Optimal Policy Derivation for Racetrack Completion Using Monte Carlo Control Methods

Darcy Corson

October 20, 2023

1 Introduction and Motivation

In the context of Reinforcement Learning (RL), Monte Carlo (MC) methods fall under the category of learning methods. Specifically, they are used for learning the value of actions and states based on experience, which is typically represented as a series of episodes within the environment. These methods are especially valuable when detailed information about the environment is not accessible. This is because these methods facilitate learning through interaction, allowing an agent to understand and strategize within its environment without needing a comprehensive model. Thus, in the absence of pre-defined rules, the agent embarks on multiple episodes, drawing on these accumulated experiences to inform future decision-making.

The motivation behind solving Exercise 5.12, the "Racetrack" problem, presented by Richard S. Sutton and Andrew G. Barto in their text, *Reinforcement Learning: an Introduction*, lies in its potential for illustrating fundamental reinforcement learning principles in a relatable, real-world scenario. As part of completing this exercise, agents must learn to balance between the competing objectives of speed and safety, as racetracks both impose strict physical boundaries and demand high performance. This creates an environment full of risk and reward, making them perfect for studying how to make choices and learn from them. Further, by using a simple, grid-based course model, and including a random "noise" factor, the exercise mimics the complexities that a driver might face in real-life, such as changing track conditions, obstacles, or other unpredictable disturbances.

This study attempts to compute the optimal policy for navigating the racetrack from various starting states, achieved by employing MC methods. These methods are particularly appropriate for this task because they allow for exploration and exploitation of possible actions in the race car's discrete and varied action space. The work explained here intends to demonstrate both the derivation of an optimal policy, and the adaptability and resilience of MC methods in the face of the uncertainty imposed by the problem's "noise" factor.

2 Background

2.1 Monte Carlo Methods

As mentioned above, in RL, MC methods are primarily characterized by their model-free paradigm that learns directly from episodes of experience. These methods leverage randomness, sampling different episodes to approximate the expected value of states. This circumvents the need for a known model of the environment. In environments where uncertainty is pervasive, this feature of MC methods makes them particularly useful.

Central to MC methods is the principle of averaging sample returns to estimate values. This technique is known as empirical mean approximation. These values usually represent the expected cumulative future rewards from a particular state. Mathematically, this is expressed as

$$V(s) = E[G_t | S_t = s]$$

where V(s) is the value of state s and G_t represents the return after time t. In the context of MC methods, G_t is defined as the cumulative discounted reward from time t moving forward, mathematically:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

where γ is the discount factor, which determines the present value of future rewards. Examination of this formula reveals the foresight embedded within MC estimations, as it considers the entire set of future rewards while mitigating the impact of those that are temporarily distant. This aspect of the calculation ensures a sensible balance between immediate and future gains.

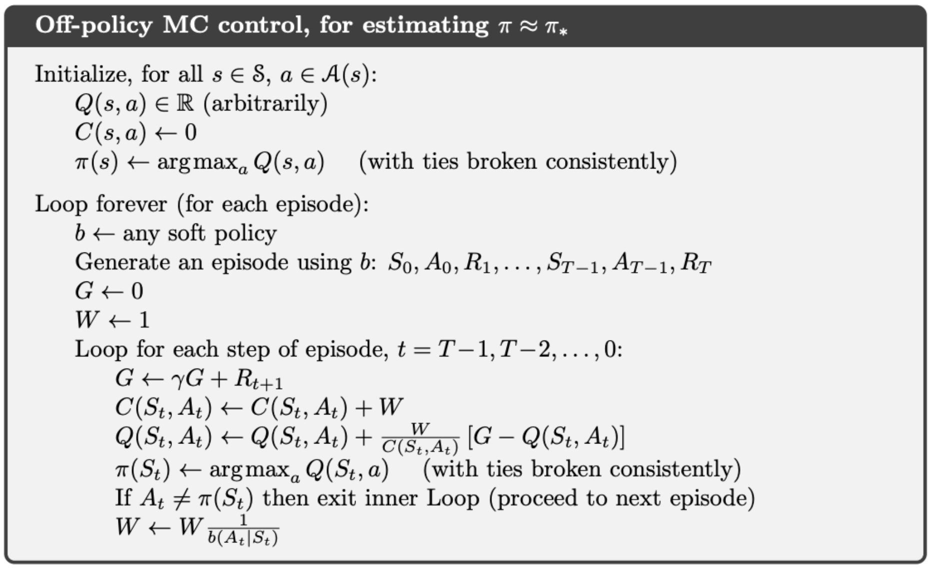
Also central to MC methods in RL is their episodic nature. In this context, an episode is defined as a sequence of states, actions, and rewards that culminates in a terminal state. For MC methods, learning occurs at the end of each episode. This is important because it implies that value estimates (V(s)) are only updated after an episode has finished, thereby using the full sequence of experienced rewards to compute G_t. This strategy is particularly useful when the consequences of individual choices, represented by actions, manifest over extended time horizons in unpredictable environments.

In RL, the process of policy evaluation involves the iterative update of value functions, as prescribed by the policy. For MC methods, this update uses the mean value return for each state across episodes, and it is continuously refined as more returns are observed. After many episodes, these estimates converge to the expected values under the current policy. During policy improvement, the agent enhances its decision-making strategy based on the updated knowledge of the value functions. Often, this leads to a new policy that is greedily chosen with respect to current value function estimates.

2.2 Off-Policy Methods in Monte Carlo

Off-policy methods are a notable subset of MC methods. Off-policy methods, particularly off-policy MC control, offer a strategic divergence from traditional learning paths, emphasizing learning the optimal policy, $\pi*$, through the exploration of state-action pairs distinct from the current policy being improved. These techniques prove crucial in scenarios where a learning agent's real-world experiences do not directly contribute to the evaluation and refinement of the current strategy. Off-policy learning hinges on the contrast between two policies: the "behavior policy," outlining current actions, and the "target policy," representing the ideal set of actions. By differentiating these policies, the agent can leverage experiences beyond those available through its current strategy, potentially hastening the learning curve by drawing from a more extensive knowledge pool.

An important element in off-policy MC control is its commitment to thorough exploration. This exploration, necessary for ensuring that no possible valuable strategy is overlooked, is commonly facilitated by incorporating ϵ-greedy mechanisms. These mechanisms maintain a careful balance within the MC system, promoting adherence to proven strategies while permitting occasional exploration of novel actions.



```
Off-policy MC control, for estimating $\pi \approx \pi_*$

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
    $Q(s,a) \in \mathbb{R}$ (arbitrarily)
    $C(s,a) \leftarrow 0$
    $\pi(s) \leftarrow \arg\max_a Q(s,a)$    (with ties broken consistently)

Loop forever (for each episode):
    $b \leftarrow$ any soft policy
    Generate an episode using $b$: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    $W \leftarrow 1$
    Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
        $G \leftarrow \gamma G + R_{t+1}$
        $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
        $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$
        $\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$    (with ties broken consistently)
        If $A_t \neq \pi(S_t)$ then exit inner Loop (proceed to next episode)
        $W \leftarrow W \frac{1}{b(A_t|S_t)}$
```

Figure 1: An algorithm for an off-policy Monte Carlo control method for estimating the optimal policy and the optimal action-value function.

2.3 Exploitation-Exploration Dilemma and Epsilon-Greedy Strategy

The exploitation-exploration dilemma in RL refers to the challenge of choosing between exploiting known rewards by taking actions that have worked well in the past, and trying new actions to discover better solutions. Exploitation optimizes for immediate performance, potentially missing out on better options. Exploration risks short-term failure for long-term benefit. Balancing this trade-off is crucial for an RL agent's efficiency and success in adapting to diverse and dynamic environments. MC methods navigate the exploitation-exploration using techniques like the epsilon-greedy strategy within the MC framework. The epsilon-greedy algorithm responds to the exploitation-exploration dilemma by most often taking the action that is currently known to be optimal, with probability 1 - ϵ, and choosing a random action, irrespective of the action-value estimate, with probability ϵ.

MC methods rely on learning from complete episodes. Incorporating an epsilon greedy approach ensures the coverage of the state-action space is diverse. In this way, the agent is given a broad range of experiences from which to learn. This diversity is crucial because MC methods require a rich set of experiences to accurately estimate the value functions. This is because they directly learn from the actual returns following each visited state.

Further, use of the epsilon-greedy strategy helps to prevent the MC agent from converging prematurely to a suboptimal policy. This can happen if the agent doesn't explore enough, as they can get stuck in a local optimum. By maintaining a balance between exploration and exploitation, the epsilon-greedy strategy allows the agent to continue learning about potential unseen rewards in unvisited state-action pairs, enhancing the likelihood of discovering and converging to the true optimal policy over time.

2.4 Q-Learning

Q-learning is another model-free RL algorithm that provides a mechanism for agents to learn optional policies without a comprehensive model of the environment. Central to Q-learning is the use of a Q-value function which estimates the expected value (cumulative future rewards) of executing an action, a, in state, s, and following the optimal policy thereafter. This approach enables the agent to make decisions based solely on learned experiences, characterized by the following iterative update rule:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \underset{a}{max} Q(s_{t+1}, a) - Q(s_t, a_t)],$$

where:

- α represents the learning rate, determined by the weight given to new information,

- r_{t+1} is the reward received after taking action a_t,
- γ is the discount factor, which quantifies the importance of future rewards relative to immediate ones, and
- $\underset{a}{max}Q(s_{t+1}, a)$ signifies the estimate of optimal future value, encouraging the selection of actions leading to states with higher expected rewards.

By continuously updating the Q-values towards the highest reward and propagating these updates backwards, Q-learning facilitates the discovery of optimal action-selection policies. This occurs even in stochastic and non-deterministic environments.

2.5 Policy Evaluation and Improvement

The cycle of policy evaluation and policy improvement that moves agent behavior towards the optimal policy is also integral to the process of RL. Policy evaluation is the process of computing the expected cumulative value of all states in a given policy. This step provides a performance metric, letting the agent know the quality of its current strategy. Policy improvement adjusts the agent's subsequent action based on the information gathered during the evaluation phase. Through policy iteration, the agent refines its decisions, thereby ensuring that every action selected increases the expected return following the current policy. Formally, this can be represented as adopting a new policy π', such that:

$$\pi'(s) = \underset{a}{argmax}Q(s, a),$$

where π' denotes the policy derived from the state-action pair that maximizes the expected cumulative future rewards. This greedy approach towards the highest Q-values directly enhances policy performance, pushing the agent's strategy closer to the optimal behavior.

3 Solution

The solution to Exercise 5.12, described above, was comprised of two pieces: the agent and the environment. The solution introduced a simulated environment that represented a digitized racetrack. This environment was governed by precise motion dynamics, boundary conditions, and goal-oriented tasks. Within this environment, an intelligent agent was installed. The agent's primary directive was to learn the best navigation strategies, through trial and error, thereby honing its ability to make decisions based on real-time feedback. The interaction between environment and agent was governed by a set of probabilistic rules, where each agent's action triggered a specific state transition within the environment. The agent navigated the digital racetrack where success was not rewarded incrementally. Instead, every misstep, such as collision with a wall, resulted in a penalty. This framework necessitated that the agent adapt its strategy to minimize negative outcomes. Ultimately, through this iterative process of decision-making, feedback interpretation, and strategy adjustment, the agent autonomously formulated an understanding of the environment's layout and dynamics. With this information, it identified a strategy to move effectively from the track's starting line to the track's finish line.

3.1 The Environment

The simulation environment was materialized through the development of the "RaceTrack" class. More than a backdrop, the environment was a dynamic entity that simulated the real-time decision-making and motion dynamics of a race car navigating a track. At initialization, a "course" was identified. This course, delineated by strings, represented various track elements that the "init" method converted into a numerical matrix. This allowed for nuanced simulations and interactions reflective of real-world physics. Also during initialization, "noise" and minimum and maximum velocity constraints were established. This imposed a realistic operational range for the simulated car.

Central to the environment-agent interaction was the `take_action` method, a function that interpreted the agent's decisions (represented as tuples indicating velocity changes) and recalibrated the state of the environment. This recalibration considered several possible scenarios, including the vehicle remaining within track confines, colliding with boundaries, and successfully reaching the end goal. Specific outcomes, like collisions, triggered a resetting protocol, reverting the car to starting parameters at a random location on the starting line.

The `_update_velocity` and `_update_position` methods were instrumental in managing state transitions. These functions allowed the simulation to respond to agent-induced actions by recalibrating velocity and position while recognizing physical limits and collision potential. The implementation accounted for stochastic factors via 'NOISE', ensuring that the simulation mirrored the unpredictability of real-world scenarios.

Further, the environment employed utility checks (`_is_wall`, `_is_finish`, and `_is_terminal_state`) that served as interpretative mechanisms, evaluating the consequences of the agent's actions. Although binary checks; they provided critical feedback, determining the course of subsequent computational processes, either advancing the simulation or invoking the 'reset' method based on situational demands.

3.2 The Agent and Its Interaction with the Environment

Contained in the 'Agent.py' file, the agent was represented through the actions it took within the environment during the training process. The agent's behavior was governed by the strategies implemented, specifically the epsilon-greedy policy during the exploration and exploitation stages, and the learning algorithm, which was a form of Q-learning.

The agent's behavior was dictated by the policy 'pi', initialized to encourage specific actions in given states. Initially, it was set to a default action, promoting exploration. The agent operated with an epsilon-greedy strategy. This approach allowed the agent to choose, with probability 1 - ϵ, the action that had the maximum expected future rewards as per the current knowledge (exploitation). It also empowered the agent to select, with probability ϵ, a random action (exploration). This balance was allowed the agent to explore new states while also making informed decisions based on its experiences.

The agent learned by interacting with the environment (RaceTrack). These interactions occured within a loop that generated "episodes," where an episode constituted a complete sequence of the agent's journey from a starting point (the starting line) to a terminal state(the finish line). For each step within an episode, the agent observed its current state, made a decision on the action to take (based on its policy), and executed the action. The environment responded with a new state and reward, indicating the consequences of that action.

The agent's decision-making process at each step is where the ϵ-greedy strategy was applied. The variable ϵ regulated the exploration level, and it is adjusted as training progressed to reduce random actions and focus more on learned strategies for exploitation.

After each episode, the agent updated its Q-table, which was a learned representation of the quality (Q-values) of taking certain actions in certain states. This update was based on the experiences (state, action, reward sequences) it collected throughout the episode. The agent used these experiences to update its policy, changing its behavior for future episodes. The policy was modified in such a way that the actions leading to higher expected rewards were more likely to be chosen in specific states. The learning phase included an adaptation of the Q-values towards the actual rewards obtained, refining the agent's strategy and ensuring it became more efficient in navigating the race track based on the feedback from the environment.

Through continuous interaction and learning over many episodes, the agent improved its policy, which converged to an optimal policy that was capable of navigating the whole racetrack effectively.

The learning algorithm ran for N = 500,000 episodes, signifying a substantial period for the agent to learn from its environment, actions, and received rewards. The discount factor was set to 0.9, indicating the agent's preference for rewards that are expected sooner rather than later. With each episode, ϵ was reduced by multiplying the initial ϵ by the decay rate raised to the power of the episode number. This exponential decay ensured a rapid reduction at first that slows down as ϵ approached its minimum limit.

3.3 The Off-Policy Every-Visit Monte Carlo Nature of the Agent's Learning Method

The methodology employed here should be thought of as a form of Monte Carlo method for several reasons. First, full full episodes were generated using a policy (either exploratory or greedy, depending on the stage of training). Estimates were not updated based on partial episodes, but the algorithm waited until the complete sequence of states, actions, and rewards was known for a full episode before processing. This is a characteristic of Monte Carlo methods. The portion of the code related to learning from episodes can be found in the loop that processes the episode list. Here, the algorithm waits until the end of each episode, then works backward through the episode to update its action-value function estimate (Q). In this way, it uses the full return following each visited state to update the estimates.

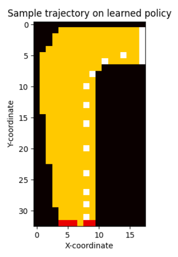
Further, a distinctive approach to learning was adopted, known as off-policy learning, to enhance the agent's learning process. This strategy marked a departure from conventional on-policy methods. Specifically, the agent's behavior was driven by an ϵ-greedy policy, balancing occasional exploration of new actions with the exploitation of existing knowledge. Simultaneously, the agent was gaining insights about a theoretical ideal policy, one that would rely solely on the most beneficial, proven actions. The complexity of managing these two differing learning processes was addressed through a technique called importance sampling. Central to this approach was the incorporation of an adjustment factor, termed 'W.' This factor played a critical role in updating the agent's understanding (reflected in the Q-values), adjusting the information gained from the exploratory ϵ-greedy policy based on how it deviated from the ideal, exploit-only policy. Therefore, through importance sampling, the agent efficiently reconciled the insights acquired from both exploration and exploitation, refining its overall learning journey.

Also, the learning algorithm distinguished itself by adopting an every-visit Monte Carlo approach as opposed to the more conventional first-visit method. This strategy is evidenced during the iterative processing of episodes, where the algorithm updates its estimates following every occurrence of a state within an episode, instead of checking for prior visits. This approach optimized the learning process, ensuring a comprehensive update mechanism that derived information from each state encounter to continually refine the policy.

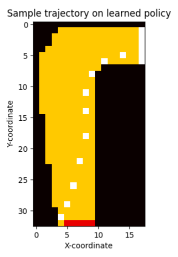
3.4 Results

3.4.1 Sample Trajectories Using Learned Policy

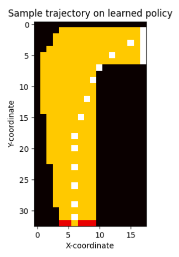
Six sample trajectories following the optimal policy for traversing the course as identified by the learning algorithm explained above (with noise turned off) are presented below.



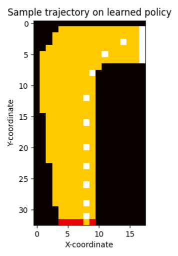
(a) Sample Trajectory 1



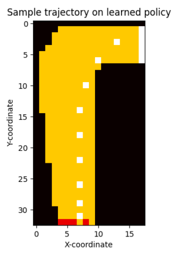
(b) Sample Trajectory 2



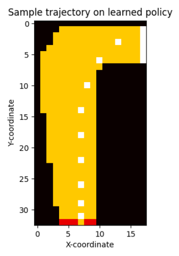
(c) Sample Trajectory 3



(d) Sample Trajectory 4



(e) Sample Trajectory 5



(f) Sample Trajectory 6

Figure 2: These six sample trajectories were created by following the optimal policy (with noise turned off). They demonstrate the validity of the Monte Carlo control method employed by the solution, because they show successful travel from the starting line to the finish line.

By turning off the environmental noise for these trajectories, the learning algorithm's performance, separate from the stochastic fluctuations usually present in dynamic environments, is isolated. These trajectories substantiate the validity of the MC control method utilized in the solution. By following the optimal policy and achieving the objective efficiently (guiding the race car through various turns without derailment) these trajectories serve as empirical evidence of the method's success.

3.4.2 Policy Exploration Heatmap

A heatmap that represented the agent's policy exploration was generated for this experiment, and is displayed below.

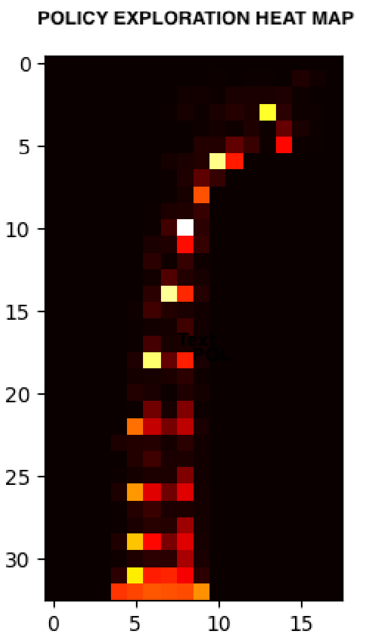


Figure 3: The heatmap's denser regions correspond to the agent's understanding of the most rewarding paths. This implies that the learning process has effectively helped the agent recognize and exploit optimal strategies for course navigation.

The heatmap accentuates areas of the track that the agent frequently accessed and that the agent tended to avoid. This visualization reveals tendencies towards certain maneuvers and paths, indicative of learned behaviors and biases within the agent's decision-making framework. For instance, the concentration of activity for the central trajectory indicates the agent's preference for that specific set of maneuvers, reflecting its evolving understanding of the track's constraints and opportunities for maximum reward (e.g. maintaining optimal speed without running off the track). The areas of sparse exploration in other regions suggests that the agent learned to successfully avoid less favorable paths (e.g. areas leading to frequent crashes). Ultimately, the heat map's denser regions correspond to the agent's understanding of the most rewarding paths, confirming that the learning process has effectively helped the agent recognize and exploit these optimal strategies for course navigation.

3.4.3 Rewards vs. Episodes Graph

For the solution described in this paper, the "Rewards vs. Episodes" graph serves as an excellent illustration of the agent's learning over episodes within the environment as structured around penalization rather than reward. The graph, which summarizes the trajectory of accrued rewards across 500,000 episodes, demonstrates the agent's learning efficacy over a substantial period. The environment designed for this experiment does not encourage the agent with rewards for favorable actions. Instead, it returns a penalty if the agent makes an undesirable decision or maneuver. As such, the convergence of the reward metric to zero represents the agent successfully learning to navigate the complexities of the environment while minimizing the frequency and magnitude of penalties incurred.

Initially, the agent undergoes a phase of exploration, characterized by a broad range of negative rewards, signifying repeated penalties. This stage is crucial for the agent to understand the consequences of incorrect actions within the set parameters of the environment. As the agent progresses through the episodes, the consistent interaction with the environment and the consequent penalties lead to an iterative refinement of its behavioral policy. This evolution is evident from the decreasing trend in negative rewards, gradually approaching zero. The zero-convergence does not imply a lack of activity or engagement with the environment but rather signifies an optimal understanding and response to it. It shows the agent's learned ability to perform actions that avoid penalties, having understood the underlying structure and rules of the environment through its interactions.

Overall, the "Rewards vs. Episodes" graph's trend towards zero is a testament to success of the solution's learning algorithm. It confirms that, over 500,000 episodes, the algorithm has effectively enabled the agent to decipher a strategy that mitigates penalties.

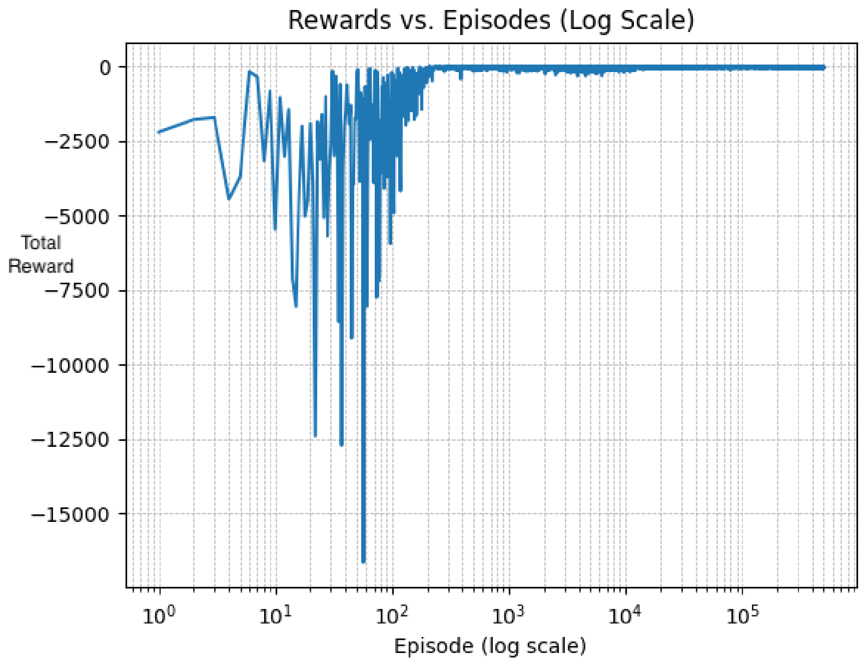


Figure 4: This graph, which summarizes the trajectory of accrued rewards across 500,000 episodes, converges to 0. This convergence confirms that, over 500,000 episodes, the reinforcement learning algorithm has effectively enabled the agent to decipher a strategy that mitigates penalties.

4 Extensions

The idea of introducing an alternative reward structure for exercise 5.12 represents a compelling avenue for future research. Revising the environmental set-up to incorporate positive reinforcements, rather than solely relying on penalties, might improve the agent's learning dynamics. It may be that an environment that only penalizes actions leads to conservative strategies, where the agent primarily learns what not to do. By introducing rewards, the agent may be encouraged to actively seek out beneficial behaviors, thus accelerating the discovery of optimal or near-optimal solutions.

5 References

Coady, Pat. "Sutton and Barto Racetrack: Monte Carlo Control." GitHub Gist, [n.d]. Web. [10/9/2023].

Ou, Tingsong. "Solving Reinforcement Learning Racetrack Exercise with Off-policy Monte Carlo Control." Towards Data Science, 23 July 2023, https://medium.com/towards-data-science/solving-reinforcement-learning-racetrack-exercise-with-off-policy-monte-carlo-control-.

Rastogi, Aditya. ”Solving Racetrack in Reinforcement Learning Using Monte Carlo Control.” Towards Data Science, 6 Jan. 2020, https://towardsdatascience.com/solving-racetrack-in-reinforcement-learning-using-monte-carlo-control-bdee2aa4f04e.

Sutton, Richard S., and Andrew G. Barto. Reinforcement Learning: An Introduction. The MIT Press, 2018.