Team Contributions

1. **Mostafa Mohamed Mostafa Elgamal (ID: 2023216)**
   Map Logic: Designing the grid, defining traffic (high-cost nodes) and buildings.
   Visualization: GUI coding and ambulance movement.

2. **Abdelhameed Adel Abdelhameed Elmehsenawy (ID: 2023311)**
   Implemented the Genetic Algorithm.

3. **Ziad Wael Awad (ID: 2023096)**
   Implemented Hill Climbing algorithm.

4. **Hadeel Mohamed Helmy Elsharkawy (ID: 2023246)**
   Implemented BFS and DFS algorithms.

5. **Shahd Ahmed Abdelaziz Elgendy (ID: 2023111)**
   Implemented A*.

6. **Margo Fouad Aziz (ID: 2023161)**
   Implemented UCS and IDS.

Smart Emergency Pathfinding System

Artificial Intelligence (AI) search algorithms play a vital role in solving real-world problems that require finding optimal or near-optimal solutions within a large search space.

This project focuses on implementing and comparing several classic AI search algorithms by applying them to a city-based ambulance pathfinding problem. The main objective is to analyze how different uninformed and informed search strategies behave in terms of speed, memory usage, and solution quality when used in a dynamic environment.

Problem Domain

The selected problem domain is Robotics and Pathfinding, modeled as a city grid where an ambulance must reach a patient as quickly as possible.

Problem Scenario

- The city is represented as a 2D grid.
- Each cell can be:
  - Road (free path)
  - Building (blocked)
  - Traffic (higher movement cost)
- The ambulance starts from a fixed location.
- The patient is located at a target cell.
- The goal is to find a path from the ambulance to the patient while minimizing time and cost.

This problem simulates real-world emergency response situations, making it suitable for evaluating different AI search strategies.

Implemented Search Algorithms

Breadth-First Search (BFS)

- How it works: BFS explores the grid level by level. It starts at the root (starting position) and explores all neighboring nodes first before moving to nodes at the next depth level.
- Advantages: Guarantees the shortest path in terms of the number of steps; simple to implement.
- Disadvantages: Memory intensive because it stores all nodes at the current depth; does not account for different costs like traffic.
- Example in our problem: The ambulance will expand all positions at distance 1 first, then distance 2, and so on until it reaches the patient.
- Best use: Small grids with uniform cost.

Depth-First Search (DFS)

- How it works: DFS explores as far as possible along one branch before backtracking.
- Advantages: Uses less memory than BFS; simple recursive implementation.
- Disadvantages: May find long, non-optimal paths; can get stuck in deep paths if the grid is large.
- Example: The ambulance keeps moving in one direction until it hits a dead end, then backtracks and tries a new path.
- Best use: Demonstration purposes, exploring all possibilities, or when memory is limited.

Uniform Cost Search (UCS)

- How it works: UCS always expands the node with the lowest path cost from the start, using a priority queue.
- Advantages: Finds the least-cost path; optimal solution.
- Disadvantages: Can be slower than BFS in grids with uniform cost; requires more memory to store the priority queue.
- Example: The ambulance prefers roads without traffic; it avoids high-cost traffic nodes unless necessary.
- Best use: Environments with variable costs like traffic, fuel consumption, or time delays.

Iterative Deepening Search (IDS)

- How it works: IDS combines DFS's low memory use with BFS completeness. It performs DFS repeatedly with increasing depth limits until it finds the goal.
- Advantages: Complete (will find a solution if it exists); low memory.
- Disadvantages: Repeats searches, so slightly slower than BFS.
- Example: The ambulance first tries paths of length 1, then 2, then 3, until it reaches the patient.
- Best use: Unknown depth of search; limited memory situations.

A Search*

- How it works: A* uses both path cost so far (g(n)) and heuristic estimate to goal (h(n)).
  It selects nodes with the smallest f(n) = g(n) + h(n).
- Heuristic Used: Manhattan Distance:

$$h(n) = |x1 - x2| + |y1 - y2|$$

- Advantages: Complete and optimal if heuristic is admissible; faster than UCS because it focuses on promising paths.
- Disadvantages: Needs a good heuristic; memory usage can be high.
- Example: The ambulance prefers routes that are both short and avoid traffic.
- Best use: Real-time pathfinding with cost-aware routes.

Hill Climbing

- How it works: Moves to the neighbor with the best heuristic value, i.e., closer to goal.
- Advantages: Very fast; uses very low memory.
- Disadvantages: Can get stuck in local maxima (good solutions but not optimal); may fail if no neighbor improves the heuristic.
- Example: The ambulance keeps moving to roads that look closer to the patient even if the path is blocked.
- Best use: Approximate solutions where speed is more important than optimality.

Genetic Algorithm

- How it works: Uses population-based search inspired by natural selection:
  - Selection: Choose better paths based on fitness (shorter distance, less traffic).
  - Crossover: Combine two paths to produce new paths.
  - Mutation: Randomly change parts of a path to explore new solutions.
- Advantages: Can solve complex optimization problems; finds near-optimal solutions.
- Disadvantages: Requires careful parameter tuning; results are probabilistic.
- Example: Each path is treated as an individual; the best paths evolve over generations to reach the patient efficiently.
- Best use: Complex optimization problems where classic search is too slow.

Implementation Details

- **Programming Language:** Python
- **Library Used:** Pygame
- **Environment:** Grid-based city simulation
- **Each algorithm is implemented as a separate function.**
- **Real-time visualization shows:**
  - Visited nodes
  - Current path
  - Final solution path
- **Time is measured during execution.**

Observations

- BFS found a solution quickly but ignored traffic cost.
- DFS sometimes found long, inefficient paths.
- UCS always found the least-cost path but was slower.
- IDS balanced memory and completeness.
- A* was the fastest and most reliable.
- Hill Climbing sometimes failed due to local maxima.
- Genetic Algorithm produced good solutions but required parameter tuning.

Comparative Analysis

Algorithm	Time	Space	Optimal	Path Cost	Notes
BFS	Medium	High	Yes (steps)	High	Ignores traffic cost
DFS	Fast	Low	No	High	Can take very long paths
UCS	Slow	High	Yes	Low	Takes traffic into account
IDS	Medium	Medium	Yes	Medium	Repeated search, memory-efficient
A*	Fast	Medium	Yes	Lowest	Uses heuristic for guidance
Hill Climbing	Very Fast	Very Low	No	Unstable	Can get stuck in local maxima
Genetic Algorithm	Variable	Medium	Near-Optimal	Low	Requires tuning, stochastic