AI 2002 – Artificial Intelligence

Assignment 1 – Question 7

Uninformed Search in a Grid Environment

Project Title: GOOD PERFORMANCE TIME APP

---

1. Introduction

In this project, I built a small AI pathfinder in Python using Pygame.
The purpose of this project is to visualize how different uninformed search algorithms explore a grid to reach a target.

The program creates a 20x20 grid. It randomly places walls, a start cell, and a target cell. Then different search algorithms are executed step-by-step so we can see how they explore the grid.

The GUI shows:

- Frontier nodes
- Explored nodes
- Final path
- Time taken
- Number of explored nodes
- Depth (for DLS and IDDFS)
- Path length

The following algorithms were implemented:

- Breadth First Search (BFS)
- Depth First Search (DFS)
- Uniform Cost Search (UCS)
- Depth Limited Search (DLS)
- Iterative Deepening DFS (IDDFS)
- Bidirectional Search

The goal was not only to find a path but also to understand how each algorithm behaves.

---

2. Grid Environment

The environment is a simple 20x20 2D grid.

Each cell can be:

- Empty
- Wall
- Start
- Target

Static walls are generated randomly at the beginning using a fixed probability.
The start and target positions are always placed on empty cells.

Movement Order

The movement order is fixed and always follows this sequence:

1. Up
2. Right
3. Down
4. Down-Right (diagonal)
5. Left
6. Up-Left (diagonal)
7. Top-Right (diagonal)
8. Bottom-Left (diagonal)

This order is important because it decides which neighbor is explored first.

GUI Colors

- Empty: Light Grey
- Wall: Black
- Start: Green

- Target: Red
- Frontier: Blue
- Explored: Yellow
- Final Path: Purple

The top of the window shows:

- Algorithm name
- Explored nodes count
- Time in milliseconds
- Depth (if applicable)
- Path length

---

3. Explanation of Each Algorithm (Implementation View)

This section explains how I implemented each algorithm in practical terms.

3.1 Breadth First Search (BFS)

For BFS, I used a queue (FIFO).

Steps:

- Start node is added to the queue.
- Remove node from front.
- Mark it as explored.
- Add all valid neighbors (in fixed order) to the queue if not visited.

BFS explores the grid level by level.
In the GUI, it looks like a wave spreading from the start.

Since all moves have equal cost, BFS always finds the shortest path in number of steps.

3.2 Depth First Search (DFS)

For DFS, I used a stack (LIFO).

Steps:

- Push start node into stack.
- Pop from top.
- Mark as explored.
- Push neighbors in reverse order.

DFS goes deep into one direction before backtracking.
In visualization, it creates long branches.

It does not guarantee the shortest path and may explore dead ends.

3.3 Uniform Cost Search (UCS)

I used a priority queue (min-heap).

Each entry stores:
(cost, cell)

Steps:

- Start cost = 0
- Remove lowest cost node
- For each neighbor, cost = current cost + 1
- Update if smaller cost is found

Since all moves cost 1, UCS behaves almost like BFS here.
With different costs, it would behave differently.

3.4 Depth Limited Search (DLS)

DLS is DFS with a depth limit.

- Recursive function is called with a limit.
- Each recursive call reduces the limit.
- If limit becomes 0, that branch stops.

If the depth limit is too small, the target may not be found.

3.5 Iterative Deepening DFS (IDDFS)

IDDFS runs DLS repeatedly with increasing depth:

Depth = 0, 1, 2, 3, ...

For each depth:

- Explored set is cleared.
- DLS runs again.

It repeats some work but eventually finds the solution with depth similar to BFS.

3.6 Bidirectional Search

This algorithm runs BFS from two sides:

- From start
- From target

Two queues and visited sets are maintained.

If a node appears in both visited sets, that is the meeting point.

The final path is built by combining both sides.

In visualization, it looks like two waves meeting in the middle.

---

4. Dynamic Obstacles and Re-Planning

The environment is dynamic.

At each step (2% probability):

- A random empty cell becomes a wall.

If this wall blocks the planned path:

- The algorithm is re-run from the current agent position.
- Frontier and explored sets are cleared.

- A new path is calculated.
- The agent follows the updated path.

This makes the environment more realistic and shows adaptive planning.

---

5. Comparison and Observations

5.1 Speed and Exploration

- BFS: Always finds shortest path but may explore many nodes.
- DFS: Can be fast or very slow depending on direction.
- UCS: Similar to BFS with equal cost.
- DLS: Works only if depth limit is sufficient.
- IDDFS: Repeats work but uses less memory.
- Bidirectional: Often explores fewer nodes when start and target are far apart.

5.2 Dynamic Obstacles

Re-planning increases time and explored count.
However, it makes the system flexible and realistic.

5.3 Visualization

The colors clearly show:

- Where algorithm will go next (Frontier)
- Where it has already been (Explored)
- Final solution path

Header statistics helped compare algorithms properly.

---

6. Conclusion

Through this project, I understood how uninformed search algorithms behave differently in the same environment.

Main learnings:

- BFS and UCS provide shortest paths.
- DFS behavior depends heavily on direction.
- DLS and IDDFS show the effect of depth control.
- Bidirectional search reduces exploration in many cases.
- Dynamic obstacles show the importance of re-planning.

Building the Pygame visualizer helped me understand these algorithms better than reading theory alone.