**Implementation of the uninformed search methods on the “agent” environment.**

Implements an AI agent that navigates a maze using various search algorithms. The maze is represented as a 2D grid where walls (1s) block movement, and the agent must find a path from the start position (3) to the goal position (2).

**Key Features:**

1. **Maze Representation**: The grid-based maze contains walls, an agent, and a goal.
2. **Agent Definition**: The OurAgent class represents the agent, which moves within the environment.
3. **Search Algorithms**: The agent can navigate the maze using:
   * **Breadth-First Search (BFS)**: Explores all possible paths level by level.
   * **Depth-First Search (DFS)**: Explores one path deeply before backtracking.
   * **Greedy Best-First Search**: Selects the next move based on the heuristic distance to the goal.
   * *A Search*\*: Uses both path cost and heuristic distance for an optimal route.
4. **Pathfinding Execution**: The user selects an algorithm, and the agent finds and prints the path.
5. **Simulation**: The environment is visualized using GraphicEnvironment, and the agent follows the computed path.

The program effectively demonstrates AI-based maze navigation by allowing different search strategies to be tested and compared.

# AI Agent Maze Navigation Analysis

This document evaluates four search algorithms used for AI agent navigation in a maze: Breadth-First Search (BFS), Depth-First Search (DFS), Greedy Best-First Search, and A\* Search. The analysis includes time complexity, space complexity, and path efficiency.

## 1. Complexity and Path Analysis

|  |  |  |  |
| --- | --- | --- | --- |
| **Algorithm** | **Time Complexity** | **Space Complexity** | **Path Length** |
| **Breadth-First Search** | O(b^d) | O(b^d) | Shortest |
| **Depth-First Search** | O(b^d) | O(d) | Longest |
| **Greedy Best-First Search** | O(b^m) | O(b^m) | Varies |
| **A\* Search** | O(b^m) (worst case) | O(b^m) | Optimal |

- **b**: branching factor (number of possible moves at each step)

- **d**: depth of the shallowest solution

- **m**: maximum depth of the search tree

## 2. Comparative Analysis

- **Algorithm with Maximum Time Complexity**: BFS and DFS (O(b^d))

- **Algorithm with Maximum Space Complexity**: BFS (O(b^d))

- **Algorithm with Longest Path**: DFS (explores one path deeply)

- **Algorithm with Shortest Path**: BFS (guarantees shortest path)

- **Most Time-Efficient Algorithm**: A\* Search (uses heuristic)

- **Most Space-Efficient Algorithm**: DFS (O(d) space complexity)

## 3. Conclusion

Each algorithm has strengths and weaknesses:

- **BFS** is best for finding the shortest path but requires large memory.

- **DFS** is memory-efficient but can get stuck in deep searches.

- **Greedy Best-First Search** is fast but not optimal.

- **A\*** is the best overall, balancing optimality and efficiency.

This analysis helps in selecting the right algorithm based on problem constraints and efficiency needs.