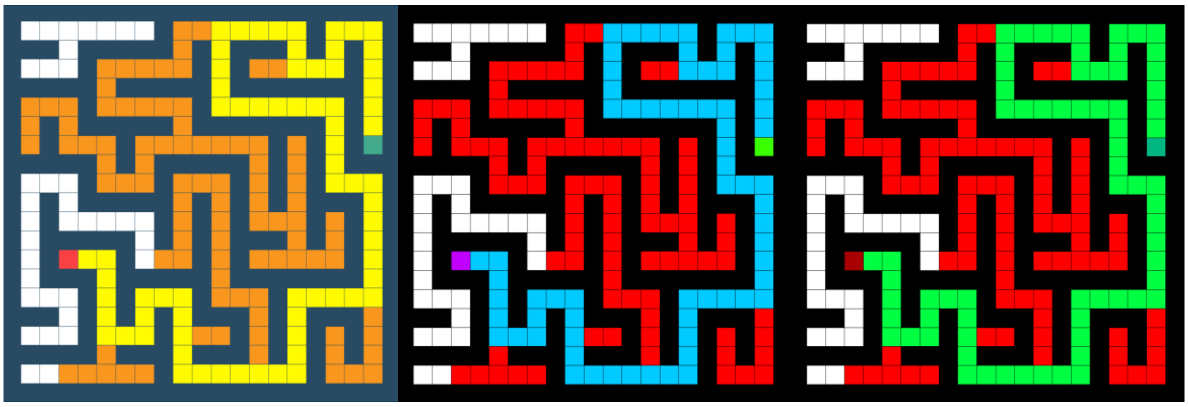
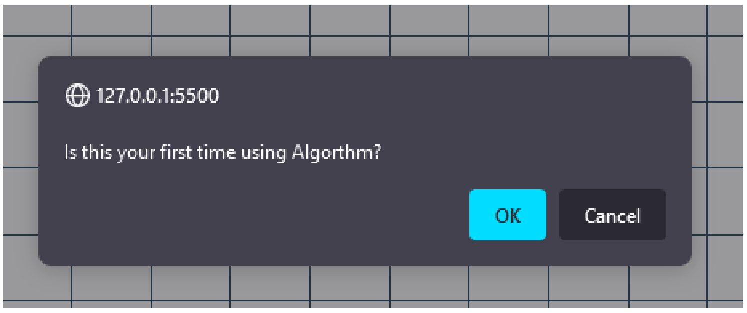


Figure 5-20 – True(Left), Deut./Prot.(Middle), Trit.(Right) Colour Palettes

Another accessibility feature was presenting new users with the tutorial tab from the start of the application. This was done by giving a prompt to users asking them if it is their first time using the application, if so then the tutorial tab is displayed.



127.0.0.1:5500

Is this your first time using Algorthm?

Figure 5-21 – First Time User Prompt

Chapter 6 Testing and Evaluation

6.1 Introduction

This chapter outlines the testing phase of the application's development life cycle. Different testing methods will be executed to help refine the application based on the results.

6.2 Demographic and Methodology

Once the application's development was complete, a questionnaire and a temporary link to the application was sent to students studying computer science and consented to participating in this study. The study's subjects (N=10) were all over 20 years old, consisting of a mixture of males (N= 8) and females (N=2). All the subjects claimed to have brief prior knowledge of some of the algorithms. Upon receiving the questionnaire, the subjects answered the pre-demo questions, then carried out a 5-10-minute-long test of the application. After this, the students answered the final questionnaires which asked about their understanding of the algorithms and the usability of the application.

6.3 Questionnaire Results

6.3.1 Pre-Test Answers

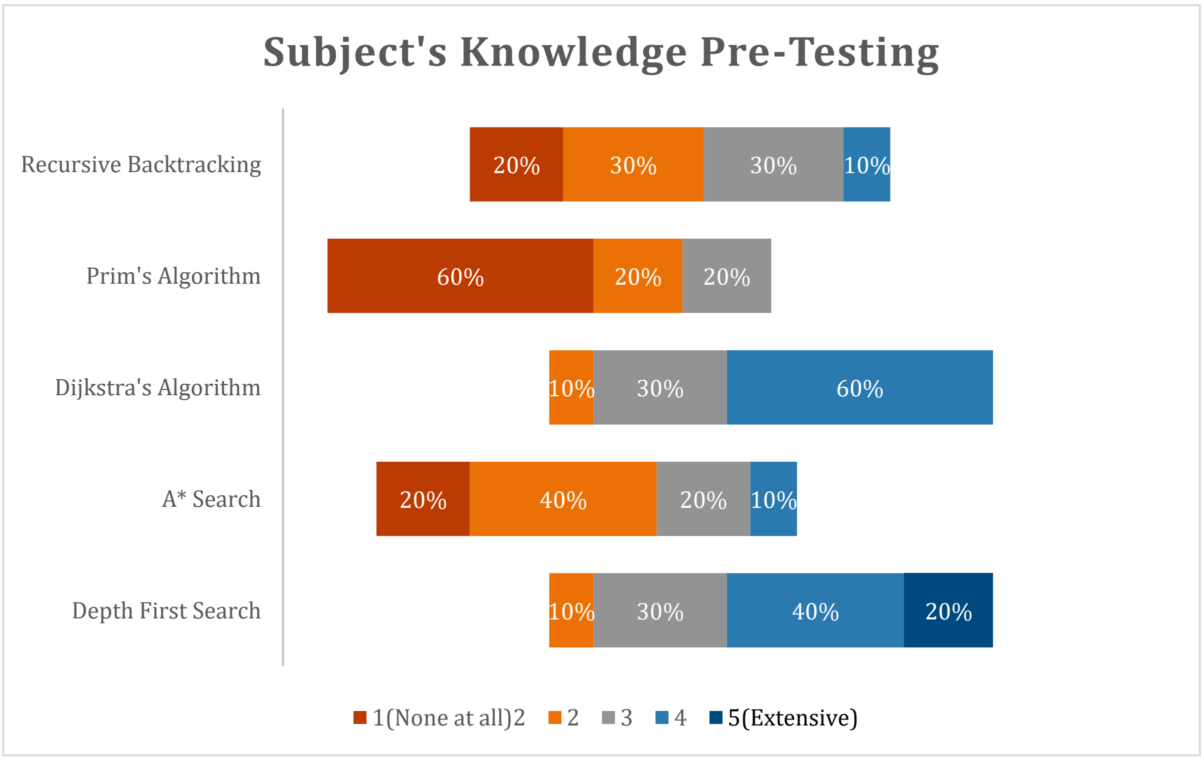


Figure 6-1 – Pre-Test Knowledge Results

6.3.2 Post-Test Answers

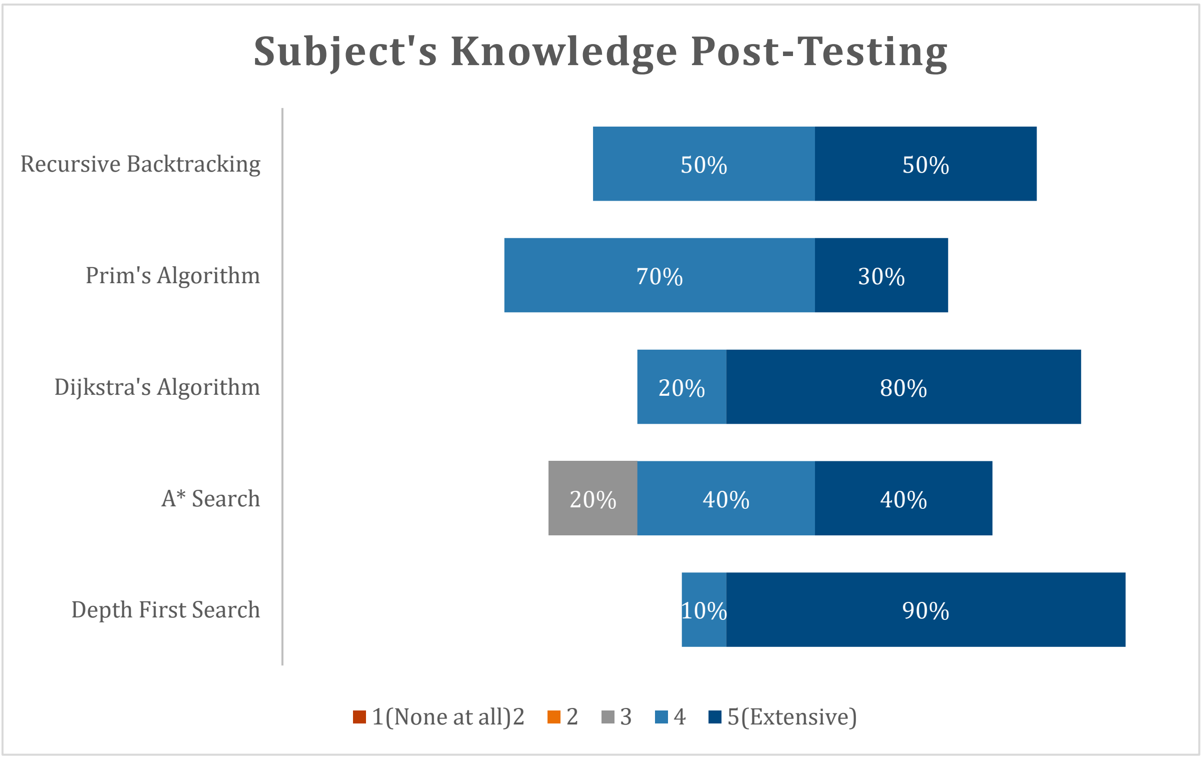


Figure 6-2 – Post-Test Knowledge Results

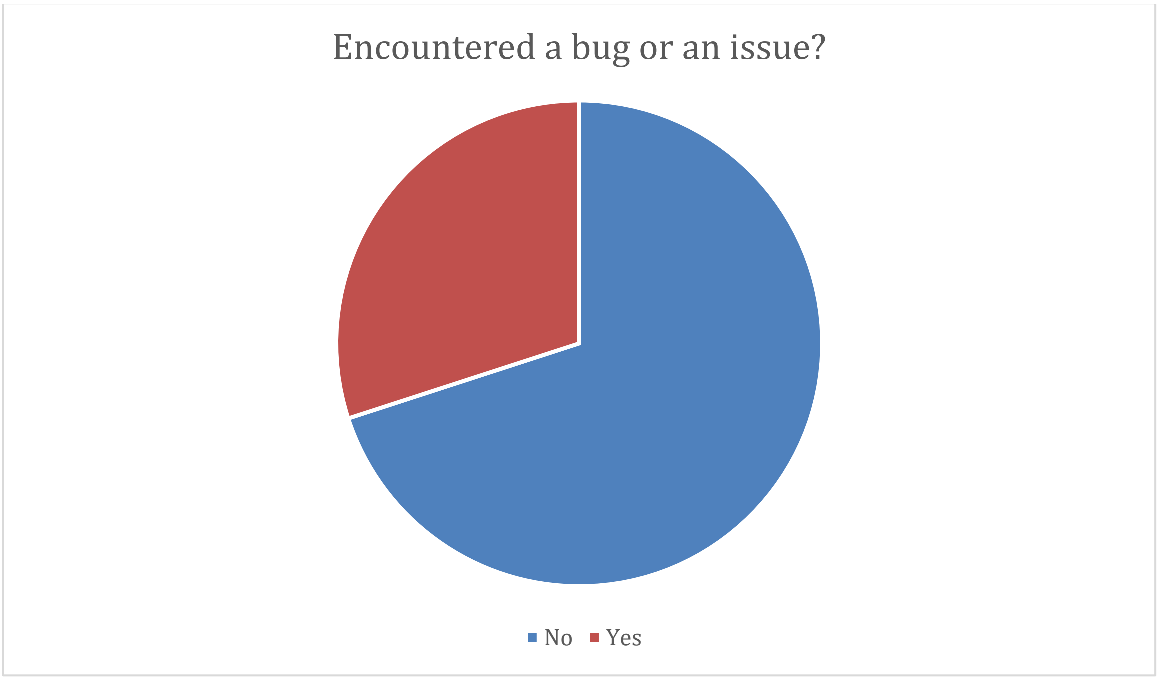


Figure 6-3 – Number of people encountered a bug or an issue

6.4 Results Discussion

Upon reviewing the results, there is a clear increase in knowledge regarding the implemented algorithms as shown in Figures 6-1 and 6-2. The subject's grasp of the maze generating algorithms was much better than their knowledge gain of pathfinding algorithms. Understandably, A* Search and Prim's Algorithm were the least known and understood algorithms both post and pre-testing. That is likely due to their complexity and lack of popularity within the field. A* Search's pseudocode is by far the most complicated out of the bunch.

When it came to usability questions, the majority (70%) did not encounter a bug/inconvenience. This result helped me understand what usability and functional features the application was lacking, and what I could use to provide a better UX. Of the three participants that encountered an inconvenience, two of the problems highlighted were due to the additional functions placement. Both subjects mentioned that the process of setting the start and end cell's positions takes an unnecessarily long amount of time. I agree and will take into consideration one of their requests (to place the start and end cell button on the top bar). The third student's inconvenience was related to setting start and end positions as well. They suggested combining both buttons together to make the interaction more fluid and lessen the time taken.

Both of those requests will be amended into the program before concluding this report. Another feature suggested was a method to view each cells properties that include, the position and the state. This could be done by hovering over a cell for a short period of time and a small prompt will pop up on top of the cell displaying the properties. One of the subjects proposed an addition of multiple end nodes into the application, where the pathfinding algorithm would have to reach one before the other. These would greatly improve the user experience.

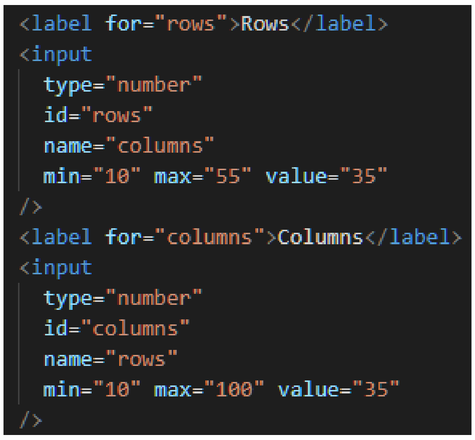
6.5 Unit Tests Table

ID	Feature Tested	Reps	Expected Result	Result
1	Generating a maze using Random Maze	10	Random maze generated	PASS
2	Generating a maze using Randomized Prim's Algorithm	10	Perfect maze generated	PASS
3	Generating a maze using Recursive Backtracking	10	Perfect maze generated	PASS
4	Generating a maze using Reverse Recursive Backtracking	10	Partially Braided maze generated	PASS

5	Finding a path between two randomly chosen points using Depth First Search	20	Path found	PASS
6	Finding a path between two randomly chosen points using Dijkstra's Algorithm	20	Path found	PASS
7	Finding a path between two randomly chosen points using A* Search	20	Path found	PASS
8	Every speed type works on all algorithms	21	All algorithms work in the 3 speed settings	PASS
9	Clear Walls button	5	Clears all walls	PASS
10	Clear Path button	5	Clears all paths	PASS
11	Clear All button	5	Clears entire grid	PASS
12	Set Start button	5	Removes previous start node and allows user to place a new one	PASS
13	Set End button	5	Removes previous end node and allows user to place a new one	PASS
14	Changing Colour Palette	6	Allows user to switch to any of the three colour palettes	PASS
15	Changing Grid Size	5	Allows user to switch to any size between 10x10 to 55x100	FAIL
16	Displaying Tutorial to new users	3	Asks if new user if so, then display tutorial tab	PASS
17	Generate Maze button without an algorithm chosen	3	Displays an error message requesting for an algorithm	PASS
18	Generate Path button without an algorithm chosen	3	Displays an error message requesting for an algorithm	PASS
19	Generate a Path button without a start and end chosen	10	Displays an error message requesting for start and end nodes	FAIL
20	Pseudocode display	14	Displays pseudocode based on the algorithm that is running	PASS

Table 3 - Unit Tests

Unit tests were carried out to discover issues in the application's features/units. As shown in Table 3, the results of the unit tests were 18/20 with the 2 failed tests being changing the grid size and generating a path button without a set start and end nodes. The first fail was due to an oversight of a way users can input the number into the grid size field. The limit on the input number is set in HTML (Front-end) as shown in the screenshot below:



```html
<label for="rows">Rows</label>
<input
  type="number"
  id="rows"
  name="columns"
  min="10" max="55" value="35"
/>
<label for="columns">Columns</label>
<input
  type="number"
  id="columns"
  name="rows"
  min="10" max="100" value="35"
/>
```

Figure 6-4 - Rows and Columns input element (HTML)

When using the arrow buttons provided with the number input, the number will not go over the limit. But when inputting the size manually, it doesn't account for the limit. This could be avoided by setting a limit in the back end.

The other failure was due to not recognising that users might forget to set a start and an end node before running a pathfinding algorithm. This can be fixed by catching an error upon executing the pathfinding algorithms.

6.6 Code Metrics

Since JavaScript is not a class-based object-oriented-programming language, it was not possible to gather the class/method code metrics e.g., Depth of Inheritance. I could not find a tool that would aid me in collecting inheritance-based code metrics for this project, but I managed to find a complexity calculator that will measure each function's cyclomatic complexity. I manually gathered some of the less complicated metrics such as LOC (lines of code), LOEC (lines of executable code), and finally LOC per main functions including the use of utility functions.

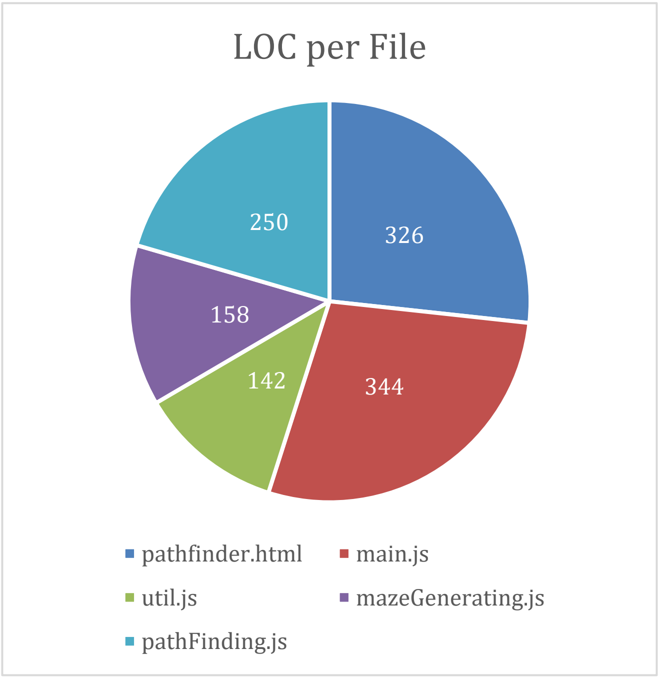


Figure 6-5 – LOC per File

Upon reviewing the code metrics collected, there appears to be a great difference in LOC and LOEC per file. This can be a result of my use of an automatic formatter that uses multiple lines for single line commands to increase readability. This could also be due to commented out work in progress or just comments in general. The size of each file is as expected, the 3 library files contain the least content, and the main functions and html files contain the majority of code.

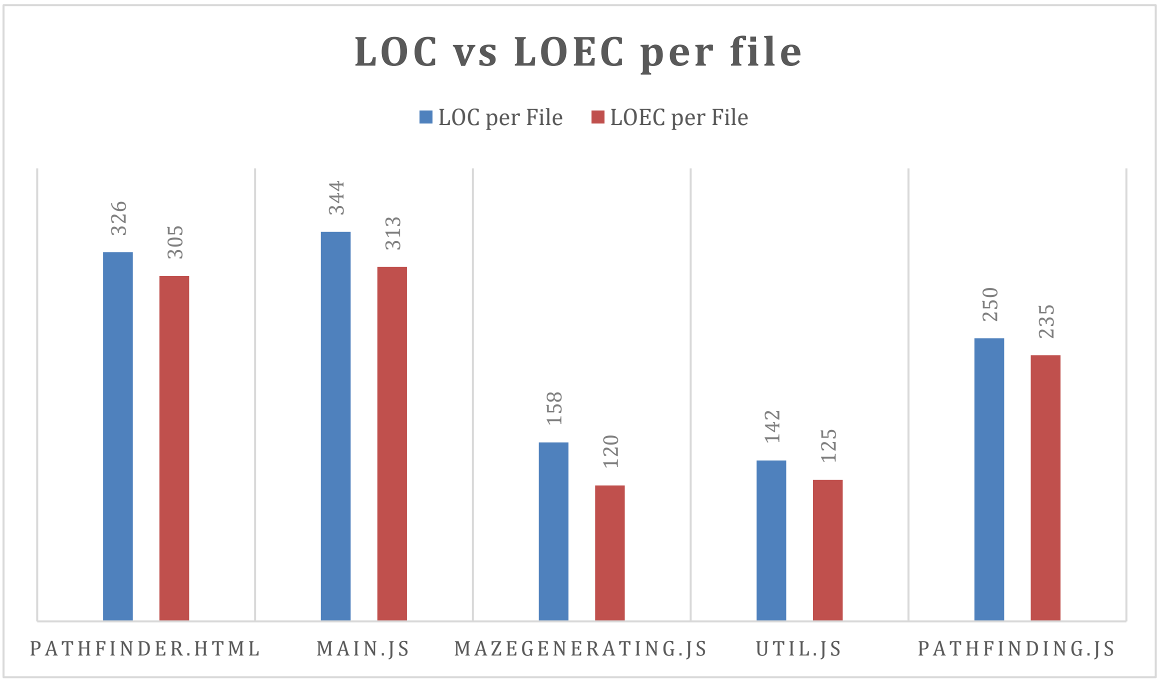


Figure 6-6 - LOC vs LOEC per file

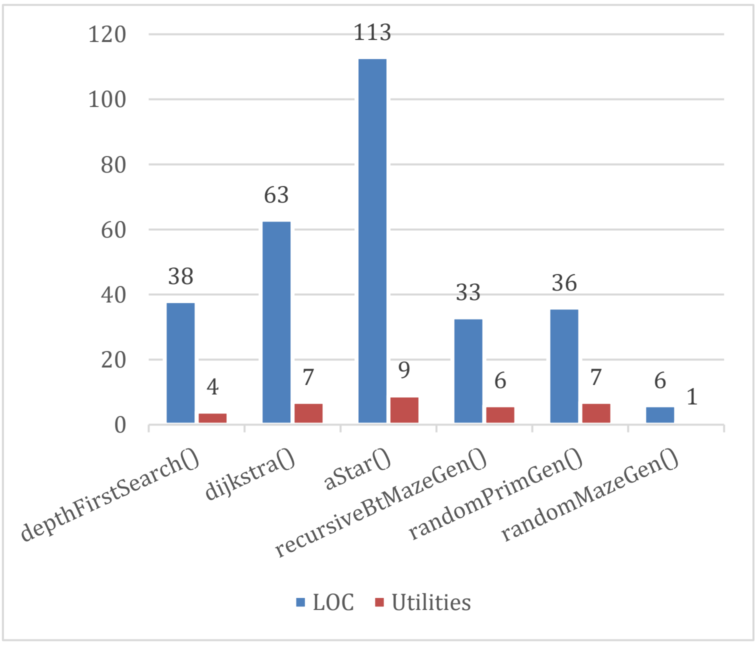


Figure 6-7 – LOC and use of Utilities per Function

A* Search is clearly the most complex algorithm out of all the algorithms implemented, and the results displayed in Figure 6-7 reflect this. The pathfinding algorithms have a greater average use of utility functions and LOC which is due to their generally larger complexity when compared maze generating algorithms. This can be proven by calculating every function's cyclomatic complexity.

Cyclomatic complexity is a quantitative based software metric used to indicate the complexity of a program. It measures the number of linearly independent branches/paths through a programs source code. Examples of paths in a code include if else statements, for loops, while loops, and nested function declarations. This is very useful when using JavaScript as it is a single-threaded language that can only execute a single path at once.

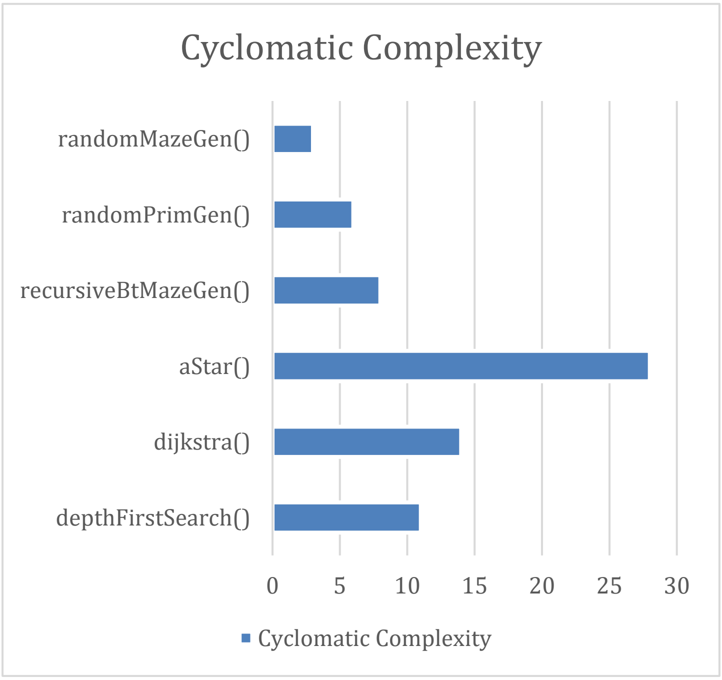


Figure 6-8 – Cyclomatic Complexity of the main functions

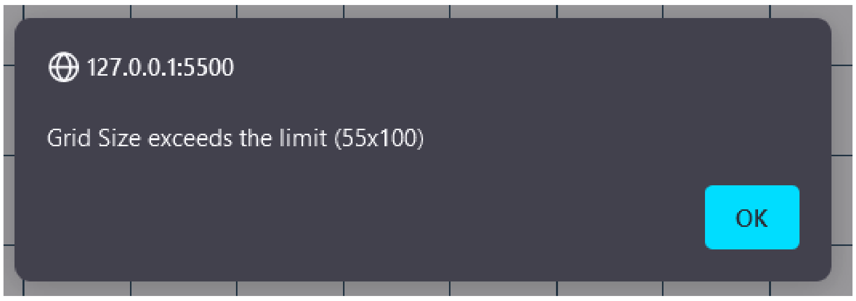
The results of the functions complexities were unexpected. A general recommendation of function complexity is to keep it less than 10. All the pathfinding algorithms have a complexity above the recommended complexity with A* Search being almost 3 times that amount. This is due to the code for those functions being too long that it is assigned to execute too many operations which leads to excessive branching.

6.7 Revisions and Fixes

After the completion of testing, there were a few fixes and features that needed to be implemented. The first two revisions are regarding the two failures that occurred in the unit tests.

Changing grid size:

The issue encountered in testing was that users could input any number into the columns or rows fields and the program would apply the change to the grid size, leading to the program breaking or the grid overflowing out of the application area. This was fixed by applying a back-end change that checks the value inputted and compares it to the limit. If the value is over the limit an error message will appear as such:

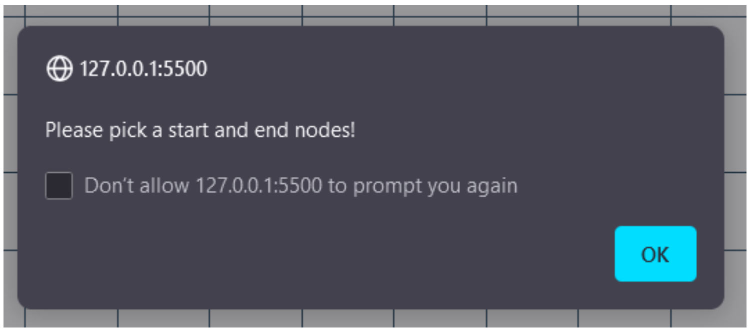

127.0.0.1:5500

Grid Size exceeds the limit (55x100)

Figure 6-9 – Grid Size Error message

Generating a path without a set start or end:

This issue involved the user requesting a path to be generated without setting a start or an end node. Some algorithms can run without an end node existing, but A* Search requires an end node. This issue was fixed by applying a check on assigning a start and end variable in the algorithms and displaying an error message accordingly:


127.0.0.1:5500

Please pick a start and end nodes!

Don't allow 127.0.0.1:5500 to prompt you again

Figure 6-10 – Pathfinding Error message

Changed position and function of set start and end button:

The buttons to set the start and end nodes with were far from reach, and the process of setting the nodes took too long. This issue was fixed by placing a button that allowed the user to set a start and set an end node directly after. This button was placed in the top navigation bar in the hopes of increasing usability and convenience for the users.

Chapter 7 Conclusions

7.1 Introduction

This report concludes in this chapter. Previously set objectives (in Aims and Objectives) and the project aim are addressed and whether they were achieved or not.

7.2 Objectives & Aims

I believe that the original aim of educating users through a visual and interactive tool was achieved. The results from the questionnaires pre and post testing clearly indicated an increase in knowledge from using the application regarding the implemented algorithms.

Table 4 – Objectives Achieved

Objectives	Achieved?	Ref
Identify and evaluate applicable software engineering methodologies and apply its structure on this project's development journey.	Yes	3.2
Thoroughly research the chosen algorithms for maze generating and path finding as well as the technology to use on implementing the application.	Yes	Chapter 2 2.8

Design a user interface that allows the users to switch between algorithms, set a start and an end as well as being able to change the size of the grid/maze.	Yes	4.3
Apply useability engineering principles throughout the entire process of designing the UI.	Yes	5.9
Create a function that allows users to draw their own mazes/environments using their mouse.	Yes	5.4
Implement two ways to generate random mazes to provide the pathfinding algorithms different environments to work with.	Yes	5.5
Implement two algorithms to find the shortest path between two cells.	Yes	5.7
Test and evaluate the user experience through questionnaires before and after using the application.	Yes	Chapter 6
Write a detailed report on the project going through all development stages and discoveries upon completing the report.	Yes	n/a
Finally implement any important changes suggested through testing and discuss plans in concerns of this applications future.	Yes	6.7

7.3 Limitations

The only limitation encountered throughout the development of this project was the limited number and demographic of participants that I worked with in gathering the data necessary to improve the application. The participants were all from the same 2-year age range of 21-23, and only 2 of the participants were female. It would have been significantly more beneficial to have an increased sample size and a wider variety of participants. A larger samples size, wider age range, and an equal gender split would yield more diverse feedback and help improve the application overall. The inclusion of colour-blind participants would have been useful to test whether the accessibility options were suitable.

7.4 Future Work

The development of this project will proceed after the completion of this report. I believe this application has great potential in shaping the future of algorithm education and can only improve the experience of those new to such an extensive and exciting part of computer science.

There are many features I would like to implement, such as the ability to move forwards or backwards within a frame, whilst allowing the user to have full control of what part of the algorithm they are watching. This could be paired with a feature that highlights the pseudocode line that is currently executing through the pseudocode tab featured in the application. The greatest feature/expansion I would like to implement in the future would be a sorting algorithms visualizer. It would be easy to implement and could serve as a great addition to visualising algorithms.

It would be beneficial to gain additional feedback, as explained in the limitations of the project. Further feedback would improve the project greatly, especially if a larger and more diverse participant group was available.

Finally, there is a plan for deploying this application for the public's use. I will host the project under the purchased domain name of Algorthm.co.uk, this is only after all the minor bugs have been addressed and the small features discussed above have been implemented.

Personal Reflection

A.1 Reflection on Project

I believe that this project's outcome was more than satisfactory. I originally set out to implement two algorithms for both categories. In the end, I implemented all of the researched algorithms. Every feature I set out to create was fully functional and completed to a (personally) acceptable standard.

If I had worked on this project again with hindsight, I would have wanted to have implemented the feature of providing the user with full control of what line of pseudocode is executing, with an ability to rewind frames, slow it down or fast forward. I believe it would have been possible to implement more algorithms or even a sorting algorithms visualizer with the hindsight knowledge gained from developing this project and the same given time.

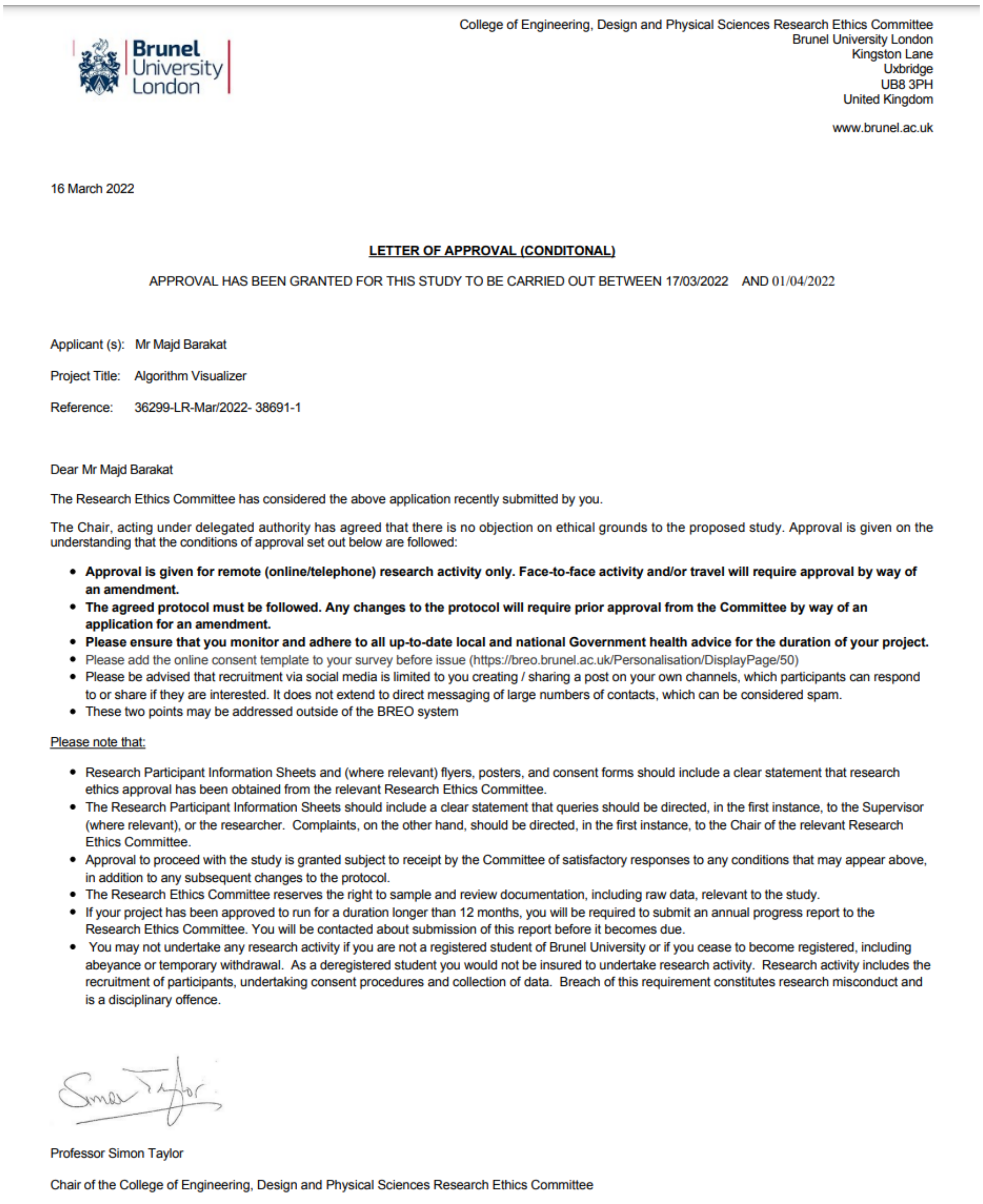
A.2 Personal Reflection

I began developing this application at the start of December 2021 and completed the foundation for the core features by the end of the month. I took a break for the entire month of January due to unpredictable circumstances that led to me having lack of a stable internet connection. Then carried out the rest of the development from February to April with steady progression.

This projects greatest downfall was that the implementation phase took much longer than anticipated, it extended from December to March. This forced me to work on the later parts of the report much later in the year, giving me around 20 days to write it. I believe the quality of the project's design, implementation, and testing documentation are not of the same quality as the previous sections. A lack of consistency can lead to a terrible reading experience and can affect the result of this report heavily.

Ethics Documentation

A.3 Ethics Confirmation



College of Engineering, Design and Physical Sciences Research Ethics Committee
Brunel University London
Kingston Lane
Uxbridge
UB8 3PH
United Kingdom

www.brunel.ac.uk

16 March 2022

LETTER OF APPROVAL (CONDITONAL)

APPROVAL HAS BEEN GRANTED FOR THIS STUDY TO BE CARRIED OUT BETWEEN 17/03/2022 AND 01/04/2022

Applicant (s):	Mr Majd Barakat
Project Title:	Algorithm Visualizer
Reference:	36299-LR-Mar/2022- 38691-1

Dear Mr Majd Barakat

The Research Ethics Committee has considered the above application recently submitted by you.

The Chair, acting under delegated authority has agreed that there is no objection on ethical grounds to the proposed study. Approval is given on the understanding that the conditions of approval set out below are followed:

- **Approval is given for remote (online/telephone) research activity only. Face-to-face activity and/or travel will require approval by way of an amendment.**
- **The agreed protocol must be followed. Any changes to the protocol will require prior approval from the Committee by way of an application for an amendment.**
- **Please ensure that you monitor and adhere to all up-to-date local and national Government health advice for the duration of your project.**
- Please add the online consent template to your survey before issue (https://breo.brunel.ac.uk/Personalisation/DisplayPage/50)
- Please be advised that recruitment via social media is limited to you creating / sharing a post on your own channels, which participants can respond to or share if they are interested. It does not extend to direct messaging of large numbers of contacts, which can be considered spam.
- These two points may be addressed outside of the BREO system

Please note that:

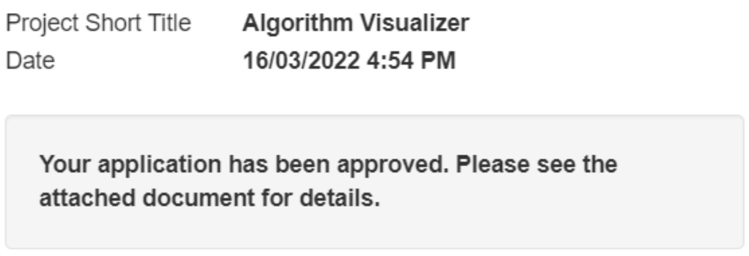
- Research Participant Information Sheets and (where relevant) flyers, posters, and consent forms should include a clear statement that research ethics approval has been obtained from the relevant Research Ethics Committee.
- The Research Participant Information Sheets should include a clear statement that queries should be directed, in the first instance, to the Supervisor (where relevant), or the researcher. Complaints, on the other hand, should be directed, in the first instance, to the Chair of the relevant Research Ethics Committee.
- Approval to proceed with the study is granted subject to receipt by the Committee of satisfactory responses to any conditions that may appear above, in addition to any subsequent changes to the protocol.
- The Research Ethics Committee reserves the right to sample and review documentation, including raw data, relevant to the study.
- If your project has been approved to run for a duration longer than 12 months, you will be required to submit an annual progress report to the Research Ethics Committee. You will be contacted about submission of this report before it becomes due.
- You may not undertake any research activity if you are not a registered student of Brunel University or if you cease to become registered, including abeyance or temporary withdrawal. As a deregistered student you would not be insured to undertake research activity. Research activity includes the recruitment of participants, undertaking consent procedures and collection of data. Breach of this requirement constitutes research misconduct and is a disciplinary offence.

Simon Taylor

Professor Simon Taylor

Chair of the College of Engineering, Design and Physical Sciences Research Ethics Committee

Figure 7-1 – BREO Approval email



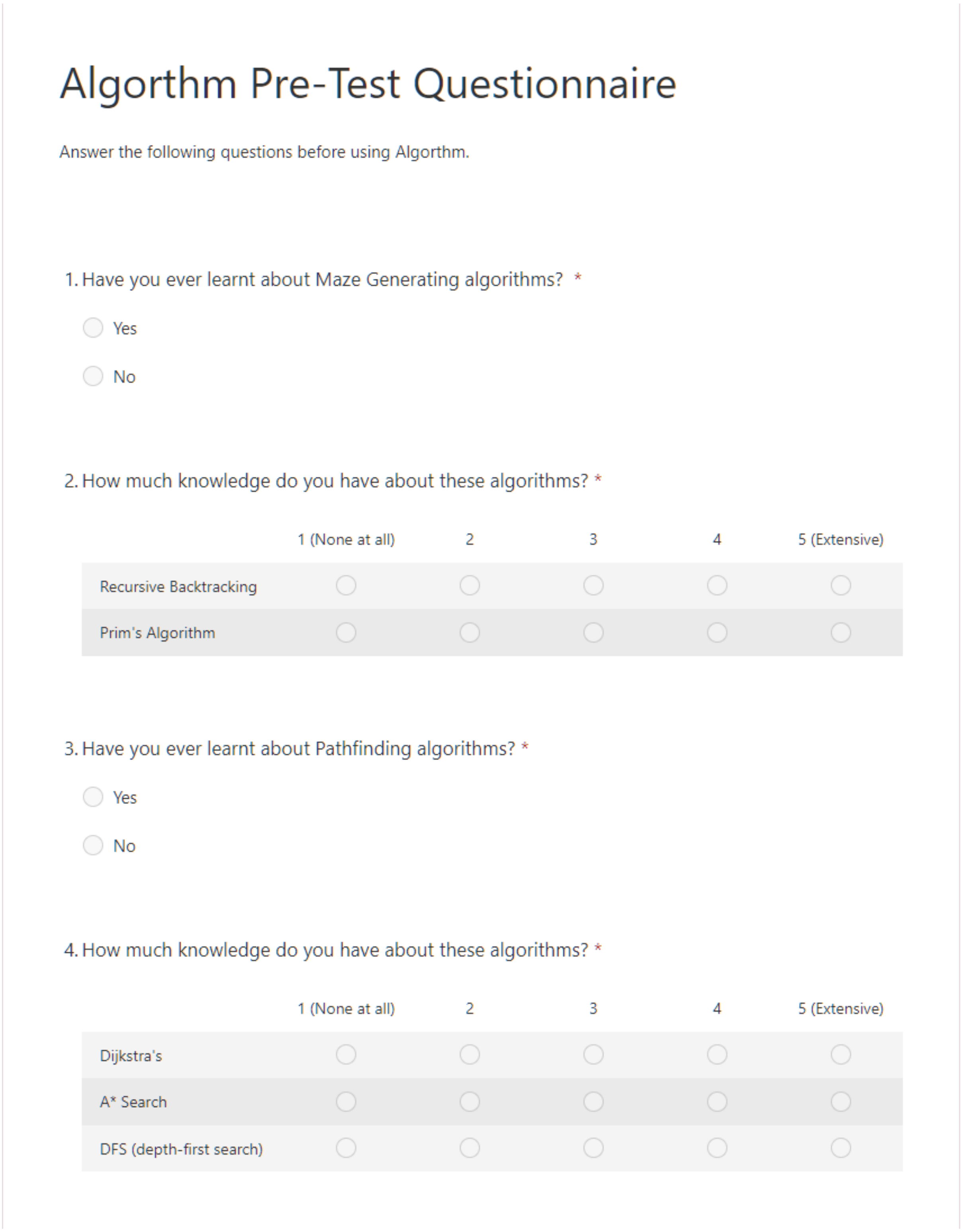
Project Short Title	**Algorithm Visualizer**
Date	**16/03/2022 4:54 PM**

Your application has been approved. Please see the attached document for details.

Figure 7-2 – BREO Approval website

Other Appendices

Questionnaires



Algorthm Pre-Test Questionnaire

Answer the following questions before using Algorthm.

1. Have you ever learnt about Maze Generating algorithms? *

○ Yes

○ No

2. How much knowledge do you have about these algorithms? *

	1 (None at all)	2	3	4	5 (Extensive)
Recursive Backtracking	○	○	○	○	○
Prim's Algorithm	○	○	○	○	○

3. Have you ever learnt about Pathfinding algorithms? *

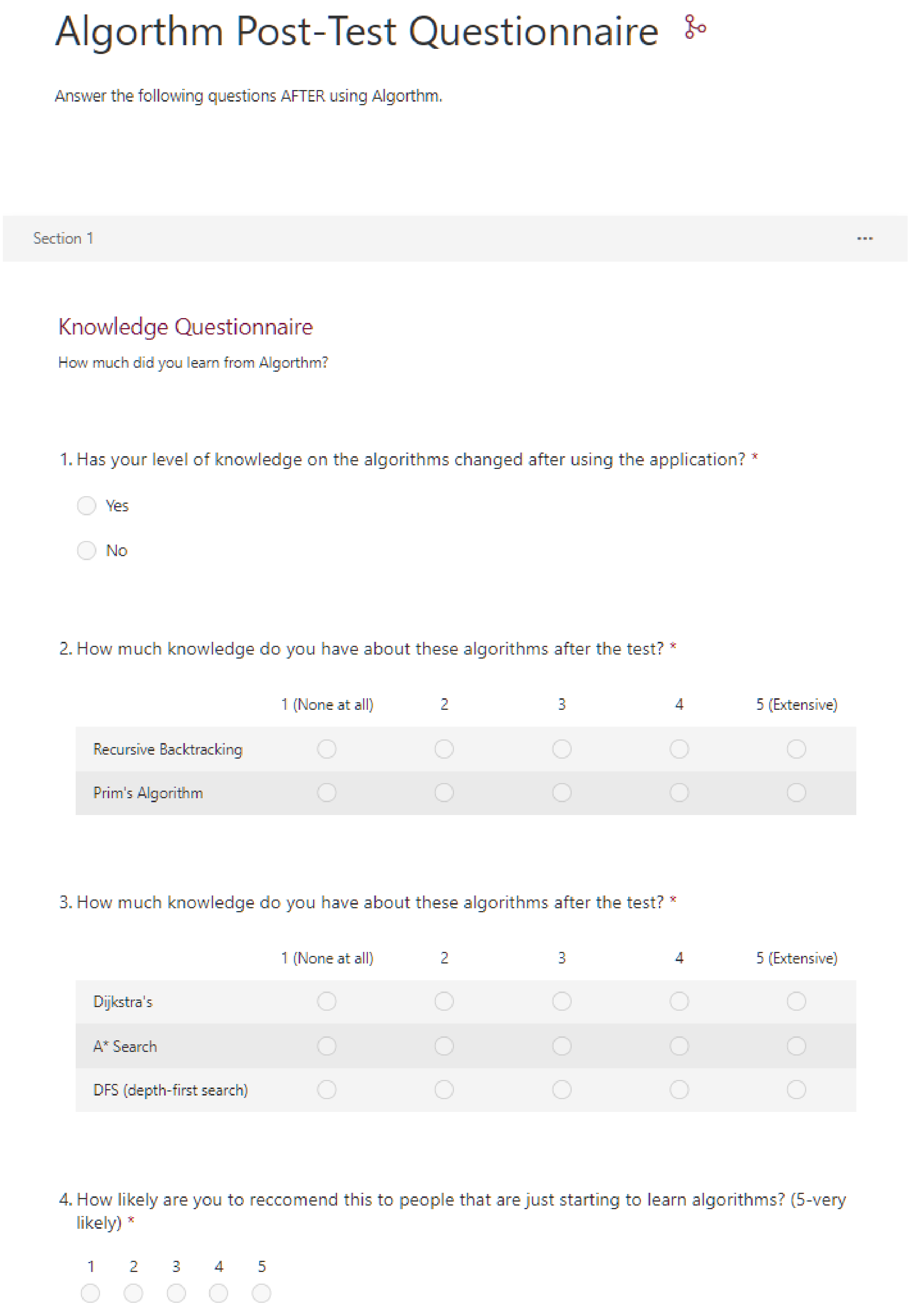
○ Yes

○ No

4. How much knowledge do you have about these algorithms? *

	1 (None at all)	2	3	4	5 (Extensive)
Dijkstra's	○	○	○	○	○
A* Search	○	○	○	○	○
DFS (depth-first search)	○	○	○	○	○

Figure 7-3 – Pre-Test Questionnaire



Algorthm Post-Test Questionnaire

Answer the following questions AFTER using Algorthm.

Section 1 ...

Knowledge Questionnaire

How much did you learn from Algorthm?

1. Has your level of knowledge on the algorithms changed after using the application? *

○ Yes

○ No

2. How much knowledge do you have about these algorithms after the test? *

	1 (None at all)	2	3	4	5 (Extensive)
Recursive Backtracking	○	○	○	○	○
Prim's Algorithm	○	○	○	○	○

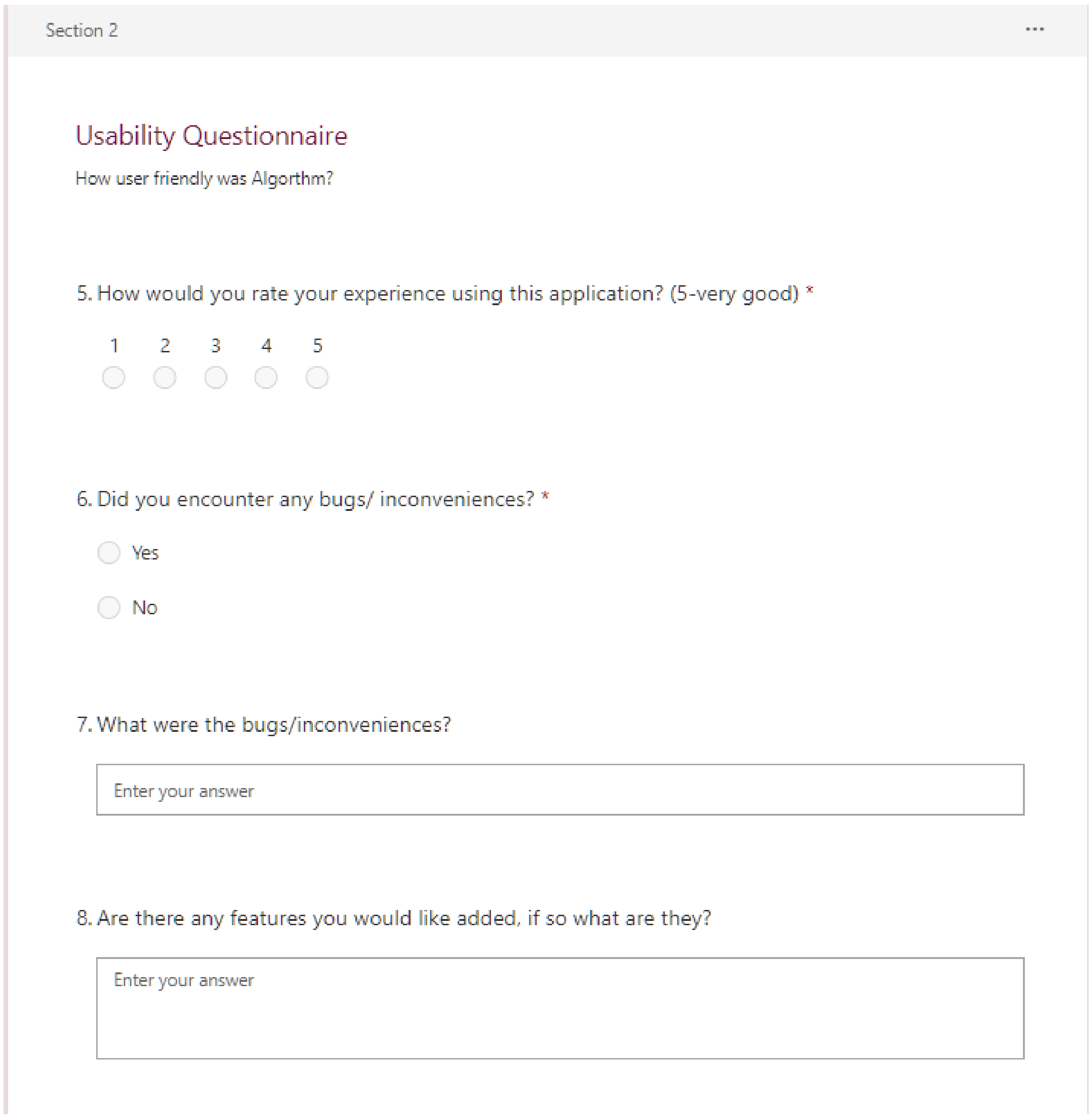
3. How much knowledge do you have about these algorithms after the test? *

	1 (None at all)	2	3	4	5 (Extensive)
Dijkstra's	○	○	○	○	○
A* Search	○	○	○	○	○
DFS (depth-first search)	○	○	○	○	○

4. How likely are you to reccomend this to people that are just starting to learn algorithms? (5-very likely) *

1	2	3	4	5
○	○	○	○	○

Figure 7-4 – Post-Test Questionnaire Section 1


Section 2 ···

Usability Questionnaire

How user friendly was Algorthm?

5. How would you rate your experience using this application? (5-very good) *

1	2	3	4	5
○	○	○	○	○

6. Did you encounter any bugs/ inconveniences? *

○ Yes

○ No

7. What were the bugs/inconveniences?

Enter your answer

8. Are there any features you would like added, if so what are they?

Enter your answer

Figure 7-5 – Post-Test Questionnaire Section 2

Chapter 8 Bibliography

AAUW, 1998. Gender Gaps: Where Schools Still Fail Our Children. Executive Summary. pp. 4-5.

Babson Survey Research Group, 2015. *Online Report Card - Tracking Online Education in the United States,* s.l.: s.n.

Boyde, J., 2014. *A Down-To-Earth Guide To SDLC Project Management.* 2 ed. s.l.:Joshua Boyde.

Buck, J., 2015. *Mazes For Programmers.* s.l.:The Pragmatic Programmers, LLC..

Buck, J., 2015. *The Buckblog.* [Online]
Available at: https://weblog.jamisbuck.org/2015/10/31/mazes-blockwise-geometry.html
[Accessed 1 April 2022].

Chakraborty, A., 2019. *Applications of DFS and BFS in Data Structures.* [Online]
Available at: https://www.tutorialspoint.com/applications-of-dfs-and-bfs-in-data-structures
[Accessed 1 April 2022].

Garcia, S. I., 2018. *L0 Norm, L1 Norm, L2 Norm & L-Infinity Norm.* [Online]
Available at: https://montjoile.medium.com/l0-norm-l1-norm-l2-norm-l-infinity-norm-7a7d18a4f40c#:~:text=Also%20known%20as%20Manhattan%20Distance,the%20vector%20are%20weighted%20equally
[Accessed 1 April 2022].

GitHub, 2022. *Algorthm Repository.* [Online]
Available at: https://github.com/MajdBarakat/Algorthm

Interaction Design Foundation, n.d. *Accessibility.* [Online]
Available at: https://www.interaction-design.org/literature/topics/accessibility
[Accessed 1 April 2022].

Isaac Computer Science, n.d. *Isaac Computer Science.* [Online]
Available at:
https://isaaccomputerscience.org/concepts/dsa_search_a_star?examBoard=all&stage=all
[Accessed 1 April 2022].

Jepson, A. & Perl, T., 2002. Priming the pipeline. *ACM SIGCSE Bulletin,* 34(2), pp. 36-39.

Kinnunen, P. & Malmi, L., 2006. *Why students drop out CS1 course?,* New York: Association for Computing Machinery.

Krayzman, L., Kumar, N. & Lawrence, S., n.d. *Pathfinding InfoPages.* [Online]
Available at: https://mbhs.edu/~lpiper/pathfinding/
[Accessed 1 April 2022].

Ling, E. T. Y., 2015. *Teaching Algorithms with Web-based Technologies,* Singapore: National University of Singapore.

Martin, R. C., 2008. *Clean Code.* s.l.:Pearson.

Mayer, R. E. & Massa, L. J., 2003. Three Facets of Visual and Verbal Learners: Cognitive Ability,Cognitive Style, and Learning Preference. *Journal of Educational Psychology,* 95(4), p. 833– 846.

Memon, M., 2019. *16 Important UX Design Principles for Newcomers.* [Online]
Available at: https://www.springboard.com/blog/design/ux-design-principles/
[Accessed 1 April 2022].

Moreillon, J., 2015. Increasing Interactivity in the Online Learning Environment: Using Digital Tools to Support Students in Socially Constructed Meaning-Making. *TechTrends,* pp. 40-46.

Nichols, D., n.d. *Coloring for Colorblindness.* [Online]
Available at: https://davidmathlogic.com/colorblind/
[Accessed 1 April 2022].

Norman, J. M., 2021. *History Of Information.* [Online]
Available at: https://www.historyofinformation.com/detail.php?entryid=4379
[Accessed 1 April 2022].

Patel, A., 2011. *Amit's Thoughts on Pathfinding.* [Online]
Available at: http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html
[Accessed 1 April 2022].

Plain Language Association International , 2022. *What is plain language?.* [Online]
Available at: https://plainlanguagenetwork.org/plain-language/what-is-plain-language/
[Accessed 1 April 2022].

Porter, J., n.d. *Principles of User Interface Design.* [Online]
Available at: http://bokardo.com/principles-of-user-interface-design/
[Accessed 1 April 2022].

Pullen, W. D., 2021. *Maze Classification.* [Online]
Available at: http://www.astrolog.org/labyrnth/algrithm.htm#solve
[Accessed 1 April 2022].

Sadikan, S. F. N. & Yassin, S. F. M., 2012. Role of Interactive Computer Programming Courseware in Facilitating Teaching and Learning Process Based on Perception of Students in Bangi, Selangor, Malaysia. *International Journal of Scientific & Engineering Research,* 3(8), pp. 2-4.

Sommerville, I., 2016. *Software Engineering.* 10 ed. Pearson Education: s.n.

UIS, n.d. *Strengths and Weaknesses of Online Learning.* [Online]
Available at: https://www.uis.edu/ion/resources/tutorials/online-education-overview/strengths-and-weaknesses/
[Accessed 1 April 2022].

University of Klagenfurt, n.d. *Virtual Exhibitions in Informatics.* [Online]
Available at: http://cs-exhibitions.uni-

klu.ac.at/index.php?id=193#:~:text=Algorithms%20have%20a%20long%20history,of%20the%20term%20%E2%80%9CAlgorithm%E2%80%9D
[Accessed 1 April 2022].

Upadhyay, A., 2022. *Why Data Structures and Algorithms Are Important to Learn?.* [Online]
Available at: https://www.geeksforgeeks.org/why-data-structures-and-algorithms-are-important-to-learn/
[Accessed 1 April 2022].

Yang, X.-S., 2019. *Introduction to Algorithms for Data Mining and Machine Learning.* London: Academic Press.