Optimizing Blackjack Strategies Using Double Q-Learning and Card Counting-Aware Policies

Filippo Focaccia Giulio Pirotta Tommaso Vezzoli

Abstract

This work explores the use of reinforcement learning (RL) via Double Q-Learning to develop advanced strategies for the game of Blackjack. We design a custom OpenAI Gym-compatible environment that incorporates core Blackjack mechanics, including splitting, doubling, and card counting through the true count. Double Q-Learning is implemented across multiple configurations to study its ability to learn robust decision policies. We demonstrate that agents trained using Double Q-Learning can adapt to complex state-action spaces and exhibit behavior aligned with well-known Blackjack strategies, including dynamic bet sizing in response to card-counting signals. Our results highlight the effectiveness of Double Q-Learning in environments with sparse rewards and multiple decision branches.

1 Introduction

Blackjack is a classic card game that involves sequential decision-making under uncertainty. Its rich combination of probabilistic outcomes, reward shaping, and rule-based structure makes it an ideal testbed for reinforcement learning. Traditional strategies such as the Basic Strategy and card counting are well-studied, yet their integration into learning-based agents remains an area of active research.

This project investigates the following research question: *Can Double Q-Learning effectively learn Blackjack strategies—including betting, splitting, and counting-based adaptations—in a custom simulation environment?*

Our contributions include:

- A Gym-compatible Blackjack environment supporting hit, stand, double, split, and betting actions
- Incorporation of the Hi-Lo card counting system via true count in the observation space
- Evaluation of multiple Double Q-Learning agents under varying rule configurations
- Visualization and comparison with basic strategy charts

2 Related Work

Card games like Blackjack have long been studied in the context of probabilistic modeling and decision theory. The Basic Strategy defines optimal actions under fixed deck distributions, while the Hi-Lo system enables advantage play through card counting.

RL applications to Blackjack have primarily focused on Q-Learning in simplified environments. However, Double Q-Learning has been shown to mitigate overestimation bias in Q-value updates, which is critical in Blackjack due to its sparse and delayed rewards. This work extends previous approaches by combining card counting signals with Double Q-Learning in a full-featured simulation.

3 Methodology

3.1 Environment Design

We implement a Blackjack simulation using Python classes to represent cards, decks, hands, players, and dealers. The environment supports:

- Variable numbers of decks and reshuffling thresholds
- Action space including hit, stand, double, split, and bet selection
- State transitions adhering to Blackjack rules (e.g., one-card splits on aces)

3.2 Observation Space

The RL agent receives the following observations:

- Player score (1–31)
- Dealer upcard value (1–10)
- Indicator for soft hands (1 if true, 0 otherwise)
- Indicator for ability to split
- Pair value if a pair is present
- True count (Hi-Lo card counting normalized by decks remaining)

3.3 Action Space and Rewards

The agent chooses two types of actions:

- **Bet Action**: Select bet size before hand is dealt
- **Move Action**: Choose from hit, stand, double, or split

Rewards:

- Win = +1, Loss = -1, Natural Blackjack win = +1.5
- Rewards from playing actions like double down or split are multiplied according to the development of the match (e.g., playing double down and win will give reward +2)
- Neutral reward (0) on ties (including hands where the agent plays split and wins one hand and lose the other)

3.4 Q-Learning

Q-Learning is a foundational reinforcement learning algorithm that estimates the expected cumulative reward (Q-value) for taking a specific action a in a given state s. The standard update rule is given by Bellman equation:

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

where s' is the next state after the action, and a' is the best action from s'. α is the learning rate, γ the discount factor, and r the immediate reward. This method gradually improves estimates through iterative updates and ϵ-greedy exploration, converging to optimal values under the right conditions.

3.5 Double Q-Learning

We train agents using the Double Q-Learning algorithm, which maintains two Q-tables (or approximators) and uses them alternately for action selection and target value updates:

$$Q_1(s,a) \leftarrow Q_1(s,a) + \alpha \left[r + \gamma Q_2(s', \arg\max_{a'} Q_1(s',a')) - Q_1(s,a) \right]$$

where s is the current state in which the agent plays action a, s' is the state on which it ends up after the action, and a' is the best action the agent would play in state s'.
This decouples max action selection from the value estimation, reducing positive bias and improving policy stability. Training is performed using ϵ-greedy exploration with scheduled decay.

4 Training

4.1 Setup

To build a robust Blackjack-playing agent, we implemented Double Q-Learning in a phased manner, gradually increasing the action complexity and observation granularity. The training was split across four main stages, corresponding to separate notebooks:

- **Stage 1:** `dq_hit_stand.ipynb` – Hit/Stand only
- **Stage 2:** `dq_hit_stand_dd.ipynb` – Adds Double Down

- **Stage 3:** `betting.ipynb` – Adds dynamic betting using true count
- **Stage 4:** `split.ipynb` – Adds Split action and multi-hand logic

Each training step built upon the previous Q-table, progressively extending the action space and refining the learned strategy.
In all stages, updates followed the Double Q-learning formulation with two Q-tables (Q_1, Q_2), and learning alternated between them every step.

Note on State Pruning: In all training stages except the final split phase, states where the player's hand sum was below a certain threshold were excluded from training. In such states, hitting is universally advantageous, so the agent defaults to a hardcoded `hit` without learning, reducing the state space and focusing training on meaningful decision points.

4.2 Evaluation Metrics

To evaluate each stage, we used both general and stage-specific metrics:

- **Win Rate:** Proportion of hands won by the agent
- **Loss Rate:** Proportion of hands lost
- **Draw Rate:** Proportion of ties or pushes
- **Average Reward:** Mean return per episode across evaluation rounds

Additional stage-specific metrics included:

- **Stage 2:** Frequency and performance of double-down decisions, win rate when doubling
- **Stage 3:** Average bet size across different true count buckets, profit as a function of true count
- **Stage 4:** Split frequency, win rate post-split, and hand count per round

All metrics were computed on evaluation runs using the final learned policy with no exploration ($\epsilon = 0$).

Stage 1: Hit/Stand Training

In the initial stage, the agent could only choose between `hit` and `stand`. The Q-table was not initialized to zeros but strategically biased: states with high player sums favored `stand`, while lower sums favored `hit`. This accelerated convergence toward realistic strategies.

- **Discount factor (γ):** 1.0, to emphasize long-term returns over immediate rewards
- **Epsilon decay:** from 1.0 to 0.01 with decay rate 0.99995

- **Learning rate (α):** Adaptive per state-action pair, with decay parameter $\lambda = 0.00005$ and an initial learning rate $\alpha_{\text{initial}} = 0.1$
- **Episodes:** 1,000,000

The learning rate α used two levels of adaptation:

1. At the end of each episode, the base learning rate was updated as:

$$\alpha^{t+1} = \frac{\alpha_{\text{initial}}}{1 + \lambda \cdot t}$$

   where t is the episode

2. During Q-value updates, this base learning rate was made adaptive to the state-action pair visit count $N(s, a)$ up to time t:

$$\alpha_{s,a}^{t} = \max\left(\frac{\alpha^{t}}{1 + 0.005 \cdot N(s, a)},\ \frac{\alpha^{t}}{10}\right)$$

This formulation balances exploration with convergence. Early in training, it encourages rapid updates, while later steps introduce stability. The lower bound prevents the learning rate from vanishing too quickly, ensuring continued adaptation even for frequently visited state-action pairs.

Stage 2: Adding Double Down

In this phase, the `double` action was introduced, necessitating expansion of the Q-table. The initialization was carefully constructed:

- Q-values for `hit` and `stand` were carried over from Stage 1
- Q-values for `double` were initialized with small biases based on Blackjack game theory, being slightly favorable in advantageous states (e.g., player 11 vs. dealer 6)

Training proceeded in two phases, both with the same learning rate scheduler as in Stage 1:

1. **Exploration phase (2,000,000 rounds):** Epsilon decayed as in Stage 1, but during random exploration steps, `double` was sampled more frequently to ensure visibility; Q-values for Hit and Stand are not updated in this phase
2. **Fine-tuning phase (2,000,000 rounds):** the learning rate was scheduled as usual, but during the Q-table update step, only 25% of the computed learning rate was used. This scaling reduces the update magnitude and ensures more conservative refinement of the policy, especially important when integrating the new action (double) into a Q-table that already encodes meaningful values for hit and stand.

This design allowed the agent to integrate `double` into the policy, leading to faster and more stable convergence. Empirically, most policies converged before the full 2M rounds in the fine-tuning stage.

Stage 3: Betting Based on Count

This phase introduced dynamic betting decisions informed by card counting. The agent no longer relied on previous Q-tables, as the structure and semantics of the task were entirely different. The state consisted solely of the observed true count at the beginning of each round, and the actions corresponded to bet sizes.

- **Independent Q-table:** Separate from earlier stages, indexed by true count buckets
- **True count buckets:** 0 for tc ≤ 1, 1 for tc $\in [2, 4]$, and 2 for tc ≥ 5
- **Q-table initialization:** Biased heuristically—lower bets preferred for lower counts, and higher bets for high counts
- **Episodes:** 4,000,000

Learning rate scheduling: This stage used a hybrid learning rate scheduler that balances exploration and convergence through two mechanisms:

1. For frequently visited state-action pairs (i.e., visit count > 100), the learning rate decays more slowly:

$$\alpha = \max(\alpha_{\min},\ \alpha_{\text{initial}} \cdot 0.9995^{\lfloor \text{visits}/100 \rfloor})$$

2. For infrequent pairs, the decay follows an exponential schedule:

$$\alpha = \max(\alpha_{\min},\ \alpha_{\text{initial}} \cdot (\lambda)^t)$$

Where $\alpha_{\text{initial}} = 0.1$, $\alpha_{\min} = 0.0005$, and $\lambda = 0.99999$ (decay parameter). This schedule retains adaptability for high-traffic states while stabilizing updates for less common true counts.

Exploration: Epsilon decayed similarly to earlier stages but was biased to prefer larger bets under favorable counts during exploration.

To improve learning stability, rewards were clipped to the interval [`-1.2`, `+1.2`]. This was necessary since certain rounds (e.g., winning with `double`) could produce rewards beyond this range. Without clipping, such large values might lead to destabilizing Q-updates and biased learning.

Stage 4: Supporting Splits

Finally, the most complex addition—splits—required redesigning the environment to allow multi-hand play. We extended the observation space with 'can_split' and 'pair_value', and modified the action handler to process multiple hands sequentially.

- **Environment change:** now supports multi-hand tracking and recursive play
- **Separate Q-table:** States are defined as pairs of card value and dealer upcard. Actions are `split` or `not split`.

- **Q-table initialization:** Simple and unbiased. Prior attempts with heuristics failed to produce better convergence.
- **Learning rate:** Same schedule and adaptive formula as in Stage 3
- **Exploration:** Standard decay, with state-dependent epsilon based on visit counts
- **Episodes:** 3,000,000

The Q-value update logic accounted for the special nature of splits. Specifically, the next state after a `split` rarely leads to another split (i.e., two identical cards again). Thus, we used the Q-values from the main state-action Q-table (from Stage 2) to estimate the future return following a split, rather than referencing the split Q-table.

Reward clipping as in Stage 3 was used here as well, due to the compound effect of multiple hands and potential high rewards from splitting aces or doubling after a split. This kept updates stable and prevented rare large rewards from skewing value estimation.

5 Results

5.1 Basic Strategy Comparison Heatmaps

The heatmaps compare the agent's learned strategies at various training stages to the corresponding basic strategies for both soft and hard hands, as well as split decisions.
In the figures, on the X-axes we have the dealer upcard value, and on the Y-axes the value of the player score (fig. 1-2) or of the player's card if it is a pair (fig. 3).

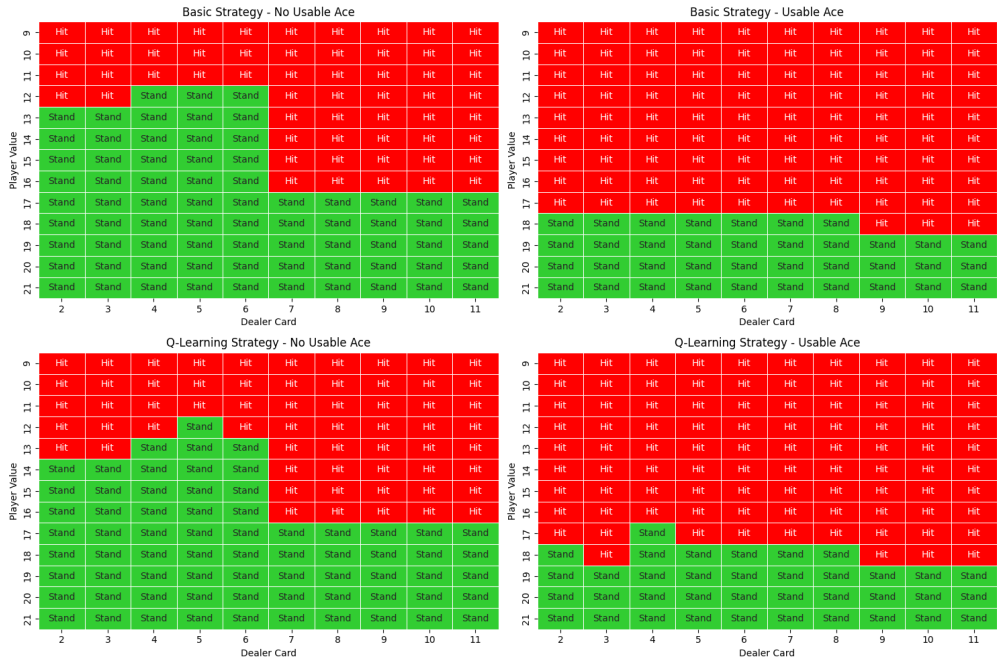

Figure 1: Heatmaps showing the player's preferred actions under the basic strategy for Hit/Stand vs. learned strategy from Stage 1. Green = Stand, Red = Hit.

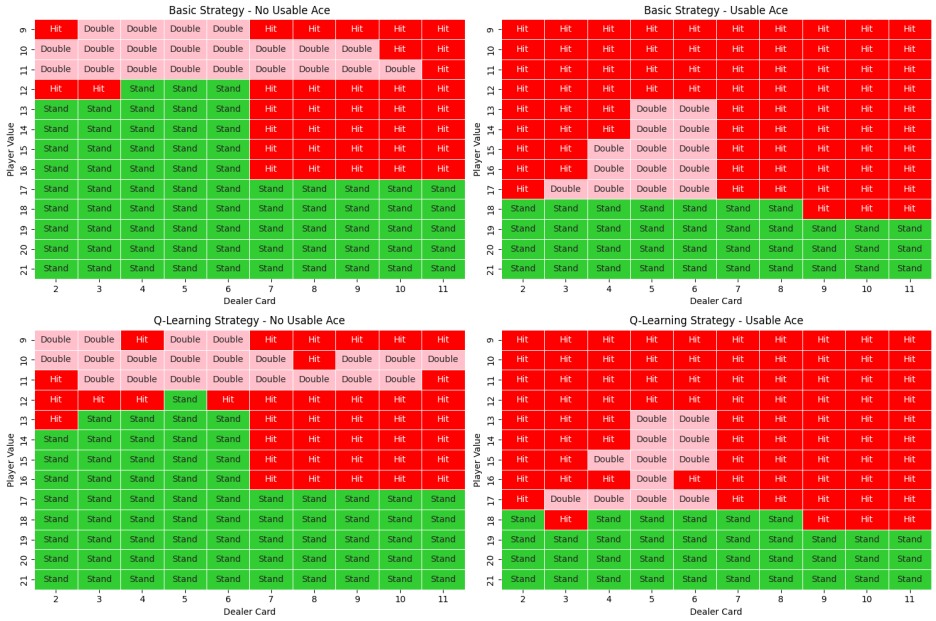

Figure 2: Heatmaps showing the player's preferred actions under the basic strategy for Hit/Stand/Double vs. learned strategy from Stage 2. Green = Stand, Red = Hit, Pink = Double Down.

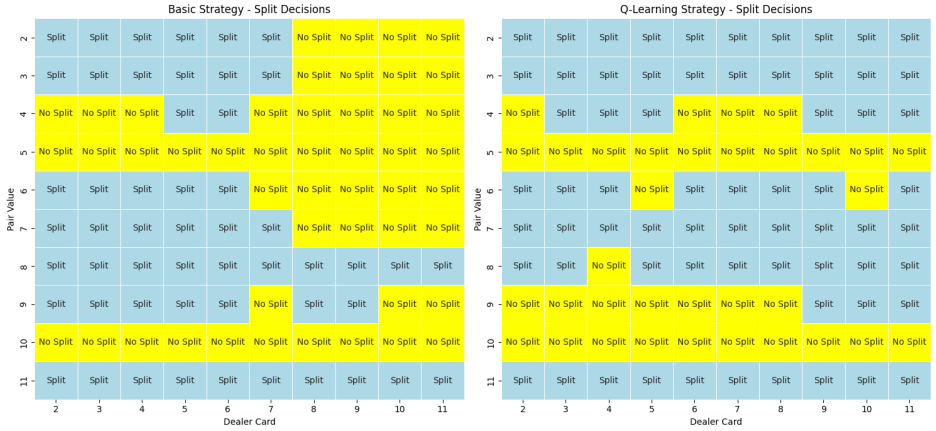

Figure 3: Heatmaps showing the player's preferred actions under the basic strategy for split vs. learned splitting strategy from Stage 3. Light Blue = Split, Yellow = No Split.

Interpretation: We can easily notice that the agent learns extremely well the strategy for hit, stand, and double moves, while the split strategy heatmaps show greater deviation from the basic strategy. This is expected due to the relative rarity and complexity of split situations. Despite these discrepancies, the agent still learned profitable patterns (e.g., always splitting 8s and Aces), as shown in its evaluation metrics.
Overall, the visualizations confirm that the agent effectively internalized key strategic principles of Blackjack while making context-sensitive adjustments based on its learned Q-values.

5.2 Learned Betting Strategy

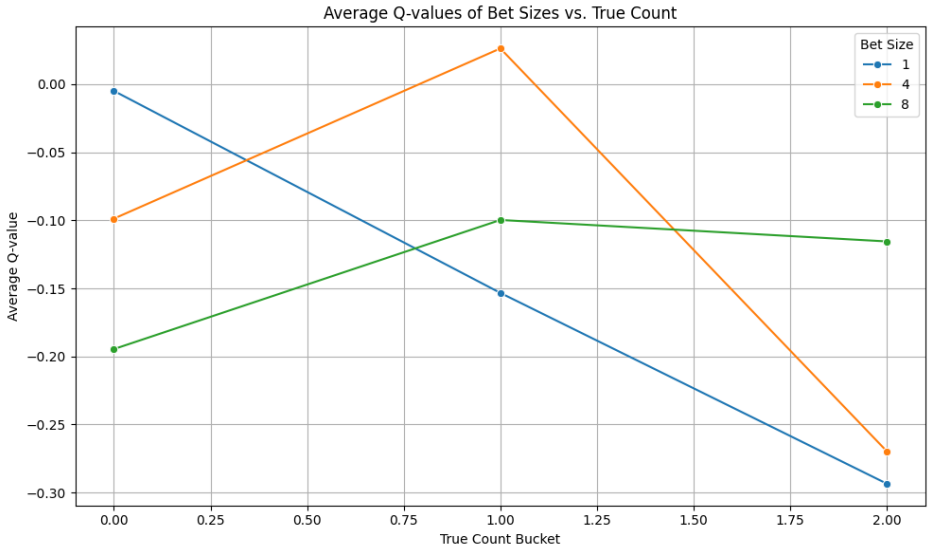

Figure 4: Average Q-values of different bet sizes as a function of true count, obtained from Stage 3.

Interpretation: The line plot above visualizes the average Q-values associated with each available bet size (1, 4, 8) across different true count buckets. These Q-values represent the expected long-term reward of placing each bet size given the current bucketed true count.

For example, suppose the agent observes a true count of 3. According to the bucketing system, this value falls into bucket 1 (representing true counts between 2 and 4 inclusive). At this bucket, the graph shows that betting 4 has the highest Q-value (approximately 0.07), followed by betting 8 (around -0.10), and betting 1 is the least favorable (around -0.15). Therefore, the agent should place a bet of size 4 to maximize expected reward.

From Q-values to Policy: The agent derives its betting strategy by simply selecting the bet size with the highest Q-value at the current true count bucket. That is:

- Observe the current true count
- Identify the corresponding bucket (0: ≤ 1, 1: 2–4, 2: ≥ 5)
- Choose the bet size whose Q-value is highest in that bucket

This approach effectively captures the principle behind card counting: increase bet size when the deck is favorable (higher true count), and reduce exposure when the odds are against the player.

Evaluation Summary Table

To quantitatively compare the performance of the learned policies at each stage, we evaluate both the basic strategy and each trained agent over 1 million simulated episodes. The table below reports key performance metrics:

Strategy	**Win Rate**	**Draw Rate**	**Loss Rate**	**Avg. Reward**
Stage 1 (Hit/Stand)	0.4324	0.0867	0.4809	-0.0258
Basic Stage 1 (Hit/Stand)	0.4333	0.0868	0.4799	-0.0240
Stage 2 (+ Double)	0.4320	0.0860	0.4820	-0.0092
Basic Stage 2 (+ Double)	0.4325	0.0860	0.4815	-0.0077
Stage 3 (+ Betting)	0.4328	0.0863	0.4808	-0.0044
Basic Stage 3 (+ Betting)	0.4328	0.0863	0.4808	-0.0025
Stage 4 (+ Split)	0.4337	0.0892	0.4770	0.0148
Basic Stage 4 (+ Split)	0.4336	0.0875	0.4789	0.0165

Table 1: Performance metrics computed on 1 million evaluation hands for each strategy.

These results are subject to natural variance due to the inherent randomness of the Blackjack environment. While the win, draw, and loss rates remain relatively stable across all strategies, both the learned and basic strategies demonstrate a clear trend of increasing average reward per hand as training stages progress. This suggests the incremental inclusion of complex actions (double, betting, split) enables agents to improve expected returns, ultimately reaching positive average outcomes in the final stage.

6 Discussion

Double Q-Learning agents successfully learned Blackjack policies consistent with domain knowledge:

- Agents learned to hit on low totals and stand on higher ones
- Inclusion of split and double increased expected return, despite the strategies of these moves being more complex and less frequently encountered
- Betting policies showed sensitivity to the true count, increasing bets with favorable decks

Notably, the strategy tables for advanced agents aligned closely with the basic strategy for hit, stand, and double actions, exhibiting only minor deviations in borderline scenarios. The learned split strategy deviated more substantially, which is expected due to the low frequency and higher variability of split opportunities. Nevertheless, the learned policy still led to a positive net gain, demonstrating effective generalization under sparse feedback.

The betting behavior notably mimicked human card counters, adjusting aggression in correlation with favorable deck conditions as indicated by the true count.

Limitations

- Large state-action space led to slow convergence, particularly in stages with expanded action sets
- The environment was limited to a single player versus the dealer, which omits strategic interactions and betting competition present in multi-player Blackjack
- The agent currently supports only three discrete bet sizes, which constrains policy flexibility and granularity in bet sizing strategy

7 Conclusion

This project demonstrates the effectiveness of Double Q-Learning for training Blackjack agents under realistic game rules. Through a structured, multi-stage training process, agents learned optimal and near-optimal strategies across diverse action sets, including split, double down, and betting based on card counting.

Despite the sparse feedback and combinatorial state space, the agents converged to policies that align closely with known basic strategy and showed strategic flexibility, especially in betting behavior. Even in more variable and rare decisions such as splitting, the learned strategies yielded strong performance in evaluation metrics.
Future directions for this work include:

- **Multi-player environments:** Extending the environment to support multiple players would introduce competitive dynamics, enabling agents to learn not just from the dealer, but from observing or responding to other players' bets and actions.
- **Flexible bet sizing:** Moving beyond three discrete bet sizes to a more continuous or larger discrete set could allow agents to express finer-grained confidence and potentially increase profits.
- **Curriculum-guided training:** A more structured training schedule—possibly using a simulator or heuristic-based sampling—could first expose the agent to common, simple situations before gradually introducing rare or complex scenarios like splits. This may help stabilize convergence and ensure core strategies are reliably acquired.
- **Function approximation:** Incorporating neural networks (the so called Deep Q-Network) to approximate Q-values would enable generalization across similar but unseen states, helping scale to larger state spaces like deck composition tracking or variable opponent modeling.

Overall, the framework and findings outlined here provide a strong foundation for intelligent game-playing agents and can be extended to other sequential decision-making domains involving uncertainty, long-term rewards, and evolving strategies.

References

[1] Watkins, C. J. C. H., & Dayan, P. (1992). Q-learning. *Machine Learning*, 8(3-4), 279–292. https://link.springer.com/article/10.1007/BF00992698

[2] Griffin, P. A. (1999). *The Theory of Blackjack: The Complete Card Counter's Guide to the Casino Game of 21* (6th ed.). Huntington Press.

[3] Thorp, E. O. (1966). *Beat the Dealer: A Winning Strategy for the Game of Twenty-One*. Vintage Books.

[4] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). OpenAI Gym. *arXiv preprint arXiv:1606.01540*. https://arxiv.org/abs/1606.01540

[5] Buramdoyal, A. (2021). Variations on the Reinforcement Learning Performance of Blackjack. *arXiv preprint arXiv:2101.00381*. https://arxiv.org/abs/2101.00381

[6] van Hasselt, H. (2010). Double Q-learning. In *Advances in Neural Information Processing Systems* (NeurIPS), 23. https://proceedings.neurips.cc/paper_files/paper/2010/file/091d584fced301b442654dd8c23b3fc9-Paper.pdf