Maze-solving using A\*

Computer Science Non-Examined Assessment

2021 - 2022

Eleanor Forrest

Candidate Number: 6084

Centre Number: 57017

Table of Contents

[1 Analysis 5](#_Toc98757658)

[1.1 Defining the problem 5](#_Toc98757659)

[1.1.1 Introduction 5](#_Toc98757660)

[1.1.2 Pathfinding algorithms 5](#_Toc98757661)

[1.1.3 The Project 6](#_Toc98757662)

[1.2 End-user Feedback 7](#_Toc98757663)

[1.3 Research 9](#_Toc98757664)

[1.3.1 Current Software 9](#_Toc98757665)

[1.3.2 Key Algorithms 13](#_Toc98757666)

[1.4 Project Objectives 18](#_Toc98757667)

[1.5 Acceptable Limitations 20](#_Toc98757668)

[1.6 Data Volumes 20](#_Toc98757669)

[2 Documented design 21](#_Toc98757670)

[2.1 Overall System Design 21](#_Toc98757671)

[2.2 User Account Management 23](#_Toc98757672)

[2.2.1 User Account Management Interface Design 23](#_Toc98757673)

[2.2.2 Database Design 24](#_Toc98757674)

[2.2.3 SQL 26](#_Toc98757675)

[2.3 Menu 26](#_Toc98757676)

[2.4 Maze Game 27](#_Toc98757677)

[2.4.1 Maze Game Interface Design 27](#_Toc98757678)

[2.4.2 Randomised Depth-First Search 32](#_Toc98757679)

[2.4.3 Playing the Game 33](#_Toc98757680)

[2.4.4 Solving the Maze 33](#_Toc98757681)

[2.5 A\* Simulation 34](#_Toc98757682)

[2.5.1 A\* Simulation Interface Design 34](#_Toc98757683)

[2.5.2 The A\* Algorithm 35](#_Toc98757684)

[2.5.3 Objects 38](#_Toc98757685)

[2.5.4 Visualising the Algorithm 40](#_Toc98757686)

[2.6 User Feedback 41](#_Toc98757687)

[2.7 Object Oriented Programming design 42](#_Toc98757688)

[2.7.1 The Cell class 42](#_Toc98757689)

[2.7.2 The Sprite class 44](#_Toc98757690)

[2.7.3 The Stopwatch Class and Threading 45](#_Toc98757691)

[2.8 File organisation 46](#_Toc98757692)

[2.9 Key variables 47](#_Toc98757693)

[3 Technical solution 50](#_Toc98757694)

[3.1 Code contents 50](#_Toc98757695)

[3.2 Achieved objectives 52](#_Toc98757696)

[4 Testing 56](#_Toc98757697)

[4.1 User Experience based testing 56](#_Toc98757698)

[4.2 Detailed testing 68](#_Toc98757699)

[4.2.1 User Account Management 69](#_Toc98757700)

[4.2.2 Menu 70](#_Toc98757701)

[4.2.3 Maze Game 70](#_Toc98757702)

[4.2.4 A\* Simulation 76](#_Toc98757703)

[4.2.5 Fixing failed tests 79](#_Toc98757704)

[4.3 Extra user interface evidence 84](#_Toc98757705)

[4.4 Testing against project objectives 92](#_Toc98757706)

[5 Evaluation 96](#_Toc98757707)

[5.1 Whole system evaluation 96](#_Toc98757708)

[5.2 Differences from the Design section 97](#_Toc98757709)

[5.3 Evaluation against project objectives 98](#_Toc98757710)

[5.4 End-user evaluation 101](#_Toc98757711)

[5.4.1 Feedback from my end user 101](#_Toc98757712)

[5.4.2 Analysis of feedback 102](#_Toc98757713)

[5.4.3 Other suggestions 103](#_Toc98757714)

[5.5 Conclusion 103](#_Toc98757715)

[6 References 105](#_Toc98757716)

[6.1 Bibliography 105](#_Toc98757717)

[6.2 Table of Figures 105](#_Toc98757718)

[7 Appendix 107](#_Toc98757719)

[7.1 Appendix A – Prototype of the A\* algorithm 107](#_Toc98757720)

[7.2 Appendix B – Technical Solution 109](#_Toc98757721)

[7.2.1 Class definitions 109](#_Toc98757722)

[7.2.2 Key Algorithms 115](#_Toc98757723)

[7.2.3 Main program and Menu 120](#_Toc98757724)

[7.2.4 User Account Management 127](#_Toc98757725)

[7.2.5 Maze Game 128](#_Toc98757726)

[7.2.6 A\* Simulation 148](#_Toc98757727)

# 1 Analysis

## 1.1 Defining the problem

### 1.1.1 Introduction

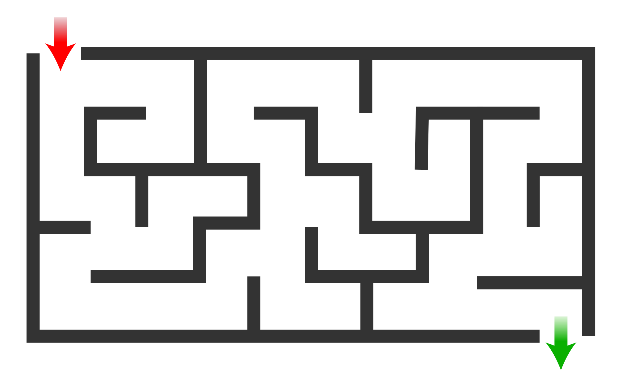
Mazes have been a popular form of puzzle for years, whether by tracing a route using paper and pencil or in the form of a large, walled labyrinth that the puzzler traverses on foot. The idea is to start at the entrance and navigate through the walls to the exit, avoiding a multitude of incorrect paths which lead to dead ends. The correct route is often convoluted and full of turns, making it a challenge to try to find a way through.

Figure : A simple maze (Wikipedia, 2021).

For a computer to attempt such a puzzle, a set of well-defined, logical rules must be outlined in order to find a solution. This is called an algorithm. Possibly the simplest of these is the ‘wall follower’ rule: keep one hand on a wall at all times, and you will almost always reach the exit eventually. While this does provide a solution, it often won’t be the fastest route through the maze. However, there are plenty of other algorithms which will find the shortest pathway.

### 1.1.2 Pathfinding algorithms

In order for a maze to be simulated by a computer, it must be modelled as a graph (a set of nodes connected together by edges). In this case, the traversable passages are represented as nodes, and openings in the walls of the maze are edges that can be crossed – so if there is a wall between two passages, there is no edge between the corresponding nodes, as you can’t travel that way. We are trying to find the shortest path from the start node to the end node.

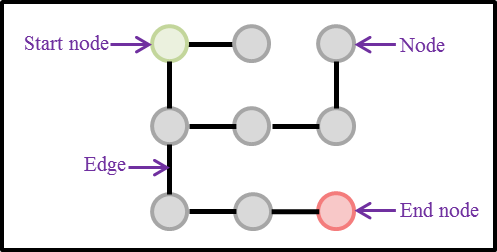


Figure : An example of a graph that could represent a maze.

There are many different approaches to this problem:

* Explore equally in all directions until the exit is found (Breadth First Search)
* Prioritise shorter paths by calculating the distance of each node from the start node (Dijkstra’s Algorithm)
* Prioritise shorter paths by estimating the distance of each node from the end node (Greedy Best First Search)
* Use both the distance from the start node and the estimated distance from the end node to calculate the optimal path (A\*)

Of these, the A\* pathfinding algorithm is optimal for maze solving. Breadth First Search and Dijkstra’s Algorithm often find the quickest solution, but spend a lot of time exploring away from the goal. Greedy Best First Search prioritises paths closer to the goal, but may not find the optimal solution if it encounters an obstacle – namely a wall in the maze. A\* uses a combination of Dijkstra’s Algorithm and Greedy Best First Search in its calculations, resulting in it finding the optimal solution while exploring much less than Dijkstra’s Algorithm does.

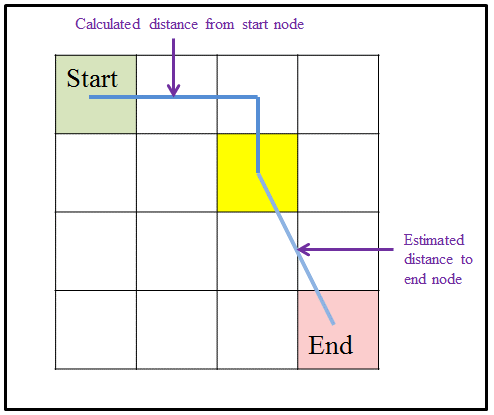


Figure : The A\* algorithm includes the distance from both the start node and the end node in its calculations for each node.

### 1.1.3 The Project

However, there are very few resources that demonstrate this algorithm in action, especially in a way that the user can interact with. It is difficult to compare the algorithm with how a human would approach a maze puzzle. Therefore, in this project I will create a resource that allows a user to see and interact with the A\* algorithm.

The project will consist of two modes. In the first, the user will be able to play a maze that has been randomly generated by the program, attempting to complete it as quickly as possible. This time will be used as a score system, with their fastest times being saved to a database. Should they struggle, they can choose to see the answer, in which case the A\* algorithm will be used to produce and display the solution to the maze.

In the second mode, the user can create their own maze to be solved by the algorithm. They start with a blank grid and can add walls and obstacles, as well as a start and end node. Once they are happy with their maze, the A\* algorithm will be used to find the shortest path through. This will include changing the colour of the cells that are being checked, with a time delay to clearly show the process in action.

This program will require a variety of technical skills, including simulating and traversing the graph, recursive algorithms (used when generating the maze) and implementing the A\* algorithm. In addition to this, some challenges I will face while completing the project include:

* Learning to use the Tkinter module to create a suitable and navigable user interface.
* Planning deadlines for each aspect of the program and documentation.
* Getting feedback on time and responding effectively.
* Using SQL and a database to manage the user’s scores, something I have very little experience with.

Currently, there are few resources allowing for easy visualisation of the A\* pathfinding algorithm and how it performs in a maze environment. I am creating this resource to provide a fun way to view and interact with the algorithm, as well as let the user have a go at some mazes themselves.

## 1.2 End-user Feedback

The target audience for my game is anyone with an interest in maze puzzles, but in particular students and young people who are inquisitive into how a computer could attempt to solve such a puzzle. Therefore the interface should be interactive and educational, allowing them to explore how the algorithm solves different styles of mazes and gain an understanding into how it works. The maze section should be easy to navigate and controls should be intuitive, encouraging the user. Customisability will allow them to tailor the experience to their individual preferences.

I spent some time discussing my project with my friend and fellow puzzle enthusiast Noah, my end user, in the form of an informal interview. They were able to specify features that would be necessary to include, as well as some that would help it to be appealing and user-friendly:

**Do you have any ideas as to how the maze section would look?**

‘It should be clear where the start and end of the maze is, and there should be a sprite showing your current position. Also, when the sprite moves the cells it has traversed should light up – the aim is to make it to the end while lighting up the fewest amount of squares. I really think the interface should be colourful and fun – or at least there should be an option to make it so. When the user completes the maze there needs to be some kind of celebration to reward the user and encourage them to play again.’

**What features do you think should be included? What extra features would make the game more enjoyable?**

‘There should definitely be a time aspect to the puzzle, so I can try to beat my high score. You should also be able to see the answer, with the least possible number of squares lit up. If you have time, you could include different modes, such as one where you try to collect objects such as coins scattered in the maze, or a mode where you can choose to remove a select number of walls to make it easier.’

**What control system would be easiest to use?**

‘I think it would be best to use the arrow keys for movement. However, you could allow the user to plan a route using the mouse and then follow it using the arrows.’

**What would make the game less enjoyable?**

‘If the timer is too big it could become a source of stress. Also don’t make the mazes too hard!’

**Do you have any ideas on how to make the A\* section interactive to the user?**

‘You should be able to bring up two signs: one explaining how to use the software and the other giving an insight into how the algorithm works and what it is doing. While these don’t need to be brought up immediately, links to the signs should be easily identifiable. At the bottom of the grid should be space to select the block you would like to place with the mouse. Also, when the algorithm is running it should use different colours for the squares it is checking and those it has already checked so it is easy to distinguish between them.’

**What kinds of obstacles could I include?**

‘It would be good to have a couple of obstacles that take a different amount of time to cross, such as water or cobwebs. There could also be warp panels that allow for teleportation to another space on the board. That would be an interesting mechanic.’

**Do you have anything else to add?**

‘The sprites used don’t need to be too complicated – it can be more fun just using stylised blobs and simple shapes, instead of thinking too hard about what each object represents.’

From this interview I was able to gain a clear idea of how I can make the software accessible to my target audience and which features would really help it to stand out. Some of their ideas may be too challenging to code in the time frame I have, however the key points I have gained from this interview are:

* Make sure the maze section is intuitive – you should be able to look at it and clearly see where the start and end is as well as where you are currently. Including a trail from the start will help the user to see where they have already travelled.
* Ensure that the interface is customisable so that the user can tailor it to their preferences. This will mean that they can select colour schemes based on their mood and personality, making it more appealing to a wider range of users.
* Include a timer and high score system to add an element of competition to the maze, encouraging the user to play again. A ‘congratulations’ type message would aid with this encouragement.
* The user should control a simple sprite, which moves using the arrow keys.
* The user should be able to view instructions on how to use the A\* section, as well as learn about why the algorithm is used and how it works.
* Use different colours to show the cells that the algorithm is checking or has already visited, in order to allow the user to easily visualise it.
* To make the A\* section more engaging, include obstacles that vary how the algorithm calculates the priority of a certain cell.

## 1.3 Research

### 1.3.1 Current Software

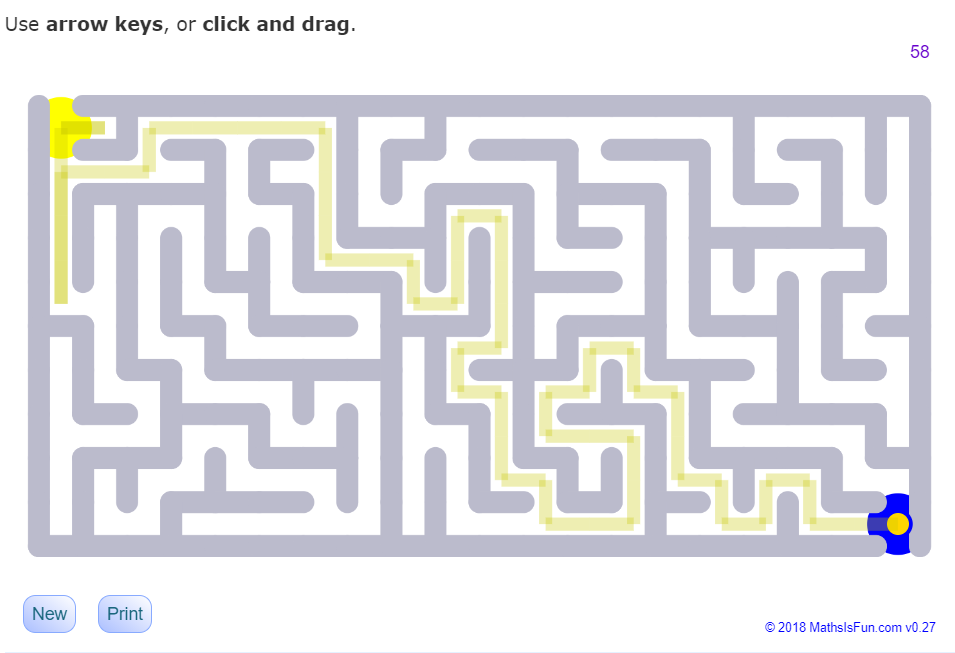
I also researched some already-existing software to help inform my project. The first of these was an online maze game hosted on the [*MathIsFun*](https://www.mathsisfun.com/measure/mazes.html) website. While there are a range of maze puzzles available online, this site stood out to me because it was one of the few where the mazes appear to be randomly generated instead of pre-set – like my product will be.

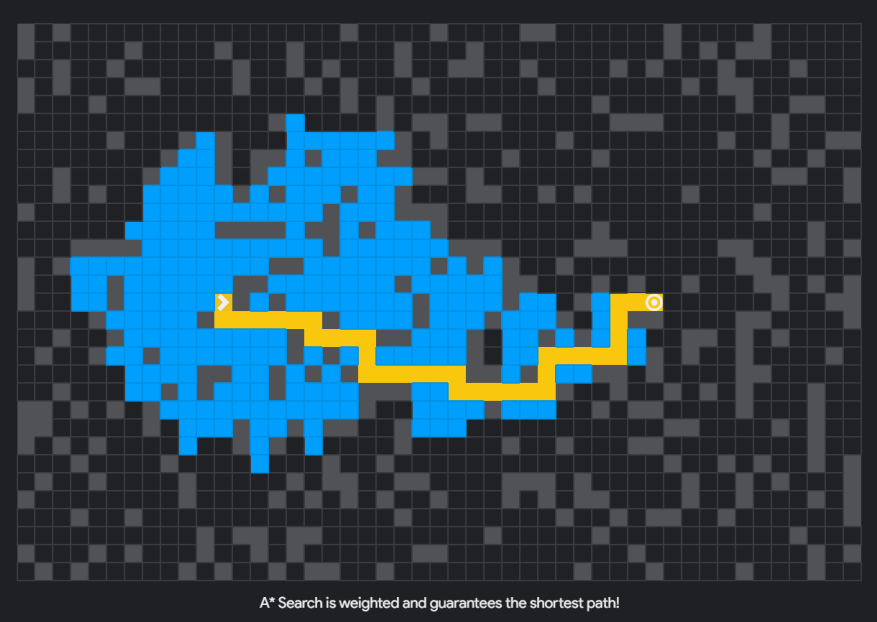
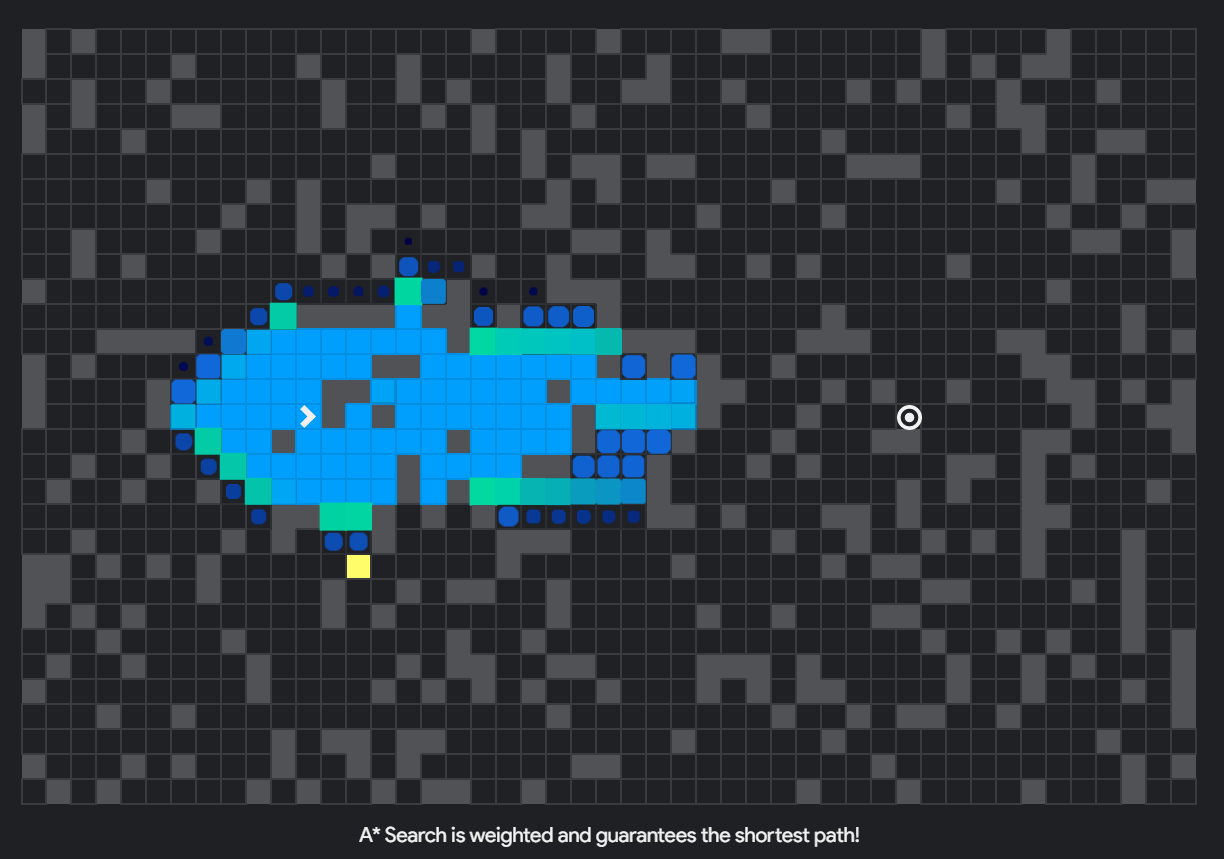
Figure : Screenshot from the MathIsFun maze platform (Pierce, 2020).

Figure 4 presents a screenshot of one of the mazes produced by the site. You play as the yellow dot, using the arrow keys to navigate one space at a time to the blue circle.

|  |  |
| --- | --- |
| Successful features   * The maze is simple and it is easy to see what you are aiming to do. While this isn’t particularly hard for a maze puzzle, it is made clear where the exit is and there are no distractions. * Instructions are made clear to the user, and it is obvious where you need to click to try a new puzzle. This is essential to user experience. * You can see the path you have taken in order to reach the goal. * When a maze is complete, a bubble appears stating your time and number of steps taken. There is a step counter in the top right corner but no timer while you are playing, reducing stress. * You can choose a difficulty level for the maze, as well as a range of different styles for the walls, such as using dots or rainbow coloured rectangles. | Unsuccessful features   * The controls are a little jagged – the maze is based on an invisible grid, and each time you press an arrow key, the dot jumps to the next space. Holding the key down moves multiple spaces at once which can be difficult to control. Using the mouse is even more awkward, as the dot attempts to follow the mouse while being confined to the walls of the maze. * As seen in Figure 4, a maze may be generated where it is impossible to access certain areas. This was quite a common occurrence which was disappointing. * Choosing hard difficulty pushes more walls into the same sized space, which becomes quite small and difficult to see. * The walls are curved but you can only move on a set grid. Also some of the customisation options make it very difficult to work out where the walls actually are. |
| What I will take forward into my project   * Make sure it is clear to the user how to play and where the entrance and exit to the maze are, as well as how to start a new puzzle or see the answer to the current one. * Include a trail to show where the user has been so far. While the one used in this software is sufficient, I think it would be more appealing to have the whole cell change colour, thus removing the white edges and overlapping corners. * Show the time taken once the user completes the maze. I think including a timer will add some pressure to the game making it more enjoyable, however this mustn’t be too prominent and therefore a distraction. * Include a difficulty option (likely easy, medium and hard puzzles); however ensure that the more difficult mazes don’t become too small to follow. * I liked the fact that there are customisation options for the maze; however, the ones available weren’t very appealing to me. I think that instead of just changing the colour and shape of the walls and leaving the background, I will let the user select between colour themes which affect both the background and the walls, having a larger effect on the overall mood. Keeping the walls rectangular will ensure that it is obvious where you can travel and where you can’t. * Personally, I found that using the arrow keys was much more natural when traversing the maze, so this is the control scheme I will use. * If possible, ensure that all areas of the maze will always be accessible. This may depend on the algorithm I use to create the maze. | |

The second half of my project was less commonly found online. I found a site called [PathVis](https://pathvis.datsko.dev/) however, which allowed for visualisation of A\* as well as other algorithms such as Dijkstra’s and Greedy Best First Search.

Figure : Screenshots from the PathViz platform during (top) and after (bottom) an A\* visualisation (PathViz, n.d.).



The top screenshot was taken during the visualisation, after I had chosen to set up a ‘basic random maze’ wall pattern in the grid. The bottom screenshot is what is presented once the target has been found.

|  |  |
| --- | --- |
| Successful features   * You can move the start and target around the grid, and it is clear which is which. * It is easy to draw and remove walls simply by clicking or dragging the mouse. * There is a control panel at the side of the screen allowing you to choose an algorithm, create a maze, clear the board and adjust the speed of the visualisation, as well as a button to start the algorithm. They each have icons to help the user navigate them. * The cells that are being checked by the algorithm appear on screen, growing and becoming a light green, before fading to blue once they have been checked. * Once the algorithm has found the shortest path, it terminates and redraws the shortest path in a contrasting yellow. * There is a set of slides available explaining what the software does and how to use it, and giving a brief overview of each algorithm. | Unsuccessful features   * The dark theme means that it is difficult to distinguish the walls from the background of the grid. Moreover, the colour of a square changing from blue to green and then back to blue makes it difficult to tell newly checked cells apart from already checked ones. * The visualisation has a yellow square that jumps around the grid to show each cell that is checked at a time. While this does the job, I find it too rapid to follow even at slower speeds. * The program does not explain how each algorithm actually works in detail. |
| What I will take forward into my project   * Allow the user to draw walls with the mouse and choose a start and end position. * Include options to clear the board, start the algorithm and, if I have time, adjust the visualisation speed. * Use different colours to represent the cells in the frontier (all those currently being checked) and those already visited. Make sure these are distinct to avoid confusion, and cycle through the algorithm slowly enough that the user can watch as the frontier changes. * Once the shortest path has been found, highlight it in a different colour. Make sure the algorithm terminates at this point. * Include an option to bring up a screen explaining how the algorithm works and calculates which cells to check. | |

While my project will be inspired by both of these, there are a few other ideas I have that weren’t present in these projects. The first is an account system, where a user’s fastest times in the maze section are saved to a database, so that they can try to achieve new personal bests. Another is to include obstacles in the A\* section that take time to cross or transport the frontier to another area (i.e. a warp panel). Finally, I think that when the user chooses to see the solution to a maze, the A\* algorithm must be used to solve it, so that they can see how it responds to the maze that they have just attempted. My program needs to be educational and customisable to suit the preferences of my end users.

### 1.3.2 Key Algorithms

The program will require two major algorithms: one for maze generation, and, of course, the A\* algorithm itself. In order to generate the maze, I had a couple of options for algorithms I could use, each producing a slightly different style of maze. I chose to use a method called **randomised depth-first search**, which tends to create many long corridors with less branching paths.

Firstly, the maze should be modelled as a complete grid, represented using a graph, with each cell as a node and each wall as an edge. The algorithm is then called with the first cell given as a parameter. Figure6 presents a flow diagram of the algorithm:

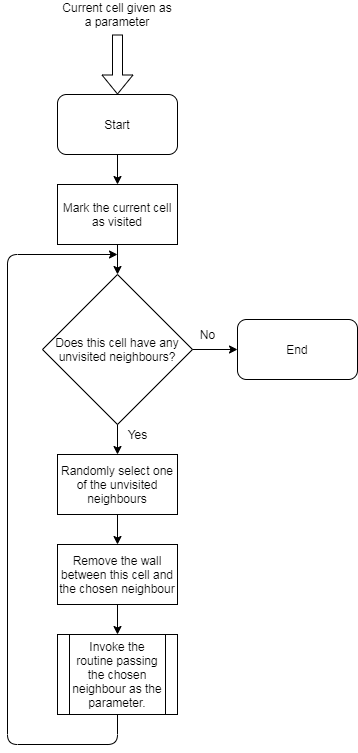


Figure : A flowchart of the randomised depth-first search algorithm.

The algorithm randomly selects a neighbouring cell (a cell directly above, below, to the right or to the left of the current cell) that hasn’t yet been visited, removes the wall between the two cells and then calls the same routine using the neighbour cell. This causes it to keep being called until a cell with no unvisited neighbours is reached, at which point it will backtrack through the cells it has visited until a cell with some unvisited neighbours is found. The algorithm always visits every cell, before returning to the first cell. Therefore, when the first cell has no unvisited neighbours the routine ends.

A potential issue with this algorithm is a stack overflow error if the subroutine calls itself too many times. In this case however, the maximum depth of recursion will never exceed the number of cells in the grid, as once a dead end is reached it backtracks out of the recursion. In Python, the maximum depth is around 1000, meaning as long as my grid is smaller than √1000 (31\*31) cells I won’t hit this limit.

The A\* algorithm is a weighted pathfinding algorithm, which means that it calculates a cost to travel to each cell, and prioritises cells with the lowest cost. It does this using two factors: the recorded distance from the start node, and an estimation of the distance to the end node. Calculating the distance from the start node is easy – just keep track of how many cells have been traversed so far – but the distance from the end node cannot be determined. Instead, the algorithm estimates it using a heuristic – the cost of a cell is determined using the formula:

***f(n) = g(n) + h(n)***

where ***n*** is the cell being checked, ***f(n)*** is the total cost of travelling to that cell, ***g(n)*** is the distance from the start node and ***h(n)*** is a heuristic. I explain how this heuristic is calculated in Section [2.5.2](#_2.5.2_The_A*). Here is a flow diagram presenting the A\* algorithm:

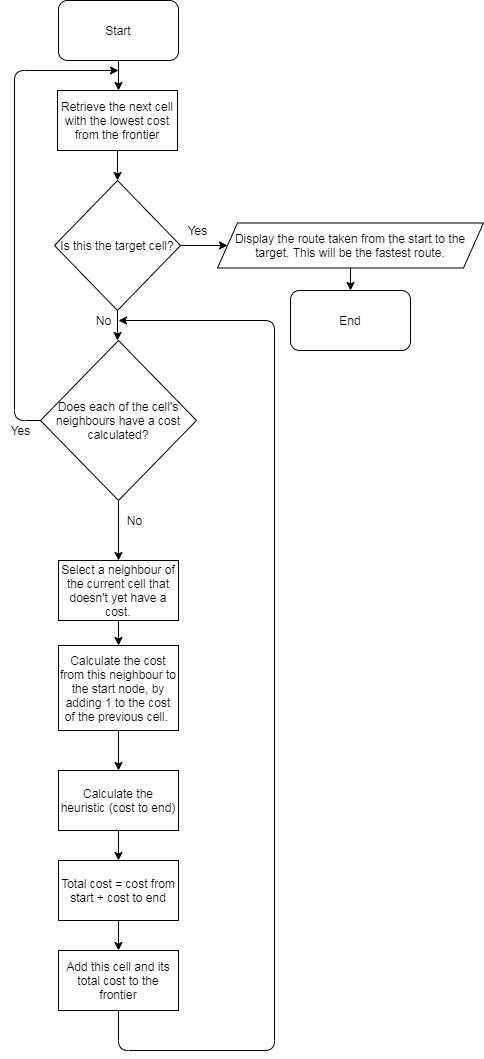


Figure : A flowchart of the A\* algorithm.

The frontier is a priority queue, meaning that each cell is added along with its calculated cost. When a cell is retrieved from the queue, this cell will always be the cell with the lowest cost, so paths with the lowest cost are prioritised; hence the route returned is guaranteed to be the shortest route.

I decided to develop a prototype of this algorithm, as it is a core part of both sections of my program. This prototype had no user interface and simply tested if, when provided the size of grid and target cell, the algorithm could find the shortest route to the target cell from the first (top left) cell. See [Appendix A](#_7.1_Appendix_A) for the code used in this prototype.

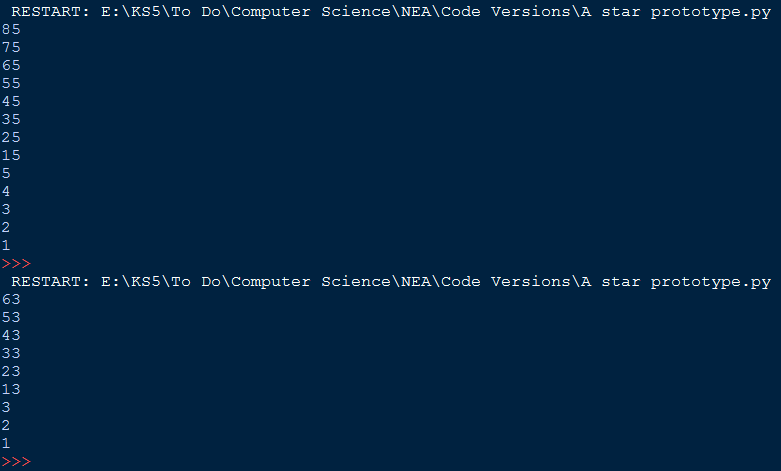
I tested this using a 10\*10 grid, with a range of target cells. Each time, the shortest route that involved traversing the top row and then going vertically downwards was selected:

Figure : A screenshot of the output from the prototype, using 85 and 63 as target cells.

This is always correct, and while there are other shortest pathways through an empty grid, my program will only ever produce one, so this was sufficient. However, alongside this, I was able to test if ithe prototype could navigate a wall, by adding the *and* statement: ‘if checkingNeighbour not in cameFrom and checkingNeighbour != 55:’. Here is the result:

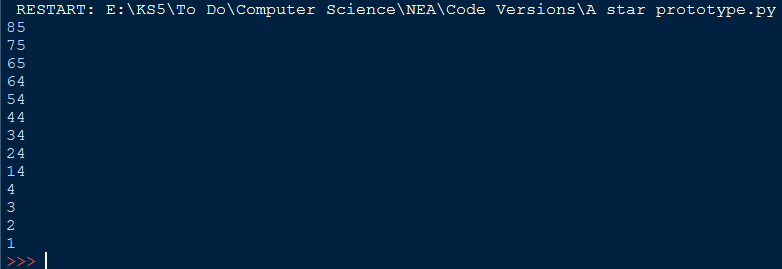
Notice the algorithm now makes use of the fourth column in order to avoid the makeshift wall in the grid. This proves that it does in fact work as intended when walls are present, as in my project.

Figure : Testing the prototype after cell 55 was removed.

Along with these algorithms, some other processes I need to consider include: simulating the grids as graphs; handling user inputs while they are playing or creating a maze; and recording their details and high scores to a database so that they can be retrieved. On top of this I need to create a functional and interactive user interface so that the system can be used and enjoyed by my end users.

## 1.4 Project Objectives

Here is a list of the requirements for my project:

1. Upon starting, the program will prompt the user to register or log in to their account.
   1. They will have to register a username and password.
      1. All usernames are stored in a database. This username must be checked against those in the database to make sure it is unique. This database will be accessed using SQL.
      2. If the username is taken the user should be alerted.
      3. The password will also be stored in the database. It should be encrypted using a hashing algorithm instead of being stored simply as plaintext.
   2. Once the account has been successfully created (or a user with an already existing account has logged in) they will be directed to the main menu.
2. The main menu will allow the user to select which mode they wish to play.
   1. There will be a button for the maze game and a button for the A\* simulator.
   2. They will also be able to sign out from this screen, returning to the first screen.
3. In the maze section:
   1. The user will select a difficulty (easy, medium or hard).
   2. A maze will be generated using the randomized depth-first search algorithm, its size depending on the difficulty selected in objective 3.1.
   3. The user can then play the maze, using the arrow keys to navigate a simple sprite from the start cell to the target cell.
      1. Instructions to use the arrow keys to play should be clearly displayed.
      2. When the sprite moves a trail should be displayed showing where it has been.
      3. There will be a stopwatch displayed on this screen so the user can see the current time taken.
   4. When the user completes the maze, a congratulations message will be produced, which states the time taken and the user’s current high score.
      1. The user’s fastest time (on this difficulty) is taken as their highest score and saved to the database.
      2. Should the user beat their high score, this will be updated in the database and the new score will be displayed the next time they beat a maze of the same difficulty.
   5. There will be an option to see the answer. When clicked the A\* algorithm will be used and the shortest route through the maze will be displayed.
      1. No time will be recorded for this attempt. Instead the user must click the play again button (see objective 3.6) to start a new maze.
   6. There will also be options to play again, change difficulty level or return to the menu.
   7. There will be a customisation option which allows the user to select a colour theme for the maze.
4. In the A\* section:
   1. An empty grid will be displayed.
   2. At the bottom of the screen the user can select what to place with their mouse:
      1. A solid wall.
      2. A start node (only one).
      3. An end node (only one).
      4. An obstacle which takes longer to cross than an empty cell.
      5. A warp panel that moves the frontier across the grid.
   3. Once the user has drawn their maze, they can select the start button.
      1. If a start and end node haven’t been placed the user should be notified.
      2. The A\* algorithm will then run, changing the colour of each cell it visits. There will be a short delay so that the user can see the changes occurring.
      3. When the shortest path has been found it will be highlighted in another colour.
   4. There should be an option to clear the grid.
   5. There should be a button containing information about this section of the software:
      1. How to interact with the area (place objects etc.)
      2. What the A\* algorithm is and does (finds the shortest path from one node to another)
      3. How the algorithm works (explaining how the cost for each cell is calculated etc.)
   6. There should also be an option to return to the menu.

## 1.5 Acceptable Limitations

Some potential limitations of my project include:

* Customization will likely be limited to changing the colour of walls, buttons and backgrounds due to complexity.
* Creating a maze may be slow due to the recursive function.
* I will be unlikely to have time to incorporate entirely new game modes into the maze section as suggested by Noah.
* Similarly, I am unlikely to have time to allow patterns (such as the basic random maze from Figure 5) to be generated within the A\* section – it will all have to be done by the user.

## 1.6 Data Volumes

My program will store the following data per user:

* Username
* Password
* High score on Easy difficulty
* High score on Medium difficulty
* High score on Hard difficulty

My program will not have a large number of users, so I don’t foresee any issues with data storage affecting my project in any way.

# 2 Documented design

## 2.1 Overall System Design

My program is split into four major sections, as demonstrated in the objectives. The first is User Account Management – when the program is run the user will be asked to register or log in to an existing account. This section handles communicating this data to the database, and directs the user to the second section, the Menu. Here they can choose to play the Maze Game or A\* Simulation, or sign out.

In the Maze Game section, the user will be asked to select a difficulty (easy, medium or hard). A maze will then be generated using randomized depth-first search, its size dependent on this difficulty. Instructions to use the arrow keys to play will be displayed as well as a stopwatch, and the user can attempt to navigate a sprite from the start cell to the target cell. When they reach it a congratulations message will be produced, which displays their time taken and current high score for this difficulty. If they beat their previous high score, the new time is displayed here and updated in the database. There will also be options to create a new maze, change difficulty, choose a colour theme or return to the menu, as well as an option to see the answer, in which case the A\* algorithm will be used to find and display the shortest route.

In the A\* Simulation section, an empty grid will be displayed, with a selection of objects to be added (walls, start and end nodes, obstacles and warp panels) at the bottom. The user can click these and click on the grid to place them. Once done, they will select the run button and, if a start and end node have been placed, the A\* algorithm will be displayed as it works on their grid. Cells in the frontier will become one colour, and already visited cells another. Once the shortest route is found it will be displayed in a third colour, making it stand out to the user. There will also be options to clear the grid, view information about the algorithm and return to the menu.

Figure 10presents a hierarchy diagram for the whole project, giving brief details of the requirements for each section.

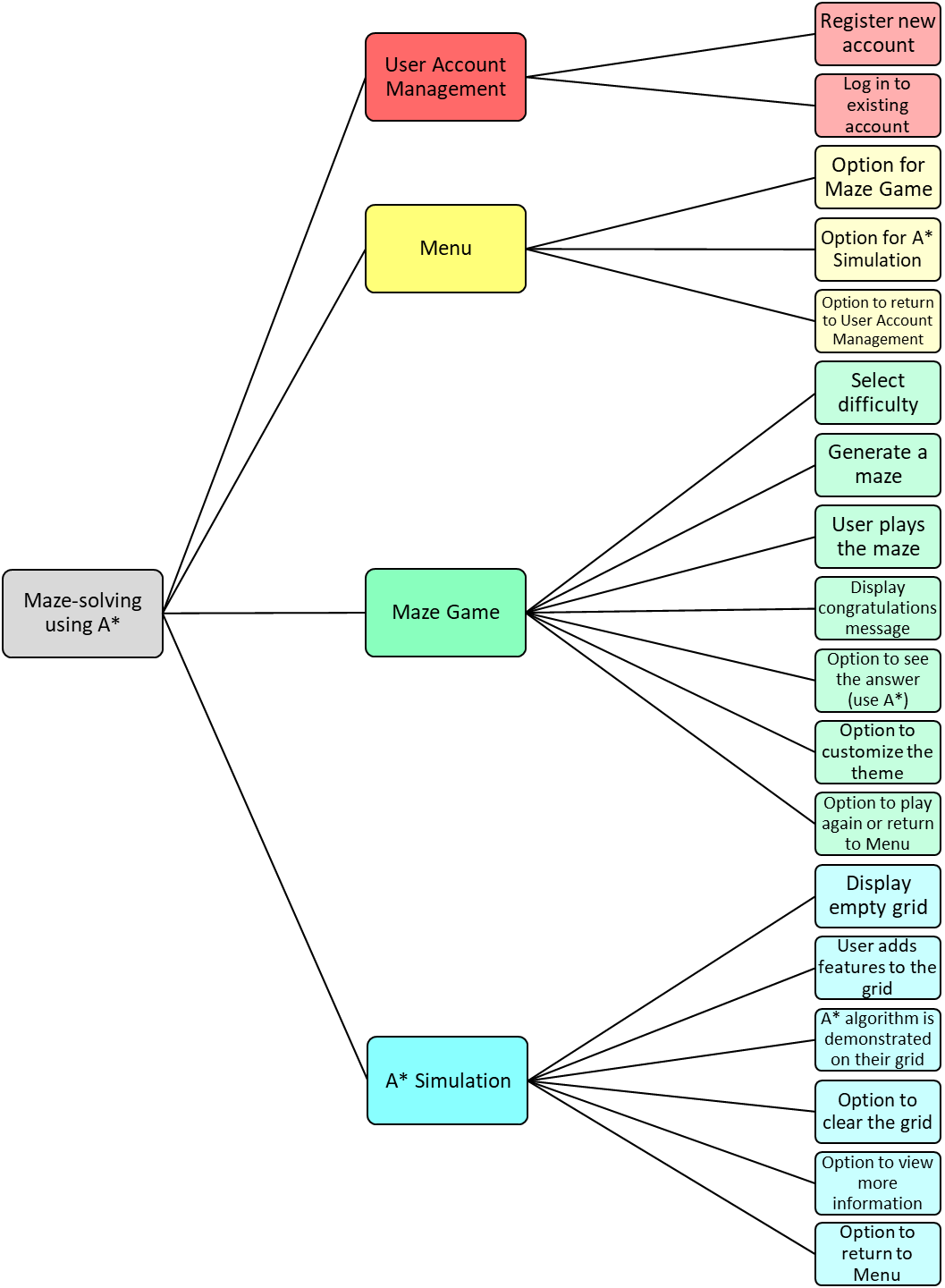


Figure : A hierarchy diagram of the system as a whole.

By splitting the project into sections, I can complete and test one section before focusing on the next, without having to worry about whether the previous sections will still work. This is also true for some of the sub-sections.

## 2.2 User Account Management

### 2.2.1 User Account Management Interface Design

The first page of the program will ask the user if they wish to register or log in to an account. It will look something like this:

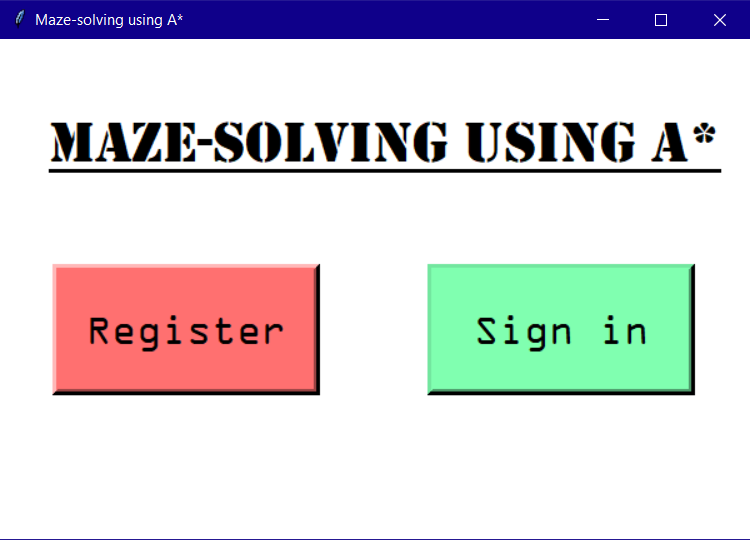
They will then be met with a form, asking them to enter a username and password:

Figure : A design of the first page of User Account Management.

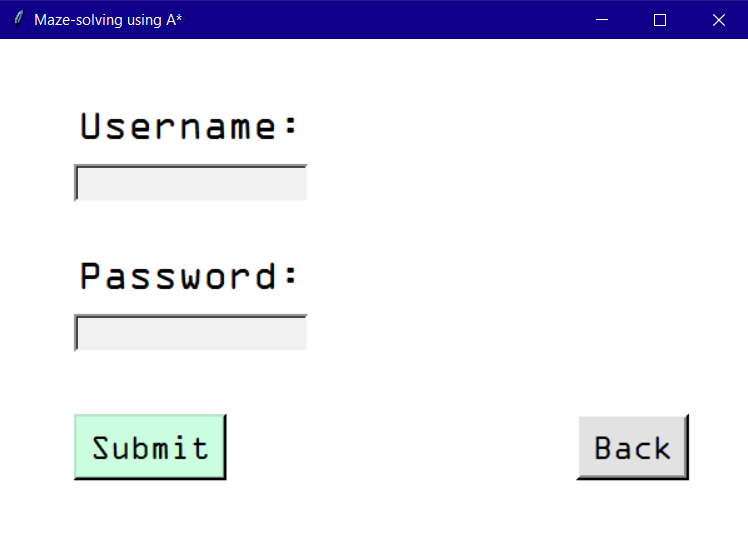


Figure : A design of the User Account Management login form.

### 2.2.2 Database Design

This information is then communicated with the database. Figure 13 presents a plan of the data table I will use:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **User ID** | **Username** | **Password** | **Easy High Score** | **Medium High Score** | **Hard High Score** |
| 1 | eleanor1 | dvb5dgf79hf9… | 00:07 | 00:15 | 01:30 |
| 2 | smithj | 855u3gls6w2… | 00:10 | 00:27 | 00:57 |
| 3 | mr\_kitten | j78d2346gre9… | 00:04 | None | 01:18 |

Figure : Data table recording user credentials and high scores

When registering, the username must be checked against all other usernames in the database to make sure it is unique. If not, the user should be prompted to enter a different username. When their username is unique, the password will be hashed using the SHA-256 hashing algorithm, which turns it into a 256-bit (64-character) string of seemingly random characters. This keeps the passwords secure: if a hacker were to infiltrate the database and find the resulting hash, it would be practically impossible to discern what the original password was and therefore gain access to the account. The username and hashed password will then be stored in the database, and the other records containing the user’s high scores will be initialised to None.

When signing in, the username is checked against existing usernames in the database, and if it does not match any the user should be alerted. The entered password is then hashed using the same algorithm, and if the two hashed passwords match the user is deemed to have logged in to their account.

Here is a data flow diagram to illustrate in which sections of the program data is communicated with the database:

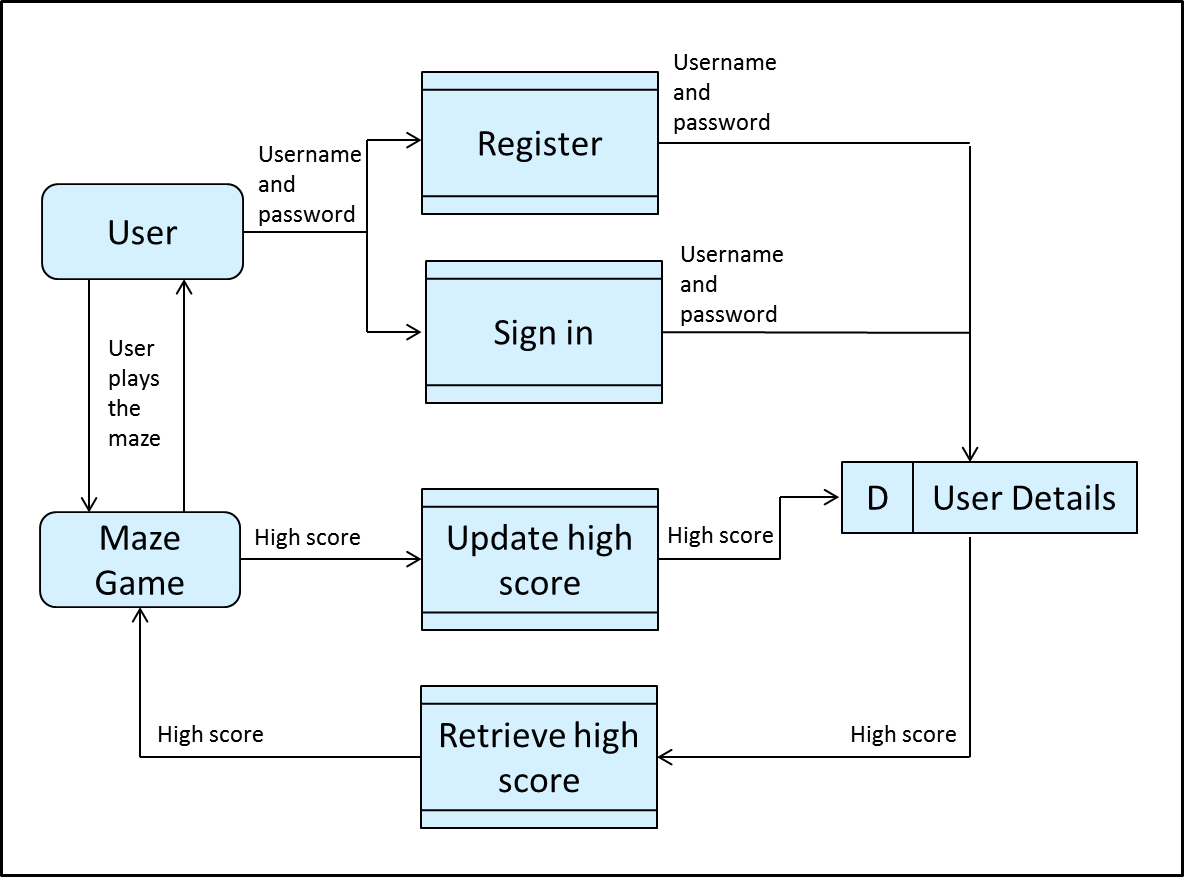


Figure : A Data Flow Diagram illustrating when data is passed to and from the database.

The database is accessed in four places throughout the program: when the user registers or signs in during User Account Management; and twice during Maze Game when the high score is retrieved or updated.

### 2.2.3 SQL

To communicate with the database, I will need to use Structured Query Language (SQL), which I will import into a Python program using the mysql.connector module. A few examples of statements I will use include:

* "CREATE DATABASE user\_accounts" to create the database.
* "CREATE TABLE user\_details (userID INT AUTO\_INCREMENT PRIMARY KEY, username VARCHAR(255), password VARCHAR(255), easy\_high\_score VARCHAR(5), medium\_high\_score VARCHAR(5), hard\_high\_score VARCHAR(5))" to create the table. This initialises the maximum number of characters in each field and causes the userID to be incremented for each new registered user.
* "SELECT password FROM user\_details WHERE username = *‘username’*" to retrieve the stored password.
* "DELETE FROM User\_Details WHERE username = '*username*'" to delete a record from the table.

I will also likely use statements to show the data in the table when I am developing the login system, though they will not be used in the program itself.

## 2.3 Menu

The menu acts as a hub from which the user can select a mode or return to the User Account Management section. Figure 15presents a simple design for the menu:

## 2.4 Maze Game

Figure : A design for the Menu.

### 2.4.1 Maze Game Interface Design

Upon choosing the maze game, the user will be asked which difficulty they would like to attempt:

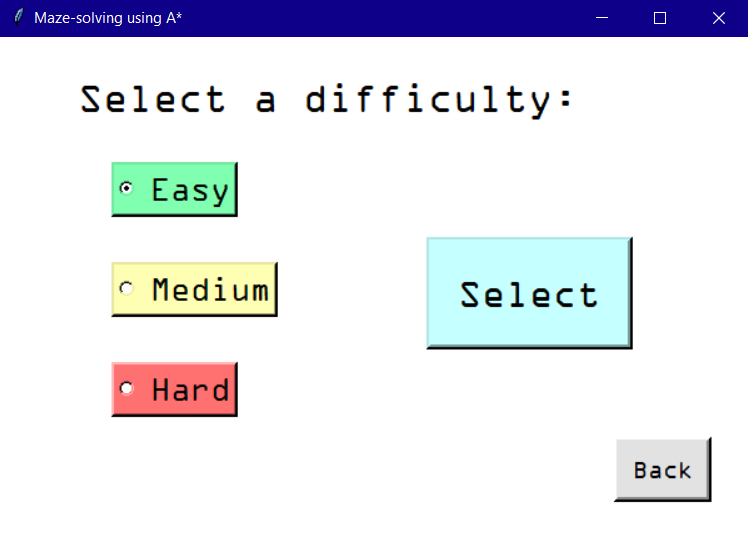
They will then be sent to the main page for this section. A maze will be generated with the entrance in the top left corner and exit in the bottom right corner (indicated by different colour squares). The sprite will begin in the entrance cell. When the user moves the sprite, the cells it has traversed should change colour to reflect its path. Outside of the maze, there is a stopwatch, instructions to use the arrow keys, and buttons allowing the user to create a new maze, change difficulty level, see the answer, customise the theme or return to the menu.

Figure : A design for the Maze Game difficulty selection page.

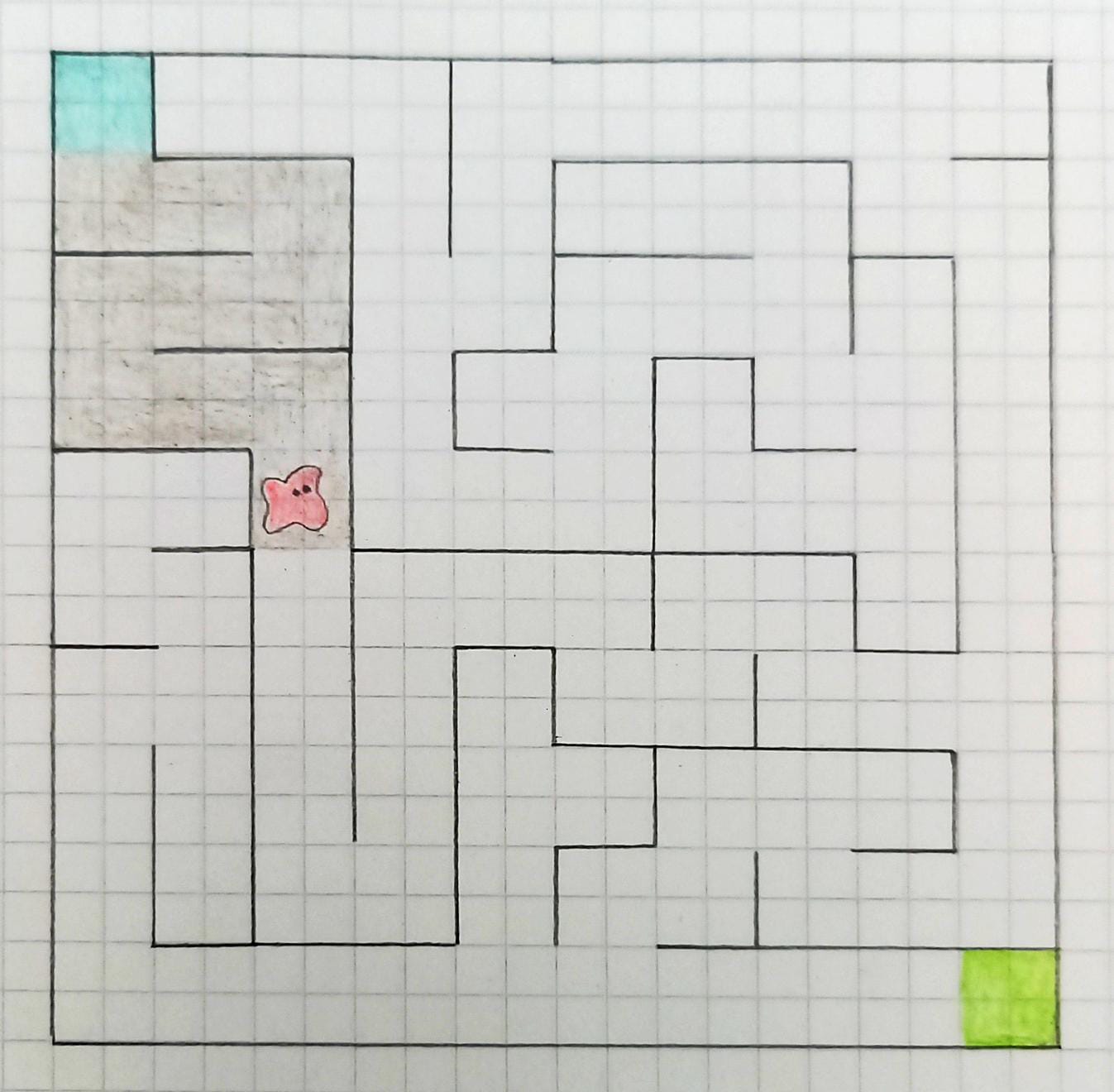


Figure : A sketch of the maze from the Maze Game main page.

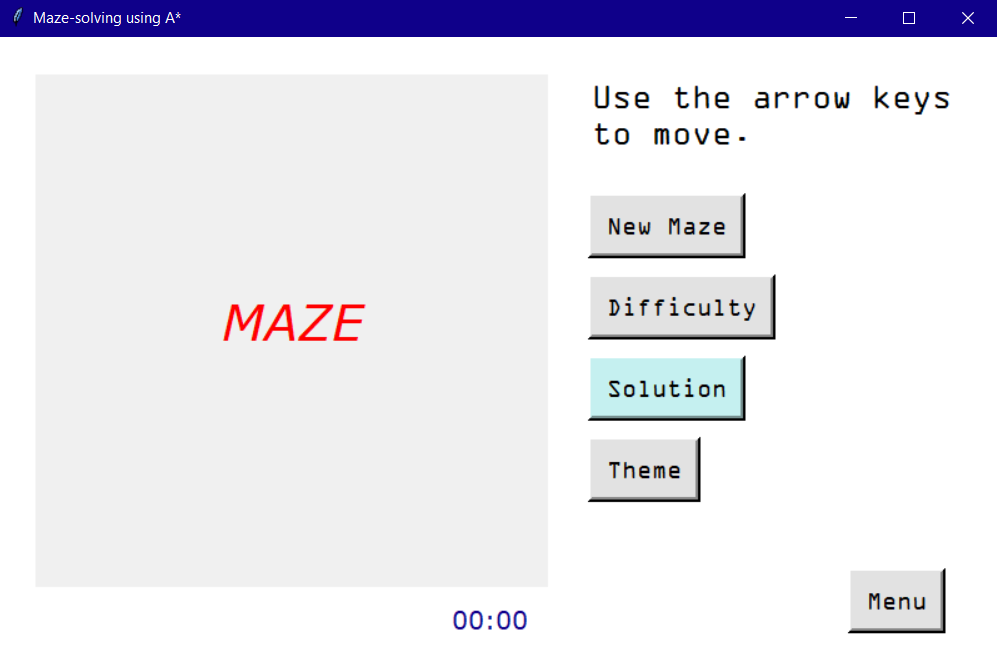
Once the user completes the maze, a page will be produced congratulating them and displaying their score and current high score:

Figure : A design for the main Maze Game page.

The user will be able to select a few colour themes for this section, which change the colour of the maze walls and background, overall page background, buttons and the sprite’s trail. The congratulations page will also change. My ideas for themes include:

Figure : A design of the Maze Game congratulations page.

* Simple black and white theme with a grey trail (this is the default)
* Light blue theme with darker blue walls and a white trail
* Green theme using light green for the trail and dark green for the walls
* Pink theme with a deep pink background, light pink maze background, red walls and a purple trail
* Neon theme with a black background, grey maze background, white walls, and a trail that turns each cell a different colour: bright blue, pink, yellow and purple

The first four styles will all be similarly coded, just changing the colour value for certain sections. The neon theme will be hardest to code as I will have to predetermine which colour each cell will turn, or use an algorithm that ensures the same colour does not appear next to itself.

### 2.4.2 Randomised Depth-First Search

In order to generate the maze, I will use the randomised depth-first search algorithm described in Section [1.3.2](#_1.3.2_Key_Algorithms). Firstly, I need to model the empty grid as a graph, with the nodes representing cells and the edges representing connections between cells (i.e. not a wall). I will use an adjacency list to implement the graph: I will create a list of all the nodes in the grid, and each node can be used to create a list of nodes that can be traversed to. For example, the top left cell (0,0) points to cells (0,1) and (1,0). I will create a class called Cell (see Section [2.7.1](#_2.6.1_The_Cell)) and create instances of it for each cell in the grid, with a method to list each of its neighbours.

I can then use the randomised depth-first search on this graph in order to create a maze. Here is a representation of the algorithm using pseudocode:

SUBROUTINE depthFirstRecursion(current:Cell, allCells:[Cell…\*], cellsList:[(int, int)…\*], totalPassages:[str…\*])

//Generates the maze (by creating a list of walkable passages between cells)

current.setVisited(TRUE)

neighbours 🡨 current.getNeighbours()

SHUFFLE neighbours //so they are selected in a random order

FOR neighbour IN neighbours

position 🡨 cellsList.POSITION(neighbour)

neighbour 🡨 allCells[position] //find the desired neighbour from the allCells list of Cell objects

IF neighbourVisited = FALSE THEN

//add this passage to the list of passages for each cell

currentxy 🡨 current.getxy()

neighbourxy  neighbour.getxy()

passage  (currentxy, neighbourxy)

currentPassages  current.getPassages()

neighbourPassages  neighbour.getPassages()

currentPassages.APPEND(passage)

neighbourPassages.APPEND(passage)

totalPassages.APPEND(passage)

//invoke the routine with the neighbour cell

totalPassages  depthFirstRecursion(neighbour, allCells, cellsList, totalPassages)

ENDIF

ENDFOR

RETURN totalPassages

ENDSUBROUTINE

Note that *allCells* is a list of all the cells as Cell object instances, and *cellsList* is a list of cells in coordinate form: (x, y). The *getNeighbours()* method returns a list of neighbours in coordinate form, and so they must be found within the *allCells* list before they can be used. This prevents different instances with the same attributes from being created.

The subroutine is initially passed the cell with coordinates (0,0). It marks this cell as visited, making it the only cell that has currently been visited. So whichever neighbouring cell it picks, (0,1) or (1,0), will be visited next. If it were to then pick a cell that has been visited, it simply picks again. When the subroutine lands on a cell where all of the cell’s neighbours have been visited, it then backs out of the recursion until the last cell it visited that still has unvisited neighbours. It then explores in that direction. In the worst possible case, this could lead to a depth of recursion of **n**(the number of cells in the grid) if it never had to backtrack (e.g. by making a perfect spiral). However, this is very unlikely to be the case.

### 2.4.3 Playing the Game

The sprite will begin in the top-left cell, and must follow the graph of passages to the bottom-right cell. I will need to handle the inputs from the user, so that when they press an arrow key, the sprite should move in the specified direction if there is a passage there. The Tkinter package allows me to bind certain keypresses to a specific function, so when the user presses an arrow key I can map this to a subroutine which checks whether the sprite can move in that direction.

The program will keep track of the sprite’s current cell, and use this cell’s *getPassages()* method to check whether the specified neighbouring cell is traversable. If so, I can then use another subroutine to define the actual movement of the sprite in the given direction, allowing me to adjust the trail in this subroutine.

When the sprite reaches the end cell, the stopwatch should be paused and the congratulations page should be produced. If the user’s time is lower than their previous time (or no time has yet been recorded for this difficulty), it should be updated in the database. I will do this using the SQL statement:

“UPDATE user\_details SET ‘*difficulty\_high\_score’* = ‘*score’* WHERE username = ‘*username’”*.

### 2.4.4 Solving the Maze

If the user selects the Solution button, the A\* algorithm (see Section [2.5.2](#_2.5.2_The_A*)) will be used on the graph in order to find the solution to the maze. For each cell that is a part of this solution, the cell’s colour should change in order to illustrate the shortest route.

## 2.5 A\* Simulation

### 2.5.1 A\* Simulation Interface Design

When the A\* Simulation section is chosen, the user will be met with this page:

‘GRID’ will simply contain an empty grid to begin with. The user can choose to add objects (see Section [2.5.3](#_2.5.3_Objects)) to the grid, including a start and end cell. Then, when they select the ‘Run A\*’ button, the program will cycle through the A\* algorithm from start cell to end cell. I have included a design for this stage in Section [2.5.4](#_2.5.4_Visualising_the).

Figure : A design of the main A\* Simulation page.

If the user selects the ‘What’s going on?’ button, a page will be brought up with more information about the A\* algorithm and how to use this section to interact with it:

### 2.5.2 The A\* Algorithm

Figure : A design for the A\* Simulation information page.

This A\* instance will differ slightly from the one used in the Maze Game section, as it must allow for obstacles and the start and end cells may be anywhere on the grid. However, as the cells used in the Maze Game section are basic instances of the Cell class, by including selection statements to check the cell’s type, I can use the same algorithm for both sections (see Section [2.5.3](#_2.5.3_Objects)).

The cost of each cell is calculated using the formula ***f(n) = g(n) + h(n)*** discussed in Section [1.3.2](#_1.3.2_Key_Algorithms). Retrieving the cost from the start cell is simple: just add 1 to the cost from the start for the previous cell. This will require two dictionaries: one to list each cell’s cost from the start, and the other to list each cell’s previous cell – which is conveniently also required to reconstruct the shortest path. However, the heuristic is more complex. In order to guarantee the shortest path, it must not overestimate the distance to the end node. However, the closer ***h(n)*** is to 0, the more the A\* algorithm will explore in unpromising directions, turning back into Dijkstra’s algorithm as the heuristic approaches 0. So the closer the heuristic is to the actual distance to the end node, the faster A\* will run.

To calculate the heuristic, I will use *Manhattan Distance*. There were a few options for different heuristics I could choose, however Manhattan Distance is the simplest for a square grid without diagonal movement, as both sections of my program will use. Here is the formula for Manhattan Distance:

**ǀ*x0 – x1*ǀ+ ǀ*y0 – y1*ǀ**

where **x0** is the x-coordinate of the start cell, **x1** is the x-coordinate of the end cell, **y0** is the y-coordinate of the start cell and **y1** is the y-coordinate of the end cell. In words, find the distance between the start and target cell’s x and y coordinates, convert both to a positive integer and add them together.

Here is the A\* algorithm in pseudocode:

SUBROUTINE heuristic(startxy:(int, int), endxy:(int, int))

RETURN ǀstartxy[0] – endxy[0]ǀ + ǀstartxy[1] – endxy[1]ǀ

ENDSUBROUTINE

SUBROUTINE calculateRoute(currentxy:(int, int), cameFrom:{(int, int): (int, int)…\*})

//reproduce the shortest route

solution 🡨 [currentxy]

WHILE cameFrom[currentxy] <> NONE

solution.APPEND(cameFrom[currentxy])

currentxy 🡨 cameFrom[currentxy]

ENDWHILE

solution.REVERSE()

RETURN solution

ENDSUBROUTINE

SUBROUTINE AStar(size:int, start:Cell, target:Cell, allWalls:[((int, int), (int, int)…\*])

//Use the A\* algorithm to find the shortest route to the target cell

frontier 🡨 PRIORITYQUEUE() //I can import this function from the queue library in Python

frontier.PUT(start, 1)

totalCosts  {}

costsFromStart  {}

cameFrom  {}

cameFrom[start.getxy()]  NONE

current  frontier.GET()

currentxy  current.getxy()

targetxy  target.getxy()

WHILE currentxy <> targetxy

neighbours  current.getNeighbours()

FOR checkingNeighbour IN neighbours

neighbourxy  checkingNeighbour.getxy()

IF neighbourxy NOT IN cameFrom THEN

Calculate possible wall between these two cells

IF possibleWall NOT IN allWalls THEN

IF currentxy = startxy THEN

costSoFar  1 //the start cell has no previous cell

ELSE

costSoFar  costsFromStart[currentxy] + 1

ENDIF

costToNeighbour  costSoFar + heuristic(neighbourxy, targetxy)

IF neighbourxy NOT IN totalCosts OR costToNeighbour < totalCosts[neighbourxy] THEN

//if this cell has already been checked, only update its cost if the new cost is smaller than its previous cost.

totalCosts[neighbourxy]  costToNeighbour

costsFromStart[neighbourxy]  costSoFar

frontier.PUT((costToNeighbour, checkingNeighbour)) //use the cell’s cost as its priority

cameFrom[neighbourxy]  currentxy

ENDIF

ENDIF

ENDIF

ENDFOR

current  frontier.GET()

ENDWHILE

RETURN calculateRoute(targetxy, cameFrom)

ENDSUBROUTINE

The frontier uses the actual Cell objects (current, target), while all other dictionaries represent them using their coordinates. This is again to prevent multiple instances of the same cell from being produced (in this case, the WHILE loop would never evaluate to TRUE and so would continue infinitely).

### 2.5.3 Objects

The user can choose to add a selection of objects to the graph. These are:

* Start Cell
* End Cell
* Wall
* Puddle
* Warp Entrance
* Warp Exit

Each will be a subclass of the Cell class (see Section [2.7.1](#_2.7.1_The_Cell)) with a unique attribute *type*. Start and end cells will have this attribute set to ‘start’ or ‘end’. When the user selects the ‘Run A\*’ button, the program should locate these cells in the grid and return their coordinates, which can then be passed into the A\* subroutine. If either or both cells have not been placed, the user should be prompted to add them and the algorithm should not run.

Wall cells will have the type attribute set to ‘wall’. When they are created, each of the cell’s edges should be added to the list of all walls: in Maze Game this is also how walls are represented, allowing me to use the A\* subroutine in both sections.

Puddles are cells that will have a different weight to cross than an empty cell (empty cells have a weight of 1, and puddles have a weight of 5). This means that the total cost of travelling through this cell is 5, so the shortest route may go around it rather than passing over it. This can be achieved by adjusting the cost calculation to check the neighbour cell’s *cost* attribute:

costSoFar  costsFromStart[currentxy] + checkingNeighbour.getCost()

Warp panels will require two different subclasses, WarpIn and WarpOut. For simplicity, I will limit the user to only having one of each on their grid. When the WarpIn cell is checked, instead of checking its own neighbours, it should check the neighbours of the WarpOut cell, adding those to the frontier. This should be irreversible: the WarpOut cell will be treated as an empty cell if crossed normally.

Figure 22 presents some designs for each object:

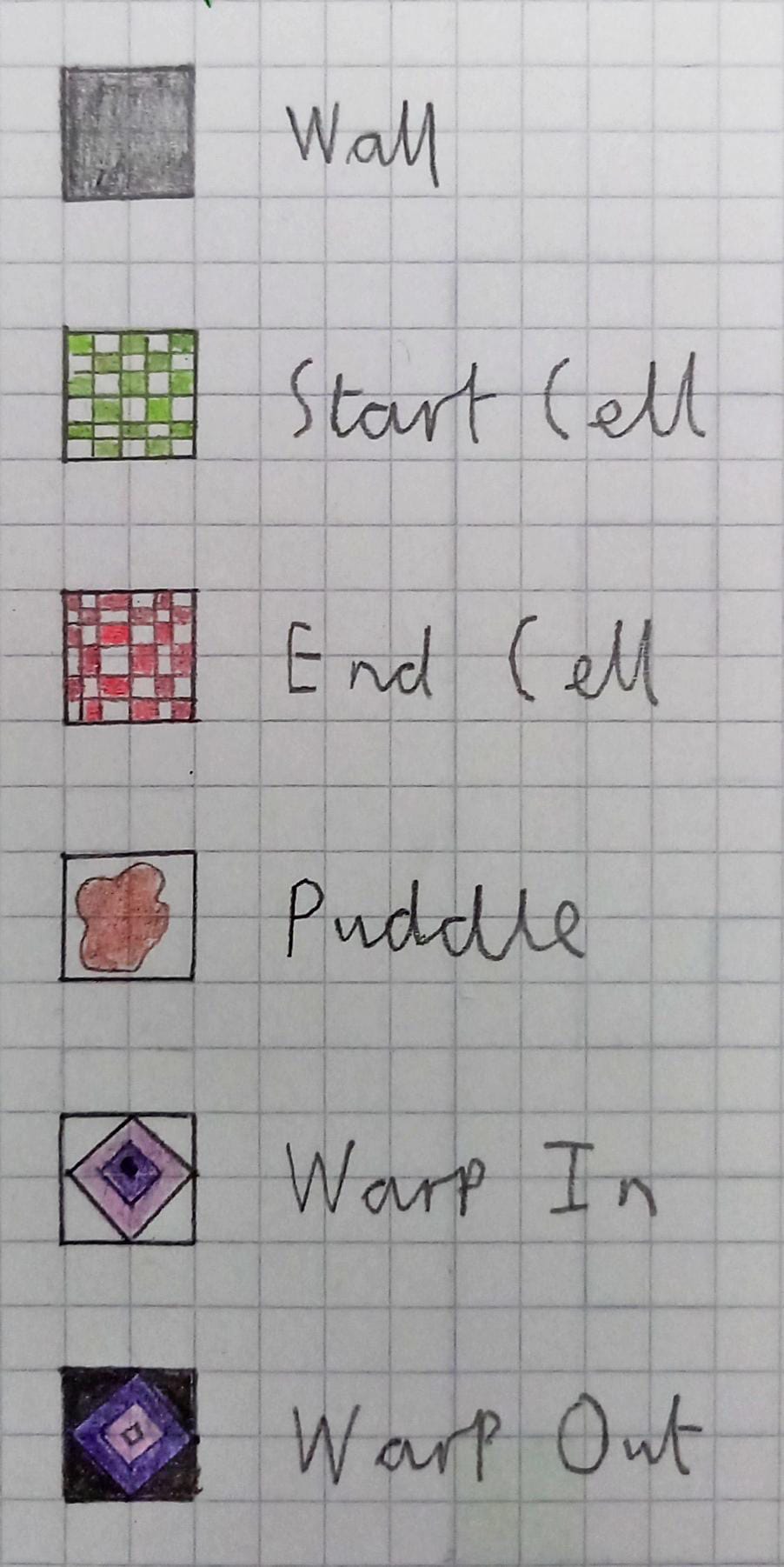


Figure : A sketch of the objects used in A\* Simulation.

### 2.5.4 Visualising the Algorithm

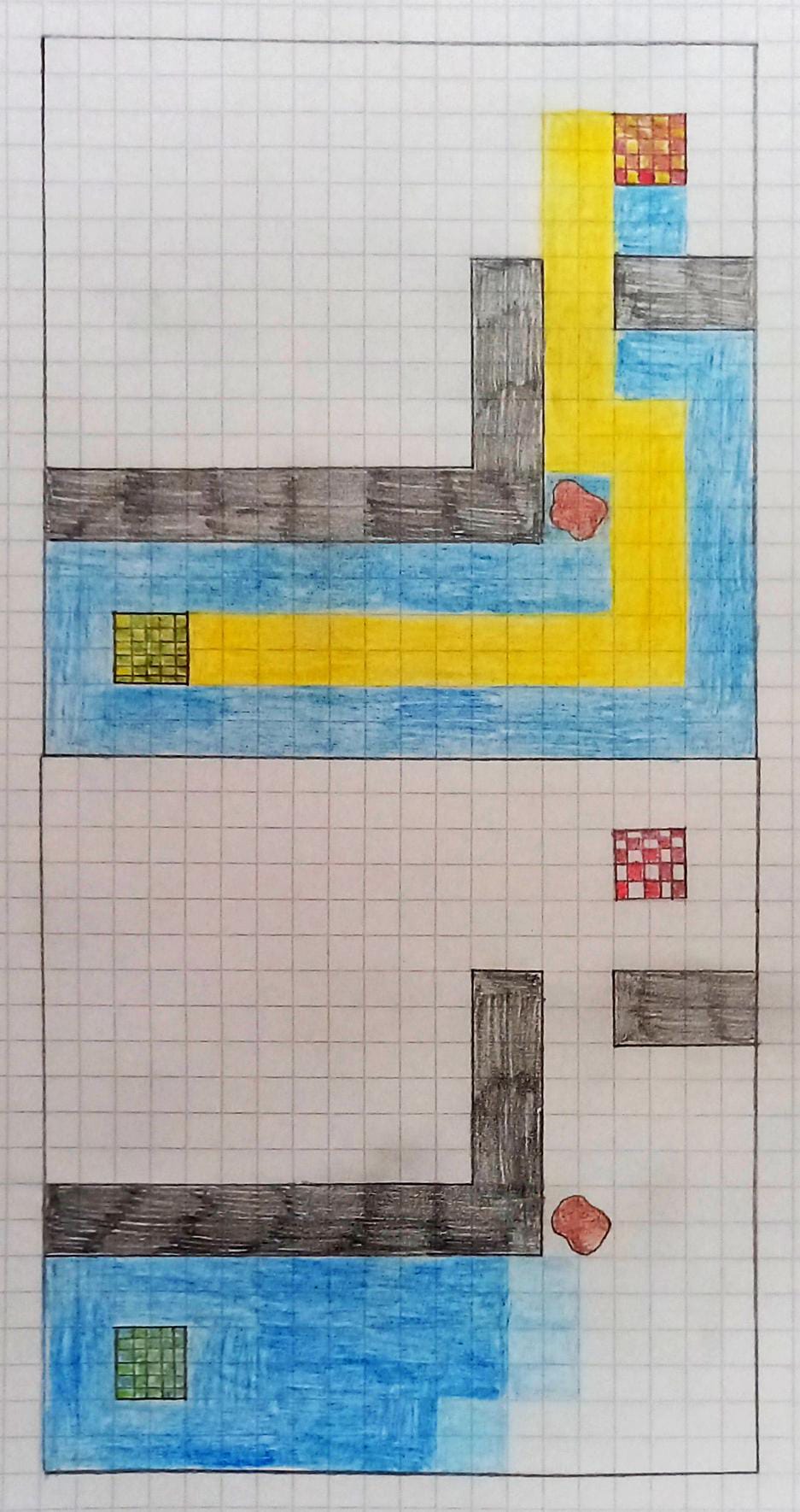
As discussed earlier, visualisation is a key part of this software, as it will allow the user to see the A\* algorithm in action and gain a better understanding of how it works. Initially, all empty cells in the grid will be white. When the user selects the ‘Run A\*’ button, the algorithm will be stepped through with a break in between, allowing the user to watch it find the solution. The PathViz site I looked at in Section [1.3.2](#_1.3.2_Key_Algorithms) used a different colour for the current cell, however I found this jumped around the grid too rapidly to follow. I will however, use a light blue to show the cells in the frontier, as this is easier to comprehend and doesn’t change as quickly. Cells that have already been visited will be a darker blue. Finally, once the shortest path has been found it should be highlighted in a contrasting yellow. Figure 23 presents my designs for this stage:

Figure : A sketch of the A\* Simulation running (left) and complete (right).

## 2.6 User Feedback

I presented my designs of the user interface to Noah so that they could give me feedback on the features they liked, and ideas I could work on integrating into the design. Similarly to before, the meeting took place in the form of an informal interview, where I showed them each design and then asked related questions. I also discussed my ideas for themes for the Maze Game section during this process. Here is their feedback:

**Do the designs make it clear how to access each section? Is it easy to navigate?**

‘I found the designs simple and intuitive to navigate. The different colours really help me differentiate between each option. If I could suggest one change it would be to make the Maze Game and A\* Simulation buttons two different colours so they stand out from each other.’

**Do you have any comments on the general design features such as font and colours, or the different themes I have suggested?**

‘I like the font choice; it really gives the program some character. The default theme shown in the designs works well, being colourful enough not to seem monotonous. The other themes suggested for the Maze Game section all sound plausible and I am excited to see how this feature turns out.’

**What were your thoughts on the maze sketch?**

‘I thought that the simple character worked really well. Perhaps adding a flag at the end or writing *start* and *end* in those positions would help the user to understand where they are supposed to go.’

**What were your thoughts on the A\* Simulation sketches?**

‘The warp in and warp out objects could use some labels so that it is clear which is which. I like the different colours used during the visualisation but I think that mentioning them in the information page would help the user to understand what they represent.’

**Does the information page explain the A\* algorithm well? Did you understand everything?**

‘I understood the explanation well and think it went into a good amount of detail. An improvement would be to add some colours to this section so it is less dull.’

**Do you have any other comments?**

‘The maze itself would be best without including grid markings, however I do think these are necessary for the A\* Simulation section. Also, including a play as guestoption would be beneficial, so the user doesn’t have to go through the hassle of signing in every time they use the system.’

This feedback was really useful, allowing me to refine my design in order to meet the needs of my end user. The features I will edit or add to my design as a result of this interview are:

* Make the buttons in the Menu section different colours to make them more unique.
* Add details to the maze to make it clearer where the start and end are (perhaps by using the start and end cell images from A\* Simulation).
* Add labels to the objects in A\* Simulation so that it is clear what each object is.
* Add a description of how the colours in the A\* Simulation are used to represent the frontier, visited cells and shortest path.
* Add some colouration to the A\* Simulation information page.
* Possibly include a play as guest option so users don’t have to sign in.

## 2.7 Object Oriented Programming design

### 2.7.1 The Cell class

My project makes use of object oriented principles in multiple areas. First, a class called Cell will be used to model the empty cells in the grid in both Maze Game and A\* Simulation. Each cell will have the following attributes and methods.

|  |
| --- |
| Cell |
| xy: (int, int)  visited : Boolean  neighbours : [(int, int)…\*]  edges : [((int, int), (int, int))…\*]  passages : [str … \*]  colour : str  type : str  cost : int |
| getxy()  getVisited()  setVisited(value)  getNeighbours()  getEdges()  getPassages()  getColour()  setColour(value)  getType()  getCost()  calculateNeighbours(height, width)  calculateEdges() |

Figure : The attributes and methods of the Cell class.

* *xy* is the cell’s coordinates, starting with (0,0) as the top left cell.
* *visited* is a flag used in the randomised depth-first search algorithm, False by default.
* *neighbours* is a list of the cell’s neighbouring cells; for example cell (3, 5) has neighbours (2, 5), (4, 5), (3, 4) and (3, 6).
* *edges* is a list of the cell’s four edges, in the form ((start coordinates), (end coordinates)). For example, cell (3, 5) has its top edge at ((2.5, 4.5), (3.5, 4.5)).
* *passages* is a list of traversable passages from this cell (for example, you may be able to move up but not down due to a wall being in the way). This is used for collision detection in Maze Game.
* *colour* is the cell’s RGB colour value, depending on the current theme.
* *type* is the cell’s type, which is ‘empty’ for Cell objects.
* *cost* is the cell’s cost to cross, which is 1 for all cell types except puddles.
* *calculateNeighbours(height, width)* determines the cell’s neighbours, using the height and width of the grid to check whether a neighbour falls within the bounds of the grid.
* *calculateEdges()* similarly calculates the cell’s edges.

The objects in A\* Simulation each inherit from the Cell class. The below class diagram highlights which attributes are unique to these subclasses.

|  |
| --- |
| StartCell/EndCell/Wall/Puddle/WarpIn/WarpOut |
| colour : Image  type : str  cost : int |

Figure : The attributes of each of Cell's subclasses.

* *colour* is now an Image object from the Python Imaging Library, using a design I have created instead of a flat colour. The only exception is Wall which remains a flat colour, black.
* *type* is a string unique to each subclass such as ‘start’.
* *cost* is set to 5 for a Puddle cell, but remains 1 for all other cells.

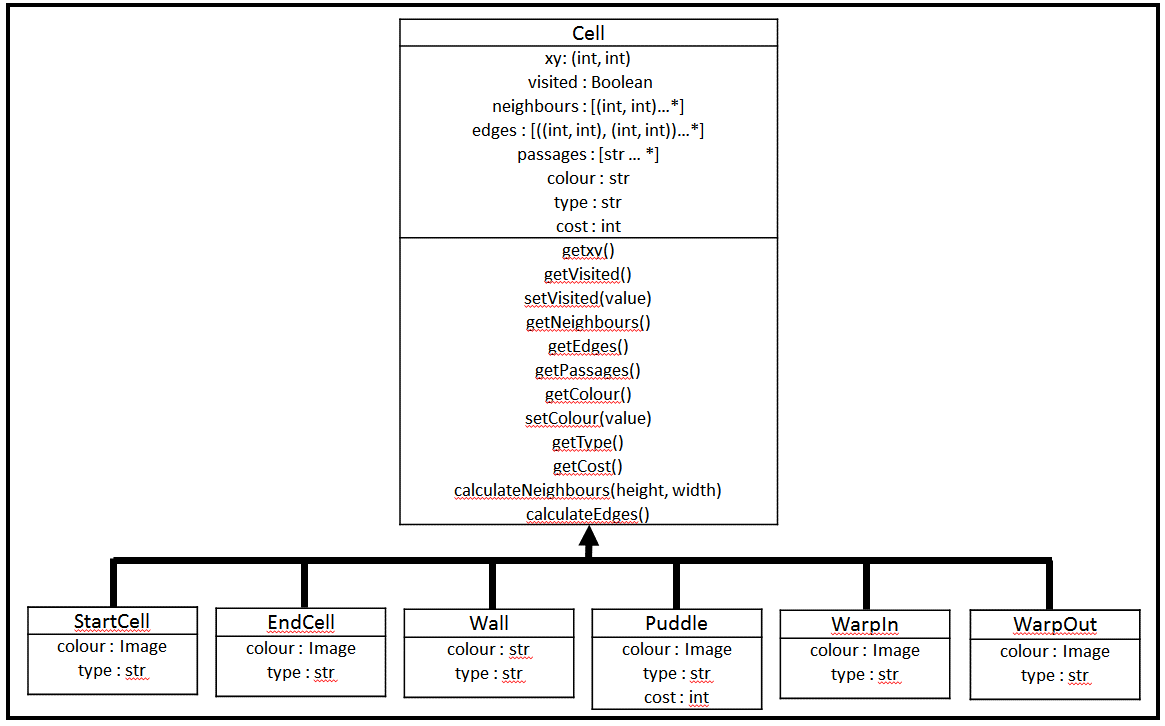
Here is a class diagram illustrating this inheritance:

Figure : Class diagram containing Cell and its subclasses.

### 2.7.2 The Sprite class

Another class in my program is the Sprite from Maze Game (the character that is controlled by the player). It will require the following attributes and methods:

|  |
| --- |
| Sprite |
| xy: (int, int)  cell : Cell |
| getxy()  setxy(value)  getCell(cellsList, allCells)  canMove(direction) |

Figure : The attributes and methods of the Sprite class.

* *xy* is the sprite’s current coordinates.
* *cell* is the Cell object that the sprite is currently on, according to its coordinates.
* *getCell(cellsList, allCells)* is a getter method, but will also handle converting between coordinate and Cell form using *cellsList and allCells* (see Section [2.4.2](#_2.4.2_Randomised_Depth-First) for an explanation of these lists).
* *canMove(direction)* is used to check whether the sprite can move in the direction pressed by the user, by making use of its current cell’s *passages* attribute to check whether there is a passage there. If so, it will set the sprite’s *xy* attribute to that of the new cell.

### 2.7.3 The Stopwatch Class and Threading

Similarly, I will use a class to implement the stopwatch in Maze Game.

|  |
| --- |
| Stopwatch |
| total : int  seconds : int  minutes : int  display : str |
| getMinutes()  getDisplay()  increment()  updateDisplay()  reset() |

Figure : The attributes and methods of the Stopwatch class.

* *total* is the total number of seconds that have passed since this maze was created.
* *seconds* is the number of seconds that have passed this minute.
* *minutes* is the number of minutes that have passed.
* *display* is the stopwatch’s display (the combination ‘*minutes*:*seconds*’)
* *increment()* is used to add one second, updating *total, seconds* and *minutes* each time.
* *updateDisplay()* uses the values of *seconds* and *minutes* to update *display* accordingly.
* *reset()* resets the stopwatch to ‘00:00’ when a new maze is created.

In order to have the stopwatch running simultaneously with the rest of Maze Game, I will need to make use of threading: a mechanic from Python’s threading module which allows different parts of a program to run at the same time. A thread will be used to manage the stopwatch, with this thread ending and a new thread being created each time the user presses the ‘New Maze’ button. However, Python does not provide a simple method to end a thread, so I will work around this by using a class:

|  |
| --- |
| StopwatchThread |
| running : Boolean  thread : Thread |
| stop()  updateStopwatch(stopwatch)  getThread()  setThread(value) |

Figure : The attributes and methods of the StopwatchThread class.

* *running* is a flag used to stop the thread.
* *thread* is the actual Thread object from the threading module.
* *stop()* sets *running* to False, so the thread can end.
* *updateStopwatch(stopwatch)* is the main subroutine used in the thread, which will increment the stopwatch each second and display the result on screen. It continues to do so until *running* becomes False, the user completes the maze or the stopwatch reaches ’60:00’.

This way, I can create a new instance of StopwatchThread each time a new maze is created, which will include a new instance of the Thread module. I will also have to ensure that the current thread is closed whenever the user exits the main Maze Game window.

## 2.8 File organisation

My solution will be spread across several different programs, in order to encapsulate related modules and allow me to develop and test sections individually. The programs I intend to use are as follows:

* ‘mazeSolvingUsingAStar’ – The program that is run by the user. This will contain the user interfaces of the User Account Management and Menu sections, as these are very simple, and handle navigation between each section of the solution.
* ‘userAccountManagement’ – Used to handle communication with the database so the user can register or log in to their account.
* ‘mazeGame’ – Handles creating the maze, allowing the user to play it and other functions from Maze Game.
* ‘AStarSimulation’ – Allows the user to create their own maze and produces a visualisation of the A\* algorithm, as well as other functions from A\* Simulation.
* ‘randomisedDepthFirstSearch’ – As one of my major algorithms, I have decided to separate this from the rest of Maze Game so that I can test it easily.
* ‘AStarAlgorithm’ – My other major algorithm, which will be accessed by both ‘mazeGame’ and ‘AStarSimulation’, so I will keep it in a separate program to make this easier.
* ‘cellClasses’ – Contains the Cell class and all of its subclasses.
* ‘spriteClass’ – Contains the Sprite class.
* ‘stopwatchClass’ – Contains the Stopwatch class.

The user accounts will be stored on a MySQL database called ‘user\_accounts’.

## 2.9 Key variables

Below is a table detailing all of the major variables, objects and complex data structures I will use in my project (excluding those associated with Tkinter):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Data Type** | **Description** | **Example** | **Location** |
| option | str | Used to determine which part of the solution to navigate to (for example, if they pressed the ‘Back’ button). | ‘menu’ | ‘mazeSolvingUsingAStar’ |
| username | str | The user’s selected username. | ‘eleanor1’ | ‘mazeSolvingUsingAStar’, ‘userAccountManagement’, ‘mazeGame’ |
| password | str | The user’s password (hashed using the SHA-265 algorithm). | ‘dvb5dgf79hf9…’ | ‘mazeSolvingUsingAStar’, ‘userAccountManagement’ |
| db | MySQLConnection object | Used to create a connection to the database. | N/A | ‘userAccountManagement’, ‘mazeGame’ |
| cursor | MySQLCursor object | Used to execute SQL commands to communicate with the database. | N/A | ‘userAccountManagement’, ‘mazeGame’ |
| current | Cell | The cell that is currently being visited. | Cell at (3, 5) | ‘randomisedDepthFirstSearch’, ‘AStarAlgorithm’ |
| neighbour | Cell | The currently selected neighbour cell (of the current cell). | Cell at (3, 6) | ‘randomisedDepthFirstSearch’, ‘AStarAlgorithm’ |
| passage | Tuple | The passage between two cells. | ((3, 5), (3, 6)) | ‘randomisedDepthFirstSearch’ |
| passages | List | A list of all traversable passages within the grid. | [((3, 5), (3, 6)), …] | ‘randomisedDepthFirstSearch’, ‘mazeGame’ |
| allCells | List | A list of all cells within the grid (as Cell objects). As each cell can point to its neighbouring cells this is an adjacency list. | [Cell at (0,0), Cell at (0,1)…] | ‘randomisedDepthFirstSearch’, ‘mazeGame’,  ‘AStarAlgorithm’, ‘AStarSimulation’ |
| cellsList | List | A list of all cells within the grid (as coordinates). | [(0,0), (0,1)…] | ‘randomisedDepthFirstSearch’, ‘mazeGame’,  ‘AStarAlgorithm’, ‘AStarSimulation’ |
| size | int | The size of the grid (number of cells per row). | 5 | ‘mazeGame’ |
| default, blue, green, pink and neon | Dictionary | Each dictionary holds properties of the user interface as keys, and a predetermined colour for this property as values. | {‘walls’ : ‘#000000’…} | ‘mazeGame’ |
| theme | Dictionary | The current theme in use (one of default, blue, green, pink and neon). | {‘walls’ : ‘#000000’…} | ‘mazeGame’ |
| sprite | Sprite | The Sprite object controlled by the user. | N/A | ‘mazeGame’ |
| allWalls | List | A list of all walls within the grid. | [((2.5, 3.5), (2.5, 4.5)), …] | ‘mazeGame’, ‘AStarSimulation’ |
| finished | Boolean | Signals whether the user has completed this maze yet (and therefore whether to bring up the congratulations page). | False | ‘mazeGame’ |
| stopwatch | Stopwatch | The Stopwatch object. | N/A | ‘mazeGame’ |
| thread | Thread | The current Thread object. | N/A | ‘mazeGame’ |
| solution | List | A list of cells in coordinate form that are part of the solution to the maze. | [(0,0), (0,1),…] | ‘mazeGame’, ‘AStarAlgorithm’, ‘AStarSimulation’ |
| frontier | Priority Queue | A priority queue of cells to be checked, with their total cost used as their priority. Uses Python’s built-in queue library. | [(20, Cell at (3,5)), …] | ‘AStarAlgorithm’ |
| totalCosts | Dictionary | Contains each cell that has been checked and its total cost. | {Cell at (2,2) : 25, …] | ‘AStarAlgorithm’ |
| costsFromStart | Dictionary | Contains each cell that has been checked and its distance from the start cell. | {Cell at (7,0) : 7} | ‘AStarAlgorithm’ |
| cameFrom | Dictionary | Contains each cell that has been checked and the cell visited immediately before it. | {Cell at (7,0) : Cell at (6,0), …] | ‘AStarAlgorithm’ |
| possibleWall | Tuple | The edge between the current cell and its neighbour currently being checked. | ((2.5, 4.5), (3.5, 4.5)) | ‘AStarAlgorithm’ |
| costSoFar | int | The cost to travel from the start cell to the current cell. | 15 | ‘AStarAlgorithm’ |
| costToNeighbour | int | The cost to a cell: the sum of the distance from the start cell and the heuristic distance to the end cell. | 23 | ‘AStarAlgorithm’ |
| height and width | int | The height and width of the grid (replacing the size variable from ‘mazeGame’). | 10, 24 | ‘AStarSimulation’ |
| currentObject | str | The type of item that the user has selected to be added to the grid when they click a cell. | ‘wall’ | ‘AStarSimulation’ |
| editGrid | Boolean | Used to prevent the user from making changes to the grid while the algorithm is running. | False | ‘AStarSimulation’ |

# 3 Technical solution

## 3.1 Code contents

The full listing of my code is situated in [Appendix B](#_7.2_Appendix_B). The program consists of 9 files as described in Section [2.8](#_2.8_File_organisation). Below is a contents table containing notable locations or features of my program and their initial page numbers.

|  |  |  |
| --- | --- | --- |
| **Location or feature** | **Description** | **Page Number** |
| ‘cellClasses’ | The definition of the Cell class and its subclasses. | 109 |
| Adjacency list pointers | The method *calculateNeighbours()* allows cells to produce a list of their neighbouring cells (the list *allCells* is therefore an adjacency list). | 111 |
| Inheritance | The object subclasses each inherit attributes and methods from the Cell class. | 111 |
| ‘spriteClass’ | The definition of the Sprite class. | 113 |
| ‘stopwatchClass’ | The definition of the Stopwatch class. | 114 |
| ‘randomisedDepthFirstSearch’ | Program in which a maze is generated. | 115 |
| Depth-first search | Used to traverse the graph representing the grid, allowing a maze to be created on it. | 115 |
| Recursion | Used as part of the depth-first search algorithm. | 116 |
| ‘AStarAlgorithm’ | Program where the A\* Algorithm is implemented. | 117 |
| Manhattan Distance heuristic | Estimates the distance from a particular cell to the target cell. | 117 |
| Priority queue manipulation | *frontier* is a priority queue used within the algorithm to store cells to be checked next. Various lists and dictionaries are also used. | 117 |
| ‘mazeSolvingUsingAStar’ | Main program used for simple Tkinter output and project navigation. | 120 |
| Exception handling | I used exception handling in various places in my program to deal with missing imports, exceptions caused when the program is terminated etc. | 120 |
| ‘userAccountManagement’ | Program which handles communication with the database so users can register or sign in to an account. | 127 |
| Database access using SQL 1 | SQL is used to query the database, retrieving or setting usernames and passwords. | 127 |
| SHA-256 password hashing | The SHA-256 algorithm hashes the user’s password so they cannot be seen by an infiltrator. It is also used when signing in to check that the given password matches the one in the database. | 127 |
| ‘mazeGame’ | Program that allows the user to play mazes of different difficulties and keep track of their time taken. | 128 |
| MazeClass definition | Tkinter buttons can have variables passed into them, but not returned. This makes it difficult to manage variables that are required throughout different subroutines. MazeClass is a simple class implementing each variable as an attribute, so that I can pass a MazeClass object into each subroutine, and then access and modify whichever variables I need to. | 128 |
| stopwatchThread class definition | This class is used to handle multithreading, allowing the stopwatch to increment while the main program is running. Ordinarily Python does not provide a method to stop a thread, so in this class I have created the *stop()* method, which allows me to end the current thread when a new maze is created. | 133 |
| Database access using SQL 2 | SQL is used to query the database, retrieving or setting high scores. | 138 |
| ‘AStarSimulation’ | Program that allows the user to create a customised grid for the A\* algorithm to navigate. | 148 |
| AStarClass definition | A class which functions the same as MazeClass, allowing me to pass variables into and out of subroutines. | 149 |
| Linear search | Used to check if a certain cell type is present within *allCells*. | 158 |

For more information, refer to the Design section or comments within the code itself.

## 3.2 Achieved objectives

The below table contains evidence of where my objectives have been met within my code. See Section [4.4](#_4.4_Testing_against) for a guide to visual evidence of these objectives being met.

|  |  |  |
| --- | --- | --- |
| **Objective** | **Subroutine** | **Page Number** |
| 1. Upon starting, the program will prompt the user to register or log in to their account. | mazeSolvingUsingAStar.displayLogin() | 120 |
| 1.1. They will have to register a username and password. | N/A | N/A |
| 1.1.1. All usernames are stored in a database. This username must be checked against those in the database to make sure it is unique. This database will be accessed using SQL. | userAccountManagement.register() | 122 |
| 1.1.2. If the username is taken the user should be alerted. | mazeSolvingUsingAStar.register() | 122 |
| 1.1.3. The password will also be stored in the database. It should be encrypted using a hashing algorithm instead of being stored simply as plaintext. | userAccountManagement.register() | 122 |
| 1.2. Once the account has been successfully created (or a user with an already existing account has logged in) they will be directed to the main menu. | mazeSolvingUsingAStar.navigate() | 126 |
| 2. The main menu will allow the user to select which mode they wish to play. | mazeSolvingUsingAStar.displayMenu() | 124 |
| 2.1. There will be a button for the maze game and a button for the A\* simulator. | mazeSolvingUsingAStar.displayMenu() | 124 |
| 2.2. They will also be able to sign out from this screen, returning to the first screen. | mazeSolvingUsingAStar.displayMenu() | 124 |
| 3. In the maze section: | N/A | N/A |
| 3.1. The user will select a difficulty (easy, medium or hard). | mazeSolvingUsingAStar.mazeDifficulty() | 125 |
| 3.2. A maze will be generated using the randomized depth-first search algorithm, its size depending on the difficulty selected in objective 3.1. | randomisedDepthFirstSearch.depthFirstRecursion()  mazeGame.createMaze() | 115  144 |
| 3.3. The user can then play the maze, using the arrow keys to navigate a simple sprite from the start cell to the target cell. | spriteClass.sprite.canMove()  mazeGame.moveSprite() | 113  145 |
| 3.3.1. Instructions to use the arrow keys to play should be clearly displayed. | mazeGame.main() | 146 |
| 3.3.2. When the sprite moves a trail should be displayed showing where it has been. | mazeGame.editCell() | 142 |
| 3.3.3. There will be a stopwatch displayed on this screen so the user can see the current time taken. | stopwatchClass.\_updateDisplay()  mazeGame.stopwatchThread.\_updateStopwatch() | 115  134 |
| 3.4. When the user completes the maze, a congratulations message will be produced, which states the time taken and the user’s current high score. | mazeGame.findHighScore()  mazeGame.congratulations() | 137  139 |
| 3.4.1. The user’s fastest time (on this difficulty) is taken as their highest score and saved to the database. | mazeGame.findHighScore() | 137 |
| 3.4.2. Should the user beat their high score, this will be updated in the database and the new score will be displayed the next time they beat a maze of the same difficulty. | mazeGame.findHighScore() | 137 |
| 3.5. There will be an option to see the answer. When clicked the A\* algorithm will be used and the shortest route through the maze will be displayed. | AStarAlgorithm.AStar()  mazeGame.solveMaze() | 117  137 |
| 3.5.1. No time will be recorded for this attempt. Instead the user must click the play again button (see objective 3.6) to start a new maze. | mazeGame.main() | 146 |
| 3.6. There will also be options to play again, change difficulty level or return to the menu. | mazeGame.main() | 146 |
| 3.7. There will be a customisation option which allows the user to select a colour theme for the maze. | mazeGame.themePage()  mazeGame.changeTheme() | 135  136 |
| 4. In the A\* section: | N/A | N/A |
| 4.1. An empty grid will be displayed. | AStarSimulation.drawCell()  AStarSimulation.main() | 155  161 |
| 4.2. At the bottom of the screen the user can select what to place with their mouse: | AStarSimulation.main() | 161 |
| 4.2.1. A solid wall. | cellClasses.Wall  AStarSimulation.main() | 112  161 |
| 4.2.2. A start node (only one). | cellClasses.StartCell  AStarSimulation.main() | 111  161 |
| 4.2.3. An end node (only one). | cellClasses.EndCell  AStarSimulation.main() | 112  161 |
| 4.2.4. An obstacle which takes longer to cross than an empty cell. | cellClasses.Puddle  AStarSimulation.main() | 112  161 |
| 4.2.5. A warp panel that moves the frontier across the grid. | cellClasses.WarpIn  cellClasses.WarpOut  AStarSimulation.main() | 112  112  161 |
| 4.3. Once the user has drawn their maze, they can select the start button. | AStarSimulation.main() | 161 |
| 4.3.1. If a start and end node haven’t been placed the user should be notified. | AStarSimulation.checkGrid() | 158 |
| 4.3.2. The A\* algorithm will then run, changing the colour of each cell it visits. There will be a short delay so that the user can see the changes occurring. | AStarSimulation.wait()  AStarSimulation.runAStar() | 159  159 |
| 4.3.3. When the shortest path has been found it will be highlighted in another colour. | AStarSimulation.runAStar() | 159 |
| 4.4. There should be an option to clear the grid. | AStarSimulation.clearGrid() | 154 |
| 4.5. There should be a button containing information about this section of the software: | AStarSimulation.main() | 161 |
| 4.5.1. How to interact with the area (place objects etc.) | AStarSimulation.infoPage() | 152 |
| 4.5.2. What the A\* algorithm is and does (finds the shortest path from one node to another) | AStarSimulation.infoPage() | 152 |
| 4.5.3. How the algorithm works (explaining how the cost for each cell is calculated etc.) | AStarSimulation.infoPage() | 152 |
| 4.6. There should also be an option to return to the menu. | AStarSimulation.main() | 161 |

# 4 Testing

## 4.1 User Experience based testing

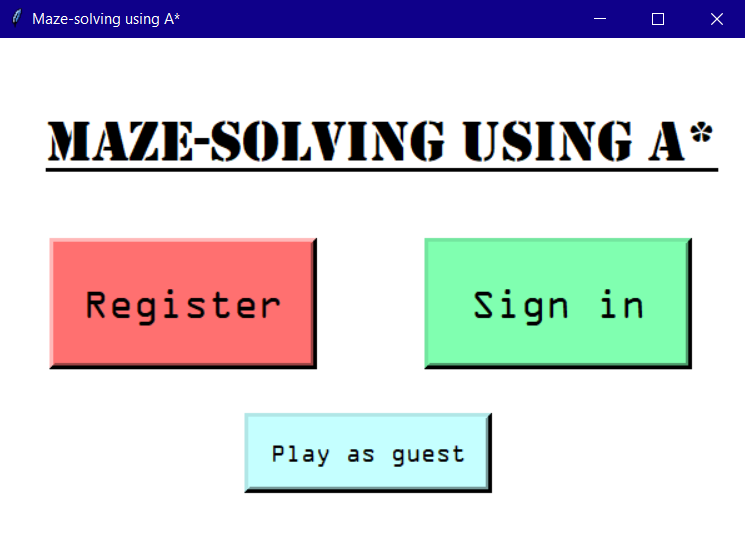
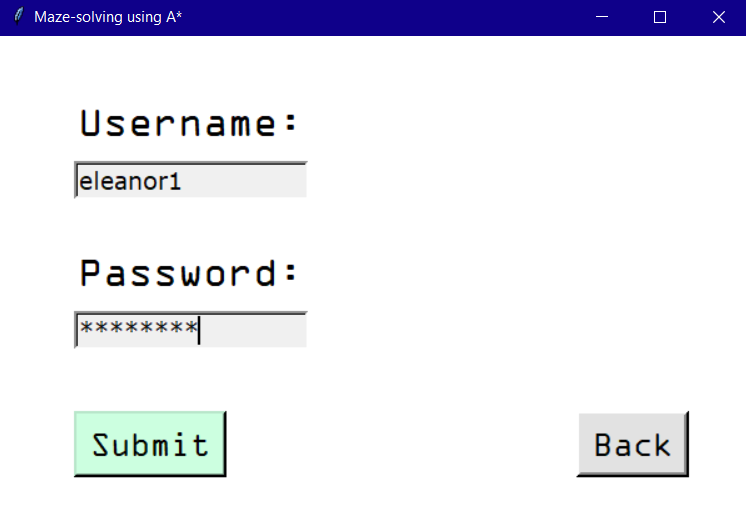
My first set of tests focus on the system as a whole and how a user may experience the program. It also contains some evidence of my final user interfaces; those not present here are displayed in Section [4.3](#_4.3_Extra_user). I created an account called ‘eleanor1’and interacted with the program as I would expect a user might. Below are screenshots of the program at each stage:

Figure : The first page of the program, allowing me to choose to register, sign in or play as guest.

Figure 30 shows the page I was met with when I loaded up the program. I clicked ‘Register’, which took me to this form:

Figure : The form allowing me to register my account.

When I clicked ‘Submit’, this message appeared:

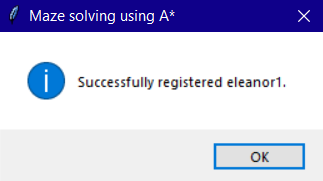


Figure : The message produced when I clicked ‘Submit’.

My account was successfully created. I clicked ‘OK’, and the menu was produced:

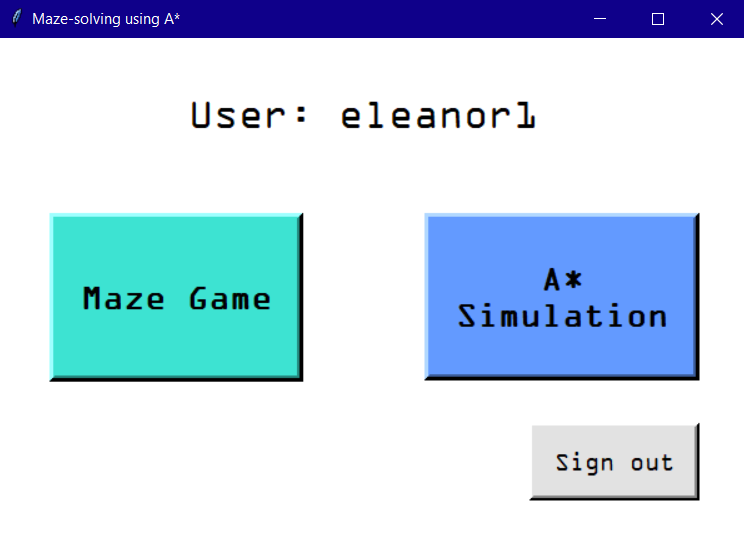


Figure : The menu page, allowing me to pick which option to play. My username is displayed at the top.

First, I clicked on ‘Maze Game’, which took me to the difficulty selection page:

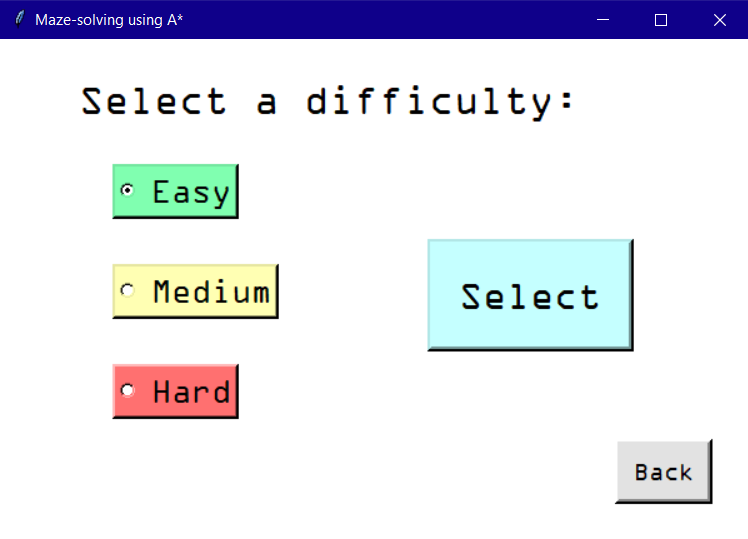


Figure : The difficulty selection page, with Easy selected.

I chose Easy and pressed ‘Select’. The main Maze Game page was produced with a 5\*5 maze, and the stopwatch started instantly. I used the arrow keys to move the sprite (which started on the green start cell):

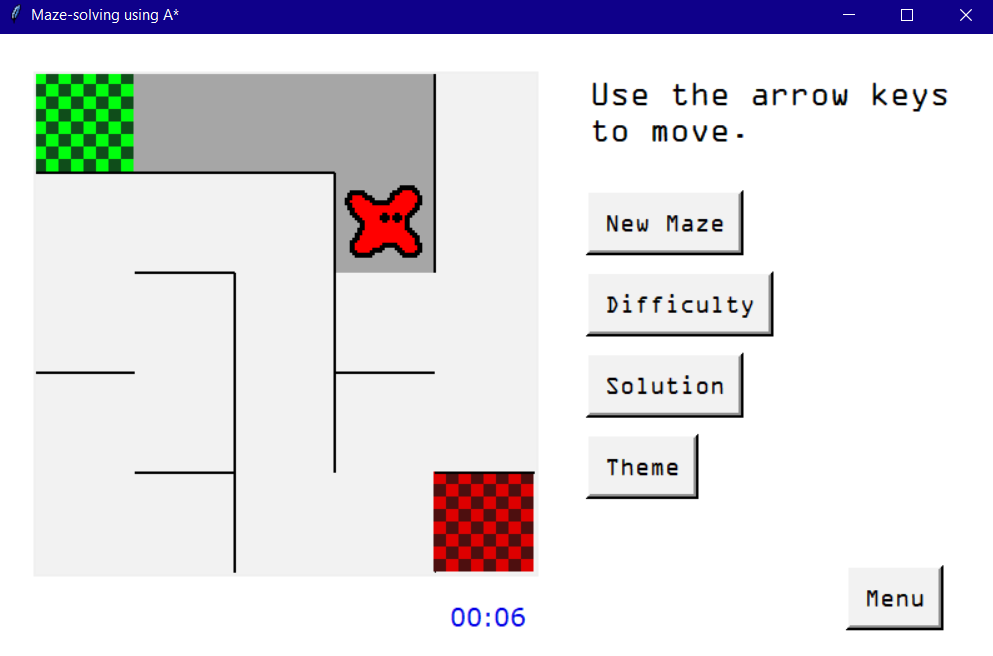


Figure : A 5\*5 maze is produced, with a clear sprite, start cell and target cell. I have moved the sprite so that the trail can be clearly seen.

The theme used when the maze was first generated was the Default theme (screenshots of the other themes are in Section [4.3](#_4.2.5_Extra_user)). If I moved the sprite back along its trail, the cells remained dark grey. The stopwatch incremented every second. I navigated the sprite to the red target cell, and as soon as I touched it the congratulations page was displayed:



Figure : The congratulations page, with my time taken, Easy high score and a message to say that this is a new high score.

As it was my first attempt at this difficulty, I immediately achieved a new high score. This page was produced on top of the maze page, so when I pressed ‘Back’ the focus returned to the main maze page. I could still move around on the old maze, and this page was not brought up if I crossed the target cell again.

I pressed ‘New Maze’ to create a new maze. This time I finished it in 11 seconds, and the congratulations page looked like this:



Figure : The congratulations page, but this time I did not achieve a new high score.

I then pressed the ‘Difficulty’ button, which sent me back to the difficulty selection page, and selected Hard this time. A 15\*15 maze was produced. I decided to press the ‘Solution’ button, which displayed the solution to the maze and stopped the timer:

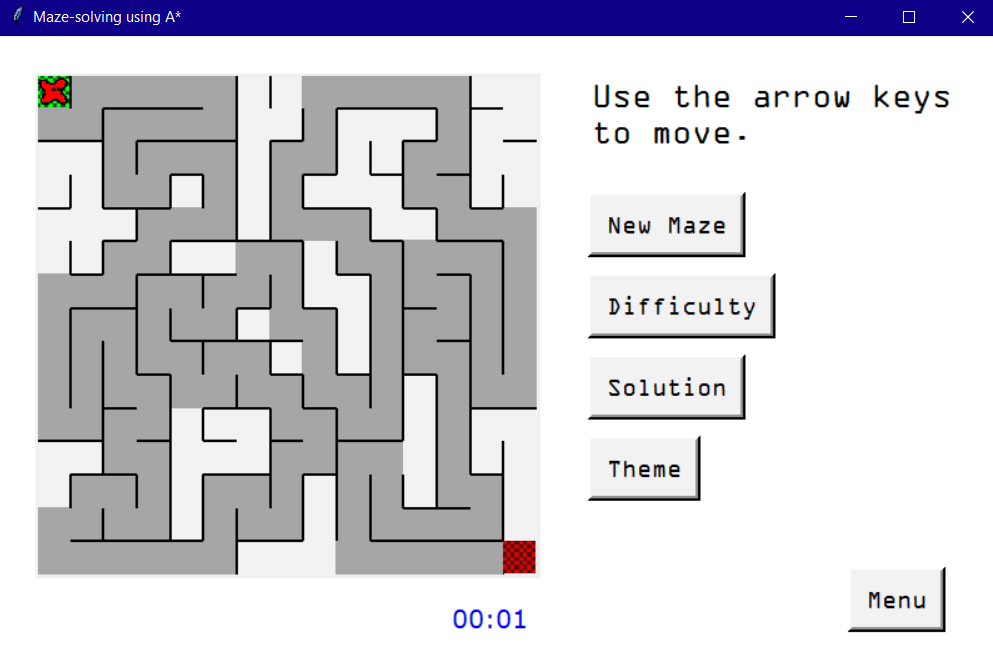


Figure : The solution to a Hard maze, produced when I click the ‘Solution’ button.

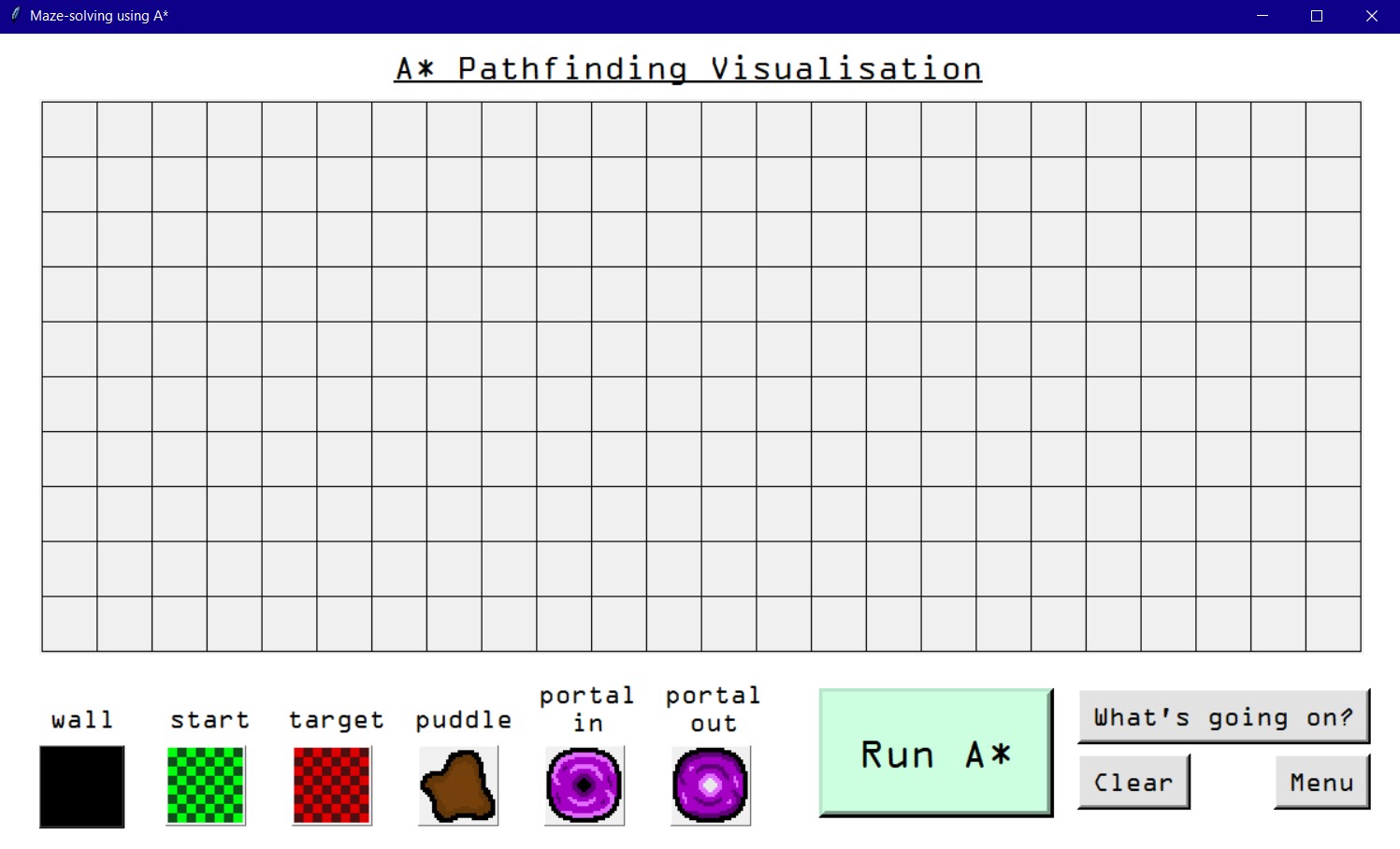
I then clicked ‘Menu’, returning me to the menu page, and clicked ‘A\* Simulation’. I was met with this page:

Figure : The A\* Simulation grid in its initial state.

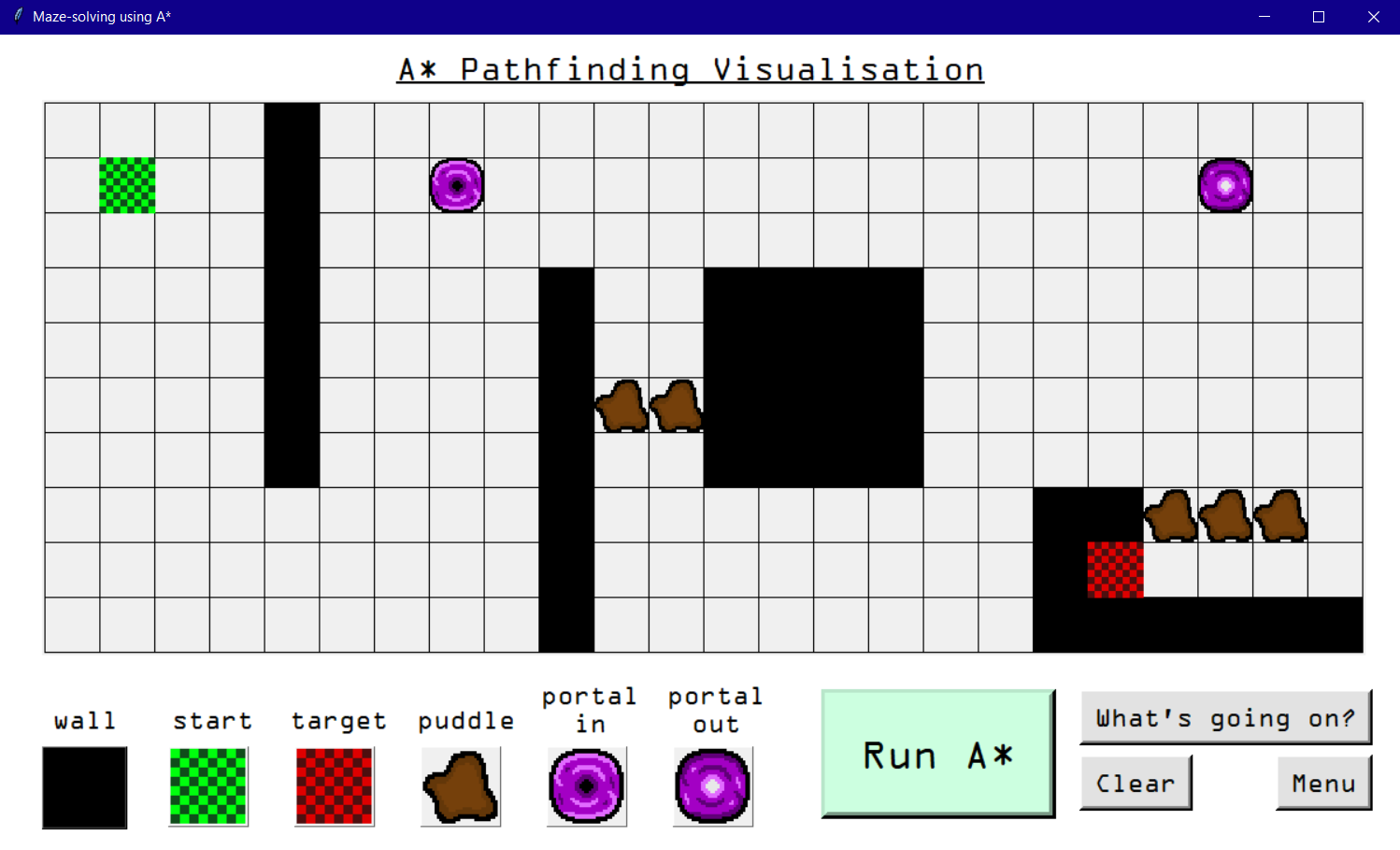
When I clicked one of the objects, I could then click on the grid to place that object there, and click again to remove it. I created a maze to try out the algorithm:

Figure : A simple maze I created for the A\* algorithm to solve.

When I clicked ‘Run A\*’, the program demonstrated the algorithm, clearly showing the cells in the frontier and those that had been visited.

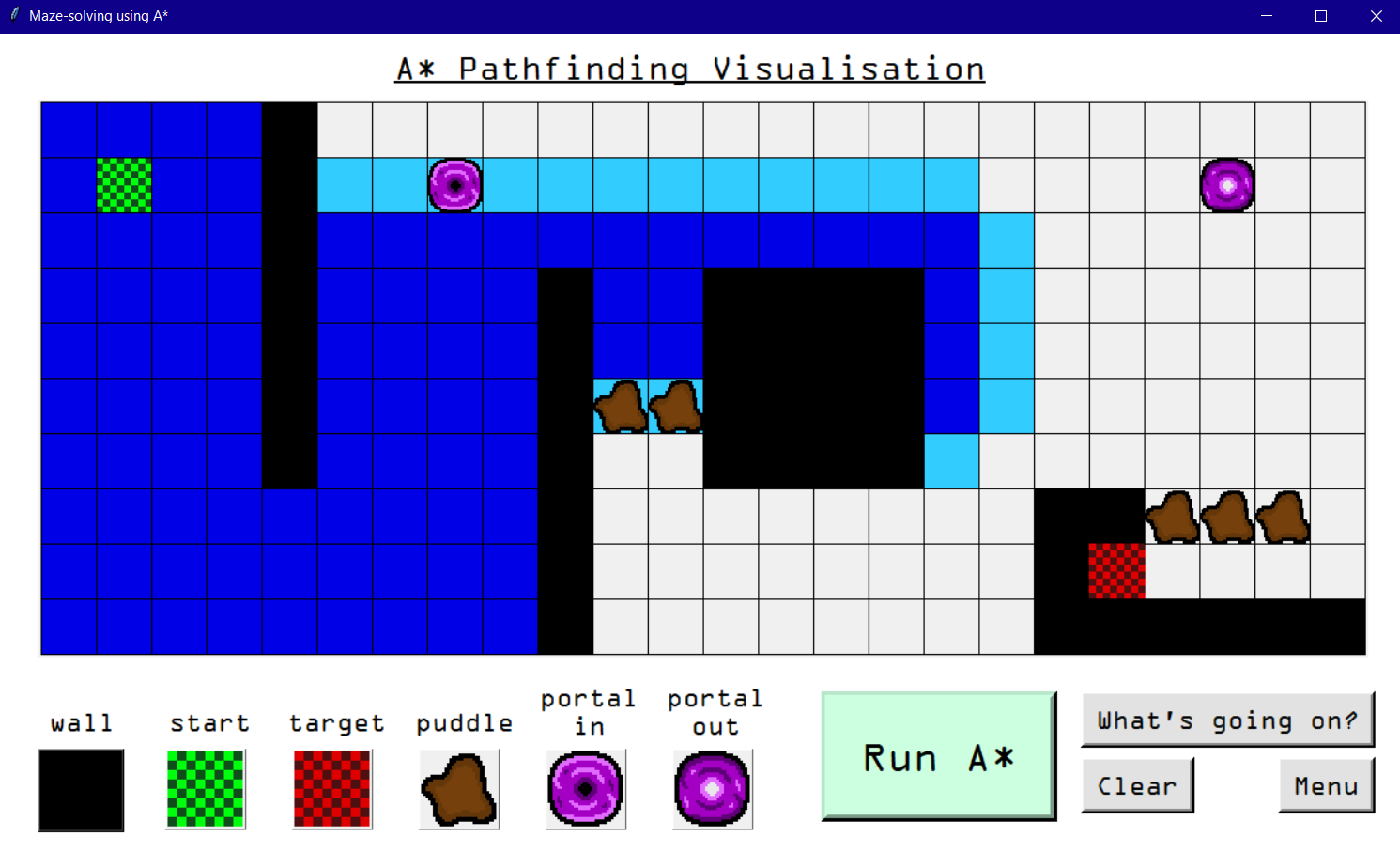
When the target cell was found, the shortest path was highlighted in a bright yellow.

Figure : Partway through a demonstration of the A\* algorithm solving my maze.

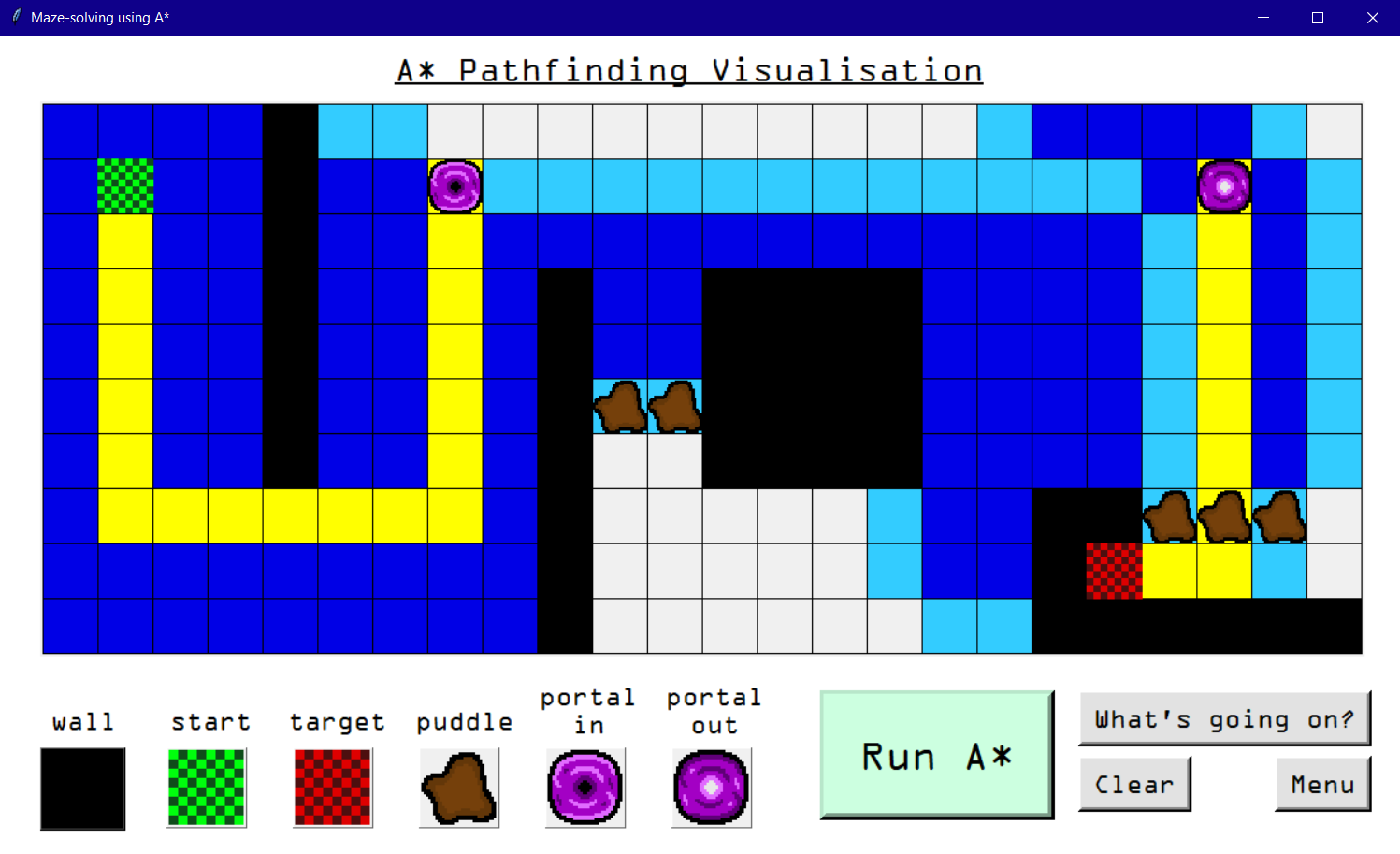
When I pressed ‘Clear’, the grid reset to being empty (Figure 40). When I clicked the ‘What’s going on?’ button, the information page brought up was drastically different from my initial design:

Figure : The solution to my maze was highlighted in yellow.

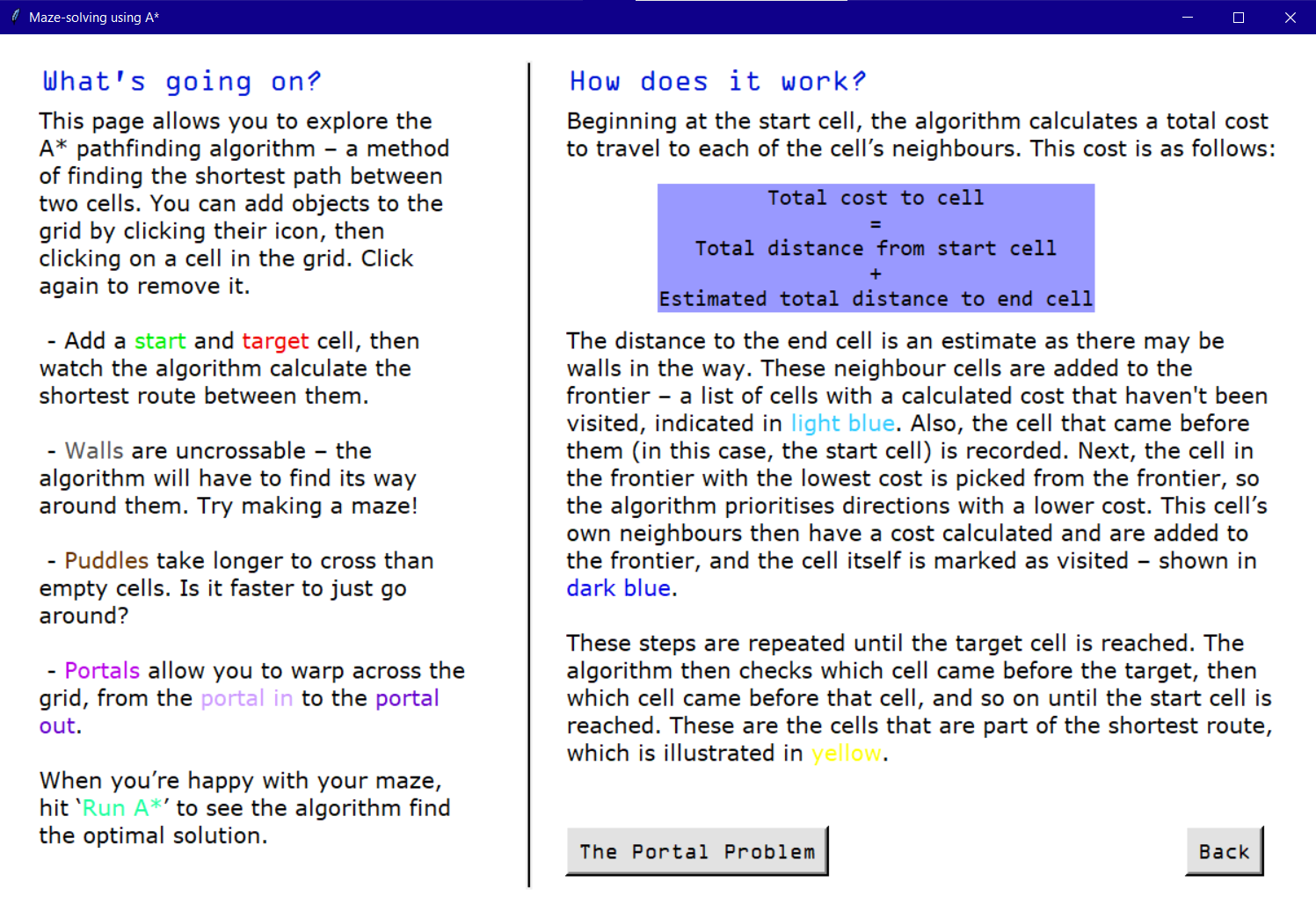
There was also an additional page brought up when I pressed the ‘The Portal Problem’ button:

Figure : The information page produced by pressing the ‘What's going on?’ button.

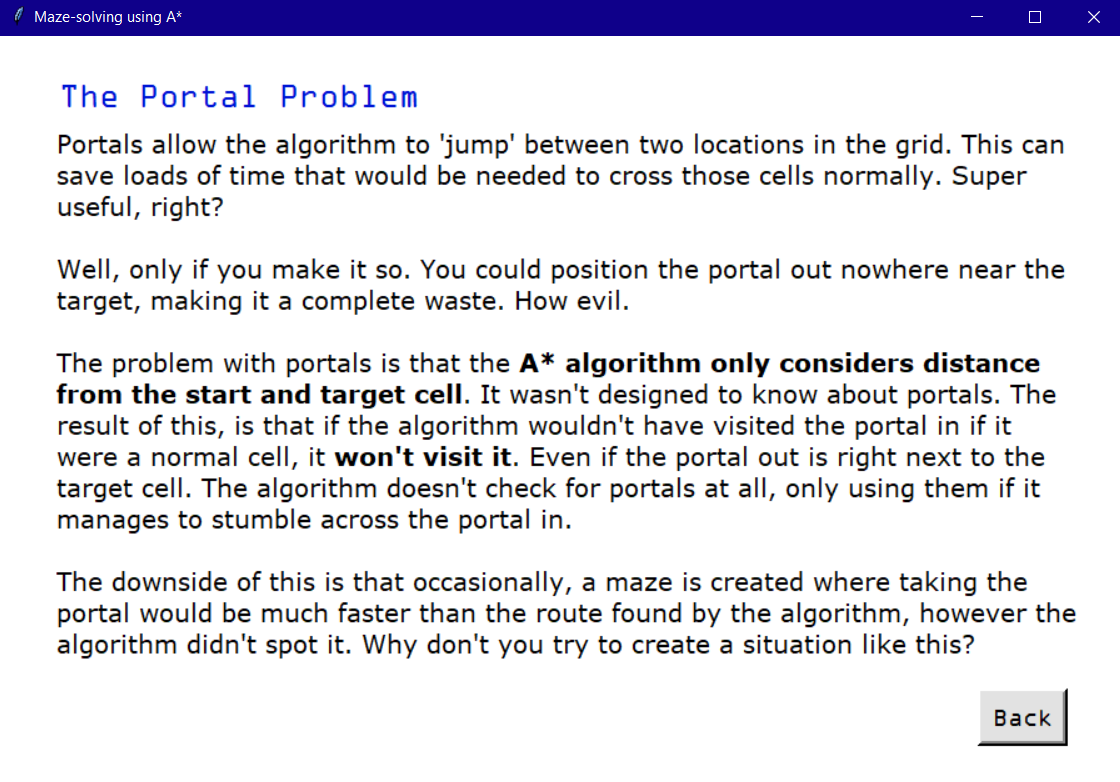
This page discusses something I discovered during the development of my solution. As explained, if a portal is placed so that it would never be visited by the algorithm, the algorithm won’t ever actually check it, even if it would produce a faster solution. I discovered this while creating my own mazes to check that the portals were working correctly, and thought it was worth discussing in detail, as a user may be confused as to why this is the case.

Figure : An additional page explaining a quirk of using portals with the A\* algorithm.

This section of my testing demonstrates how a user may use the system as intended, however other features I didn’t demonstrate here such as error handling and the colour themes in Maze Game are presented in Section [4.3](#_4.2.5_Extra_user).

## 4.2 Detailed testing

In this section I carried out thorough tests of each section of my program. A summary of these tests are shown in the below test tables. Solutions to failed tests are in Section [4.2.5](#_4.2.5_Fixing_failed).

### 4.2.1 User Account Management

I made use of the SQL statement "SELECT \* FROM user\_details" to check the database was being updated correctly. However, this was also be proved by the account being successfully updated in the Maze Game section.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test #** | **Test description** | **Database access** | **Expected result** | **Actual result** | **Pass / Fail** |
| 1.1 | The user can only choose to register or login if there was a successful connection to MySQL. | Yes | The user can select the register and sign in buttons. | The buttons are perfectly usable. | Pass |
| 1.2 | No | The user should be alerted that the connection failed and the buttons should be disabled. | A warning is displayed and the user cannot click the register or sign in buttons. | Pass |
| 1.3 | The register and sign in form should reject blank inputs. | Yes | If the user presses ‘Submit’ without entering a username or password, the program should prompt them to enter one. | The program displays ‘Please enter a username’ if the username is not present, or ‘Please enter a password’ if there is a username but not a password. | Pass |
| 1.4 | The username and password should not be longer than 265 characters. | Yes | The user should be alerted that their username or password is too long. | An error is produced because the data is too long for the respective database field. | Fail |
| 1.5 | The user should not be allowed to register an account with the name ‘Guest’, as this is used for the guest account. | Yes | If the username is ‘Guest’, the program should reject the registration. | A warning ‘That username is in use. Please pick another’ is displayed. | Pass |
| 1.6 | The user should not be allowed to register with a username already in use by someone else. | Yes | If the username is already in use, the program should reject the registration. | A warning ‘That username is in use. Please pick another’ is displayed. | Pass |
| 1.7 | If the username is unique, the user’s account should be created. | Yes | An account with the username, hashed password and None in the high score fields should be created. | The account is created as expected and the menu is displayed. | Pass |
| 1.8 | The user should be alerted if their username or password is incorrect. | Yes | A warning should be displayed that the credentials are incorrect. | A warning, ‘The username or password is incorrect’ is displayed. | Pass |
| 1.9 | If the database cannot be accessed the user should be alerted. | No | A warning should be displayed if the database cannot be accessed (but MySQL can) | A warning message is displayed and the user is advised to play as a guest. The program continues as if the ‘Play as guest’ button was pressed. | Pass |
| 1.10 | If the user selects ‘Play as guest’ the database is not accessed. | N/A | The menu should be produced and the database is not edited. | The program proceeds to the menu using the guest account. | Pass |

### 4.2.2 Menu

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test #** | **Test description** | **Current user** | **Expected result** | **Actual result** | **Pass / Fail** |
| 2.1 | The menu is displayed, with the user’s name at the top. | eleanor1 | The menu displays with this username. | The menu displays with text ‘User: eleanor1’ at the top. | Pass |
| 2.2 | Guest | The menu displays; the name at the top is ‘Guest’ | The menu displays with text ‘User: Guest’ at the top. | Pass |

### 4.2.3 Maze Game

Note that the ‘Relevant variables’ field may either refer to a variable or an attribute of the MazeClass class. Tests involving the database were carried out using the same SQL statement as in the User Account Management tests.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test #** | **Test description** | **Relevant variables** | **Expected result** | **Actual result** | **Pass / Fail** |
| 3.1 | If the user doesn’t select a difficulty they should not be allowed to proceed. | N/A | A warning will be produced asking the user to select a difficulty. | A warning message is produced and the screen remains on difficulty selection. | Pass |
| 3.2 | The correct sized maze is generated based on the chosen difficulty. The sprite, start cell and stop cell are each scaled appropriately. | Difficulty = ‘Easy’ | A 5\*5 maze is generated. | A 5\*5 maze is generated, and the images are appropriately scaled. | Pass |
| 3.3 | Difficulty = ‘Medium’ | A 10\*10 maze is generated. | A 10\*10 maze is generated, and the images are appropriately scaled. | Pass |
| 3.4 | Difficulty = ‘Hard’ | A 15\*15 maze is generated. | A 15\*15 maze is generated, and the images are appropriately scaled. | Pass |
| 3.5 | The stopwatch increments every second once the maze is generated. | N/A | The stopwatch should work as one would expect, increasing by one every second, and updating each digit correctly (e.g. when a minute passes the ‘minutes’ digit increases and the ‘seconds’ digit returns to 0). | The stopwatch begins as soon as the maze is created. The digits are updated correctly. | Pass |
| 3.6 | After an hour the stopwatch should no longer increment. | N/A | The stopwatch should stop at ‘60:00’. | The stopwatch no longer updates after ‘60:00’ as expected. | Pass |
| 3.7 | The arrow keys can be used to move. | N/A | If there is not a wall in the way, the sprite should move in the direction of the arrow key that was pressed. | The sprite moves when the user presses the respective arrow key. | Pass |
| 3.8 | The sprite should not be able to pass through walls. | N/A | If there is a wall in the way the sprite should not move to the next cell. | If there is a wall then the sprite does not move in that direction. | Pass |
| 3.9 | A trail should be displayed showing where the sprite has been. | N/A | If the sprite has not yet visited its current cell, that cell’s colour should change, creating a trail. | As the sprite reaches a new cell its colour is changed as expected. (This does not apply to the target cell.) | Pass |
| 3.10 | When the user reaches the target cell, a congratulations page should be displayed. This contains the user’s score, the current high score on that difficulty, and a message saying ‘New High Score’ if appropriate. | Username = ‘eleanor1’  Old High Score = ‘00:10’  Time Taken = ‘00:08’ | If the user has beaten their current high score, their new time should be displayed in both the ‘Your Time’ and ‘High Score’ sections, and ‘New High Score’ should be displayed. | ‘00:08’ is the time displayed in both sections, and ‘New High Score’ is displayed as expected. | Pass |
| 3.11 | Username = ‘eleanor1’  Old High Score = ‘00:08’  Time Taken = ‘00:12’ | If the user has not beaten their high score, the old high score should be displayed in the ‘High Score’ section and ‘New High Score’ should not be displayed. | ‘00:12’ is displayed under ‘Your Time’, ‘00:08’ under ‘High Score’ and ‘New High Score’ is not displayed. | Pass |
| 3.12 | Username = ‘Guest’  Time Taken = ‘00:15’ | The guest user’s time taken should be displayed, however there should be no attempt to display a high score and ‘New High Score’ should not be displayed. | ’00:15’ is displayed under ‘Your Time’, ‘XX:XX’ under ‘High Score’ and ‘New High Score’ is not displayed. | Pass |
| 3.13 | The congratulations page’s ‘High Score’ section should state the difficulty being played. | Difficulty = ‘Easy’ | The current difficulty should be written before ‘High Score’. | ‘Easy High Score’ is displayed. | Pass |
| 3.14 | Difficulty = ‘Medium’ | ‘Medium High Score’ is displayed. | Pass |
| 3.15 | Difficulty = ‘Hard’ | ‘Hard High Score’ is displayed. | Pass |
| 3.16 | If the user has achieved a new high score, this should be updated in the database. | Difficulty = ‘Easy’  Old High Score = ‘00:10’  Time Taken = ‘00:08’ | The value in the easy\_high\_score field of the database for this user should be updated. | The value ‘00:10’, which was previously in this field, is overwritten with ’00:08’ | Pass |
| 3.17 | Difficulty = ‘Medium’  Old High Score = ‘00:22’  Time Taken = ‘00:17’ | The value in the medium\_high\_ score field of the database for this user should be updated. | The value ‘00:22’, which was previously in this field, is overwritten with ’00:17’ | Pass |
| 3.18 | Difficulty = ‘Hard’  Old High Score = ‘00:30’  Time Taken = ‘00:25’ | The value in the hard\_high\_score field of the database for this user should be updated. | The value ‘00:30’, which was previously in this field, is overwritten with ’00:25’ | Pass |
| 3.19 | The stopwatch should no longer increment once the congratulations page is called. | Time Taken = ’00:22’ | The stopwatch should stop, even if the user continues to move around on this maze. | Once the congratulations page is called the stopwatch remains at ’00:22’. | Pass |
| 3.20 | After the congratulations page has been called once, if the user crosses the target cell again on the same maze it should not be brought up. | N/A | The user is still able to navigate the maze they were just on, however the congratulations page should not be produced again. | If the user attempts to cross the target cell a second time the congratulations page is not produced. | Pass |
| 3.21 | The user can press ‘New Maze’ to create a new maze. | N/A | A new maze should be generated, the trail should be removed, the sprite returned to the start cell and the stopwatch reset to ‘00:00’. | A new maze is generated as expected, with the sprite, trail and stopwatch each reset to their original states. | Pass |
| 3.22 | If the user attempts a new maze without closing the congratulations page, upon completion of the new maze the old congratulations page should close before a new one is opened. | N/A | Only one congratulations page should ever be open on the user’s device. | With one congratulations page open, if I click back on the main maze page and complete a new maze a new congratulations page is produced. I can repeat this indefinitely. | Fail |
| 3.23 | If the user attempts to change difficulty, return to the menu or press X on the main maze page while a congratulations page is open, it should be closed before the program proceeds. | N/A | The congratulations page should be automatically closed by the program. | The user cannot proceed until they manually close the congratulations page. | Fail |
| 3.24 | The user can press ‘Solution’ to view the solution to the current maze. | N/A | The solution should be calculated using the A\* algorithm, and cells in that solution should become the colour of the trail. If the user has strayed away from the shortest path, cells not part of it should return to the original colour. | Cells in the solution become the colour of the trail, and those not in the solution become the original colour of the maze. | Pass |
| 3.25 | After pressing ‘Solution’ a time should not be recorded for this maze. | Time Taken: ’00:17’ | The stopwatch should freeze when ‘Solution’ is pressed, and if the user crosses the target cell the congratulations page should not be produced and the database should not be accessed. | The stopwatch remains at ’00:17’. The congratulations page is not called when the user crosses the target cell, and the user’s high scores are unchanged. | Pass |
| 3.26 | The user can press ‘Themes’ to select a colour theme for the maze. | N/A | A page is produced with five buttons allowing the user to select a theme. Once a button is pressed, the maze page should be recreated using colours associated with this new theme. | When the user selects a theme (even if it is the same as the current theme), the main maze page is recreated using the colours of the chosen theme. | Pass |
| 3.27 | There should only be one themes page open at a time. | N/A | If the user presses the ‘Themes’ button with a themes page already open, a second one should not be brought up. | A new themes page is not brought up (nothing happens). | Pass |
| 3.28 | There should not be a congratulations page and themes page open at the same time. | N/A | If the user completes a maze with the themes page open, or presses ‘Themes’ with a congratulations page open, the page that was previously open should close before the next one opens. | In both cases both pages were open at the same time. | Fail |
| 3.29 | If the user attempts to change difficulty, return to the menu, or press X on the main maze page while the themes page is open, it should be closed before the program proceeds. | N/A | The themes page should be automatically closed by the program. | Pressing X closes the themes page as expected, however with the other two options, the user cannot proceed until they manually close the themes page. If they instead press one of the theme options an error is produced. | Fail |

### 4.2.4 A\* Simulation

Similarly to Maze Game, the ‘Relevant variables’ field may either refer to a variable or an attribute of AStarClass.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test #** | **Test description** | **Relevant variables** | **Expected result** | **Actual result** | **Pass / Fail** |
| 4.1 | The user can click on the grid to place the currently selected object. | 0 ≤ x-coordinate of mouse ≤ 23  0 ≤ y-coordinate of mouse ≤ 9  (relative to the topleft cell which is (0,0)) | The currently selected object is created in that position on the grid. | The object appears in that space as expected. | Pass |
| 4.2 | Clicking a cell with the current object already in it will remove that object. | N/A | When I click on the object, its image should be removed. | The cell reverts back to an empty cell. | Pass |
| 4.3 | Clicking a cell that isn’t empty but doesn’t contain the currently selected object will place the currently selected object in that space. | oldType = *anything other than ‘wall’* | The object in that cell will be replaced by the new object. | The old object is replaced by the new object as expected. | Pass |
| 4.4 | oldType = ‘wall’ | The wall appears to have been replaced, however its edges have not been removed from the list of walls, causing the cell to still be treated as a wall by the algorithm. | Fail |
| 4.5 | The user should only be able to place objects within the grid. | x-coordinate of mouse <0 | No cell should be placed as it wouldn’t be visible. | No cell is placed (nothing happens). | Pass |
| 4.6 | y-coordinate of mouse <0 | Pass |
| 4.7 | x-coordinate of mouse >23 | An error occurs as the mouse coordinates do not fall within my list of cell coordinates. | Fail |
| 4.8 | y-coordinate of mouse >9 | Fail |
| 4.9 | Only one of the following cell types can exist on the grid:   * Start Cell * End Cell * Warp In * Warp Out | currentObject = ‘startcell’ | If the user attempts to place more than one of these objects, a suitable error message should be produced and the object not placed. | An error message ‘You can only place one start cell’ is produced. | Pass |
| 4.10 | currentObject = ‘endcell’ | An error message ‘You can only place one target cell’ is produced. | Pass |
| 4.11 | currentObject = ‘warpin’ | An error message ‘There can only be one portal system’ is produced. | Pass |
| 4.12 | currentObject = ‘warpout’ | Pass |
| 4.13 | A start and target cell must be present on the grid in order to run the algorithm. | N/A | If one or both of these cells are missing the user should be alerted when they press ‘Run A\*’. | An error message ‘The grid is missing a start cell’ is produced if there is no start cell. The message ‘The grid is missing a target cell’ appears if there is a start cell but no target cell. | Pass |
| 4.14 | If portals are used both a portal in and portal out must be present on the grid. | N/A | A message should appear stating which object is missing when the user presses ‘Run A\*’. | If the portal in is missing an error message ‘The portal needs an entrance’ is produced. | Pass |
| 4.15 | N/A | If the portal out is missing an error message ‘The portal needs an exit’ is produced. | Pass |
| 4.16 | If the grid is valid the A\* visualisation should begin when the user presses ‘Run A\*’. | N/A | The fastest path should be found and the user can watch the algorithm as it checks cells, and highlights the fastest route when it reaches the target cell. | The visualisation runs as expected. | Pass |
| 4.17 | The algorithm should still run even if it is impossible to reach the target cell. | N/A | The user can still watch the algorithm adding cells to the frontier and visited list. | The algorithm still runs until all cells it was able to visit have been visited. | Pass |
| 4.18 | The user should not be able to edit the grid while the visualisation is running. | N/A | If the user clicks the grid while the visualisation is playing, no object should be placed and the visualisation does not stop. | The grid is unchanged and the visualisation continues uninterrupted. | Pass |
| 4.19 | When the visualisation has finished the user can make amendments to their grid. | N/A | The objects and colours remain in place, and the user is able to add and remove objects as they please. | The objects and colours remain as they were when the visualisation finished, and the user is free to add or remove objects. | Pass |
| 4.20 | If the user presses ‘Run A\*’ again, the colours should clear before the algorithm is reapplied to the new grid. | N/A | The objects should remain exactly where they are but the colours on the grid should revert to white, then the visualisation begins again. | The colours on the grid are removed and the visualisation runs on the intended grid perfectly. | Pass |
| 4.21 | The user can press ‘Clear’ to clear the grid. | N/A | All objects and colours should be removed from the grid. | The grid resets to empty. Existing walls are deleted from the list of walls, so the next instance of the algorithm does work as intended. | Pass |
| 4.22 | The ‘Clear’ button should not work while the visualisation is running. | N/A | Until the visualisation finishes, nothing should happen if the user presses ‘Clear’. | The visualisation continues without interruption. | Pass |
| 4.23 | The user can press ‘What’s going on?’ to bring up a page with information about the algorithm, and from there press ‘The Portal Problem’ to bring up extra info regarding portals. | N/A | The two pages should be produced upon pressing the button. | The pages display as intended. | Pass |
| 4.24 | There should be no more than one of each of these pages open at a time. | N/A | Pressing the button to open one of these pages when it is already open should have no effect. | Both pages would open multiple times if the button to call them was pressed more than once without closing them. | Fail |
| 4.25 | If the user closes the main A\* page or attempts to return to the menu while these pages are open, they should close. | N/A | The information and portal problem pages should automatically close when the main A\* page closes. | The pages remain open, and the user cannot proceed until both have been closed. | Fail |

### 4.2.5 Fixing failed tests

Below is a table detailing the failed tests and how I was able to fix them.

|  |  |  |  |
| --- | --- | --- | --- |
| **Test #** | **Problem** | **Explanation of solution** | **Notes** |
| 1.4 | If the user tries to register with a username or password longer than 256 characters an error occurs. | Next to the check for if one of the fields is empty, I added a check for if the length of the username or password is greater than 256, producing a warning if so and not allowing the user to proceed. | N/A |
| 3.22 | Multiple congratulations pages can be open at once. | I edited one of my variables *themePageOpen* to be *topPage*, which would be assigned the Tkinter window *congratulationsWindow* when the congratulations subroutine is called. Then, at the start of the subroutine I included a check for whether *topPage* was not None, destroying the window if so. I also had to edit the ‘WM\_DELETE\_WINDOW’ protocol of the congratulations page to point to a subroutine *closeTopPage(),* in which I reset *topPage* to None, so that if a user did close the congratulations page it would open as expected the next time they completed the maze. | This change heavily influenced my solutions to tests 3.23, 3.28 and 4.24. |
| 3.23 | The congratulations page doesn’t automatically close when the main maze page is closed. | In the *main()* subroutine, I adjusted the ‘WM\_DELETE\_WINDOW’ protocol of the maze page to point to a new subroutine *closeWindows()* in which I closed the maze page, and if *topPage* was not None, closed that page too. I added a similar check to the *changeDifficulty()* and *returnToMenu()* subroutines so that the congratulations page would also be destroyed when the user selected these options. | I had already considered using ‘WM\_DELETE\_WINDOW’ to close the themes page if the user pressed X on the main maze window. I had overlooked the fact that it was also needed for the congratulations page. |
| 3.28 | Both a congratulations page and themes page can be open at the same time. | By naming the variable *topPage* instead of *congratulationsPage*, I was able to also assign it the themes page when this was opened. I added a check for *topPage* before the themes page was created, similarly to the one in the *congratulations()* subroutine, and this meant that one of these pages would be closed before the other opened. | I also had to amend the call to subroutine *closeThemePage()* to refer to *closeTopPage()*, the same subroutine used to close the congratulations page. |
| 3.29 | The themes page doesn’t automatically close when the main maze page is closed using the Difficulty or Menu buttons. | The checks I added to the *changeDifficulty()* and *returnToMenu()* subroutines for Test 3.23 also solved this issue. | N/A |
| 4.4 | When overwriting a wall with another object, the walls are not removed from the list of walls so the algorithm behaves as if the wall was still there. | I already had a check for if the cell’s old type was a wall, and if so remove the walls from the list of all walls, however this check was situated inside another check for if the currently selected type was the same as the old type (so this would work if the user’s currently selected type was wall, but not if it was anything else). Moving these statements outside of this check fixed the issue, as the walls would be removed regardless of what the currently selected object was. | N/A |
| 4.7 | If the user manages to click just to the right of the grid, an error is raised as the algorithm tries to edit a cell that doesn’t exist. | I added a simple if statement to check that the mouse coordinates were within the range of the grid. | Both issues were fixed with one line of code. |
| 4.8 | If the user manages to click just below the grid, an error is raised as the algorithm tries to edit a cell that doesn’t exist. |
| 4.24 | Multiple of the information and portal problem pages can be open at a time. | I created two variables, *infoPage* and *portalPage*, which functioned much the same as *topPage* from Maze Game. I added checks for whether these variables existed at the start of the respective page’s definition, and deleted them if so, before updating them after the new page is created. I also added subroutines *closeInfoPage()* and *closePortalPage()* so that the pages would behave as expected once closed. | As both of these pages can exist at the same time, I had to add two variables to contain them instead of one like with the congratulations and themes pages. |
| 4.25 | The information and portal problem pages don’t automatically close when the main A\* page is closed. | Similarly to in Maze Game, I adjusted the ‘WM\_DELETE\_WINDOW’ protocol of the main A\* page to point to a subroutine *closeWindows()*, where I closed the main page, checked for the existence of the two other pages, and if they existed, closed them too. | N/A |

Here is some of the most relevant code used in my solutions:

**Test 1.4:**

Checking if the username or password is too long:

elif len(username) >256:

    messagebox.showwarning(title = 'Maze solving using A\*', message = 'The username is too long.')

elif len(password) >256:

    messagebox.showwarning(title = 'Maze solving using A\*', message = 'The password is too long.')

…

**Test 3.22:**

Setting *topPage*:

variables.setTopPage(congratulationsWindow)

Destroying the congratulations page:

if variables.getTopPage():

    variables.getTopPage().destroy()

#endif

Adjusting the congratulations page’s closing protocol:

congratulationsWindow.protocol('WM\_DELETE\_WINDOW', lambda: closeTopPage(variables))

The *closeTopPage()* subroutine:

def closeTopPage(variables):

    '''Properly close the congratulations or themes window'''

variables.getTopPage.destroy()

    variables.setTopPage(None)

#endsub

**Test 3.23:**

Adjusting the main maze page’s closing protocol:

mazeWindow.protocol('WM\_DELETE\_WINDOW', lambda: closeWindows(mazeWindow, variables))

The *closeWindows()* subroutine:

def closeWindows(mazeWindow, variables):

    '''Close the congratulations or themes window if the user closes the main window'''

    mazeWindow.destroy()

    if variables.getTopPage():

        variables.getTopPage().destroy()

    #endif

#endsub

(The new code added to the *changeDifficulty()*  and *returnToMenu()* subroutines was the same as the ‘Destroying the congratulations page’ code from Test 3.22).

**Test 3.28:**

(The check for *topPage* was the same as the ‘Destroying the congratulations page’ code from Test 3.22, and the themes page is also closed using the *closeTopPage()* subroutine seen in this test).

**Test 4.4:**

The code to remove the walls (which I simply had to move):

#Remove the walls if appropriate

if oldType == 'wall':

    walls.remove(newCell.getEdges()[0])

    walls.remove(newCell.getEdges()[1])

    walls.remove(newCell.getEdges()[2])

    walls.remove(newCell.getEdges()[3])

#endif

**Test 4.7 and Test 4.8:**

Checking the upper bounds of the cell coordinates:

if cellx <=23 and celly <= 9:

…

**Test 4.24:**

(The code is the same as in Test 3.22, just replacing *topPage* with *infoPage* or *portalPage* and *congratulationsWindow* with *infoWindow* or *portalWindow* as required).

**Test 4.25:**

The *closeWindows()* subroutine:

def closeWindows(AStarWindow, variables):

    '''Close the information or portal problem window if the user closes the main window'''

    AStarWindow.destroy()

    if variables.getInfoPage():

        variables.getInfoPage().destroy()

    #endif

    if variables.getPortalPage():

        variables.getPortalPage().destroy()

    #endif

#endsub

## 4.3 Extra user interface evidence

In this section are screenshots of some of the pages in my program not already evidenced in Section [4.1](#_4.1_User_Experience).

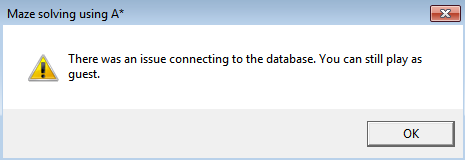


Figure : The error message produced when I open the program on a device not connected to my database.

If the device I am using cannot connect to the database, this message is produced when the program is opened. The ‘Register’ and ‘Sign In’ buttons become disabled, but I can still press ‘Play as guest’.

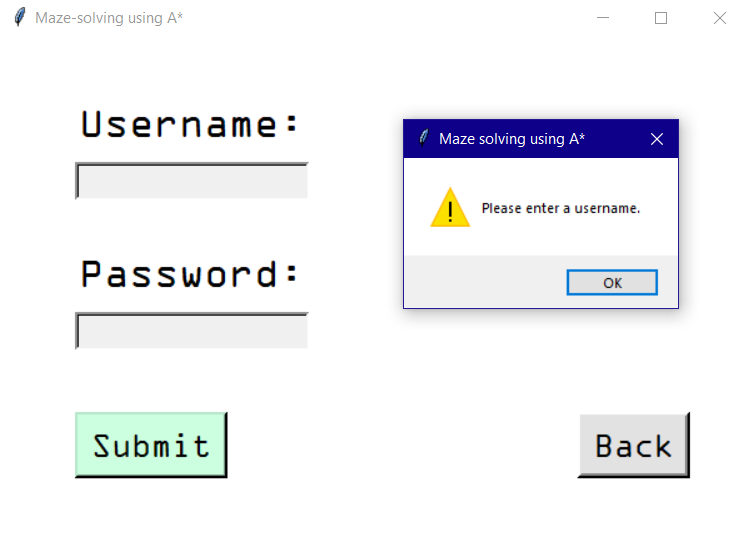
This text box is commonly used if the user makes an erroneous input. When registering:

Figure : The error message produced if the user presses ‘Submit’ without inputting a username.

* If they do not enter a username, the message ‘Please enter a username’ is produced as in Figure 46.
* If they enter a username but leave the password field blank, the message ‘Please enter a password’ is produced.
* If they enter a username already in use, the message ‘That username is already in use. Please pick another’ is displayed.
* If they enter the username ‘Guest’, the message ‘That username is already in use. Please pick another’ is displayed.
* (After testing only) If the user enters a username longer than 256 characters, the message ‘The username is too long’ is produced.
* (After testing only) If the user enters a password longer than 256 characters, the message ‘The password is too long’ is produced.

When signing in:

* The messages produced if one of the fields is empty when registering are also produced when signing in.
* If the credentials are incorrect the message ‘The username or password is incorrect’ is produced.



Figure : The message produced when I successfully sign in to an account.

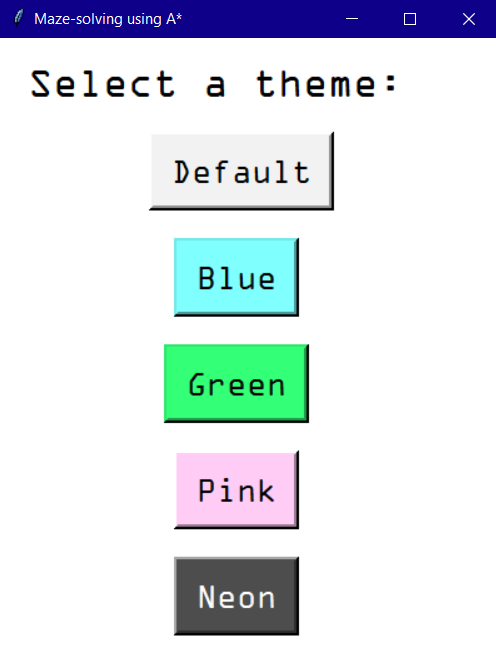


Figure : The themes page, allowing the user to select a colour theme for the maze.

This page’s background and text colour is dependent on the theme currently selected; however the buttons remain exactly as they are here.

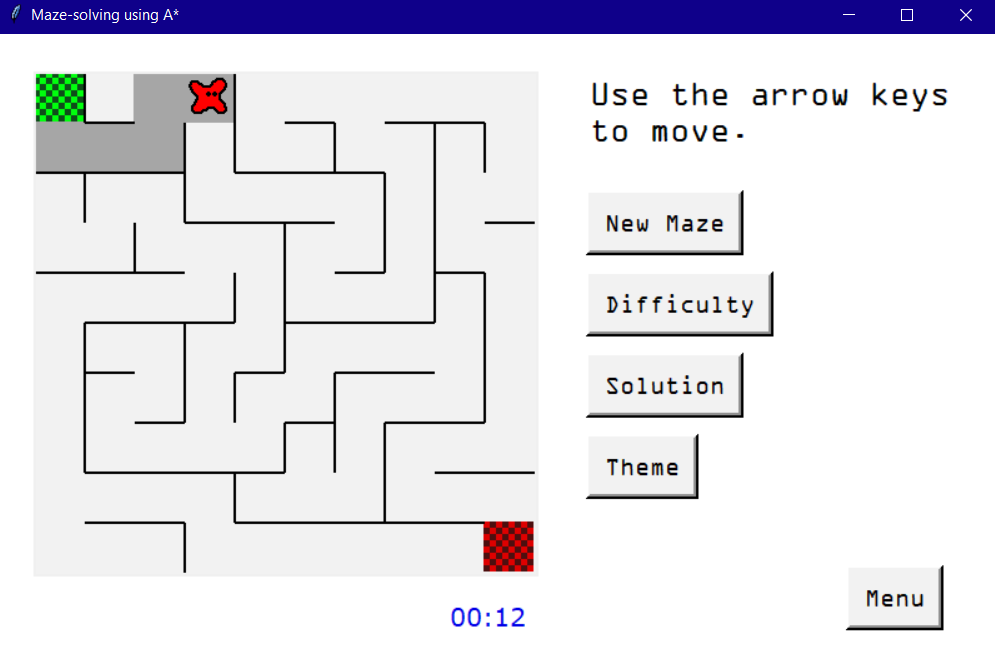
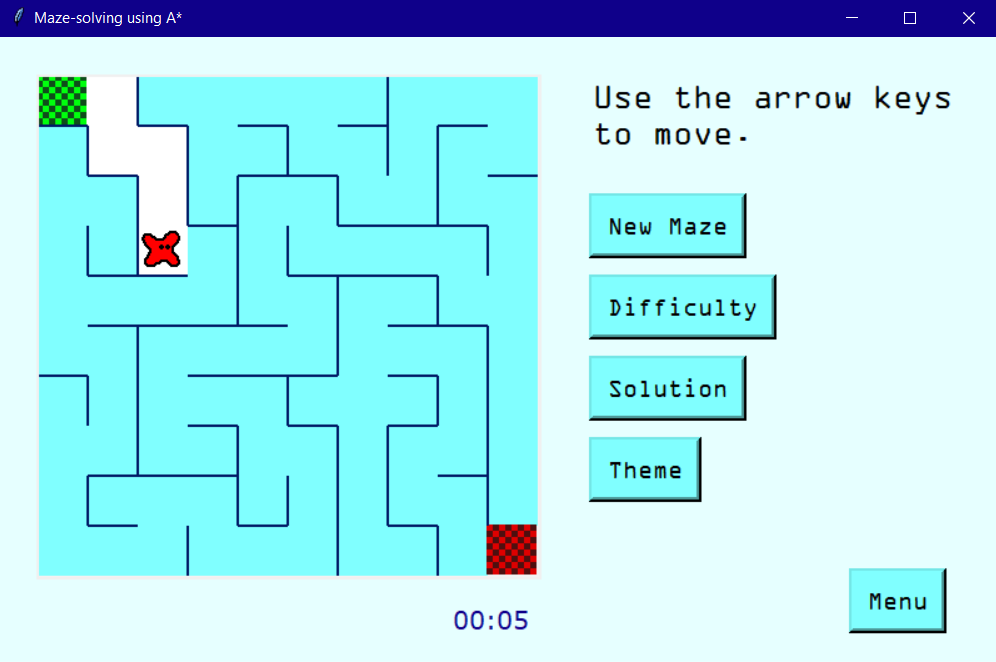


Figure : A medium maze in the Blue theme.

Figure : A medium (10\*10) maze using the Default theme.

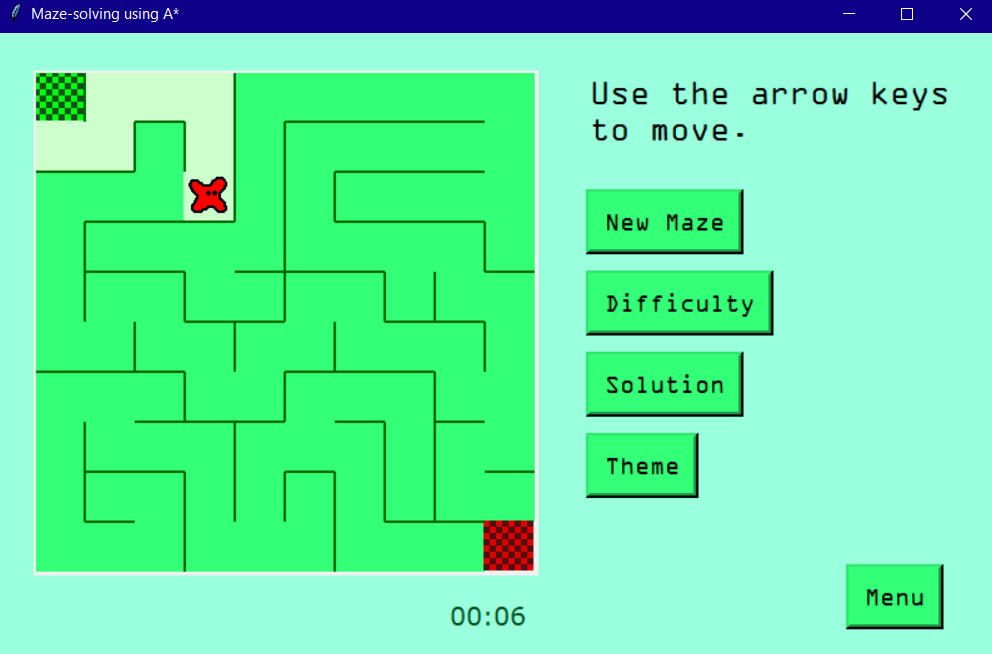
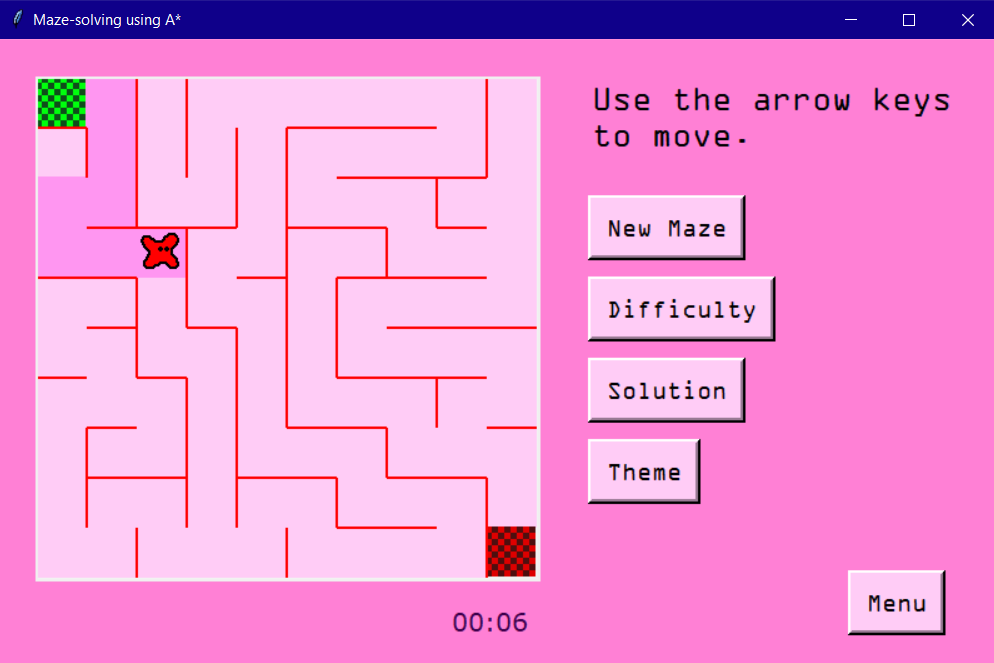


Figure : A medium maze in the Pink theme.

Figure : A medium maze in the Green theme.

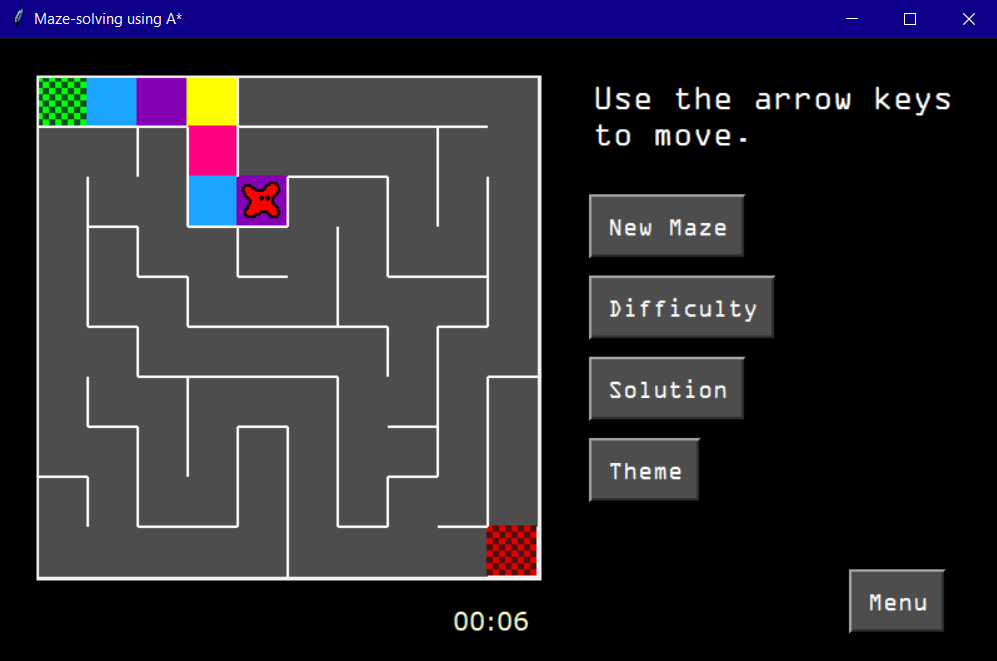
The Easy and Hard difficulties also support these themes.

Figure : A medium maze in the Neon theme.



Figure : The congratulations page when using the guest account, using the Blue theme.

When using the guest account, the database is not accessed and the high score label is replaced with ‘XX:XX’. The congratulations page has its theme matched to that of the maze; different versions exist for each of the other themes.

In the A\* Simulation, the same Tkinter message box as in Figure 46 is used to display a warning if the user presses ‘Run A\*’ with the following conditions:

* There is no start cell on the grid - the message ‘The grid is missing a start cell’ is produced.
* There is a start cell but no target cell - the message ‘The grid is missing a target cell’ is produced.
* There is a portal in but not a portal out - the message ‘The portal needs an exit’ is produced.
* There is a portal out but not a portal in - the message ‘The portal needs an entrance’ is produced.

The following messages are produced as soon as the user meets the criteria:

* The user attempts to add more than one start cell to the grid - the message ‘You can only place one start cell’ is produced.
* The user attempts to add more than one target cell - the message ‘You can only place one target cell’ is produced.
* The user attempts to add more than one of either type of portal - the message ‘There can only be one portal system’ is produced.

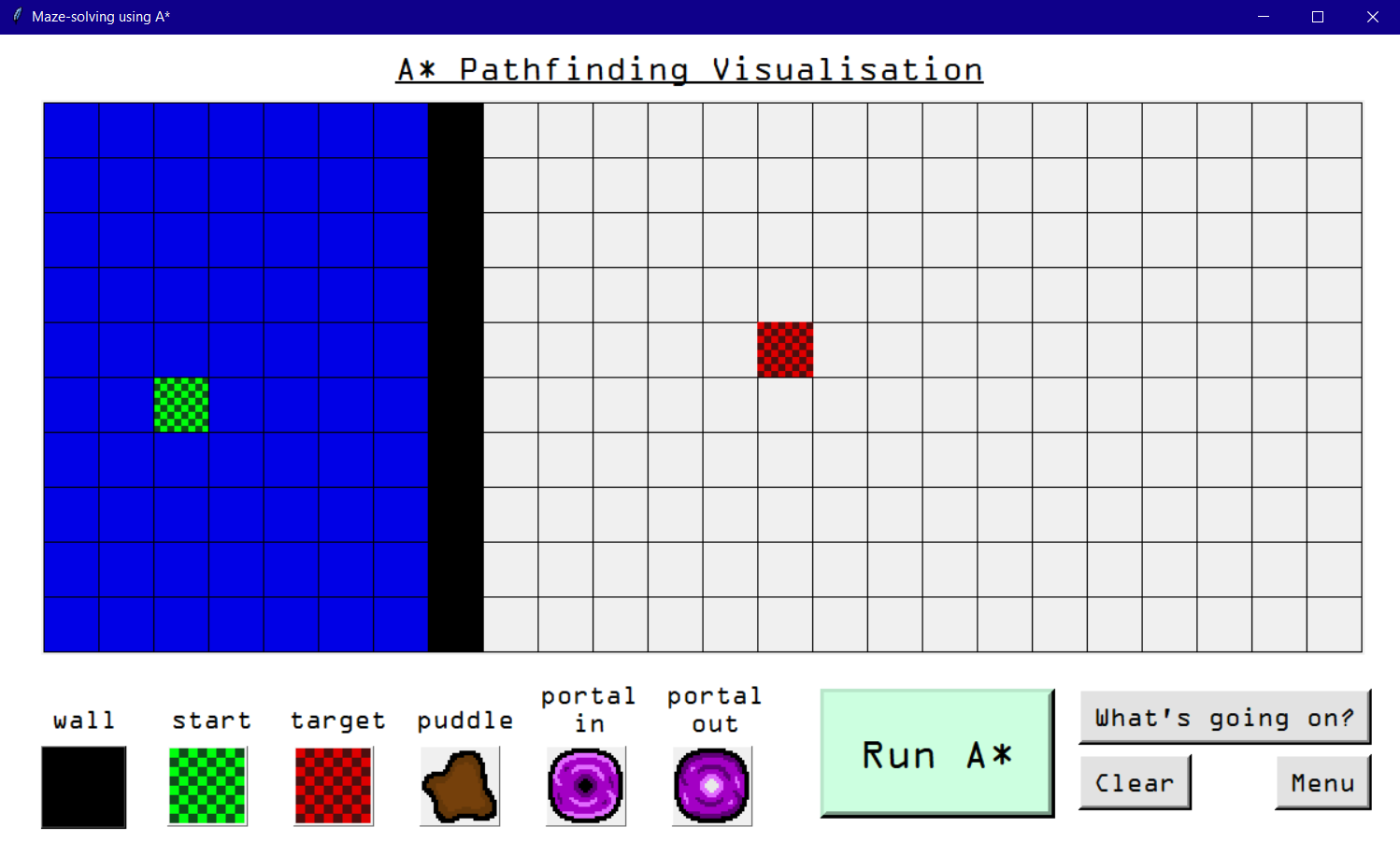
I can still watch the algorithm stepping through cells until all cells it can reach have been visited. At this point I am free to edit the grid again.

Figure : The result when the user creates an impossible grid in A\* Simulation.

## 4.4 Testing against project objectives

Below is a table referring to where in the Testing section there is evidence of each of my project objectives being met.

|  |  |  |
| --- | --- | --- |
| **Objective** | **Test Location** | **Page Number** |
| 1. Upon starting, the program will prompt the user to register or log in to their account. | User Experience based testing | 56 |
| 1.1. They will have to register a username and password. | User Experience based testing | 57 |
| 1.1.1. All usernames are stored in a database. This username must be checked against those in the database to make sure it is unique. This database will be accessed using SQL. | No specific test for this, but see how the previous high score is retained in User Experience based testing | 62 |
| 1.1.2. If the username is taken the user should be alerted. | Extra user interface evidence | 85 |
| 1.1.3. The password will also be stored in the database. It should be encrypted using a hashing algorithm instead of being stored simply as plaintext. | No specific test for this, but successfully signing in in Extra user interface evidence shows that the password is being stored | 86 |
| 1.2. Once the account has been successfully created (or a user with an already existing account has logged in) they will be directed to the main menu. | User Experience based testing | 58 |
| 2. The main menu will allow the user to select which mode they wish to play. | User Experience based testing | 58 |
| 2.1. There will be a button for the maze game and a button for the A\* simulator. | User Experience based testing | 58 |
| 2.2. They will also be able to sign out from this screen, returning to the first screen. | User Experience based testing | 58 |
| 3. In the maze section: | N/A | N/A |
| 3.1. The user will select a difficulty (easy, medium or hard). | User Experience based testing | 59 |
| 3.2. A maze will be generated using the randomized depth-first search algorithm, its size depending on the difficulty selected in objective 3.1. | Easy - User Experience based testing  Medium - Extra user interface evidence  Hard - User Experience based testing | 60  88  63 |
| 3.3. The user can then play the maze, using the arrow keys to navigate a simple sprite from the start cell to the target cell. | User Experience based testing | 60 |
| 3.3.1. Instructions to use the arrow keys to play should be clearly displayed. | User Experience based testing | 60 |
| 3.3.2. When the sprite moves a trail should be displayed showing where it has been. | User Experience based testing | 60 |
| 3.3.3. There will be a stopwatch displayed on this screen so the user can see the current time taken. | User Experience based testing | 60 |
| 3.4. When the user completes the maze, a congratulations message will be produced, which states the time taken and the user’s current high score. | User Experience based testing | 62 |
| 3.4.1. The user’s fastest time (on this difficulty) is taken as their highest score and saved to the database. | No specific test for this but evidenced by the fastest time being displayed on the second congratulations page in User Experience based testing | 62 |
| 3.4.2. Should the user beat their high score, this will be updated in the database and the new score will be displayed the next time they beat a maze of the same difficulty. | No specific test for this, but when the user plays a difficulty for the first time they automatically get a new high score, so see the two congratulations pages in User Experience based testing | 61  62 |
| 3.5. There will be an option to see the answer. When clicked the A\* algorithm will be used and the shortest route through the maze will be displayed. | User Experience based testing | 63 |
| 3.5.1. No time will be recorded for this attempt. Instead the user must click the play again button (see objective 3.6) to start a new maze. | User Experience based testing | 63 |
| 3.6. There will also be options to play again, change difficulty level or return to the menu. | User Experience based testing | 60 |
| 3.7. There will be a customisation option which allows the user to select a colour theme for the maze. | Extra user interface evidence | 87  88  89  90 |
| 4. In the A\* section: | N/A | N/A |
| 4.1. An empty grid will be displayed. | User Experience based testing | 63 |
| 4.2. At the bottom of the screen the user can select what to place with their mouse: | User Experience based testing | 63 |
| 4.2.1. A solid wall. | User Experience based testing | 63 |
| 4.2.2. A start node (only one). | User Experience based testing | 63 |
| 4.2.3. An end node (only one). | User Experience based testing | 63 |
| 4.2.4. An obstacle which takes longer to cross than an empty cell. | User Experience based testing | 63 |
| 4.2.5. A warp panel that moves the frontier across the grid. | User Experience based testing | 63 |
| 4.3. Once the user has drawn their maze, they can select the start button. | User Experience based testing | 64 |
| 4.3.1. If a start and end node haven’t been placed the user should be notified. | Extra user interface evidence | 91 |
| 4.3.2. The A\* algorithm will then run, changing the colour of each cell it visits. There will be a short delay so that the user can see the changes occurring. | User Experience based testing | 65 |
| 4.3.3. When the shortest path has been found it will be highlighted in another colour. | User Experience based testing | 66 |
| 4.4. There should be an option to clear the grid. | User Experience based testing | 63 |
| 4.5. There should be a button containing information about this section of the software: | User Experience based testing | 63 |
| 4.5.1. How to interact with the area (place objects etc.) | User Experience based testing | 67 |
| 4.5.2. What the A\* algorithm is and does (finds the shortest path from one node to another) | User Experience based testing | 67 |
| 4.5.3. How the algorithm works (explaining how the cost for each cell is calculated etc.) | User Experience based testing | 67 |
| 4.6. There should also be an option to return to the menu. | User Experience based testing | 63 |

# 5 Evaluation

## 5.1 Whole system evaluation

Personally, I am satisfied with the overall result of my program. As a whole, it feels like a cohesive unit and is easy to navigate. I am happy with the user interface and feel that both of the major sections are easy and enjoyable to use, and that there are no lingering errors (since fixing my failed tests). By allowing the user to play mazes of varying difficulties, and then create their own maze for the A\* algorithm to solve, I feel that my program is an effective solution to my original problem, creating an interface that allows the user to interact with the algorithm and compare it to their own maze-solving approach.

However, if I were to further develop the solution, I would allow an option for the user to visualise the A\* algorithm working from within the maze section. Currently the algorithm is used to solve the maze when the user presses the ‘Solution’ button, however, including the visualisation from A\* Simulation here would allow the user to visually apply the algorithm to a maze that they have attempted themselves. This could be done by reusing the *runAStar()* subroutine from A\* Simulation within Maze Game, with some modifications to allow it to work with the maze canvas. This would make user-versus-algorithm comparison even easier.

Another potential extension of the program could be to save the user’s created mazes from A\* Simulation in the database, so that they can revisit their previously generated mazes. This was something I considered when I was first conceptualising my program, but I decided that it wasn’t a core requirement. However, due to the way I have used the *allCells* variable to keep track of the grid layout, I could create an array of the cell type in each position and add this to the database using further SQL commands, and retrieve it in order to recreate a loaded maze.

Finally, options to delete the account or change the password are commonly seen with most login systems, and would be useful tools to include. They would require making use of the SQL DELETE and UPDATE statements, which wouldn’t be too difficult. However, in many systems these features make use of email address verification to authenticate the user wishing to make these changes. If I were to include this I would need another field in my database for the email address, and I would have to create an email which would be sent to this address to verify the user, which would be more of a challenge.

There were, as well, some minor inconveniences that I could attempt to fix. When the program creates a new Tkinter window, the device does not automatically focus on that window. In most cases this is no big deal, as the user can just click where they wish and this click automatically brings the window to focus, however when the main Maze Game window is created, the user has to click on the window before they can use the arrow keys to move the sprite. The stopwatch, however, starts as soon as the maze is created – so the user wastes seconds clicking on the window. I attempted to fix the focus issue after development but did not find a suitable Python or Tkinter method to do this. However, a simpler solution would be to not start the stopwatch until the user makes their first move (likely by using a Boolean variable *firstMove*, or something similar). This was actually pointed out to me by another of my friends, Ben, when I was showing him my progress.

Additionally, recall that in Section [1.3.1](#_1.3.1_Current_Software), when discussing the MathIsFun software I made a comment on how sometimes mazes could be generated with large portions being inaccessible until after the target cell is reached. This is unfortunately also present in my program (though the user can still explore it after completing the maze if they so wish). It is due to the nature of the randomised depth-first search algorithm, and to avoid it I would have had to have picked a different algorithm, or dynamically chosen a location for the target cell at the end of a branch, which may have led to even easier mazes being generated accidentally.

## 5.2 Differences from the Design section

My program ended up following my design pretty closely, however there were a few major changes that I will discuss here.

* Inclusion of a play as guest option – following feedback from Noah I decided to create this option. It wasn’t particularly difficult to code, however it has saved me lots of time while editing my program from devices not connected to my database and during testing.
* The A\* Simulation information page has split into two – a page with information about the section and algorithm, and one about the issue with portals, which I discovered while developing the program code. See Section [4.1](#_4.1_User_Experience) for more information and evidence of the new page designs.
* In the A\* Algorithm, *frontier* now uses coordinates instead of Cell objects. While finding the cell’s neighbours required them in object form, other methods such as sorting the priority queue required them in coordinate form. In general, it was easiest to keep them in coordinate form in the priority queue and convert to Cell form when required.
* The designs of the portal in and portal out are quite different from my initial design – I am very satisfied with the final result.
* Due to the nature of Tkinter buttons not allowing a variable to be returned, I had to create two extra classes, MazeClass and AStarClass, with attributes correlating to each variable used in multiple subroutines, as well as a getter and setter for each. In both sections I created instances of these classes titled ‘variables’, which could be passed into the subroutine being called and the required attributes accessed using this. This was a time consuming process, but prevented my program from being littered with global variables.

## 5.3 Evaluation against project objectives

Below is a table discussing how my program has met each of my initial objectives.

|  |  |  |
| --- | --- | --- |
| **Objective** | **Met** | **Comments** |
| 1. Upon starting, the program will prompt the user to register or log in to their account. | Yes | The user can choose to register, login or play as a guest. |
| 1.1. They will have to register a username and password. | Yes | If the user selects ‘Register’ a form is produced asking for a username and password. |
| 1.1.1. All usernames are stored in a database. This username must be checked against those in the database to make sure it is unique. This database will be accessed using SQL. | Yes | The SQL statement ‘SELECT username FROM user\_details’ is used to retrieve all usernames and check them against the entered username. |
| 1.1.2. If the username is taken the user should be alerted. | Yes | If the entered username matches any of the already existing usernames, the message ‘That username is already in use. Please pick another.’ is displayed and the program does not continue. |
| 1.1.3. The password will also be stored in the database. It should be encrypted using a hashing algorithm instead of being stored simply as plaintext. | Yes | The password is hashed using the SHA-256 hashing algorithm. When logging in, the entered password is also hashed using this algorithm, and the program checks whether this result matches the hashed password in the database in order to authenticate the user. |
| 1.2. Once the account has been successfully created (or a user with an already existing account has logged in) they will be directed to the main menu. | Yes | A message ‘Successfully registered \_\_\_’ (if registering) or ‘Welcome \_\_\_’ (if logging in) is displayed before the menu is produced. |
| 2. The main menu will allow the user to select which mode they wish to play. | Yes | There are buttons to play ‘Maze Game’ or ‘A\* Simulation’, or to sign out. |
| 2.1. There will be a button for the maze game and a button for the A\* simulator. | Yes |
| 2.2. They will also be able to sign out from this screen, returning to the first screen. | Yes | Pressing ‘Sign Out’ logs out the current user, reproducing the first page. |
| 3. In the maze section: | | |
| 3.1. The user will select a difficulty (easy, medium or hard). | Yes | A page is created with radio buttons allowing the user to select a difficulty. |
| 3.2. A maze will be generated using the randomized depth-first search algorithm, its size depending on the difficulty selected in objective 3.1. | Yes | Easy – 5\*5 maze, Medium – 10\*10 maze, Hard – 15\*15 maze. The Default theme is used immediately after difficulty selection. |
| 3.3. The user can then play the maze, using the arrow keys to navigate a simple sprite from the start cell to the target cell. | Yes | The user can use the arrow keys to move the sprite across the maze, in any direction not blocked by a wall. |
| 3.3.1. Instructions to use the arrow keys to play should be clearly displayed. | Yes | The message ‘Use the arrow keys to move.’ is displayed in the top right. |
| 3.3.2. When the sprite moves a trail should be displayed showing where it has been. | Yes | When the sprite crosses a cell, that cell’s colour changes to the colour of the trail. If it backtracks, the cells remain this new colour, therefore indicating all the cells the sprite has visited. In the Neon theme the trail is four colours instead of one. |
| 3.3.3. There will be a stopwatch displayed on this screen so the user can see the current time taken. | Yes | The stopwatch increments until the user completes the maze, creates a new maze, or it reaches ’60:00’. |
| 3.4. When the user completes the maze, a congratulations message will be produced, which states the time taken and the user’s current high score. | Yes | This information is displayed in a new window. If playing as a guest, the high score is displayed as ‘XX:XX’. |
| 3.4.1. The user’s fastest time (on this difficulty) is taken as their highest score and saved to the database. | Yes | This is achieved using the SQL statement ‘'UPDATE user\_details SET easy\_high\_score = %s WHERE username = %s'. If it is the user’s first attempt at this difficulty, their score is automatically saved to the database and displayed as their high score on the congratulations page. If they attempt the difficulty again and don’t beat their score the high score in the database is displayed instead; if they do the new score is saved and displayed. |
| 3.4.2. Should the user beat their high score, this will be updated in the database and the new score will be displayed the next time they beat a maze of the same difficulty. | Yes |
| 3.5. There will be an option to see the answer. When clicked the A\* algorithm will be used and the shortest route through the maze will be displayed. | Yes | The cells that are part of the solution become the colour of the trail. |
| 3.5.1. No time will be recorded for this attempt. Instead the user must click the play again button (see objective 3.6) to start a new maze. | Yes | The user is still able to navigate the maze, however the stopwatch is frozen and the congratulations page will not be created if they reach the target cell. |
| 3.6. There will also be options to play again, change difficulty level or return to the menu. | Yes | The button ‘New Maze’ generates a new maze, resetting the stopwatch and the colour of the cells and returning the sprite to the start cell. Buttons ‘Difficulty’ and ‘Menu’ return the user to the difficulty selection screen or menu respectively. |
| 3.7. There will be a customisation option which allows the user to select a colour theme for the maze. | Yes | The button ‘Themes’ produces a window allowing the user to select a colour theme. When they choose one the entire maze window is recreated using the new theme. The congratulations and themes pages are also coloured using this new theme. |
| 4. In the A\* section: | | |
| 4.1. An empty grid will be displayed. | Yes | The grid is 24\*10, filled with light grey empty cells. |
| 4.2. At the bottom of the screen the user can select what to place with their mouse: | Yes | There are six buttons, each containing the sprites for each object. There is a label above each (‘wall’, ‘start’, ‘target’, ‘puddle’, ‘portal in’, ‘portal out’). The user can click one of these buttons and then on the grid to add this object to the grid. |
| 4.2.1. A solid wall. | Yes |
| 4.2.2. A start node (only one). | Yes |
| 4.2.3. An end node (only one). | Yes |
| 4.2.4. An obstacle which takes longer to cross than an empty cell. | Yes |
| 4.2.5. A warp panel that moves the frontier across the grid. | Yes |
| 4.3. Once the user has drawn their maze, they can select the start button. | Yes | The button is titled ‘Run A\*’. |
| 4.3.1. If a start and end node haven’t been placed the user should be notified. | Yes | The message ‘The grid is missing a start cell’ or ‘The grid is missing a target cell’ is produced once the ‘Run A\*’ button is clicked. |
| 4.3.2. The A\* algorithm will then run, changing the colour of each cell it visits. There will be a short delay so that the user can see the changes occurring. | Yes | The algorithm is cycled through with a delay, changing the colour of cells: those in the frontier become light blue and those that have been visited become dark blue. |
| 4.3.3. When the shortest path has been found it will be highlighted in another colour. | Yes | Cells in the solution are drawn in yellow (still with a time delay so the user can see it form). |
| 4.4. There should be an option to clear the grid. | Yes | Pressing the ‘Clear’ button will revert all cells back to empty cells. Colours are reverted back to grey. |
| 4.5. There should be a button containing information about this section of the software: | Yes | The button is titled ‘What’s going on?’ |
| 4.5.1. How to interact with the area (place objects etc.) | Yes | This is explained within the information page produced by pressing ‘What’s going on?’ One side of the page explains how to use this section, the other goes into detail about the A\* algorithm and how it changes the colours of cells. Additionally, there is an extra page discussing how the algorithm interacts with portals, so any curious or confused users can learn more. |
| 4.5.2. What the A\* algorithm is and does (finds the shortest path from one node to another) | Yes |
| 4.5.3. How the algorithm works (explaining how the cost for each cell is calculated etc.) | Yes |
| 4.6. There should also be an option to return to the menu. | Yes | There is a button titled ‘Menu’, which returns to the menu screen. |

## 5.4 End-user evaluation

### 5.4.1 Feedback from my end user

I again met with Noah to gain their feedback on my finished program. Unfortunately the device we were using was unable to connect to my database, so they were limited to using the play as guest feature; I did however show them screenshots from when I have used the login system successfully. They were able to use the rest of the program using the guest account.

After they had spent some time playing with my program, I asked them the following questions:

**How well does the program meet its aim, to produce a system which allows the user to interact with the A\* algorithm and apply it to a maze setting?**

‘I think that the program meets this aim very well. It is easy to learn about the algorithm in the second section – the information page is interesting and very clear – and then compare it with my own approach to solving a maze, both by looking at the solution to a maze I have created while the algorithm explores it, and by testing my own methods in the maze section. However, only I can play the computer-generated mazes. Allowing the algorithm to solve these mazes too would give an even easier comparison.’

**Did the program meet your initial expectations? Were your comments addressed?**

‘Yes. I was impressed with how well the program followed its design, and all of the important features we discussed were included and work well. I was pleased to see the play as guest feature I had suggested made it into the program. My favourite feature is definitely the range of colour themes that the user can choose from. I especially love the Neon theme: it looks really cool and I love how the colours in the trail form a pattern on the grid!’

**How well does the program work as a whole? Is it easy to understand and navigate?**

‘I found the program really simple to use. There aren’t any unnecessary extras and the overall layout is intuitive. Also, the instructions “Use the arrow keys to move” are clearly displayed in the maze section.’

**How well does the account system work?**

‘I was only able to use the guest account, but that worked well. I was still able to see my time taken on the current maze, so I could remember a few of my scores anyway. But I think that having an account would provide more of an incentive to beat my high scores, because I could see just how close I came to my previous score, and record my progress. From the screenshots you showed me, however, the login system looks easy to navigate.’

**Is the maze section fun to use? Does it work as you would expect?**

‘The maze game is definitely fun and very addictive! Using the arrow keys works really well – the controls couldn’t be simpler. As a whole the section is pretty self-explanatory: the start and end cells being bright green and red flags make it easy to see what the aim is.’

**Is the A\* section effective? Is it easy to use? Are the information pages improved from my initial design?**

‘The A\* section is laid out well and it is easy to select objects and add them to the grid. I found the visualisation very satisfying to watch, especially after I’d looked at the information page to see what the colours meant. Having a colour key to explain what each object does was really helpful. I also liked how there were some fun little comments in there, making it less of a drag to read. The page about the portal problem is engaging, because there was a challenge to try to create that scenario, so I had to try it for myself!’

**What features could I have added to improve the system?**

‘I liked the design of the character in the maze; however I would love to have some customisation options for it. For the maze section, a mode where the walls were invisible would be really challenging. And for the A\* section, it would be very fun to have multiple portals on the grid at once.’

**Do you have anything else to add?**

‘I really enjoyed using the system, especially trying to beat my high score in each maze difficulty! I liked that the mazes were random, so there would never be any repeats. After playing the maze, it was really interesting to learn about the mechanics behind how it is solved in the A\* section.’

### 5.4.2 Analysis of feedback

It is evident from this feedback that Noah enjoyed using the program, and felt that it was intuitive, educational and fun to use – all qualitites that I had hoped to achieve to make the system suitable for my target audience. They praised the customisablity options and remarked that the game was addictive, even despite not being able to use an account to save scores.

They suggested that the A\* algorithm could also be used to solve mazes created using the randomised depth first search, which matched pretty well with my idea of reusing the visualisation within these mazes. This would be pretty feasible to implement in an extension of my project. Alternatively, Noah’s idea could be met by including a maze generation feature in the A\* section, similarly to how the grid from the PathVis site could have patterns generated onto it. However, I think this wouldn’t give as clear a comparison between the user and the algorithm.

Noah commented that the maze section was very self explanatory. When we met to discuss the design of the program they explicitly mentioned adding a start and end flag to the maze to aid with this, and it appears from their feedback that they are happy with the result.

They were also clearly impressed with the changes I’d made to the information page in the A\* section, especially the coloured key which helped them to use the visualisation properly.

They also suggested some additional features to add to the user’s enjoyment: character customisation, a maze with invisible walls, and extra portals in the A\* section. While I believe these wouldn’t be essential additions – they wouldn’t really contribute to the overall performance of the system to solve my problem – they would be fun additions from a user perspective. Character customisation would simply require a new character selection page and some more sprite designs. I could hide the walls easily by simply not using the *drawWalls()* subroutine. As for more portals however, I would need to come up with a way of distinguishing between each portal in/portal out pair, and ensuring that the correct portal out is chosen depending on the current portal in, which would be more difficult (and quite chaotic, in my opinion).

### 5.4.3 Other suggestions

While Noah was my main end user for this project, some of my other friends were able to see my program while I was developing and testing it. A few suggestions were raised for further improvements, which I will discuss here.

* ‘Only start the stopwatch when the user makes their first move’ (see Section [5.1](#_5.1_Whole_system)). This would be pretty simple to do using a Boolean flag.
* ‘Make the walls more visible, especially in the Neon theme’. Due to the trail using four different bright colours and highlighting the cells of the grid, the walls can become difficult to see. I would have to increase the width of the lines to improve this. (Noah didn’t seem to mind however – they even suggested making it a feature!)
* ‘Let the user drag the mouse across the A\* grid to draw walls’. I would need to research how Tkinter handles the mouse being dragged in order to implement this.

Of course, this feedback wasn’t supposed to be formal or suggest groundbreaking changes to the program, just small features to increase convenience.

## 5.5 Conclusion

It is clear from Noah’s feedback that my program is a useful and effective tool, allowing users to learn about the A\* algorithm through visualisation and apply it to a maze scenario, while also being able to play mazes themselves and compare their approach to the algorithm’s. I believe that the program met all of its project objectives, having exceeded them in some areas. Some extensions were suggested in order to improve the system, for user convenience and enjoyment. Also, Noah and I both seemed to agree that I could have included the visualisation in the Maze Game section, to make it clearer that this algorithm is being used to solve the maze, and allow for further comparison between user and algorithm. I hope to now be able to share my program with other students so that they can explore the A\* algorithm for themselves.

# 6 References

## 6.1 Bibliography

PathViz. (n.d.). *Pathfinding Visualiser*. Retrieved from PathVis: https://pathvis.datsko.dev/

Pierce, R. (2020, July 20). *Play Mazes Online or Print Them*. Retrieved from Math Is Fun: https://www.mathsisfun.com/measure/mazes.html

Wikipedia. (2021). *Maze*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Maze#Maze\_construction

## 6.2 Table of Figures

[Figure 1: A simple maze (Wikipedia, 2021). 5](file:///E:\KS5\To%20Do\Computer%20Science\NEA\Report%20Versions\Version%2020%20done%20maybe.docx#_Toc98757728)

[Figure 2: An example of a graph that could represent a maze. 5](file:///E:\KS5\To%20Do\Computer%20Science\NEA\Report%20Versions\Version%2020%20done%20maybe.docx#_Toc98757729)

[Figure 3: The A\* algorithm includes the distance from both the start node and the end node in its calculations for each node. 6](file:///E:\KS5\To%20Do\Computer%20Science\NEA\Report%20Versions\Version%2020%20done%20maybe.docx#_Toc98757730)

[Figure 4: Screenshot from the MathIsFun maze platform (Pierce, 2020). 9](file:///E:\KS5\To%20Do\Computer%20Science\NEA\Report%20Versions\Version%2020%20done%20maybe.docx#_Toc98757731)

[Figure 5: Screenshots from the PathViz platform during (top) and after (bottom) an A\* visualisation (PathViz, n.d.). 11](file:///E:\KS5\To%20Do\Computer%20Science\NEA\Report%20Versions\Version%2020%20done%20maybe.docx#_Toc98757732)

[Figure 6: A flowchart of the randomised depth-first search algorithm. 14](file:///E:\KS5\To%20Do\Computer%20Science\NEA\Report%20Versions\Version%2020%20done%20maybe.docx#_Toc98757733)

[Figure 7: A flowchart of the A\* algorithm. 16](file:///E:\KS5\To%20Do\Computer%20Science\NEA\Report%20Versions\Version%2020%20done%20maybe.docx#_Toc98757734)

[Figure 8: A screenshot of the output from the prototype, using 85 and 63 as target cells. 17](file:///E:\KS5\To%20Do\Computer%20Science\NEA\Report%20Versions\Version%2020%20done%20maybe.docx#_Toc98757735)

[Figure 9: Testing the prototype after cell 55 was removed. 18](file:///E:\KS5\To%20Do\Computer%20Science\NEA\Report%20Versions\Version%2020%20done%20maybe.docx#_Toc98757736)

[Figure 10: A hierarchy diagram of the system as a whole. 22](file:///E:\KS5\To%20Do\Computer%20Science\NEA\Report%20Versions\Version%2020%20done%20maybe.docx#_Toc98757737)

[Figure 11: A design of the first page of User Account Management. 23](file:///E:\KS5\To%20Do\Computer%20Science\NEA\Report%20Versions\Version%2020%20done%20maybe.docx#_Toc98757738)

[Figure 12: A design of the User Account Management login form. 24](file:///E:\KS5\To%20Do\Computer%20Science\NEA\Report%20Versions\Version%2020%20done%20maybe.docx#_Toc98757739)

[Figure 13: Data table recording user credentials and high scores 24](#_Toc98757740)

[Figure 14: A Data Flow Diagram illustrating when data is passed to and from the database. 25](#_Toc98757741)

[Figure 15: A design for the Menu. 27](file:///E:\KS5\To%20Do\Computer%20Science\NEA\Report%20Versions\Version%2020%20done%20maybe.docx#_Toc98757742)

[Figure 16: A design for the Maze Game difficulty selection page. 28](file:///E:\KS5\To%20Do\Computer%20Science\NEA\Report%20Versions\Version%2020%20done%20maybe.docx#_Toc98757743)

[Figure 17: A sketch of the maze from the Maze Game main page. 29](file:///E:\KS5\To%20Do\Computer%20Science\NEA\Report%20Versions\Version%2020%20done%20maybe.docx#_Toc98757744)

[Figure 18: A design for the main Maze Game page. 30](file:///E:\KS5\To%20Do\Computer%20Science\NEA\Report%20Versions\Version%2020%20done%20maybe.docx#_Toc98757745)

[Figure 19: A design of the Maze Game congratulations page. 31](file:///E:\KS5\To%20Do\Computer%20Science\NEA\Report%20Versions\Version%2020%20done%20maybe.docx#_Toc98757746)

[Figure 20: A design of the main A\* Simulation page. 34](file:///E:\KS5\To%20Do\Computer%20Science\NEA\Report%20Versions\Version%2020%20done%20maybe.docx#_Toc98757747)

[Figure 21: A design for the A\* Simulation information page. 35](file:///E:\KS5\To%20Do\Computer%20Science\NEA\Report%20Versions\Version%2020%20done%20maybe.docx#_Toc98757748)

[Figure 22: A sketch of the objects used in A\* Simulation. 39](file:///E:\KS5\To%20Do\Computer%20Science\NEA\Report%20Versions\Version%2020%20done%20maybe.docx#_Toc98757749)

[Figure 23: A sketch of the A\* Simulation running (left) and complete (right). 40](file:///E:\KS5\To%20Do\Computer%20Science\NEA\Report%20Versions\Version%2020%20done%20maybe.docx#_Toc98757750)

[Figure 24: The attributes and methods of the Cell class. 42](#_Toc98757751)

[Figure 25: The attributes of each of Cell's subclasses. 43](#_Toc98757752)

[Figure 26: Class diagram containing Cell and its subclasses. 44](file:///E:\KS5\To%20Do\Computer%20Science\NEA\Report%20Versions\Version%2020%20done%20maybe.docx#_Toc98757753)

[Figure 27: The attributes and methods of the Sprite class. 44](#_Toc98757754)

[Figure 28: The attributes and methods of the Stopwatch class. 45](#_Toc98757755)

[Figure 29: The attributes and methods of the StopwatchThread class. 46](#_Toc98757756)

[Figure 30: The first page of the program, allowing me to choose to register, sign in or play as guest. 56](#_Toc98757757)

[Figure 31: The form allowing me to register my account. 57](#_Toc98757758)

[Figure 32: The message produced when I clicked ‘Submit’. 57](file:///E:\KS5\To%20Do\Computer%20Science\NEA\Report%20Versions\Version%2020%20done%20maybe.docx#_Toc98757759)

[Figure 33: The menu page, allowing me to pick which option to play. My username is displayed at the top. 58](#_Toc98757760)

[Figure 34: The difficulty selection page, with Easy selected. 59](#_Toc98757761)

[Figure 35: A 5\*5 maze is produced, with a clear sprite, start cell and target cell. I have moved the sprite so that the trail can be clearly seen. 60](#_Toc98757762)

[Figure 36: The congratulations page, with my time taken, Easy high score and a message to say that this is a new high score. 61](#_Toc98757763)

[Figure 37: The congratulations page, but this time I did not achieve a new high score. 62](#_Toc98757764)

[Figure 38: The solution to a Hard maze, produced when I click the ‘Solution’ button. 63](#_Toc98757765)

[Figure 39: The A\* Simulation grid in its initial state. 63](file:///E:\KS5\To%20Do\Computer%20Science\NEA\Report%20Versions\Version%2020%20done%20maybe.docx#_Toc98757766)

[Figure 40: A simple maze I created for the A\* algorithm to solve. 64](file:///E:\KS5\To%20Do\Computer%20Science\NEA\Report%20Versions\Version%2020%20done%20maybe.docx#_Toc98757767)

[Figure 41: Partway through a demonstration of the A\* algorithm solving my maze. 65](file:///E:\KS5\To%20Do\Computer%20Science\NEA\Report%20Versions\Version%2020%20done%20maybe.docx#_Toc98757768)

[Figure 42: The solution to my maze was highlighted in yellow. 66](file:///E:\KS5\To%20Do\Computer%20Science\NEA\Report%20Versions\Version%2020%20done%20maybe.docx#_Toc98757769)

[Figure 43: The information page produced by pressing the ‘What's going on?’ button. 67](file:///E:\KS5\To%20Do\Computer%20Science\NEA\Report%20Versions\Version%2020%20done%20maybe.docx#_Toc98757770)

[Figure 44: An additional page explaining a quirk of using portals with the A\* algorithm. 68](file:///E:\KS5\To%20Do\Computer%20Science\NEA\Report%20Versions\Version%2020%20done%20maybe.docx#_Toc98757771)

[Figure 45: The error message produced when I open the program on a device not connected to my database. 84](file:///E:\KS5\To%20Do\Computer%20Science\NEA\Report%20Versions\Version%2020%20done%20maybe.docx#_Toc98757772)

[Figure 46: The error message produced if the user presses ‘Submit’ without inputting a username. 85](file:///E:\KS5\To%20Do\Computer%20Science\NEA\Report%20Versions\Version%2020%20done%20maybe.docx#_Toc98757773)

[Figure 47: The message produced when I successfully sign in to an account. 86](file:///E:\KS5\To%20Do\Computer%20Science\NEA\Report%20Versions\Version%2020%20done%20maybe.docx#_Toc98757774)

[Figure 48: The themes page, allowing the user to select a colour theme for the maze. 87](file:///E:\KS5\To%20Do\Computer%20Science\NEA\Report%20Versions\Version%2020%20done%20maybe.docx#_Toc98757775)

[Figure 49: A medium maze in the Blue theme. 88](file:///E:\KS5\To%20Do\Computer%20Science\NEA\Report%20Versions\Version%2020%20done%20maybe.docx#_Toc98757776)

[Figure 50: A medium (10\*10) maze using the Default theme. 88](file:///E:\KS5\To%20Do\Computer%20Science\NEA\Report%20Versions\Version%2020%20done%20maybe.docx#_Toc98757777)

[Figure 51: A medium maze in the Pink theme. 89](file:///E:\KS5\To%20Do\Computer%20Science\NEA\Report%20Versions\Version%2020%20done%20maybe.docx#_Toc98757778)

[Figure 52: A medium maze in the Green theme. 89](file:///E:\KS5\To%20Do\Computer%20Science\NEA\Report%20Versions\Version%2020%20done%20maybe.docx#_Toc98757779)

[Figure 53: A medium maze in the Neon theme. 90](file:///E:\KS5\To%20Do\Computer%20Science\NEA\Report%20Versions\Version%2020%20done%20maybe.docx#_Toc98757780)

[Figure 54: The congratulations page when using the guest account, using the Blue theme. 91](file:///E:\KS5\To%20Do\Computer%20Science\NEA\Report%20Versions\Version%2020%20done%20maybe.docx#_Toc98757781)

[Figure 55: The result when the user creates an impossible grid in A\* Simulation. 92](file:///E:\KS5\To%20Do\Computer%20Science\NEA\Report%20Versions\Version%2020%20done%20maybe.docx#_Toc98757782)

# 7 Appendix

## 7.1 Appendix A – Prototype of the A\* algorithm

from queue import PriorityQueue

def coordinates(cell, size):

     '''Given a cell's index number calculate its x and y coordinates, starting from the top left'''

     x = cell % size

     y = cell // size +1

     if x == 0:

        x = size  #eg 30 % 10 is 0, but should be placed in row 10

        y = y -1  # 30 // 10 +1 is 4, but should be placed in row 3

     #endif

     return x, y

#endsub

def calculateNeighbours(cell, size):

    '''Given a cell's index number, work out its neighbouring cells'''

    neighbours = []

    north = cell - size

    neighbours = checkNeighbour(north, size, neighbours)

    if cell % size !=0: #there is no east neighbour for the rightmost column

        east = cell + 1

        neighbours = checkNeighbour(east, size, neighbours)

    #endif

    south = cell + size

    neighbours = checkNeighbour(south, size, neighbours)

    if (cell -1) % size != 0 and cell !=1: #there is no west neighbour for the leftmost column

        west = cell -1

        neighbours = checkNeighbour(west, size, neighbours)

    #endif

    return neighbours

#endsub

def checkNeighbour(n, size, neighboursList):

    '''Check if a neighbour is valid (does not fall outside the grid)'''

    if n <=0 or n > size\*\*2:

        return neighboursList #do nothing if the neighbour is invalid

    else:

        neighboursList.append(n)

        return neighboursList

    #endif

#endsub

def heuristic(startx, starty, endx, endy):

     '''Calculates the heuristic used to guess the distance to the target cell using Manhattan Distance'''

     return (abs(startx - endx) + abs(starty - endy)) #the sum of the differences between the current and target cell

#endsub

def output(a, cameFrom):

     '''Produce the shortest path calculated in the A\* algorithm'''

     print(a)

     while cameFrom[a] != None:

          print(cameFrom[a])

          a = cameFrom[a]

     #endwhile

#endsub

def AStar(size):

     '''Use the A\* algorithm to find the shortest route to the target cell in a grid'''

     frontier = PriorityQueue()

     frontier.put(1, 0)

     totalCosts = {} # (cell number, cost to cell)

     costsFromStart = {} # (cell number, cost from start)

     cameFrom = {} #  (cell number, previous cell number)

     cameFrom[1] = None

     currentCell = frontier.get()

     targetCell = 85

     targetCellx, targetCelly = coordinates(targetCell, size)

     while currentCell != targetCell:

        neighbours = calculateNeighbours(currentCell, size)

        for checkingNeighbour in neighbours: #checks each neighbour individually

                neighbourx, neighboury = coordinates(checkingNeighbour, size)

                if checkingNeighbour not in cameFrom:

                    if currentCell == 1:

                         costSoFar = 1

                    else:

                         costSoFar = costsFromStart[currentCell] + 1

                    #endif

                    costToNeighbour = costSoFar + heuristic(neighbourx, neighboury, targetCellx, targetCelly)

                    if checkingNeighbour not in totalCosts or costToNeighbour < totalCosts[checkingNeighbour]:

                         totalCosts[checkingNeighbour] = costToNeighbour

                         costsFromStart[checkingNeighbour] = costSoFar

                         frontier.put((costToNeighbour, checkingNeighbour)) #use the cell's cost as its priority

                         cameFrom[checkingNeighbour] = currentCell

                    #endif

                #endif

#endif

        #endfor

        currentCell = (frontier.get())[1]

     #endwhile

     output(targetCell, cameFrom)

#endsub

AStar(10) #calls the algorithm using a 10x10 grid

## 7.2 Appendix B – Technical Solution

Refer to Section [3.1](#_3.1_Code_contents) for a contents table, or Section [3.2](#_3.2_Achieved_objectives) for where each of my objectives have been met within my code. This is the code from before I began testing, so the changes discussed in Section [4.2.5](#_4.2.5_Fixing_failed) are not present here.

### 7.2.1 Class definitions

Program name: ‘**cellClasses**’

Contains the Cell class and its subclasses: StartCell, EndCell, Wall, Puddle, WarpIn and WarpOut, used by both Maze Game and A\* Section.

#The Cell class, from which grids can be created, and its subclasses, different objects for the A\* section.

from tkinter import PhotoImage

class Cell:

    def \_\_init\_\_(self, x, y, height, width, theme):

        self.\_\_xy = (x, y)

        self.\_visited = False

        self.\_\_neighbours = self.\_\_calculateNeighbours(height, width)

        self.\_\_edges = self.\_\_calculateEdges()

        self.\_passages = []  #used when checking if the sprite can move here in Maze Game

        self.\_colour = theme['background']

        self.\_type = 'empty' #determines which kind of cell a particular Cell object is

        self.\_cost = 1 #cost to cross this cell

    #endsub

    def getxy(self):

        return self.\_\_xy

    #endsub

    def getVisited(self):

        return self.\_visited

    #endsub

    def setVisited(self, new):

        self.\_visited = new

    #endsub

    def getNeighbours(self):

        return self.\_\_neighbours

    #endsub

    def getEdges(self):

        return self.\_\_edges

#endsub

    def getPassages(self):

        return self.\_passages

    #endsub

    def getColour(self):

        return self.\_colour

    #endsub

    def setColour(self, new):

        self.\_colour = new

    #endsub

    def getType(self):

        return self.\_type

    #endsub

    def getCost(self):

        return self.\_cost

    #endsub

    def \_\_calculateNeighbours(self, height, width):

        '''Given a cell's coordinates, return a list of its neighbouring cells'''

        directions = [[0,1], [1,0], [0,-1], [-1,0]]

        neighbours = []

        for direction in directions:

            neighbour = [self.\_\_xy[0] + direction[0], self.\_\_xy[1] +direction[1]]

            if -1<neighbour[0] <width and -1<neighbour[1]<height:

                neighbours.append(neighbour) #only add the neighbour if it falls inside the grid.

            #endif

        #endfor

        return neighbours

    #endsub

    def \_\_calculateEdges(self):

        '''Calculate the coordinates of the four edges of this cell'''

        edge1 = ((self.\_\_xy[0]-0.5, self.\_\_xy[1]-0.5), (self.\_\_xy[0]+0.5, self.\_\_xy[1]-0.5))

        edge2 = ((self.\_\_xy[0]-0.5, self.\_\_xy[1]-0.5), (self.\_\_xy[0]-0.5, self.\_\_xy[1]+0.5))

        edge3 = ((self.\_\_xy[0]+0.5, self.\_\_xy[1]-0.5), (self.\_\_xy[0]+0.5, self.\_\_xy[1]+0.5))

        edge4 = ((self.\_\_xy[0]-0.5, self.\_\_xy[1]+0.5), (self.\_\_xy[0]+0.5, self.\_\_xy[1]+0.5))

        edges = [edge1, edge2, edge3, edge4]

        return edges

    #endsub

#endclass

class StartCell(Cell):

    def \_\_init\_\_(self, x, y, height, width, theme):

        super().\_\_init\_\_(x, y, height, width, theme)

        self.\_type = 'start'

        self.\_colour = PhotoImage(file = 'startcell.png')

    #endsub

#endclass

class EndCell(Cell):

    def \_\_init\_\_(self, x, y, height, width, theme):

        super().\_\_init\_\_(x, y, height, width, theme)

        self.\_type = 'end'

        self.\_colour = PhotoImage(file = 'endcell.png')

    #endsub

#endclass

class Wall(Cell):

    def \_\_init\_\_(self, x, y, height, width, theme):

        super().\_\_init\_\_(x, y, height, width, theme)

        self.\_type = 'wall'

        self.\_colour = '#000000'

    #endsub

#endclass

class Puddle(Cell):

    def \_\_init\_\_(self, x, y, height, width, theme):

        super().\_\_init\_\_(x, y, height, width, theme)

        self.\_type = 'puddle'

        self.\_colour = PhotoImage(file = 'puddle.png')

        self.\_cost = 5

    #endsub

#endclass

class WarpIn(Cell):

    def \_\_init\_\_(self, x, y, height, width, theme):

        super().\_\_init\_\_(x, y, height, width, theme)

        self.\_type = 'warp in'

        self.\_colour = PhotoImage(file = 'warpin.png')

    #endsub

#endclass

class WarpOut(Cell):

    def \_\_init\_\_(self, x, y, height, width, theme):

        super().\_\_init\_\_(x, y, height, width, theme)

        self.\_type = 'warp out'

        self.\_colour = PhotoImage(file = 'warpout.png')

    #endsub

#endclass

Program name: ‘**spriteClass**’

Contains the Sprite class used to model the sprite controlled by the user in Maze Game.

class Sprite():

    '''The sprite that the user navigates through the maze'''

    def \_\_init\_\_(self):

        self.\_xy = (0, 0)

        self.\_cell = () #the cell the sprite is currently on

    #endsub

    def getxy(self):

        return self.\_xy

    #endsub

    def setxy(self, new):

        self.\_xy = new

    #endsub

    def getCell(self, cellsList, allCells):

        '''Find the Cell object the sprite is on based on its coordinates'''

        cellPosition = cellsList.index([self.\_xy[0], self.\_xy[1]])

        return allCells[cellPosition]

    #endsub

    def canMove(self, direction, cellsList, allCells):

        '''Test whether the sprite can move in a given direction'''

        move = False

        if direction == 'up':

            movement = (0, -1)

        elif direction == 'down':

            movement = (0, 1)

        elif direction == 'left':

            movement = (-1, 0)

        elif direction == 'right':

            movement = (1, 0)

        #endif

        currentCell = self.\_xy

        newCell = (currentCell[0] + movement[0], currentCell[1] + movement[1]) #adds the direction to the current cell coordinates

        newPassage = (currentCell, newCell)

        self.\_cell = self.getCell(cellsList, allCells)

        if newPassage in self.\_cell.getPassages():

            move = True

        #endif

        newPassage = (newCell, currentCell) #the passage is traversable in both directions, but only one will be listed in the cell's list of passages

        if newPassage in self.\_cell.getPassages():

            move = True

        #endif

        if move == True:

            self.\_xy = newCell

        return move, movement

    #endsub

#endclass

Program name: ‘**stopwatchClass**’

Contains the Stopwatch class used to keep track of the user’s time in Maze Game.

class Stopwatch():

    '''The stopwatch used to record the time taken to complete the maze'''

    def \_\_init\_\_(self):

        self.\_total = -1 #total number of seconds that have passed. -1 means it immediately increments to 0

        self.\_seconds = 0

        self.\_minutes = 0

        self.\_display = '00:00'

    #endsub

    def getTotal(self):

        return self.\_total

    #endsub

    def getMinutes(self):

        return self.\_minutes

    #endsub

    def getDisplay(self):

        return self.\_display

    #endsub

    def increment(self):

        '''Add 1 second'''

        self.\_total+=1

        self.\_minutes = self.\_total // 60

        self.\_seconds = self.\_total % 60

        self.\_updateDisplay()

    #endsub

    def \_updateDisplay(self):

        '''Change the display that will be seen by the user'''

        secs = str(self.\_seconds)

        if len(secs) == 1:

            secs = '0'+secs

        #endif

        mins = str(self.\_minutes)

        if len(mins) == 1:

            mins = '0'+mins

        #endif

        self.\_display = '{0}{1}:{2}{3}'.format(mins[0], mins[1], secs[0], secs[1])

    #endsub

    def reset(self):

        '''Reset the stopwatch back to "00:00"'''

        self.\_\_init\_\_()

    #endsub

#endclass

### 7.2.2 Key Algorithms

Program name: ‘**randomisedDepthFirstSearch**’

The recursive randomised depth-first search graph traversal algorithm used to generate a maze in Maze Game.

#The randomised depth first search algorithm used to create a maze

import random

from cellClasses import Cell, StartCell, EndCell

def depthFirstRecursion(current, allCells, cellsList, totalPassages):

    '''Create a list of passages by using the randomised depth first search algorithm on the grid'''

    current.setVisited(True)

    neighbours = current.getNeighbours()

    random.shuffle(neighbours) #hence the algorithm is randomised so a maze can be created

    for neighbour in neighbours:

        position = cellsList.index(neighbour)

        neighbour = allCells[position]

        if neighbour.getVisited() == False:

            #add the passage between these cells to each cell's passage list and the list of all passages (forming an adjacency list)

            currentxy = current.getxy()

            neighbourxy = neighbour.getxy()

            passage = (currentxy, neighbourxy)

            currentPassages = current.getPassages()

            neighbourPassages = neighbour.getPassages()

            currentPassages.append(passage)

            neighbourPassages.append(passage)

            totalPassages.append(passage)

            totalPassages = depthFirstRecursion(neighbour, allCells, cellsList, totalPassages) #the algorithm is then recalled using the neighbour cell. It unwinds once all of a cell's neighbours have already been visited.

        #endif

    #endfor

    return totalPassages

#endsub

def main(size, theme):

    '''Initialise the grid ready to pass into the recursive subroutine'''

    allCells = [] #allCells is a list of Cell objects for each cell in the grid

    cellsList=[] #cellsList is a list of these cells in coordinate form (x, y), preventing multiple instances of the same cell from being created

    for x in range(size):

        for y in range(size):

            cellsList.append ([x, y])

            if x==0 and y==0:

                allCells.append(StartCell(x, y, size, size, theme))

            elif x==size-1 and y==size-1:

                allCells.append(EndCell(x, y, size, size, theme))

            else:

                allCells.append(Cell(x, y, size, size, theme))

            #endif

        #endfor

    #endfor

    current = allCells[0] #the start cell

    totalPassages = []

    totalPassages = depthFirstRecursion(current, allCells, cellsList, totalPassages)

    return(totalPassages, allCells, cellsList)

#endsub

Program name: ‘**AStarAlgorithm**’

The A\* shortest path algorithm used to solve the maze in Maze Game and customised and visualised in A\* Section.

#The A\* pathfinding algorithm used to find the shortest route from start cell to end cell

from queue import PriorityQueue

def heuristic(currentCell, targetCell):

    '''Calculates the heuristic used to guess the distance to the target cell using Manhattan Distance'''

    currentxy = currentCell.getxy()

    targetxy = targetCell.getxy()

    return (abs(currentxy[0] - targetxy[0]) + abs(currentxy[1] - targetxy[1])) #manhattan distance formula

#endsub

def calculateRoute(cell, cameFrom):

    '''Retrieve the shortest route by tracing back from target cell to start cell'''

    shortestRoute = [cell.getxy()]

    while cameFrom[cell] != None:

        shortestRoute.append(cameFrom[cell].getxy())

        cell = cameFrom[cell]

    #endwhile

    shortestRoute.reverse()#produces the route from start to target

    return shortestRoute

#endsub

def AStar(startCell, targetCell, allWalls, cellsList, allCells, warpOutCell):

    '''Use the A\* algorithm to find the shortest route to the target cell in a grid'''

    frontier = PriorityQueue() #The frontier is a priority queue of cells currently being checked

    frontier.put(startCell.getxy(), 1)

    totalCosts = {} #A dictionary of each cell and its total cost

    costsFromStart = {} #A dictionary of each cell and its cost from the start cell

    cameFrom = {startCell:None} #A dictionary of cells that have been visited and the cell directly before them

    colourChanges = [] #Records all cells which should have their colour changed, and to what colour, so that they can be updated in A\* Simulation only

    currentCell = frontier.get() #In order to sort the frontier, the cells must be in coordinate form, however to get the cell's neighbours it must be in Cell form. The two lines below convert from coordinate form to Cell form.

    position = cellsList.index([currentCell[0], currentCell[1]])

    currentCell = allCells[position]

    while currentCell != targetCell and currentCell != -1: #if it is impossible to reach the target the frontier becomes empty, so currentCell is set to -1

        #Calculate the cell's neighbours

        if currentCell.getType() != 'warp in':

            neighbours = currentCell.getNeighbours()

        else:

            neighbours = warpOutCell.getNeighbours()

        #endif

        for neighbour in neighbours: #check each neigbour individually

            #convert the neighbour's coordinates to its corresponding Cell object

            position = cellsList.index([neighbour[0], neighbour[1]])

            neighbour = allCells[position]

            if neighbour not in cameFrom: #don't check if this neighbour has already been visited

                #find the possible wall between these cells

                possibleWall = -1

                currentxy = currentCell.getxy()

                neighbourxy = neighbour.getxy()

                direction = (neighbourxy[0]-currentxy[0], neighbourxy[1] - currentxy[1])

                if direction == (1, 0):

                    possibleWall = ((currentxy[0]+0.5, currentxy[1] -0.5), (currentxy[0]+0.5, currentxy[1]+0.5))

                elif direction == (-1, 0):

                    possibleWall = ((currentxy[0]-0.5, currentxy[1] -0.5), (currentxy[0]-0.5, currentxy[1]+0.5))

                elif direction == (0, 1):

                    possibleWall = ((currentxy[0]-0.5, currentxy[1] +0.5), (currentxy[0]+0.5, currentxy[1]+0.5))

                elif direction == (0, -1):

                    possibleWall = ((currentxy[0]-0.5, currentxy[1] -0.5), (currentxy[0]+0.5, currentxy[1]-0.5))

                #endif

                if possibleWall not in allWalls: #test whether this wall is actually present

                    if currentCell == startCell:

                        costSoFar = 1 #the first cell has no previous cell

                    else:

                        costSoFar = costsFromStart[currentCell] + neighbour.getCost() # +5 for puddles, +1 for all other cells

                    #endif

                    #Calculate the total cost using the cost so far and a heuristic to guess the distance to the end cell

                    costToNeighbour = costSoFar + heuristic(neighbour, targetCell)

                    #update each dictionary and frontier

                    if neighbour not in totalCosts or costToNeighbour < totalCosts[neighbour]: #only use this total if it is lower than any previously recorded totals (or it wouldn't be the shortest path)

                        totalCosts[neighbour] = costToNeighbour

                        costsFromStart[neighbour] = costSoFar

                        frontier.put((int(costToNeighbour), neighbour.getxy())) #the cell's cost is used as its priority, so cells with the lowest cost are checked before those with a higher cost.

                        colourChanges.append((neighbour, '#33ccff')) #light blue as the neighbour has been added to the frontier

                        cameFrom[neighbour] = currentCell

                    #endif

                #endif

            #endif

        #endfor

        colourChanges.append((currentCell, '#0000e6')) #dark blue as the current cell has now been visited

        #retrieve the next cell to be checked from the frontier

        if frontier.empty() == False:

            currentCell = (frontier.get())[1]

            position = cellsList.index([currentCell[0], currentCell[1]])

            currentCell = allCells[position]

        else:

            currentCell = -1

        #endif

    #endwhile

    #retrieve the shortest path

    if currentCell != targetCell:

        return -1, colourChanges

    #endif

    return calculateRoute(targetCell, cameFrom), colourChanges

#endsub

### 7.2.3 Main program and Menu

Program name = ‘**mazeSolvingUsingAStar**’

The main program that is run by the user, which controls navigating through the program, contains all of the user interfaces for both User Account Management and Menu, as well as the difficulty selection page from Maze Game.

#Imports

import tkinter

from tkinter import messagebox

try:

    import mysql.connector

    databaseError = False

except ImportError:

    databaseError = True #once the first window is created the user can be warned that there was an error. Else an error message would be produced with a blank window. This global variable is unavoidable if I am to keep imports at the top of my file, as recommended by the PEP8 Python style standard.

#endtry

import userAccountManagement

import mazeGame

import AStarSimulation

#User Account Management

def displayLogin():

    '''Produce the display allowing the user to choose whether to register or sign in to an account.'''

    loginWindow = tkinter.Tk()

    loginWindow.title('Maze-solving using A\*')

    loginWindow.geometry('600x400')

    loginWindow.configure(bg='#ffffff')

    title = tkinter.Label(loginWindow, text = 'Maze-solving using A\*', bg= '#ffffff', font = '"Stencil" 35  underline', )

    title.place(relx= 0.06, rely= 0.13)

    registerButton = tkinter.Button(loginWindow, bg='#ff7070', activebackground='#dd5050', bd='5',  text= 'Register', font= '"OCR A Extended" 25', fg='#000000', padx='10', pady='20', justify='center', relief='raised', command=lambda: navigate('register', loginWindow, None))

    registerButton.place(relx = 0.07, rely = 0.4)

    signInButton = tkinter.Button(loginWindow, bg='#80ffb0', activebackground='#60dd90', bd= '5', text= 'Sign in', font= '"OCR A Extended" 25', fg='#000000', padx='20', pady= '20', justify='center', relief='raised', command=lambda: navigate('sign in', loginWindow, None))

    signInButton.place(relx = 0.57, rely = 0.4)

    guestButton = tkinter.Button(loginWindow, bg='#c5ffff', activebackground = '#a9f0f9',  bd= '5', text= 'Play as guest', font= '"OCR A Extended" 15', fg='#000000', padx='10', pady= '10', justify='center', relief='raised', command=lambda: navigate('menu', loginWindow, 'Guest'))

    guestButton.place(relx = 0.33, rely = 0.75)

    if databaseError:

        messagebox.showwarning(title = 'Maze solving using A\*', message = 'There was an issue connecting to the database. You can still play as guest.')

        registerButton['state'] = 'disabled'

        signInButton['state'] = 'disabled'

    #endif

    loginWindow.mainloop()

#endsub

def displayForm(option):

    '''Produce the form into which the user can input a username or password.'''

    formWindow=tkinter.Tk()

    formWindow.title('Maze-solving using A\*')

    formWindow.geometry('600x400')

    formWindow.configure(bg= '#ffffff')

    usernameLabel = tkinter.Label(formWindow, text='Username:', font= '"OCR A Extended" 25', fg='#000000', bg = '#ffffff'  )

    usernameLabel.place(relx= '0.1', rely= '0.12')

    usernameForm = tkinter.Entry(formWindow, bg = '#f0f0f0', fg = '#000000', width = '15', font = '"Verdana" 14', bd='3')

    usernameForm.place(relx = '0.1', rely ='0.25' )

    passwordLabel = tkinter.Label(formWindow, text='Password:', font= '"OCR A Extended" 25', fg='#000000', bg = '#ffffff'  )

    passwordLabel.place(relx= '0.1', rely= '0.42')

    passwordForm = tkinter.Entry(formWindow, bg = '#f0f0f0', fg = '#000000', width = '15', font = '"Verdana" 14', bd='3', show='\*' )

    passwordForm.place(relx = '0.1', rely ='0.55' )

    if option == 'register':

        submitButton = tkinter.Button(formWindow, bg='#ccffe0', activebackground='#aaddc0', bd= 3, fg= '#000000', font='"OCR A Extended" 20', text='Submit', command=lambda: register(usernameForm.get(), passwordForm.get(), formWindow))

    elif option == 'sign in':

        submitButton = tkinter.Button(formWindow, bg='#ccffe0', activebackground='#aaddc0', bd= 3, fg= '#000000', font='"OCR A Extended" 20', text='Submit', command=lambda: signIn(usernameForm.get(), passwordForm.get(), formWindow))

    #endif

    submitButton.place(relx='0.1', rely = '0.75')

    backButton = tkinter.Button(formWindow, text = 'Back', font = '"OCR A Extended" 20', bg = '#e2e2e2', activebackground='#cccccc', fg = '#000000', relief = 'raised', padx = 0, pady = 0, bd = 4, command=lambda: navigate('display login', formWindow, None))

    backButton.place(relx = 0.77, rely = 0.75)

    formWindow.mainloop()

#endsub

def register(username, password, window):

    '''Allow the user to register a new account'''

    #prevent null inputs

    if username.isspace() or username == '':

        messagebox.showwarning(title = 'Maze solving using A\*', message = 'Please enter a username.')

    elif password.isspace() or password == '':

        messagebox.showwarning(title = 'Maze solving using A\*', message = 'Please enter a password.')

    else:

        try:

            db = mysql.connector.connect(host = 'localhost', user = 'root', password = 'root', database = 'user\_accounts') #attempt to connect to the database

            success = userAccountManagement.register(username, password, db)

            if success:

                messagebox.showinfo(title = 'Maze solving using A\*', message = 'Successfully registered {0}.'.format(username))

                navigate('menu', window, username)

            else:

                messagebox.showwarning(title = 'Maze solving using A\*', message = 'That username is already in use. Please pick another.')

            #endif

        except mysql.connector.errors.ProgrammingError: #the current device successfully imported mysql.connector but could not connect to this specific database.

            messagebox.showwarning(title = 'Maze solving using A\*', message = 'There was an issue connecting to the database. You can still play as a guest.')

            navigate('menu', window, 'Guest')

        #endtry

    #endif

#endsub

def signIn(username, password, window):

    '''Allow a user to sign in to an existing account'''

    #prevent null inputs

    if username.isspace() or username == '':

        messagebox.showwarning(title = 'Maze solving using A\*', message = 'Please enter a username.')

    elif password.isspace() or password == '':

        messagebox.showwarning(title = 'Maze solving using A\*', message = 'Please enter a password.')

    else:

        try:

            db = mysql.connector.connect(host = 'localhost', user = 'root', password = 'root', database = 'user\_accounts') #attempt to connect to the database

            success = userAccountManagement.signIn(username, password, db)

            if success:

                messagebox.showinfo(title = 'Maze solving using A\*', message = 'Welcome {0}.'.format(username))

                navigate('menu', window, username)

            else:

                messagebox.showwarning(title = 'Maze solving using A\*', message = 'The username or password is incorrect.')

            #endif

        except mysql.connector.errors.ProgrammingError:

            messagebox.showwarning(title = 'Maze solving using A\*', message = 'There was an issue connecting to the database. You can still play as a guest.')

            navigate('menu', window, 'Guest')

        #endtry

    #endif

#endsub

#Menu

def displayMenu(username):

    '''Produce the menu for the user to choose which game mode to play'''

    menuWindow=tkinter.Tk()

    menuWindow.title('Maze-solving using A\*')

    menuWindow.geometry('600x400')

    menuWindow.configure(bg= '#ffffff')

    userLabel= tkinter.Label(menuWindow, text='User: {0}'.format(username), font= '"OCR A Extended" 25', fg='#000000', bg = '#ffffff')

    userLabel.place(relx = 0.25, rely = 0.1)

    mazeButton = tkinter.Button(menuWindow, bg='#3de3d2', activebackground = '#1dc9b8', bd='5',  text= 'Maze Game', font= '"OCR A Extended" 20 bold', fg='#000000', padx='10', pady='40', justify='center', relief='raised', command=lambda: navigate('maze', menuWindow, username))

    mazeButton.place(rely='0.35', relx='0.07')

    aStarButton = tkinter.Button(menuWindow, bg='#639aff', activebackground = '#337aff', bd= '5', text= 'A\*\nSimulation', font= '"OCR A Extended" 20 bold', fg='#000000', padx='10', pady= '25', justify='center', relief='raised', command=lambda: navigate('A\*', menuWindow, username))

    aStarButton.place(rely='0.35', relx= '0.57' )

    signOutButton = tkinter.Button (menuWindow, bg='#e2e2e2', activebackground='#cccccc',bd= '4', text= 'Sign out', font= '"OCR A Extended" 15', fg='#000000', padx='10', pady= '10', justify='center', relief='raised', command=lambda: navigate('display login', menuWindow, None))

    signOutButton.place(rely='0.77', relx= '0.71' )

    menuWindow.mainloop()

#endsub

#Maze Game

def mazeDifficulty(username):

    '''Allow the user to select a difficulty for the maze'''

    mazeDifficultyWindow=tkinter.Tk()

    mazeDifficultyWindow.title('Maze-solving using A\*')

    mazeDifficultyWindow.geometry('600x400')

    mazeDifficultyWindow.configure(bg= '#ffffff')

    var = tkinter.IntVar() #contains the size the maze should be based on which difficulty is picked.

    text = tkinter.Label(mazeDifficultyWindow,  text = 'Select a difficulty:', bg= '#ffffff', font = '"OCR A Extended" 25 ', fg = '#000000' )

    text.place(relx= '0.1', rely= '0.07')

    easy = tkinter.Radiobutton(mazeDifficultyWindow, text = 'Easy', font = '"OCR A Extended" 20', bg = '#80ffb0', activebackground='#4dff91', fg = '#000000', bd = 3, relief= 'raised', variable = var, value = 5)

    easy.place(relx = '0.15', rely = 0.25)

    medium = tkinter.Radiobutton(mazeDifficultyWindow, text = 'Medium', font = '"OCR A Extended" 20', bg = '#ffffb3', activebackground='#ffff80', fg = '#000000', bd = 3, relief= 'raised', variable = var, value = 10)

    medium.place(relx = '0.15', rely = 0.45)

    hard = tkinter.Radiobutton(mazeDifficultyWindow, text = 'Hard', font = '"OCR A Extended" 20', bg = '#ff7070', activebackground='#ff4d4d', fg = '#000000', bd = 3, relief= 'raised', variable = var, value = 15)

    hard.place(relx = '0.15', rely = 0.65)

    selectButton = tkinter.Button(mazeDifficultyWindow, text = 'Select', font = '"OCR A Extended" 23', bg = '#c5ffff', activebackground = '#a9f0f9', fg = '#000000', relief = 'raised', padx = 10, pady = 15, bd = 4, command=lambda:getSize(mazeDifficultyWindow, var, username))

    selectButton.place(relx = 0.57, rely = 0.4)

    backButton = tkinter.Button(mazeDifficultyWindow, text = 'Back', font = '"OCR A Extended" 15', bg = '#e2e2e2', activebackground = '#cccccc', fg = '#000000', relief = 'raised', padx = 5, pady = 5, bd = 4, command = lambda:navigate('menu', mazeDifficultyWindow, username))

    backButton.place(relx = 0.82, rely = 0.8)

    mazeDifficultyWindow.mainloop()

#endsub

def getSize(window, var, username):

    '''Retrieve the size of the maze and call Maze Game'''

    size = var.get()

    if size == 0: #no button was selected

        messagebox.showwarning(title = 'Maze solving using A\*', message = 'Please select a difficulty.')

    else:

        window.destroy()

        changeDifficulty, returnToMenu = mazeGame.navigateMazeSection(size, username) # call Maze Game

        #redirect to the correct page

        if changeDifficulty == True:

            navigate('maze', None, username)

        elif returnToMenu == True:

            navigate('menu', None, username)

        #endif

    #endif

#endsub

#Navigation

def navigate(option, window, username):

    '''Navigate between different parts of the program.'''

    if window:

        window.destroy() #closes the current window

    #endif

    if option == 'display login':

        displayLogin()

    elif option == 'register' or option == 'sign in':

        displayForm(option)

    elif option == 'menu':

        displayMenu(username)

    elif option == 'maze':

        mazeDifficulty(username)

    elif option == 'A\*':

        AStarSimulation.navigateAStarSimulation() #call A\* Simulation

        navigate('menu', None, username)

    #endif

#endsub

if \_\_name\_\_ == '\_\_main\_\_':

    navigate('display login', None, None)

#endif

### 7.2.4 User Account Management

Program name: ‘**userAccountManagement**’

Communicates with the database allowing the user to register or sign in to an account.

import hashlib #to hash passwords

def register(username, password, db):

    '''Allow users to create a new account'''

    #check if the chosen username is already in use

    if username == 'Guest':

        return False #as this is the guest account

    #endif

    cursor = db.cursor()

    cursor.execute('SELECT username FROM user\_details')

    for names in cursor:

        for name in names:

            if name == username:

                return False #so the program doesn't continue

            #endif

        #endif

    #endfor

    #hash the user's password using SHA256 so it is unrecognisable

    password = str(hashlib.sha256(password.encode('utf-8')).hexdigest())

    #add a record to the database with None in the high score fields

    sql = 'INSERT INTO user\_details (username, password) VALUES (%s, %s)'

    values = (username, password)

    cursor.execute(sql, values)

    db.commit()

    return True

#endsub

def signIn(username, password, db):

    '''Allow the user to sign in to their existing account'''

    #lookup the password hash in the database

    cursor = db.cursor()

    sql = 'SELECT password FROM user\_details WHERE username = %s'

    value = (username, )

    cursor.execute(sql, value)

    result = cursor.fetchone()

    if result:

        for oldhash in result:

            #calculate the hash of the newly typed password and check against this hash

            newhash = str(hashlib.sha256(password.encode('utf-8')).hexdigest())

            if oldhash == newhash:

                return True

            #endif

        #endfor

    else:

        return False #the username was not in the database

    #endif

#endsub

### 7.2.5 Maze Game

Program name: ‘**mazeGame**’

Randomly generates a maze which the user can play, keeping track of their time taken and updating the database with their fastest time. Users can choose to view the solution, switch difficulty, generate new mazes or switch colour themes.

#The main part of Maze Game

import time

import tkinter

from threading import Thread

try:

    import mysql.connector

except ImportError:

    pass #an error message for this is already produced in the main program

#endtry

import randomisedDepthFirstSearch

import spriteClass

import stopwatchClass

import AStarAlgorithm

#Objects

class MazeClass():

    '''This class is used so that variables can be adjusted by functions that are called from a Tkinter button or keypress.'''

    def \_\_init\_\_(self, size, username, mazeWindow):

        self.\_\_SIZE = size

        self.\_\_USERNAME = username

        self.\_\_DEFAULT = {'main': '#ffffff', 'text': '#000000', 'accent': '#0000e6', 'background': '#f2f2f2', 'activebutton': '#d9d9d9', 'trail1':'#a6a6a6', 'trail2': '#a6a6a6', 'trail3':'#a6a6a6', 'trail4':'#a6a6a6', 'walls': '#000000'}

        self.\_\_BLUE = {'main': '#e6ffff', 'text': '#000000', 'accent': '#0f008a', 'background': '#80ffff', 'activebutton': '#7ceeee', 'trail1':'#ffffff', 'trail2': '#ffffff', 'trail3':'#ffffff', 'trail4':'#ffffff', 'walls': '#001a66'}

        self.\_\_GREEN = {'main': '#99ffdd', 'text': '#000000', 'accent': '#004d1a', 'background': '#33ff77', 'activebutton': '#00ff55', 'trail1':'#ccffcc', 'trail2': '#ccffcc', 'trail3':'#ccffcc', 'trail4':'#ccffcc', 'walls': '#006600'}

        self.\_\_PINK = {'main': '#ff80d5', 'text': '#000000', 'accent': '#3d004d', 'background': '#ffccf6', 'activebutton': '#ff99ec', 'trail1':'#ff96f1', 'trail2': '#ff96f1', 'trail3':'#ff96f1', 'trail4':'#ff96f1', 'walls': '#ff0000'}

        self.\_\_NEON = {'main': '#000000', 'text': '#ffffff', 'accent': '#ffffcc', 'background': '#4d4d4d', 'activebutton': '#ffff00', 'trail1':'#ff0080', 'trail2': '#1aa3ff', 'trail3':'#8600b3', 'trail4':'#ffff00', 'walls': '#ffffff'}

        self.\_theme = self.\_\_DEFAULT #the theme currently in use out of the above 5 options

        self.\_mazeWindow = mazeWindow

        self.\_\_SCALE = 400/size #used for scaling so the maze fills the canvas

        self.\_\_TRANSLATE = self.\_\_SCALE /2 #used to translate cells to the correct position.

        self.\_mazeArea = tkinter.Canvas(mazeWindow, bd = 0, bg = self.\_theme['background'], height = 400, width = 400 ) #the tkinter canvas which the maze is drawn on

        self.\_sprite = spriteClass.Sprite()

        self.\_spriteImage = None #the image of the sprite that is drawn on the canvas

        self.\_startImage = None

        self.\_passages = []

        self.\_allCells = []

        self.\_cellsList = []

        self.\_drawnCells = {} #of the form {cell's xy coordinates : Tkinter rectangle corresponding to this cell}

        self.\_allWalls = []

        self.\_drawnWalls = []

        self.\_finished = False #used to determine whether the congratulations page should be displayed

        self.\_stopwatch = None

        self.\_thread = None

        self.\_themePageOpen = False #prevents the user from opening the themes page multiple times

        self.\_changeDifficulty = False

        self.\_returnToMenu = False

    #endsub

    def getSize(self):

        return self.\_\_SIZE

    #endsub

    def getUsername(self):

        return self.\_\_USERNAME

    #endsub

    def getDefaultTheme(self):

        return self.\_\_DEFAULT

    #endsub

    def getBlueTheme(self):

        return self.\_\_BLUE

    #endsub

    def getGreenTheme(self):

        return self.\_\_GREEN

    #endsub

    def getPinkTheme(self):

        return self.\_\_PINK

    #endsub

    def getNeonTheme(self):

        return self.\_\_NEON

    #endsub

    def getTheme(self):

        return self.\_theme

    #endsub

    def setTheme(self, new):

        self.\_theme = new

    #endsub

    def getMazeWindow(self):

        return self.\_mazeWindow

    #endsub

    def setMazeWindow(self, new):

        self.\_mazeWindow = new

    #endsub

    def getScale(self):

        return self.\_\_SCALE

    #endsub

    def getTranslate(self):

        return self.\_\_TRANSLATE

    #endsub

    def getMazeArea(self):

        return self.\_mazeArea

    #endsub

    def setMazeArea(self, new):

        self.\_mazeArea = new

    #endsub

    def getSprite(self):

        return self.\_sprite

    #endsub

    def setSprite(self, new):

        self.\_sprite = new

    #endsub

    def getSpriteImage(self):

        return self.\_spriteImage

    #endsub

    def setSpriteImage(self, new):

        self.\_spriteImage = new

    #endsub

    def getStartImage(self):

        return self.\_startImage

    #endsub

    def setStartImage(self, new):

        self.\_startImage = new

    #endsub

    def getPassages(self):

        return self.\_passages

    #endsub

    def setPassages(self, new):

        self.\_passages = new

    #endsub

    def getAllCells(self):

        return self.\_allCells

    #endsub

    def setAllCells(self, new):

        self.\_allCells = new

    #endsub

    def getCellsList(self):

        return self.\_cellsList

    #endsub

    def setCellsList(self, new):

        self.\_cellsList = new

    #endsub

    def getDrawnCells(self):

        return self.\_drawnCells

    #endsub

    def setDrawnCells(self, new):

        self.\_drawnCells = new

    #endsub

    def getAllWalls(self):

        return self.\_allWalls

    #endsub

    def setAllWalls(self, new):

        self.\_allWalls = new

    #endsub

    def getDrawnWalls(self):

        return self.\_drawnWalls

    #endsub

    def setDrawnWalls(self, new):

        self.\_drawnWalls = new

    #endsub

    def getFinished(self):

        return self.\_finished

    #endsub

    def setFinished(self, new):

        self.\_finished = new

    #endsub

    def getStopwatch(self):

        return self.\_stopwatch

    #endsub

    def setStopwatch(self, new):

        self.\_stopwatch = new

    #endsub

    def getThread(self):

        return self.\_thread

    #endsub

    def setThread(self, new):

        self.\_thread = new

    #endsub

    def getThemePageOpen(self):

        return self.\_themePageOpen

    #endsub

    def setThemePageOpen(self, new):

        self.\_themePageOpen = new

    #endsub

    def getChangeDifficulty(self):

        return self.\_changeDifficulty

    #endsub

    def setChangeDifficulty(self, new):

        self.\_changeDifficulty = new

    #endsub

    def getReturnToMenu(self):

        return self.\_returnToMenu

    #endsub

    def setReturnToMenu(self, new):

        self.\_returnToMenu = new

    #endsub

#endclass

class stopwatchThread():

    '''Used to update the stopwatch. Threads run simultaneously with the main program.'''

    def \_\_init\_\_(self, x, variables):

        self.\_running = True

        self.\_t = Thread(target = self.\_updateStopwatch, args = (x, variables))

        self.\_t.setDaemon(True) #so the thread ends when the window is closed

        self.\_t.start()

    #endsub

    def stop(self):

        '''Causes the thread to end'''

        self.\_running = False

    #endsub

    def \_updateStopwatch(self, x, variables):

        '''Used to update the displayed stopwatch each time it increments'''

        stopwatch = variables.getStopwatch()

        try:

            minutes = stopwatch.getMinutes()

            while minutes < 60 and variables.getFinished() == False and self.\_running == True: #don't try to update the stopwatch after one hour or when the maze has been completed

                stopwatch.increment()

                timeTaken = stopwatch.getDisplay()

                stopwatchDisplay = tkinter.Label(variables.getMazeWindow(), text= timeTaken, font = '"Verdana" 15', fg = variables.getTheme()['accent'], bg = variables.getTheme()['main'])

                stopwatchDisplay.place(relx = 0.45, rely = 0.9)

                time.sleep(1)

                minutes = stopwatch.getMinutes()

            #endwhile

        except (RuntimeError, tkinter.TclError): #There may be an error if the user closes the program while the thread is running

            pass

        #endtry

        variables.setStopwatch(stopwatch)

    #endsub

    def gett(self):

        return self.\_t

    #endsub

    def sett(self, new):

        self.\_t = new

    #endsub

#endclass

#Changing the theme

def closeThemePage(themesWindow, variables):

    '''Closes the themes page properly when the user presses X'''

    variables.setThemePageOpen(False)

    themesWindow.destroy()

#endsub

def themePage(size, variables):

    '''Produce a page allowing the user to choose a theme'''

    if variables.getThemePageOpen() == False: #prevent the page from being open more than once at a time

        try:

            variables.setThemePageOpen(True)

            themesWindow = tkinter.Tk()

            themesWindow.title('Maze-solving using A\*')

            themesWindow.geometry('400x500')

            themesWindow.configure(bg = variables.getTheme()['main'])

            themesWindow.protocol('WM\_DELETE\_WINDOW', lambda: closeThemePage(themesWindow, variables)) #means if the user closes the window using X this window can be opened again, otherwise the themes button would do nothing.

            title = tkinter.Label(themesWindow, text = 'Select a theme:', font = '"OCR A Extended" 25', fg = variables.getTheme()['text'], bg = variables.getTheme()['main'])

            title.place(relx =0.05, rely = 0.03)

            default = tkinter.Button(themesWindow, text = 'Default', font = '"OCR A Extended" 20', fg = variables.getDefaultTheme()['text'], bg = variables.getDefaultTheme()['background'], activebackground = variables.getDefaultTheme()['activebutton'], relief = 'raised', padx = 5, pady = 5, bd = 4, command = lambda:changeTheme(size, variables, themesWindow, variables.getDefaultTheme()))

            default.place(relx = 0.3, rely = 0.15)

            blue = tkinter.Button(themesWindow, text = 'Blue', font = '"OCR A Extended" 20', fg = variables.getBlueTheme()['text'], bg = variables.getBlueTheme()['background'], activebackground=variables.getBlueTheme()['activebutton'], relief = 'raised', padx = 5, pady = 5, bd = 4, command = lambda:changeTheme(size, variables, themesWindow, variables.getBlueTheme()))

            blue.place(relx = 0.35, rely = 0.32)

            green = tkinter.Button(themesWindow, text = 'Green', font = '"OCR A Extended" 20', fg = variables.getGreenTheme()['text'], bg = variables.getGreenTheme()['background'], activebackground=variables.getGreenTheme()['activebutton'], relief = 'raised', padx = 5, pady = 5, bd = 4, command = lambda:changeTheme(size, variables, themesWindow, variables.getGreenTheme()))

            green.place(relx = 0.33, rely = 0.49)

            pink = tkinter.Button(themesWindow, text = 'Pink', font = '"OCR A Extended" 20', fg = variables.getPinkTheme()['text'], bg = variables.getPinkTheme()['background'], activebackground=variables.getPinkTheme()['activebutton'], relief = 'raised', padx = 5, pady = 5, bd = 4, command = lambda:changeTheme(size, variables, themesWindow, variables.getPinkTheme()))

            pink.place(relx = 0.35, rely = 0.66)

            neon = tkinter.Button(themesWindow, text = 'Neon', font = '"OCR A Extended" 20', fg = variables.getNeonTheme()['text'], bg = variables.getNeonTheme()['background'], activebackground=variables.getNeonTheme()['activebutton'], relief = 'raised', padx = 5, pady = 5, bd = 4, command = lambda:changeTheme(size, variables, themesWindow, variables.getNeonTheme()))

            neon.place(relx = 0.35, rely = 0.83)

            themesWindow.mainloop()

        except tkinter.TclError: #if the user closes the main maze window this error will occur

            themesWindow.destroy()

        #endtry

    #endif

#endsub

def changeTheme(size, variables, themesWindow, newTheme):

    '''Reset the main maze window with the new theme'''

    variables.setThemePageOpen(False)

    themesWindow.destroy()

    variables.setTheme(newTheme)

    #reset the sprite's position

    sprite = variables.getSprite()

    sprite.setxy((0,0))

    variables.setSprite(sprite)

    #end the current stopwatch thread

    thread = variables.getThread()

    thread.stop()

    t = thread.gett()

    t.join() #ends the thread properly

    thread.sett(t)

    variables.setThread(thread)

    main(size, variables)

#endsub

#Solving the maze

def solveMaze(variables):

    '''Use the A\* algorithm to find and display a solution to the maze'''

    #find the solution

    solution, a = AStarAlgorithm.AStar(variables.getAllCells()[0], variables.getAllCells()[-1], variables.getAllWalls(), variables.getCellsList(), variables.getAllCells(), None)

    #display the solution on the canvas

    for cell in variables.getAllCells()[1:-1]:

        cell.setColour(variables.getTheme()['background']) #clear the grid first

        drawCell(cell, variables)

    #endfor

    for cell in solution[1:-1]: #omit the end cell so congratulations isn't called

        position = variables.getCellsList().index([cell[0], cell[1]])

        cell = variables.getAllCells()[position]

        editCell(cell, variables)

    #endfor

    mazeArea = variables.getMazeArea()

    try:

        for wall in variables.getDrawnWalls():

            mazeArea.tag\_raise(wall) #brings the walls to the top layer

        #endfor

        mazeArea.tag\_raise(variables.getSpriteImage())

    except tkinter.TclError: #may be caused when the tkinter window quits

        pass

    #endtry

    variables.setMazeArea(mazeArea)

    variables.setFinished(True) #so no time is recorded

#endsub

#Congratulating the player

def findHighScore(newTime, newTimeDisplay, variables):

    '''Retrieve the user's high score from the database'''

    newHighScore = False

    #find the current difficulty

    size = variables.getSize()

    if size == 5:

        difficulty = 'Easy'

    elif size == 10:

        difficulty = 'Medium'

    elif size == 15:

        difficulty = 'Hard'

    #endif

    username = variables.getUsername()

    if username == 'Guest':

        return difficulty, 'XX:XX', newHighScore #for guests the database isn't accessed

    else:

        db = mysql.connector.connect(host = 'localhost', user = 'root', password = 'root', database = 'user\_accounts')

        cursor = db.cursor()

        #retrieve the old high score

        if difficulty == 'Easy':

            sql = 'SELECT easy\_high\_score FROM user\_details WHERE username = %s'

        elif difficulty == 'Medium':

            sql = 'SELECT medium\_high\_score FROM user\_details WHERE username = %s'

        elif difficulty == 'Hard':

            sql = 'SELECT hard\_high\_score FROM user\_details WHERE username = %s'

        #endif

        value = (username,)

        cursor.execute(sql, value)

        result = cursor.fetchone()

        if result == (None,): #the user hasn't attempted this difficulty before

            newHighScore = True

            highScore = newTimeDisplay

        else:

            for oldTime in result:

                highScore = oldTime #by default the previous high score is the high score

                #convert to total number of seconds

                minutes = int(oldTime[:2])

                seconds = int(oldTime[3:])

                oldTime = minutes\*60 + seconds

                #compare the scores

                if newTime < oldTime:

                    newHighScore = True

                    highScore = newTimeDisplay

                #endif

            #endfor

        #endif

        #update the high score in the database

        if newHighScore:

            if difficulty == 'Easy':

                sql = 'UPDATE user\_details SET easy\_high\_score = %s WHERE username = %s'

            elif difficulty == 'Medium':

                sql = 'UPDATE user\_details SET medium\_high\_score = %s WHERE username = %s'

            elif difficulty == 'Hard':

                sql = 'UPDATE user\_details SET hard\_high\_score = %s WHERE username = %s'

            #endif

            values = (newTimeDisplay, username,)

            cursor.execute(sql, values)

            db.commit()

        #endif

        return difficulty, highScore, newHighScore

    #endif

#endsub

def congratulations(variables):

    '''Produce a page congratulating the user for completing the maze'''

    #check if this score is a new high score

    time = variables.getStopwatch().getTotal()

    timeDisplay = variables.getStopwatch().getDisplay()

    difficulty, highScore, newHighScore = findHighScore(time, timeDisplay, variables)

    congratulationsWindow=tkinter.Tk()

    congratulationsWindow.title('Maze-solving using A\*')

    congratulationsWindow.geometry('600x450')

    congratulationsWindow.configure(bg = variables.getTheme()['main'])

    title = tkinter.Label(congratulationsWindow, text = 'CONGRATULATIONS!', font = 'Stencil 40', fg = variables.getTheme()['text'], bg = variables.getTheme()['main'])

    title.place(relx =0.07, rely = 0.03)

    timelabel = tkinter.Label(congratulationsWindow, text = 'Your Time:', font = '"OCR A Extended" 25', fg = variables.getTheme()['text'], bg = variables.getTheme()['main'])

    timelabel.place(relx = 0.35, rely = 0.21)

    score = tkinter.Label(congratulationsWindow, text = timeDisplay, font = '"Verdana" 22 bold', fg = variables.getTheme()['accent'], bg = variables.getTheme()['main'])

    score.place(relx = 0.42, rely = 0.31)

    hslabel = tkinter.Label(congratulationsWindow, text = '{0} High Score:'.format(difficulty), font = '"OCR A Extended" 25', fg = variables.getTheme()['text'], bg = variables.getTheme()['main'])

    hslabel.place(relx = 0.25, rely = 0.47)

    hscore = tkinter.Label(congratulationsWindow, text = highScore, font = '"Verdana" 22 bold', fg = variables.getTheme()['accent'], bg = variables.getTheme()['main'])

    hscore.place(relx = 0.42, rely = 0.57)

    newhs = tkinter.Label(congratulationsWindow, text = '\*\*\* NEW HIGH SCORE \*\*\*', font = '"OCR A Extended" 20', fg = variables.getTheme()['accent'], bg = variables.getTheme()['main'])

    if newHighScore == True:

        newhs.place(relx = 0.2, rely = 0.72)

    #endif

    back = tkinter.Button(congratulationsWindow, text = 'Back', font = '"OCR A Extended" 15', bg = variables.getTheme()['background'], activebackground=variables.getTheme()['activebutton'], fg = variables.getTheme()['text'], relief = 'raised', padx = 5, pady = 5, bd = 4, command=lambda:congratulationsWindow.destroy())

    back.place(relx = 0.44, rely = 0.84)

    congratulationsWindow.mainloop()

#endsub

#Creating the maze

def drawCell(cell, variables):

    '''Draw one of the cells in the maze'''

    xy = cell.getxy() #this is the centre of a cell, so topleft and bottomright are the corners of this cell.

    topleft = (xy[0]-0.5, xy[1]-0.5)

    bottomright = (xy[0]+0.5, xy[1]+0.5)

    global colour

    colour = cell.getColour()

    mazeArea = variables.getMazeArea()

    drawnCells = variables.getDrawnCells()

    #delete the cell if already present. This improves performance after cells have been redrawn lots of times.

    if xy in drawnCells:

        mazeArea.delete(drawnCells[xy])

        del drawnCells[xy]

    #endif

    #draw the cell

    if type(colour) == str:

        rectangle = mazeArea.create\_rectangle(topleft[0]\*variables.getScale()+variables.getTranslate(), topleft[1]\*variables.getScale()+variables.getTranslate(), bottomright[0]\*variables.getScale()+variables.getTranslate(), bottomright[1]\*variables.getScale()+variables.getTranslate(), fill = colour, outline = colour)

    else: #used by the start and end cell

        #scale the image depending on the maze size

        if variables.getSize() == 5:

            colour = variables.getMazeWindow().colour = colour.zoom(5, 5)

            colour = variables.getMazeWindow().colour = colour.subsample(2, 2)

        elif variables.getSize() == 10:

            colour = variables.getMazeWindow().colour = colour.zoom(5, 5)

            colour = variables.getMazeWindow().colour = colour.subsample(4, 4)

        elif variables.getSize() == 15:

            colour = variables.getMazeWindow().colour = colour.zoom(5, 5)

            colour = variables.getMazeWindow().colour = colour.subsample(6, 6)

        #endif

        if cell.getType() == 'start':

            variables.setStartImage(colour) #This prevents the start cell from being garbage collected and not displaying on the canvas.

            rectangle = mazeArea.create\_image(topleft[0]\*variables.getScale()+variables.getTranslate(), topleft[1]\*variables.getScale()+variables.getTranslate(), image=variables.getStartImage(), anchor='nw')

        else:

            rectangle = mazeArea.create\_image(topleft[0]\*variables.getScale()+variables.getTranslate(), topleft[1]\*variables.getScale()+variables.getTranslate(), image=variables.getMazeWindow().colour, anchor='nw')

        #endif

    #endif

    drawnCells.update({xy:rectangle}) #add this newly drawn cell to drawnCells

    variables.setMazeArea(mazeArea)

    variables.setDrawnCells(drawnCells)

#endsub

def editCell(currentCell, variables):

    '''Check if this is the end cell, and if not, change it's colour.'''

    type=currentCell.getType()

    if type == 'end' and variables.getFinished() == False:

        variables.setFinished(True) #this means the congratulations page is only called once

        congratulations(variables) #call the congratulations page

    elif type == 'empty':

        xy=currentCell.getxy()

        if (xy[0]+ xy[1]) % 4 == 0: #this if statement determines which trail colour to use when using the NEON theme

            currentCell.setColour(variables.getTheme()['trail1'])

        elif ((xy[0]+ xy[1])) % 4 == 1:

            currentCell.setColour(variables.getTheme()['trail2'])

        elif ((xy[0]+ xy[1])) % 4 == 2:

            currentCell.setColour(variables.getTheme()['trail3'])

        elif ((xy[0]+ xy[1])) % 4 == 3:

            currentCell.setColour(variables.getTheme()['trail4'])

        #endif

        drawCell(currentCell, variables)

    #endif

#endsub

def createWalls(variables):

    '''Use randomised depth-first search to create a list of passages, then generate a corresponding list of walls.'''

    allWalls = []

    #create an instance of each wall, horizontal and vertical.

    for i in range (0, variables.getSize()):

        for j in range(0, variables.getSize()):

            wall = ((i, j), (i+1, j))

            allWalls.append(wall)

            wall = ((i, j), (i, j+1))

            allWalls.append(wall)

        #endfor

    #endfor

    #Translate each wall so that they are drawn around each cell rather than through them. (walls are translated left and up; the right and bottom sides of the maze are ignored as those walls wouldn't be seen.)

    newWalls = []

    for wall in allWalls: #eg ((3,2),(4,2)) - this wall will be a horizontal wall above cell (3, 2)

        start = wall[0] #(3,2)

        st1 = start[0] -0.5 #2.5 - between cells (2, y) and (3, y)

        st2 = start[1] -0.5 #1.5

        end = wall[1] #(4,2)

        en1 = end[0] -0.5 #3.5

        en2 = end[1] -0.5 #1.5

        wall = ((st1, st2), (en1, en2)) #((2.5, 1.5), (3.5, 1.5)) This wall is therefore the upper boundary of cell (3,2)

        newWalls.append(wall)

    #endfor

    #newWalls is currently a grid. The walls crossed by the list of passages must be removed.

    for p in variables.getPassages():

        for w in newWalls:

            if ((w[0][0]+w[1][0])/2 == (p[0][0]+p[1][0])/2) and ((w[0][1]+w[1][1])/2 == (p[0][1]+p[1][1])/2): #check if the midpoint of the wall and passage match. Midpoint = (startx+endx)/2 or (starty + endy) /2

                newWalls.remove(w)

            #endif

        #endfor

    #endfor

    variables.setAllWalls(newWalls)

#endsub

def drawWalls(variables):

    '''Given the list of walls, draw them on the maze'''

    drawnWalls = []

    mazeArea = variables.getMazeArea()

    for wall in variables.getAllWalls():

        start = wall[0]

        end = wall[1]

        drawnWalls.append(mazeArea.create\_line((start[0])\*variables.getScale()+variables.getTranslate()+1, (start[1])\*variables.getScale()+variables.getTranslate()+1, (end[0])\*variables.getScale()+variables.getTranslate()+1, (end[1])\*variables.getScale()+variables.getTranslate()+1, fill=variables.getTheme()['walls'], width = 2))

    #endfor

    variables.setDrawnWalls(drawnWalls)

    variables.setMazeArea(mazeArea)

#endsub

def drawSprite(variables, size):

    '''Draw the sprite onto the maze'''

    xy = variables.getSprite().getxy()

    s = variables.getMazeWindow().s = tkinter.PhotoImage(file='sprite.png')

    #scale the sprite depending on the size of the maze

    if size == 5:

        s = variables.getMazeWindow().s = s.zoom(2, 2)

    elif size == 15:

        s = variables.getMazeWindow().s = s.zoom(3, 3) #the values in zoom and subsample must be integers. This is the equivalent of \*3 then /4, = \*0.75.

        s = variables.getMazeWindow().s = s.subsample(4, 4)

    #endif

    mazeArea = variables.getMazeArea()

    spriteImage = mazeArea.create\_image(xy[0]+variables.getTranslate() - int(0.75\*(variables.getTranslate()))-1, xy[1]+variables.getTranslate() - int(0.75\*(variables.getTranslate()))-1, image=s, anchor='nw')

    try:

        mazeArea.tag\_raise(spriteImage)

    except tkinter.TclError: #may be caused when the tkinter window quits

        pass

    #endtry

    variables.setSpriteImage(spriteImage)

    variables.setMazeArea(mazeArea)

#endsub

def createMaze(variables):

    '''Call the relevant subroutines to create and display the maze'''

    #create a new maze

    passages, allCells, cellsList = randomisedDepthFirstSearch.main(variables.getSize(), variables.getTheme())

    variables.setAllCells(allCells)

    variables.setCellsList(cellsList)

    variables.setPassages(passages)

    #draw the cells

    for cell in variables.getAllCells():

        drawCell(cell, variables)

    #endfor

    #draw the walls

    createWalls(variables)

    drawWalls(variables)

    #position the sprite

    sprite = variables.getSprite()

    oldxy = sprite.getxy()

    mazeArea = variables.getMazeArea()

    mazeArea.move(variables.getSpriteImage(), (oldxy[0]\*- 1\*variables.getScale()), (oldxy[1]\*-1\*variables.getScale())) #the -1 moves it back to (0, 0)

    try:

        mazeArea.tag\_raise(variables.getSpriteImage())

    except tkinter.TclError: #may be caused when the tkinter window quits

        pass

    #endtry

    variables.setMazeArea(mazeArea)

    sprite.setxy((0,0)) #reset the sprite's coordinates

    variables.setSprite(sprite)

    variables.setFinished(False)

    #reset the stopwatch

    stopwatch = variables.getStopwatch()

    stopwatch.reset()

    variables.setStopwatch(stopwatch)

    thread = variables.getThread()

    if thread:

        thread.stop() #close the current thread before opening a new one

    #endif

    variables.setThread(stopwatchThread([], variables)) #allows the stopwatch to run in the background

#endsub

#Playing the maze

def moveSprite(direction, variables):

    '''Move the sprite in the given direction if there is a passage there'''

    #check if there is a passage in this direction

    move, movement = variables.getSprite().canMove(direction, variables.getCellsList(), variables.getAllCells())

    if move:

        mazeArea = variables.getMazeArea()

        mazeArea.move(variables.getSpriteImage(), movement[0]\*variables.getScale(), movement[1]\*variables.getScale()) #moves the image one cell in the given direction

        variables.setMazeArea(mazeArea)

        currentCell = variables.getSprite().getCell(variables.getCellsList(), variables.getAllCells())

        editCell(currentCell, variables)

        try:

            for wall in variables.getDrawnWalls():

                mazeArea.tag\_raise(wall) #brings the walls to the top layer (visible above the newly drawn cell)

            #endfor

            mazeArea.tag\_raise(variables.getSpriteImage())

        except tkinter.TclError: #may be caused when the tkinter window quits

            pass

        #endtry

        variables.setMazeArea(mazeArea)

    #endif

#endsub

#Exiting Maze Game

def changeDifficulty(variables):

    '''Exit Maze Section and return to the difficulty selection screen'''

    thread = variables.getThread()

    thread.stop() #terminate the thread before closing the window

    t = thread.gett()

    t.join()

    variables.getMazeWindow().destroy()

    variables.setChangeDifficulty(True)

#endsub

def returnToMenu(variables):

    '''Exit Maze Section and return to the menu'''

    thread = variables.getThread()

    thread.stop() #terminate the thread before closing the window

    t = thread.gett()

    t.join()

    variables.getMazeWindow().destroy()

    variables.setReturnToMenu(True)

#endsub

#Main window

def main(size, variables):

    '''The main subroutine for Maze Game, used to handle tkinter output and call subroutines to set up and play the maze'''

    #Produce the background window

    mazeWindow = variables.getMazeWindow()

    mazeWindow.title('Maze-solving using A\*')

    mazeWindow.geometry('800x500')

    mazeWindow.configure(bg= variables.getTheme()['main'])

    mazeArea = variables.getMazeArea()

    mazeArea.pack(anchor = 'nw', padx = 30, pady = 30)

    variables.setMazeArea(mazeArea)

    instructions = tkinter.Label(mazeWindow, text = 'Use the arrow keys\nto move.', justify= 'left', fg= variables.getTheme()['text'], bg= variables.getTheme()['main'], font = '"OCR A Extended" 20')

    instructions.place(relx = 0.59, rely = 0.06)

    variables.setStopwatch(stopwatchClass.Stopwatch())

    newButton = tkinter.Button(mazeWindow, text = 'New Maze', font = '"OCR A Extended" 15', bg = variables.getTheme()['background'], activebackground=variables.getTheme()['activebutton'], fg = variables.getTheme()['text'], relief = 'raised', padx = 5, pady = 5, bd = 4, command = lambda: createMaze(variables))

    newButton.place(relx = 0.59, rely = 0.25)

    difficultyButton = tkinter.Button(mazeWindow, text = 'Difficulty', font = '"OCR A Extended" 15', bg = variables.getTheme()['background'], activebackground=variables.getTheme()['activebutton'], fg = variables.getTheme()['text'], relief = 'raised', padx = 5, pady = 5, bd = 4, command = lambda:changeDifficulty(variables))

    difficultyButton.place(relx = 0.59, rely = 0.38)

    answerButton = tkinter.Button(mazeWindow, text = 'Solution', font = '"OCR A Extended" 15', bg = variables.getTheme()['background'], activebackground=variables.getTheme()['activebutton'], fg = variables.getTheme()['text'], relief = 'raised', padx = 5, pady = 5, bd = 4, command = lambda:solveMaze(variables))

    answerButton.place(relx = 0.59, rely = 0.51)

    themeButton = tkinter.Button(mazeWindow, text = 'Theme', font = '"OCR A Extended" 15', bg = variables.getTheme()['background'], activebackground=variables.getTheme()['activebutton'], fg = variables.getTheme()['text'], relief = 'raised', padx = 5, pady = 5, bd = 4, command = lambda:themePage(size, variables))

    themeButton.place(relx = 0.59, rely = 0.64)

    menuButton = tkinter.Button(mazeWindow, text = 'Menu', font = '"OCR A Extended" 15', bg = variables.getTheme()['background'], activebackground=variables.getTheme()['activebutton'], fg = variables.getTheme()['text'], relief = 'raised', padx = 5, pady = 5, bd = 4, command = lambda: returnToMenu(variables))

    menuButton.place(relx = 0.85, rely = 0.85)

    #Draw the sprite for the first time

    drawSprite(variables, size)

    mazeArea = variables.getMazeArea()

    try:

        mazeArea.tag\_raise(variables.getSpriteImage())

    except tkinter.TclError: #may be caused when the tkinter window quits

        pass

    #endtry

    variables.setMazeArea(mazeArea)

    #Create the maze

    createMaze(variables)

    #Bind keyboard inputs to moving the sprite

    mazeWindow.bind('<Left>', lambda event: moveSprite('left', variables))

    mazeWindow.bind('<Right>', lambda event: moveSprite('right', variables))

    mazeWindow.bind('<Up>', lambda event: moveSprite('up', variables))

    mazeWindow.bind('<Down>', lambda event: moveSprite('down', variables))

    variables.setMazeWindow(mazeWindow)

    variables.getMazeWindow().mainloop()

#endsub

#Initialising Maze Game

def navigateMazeSection(size, username):

    '''Call main using the default theme, and exit if changeDifficulty or returnToMenu become true.'''

    mazeWindow = tkinter.Tk()

    variables = MazeClass(size, username, mazeWindow) #Tkinter buttons do not allow variables to be returned. To avoid this issue I have made each variable an attribute of 'MazeClass', so that they can be accessed, and more importantly, edited by any subroutine where 'variables' is passed in.

    main(size, variables)

    return variables.getChangeDifficulty(), variables.getReturnToMenu()

#endsub

### 7.2.6 A\* Simulation

Program name: ‘**AStarSimulation**’

Contains a grid to which the user can add a start and end cell, walls, puddles and warp panels, and demonstrates the A\* pathfinding algorithm on this grid. They can also clear the grid or view information about this section of the program.

#The main part of A\* Simulation

from math import floor

import tkinter

from tkinter import messagebox

from cellClasses import Cell, StartCell, EndCell, Wall, Puddle, WarpIn, WarpOut

from AStarAlgorithm import AStar

#Objects

class AStarClass():

    '''This class contains all the variables required across different subroutines. This is because subroutines called using a button cannot return a variable.'''

    def \_\_init\_\_(self):

        self.\_\_HEIGHT = 10

        self.\_\_WIDTH = 24

        self.\_currentObject = 'wall'

        self.\_\_SCALE = 47

        self.\_\_TRANSLATE = 25 #these constants are used to scale and translate the grid to the correct proportion

        self.\_cellsList = []

        self.\_allCells = []

        self.\_walls = [] #a list of walls (the sides of each Wall cell placed on the grid)

        self.\_gridArea = None

        self.\_images = [] #a list of images to be displayed on the grid so they don't get garbage collected by tkinter and not displayed.

        self.\_editGrid = True

        self.\_returnToMenu = False

    #endsub

    def getHeight(self):

        return self.\_\_HEIGHT

    #endsub

    def getWidth(self):

        return self.\_\_WIDTH

    #endsub

    def getCurrentObject(self):

        return self.\_currentObject

    #endsub

    def setCurrentObject(self, new):

        self.\_currentObject = new

    #endsub

    def getScale(self):

        return self.\_\_SCALE

    #endsub

    def getTranslate(self):

        return self.\_\_TRANSLATE

    #endsub

    def getCellsList(self):

        return self.\_cellsList

    #endsub

    def setCellsList(self, new):

        self.\_cellsList = new

    #endsub

    def getAllCells(self):

        return self.\_allCells

    #endsub

    def setAllCells(self, new):

        self.\_allCells = new

    #endsub

    def getWalls(self):

        return self.\_walls

    #endsub

    def setWalls(self, new):

        self.\_walls = new

    #endsub

    def getGridArea(self):

        return self.\_gridArea

    #endsub

    def setGridArea(self, new):

        self.\_gridArea = new

    #endsub

    def getImages(self):

        return self.\_images

    #endsub

    def setImages(self, new):

        self.\_images = new

    #endsub

    def getEditGrid(self):

        return self.\_editGrid

    #endsub

    def setEditGrid(self, new):

        self.\_editGrid = new

    #endsub

    def getReturnToMenu(self):

        return self.\_returnToMenu

    #endsub

    def setReturnToMenu(self, new):

        self.\_returnToMenu = new

    #endsub

#endclass

#Displaying information

def portalPage():

    '''Display a page explaining why portals may not be discovered.'''

    portalWindow = tkinter.Tk()

    portalWindow.title('Maze-solving using A\*')

    portalWindow.geometry('900x600')

    portalWindow.configure(bg = '#ffffff')

    title = tkinter.Label(portalWindow, text = 'The Portal Problem', font = '"OCR A Extended" 20 ', fg = '#0015d4', bg = '#ffffff')

    title.place(relx = 0.05, rely = 0.05)

    text1 = tkinter.Text(portalWindow, font = 'Verdana 15', fg = '#000000', bg = '#ffffff', height = 100, width = 63, wrap = 'word', bd = 0)

    text1.insert('insert', 'Portals allow the algorithm to \'jump\' between two locations in the grid. This can save loads of time that would be needed to cross those cells normally. Super useful, right?\n\nWell, only if you make it so. You could position the portal out nowhere near the target, making it a complete waste. How evil.\n\nThe problem with portals is that the A\* algorithm only considers distance from the start and target cell. It wasn\'t designed to know about portals. The result of this, is that if the algorithm wouldn\'t have visited the portal in if it were a normal cell, it won\'t visit it. Even if the portal out is right next to the target cell. The algorithm doesn\'t check for portals at all, only using them if it manages to stumble across the portal in.\n\nThe downside of this is that occasionally, a maze is created where taking the portal would be much faster than the route found by the algorithm, however the algorithm didn\'t spot it. Why don\'t you try to create a situation like this?')

    text1.config(state = 'disabled')

    text1.tag\_add('algorithm', 5.37, 5.104)

    text1.tag\_configure('algorithm', font = 'Verdana 15 bold')

    text1.tag\_add('no visit', 5.257, 5.272)

    text1.tag\_configure('no visit', font = 'Verdana 15 bold')

    text1.place(relx = 0.05, rely = 0.12)

    backButton = tkinter.Button(portalWindow, text = 'Back', font = '"OCR A Extended" 15', fg = '#000000', bg = '#e2e2e2', activebackground = '#cccccc', relief = 'raised', padx = 2, pady = 2, bd = 4, command = lambda:portalWindow.destroy())

    backButton.place(relx = 0.87, rely = 0.87)

    portalWindow.mainloop()

#endsub

def infoPage():

    '''Display a page giving information on what the algoithm is and how it works'''

    infoWindow = tkinter.Tk()

    infoWindow.title('Maze-solving using A\*')

    infoWindow.geometry('1200x800')

    infoWindow.configure(bg = '#ffffff')

    title1 = tkinter.Label(infoWindow, text = 'What\'s going on?', font = '"OCR A Extended" 20 ', fg = '#0015d4', bg = '#ffffff')

    title1.place(relx = 0.03, rely = 0.03)

    title2 = tkinter.Label(infoWindow, text = 'How does it work?', font = '"OCR A Extended" 20 ', fg = '#0015d4', bg = '#ffffff')

    title2.place(relx = 0.43, rely = 0.03)

    text1 = tkinter.Text(infoWindow, font = '"Verdana" 15', fg = '#000000', bg = '#ffffff', height = 100, width = 30, wrap = 'word', bd = 0)

    text1.insert(tkinter.INSERT, 'This page allows you to explore the A\* pathfinding algorithm – a method of finding the shortest path between two cells. You can add objects to the grid by clicking their icon, then clicking on a cell in the grid. Click again to remove it.\n\n - Add a start and target cell, then watch the algorithm calculate the shortest route between them.\n\n - Walls are uncrossable – the algorithm will have to find its way around them. Try making a maze!\n\n - Puddles take longer to cross than empty cells. Is it faster to just go around?\n\n - Portals allow you to warp across the grid, from the portal in to the portal out.\n\nWhen you’re happy with your maze, hit ‘Run A\*’ to see the algorithm find the optimal solution.')

    text1.config(state='disabled')

    #colour certain words

    text1.tag\_add('start', 3.8, 3.14)

    text1.tag\_configure('start', foreground = '#00ee00')

    text1.tag\_add('target', 3.18, 3.25)

    text1.tag\_configure('target', foreground = '#ee0000')

    text1.tag\_add('walls', 5.3, 5.8)

    text1.tag\_configure('walls', foreground = '#505050')

    text1.tag\_add('puddles', 7.3, 7.11)

    text1.tag\_configure('puddles', foreground = '#663300')

    text1.tag\_add('portals', 9.3, 9.11)

    text1.tag\_configure('portals', foreground = '#b800e6')

    text1.tag\_add('portal in', 9.54, 9.64)

    text1.tag\_configure('portal in', foreground = '#cc99ff')

    text1.tag\_add('portal in', 9.54, 9.64)

    text1.tag\_configure('portal in', foreground = '#cc99ff')

    text1.tag\_add('portal out', 9.72, 9.82)

    text1.tag\_configure('portal out', foreground = '#6600cc')

    text1.tag\_add('run a\*', 11.39, 11.45)

    text1.tag\_configure('run a\*', foreground = '#1aff9c')

    text1.place(relx = 0.03, rely = 0.08)

    line = tkinter.Canvas(infoWindow, bg = '#000000', height = 750, width = 2)

    line.place(relx = 0.4, rely = 0.03)

    text2 = tkinter.Text(infoWindow, font = '"Verdana" 15', fg = '#000000', bg = '#ffffff', height = 100, width = 50, wrap = 'word', bd = 0)

    text2.insert('insert', 'Beginning at the start cell, the algorithm calculates a total cost to travel to each of the cell’s neighbours. This cost is as follows:\n\n\n\n\n\n\nThe distance to the end cell is an estimate as there may be walls in the way. These neighbour cells are added to the frontier – a list of cells with a calculated cost that haven\'t been visited, indicated in light blue. Also, the cell that came before them (in this case, the start cell) is recorded. Next, the cell in the frontier with the lowest cost is picked from the frontier, so the algorithm prioritises directions with a lower cost. This cell’s own neighbours then have a cost calculated and are added to the frontier, and the cell itself is marked as visited – shown in dark blue. \n\nThese steps are repeated until the target cell is reached. The algorithm then checks which cell came before the target, then which cell came before that cell, and so on until the start cell is reached. These are the cells that are part of the shortest route, which is illustrated in yellow.')

    text2.config(state = 'disabled')

    text2.tag\_add('light blue', 8.207, 8.217)

    text2.tag\_configure('light blue', foreground = '#33ccff')

    text2.tag\_add('dark blue', 8.578, 8.587)

    text2.tag\_configure('dark blue', foreground = '#0000e6')

    text2.tag\_add('yellow', 10.282, 10.289)

    text2.tag\_configure('yellow', foreground = '#ffff00')

    text2.place(relx = 0.43, rely = 0.08)

    text3 = tkinter.Label(infoWindow, text = 'Total cost to cell\n=\nTotal distance from start cell\n+\nEstimated total distance to end cell', font = '"Consolas" 15', fg = '#000000', bg = '#9999ff', bd = 0, justify = 'center')

    text3.place(relx = 0.5, rely = 0.17)

    portalButton = tkinter.Button(infoWindow, text = 'The Portal Problem', font = '"OCR A Extended" 15', fg = '#000000', bg = '#e2e2e2', activebackground = '#cccccc', relief = 'raised', padx = 2, pady = 2, bd = 4, command = portalPage)

    portalButton.place(relx = 0.43, rely = 0.9)

    backButton = tkinter.Button(infoWindow, text = 'Back', font = '"OCR A Extended" 15', fg = '#000000', bg = '#e2e2e2', activebackground = '#cccccc', relief = 'raised', padx = 2, pady = 2, bd = 4, command = lambda:infoWindow.destroy())

    backButton.place(relx = 0.9, rely = 0.9)

    infoWindow.mainloop()

#endsub

#Clearing the grid

def clearGrid(variables):

    '''Clear the canvas so the user can start over'''

    if variables.getEditGrid() == True:

        allCells = variables.getAllCells()

        for cell in allCells:

            oldColour = cell.getColour() #colour is used to reset blue and yellow cells as well as non-empty cells

            if oldColour != '#f0f0f0':

                #replace the cell instance with that of an empty cell

                xy = cell.getxy()

                allCells.remove(cell)

                del cell

                position = variables.getCellsList().index([xy[0], xy[1]])

                cell = Cell(xy[0], xy[1], variables.getHeight(), variables.getWidth(), {'background': '#f0f0f0'})

                allCells.insert(position, cell)

                variables.setAllCells(allCells)

                drawCell(cell, variables) #redraw the empty cell

            #endif

        #endfor

    #endif

    variables.setWalls([]) #remove all walls

#endsub

#Creating the grid

def drawCell(cell, variables):

    '''Given a cell, draw it to the grid'''

    xy = cell.getxy()

    topleft = (xy[0] -0.5, xy[1] -0.5)

    bottomright = (xy[0]+0.5, xy[1]+0.5)

    colour = cell.getColour()

    gridArea = variables.getGridArea()

    images = variables.getImages()

    if type(colour) == str:

        gridArea.create\_rectangle(topleft[0]\*variables.getScale()+variables.getTranslate(), topleft[1]\*variables.getScale()+variables.getTranslate(), bottomright[0]\*variables.getScale()+variables.getTranslate(), bottomright[1]\*variables.getScale()+variables.getTranslate(), fill = cell.getColour(), outline = '#000000')

    else:

        colour = colour.zoom(3, 3)

        colour = colour.subsample(2, 2)

        position = variables.getAllCells().index(cell)

        images[position] = colour

        gridArea.create\_image(topleft[0]\*variables.getScale()+variables.getTranslate(), topleft[1]\*variables.getScale()+variables.getTranslate(), image = images[position], anchor = 'nw')

    #endif

    variables.setGridArea(gridArea)

    variables.setImages(images)

#endsub

def addObject(event, variables):

    '''Add the selected cell type to the grid'''

    if variables.getEditGrid() == True: #don't make changes while the algorithm is running

        #Find the position of the mouse

        mousex = int(event.x)

        mousey = int(event.y)

        mousex /= variables.getScale() #Descale these coordinates so they fit within a cell

        mousey /= variables.getScale()

        cellx = floor(mousex) #get the integer value

        celly = floor(mousey)

        cellPosition = variables.getCellsList().index([cellx, celly]) #Find the corresponding Cell object

        cell = variables.getAllCells()[cellPosition]

        #Replace this cell with one of the chosen type

        oldType = cell.getType()

        xy = cell.getxy()

        allCells = variables.getAllCells()

        position = allCells.index(cell)

        allCells.remove(cell)

        del cell

        walls = variables.getWalls()

        images = variables.getImages()

        newCell = Cell(xy[0], xy[1], variables.getHeight(), variables.getWidth(), {'background': '#f0f0f0'}) #the default is an empty cell

        if oldType == variables.getCurrentObject():

            images[position] = '' #reset the image at this position to empty

            newCell = Cell(xy[0], xy[1], variables.getHeight(), variables.getWidth(), {'background': '#f0f0f0'})

            #Remove the walls if appropriate

            if oldType == 'wall':

                walls.remove(newCell.getEdges()[0])

                walls.remove(newCell.getEdges()[1])

                walls.remove(newCell.getEdges()[2])

                walls.remove(newCell.getEdges()[3])

            #endif

        else:

            #first reset the cell's colour

            drawCell(newCell, variables)

            if variables.getCurrentObject() == 'wall':

                newCell = Wall(xy[0], xy[1], variables.getHeight(), variables.getWidth(), {'background': '#f0f0f0'})

                #add a wall for each side of this cell

                walls.append(newCell.getEdges()[0])

                walls.append(newCell.getEdges()[1])

                walls.append(newCell.getEdges()[2])

                walls.append(newCell.getEdges()[3])

            elif variables.getCurrentObject() == 'start':

                found, c = alreadyPlaced('start', variables)

                if found == False: #there can only be one start cell

                    newCell = StartCell(xy[0], xy[1], variables.getHeight(), variables.getWidth(), {'background': '#f0f0f0'})

                else:

                    messagebox.showwarning(title = 'Maze solving using A\*', message = 'You can only place one start cell.')

                #endif

            elif variables.getCurrentObject() == 'end':

                found, c = alreadyPlaced('end', variables)

                if found == False:

                    newCell = EndCell(xy[0], xy[1], variables.getHeight(), variables.getWidth(), {'background': '#f0f0f0'})

                else:

                    messagebox.showwarning(title = 'Maze solving using A\*', message = 'You can only place one target cell.')

                #endif

            elif variables.getCurrentObject() == 'puddle':

                newCell = Puddle(xy[0], xy[1], variables.getHeight(), variables.getWidth(), {'background': '#f0f0f0'})

            elif variables.getCurrentObject() == 'warp in':

                found, c = alreadyPlaced('warp in', variables)

                if found == False:

                    newCell = WarpIn(xy[0], xy[1], variables.getHeight(), variables.getWidth(), {'background': '#f0f0f0'})

                else:

                    messagebox.showwarning(title = 'Maze solving using A\*', message = 'There can only be one portal system.')

                #endif

            elif variables.getCurrentObject() == 'warp out':

                found, c = alreadyPlaced('warp out', variables)

                if found == False:

                    newCell = WarpOut(xy[0], xy[1], variables.getHeight(), variables.getWidth(), {'background': '#f0f0f0'})

                else:

                    messagebox.showwarning(title = 'Maze solving using A\*', message = 'There can only be one portal system.')

                #endif

            #endif

        #endif

        allCells.insert(cellPosition, newCell)

        variables.setAllCells(allCells)

        variables.setImages(images)

        variables.setWalls(walls)

        drawCell(newCell, variables) #display the new cell on screen

    #endif

#endsub

#Checking the grid is valid

def alreadyPlaced(desiredType, variables):

    '''Check whether a certain type of cell has been placed on the grid and return it, using a linear search'''

    found = False

    for cell in variables.getAllCells():

        if cell.getType() == desiredType:

            found = True

            return found, cell #the loop ends as the cell was found

        #endif

    #endfor

    return found, -1

#endsub

def checkGrid(variables, AStarWindow):

    '''Check that the user's grid is valid (has a start and end cell and both or no portals)'''

    variables.setEditGrid(False) #so the user cannot make changes while the algorithm is running

    startPresent, startCell = alreadyPlaced('start', variables)

    endPresent, endCell = alreadyPlaced('end', variables)

    warpInPresent, warpInCell = alreadyPlaced('warp in', variables)

    warpOutPresent, warpOutCell = alreadyPlaced('warp out', variables)

    #produce an error message if the grid is invalid

    if startPresent == False:

        messagebox.showwarning(title = 'Maze solving using A\*', message = 'The grid is missing a start cell.')

    elif endPresent == False:

        messagebox.showwarning(title = 'Maze solving using A\*', message = 'The grid is missing a target cell.')

    elif warpInPresent == True and warpOutPresent == False:

        messagebox.showwarning(title = 'Maze solving using A\*', message = 'The portal needs an exit.')

    elif warpInPresent == False and warpOutPresent == True:

        messagebox.showwarning(title = 'Maze solving using A\*', message = 'The portal needs an entrance.')

    else:

        runAStar(variables, startCell, endCell, warpOutCell, AStarWindow)

    #endif

    variables.setEditGrid(True)

#endsub

#Running the A\* algorithm

def wait(AStarWindow):

    '''Used to add a break between displaying changes to the frontier'''

    var = tkinter.IntVar()

    AStarWindow.after(50, var.set, 1)

    AStarWindow.wait\_variable(var)

#endsub

def runAStar(variables, startCell, endCell, warpOutCell, AStarWindow):

    '''Use the A\* algorithm to find the solution, and display it on the grid.'''

    #Reset cells whose colour was changed in a previous run

    for cell in variables.getAllCells():

        oldColour = cell.getColour()

        if oldColour != '#f0f0f0':

            cell.setColour('#f0f0f0')

            drawCell(cell, variables)

            if cell.getType() != 'empty': #redraw the sprite on top of the blank square if the cell was not empty

                cell.setColour(oldColour)

                drawCell(cell, variables)

            #endif

        #endif

    #endfor

    #Calculate the fastest route through the grid

    shortestRoute, colourChanges = AStar(startCell, endCell, variables.getWalls(), variables.getCellsList(), variables.getAllCells(), warpOutCell)

    #Change each cell's colour as specified in the AStar subroutine

    for change in colourChanges: #change is in the form (cell, colour)

        cell = change[0]

        oldColour = cell.getColour()

        cell.setColour(change[1])

        drawCell(cell, variables)

        if cell.getType() != 'empty': #redraw the object on top of the new colour

            cell.setColour(oldColour)

            drawCell(cell, variables)

        #endif

        wait(AStarWindow)

    #endfor

    #Draw the solution in yellow

    if shortestRoute != -1: #if the algorithm did not fail

        for cell in shortestRoute:

            position = variables.getCellsList().index([cell[0], cell[1]])

            cell = variables.getAllCells()[position]

            if cell.getType() == 'empty':

                cell.setColour('#ffff00')

                drawCell(cell, variables)

            else:

                oldImage = cell.getColour()

                cell.setColour('#ffff00')

                drawCell(cell, variables)

                cell.setColour(oldImage)

                drawCell(cell, variables)

            #endif

            #change the colour of the portal out if part of the fastest route (as it isn't in solution)

            if cell.getType() == 'warp in':

                oldImage = warpOutCell.getColour()

                warpOutCell.setColour('#ffff00')

                drawCell(warpOutCell, variables)

                warpOutCell.setColour(oldImage)

                drawCell(warpOutCell, variables)

            #endif

            wait(AStarWindow)

        #endfor

    #endif

#endsub

#Exiting A\* Section

def returnToMenu(window, variables):

    if variables.getEditGrid() == True:

        window.destroy()

        variables.setReturnToMenu(True)

    #endif

#endsub

#Main window

def main(variables):

    '''The main subroutine for A\* Simulation, used to display the page and handle user input'''

    #Produce the background window

    AStarWindow = tkinter.Tk()

    AStarWindow.title('Maze-solving using A\*')

    AStarWindow.geometry('1200x700')

    AStarWindow.configure(bg = '#ffffff')

    title = tkinter.Label(AStarWindow, text = 'A\* Pathfinding Visualisation', font = '"OCR A Extended" 22 underline', fg = '#000000', bg = '#ffffff')

    title.place(relx = 0.28, rely = 0.015)

    gridArea = tkinter.Canvas(AStarWindow, bd = 0, bg = '#f0f0f0', height = 471, width = 1129 )

    gridArea.place(relx = 0.03, rely = 0.08)

    variables.setGridArea(gridArea)

    wallButton = tkinter.Button(AStarWindow, bg = '#000000', activebackground = '#303030' , height = 4, width = 9, command = lambda: variables.setCurrentObject('wall'))

    wallButton.place(relx = 0.03, rely = 0.87)

    startImage = tkinter.PhotoImage(file='startcell.png').zoom(2,2)

    startButton = tkinter.Button(AStarWindow, image= startImage, height = 65, width = 65, bd = 1, command = lambda: variables.setCurrentObject('start'))

    startButton.place(relx = 0.12, rely = 0.87)

    endImage = tkinter.PhotoImage(file='endcell.png').zoom(2,2)

    endButton = tkinter.Button(AStarWindow, image= endImage, height = 65, width = 65, bd = 1, command = lambda: variables.setCurrentObject('end'))

    endButton.place(relx = 0.21, rely = 0.87)

    puddleImage = tkinter.PhotoImage(file = 'puddle.png').zoom(2,2)

    puddleButton = tkinter.Button(AStarWindow, image= puddleImage, height = 65, width = 65, bd = 1, command = lambda: variables.setCurrentObject('puddle'))

    puddleButton.place(relx = 0.3, rely = 0.87)

    warpInImage = tkinter.PhotoImage(file = 'warpin.png').zoom(2,2)

    warpInButton = tkinter.Button(AStarWindow, image= warpInImage, height = 65, width = 65, bd = 1, command = lambda: variables.setCurrentObject('warp in'))

    warpInButton.place(relx = 0.39, rely = 0.87)

    warpOutImage = tkinter.PhotoImage(file = 'warpout.png').zoom(2,2)

    warpOutButton = tkinter.Button(AStarWindow, image= warpOutImage, height = 65, width = 65, bd = 1, command = lambda: variables.setCurrentObject('warp out'))

    warpOutButton.place(relx = 0.48, rely = 0.87)

    wallLabel = tkinter.Label(AStarWindow, text = 'wall', font = '"OCR A Extended" 17', fg = '#000000', bg = '#ffffff')

    wallLabel.place(relx = 0.035, rely = 0.815)

    startLabel = tkinter.Label(AStarWindow, text = 'start', font = '"OCR A Extended" 17', fg = '#000000', bg = '#ffffff')

    startLabel.place(relx = 0.12, rely = 0.815)

    endLabel = tkinter.Label(AStarWindow, text = 'target', font = '"OCR A Extended" 17', fg = '#000000', bg = '#ffffff')

    endLabel.place(relx = 0.204, rely = 0.815)

    puddleLabel = tkinter.Label(AStarWindow, text = 'puddle', font = '"OCR A Extended" 17', fg = '#000000', bg = '#ffffff')

    puddleLabel.place(relx = 0.295, rely = 0.815)

    warpInLabel = tkinter.Label(AStarWindow, text = 'portal\nin', font = '"OCR A Extended" 17', fg = '#000000', bg = '#ffffff')

    warpInLabel.place(relx = 0.383, rely = 0.785)

    warpOutLabel = tkinter.Label(AStarWindow, text = 'portal\nout', font = '"OCR A Extended" 17', fg = '#000000', bg = '#ffffff')

    warpOutLabel.place(relx = 0.473, rely = 0.785)

    run = tkinter.Button(AStarWindow,  text = 'Run A\*', font = '"OCR A Extended" 27', bg = '#ccffe0', activebackground = '#99ffc2', fg = '#000000', relief = 'raised', padx = 15, pady = 20, bd = 6, command = lambda: checkGrid(variables, AStarWindow))

    run.place(relx = 0.586, rely = 0.8)

    clear = tkinter.Button(AStarWindow,  text = 'Clear', font = '"OCR A Extended" 17', bg = '#e2e2e2', activebackground = '#cccccc', fg = '#000000', relief = 'raised', padx = 2, pady = 2, bd = 4, command = lambda: clearGrid(variables))

    clear.place(relx = 0.77, rely = 0.88)

    info = tkinter.Button(AStarWindow,  text = 'What\'s going on?', font = '"OCR A Extended" 17', bg = '#e2e2e2', activebackground = '#cccccc', fg = '#000000', relief = 'raised', padx = 2, pady = 2, bd = 4, command = infoPage)

    info.place(relx = 0.77, rely = 0.8)

    menu = tkinter.Button(AStarWindow,  text = 'Menu', font = '"OCR A Extended" 17', bg = '#e2e2e2', activebackground = '#cccccc', fg = '#000000', relief = 'raised', padx = 2, pady = 2, bd = 4, command = lambda:returnToMenu(AStarWindow, variables))

    menu.place(relx = 0.91, rely = 0.88)

    #Fill the grid with empty cells

    cellsList = variables.getCellsList()

    allCells = variables.getAllCells()

    images = variables.getImages()

    for x in range(variables.getWidth()):

        for y in range(variables.getHeight()):

            cellsList.append([x, y])

            allCells.append(Cell(x, y, variables.getHeight(), variables.getWidth(), {'background': '#f0f0f0'}))

            images.append('')

        #endfor

    #endfor

    for cell in allCells:

        drawCell(cell, variables)

    #endfor

    variables.setCellsList(cellsList)

    variables.setAllCells(allCells)

    variables.setImages(images)

    #Detect when the user clicks on the grid

    gridArea.bind('<Button-1>', lambda event: addObject(event, variables))

    variables.setGridArea(gridArea)

    AStarWindow.mainloop()

#endsub

#Initialising A\* Simulation

def navigateAStarSimulation():

    variables = AStarClass() #Tkinter buttons do not allow variables to be returned. To avoid this issue I have made each variable an attribute of 'AStarClass', so that they can be accessed, and more importantly, edited by any subroutine where 'variables' is passed in.

    main(variables)

    return variables.getReturnToMenu()

#endsub