# Risk vs. Reward: A Comparative Analysis of Reinforcement Learning Strategies in the Stochastic Racetrack Problem

**Anthony (AJ) Mann** ANTHONYSTEVENMANN@GMAIL.COM
*Artificial Intelligence Student*
*Montana State University*
*Bozeman, MT 59718, USA*

**Jackson Thorn** JTHORN2448@GMAIL.COM
*Artificial Intelligence Student*
*Montana State University*
*Bozeman, MT 59718, USA*

## Abstract

This paper aims to explore the application of model-based and model-free reinforcement learning algorithms to the stochastic racetrack problem. Value Iteration is implemented to compute the theoretical optimal policy using a known transition model, while Q-Learning and SARSA are employed to approximate optimal policies through temporal difference learning without prior knowledge of the environment. The algorithms are evaluated across three distinct tracks under two crash-penalty scenarios: resetting to the nearest valid position (NRST) and resetting to the starting line (STRT). We found that Value Iteration successfully converged to the mathematically optimal trajectory, Q-Learning converged to more aggressive paths, and SARSA favored safer central trajectories.

## 1 Introduction

### 1.1 Stochastic Racetrack Problem

The racetrack problem is a stochastic control challenge used to evaluate the decision making process of reinforcement learning agents under uncertainty. The goal is to navigate a vehicle from the start of the track to the end of the track in the minimum number of time steps. The agent must adhere to the boundaries of the track, of which there are three: the 'W' track, which is essentially a straight shot to the finish line with two dead-end branches, the 'U' track which is shaped like the letter U, and the '2' track which is shaped like the number two. There are two penalty mechanisms if the agent collides with a track wall: NRST is a "soft" crash that resets the agent to the nearest valid track position and STRT is a "hard" crash that forces the agent to restart from the beginning of the track, while retaining knowledge of the previous run. These constraints require the algorithms not only to find the shortest path, but to account for risk.

### 1.2 Algorithms

To address this control problem, we implemented three distinct reinforcement learning algorithms. Value iteration acts as a model-based baseline - it utilizes dynamic programming and full knowledge of the transition probabilities to solve the Bellman optimality equation, guaranteeing convergence to the global optimum. Q-Learning represents a model-free, off-policy approach. It estimate the optimal action-value function by updating Q-values based on the maximum potential future reward, allowing the agent to learn the optimal policy regardless of its exploration strategy. Finally, SARSA is a model-free, on-policy alternative. Unlike Q-learning, it updates value estimates based on

the action actually executed by the current policy, thereby incorporating the risk associated with exploration steps into the learned values.

### 1.3 Hypothesis

We hypothesize that Value Iteration will produce the absolute shortest paths, serving as the ground truth for performance due to its access to the complete environment model. As for Q-Learning, we expect it to converge to a trajectory similar to Value iteration, prioritizing a more aggressive pathing approach despite the risk of collision. In contrast, we hypothesize that SARSA will account for the risk of stochastic failure and exploration into its value estimates, adhering to the center of the track and resulting in a longer path.

## 2 Experimental Approach

Each algorithm was evaluated on all three tracks. For every track and crash-handling choice, we ran the algorithm ten times, resulting in 20 experiments per algorithm per track. During each run, we recorded the length of the resulting optimal path, the number of learning episodes executed, and the corresponding learning curve. Start states were selected randomly by first enumerating all valid starting positions and then sampling uniformly from that set. Agent velocities were constrained to the range $[-5, 5]$ for both axes, and actions were limited to accelerations of $-1, 0$, or 1 in each direction

A stochastic element was also present in the environment: with a 20% probability, the agent failed to accelerate regardless of the chosen action. This stochasticity was incorporated directly into the transition model (for Value Iteration) and the environment dynamics used during learning (for Q-learning and SARSA), and is reflected in the results presented in the section below.

### 2.1 Value Iteration

We applied value iteration to compute an optimal policy for each racetrack. The state space consists of all valid $(x, y)$ positions combined with all valid velocity pairs $(v_x, v_y)$. For each state, we iteratively applied the Bellman update using the minimum expected cost over all actions, incorporating a discount factor of $\gamma = 0.99$. Value iteration was initialized with $V(s) = 0$ for all states and was executed until convergence, which is defined as the first iteration where the maximum change $\delta$ across all state values fell below a threshold of $\theta = 0.000001$.

### 2.2 Q-Learning

We implemented Q-Learning to approximate the optimal action-value function $Q*(s, a)$ using an off-policy temporal difference approach. The Q-table was initialized with $Q(s, a) = 0$ for all state-action pairs; given the negative reward structure (-1 per step, minimization), this served as an optimistic initialization to encourage early exploration. We employed an $\epsilon$-greedy exploration strategy, where $\epsilon$ decayed exponentially from 0.5 to a minimum of 0.01 over the course of training, shifting behavior from exploration to exploitation. The learning rate $\alpha$ was similarly decayed to ensure stability as the agent converged. The Q-values were updated using a discount factor of $\gamma = 0.9$ according to the standard update rule, which targets the maximum estimated value fo the next state regardless of the action actually selected by the behavior policy.

### 2.3 SARSA

We applied SARSA to learn the action-value function in an on-policy manner. Unlike Q-Learning, SARSA updates its value estimates based on the actual next action chosen by the agent's current policy, rather than the theoretical maximum. This distinction forces the algorithm to account for the randomness of the exploration strategy during the learning process. We utilized the same state representation and initialization as in Q-Learning. The algorithm was executed using a decaying $\epsilon$-greedy policy and a discount factor of $\gamma = 0.9$. SARSA is expected to learn more conservative policies that avoid states where random actions could lead to high-penalty collisions.

### 2.4 Extracting the Optimal Path

After training or policy computation, the optimal policy for each algorithm was used to generate trajectories. The policy acts as a mapping from each state to the action the agent should take. Starting from a randomly selected start state, the agent repeatedly executed the action prescribed by the policy and transitioned to the next state according to the environment dynamics. This process continued until a terminal state at the finish line was reached, producing a sequence of visited states that constitutes the optimal path. If an action would result in leaving the track or colliding with a wall or corner, the corresponding crash rule was applied to update the agent's position and velocity accordingly.

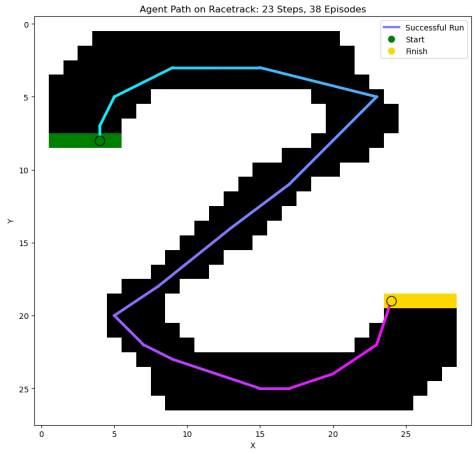
### 2.5 Crash Rules

During updates, we monitored invalid actions, including leaving the track, colliding with walls, or clipping corners. Corner collisions were detected using Bresenham's line algorithm, which checks all cells along the line connecting the agent's previous and next positions. Any of these invalid actions resulted in a crash. Crashes were handled according to the chosen mode: $STRT$ returned the agent to the designated start state, while $NRST$ returned the agent to the nearest legal cell. In both cases, the agent's velocity was reset to zero. Crossing the finish line overrides any collisions or crashes, ensuring the agent successfully completes the track regardless of its trajectory immediately prior.

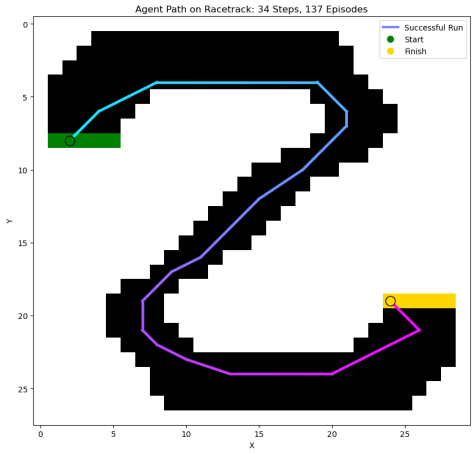
## 3 Results

The following plots are representative examples from one of the ten test runs performed for each crash-handling condition.

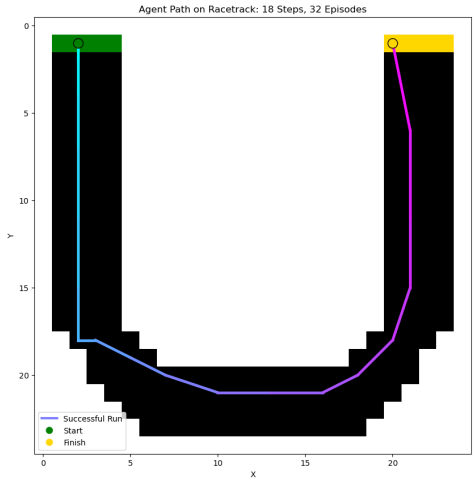
### 3.1 Value Iteration



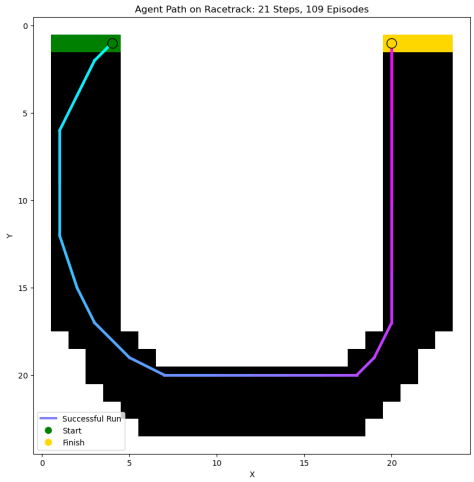

(a) 2-Track (NRST)



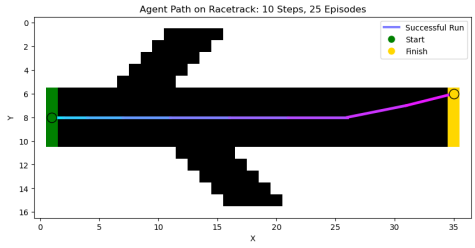

(b) 2-Track (STRT)



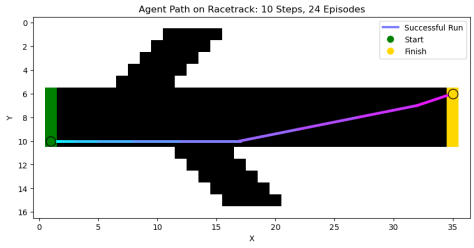

(c) U-Track (NRST)




(d) U-Track (STRT)



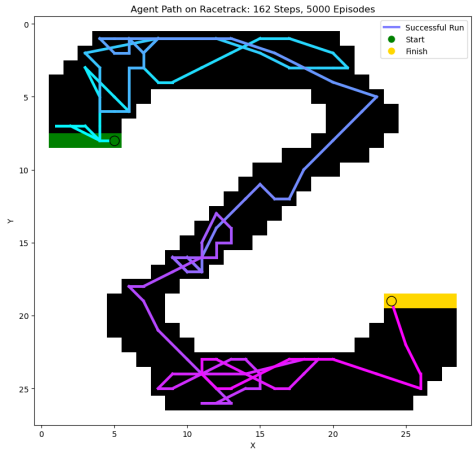

(e) W-Track (NRST)



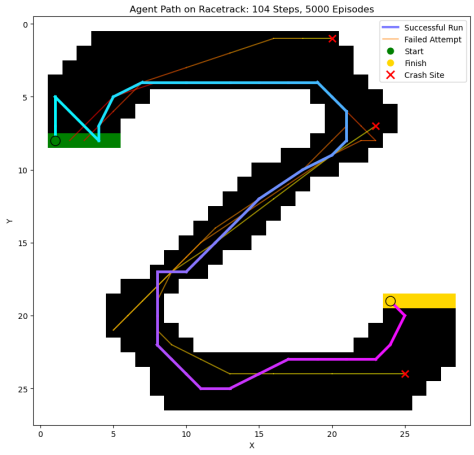

(f) W-Track (STRT)

Figure 1: Value Iteration Results. The left column displays the Nearest Position (NRST) penalty, while the right column displays the Restart (STRT) penalty.

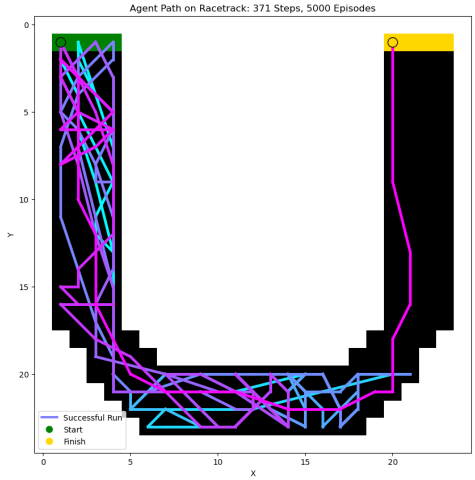
### 3.2 Q-Learning



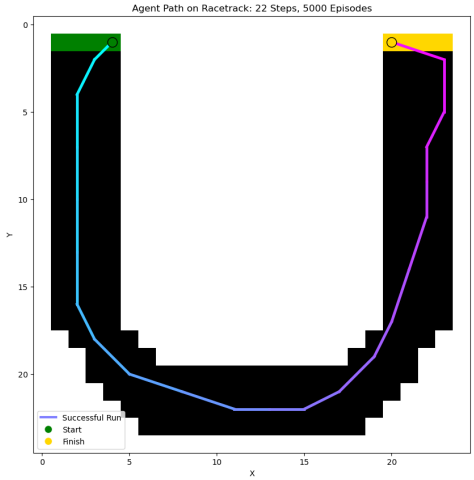

(a) 2-Track (NRST)



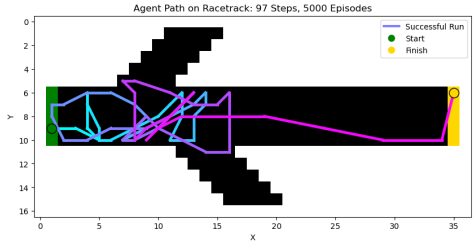

(b) 2-Track (STRT)



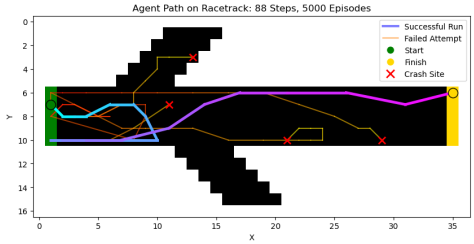

(c) U-Track (NRST)




(d) U-Track (STRT)



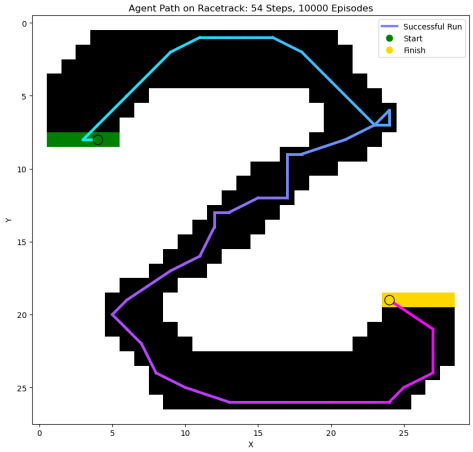

(e) W-Track (NRST)



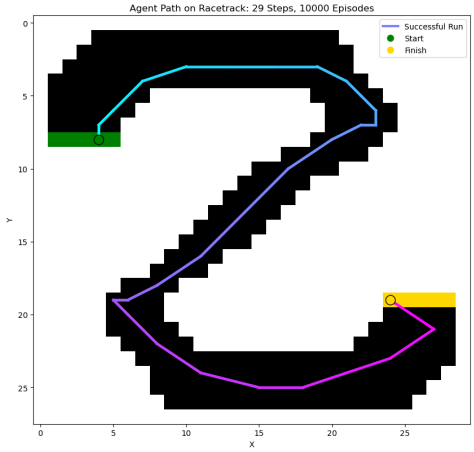

(f) W-Track (STRT)

Figure 2: Q-Learning Results. The left column displays the Nearest Position (NRST) penalty, while the right column displays the Restart (STRT) penalty.

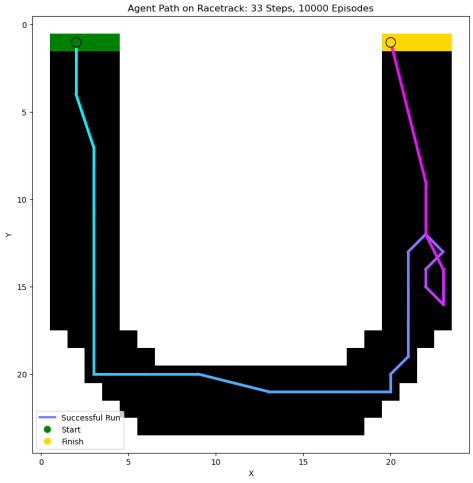
### 3.3 SARSA



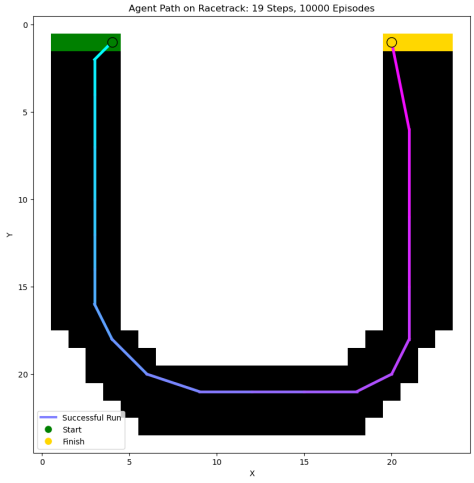

(a) 2-Track (NRST)



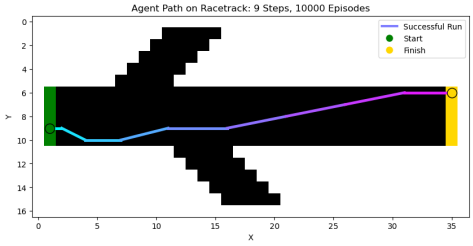

(b) 2-Track (STRT)



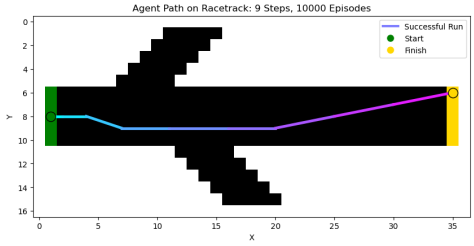

(c) U-Track (NRST)




(d) U-Track (STRT)




(e) W-Track (NRST)




(f) W-Track (STRT)

Figure 3: SARSA Results. The left column displays the Nearest Position (NRST) penalty, while the right column displays the Restart (STRT) penalty.

**For Value Iteration:**

- For "2-Track," Value Iteration found an optimal path in 23 steps for the NRST condition and 34 steps for the STRT condition. The STRT condition required almost 100 more episodes to converge. The NRST path is more aggressive, with abrupt velocity changes and sharp cornering, while the STRT path is smoother and more like what a professional human driver would take. Towards the end of the STRT path, the agent did not accelerate properly, causing a sudden change in direction.
- For "U-Track," the optimal path was 18 steps for NRST and 21 steps for STRT. Similar to the previous track, STRT took longer to converge, requiring 77 additional episodes. The NRST path shows sharper cornering and even a crash after the first long straight, whereas the STRT path is smoother and safer.
- For "W-Track," both NRST and STRT paths required 10 steps. Unlike the other tracks, the STRT path converged slightly faster by one episode, likely due to the track's simplicity. Both paths are almost straight and free of collisions.

**For Q-Learning:**

- For "2-Track," the Q-Learning agent found a path of 104 steps for the NRST condition and 162 steps for the STRT condition. The behavior difference is distinct: the NRST path is chaotic and visibly "bounces" along the track, utilizing the low crash penalty to make forward progress despite errors. In contrast, the STRT plot shows a high density of failed attempts (orange lines), indicating that the agent struggled to learn a safe policy. The final STRT path is significantly longer because it takes a wide, conservative route to avoid the risk of sending the agent back to the start.
- For "U-Track," there was a massive divergence in performance. The NRST agent exploited the soft penalty to hug the inner wall, accepting a crash (indicated by the red X) to maintain a tight racing line, finishing in just 22 steps. However, the STRT agent failed to converge to an efficient policy, taking 371 steps to complete the track. The harsh restart penalty likely caused the agent to fall into a suboptimal policy of erratic, low-velocity movements to minimize the probability of a fatal crash, rather than learning the smooth curve required for speed.
- For "W-Track," the STRT condition actually outperformed the NRST condition, finishing in 88 steps compared to 97. While the NRST agent shows a zigzagging pattern—likely correcting for minor collisions or velocity errors—the STRT agent learned a cleaner, straighter trajectory. The narrow nature of the W-Track means that any deviation typically results in a crash; consequently, the STRT agent was forced to learn the precise "needle-threading" line to survive, resulting in a more efficient final path than the permissive NRST agent.

**For SARSA:**

- For "2-Track," SARSA demonstrated a significant divergence in strategy based on the penalty. The NRST agent adopted a highly conservative approach, finishing in 54 steps by taking wide turns and keeping maximum distance from obstacles. In contrast, the STRT agent found a much more efficient path of 29 steps. Because a single crash in the STRT condition results in a total restart, the agent likely learned that the safest way to minimize the probability of a stochastic error accumulating over time was to minimize the time on the track itself, leading to a straighter, faster line that still maintained a safe margin from the walls.

- For "U-Track," the results mirrored the previous track. The NRST agent converged to a 33-step path that strictly followed the centerline, prioritizing safety over speed. The STRT agent, however, converged to a 19-step path. This trajectory is noticeably tighter to the interior wall than the NRST path. Unlike the Q-Learning STRT agent, which failed to converge on this track, the SARSA STRT agent successfully balanced risk and reward, finding a path that was fast enough to minimize exposure to randomness but safe enough to avoid the restart penalty.
- For "W-Track," both conditions converged to an identical 9-step path. Due to the constrained geometry of the track, there is little distinction between a "safe" path and an "optimal" path; deviations in any direction increase crash risk. SARSA successfully identified the straight-line solution in both scenarios, avoiding the oscillating behavior observed in the NRST Q-Learning experiments.

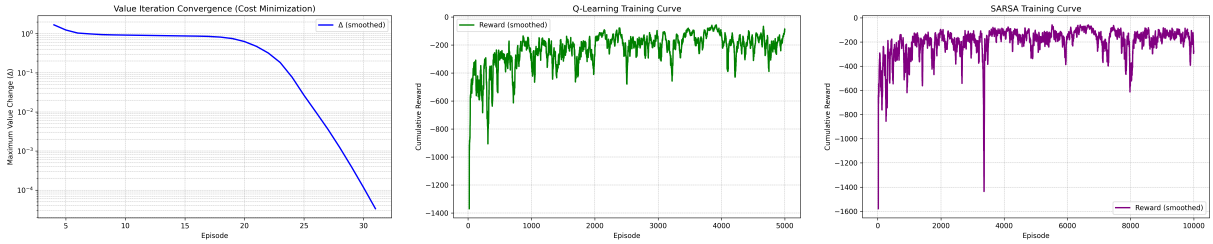



Figure 4: Comparison of learning curves on the "U-track" (CRASH_POS = "NRST"): Value Iteration $\Delta$ showing maximum value change, Q-Learning cumulative reward, and SARSA cumulative reward.

The learning curves illustrate how each algorithm reduces cumulative cost over training. For Value Iteration, the $\Delta$ plot shows the maximum change in the value function per iteration, which decreases exponentially as the value estimates converge to the optimal policy, directly minimizing cost. In Q-Learning, the cumulative reward per episode starts highly negative (reflecting large initial costs) and increases toward zero as the agent learns to take shorter, lower-cost paths, with occasional fluctuations due to stochastic transitions and $\epsilon$-greedy exploration. SARSA exhibits a similar trend, but because it is on-policy and updates based on the actions actually taken—including exploratory moves—its cost reduction is often slightly slower and exhibits larger fluctuations. Together, these curves demonstrate the convergence behavior and efficiency of model-based versus model-free reinforcement learning approaches in the racetrack environment, with all three ultimately learning to minimize total cost.

## 4 Discussion

In summary, the experiments demonstrate a trade-off between optimality and safety. Value Iteration provided the theoretical lower bound for step counts, revealing that the optimal policy shifts significantly, becoming wider and more conservative, when the cost of failure increases from a wall penalty (NRST) to a restart (STRT). Among the model-free approaches, Q-Learning's off-policy nature drove it toward aggressive, time-minimizing trajectories that under the NRST penalty but often produced inefficient paths under the STRT constraints due to frequent crashing. Conversely, SARSA's on-policy learning produced risk-averse behaviors that, while occasionally slower, proved more reliable in high-penalty scenarios. This confirms that for environments with stochastic dy-

namics an algorithm that incorporates the risk of the agent's own behavior into its value estimates is often better than one that assumes perfect execution.

### 4.1 Challenges

Value Iteration struggled to find an optimal path for the "2-track" with $CRASH\_POS = STRT$. The agent often got stuck near the finish line. We believe this occurred because, in the absence of a crash penalty, the agent considered crashing into walls to be "cheaper" than making valid moves. To address this, we introduced a small crash penalty of 2, which promptly resolved the issue and allowed the algorithm to find the correct optimal path.

Q-learning and SARSA were significantly slower compared to Value Iteration. Both algorithms required many more episodes of training before converging on the optimal path. This is expected, as these are model-free, trial-and-error methods that learn through interaction with the environment, whereas Value Iteration is a model-based algorithm that computes the optimal policy directly from the transition dynamics. As a result, Q-learning and SARSA needed more episodes to adequately explore the state space and accurately estimate the action values, especially in tracks with more complex layouts or stochastic elements.

## 5 Summary

Overall, the coding portion of this project was very enjoyable. Compared to other projects this semester, it felt the most seamless and engaging. It was rewarding to design the experiments and observe the agents compute their optimal paths. The inclusion of crash cases and stochasticity made the task more interesting, as no single optimal path was identical across all 10 experiments. Value Iteration proved to be extremely efficient, taking at most two minutes to converge for the "2-track" with $CRASH\_POS = STRT$, confirming our expectation that it would produce the shortest possible paths. The behaviors of Q-Learning and SARSA also aligned with our hypotheses, with Q-Learning favoring more aggressive trajectories and SARSA accounting for the risk of stochastic failures, resulting in safer but slightly longer paths.

## 6 Appendix

**Jackson wrote the following:**

- Q-learning and SARSA
- Paper - Abstract, 1, 1.1, 1.2, 1.3, 2.2, 2.3, 4, Results

**AJ wrote the following:**

- Value Iteration
- Paper - 2, 2.1, 2.4, 2.5, 4.1, 5, Results, References

## References

[1] Bresenham. Bresenham Python Package. Available at `https://pypi.org/project/bresenham/`. Accessed: 2025-12-05.