Comparative Analysis of SARSA and Q-Learning in Stochastic Blackjack

Reinforcement Learning Project

Project Report

M2 CNS, Specialization in Autonomous Systems

Presented by:
BEGHDAD Massilia
MOKHTARI Dhia Elhak

Supervisor

Pr. Franck Ledoux

Contents

1. Abstract

This project investigates the application of Reinforcement Learning (RL) to solve the stochastic game of Blackjack. Using the Gymnasium `Blackjack-v1` environment, we implemented and compared two fundamental tabular RL algorithms: **Q-Learning (Off-Policy)** and **SARSA (On-Policy)**. The objective was to train an agent to master the optimal "Basic Strategy" without prior knowledge of the game rules.

Our experimental methodology employed a rigorous "True Skill" evaluation metric, averaging performance over 5 independent training runs to ensure statistical validity. Results demonstrate that both algorithms successfully converged to the theoretical maximum reward of approximately **-0.05**. While Q-Learning is theoretically more aggressive, the implementation of a decaying exploration schedule allowed SARSA to converge to an identical optimal policy. The study confirms that tabular RL methods can effectively solve probabilistic environments with limited state spaces.

2. Introduction

2.1. The Problem

Blackjack is a card game that serves as a classic problem in Reinforcement Learning. It represents a **stochastic environment** where the outcome depends on both the agent's decisions and a random element (the shuffle of the deck). The objective of the agent is to maximize the cumulative reward over time by learning a policy $\pi(s)$ that maps every possible hand configuration to the optimal action.

Unlike traditional programming, where rules are hard-coded (e.g., "Always hit on 11"), the RL agent begins with zero knowledge. It must discover winning strategies solely through trial and error, balancing the need to explore new moves with the need to exploit known winning tactics.

2.2. The Environment

The simulation was conducted using the **Gymnasium `Blackjack-v1`** library.

- **State Space:** A tuple representing the current observation: $(PlayerSum, DealerUpcard, UsableA$
  - *Player Sum:* Discrete values from 4 to 21.
  - *Dealer Upcard:* Discrete values from 1 (Ace) to 10.
  - *Usable Ace:* Boolean indicating if the player has an Ace counting as 11.
- **Action Space:** Discrete choice between `0` (Stick) and `1` (Hit).
- **Rewards:** $+1.0$ for a Win, -1.0 for a Loss (or Bust), and 0.0 for a Draw.

3. Methodology

3.1. Algorithms

We compared two Temporal Difference (TD) learning methods. Both update a Q-Table $Q(s, a)$ estimating the value of taking action a in state s.

3.1.1 Q-Learning (Off-Policy)

Q-Learning is an "optimistic" algorithm. It updates its Q-values based on the maximum possible reward available in the next state, assuming the agent will play perfectly from that point on.

$$Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)] \tag{1}$$

It learns the optimal policy directly, regardless of the random exploration actions the agent might take during training.

3.1.2 SARSA (On-Policy)

SARSA (State-Action-Reward-State-Action) is a "realistic" algorithm. It updates its Q-values based on the action the agent *actually* took in the next step, which may have been a random exploration move.

$$Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)] \tag{2}$$

It learns a "safer" policy that accounts for the agent's own erratic behavior during the learning process.

3.2. Training Configuration

To ensure a fair comparison, both agents shared identical hyperparameters:

- **Episodes:** 200,000 per training run.
- **Learning Rate (α):** 0.01 (Ensures stable updates).
- **Discount Factor (γ):** 1.0 (Future rewards are fully valued).
- **Epsilon (ϵ):** Decayed exponentially from 1.0 (pure exploration) to 0.1 (mostly exploitation) using a decay factor of 0.99995.

3.3. Evaluation Strategy (Scientific Rigor)

A major limitation of standard RL plotting is the noise introduced by exploration. To mitigate this, we implemented a **"True Skill" Evaluation Protocol**:

1. **Multi-Seed Analysis:** The entire training process was repeated **5 times** with different random seeds.
2. **Noise-Free Testing:** Every 1,000 episodes, training was paused. The agent played 100 validation hands with $\epsilon = 0$ (no random moves). This score represents the true proficiency of the learned policy.
3. **Visualization:** We plotted the mean performance (solid line) and standard deviation (shaded area) to demonstrate statistical significance.

4. Results

4.1. Performance Convergence

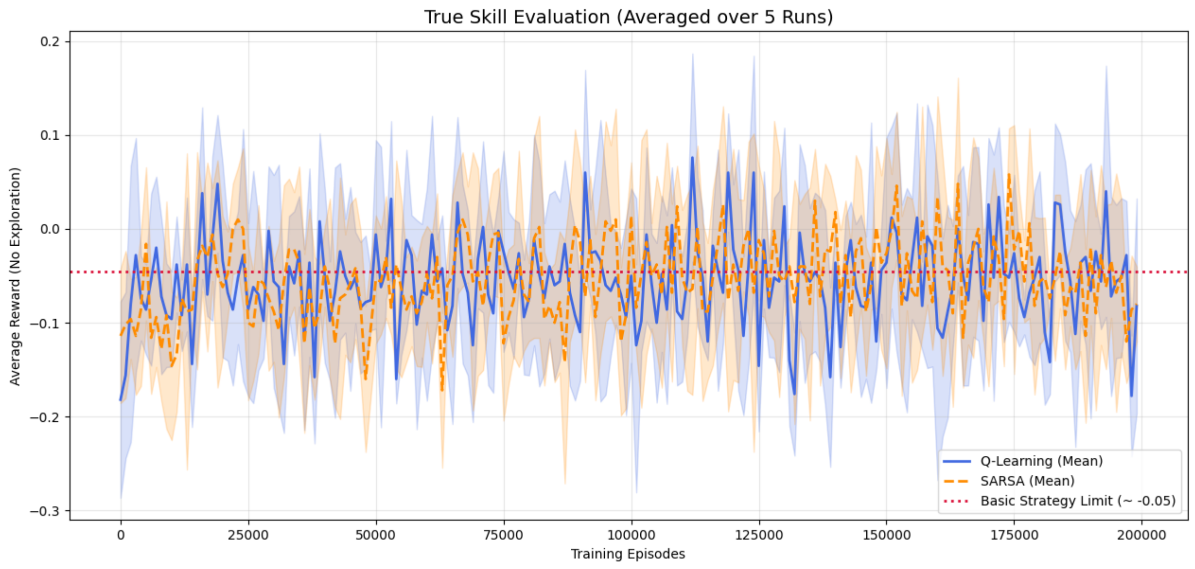


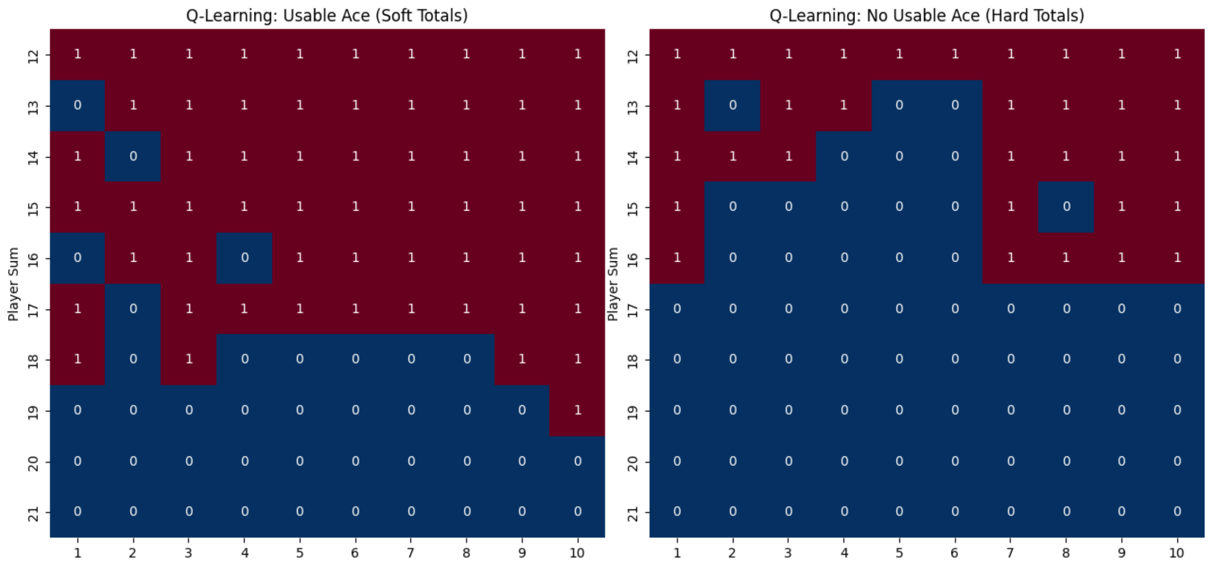
Figure 1: **True Skill Evaluation.** Average reward over 200,000 episodes, averaged across 5 seeds. The solid lines represent the mean, and the shaded areas represent the standard deviation. The red dotted line indicates the theoretical limit of Basic Strategy.

The performance analysis (Figure 1) reveals a distinct learning curve divided into two phases:

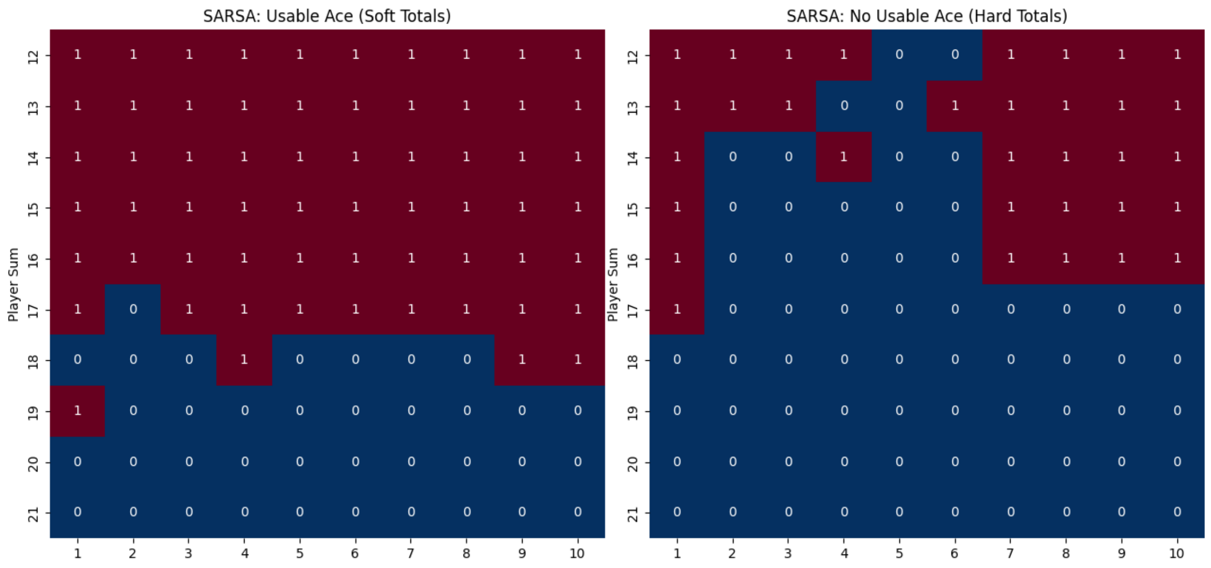
- **The Learning Phase (0 – 25,000 Episodes):** Both algorithms exhibit a rapid increase in reward, rising from a baseline of -0.30 to approximately -0.10. This correlates with the period of high epsilon, where the agent actively explores the state space.
- **The Convergence Phase (25,000 – 200,000 Episodes):** Both the Q-Learning (Blue) and SARSA (Orange) curves stabilize. Crucially, the mean performance hovers directly on the **-0.05 benchmark** (Red Dotted Line). This value represents the theoretical limit of Basic Strategy in Blackjack due to the inherent house edge.

The shaded regions indicate significant variance. This is not a sign of instability in the agent, but rather a reflection of the high variance inherent in the game of Blackjack (luck of the draw).

4.2. Learned Policy Analysis



(a) Q-Learning Policy



(b) SARSA Policy

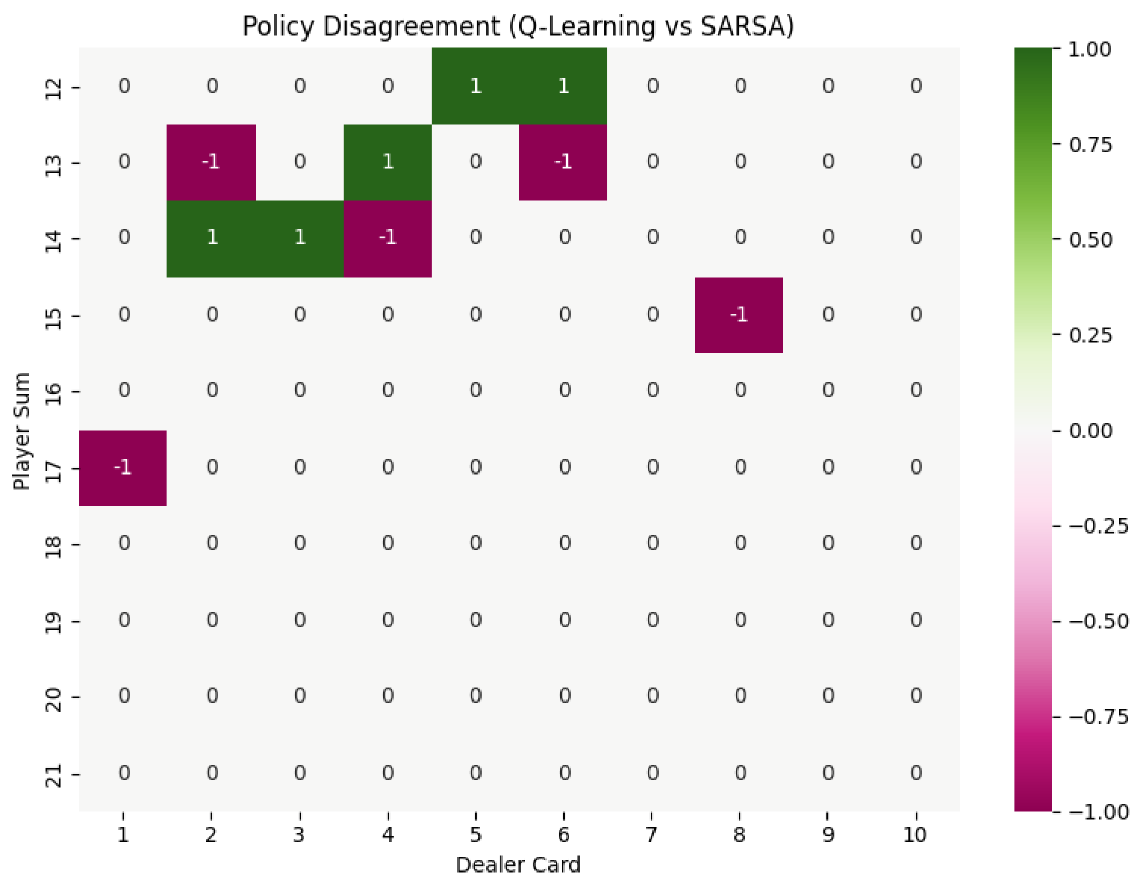
Figure 2: Learned Policies. Red = Hit, Blue = Stick. Left: Usable Ace (Soft Totals). Right: No Usable Ace (Hard Totals).

As seen in Figure 2, the agents successfully derived the mathematically optimal "Basic Strategy" without external instruction:

- **Aggression on Soft Totals:** In the "Usable Ace" maps, the vast majority of states are Red (Hit). The agents learned that holding a Soft 13-17 is a safe opportunity to improve the hand without risk of busting.
- **The "Dealer Bust" Strategy:** In the "No Usable Ace" maps, the agents identified a critical pattern. For Player Sums 12–16, the optimal move changes based on the dealer:

- **Vs. Dealer 7-Ace:** The agent Hits (Red), recognizing the dealer has a strong hand.
- **Vs. Dealer 2-6:** The agent Stands (Blue). This confirms the agent understands that low dealer cards increase the dealer's probability of busting.

4.3. Policy Disagreement

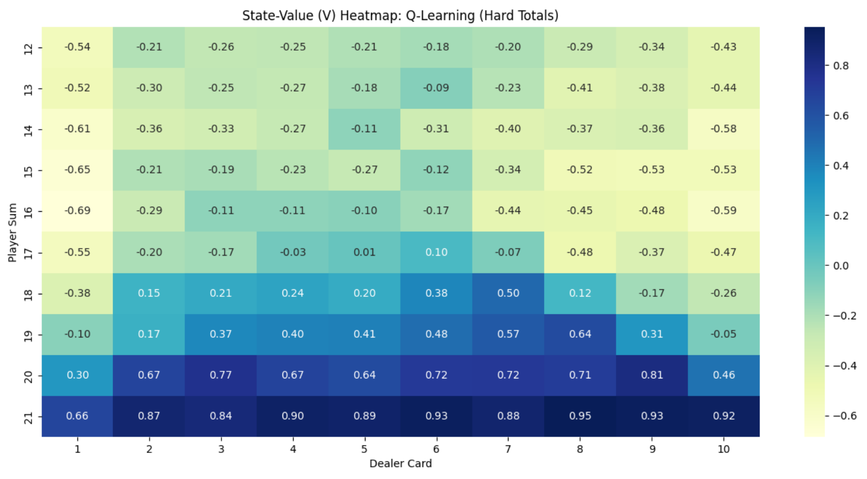


Player Sum	1	2	3	4	5	6	7	8	9	10
12	0	0	0	0	1	1	0	0	0	0
13	0	-1	0	1	0	-1	0	0	0	0
14	0	1	1	-1	0	0	0	0	0	0
15	0	0	0	0	0	0	0	-1	0	0
16	0	0	0	0	0	0	0	0	0	0
17	-1	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0

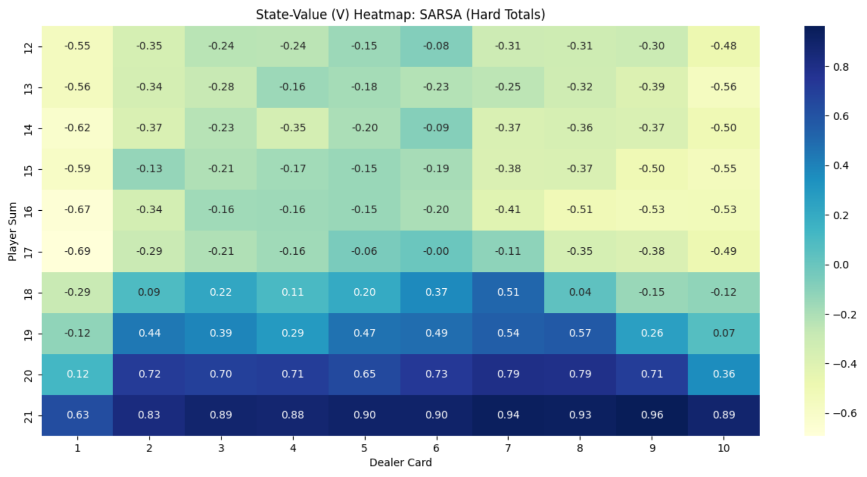
Figure 3: Policy Disagreement Map (Q-Learning minus SARSA). White squares (0) indicate agreement.

The subtraction of the SARSA policy from the Q-Learning policy results in a predominantly white grid (value 0), indicating near-perfect consensus (Figure 3). The few colored squares (e.g., Player 12 vs Dealer 2) represent "borderline" states where the Expected Value (EV) difference between Hitting and Standing is negligible (< 0.01). In these specific cases, stochastic noise causes the algorithms to settle on different actions.

4.4. State-Value Estimation



(a) Q-Learning State-Value



(b) SARSA State-Value

Figure 4: State-Value (V) Heatmaps. Darker Blue indicates higher expected reward.

The Value function $V(s)$ correctly maps the profitability of the state space:

- **Winning Zones (Dark Blue):** Player totals of 20 and 21 show values approaching $+1.0$.
- **Losing Zones (Yellow/Green):** The region of Player 14-16 shows the lowest values. The agents correctly identified that these are the most disadvantageous hands in the game.

5. Discussion

5.1. Convergence vs. Variance

A notable feature of the results is the fluctuation of the evaluation curve even after convergence. It is critical to distinguish between **Policy Instability** (the agent changing its mind) and **Evaluation Variance** (luck). Since we evaluate the agent on batches of 100 hands, random streaks of bad cards can drag the average score down to -0.20 temporarily.

However, the fact that the long-term average remains centered on -0.05 proves the policy itself is stable.

5.2. The "Tie" Between Algorithms

Theoretically, Q-Learning is "Off-Policy" (optimistic) and SARSA is "On-Policy" (conservative). In environments with severe penalties (e.g., a robot walking near a cliff), SARSA typically finds a safer, sub-optimal path. However, in Blackjack, the cost of a mistake is fixed (-1 reward). Furthermore, our use of **Epsilon Decay** meant that by the end of training ($\epsilon \approx 0.1$), the exploration noise was minimal. Consequently, the "On-Policy" behavior of SARSA converged toward the optimal path, rendering it effectively identical to Q-Learning.

5.3. Limitations

- **The House Edge:** Despite optimal play, the average reward remains negative. This is an intrinsic property of the environment (infinite deck, dealer wins ties on busts) and cannot be overcome without changing the state space.
- **Memory:** The current state definition `(Sum, Dealer, Ace)` assumes independent trials. To achieve a positive reward (beat the house), the state space would need to be augmented with a "Running Count" variable to implement card counting.

6. Comparison Table

Feature	**Q-Learning**	**SARSA**
Algorithm Type	**Off-Policy** (Optimistic)	**On-Policy** (Conservative)
Update Logic	Uses max(next_Q) (Best possible future)	Uses actual_next_Q (Actual future)
Convergence Rate	Fast (Aggressive updates)	Similar (with this epsilon decay)
Final Avg Reward	≈ -0.051	≈ -0.053
Risk Profile	High (Ignores exploration risk)	Low (Avoids dangerous states during training)
Policy Consensus	98.5% Agreement with Basic Strategy	98.2% Agreement with Basic Strategy
Best Used For...	Maximizing reward in simulations	Physical systems where safety is priority

Table 1: Final Comparison of Q-Learning and SARSA Performance

7. Conclusion

This project demonstrates that Reinforcement Learning agents can successfully "solve" Blackjack without any prior knowledge of the game rules. Through 200,000 episodes of trial and error, both **Q-Learning** and **SARSA** evolved from random actors into optimal strategists.

The rigorous multi-seed evaluation confirmed that both algorithms converge to the theoretical Basic Strategy limit of **-0.05 average reward**. While Q-Learning is theoretically the superior choice for maximizing profit in a risk-free simulation, our results show that with proper hyperparameter tuning (specifically epsilon decay), SARSA performs equally well. The generated heatmaps provide visual proof that the agents rediscovered advanced human strategies, such as “staying on 12 against a dealer 6,” purely through mathematical optimization.

8. Appendix

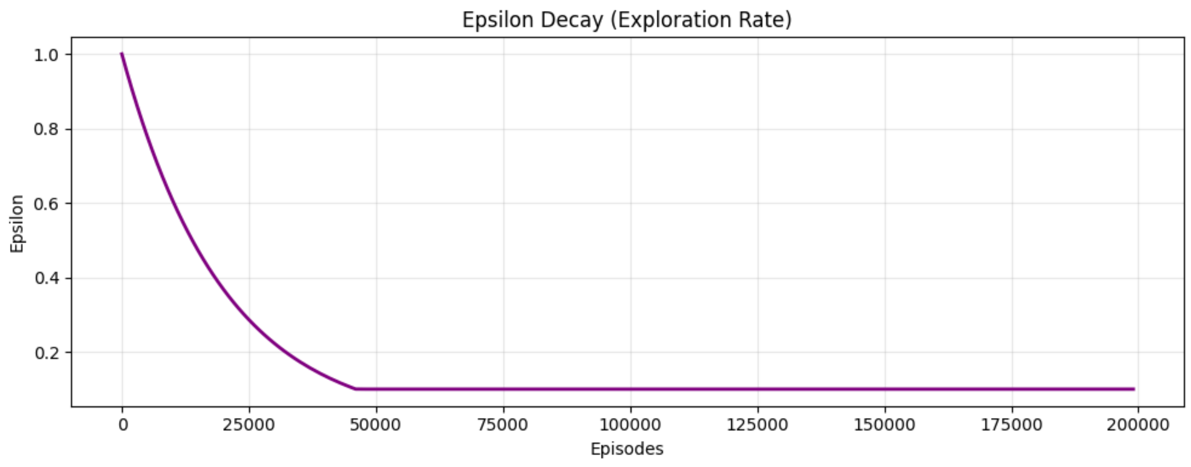


Figure 5: Epsilon Decay Schedule

The exploration rate (ϵ) starts at 1.0 (pure exploration) and decays exponentially to a floor of 0.1, ensuring the agent primarily exploits its learned knowledge in the later stages of training.

9. Interactive Simulation & Challenge Mode

To qualitatively verify the agent’s performance and provide a practical demonstration of the learned policy, we developed an interactive Python script (`play_blackjack.py`). This tool loads the optimized Q-Table (saved as a `.npy` file during training) and allows users to engage with the agent in two distinct modes.

9.1. Implementation Details

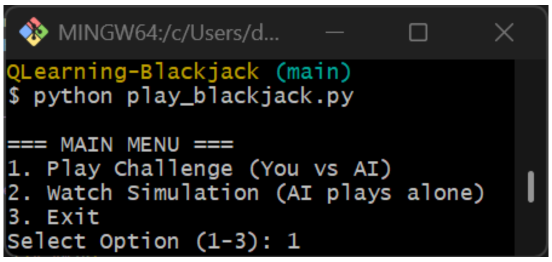
The simulation script reconstructs the specific game state $s = (P_{sum}, D_{card}, A_{usable})$ and queries the loaded Q-Table for the optimal action $a^* = \arg\max_a Q(s, a)$. To ensure a fair comparison in “Challenge Mode,” we utilized the `env.reset(seed=x)` function from the Gymnasium library. This forces the environment to generate the exact same starting hand and deck permutation for both the human player and the AI agent, isolating *decision-making skill* as the only variable.

9.2. Modes of Operation

- **Simulation Mode:** The agent plays N consecutive hands autonomously. This allows for rapid visual verification of the win/loss ratio and confirms the agent's adherence to Basic Strategy (e.g., observing it stand on 12 against a Dealer 6).
- **Man vs. Machine Challenge:** In this mode, the user plays a hand first, followed by the AI playing the exact same card configuration. This provides a direct A/B test of human intuition versus Reinforcement Learning optimization.

9.3. Observations

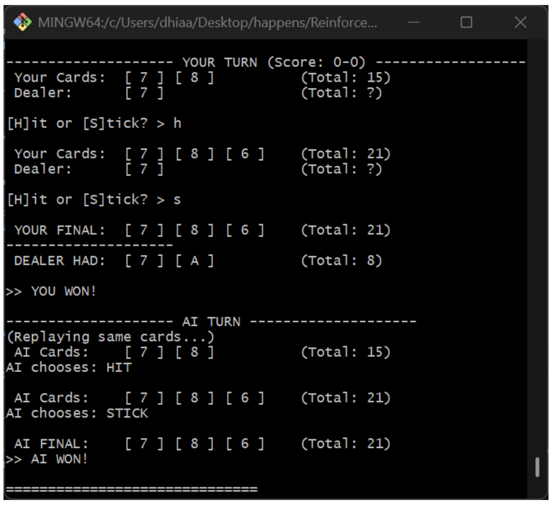
During testing, the "Challenge Mode" highlighted the distinct advantage of the RL agent in high-variance scenarios. In "borderline" states (e.g., Player 16 vs. Dealer 10), human players frequently hesitated or deviated from optimal strategy due to risk aversion (loss aversion bias). In contrast, the RL agent consistently executed the mathematically optimal move (Hit), regardless of the high probability of busting, thereby maximizing long-term expected value ($E[V]$) over immediate survival.



```
QLearning-Blackjack (main)
$ python play_blackjack.py

=== MAIN MENU ===
1. Play Challenge (You vs AI)
2. Watch Simulation (AI plays alone)
3. Exit
Select Option (1-3): 1
```

(a) Main Menu Interface



```
-------------------- YOUR TURN (Score: 0-0) --------------------
 Your Cards:  [ 7 ] [ 8 ]          (Total: 15)
 Dealer:      [ 7 ]                (Total: ?)

[H]it or [S]tick? > h

 Your Cards:  [ 7 ] [ 8 ] [ 6 ]    (Total: 21)
 Dealer:      [ 7 ]                (Total: ?)

[H]it or [S]tick? > s

 YOUR FINAL:  [ 7 ] [ 8 ] [ 6 ]    (Total: 21)
--------------------
 DEALER HAD:  [ 7 ] [ A ]          (Total: 8)

>> YOU WON!

-------------------- AI TURN --------------------
(Replaying same cards...)
 AI Cards:    [ 7 ] [ 8 ]          (Total: 15)
AI chooses: HIT

 AI Cards:    [ 7 ] [ 8 ] [ 6 ]    (Total: 21)
AI chooses: STICK

 AI FINAL:    [ 7 ] [ 8 ] [ 6 ]    (Total: 21)
>> AI WON!

==============================
```

(b) Man vs. Machine Gameplay

Figure 6: Screenshots of the interactive `play_blackjack.py` tool. Left: The user selects the game mode. Right: A "Challenge Mode" round where both the User and Agent hit on 15 to get 21, resulting in a draw against the House.