AI Maze Generator and Solver

This project creates and solves mazes using AI algorithms. We use the A* Search method to generate smart mazes and find optimal paths. Watch a live demonstration of AI navigating through complex mazes.

By-
Abhigyat (00519051923)
Abhisaar (01119051923)
Liban (01519051923)

Introduction to Maze Generation

Generation Algorithms

- Randomized generation
- Recursive Backtracker for complex patterns

Data Structures

Heap – priority queue for A* search

Dictionary – Scores and path recommendation

List-Direction, paths, and maze grid

Set- to maintain closed list for visited nodes

AI Maze Solving Techniques

A* Search Algorithm

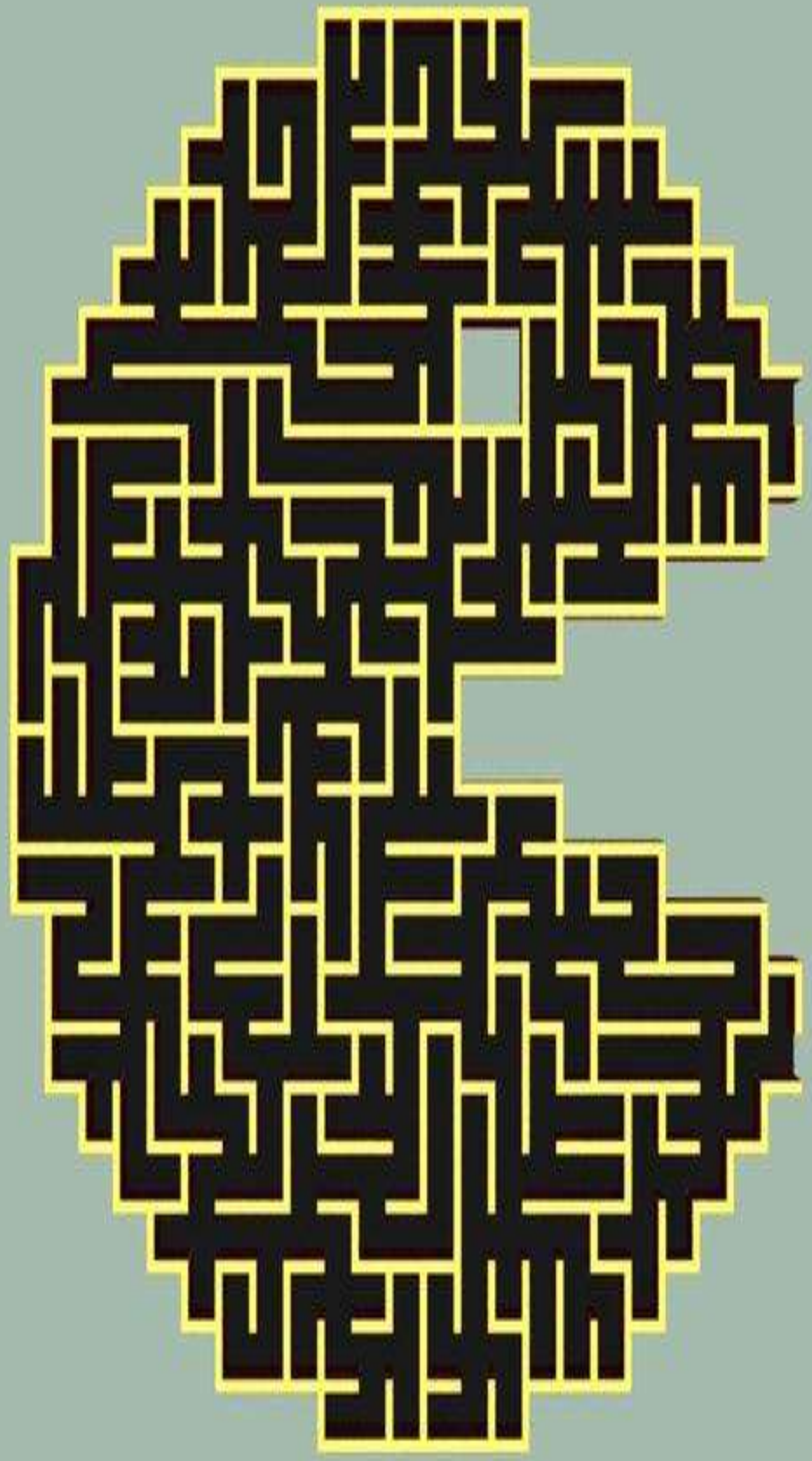
Finds shortest path efficiently with best-first approach using heuristics.

Heuristic Function

Manhattan distance guides search prioritizing close nodes to goal.

Optimization

Memory and speed improvements ensure fast, scalable solving of mazes.



P.E.A.S.

Performance Measure

•Successfully finds a path from start to goal.
•Minimizes the number of steps or time taken to solve the maze.

Environment

•The maze grid (2D array) with walls and open cells.
•Start and goal positions within the maze.

Actuators

•The agent’s ability to move in the maze (up, down, left, right).
•Marking cells as visited or part of the path.

Sensors

•Reading the current state of the maze (walls, open paths).
•Detecting the agent’s current position.

Implementation Details

Programming Language

Python for flexible algorithm development and rapid prototyping.

Key Libraries

NumPy, MatPlotLib, heapq, imageio

System Architecture

Modular to allow easy switching between maze algorithms.

Results and Performance Metrics

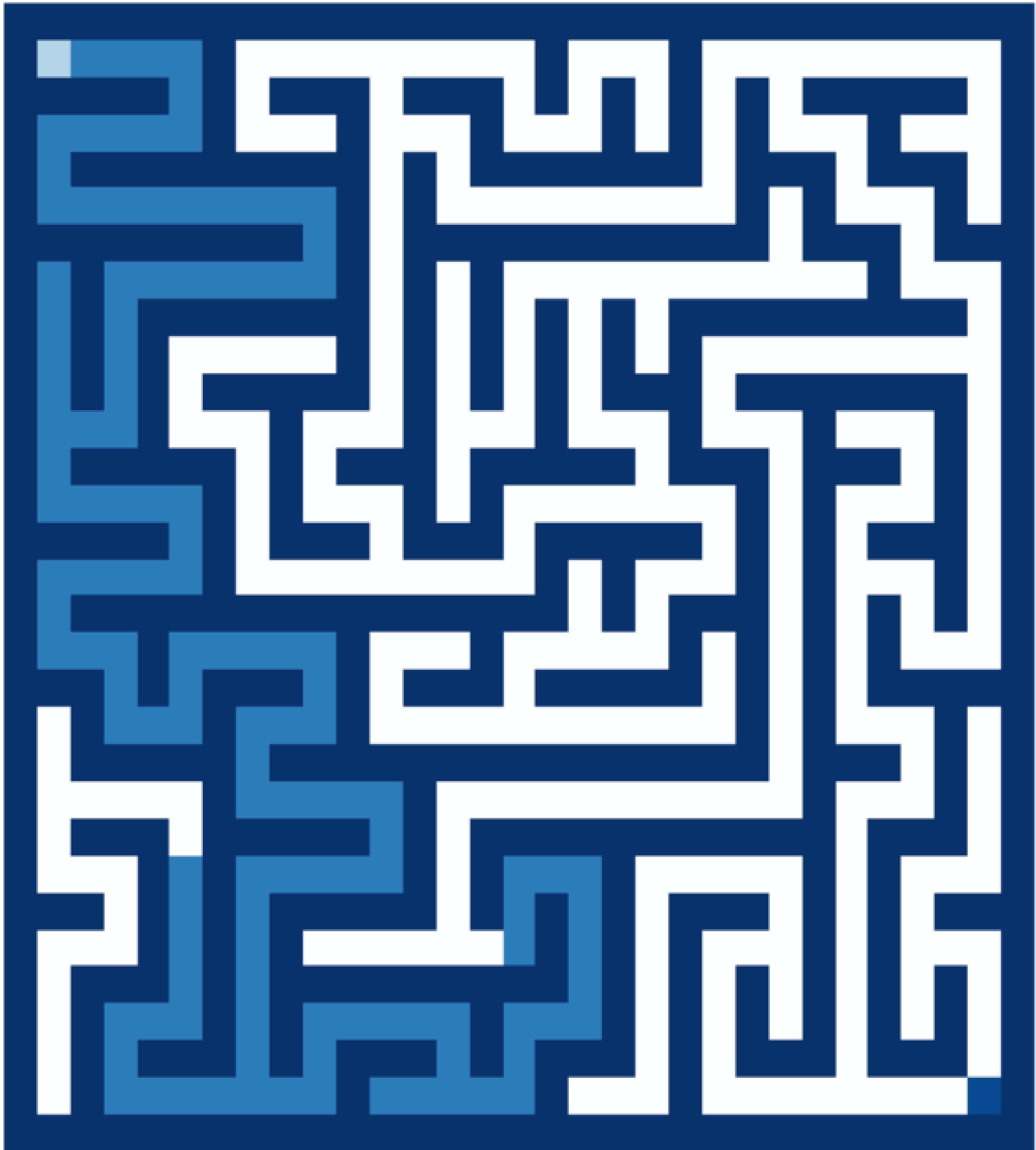
Maze Size	50x50 (can change)
Generation Speed	1 min on avg.
Solving Efficiency	A* solves optimally and is very optimal for mazes up to 100x100
Algorithm Benchmark	Outperforms DFS in path cost and speed

Visualization and User Interface

Each frame show the progress

Colors:
White(1):wall
Gray(0.7):Visited Nodes
Light Gray(0.3):Current Path
Start(0.1) and Goal(0.9) marked distinctly

Main Function Flow:
1.Takes user input for width & height
2.Generates an odd-sized maze
3.Finds path using A*
4.Saves a GIF visualizing the pathfinding



Challenges and Limitations

High Memory Use

Large mazes (>100x100) consume significant RAM.

Algorithm Limits

A* can struggle in highly complex maze layouts.

Future Improvements

Plan to add parallel processing and better heuristics.

Conclusion and Future Directions

The A* algorithm is a powerful and widely used pathfinding and graph traversal technique, combining the strengths of Dijkstra’s Algorithm and Greedy Best-First Search

By using heuristics along with cost functions, A* efficiently finds the shortest path in various applications such as robotics, game development, and AI planning.

Future work can focus on enhancing the A* algorithm for real-time application, large-scale maps, and dynamic environments.

.