AI Path Planner :Multi-Algorithm Grid-Based Search Visualization

CMPT 310 – D100 Introduction to Artificial Intelligence (Fall 2025)

Overview

This guide explains how to build and run a multi-algorithm **AI Path Planner** that visualizes and compares classical AI search algorithms : **Breadth-First Search (BFS)**, **Uniform Cost Search (UCS)**, and **A*** (Manhattan & Euclidean).
It generates a randomized 20 × 20 grid world, computes shortest paths, and evaluates execution time, path cost, and explored nodes through animated visuals and charts.

Environment Setup

Requirements:

Python 3.10+

pip install numpy matplotlib

Running:

cd src

python main.py

Outputs:

Outputs are directly stored in →

results/figures/ → performance_*.png, heuristics_*.png

results/logs/ → simultaneous_*.json

Project Structure

File	Purpose
grid_generator.py	Generates 20×20 grid with adjustable obstacle probability
bfs.py, ucs.py, astar.py	Core search algorithm implementations
visualize.py	Displays animated grid and path visualizations
main.py	Runs all algorithms and creates comparison graphs
results/	Contains output charts and JSON logs

Step-by-Step Workflow

Step 1 -Grid Generation

Creates a NumPy matrix (0 = free, 1 = obstacle) with start (0,0) and goal (19,19).

Step 2 - Algorithm Execution

Each algorithm is run independently to search for the shortest path using its own strategy.

Step 3 - Visualization

Explored nodes appear in light blue; final path is drawn in yellow.
Four subplots show BFS, UCS, A*(Manhattan), and A*(Euclidean) side by side.

Step 4 - Evaluation & Logging

Metrics recorded for each algorithm: execution time, nodes explored, path length, and cost.

Step 5 - Performance Graphs

Bar charts compare search efficiency and heuristic accuracy.
All plots auto-saved in results/figures/.

Goal-Conditioned Reinforcement Learning

1. Interactive Goal Selection: Users click on grid cells to choose multiple pickup locations. Invalid selections (obstacles) are automatically rejected.

2. Goal-Conditioned Q-Learning Training : The agent learns to reach selected goals using reinforcement learning.

State representation: (position_x, position_y, goal_x, goal_y)

Training setup:

- 4 segments × 600 episodes = 2400 total episodes
- ε-greedy policy with gradual decay
- Tracks success rate and average reward per 100 episodes
- Converges to ≈ 100 % success per segment

3. RL vs A* Comparison : After training, both algorithms navigate the same multi-goal route
Takeaway: The RL agent achieves fast, low-exploration navigation once trained, while A* remains more path-optimal.

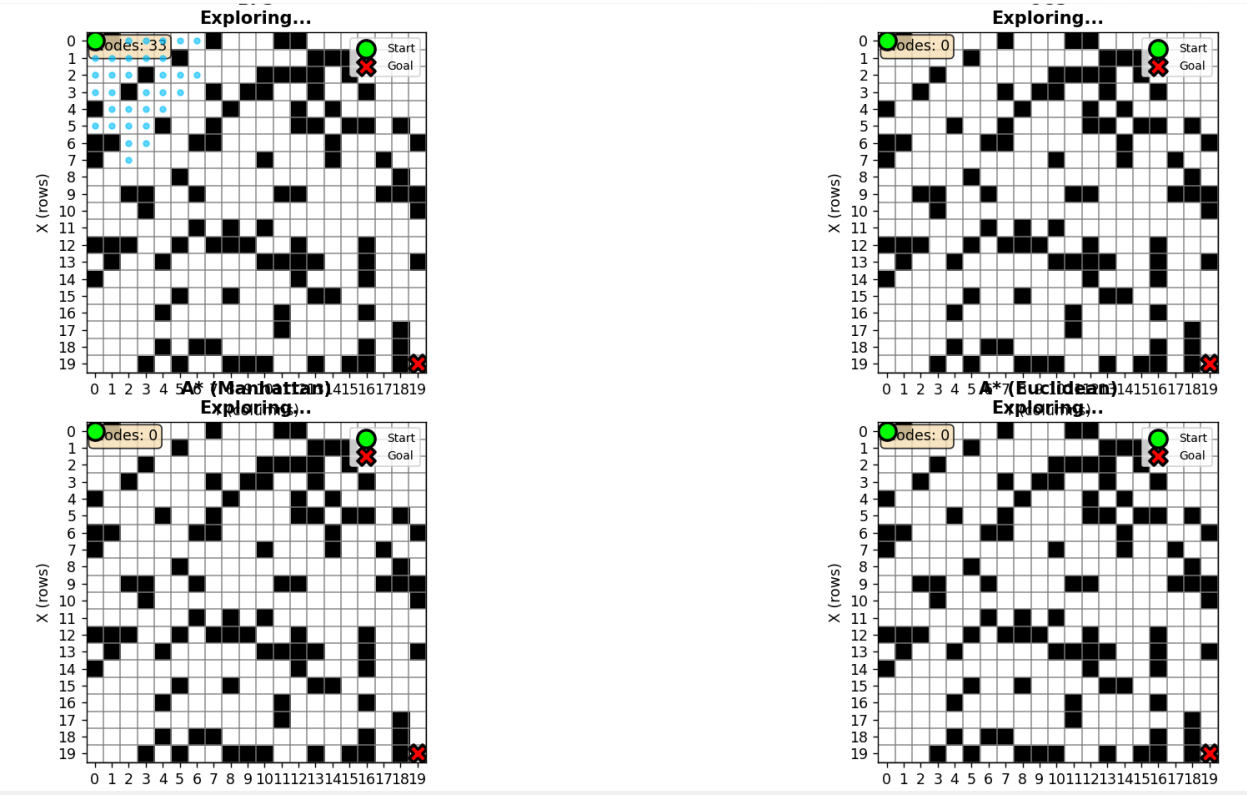
Common Errors & Fixes

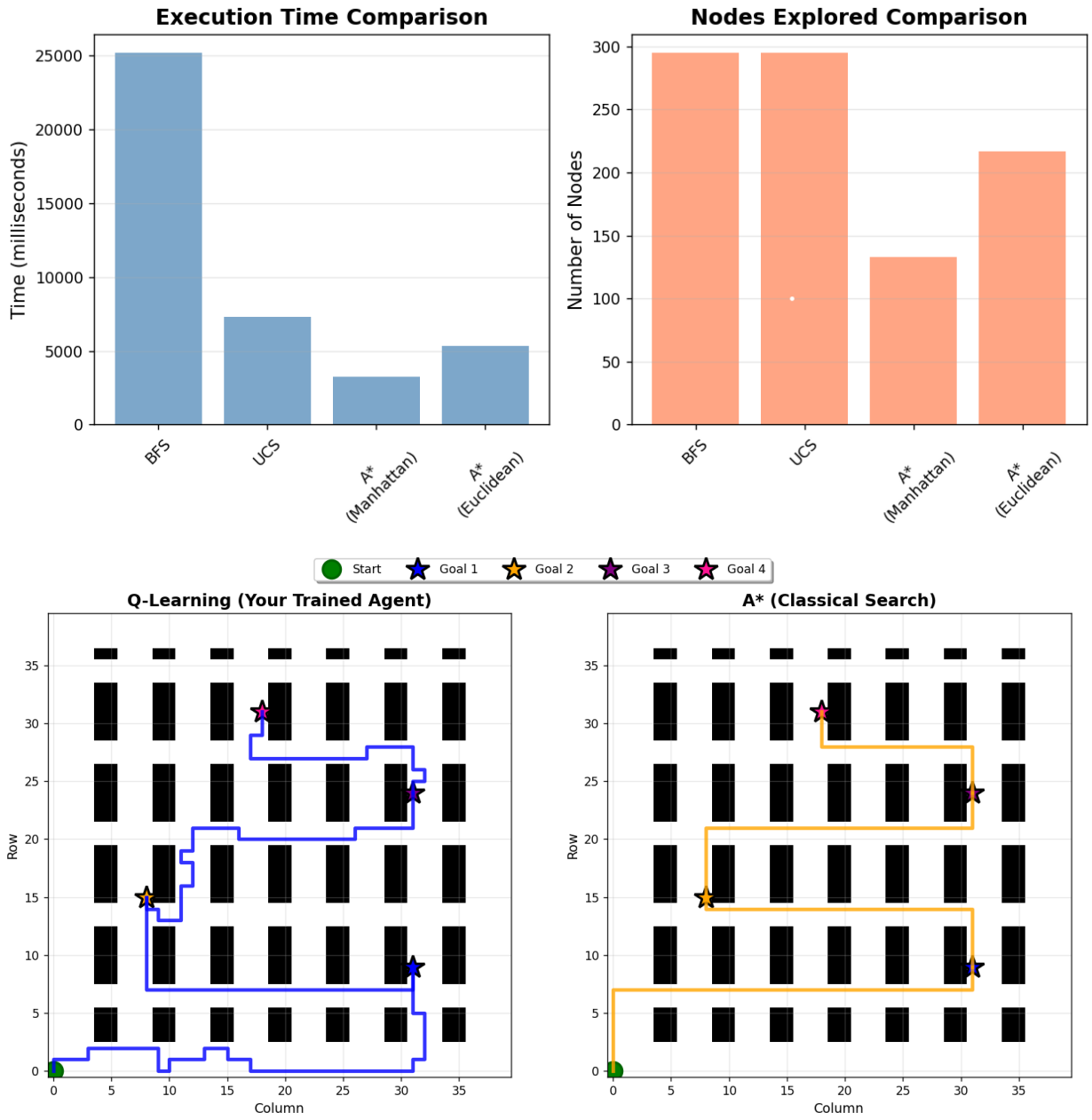
Error	Cause	Fix
ModuleNotFoundError: matplotlib	Missing dependency	pip install matplotlib
Blank window	GUI backend issue	Run from VS Code or disable animation

Troubleshooting Tips

- Reduce obstacle probability if no path is found.
- Increase delay (0.05 s) for slower animations.
- Use different random seeds for varied maps.

Visual Examples





License

This guide is produced for SFU CMPT 310 (Fall 2025). You retain copyright; SFU may share for academic use.