# AI Project Report

Maze Solving Using Search Algorithms

By : Sherif Yasser Elsaid

Course : Artificial Intelligence

Student ID : 2023110

Under Guidance :

Eng. Ahmed Sobhy

Eng. Omar Khaled

## Introduction

Maze-solving problems are a popular way to study problem-solving techniques in Artificial Intelligence.  At first glance, the problem seems simple: moving from a starting point to a goal. However, once obstacles and different movement costs are introduced, the problem becomes much more challenging. In this project, we focus on solving weighted maze problems, where not all paths have the same cost. Some routes are shorter but expensive, while others are longer but cheaper. This setup helps reveal how different algorithms make decisions under constraints.

## Problem Description

The main problem of this project is to find an efficient path from a start position (S) to a goal position (G) in a maze that contains walls and weighted paths. Unlike traditional mazes where every step has the same cost, this project assigns different costs to different cells. As a result, the shortest path is not always the best one. The challenge is to:

Choose a path that balances distance and cost

Avoid obstacles and high-cost areas

Reach the goal efficiently

## Why This Problem Matters

This type of problem closely represents real-world scenarios , such as:

Navigation systems choosing faster or cheaper routes

Robots avoiding dangerous or costly areas

Games where characters must manage risk and efficiency

Understanding this problem helps explain how intelligent systems make decisions beyond simple distance calculations.

## Maze Design Challenges

Designing the mazes was one of the hardest parts of the project . Simple mazes caused most algorithms to behave similarly, making comparison meaningless. To solve this, the mazes were carefully redesigned to include:

Multiple valid routes to the goal

Dead ends and misleading paths

High-cost cells placed on short paths

Low-cost cells placed on longer paths

Three different maze sizes (10×10, 20×20, and 30×30) were used, each with a unique structure.

## Problem Constraints

The maze problem follows these rules:

- Obstacles: Walls (#) block the way.
- Variable Costs: Unlike standard mazes, some paths are "Harder" to walk on (weighted 2, 3, or 5), while others are "Easier" (1).
- Efficiency: We don't just want any path; we want the cheapest path with the least computational effort.

## Environment & Technical Setup

The project is built using C++ for high performance.

- Grid System: A 2D array representation.
- Heuristics: We used Manhattan Distance to estimate the distance to the goal.
- Data Structures: Custom struct objects, priority queue for cost management, and vector for path reconstruction.

## Brief on Search Algorithms Used

### 1. Uninformed Search (Blind Search)

#### 1. BFS (Breadth-First Search)

- Logic: Explores all neighbors at the current depth before moving deeper.
- Pros: Guarantees the path with the fewest steps.
- Cons: Does not "see" weights; it might take a very expensive path just because it's shorter in tiles.

#### 2. DFS (Depth-First Search)

- Logic: Goes as deep as possible down one path before backtracking.
- Pros: Low memory usage.
- Cons: Often find extremely long, "bad" paths.

### 2. Informed Search (Heuristic Search)

#### 1. Hill Climbing

- Logic: A greedy algorithm that only moves to the neighbor that looks closest to the goal.
- Pros: Extremely fast and uses almost no memory.
- Cons: Often fails in complex mazes because it gets stuck in dead ends .

#### 2. UCS (Uniform Cost Search)

- Logic: Always expand the node with the lowest cumulative cost g(n).

- Pros: Guaranteed finding the cheapest path.
- Cons: Explores in a circle; wastes time looking at paths that lead away from the goal.

### 3. A* (A-Star Search) 🚀 *

- Logic: The "Gold Standard." It uses f(n) = g(n) + h(n).
- Pros: It combines the cost spent (g) with the estimated distance to the goal (h). It finds the cheapest path while ignoring useless directions.
- Cons: Requires a good heuristic function.

## Expected Differences in Solutions

Due to weighted paths, algorithms are expected to behave differently:

- Some may prefer shorter paths
- Others may prioritize lower cost
- Some may fail due to greedy or incomplete decisions
- This makes the problem ideal for observing real differences in algorithm behavior.

## Observed Behavior

After testing the algorithms, several patterns were observed:

- Some algorithms reached the goal quickly but with higher cost
- Others took longer routes to reduce total cost
- Certain algorithms struggled or failed in complex mazes
- These results highlight how maze design and cost distribution affect performance.

## Scientific Analysis & Findings

- Through our testing on Easy, Hard, and Insane mazes, we observed the following:
- Optimality: UCS and A* always find the same total path cost. They are the only algorithms in this project that are truly "Optimal."
- Performance: A* consistently expanded to fewer nodes than UCS. This proves that giving an algorithm a "hint" (Heuristic) makes it significantly smarter.
- Path Quality: BFS is great for simple mazes, but in weighted mazes, its paths are more "expensive" compared to A*.
- Reliability: Hill Climbing is too simple for "Insane" levels; without the ability to backtrack or look at total cost, it fails where others succeed.

## Conclusion

**This project demonstrates that solving a maze is not just about reaching the destination , but about making smart decisions under constraints. By focusing on the problem itself and carefully designing the mazes, the project successfully highlights how different strategies lead to different outcomes.**

## Future Improvements

**Possible future extensions include:**

- **Dynamic or changing maze costs**
- **Real-time visualization of algorithm paths**
- **More advanced heuristics**
- **Larger and more complex environments**