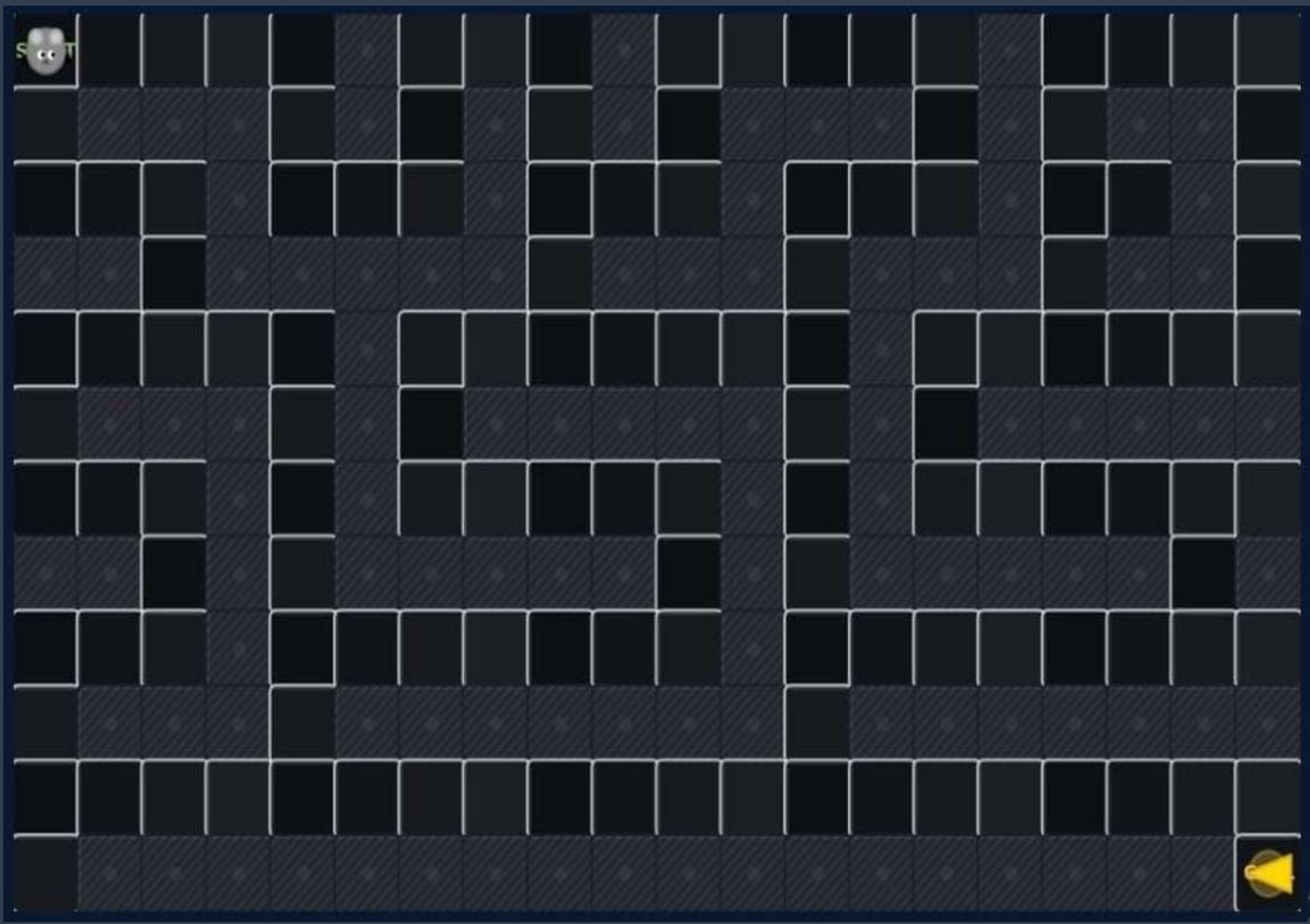
Maze Pathfinding Proposal



Team members

Name	Section
أسماء أحمد السيد محمد بهنسي عيسى	C1
أميرة خميس هنداوى خميس ابوسويد	C1
اسراء فوزي السيد عبدالقادر	C1
أسماء زكريا عمر محمد الخياط	C1

1. Introduction

In this project, We implemented and compared five important search algorithms to solve maze pathfinding problems. The goal is to find a path from a starting point to a goal while avoiding walls and obstacles. We chose this problem because it is visual and easy to understand, and it helps show how different search algorithms explore a space and find solutions. The maze also allows us to compare speed, memory usage, and path quality for each algorithm.

2. Problem Formulation

State Space:

The maze is a 20×12 grid. Each cell represents a position in the maze:
- All positions: (x, y) where $0 \le x < 20$, $0 \le y < 12$
- Total positions: 240
- Valid positions: only cells without walls (value 0)

Start State: (0, 0) – top-left corner of the maze

Goal State: (19, 11) – bottom-right corner of the maze

Actions:

From any position, the agent can move:
1. Up → (x, y-1)
2. Down → (x, y+1)
3. Left → (x-1, y)
4. Right → (x+1, y)

Constraints:

- Can only move to adjacent cells
- Cannot move into walls
- Cannot move outside the maze
- Each move has a uniform cost of 1

Cost Function:

Every move costs 1, so path length = total cost.

3. Algorithms used in the project

A. Uninformed (Blind) Search Algorithms

These algorithms do not use any information about the goal location. They explore the maze without guidance.

- Breadth-First Search (BFS)
- Depth-First Search (DFS)
- Iterative Deepening Search (IDS)
- Uniform Cost Search (UCS)

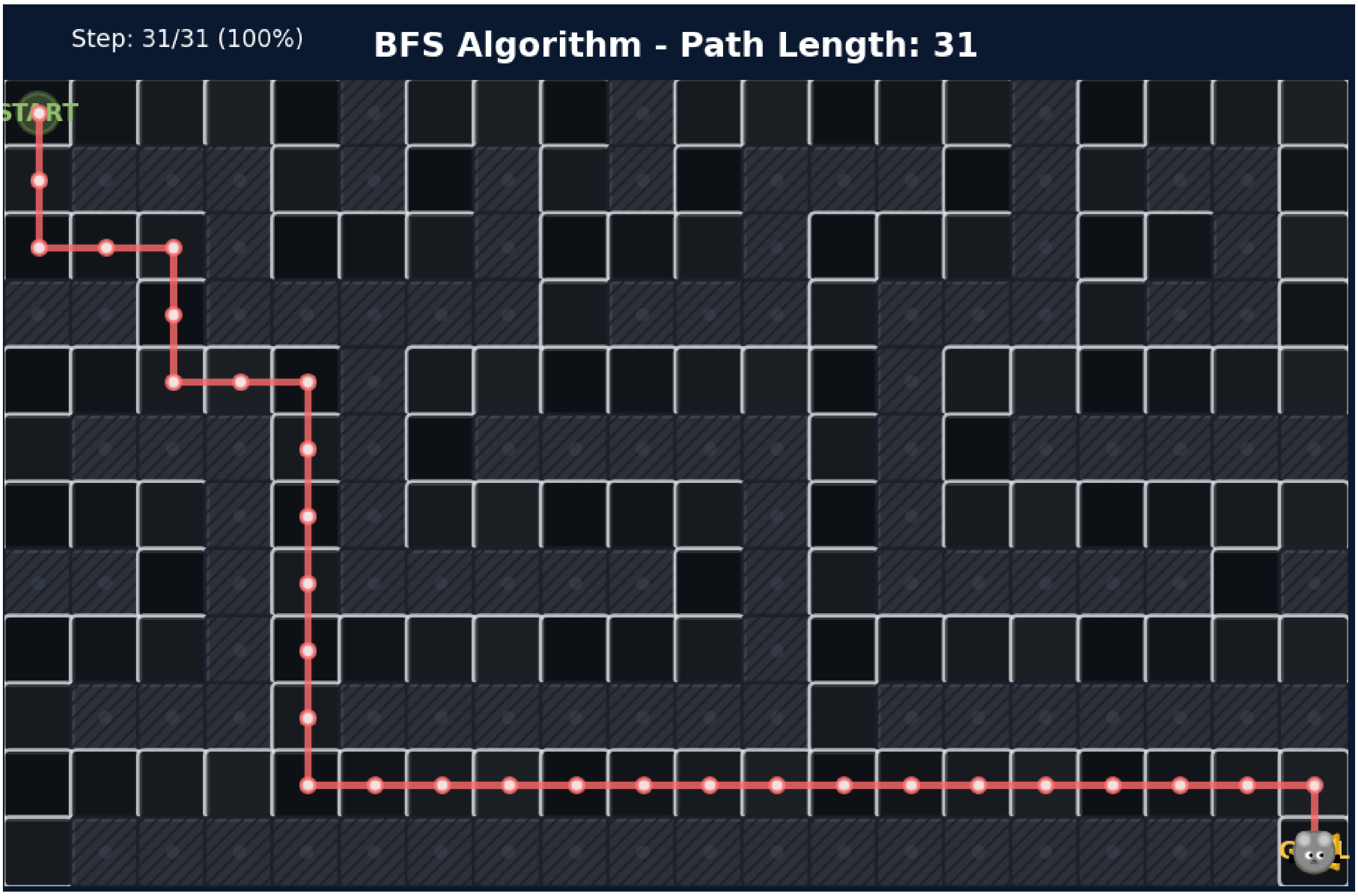
B. Informed (Heuristic) Search Algorithm

These algorithms use additional information (heuristics) to guide the search toward the goal efficiently.

- A* Search algorithm

3. Algorithms Implementation

BFS (Breadth-First Search):



BFS explores all nodes at the current depth before moving deeper. It uses a queue to keep track of nodes to explore.

How it was used in the project:

•Start with the starting position in the queue

•Remove nodes from the front of the queue

•Check if the node is the goal

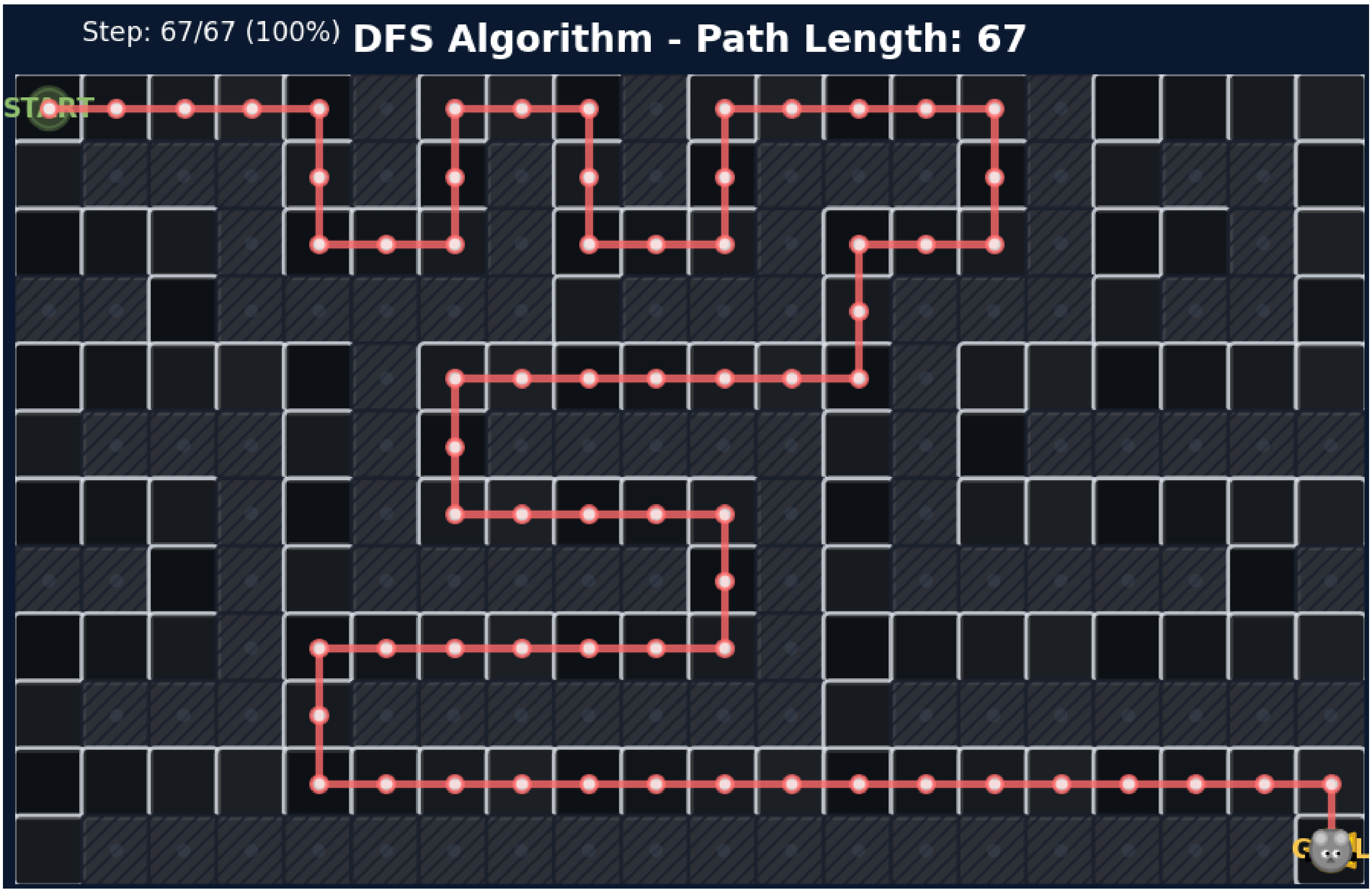
•Add all valid neighbors to the queue

•Keep track of parents to reconstruct the path

Why it works for the maze:

- Always finds the shortest path in a uniform-cost maze
- Complete (will find a solution if it exists)
- Simple to implement

DFS (Depth-First Search):



DFS explores one path as far as possible before backtracking. It uses a stack to manage nodes.

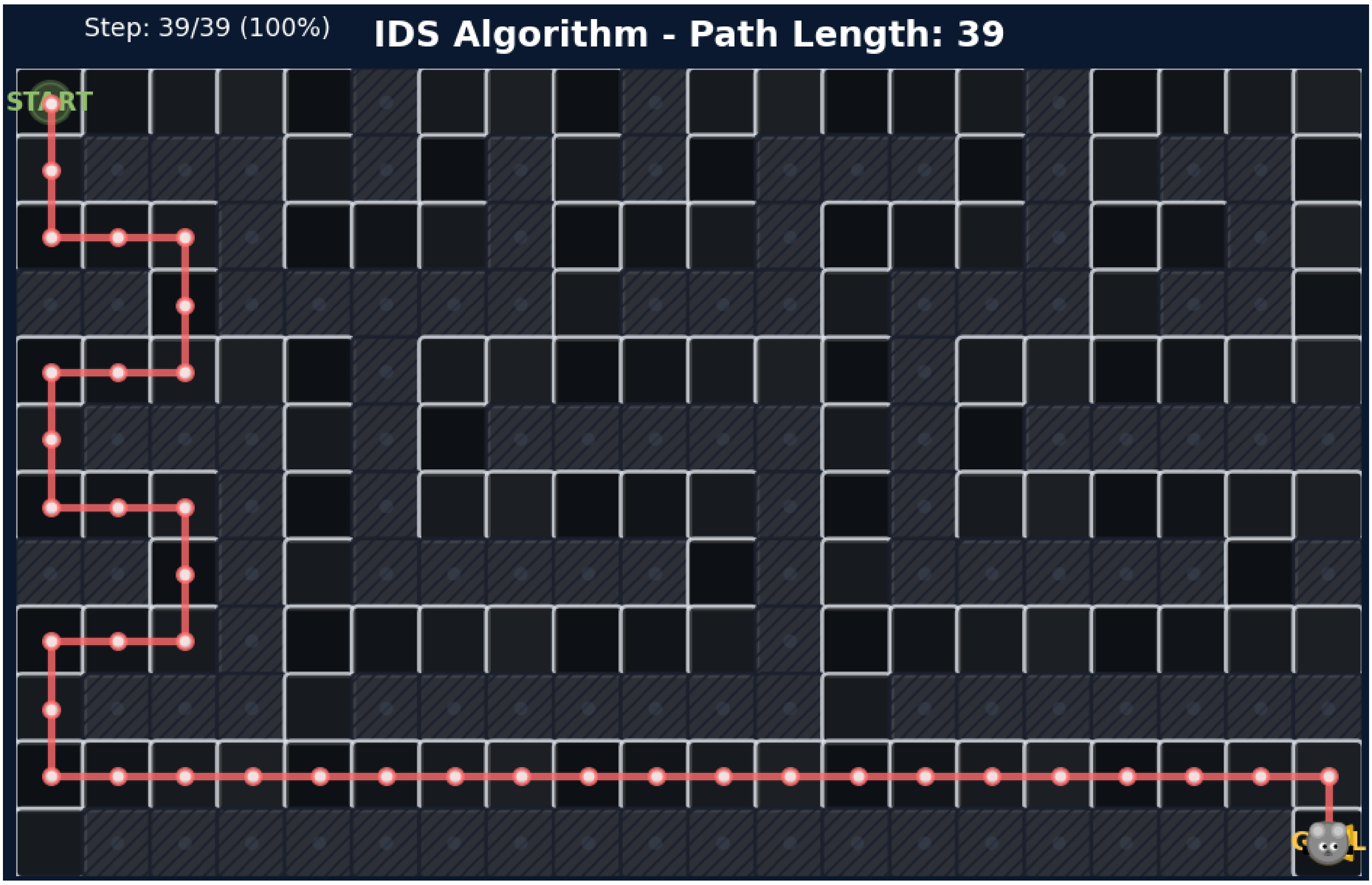
How it was used in the project:

- Start with the starting position in the stack
- Pop nodes from the top
- Check if it is the goal
- Add neighbors to the stack
- Automatically backtrack when hitting dead-ends

Why it works for the maze:

- Uses less memory than BFS
- Can find a solution quickly, even if it's not optimal
- Good when memory is limited

IDS (Iterative Deepening Search):



IDS combines DFS memory efficiency with BFS completeness. It runs DFS with increasing depth limits until the goal is found.

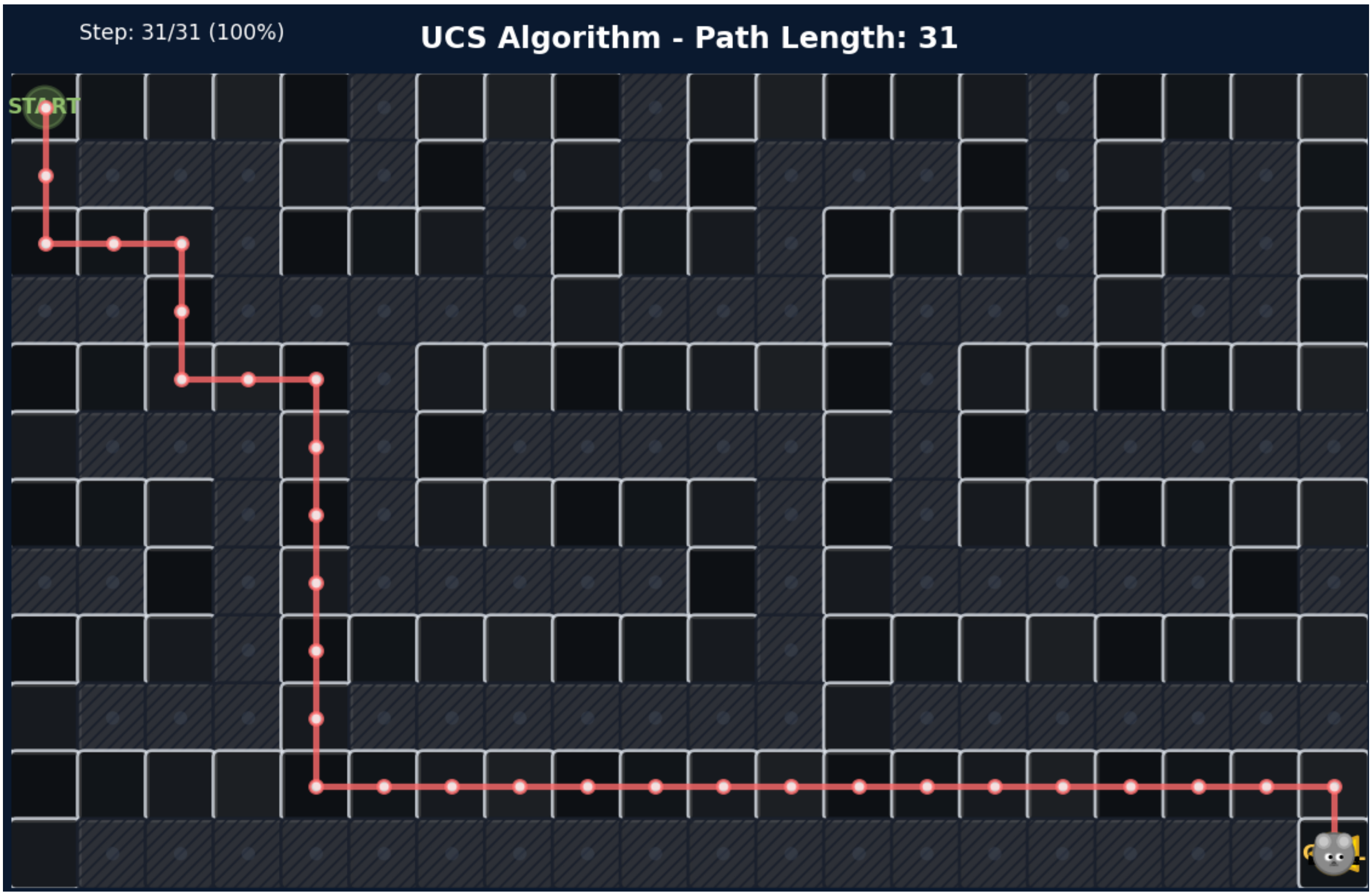
How it was used in the project:

- Start with depth limit = 0
- Run depth-limited DFS
- Increase depth gradually until goal is found

Why it works for the maze:

- Finds the shortest path
- Uses less memory than BFS
- Good compromise between memory and completeness

UCS (Uniform Cost Search):



UCS expands the node with the lowest cost from the start. It is similar to Dijkstra's algorithm.

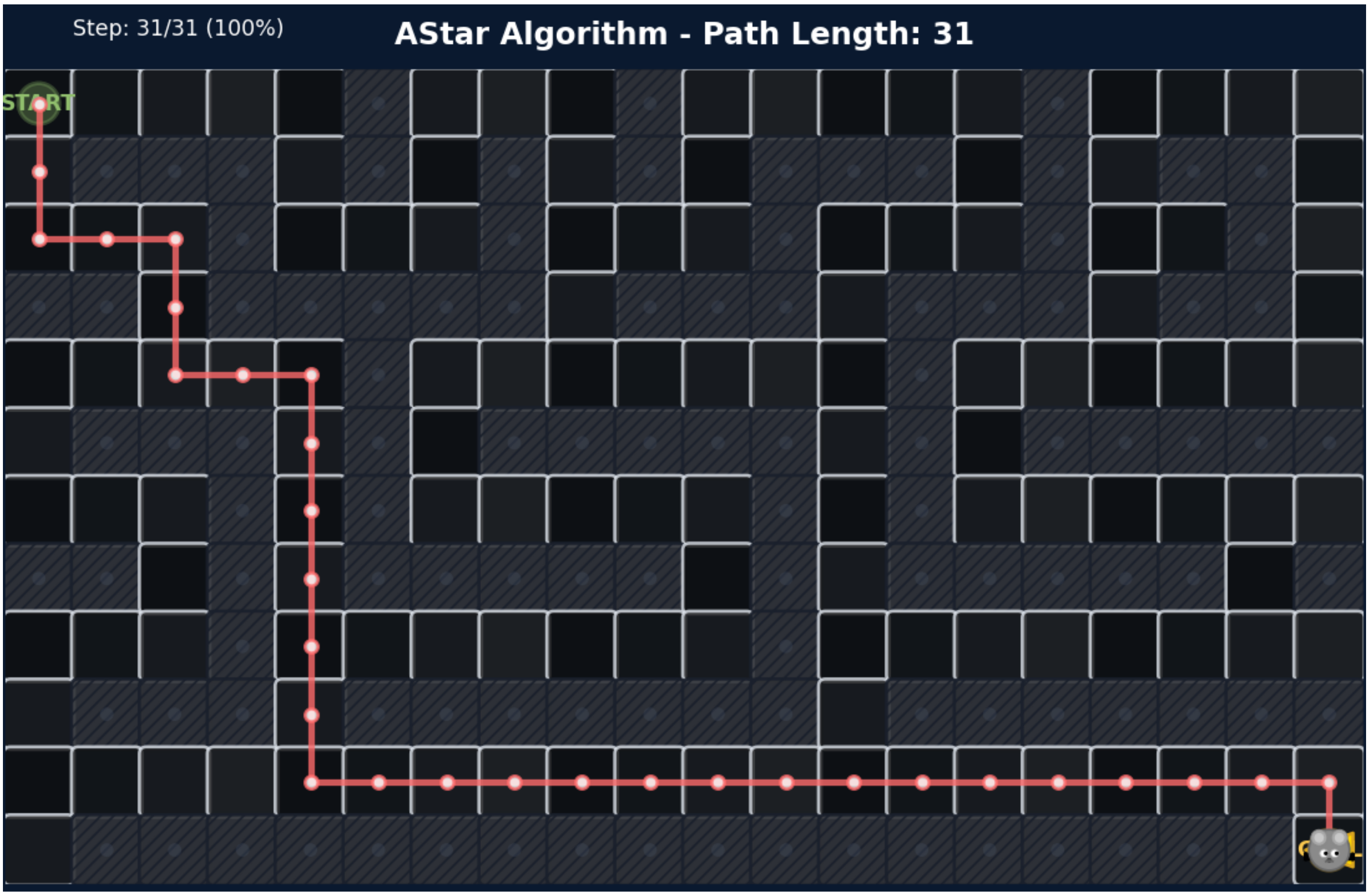
How it was used in the project:

- Use a priority queue with the start node at cost 0
- Expand the node with the lowest cost
- Add neighbors to the queue with updated costs
- Track the lowest cost paths to each node

Why it works for the maze:

- Guarantees the shortest path
- Can handle varying move costs (not needed here, but useful in general)

A* (A-Star Search):



A* improves UCS by adding a heuristic function to estimate the distance to the goal. Nodes are expanded based on

$$f(n) = g(n) + h(n).$$

where g(n) is the cost from start and h(n) is the heuristic.

How it was used in the project:

- Start with the start node in a priority queue
- f(n) = g(n) + h(n), where h(n) is the Manhattan distance to goal
- Expand node with smallest f(n)
- Add valid neighbors to the queue with updated scores
- Track parents to reconstruct the path

Why it works for the maze:

- Most efficient search for grid-based problems
- Guarantees the shortest path if the heuristic is admissible
- Reduces unnecessary exploration

4. Heuristic Function for A*

Chosen Heuristic: Manhattan Distance
Formula: $h(n) = |current\_x - goal\_x| + |current\_y - goal\_y|$
Why it works:
- Matches the 4-direction movement in the maze
- Never overestimates the cost (admissible)
- Easy to calculate
- Guides search efficiently

5. Experimental Results

Algorithm	Steps	Cost	Expanded nodes	Time(ms)	Memory	Is optimal?	Success Rate
BFS	31	30	108	0.20	216	Yes	100%
DFS	67	66	73	0.08	110	NO	100%
IDS	39	38	77	5.78	92	NO	100%
UCS	31	30	106	0.24	265	Yes	100%
A*	31	30	50	0.09	150	Yes	100%

Note: Values are approximate avarages from multiple runs

6. Analysis & Comparison

Observations:

1. **BFS and UCS** expanded nearly the same number of nodes (BFS 108, UCS 106) because all moves have **uniform cost**, so UCS behaves like BFS. Both algorithms found the **shortest path** (Steps = 31, Cost = 30).

2. **DFS** was the **fastest** (0.08 ms) and used relatively little memory (110), but it found a **longer path** (Steps = 67, Cost = 66), so it is **not optimal**.

3. **IDS** was the **slowest** (5.78 ms) but used the **least memory** (92). It found a path longer than BFS/UCS/A* (Steps = 39, Cost = 38), so it is **not fully optimal**, but it guarantees a solution.

4. A* gave the **best balance of speed, optimality, and efficiency**: it found the **shortest path**, used moderate memory (150), and expanded the **fewest nodes** (50), thanks to its Manhattan distance heuristic.

Memory Usage Ranking:

1. **UCS (265)**: uses the most memory because it stores all possible cells along with the cost for each cell.

2. **BFS (216):** stores all cells at the current depth level in the maze.

3.A*(150): keeps track of open and closed cells, plus extra data for the heuristic calculations.

4. DFS (110): only stores the current path of cells being explored in the stack.

5. IDS (92): uses the least memory because it stores only the current path of cells for each depth-limited iteration

Why These Results Occur:

- Uniform move costs make BFS and UCS behave similarly in terms of expanded nodes and path length.
- A* uses the Manhattan heuristic, which guides the search efficiently toward the goal, reducing unnecessary exploration.
- DFS explores deeply in one direction, so the path length depends heavily on the order of exploration.
- IDS repeats DFS multiple times with increasing depth limits, which increases runtime but keeps memory usage low.

7. Conclusion

Best Algorithm: A* Search with Manhattan distance

Reasons:
- Guarantees shortest path
- Efficient and fast
- Works well on larger mazes
- Practical for real-world pathfinding

When to Use Other Algorithms:
- BFS: small mazes, memory not an issue, need shortest path
- DFS: fast solution acceptable, memory limited, path may not be shortest
- IDS: memory-efficient and complete, can tolerate slower execution
- UCS: for variable move costs, need guaranteed optimal path
- A*: good heuristic, need optimal and fast solution

This project shows the trade-offs of different search algorithms and how heuristics improve performance. It provides insight into choosing the best algorithm for pathfinding tasks.

Team Contributions

Student ID	Stident Name	Algorithm	Other tasks
2023042	Asmaa Ahmed Elsayd	• Breadth-First Search (BFS) algorithm implementation • Depth-First Search (DFS) algorithm implementation	•Maze design and implementation
202350	Amira Khamis Hendawy	•Iterative Deepening Search (IDS) algorithm implementation	• Writing the final project report • Performance analysis and comparison of algorithms • GitHub Repo
2023035	Esraa Fawzy Elsayd	•Uniform Cost Search (UCS) algorithm implementation	•main program development •user input handling.
2023043	Asmaa Zakaria Omar	•A* Search algorithm implementation	• README documentation