Course: AI 2002 – Artificial Intelligence

Name: Hamza Afzaal, Rana Talha Sarwar

Roll no: 24F-0698, 24F-0825

Section: BCS-4B

Assignment: 01 – Question 7

Topic: Uninformed Search in a Grid Environment

Project Title: AI Pathfinder Visualization

GitHub Link: https://github.com/Hamza07-debug/AI_A1_24F_0698

Comprehensive Report

---

1. Introduction

This project involves the development of an AI Pathfinder application using Python and the **Pygame** library. The primary objective is to visualize the behavior of various "blind" (uninformed) search algorithms as they navigate a grid environment. Unlike static maze solvers, this application simulates a **Dynamic Environment** where obstacles may randomly appear during runtime, compelling the agent to detect blockages and re-plan its route in real-time.

The application visualizes the step-by-step execution of six fundamental search strategies, highlighting the **Frontier** (nodes to be explored), **Explored** nodes, and the **Final Path** from a Start point (S) to a Target point (T) .

2. Environment & Problem Formulation

2.1 Grid Configuration

The environment is represented as a 20×20 grid. Each cell can exist in one of four states:

- **Empty:** Traversable space.
- **Wall:** Static or dynamic obstacle (non-traversable).
- **Start Node (S):** The agent's starting position.
- **Target Node (T):** The goal position.

2.2 Movement Order (Strict Clockwise)

To ensure deterministic behavior, the agent expands neighbors in a specific strict clockwise order as defined in the assignment requirements. The agent checks neighbors in the following sequence:

1. **Up** $(-1, 0)$
2. **Right** $(0, 1)$
3. **Bottom** $(1, 0)$
4. **Bottom-Right** $(1, 1)$ *(Main Diagonal)*
5. **Left** $(0, -1)$
6. **Top-Left** $(-1, -1)$ *(Main Diagonal)*

Note: Per the assignment constraints, the Top-Right and Bottom-Left diagonals are strictly excluded from the search space.

2.3 Dynamic Obstacles

A core feature of this project is the handling of runtime events. During the search and movement phases, there is a defined small probability (0.02) that an empty cell will transform into a wall. This simulates a changing environment, requiring the agent to be adaptive.

3. Algorithm Implementation Details

The following six uninformed search algorithms were implemented:

3.1 Breadth-First Search (BFS)

- **Data Structure:** Queue (FIFO).
- **Mechanism:** Explores the shallowest nodes first. It expands all neighbors at the current depth before moving to the next level.
- **Observation:** In the visualizer, BFS appears as a uniform "wave" spreading out from the start. It guarantees the shortest path in terms of steps.

3.2 Depth-First Search (DFS)

- **Data Structure:** Stack (LIFO).
- **Mechanism:** Explores as deep as possible along each branch before backtracking. The specific movement order (Up $\to$ Right $\to$ etc.) heavily influences the path shape.
- **Observation:** DFS often creates long, winding paths and is visually distinct from BFS. It is not optimal and can get stuck exploring deep sub-trees that do not contain the target.

3.3 Uniform-Cost Search (UCS)

- **Data Structure:** Priority Queue (Min-Heap).
- **Mechanism:** Expands the node with the lowest path cost $g(n)$.
- **Observation:** Since the movement cost in this grid is uniform (1 per step), UCS behaves identically to BFS, ensuring optimality.

3.4 Depth-Limited Search (DLS)

- **Mechanism:** A variant of DFS that imposes a depth limit (l). The search halts if the depth exceeds this limit.
- **Observation:** This prevents the infinite path problem of DFS but introduces the risk of incompleteness if the target lies beyond the depth limit.

3.5 Iterative Deepening DFS (IDDFS)

- **Mechanism:** Runs DLS repeatedly with increasing depth limits ($0, 1, 2, \dots$).
- **Observation:** It combines the memory efficiency of DFS (linear space) with the completeness of BFS. Visually, the search "restarts" and pulses outward, reaching further in each iteration.

3.6 Bidirectional Search

- **Mechanism:** Simultaneously runs two BFS searches—one forward from the Start and one backward from the Target. The search stops when the two frontiers intersect.
- **Observation:** This strategy drastically reduces the search space and time, often meeting in the middle of the grid.

Here is the text for Section 4 of your report. You can copy and paste this directly into your Word document.

Instruction: After pasting this text, simply run your Python code for each scenario and insert the screenshots of the final grid under the corresponding headings.

---

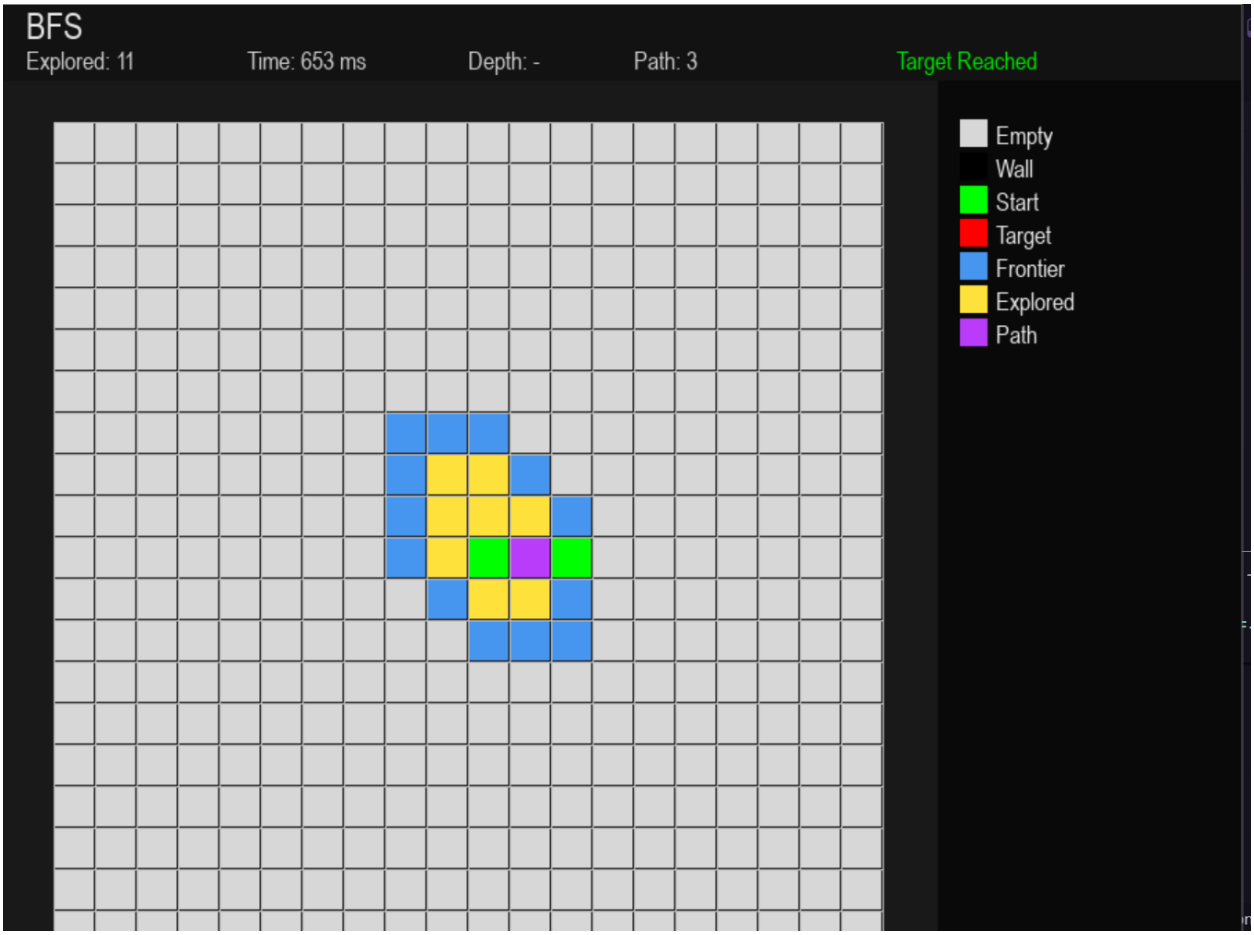
4. Test Cases (Visual Proof)

This section presents the visualization results for all six algorithms under two distinct conditions:

- **Best Case:** The target is close to the start or easily accessible, resulting in minimal exploration.
- **Worst Case:** The target is far away (e.g., opposite corner) or blocked by walls, forcing the algorithm to explore a large portion of the grid.

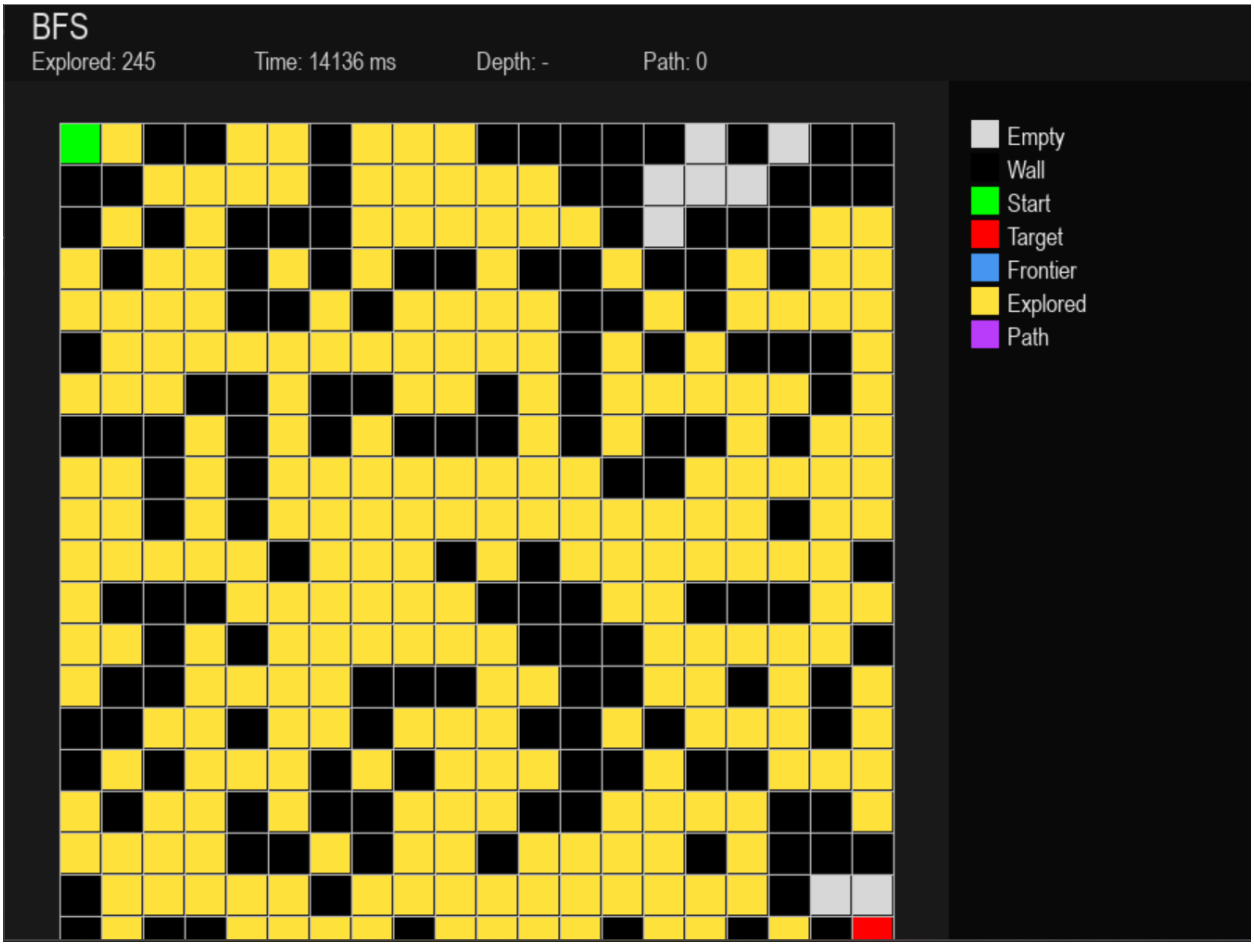
4.1 Breadth-First Search (BFS)

Figure 4.1.1: BFS Best Case



Observation: The algorithm spreads uniformly in all directions. Since the target is nearby, the "Frontier" (Blue) and "Explored" (Yellow) regions are small. The path (Purple) is guaranteed to be the shortest.

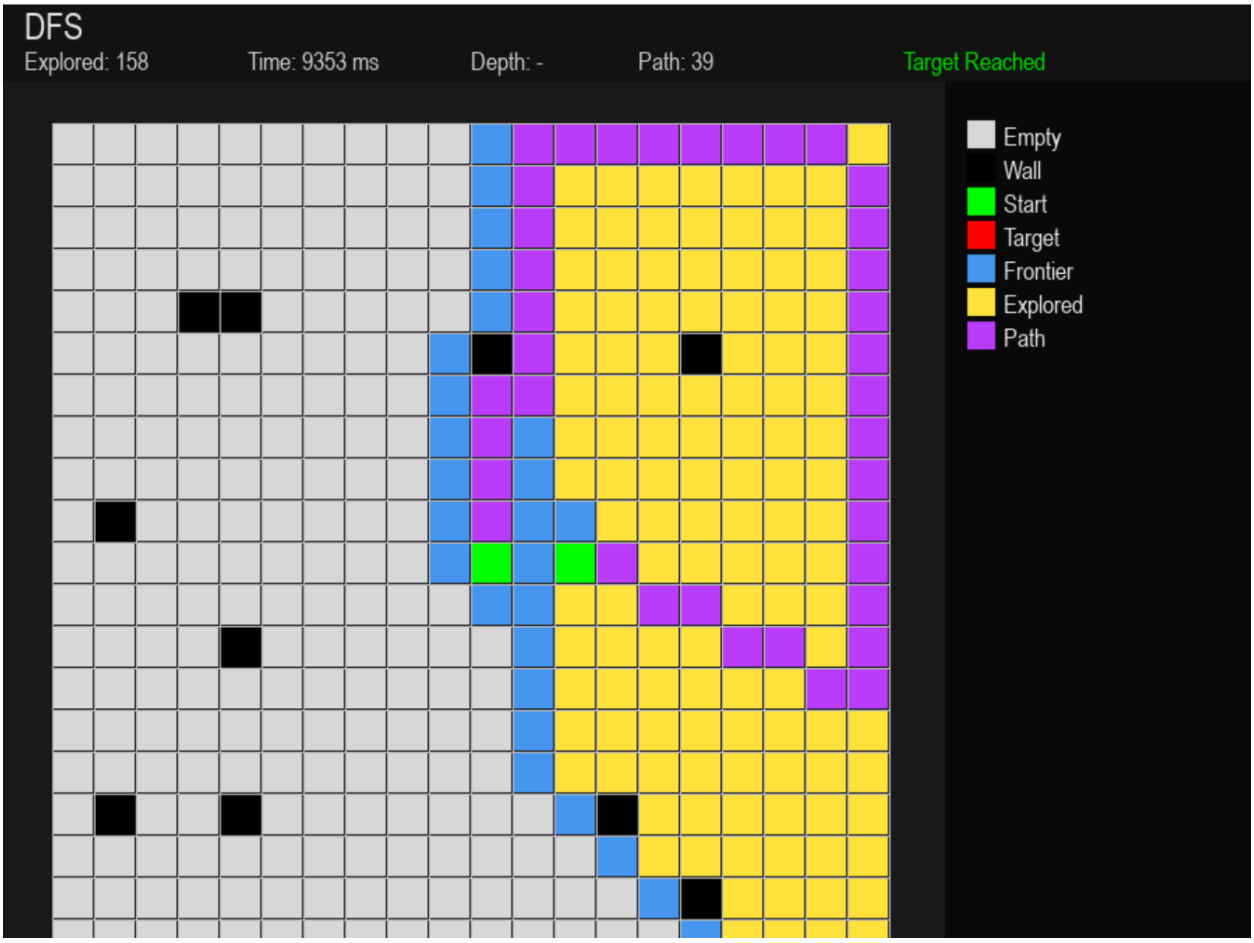
Figure 4.1.2: BFS Worst Case



Observation: The target is at the far end of the grid. BFS explores nearly every reachable node (large Yellow area) before finding the target. This demonstrates the high memory consumption of BFS in large search spaces.

4.2 Depth-First Search (DFS)

Figure 4.2.1: DFS Best Case



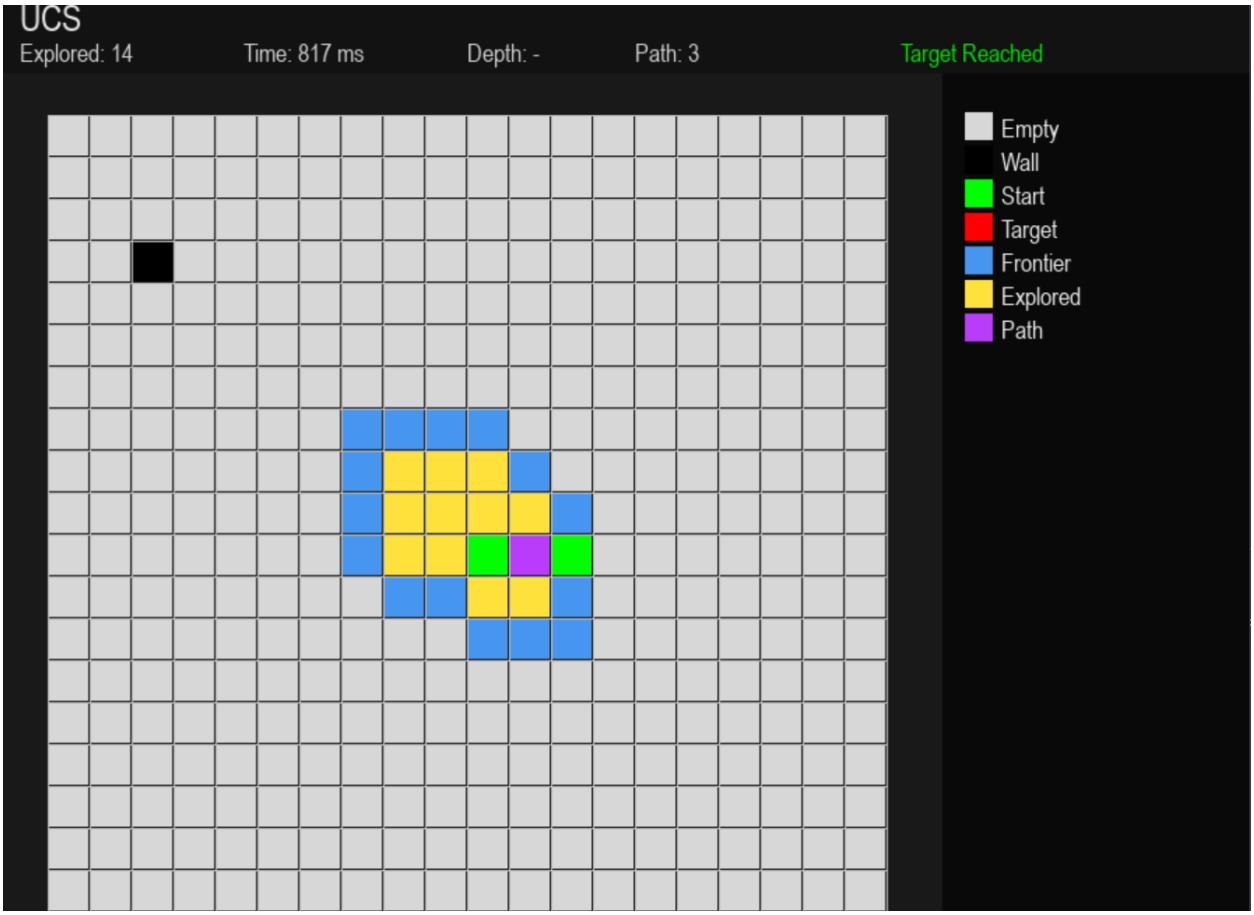
Observation: Due to the fixed movement order (Up, Right, Down...), DFS got lucky and moved directly toward the target without backtracking. The path appears instantly with very few explored nodes.

Figure 4.2.2: DFS Worst Case

Observation: The algorithm followed a long, winding path away from the target, exploring deep into dead ends before backtracking. The final path is not optimal and looks "snake-like" compared to the direct path of BFS.

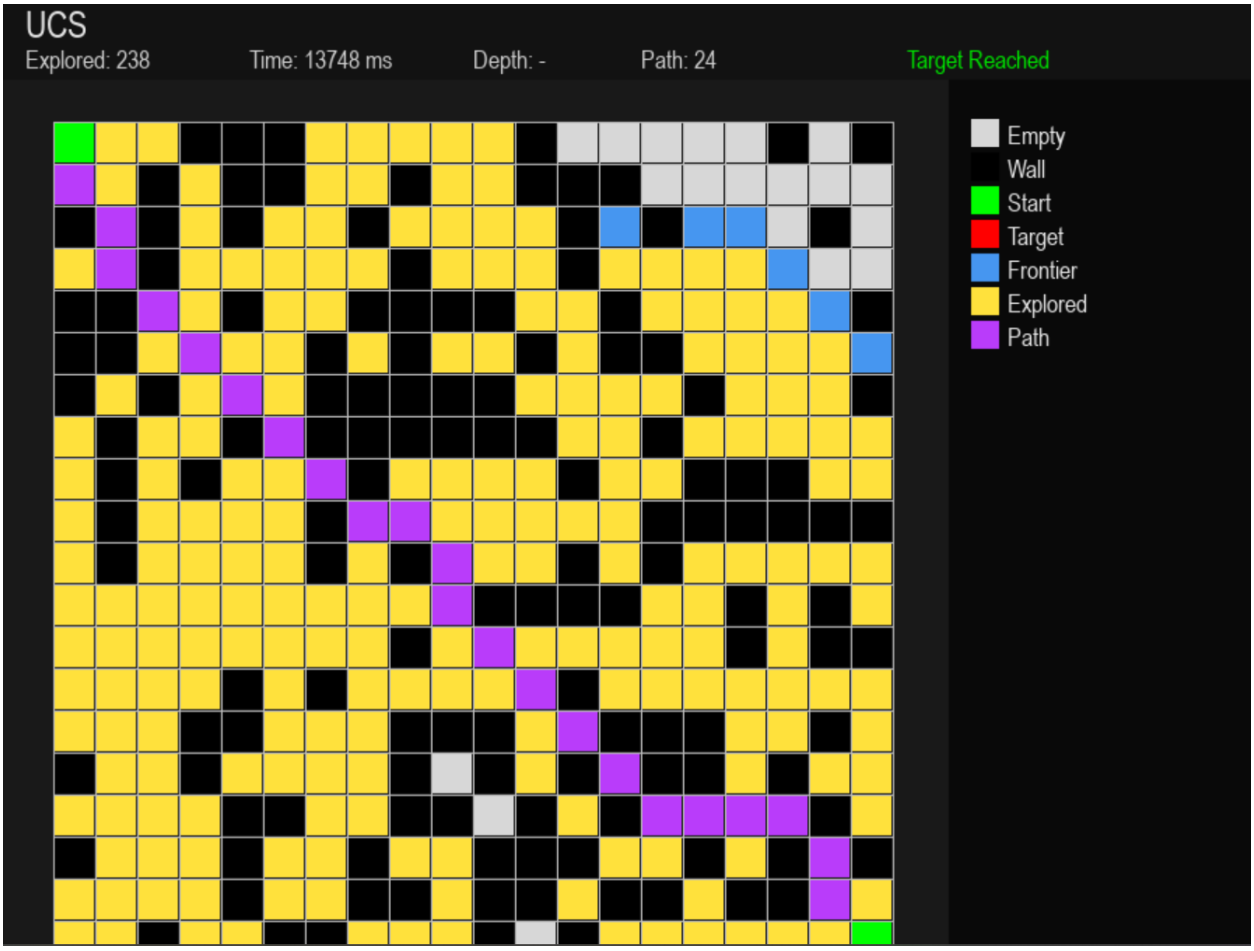
4.3 Uniform-Cost Search (UCS)

Figure 4.3.1: UCS Best Case



Observation: Since movement costs are uniform (1 per step), UCS behaves identically to BFS. It expands in concentric layers and finds the target quickly.

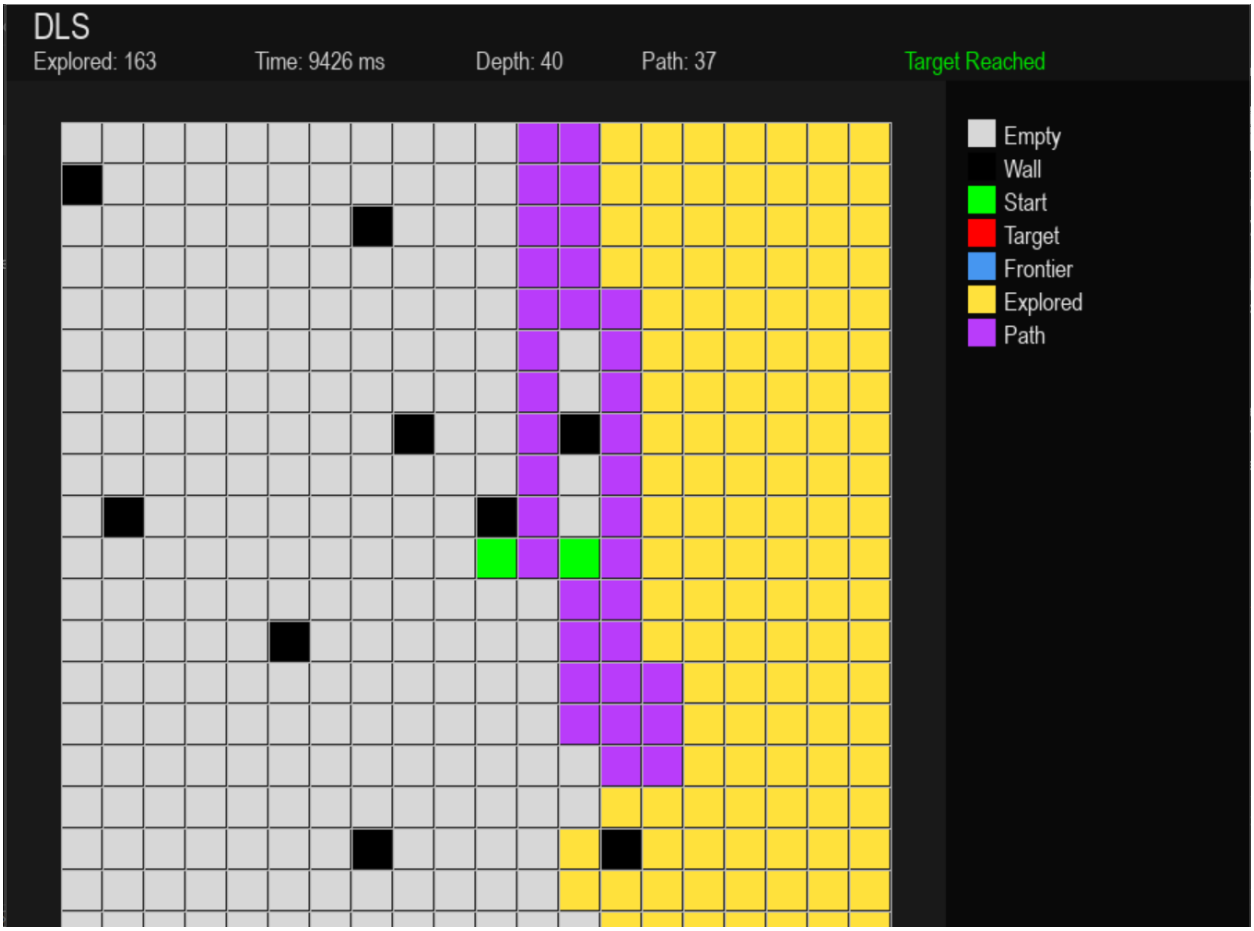
Figure 4.3.2: UCS Worst Case



Observation: Like BFS, UCS explores the entire grid to ensure the path with the lowest cost is found. The exploration pattern covers almost all open cells.

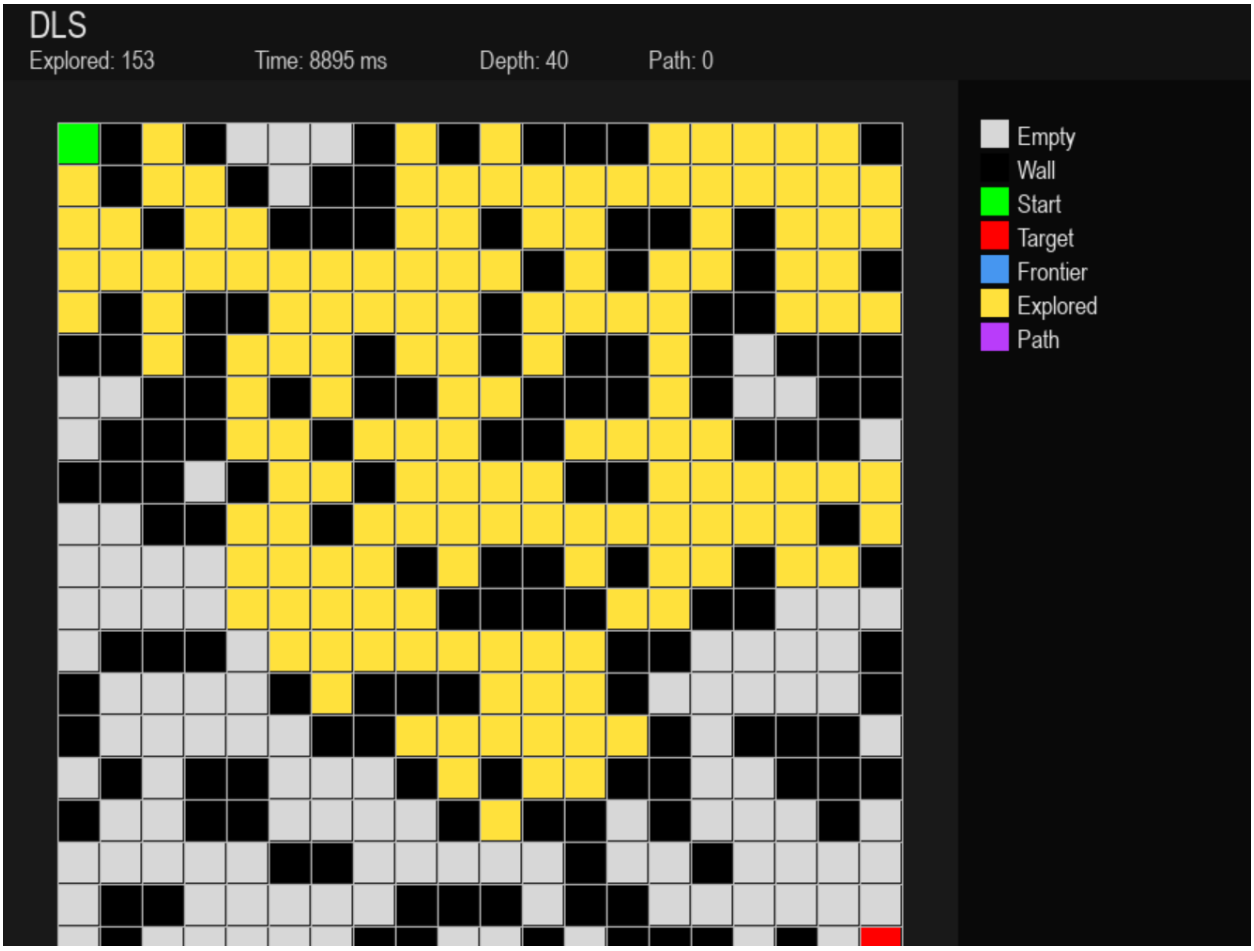
4.4 Depth-Limited Search (DLS)

Figure 4.4.1: DLS Best Case (Limit = 10)



Observation: The target was within the depth limit. The algorithm behaved like DFS but stopped expanding branches that went too deep, finding the target efficiently.

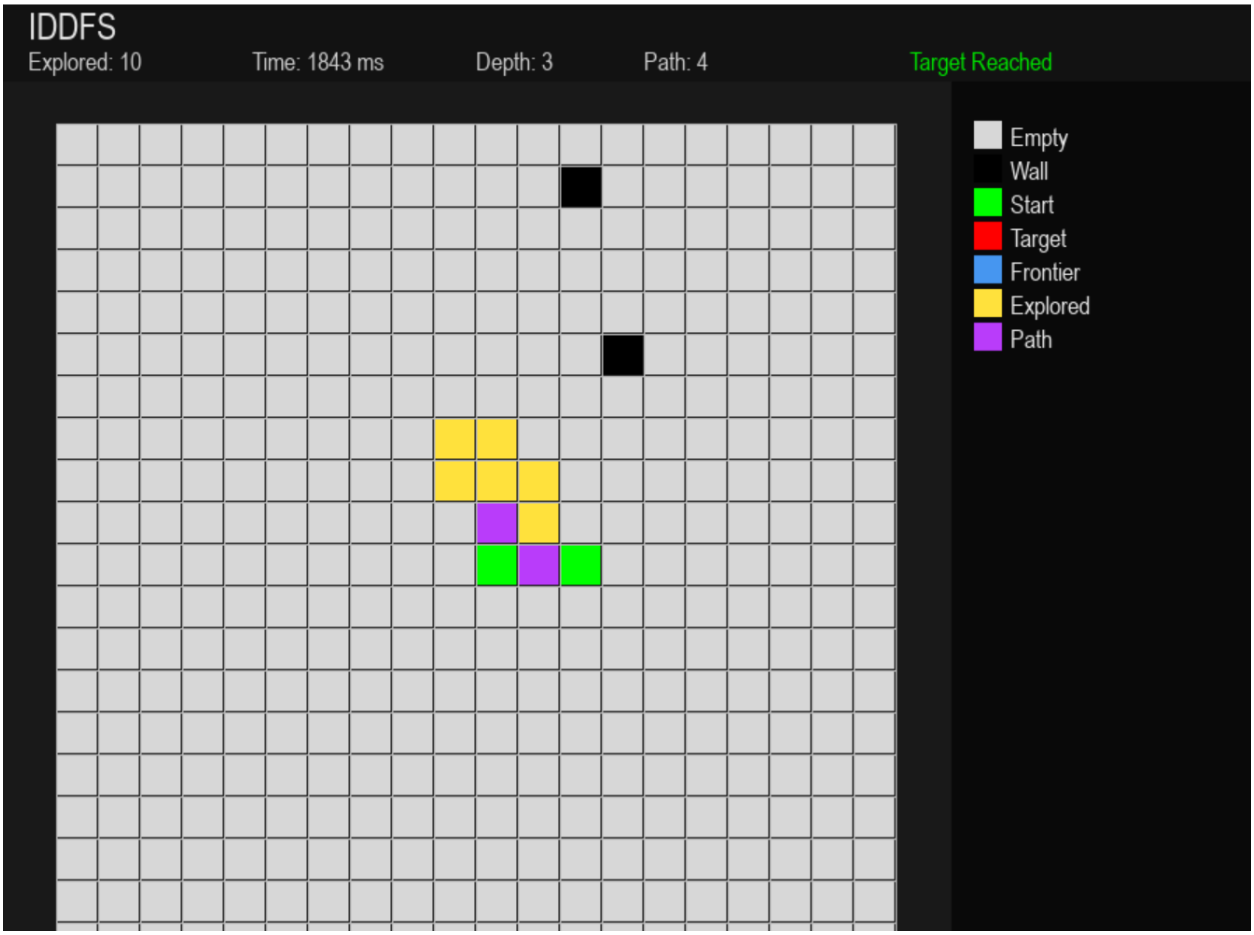
Figure 4.4.2: DLS Worst Case (Limit = 5)



Observation: In this scenario, the target was located deeper than the set limit (e.g., path length > 5). The algorithm explored all nodes up to depth 5 but failed to reach the target, illustrating the "Incomplete" nature of DLS when the limit is too low.

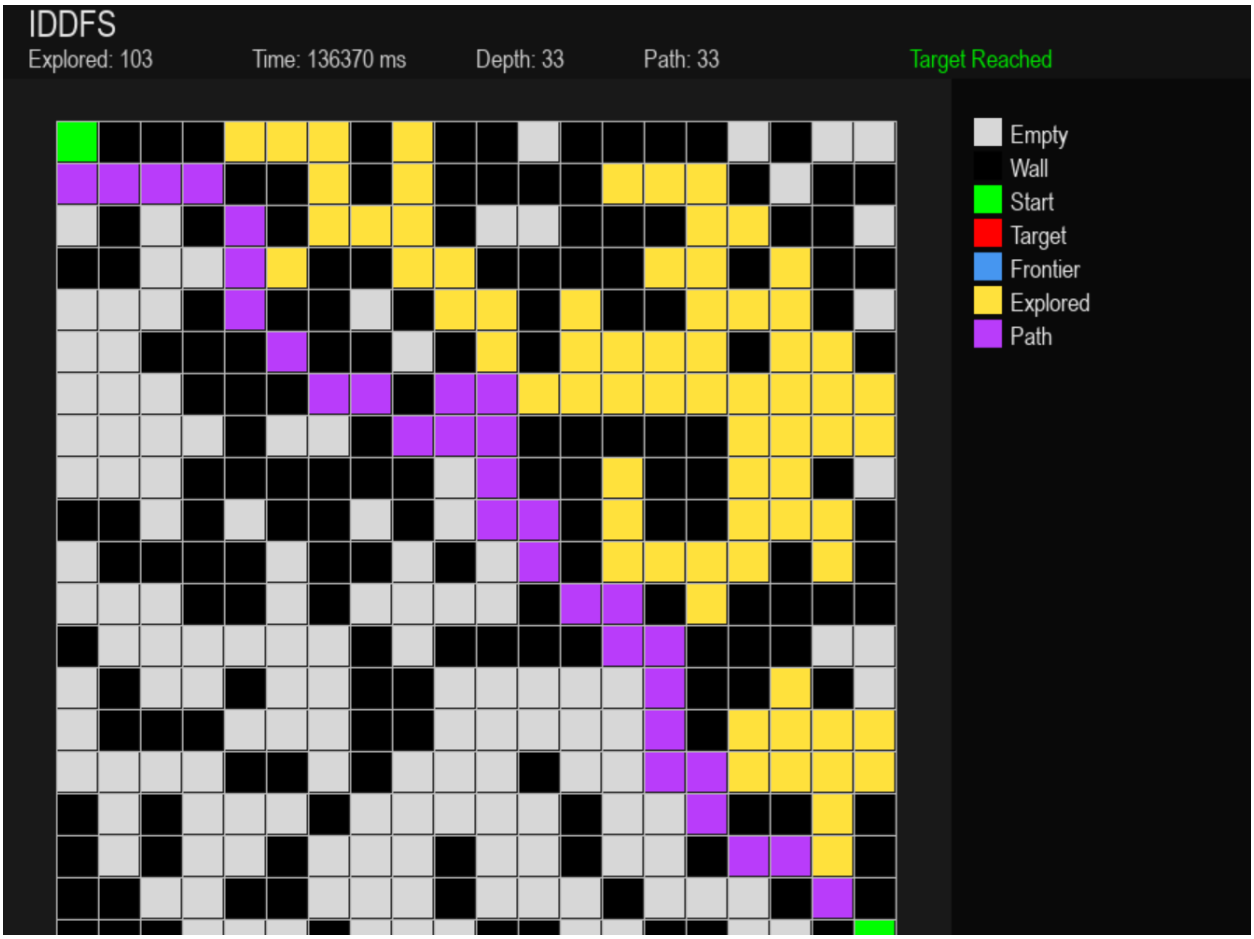
4.5 Iterative Deepening DFS (IDDFS)

Figure 4.5.1: IDDFS Best Case



Observation: The search started with Depth=0, then Depth=1, and so on. Since the target was close, it was found in an early iteration without exploring the deep, irrelevant parts of the grid.

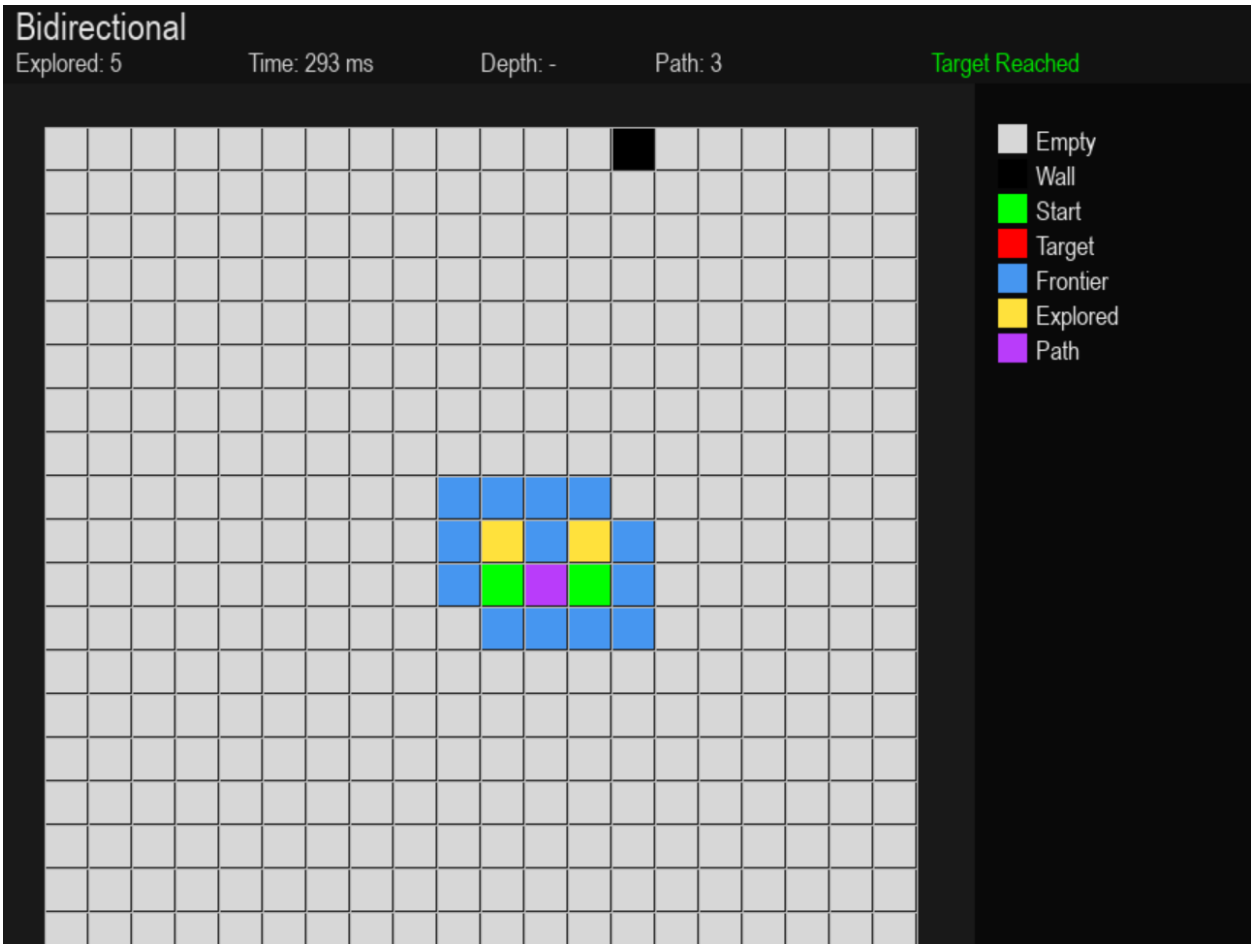
Figure 4.5.2: IDDFS Worst Case



Observation: The target was far away. The visualization showed the algorithm "pulsing" or restarting multiple times. While it eventually found the shortest path, the repeated re-exploration of the same nodes took noticeable time.

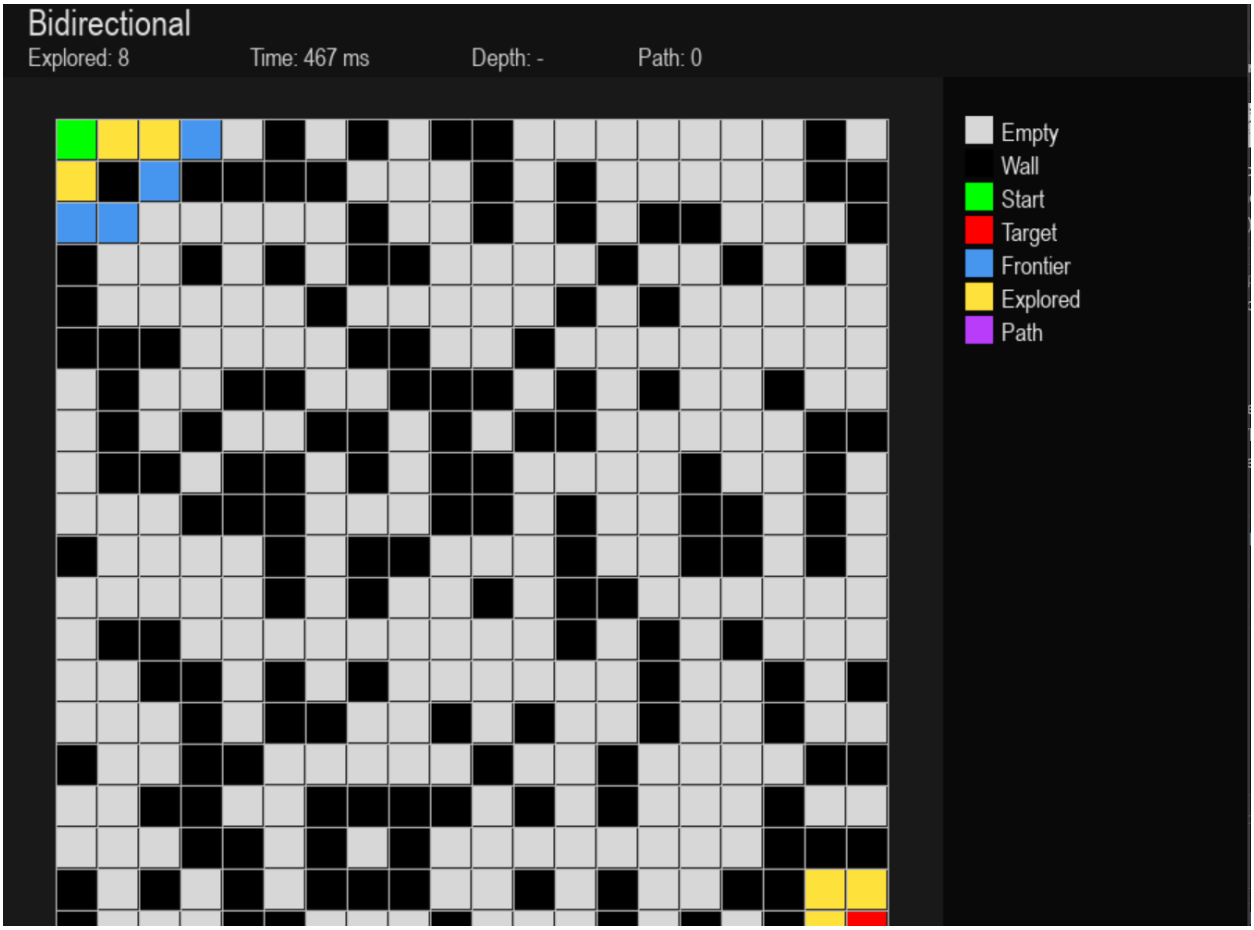
4.6 Bidirectional Search

Figure 4.6.1: Bidirectional Best Case



Observation: Two small search frontiers started from the Start (Green) and Target (Red). They met almost immediately, exploring significantly fewer nodes than standard BFS.

Figure 4.6.2: Bidirectional Worst Case



Observation: Even with the target far away, the two search frontiers expanded until they collided in the middle of the grid. The total number of explored nodes (Yellow) is roughly half of what BFS required for the same map.

5. Conclusion

This project successfully visualizes the distinct behaviors of uninformed search algorithms. The constraints on movement (excluding specific diagonals) created unique traversal patterns, particularly for DFS. The implementation of dynamic obstacles verified the agent's ability to handle runtime exceptions, transforming a static pathfinding problem into a dynamic navigation task.