Introduction

This report details the design and implementation of a maze-solving algorithm developed by our group. The objective of this project was to create a program capable of navigating through complex mazes autonomously.

Key Idea

The core idea of our maze solver is to employ an algorithm which is able to navigate from the maze's entrance to its exit.

Challenges of Maze Solving

Complexity of Maze Structures

Mazes can be intricate with numerous paths such as loops and dead-ends. The solver needs to backtrack effectively upon encountering dead-ends or loops to explore alternate pathways.

Pathfinding in Unknown Environments

The solver must be able to navigate without prior knowledge of the maze's layout.

Optimal Route Selection

Finding the shortest, most efficient path adds complexity to the project.

Utilities of Maze Solving

Robotics and Autonomous Systems

Autonomous robots often need to navigate complex environments without human intervention, therefore efficient maze-solving algorithms can be adapted for robot pathfinding applications

Game Design and Development

Many video games incorporate maze-like environments. Developing robust maze-solving algorithms can enhance the realism and complexity of game AI, resulting in more engaging gameplay

Initial Goals to Achieved Results

Our initial goals for the maze solver were to develop an algorithm capable of navigating through complex mazes autonomously and to implement various search algorithms to compare their effectiveness. The following section shows how well we met these goals:

Meeting Initial Goals:

Algorithm Development: We successfully implemented several search algorithms: Depth First Search, Breadth First Search, Uniform Cost Search and A* Search within our maze solver

Maze Navigation: The solver can effectively navigate through different types of mazes, finding paths from the entrance to the exit using the chosen algorithms

Comparative Analysis: We conducted comprehensive tests and evaluations to compare the performance of these algorithms in terms of pathfinding efficiency

Extent of Goal Achievement

We largely achieved our goal of developing and implementing a maze solver with multiple search algorithms. Furthermore our analysis provided valuable insights into the strengths and weaknesses of each algorithm in maze-solving scenarios.

Unexpected Findings

The impact of heuristic functions on informed search algorithms (Greedy Search, A* Search) was a lot more notable than expected, influencing the optimality of the solutions.

Strengths and Weaknesses of each Algorithm

Depth First Search

Strengths:

- Memory-efficient for deep maze exploration
- Can reliably find any path

Weaknesses:

- Not optimal for finding the shortest path
- Can get stuck in deep paths

Breadth First Search

Strengths:

- Guarantees finding the shortest path in terms of steps

Weaknesses:

- Requires more memory due to node storage
- Slow for large mazes due to memory overhead

Uniform Cost Search

Strengths:

- Finds the optimal path based on path cost
- Suitable for mazes with varying path costs

Weaknesses:

- Complexity increases with higher costs and complex structures
- Performance affected by maze size and cost distribution

Greedy Search

Strengths:

- Fast and efficient for simper mazes
- Prioritizes nodes based on heuristic estimates

Weaknesses:

- May not guarantee the optimal solution
- Can overlook potentially better paths based on local heuristic information

A* Search

Strengths:

- Guarantees optimal path solution with suitable heuristic
- Balances path cost and heuristic estimates effectively

Weaknesses:

- Requires a consistent and admissible heuristic
- Computationally intensive compared to uninformed search algorithms

Our Solution

Overview: We developed an AI Maze Solver using two types of search strategies: uninformed and informed. These strategies include:

1. **Uninformed Search Strategies:**
    - **Breadth-First Search (BFS):** Explores all nodes at the current level before moving on to the next.
    - **Depth-First Search (DFS):** Explores nodes deeply before backtracking to cover unexplored paths.
    - **Uniform-Cost Search** (**UCS**) Explores nodes that find the lowest-cost path between a starting point and a goal in a weighted graph.
2. **Informed Search Strategies:**
    - **A***: Utilizes a heuristic function to estimate the cost of reaching the goal, thus choosing paths that are more likely to succeed. heuristic function + path cost
    - **Greedy**: Explores nodes that have shortest distance based on the heuristic function straight line test.

Development Steps:

1. **Maze Representation:**
    - I represented the maze as a 2D grid where cells are either open (passable) or walls (impassable).
    - Start and goal cells are marked clearly.
2. **Graph Construction:**
    - Each open cell was treated as a graph node, with edges connecting it to its neighboring open cells.
    - The graph was used to facilitate search algorithms.
3. **Algorithm Implementation:**
    - BFS ,DFS, UCS were implemented as basic graph traversal algorithms, which identify all valid paths to the goal.
    - A* and Greedy relied on a heuristic function, such as the Manhattan distance or, to prioritize paths that reduce the estimated remaining distance to the goal.
4. **Matplotlib Visualization:**
    - Matplotlib was used to visually represent the maze and the solution paths.
    - Different colors distinguish between explored and unexplored paths.
    - Different color for path to distinguish between uninformed and informed

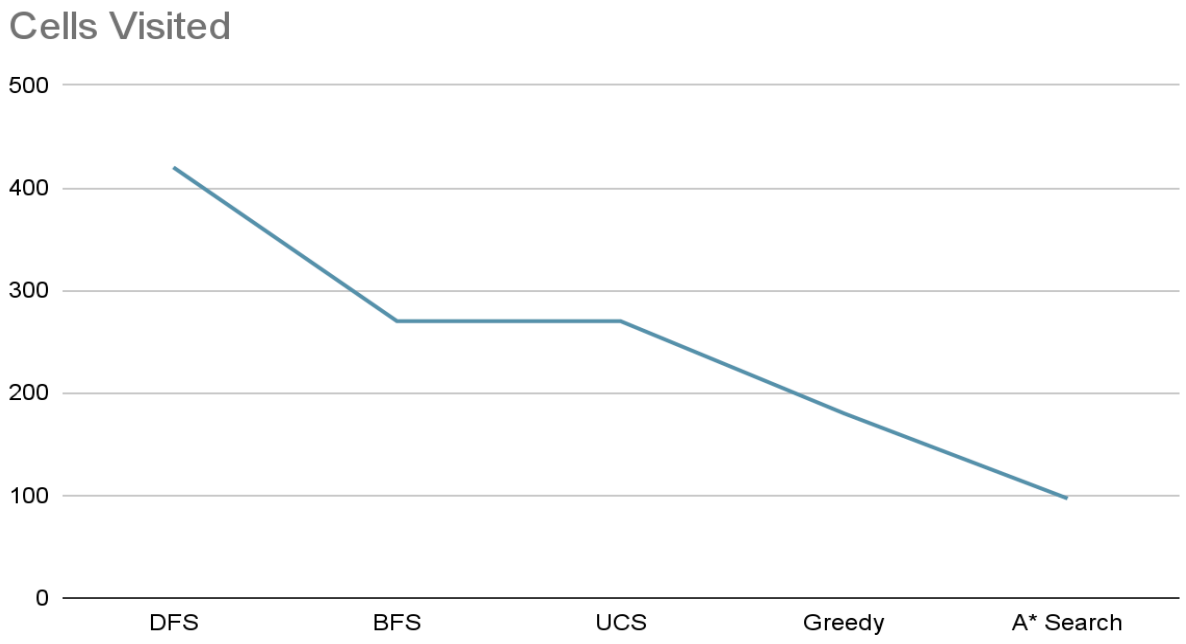
Use of Pretrained Models/Libraries:

- **Matplotlib:** Chosen for its simplicity and power in displaying 2D graphics, it allowed me to visualize both the maze and solution paths effectively.
- **Heap Data Structure:** For UCS, Greddy, and A* since exploring nodes with lowest cost is essential, popping from a heap was very ideal.

Fine-Tuning and Adaptations:

- We modified the heuristic function for the A* algorithm to adjust to various maze complexities, allowing more accurate predictions in complex maze structures.

Results



Graph displaying the number of cells each algorithm visited before reaching the goal

Contributions

Muse Degu: Informed Search Algorithms (Creating Maze Logistics, Greedy Search, A* Search)
Tristan Yeo: Uninformed Search Algorithms (DFS, BFS, UCS)