Reinforcement Learning Project

Tabular Control of `Pendulum-v1`

Tomás Silva[1], Autor Dois[2], and Autor Três[3]

[1]20230982
[2]
[3]

June 27, 2025

1 Introduction

To explore the capabilities of reinforcement learning (RL) algorithms, we developed two agents designed to solve distinct environments from the Gymnasium toolkit. The main objective was to implement and analyze RL methods that can handle environments with varying complexities in their state and action spaces.

We selected two environments to represent different types of problems:

- **Inverted Pendulum** (`Pendulum-v1`) – a continuous control task characterized by non-linear dynamics.
- **Blackjack** (`Blackjack-v1`) – a discrete, episodic decision-making problem well-suited for tabular methods.

The Inverted Pendulum environment is a classical control challenge in which the agent must swing a pendulum from a random starting position to the upright position and maintain balance. Since the state and action spaces are continuous, we applied discretization techniques to enable the use of standard tabular algorithms such as SARSA and Q-Learning.

The Blackjack environment simulates a simplified version of the card game, making it ideal for evaluating tabular RL algorithms. For this task, we implemented and compared the performance of Monte Carlo control, SARSA, and Q-Learning.

By applying different RL approaches to both environments, we aimed to gain insight into algorithm design, learning performance, and adaptability across continuous and discrete domains.

2 Problem statement

In this section, we present the two environments selected for this project—`Pendulum-v1` and `Blackjack-v1` and outline the reinforcement learning (RL) methods used to address each. These environments were chosen to highlight different types of decision-making challenges: continuous control for the Pendulum and discrete, episodic decision-making for Blackjack.

2.1 Pendulum-v1 Environment Description

The `Pendulum-v1` environment from the Gymnasium suite simulates a classic control problem: the inverted pendulum swing-up task. The system consists of a pendulum attached to a fixed hinge, with the other end free to rotate. The pendulum starts in a random position, and the

agent's objective is to apply torque to swing it upright and balance it vertically above the pivot point.

State and Action Space

`Pendulum-v1` features continuous state and action spaces. The state at each time step is given by the tuple:

$$(\cos\theta, \sin\theta, \dot{\theta}),$$

where θ is the angle of the pendulum from the vertical and $\dot{\theta}$ is the angular velocity.

The action is a continuous torque $a \in [-2, 2]$ applied by the agent at each time step.

System Dynamics

The environment exhibits nonlinear dynamics due to the trigonometric and differential relationships between the angle, angular velocity, and torque. This nonlinearity makes the control task challenging, as small variations in torque can cause drastically different outcomes depending on the pendulum's state.

Reward Function

The reward at each time step is defined as:

$$r_t = -\theta_t^2 + 0.1\dot{\theta}_t^2 + 0.001a_t^2,$$

which penalizes deviations from the upright position, high angular velocity, and large torque values. The optimal reward is:

$$R_{\text{opt}} = 0,$$

achieved when $\theta = 0$, $\dot{\theta} = 0$, and $a = 0$.

Episode Properties

Each episode runs for up to 200 time steps. A random policy typically results in a cumulative reward around -1500, which serves as a baseline for assessing learning performance.

2.2 Blackjack-v1 Environment Description

`Blackjack-v1` simulates a simplified version of the classic card game, designed to evaluate reinforcement learning algorithms in a discrete, episodic decision-making setting. The goal is to beat the dealer by obtaining a hand with a total value closer to 21, without exceeding it.

Initial Setup

Each game begins as follows:

- The dealer receives one visible card and one hidden card.
- The player receives two visible cards.

All cards are drawn from an infinite deck (sampling with replacement). The card values are:

- Face cards (Jack, Queen, King): 10 points.
- Number cards (2–10): face value.
- Aces: 11 (usable ace) or 1, depending on which is more advantageous to the player without causing a bust.

Game Rules

During play:

- The player may choose to **hit** (draw a card) or **stick** (end their turn).
- If the player's total exceeds 21, they bust and automatically lose.
- Once the player sticks, the dealer reveals their hidden card and draws until reaching a total of at least 17.

Outcomes

The game's outcome is determined as follows:

- If the dealer busts, the player wins.
- If neither busts, the hand closest to 21 wins.
- A tie results in a draw.

Environment Characteristics

This environment provides:

- A controlled, discrete state and action space.
- A finite action space: **hit** or **stick**.
- An ideal setting for model-free, value-based reinforcement learning methods such as Monte Carlo, SARSA, and Q-Learning.

3 Methodology

In this section, we outline the methodology used to address both the Pendulum-v1 and Blackjack-v1 environments from the Gymnasium toolkit. We begin by providing a brief description of each environment, followed by a detailed explanation of the reinforcement learning approaches and implementation strategies applied to solve them.

3.1 Pendulum-v1: Continuous Control with Tabular RL

`Pendulum-v1` poses a continuousstate, continuousaction swing-up problem, where the agent must apply a torque $a \in [-2, 2]$ to balance the pendulum upright. Because classical tabular algorithms such as **Q-Learning** and **SARSA** require finite state–action sets, we first introduced a *discretisation layer*:

- **State discretisation**: the continuous observation $(\cos\theta, \sin\theta, \dot{\theta})$ was binned into fixed intervals for angle and angular velocity.
- **Action discretisation**: the torque range $[-2, 2]$ was quantised into N equally spaced torque levels.

A custom Gymnasium-compatible *wrapper* implements this mapping while preserving the original environment dynamics and reproducible seeding via `reset()` and `seed()`. The resulting finite MDP allows us to learn a Q-table $Q(s, a)$ through repeated trial-and-error interactions. Both SARSA (on-policy) and Q-Learning (off-policy) were trained until convergence of the discretised value function.

3.2 Blackjack-v1: Model-Free Value Methods

`Blackjack-v1` is inherently discrete and episodic, making it ideal for tabular, model-free RL. We compared three value-based algorithms:

- **Monte Carlo control** — first-visit estimation of $Q(s, a)$ with ε-greedy improvement.
- **SARSA** — on-policy TD(0) updates.
- **Q-Learning** — off-policy TD(0) with $\max_{a'} Q(s', a')$ targets.

Each algorithm iteratively estimated action-value functions and refined its policy across many independent episodes of play, leveraging the environment's small, finite state space: (player sum, dealer card, usable ace).

By discretising `Pendulum-v1` we made a continuous control task amenable to tabular RL, while `Blackjack-v1` allowed direct application of model-free value methods. This dual-environment study highlights how algorithm design and preprocessing must adapt to the underlying state and action structure of each problem.

3.3 Pendulum-v1: Discretisation Strategy

To enable tabular **Q-Learning** and **SARSA** in the continuous `Pendulum-v1` environment, we map both the state and action spaces onto finite grids. The resulting Q–table can then be indexed by discrete state–action pairs.

Global discretisation parameters

- **State space**: Each observation is the vector $(\cos\theta,\ \sin\theta,\ \dot{\theta})$. We quantise it into
  - 25 bins for the angle θ,
  - 25 bins for the angular velocity $\dot{\theta}$.

  Hence the discrete state space contains $25 \times 25 = 625$ states.
- **Action space**: The continuous torque $a \in [-2, 2]$ is reduced to the fixed set

$$\mathcal{A} = \{-2,\ -1,\ 0,\ 1,\ 2\},$$

  giving 5 discrete actions.

Key helper functions

[label=0.]`discretised_state()`

1.
   - *Input*: $(\cos\theta, \sin\theta, \dot{\theta})$
   - *Output*: (angle_bin, vel_bin)
   - *Procedure*:
     [label=.]Recover $\theta = \text{atan2}(\sin\theta, \cos\theta)$. Clip θ and $\dot{\theta}$ to predefined ranges. Assign both values to their respective bins.

(2) `initialize_Q_table()`

- Creates a zero-initialised array with shape $[25, 25, 5]$.
- Each entry $Q[s, a]$ stores the expected return for state s and action a.

3. `epsilon_greedy_policy()`

- Implements the ε-greedy rule:

$$a = \begin{cases} \text{random action}, & \text{with prob. } \varepsilon, \\ \arg\max_{a'} Q(s, a'), & \text{with prob. } 1 - \varepsilon. \end{cases}$$

- ε can be annealed to reduce exploration over time.

Discretisation trade-offs

- **Coarse grids** $\rightarrow$ fewer states, faster learning, smaller Q-tables.
- **Fine grids** $\rightarrow$ richer state representation but higher memory and sample complexity.

The same helper functions and binning logic can be reused in other continuous-state environments (e.g. `MountainCar`), demonstrating the portability of the discretisation approach.

3.4 Algorithms Used

We employed three classical, model-free reinforcement learning algorithms—**Q-Learning**, **SARSA**, and **Monte Carlo First-Visit**—to solve the selected Gymnasium environments. Action selection was governed by four exploration–exploitation schemes: *greedy*, *ε-greedy*, *decaying ε*, and *softmax*. These policies ensure sufficient exploration while allowing the agent to converge toward high–value actions.

3.5 Q-Learning (off-policy)

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \Big[r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \Big]. \tag{1}$$

- *Off-policy*: the target uses the *greedy* action $\arg\max_{a'} Q(s_{t+1}, a')$, not necessarily the action taken.
- Typically exhibits *fast convergence*, but can be unstable if exploration is too aggressive.

3.6 SARSA (on-policy)

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \Big[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \Big]. \tag{2}$$

- *On-policy*: updates use the *actual* next action a_{t+1} produced by the current policy.
- Generally *more stable and conservative* than Q-Learning in stochastic settings.

3.7 Monte Carlo Control (first visit)

- Estimates $Q(s, a)$ by averaging the return G obtained from the *first* occurrence of (s, a) in each episode.
- Learns from *complete episodes* rather than step-by-step TD updates, making it well suited to episodic tasks such as `Blackjack-v1`.

3.8 Hyper-parameter Grid Search

To tune learning rate α and discount factor γ we executed a 4×3 grid search:

$$\alpha \in \{0.05, 0.07, 0.10, 0.15\}, \qquad \gamma \in \{0.90, 0.95, 0.99\}.$$

- **Training:** each (α, γ) pair was trained for 1000 episodes.
- **Evaluation:** performance was measured by the average return over the last 100 episodes, $\text{mean}_{\text{last } 100}$, serving as our convergence and policy-quality metric.

4 Evaluation of Results

4.1 Pendulum-v1 Environment

Comparative Performance

Algorithm	Best 100-ep Mean	Final 100-ep Mean	α	γ
Q-Learning	-170.39	-170.39	0.15	0.95
SARSA	-172.48	-176.99	0.15	0.90

Table 1: Performance comparison of Q-Learning and SARSA on Pendulum-v1 over 5000 episodes.

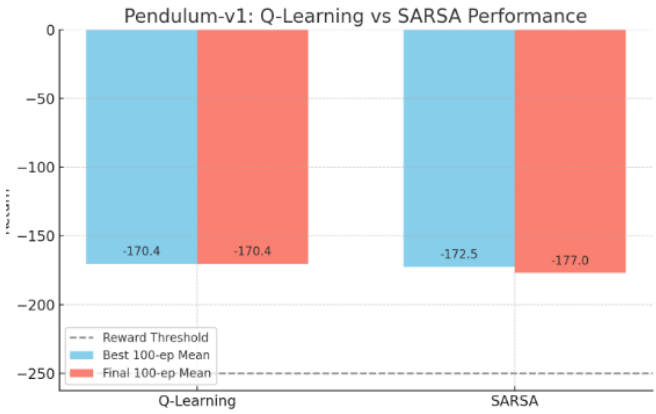

Figure 1: Pendulum Q-Learning

- **Q-Learning** achieved the best overall performance, outperforming SARSA in both the *best* and *final* 100-episode average returns.
- **SARSA** displayed a slightly lower peak return and a modest decline toward the end of training, indicating less stable convergence.
- The higher discount factor in Q-Learning ($\gamma = 0.95$) likely enabled the agent to place more emphasis on long-term rewards than SARSA, which used $\gamma = 0.90$.
- Both algorithms shared the same learning rate ($\alpha = 0.15$), suggesting that the difference in performance is primarily attributable to the choice of γ rather than the step-size parameter.

To better understand the agent performances, we plotted learning curves to compare various algorithms using different policy strategies like Greedy, Epsilon-Greedy, Decay Epsilon, and Softmax.

Q-Learning Performance by Policy

- **Greedy Policy:** Achieves the best overall performance. It demonstrates a rapid learning rate and a stable learning curve. This policy effectively balances exploration and exploitation, enabling the agent to converge quickly and maintain near-optimal performance throughout training.

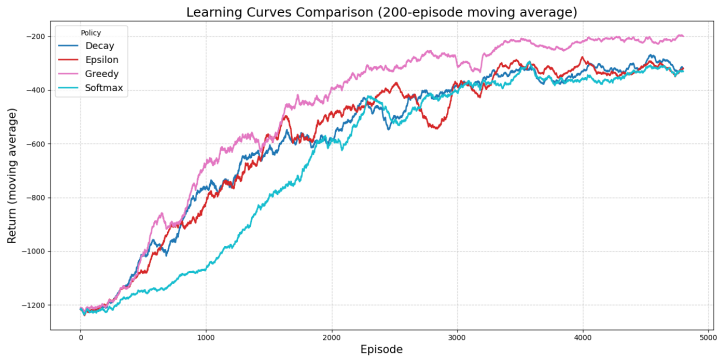

Figure 2: Pendulum Q-Learning

- **Epsilon-Greedy and Decay Epsilon Policies:** Both policies exhibit moderately high performance with reasonable learning speeds. While their learning curves are slightly more oscillatory compared to the Greedy policy, they still indicate effective learning and policy improvement over time.
- **Softmax Policy:** Yields the lowest performance among all policies. It exhibits the slowest learning rate, with delayed convergence and lower average returns. This suggests that Softmax was less effective in guiding the agent toward high-value actions in this environment.

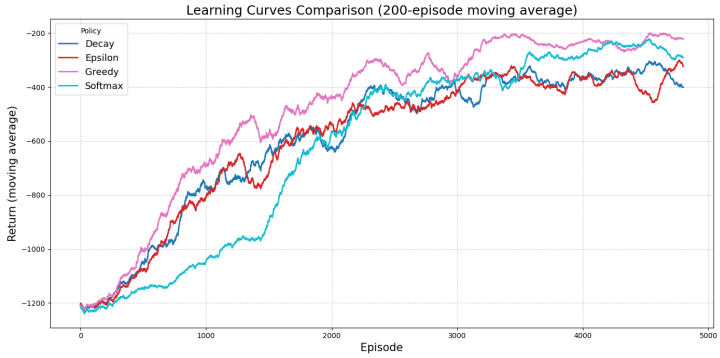

Figure 3: Pendulum SARSA

SARSA Performance by Policy

- **Greedy Policy:** Once again shows the highest performance and stability. However, due to its lack of exploration, it risks converging to suboptimal policies by not sufficiently exploring alternative strategies.
- **Epsilon-Greedy and Decay Epsilon Policies:** Both demonstrate comparable learning behavior, with Decay Epsilon offering slightly better convergence characteristics. Despite this, they fall short of the Greedy policy in terms of final performance and show limited ability to generalize across states.
- **Softmax Policy:** While initially slower to learn, the Softmax policy eventually surpasses both Epsilon-Greedy and Decay Epsilon in terms of stability and average return. This indicates a more consistent and robust long-term learning trajectory, potentially due to its smoother action selection mechanism.

4.2 Blackjack-v1 Environment

For Blackjack, the policies used are Greedy, Epsilon-Greedy, Decay Epsilon, Softmax, and Random

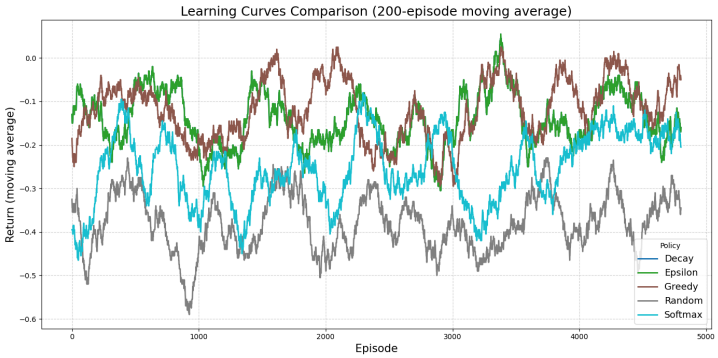

Figure 4: Blackjack Q-Learning

Q-Learning: Performance by Policy

- **Greedy Policy:** Achieves the highest performance, with noticeable peaks and oscillations. These fluctuations likely stem from the algorithm over-exploiting certain high-reward strategies without sufficient exploration, preventing full stabilization.
- **Epsilon-Greedy Policy:** Delivers good overall performance, slightly below that of the Greedy policy. It maintains moderate exploration, but this comes at the cost of convergence, as the policy does not fully stabilize.
- **Softmax Policy:** Shows moderate performance and successfully stabilizes over time. This suggests that Softmax helps the agent converge to a consistent, albeit suboptimal, strategy.
- **Random Policy:** Performs the worst, as expected. It neither learns nor stabilizes, providing a flat baseline that illustrates the necessity of policy-driven learning.
- **Decay Epsilon Policy:** The curve for this policy was not visible in the plot, likely due to a plotting or implementation error. Further debugging is needed to confirm its actual performance.

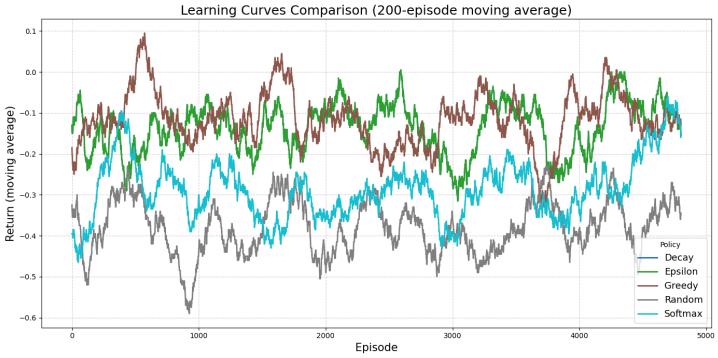

Figure 5: Blackjack SARSA

SARSA: Performance by Policy

- **Greedy Policy:** Demonstrated the best overall performance, with faster convergence and better generalization. This indicates a more reliable learning of near-optimal strategies.
- **Epsilon-Greedy:** Achieved performance close to the Greedy policy, though with noticeable instability over time. This reflects a balance between exploration and exploitation, but with some trade-off in convergence stability.
- **Softmax Policy:** Showed steady improvement throughout training but did not reach the performance levels of Greedy or Epsilon-Greedy. Additionally, it failed to stabilize fully, suggesting limited effectiveness in this environment.
- **Random Policy:** Consistently produced the poorest and most unstable results, reinforcing its inefficacy for learning meaningful strategies.
- **Decay Epsilon:** Performance data for this policy was not visible in the resulting graphs, possibly due to implementation or rendering issues.

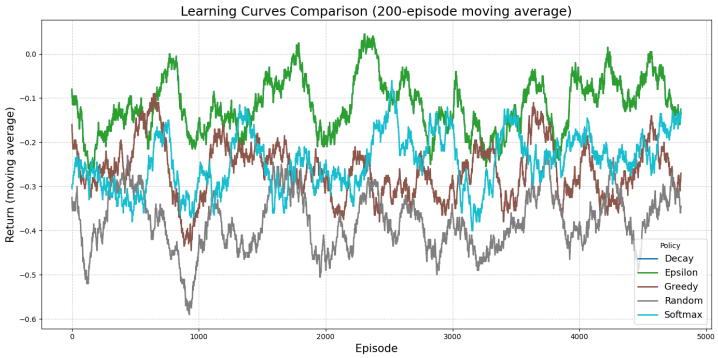

Figure 6: Blackjack Monte Carlo

Monte Carlo: Performance by Policy

- **Epsilon-Greedy Policy:** Achieves the highest overall performance. The learning curve displays distinct peaks, reflecting successful exploitation of high-reward trajectories. It also stabilizes over time, indicating effective generalization and convergence.
- **Softmax Policy:** Performs moderately well, outperforming both Greedy and Random policies. However, it does not reach the performance level of Epsilon-Greedy, suggesting a slower or less effective learning process.
- **Greedy and Random Policies:** Both perform poorly in this environment. The Greedy policy fails due to lack of exploration, potentially getting stuck in suboptimal strategies, while the Random policy, as expected, lacks any directed learning and fails to stabilize or improve.

5 Conclusion

This project provided a practical opportunity to evaluate two distinct Gymnasium environments using various reinforcement learning algorithms. By exploring both continuous and discrete domains, we gained insights into the strengths and limitations of different RL strategies.

For the **Blackjack** environment, the Monte Carlo method proved to be the most effective overall, delivering the highest and most stable performance. Q-Learning and SARSA with a greedy policy also achieved promising results, indicating their potential when accompanied by well-tuned parameters. In contrast, the random policy consistently underperformed across all algorithms, as expected due to its lack of directed exploration or learning.

For the **Pendulum-v1** environment, both SARSA and Q-Learning paired with a greedy policy yielded good results. However, Q-Learning demonstrated greater stability and convergence toward an optimal strategy, likely due to its off-policy nature and higher discount factor.

In summary, this project highlighted the significance of selecting appropriate algorithms and policies, as well as the critical role of hyperparameter tuning. These factors had a substantial impact on performance and learning outcomes and are essential considerations in any real-world reinforcement learning application.

6 Annex A – References

1. Sutton, R. S., & Barto, A. G. *Reinforcement Learning: An Introduction.*
   The foundational textbook covering all major RL concepts including Q-Learning, SARSA, and Monte Carlo methods.
   Free PDF (official): `http://incompleteideas.net/book/RLbook2020.pdf`

2. **Gymnasium Documentation (Successor to OpenAI Gym)**
   Official documentation for all environments including `Blackjack-v1` and `Pendulum-v1`.
   `https://gymnasium.farama.org/`

3. Alpalhão, N. (2025). *Reinforcement Learning – Lecture Slides and Practical Class Notebooks.*
   Unpublished course material, Master in Data Science and Advanced Analytics, NOVA IMS.