A Project Documentation Submitted for AQA A-Level Computer Science Non-Exam Assessment (NEA)

**A Pathfinding Algorithm Simulator:**

**Design and Implementation**

|  |  |  |
| --- | --- | --- |
| **Centre Number** | **：** | **XXXXX** |
| **Candidate Number** | **：** | **XXXX** |
| **Candidate Name** | **：** | **Yanbo Leng** |



**Bodiam, Robertsbridge TN32 5UJ**

**February, 2024**

# Abstract

The simulator integrates Randomised Prim’s Algorithm to create dynamically generated mazes to visualise and analyse the efficiency of different pathfinding algorithms, including A\*, Dijkstra, Depth-First and Breadth-First search in such a manner that the user can choose whether to complete the maze manually or automatically. It also includes performance analysis tools for comparing algorithm efficiency and a login & registration system to track the user’s maze exploration progress.

**Keywords:** Pathfinding Algorithms, Randomised Prim's Algorithm, Maze Generation, Algorithm Efficiency Analysis, A\* Search, Dijkstra's Algorithm, Depth-First Search, Breadth-First Search, Performance Analysis, User Interaction, Maze Exploration, Educational Tool, Login System, User Progress Tracking

Table of Contents

[Abstract 1](#_Toc159509272)

[1. Analysis 4](#_Toc159509273)

[1.1 Background 4](#_Toc159509274)

[1.2 Initial Research 5](#_Toc159509275)

[1.2.1 Interviews 5](#_Toc159509276)

[1.2.2 Similar Products 7](#_Toc159509277)

[1.3 Further Research 8](#_Toc159509278)

[1.3.1 Maze Generating Algorithm Selection 8](#_Toc159509279)

[1.3.2 Comparative Data Selection for Pathfinding Algorithms 11](#_Toc159509280)

[1.4 End-user 11](#_Toc159509281)

[1.4.1 Prospective Users 11](#_Toc159509282)

[1.4.2 User Needs 12](#_Toc159509283)

[1.5 Objectives 12](#_Toc159509284)

[1.5.1 Introduction 12](#_Toc159509285)

[1.5.2 Final Objective List 12](#_Toc159509286)

[1.6 Solutions 16](#_Toc159509287)

[2. Documented Design 18](#_Toc159509288)

[2.1 System Design 18](#_Toc159509289)

[2.1.1 Overall System Flowchart 18](#_Toc159509290)

[2.1.2 Maze and Pathfinding System Flowchart 19](#_Toc159509291)

[2.1.3 Player Movement and Control Flowchart 20](#_Toc159509292)

[2.2 Algorithms 20](#_Toc159509293)

[2.2.1 Randomised Prim's Algorithm 20](#_Toc159509294)

[2.2.2 Depth-first Search 22](#_Toc159509295)

[2.2.3 Breadth-first Search 23](#_Toc159509296)

[2.2.4 Dijkstra’s Algorithm 24](#_Toc159509297)

[2.2.5 A\* Search 25](#_Toc159509298)

[2.3 Data Structures 25](#_Toc159509299)

[2.3.1 Dynamic Array 25](#_Toc159509300)

[2.3.2 Stack 26](#_Toc159509301)

[2.3.3 Queue 27](#_Toc159509302)

[2.3.4 Priority Queue (Min Heap) 27](#_Toc159509303)

[2.3.5 Set 27](#_Toc159509304)

[2.3.6 Enumerations (Enum) 27](#_Toc159509305)

[2.3.7 2D Tuple 28](#_Toc159509306)

[2.3.8 List 28](#_Toc159509307)

[2.4 Database Design 28](#_Toc159509308)

[2.4.1 ER Diagram 28](#_Toc159509309)

[2.4.2 Data Dictionaries 28](#_Toc159509310)

[2.4.3 SQL Statements 29](#_Toc159509311)

[2.4.4 Normalisation 30](#_Toc159509312)

[2.4.5 DFD Diagram 30](#_Toc159509313)

[2.5 Object-Oriented Design 30](#_Toc159509314)

[2.5.1 Class Description 30](#_Toc159509315)

[2.5.2 UML Class Diagram 33](#_Toc159509316)

[2.6 GUI Design 34](#_Toc159509317)

[2.6.1 Main Menu 34](#_Toc159509318)

[2.6.2 Common UI 35](#_Toc159509319)

[2.6.3 Maze and Pathfinding Interface 36](#_Toc159509320)

[2.6.4 Congratulations Interface 38](#_Toc159509321)

[2.6.5 Game Level Interface 39](#_Toc159509322)

[2.6.6 Login/Register Page 40](#_Toc159509323)

[2.6.7 Secondary Menu 43](#_Toc159509324)

[2.7 File Structure 44](#_Toc159509325)

[2.7.1 File Arrangement 44](#_Toc159509326)

[2.7.2 Resources Organisation 45](#_Toc159509327)

[2.7.3 Additional Considerations 46](#_Toc159509328)

[3. Testing 47](#_Toc159509329)

[3.1 Compilation Environment 47](#_Toc159509330)

[3.2 Overall System Test Table 47](#_Toc159509331)

[3.2.1 User Interface Testing (Objective 1) 47](#_Toc159509332)

[3.2.2 Maze Generation Testing (Objective 2) 48](#_Toc159509333)

[3.2.3 Manual Exploration Mode Testing (Objective 3) 49](#_Toc159509334)

[3.2.4 Process and Comparative Data Visualisation Testing (Objectives 4 & 5 & 6) 50](#_Toc159509335)

[3.2.5 Login and Register System Testing (Objective 7) 52](#_Toc159509336)

[3.2.6 Maze Level Selection Menu Testing (Objective 8) 53](#_Toc159509337)

[3.3 Major Bugs and Fixes 54](#_Toc159509338)

[3.4 Testing Search Algorithm Efficiency 56](#_Toc159509339)

[3.4.1 Purpose of Testing 56](#_Toc159509340)

[3.4.2 Testing Environment and Tools 56](#_Toc159509341)

[3.4.3 Testing Method 56](#_Toc159509342)

[3.4.4 Results Table 56](#_Toc159509343)

[3.4.5 Summary of Test Results 57](#_Toc159509344)

[4. Technical Solution 59](#_Toc159509345)

[4.1 Index Table 59](#_Toc159509346)

[4.2 System Architecture and Modular Design 62](#_Toc159509347)

[4.3 Source Code 63](#_Toc159509348)

[4.3.1 main.py 63](#_Toc159509349)

[4.3.2 maze.py 77](#_Toc159509350)

[4.3.3 player.py 81](#_Toc159509351)

[4.3.4 pathfinder.py 82](#_Toc159509352)

[4.3.5 authentication.py 89](#_Toc159509353)

[4.3.6 game\_level\_interface.py 96](#_Toc159509354)

[5. Evaluation 100](#_Toc159509355)

[5.1 Evaluation against Objectives 100](#_Toc159509356)

[5.2 Conclusion 106](#_Toc159509357)

[Appendix 106](#_Toc159509358)

[I. References 106](#_Toc159509359)

[II. Image Source 107](#_Toc159509360)

# 1. Analysis

## 1.1 Background

My interest in pathfinding algorithms as an academic project was first sparked as a result of a coding challenge from [codewars.com](https://www.codewars.com/) in the form of “Optimised Pathfinding Algorithm.” I began to consider what “pathfinding” really meant and have concluded that it is not merely a series of computational steps but a crucial element of efficient route optimisation within various scenarios, such as artificial intelligence, robotics and gaming. Given that this topic has real-life significance and has drawn my interest, I decided to delve deeper into it at a higher level.

Drawing on my studies in pathfinding, I have sought to broaden my understanding of different pathfinding algorithms, such as Dijkstra’s, depth-first, breadth-first and A\* Search, and their efficiency and performance. However, there is a lack of interactive and engaging tools that allow users to visualise and compare these algorithms in action. Moreover, most existing tools only use static and simple mazes, which do not reflect the complexity and dynamism of real-world scenarios.

Thus, I look forward to developing a tool to perform a comparative analysis of pathfinding algorithms in dynamic and complex mazes, including showing the principles that underlie pathfinding algorithms in different environmental complexities, will entail.

## 1.2 Initial Research

### 1.2.1 Interviews

#### Interview 1

Date: *11/27/23*

Interviewee: *Alex T.*

***1) Can you tell me your background and experience with pathfinding algorithms?***

* I worked in the video game development scene for about five years. I worked on a mix of indie and AAA titles, where pathfinding algorithms are often used to create NPC behaviours.

***2) In the context of game development, how crucial are pathfinding algorithms, and what challenges have you encountered?***

* In many games, especially MMORPG games, NPCs have the function of automatic pathfinding, which can save a lot of recording game coordinate points as well as long-term keyboard operations during the game while ensuring NPCs can move to the designated position as quickly as possible. One of the challenges we encounter is to try to improve the efficiency of the pathfinding algorithm. You want NPCs to move realistically, but you also can't afford to sacrifice too much processing power.

***3) How do you typically approach integrating and testing pathfinding algorithms in a game development environment?***

* It depends on the game's scale and requirements. We might use algorithms like A\* to calculate paths for more giant open-world games. We usually run extensive testing to ensure NPCs navigate smoothly in various environments and scenarios.

***4) How important is it to have a tool to visualise and compare these algorithms?***

* This is very valuable. Visualisation helps not only with debugging but also with understanding how the algorithm behaves in different situations. This works like a debugging assistant, allowing you to see if an NPC has taken an unexpected route or gotten stuck.

***5) What specific features or functionality would you like to see in a pathfinding simulator to make life as a game developer easier?***

* You may want to add a function that can visually compare data between different algorithms, such as maze size, pathfinding time completed by the pathfinding algorithm, number of nodes explored, etc., so that users can compare the efficiency of different pathfinding algorithms.

#### Interview 2

Date: *12/11/23*

Interviewee: *Aiden S.*

***1) Can you tell me your background and experience with pathfinding algorithms?***

* I am a GCSE Computer Science student and have had some classes on basic data structures and algorithms. I remember our teacher taught us about Dijkstra’s shortest path algorithm. She introduced the computational steps on the board, but everything was so abstract. Trying to comprehend the pseudocode made it feel like I was decoding an alien language.

***2) Did you have any tools or resources that helped you understand it?***

* Not as much as I would have liked. We had some videos to show basic visualisation, but they were dull and obscure. I don’t like passively studying. I particularly hope that an educational tool allows us to explore the underlying concepts of knowledge actively.

***3) How would you envision this educational tool, and what features should it allow you to understand the concepts of different pathfinding algorithms better?***

* First, it should feature visualisations of the implementation of individual pathfinding algorithms, which can help students understand how they work. Furthermore, it should have some factors that users can interact with, such as whether they can explore the maze manually. Not only will this make the entire tool more interesting, but it will also allow users to understand why these pathfinding algorithms exist.

#### Analysis of Interviews

The interview with Alex confirmed the significance of visualising algorithms. In addition, to help programmers choose appropriate algorithms in different situations further, he pointed out that I can also visualise the maze-solving results of varying pathfinding algorithms in mazes of various sizes. So, I will add this functionality to my emulator.

The interview with Aiden made me realise that in addition to allowing our simulator to help programmers choose the most appropriate pathfinding algorithm in different situations, I can also add some specific functions so that this simulator can also help students better understand the principles of pathfinding algorithms. For example, as Aiden suggests, allow users to navigate mazes manually.

In addition, I can also add a database to store user information. Users can register their accounts and save progress from improving the user experience.

### 1.2.2 Similar Products

#### Product 1

<https://emmilco.github.io/path_finder/> by Elliot Milco is a website that demonstrates and introduces different algorithms for making and solving mazes. It not only provides visualisations of a range of pathfinding algorithms, including depth-first, breadth-first, Dijkstra’s and A\* search, but also their maze-solving results, as shown in Figure 1:

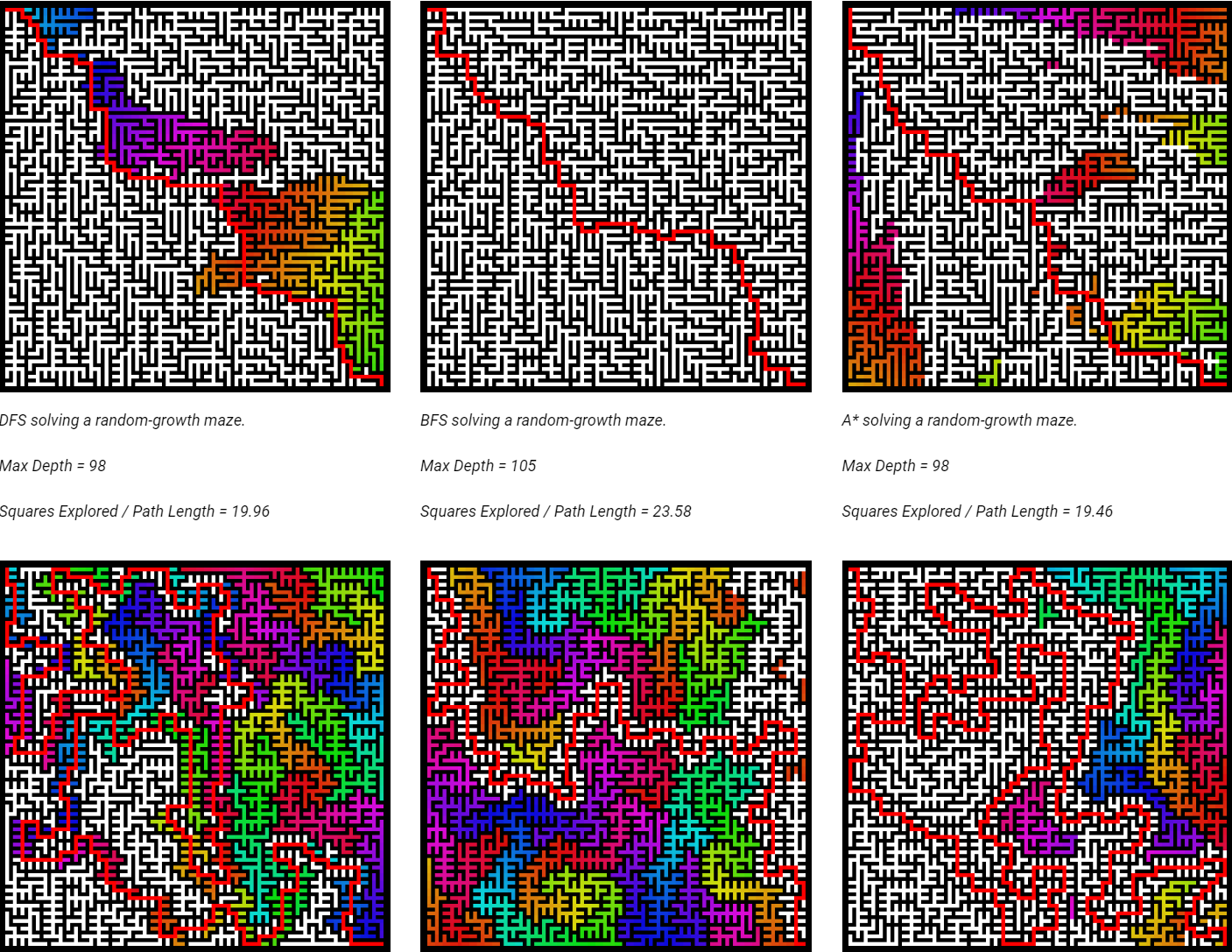


Figure 1

However, I find it challenging to draw convincing conclusions based on the information provided on this website. Firstly, as shown in Figure 1, although each algorithm has maze-solving results, the experimental variables are not controlled: each algorithm performs on different mazes. Secondly, the maze size is fixed, and the complexity and size of the maze may impact the efficiency of different algorithms.

#### Analysis of Products

I should strictly control the variables in my simulator and compare different pathfinding algorithms in the same environment. Also, my maze complexity should not be static, and I should provide mazes of varying complexity to reflect the varying complexity of real-life problems.

## 1.3 Further Research

### 1.3.1 Maze Generating Algorithm Selection

Creating the maze environment is also one of the primary challenges in my project. The generated maze environment must be complex and fair, and it should also avoid excessive space or accumulation of obstacles while minimising the occurrence of dead ends or impassable paths. To solve this problem, I mainly studied three representative maze generation algorithms: recursive backtracker (a.k.a. depth-first algorithm), randomised prim’s algorithm and recursive division.

#### Recursive Backtracker

***Time Complexity:*** O(V+E), where V is the number of vertices (cells), and E is the number of edges (walls).

***Space Complexity:*** O(V) for the visited array.

***Principles:***

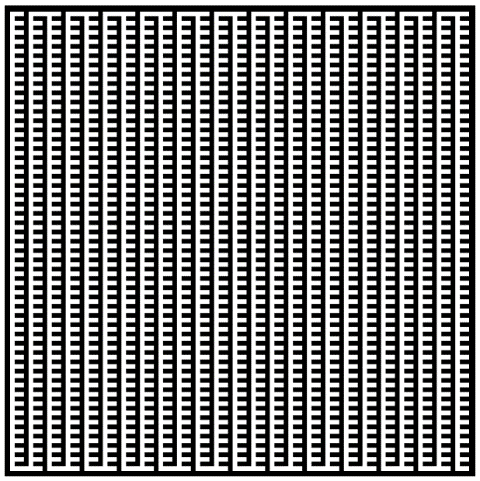
****The Recursive Backtracker Algorithm begins with exploring adjacent unvisited maze units of the starting point. When an unvisited path is found, it is prioritised, and the algorithm continues to explore further. If a dead-end is reached, the algorithm backtracks to the previous decision point and explores alternative paths.

Figure 2

The implementation idea of this algorithm is relatively simple. The situation in Figure 2 would appear if it were not a variant. The path is evident and unsuitable for a complex maze environment.

#### Randomised Prim’s Algorithm

***Time Complexity:*** O(ElogV)

***Space Complexity:*** O(V)

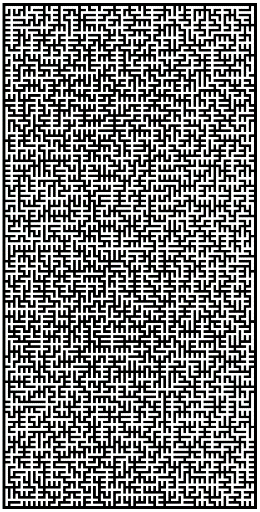
***Principles:*** Compared with the Recursive Backtracker Algorithm, the Randomised Prim’s Algorithm does not prioritise the recently selected maze units but randomly selects from the entire list of maze units.

Figure 3

This characteristic introduces a structure with more diverse paths, thus increasing complexity. This approach helps make the maze feel natural and interesting.

The flow chart of this algorithm is shown below:

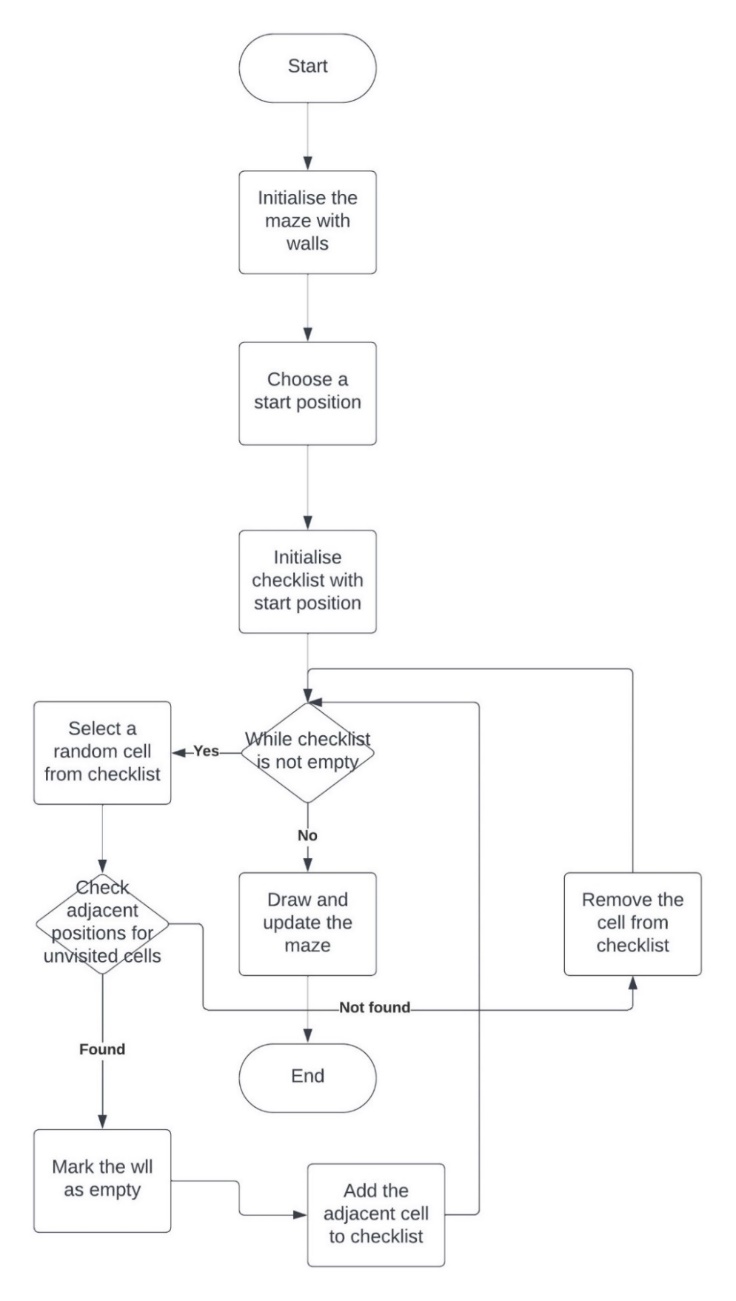


Figure 4

#### Recursive Division

***Time Complexity:*** O(VlogV),

***Space Complexity:*** O(V)

***Principles:*** The Recursive Division Algorithm initially divides the map into four sub-maps with a cross. To ensure connectivity, holes are dug on three walls in each sub-map. This process is then recursively applied to each resulting sub-map until a map is available for further division.

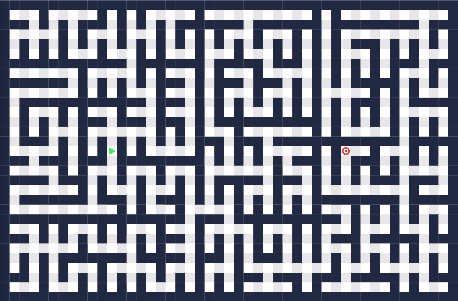
******Although this algorithm is very efficient, the generated maze has prominent rectangular components, many straight roads and untwisted paths, as shown below, so the generated maze is not fascinating compared with the Randomised Prim’s Algorithm.

Figure 5

#### Analysis of Algorithms

After analysis of these algorithms, it is obvious that each algorithm has its advantages and disadvantages.

From a complexity point of view, while Randomised Prim's Algorithm has a O(ElogV) time complexity, it might perform better than Recursive Backtracker regarding speed. This is beneficial when generating mazes of larger sizes. Besides, the space complexity of Randomised Prim's is O(V+E). Compared to Recursive Division, which relies heavily on the call stack, Prim's uses less memory.

From an algorithmic point of view, Recursive Backtracker tends to create very simple mazes with long and winding paths. Recursive Division splitting divides the maze into rectangular parts, creating many straight paths and walls. Randomised Prim’s Algorithm, on the other hand, tends to produce mazes with a balanced complexity distribution, providing a mix of open space and obstacles. Therefore, it is probably the best choice for maze generation in my project. This feature can provide a fair and challenging environment for comparing the efficiency of different pathfinding algorithms.

From an entertainment perspective, Recursive Backtracker and Recursive Division could generate less attractive mazes than Randomised Prim's. Since this tool is also educational, generating visually appealing mazes would be nice.

I used Randomised Prim’s Algorithm to create a maze environment for my project. Its time and space complexity ensures strong computing performance without affecting memory usage. Additionally, it can generate visually complex yet structured mazes, which aligns with this project's goals.

### 1.3.2 Comparative Data Selection for Pathfinding Algorithms

While visualising the steps of the pathfinding algorithm to satisfy computer students in learning the mechanism behind the algorithm, my project will also facilitate professional programmers to compare the efficiency of the pathfinding algorithm. In implementing the visual pathfinding algorithm, the solution results and efficiency of the pathfinding algorithm in the maze can be directly displayed on the screen in a table. Therefore, it is necessary to study the key indicators to determine the efficiency of a pathfinding algorithm.

The first thing I think of is the path length of the maze solution result. In other words, it is the length of the shortest path found by the algorithm because, in most virtual road algorithm scenarios, the meaning of pathfinding is to find the shortest path, such as in reality. In reality, there are navigation GPS, routing in logistics or NPC in games. Therefore, this indicator can help users make choices in specific circumstances to see which algorithm can find the shorter path.

However, just finding the shortest path without considering other factors is not enough. For example, an intuitive indicator is the time the algorithm takes to find the shortest path (a.k.a. execution time). This factor is especially important in applications where time is a critical factor. For example, in a real-time system, a faster algorithm may be preferred even if the algorithm is slightly less efficient in other aspects. Adding this metric can help determine which algorithm is best for a specific type of maze.

In addition, after research, the explored nodes can also be a good metric. It can compare the number of nodes visited by algorithms during the pathfinding process in environments of different complexity. If an algorithm can find the correct path by examining fewer nodes, it may mean that it considers weights like a greedy algorithm rather than traversing all nodes like a brute-force algorithm. This metric is crucial for understanding the nature of the algorithm and its scalability and adaptability to different complex environments.

## 1.4 End-user

### 1.4.1 Prospective Users

* Programmers who need a simulator to help them choose the most efficient pathfinding algorithms for different environmental complexities.
* Computer science students and educators who lack an interactive educational tool demonstrating the mechanisms behind various pathfinding algorithms.

### 1.4.2 User Needs

* Intuitive and easy-to-use interface
* Access to various pathfinding algorithms for comparison and selection
* Maze generation for varied complexity levels
* Clear visualisation of algorithmic processes for a better understanding
* Comparative analysis features to assess algorithm efficiency
* Access to performance data and analytics for deeper insights
* User account system for tracking progress and personalised experiences
* Manual exploration functionality for active engagement

## 1.5 Objectives

### 1.5.1 Introduction

My ultimate objective of this project is to make an attractive educational hub for visualising and analysing different pathfinding algorithms in dynamically generated mazes. This project aims to help users explore and learn different pathfinding algorithms. The main functions that achieve this goal are marked as success criteria in the table below.

To make this Hub more attractive and exciting. I want to add some creative features while meeting the needs of all users. I have marked the implementation of these features as sub-success criteria in the list below.

### 1.5.2 Final Objective List

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Objective | | Explanation | Success Criteria | | Sub-success Criteria | |
| **1.** | Intuitive and Attractive User Interface | Intuitive interface design allows easy interaction and caters to users with varying levels of expertise. | a | A minimalist-style main page, including three buttons, play, login and registration. | i | Add BGM to the background. |
| b | A clear login and register interface, including a title (indicating registration or login), a return to the main menu button, username and password input boxes and a submit button |
| c | When the user is logged in, a maze-level selection page will appear when entering the game. The buttons for the maze level will be updated as the player progresses in unlocking the levels. This page should also include a return button. The user can continue the game after clicking this button. | ii | Add sound effects when clicking buttons. |
| d | Streamlined controls for algorithm selection, maze level selection, and result observation. | iii | Add version number to the title of the main menu |
| iv | Add sound effects when the main menu's title animation is triggered. |
|  |  |  | e | A table comparing the efficiency of different pathfinding algorithms should be visible below the maze and the Select Pathfinding Algorithm buttons. | v | When the status is the congratulations menu, add a crowd cheer sound. |
| f | Add a congratulations menu. This page should pop up when the user manually controls the character to reach the target point or presses the skip button. This page should include a congratulations, a next-level button, and a return button. |
|  |  |  | g | When the user clicks Return in the game after logging in, a second menu should appear, including the functions of logging out and continuing the game. |  |  |
| **2.** | Dynamic Maze Generation Using Randomised Prim’s Algorithm | After researching and investigating different maze generation methods, I chose to use the randomised Prim algorithm, which can help create perfect mazes. Thus providing a complex and exciting environment for the comparison and visualisation of different pathfinding algorithms.  \* For 2. e), visualising the maze generation steps can help initialise the pathfinding algorithm data to display the data table of various pathfinding algorithms. Of course, this can also improve the program's aesthetics and attract more users. | a | Implement the Randomised Prim’s Algorithm to generate mazes of varying complexity and sizes, ensuring no two mazes are identical. |  |  |
| b | The generated map is a perfect maze (a.k.a. standard maze). It refers to a maze with no loops or unreachable positions. It is making sure that there is a path between the start point and the target point. |
| c | Allow users to select different complexity levels for maze generation. (Achieve different complexities by changing the maze size) |  |  |
| d | Generate a target point in the lower right corner of the maze, and walls should not all surround this target point. |
| e | Visualisation of maze generation steps |
| f | Add a skip button. If the player is unwilling to manually control the character to reach the end and enter the next level, they can press this button to skip it and go directly to the congratulations page. |
| g | Add a previous level button so that if the user clicks on it, the maze size should be regenerated to the size of the previous level. This allows players to compare whether the efficiency of different pathfinding algorithms changes with the size of the maze. |
| h | Add a return button that will return the user to the main menu if they click it. |
| **3.** | Manual Exploration Mode in Mazes | This feature is not only to improve entertainment to attract users but also to allow users to customise the starting point so that the pathfinding algorithm can start to find the location of the target point in a maze from the user-defined starting point. | a | The player's initial position in the maze should be in the upper left corner, which walls should not all surround. | i | A sound effect will be triggered every time the player enters the maze. |
| b | Players can use the small keyboard or “WASD” to control character movement in a maze. | ii | A sound effect is made every time the player steps into the maze. |
| c | If the user reaches the target point, the congratulations menu will be displayed | iii | Every time the player takes a step, the sound effect should be randomly selected from a list of sound effects instead of having just one sound effect. |
|  |  | iv | A sound effect will be triggered if the player reaches the target point. |
| **4.** | Enable Process Visualisation of Pathfinding Algorithms |  |  | Clear visualisation of how the algorithm navigates around the maze walls to reach the target point  Implement visual cues to indicate the progression of algorithms through the maze.  Ensure algorithms can be run in a single environment to control variables. |  |  |
| **5.** | Implementation of Diverse Pathfinding Algorithms |  | a | Depth-first Search |  |  |
| b | Breadth-first Search |
| c | Dijkstra’s Algorithm |
| d | A\* Search |
| e | Ensure algorithms can solve mazes generated by the system effectively. |
|  |  |
| **6.** | Performance Analysis and Comparative Data Display |  | a | Develop a system to compare the performance of different pathfinding algorithms on the same maze. |  |  |
| b | Metrics include path length, computation time, and efficiency. |
| **7.** | Implement Login and Register System |  | a | User account creation, login system, and maze-solving history tracking for personalised user experiences. | i | Develop a secure user authentication and registration system using appropriate hashing and password encryption. |
| b |  |
| **8.** | Implement Maze Level Selection Menu |  |  | A user-friendly interface that allows a selection of maze complexities. |  |  |

## 1.6 Solutions

#### Python3

For the development of my project, I choose Python3 as my programming language. This is not only because it comes with many valuable and powerful libraries but also because it supports a variety of mathematical operations that will help **create** relatively complex algorithms such as Dijkstra and A\*. I have done many projects using this programming language before, so I am familiar with it.

#### Pygame

For libraries for developing GUIs, I used Pygame, a library specifically designed for writing video games. It enables the creation of interactive and visually appealing GUIs and helps **visualise** pathfinding algorithm steps.

#### SQLite3

Additionally, I decided to use SQLite3 as the database management system for this project because SQLite3 allows portability and eliminates the need for a separate database server, making deployment easier for end users. Its portability ensures that this educational tool can be easily set up and used in **various** settings.

#### Integration Strategy

Python 3 is responsible for implementing the pathfinding algorithm, managing data processing, and the basic logic of the project. Pygame is integrated with Python and is responsible for developing graphical user interfaces, rendering graphics, managing user events, and playing sounds. SQLite3 will collect user data and maze configuration. It will be integrated within the Python environment and handle data storage and retrieval, ensuring the user's game progress is saved.

# 2. Documented Design

## 2.1 System Design

### 2.1.1 Overall System Flowchart

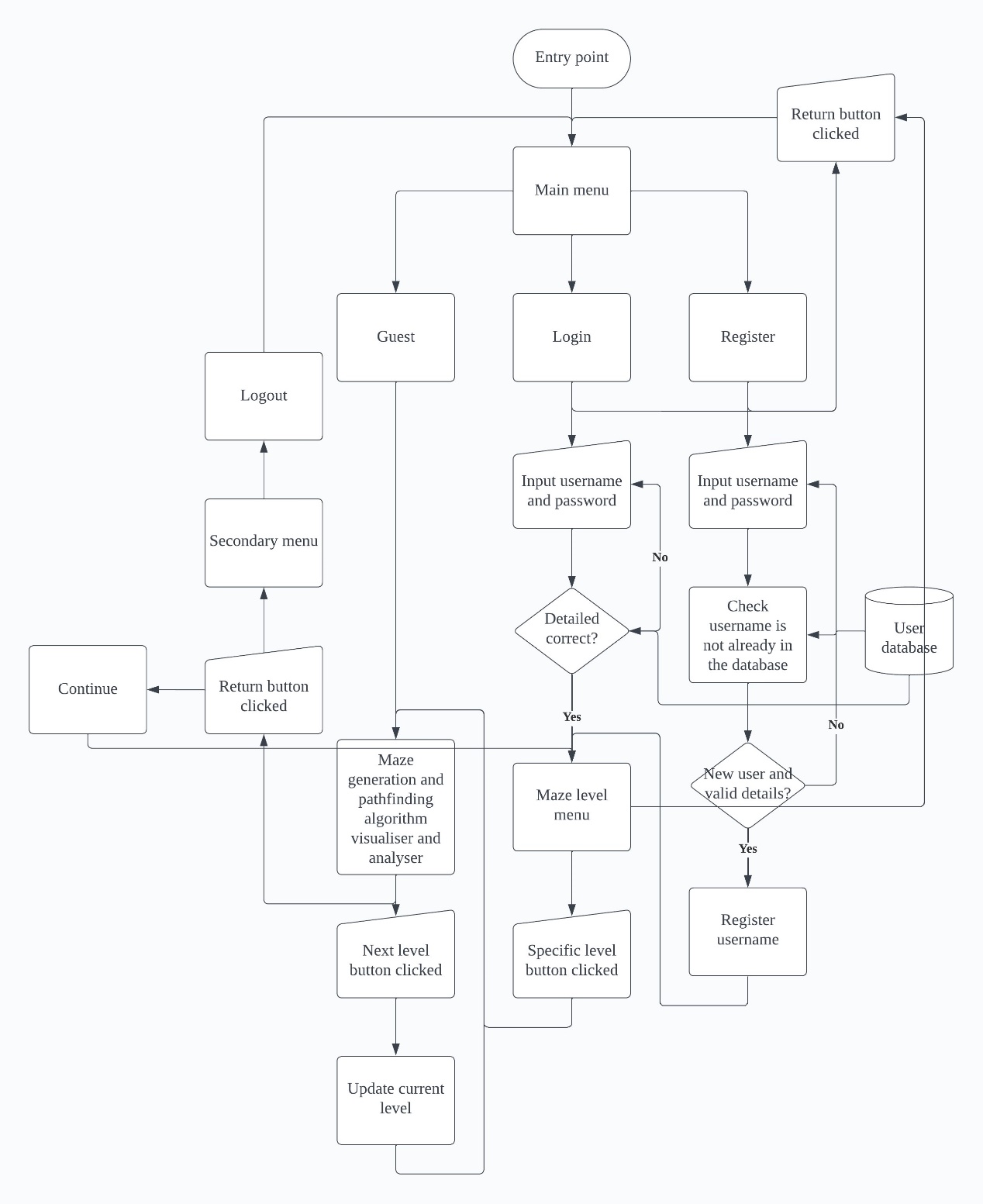


Figure 6

### 2.1.2 Maze and Pathfinding System Flowchart

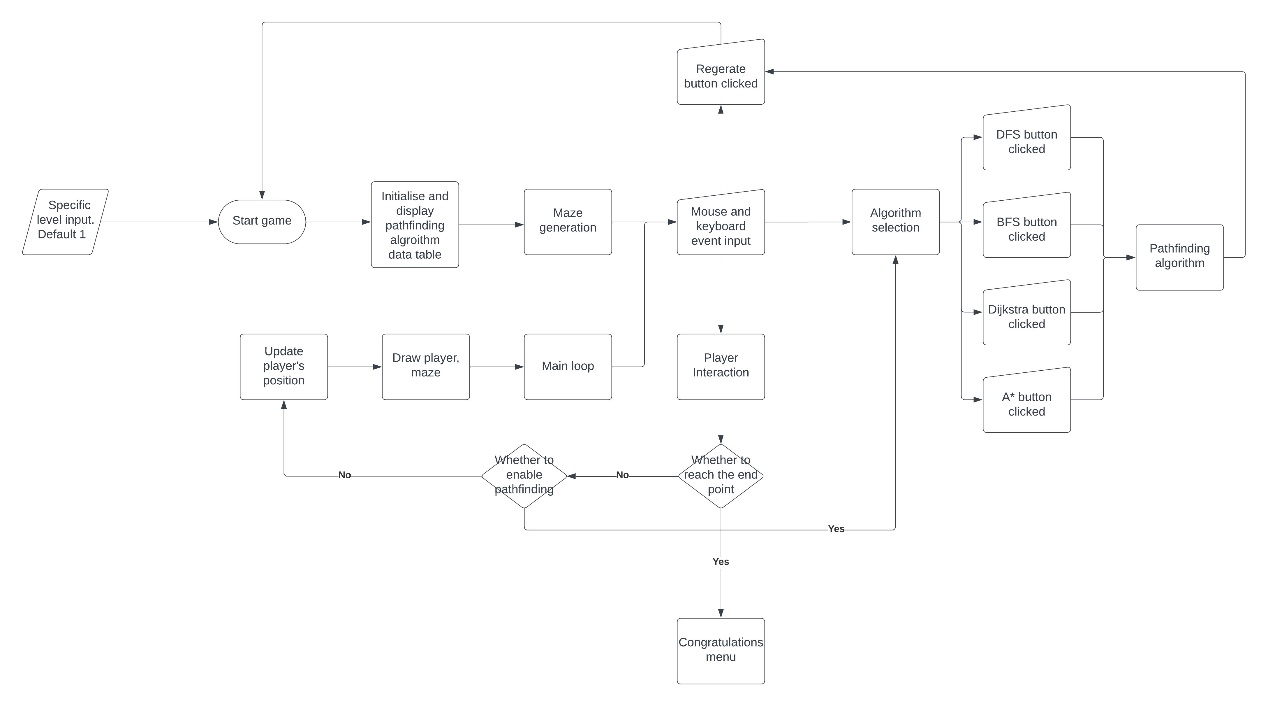


Figure 7

The above flowchart represents the system logic of the main game, expands and illustrates the steps of the maze generation and pathfinding algorithm visualiser and analyser in the overall system flowchart in Section 2.1.1, and illustrates the operation process from start to end. It starts with the player's level selection, defaulting to level 1 if no specific input is given. The game then initialises and displays the maze and pathfinding algorithm data sheet. The game's main loop handles the rendering of the player and the maze, capturing player interaction via mouse and keyboard input.

Player decisions lead to selecting the pathfinding algorithm, where buttons corresponding to different algorithms (DFS, BFS, Dijkstra, A\*) can be clicked to activate them. Based on the player's in-game actions (such as moving within a maze or selecting a pathfinding algorithm), the game updates the player's position or executes the selected pathfinding algorithm.

The final condition of this flowchart is to check if the player has reached the end of the maze, causing a congratulations menu to appear, indicating that the level is complete.

### 2.1.3 Player Movement and Control Flowchart

Use the if statement to write the four-direction (up, down, left, and right) program framework as follows:

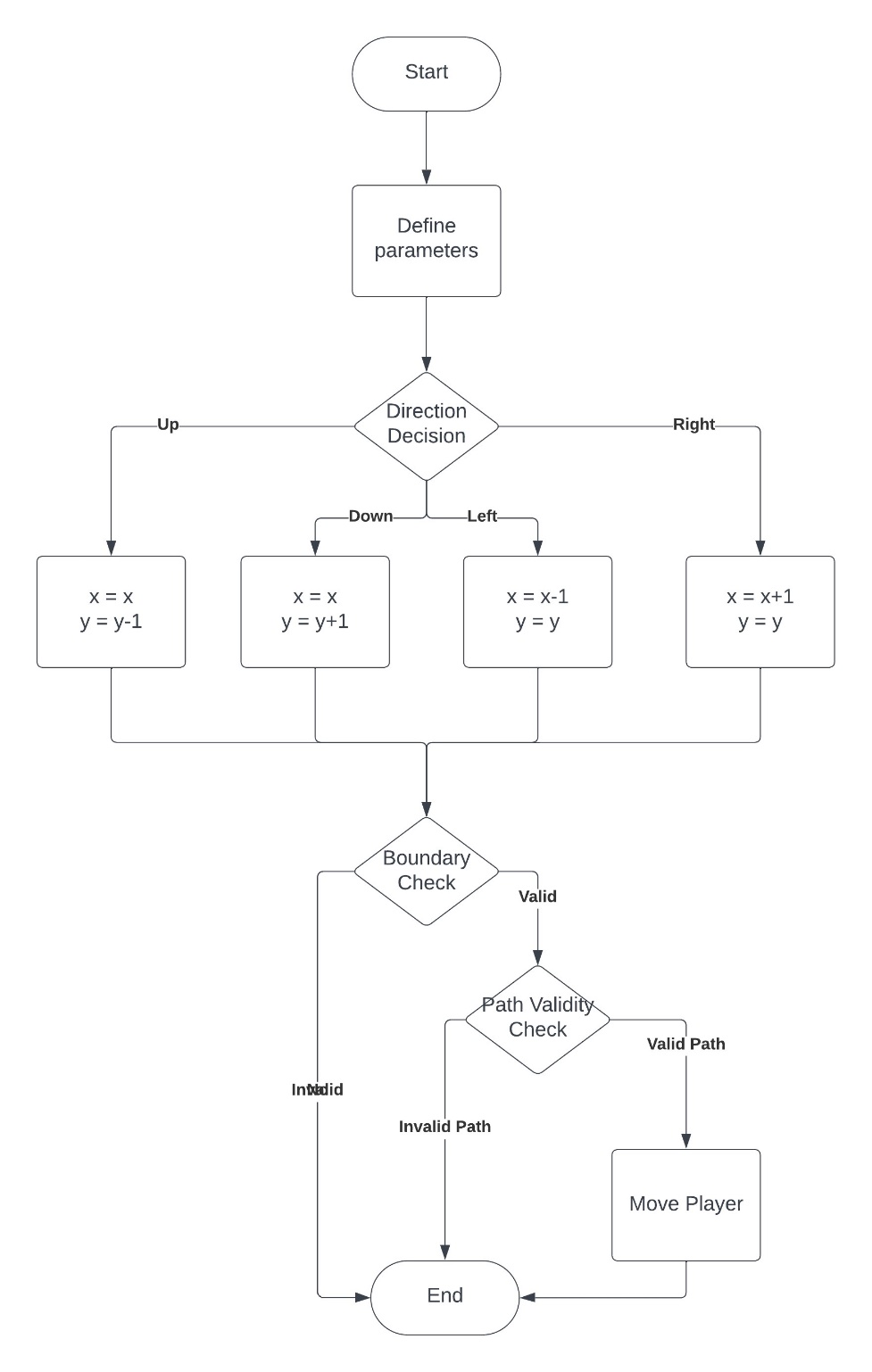


Figure 8

## 2.2 Algorithms

### 2.2.1 Randomised Prim's Algorithm

After researching and investigating different maze generation methods in the analysis part, I chose to use the randomised Prim algorithm for the maze generation, which can help create perfect mazes (a.k.a. standard mazes). It refers to a maze with no loops or unreachable positions. It is making sure that there is a path between the start point and the target point.

Because this algorithm generates mazes by breaking through the walls, initialising a map with all cells separated by walls is necessary before developing a maze. Take the 3×3 map below as an example. Odd-numbered intersections represent paths; the remaining points are walls. The maze is initially surrounded by walls.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 0 |  |  |  |  |  |  |  |
| 1 |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  |  |

From the graph, we can see the position of a path in the map, such as (3, 1), (1, 5); the values of x and y always show odd numbers. They can also be expressed as (2x+1, 2y+1), where . Note that the maze must have an odd number of rows and columns.

The steps of this algorithm can be described from a set perspective.

To facilitate the description, we first label all the path units of this 3×3 map with numbers, as shown in the figure below: 1 represents (1, 1), 2 illustrates (3, 1) and so on.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 0 |  |  |  |  |  |  |  |
| 1 |  | 1 |  | 2 |  | 3 |  |
| 2 |  |  |  |  |  |  |  |
| 3 |  | 4 |  | 5 |  | 6 |  |
| 4 |  |  |  |  |  |  |  |
| 5 |  | 7 |  | 8 |  | 9 |  |
| 6 |  |  |  |  |  |  |  |

Initially, the 3×3 map has nine path units separated by wall units, meaning every path only contains one path unit. If we use a set to represent the path , the elements in the set are path unit numbers; when we initialising the map, we have:

Then, randomly select a wall unit, assuming that the two path units on both sides of the wall unit are as , where .

If , it means that the path unit and the path unit are already connected, and the elements originally in would have merged into (assuming ).In such a case, there is no need to take any action.

But if , which means that the path unit and the path unit are not connected. In this case, we merge the sets and by combining all the elements from both sets. We update the set of the minimum index () to be the union of and so that the path unit and the path unit can be connected.

Repeat these processes until all the initial sets combine into one set: .

Indicating that every path unit within the maze is now connected, and a fully linked set of path units representing a traversable maze has been established.

### 2.2.2 Depth-first Search

My maze navigation system will use the Depth-first Search (DFS) based on stack implementation approach. It explores the maze by advancing as far as possible along each branch from the starting point (referred to as ‘’) before backtracking.

This exploration is managed using a LIFO (Last in, first out) method, where a stack data structure (denoted as ‘’, where ) is used. In this context, ‘’ represent the possible positions within the maze, and ‘’ are sequences of nodes depicting a route taken. Each element in the stack ‘’ is a pair consisting of a maze node and the corresponding path to that node. Additionally, a visited set (, where ), which is initially empty, records all nodes that have been explored to prevent revisiting them. The starting node, ‘’, is an element of the set of all possible nodes ().

Initially, create a stack, add the starting position, and initialise an empty set for visited nodes.

Then, while the stack () is not empty, pop the top element from to check whether this position is the target position.

In the beginning, ; if the starting point is the end node, return the path as the solution. If not in , add it to .

For each direction from , calculate . If is within maze boundaries and not in : Push onto S.

Since I use the randomised Prim algorithm (see 2.2.1) to generate mazes, the generated maze is theoretically perfect (no loops, no mazes in unreachable areas), but during the debugging process, it is not ruled out that the maze may be generated incorrectly. Therefore, it is necessary to add a completion check: if becomes empty and no solution is found, indicate failure (e.g., return an empty set).

***Initialisation:***

**Starting Node:**

**Visited Set ():**

**Stack ():**

***Algorithm Process:***

1. **While Loop:**

a. Continue as long as .

b. Dequeue the first element from

c. **Path Checking:**

i. If , return path

ii. If , then

d. **Node Exploration:**

i. For each direction from current node, calculate new node.

ii. If new node is valid and :

2. **Completion Check:**

a. If and no solution is found, return

### 2.2.3 Breadth-first Search

Breadth-first Search (BFS) follows a similar basic framework to Depth-first Search (DFS) when traversing the graph, but the data structures they use determine the order and nature of their exploration (see 2.3), BFS uses a FIFO (First in, first out) approach, where a queue data structure (denoted as ‘’) is used for BFS to explore all neighbouring nodes at the current depth level before moving deeper into the maze.

Similar to DFS, BFS uses a set () to track visited nodes, ensuring each node is processed only once.

***Initialisation:***

**Starting Node:**

**Visited Set ():**

**Queue ():**

***Algorithm Process:***

1. **While Loop:**

a. Continue as long as .

b. Dequeue the first element from

c. **Path Checking:**

i. If , return path

ii. If , then

d. **Node Exploration:**

i. For each direction from current node, calculate new node.

ii. If new node is valid and :

2. **Completion Check:**

a. If and no solution is found, return

### 2.2.4 Dijkstra’s Algorithm

DFS and BFS can be viewed as brute-force algorithms, exploring paths without considering the weights or costs associated with each step. In contrast, Dijkstra’s algorithm employs a greedy approach. It selects the node with the smallest known distance from the start node at each step, ensuring the shortest path is incrementally built.

Initialise all distances to infinity, except the start node which is set to 0.

Initially contains all nodes with their tentative distances.

***Initialisation:***

**Starting Node:**

**Distance (D):**

**Visited Nodes (V):**

**Priority Queue (Q):**

***Algorithm Process:***

1. **While Loop:**

a. Continue as long as

b. Select and remove the node with the smallest distance from Q:

c. If is in , skip to the next iteration

d. Add to

2. **Relaxation:**

a. For each neighbour of :

i. Calculate new distance:

ii. If :

Update

Update ’s distance

3. **Completion:**

When is empty, contains the shortest distances from the start node to all other nodes.

### 2.2.5 A\* Search

The A\* Search Algorithm combines path cost and heuristic evaluation to solve the maze.

This approach uses a priority queue (denoted as ‘’, where ) to manage node exploration based on both actual path cost from the start node and an estimated cost to the goal (). ‘’ represent possible positions within the maze, and ‘’ are the cumulative values of and , determining the exploration priority.

Initially, the priority queue is set with the starting node, and a set of visited nodes (, where ) is initialised to track explored nodes. The starting node (‘’) is thus an element of the set of all possible nodes (), with and and .

***Initialisation:***

Starting Node:

Visited Set (V):

Priority Queue (P):

***Algorithm Process:***

1. **While Loop**:

a. Continue as long as .

b. Select and remove the node with the lowest from :

c. Path Checking:

i. If , reconstruct and return the path.

ii. If , then .

d. Node Exploration:

i. For each direction from the , calculate new node.

ii. If new node is within boundaries, not in , and offers a lower :

.

2. **Completion Check**:

a. If and no solution is found, return .

## 2.3 Data Structures

### 2.3.1 Dynamic Array

#### Maze Representation

The maze environment can be represented as a 2D dynamic array. This data structure provides a grid-like structure to the initialisation maze, as shown below. Each unit in the array corresponds to a specific position within the maze, and its value corresponds to the map type, a wall unit (represented by 1) or a path unit (represented by 0).

This data structure makes manipulating the maze elements easier during maze generation and visualisation.

#### Checklist for the Generation of Mazes

While executing the Randomised Prim’s Algorithm, a dynamic list (implemented as a list in Python) is used as a checklist. The checklist keeps track of maze units that need exploration. The dynamic nature of the list allows for the addition and removal of maze units as the algorithm progresses.

checklist = [(x1, y1), (x2, y2), ...]

Note that the maze units for exploration are randomly selected for the Randomised Prim's Algorithm, so the checklist does not abide by any specific order (neither FIFO nor LIFO).

### 2.3.2 Stack

A stack follows a LIFO (Last In, First Out) principle. It only allows adding or removing elements at the top of the stack, which is suitable for the latest data-related scenarios.

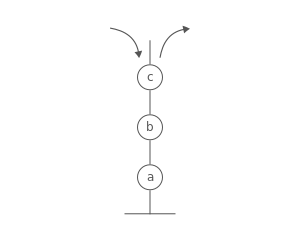


Figure 10

#### DFS Search

In DFS, start at a node and follow a deeper path until you can no longer continue, then backtrack to explore other paths. The algorithm uses stacks to manage the order of nodes to be examined next.

Because stacks are LIFO (last in, first out), nodes recently added to the stack will be removed and processed first. This approach allows the algorithm to explore each path in depth.

### 2.3.3 Queue

A queue implements a FIFO (First In, First Out) approach. Elements are added at the end and removed from the front, ensuring the order of elements is maintained.

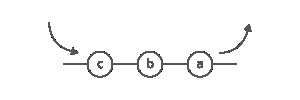


Figure 11

#### BFS Search

BFS starts from a node, explores all neighbouring nodes, and then moves to the next level of neighbouring nodes.

Queues ensure that nodes are explored in the order they were discovered, thus maintaining the level-wise traversal.

### 2.3.4 Priority Queue (Min Heap)

A priority queue organises elements according to their priority, allowing the highest priority elements to be quickly retrieved. It is usually implemented using a heap to provide efficient insertion and deletion.

#### Dijkstra’s and A\*

In the Dijkstra and A\* search algorithms, a priority queue manages nodes based on a cost metric (cumulative distance for Dijkstra and f-score for A\*), allowing the algorithm to choose the best path at each step.

### 2.3.5 Set

A set is a collection that stores elements that do not repeat and are not in any particular order, making insertion and removal easy. In my project, a set is used in the pathfinding algorithm to keep track of visited nodes, ensuring that each node is processed only once.

### 2.3.6 Enumerations (Enum)

An enumeration is a set of symbolic names bound to unique and constant values. They make code more readable and less error-prone by limiting the range of acceptable values for a variable. Enums define specific directions and map entry types in my project.

### 2.3.7 2D Tuple

A 2D tuple is an immutable and ordered collection of elements. This data type is suitable for storing data consisting of multiple related items. They represent coordinates (x, y) in the maze.

### 2.3.8 List

A list in Python is an ordered, dynamic data structure. In this project, they store random congratulations messages and display them randomly on the congratulations page because lists can be easily indexed, modified or expanded.

## 2.4 Database Design

This project uses the database to manage user data, including the user's account name, password, and game progress level. Among them, the storage of game progress levels can track the maze levels unlocked by players, thereby providing players with a more personalised experience and achieving this program's educational and entertainment purposes.

### 2.4.1 ER Diagram

Because the only purpose of adding the database to this project is to record the player's progress in unlocking the maze levels, it only contains one entity. The database part is implemented using SQLite; four fields are stored in the database: id, username, password, and level.

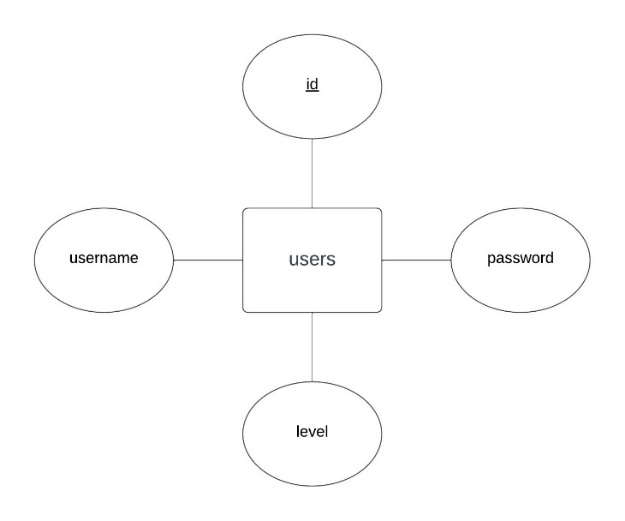


Figure 12

### 2.4.2 Data Dictionaries

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Table Name | | users | | | |
| Primary Key(s) | | id | | | |
| Foreign Key(s) | | N/A | | | |
| Data Item | Data Type | | Description | Constraints | Sample Data |
| id | Integer | | A unique identifier for each user. | Primary key, auto-increment | 3 |
| username | Text | | The user chooses the username. | Not null, unique, maximum Length 20 characters | hello |
| password | Text | | The user sets the password. | Not null, maximum length 20 characters | world |
| level | Integer | | Represents the user's current level or progress in the game. | The default value is 1 | 12 |

### 2.4.3 SQL Statements

#### Create Table Statement

Initialise the users table in the database

        CREATE TABLE IF NOT EXISTS users (

            id INTEGER PRIMARY KEY AUTOINCREMENT,

            username TEXT NOT NULL,

            password TEXT NOT NULL,

            level INT DEFAULT 1

        )

#### Insert Statement

This statement adds a new user to the users table; the “?” placeholders are replaced with the username and password provided by the user during registration.

INSERT INTO users (username, password) VALUES (?, ?);

#### Select Statement

This statement retrieves a user’s login details; the “?” placeholders are replaced with the username and password entered by the user.

SELECT \* FROM users WHERE username=?

#### Update Statement

This statement updates the level of a user based on their game progress; the first “?” is replaced with the new level, and the second “?” with the username.

update users SET level=? WHERE username=?

### 2.4.4 Normalisation

Since the only purpose of this project is to store game progress, the database is not complex and is already in 1NF since each table column has atomic values and no repeating groups. For example, in the users table, each column (id, username, password, level) contains only one value per record.

### 2.4.5 DFD Diagram

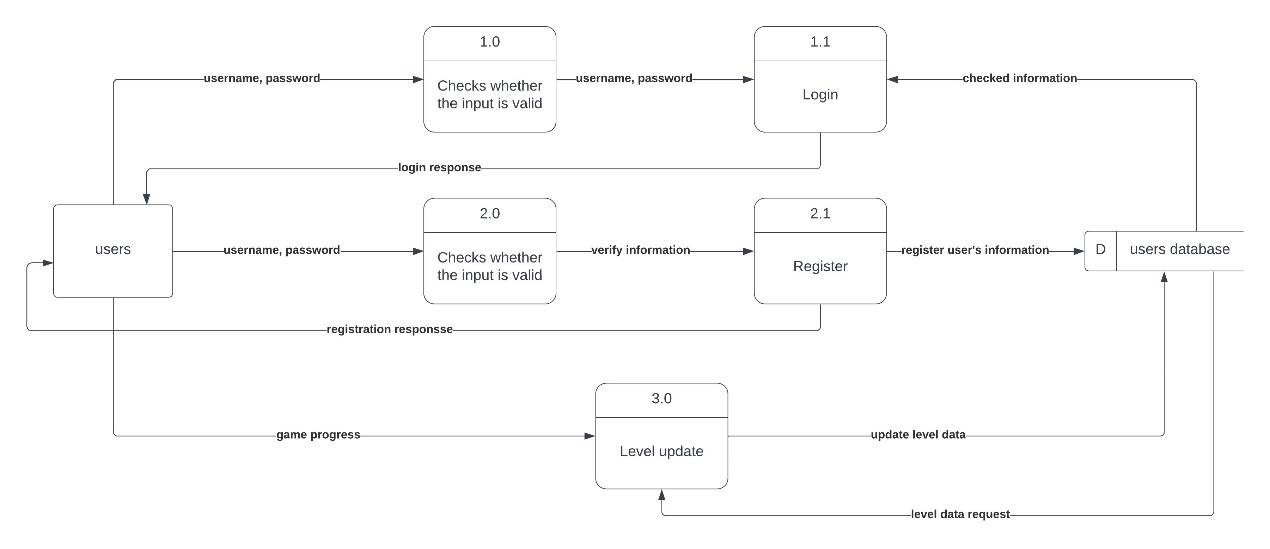


Figure 13

## 2.5 Object-Oriented Design

### 2.5.1 Class Description

#### Player (Class)

|  |
| --- |
| Player |
| + x: int  + y: int |
| + \_\_init\_\_(self, start\_x: int, start\_y: int): None  + move(self, direction: PLAYER\_DIRECTION): None |

This class encapsulates the player’s position within the maze and provides players with the function of moving in the maze. Attribute x and y represent the player’s coordinates.

The move method is consistent with the PLAYER\_DIRECTION of enumeration and updates the player’s position to show the interaction with the game’s maze environment.

#### Map (Class)

|  |
| --- |
| Map |
| + width: int  + height: int  - map: list[list[int]]  + player: Player |
| + \_\_init\_\_(self, maze\_level: int): None  + resetMap(self, value: MAP\_ENTRY\_TYPE): None  + setMap(self, x: int, y: int, value: MAP\_ENTRY\_TYPE): None  + isVisited(self, x: int, y: int): bool  + drawMap(self, screen: pygame.Surface, CELL\_SIZE: int = 15): None |

The Map class is used to create and manage the maze's structure. It defines the maze environment. Properties in this class include width and height (which determine the dimensions of the maze) and map (a 2D list that represents the maze's grid layout). The core of this class is the player attribute, which is an instance of the Player class and represents the player's position in the maze. The resetMap method initialises and clears the maze, the setMap method sets the cell value to define the maze size, and the DrawMap method renders the maze on the screen.

#### LevelObject (Class)

|  |
| --- |
| LevelObject |
| + x: int  + y: int  + image: pygame.Surface  + rect: pygame.Rect  + text: str  - margin: int  - font: pygame.font.Font |
| + \_\_init\_\_(self, x: int, y: int, text: str, obj\_size: list, font\_size: int): None  + update\_text(self): None  + collision(self, mouse\_pos: tuple): bool |

The LevelObject class is a dynamic element in the game level selection interface, as a subclass of pygame.sprite.Sprite, it enables interactive components to be displayed on the level selection screen. Each LevelObject instance represents a clickable level button, characterised by properties such as x and y for positioning and image and rect for graphical representation. This class also has text properties to display level numbers or information. Essential functions include update\_text for rendering text onto an object and collision for detecting user interaction.

#### GameLevelInterface (Class)

|  |
| --- |
| GameLevelInterface |
| + win\_size: tuple[int, int]  + screen: pygame.Surface  + image\_background: pygame.Surface  + user: str  + common\_methond: tuple  + return\_main\_text: tuple[pygame.Rect, pygame.Surface]  + game\_status\_code: list[int]  + levels\_column: int  + levels: list[list[int]]  + levels\_objs: list[LevelObject]  + all\_sprites: pygame.sprite.Group  + first: bool |
| + \_\_init\_\_(self, \*\*kwargs): None  + query\_db\_level(self): int  + deal(self): int  + draw(self): None |

The GameLevelInterface class manages the game's level selection interface. It is a graphical and interactive front end where players can select maze levels. This class handles the dynamic display of level options, each represented by an instance of the LevelObject class.

#### Pathfinder (Pseudo Class)

|  |
| --- |
| Pathfinder |
|  |
| + searchDFS(maze, start, end, screen, delay)  + searchBFS(maze, start, end, screen, delay)  + searchDijkstra(maze, start, end, screen, delay)  + searchAStar(maze, start, end, screen, delay) |

The Pathfinder pseudo-class represents a collection of various pathfinding algorithms for maze navigation. Although not a class, it symbolically groups different algorithms, including depth-first search, breadth-first search, Dijkstra, and A\* search. Each algorithm is encapsulated as a method for finding paths through a maze with different strategies and efficiencies.

#### Authentication (Pseudo Class)

|  |
| --- |
| Authentication |
|  |
| + login(username, password)  + register(username, password)  + query\_db\_level(username) |

The Authentication pseudo-class conceptually encapsulates the user authentication system, which includes login and registration processes and handling of user data.

### 2.5.2 UML Class Diagram

#### Core Gameplay Mechanics

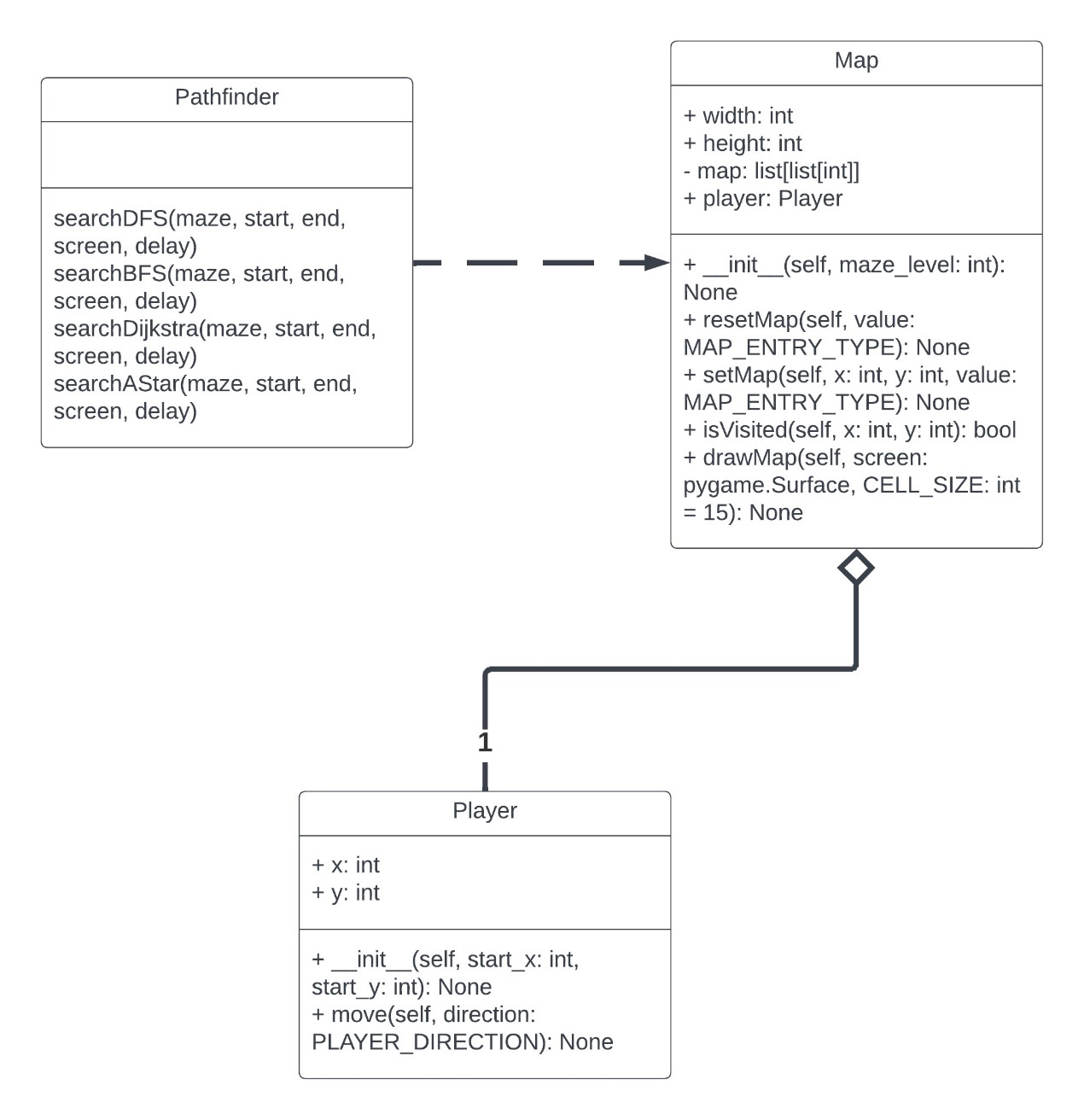
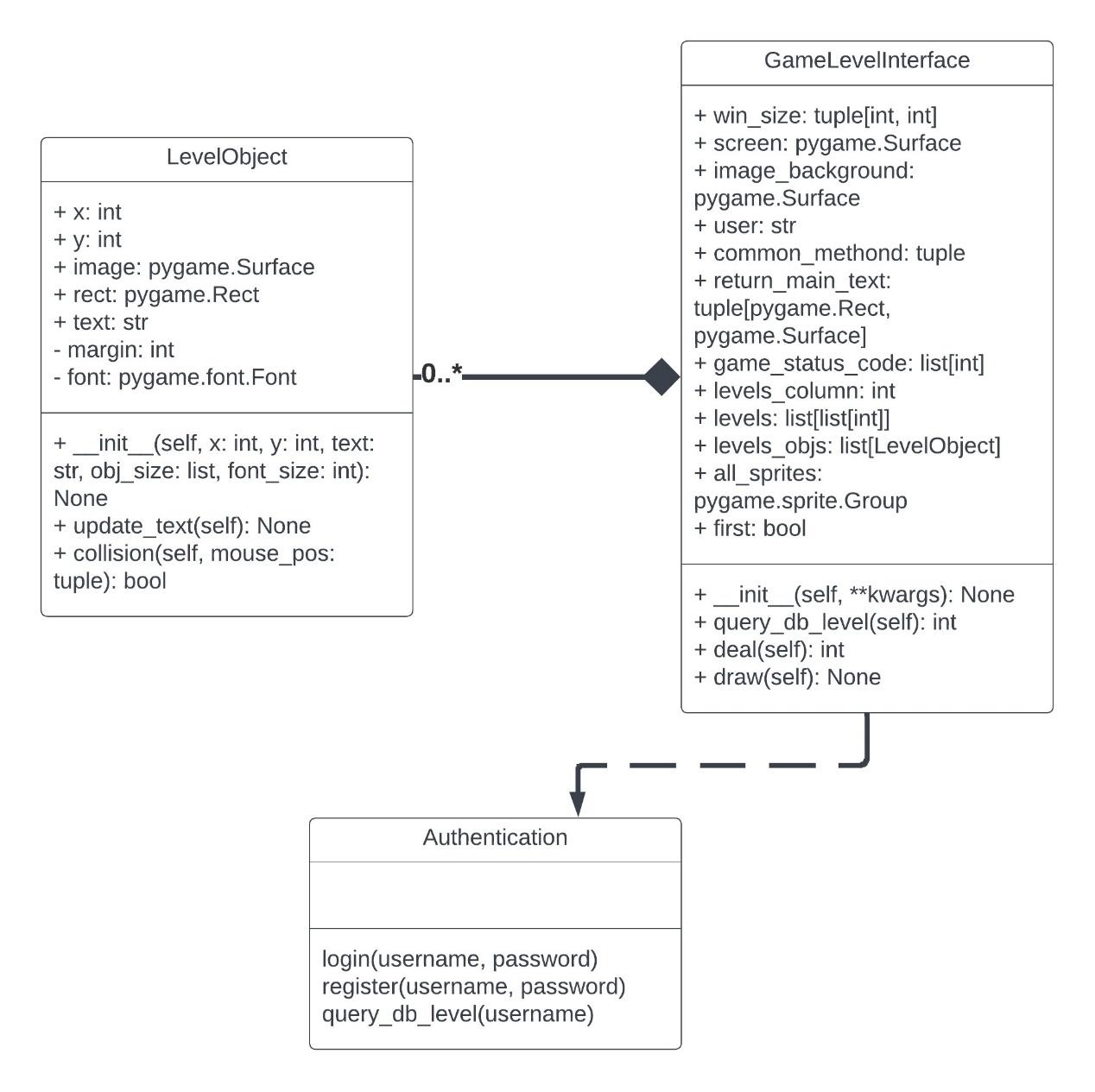


Figure 14

#### User Interface and Authentication



## 2.6 GUI Design

### 2.6.1 Main Menu

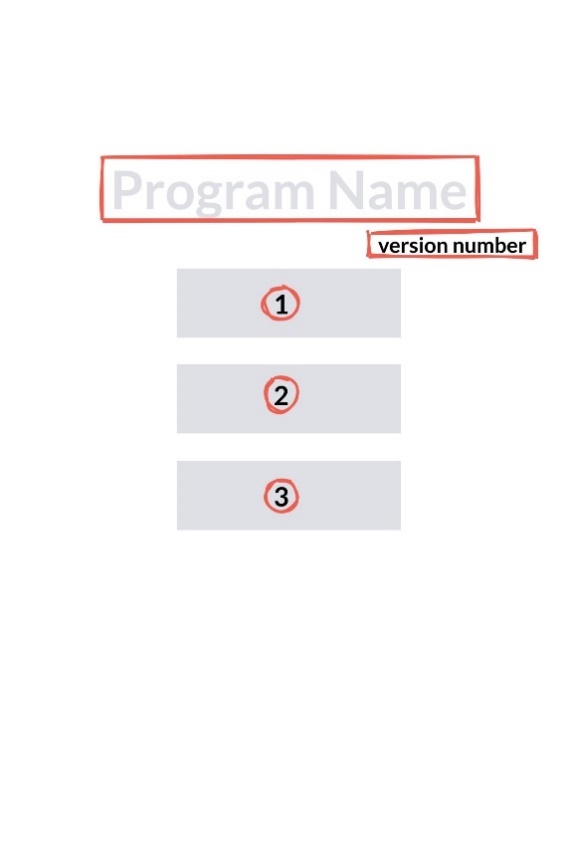
 

Figure 15 Figure 16

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Component | | Functionality | | Technical Implementation | |
| 1 | Button 1 (Play) | a | Starts the game or takes the user to the level selection screen. | i | Change the Gamestate to either start the game directly or move to the level selection menu, depending on whether the user is logged in. |
| ii | Use the drawButton function for visual representation. |
| 2 | Button 2 (Login) | b | Open the login interface for user authentication. | i | On click, it changes the game state to the login screen. |
| ii | Interact with the SQLite database for user authentication. |
| 3 | Button 3 (Register) | c | Navigate to the user registration interface. | i | This leads to the registration screen, where new users can create an account. |
| ii | It is linked to the SQLite database for storing new user information. |
| 4 | Program Name | d | A visual identifier for the program | i | Use pygame.image.load() to load the logo image from a file. The logo is then positioned at the top centre or another prominent location on the screen using blit() to draw it on the surface. |
| 5 | Version number | a | Informs the user about the current version of the NAVI program they are using. | i | Render the version number as text using pygame.font.Font().render() |

### 

### 2.6.2 Common UI

Because this combination is frequently used in the following scenarios, a separate function is created to improve overall code efficiency.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Component | | Functionality | | Technical Implementation | |
| **1** | Login State | **a** | Show the user's current status, e.g., logged in as [Username]. | **i** | Uses conditional rendering to display user-specific information (like username) or a generic login prompt for guests. |
| **ii** | Interact with the SQLite database for user authentication and data retrieval. |
| **iii** | Render the user's status. |
| **2** | Return Button | **a** | Return to the main menu and exit the maze environment. | **i** | Reset the Gamestate to the initial state, which represents the main menu. |
| **ii** | This transition is conditionally altered if the user is logged in, taking them to a different menu state. |

### 2.6.3 Maze and Pathfinding Interface



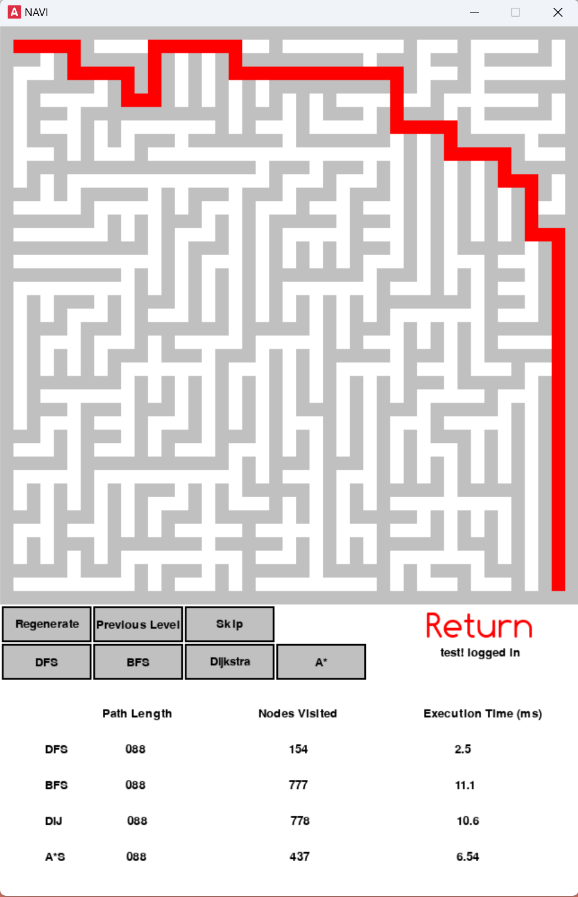
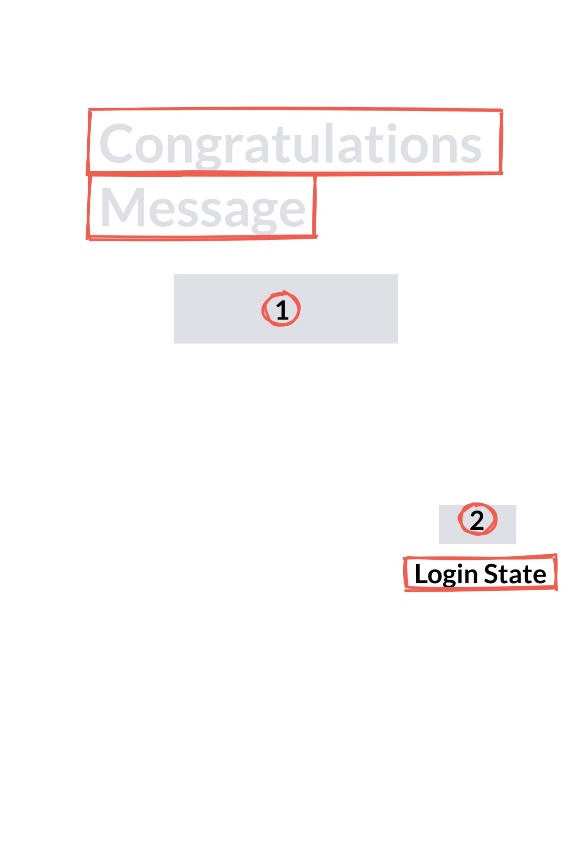


Figure 17 Figure 18

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Component | | Functionality | | Technical Implementation | |
| 1 | Button 1 (Regenerate) | a | Trigger the regeneration of the maze. | i | When pressed, it sets a Boolean flag generate\_maze to True. |
| ii | This flag is checked in the main game loop, and when it is True, it calls the maze generation function to redraw the maze. |
| iii | Use the drawButton function to create the button's visual appearance and the pygame. event.get() function to handle the button press event. |
| 2 | Button 2 (Previous Level) | a | Decrease the maze complexity by lowering the maze level. | i | Adjust the maze\_level variable to reduce the complexity. |
| ii | Call the maze generation algorithm with the new level setting. |
| iii | Integrated into the main loop for dynamic adjustment during gameplay. |
| 3 | Button 3 (Skip) | a | Directly switch to the congratulations menu. | i | Change the Gamestate to a predefined state that represents the congratulations screen. |
| ii | This skips the current gameplay and displays the winning message or next-level options. |
| 4 | Common UI | a | (See 2.6.3) | i | (See 2.6.3) |
| 5 | Button 5 to 8 (DFS; BFS; Dijkstra; A\*) | a | Switch the game state to execute the selected pathfinding algorithm. | i | Each button is linked to a specific pathfinding algorithm. |
| ii | Upon pressing any of these buttons, the game state changes to the corresponding algorithm execution, which then runs on the current maze layout. |
| 6 | Comparative Data Display | a | Show a real-time table with metrics like path length, nodes visited, and execution time for each algorithm. | i | Use text rendering to display dynamic data within the game interface. |
| ii | Data is calculated and updated based on the pathfinding algorithms’ performances in the current maze. |
| 7 | Maze Environment | a | Respond to user input to modify the maze layout and show pathfinding algorithm progress. | i | The maze object is manipulated based on user interactions (like button presses for regeneration). |
| ii | The progress of pathfinding algorithms is visually represented in the maze, updating in real-time as the algorithms run. |

### 

### 2.6.4 Congratulations Interface



#

Figure 19 Figure 20

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Component | | Functionality | | Technical Implementation | |
| 1 | Button 1 (Next Level) | a | Navigate the user to the next level or challenge. | i | On click, increments the level and initialises the next challenge. |
| 2 | Button 2 (Return) | a | Allow the user to exit to the main menu. | i | Reset the game state to the main menu on click. |
| 3 | Congratulations Message | a | Display a congratulatory message upon successful completion of a level or skipped. | i | Dynamically choose a congratulations text from a list and display it on the screen. |
| ii | Possibly animate the text for added impact, such as a brief scaling or colour change effect. |
| 4 | Login State | a | (See 2.6.3) | i | (See 2.6.3) |

### 2.6.5 Game Level Interface

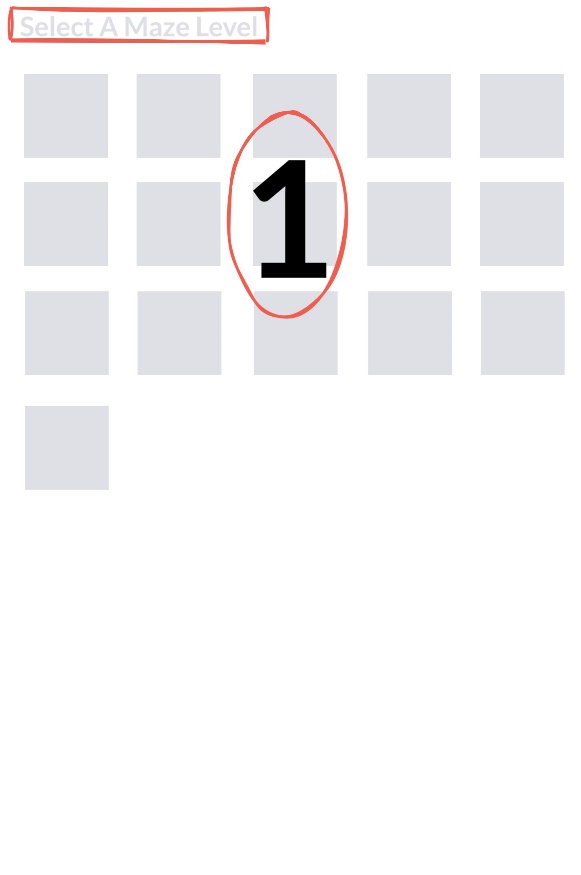
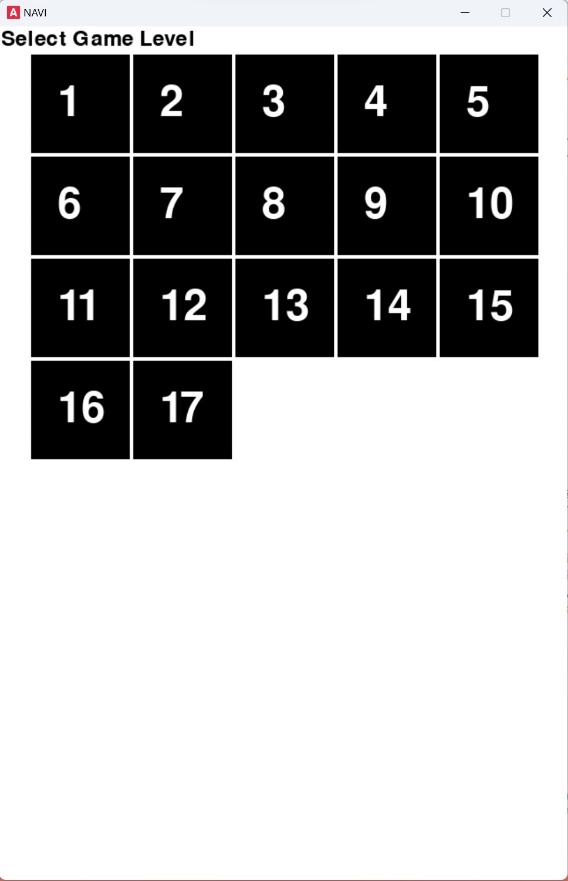


Figure 21 Figure 22

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Component | | Functionality | | Technical Implementation | |
| 1 | Button Group 1 (Level Selection Buttons) | a | Enable the selection of different game levels. | i | Create buttons dynamically based on the number of levels available. |
| ii | Each button should display the level number and be large enough for easy interaction. |
| iii | Arrange the buttons in a grid layout for an organised appearance. |
| iv | Use the LevelObject class to create and manage these buttons. |
| 2 | Text Display (Title) | a | Display the title “Select Game Level”. | i | Render the title with pygame. font.Font |

### 2.6.6 Login/Register Page

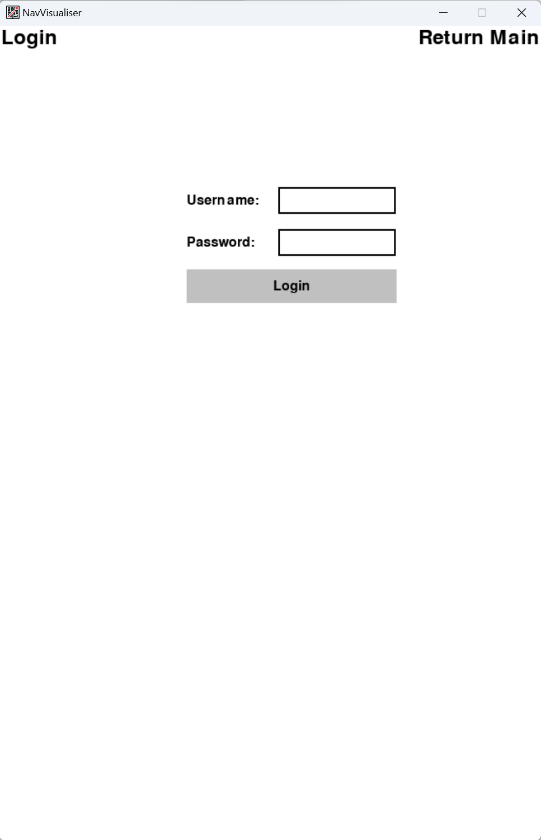


Figure 23 Figure 24

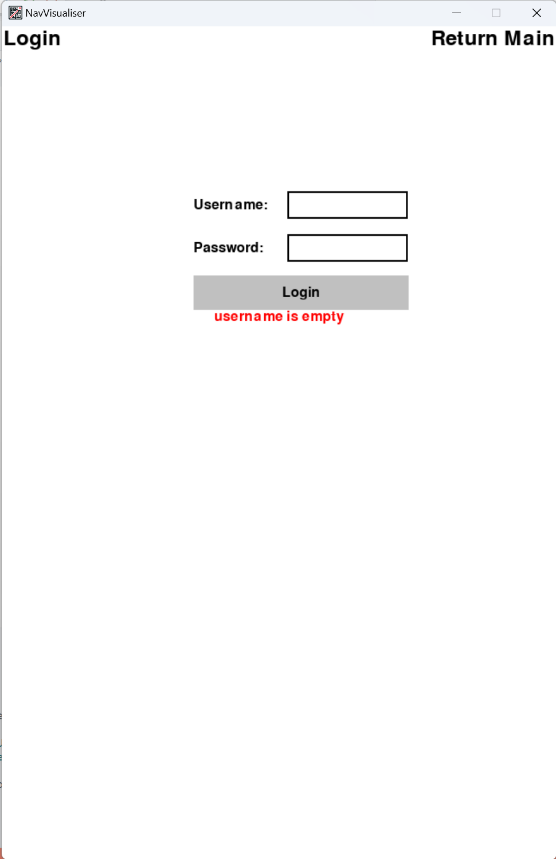
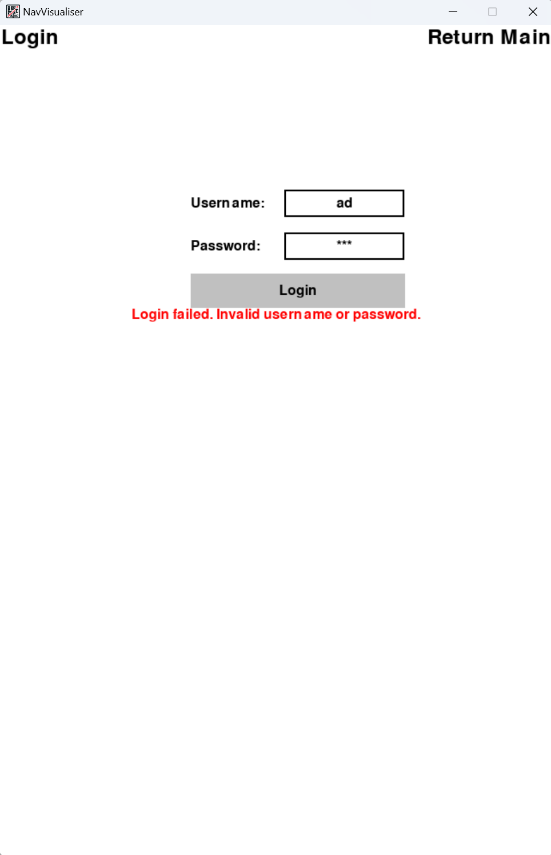


Figure 25 Figure 26

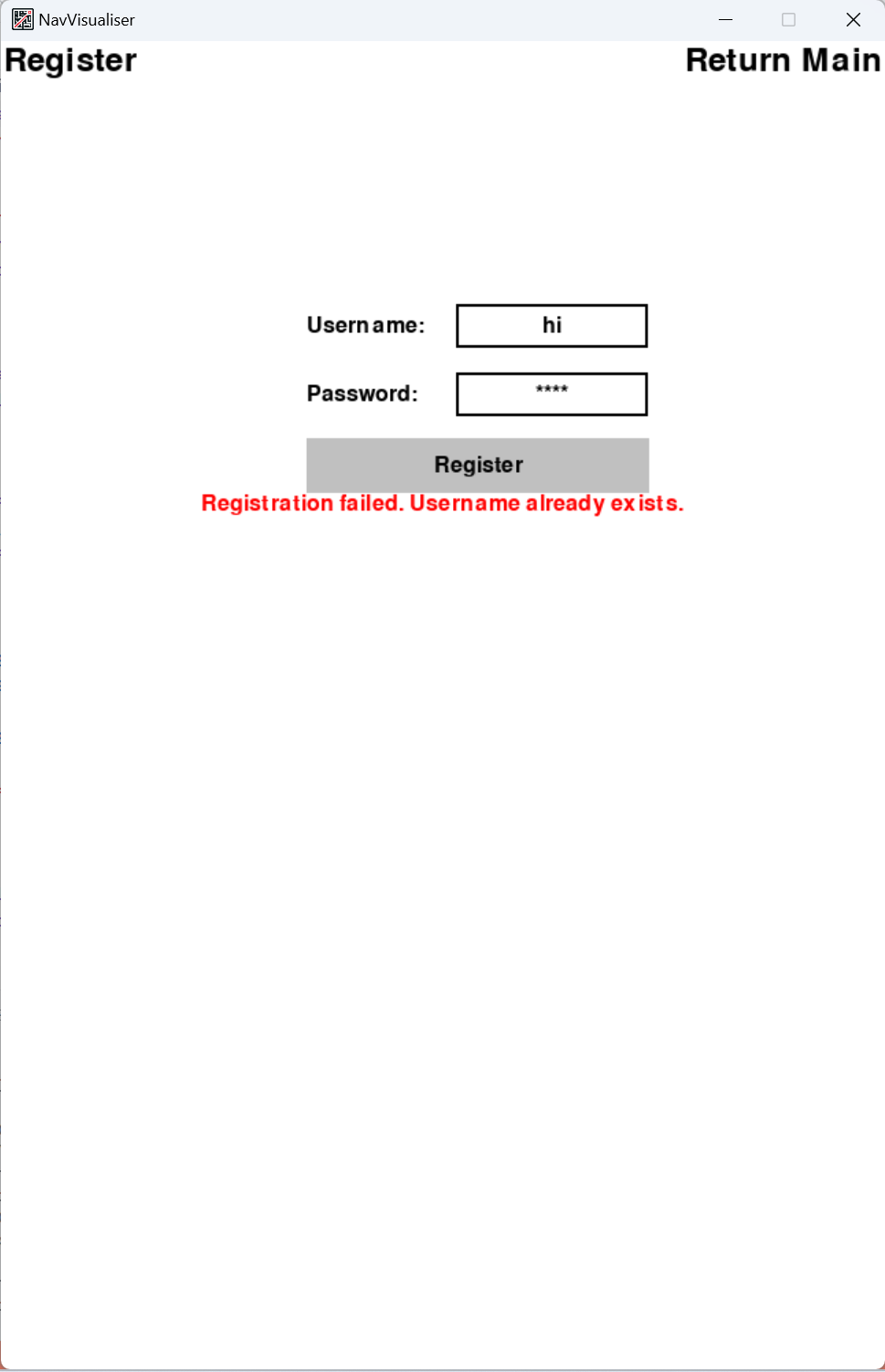


Figure 27

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Component | | Functionality | | Technical Implementation | |
| 1 | Input Fields 1 and 2 (Username and Password) | a | Clickable and typable fields. | i | Interactive text box initialised empty. Incorporate event handling to capture and store user input. |
| b | Password characters should be masked for privacy. | ii | Event listeners for mouse clicks and keyboard inputs to activate and capture data from input fields. |
| c | Include placeholder text, e.g., “Enter Username” and “Enter Password”. |  |  |
| 2 | Button 3 (Login; Register) | a | Trigger the authentication process with the entered credentials. | i | A clickable button that, upon activation, validates user credentials against the database for login or registers a new user account. |
| ii | Use SQLite3 Python library for database operations, including querying user data, inserting new users, and validating login credentials. |
| 3 | Title Text (Login or Register) | a | Display the current interface's context - either “Login” or “Register.” | i | Render the title. Dynamically switch the text based on the interface state. |
| 4 | Return Main Button | a | Provide an option to navigate back to the main interface. | i | A button that changes the application state to the main menu upon clicking. |
| 5 | Error Message Display | a | Show error messages related to authentication failures or input validation. | i | Text fields are positioned strategically to be visible in cases of incorrect input or authentication issues. Updates dynamically based on the authentication process's outcome. |

### 2.6.7 Secondary Menu

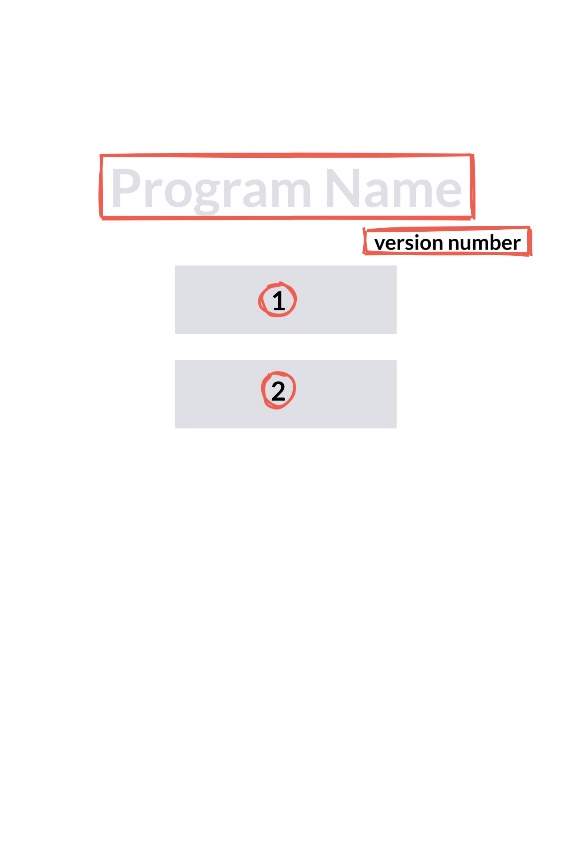
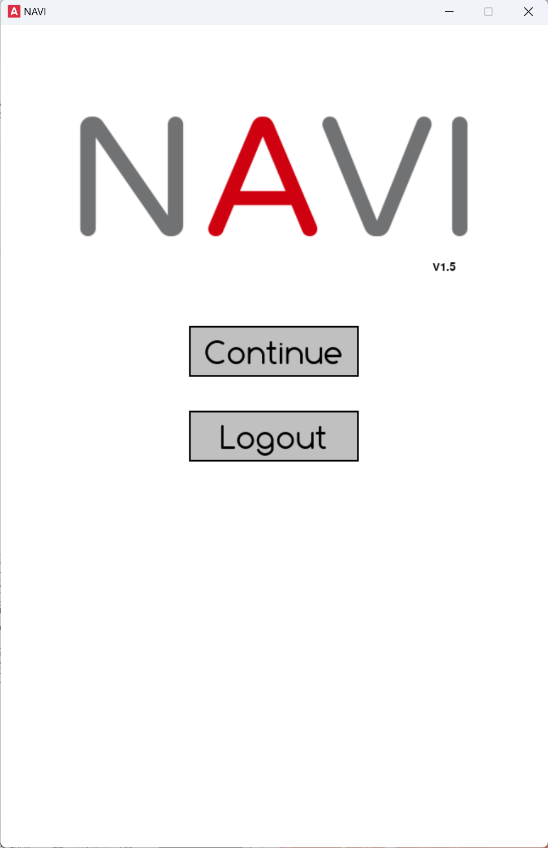


Figure 28 Figure 29

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Component | | Functionality | | Technical Implementation | |
| 1 | Button 1 | a | Allow the user to continue the game. | i | Define a pygame.Rect object to specify the button's position and dimensions. Render the button's label using pygame.font.Font().render() and draw the button using pygame.draw.rect(). Use event handling (pygame.event.get()) to detect clicks on the button's area and trigger the game to start or resume. |
| 2 | Button 2 | a | Enable the user to log out of their account safely. This action should save any progress and return the user to the main menu | i | Similar to the Play button, define a pygame.Rect for the Logout button's position and dimensions. Render the label and draw the button on the screen. Event handling is used to detect clicks on this button, which triggers the logout process. This involves calling a function to save the user's progress to the database (using SQLite3 in Python for database operations) and then transitioning back to the login/register screen. |
| 3 | Program Name | a | A visual identifier for the program | i | Use pygame.image.load() to load the logo image from a file. The logo is then positioned at the top centre or another prominent location on the screen using blit() to draw it on the surface. |
| 4 | Version number | a | Informs the user about the current version of the NAVI program they are using. | i | Render the version number as text using pygame.font.Font().render() |

## 2.7 File Structure

### 2.7.1 File Arrangement

* resources/
  + bgm/
    - 0.wav
    - 1.wav
    - 2.wav
    - 3.wav
    - 4.wav
    - 5.wav
    - 6.wav
    - 7.wav
    - 8.wav
    - 9.wav
    - 10.wav
  + Image/
    - icon.png
    - main\_menu.png
    - medal.png
  + Sound/
    - button.mp3
    - clap.mp3
    - move.mp3
    - next.map
    - reach.mp3
* authentication.py
* game\_level\_interface.py
* main.py
* maze.py
* pathfinder.py
* player.py
* requirements.txt
* user\_data.db
* README.md

### 2.7.2 Resources Organisation

#### “bgm” directory

The “resources” directory serves as the root for all visual and auditory resources. It is divided into three main subdirectories: “bgm”, “image”, and “sound”, each representing different types of assets used throughout the simulator.

The “bgm” folder contains various WAV files (uncompressed file format). These files provide a range of high-quality background music tracks that make this simulator more entertaining. To make it easier to reference and dynamically load tracks, I changed the names of these tracks to be arranged using a numerical arrangement system. In detail, I used a random selection mechanism to trigger the replacement of different BGMs. The method is as follows:

pygame.mixer.music.load('resources/bgm/%s.wav' % random.randint(0, 10))

This creates unpredictability and a sense of freshness when users use the simulator. This randomness ensures that the background music never becomes monotonous, increasing the game's overall fun and thereby better achieving the educational purpose of this project.

However, as it's worth noting, this version, intended to make the test video in the document, uses the audio format WAV, which would make the overall file very large. So, if possible, I will convert these audio formats to compressed file formats such as MP3 to reduce file size.

#### “sound” directory

This directory hosts sound effects, including interactions like button clicks (“button.mp3”), applause sound (“clap.mp3”), movements in mazes (“move.mp3”), level progression sound (“next.mp3”), and reaching a target point sound (“reach.mp3”). Adding these sounds is not technically challenging, but it can significantly improve the user experience to meet more user needs.

#### “image” folder

The “image” folder contains images used in the simulator’s user interface. These include an icon, the main menu logo and the gold medal image used on the congratulations page, as shown below:



Figure 30

### 2.7.3 Additional Considerations

**“requirements.txt” -** Lists all external libraries required for the project, ensuring consistent setup environments.

**“README.md”** – To provide an overview, installation instructions, and usage details.

# 3. Testing

## 3.1 Compilation Environment

This project was compiled and passed under Windows. The required environment is python3, and the library used to write the GUI is Pygame. Before running the program, configure the dependencies needed, that is, open cmd under the path of main.py and run:

pip install –r requirement.txt

After configuring the dependencies, run the program by entering:

python main.py

## 3.2 Overall System Test Table

**Testing Video on YouTube:** https://youtu.be/lRSIYmDF1\_c



### 3.2.1 User Interface Testing (Objective 1)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | Description | Test Data/Input | Expected Outcome | Pass/ Fail | Evide  nce |
| 1 | Verify that the homepage loads correctly in a minimalist style, including play, login, and register buttons. | Launch application. | The main page is displayed with a clear, minimalist design containing three buttons: “Play”, “Login”, and “Registration”. | Pass | 0:06 |
| 2 | Test the responsiveness of the login and registration interface. | Click on the  “Play”, “Login”,  and “Regi  -stration” buttons. | Each button click leads to their respective interfaces with clear titles, input boxes for username and password, a submit button, and a return to the main menu button. Inputs should be easily readable, and submit buttons should be clickable. | Pass | 0:11 |
| 3 | Test the maze-level selection page's user interface after login. | Successful login. | After logging in, the maze level selection page displays buttons for the different maze levels, which update as the player progresses. The interface includes a functional back button. | Pass | 0:20 |
| 4 | Evaluate simplified controls for algorithm selection, maze level selection, and display for comparative data. | Interaction with maze level  buttons | Components are intuitive and provide users with immediate, clear feedback. The selection of maze levels is smooth, and there is a comparative data table with visible, dynamic pathfinding algorithm efficiency below the maze. | Pass | 0:21 |
| 5 | Confirm the functionality of the congratulations menu after completing a level or skipping. | Completion or skipping of a maze level. | A congratulations menu appears with aesthetically pleasing design elements, including dynamic messages, a next-level button, a medal image and a return button. | Pass | 0:29 |

### 3.2.2 Maze Generation Testing (Objective 2)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | Description | Test Data/Input | Expected Outcome | Pass/ Fail | Evidence |
| 6 | Validate the initialisation of the maze generation process. | Trigger maze generation with default settings. | The algorithm initialises and starts generating a maze without errors, beginning with a grid filled with walls. | Pass | 0:49 |
| 7 | Verify that maze generation leads to a perfect maze without loops or unreachable areas. | Complete the maze generation process. | The generated maze is a perfect maze with exactly one unique path between any two points in the maze. No loops or isolated sections should exist. | Pass | 0:50 |
| 8 | Check the algorithm's ability to generate mazes of varying complexity and sizes. | Generate mazes with different complexity levels by adjusting maze size parameters. | The algorithm successfully generates mazes of specified sizes and complexities, with the complexity appropriately reflected in the maze's layout. | Pass | 1:00 |
| 9 | Evaluate the uniqueness of each generated maze. | Generate multiple mazes with the same size and complexity settings by clicking the “Regenrate” button. | Each generated maze should be unique, showing the algorithm's randomness and variability in maze design. | Pass | 3:54 |
| 10 | Test the generation of the target point and validate its placement. | Observe the placement of the target point in multiple maze generations. | The target point is placed in a reachable location not immediately adjacent to the start point, preferably in the lower right corner of the maze, and always accessible from the start point. | Pass | 0:49 |

### 3.2.3 Manual Exploration Mode Testing (Objective 3)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | Description | Test Data/Input | Expected Outcome | Pass/ Fail | Evidence |
| 11 | Test the functionality of player movement controls within the maze. | Use keyboard inputs (“WASD” or arrow keys) to move the player in all four directions. | The player character moves smoothly in the direction corresponding to the key press, with movement restricted by walls and boundaries within the maze. | Pass | 1:20 |
| 12 | Verify that the player can start from the designated initial position. | Start or reset the maze exploration. | Each time the exploration mode begins or is reset, the player starts from the upper left corner or a predetermined starting point within the maze. | Pass | 7:13 |
| 13 | Confirm the appearance of the congratulations menu upon reaching the target point. | Navigate the player to the target point in the maze. | Upon reaching the target point, the congratulations menu appears with options for proceeding to the next level or returning to the main menu. | Pass | 1:37 |
| 14 | Test the functionality of the manual exploration mode with different maze complexities. | Engage in manual exploration in mazes of varying complexity. | The player can manually explore mazes of any complexity, with the game accurately tracking and responding to player movement within the different layouts. | Pass | 8:58 |
| 15 | Evaluate interactive feedback during manual exploration (e.g., sound effects, visual cues). | Perform movement actions within the maze (movement, reaching the target). | Appropriate sound effects and visual cues are triggered by actions such as moving, hitting a wall, or reaching the target, enhancing the exploration experience. | Pass | 1:20 |

### 3.2.4 Process and Comparative Data Visualisation Testing (Objectives 4 & 5 & 6)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | Description | Test  Data/Input | Expected Outcome | Pass/ Fail | Evide  nce |
| 16 | Verify the visualisation starts  correctly for each pathfinding algorithm. | Select each pathfinding algorithm (DFS, BFS, Dijkstra, A\*) from the interface to start the visualisation. | Upon selection, the visualisation of the chosen algorithm begins immediately, with clear visual cues indicating the starting point, the algorithm's progression, and exploration of the maze. | Pass | 2:10 |
| 17 | Test the accuracy  and completeness of the pathfinding process visualisation. | Complete runs of each pathfinding algorithm in a variety of mazes. | The visualisation accurately represents the entire pathfinding process, from start to finish, including the exploration of nodes and the final path taken to reach the target. Each step should be clearly distinguishable, and the final path should be correct according to the algorithm's logic. | Pass | 2:16 |
| 18 | Assess the visual clarity and educational value of the visualisation features. | Observation of the visualisation features, such as highlighted current node explored paths and the final path. | All visualisation features (e.g., highlighting, colour changes) are clearly visible and contribute to understanding the algorithm's functionality. Users should be able to distinguish between explored nodes, the current focus of the algorithm, and the chosen path. | Pass | 2:10 |
| 19 | Ensure  that the visualisation process  can be run  in a  controlled environment. | Run each pathfinding algorithm on the same maze to observe and compare their behaviour consistently. | Each algorithm operates within the same environmental conditions, allowing for direct comparison of their pathfinding methods and efficiency. The visualisations should reflect the unique characteristics and advantages of each algorithm. | Pass | 2:10 |
| 21 | Test the functionality  of Depth-first Search in  solving a maze. | Run DFS on a variety of mazes with different layouts and complexities. | DFS successfully finds a path from the start to the target point, if one exists, prioritising depth traversal over breadth. The path may not be the shortest, but it must be valid. | Pass | 3:41 |
| 22 | Verify the functionality  of Breadth-  first Search  in maze solving. | Apply BFS to various mazes, ensuring a mix of simple and complex structures. | BFS finds the shortest path from the start to the target point, exploring nodes level by level. The algorithm should efficiently manage exploration without unnecessary deviations. | Pass | 3:44 |
| 23 | Test  Dijkstra’s Algorithm  for correct-  ness and efficiency. | Execute Dijkstra's Algorithm on mazes with varying path lengths and obstacles. | Dijkstra’s Algorithm accurately calculates the shortest path in terms of distance to the target, considering all paths' weights. The algorithm should handle varying complexities without errors. | Pass | 3:47 |
| 24 | Evaluate the A\* Search algorithm for optimal pathfinding. | Use A\* Search in mazes designed with multiple routes and varying difficulties. | A\* Search efficiently finds the shortest path by combining costs and heuristics, outperforming Dijkstra's in terms of execution time and resource usage while still being accurate. | Pass | 3:50 |
| 25 | Cross-  compare all implemented algorithms on the same set of mazes. | Run DFS, BFS, Dijkstra's,  and A\* on identical mazes,  noting their pathfinding choices, execution time, and path lengths. | Each algorithm solves the maze according to its theoretical basis, with observable differences in path choice, efficiency, and computational overhead. BFS and A\* should consistently find the shortest path, with A\* potentially doing so more efficiently. DFS may find longer paths, and Dijkstra’s should perform optimally where path costs vary. | Pass | 2:10 |
| 26 | Verify the accurate recording of path length for each algorithm. | Solve a  maze using each pathfinding algorithm (DFS, BFS, Dijkstra’s, A\*). | The system accurately records and displays the path length (number of steps) each algorithm takes to reach the target. BFS and A\* should generally record the shortest path lengths due to their nature. | Pass | 2:10 |
| 27 | Test computation time recording for each algorithm. | Execution  of each pathfinding algorithm  on identical mazes. | The system accurately measures and displays each algorithm's computation time, reflecting efficiency differences. A\* and BFS are expected to show better performance in terms of speed compared to DFS and potentially Dijkstra’s, depending on the maze complexity. | Pass | 2:10 |
| 28 | Evaluate the accurate tracking of  the number  of nodes explored. | Apply each algorithm to a set of mazes. | The system records and displays the total number of nodes explored by each algorithm before finding the target. This metric should reflect the exploratory efficiency, with A\* and BFS exploring fewer nodes than DFS and Dijkstra’s in most cases. | Pass | 2:10 |
| 29 | Test the comparative analysis display for clarity and accuracy. | Review the comparative analysis display after running each algorithm on the same maze. | The comparative analysis display clearly and accurately presents the data for path length, computation time, and nodes explored, enabling easy comparison across algorithms. The display should be user-friendly and facilitate educational insights. | Pass | 2:10 |
| 30 | Validate the response-  iveness and real-time update  feature of  the comparative data display. | Change algorithms or maze configurations to observe updates in the comparative data display. | The comparative data display updates in real-time with new data as different algorithms are run or when the maze configuration changes, without significant delays or inaccuracies. | Pass | 2:10 |

### 3.2.5 Login and Register System Testing (Objective 7)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | Description | Test Data/Input | Expected  Outcome | Pass/ Fail | Evide  nce |
| 31 | Test the registration process for  new users. | Enter a unique username and password into the registration form. | The system successfully creates a new user account, stores the user's data securely, and provides a confirmation message or redirects the user to the login page. | Pass | 5:48 |
| 32 | Verify the login functionality for existing users. | Enter valid login credentials for an existing user account. | The system authenticates the user and gives access to their account, redirecting them to the main application or user dashboard. | Pass | 6:20 |
| 33 | Test login attempt with invalid credentials. | Enter an incorrect username and/or password. | The system denies access, displaying an error message to the username or password that is incorrect without specifying which one for security reasons. | Pass | 4:32 |
| 34 | Assess the system's response to a registration attempt with an existing username. | Try to register a new account using a username that already exists in the database. | The system prevents the creation of a new account, providing a clear error message that the username has already been taken. | Pass | 5:03 |
| 35 | Evaluate the password encryption and security measures. | Register a new account and inspect the method  of storing user credentials  in the database. | The system uses secure hashing algorithms (e.g., bcrypt, SHA-256) to store passwords, ensuring they are not stored in plain text. | Pass | 5:21 |
| 36 | Check the logout functionality. | Log in and then use the logout option. | The system successfully logs the user out, terminates the session, and redirects the user to the main page. | Pass | 6:13 |

### 3.2.6 Maze Level Selection Menu Testing (Objective 8)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | Description | Test Data/Input | Expected Outcome | Pass/ Fail | Evide  nce |
| 37 | Verify the display and functionality of the maze level selection menu. | Access the maze level selection menu after logging in. | The menu displays all available levels; The user can select any unlocked level to start playing. | Pass | 7:59 |
| 38 | Test level unlocking mechanism upon completion of the preceding level. | Complete a level and return to the level selection menu. | The menu's next level becomes unlocked and selectable, reflecting the user's progress. The system should accurately track and update the levels unlocked by the user. | Pass | 8:11 |
| 39 | Evaluate the responsiveness and navigational flow of the level selection interface. | Navigate through the level selection menu, including selecting levels and accessing other menu options. | The interface responds quickly to user input with no noticeable delay. Navigation of the level selection menu and other application sections is intuitive and consistent. | Pass | 7:59 |
| 40 | Confirm the integration of the level selection menu with the user's account and progress tracking. | Login  with different user accounts having varying levels of progress. | The level selection menu should reflect each account's progress and unlock the correct level based on the user's game progress. | Pass | 9:50 |

## 3.3 Major Bugs and Fixes

|  |  |  |  |
| --- | --- | --- | --- |
| Bug ID | Issue Area | Description/ Screenshot | Technical Solution |
| 1 | Level Selection Display for New Users | Incorrect level option display for new users and incorrect row display logic. | Ensure the initialisation logic for new users correctly sets the starting level to 1. For row display issues, correct the algorithm responsible for the display level. Correct the count of display levels before adding a new level to the row to ensure that a new row starts after every five levels. |
| 2 | Repeated Rendering in State 11 | Common UI text is repeatedly rendered with any input event. | Adjust the position of the Common UI function in the code and put it before the event detection to avoid repeated rendering, that is, within the if statement that generates the maze. |
| 3 | Delayed UI Element Display | UI elements under the maze require an input event to display. | Call the function responsible for displaying these UI elements during the initial setup of state 11, not tied to an event handler. Ensure all elements are set to be visible right after the maze is rendered. |
| 4 | Maze Level Generation | Inconsistencies in maze level generation and selection, with a total of 17 levels expected. | Verify the algorithm that maps selected levels to maze sizes, ensuring the mapping correctly reflects the range from 11 to 43 with intervals of 2. A loop or a mapping function that generates these values should be corrected for off-by-one errors or incorrect incrementing. |

## 3.4 Testing Search Algorithm Efficiency

### 3.4.1 Purpose of Testing

The main goal of the tests was to evaluate the performance of different pathfinding algorithms (DFS, BFS, Dijkstra and A\*) for navigating mazes of varying complexity. Key performance metrics include number of nodes visited and execution time in milliseconds.

### 3.4.2 Testing Environment and Tools

The tests ran on a system equipped with:

CPU: Intel i9

RAM: 32 GB

### 3.4.3 Testing Method

The test environment is to generate square mazes of different sizes through the algorithm (tested every two levels).

Each pathfinding algorithm was executed three times on each maze size category, and an average was calculated to mitigate any anomalies.

Execution time was measured in milliseconds using Python's time module, while the number of nodes visited was tracked via algorithm-specific counters implemented in each algorithm's code.

### 3.4.4 Results Table

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Maze Level | Algorithm | Test 1  (Node) | Test 2  (Nodes) | Test 3  (Nodes) | Average Nodes Visited | Test 1  (Time) | Test 2  (Time) | Test 3  (Time) | Average Execution Time |
| 1 | DFS | 20 | 20 | 24 | 21.3 | 0.00 | 0.77 | 0.00 | 0.77 |
| 1 | BFS | 46 | 34 | 32 | 37.3 | 0.00 | 0.00 | 1.00 | 1.00 |
| 1 | Dijkstra | 46 | 34 | 32 | 37.3 | 0.94 | 0.96 | 0.00 | 0.95 |
| 1 | A\* | 39 | 29 | 25 | 31.0 | 0.00 | 0.54 | 1.00 | 0.51 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 3 | DFS | 42 | 38 | 66 | 48.7 | 1.53 | 0.00 | 0.00 | 1.53 |
| 3 | BFS | 73 | 67 | 62 | 67.3 | 1.07 | 1.02 | 1.23 | 1.11 |
| 3 | Dijkstra | 74 | 68 | 66 | 69.3 | 1.00 | 1.00 | 1.01 | 1.00 |
| 3 | A\* | 53 | 50 | 50 | 51.0 | 0.00 | 1.00 | 0.00 | 1.00 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 5 | DFS | 128 | 148 | 106 | 127.3 | 1.01 | 2.02 | 1.98 | 1.67 |
| 5 | BFS | 156 | 156 | 114 | 142.0 | 1.58 | 1.51 | 1.03 | 1.37 |
| 5 | Dijkstra | 156 | 156 | 118 | 142.0 | 2.13 | 2.26 | 0.99 | 1.79 |
| 5 | A\* | 155 | 136 | 85 | 125.3 | 1.88 | 1.57 | 0.98 | 1.48 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | DFS | 228 | 64 | 200 | 164.0 | 2.88 | 0.55 | 2.03 | 1.82 |
| 7 | BFS | 208 | 237 | 219 | 221.3 | 2.02 | 2.03 | 3.00 | 2.35 |
| 7 | Dijkstra | 214 | 238 | 222 | 224.7 | 2.00 | 3.00 | 1.99 | 2.33 |
| 7 | A\* | 162 | 164 | 167 | 164.3 | 2.00 | 1.98 | 2.05 | 2.01 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 9 | DFS | 220 | 208 | 72 | 166.7 | 2.00 | 2.51 | 1.00 | 1.84 |
| 9 | BFS | 332 | 312 | 320 | 321.3 | 4.24 | 3.00 | 3.00 | 3.41 |
| 9 | Dijkstra | 334 | 314 | 322 | 323.3 | 3.98 | 3.00 | 4.29 | 3.76 |
| 9 | A\* | 253 | 214 | 179 | 215.3 | 2.78 | 4.01 | 1.91 | 2.90 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 11 | DFS | 228 | 206 | 208 | 214.0 | 3.02 | 2.46 | 2.22 | 2.57 |
| 11 | BFS | 324 | 390 | 231 | 315.0 | 3.00 | 4.00 | 3.03 | 3.34 |
| 11 | Dijkstra | 332 | 390 | 236 | 319.3 | 4.09 | 4.61 | 2.17 | 3.62 |
| 11 | A\* | 170 | 355 | 142 | 222.3 | 2.02 | 4.90 | 2.63 | 3.18 |

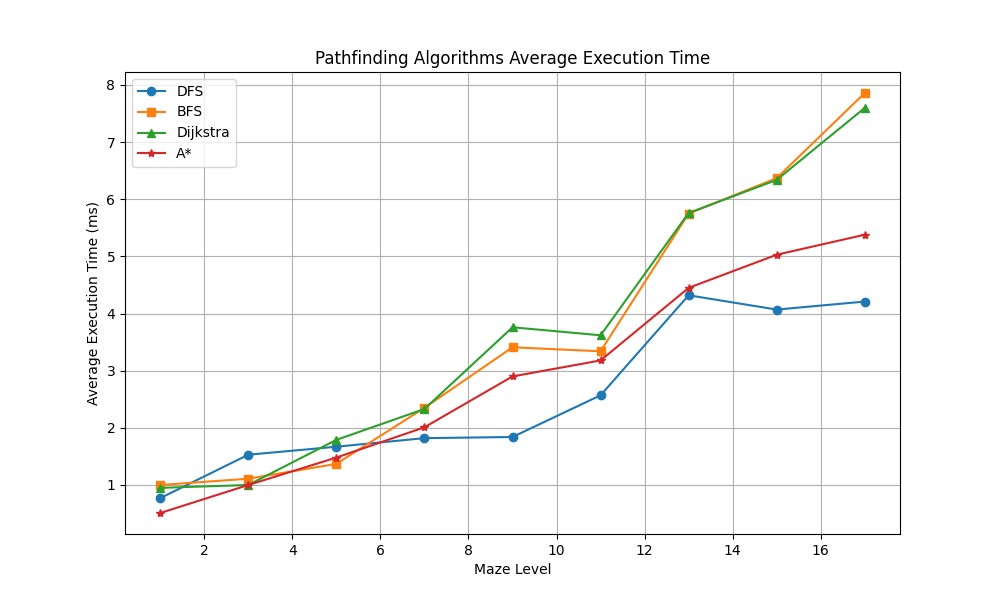
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 13 | DFS | 418 | 394 | 444 | 418.7 | 4.00 | 4.98 | 3.98 | 4.32 |
| 13 | BFS | 455 | 452 | 573 | 493.3 | 5.59 | 5.02 | 6.63 | 5.75 |
| 13 | Dijkstra | 466 | 460 | 574 | 500.0 | 5.50 | 5.00 | 6.77 | 5.76 |
| 13 | A\* | 302 | 318 | 472 | 364.0 | 3.53 | 4.02 | 5.81 | 4.45 |

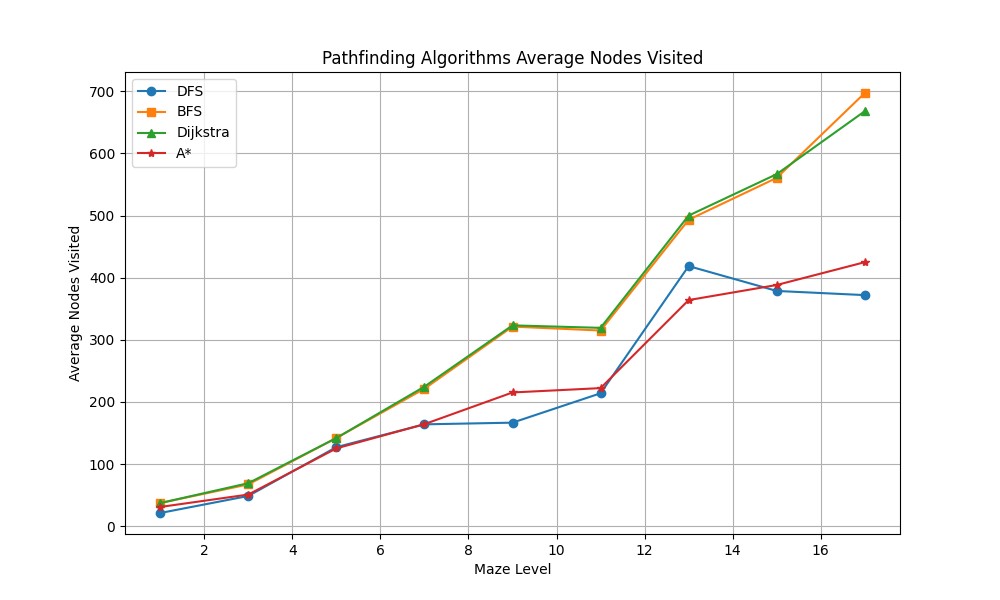
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 15 | DFS | 398 | 506 | 232 | 378.7 | 4.00 | 5.21 | 3.01 | 4.07 |
| 15 | BFS | 530 | 479 | 673 | 560.7 | 6.01 | 6.09 | 7.00 | 6.37 |
| 15 | Dijkstra | 538 | 488 | 674 | 566.7 | 6.00 | 5.00 | 8.01 | 6.34 |
| 15 | A\* | 358 | 343 | 464 | 388.3 | 5.01 | 4.04 | 6.04 | 5.03 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 17 | DFS | 204 | 512 | 400 | 372.0 | 2.00 | 6.57 | 4.07 | 4.21 |
| 17 | BFS | 538 | 847 | 607 | 697.3 | 6.55 | 10.0 | 7.02 | 7.86 |
| 17 | Dijkstra | 538 | 852 | 614 | 668.0 | 6.25 | 9.06 | 7.49 | 7.60 |
| 17 | A\* | 367 | 552 | 356 | 425.0 | 4.03 | 7.10 | 5.00 | 5.38 |

### 3.4.5 Summary of Test Results

I used the matplotlib library in Python to draw graphs to visualise the performance of different algorithms at different maze levels, including the average number of nodes explored and the average execution time, as shown below:





The two charts demonstrate that the DFS show high efficiency in small to medium size mazes with a low number of nodes visited and execution time. However, as the maze level increases, the growth of its execution time and the number of node visits becomes unstable, especially when the maze reaches 13 and above, the growth rate accelerates significantly. This indicates that DFS is less efficient when dealing with large-scale mazes, possibly because it tends to explore in-depth and is prone to falling into longer paths in complex or large mazes.

BFS and Dijkstra’s algorithms show similar trends in the number of node visits and execution time in all maze levels, with the execution time and number of node visits of both algorithms growing linearly as the maze size increases. This shows that they maintain relatively stable efficiency when processing mazes of different sizes. Still, in giant mazes, the number of node visits and execution time is significantly higher than those of DFS and A\* algorithms.

The A\* algorithm shows relatively good balance in different levels of mazes, and the number of node visits and execution time remains at a low growth rate. Especially in giant mazes (level 13 and above), A\* Search shows better efficiency than BFS and Dikstra algorithms. This may be attributed to the fact that A\* Search uses heuristic evaluation during the exploration process, effectively reducing unnecessary node exploration.

# 4. Technical Solution

## 4.1 Index Table

|  |  |  |  |
| --- | --- | --- | --- |
| Section | | Description/Relevance | Technical Solution |
| Key Algorithms | | | |
| 1 | Randomised Prims’s Algorithm | Random edge selection in a growing maze is utilised to iteratively connect isolated subgraphs (cells) until a single spanning tree is formed. This algorithm can generate perfect mazes and provide a fair environment for comparing different pathfinding algorithms. | [Section 4.3.2](#_4.2.2_maze.py) |
| [Section 2.2.1](#_2.1.1_Overall_System) |
| 2 | Depth-First Search | Employs a stack data structure (recursion) to explore as far as possible along a branch before backtracking. It is a simple yet effective approach for unweighted graphs. | [Section 4.3.4](#_4.2.4_pathfinder.py) |
| [Section 2.2.2](#_2.2.2_Depth-first_Search) |
| 3 | Breadth-First Search | It uses a queue to explore all neighbours of a node at the current depth level before moving to the nodes at the next level. It is an algorithm in unweighted graphs that explores equally in all directions. | [Section 4.3.4](#_4.2.4_pathfinder.py) |
| [Section 2.2.3](#_2.2.3_Breadth-first_Search) |
| 4 | Dijkstra’s Search | Employs a priority queue to continuously select the next closest unexplored node, updating distances to each node as it progresses. This algorithm is essential for weighted graphs where the objective is to find the shortest path based on edge weights. | [Section 4.3.4](#_4.2.4_pathfinder.py) |
| [Section 2.2.4](#_2.2.4_Dijkstra’s_Algorithm) |
| 5 | A\* Search | Estimate the cost to the target point from each node, using the priority queue to select the node that minimises the sum of the travelled distance and the heuristic. It is efficient to find the target by narrowing down the search area. | [Section 4.3.4](#_4.2.4_pathfinder.py) |
| [Section 2.2.5](#_2.2.5_A*_Search) |

|  |  |  |  |
| --- | --- | --- | --- |
| Key Data Structures | | | |
| 1 | Dynamic Array | Provides a resizable array structure that allows efficient random access and dynamic resizing to accommodate added or deleted data. It is used to store and manipulate maze structures. | [Section 4.3.2](#_4.2.2_maze.py) |
| [Section 2.3.1](#_2.3.1_Dynamic_Array) |
| 2 | Stack | Because of the recursive nature of DFS and managing backtracking during maze generation or solution. So, the LIFO data structure is used. | [Section 4.3.4](#_4.2.4_pathfinder.py) |
| [Section 2.3.2](#_2.3.2_Stack) |
| 3 | Queue | A FIFO data structure is used for BFS’s level-by-level exploration of nodes, ensuring the shortest path in unweighted graphs. | [Section 4.3.4](#_4.2.4_pathfinder.py) |
| [Section 2.3.3](#_2.3.3_Queue) |
| 4 | Priority Queue (Min Heap) | Because greedy algorithms such as Dijkstra and A\* are involved, this data structure allows efficient extraction of the lowest-cost nodes that can be used to manage the boundaries of explored nodes. | [Section 4.3.4](#_4.2.4_pathfinder.py) |
| [Section 2.3.4](#_2.3.4_Priority_Queue) |
| 5 | Set | Used to keep track of visited nodes or unique elements that are not repeated to avoid loops in pathfinding algorithms and ensure that each node is processed only once. | [Section 4.3.4](#_4.2.4_pathfinder.py) |
| [Section 2.3.5](#_2.3.5_Set) |
| 6 | Enumerations (Enum) | Enhanced code readability and error checking for defining direction or state. | [Section 4.3.1](#_4.2.1_main.py)  [Section 4.3.4](#_4.2.4_pathfinder.py) |
| [Section 2.3.6](#_2.3.6_Enumerations_(Enum)) |
| 7 | 2D Tuple | Represents coordinates in a grid or maze, allowing direct representation of positions or dimensions within a two-dimensional space. | [Section 4.3.4](#_4.2.4_pathfinder.py) |
| [Section 2.3.7](#_2.3.7_2D_Tuple) |
| 8 | List | Acts as a flexible, dynamic container used throughout the application to maintain an ordered collection of elements, from algorithm output to a list of UI components to display random congratulations messages. | [Section 4.3.1](#_4.2.1_main.py) |
| [Section 2.3.8](#_2.3.8_List) |

|  |  |  |  |
| --- | --- | --- | --- |
| Database Design and Management | | | |
| 1 | Create Table Statement | Uses SQLite3 to store, retrieve and manage user data to save usernames and passwords, game progress and other configurations. | [Section 4.3.5](#_4.2.5_authentication.py) |
| [Section 2.4.3](#_Create_Table_Statement) |
| 2 | Insert Statement | [Section 4.3.5](#_4.2.5_authentication.py) |
| [Section 2.4.3](#_Insert_Statement) |
| 3 | Select Statement | [Section 4.3.5](#_4.2.5_authentication.py) |
| [Section 2.4.3](#_Select_Statement) |
| 4 | Update Statement | [Section 4.3.5](#_4.2.5_authentication.py) |
| [Section 2.4.3](#_Update_Statement) |

|  |  |  |  |
| --- | --- | --- | --- |
| GUI Components and User Interface | | | |
| 1 | Main Menu |  | [Section 4.3.1](#_4.2.1_main.py) |
| [Section 2.6.1](#_2.6.1_Main_Menu) |
| 2 | Common UI |  | [Section 4.3.1](#_4.2.1_main.py) |
| [Section 2.6.2](#_2.6.2_Common_UI) |
| 3 | Maze and Pathfinding Interface |  | [Section 4.3.1](#_4.2.1_main.py) |
| [Section 2.6.3](#_2.6.3_Maze_and) |
| 4 | Congratulations Interface |  | [Section 4.3.1](#_4.2.1_main.py) |
| [Section 2.6.4](#_2.6.4_Congratulations_Interface) |
| 5 | Game Level Interface |  | [Section 4.3.6](#_4.2.6_game_level_interface.py) |
| [Section 2.6.5](#_2.6.5_Game_Level) |
| 6 | Login/Register Page |  | [Section 4.3.5](#_4.2.5_authentication.py) |
| [Section 2.6.6](#_2.6.6_Login/Register_Page) |
| 7 | Secondary Menu |  | [Section 4.3.1](#_4.2.1_main.py) |
| [Section 2.6.7](#_2.6.7_Secondary_Menu) |

|  |  |  |  |
| --- | --- | --- | --- |
| Additional Key Components | | | |
| 1 | Player Movement and Control | Capture user input (such as keyboard arrows or WASD keys) via the Pygame module and convert it into on-screen character movement. This requires updating the player's position based on input while ensuring that collisions with walls or obstacles are handled appropriately, preventing the player from walking through them. | [Section 4.3.3](#_4.2.3_player.py) |
| [Section 2.1.3](#_2.1.3_Player_Movement) |

## 4.2 System Architecture and Modular Design

To facilitate updating and maintaining the code, I designed a modular architecture to separate the core part of the algorithm from the GUI. This allows the core functions encapsulated in the module to be independent of the GUI, thus simplifying the independent testing of the core algorithm.

This architecture consists of multiple python files responsible for different responsibilities, as shown below:

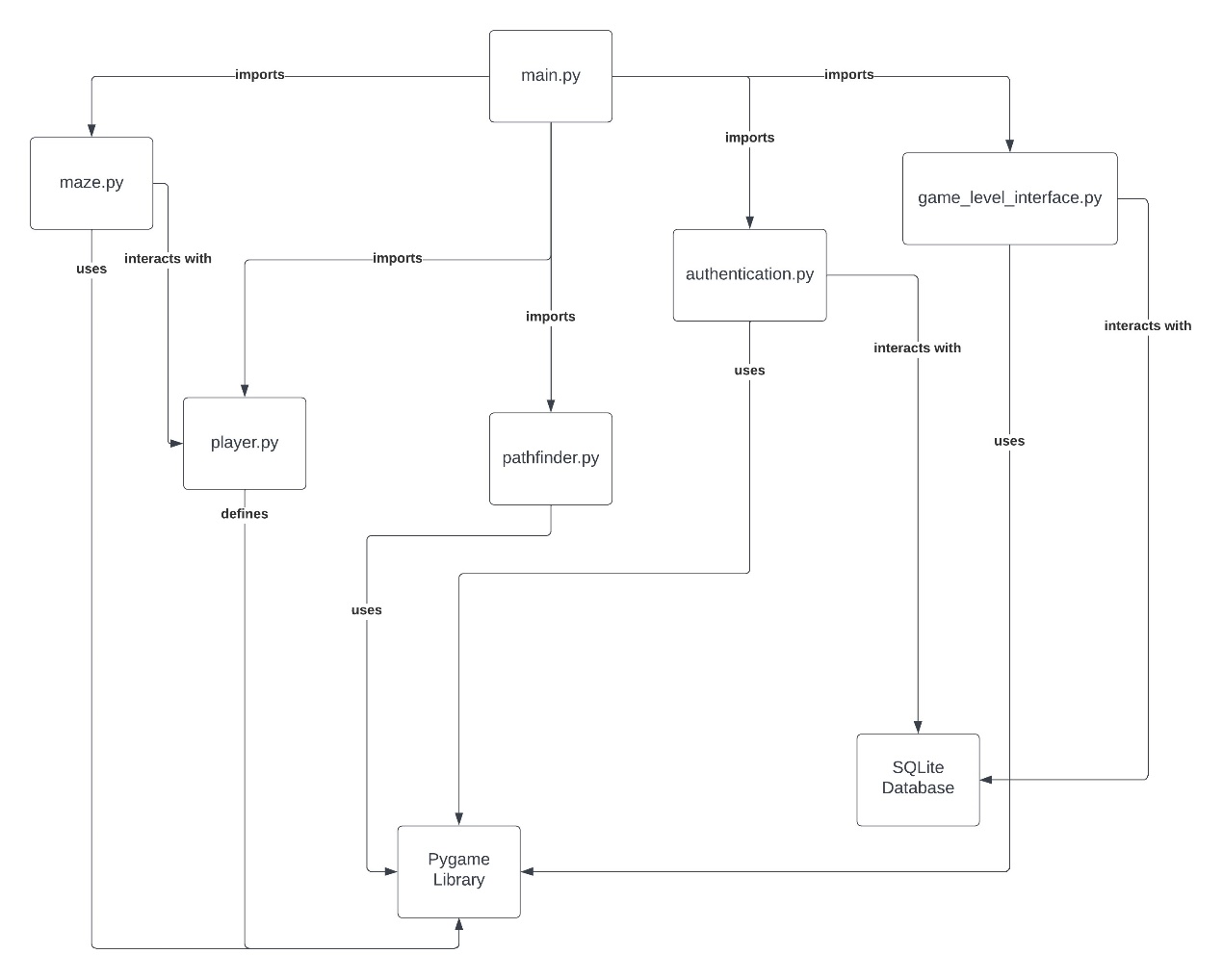


Figure 40

‘main.py’ is the entry point, importing and using other modules. It interacts with the ‘game\_level\_interface.py’ to manage the game flow and levels and with ‘authentication.py’ to handle the user’s login and registration system. The user's gaming progress and account are stored in a SQLite database. For the pathfinding visualiser and comparator, main.py delegates to pathfinder.py, which interacts with mazes generated by ‘maze.py’. The ‘player.py’ module represents and manages the player's actions within the game environment.

## 4.3 Source Code

**Github Address**: <https://github.com/xasolla/NAVI-Simulator>

### 4.3.1 main.py

Acts as a central hub integrating all other modules within the Pygame GUI framework. It coordinates the overall program flow and user interactions.

1. import pygame, sys, sqlite3, traceback, random

2. from pygame.locals import \*

3. from maze import \*

4. from player import \*

5. from pathfinder import \*

6. import authentication

7. import game\_level\_interface

8.

9. # Initialise pygame

10. pygame.init()

11. pygame.display.set\_caption('NAVI')

12. screen = pygame.display.set\_mode((645, 970))

13. pygame.mixer.init()

14.

15. # Define fonts

16. font\_large = pygame.font.Font('font/Comfortaa\_Bold.ttf', 80)

17. font\_medium = pygame.font.Font('font/Comfortaa\_Regular.ttf', 40)

18. font\_small = pygame.font.Font(None, 20)

19.

20. # Define colors

21. color\_black = (0, 0, 0)

22. color\_white = (255, 255, 255)

23. color\_grey = (192, 192, 192)

24. color\_red = (255, 0, 0)

25. color\_light\_grey = (211, 211, 211)

26.

27. size = width,height = 645, 970

28. level = 11;

29.

30. # Define icon

31. icon = pygame.image.load('resources/image/icon.png')

32. pygame.display.set\_icon(icon)

33.

34. # Sound effects

35. sound\_button = pygame.mixer.Sound('resources/sound/button.mp3')

36. sound\_button.set\_volume(1)

37. sound\_move = pygame.mixer.Sound('resources/sound/move.mp3')

38. sound\_move.set\_volume(2)

39. sound\_next = pygame.mixer.Sound('resources/sound/next.mp3')

40. sound\_next.set\_volume(0.5)

41. sound\_reach = pygame.mixer.Sound('resources/sound/reach.mp3')

42. sound\_reach.set\_volume(1)

43. sound\_clap = pygame.mixer.Sound('resources/sound/clap.mp3')

44. sound\_clap.set\_volume(0.5)

45.

46. # Draw the button

47. def drawButton(screen, rect, text, font, text\_color, button\_color, button\_size=None):

48.     # Draw the button surface

49.     button\_surface = pygame.Surface((button\_size[0], button\_size[1])) if button\_size else pygame.Surface(rect.size)

50.     button\_surface.fill(button\_color)

51.     # Draw the button border

52.     pygame.draw.rect(button\_surface, color\_black, (0, 0, button\_surface.get\_width(), button\_surface.get\_height()), 2)

53.     # Draw the text on the button

54.     text\_surface = font.render(text, True, text\_color)

55.     text\_rect = text\_surface.get\_rect(center=(button\_surface.get\_width() / 2, button\_surface.get\_height() / 2))

56.     # Blit the text on the button

57.     button\_surface.blit(text\_surface, text\_rect)

58.     screen.blit(button\_surface, rect)

59.

60. def main():

61.     # Connect to SQLite database (or create it if not exists)

62.     conn = sqlite3.connect('user\_data.db')

63.     cursor = conn.cursor()

64.     # Create a table to store user information

65.     cursor.execute('''

66.         CREATE TABLE IF NOT EXISTS users (

67.             id INTEGER PRIMARY KEY AUTOINCREMENT,

68.             username TEXT NOT NULL,

69.             password TEXT NOT NULL,

70.             level INT DEFAULT 1

71.         )

72.     ''')

73.     # Commit changes and close the database connection

74.     conn.commit()

75.     conn.close()

76.

77.     generate\_maze = False # Variable to control maze generation

78.

79.     # Main menu text

80.     text\_game\_main\_status = font\_small.render(u'v1.1.6', True, color\_black)

81.     text\_game\_main\_status\_rect = text\_game\_main\_status.get\_rect(center=(screen.get\_rect().centerx + 200, screen.get\_rect().centery - 200))

82.

83.     # Main menu logo

84.     image\_main\_menu = pygame.image.load('resources/image/main\_menu.png').convert()

85.     image\_main\_menu\_rect = image\_main\_menu.get\_rect(center=(screen.get\_rect().centerx, screen.get\_rect().centery - 300))

86.

87.     # Medal image

88.     image\_medal = pygame.image.load('resources/image/medal.png').convert\_alpha()

89.     image\_medal\_rect = image\_medal.get\_rect(center=(screen.get\_rect().centerx, screen.get\_rect().centery - 200))

90.

91.     # Play button

92.     button\_play\_rect = pygame.Rect(0, 0, 200, 60)

93.     button\_play\_rect.center = (screen.get\_rect().centerx, screen.get\_rect().centery - 100)

94.     drawButton(screen, button\_play\_rect, 'Play', font\_medium, color\_black, color\_grey)

95.

96.     # Login button

97.     button\_login\_rect = pygame.Rect(0, 0, 200, 60)

98.     button\_login\_rect.center = (screen.get\_rect().centerx, screen.get\_rect().centery)

99.     drawButton(screen, button\_login\_rect, 'Login', font\_medium, color\_black, color\_grey)

100.

101.     # Register button

102.     button\_register\_rect = pygame.Rect(0, 0, 200, 60)

103.     button\_register\_rect.center = (screen.get\_rect().centerx, screen.get\_rect().centery+100)

104.     drawButton(screen, button\_register\_rect, 'Register', font\_medium, color\_black, color\_grey)

105.

106.     # Next Level button

107.     button\_next\_rect = pygame.Rect(0, 0, 200, 60)

108.     button\_next\_rect.center = (screen.get\_rect().centerx, screen.get\_rect().centery)

109.

110.     # Regenerate button

111.     button\_regenerate\_rect = pygame.Rect(0, 0, 100, 40)

112.     button\_regenerate\_rect.center = (screen.get\_rect().centerx - 270, screen.get\_rect().centery +182)

113.

114.     # return button

115.     button\_return\_rect = pygame.Rect(0, 0, 100, 40)

116.     button\_return\_rect.center = (screen.get\_rect().centerx +270, screen.get\_rect().centery +182)

117.

118.     # Previous Level button

119.     button\_previous\_rect = pygame.Rect(0, 0, 100, 40)

120.     button\_previous\_rect.center = (screen.get\_rect().centerx - 168, screen.get\_rect().centery + 182)

121.

122.     # Skip button

123.     button\_skip\_rect = pygame.Rect(0, 0, 100, 40)

124.     button\_skip\_rect.center = (screen.get\_rect().centerx-66, screen.get\_rect().centery + 182)

125.

126.     # DFS button

127.     button\_dfs\_rect = pygame.Rect(0, 0, 100, 40)

128.     button\_dfs\_rect.center = (screen.get\_rect().centerx - 270, screen.get\_rect().centery + 224)

129.

130.     # BFS button

131.     button\_bfs\_rect = pygame.Rect(0, 0, 100, 40)

132.     button\_bfs\_rect.center = (screen.get\_rect().centerx - 168, screen.get\_rect().centery + 224)

133.

134.     # Dijkstra button

135.     button\_dijkstra\_rect = pygame.Rect(0, 0, 100, 40)

136.     button\_dijkstra\_rect.center = (screen.get\_rect().centerx - 66, screen.get\_rect().centery + 224)

137.

138.     # A\* button

139.     button\_astar\_rect = pygame.Rect(0, 0, 100, 40)

140.     button\_astar\_rect.center = (screen.get\_rect().centerx + 36, screen.get\_rect().centery + 224)

141.

142.     # Return to main menu button

143.     return\_main\_text = font\_medium.render('Return', True, color\_red)

144.     return\_main\_text\_rect = return\_main\_text.get\_rect(center=(screen.get\_rect().centerx + 213, screen.get\_rect().centery + 182))

145.

146.     # Maze setup

147.     maze\_min\_level = 11

148.     maze\_max\_level = 43

149.     maze\_level = maze\_min\_level

150.     maze = Map(maze\_level)

151.     maze.drawMap(screen)

152.

153.     # Clear the DFS path

154.     dfs\_path = set()

155.     bfs\_path = set()

156.     dijkstra\_path = set()

157.     astar\_path = set()

158.

159.     # Game states

160.     game\_main\_status = 0

161.     game\_level\_interface\_status = 30

162.     user, level = None, None

163.     Gamestate = game\_main\_status

164.

165.     # Sentences for winning

166.     sentences = [

167.         'You Win!',

168.         'Well Played!',

169.         'Marvellous!',

170.         'Congratulations!',

171.         'Phenomenal!',

172.         'Spectacular!',

173.         'Brilliant Victory!'

174.     ]

175.

176.     # Main game loop

177.     while True:

178.         for event in pygame.event.get():

179.             if event.type == QUIT:

180.                 pygame.quit()

181.                 save\_level\_db(user, level)

182.                 sys.exit()

183.

184.             if not pygame.mixer.music.get\_busy():

185.                 pygame.mixer.music.load('resources/bgm/0.wav')

186.                 pygame.mixer.music.play(-1, 0.0)

187.

188.             # Mouse click event

189.             if Gamestate == game\_main\_status:

190.                 if user is None:

191.                     if event.type == MOUSEBUTTONDOWN:

192.                         if event.button == 1 and button\_play\_rect.collidepoint(event.pos):

193.                                 if user is None:

194.                                     Gamestate = 11

195.                                     maze\_level = min(maze\_level + 2, maze\_max\_level)

196.                                     maze = Map(maze\_level)

197.                                     maze.resetMap(MAP\_ENTRY\_TYPE.MAP\_BLOCK)

198.                                     generate\_maze = True

199.                                     dfs\_path = set()

200.                                     bfs\_path = set()

201.                                     dijkstra\_path = set()

202.                                     astar\_path = set()

203.                                     sound\_next.play()

204.                                 else:

205.                                     Gamestate = game\_level\_interface\_status

206.                                     sound\_button.play()

207.                         elif event.button ==1 and button\_login\_rect.collidepoint(event.pos):

208.                             Gamestate = 21 # Login

209.                             sound\_button.play()

210.                         elif event.button ==1 and button\_register\_rect.collidepoint(event.pos):

211.                             Gamestate = 22 # Register

212.                             sound\_button.play()

213.                     screen.fill(color\_white)

214.                     screen.blit(image\_main\_menu, image\_main\_menu\_rect)

215.                     screen.blit(text\_game\_main\_status, text\_game\_main\_status\_rect)

216.                     drawButton(screen, button\_play\_rect, 'Play', font\_medium, color\_black, color\_grey)

217.                     drawButton(screen, button\_login\_rect, 'Login', font\_medium, color\_black, color\_grey)

218.                     drawButton(screen, button\_register\_rect, 'Register', font\_medium, color\_black, color\_grey)

219.                 else:

220.                     # Display the secondary menu for logged in users

221.                     button\_logout\_rect = pygame.Rect(0, 0, 200, 60)

222.                     button\_logout\_rect.center = (screen.get\_rect().centerx, screen.get\_rect().centery)

223.

224.                     if event.type == MOUSEBUTTONDOWN and button\_play\_rect.collidepoint(event.pos):

225.                         Gamestate = game\_level\_interface\_status

226.                         sound\_button.play()

227.                     elif event.type == MOUSEBUTTONDOWN and button\_logout\_rect.collidepoint(event.pos):

228.                         save\_level\_db(user, level) # Save the current level of the user to the database

229.                         user, level = None, None # Logout the user

230.                         screen.fill(color\_white)

231.                         drawButton(screen, button\_play\_rect, 'Play', font\_medium, color\_black, color\_grey)

232.                         drawButton(screen, button\_login\_rect, 'Login', font\_medium, color\_black, color\_grey)

233.                         drawButton(screen, button\_register\_rect, 'Register', font\_medium, color\_black, color\_grey)

234.                         continue

235.                     # Draw main menu interface

236.                     screen.fill(color\_white)

237.                     screen.blit(image\_main\_menu, image\_main\_menu\_rect)

238.                     screen.blit(text\_game\_main\_status, text\_game\_main\_status\_rect)

239.

240.                     drawButton(screen, button\_play\_rect, 'Continue', font\_medium, color\_black, color\_grey)

241.                     drawButton(screen, button\_logout\_rect, 'Logout', font\_medium, color\_black, color\_grey)

242.

243.             if Gamestate == 21: # Login

244.                 Gamestate, user = authentication.handle\_login\_events(game\_main\_status=game\_main\_status,

245.                                                 Gamestate\_code = [game\_level\_interface\_status],

246.                                                 win\_size=size)

247.                 if user is not None:

248.                     level = query\_db\_level(user=user) # Get the current level of the user from the database

249.

250.             if Gamestate == 22: # Register

251.                 Gamestate, user = authentication.handle\_register\_events(game\_main\_status=game\_main\_status,

252.                 Gamestate\_code = [game\_level\_interface\_status],

253.                                                 win\_size=size,

254.                                                 is\_register=True)

255.                 if user is not None:

256.                     level = query\_db\_level(user=user)

257.

258.             if Gamestate == game\_level\_interface\_status:

259.                 game\_level\_interface\_cls = game\_level\_interface.GameLevelInterface(

260.                                                                win\_size=size,

261.                                                                 game\_status\_code = [game\_main\_status, 11],

262.                                                                screen=screen,

263.                                                                user=user,

264.                                                                level=level,

265.                                                                return\_main\_text=(return\_main\_text\_rect, return\_main\_text),

266.                                                                common\_methond=(common\_ui,[user, screen, return\_main\_text, return\_main\_text\_rect]))

267.                 Gamestate = game\_level\_interface\_cls.deal()

268.

269.                 # If the user has selected a level, then start the game

270.                 if isinstance(Gamestate, list) and len(Gamestate) == 2:

271.                     new\_game\_state, selected\_level = Gamestate

272.

273.                     if new\_game\_state == 11:

274.                         # Convert the selected level to an odd number for the maze generation

275.                         maze\_level = max(min(selected\_level, maze\_max\_level), maze\_min\_level)

276.                         if maze\_level % 2 == 0:

277.                             maze\_level += 1  # Make sure the maze level is odd

278.

279.                         screen.fill(color\_white)

280.                         maze = Map(maze\_level)

281.                         common\_ui(user, screen, return\_main\_text, return\_main\_text\_rect)

282.                         generate\_maze = True

283.                         Gamestate = 18

284.

285.             if Gamestate == 11: # Gamestate for maze generation and pathfinding

286.                 # Draw the maze

287.                 if generate\_maze:

288.                     screen.fill(color\_white)

289.                     doRandomPrim(maze,screen)

290.                     generate\_maze = False  # Reset the flag to prevent continuous maze generation

291.                     Gamestate = 18

292.                     common\_ui(user, screen, return\_main\_text, return\_main\_text\_rect)

293.                     drawButton(screen, button\_regenerate\_rect, 'Regenerate', font\_small, color\_black, color\_grey, button\_size=(100, 40))

294.                     drawButton(screen, button\_previous\_rect, 'Previous Level', font\_small, color\_black, color\_grey, button\_size=(100, 40))

295.                     drawButton(screen, button\_skip\_rect, 'Skip', font\_small, color\_black, color\_grey, button\_size=(100, 40))

296.

297.                     drawButton(screen, button\_dfs\_rect, 'DFS', font\_small, color\_black, color\_grey, button\_size=(100, 40))

298.                     drawButton(screen, button\_bfs\_rect, 'BFS', font\_small, color\_black, color\_grey, button\_size=(100, 40))

299.                     drawButton(screen, button\_dijkstra\_rect, 'Dijkstra', font\_small, color\_black, color\_grey, button\_size=(100, 40))

300.                     drawButton(screen, button\_astar\_rect, 'A\*', font\_small, color\_black, color\_grey, button\_size=(100, 40))

301.

302.                 maze.drawMap(screen) # Draw the maze on the screen

303.                 if event.type == MOUSEBUTTONDOWN:

304.                     if event.button == 1:

305.                         if button\_regenerate\_rect.collidepoint(event.pos): # Regenerate the maze and reset the path

306.                             dfs\_path = set()

307.                             bfs\_path = set()

308.                             dijkstra\_path = set()

309.                             astar\_path = set()

310.                             generate\_maze = True

311.                             maze.resetMap(MAP\_ENTRY\_TYPE.MAP\_BLOCK)

312.                             sound\_button.play()

313.

314.                         if button\_previous\_rect.collidepoint(event.pos):

315.                             maze\_level = max(maze\_level - 2, maze\_min\_level)  # choose the level between 11 and 43, the level is odd

316.                             maze = Map(maze\_level)

317.                             maze.resetMap(MAP\_ENTRY\_TYPE.MAP\_BLOCK)

318.                             dfs\_path = set()

319.                             bfs\_path = set()

320.                             dijkstra\_path = set()

321.                             astar\_path = set()

322.                             generate\_maze = True

323.                             sound\_button.play()

324.

325.                         if button\_skip\_rect.collidepoint(event.pos):

326.                             selected\_sentence = sentences[random.randint(0, len(sentences) - 1)]  # Randomly select a congrats sentence from the list

327.                             sound\_reach.play()

328.                             dfs\_path = set()

329.                             bfs\_path = set()

330.                             dijkstra\_path = set()

331.                             astar\_path = set()

332.                             Gamestate = 100 # Gamestate for winning

333.

334.                         if button\_return\_rect.collidepoint(event.pos):

335.                             Gamestate = game\_main\_status

336.                             sound\_button.play()

337.                             dfs\_path = set()

338.                             bfs\_path = set()

339.                             dijkstra\_path = set()

340.                             astar\_path = set()

341.                             continue

342.

343.                         # Check if the mouse click is within the maze area

344.                         if (maze.player.x, maze.player.y) in dfs\_path:

345.                             pygame.draw.rect(screen, color\_red, (maze.player.x \* CELL\_SIZE, maze.player.y \* CELL\_SIZE, CELL\_SIZE, CELL\_SIZE))

346.                         if (maze.player.x, maze.player.y) in bfs\_path:

347.                             pygame.draw.rect(screen, color\_red, (maze.player.x \* CELL\_SIZE, maze.player.y \* CELL\_SIZE, CELL\_SIZE, CELL\_SIZE))

348.                         if (maze.player.x, maze.player.y) in dijkstra\_path:

349.                             pygame.draw.rect(screen, color\_red, (maze.player.x \* CELL\_SIZE, maze.player.y \* CELL\_SIZE, CELL\_SIZE, CELL\_SIZE))

350.                         if (maze.player.x, maze.player.y) in astar\_path:

351.                             pygame.draw.rect(screen, color\_red, (maze.player.x \* CELL\_SIZE, maze.player.y \* CELL\_SIZE, CELL\_SIZE, CELL\_SIZE))

352.

353.                         if button\_dfs\_rect.collidepoint(event.pos):

354.                             Gamestate = 12

355.                             sound\_button.play()

356.                         if button\_bfs\_rect.collidepoint(event.pos):

357.                             Gamestate = 13

358.                             sound\_button.play()

359.                         if button\_dijkstra\_rect.collidepoint(event.pos):

360.                             Gamestate = 14

361.                             sound\_button.play()

362.                         if button\_astar\_rect.collidepoint(event.pos):

363.                             Gamestate = 15

364.                             sound\_button.play()

365.

366.                 elif event.type == KEYDOWN:

367.                     if event.key == K\_UP or event.key == K\_w:

368.                         move\_player(maze, PLAYER\_DIRECTION.UP)

369.                         sound\_move.play()

370.                     elif event.key == K\_DOWN or event.key == K\_s:

371.                         move\_player(maze, PLAYER\_DIRECTION.DOWN)

372.                         sound\_move.play()

373.                     elif event.key == K\_LEFT or event.key == K\_a:

374.                         move\_player(maze, PLAYER\_DIRECTION.LEFT)

375.                         sound\_move.play()

376.                     elif event.key == K\_RIGHT or event.key == K\_d:

377.                         move\_player(maze, PLAYER\_DIRECTION.RIGHT)

378.                         sound\_move.play()

379.

380.                 # Check if the player has reached the goal

381.                 if maze.player.x == maze.width - 2 and maze.player.y == maze.height - 2:

382.                     sound\_reach.play()

383.                     selected\_sentence = sentences[random.randint(0, len(sentences) - 1)]

384.                     Gamestate = 100

385.

386.             if Gamestate == 18: # Gamestate for maze generation and pathfinding

387.                 # Placeholder values for the table

388.                 dfs\_nodes\_visited, dfs\_path\_length, dfs\_execution\_time = dataDFS(maze, (maze.player.x, maze.player.y), (maze.width - 2, maze.height - 2))

389.                 bfs\_nodes\_visited, bfs\_path\_length, bfs\_execution\_time = dataBFS(maze, (maze.player.x, maze.player.y), (maze.width - 2, maze.height - 2))

390.                 dij\_nodes\_visited, dij\_path\_length, dij\_execution\_time = dataDijkstra(maze, (maze.player.x, maze.player.y), (maze.width - 2, maze.height - 2))

391.                 astar\_nodes\_visited, astar\_path\_length, astar\_execution\_time = dataAStar(maze, (maze.player.x, maze.player.y), (maze.width - 2, maze.height - 2))

392.

393.                 # Draw the table with placeholder values

394.                 text\_type = font\_small.render('      Path Length                        Nodes Visited                        Execution Time (ms)', True, color\_black)

395.                 text\_result1 = font\_small.render('DFS                ' + str(dfs\_path\_length).zfill(3) + '                                        '

396.                                                  + str(dfs\_nodes\_visited).zfill(3) + '                                         '

397.                                                  + format(dfs\_execution\_time, '.3g'), True, color\_black

398.                                                  )

399.                 text\_result2 = font\_small.render('BFS                '

400.                                                  + str(bfs\_path\_length).zfill(3) + '                                        '

401.                                                  + str(bfs\_nodes\_visited).zfill(3) + '                                         '

402.                                                  + format(bfs\_execution\_time, '.3g'), True, color\_black

403.                                                  )

404.                 text\_result3 = font\_small.render('DIJ                  ' + str(dij\_path\_length).zfill(3)

405.                                                  + '                                        '

406.                                                  + str(dij\_nodes\_visited).zfill(3)

407.                                                  + '                                         '

408.                                                  + format(dij\_execution\_time, '.3g'), True, color\_black

409.                                                  )

410.                 text\_result4 = font\_small.render('A\*S                 '

411.                                                  + str(astar\_path\_length).zfill(3)

412.                                                  + '                                        '

413.                                                  + str(astar\_nodes\_visited).zfill(3)

414.                                                  + '                                         '

415.                                                   + format(astar\_execution\_time, '.3g'), True, color\_black

416.                                                  )

417.

418.                 # Adjust the positions for visibility

419.                 screen.blit(text\_type, (90, 760))

420.                 screen.blit(text\_result1, (50, 800))

421.                 screen.blit(text\_result2, (50, 840))

422.                 screen.blit(text\_result3, (50, 880))

423.                 screen.blit(text\_result4, (50, 920))

424.

425.                 Gamestate = 11

426.

427.             if Gamestate == 12: # Gamestate for DFS pathfinding

428.                 dfs\_path = searchDFS(maze, (maze.player.x, maze.player.y), (maze.width - 2, maze.height - 2), screen)  # Added screen parameter

429.                 Gamestate = 11

430.             if Gamestate == 13: # Gamestate for BFS pathfinding

431.                 bfs\_path = searchBFS(maze, (maze.player.x, maze.player.y), (maze.width - 2, maze.height - 2), screen)

432.                 Gamestate = 11

433.             if Gamestate == 14: # Gamestate for Dijkstra pathfinding

434.                 dijkstra\_path = searchDijkstra(maze, (maze.player.x, maze.player.y), (maze.width - 2, maze.height - 2), screen)

435.                 Gamestate = 11

436.             if Gamestate == 15: # Gamestate for A\* pathfinding

437.                 astar\_path = searchAStar(maze, (maze.player.x, maze.player.y), (maze.width - 2, maze.height - 2), screen)

438.                 Gamestate = 11

439.

440.             if Gamestate == 100: # Gamestate for winning

441.                 sound\_clap.play()

442.                 screen.fill(color\_white)

443.

444.                 # Randomly select a sentence

445.                 text\_win = font\_large.render(selected\_sentence, True, color\_black)

446.

447.                 text\_win\_rect = text\_win.get\_rect(center=(screen.get\_rect().centerx, screen.get\_rect().centery - 400))

448.                 screen.blit(text\_win, text\_win\_rect)

449.                 screen.blit(image\_medal, image\_medal\_rect)

450.                 drawButton(screen, button\_next\_rect, 'Next Level', font\_medium, color\_black, color\_grey, button\_size=(200, 60))

451.                 common\_ui(user, screen, return\_main\_text, return\_main\_text\_rect)

452.

453.                 if event.type == MOUSEBUTTONDOWN:

454.                     sound\_clap.stop()

455.                     if event.button == 1 and button\_next\_rect.collidepoint(event.pos):

456.                         Gamestate = 11

457.                         if user is not None:

458.                             level += 1

459.                             save\_level\_db(user, level)

460.                         pygame.mixer.music.stop()

461.                         if not pygame.mixer.music.get\_busy():

462.                             sound\_next.play()

463.                             pygame.mixer.music.load('resources/bgm/%s.wav' % random.randint(0, 10))

464.                             pygame.mixer.music.set\_volume(1.5)

465.                             pygame.mixer.music.play(-1, 0.0)

466.

467.                         maze\_level = min(maze\_level + 2, maze\_max\_level)

468.                         maze = Map(maze\_level)

469.                         maze.resetMap(MAP\_ENTRY\_TYPE.MAP\_BLOCK)

470.                         generate\_maze = True

471.                         dfs\_path = set()

472.                         bfs\_path = set()

473.                         dijkstra\_path = set()

474.                         astar\_path = set()

475.                         sound\_button.play()

476.                     if event.button == 1 and button\_return\_rect.collidepoint(event.pos):

477.                         Gamestate = game\_main\_status

478.                         sound\_button.play()

479.                         dfs\_path = set()

480.                         bfs\_path = set()

481.                         dijkstra\_path = set()

482.                         astar\_path = set()

483.                         continue

484.

485.         # Draw the shortest path in red

486.         for position in dfs\_path:

487.             pygame.draw.rect(screen, color\_red, (position[0] \* CELL\_SIZE, position[1] \* CELL\_SIZE, CELL\_SIZE, CELL\_SIZE))

488.         for position in bfs\_path:

489.             pygame.draw.rect(screen, color\_red, (position[0] \* CELL\_SIZE, position[1] \* CELL\_SIZE, CELL\_SIZE, CELL\_SIZE))

490.         for position in dijkstra\_path:

491.             pygame.draw.rect(screen, color\_red, (position[0] \* CELL\_SIZE, position[1] \* CELL\_SIZE, CELL\_SIZE, CELL\_SIZE))

492.         for position in astar\_path:

493.             pygame.draw.rect(screen, color\_red, (position[0] \* CELL\_SIZE, position[1] \* CELL\_SIZE, CELL\_SIZE, CELL\_SIZE))

494.

495.         # Update screen

496.         pygame.display.update()

497.         # Tick clock

498.         pygame.time.Clock().tick(60)

499.

500. # Common UI for all game states except the main menu and login/register interface

501. def common\_ui(user, screen:pygame.Surface,

502.               return\_main\_text:pygame.Surface,

503.               return\_main\_text\_rect:Rect):

504.     screen.blit(return\_main\_text, return\_main\_text\_rect)

505.     if user is None:

506.         user = 'Visitor!not login'

507.     else:

508.         user += '! logged in'

509.     margin = 2

510.     user\_text\_surface = font\_small.render(user , True, (color\_black))

511.     user\_text\_rect = user\_text\_surface.get\_rect()

512.     user\_text\_rect.center = (return\_main\_text\_rect.center[0],

513.                              return\_main\_text\_rect.bottom+user\_text\_rect.height//2 +margin)

514.     screen.blit(user\_text\_surface, user\_text\_rect)

515.

516. # Query the database to get the current level of the user

517. def query\_db\_level(user):

518.     with sqlite3.connect('user\_data.db') as conn:

519.         cursor = conn.cursor()

520.         cursor.execute('SELECT level FROM users WHERE username=?', (user,))

521.         level = cursor.fetchone()

522.         return level[0]

523.

524. # Save the current level of the user to the database

525. def save\_level\_db(user, level):

526.     with sqlite3.connect('user\_data.db') as conn:

527.         cursor = conn.cursor()

528.         cursor.execute('update users SET level=? WHERE username=?', (level, user))

529.         conn.commit()

530.

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

532.     try:

533.         main()

534.     except SystemExit:

535.         pass

536.     except:

537.         traceback.print\_exc() # Print the traceback to the console

538.         pygame.quit()

539.         input() # Wait for the user to press enter before closing the window

540.

### 4.3.2 maze.py

Generates complex and perfect mazes using the Randomised Prim algorithm. This module provides the environment for the pathfinding algorithms to operate within.

1. import pygame

2. from random import randint, choice

3. from pygame.locals import \*

4. from enum import Enum

5. from player import \*

6.

7. # Define colors

8. color\_path = (255, 255, 255)

9. color\_wall = (192, 192, 192)

10. color\_start = (255, 0, 0)

11. color\_end = (255, 215, 0)

12.

13. CELL\_SIZE = 15 # Size of each cell in the maze

14.

15. # Enum to represent the type of map entry

16. class MAP\_ENTRY\_TYPE(Enum):

17.     MAP\_EMPTY = 0

18.     MAP\_BLOCK = 1

19.

20. # Enum to represent the direction of the wall

21. class WALL\_DIRECTION(Enum):

22.     WALL\_LEFT = 0

23.     WALL\_UP = 1

24.     WALL\_RIGHT = 2

25.     WALL\_DOWN = 3

26.

27. # Class to represent the maze map

28. class Map:

29.     def \_\_init\_\_(self, maze\_level):

30.         self.width = maze\_level

31.         self.height = maze\_level

32.         self.map = [[0 for x in range(self.width)] for y in range(self.height)]  # 2D array, 0 represents empty, 1 represents wall

33.         self.player = Player(1, 1)  # Initialize player at (1, 1)

34.

35.     # Reset the map to all walls

36.     def resetMap(self, value):

37.         for y in range(self.height):

38.             for x in range(self.width):

39.                 self.setMap(x, y, value)

40.

41.     # Set the value of the map at position (x, y) to value (0 or 1)

42.     def setMap(self, x, y, value):

43.         if value == MAP\_ENTRY\_TYPE.MAP\_EMPTY:

44.             self.map[y][x] = 0

45.         elif value == MAP\_ENTRY\_TYPE.MAP\_BLOCK:

46.             self.map[y][x] = 1

47.

48.     # Check if the position (x, y) is visited

49.     def isVisited(self, x, y):

50.         return self.map[y][x] != 1

51.

52.     # Draw the maze, the player and the end point on the screen

53.     def drawMap(self, screen, CELL\_SIZE=CELL\_SIZE):

54.         for y in range(self.height):

55.             for x in range(self.width):

56.                 color = (color\_wall) if self.map[y][x] == 1 else (color\_path)

57.                 pygame.draw.rect(screen, color, (x \* CELL\_SIZE, y \* CELL\_SIZE, CELL\_SIZE, CELL\_SIZE))

58.

59.         # Draw the player

60.         pygame.draw.rect(screen, color\_start, (self.player.x \* CELL\_SIZE, self.player.y \* CELL\_SIZE, CELL\_SIZE, CELL\_SIZE))

61.

62.         # Draw the end point

63.         pygame.draw.rect(screen, color\_end, ((self.width - 2) \* CELL\_SIZE, (self.height - 2) \* CELL\_SIZE, CELL\_SIZE, CELL\_SIZE))

64.

65. # Check if the adjacent positions of (x, y) are visited

66. def checkAdjacentPos(map, x, y, width, height, checklist):

67.     directions = []

68.

69.     # Check left

70.     if x > 0 and not map.isVisited(2 \* (x - 1) + 1, 2 \* y + 1):

71.         directions.append(WALL\_DIRECTION.WALL\_LEFT)

72.

73.     # Check up

74.     if y > 0 and not map.isVisited(2 \* x + 1, 2 \* (y - 1) + 1):

75.         directions.append(WALL\_DIRECTION.WALL\_UP)

76.

77.     # Check right

78.     if x < width - 1 and not map.isVisited(2 \* (x + 1) + 1, 2 \* y + 1):

79.         directions.append(WALL\_DIRECTION.WALL\_RIGHT)

80.

81.     # Check down

82.     if y < height - 1 and not map.isVisited(2 \* x + 1, 2 \* (y + 1) + 1):

83.         directions.append(WALL\_DIRECTION.WALL\_DOWN)

84.

85.     if len(directions):

86.         direction = choice(directions)

87.         # Mark adjacent positions as empty

88.         if direction == WALL\_DIRECTION.WALL\_LEFT:

89.             map.setMap(2 \* (x - 1) + 1, 2 \* y + 1, MAP\_ENTRY\_TYPE.MAP\_EMPTY)

90.             map.setMap(2 \* x, 2 \* y + 1, MAP\_ENTRY\_TYPE.MAP\_EMPTY)

91.             checklist.append((x - 1, y))

92.         elif direction == WALL\_DIRECTION.WALL\_UP:

93.             map.setMap(2 \* x + 1, 2 \* (y - 1) + 1, MAP\_ENTRY\_TYPE.MAP\_EMPTY)

94.             map.setMap(2 \* x + 1, 2 \* y, MAP\_ENTRY\_TYPE.MAP\_EMPTY)

95.             checklist.append((x, y - 1))

96.         elif direction == WALL\_DIRECTION.WALL\_RIGHT:

97.             map.setMap(2 \* (x + 1) + 1, 2 \* y + 1, MAP\_ENTRY\_TYPE.MAP\_EMPTY)

98.             map.setMap(2 \* x + 2, 2 \* y + 1, MAP\_ENTRY\_TYPE.MAP\_EMPTY)

99.             checklist.append((x + 1, y))

100.         elif direction == WALL\_DIRECTION.WALL\_DOWN:

101.             map.setMap(2 \* x + 1, 2 \* (y + 1) + 1, MAP\_ENTRY\_TYPE.MAP\_EMPTY)

102.             map.setMap(2 \* x + 1, 2 \* y + 2, MAP\_ENTRY\_TYPE.MAP\_EMPTY)

103.             checklist.append((x, y + 1))

104.         return True

105.     else:

106.         return False

107.

108. # Randomised Prim's algorithm to generate maze

109. def randomPrim(map, width, height, screen, CELL\_SIZE=CELL\_SIZE):

110.

111.     startX, startY = (randint(1, width - 1), randint(1, height - 1)) # random generate maze position

112.

113.     # Mark start and end positions empty

114.     map.setMap(2 \* startX + 1, 2 \* startY + 1, MAP\_ENTRY\_TYPE.MAP\_EMPTY)

115.

116.     checklist = []

117.     checklist.append((startX, startY))

118.     while len(checklist):

119.         entry = choice(checklist)

120.         if not checkAdjacentPos(map, entry[0], entry[1], width, height, checklist):

121.             # The entry has no unvisited adjacent entry, so remove it from the checklist

122.             checklist.remove(entry)

123.

124.         # Draw the maze after each step

125.         map.drawMap(screen)

126.

127.         pygame.display.flip()

128.

129.         pygame.time.delay(1)

130.

131. # Generate a random maze

132. def doRandomPrim(map, screen):

133.     map.resetMap(MAP\_ENTRY\_TYPE.MAP\_BLOCK)  # Reset map to all walls

134.     randomPrim(map, (map.width - 1) // 2, (map.height - 1) // 2, screen)

135.

### 4.3.3 player.py

Forms part of the GUI, defining the player character and interactions within the maze, including movement and collision detection.

1. from enum import Enum

2.

3. # Enumeration for player directions

4. class PLAYER\_DIRECTION(Enum):

5.     UP = 0

6.     DOWN = 1

7.     LEFT = 2

8.     RIGHT = 3

9.

10. # Class to represent the player

11. class Player:

12.     def \_\_init\_\_(self, start\_x, start\_y):

13.         self.x = start\_x

14.         self.y = start\_y

15.

16.     # Function to move the player in the specified direction, updating the player's position if possible

17.     def move(self, direction):

18.         if direction == PLAYER\_DIRECTION.UP:

19.             self.y -= 1

20.         elif direction == PLAYER\_DIRECTION.DOWN:

21.             self.y += 1

22.         elif direction == PLAYER\_DIRECTION.LEFT:

23.             self.x -= 1

24.         elif direction == PLAYER\_DIRECTION.RIGHT:

25.             self.x += 1

26.

27. # Function to move the player in the specified direction if possible (i.e. if the new position is within the maze boundaries and is a valid path)

28. def move\_player(map, direction):

29.     new\_x, new\_y = map.player.x, map.player.y

30.

31.     if direction == PLAYER\_DIRECTION.UP:

32.         new\_y -= 1

33.     elif direction == PLAYER\_DIRECTION.DOWN:

34.         new\_y += 1

35.     elif direction == PLAYER\_DIRECTION.LEFT:

36.         new\_x -= 1

37.     elif direction == PLAYER\_DIRECTION.RIGHT:

38.         new\_x += 1

39.

40.     # Check if the new position is within the maze boundaries and is a valid path

41.     if 0 <= new\_x < map.width and 0 <= new\_y < map.height and map.map[new\_y][new\_x] == 0:

42.         map.player.move(direction)

43.

### 4.3.4 pathfinder.py

Implements various pathfinding algorithms, including depth-first search, breadth-first search, Dijkstra, and A\* search algorithms. It allows comparative analysis and visualisation of these algorithms running in mazes generated by ‘maze.py’

1. import pygame, sys, heapq, time

2. from enum import Enum

3. from pygame.locals import \*

4.

5. color\_red = (255, 0, 0)

6.

7. CELL\_SIZE = 15

8.

9. # Enumeration for player directions (used in player.py)

10. class DIRECTION(Enum):

11.     UP = (0, -1)

12.     DOWN = (0, 1)

13.     LEFT = (-1, 0)

14.     RIGHT = (1, 0)

15.

16. # Depth-First Search algorithm for maze solving

17. def searchDFS(maze, start, end, screen, delay=30):

18.     stack = [(start, [])]

19.     visited = set()

20.

21.     while stack:

22.         for event in pygame.event.get():

23.             if event.type == QUIT:

24.                 pygame.quit()

25.                 sys.exit()

26.

27.         current\_position, path = stack.pop()

28.         x, y = current\_position

29.

30.         if current\_position == end:

31.             return set(path + [current\_position])

32.

33.         if current\_position not in visited:

34.             visited.add(current\_position)

35.

36.             for direction in DIRECTION:

37.                 new\_x, new\_y = x + direction.value[0], y + direction.value[1]

38.

39.                 if 0 <= new\_x < maze.width and 0 <= new\_y < maze.height and maze.map[new\_y][new\_x] == 0:

40.                     stack.append(((new\_x, new\_y), path + [current\_position]))

41.

42.                     # Visualize the process by drawing the maze and the current path

43.                     maze.drawMap(screen)

44.                     for position in path + [current\_position]:

45.                         pygame.draw.rect(screen, color\_red, (position[0] \* CELL\_SIZE, position[1] \* CELL\_SIZE, CELL\_SIZE, CELL\_SIZE))

46.

47.                     # Delay to slow down visualization (in milliseconds)

48.                     pygame.time.delay(delay)

49.                     # Update display

50.                     pygame.display.flip()

51.     return set()  # No path found

52.

53. def searchBFS(maze, start, end, screen, delay=30):

54.     queue = [(start, [])]

55.     visited = set()

56.

57.     while queue:

58.         for event in pygame.event.get():

59.             if event.type == QUIT:

60.                 pygame.quit()

61.                 sys.exit()

62.

63.         current\_position, path = queue.pop(0)

64.         x, y = current\_position

65.

66.         if current\_position == end:

67.             return set(path + [current\_position])

68.

69.         if current\_position not in visited:

70.             visited.add(current\_position)

71.

72.             for direction in DIRECTION:

73.                 new\_x, new\_y = x + direction.value[0], y + direction.value[1]

74.

75.                 if 0 <= new\_x < maze.width and 0 <= new\_y < maze.height and maze.map[new\_y][new\_x] == 0:

76.                     queue.append(((new\_x, new\_y), path + [current\_position]))

77.

78.                     # Visualize the process by drawing the maze and the current path

79.                     maze.drawMap(screen)

80.                     for position in path + [current\_position]:

81.                         pygame.draw.rect(screen, color\_red, (position[0] \* CELL\_SIZE, position[1] \* CELL\_SIZE, CELL\_SIZE, CELL\_SIZE))

82.

83.                     # Update display

84.                     pygame.display.flip()

85.

86.                     # Delay to slow down visualization (in milliseconds)

87.                     pygame.time.delay(delay)

88.

89.     return set()  # No path found

90.

91. def searchDijkstra(maze, start, end, screen, delay=30):

92.     priority\_queue = [(0, start, [])]

93.     visited = set()

94.

95.     while priority\_queue:

96.         for event in pygame.event.get():

97.             if event.type == QUIT:

98.                 pygame.quit()

99.                 sys.exit()

100.

101.         cost, current\_position, path = heapq.heappop(priority\_queue)

102.         x, y = current\_position

103.

104.         if current\_position == end:

105.             return set(path + [current\_position])

106.

107.         if current\_position not in visited:

108.             visited.add(current\_position)

109.

110.             for direction in DIRECTION:

111.                 new\_x, new\_y = x + direction.value[0], y + direction.value[1]

112.

113.                 if 0 <= new\_x < maze.width and 0 <= new\_y < maze.height and maze.map[new\_y][new\_x] == 0:

114.                     new\_cost = cost + 1  # Uniform cost for each move

115.

116.                     heapq.heappush(priority\_queue, (new\_cost, (new\_x, new\_y), path + [current\_position]))

117.

118.                     # Visualize the process by drawing the maze and the current path

119.                     maze.drawMap(screen)

120.                     for position in path + [current\_position]:

121.                         pygame.draw.rect(screen, color\_red, (position[0] \* CELL\_SIZE, position[1] \* CELL\_SIZE, CELL\_SIZE, CELL\_SIZE))

122.

123.                     # Update display

124.                     pygame.display.flip()

125.

126.                     # Delay to slow down visualization (in milliseconds)

127.                     pygame.time.delay(delay)

128.

129.     return set()  # No path found

130.

131. def heuristic(node, goal):

132.     # A simple Manhattan distance heuristic for A\*

133.     return abs(node[0] - goal[0]) + abs(node[1] - goal[1])

134.

135. def searchAStar(maze, start, end, screen, delay=30):

136.     priority\_queue = [(0, start, [])]

137.     visited = set()

138.

139.     while priority\_queue:

140.         for event in pygame.event.get():

141.             if event.type == QUIT:

142.                 pygame.quit()

143.                 sys.exit()

144.

145.         cost, current\_position, path = heapq.heappop(priority\_queue)

146.         x, y = current\_position

147.

148.         if current\_position == end:

149.             return set(path + [current\_position])

150.

151.         if current\_position not in visited:

152.             visited.add(current\_position)

153.

154.             for direction in DIRECTION:

155.                 new\_x, new\_y = x + direction.value[0], y + direction.value[1]

156.

157.                 if 0 <= new\_x < maze.width and 0 <= new\_y < maze.height and maze.map[new\_y][new\_x] == 0:

158.                     new\_cost = cost + 1  # Uniform cost for each move

159.                     heuristic\_cost = heuristic((new\_x, new\_y), end)

160.

161.                     heapq.heappush(priority\_queue, (new\_cost + heuristic\_cost, (new\_x, new\_y), path + [current\_position]))

162.

163.                     # Visualize the process by drawing the maze and the current path

164.                     maze.drawMap(screen)

165.                     for position in path + [current\_position]:

166.                         pygame.draw.rect(screen, color\_red, (position[0] \* CELL\_SIZE, position[1] \* CELL\_SIZE, CELL\_SIZE, CELL\_SIZE))

167.

168.                     # Update display

169.                     pygame.display.flip()

170.

171.                     # Delay to slow down visualization (in milliseconds)

172.                     pygame.time.delay(delay)

173.

174.     return set()  # No path found

175.

176. def dataDFS(maze, start, end):

177.     start\_time = time.time()  # Start time measurement

178.     stack = [(start, [])]

179.     visited = set()

180.     nodes\_explored = 0

181.

182.     while stack:

183.         current\_position, path = stack.pop()

184.         x, y = current\_position

185.

186.         if current\_position == end:

187.             end\_time = time.time()

188.             time\_taken = (end\_time - start\_time) \* 1000  # Convert to milliseconds

189.             return  nodes\_explored, len(path), time\_taken

190.

191.         if current\_position not in visited:

192.             visited.add(current\_position)

193.             nodes\_explored += 1  # Increment nodes explored

194.

195.             for direction in DIRECTION:

196.                 new\_x, new\_y = x + direction.value[0], y + direction.value[1]

197.

198.                 if 0 <= new\_x < maze.width and 0 <= new\_y < maze.height and maze.map[new\_y][new\_x] == 0:

199.                     stack.append(((new\_x, new\_y), path + [current\_position]))

200.

201.     time\_taken = time.time() - start\_time

202.     return 0, 0, time\_taken  # No path found

203.

204. def dataBFS(maze, start, end):

205.     start\_time = time.time()  # Start time measurement

206.     queue = [(start, [])]

207.     visited = set()

208.     nodes\_explored = 0

209.

210.     while queue:

211.         current\_position, path = queue.pop(0)

212.         x, y = current\_position

213.

214.         if current\_position == end:

215.             end\_time = time.time()

216.             time\_taken = (end\_time - start\_time) \* 1000  # Convert to milliseconds

217.             return nodes\_explored, len(path), time\_taken

218.

219.         if current\_position not in visited:

220.             visited.add(current\_position)

221.             nodes\_explored += 1  # Increment nodes explored

222.

223.             for direction in DIRECTION:

224.                 new\_x, new\_y = x + direction.value[0], y + direction.value[1]

225.

226.                 if 0 <= new\_x < maze.width and 0 <= new\_y < maze.height and maze.map[new\_y][new\_x] == 0:

227.                     queue.append(((new\_x, new\_y), path + [current\_position]))

228.

229.     time\_taken = time.time() - start\_time

230.     return 0, 0, time\_taken  # No path found

231.

232. def dataDijkstra(maze, start, end):

233.     start\_time = time.time()  # Start time measurement

234.     priority\_queue = [(0, start, [])]

235.     visited = set()

236.     nodes\_explored = 0

237.

238.     while priority\_queue:

239.         cost, current\_position, path = heapq.heappop(priority\_queue)

240.         x, y = current\_position

241.

242.         if current\_position == end:

243.             end\_time = time.time()

244.             time\_taken = (end\_time - start\_time) \* 1000

245.             return nodes\_explored, len(path), time\_taken

246.

247.         if current\_position not in visited:

248.             visited.add(current\_position)

249.             nodes\_explored += 1  # Increment nodes explored

250.

251.             for direction in DIRECTION:

252.                 new\_x, new\_y = x + direction.value[0], y + direction.value[1]

253.

254.                 if 0 <= new\_x < maze.width and 0 <= new\_y < maze.height and maze.map[new\_y][new\_x] == 0:

255.                     new\_cost = cost + 1  # Uniform cost for each move

256.

257.                     heapq.heappush(priority\_queue, (new\_cost, (new\_x, new\_y), path + [current\_position]))

258.

259.     time\_taken = time.time() - start\_time

260.     return 0, 0, time\_taken  # No path found

261.

262. def dataAStar(maze, start, end):

263.     start\_time = time.time()  # Start time measurement

264.     priority\_queue = [(0, start, [])]

265.     visited = set()

266.     nodes\_explored = 0

267.

268.     while priority\_queue:

269.         cost, current\_position, path = heapq.heappop(priority\_queue)

270.         x, y = current\_position

271.

272.         if current\_position == end:

273.             end\_time = time.time()

274.             time\_taken = (end\_time - start\_time) \* 1000  # Convert to milliseconds

275.             return nodes\_explored, len(path), time\_taken

276.

277.         if current\_position not in visited:

278.             visited.add(current\_position)

279.             nodes\_explored += 1  # Increment nodes explored

280.

281.             for direction in DIRECTION:

282.                 new\_x, new\_y = x + direction.value[0], y + direction.value[1]

283.

284.                 if 0 <= new\_x < maze.width and 0 <= new\_y < maze.height and maze.map[new\_y][new\_x] == 0:

285.                     new\_cost = cost + 1  # Uniform cost for each move

286.                     heuristic\_cost = heuristic((new\_x, new\_y), end)

287.

288.                     heapq.heappush(priority\_queue, (new\_cost + heuristic\_cost, (new\_x, new\_y), path + [current\_position]))

289.

290.     time\_taken = time.time() - start\_time

291.     return 0, 0, time\_taken  # No path found

292.

293.

### 4.3.5 authentication.py

Implementation of login and registration system. It interacts with the SQLite database to store and retrieve users’ databases.

1. import pygame, sys, sqlite3

2. from pygame.locals import \*

3.

4. pygame.init()

5. size = width,height = 645,970

6. bg = (0, 0, 0)

7. screen = pygame.display.set\_mode(size)

8.

9. # Define fonts

10. font\_large = pygame.font.Font('font/Comfortaa\_Bold.ttf', 80)

11. font\_medium = pygame.font.Font('font/Comfortaa\_Regular.ttf', 40)

12. font\_small = pygame.font.Font(None, 20)

13.

14. # Define colors

15. color\_black = (0, 0, 0)

16. color\_white = (255, 255, 255)

17. color\_grey = (192, 192, 192)

18. color\_red = (255, 0, 0)

19. color\_light\_grey = (211, 211, 211)

20.

21. username\_input\_rect = None

22. passowrd\_input\_rect = None

23. exit\_login\_rect = None

24. login\_button\_rect = None

25. username\_input\_active = False

26. password\_input\_active = False

27. username\_text = ""

28. password\_text = ""

29. margin = 2

30. top\_title\_font\_size = 36

31. user\_ps\_len = 10

32.

33. def draw\_text\_input(label, position):

34.     text\_font = pygame.font.Font(None, 24)

35.     text = text\_font.render(label, True, (color\_black))

36.     text\_rect = text.get\_rect()

37.     text\_rect.topleft = position

38.     screen.blit(text, text\_rect)

39.

40.     input\_rect = pygame.Rect(position[0] + 110, position[1], 140, 32)

41.     input\_rect.centery = text\_rect.centery

42.     draw\_rect = pygame.draw.rect(screen, (color\_black), input\_rect, 2)

43.     return draw\_rect, text\_rect

44.

45. def draw\_button(label, position, width, height):

46.     button\_rect = pygame.Rect(position[0], position[1], width, height)

47.     if label == 'Login' or label == 'Register':

48.         text\_font  = pygame.font.Font(None, 24)

49.         pygame.draw.rect(screen, (color\_grey), button\_rect)

50.     else:

51.         text\_font = pygame.font.Font(None, 24)

52.     text = text\_font.render(label, True, (color\_black))

53.     text\_rect = text.get\_rect()

54.     text\_rect.center = button\_rect.center

55.     screen.blit(text, text\_rect)

56.     return button\_rect

57.

58. # Login interface

59. def draw\_login\_interface(\*\*kwargs):

60.     screen.fill(color\_white)

61.     ps\_error\_msg, user\_error\_msg = kwargs.get('ps\_error\_msg'), kwargs.get('user\_error\_msg')

62.     is\_register = kwargs.get('is\_register', False)

63.     if is\_register:

64.         login\_label = 'Register'

65.     else:

66.         login\_label = "Login"

67.

68.     log\_msg = kwargs.get('log\_msg')

69.

70.

71.     # add title

72.     text\_font = pygame.font.Font(None, top\_title\_font\_size)

73.     text\_title = text\_font.render(login\_label, True, (color\_black))

74.     text\_title\_rect = text\_title.get\_rect()

75.     text\_title\_rect.topleft = (margin, margin)

76.     screen.blit(text\_title, text\_title\_rect)

77.

78.     global username\_input\_rect, passowrd\_input\_rect, login\_button\_rect, exit\_login\_rect

79.     # add username and password input box

80.     username\_input\_rect, username\_rect = draw\_text\_input("Username:", (width / 2 - 100, 200))

81.     passowrd\_input\_rect, ps\_rect = draw\_text\_input("Password:", (width / 2 - 100, 250))

82.

83.     # exit login ui

84.     exit\_button\_width, exit\_button\_height = 100, 40

85.     return\_main\_text = text\_font.render(u'Return Main', True, (color\_black))

86.     exit\_login\_rect = return\_main\_text.get\_rect()

87.     exit\_login\_rect.topleft = (width - exit\_login\_rect.width -margin, margin)

88.     screen.blit(return\_main\_text, exit\_login\_rect)

89.

90.     # add login button

91.     login\_button\_rect = draw\_button(login\_label, (ps\_rect.left, passowrd\_input\_rect.bottom+16),

92.                              exit\_button\_width + username\_input\_rect.width +10, exit\_button\_height)

93.

94.     text\_font = pygame.font.Font(None, 24)

95.     error\_color = (color\_red)

96.     if ps\_error\_msg is not None:

97.         text = text\_font.render(ps\_error\_msg, True, error\_color)

98.         text\_rect = text.get\_rect()

99.         text\_rect.topleft = (passowrd\_input\_rect.right+margin, passowrd\_input\_rect.top)

100.         screen.blit(text, text\_rect)

101.     if user\_error\_msg is not None:

102.         text = text\_font.render(user\_error\_msg, True, error\_color)

103.         text\_rect = text.get\_rect()

104.         text\_rect.topleft = (username\_input\_rect.right+margin, username\_input\_rect.top)

105.         screen.blit(text, text\_rect)

106.     if log\_msg is not None:

107.         text = text\_font.render(log\_msg, True, error\_color)

108.         text\_rect = text.get\_rect()

109.         text\_rect.center = (width//2, login\_button\_rect.bottom+text\_rect.height//2)

110.         screen.blit(text, text\_rect)

111.

112. # Register interface

113. def draw\_register\_interface():

114.     screen.fill(color\_white)

115.

116.     # add title

117.     text\_font = pygame.font.Font(None, 36)

118.     text\_title = text\_font.render(u'Register', True, (255, 255, 255))

119.     text\_title\_rect = text\_title.get\_rect()

120.     text\_title\_rect.center = (width / 2, 100)

121.     screen.blit(text\_title, text\_title\_rect)

122.

123.     # add username and password input box

124.     draw\_text\_input("Username:", (width / 2 - 100, 200))

125.     draw\_text\_input("Password:", (width / 2 - 100, 250))

126.

127.     # add register button

128.     draw\_button("Register", (width / 2 - 50, 320), 100, 40)

129.

130. # Login interface

131. def handle\_login\_events(\*\*kwargs):

132.     global username\_input\_active, password\_input\_active, \

133.             username\_text,password\_text, size, width, height, screen

134.     game\_main\_status = kwargs.get('game\_main\_status', 0)

135.     Gamestate\_code:list = kwargs.get('Gamestate\_code')

136.     size = width, height = kwargs.get('win\_size')

137.     is\_register = kwargs.get('is\_register', False)

138.     screen = pygame.display.set\_mode(size)

139.

140.     is\_exit = False

141.     ps\_error\_msg, user\_error\_msg = None, None

142.     log\_msg = None

143.     while not is\_exit:

144.         kwargs['ps\_error\_msg'] = ps\_error\_msg

145.         kwargs['user\_error\_msg'] = user\_error\_msg

146.         kwargs['log\_msg'] = log\_msg

147.         draw\_login\_interface(\*\*kwargs)

148.         for event in pygame.event.get():

149.             if event.type == QUIT:

150.                 pygame.quit()

151.                 sys.exit()

152.             elif event.type == MOUSEBUTTONDOWN:

153.                 if event.button == 1:

154.                     # mouse\_pos = event.pos

155.                     # handle username input box

156.                     # if 200 <= mouse\_pos[0] <= 340 and 200 <= mouse\_pos[1] <= 232:

157.                     if username\_input\_rect.collidepoint(event.pos):

158.                         username\_input\_active = True

159.                         password\_input\_active = False

160.                     # handle password input box

161.                     # elif 200 <= mouse\_pos[0] <= 340 and 250 <= mouse\_pos[1] <= 282:

162.                     elif passowrd\_input\_rect.collidepoint(event.pos):

163.                         username\_input\_active = False

164.                         password\_input\_active = True

165.                     # handle login button

166.                     elif login\_button\_rect.collidepoint(event.pos):

167.                         if is\_register:

168.                             log\_msg = register(username\_text, password\_text)

169.                         else:

170.                             log\_msg = login(username\_text, password\_text)

171.                         if log\_msg == True:

172.                             return Gamestate\_code[0], username\_text #game\_level\_interface

173.                     elif exit\_login\_rect.collidepoint(event.pos):

174.                         return game\_main\_status, None

175.

176.             elif event.type == KEYDOWN:

177.                 # handle keyboard events

178.                 if username\_input\_active or password\_input\_active:

179.                     if event.key == K\_RETURN:

180.                         # if press enter, login

181.                         if is\_register:

182.                             log\_msg = register(username\_text, password\_text)

183.                         else:

184.                             log\_msg = login(username\_text, password\_text)

185.                         if log\_msg == True:

186.                             return Gamestate\_code[0], username\_text #game\_level\_interface

187.                     elif event.key == K\_BACKSPACE:

188.                         # if press backspace, delete the last character

189.                         if username\_input\_active:

190.                             username\_text = username\_text[:-1]

191.                         elif password\_input\_active:

192.                             password\_text = password\_text[:-1]

193.                     else:

194.                         # add the character to the input box

195.                         if username\_input\_active:

196.                             username\_text += event.unicode

197.                         elif password\_input\_active:

198.                             password\_text += event.unicode

199.

200.         # display the input box

201.         # draw\_text\_input("Username: " + username\_text, (width / 2 - 100, 200))

202.         # draw\_text\_input("Password: " + "\*" \* len(password\_text), (width / 2 - 100, 250))

203.         user\_error\_msg, username\_text = draw\_input\_text(username\_input\_rect, username\_text, (width / 2 - 100, 200))

204.         ps\_error\_msg, password\_text = draw\_input\_text(passowrd\_input\_rect, "\*" \* len(password\_text), (width / 2 - 100, 250))

205.

206.         # handle login button

207.         # draw\_button("Login", (width / 2 - 50, 320), 100, 40)

208.         pygame.display.flip()

209.     # clock.tick(60)

210.

211. # draw input text and return error message and label text

212. def draw\_input\_text(rect\_ui:pygame.Rect, label, position):

213.     error\_msg = None

214.     text\_font = pygame.font.Font(None, 24)

215.     if len(label) > user\_ps\_len:

216.         label = label[:10]

217.         error\_msg = f'over {user\_ps\_len},only use previce{user\_ps\_len} '

218.     text = text\_font.render(label, True, (color\_black))

219.     text\_rect = text.get\_rect()

220.     text\_rect.center = rect\_ui.center

221.     screen.blit(text, text\_rect)

222.     draw\_rect = pygame.draw.rect(screen, (color\_black), rect\_ui, 2)

223.     return error\_msg, label

224.

225. # handle register events

226. def handle\_register\_events(\*\*kwargs):

227.     return handle\_login\_events(\*\*kwargs)

228.

229. # login logic

230. def login(username, password):

231.     # Connect to SQLite database

232.     msg = None

233.     if username is None or username == '':

234.         return 'username is empty'

235.     if password is None or password == '':

236.         return 'password is empty'

237.     conn = sqlite3.connect('user\_data.db')

238.     cursor = conn.cursor()

239.

240.     # Check if the user exists and the password is entered correctly

241.     cursor.execute('SELECT \* FROM users WHERE username=? AND password=?', (username, password))

242.     user\_data = cursor.fetchone()

243.

244.     if user\_data:

245.         msg = True

246.     else:

247.         msg = "Login failed. Invalid username or password."

248.

249.     # Commit changes and close the database connection

250.     conn.commit()

251.     conn.close()

252.     return msg

253.

254. # register logic

255. def register(username, password):

256.     msg = None

257.     if username is None or username == '':

258.         return 'username is empty'

259.     if password is None or password == '':

260.         return 'password is empty'

261.     # Connect to SQLite database

262.     conn = sqlite3.connect('user\_data.db')

263.     cursor = conn.cursor()

264.

265.     # Check if the username already exists

266.     cursor.execute('SELECT \* FROM users WHERE username=?', (username,))

267.     existing\_user = cursor.fetchone()

268.

269.     if existing\_user:

270.         msg = "Registration failed. Username already exists."

271.     else:

272.         # Insert the new user into the database

273.         cursor.execute('INSERT INTO users (username, password) VALUES (?, ?)', (username, password))

274.         msg = True

275.

276.     # Commit changes and close the database connection

277.     conn.commit()

278.     conn.close()

279.     return msg

280.

### 4.3.6 game\_level\_interface.py

Tracks and records user progress through different levels of maze challenges.

1. import pygame, sys, sqlite3

2.

3. # Define colors

4. color\_black = (0, 0, 0)

5. color\_white = (255, 255, 255)

6. color\_grey = (192, 192, 192)

7. color\_red = (255, 0, 0)

8. color\_light\_grey = (211, 211, 211)

9.

10. # Define the level object class, used to display the level button

11. class LevelObject(pygame.sprite.Sprite):

12.     def \_\_init\_\_(self, x, y, text, obj\_size:list, font\_size) -> None:

13.         super(LevelObject, self).\_\_init\_\_()

14.         super().\_\_init\_\_()

15.         self.margin = 4

16.         w, h = obj\_size

17.         w = w -self.margin

18.         h = h -self.margin

19.         obj\_size = (w, h)

20.         self.image = pygame.Surface(obj\_size)  # assume the size of the object is 50\*50

21.         self.image.fill(color\_black)  # fill the object with grey color

22.         self.rect = self.image.get\_rect()

23.

24.         self.rect.topleft = (x+self.margin, y+self.margin)

25.         self.text = str(text)

26.         self.font = pygame.font.SysFont('None', font\_size)  # use default font and size

27.         self.update\_text()

28.

29.     def update\_text(self):

30.         text = self.font.render(self.text, True, (color\_white))  # render the text

31.         self.image.blit(text, (30, 30))  # display the text, offset by 5 pixels from the top left corner

32.

33.     def collision(self, mouse\_pos):

34.         mouse\_x, mouse\_y = mouse\_pos

35.         # judge if the mouse is in the rect

36.         if self.rect.collidepoint(mouse\_x, mouse\_y):

37.             return True

38.         return False

39.

40. # Define the game level interface class, used to select the game level

41. class GameLevelInterface(object):

42.     def \_\_init\_\_(self, \*\*kwargs) -> None:

43.         self.top\_title\_font\_size = 36

44.         self.margin = 2

45.         self.win\_size = kwargs.get('win\_size')

46.         self.screen = kwargs.get('screen')

47.         self.user = kwargs.get('user')

48.         self.common\_methond = kwargs.get('common\_methond')

49.         self.return\_main\_text = kwargs.get('return\_main\_text')

50.         self.game\_status\_code = kwargs.get('game\_status\_code')

51.         self.levels\_column = 5

52.         # Generate level buttons

53.

54.         self.maze\_min\_level = 11

55.         self.maze\_max\_level = 43

56.         self.level\_interval = 2

57.         self.total\_maze\_levels = (self.maze\_max\_level - self.maze\_min\_level) // self.level\_interval + 1

58.         levels = kwargs.get('level')

59.         self.levels = []

60.

61.         for one in range(0, levels // 5):

62.             start\_level = one \* self.levels\_column + 1

63.             end\_level = start\_level + self.levels\_column

64.             self.levels.append(list(range(start\_level, end\_level)))

65.

66.         # Handle the remaining levels, if any

67.         remaining\_levels = levels % 5

68.         if remaining\_levels > 0:

69.             start\_level = levels - remaining\_levels + 1

70.             self.levels.append(list(range(start\_level, levels + 1)))

71.

72.         self.levels\_objs = []

73.         self.all\_sprites = pygame.sprite.Group()  # create a sprite group to manage all sprites

74.         self.first = True

75.

76.     # When a level is selected, map it to the corresponding maze level

77.     def get\_maze\_level(self, selection\_level):

78.         maze\_level = self.maze\_min\_level + (selection\_level - 1) \* self.level\_interval

79.         return min(maze\_level, self.maze\_max\_level)

80.

81.     # Query the current level of the user

82.     def query\_db\_level(self):

83.         with sqlite3.connect('user\_data.db') as conn:

84.             cursor = conn.cursor()

85.             cursor.execute('SELECT level FROM users WHERE username=?', (self.user,))

86.             level = cursor.fetchone()

87.             return level[0]

88.

89.     # Update the level of the user in the database, if the current level is higher than the previous level of the user, update it

90.     def deal(self):

91.         while True:

92.             self.draw()

93.             for event in pygame.event.get():

94.                 if event.type == pygame.QUIT:

95.                     pygame.quit()

96.                     sys.exit()

97.                 elif event.type == pygame.MOUSEBUTTONDOWN:

98.                     for level\_obj in self.levels\_objs:

99.                         if level\_obj.collision(event.pos):

100.                             selection\_level = int(level\_obj.text)

101.                             maze\_level = self.get\_maze\_level(selection\_level)

102.                             return [self.game\_status\_code[1], maze\_level]

103.

104.             self.all\_sprites.update()

105.             self.all\_sprites.draw(self.screen)

106.             pygame.display.flip()

107.

108.     # Draw the game level interface, including the title, the level buttons, and the return button

109.     def draw(self):

110.         self.screen.fill(color\_white)

111.

112.         select\_game\_level\_text\_font = pygame.font.Font(None, self.top\_title\_font\_size)

113.         select\_game\_level\_text = select\_game\_level\_text\_font.render('Select Game Level', True, (color\_black))

114.         text\_title\_rect = select\_game\_level\_text.get\_rect()

115.         text\_title\_rect.topleft = (self.margin, self.margin)

116.         self.screen.blit(select\_game\_level\_text, text\_title\_rect)

117.

118.         if self.first:

119.             level\_text\_font\_size = self.top\_title\_font\_size\*2

120.             obj\_topleft\_x = self.margin\*16

121.             obj\_size = [(self.win\_size[0] - obj\_topleft\_x\*2)//self.levels\_column,

122.                         (self.win\_size[0] - obj\_topleft\_x\*2)//self.levels\_column]

123.

124.             for outter\_index, i in enumerate(self.levels):

125.                 for index, j in enumerate(i):

126.                     self.levels\_objs.append(LevelObject(obj\_topleft\_x + index \*obj\_size[0],

127.                                                 text\_title\_rect.bottom+self.margin + outter\_index\*obj\_size[0],

128.                                                 j,

129.                                                 obj\_size,

130.                                                 level\_text\_font\_size)

131.                                     )

132.             self.all\_sprites.add(self.levels\_objs)

133.             self.first = False

134.

135.

# 5. Evaluation

## 5.1 Evaluation against Objectives

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Objective | | Success Criteria | |  | Sub-success Criteria | | | Evaluation |
| **1.** | Intuitive and Attractive User Interface | a | A minimalist-style main page, including three buttons, play, login and registration. | **√** | i | Add BGM to the background. | **√** | From an aesthetic design perspective, the minimalist design of the UI enhances the user experience and avoids unnecessary visual fatigue, allowing users to focus better on the functionality of the simulator itself. After at least three comprehensive beautifications, I deleted unnecessary background images (which can also improve the loading speed of the program), added high-contrast button backgrounds to most buttons (instead of pure text), and selected in the maze. The interface offers high-contrast buttons. From a technical perspective, I also added many dynamic UI elements, such as game level selection that updates as the user progresses through the game; the congratulations message is not just a boring, static message but a random selection from a list. The user's login status will be displayed at any time, etc.  Future revisions to the UI may include user customisation options for interface themes, such as adding background music volume adjustments, changing themes, etc.  \* For iv), my original plan was to add a dynamic typing effect to the main page's title and a typing audio. Because I have implemented this function in a game before, but in actual testing, I found that this typing effect is unnecessary. |
| b | A clear login and register interface, including a title (indicating registration or login), a return to the main menu button, username and password input boxes and a submit button | **√** |
| c | When the user is logged in, a maze-level selection page will appear when entering the game. The buttons for the maze level will be updated as the player progresses in unlocking the levels. This page should also include a return button. The user can continue the game after clicking this button. | **√** | ii | Add sound effects when clicking buttons. | **√** |
| d | Streamlined controls for algorithm selection, maze level selection, and result observation. | **√** | iii | Add the version number to the title of the main menu. | **√** |
| iv | Add sound effects when the main menu's title animation is triggered. | × |
|  |  | e | A table comparing the efficiency of different pathfinding algorithms should be visible below the maze and the Select Pathfinding Algorithm buttons. | **√** | v | When the status is the congratulations menu, add a crowd cheer sound. | **√** |
| f | Add a congratulations menu. This page should pop up when the user manually controls the character to reach the target point or presses the skip button. This page should include a congratulations, a next-level button, and a return button. | **√** |
|  |  | g | When the user clicks Return in the game after logging in, a second menu should appear, including the functions of logging out and continuing the game. | **√** |  |  |  |
| **2.** | Dynamic Maze Generation Using Randomised Prim’s Algorithm | a | Implement the Randomised Prim’s Algorithm to generate mazes of varying complexity and sizes, ensuring no two mazes are identical. | **√** |  |  |  | After many tests, the randomised Prim algorithm has indeed achieved the characteristics of generating a perfect maze. Not only complex but also non-repetitive mazes can be created with this algorithm. In order to allow the data of the pathfinding algorithm to be pre-loaded, I also added a delay in the visualisation of the maze generation. The user can also understand the steps of randomised Prim algorithm generation. In addition, the performance of the algorithm is optimised for minimal computational overhead, even if It can also ensure the rapid generation of complex mazes on low-configuration devices.  In the future, various maze generation algorithms may be added, allowing users to choose different maze generation algorithms for comparison just like they choose pathfinding algorithms. |
| b | The generated map is a perfect maze (a.k.a. standard maze). It refers to a maze with no loops or unreachable positions. It is making sure that there is a path between the start point and the target point. | **√** |
| c | Allow users to select different complexity levels for maze generation. (Achieve different complexities by changing the maze size) | **√** |  |  |  |
| d | Generate a target point in the lower right corner of the maze, and walls should not all surround this target point. | **√** |
| e | Visualisation of maze generation steps | **√** |
| f | Add a skip button. If the player is unwilling to manually control the character to reach the end and enter the next level, they can press this button to skip it and go directly to the congratulations page. | **√** |
| g | Add a previous level button so that if the user clicks on it, the maze size should be regenerated to the size of the previous level. This allows players to compare whether the efficiency of different pathfinding algorithms changes with the size of the maze. | **√** |
| h | Add a return button that will return the user to the main menu if they click it. | **√** |
| **3.** | Manual Exploration Mode in Mazes | a | The player's initial position in the maze should be in the upper left corner, which walls should not all surround. | **√** | i | A sound effect will be triggered every time the player enters the maze. | **√** | At the request of the interviewees, I also added a manual exploration mode, which proved to be a very correct determination. I used event-driven programming to solve the user control maze logic representation integration, ensuring the character could move within the specified range and collision detection would work.  There is a slight regret here. Initially, I planned to randomly generate the starting and target points in the maze. Still, I finally changed them for various technical reasons to develop them in the maze's upper left and lower right corners. This can be optimised in the future to improve randomness.  \* For iii), I made a random sound effect according to the background music method, but the effect was terrible and was not harmonious with the background music, so I discarded it. |
| b | Players can use the small keyboard or “WASD” to control character movement in a maze. | **√** | ii | A sound effect is made every time the player steps into the maze. | **√** |
| c | The congratulations menu will be displayed if the user reaches the target point. | **√** | iii | Every time the player takes a step, the sound effect should be randomly selected from a list of sound effects instead of having just one sound effect. | × |
| iv | A sound effect will be triggered if the player reaches the target point. | **√** |
| **4.** | Enable Process Visualisation of Pathfinding Algorithms | a | Clear visualisation of how the algorithm navigates around the maze walls to reach the target point | **√** |  |  |  | The visual algorithm is implemented by mapping algorithm decisions to the screen through Pygame's graphics rendering functions. This function helps me debug and achieve educational significance in this project. But because the delay I added is fixed, as the complexity of the maze increases, the visualisation time may be too long, so the function of user-defined delay time may be added in the future. |
| b | Implement visual cues to indicate the progression of algorithms through the maze. | **√** |
| c | Ensure algorithms can be run in a single environment to control variables. | **√** |
| **5.** | Implementation of Diverse Pathfinding Algorithms | a | Depth-first Search | **√** |  |  |  | In this project, I implemented a total of 4 different pathfinding algorithms to provide users with diverse choices. In the future, I may add more pathfinding algorithms, such as Jump Point Search and other more advanced algorithms. |
| b | Breadth-first Search | **√** |
| c | Dijkstra’s Algorithm | **√** |
| d | A\* Search | **√** |
| e | Ensure algorithms can solve mazes generated by the system effectively. | **√** |
| **6.** | Performance Analysis and Comparative Data Display | a | Develop a system to compare the performance of different pathfinding algorithms on the same maze. | **√** |  |  |  | Algorithm analysis tools specially developed for programmers use statistical methods to present the algorithm performance of different indicators. This helps users understand the pros and cons of different algorithms in varying complex environments.  More indicators may be added in the future, such as the Big O algorithm, etc. |
| b | Metrics include path length, computation time, and efficiency. | **√** |
| **7.** | Implement Login and Register System | a | User account creation, login system, and maze-solving history tracking for personalised user experiences. | **√** | i | Develop a secure user authentication and registration system using appropriate hashing and password encryption. | **√** | In the project's current version, the only purpose of introducing the login registration system is to track the progress of players unlocking the maze. The database is relatively simple, and no second database is connected with the current one. However, I still considered security issues in the database design because some users may use their personal information as login names and passwords, so I appropriately encrypted the fields filled in by users. However, in the future, An achievement system and a pathfinding algorithm knowledge database will be added to enhance the educational significance of this simulator. |
| **8.** | Implement Maze Level Selection Menu |  | A user-friendly interface that allows a selection of maze complexities. | **√** |  |  |  | The game level interface is also a significant difficulty in this project because this level interface is dynamic. The interface will continuously update the unlocked levels as the user completes the level. After constant attempts, this feature was finally implemented. As long as the user is logged in, their game progress will be saved, and its final interface will be user-friendly, using high-contrast buttons to provide users with different maze-size options. , users can access different game levels by pressing these buttons. An achievement system may be added below this interface, similar to this maze-level system. As users continue to experience the simulator, some achievements, such as completing all levels of mazes, will be displayed on the achievement page. |

## 5.2 Data Collection Conclusion

The data and charts in [Section 3.4](#_3.4_Testing_Search) show that as the maze complexity increases, the performance differences of different algorithms become more apparent. This emphasises the critical importance of considering the problem's complexity and the algorithm's space-time efficiency when selecting an algorithm suitable for a specific situation (such as maze solving).

In practical applications, the choice of algorithm should be based on specific usage scenarios. For example, DFS may be an efficient choice in small or medium-complexity mazes, while in giant or high-complexity mazes, the A\* algorithm may be superior.

For each algorithm, analysing its performance at different maze levels can reveal opportunities for performance optimisation. For example, the DFS algorithm can be optimised to reduce path backtracking in giant mazes, or the heuristic function of the A\* algorithm can be improved to improve its efficiency further on all maze levels.

# Appendix

## I. References

Maze generation algorithm. (2024, January 3). In *Wikipedia*. <https://en.wikipedia.org/wiki/Maze_generation_algorithm>

Pathfinding. (2023, November 13). In *Wikipedia*. <https://en.wikipedia.org/wiki/Pathfinding>

Pygame Front Page — pygame v2.6.0 documentation. (n.d.). <https://www.pygame.org/docs/>

Matplotlib. (n.d.). Matplotlib: Python plotting — Matplotlib 3.3.4 documentation. Matplotlib.org. <https://matplotlib.org/stable/index.html>

PathFinder: The Amazing Maze Algorithm Demonstrator! (n.d.). Emmilco.github.io. <https://emmilco.github.io/path_finder/>

## II. Image Source

Figure [1,2,3]: Screenshot from <https://emmilco.github.io/path_finder/>

Figure [5]: Randomly generated from <https://angeluriot.com/maze_solver/>

Figure [10, 11]: Screenshot from <https://www.interviewcake.com/concept/python3/stack?course=fc1&section=queues-stacks>