Blackjack Reinforcement Learning Agent

Artem BRUDASTOV, Won Suk CHO, Marion PAULE

***Abstract*—In this project, reinforcement learning algorithms for blackjack are evaluated over different training durations. Expected SARSA achieves the highest and most stable win rates, while Q-learning and Double Q-learning remain competitive with sufficient training. The study highlights trade-offs in stability, sampling efficiency and training time and improves the understanding of efficient strategies in stochastic, episodic environments.**

I. Introduction

Reinforcement learning (RL) is under the umbrella of machine learning that utilizes on how agents should act in a certain given environment to maximize rewards through trial and error. This research will provide a specific focus on comparing and analyzing different RL algorithms. The environment that the agents will be tested on will be the blackjack game using the `gymnasium` library. The agents will be trained with the following algorithms. (1) Random Agent - for baseline comparison, (2) Monte Carlo, (3) Q-Learning, (4) Double Q-Learning, (5) State–action–reward–state–action (SARSA), (6) Expected SARSA, and (7) Q-Learning + Expected SARSA - Hybrid Agent. These implementations are expected to show the comparison between different algorithms and how it performs in the blackjack environment.

The game blackjack has a well-defined set of rules, stochastic and episodic. The most important part is that it is a ”house” or a non-player biased game, where the winning ratio is low for the player. These aspects make it particularly relevant for RL. The agent has to make decisions with incomplete information and should be able to make optimal decisions based on estimated value functions. Examining and testing these algorithms in blackjack should provide quantitative insights into convergence behavior, overestimation errors, and strategy optimization.

However, applying different RL algorithms to Blackjack has some challenges. (1) The stochastic nature or the randomness of the game complicates the learning process. (2) The agent should take into consideration both player's hand, and the dealer's face-up card in order to gauge the usability of different cards, which requires efficient exploration strategies. (3) The agents need to be trained on an infinite deck, where cards reset after each draw. This prevents the agent from using strategies such as card counting, which reduces realism but ensures consistency of state distributions. Therefore, the scope and limitations of our main study are mainly linked to the limitations of Open AI's Blackjack environment. Indeed, our agents are tested on infinite decks of cards of Open AI's, which prevents the exploration of different blackjack strategies such as card storage or card counting. Furthermore, our agents do not consider actions beyond “hit” and “stand”, such as doubling down or splitting pairs, which are integral parts of real blackjack deals. Therefore, as a bonus, we decided to adapt one of our agents (Q-Learning agent) on RLCard's finite deck environment to see the difference in training convergence.

A review of related research has dealt with RL in blackjack. Sutton and Barto introduced Expected SARSA with variance-reducing, and theoretical convergence approach. Van Hasselt [4] proposed Double Q-Learning to reduce the overestimation errors common in traditional Q-Learning. Geiser and Hasseler [2] applied SARSA and Q-learning to blackjack and tested the performance with simple strategy tables. Moreover, hybrid and double estimators was tested in stochastic environments to improve stability and accuracy, which can be seen in [3] and [8]. However, it is important to note that direct empirical comparisons of these algorithms under identical conditions are still limited.

This research implements and evaluates seven agents, which are (1) Random, (2) Monte Carlo, (3) Q-Learning, (4) Double Q-Learning, (5) State–action–reward–state–action (SARSA), (6) Expected SARSA, and (7) Q-Learning + Expected SARSA - Hybrid Agent in the same blackjack environment. We evaluate their performance in terms of win rate, convergence speed and stability. Our approach makes sure of the uniformity in testing, standardized reward structures, and consistent state representations that enables us to isolate algorithmic differences. Each agent was trained over different number of episodes to evaluate and compare its learning behavior.

Our final results show that Q-learning consistently achieves desirable win rates over varying numbers of episodes, however, it is important to note that its performance is comparable to other methods once different agents have stabilized. Although SARSA resulted to be the most effective algorithm in delivering the highest win rates with strong stability, especially over longer training sessions. Double Q-learning and SARSA show reliable and stable performance with minimal variance on increasing episodes. The hybrid approach combining Q-Learning and Expected SARSA is competitive, but does not significantly outperform the individual methods. Monte Carlo methods tend to have high variance in shorter runs, but gradually become a strong competitor with longer training, confirming their robustness with enough number of episodes. Overall, on-policy methods such as SARSA and Expected SARSA provide stable and conservative strategies, while off-policy methods such as Q-learning and Double Q-learning provide efficient learning and balanced performance, especially when the number of episodes is in the 10,000 mark.

The full source code and implementation details are available for reproducibility at the following repository:

https://colab.research.google.com/drive/1T0S0Lh8qI8rWQrsQW4RLFcuGqgx0RMm5

II. Background

In reinforcement learning (RL), a computer learns to make decisions by learning from mistakes [6]. At each time step t, the agent observes the current state $s_t \in \mathcal{S}$, chooses an action $a_t \in \mathcal{A}$ based on the strategy π, receives a reward r_{t+1} and moves to the state s_{t+1}. The goal is to find the optimal strategy π^* that maximizes the expected return, the discounted sum of future rewards:

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}, \tag{1}$$

where γ is the discount factor that determines the importance of future rewards.

Blackjack is chosen as the environment due to its randomness and its unique start and end [1]. We assume an infinite number of cards and remove rules such as double down. The state s consists of the player's hand, the dealer's visible card and whether the player has a useful ace. The actions a are hit (take a card) or stand (stay). The game ends when the player folds, stands or the dealer stops. The rewards are $+1$ for a win, -1 for a loss and 0 for a draw.

A. Value Functions and Policies

A policy $\pi(a|s)$ gives the probability of taking action a in state s. The action-value function $Q^\pi(s,a)$ estimates the expected return from state s, taking action a, and following policy π:

$$Q^\pi(s,a) = \mathbb{E}_\pi[G_t | s_t = s, a_t = a]. \tag{2}$$

The optimal action value function $Q^*(s,a)$ provides the highest expected return for each state-action pair. The optimal strategy π^* selects actions that maximize $Q^*(s,a)$.

B. Algorithms Overview

This project compares RL algorithms that update the action value function in different ways.

- **Random Agent**: Serves as a basis by selecting actions evenly at random.
- **Monte Carlo (MC)**: Estimates $Q(s,a)$ as the average return from complete episodes, updated only after each episode based on the total return G_t [6].
- **Q-Learning**: An off-policy Temporal Difference (TD) method updating $Q(s,a)$ toward the maximum next-state action value.

$$\begin{gathered} Q(s,a) \leftarrow \\ Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right], \end{gathered} \tag{3}$$

  where α is the learning rate [8].
- **SARSA**: An on-policy TD method updating $Q(s,a)$ using the next action taken.

$$\begin{gathered} Q(s,a) \leftarrow \\ Q(s,a) + \alpha \left[r + \gamma Q(s',a') - Q(s,a) \right], \end{gathered} \tag{4}$$

- **Expected SARSA**: A variance-reduced extension of SARSA that updates $Q(s,a)$ using the expected value of all next actions under the current policy [7].

$$\begin{gathered} Q(s,a) \leftarrow \\ Q(s,a) + \alpha \left[r + \gamma \textstyle\sum_{a'} \pi(a'|s') Q(s',a') - Q(s,a) \right], \end{gathered} \tag{5}$$

- **Double Q-Learning**: Reduces Q-Learning's overestimation bias by maintaining two independent value functions, Q_A and Q_B, updated alternately [4].

$$\begin{gathered} Q(s,a) \leftarrow Q_A(s,a) + \\ + \alpha \left[r + \gamma Q_B(s', \operatorname{argmax}_{a'} Q_A(s',a')) - Q_A(s,a) \right], \end{gathered} \tag{6}$$

- **Hybrid Q-Learning and Expected SARSA**: Merges Q-Learning's efficiency with Expected SARSA's variance reduction, using dual estimators for balanced stability and performance [3].

C. Exploration Strategies

We use ϵ-greedy exploration to balance exploration and exploitation. With probability ϵ, the agent selects a random action; otherwise, it chooses the action maximizing $Q(s,a)$.

D. Notation Summary

The following notation is used throughout:

- $s \in \mathcal{S}$: state (player's hand, dealer's card, usable ace).
- $a \in \mathcal{A}$: action (hit or stand).
- $\pi(a|s)$: policy defining the probability of action a in state s.
- $Q(s,a)$: action-value function.
- γ: discount factor.
- α: learning rate.
- ϵ: exploration rate in ϵ-greedy policies.

E. Open AI's Blackjack Environment

In our experiments, the Blackjack environment uses the assumption of an infinite pack of cards, where cards are drawn with replacement and the deck is never exhausted. This eliminates the possibility of strategies such as card counting, which depend on keeping track of which cards have been played [2]. Although this limits the realism of our framework, it simplifies the problem and is commonly used in competition law research to create a stationary environment.

F. RLCard's Blackjack Environment

These fundamental RL concepts and algorithms form the theoretical basis of our comparative study of agents in the Blackjack environment, presented in the following sections.

III. Methodology/Approach

A. Environment Description

The environment used in this work is the `Blackjack-v1` environment from the `Gymnasium` library. Blackjack is a widely adopted testbed for reinforcement learning (RL) due to its stochastic nature, manageable state and action spaces, and clearly defined episodic structure [1]. Its simplicity allows us to focus on the learning dynamics of RL agents, while involving a degree of uncertainty and strategic decision-making.

The environment follows standard Blackjack rules with the following configuration:

- **State space** $\mathcal{S}$: Each state s is defined by the tuple (p, d, u), where p is the player's current hand sum (ranging from 4 to 21), d is the dealer's visible card (ranging from 1 to 10), and u is a binary indicator of whether the player has a usable ace.
- **Action space** $\mathcal{A}$: The agent selects from two possible actions: $a = 0$ (*stand*) or $a = 1$ (*hit*).
- **Reward function**: At the end of each episode, the agent receives a terminal reward r as follows:
  - $+1$ if the agent wins,
  - -1 if the agent loses,
  - 0 if the game results in a draw.
- **Transition dynamics**: The environment assumes an infinite deck, meaning that cards are drawn with replacement, guaranteeing a stationary distribution of outcomes between episodes. This assumption eliminates the possibility of implementing strategies based on memory or card counting and aligns with the traditional Markov Decision Process (MDP) framework of LR [2].

We disable the "natural blackjack" bonus and Slightly Advantageous Blackjack (SAB) rules by setting `natural=False` and `sab=False` in the environment configuration.

B. Implemented Agents

We implement and compare six reinforcement learning agents and one basic agent, each representing different learning strategies. All agents have the same state and action definitions and are trained under identical conditions to allow a fair comparison. The agents include:

- **Random Agent**: Selects actions uniformly at random, without considering the state.
- **Monte Carlo (MC) Agent**: Estimates state-action values $Q(s, a)$ by averaging returns from complete episodes, using first-visit Monte Carlo methods.
- **Q-Learning Agent**: An off-policy Temporal Difference (TD) method that updates $Q(s, a)$ using the maximum estimated value of the next state [8].
- **SARSA Agent**: An on-policy TD method that updates $Q(s, a)$ based on the action actually taken in the next state [6].
- **Expected SARSA Agent**: Extends SARSA by computing the expected value over all possible next actions under the current policy [7].
- **Double Q-Learning Agent**: Addresses overestimation bias by maintaining two independent value functions, Q_A and Q_B, which are updated alternately [4].
- **Hybrid Agent**: Combines aspects of Q-Learning and Expected SARSA through a fixed hybrid update mechanism inspired by hybrid TD learning techniques [3].

All agents are implemented within a common framework to ensure consistent handling of episodes, rewards and ratings.

C. Experimental Setup

We analyse the agent's performance over several training periods, with the number of episodes varying from $100 to 100{,}000$. For each number of episodes, we run three independent trials to assess stability and robustness.

After training, agent performance is evaluated over a subset of episodes by monitoring:

- **Win rate:** The proportion of episodes won by the agent.
- **Draw and loss rates:** To assess the full distribution of outcomes.
- **Dealer's advantage:** the dealer's advantage, also known as house edge, is computed as

$$Dealer\ Advantage = \frac{Expected\ profit\ for\ dealer}{Initial\ bet}$$

. In our case, since the initial bet is 1, and

$$\begin{aligned} Expected\ profit\ for\ dealer &= (L * Win\ Payout) \\ &+ (D * Draw\ Payout) + (W * Loss\ Payout) \\ &= L * 1 + D * 0 + W * (1) \\ &= L - W \end{aligned}$$

, dealer's advantage is $L - W$.

These measurements are averaged over all the trials in order to compare performance between agents and numbers of episodes.

D. Hyperparameter Configuration

Hyperparameters are chosen based on standard practice and empirical testing, as summarized in Table I.

TABLE I
HYPERPARAMETERS USED IN EXPERIMENTS

Parameter	Value
Learning rate (α)	0.1
Discount factor (γ)	1.0
Initial exploration rate (ϵ)	1.0
Minimum ϵ	0.1
Decay rate	0.9999 per episode

These parameters balance exploration and exploitation while ensuring comparability across agents.

E. Implementation Details

The project is implemented in Python using:

- `gymnasium` for the Blackjack environment.
- `NumPy` for numerical computations.
- `Matplotlib` and `Seaborn` for result visualization.

IV. Results and Discussion

This section presents and analyzes the performance of different RL agents in the Blackjack environment with different numbers of training episodes. The agents compared include Random, Monte Carlo, Q-learning, Double Q-learning, SARSA, Expected SARSA and a hybrid approach. The results are reported over multiple trials for robustness. If the nature of the game of Blackjack makes it almost impossible to have a win rate above 50%, it is still possible to optimize the metrics: Sutton and Barto theorized this in their work called *Reinforcement Learning: An Introduction*, and put the optimized win rate around 42% and the optimized loss rate at 48% [9], meaning a dealer's advantage at around 6%.

A. Learning Progression and Training Efficiency

An obvious trend in all experiments is that the performance of the agents generally improves as the number of training episodes increases. At only 100 episodes, agents such as Q-Learning and Expected SARSA show unstable and highly fluctuating results in all experiments. However, at 10,000 episodes, most agents stabilize around a win rate of 41-46%, with Expected SARSA and Hybrid agents consistently at the top end.

However, after around 10,000 episodes, the benefit of further training decreases. The difference between 10,000 and 100,000 episodes is marginal, with only small fluctuations within 1–2% of the win rates. Therefore, from a computational resource perspective, 10,000 episodes appear to be sufficient to achieve near-optimal performance in this environment. Training beyond this provides only minimal gain and is therefore superfluous for practical purposes.

B. Stability and Consistency of Agents

When evaluating RL agents, stability over multiple trials is crucial. While Q-learning shows strong peak performance in certain trials, it also suffers from occasional regressions, especially at lower episode counts (100-500 episodes). In contrast, Expected SARSA and the hybrid agent show more stable and predictable performance across different trials and episode settings.

Interestingly, the results of Monte Carlo methods are inconsistent. While they can match or even outperform other agents in some trials (e.g. trial 3 with 1000 episodes), their average performance is generally lower, especially in short training scenarios. This could be due to the Monte Carlo method's reliance on full episode returns, which can be less effective in high variance and episodic termination environments such as blackjack.

Comparative analysis of agent behavior

- **Random Agent**: Moves consistently around the 30% win rate, confirming its role as a performance base.
- **Monte Carlo**: Very variable and sensitive to the number of episodes. Has problems training with few episodes, but stabilizes after 1000 episodes.
- **Q-Learning and Double Q-Learning**: Competitive performance beyond 500 episodes, but higher variance in regimes with few episodes.
- **SARSA and Expected SARSA**: SARSA outperforms Q-learning in some situations (300-800 episodes), likely due to its on-policy nature mitigating overestimation errors. Expected SARSA outperforms regular SARSA and Q-learning on long training runs.
- **Hybrid Agent**: Robust performance, but rarely better than Expected SARSA on long-term average.

Table IV-B summarizes the observed stability and peak performance of the individual agents at a low and a high number of episodes. This comparison underlines the consistency and efficiency of the algorithms, with Expected SARSA proving to be the most stable and best performing agent in our experiments.

Agent	**Stability (Low Episodes)**	**Stability (High Num of Episodes)**	**Peak Perf.**
Q-Learning	Moderate	High	$\sim$ 42.4%
Double Q	Moderate	High	$\sim$ 42.9%
SARSA	Moderate	High	$\sim$ 43.0%
Expected SARSA	Moderate	High	$\sim$ **46.0%**
Hybrid	Variable	High	$\sim$ 42.7%
Monte Carlo	High variance	Good	$\sim$ 43.0%
Random	Poor	Poor	$\sim$ 28.0%

TABLE II
AGENT STABILITY AND PEAK PERFORMANCE.

C. Key Observations and Quantitative Insights

Expected SARSA is **the best overall agent**, leading as the best performing agents at high episode counts (10,000+), with win rates around 44-46%, and a very low dealer's advantage as shown in Table IV-C below. It converges very quickly and provides more reliable results than Monte Carlo or Double Q Learning very early. It is also the agent with the lowest loss rate, and is the most effective at reducing the loss rate as the number of episodes increases.

Agent	**Stability (High Num of Episodes)**	**Win Rate**	**Loss Rate**	**Dealer's Advantage**
Q-Learning	High	42.0%	49.0%	7.0%
Double Q	High	43.8%	47.5%	3.7%
SARSA	High	41.6%	49.8%	8.2%
Expected SARSA	High	**45.6%**	**46.1%**	**0.5%**
Hybrid	High	41.3%	50.7%	9.4%
Monte Carlo	Good	43.0%	49.0%	6.0%
Random	Poor	29.0%	66.4%	37.4%

TABLE III
AGENT STABILITY AND PERFORMANCE IN TERMS OF WIN RATE, LOSS RATE AND DEALER'S ADVANTAGE ON ONE TRAINING OF 10,000 EPISODES.

D. Future Considerations and Improvements

- **number of cards or card memory**: Including the memory of previously played cards could improve decision making in realistic situations.
- **reward customization**: Adjusting rewards for blackjacks, losses or risky moves could refine agents' strategies.
- **episode length sensitivity**: Further analysis of the effects of episode length on on-policy methods (SARSA) and off-policy methods (Q-learning) could provide deeper insights.

V. Bonus - RL Card environment

When considering our project, one aspect that concerned us was that it was impossible to train our agents to count cards since, as explained before, the deck is infinite. Therefore, we wanted to quickly explore the differences in training an agent on Open AI's infinite deck environment against RL Card's finite deck one. Let's thus first introduce the RL Card environment:

- **State space** $\mathcal{S}$: Each state s is defined by a state directory with the keys `(obs, legal_actions, raw_obs, raw_legal_actions, action_record)`, where `obs` is an array with the player's current sum and dealer's current sum, `legal_actions` and `raw_legal_actions` describe all actions the player can take, `raw_obs` is a directory describing all observations such as all cards in the player's hand and in the dealer's hand, and `action_record` is an array keeping track of all actions taken.
- **Action space** $\mathcal{A}$: Same as Open AI's one.
- **Reward function**: Same as Open AI's.
- **Transition dynamics**: The environment assumes a finite deck, meaning that cards are drawn without replacement. This assumption enables the possibility of implementing strategies based on memory or card counting.

We thus randomly chose to modify the Q-Learning agent code to train on this environment, enabling card counting and adapting the agent to the environment's variable (state space and functions).

The first element that was striking when training the agent was the number of episodes needed to make the agent learn from its environment. Indeed, the agents' win rate converged very quickly on Open AI's environment, with just 10,000 episodes for example, largely due to the simplicity of the environment: the agent only had to take into account the current sum and if it had a usable ace. Here, the state space is much more complicated, and the strategy of the agent to win is also very different: the agent needs to account for the sum, usable ace, and all cards that were drawn before. Therefore, the agent takes way more time to improve his win rate, but has however the opportunity to beat the initial odds of Blackjack. The figure (cf Figure 1) below shows the improvement of the win rate of the agent over different number of episodes.

This shows that a slight change in the environment can actually impact the reinforcement learning strategy a lot, since it changes the gameplay completely. If the rules look the same, the way to play Blackjack is profoundly impacted.

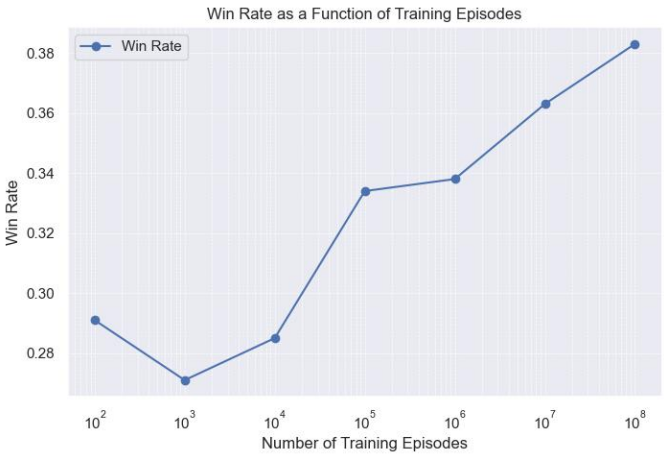


Fig. 1. Log graph of the win rate of the Q-Learning agent as a function of the number of episodes it trained on.

Some reflexions we had was to adapt all of these models and train them on a lot of episodes (100,000,000) to compare their performance in learning on a finite deck.

VI. Conclusions

This research conducted a comparative study on different RL algorithms in Open AI's Blackjack environment and analyzed their performance over different ranges of training durations. Our results show that several agents such as Q-learning, SARSA, and Double Q-learning, achieve comparable win rates after sufficient training, with Expected SARSA consistently outperforming the others in terms of both stability and peak performance. Through this testing and evaluation, we observed the tradeoffs between on-policy and off-policy methods. There was a clear result showing the importance of controlled exploration, and the diminished returns from extended training after a certain threshold. As a bonus, exploring the RL Card Blackjack environment with a finite decks introduced new strategies such as card counting that could be developed further in the future. Indeed, there is an opportunity to beat the bad odds of the game and reduce the house edge to a minimum on a deck with replacement.

Tool Reference

[1] OpenAI, "ChatGPT," https://chat.openai.com/. Used to ensure consistent IEEE formatting throughout the report and assist with the organization of the bibliography.

References

[1] A. Pérez-Uribe and E. Sanchez, "Blackjack as a test bed for learning strategies in neural networks," in *Proc. IEEE Int. Conf. Neural Networks*, 1998.

[2] J. Geiser and T. Hasseler, "Beating Blackjack – a reinforcement learning approach," Stanford University, 2022.

[3] M. Ganger, E. Duryea, and W. Hu, "Double Sarsa and Double Expected Sarsa with shallow and deep learning," *J. Data Anal. Inf. Process.*, vol. 4, pp. 159–176, 2016.

[4] H. van Hasselt, "Double Q-learning," in *Advances in Neural Information Processing Systems (NIPS)*, 2010.

[5] J. Read, "Lectures on Advanced Machine Learning and Autonomous Agents," CSC_52081 Advanced Machine Learning and Autonomous Agents, 2024.

[6] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 2020.
[7] H. van Seijen, H. van Hasselt, S. Whiteson, and M. Wiering, "A theoretical and empirical analysis of Expected Sarsa," in *Proc. IEEE Symp. Adaptive Dynamic Programming and Reinforcement Learning*, 2009.
[8] B. Jang, M. Kim, G. Harerimana, and J. W. Kim, "Q-learning algorithms: A comprehensive classification and applications," *IEEE Access*, vol. 7, pp. 133653–133670, 2019.
[9] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed., Cambridge, MA: MIT Press, 2018.

APPENDIX

This appendix includes a full PDF document containing all the detailed experimental results, including additional graphs, tables and extended evaluations for each algorithm studied. These supplementary documents provide an overview of the performance of our reinforcement learning agents in the Blackjack environment, complementing the main results discussed in the paper. The appendix is not included in the page limit and serves as an optional reference for those interested in further analysis and replication of our experiments.

Appendix: Experimental Setup
The following configuration was used for all agents and experiments to ensure consistency and reproducibility.

Environment Settings:

```
NUM_DECKS = 1           # Single deck simulated.
NATURAL = False         # No special reward for natural blackjack.
SAB = False             # Slightly Advantageous Blackjack rules disabled.
UNLIMITED_DECK = True   # Infinite deck assumption (cards drawn with replacement).
RENDER_MODE = None      # No visual rendering during training.
```

Agent Hyperparameters:

- Learning rate (`ALPHA`): 0.1
- Discount factor (`GAMMA`): 1.0
- Initial exploration rate (`EPSILON`): 1.0
- Minimum exploration rate (`MIN_EPSILON`): 0.1
- Epsilon decay rate (`EPSILON_DECAY`): 0.9999 per episode
- Number of training episodes (`NUM_EPISODES`): Variable across experiments (100 to 100,000)
- Number of evaluation episodes: 10% of the training episodes

Hardware and Software:

- MacBook Pro 14", M1 Pro Chip, 16GB RAM
- Python 3.13.1

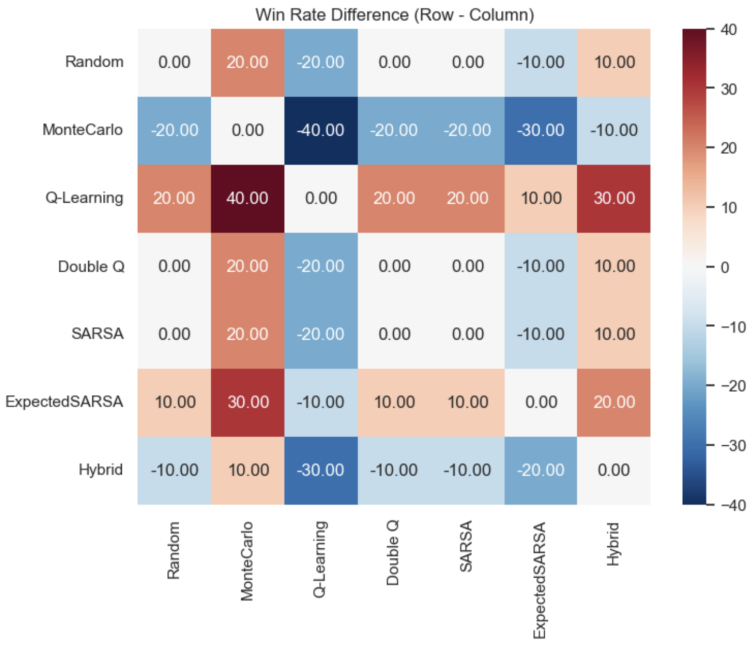
Experimental Notes:

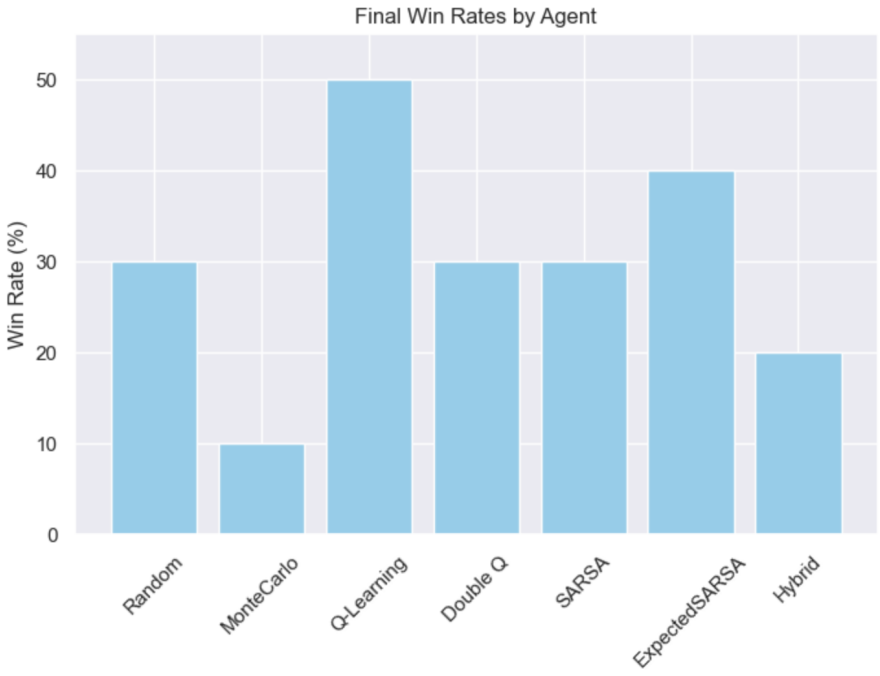
- Three consecutive runs were performed for each experiment to ensure consistency.
- The same global configuration was used for all agents for a fair comparison.
- Results are presented in text format, along with heatmaps and bar charts for visualization.

1st Trial

```
NUM_EPISODES = 100
```

```
=== Performance Comparison ===
           Agent  WinRate(%)  DrawRate(%)  LossRate(%)
0         Random        30.0         10.0         60.0
1     MonteCarlo        10.0         10.0         80.0
2     Q-Learning        50.0          0.0         50.0
3       Double Q        30.0          0.0         70.0
4          SARSA        30.0          0.0         70.0
5  ExpectedSARSA        40.0          0.0         60.0
6         Hybrid        20.0         10.0         70.0
```

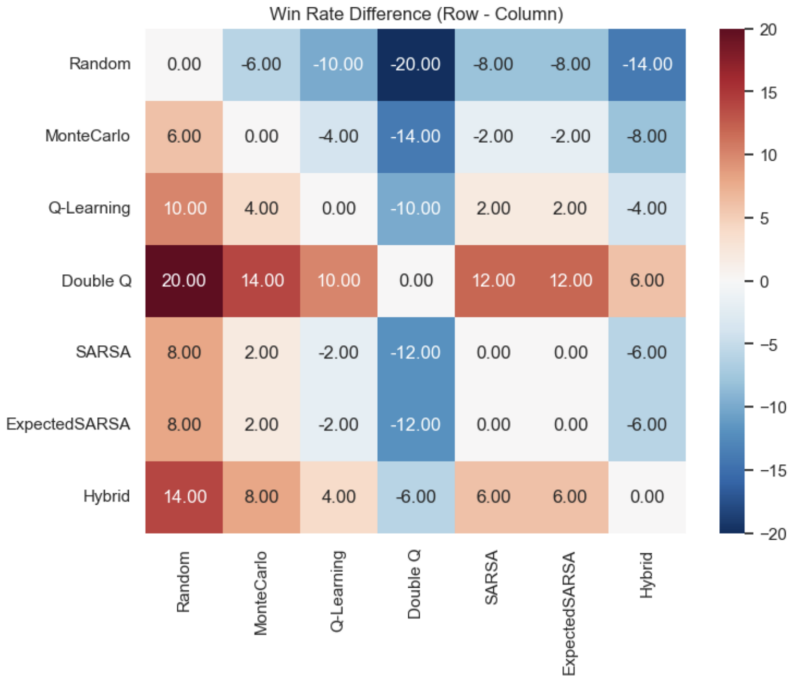


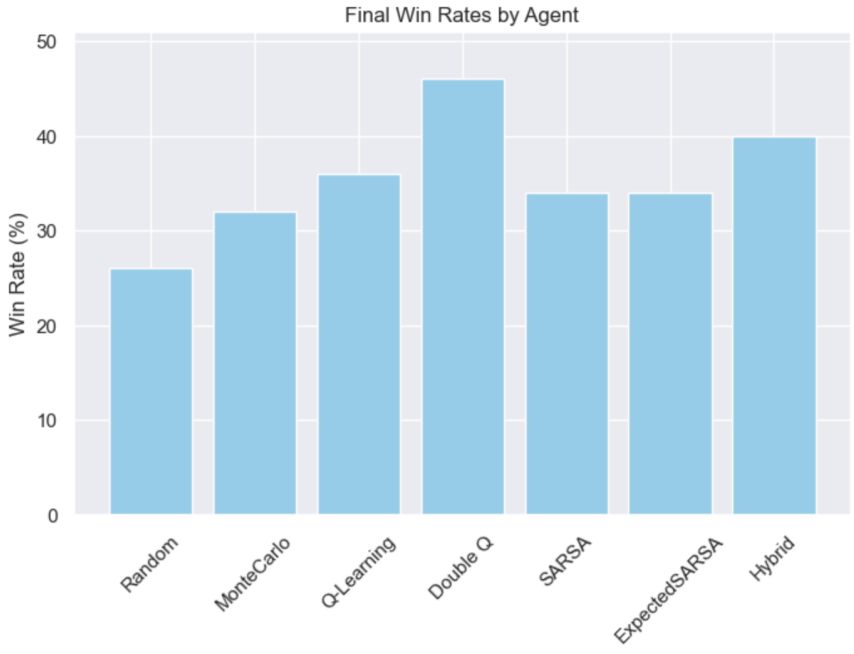


2nd Trial

```
NUM_EPISODES = 500
```

```
=== Performance Comparison ===
           Agent  WinRate(%)  DrawRate(%)  LossRate(%)
0         Random        26.0          0.0         74.0
1     MonteCarlo        32.0         10.0         58.0
2     Q-Learning        36.0          8.0         56.0
3       Double Q        46.0          4.0         50.0
4          SARSA        34.0          8.0         58.0
5  ExpectedSARSA        34.0         12.0         54.0
6         Hybrid        40.0          4.0         56.0
```

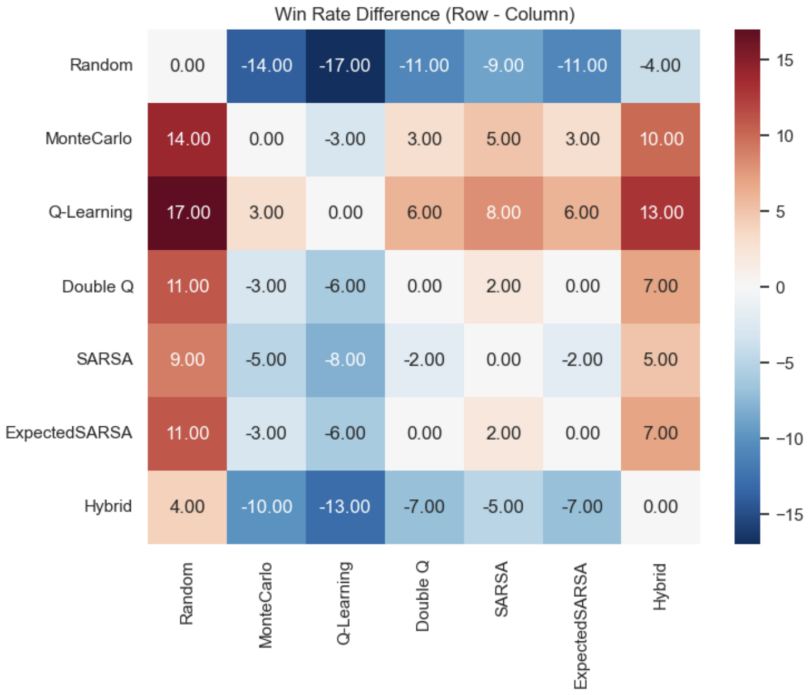


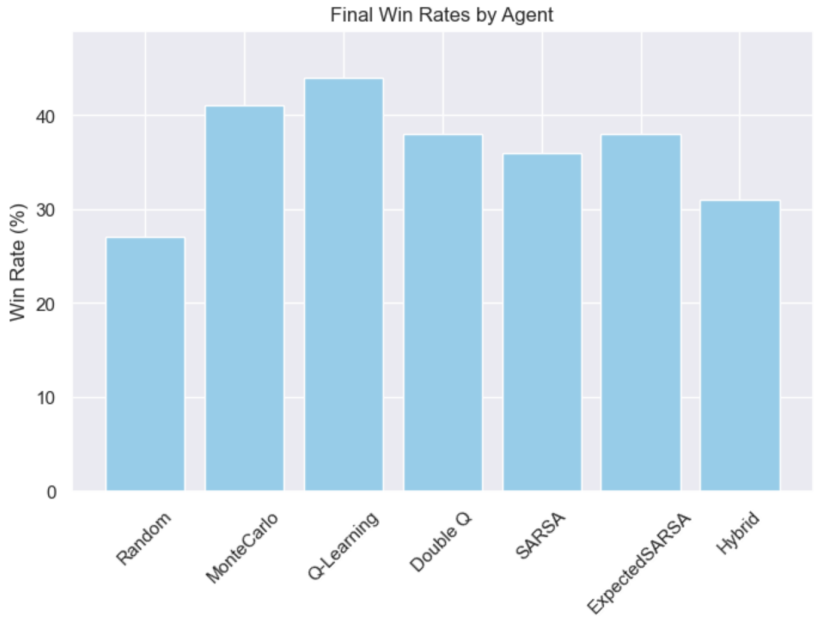


2nd Trial

```
NUM_EPISODES = 1000
```

```
=== Performance Comparison ===
           Agent  WinRate(%)  DrawRate(%)  LossRate(%)
0         Random        27.0          9.0         64.0
1     MonteCarlo        41.0          9.0         50.0
2     Q-Learning        44.0         11.0         45.0
3       Double Q        38.0          3.0         59.0
4          SARSA        36.0         12.0         52.0
5  ExpectedSARSA        38.0          7.0         55.0
6         Hybrid        31.0         13.0         56.0
```

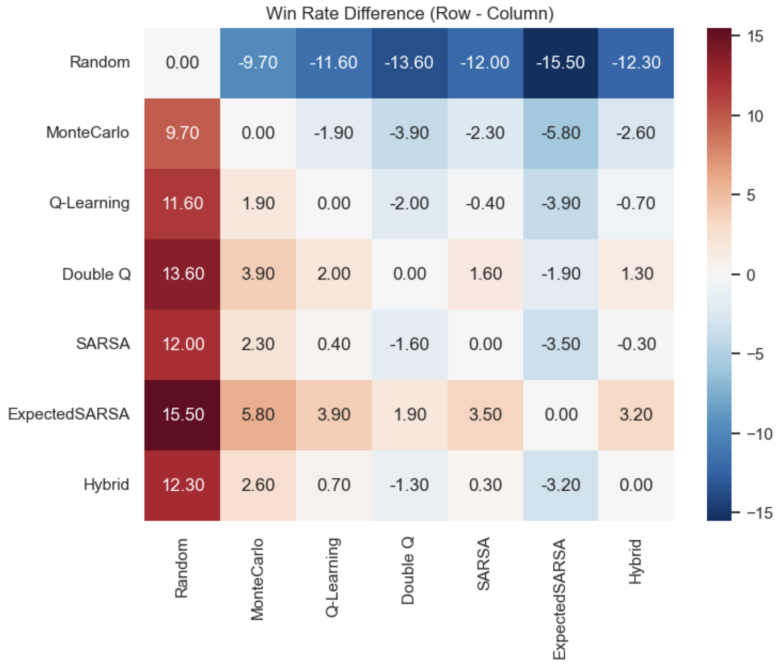


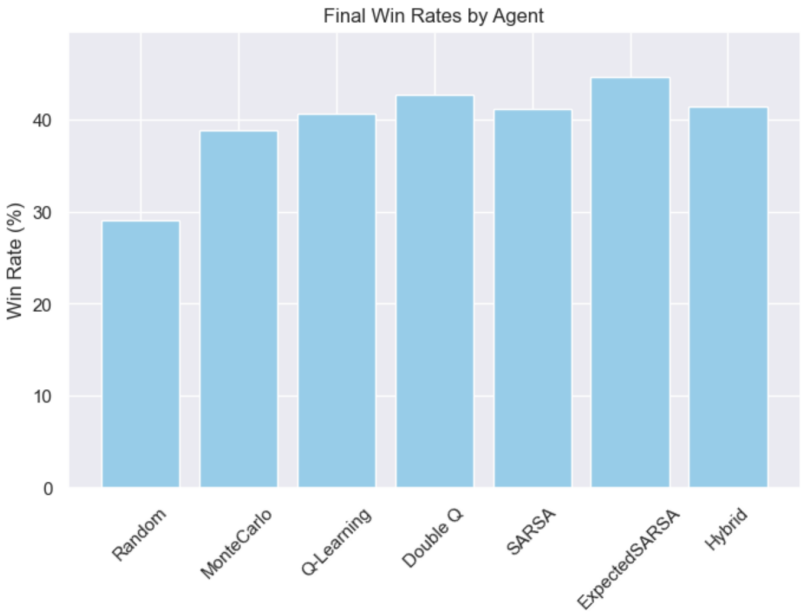


3rd Trial

```
NUM_EPISODES = 10000
```

```
=== Performance Comparison ===
           Agent  WinRate(%)  DrawRate(%)  LossRate(%)
0         Random        29.1          3.9         67.0
1     MonteCarlo        38.8          8.8         52.4
2     Q-Learning        40.7          9.6         49.7
3       Double Q        42.7          9.4         47.9
4           SARSA        41.1          8.1         50.8
5  ExpectedSARSA        44.6          9.2         46.2
6         Hybrid        41.4          8.4         50.2
```





4th Trial

```
NUM_EPISODES = 100000
```

```
=== Performance Comparison ===
           Agent  WinRate(%)  DrawRate(%)  LossRate(%)
0         Random       28.16         4.36        67.48
1     MonteCarlo       41.66         8.50        49.84
2     Q-Learning       42.31         9.05        48.64
3       Double Q       42.39         9.48        48.13
4          SARSA       42.28         8.87        48.85
5  ExpectedSARSA       42.16         9.34        48.50
6         Hybrid       42.55         8.76        48.69
```