CSE518: Artificial Intelligence Final Project Report

Group-8
Member 1: Pratham Sandesara (AU2340239)
Member 2: Dharman Patel (AU2340237)

Abstract

This project presents an autonomous planetary rover navigation system that combines deliberative planning with reactive survival behaviors. The rover operates on a 20×20 terrain grid containing varied surfaces, hazards, traps, cliffs, and recharge stations. Using an A* search planner with four heuristics—Manhattan, Euclidean, Squared Manhattan, and Logarithmic—the rover computes optimal and safe routes while adapting to changing conditions. A reflex module manages real-time decisions such as hazard avoidance, backtracking, and battery-aware rerouting. The system is visualized through a Tkinter-based GUI that displays the terrain, rover path, live status, and decision logs. Overall, the project demonstrates a robust and adaptive navigation architecture capable of safe traversal in uncertain and dangerous environments.

1 Methodology – A* Implementation

The A* search algorithm forms the core of the rover's deliberative navigation system. It computes an energy-aware, hazard-aware, and battery-feasible path using a combination of cost accumulation and heuristic distance. The methodology used for integrating A* into the rover architecture is described below.

State Representation

Each grid cell is modeled as a state with the following attributes:

- Position (x, y) on the 20×20 grid.
- Terrain type with an associated movement cost.
- Cumulative cost $g(n)$ from the start node.
- Heuristic estimate $h(n)$ chosen from four available heuristics: Manhattan, Euclidean, Squared Manhattan, and Logarithmic.

Cost Function

Movement cost strictly depends on terrain:

$$g(n) = \sum \text{terrain\_cost}(cell_i) \qquad \text{flat} = 5,\ \text{sandy} = 10,\ \text{trap} = 20,\ \text{hazardous} = 25.$$

Cliffs and rocky regions are modeled as `IMPASSABLE` and are never expanded.

Evaluation Function

The planner computes:

$$f(n) = g(n) + h(n)$$

where $h(n)$ is selected by the user in real time. This allows the rover to switch between optimal, aggressive, or exploratory behaviors.

Hazard-Aware Node Expansion

The A* implementation is modified to account for discovered dangers:

- All nodes in `known_hazards` are skipped and not expanded.
- This effectively allows the rover to **learn** the map and avoid past mistakes.
- Trap and hazardous tiles become dynamic obstacles after being stepped on once.

Battery-Feasible Path Validation

The rover only accepts a path if it is reachable according to battery remaining. If the goal is unreachable with current battery, A* is re-run toward recharge stations instead.

Recharge-Oriented A* Planning

When planning for recharging:

1. A* computes cost and path to all recharge stations.
2. Only stations reachable with current battery are considered.
3. From these, the rover selects the station with minimum Euclidean distance to the final goal.
4. This ensures the rover recharges at a strategically optimal location.

Integration With Reflex Layer

A* planning is invoked *only after* reflex checks:

- Low battery override,
- Trap/hazard detection and immediate backtracking,
- Recharge-in-place behavior.

This ensures safety overrides planning whenever necessary.

Overall Method

The combined methodology results in:

- Terrain-cost–based path planning,
- Hazard-adaptive and memory-based search,
- Battery-constrained decision-making,
- Heuristic-driven navigation flexibility.

Together, these components make the rover capable of safe, optimal, and adaptive traversal in a dangerous environment.

2 Heuristics Implemented for A* Search

Four distinct heuristics were implemented to test how different estimation strategies influence A* performance. Each heuristic provides a different way to "guess" the remaining cost to the goal.

2.1 Manhattan Distance

Calculates the cost based on grid steps (up, down, left, right), which is ideal for 4-directional movement.

$$h_M = 5 \times (|x_1 - x_2| + |y_1 - y_2|)$$

2.2 Euclidean Distance

Calculates the straight-line "as-the-crow-flies" distance. It is admissible and provides a direct path estimate.

$$h_E = 5 \times \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

2.3 Squared Manhattan

A non-admissible, aggressive heuristic that squares the Manhattan distance. This strongly prioritizes nodes that are closer to the goal, often leading to a faster (but not always cheapest) solution.

$$h_S = 5 \times (|x_1 - x_2| + |y_1 - y_2|)^2$$

2.4 Logarithmic Heuristic

A custom, admissible heuristic with a slow-growing function. It encourages broader exploration, as the cost difference between nearby and far-away nodes is less extreme.

$$h_L = 5 \times \log_2(1 + |x_1 - x_2| + |y_1 - y_2|)$$

3 Heuristic Comparison and Observations

A dedicated `Compare Heuristics` module was implemented to evaluate the performance of all four heuristics under identical map conditions. When activated, the system computes the A* path for each heuristic and overlays them simultaneously on the grid using distinct colors. The GUI legend clearly identifies the path corresponding to each heuristic.

Heuristic	**Behavior**	**Computation**	**Best Scenario**
Manhattan	Grid-aligned and stable; produces predictable paths	Linear grid distance	Regular 4-way grids
Euclidean	Smooth convergence toward goal	Straight-line distance	Open or sparse maps
Squared Manhattan	Highly goal-driven; commits early	Squared grid distance	Fast but slightly costlier routes
Logarithmic	Broad early exploration; BFS-like expansion	Logarithmic scaling	Hazard-heavy or irregular grids

Observations

- **Manhattan and Euclidean** consistently produced the most reliable and cost-efficient routes. In 4-directional movement, both yielded nearly identical paths.
- **Squared Manhattan** explored fewer nodes and reached the goal quickly, but its aggressive behavior sometimes resulted in higher energy costs.
- **Logarithmic** was the most explorative, expanding a wide search area before committing to a path. While slower, it produced safer long-range paths in maps with hazards and dead-ends.

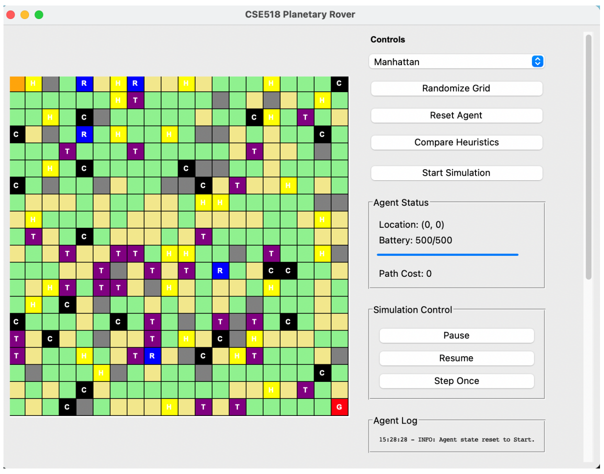
Overall, Manhattan provided the best balance between speed, cost, and predictability for general-purpose rover navigation.

4 Graphical User Interface

The GUI is built using `Tkinter` and provides a full simulation environment:

- A scrollable control panel with all simulation options.
- A dynamic, color-coded terrain map visualization.
- A live status panel showing rover location and battery level (with progress bar).
- A real-time agent log showing rover decisions, warnings, and path plans.
- Simulation controls: Pause, Resume, and Step-Once for debugging.
- The "Compare Heuristics" button for the analysis module.
- A speed adjustment slider to control the simulation's update delay.

Demonstration

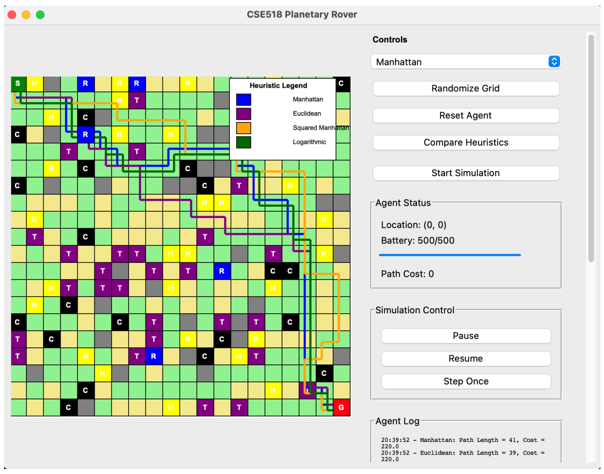


Manhattan

Location: (0, 0)
Battery: 500/500
Path Cost: 0

```
15:28:28 - INFO: Agent state reset to Start.
```

Figure 1: GUI View – Terrain Map and Controls



🟦	Manhattan
🟪	Euclidean
🟧	Squared Manhattan
🟩	Logarithmic

Manhattan

Location: (0, 0)
Battery: 500/500
Path Cost: 0

```
20:39:52 - Manhattan: Path Length = 41, Cost =
220.0
20:39:52 - Euclidean: Path Length = 39, Cost =
220.0
```

Figure 2: Compare Heuristics Feature

5 Conclusion

The project successfully delivers an autonomous rover navigation system that combines reactive safety mechanisms with deliberative A* planning. By integrating multiple heuristics, terrain-aware cost modeling, and real-time reflexes, the rover is able to navigate a complex 20×20 environment containing hazards, traps, cliffs, and recharge stations. Features such as dynamic battery management, hazard memory, and automatic backtracking significantly improve the agent's survivability and decision accuracy. Overall, the system demonstrates reliable, adaptive, and safe navigation in a partially hazardous environment, meeting all functional and performance objectives of the project.

6 Work Done

Name	**Work Done**
Pratham Sandesara	Reflex rules, battery logic, hazard handling, GUI layout, controls, sidebar, legend, heuristic testing, path tuning, recharge logic, logs, speed control, report, slides
Dharman Patel	A* algorithm, rover animation, hazard memory, impassable tiles, heuristic overlay, result logs, recharge planning, traps and cliffs, report, slides

Table 1: Team Contribution Summary

References

[1] Peter Norvig et al., *AIMA-Python: Python Implementation of Algorithms from "Artificial Intelligence: A Modern Approach"*, GitHub Repository.
Available at: https://github.com/aimacode/aima-python Accessed: 18 November 2025.