Final Semester Report: Comparing Search and Learning Methods in a Grid Race Environment

Qais Al Ramahi Diego Garcia Muhammed Goda

April 2025

Abstract

This report presents our study of several agent decision-making methods: breadth-first search (BFS), depth-first search (DFS), heuristic search (A*), minimax adversarial planning, and model-free reinforcement learning (Q-learning) - all applied to a common grid-based racing environment with resource constraints. We describe the implementation details, analyze experimental results over hundreds of runs, and discuss tradeoffs in optimality, computational cost, and adaptability. We also decided to add a bonus Q-learning component, discussing its pros and cons and also reflected on the challenges of the adversarial Police chase scenario. Limitations and directions for future work are outlined too. The code to this project can be found on `https://github.com/qaisalramahi/AI-Course-Spring-Semester-2025`

1 Introduction

The goal of this project was to apply classical and modern AI techniques to a simple racing problem. We built an OpenAI Gym environment with a Pygame-based visualizer to support rapid experimentation in a consistent framework. A car agent starts at the top-left, must reach the bottom-right goal, and navigates a sparse network of drivable tiles. A finite gas supply enforces a resource constraint that refills at a few gas stations and depletes each move. This design provides a clear testbed for comparing:

- Blind search (BFS, DFS)
- Heuristic search (A*)
- Adversarial planning (minimax)
- Reinforcement learning (Q-learning as a bonus)

We evaluated each across 100 runs, measuring path length, planning/training time, and execution time. Our contributions include a gas-aware admissible heuristic for A*, a minimax Police adversary variant, and Q-learning with reward shaping. This report synthesizes our insights.

2 Environment and Framework

2.1 Grid and Resources

The environment is a 15×15 grid. Cells belong either to *PATH_TILES*, which are drivable, or "grass" (illegal). The car has a gas tank of capacity $G_{max} = 20$. Each move consumes one unit, unless landing on a gas tile which refills to full. Running out of gas or leaving the drivable area yields a large penalty or terminates the run.

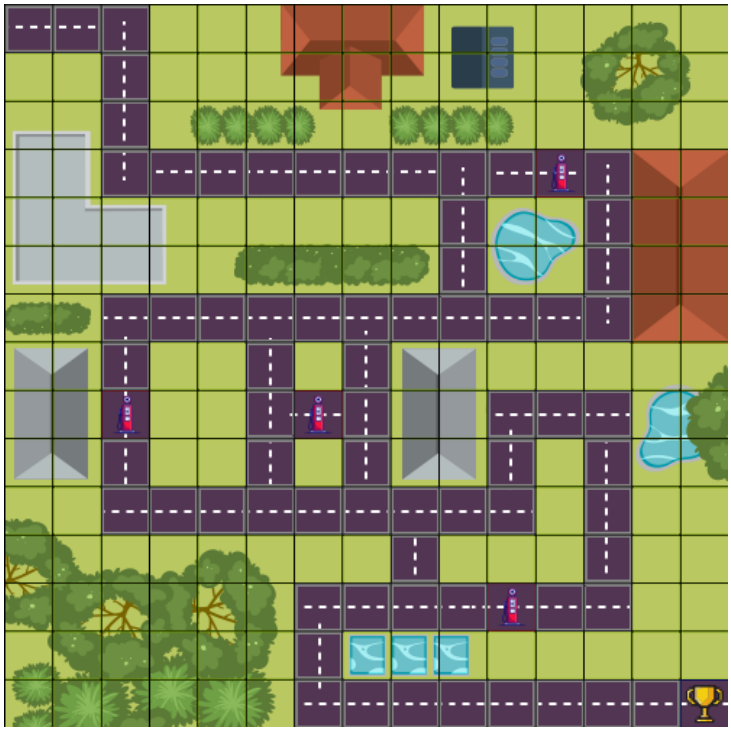

Figure 1
Shows the environment layout we used with trees, grass and a drivable road. Some cells also contain gas tanks.

2.2 Implementation Details

We extended OpenAI Gym's `Env` class for consistency with RL and planning libraries. Observations encode free vs. gas vs. agent positions, while Pygame renders background, car/Police sprites, gas level, and distance-to-goal metrics. Each algorithm is implemented in its own module exposing `solve(env)` that returns a sequence of actions.

3 Blind Search Methods

3.1 Breadth-first Search (BFS)

BFS systematically explores the state space level by level. Pseudocode:

1. Initialize a FIFO queue with $(start, [])$.
2. While queue not empty:
   - Dequeue $(s, path)$.
   - If $s = goal$, return $path$.
   - For each neighbor s' of s on PATH_TILES with valid gas:
     - Enqueue $(s', path + [a])$ and track visited.

BFS finds shortest path in terms of steps under uniform cost but explores many nodes. Memory usage grows with frontier size.

3.2 Depth-first Search (DFS)

DFS uses a stack to dive into one branch until no further moves, then backtracks. It has lower memory overhead but no optimality guarantees. DFS pseudocode mirrors BFS except using LIFO. Revisit avoidance prevents infinite loops.

3.3 Experimental Results: BFS vs. DFS

Table 1 summarizes average path lengths and timings.

Algorithm	Path (steps)	Plan Time (ms)	Exec Time (ms)
BFS	46.0	12.6	14.2
DFS	52.0	1.6	2.4

Table 1
Blind search performance over 100 runs.

BFS guarantees shorter paths but is halved in speed compared to DFS. DFS is extremely fast to plan but meanders extensively.

4 Heuristic Search: A*

4.1 Admissible Heuristic

We define the heuristic $h(p, g, G)$ as:

$$h(p,g,G) = \begin{cases} d_{man}(p,g), & G \geq d_{man}(p,g) \\ d_{man}(p,g) + \lambda \min_{s \in S_{gas}} d_{man}(p,s), & \text{otherwise} \end{cases} \tag{1}$$

where d_{man} is Manhattan distance, S_{gas} the set of gas stations, and $\lambda = 0.5$ ensures admissibility ($h \leq$ true cost). When gas suffices, it reduces to pure Manhattan.

4.2 Algorithm

Use a min-heap keyed by $f(n) = g(n) + h(n)$, track best g-scores for $(position, gas)$ states, and reconstruct the path upon reaching the goal.

4.3 Results

Algorithm	Path (steps)	Plan Time (ms)	Exec Time (ms)
A*	38.0	14.6	5.4

Table 2
A* performance over 100 runs.

A* achieved the shortest paths by leveraging domain knowledge in h. Its planning time is higher than DFS but yields a huge quality boost.

5 Adversarial Planning with Minimax

5.1 Problem Setup

We introduce a Police agent as an adversary. Turns alternate car vs. Police. The Police can initially spawn on any PATH_TILE (chosen randomly) and is allowed to drive off-road onto grass trying to mimic a potentially crazy pursuit of the "wanted" car similar to real life police chases.

5.2 Minimax with Alpha-Beta Pruning

For each state $s = (p_c, p_p, turn)$, we compute:

$$V(s) = \max_{a_c} \Big(R(s, a_c) + V(s') \Big) \qquad \text{if car's turn} \tag{2}$$

$$V(s) = \min_{a_p} \Big(R(s, a_p) + V(s') \Big) \qquad \text{if Police's turn} \tag{3}$$

with alpha-beta pruning. The evaluation function R includes:

- +10000 for reaching goal, -5000 for collision.
- $-30\times$ Manhattan distance to goal.
- $-50\times$ proximity penalty if Police within 4 tiles.
- +10 per extra escape action.

5.3 Behavioural Observations

Empirically, the Police almost always catches the car within 20 moves. Figure 2 shows a typical chase where the car is cornered by the one-tile-wide road. Because the car has limited branching in a narrow corridor, evasion is nearly impossible once the Police closes in. Allowing off-road pursuit worsens the bias. This is discussed in §6.

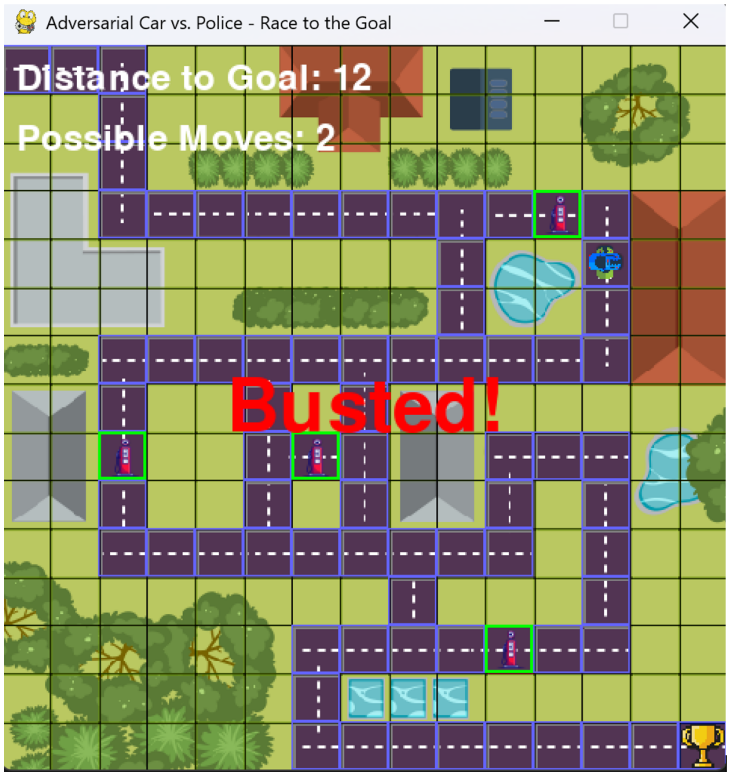

Figure 2
Typical Police chase outcome where blue car is the police and green car is our main character

6 Reinforcement Learning: Q-learning

6.1 Algorithm Details

We implemented tabular Q-learning over states (x, y, gas) and actions in {Left,Right,Up,Down}. The update:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\big[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\big] \tag{4}$$

with $\alpha = 0.1$, $\gamma = 0.99$, and an ϵ-greedy exploration decaying from $1.0 \rightarrow 0.01$ over 5000 episodes. We add shaping reward $0.2\Delta d_{man}$ per step.

6.2 Training vs. Execution

Algorithm	Path (steps)	Train Time (ms)	Exec Time (ms)
Q-learning	38.0	56600	7.7

Table 3
Q-learning performance (100 runs).

Training required tens of seconds (56.6s) to fill the Q-table; test-time lookup is fast. After extensive sampling, Q-learning matches A* path length on this deterministic map.

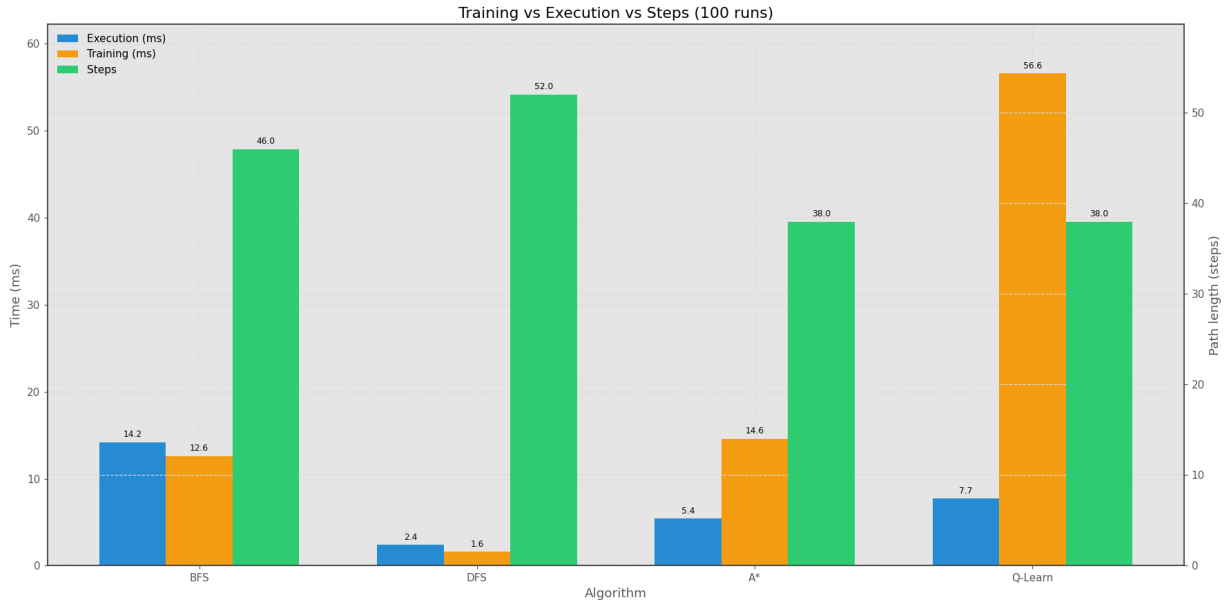

Figure 3
A graphical comparison of all algorithms taking into account three factors: training time, execution time and number of steps each algorithm took

7 Discussion

Our experiments highlight:

- **Model-based planning** (A*) is highly sample-efficient and optimal in known, deterministic environments. A high-quality heuristic yields a 17% shorter path than BFS. However, it is worth noting that had this heuristic function been poor then the performance of the

A* algorithm would have been severly diminished - performing almost the same as BFS. That is why having a gas tank-influenced, robust heuristic function here proved pivotal in the performance of the algorithm.

- **Blind search** trades off optimality for speed: DFS is fastest to generate suboptimal solutions; BFS guarantees mid-quality at moderate cost.
- **Reinforcement learning** can learn optimal policies given enough data, but is *sample-inefficient*. Q-learning needed millions of updates to catch up to A*'s one-shot planning. In this use case, the Q-learning algorithm can be seen as overkill and more useful in more stochastic environments.
- **Adversarial search** exposes limitations of narrow corridors; richer map topology or road widening would allow more meaningful evasion. But what we could clearly see is an initial attempt in a 1 vs 1 scenario where the both sides are trying to maximize their own winnings in favour of the other's.

8 Limitations and Future Work

1. The Police car's off-road ability heavily biases minimax. Future work could restrict it to only moving in PATH_TILES but for this to happen, we would need to revamp the map and create a new and bigger one which would in turn increase the complexity of the project.

2. Single start-goal tests may overstate Q-learning's utility; randomizing initial states would better evaluate generalization.

3. Include Pro-log in our project to implement logical decisions by the agent.

9 Conclusion

We demonstrated a spectrum of AI techniques on a unified task. A* outperformed others in combined speed and path quality, while Q-learning required heavy upfront cost to match it. Adversarial planning highlighted design tradeoffs in map geometry and agent capabilities. This project solidified our understanding of different traditional and more modern approaches to agentic decision-making and use cases in a "simple", beginner-friendly example.