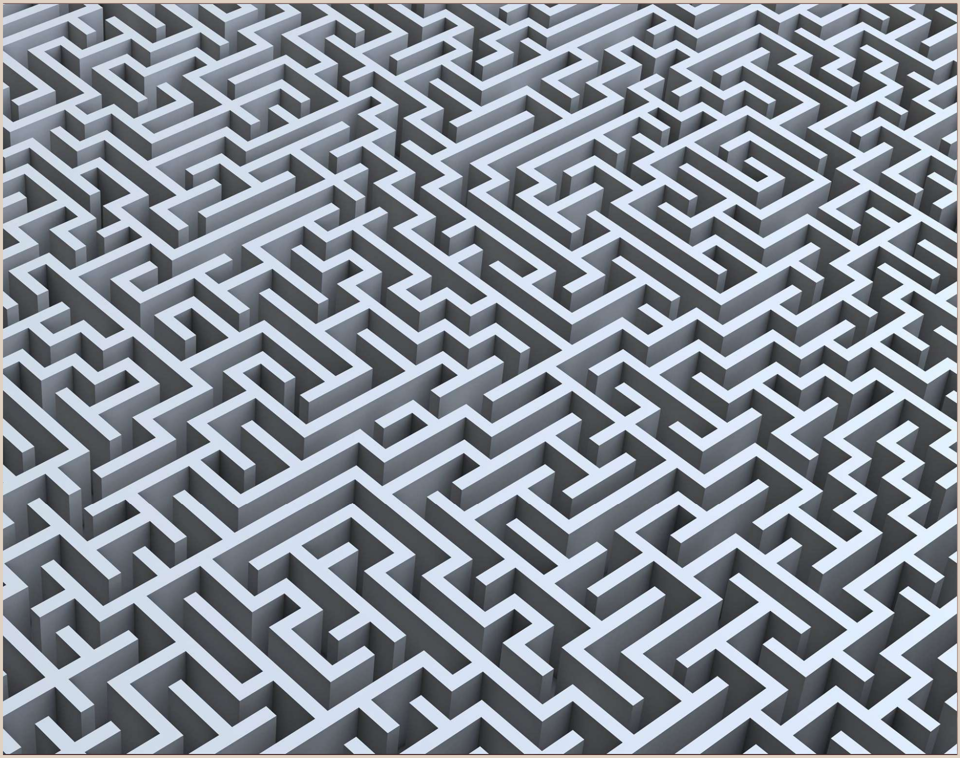
Maze Solver Project using BFS Algorithm

Introduction and Project Overview

Title and Team Information



Project Overview

The project focuses on solving mazes using BFS to find the shortest path efficiently within grid-based structures.

Team Members

Developed by Tarek Mostafa Kamel (2302162)

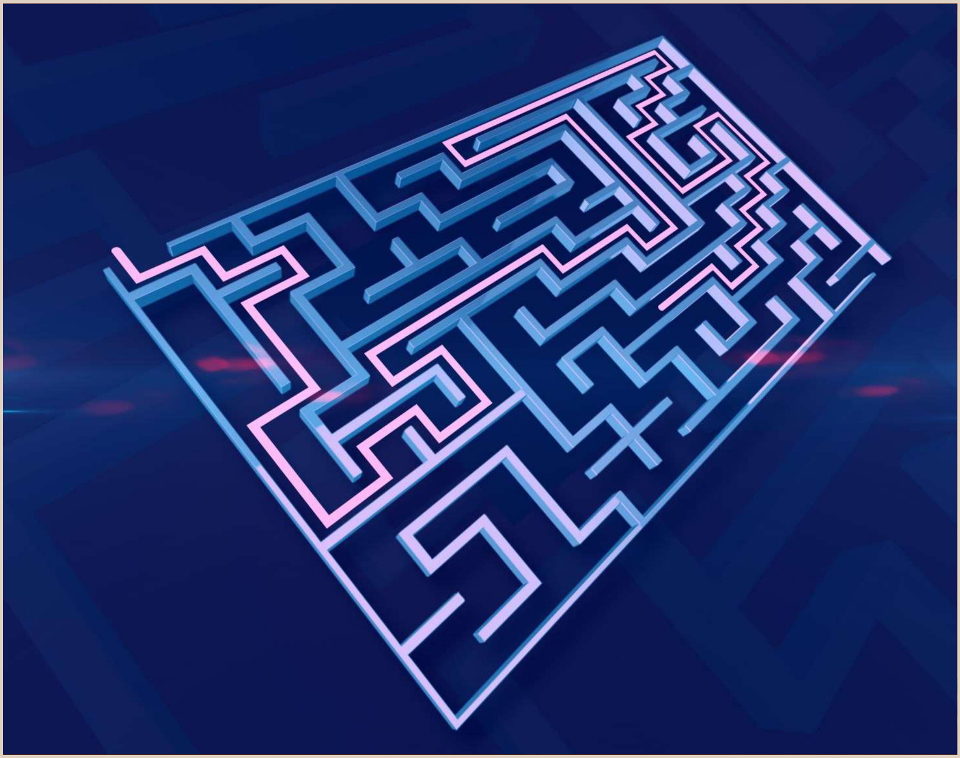
and Ibraheem Mohammed Abd El-Twab (2301915)

Breadth-First Search algorithm guarantees finding the shortest path in an unweighted maze environment.

Presentation Scope

Includes project overview, algorithm details, system design, execution, testing, and optional GUI enhancements.

Project Objective and Scope



Objective of Maze Solver

The project aims to efficiently solve mazes using AI search algorithms like BFS, highlighting pathfinding techniques.

Project Scope

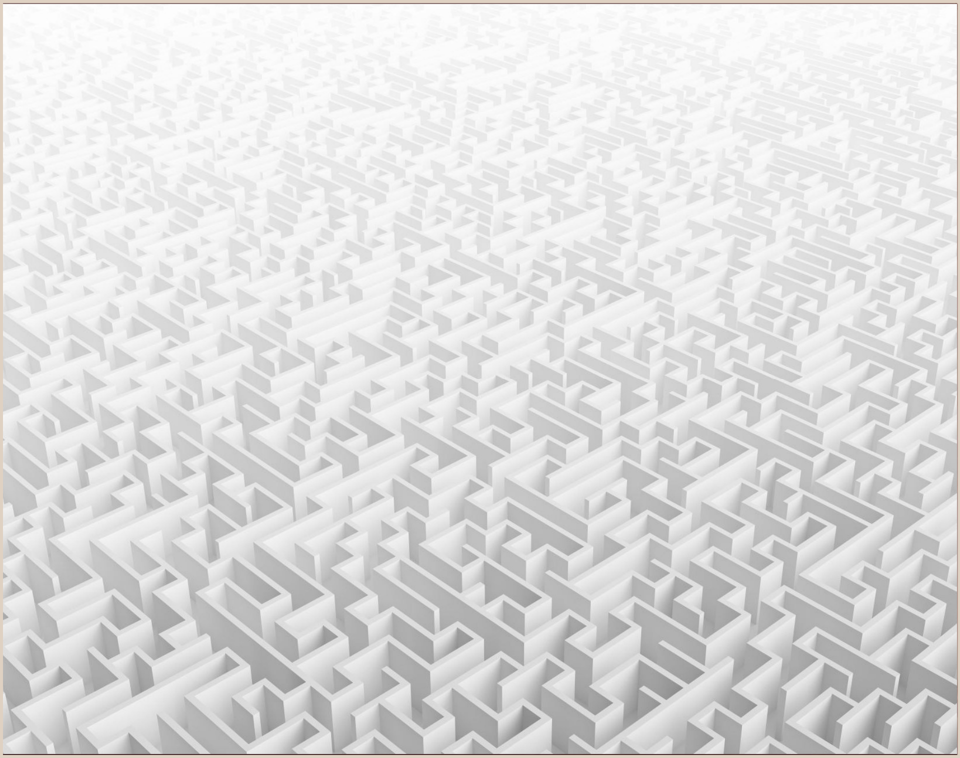
Scope includes reading maze input, identifying walls and paths, and applying BFS to find shortest routes.

Practical Implementation

Implementation uses Python focusing on clarity, functionality, and optional GUI for user interaction.

Algorithm and System Design

Breadth-First Search (BFS) Algorithm



Core BFS Concept

BFS explores nodes level by level, ensuring the shortest path in unweighted graphs or grids.

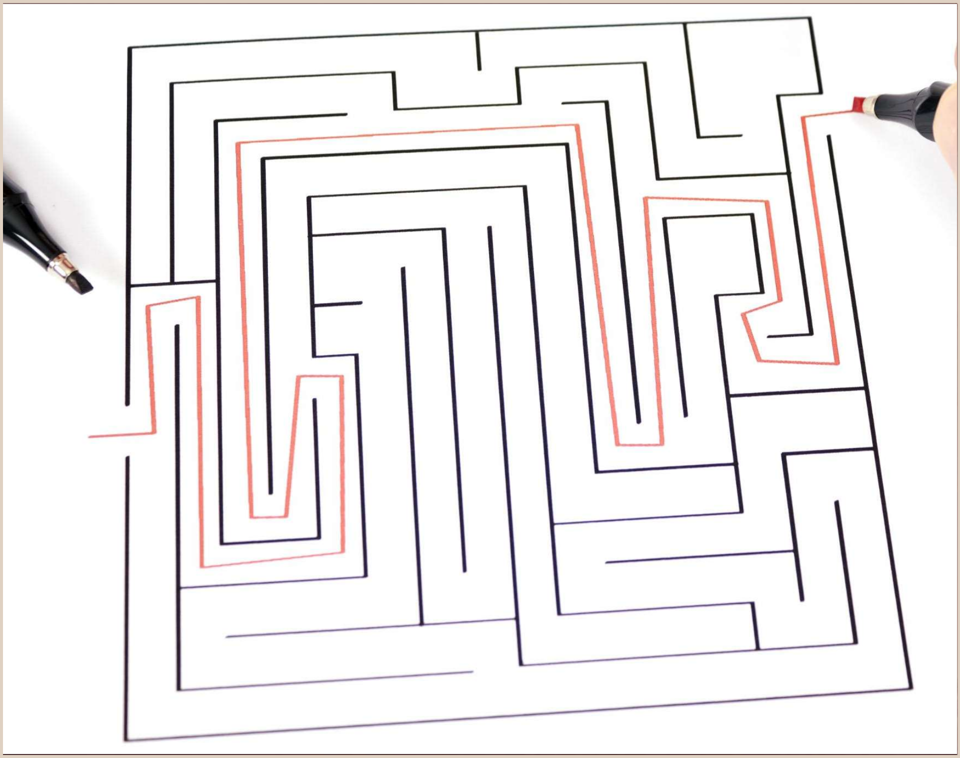
Maze Solving Application

BFS starts at the maze's start point and visits neighbors systematically to find shortest route.

Queue Data Structure

A queue manages the exploration order, enabling BFS to visit nodes in correct sequence.

System Design Overview



Input Representation

The maze is represented by a text file using symbols for walls, paths, start, and end points.

Algorithm Processing

The system uses BFS to find the shortest path by constructing an internal maze representation.

Output Display

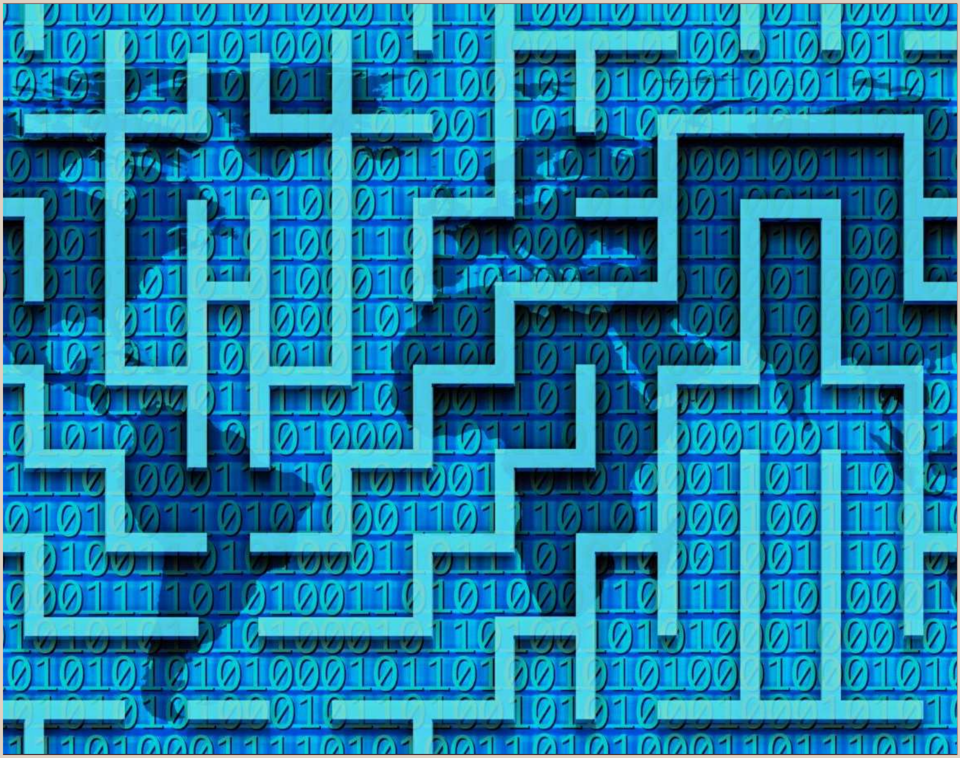
Solved maze or step sequence is displayed in the console with the path clearly marked.

Modular Design

Separate functions handle input reading, BFS processing, and output display for clarity and extensibility.

Execution and Demonstration

How to Run the Project



Project Requirements

The Maze Solver project requires Python 3.8 or newer and essential project files like maze_solver.py and maze.txt.

Execution Instructions

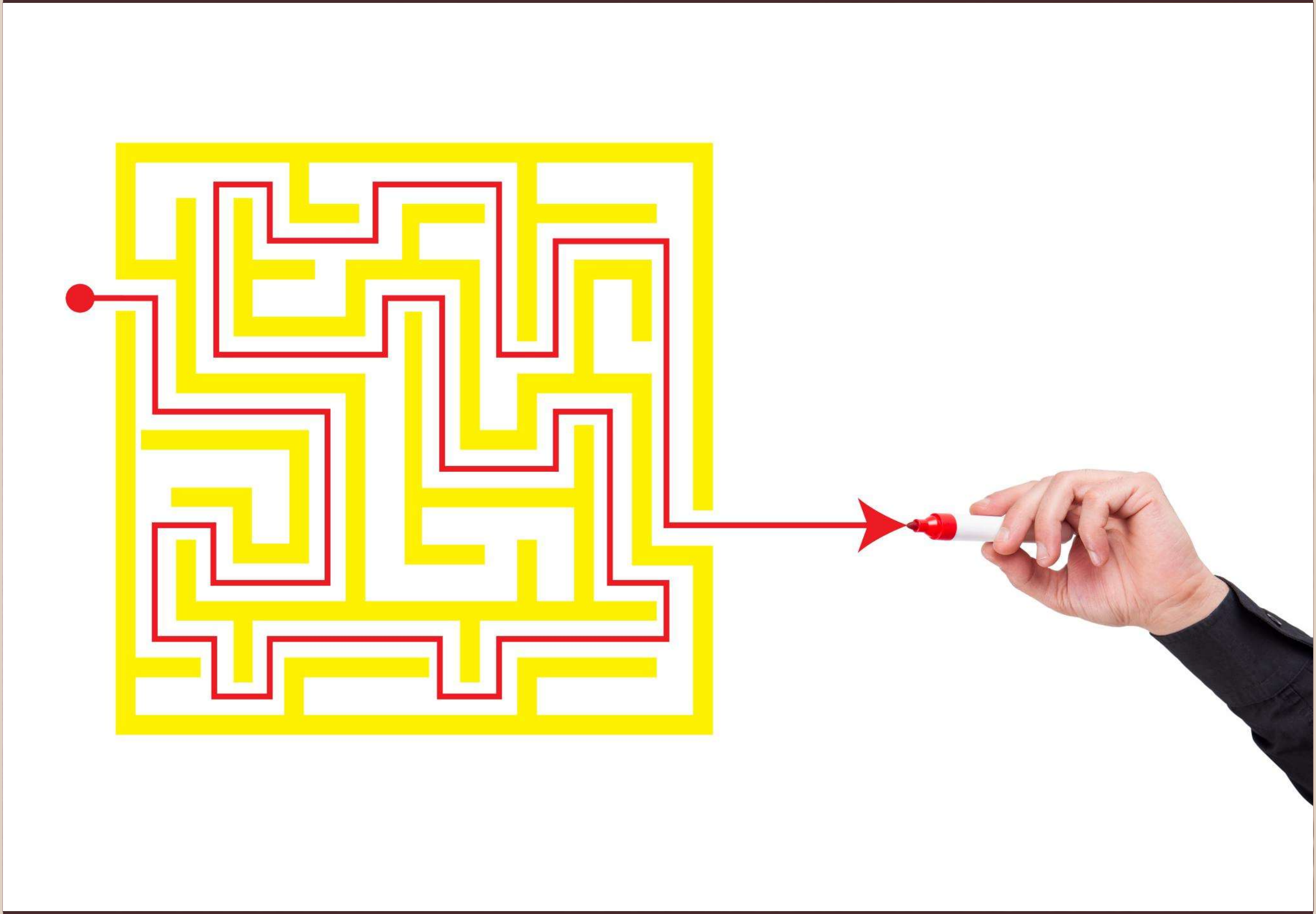
Run the solver using terminal commands specifying the maze file or default configuration for easy execution.

Maze File Format

Maze files use '#' for walls, spaces for paths, and optional 'S' and 'E' to mark start and end points.

Output Display

The program outputs the solved maze or path found, demonstrating the solution clearly in the terminal.



Input and Output Samples

Text File Input Format

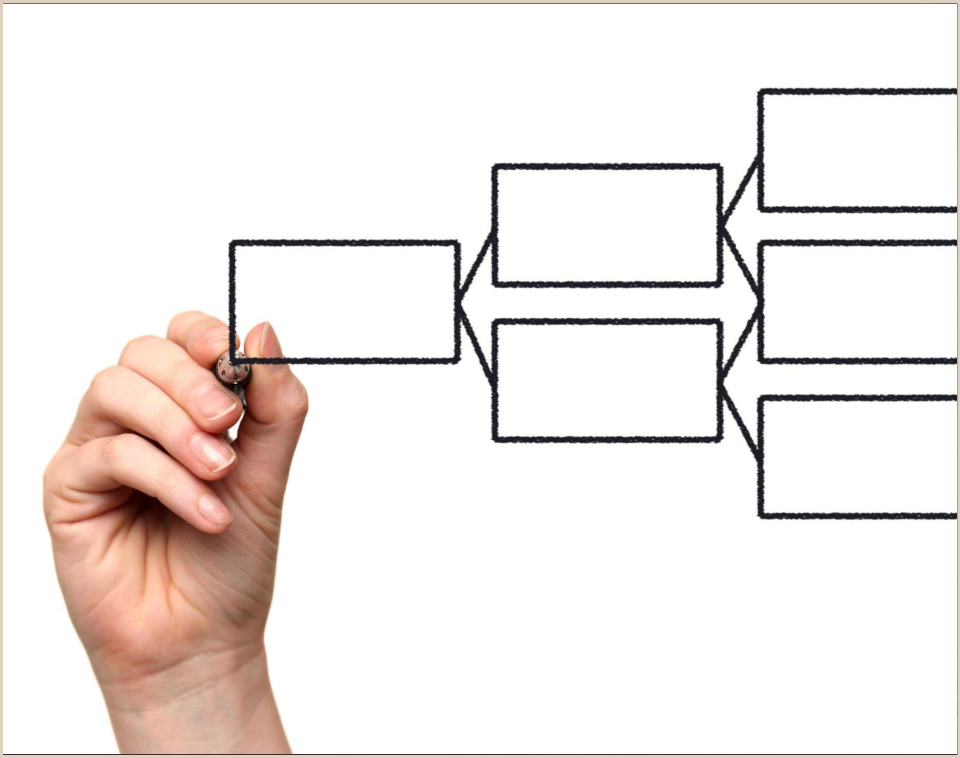
Maze input is a plain text file with '#' for walls and spaces for paths, including optional start 'S' and end 'E'.

Console Output Display

Output shows the maze with the solved path marked, helping users verify the solution visually.

Conclusion and Team Roles

Summary and Key Takeaways



AI Search Algorithm Application

The project showcases the use of BFS AI algorithm to solve real-world maze problems effectively and efficiently.

System Design and Modularity

Clear, modular system design makes the solution easy to understand, maintain, and extend for future enhancements.

Simple Execution and Reliability

The project runs on Python , confirming reliable and efficient maze-solving results with straightforward execution.

Enhancements and Future Exploration

Optional GUI improvements demonstrate usability focus and lay groundwork for exploring advanced AI concepts later.

Team Contributions

Diverse Team Roles

Team members contributed to algorithm implementation, system design, testing, and documentation.

Effective Collaboration

Clear communication and task division enabled efficient teamwork and timely project delivery.

Acknowledging Contributions

Recognizing individual efforts highlights teamwork importance and academic integrity.