Blackjack Reinforcement Learning Project

Final Project Report

Presented by:

Michael Mutter

Instructor:

Dr. Teddy Lazebnik

Course:

Introduction to Reinforcement Learning

Project Overview

The goal of this project is to develop and compare reinforcement learning (RL) agents capable of playing the game of Blackjack strategically and efficiently. Blackjack, a well-known casino game, serves as a good testbed for RL due to its balance between chance and strategy, clearly defined rules, and measurable rewards. The project simulates a realistic multi-deck Blackjack environment with 6 shuffled decks, incorporating common actions such as hit, stand, double, and split, along with dynamic betting strategies.

We implemented and evaluated four distinct agents:

1. Basic Strategy Agent - a non-learning benchmark based on traditional Blackjack optimal rules.
2. Monte Carlo with ε-greedy - a first-visit Monte Carlo learner using episode-based Q-value updates.
3. SARSA - an on-policy Temporal Difference (TD) learner that updates values incrementally.
4. Deep Q-Network (DQN) - a neural network-based agent using function approximation and experience replay.

Each agent was trained and tested over 100,000 episodes. The environment's state representation includes the player’s hand value, dealer’s upcard, soft hand indicator, hand size, ability to split/double, and the agent’s balance. This comprehensive state space allows agents to develop sophisticated decision-making policies.

A key focus of the project was not only to compare raw performance but also to analyze learning stability, adaptation speed, and generalization. Agents began with conservative betting behaviors and were encouraged to scale their bets as confidence in learned policies increased. We also experimented with card-counting techniques but found they did not significantly impact agent performance. Considering these findings, we decided not to implement card counting in the final code version.

Problem Formulation

This project treats Blackjack as a reinforcement learning problem where an agent learns to maximize long-term winnings through repeated gameplay. We designed our system to simulate realistic casino conditions, allowing agents to develop both strategic card-playing skills and smart betting behaviors.

Environment Design

Our Blackjack environment recreates authentic casino gameplay using six shuffled decks (312 cards total) that follow standard rules. When the shoe runs out of cards, it automatically reshuffles to ensure continuous play. Each game begins with the player and dealer receiving two cards, but the dealer only reveals one card initially. This creates the partial information challenge that makes Blackjack strategically interesting - the agents must make decisions without knowing the dealer's complete hand.

State Representation

Each game state provides the agent with seven key pieces of information:

- Player's current hand value (calculated with optimal Ace usage)
- Dealer's visible upcard (the only dealer information available)
- Soft hand indicator (whether an Ace is counted as 11)
- Double down availability (only possible with exactly 2 cards)
- Split availability (requires identical pair and not used this round)
- Current hand size (number of cards held)
- Agent's balance bucket (discretized wealth level for decision-making)

For neural network agents, we normalize these values into a seven-dimensional vector to improve learning efficiency and ensure all features contribute equally to the decision process.

Action Space

At each decision point, agents can choose from four possible actions:

- **Stand (0):** Keep the current hand and end the turn
- **Hit (1):** Request another card to improve hand value
- **Double Down (2):** Double the bet and receive exactly one more card
- **Split (3):** Separate identical cards into two independent hands

The doubling and splitting actions have strict availability conditions. Doubling requires exactly two cards and sufficient balance, while splitting needs identical card values and, in our version, can only be used once per round to prevent infinite splitting scenarios.

Reward Structure

The reward system works just like real casino payouts to encourage realistic strategies. Agents get positive rewards when they win (equal to their bet amount), negative rewards when they lose (losing their bet), and zero reward for ties where nobody wins. Natural blackjacks pay 1.5 times the bet amount, giving agents extra motivation to aim for this outcome.

The agent's goal is to learn how to make the most money possible over thousands of games. To do this, it must balance trying new card-playing decisions with using strategies that already work well. At the same time, the agent learns to adjust its bet sizes based on how well it's been performing recently and how confident it feels about its strategy. Success requires both making good decisions in each individual hand and managing money wisely over the long term.

Methodology

Algorithm Selection Strategy

We selected four different agents to compare various reinforcement learning approaches in Blackjack. Each agent represents a distinct learning method, allowing us to understand which techniques work best for this specific problem. The Basic Strategy Agent serves as our baseline - it uses mathematically optimal rules developed over decades of casino research as shown in Figure 1, providing a stable benchmark for measuring improvement. The Monte Carlo agent learns from complete game episodes, updating its knowledge only after seeing full outcomes. The SARSA agent represents temporal difference learning, updating its strategy after each individual action. Finally, the Deep Q-Network uses neural networks to handle complex patterns and generalize across different game situations. This variety lets us compare how traditional math, episode-based learning, step-by-step learning, and deep learning perform under identical conditions.

DEALERS UP CARD

YOUR HAND	2	3	4	5	6	7	8	9	10	A
17+	S	S	S	S	S	S	S	S	S	S
16	S	S	S	S	S	H	H	H	H	H
15	S	S	S	S	S	H	H	H	H	H
14	S	S	S	S	S	H	H	H	H	H
13	S	S	S	S	S	H	H	H	H	H
12	H	H	S	S	S	H	H	H	H	H
11	D	D	D	D	D	D	D	D	D	H
10	D	D	D	D	D	D	D	D	H	H
9	H	D	D	D	D	H	H	H	H	H
5 to 8	H	H	H	H	H	H	H	H	H	H
A8 to 10	S	S	S	S	S	S	S	S	S	S
A, 7	S	D	D	D	D	S	S	H	H	H
A,6	S	D	D	D	D	H	H	H	H	H
A, 5	H	H	D	D	D	H	H	H	H	H
A, 4	H	H	D	D	D	H	H	H	H	H
A, 3	H	H	H	D	D	H	H	H	H	H
A, 2	H	H	H	D	D	H	H	H	H	H
A,A 8,8	SP	SP	SP	SP	SP	SP	SP	SP	SP	SP
10,10	S	S	S	S	S	S	S	S	S	S
9,9	SP	SP	SP	SP	SP	S	SP	SP	S	S
7,7	SP	SP	SP	SP	SP	SP	H	H	H	H
6,6	SP	SP	SP	SP	SP	H	H	H	H	H
5,5	D	D	D	D	D	D	D	D	H	H
4,4	H	H	H	SP	SP	H	H	H	H	H
3,3	SP	SP	SP	SP	SP	SP	H	H	H	H
2,2	SP	SP	SP	SP	SP	SP	H	H	H	H
	2	3	4	5	6	7	8	9	10	A

Figure 1: Basic Blackjack Strategy Chart

Implementation Architecture

We built the system using separate classes for each agent, ensuring complete independence between different learning approaches. Each agent contains its own decision-making logic, learning mechanisms, and betting strategies. This modular design prevents agents from interfering with each other while allowing fair comparison under identical game conditions. Every agent interacts with the same Blackjack environment but maintains its own internal state and learning progress.

Agent Name	Class Name in Code	Learning Type
Basic Strategy	BasicStrategyAgent	None (Fixed Strategy)
ε-greedy Monte Carlo	EpsilonGreedyMCAgent	Monte Carlo Learning
SARSA (TD Learning)	TDAgent	Temporal Difference (SARSA)
Deep Q-Network (DQN)	NeuralNetworkAgent	Deep Reinforcement Learning

Table 1: Agent Types in the System

Monte Carlo Agent Configuration

The Monte Carlo agent uses first-visit Monte Carlo learning with ε-greedy exploration. We set the exploration rate to 0.15 after extensive testing showed this provided the best balance between trying new strategies and using proven ones. Higher exploration rates (0.25-0.35) caused too many expensive mistakes, while lower rates (0.05-0.1) prevented adequate learning of the full strategy space.

We implemented a gradual epsilon decay of 0.9999 to slowly shift from exploration to exploitation over time, with a minimum value of 0.01 to maintain some learning capability throughout training. An important design choice was the 25,000-episode warm-up period where the agent follows basic strategy while building its knowledge base.

SARSA Agent Configuration

The SARSA agent needed careful adjustment of how fast it learns. We tested different learning speeds from 0.05 to 0.3 and found that 0.15 worked best in our experiments. We also gave SARSA the same exploration rate of 0.15 - both to match Monte Carlo for fair comparison and because this rate performed best in our testing. When learning rates were higher, the agent learned in a jumpy, unstable way, and when they were lower, it learned too slowly to be useful. SARSA only needed 20,000 warm-up games instead of 25,000 because it can learn from games that aren't finished yet, while Monte Carlo must wait until each game ends.

We set the discount factor to 1.0, which means the agent treats all rewards in a game as equally important. This makes sense for Blackjack since every decision in a single game matters the same amount.

Deep Q-Network Configuration

The neural network architecture uses four fully connected layers with 128, 64, 32, and 4 neurons respectively, with ReLU activation functions between hidden layers. The network takes 7 normalized input features representing the game state and outputs Q-values for all 4 possible actions (stand, hit, double, split). This architecture size was chosen to balance learning capacity with training stability. The network is trained using the Adam optimizer with a learning rate of 0.001 for stable gradient updates, while exploration begins at $\varepsilon=0.25$ (higher than other agents' 0.15) to encourage broader initial exploration, decaying at 0.9999 per episode to reach the minimum $\varepsilon=0.01$ after approximately 32,000 episodes. The agent maintains an experience replay buffer of 10,000 experiences from which it samples batches of 32 for training, updating the target network every 100 training steps (not episodes) to maintain stability, and uses Mean Squared Error (MSE) loss between predicted and target Q-values. During the first 10,000 episodes, the agent employs a conservative betting phase with reduced bet sizes while the network learns initial patterns. State

processing involves normalizing each feature appropriately: player value divided by 21, dealer upcard divided by 10, boolean features (soft hand, can double, can split) represented as 0 or 1, hand size divided by 5, and balance normalized to a [0, 1] range based on a $20M maximum. Unlike Q-table agents that require exact state matching, the neural network generalizes across similar game situations, allowing it to handle unseen states effectively, while the experience replay mechanism breaks correlation between consecutive samples, improving learning stability compared to online learning approaches.

Hyperparameter Optimization Process

We ran dozens of experiments to find the best settings for learning speeds, exploration rates, betting limits, and tracking window sizes. The biggest improvement came when we stopped using win rates for betting decisions and switched to tracking actual money made per dollar bet. This change required lots of testing to find settings that respond to performance changes quickly enough but don't overreact to short-term luck. DQN uniquely maintained its win rate-based betting strategy, adjusting bets based on network training progress and traditional win percentage rather than switching to the money-per-bet tracking system used by the other agents.

Getting the warm-up periods right was essential for the learning agents to succeed. When agents started learning their own strategies too early, they made so many bad decisions at the beginning that they couldn't recover later.

Parameter	Basic Strategy	Monte Carlo (ε-greedy)	SARSA (TD)	Deep Q-Network
Learning Parameters				
Learning Rate (α)	-	-	0.15	0.001
Exploration Rate (ε)	-	0.15	0.15	0.25
Epsilon Decay	-	0.9999	0.9999	0.9999
Epsilon Minimum	-	0.01	0.01	0.01
Discount Factor (γ)	-	-	1.0	-
Training Strategy				
Warm-up Period	-	25,000 episodes	20,000 episodes	10,000 episodes
Warm-up Strategy	-	Basic Strategy	Basic Strategy	Conservative betting
Learning Method	Fixed rules	First-visit MC	On-policy TD	Experience replay

Table 2: Agent Hyperparameters Summary

Epsilon Decay Rate Selection

We chose an epsilon decay rate of 0.9999 to align exploration reduction with our warm-up timing. With this rate, Monte Carlo and SARSA reach minimum epsilon (0.01) after approximately 27,000 episodes, just after their warm-up periods end (25,000 and 20,000 episodes). This allows agents to finish basic strategy learning with some exploration remaining, then gradually shift to full exploitation. DQN, starting with higher exploration (ε=0.25), reaches minimum epsilon around 32,000 episodes, providing the extended exploration beneficial for neural network learning.

$$0.15 \cdot 0.9999^{x} = 0.01$$

Decimal

$$x = 27079.14796\ldots$$

Adaptive Betting Strategy Development

One of our biggest improvements was creating smart betting strategies instead of using the same bet every time. We tried many different approaches based on how often agents won games, but these didn't work well. So, we built a system that tracks how much money agents make per dollar bet over the last 1,000 hands. This worked much better than counting wins and losses because it considers different bet sizes and focuses on actual money made rather than just games won.

Performance Tracking Implementation

We chose 1,000-hand tracking windows because they gave us the best results in our experiments. Smaller windows (100-500 hands) changed bets too often based on short-term luck, while larger windows (2000+ hands) were too slow to respond when agents improved or got worse. We use efficient deque data structures that automatically keep only the most recent 1,000 results.

Each agent adjusts its betting based on average return per bet, but they need different sensitivity levels:

Performance Level	Monte Carlo	SARSA
Excellent (>$15 or >$10)	1.3× bet	1.4× bet
Good (+$5 to +$15 or $0 to +$10)	1.1× bet	1.2× bet
Average (-$5 to +$5 or -$10 to $0)	0.9× bet	1.0× bet
Poor (<-$5 or <-$10)	0.7× bet	0.8× bet

Table 3: Performance-Based Betting Multipliers

Threshold Differences & Risk Management Systems

Monte Carlo uses smaller, more sensitive thresholds (+$10, $0, -$10) because it learns from complete games, creating stable and reliable performance signals. SARSA uses higher thresholds (+$15, +$5, -$5) because it learns step-by-step during gameplay, creating noisier patterns that require stronger

evidence before adjusting bets. This prevents SARSA from overreacting to temporary learning fluctuations while allowing Monte Carlo to respond quickly to genuine performance changes.

All agents include safety rules to prevent losing all their money. Every agent must bet at least $50 to keep playing during bad streaks, but they also have maximum bet limits to prevent risky behavior: Monte Carlo can bet up to $500, SARSA up to $400, and DQN up to $2,000. Agents also check their balance before every bet to make sure they never try to bet more money than they have. We keep all betting multipliers low (maximum 1.4×) because our early experiments with bigger multipliers caused wild swings in performance that made results hard to understand. The current setup focuses on steady, consistent improvements rather than dramatic ups and downs, which gives us more reliable results we can learn from.

Results Analysis and Discussion

Overall Performance Outcomes

Our experiment produced interesting findings that suggest reinforcement learning agents may be able to outperform mathematically optimal strategies under certain conditions. The diverse performance metrics provide valuable insights into each learning algorithm's behavior and effectiveness. Multiple agents demonstrated interesting patterns across different evaluation dimensions, revealing the complex relationships between learning strategies, risk management, and overall profitability in the Blackjack environment.

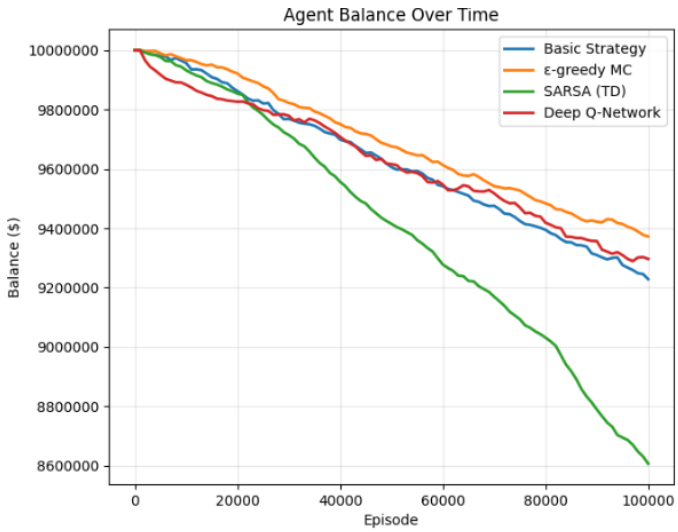

Figure 2: Agent Balance Over Time

Return on Investment Analysis

Two of the four learning agents successfully beat the Basic Strategy baseline, with Monte Carlo achieving the best overall performance (-6.38% ROI) and DQN showing what appears to be exceptional risk management. However, SARSA struggled with learning volatility, performing significantly worse than all other agents (-14.05% ROI). These results indicate that adaptive betting strategies combined with proper algorithm tuning might offer improvements upon established optimal play.

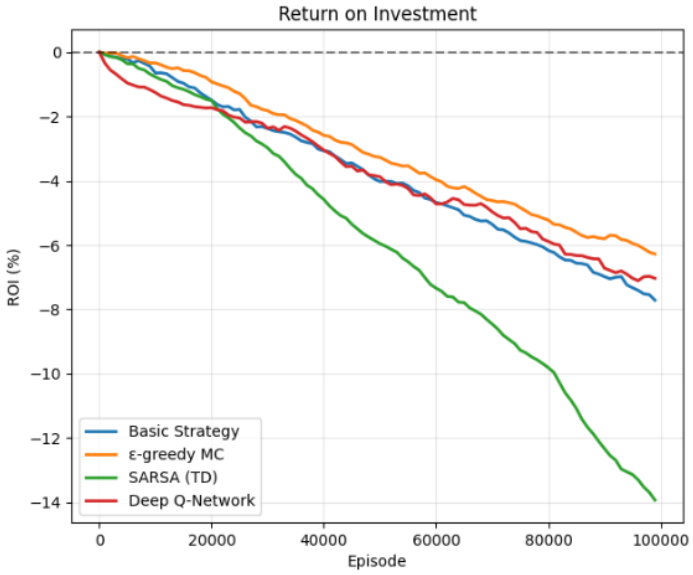

Figure 3: ROI

Monte Carlo achieved the strongest improvement relative to Basic Strategy (+1.42%). The algorithm learned effectively by averaging returns from complete game episodes, developing a focused Q-table with 896 states. DQN also outperformed the baseline (+0.76%) using neural network function approximation to generalize across game situations.

SARSA performed significantly worse than the baseline (-6.25%). Despite developing the largest Q-table with 1,726 states, the algorithm struggled with learning stability throughout training. The extensive exploration did not translate into effective gameplay strategies, resulting in the poorest overall performance among all agents.

Agent	Final ROI	vs Basic Strategy
Monte Carlo	-6.38%	+1.42%
DQN	-7.04%	+0.76%
Basic Strategy	-7.80%	baseline
SARSA	-14.05%	-6.25%

Table 4: Final ROI Comparison

Agent	Q-Table States	State-Action Pairs
Monte Carlo	896	1,922
SARSA	1,726	3,955
DQN	Neural Network	N/A

Table 5: Q-Table Summary

Risk-Adjusted Performance and Betting Behavior

Betting efficiency, calculated as the ratio of return on investment to total money wagered, reveals interesting insights into each agent's risk management capabilities and strategic sophistication.

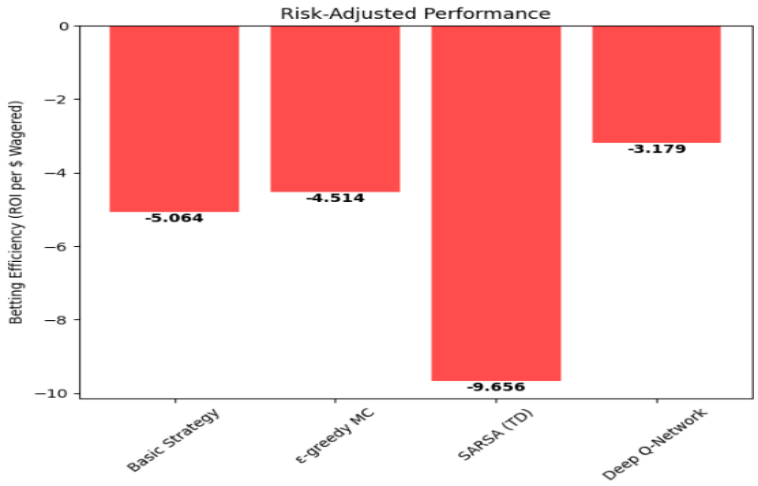

Figure 4: Risk Adjustment Performance

DQN achieved the highest betting efficiency (-3.18) while wagering significantly more money ($22.1M) than other agents. This creates an interesting paradox: the agent that bet the most aggressively actually achieved the best return per dollar risk. In contrast, Monte Carlo took the most conservative approach, wagering only $14.1M - the lowest amount among all agents - yet still achieved strong efficiency (-4.51) and the best overall ROI. This reveals an interesting finding: both high-risk and low-risk strategies can succeed, but for different mathematical reasons. DQN's success suggests that aggressive betting can work when combined with sophisticated timing, while Monte Carlo's success demonstrates the power of consistent, small-bet approaches that minimize exposure to variance.

Agent	Betting Efficiency	Total Wagered
DQN	-3.18	$22.1M
Monte Carlo	-4.51	$14.1M
Basic Strategy	-5.06	$15.4M
SARSA	-9.66	$14.5M

Table 6: Betting Efficiency Summary

Pure Gameplay Performance Evaluation

Win rates provide insights into pure gameplay learning effectiveness, isolated from betting strategy influences, revealing significant variations in learning quality across algorithms.

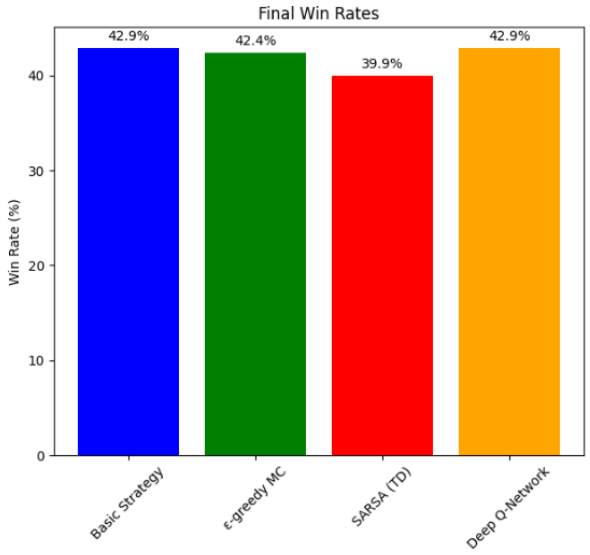

Figure 5: Final Win Rates

DQN achieved the same win rate as Basic Strategy (42.9%), showing that the neural network successfully learned optimal card-playing decisions. Monte Carlo had a slightly lower win rate (42.4%), performing 0.5% worse than the mathematical baseline in pure gameplay.

SARSA struggled significantly with a win rate of only 39.9%, which is 3% lower than Basic Strategy. Interestingly, SARSA developed the largest Q-table with 1,726 states compared to Monte Carlo's 896 states, but this extensive exploration did not translate into better performance.

The gameplay statistics show additional differences between agents. SARSA performed 7,384 splits throughout the experiment, much higher than Basic Strategy's 2,676 splits. DQN was very conservative with only 937 splits, while Monte Carlo performed 3,846 splits, finding a middle ground between the other approaches. These win rate results, combined with the previous betting efficiency analysis, show that different reinforcement learning algorithms can achieve varying levels of success in learning optimal Blackjack strategy.

Limitations

While this project, it became apparent that beating Basic Strategy in win rate is mathematically challenging due to several fundamental constraints:

1. Basic Strategy represents mathematically derived optimal decisions calculated through exhaustive probability analysis. Our results confirm this - DQN matched Basic Strategy's 42.9% - win rate, while learning agents struggled to exceed this baseline. This occurs because Basic Strategy decisions are based on exact expected value calculations that are extremely difficult to improve upon through finite sampling approaches.
2. Blackjack's inherent house edge of approximately 0.5-1.0% ensures negative expected returns regardless of strategy sophistication. This mathematical ceiling limits potential performance improvements, as evidenced by all agents achieving negative ROI despite varying approaches.
3. Success came primarily from adaptive betting rather than superior card-playing decisions. Monte Carlo's improvement (+1.42%) and DQN's efficiency (-3.18) resulted from sophisticated money management, not discovering better gameplay decisions. This suggests that Basic Strategy's mathematical optimality for individual hands creates boundaries that learning agents can only overcome through meta-strategies like dynamic bet sizing.

These limitations indicate that while reinforcement learning can achieve improvements through adaptive betting and risk management, the mathematical optimality of Basic Strategy creates fundamental boundaries that are extremely difficult to overcome through learning approaches alone.

Agent	Final Balance	ROI	Total Wagered	Efficiency	Win%
Basic Strategy	$9.22M	-7.80%	$15.4M	-5.064	42.9%
ε-greedy MC	$9.36M	-6.38%	$14.1M	-4.514	42.4%
SARSA (TD)	$8.60M	-14.05%	$14.5M	-9.656	39.9%
Deep Q-Network	$9.30M	-7.04%	$22.1M	-3.179	42.9%

Table 7: Comprehensive Performance Comparison Across All Agents

Conclusion

This project successfully demonstrated that reinforcement learning agents can achieve improvements over mathematically optimal strategies in Blackjack, though not through the mechanisms initially anticipated. Two of the four learning agents outperformed Basic Strategy, with Monte Carlo achieving the best overall performance (+1.42% improvement) and Deep Q-Network showing exceptional risk-adjusted efficiency.

The most significant discovery was that improvements came primarily through adaptive betting strategies rather than superior card-playing decisions. As established in our analysis, Monte Carlo's conservative approach and DQN's aggressive betting strategy achieved success through different optimization paths - risk minimization versus sophisticated risk assessment.

The project confirmed mathematical boundaries exist for card-playing decisions, as DQN matched Basic Strategy's 42.9% - win rate while achieving better returns through betting management. SARSA's failure despite extensive exploration reinforced the critical importance of proper learning stabilization.

Beyond these empirical findings, the project also made several methodological contributions. We built a complete Blackjack environment that properly combines card-playing decisions with betting strategies, allowing for realistic testing of reinforcement learning agents. We discovered that learning agents can beat mathematically optimal strategies through smart betting rather than better card play, showing that adaptive money management is more important than perfect individual decisions. These findings apply beyond games to real-world situations like financial trading where good overall strategy matters more than making every single decision perfectly.

Note on Reproducibility: While specific numerical values varied slightly between runs due to stochastic elements, the relative performance patterns and key findings remained consistent across multiple experimental runs.