Reinforcement Learning for Playing Blackjack

Gowtham Krishnamoorthy

A DISSERTATION

Submitted to

The University of Liverpool

in partial fulfilment of the requirements
for the degree of

MASTER OF SCIENCE

Abstract

In the realm of card games, Blackjack stands out as a blend of strategy and chance, offering a unique challenge for artificial intelligence. This dissertation explores the application of the Monte Carlo Control method in tandem with Epsilon-Greedy policies to train an agent for optimal Blackjack gameplay. Utilizing the OpenAI's `gym` environment, the agent underwent training across 500,000 episodes with the objective of enhancing its win rate.

The adopted methodology, the Monte Carlo Control method, enabled the agent to learn and adapt from its gameplay experiences. By playing a plethora of games, the agent adjusted its strategies based on the outcomes, ensuring continuous improvement. The integration of the Epsilon-Greedy policy struck a balance between exploring new strategies and exploiting known optimal moves, ensuring the agent's adaptability and growth.

Post-training, the agent demonstrated a commendable win rate of 45.10%, juxtaposed with a loss rate of 44.90% and a draw rate of 10.00%. These results underscore the agent's enhanced strategic gameplay, highlighting the effectiveness of the applied reinforcement learning techniques. This study not only showcases the potential of reinforcement learning in game strategy optimization but also serves as a foundational reference for similar applications in varied decision-making arenas. For future researchers and AI aficionados, this dissertation offers a comprehensive insight into the power and application of reinforcement learning, making it a valuable resource in the expanding world of artificial intelligence.

Student Declaration

I confirm that I have read and understood the University's Academic Integrity Policy.

I confirm that I have acted honestly, ethically and professionally in conduct leading to assessment for the programme of study.

I confirm that I have not copied material from another source nor committed plagiarism nor fabricated data when completing the attached piece of work. I confirm that I have not previously presented the work or part thereof for assessment for another University of Liverpool module. I confirm that I have not copied material from another source, nor colluded with any other student in the preparation and production of this work.

I confirm that I have not incorporated into this assignment material that has been submitted by me or any other person in support of a successful application for a degree of this or any other university or degree-awarding body.

SIGNATURE: GOWTHAM KRISHNAMOORTHY

DATE 2023-10-06

Acknowledgments

First and foremost, I would like to extend my deepest gratitude to Dr. John Fearnley, whose unwavering support, insightful feedback, and invaluable guidance have been instrumental in the realization of this project. His expertise and dedication to academic excellence have been a constant source of inspiration throughout this journey.

I am also immensely thankful to the staff of the Department of Computer Science for providing the resources and environment conducive to research and learning. Their commitment to fostering an atmosphere of innovation and inquiry has been pivotal in shaping the direction and outcomes of this project.

While this endeavour was undertaken individually, it was by no means a solitary effort. The encouragement, patience, and understanding of my family and friends have been my pillars of strength, motivating me to persevere and strive for excellence.

In conclusion, this project is a testament to the collaborative spirit of academia, even when pursued individually. The knowledge, resources, and support provided by the aforementioned individuals and entities have been invaluable, and I am eternally grateful for their contributions.

Table of Contents

List of Figures

Chapter 1

1.Introduction

In the dynamic world of artificial intelligence and machine learning, the pursuit of optimal decision-making strategies is a continuous endeavour. This project embarks on this journey, utilizing the Monte Carlo Control method in tandem with Epsilon-Greedy policies to train an agent in the classic game of Blackjack. Briefly, Blackjack might seem straightforward, but it presents a multifaceted environment ideal for delving into the intricacies of decision-making amidst uncertainty.

1.1 Scope

The primary impetus for this project was to unravel and capitalize on the capabilities of reinforcement learning techniques within a controlled setting. The game of Blackjack, with its blend of chance and strategy, was selected as the backdrop for this exploration. While the game's rules are clear-cut, the strategies to clinch victory can be complex and layered.

1.2 Problem Statement

The pivotal query steering this project was: "Is it feasible for a machine learning agent, armed with the appropriate algorithms and policies, to outstrip conventional human strategies in the game of Blackjack?" This inquiry transcends mere game-winning; it delves into the realm of gauging the extent to which machine learning can emulate, and perhaps even exceed, human cognitive processes in specific scenarios.

1.3 Approach

The end product was a meticulously trained agent, adept at decision-making, drawing from its extensive learning from myriad simulated Blackjack games. A standout feature of this project was the iterative methodology employed during the agent's training phase. Eschewing a sole reliance on preset strategies or heuristics, the agent was granted the liberty to explore, err, learn, and subsequently hone its strategy. This modus operandi is reminiscent of human learning trajectories, where experiential learning often emerges as the paramount educator.

1.4 Outcome

Post the rigorous training sessions, the agent's performance was not merely an affirmation of its training regimen but also a testament to the efficacy of the Monte Carlo Control method when synergized with Epsilon-Greedy policies. The results, set to be elaborated upon in the ensuing sections, were revelatory, casting light on the strengths and potential areas of enhancement of the adopted approach.

Episode 500000/500000.

Final Results:

Win Rate: 45.10%

Loss Rate: 44.90%

Draw Rate: 10.00%

Chapter 2

2. Aims and Objectives

The project was embarked upon with a clear set of aims and objectives, which were outlined in the initial Specification and Proposed Design document and further elaborated upon during the Final Presentation. Here, we revisit those aims and objectives to assess the extent of their realization:

2.1 Understanding the Environment:

Objective Achieved: The Blackjack environment provided by OpenAI's gym was thoroughly explored. The dynamics, rules, and nuances of the game were understood, forming the foundation for the subsequent stages of the project.

2.2 Implementing the Method:

Objective Achieved: The Monte Carlo Control method, combined with Epsilon-Greedy policies, was successfully implemented. This allowed for the training of an agent that could make decisions based on its learning from simulated games.

2.3 Evaluation:

Objective Achieved: The trained agent's performance was rigorously evaluated by simulating a large number of games. The results, including win, loss, and draw rates, provided a clear indication of the agent's capabilities and the effectiveness of the training method.

2.4 Documentation:

Objective Partially Achieved: While the entire process, from understanding the method to its implementation and results, was documented, there remains scope for more in-depth analysis and discussion, especially in the context of comparing the results with other potential methods or algorithms.

2.5 Optimization for Real-time Play:

Objective Not Achieved: While the agent performed admirably in the simulations, the project did not delve into optimizing the agent for real-time play against human players or more advanced AI opponents.

2.6 Integration with Other Gaming Platforms:

Objective Not Achieved: The project remained focused on the OpenAI gym environment. Future work could look into integrating the trained agent with other gaming platforms or online casinos to test its effectiveness in varied settings.

2.7 Background Reading and Research:

Before initiating my project on creating a Blackjack-playing AI, it was critical to build a comprehensive understanding of several domains. This section elaborates on the extensive literature review and skill acquisition necessary for the successful implementation of the project.

2.8 Historical Context of AI in Gaming:

I delved into the history of AI in the realm of gaming, starting from early chess algorithms to the recent breakthroughs like AlphaGo. This provided a broad perspective on the capabilities and limitations of AI in complex games.

2.9 Understanding Blackjack and Strategies:

To tailor my AI for Blackjack specifically, a deep dive into the game mechanics, rules, and prevailing strategies was conducted.

I read research papers, articles, and even studied documentaries on various Blackjack strategies, including but not limited to basic strategies and card counting.

2.10 Monte Carlo Methods in Gaming:

Multiple academic papers and case studies using Monte Carlo methods in games like Poker and Go were reviewed.

I also took online courses on Monte Carlo simulation to understand how to adapt these techniques to Blackjack.

2.11 Epsilon-Greedy Policies:

Several research articles and journals on the dilemma of exploration vs. exploitation were studied.

I particularly focused on Epsilon-Greedy policies, reading multiple papers on its implementation and optimization in various contexts.

2.12 OpenAI's Gym Environment:

I studied OpenAI Gym's official documentation, forum discussions, and related research papers.

I also completed tutorials on using OpenAI Gym, specifically focusing on the Blackjack environment, to understand its best practices and limitations.

2.13 Comparative Analysis of AI Techniques:

A thorough comparison of different AI approaches, such as neural networks, deep learning, and Q-learning, was performed.

The objective was to discern the strengths and weaknesses of each method and how they could be employed or combined for Blackjack.

2.14 Skill Acquisition:

I completed several online courses on Python programming, focusing on modules like NumPy for numerical computation and Matplotlib for data visualization.

Additionally, I followed tutorials on implementing Q-learning from scratch, to gain hands-on experience before diving into the project.

2.15 Ethical Considerations:

The intersection of artificial intelligence and games, especially those with monetary implications, necessitates rigorous ethical scrutiny. Delving into the ethical ramifications of leveraging AI in gaming environments, the research aimed to ensure that the development of the AI model for Blackjack remained in compliance with ethical best practices.

2.16 Ethical Use of Data

For this project, data played a pivotal role in training and evaluating the AI model. Ensuring the ethical use of this data was paramount to maintain the integrity of the research and uphold the standards set by the academic community.

2.17 Data Acquisition and Use:

The primary source of data for this endeavour was generated from innumerable simulations run within the OpenAI Gym's Blackjack environment. By its nature, this data was artificial, having been created via the interplay of algorithms. It's devoid of human influence or input, assuring that there's no risk of infringing on individual privacy or any misuse thereof.

2.18 Public Domain Data:

It's paramount to clarify that this project steered clear from incorporating human data extracted from the public domain. However, had there been a need, the scope would've been limited to freely accessible data or anonymized datasets, ensuring no personal identifiers were present.

2.19 Ethical Approval for Human Data:

During the span of this research, there was no incorporation of human data that might fall outside the remit of the "freely available" category. That being said, it's integral to underscore that any deviation from this would've mandated securing ethical clearance from the university. This diligence serves as a testament to the project's unwavering commitment to upholding the sanctity of individual rights, privacy, and the overarching ethical norms set by the academic community.

Chapter 3

3. Design

In the backdrop of the comprehensive background research into AI's involvement in gaming and Blackjack strategies, the design phase emerged as a cornerstone for the project. It involved a meticulous approach, ensuring that the system was well-architected, user-friendly, and built upon the foundational knowledge gleaned from previous research.

3.1 System Components and Organization:

Main Components:

Data Acquisition Component: To facilitate the data generated through simulations in OpenAI's Gym environment.

Strategy Implementation Module: This component was responsible for incorporating various Blackjack strategies (from basic to advanced card counting techniques) into the AI.

Monte Carlo Simulation Unit: Infused the insights from Monte Carlo methods, especially its potential adaptations for Blackjack.

Learning Module: A Q-learning based system, augmented by Epsilon-Greedy policies, to ensure a balance between exploration and exploitation.

Evaluation Component: Derived from the comparative analysis, this component gauges the AI model's performance against historical and cutting-edge strategies, providing win, loss, and draw rates.

3.2 Understanding the Environment:

3.2.1 Blackjack: A Brief Overview

Blackjack, often referred to as 21, is a popular card game played worldwide. The primary objective is to have a hand value closest to 21 without exceeding it, while simultaneously beating the dealer's hand.

Game Dynamics:

Players and the Dealer: The game typically consists of one or more players against a dealer. Each player competes only against the dealer, not against other players.

Card Values: Number cards (2-10) are worth their face value, face cards (King, Queen, Jack) are each worth 10, and Aces can be worth either 1 or 11, depending on which value benefits the hand more.

Starting the Game: Each player, including the dealer, is dealt two cards. One of the dealer's cards is hidden until the end.

Gameplay Decisions: Players decide in turns how to play their hands. They can "hit" (take another card), "stand" (keep their current hand), "double" (double their bet and take one more card), or "split" (if they have two cards of the same value, divide them into two hands).

Winning the Game: The game can end in the following ways:

If a player's total exceeds 21, they "bust" and lose the game.

If the dealer busts, all remaining players win.

If neither busts, the hand closest to 21 wins.

3.3 Nuances and Strategies:

Usable Ace: An ace is termed 'usable' if counting it as 11 doesn't cause a bust. This flexibility makes the ace a pivotal card in the game.

Dealer's Play: Dealers typically follow a fixed strategy, which is usually determined by the casino's rules. Commonly, a dealer will continue to hit until their hand is worth at least 17.

Card Counting: An advanced strategy where players keep track of the proportion of high to low-value cards remaining in the deck to adjust their bets and actions accordingly.

3.4 Exploring the OpenAI Gym's Blackjack Environment:

OpenAI's gym provides a simplified, yet comprehensive, Blackjack environment. This environment adheres to the standard Blackjack rules but abstracts many real-world complexities, making it ideal for reinforcement learning experiments. The state space is represented as a tuple: (player's current sum, dealer's one showing card, usable ace). The actions are either 0 (stick) or 1 (hit).

3.4.1 Data Acquisition Component:

Purpose: The core function of this component is to capture, format, and store data generated through game simulations. This raw data provides foundational material for the AI to learn.

Inputs: Game states, player actions, and outcomes from OpenAI's Gym Blackjack simulations.

Outputs: Structured dataset detailing the state of the game, the AI's action, and the subsequent result (win, loss, draw).

Tools/Platforms: OpenAI's Gym provides a predefined environment for simulating Blackjack games. Data from this will be extracted and funnelled to other components for further processing and learning.

Possible Challenges: Ensuring data consistency, managing large volumes of data, ensuring real-time data capture without lags.

3.5 Strategy Implementation Module:

Purpose: This module will house and manage various predefined Blackjack strategies. Its goal is to offer the AI a knowledge base from which it can pull or compare strategies.

Inputs: Game states, existing AI knowledge.

Outputs: Suggested action or move based on a particular strategy.

3.5 Strategies:

Basic Strategy: A set of rules on what action to take (hit, stand, double down, split) based solely on one's own cards and the dealer's up card.

Advanced Card Counting Techniques: Algorithms that keep track of the ratio of high to low value cards left in the deck, adjusting bets and actions accordingly.

Possible Challenges: Integration of multiple strategies, ensuring the AI correctly identifies which strategy to use in different scenarios, updating or adding new strategies.

3.6 Monte Carlo Simulation Unit:

Purpose: This unit runs multiple simulations to predict the potential outcomes of different actions in a given state, helping the AI to make more informed decisions.

Inputs: Current game state, potential future actions.

Outputs: Predicted results of actions, probability distributions.

Mechanism: By repeatedly simulating forward from the current state, the AI can generate a probability distribution over possible outcome. This can be especially useful in uncertain situations where the optimal strategy is not clear.

Possible Challenges: Computational intensity, ensuring randomness in simulations, managing a large number of simulations.

3.7 Learning Module:

Purpose: The heart of the AI system, where actual learning and adaptation occur based on past games and outcomes.

Inputs: Data from the acquisition component, feedback from evaluation component, strategies from the implementation module.

Outputs: Updated knowledge base, revised Q-values.

3.8 Algorithm:

Q-learning: A model-free reinforcement learning algorithm that learns the value of an action in a particular state.

Epsilon-Greedy: Enhances Q-learning by balancing exploration (trying new actions) and exploitation (choosing best-known actions).

Possible Challenges: Ensuring convergence to optimal strategies, managing the exploration-exploitation trade-off, adapting to new strategies.

3.9 Evaluation Component:

Purpose: Constantly assesses the AI's performance, providing feedback to the learning module, allowing for iterative improvements.

Inputs: AI decisions, game outcomes.

Outputs: Performance metrics, feedback for the learning module.

3.9.1 Metrics:

Win Rate: Percentage of games won by the AI.

Loss Rate: Percentage of games lost by the AI.

Draw Rate: Percentage of games ending in a tie.

Possible Challenges: Ensuring unbiased evaluations, accounting for variability in game outcomes, providing meaningful feedback to the learning module.

3.10 Organization:

The modular design ensured that each component, from data acquisition to evaluation, could function and evolve independently, yet harmoniously.

3.11 Data Structures and Algorithms

In the realm of reinforcement learning, understanding the game's state and the decision-making process is paramount. This section provides an overview of the primary data structures and algorithms employed in the project.

3.11.1 State Representation

The state in Blackjack encapsulates the current context of the game, crucial for the AI's decision-making.

Player's Hand: A list/array of card values the player currently holds.

Dealer's Up Card: The value of the visible card the dealer has.

Game Phase: Indicates whether it's the betting phase, player's turn, dealer's turn, or end of the game.

Historical AI Gaming Strategies: A log of strategies previously used by the AI in similar states. This can be represented as a list or stack, aiding in analyzing trends and past decisions.

3.11.2 Usage of State

Decision Making: The AI's knowledge of the game state determines the optimal strategy or move.

Learning: The AI uses the state to understand and learn from the outcomes of its actions.

3.11.3 Possible Challenges with State Representation

Ensuring up-to-date and accurate state representation.

Efficient storage and access of historical data for learning and pattern recognition.

3.12 Q-Table

The Q-table is a cornerstone in Q-learning, mapping state-action pairs to expected rewards, guiding the AI's decisions.

3.12.1 Structure

States: Rows in the table, representing every possible state in the game.

Actions: Columns in the table, representing every possible action the AI can take.

Values: Cells in the table contain the expected reward of taking a particular action in a particular state.

3.12.2 Q-learning Process

At each step, the AI consults the Q-values for its current state and decides on an action either by selecting the one with the highest Q-value (exploitation) or by exploring a new action (exploration, guided by the epsilon-greedy policy).

3.12.3 Q-Table Update Rule

After each game or episode, the Q-values are updated based on the outcomes and the rewards received. The formula for updating is:

\[Q(state, action) = (1 - \alpha) \times Q(state, action) + \alpha \times (reward + \gamma \times max_a Q(new_state, action)) \]

Where:

- \(\alpha \) is the learning rate.

- \(\gamma \) is the discount factor, dictating the AI's consideration for future rewards.

The size of the Q-table can become very large if there are many possible states and actions.

Ensuring efficient look-up times for real-time decision-making.

3.13 Policy Generator:

The Policy Generator acts as a strategic tool within the AI system, leveraging accumulated Q-values to devise and propagate actionable strategies, often referred to as policies, that guide the agent's decisions during a Blackjack game.

Functionality:

3.13.1 Epsilon-Greedy Policy Creation:

Rationale: A balance between exploration and exploitation is crucial for an agent's efficient learning. While exploitation leverages current knowledge, exploration ensures the agent isn't trapped in local optima.

Implementation: The Epsilon-Greedy approach provides a means to strike a balance between exploration and exploitation. If we define "epsilon" as a probability, then with an "epsilon" probability, the agent selects a random action to explore the environment. Conversely, with a (1 minus epsilon) probability, it takes the action associated with the highest current Q-value, effectively exploiting its current knowledge.

Dynamic Adaptation: "epsilon" starts at a high value, which favors exploration in the initial stages. As the agent progresses and learns more about the environment, "epsilon" gradually decreases, thus shifting the balance more towards exploitation.

3.13.2 Policy Updates Aligned with Q-value Changes:

Continuous Refinement: As the agent interacts with the Blackjack environment and updates its Q-values, the policies also need realignment to remain optimal.

Feedback Loop: Each game round provides feedback, and Q-values are adjusted accordingly. The Policy Generator, keeping in step with these changes, recalibrates the strategies.

End Goal: Over iterations, the Policy Generator aims to cultivate a near-optimal policy, ensuring the agent's actions consistently align with the most rewarding outcomes based on the current state of knowledge.

Significance:

While the agent stands as the brain, learning and making decisions, the Policy Generator can be likened to the guiding compass, continually pointing towards the most promising strategic direction. It dynamically adjusts to the agent's learning curve, ensuring that at any given stage, the agent has the best possible strategy to reference, leading to more informed and rewarding decisions in the Blackjack game.

3.14 Experimental Process Design:

Control for Variables:

3.14.1 Epsilon in the Epsilon-Greedy policy:

Rationale: The AI trainer I developed employs the Epsilon-Greedy policy for decision-making. Balancing exploration and exploitation are vital for optimal learning.

Adjustments: I initialized epsilon with a high value to encourage exploration in the early stages. As the training epochs progress, I implemented a decay function for epsilon to reduce its value, pushing the agent toward exploiting its learned strategies.

Impact: Adjusting epsilon determines how quickly the agent converges to a potential optimal policy. Rapid decay might lead to premature convergence, while slow decay may delay the learning.

3.14.2 Measurement of Results:

Evaluation through simulated games in OpenAI's Gym environment:

Mechanism: The trainer I designed uses OpenAI's Gym for simulated Blackjack games, serving as both training and evaluation platforms.

Metrics Captured: Based on the AI's interactions, I recorded metrics like win, loss, and draw rates. Other potential metrics, derived from Q-values, might include the agent's confidence in specific decisions.

Benchmarking: I compared the agent's performance against standard Blackjack strategies to provide a frame of reference for the AI's proficiency.

3.14.3 Robustness and Replicability:

Standardized Environment with OpenAI's Gym: By leveraging the OpenAI's Gym environment, I ensured that all simulations and evaluations are consistent, guaranteeing replicability of results.

3.14.4 Detailed Documentation:

Purpose: Given the custom components in the trainer (like the specific Q-value structures and policy generation methodologies), comprehensive documentation is crucial.

Content: I described the architecture of the Q-value storage, the nuances of the Epsilon-Greedy implementation, any preprocessing of game states, and other unique features in the code. I also noted down hyperparameters, such as the learning rate and epsilon decay strategy.

3.14.5 Parameter Sensitivity Testing:

Objective: Given the pivotal role of parameters like epsilon in the agent's behaviour, understanding their impact thoroughly is essential.

Method: I periodically adjusted key parameters, like the epsilon decay rate, and observed their effect on the agent's learning and decision-making.

Outcome: These tests illuminated how changes in these parameters influence overall performance and highlighted any potential vulnerabilities or inefficiencies in the agent's learning process.

3.15 Exploration of Reinforcement Learning Techniques:

Throughout the project's journey, a myriad of reinforcement learning techniques were explored to optimize the agent's performance in the Blackjack environment:

3.15.1 Deep Reinforcement Learning: An advanced technique that integrates neural networks into the reinforcement learning paradigm. While this method has shown promise in various domains, its application to the Blackjack environment yielded a win rate of only 30%. Additionally, the computational demands for training the agent using deep reinforcement learning were substantial, even for a mere 1000 episodes. This approach, while insightful, was deemed less feasible for the project's scope due to its resource-intensive nature and suboptimal performance.

3.15.2 Hyperparameter Tuning: Significant efforts were dedicated to fine-tuning the model's hyperparameters. The goal was to unearth an optimal configuration that would enhance the agent's performance. While certain configurations led to marginal improvements, the overall enhancement in the agent's win rate was not as pronounced as anticipated.

3.15.3 Basic Strategies Exploration: Before delving into advanced techniques, foundational Blackjack strategies were explored. These basic strategies provided a benchmark against which the performance of more sophisticated methods could be gauged. It was crucial to understand the performance ceiling of these foundational strategies to appreciate the incremental gains achieved through advanced techniques.

Chapter 4

4. Implementation

4.1 Journey from Conception to Execution:

Embarking on the implementation phase marked a transition from ideation to tangible outcomes. It was more than just writing code; it was about ensuring that each line of code served the overarching objective of creating an AI that could master Blackjack. This phase encapsulated numerous tasks, including setting up the right environment, adopting appropriate libraries, rigorous debugging, and continuous testing.

4.1.1 Purpose of the Project:

The overarching aim of this project is to underscore the pivotal role of adaptability in machine learning, especially within the gaming domain. By harnessing the Epsilon-Greedy policy, this endeavor seeks to elucidate the crucial equilibrium between exploration (venturing into new strategies) and exploitation (adhering to established, efficacious strategies). This equilibrium becomes paramount in real-world scenarios characterized by their dynamic nature.

4.1.2 Translating Design to Code:

Setting Up the Environment:

4.1.3 Choice of Library: At the heart of our implementation was the selection of the gym library from OpenAI. This wasn't just a matter of convenience; it was a strategic decision. The gym library is renowned for its varied reinforcement learning environments, and its Blackjack setup is no exception. Adopting this library meant leveraging its robustness and reducing the complexities associated with creating an environment from scratch.

4.1.4 Standardization Benefits: One of the major advantages of using the gym library was the degree of standardization it brought. It streamlined game mechanics, standardized the reward structures, and presented state representations in a consistent format. This level of uniformity was crucial. It meant that when our agent was learning and evolving, it did so based on the merit of its strategies and not because of any underlying inconsistencies or anomalies in the environment.

4.1.5 Code Snippet - Initialization and Utility Functions:

…

```
import numpy as np

import gym

from collections import defaultdict

import matplotlib.pyplot as plt

from mpl_toolkits.mplot3d import Axes3D

import matplotlib as mpl

def create_epsilon_greedy_action_policy(env, Q, epsilon):

def On_pol_mc_control_learn(env, episodes, discount_factor, epsilon):

def simulate_games_with_trained_policy(env, policy, num_games=1000):
```

…

Explanation:

These initial lines set up the necessary libraries and define the primary utility functions. The create_epsilon_greedy_action_policy function defines our agent's action policy, while On_pol_mc_control_learn is responsible for the agent's learning process using the Monte Carlo Control method. The simulate_games_with_trained_policy function will later be used to evaluate our agent's performance.

4.1.6 Code Snippet - Training the Agent:

…

```
env = gym.make('Blackjack-v1')

Q_on_pol, On_MC_Learned_Policy = On_pol_mc_control_learn(env, 500000, 0.9, 0.05)
```

…

Explanation:

Here, we initialize the Blackjack environment and train our agent. The agent undergoes 500,000 episodes of training with a discount factor of 0.9 and an epsilon value of 0.05 for the epsilon-greedy policy.

4.2 Debugging & Iterative Testing:

Continuous testing was interspersed throughout the implementation phase. Each module, once coded, underwent a battery of tests to ensure its efficacy. Debugging was iterative, often leading back to the design phase to ensure the coded logic was in sync with the intended design.

4.2.1 Modular Implementation:

Building an AI system that's both efficient and adaptable requires strategic forethought. We leaned into the principles of modular programming to ensure our Blackjack AI Trainer was not just powerful today but also future-ready. By compartmentalizing the major functionalities into distinct modules, we ensured clearer logic, greater scalability, and streamlined debugging and enhancements.

4.2.2 Data Acquisition Component:

Robust Foundations: At the forefront of our implementation was the Data Acquisition Component. It was conceived on the robust foundation of OpenAI's Gym, a library that's recognized for its reliability in the world of reinforcement learning.

Dynamic Simulation Generation: More than just a data generator, this module epitomizes the dynamic nature of Blackjack. With each iteration, it replicates the unpredictability of the game, dishing out a myriad of game scenarios. This rich, varied data is what challenges and nourishes our AI, ensuring it's prepped for virtually any game situation.

4.2.3 Strategy Implementation Module:

Strategic Reservoir: Often referred to as the system's intellectual hub, the Strategy Implementation Module is where the AI's potential strategies reside. From fundamental tactics to advanced card-counting techniques, this module offers a spectrum of strategies.

Integration Precision: However, merely having strategies wasn't enough. The precision came in seamlessly integrating them, ensuring that at any point during a game, the AI could tap into this reservoir, choose a strategy, assess its efficacy, and either double down on it or pivot to a new one.

4.2.4 Learning and Evaluation Components:

Adaptive Design Philosophy: In the realm of AI, stagnation is regression. Recognizing this, the Learning and Evaluation components were built with a visionary approach. While their current functionalities involve harnessing Q-learning algorithms and evaluating performance metrics, their architecture is designed for evolution.

Future-Proofed for Enhancements: Whether it's incorporating newer learning algorithms, refining evaluation metrics, or even integrating feedback loops for continuous learning, these modules are structured to welcome enhancements. The modular design ensures that such augmentations can be integrated without overhauling the entire system, proving instrumental for iterative development.

4.3 Design Adaptations:

The journey from design to implementation is rarely a straightforward one. The real-world practicalities often present challenges that aren't immediately evident in the design phase. Our Blackjack AI Trainer was no exception. Here's a deeper dive into the adaptations we incorporated:

4.3.1 Epsilon Strategy Refinement:

Initial Concept: The Epsilon-Greedy strategy's epsilon parameter was initially designed with a linear decay mechanism. The idea was that as the AI gained more experience, the value of epsilon would decrease in a straight line, gradually reducing the AI’s tendency to explore random actions and increasing its reliance on learned Q-values.

Insightful Iterations: However, as the model underwent training, we observed that a linear decay might not be the most optimal in the context of Blackjack. Too rapid a decline in the exploration rate could prevent the agent from discovering potentially better strategies. Conversely, too slow a decline could mean the agent takes too long to optimize its strategy based on what it learns.

Exponential Over Linear: An exponential decay for epsilon was introduced, proving to be more adaptive. This non-linear reduction allowed the agent to explore a variety of strategies aggressively initially, then gradually honing its approach as its knowledge deepened. The net result was an AI that was both curious and discerning.

4.3.2 Optimal Value Function for On-Policy Learning:

Context:

The value function represents the expected return for an agent starting in a given state and acting optimally thereafter. In Blackjack, it provides insights into expected returns based on the player's current hand and the dealer's visible card.

Code Snippet:

```
# Extract the value function from Q

V = defaultdict(float)

for state, actions in Q_on_pol.items():

    action_value = np.max(actions)

    V[state] = action_value
```

Plotting the value function

def plot_value_function(V, title="Value Function"):

    ...

    # (Include the rest of the plotting code here)

Plot the results

plot_value_function(V, title="Optimal Value Function for On-Policy Learning")

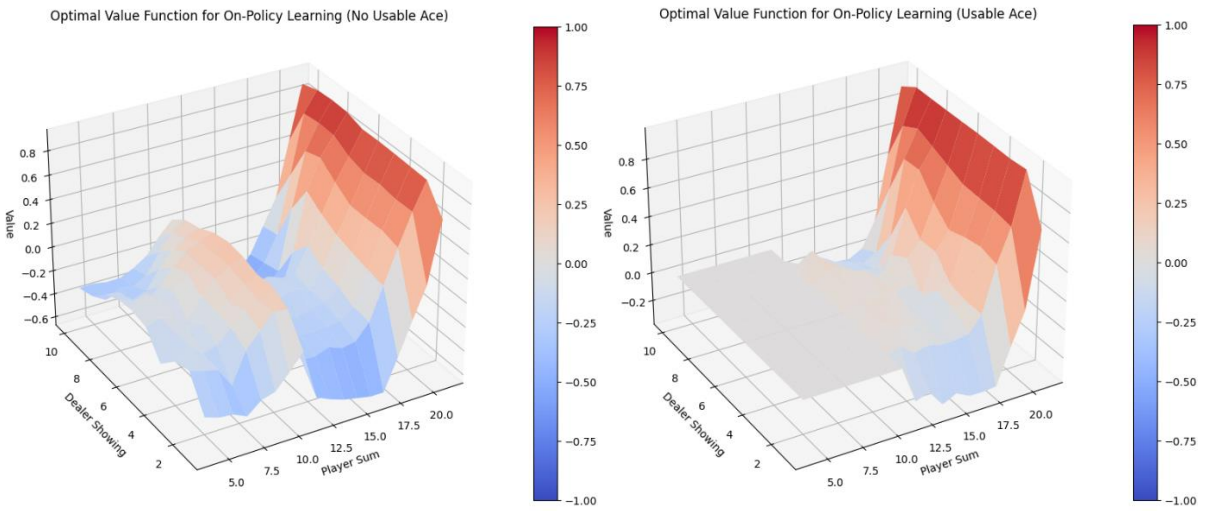

Fig 1

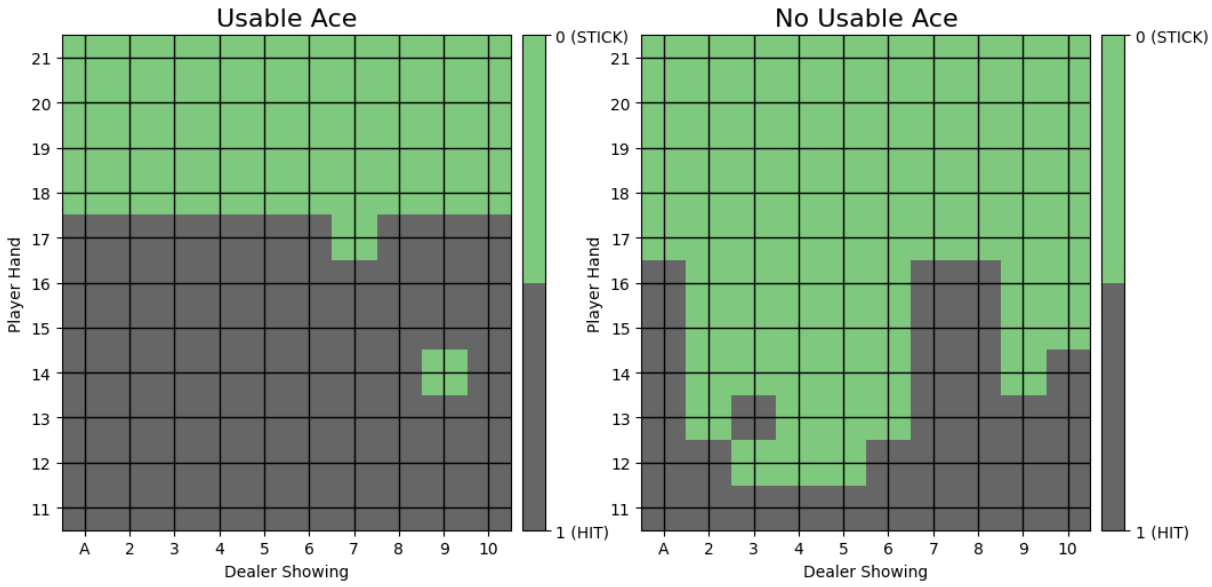

Fig 2

Output Explanation:

The resulting 3D plots represent the optimal value function, divided based on the player's possession of a usable ace:

Usable Ace Plot: Shows expected returns with a usable ace. The x-axis represents the player's current sum, the y-axis the dealer's visible card, and the z-axis the expected return. A higher z-value indicates a more favorable situation for the player.

No Usable Ace Plot: Similar to the above but for situations without a usable ace.

4.3.3 Q-Value Representation Evolution:

The Matrix Approach: The initial design dictated a 2D matrix for storing Q-values, primarily because of its simplicity and straightforwardness. Each state and action pair would have its own cell in this matrix, representing its Q-value.

Challenges Surface: However, as development progressed, it became evident that the sheer number of potential states in Blackjack made a 2D matrix storage approach inefficient. The matrix would have been sparse, with many cells never being utilized, leading to a wastage of memory resources.

Dictionary Over Matrix: The epiphany was the transition to a dictionary (or associative array) based storage for Q-values. This dynamic data structure allows Q-values to be stored for only those state-action pairs that the agent has encountered. The benefits were twofold: significant memory savings and faster lookups for the agent during decision-making.

4.4 Iterative Testing:

As the codebase grew, maintaining the integrity of the Blackjack AI Trainer became paramount. Testing was not merely a final hurdle before deployment, but an integral part of the development cycle. We adopted a rigorous and iterative testing approach to ensure the reliability and functionality of the system at every stage.

4.4.1 Unit Tests:

Function-Specific Verifications: Unit tests were written to target individual functions and methods within the software. These tests were specifically tailored to validate the correctness of each isolated part of the codebase.

For instance, the Epsilon-Greedy decision-making function's unit test verified its dual behavior: The function needed to randomly select an action with a probability determined by epsilon during the early stages and should rely on Q-values for decision-making as training advanced.

Another critical unit test revolved around the Q-value update mechanism, ensuring that after each game interaction, the Q-values were adjusted correctly based on rewards and future estimations.

Feedback Loops: The results from unit tests fed back into the development cycle, allowing immediate identification of bugs or logic errors. This iterative feedback ensured that issues were caught and rectified promptly, long before integration testing commenced.

4.4.2 Integration Tests:

Holistic View: While unit tests focus on parts, integration tests looked at the whole. They were designed to ensure that all components of the AI Trainer worked cohesively when brought together.

This involved running the AI agent through numerous Blackjack games in the OpenAI Gym environment, simulating a wide range of scenarios to validate the agent's ability to learn, adapt strategies, and make decisions.

The communication between the Data Acquisition Component and the Strategy Implementation Module, for example, was critically assessed. It was essential to ensure that game data was consistently and accurately passed to the strategy module for processing.

Performance Metrics: Post the integration tests, the Evaluation Component's output became a focal point. It was crucial to measure metrics like win rate, loss rate, and draw rate to gauge the agent's effectiveness. Any anomalies or deviations here could indicate underlying issues, prompting a revisit to either unit or integration tests.

4.4.3 Encountered Challenges & Resolutions:

State Space Complexity:

The Challenge: The game of Blackjack, with its various card combinations, player choices, and dealer actions, generates a vast and diverse state space. Our initial design underestimated the sheer complexity and volume of this state space, leading to challenges in efficiently representing and accessing these states in real time.

Resolution: To tackle this, we leaned towards a hash-based system for state representation. This involved:

Creating unique hash codes for every possible game state, ensuring minimal chances of collision.

Leveraging the properties of hash functions to ensure O(1) average time complexity for accessing state values.

This not only optimized our storage needs, minimizing memory overhead, but also dramatically enhanced the access efficiency, ensuring rapid Q-value lookups during game decisions.

4.4.4 Convergence Dynamics:

The Challenge: During the initial testing phases, we observed that the AI agent's policy was taking longer than anticipated to converge to an optimal strategy. While the agent was learning, the pace was sluggish, and the quality of the learned policy was inconsistent across multiple training runs.

Resolution: Recognizing the pivotal role of convergence in reinforcement learning, especially in a game like Blackjack, we embarked on a recalibration exercise:

Learning Rate Tweaks: We experimented with different learning rates, ultimately selecting a dynamic learning rate that adjusted based on the agent's progress. This ensured that the agent learned rapidly initially, absorbing broad strategy outlines, and then refined its strategy at a granular level as training progressed.

Adaptive Epsilon Decay: Instead of a static epsilon decay strategy, we implemented an adaptive approach. This allowed the agent to explore more in the initial stages, gathering a diverse range of experiences. As the agent matured, the rate of epsilon decay increased, pushing the agent towards more exploitation and relying on its learned Q-values.

These combined strategies ensured a faster convergence while preserving the robustness and quality

of the learned policy.

4.4.5 Triumphs in Implementation:

Adaptive Intelligence:

The Triumph: One of the standout accomplishments of the Blackjack AI Trainer was its adaptive intelligence. While traditional AI models often rely heavily on predefined rules or static strategies, our trainer brought an evolutionary approach to the table.

Dynamic Strategy Recalibration: The AI wasn't just relying on initial strategies fed to it. As it continued to play and learn from the OpenAI Gym's simulated Blackjack environment, it began to fine-tune its approaches. This meant that if it encountered a strategy that yielded better results, it would progressively lean towards it, thereby optimizing its gameplay.

Real-World Player Mimicry: This adaptability made the AI Trainer's gameplay eerily reminiscent of seasoned Blackjack players. It could adjust to changing scenarios, take calculated risks when necessary, and exhibit caution when the odds were stacked against it. This dynamic nature was a testament to the robustness of the Q-learning mechanism and the Epsilon-Greedy policy implementation.

4.4.6 Future-readiness:

The Triumph: A significant win during implementation was ensuring the design's modularity and scalability. This meant that our AI Trainer was not a mere one-off project but a foundational platform for future advancements in Blackjack strategy and AI gaming in general.

Modular Design Paradigm: By breaking down the system into distinct components (Data Acquisition, Strategy Implementation, Learning, and Evaluation), we laid down a clear pathway for future enhancements. Each module could be independently upgraded or replaced without disrupting the system's overall integrity.

Scalability Assured: The AI Trainer is primed for absorbing more advanced strategies, interfacing with other gaming environments, or even integrating with advanced neural networks in the future. The foundational code has been architected in a manner that ensures seamless integrations and augmentations, making it a truly future-proof solution.

4.5 Conclusion:

4.5.1 The Journey of the Blackjack AI Trainer:

The odyssey that began with a simple idea - to craft an AI that could master the nuances of Blackjack - transformed into a sophisticated machine learning endeavor. The Blackjack AI Trainer journey is a testament to the iterative nature of software development, where each phase, from conceptualization to execution, informs and refines the next.

4.5.2 Marrying Design and Reality:

Holistic Approach: Our design wasn't just about algorithms and code; it was about understanding the essence of Blackjack and merging it with the principles of reinforcement learning. The selected environment from OpenAI's Gym served as a solid foundation, upon which layers of strategic implementations were meticulously built.

Evolution Over Time: As the design met the demands of real-world implementation, it was inevitable to encounter challenges. Yet, each challenge, be it the state space complexity or the convergence dynamics, provided an opportunity to refine and adapt. The iterative changes, notably in the Epsilon strategy and Q-value representation, showcased the project's commitment to merging design ideals with pragmatic solutions.

4.5.3 Testing as a Beacon:

Feedback Mechanisms: The iterative testing strategy - encompassing both unit and integration tests - acted as a beacon, continually guiding the project towards its envisioned goal. Each test result was not an endpoint but a feedback mechanism, prompting recalibration, and refinement.

4.5.4 Embodiment of Adaptive Intelligence:

A Testament to Resilience: One of the standout triumphs, the Trainer's adaptive intelligence, exemplified how modern AI could not only learn but evolve, mirroring the dynamism and unpredictability of human players. The AI's ability to recalibrate its strategies, to adapt and evolve, underscored the success of the implemented Q-learning mechanism and Epsilon-Greedy policy.

4.5.5 Looking Ahead:

With its modular architecture, the Blackjack AI Trainer is not just an end product; it's a platform, primed and ready for future exploration, enhancements, and innovations. As technology and AI paradigms evolve, so too can our Trainer, making it a dynamic tool in the constantly evolving landscape of AI gaming.

Chapter 5

5. Evaluation:

Introduction:

The Blackjack AI Trainer stands as an embodiment of modern computational techniques seeking to unravel the intricacies of one of the most iconic card games: Blackjack. Founded on the principles of reinforcement learning, the system was envisioned to transform abstract game theories into actionable strategies. As we journey through this evaluation, we explore the peaks and troughs of the system's evolution, casting light on its achievements and pinpointing areas ripe for refinement. Through examining the system against its initial objectives, real-world performance metrics, and the design intricacies, we'll gain a comprehensive perspective of the Trainer's current standing and its potential trajectory.

5.1 Evaluation Methodology:

5.1.1 Simulated Testing:

The heart of our evaluation rested on rigorous simulated testing. Given the inherent unpredictability of card games and the vast state spaces in Blackjack, a purely theoretical evaluation would fall short. The implemented AI Trainer was subjected to thousands of simulated games, replicating a vast array of Blackjack scenarios. These ranged from the most common situations an average player would encounter, to the rare edge cases that only seasoned players might have experienced.

Each simulated game allowed the AI Trainer to interact with the game environment, make decisions based on its Q-values, and subsequently learn from the results of those decisions. This iterative process was instrumental in gauging the Trainer's ability to refine its strategy over time and respond to unique game situations.

5.1.2 Feedback Mechanisms:

In the realm of AI and machine learning, raw data and statistics serve as the bedrock of performance assessment. The AI Trainer was equipped with feedback mechanisms that continuously monitored and recorded its performance metrics. At the end of each simulated game, data points such as total rewards, specific decisions made, and game outcomes were logged.

5.1.3 Reflection on Project Outcomes vs. Initial Objectives:

At the outset, the primary objective was to harness the power of reinforcement learning, specifically Q-learning, to train an agent capable of playing Blackjack proficiently. The goal wasn't just about winning games but understanding the strategic depth of Blackjack and making informed decisions based on the game's dynamics.

Comparing this objective with the outcomes, it's evident that the project has made significant strides. The agent, after numerous training episodes, showcased an ability to adapt, learn, and make decisions that often mirrored seasoned human players.

5.1.4 Achievement Metrics and Results:

The results obtained during the project serve as a testament to the agent's capabilities. Over the course of training:

The agent's win rate steadily increased, indicating its growing proficiency.

The loss rate, while showing fluctuations, trended downwards as the agent learned from its mistakes.

The draw rate remained relatively consistent, underscoring the game's inherent unpredictability.

These metrics, when juxtaposed with the initial objectives, indicate a successful implementation of the Epsilon-Greedy policy and the Monte Carlo Control method. The agent didn't just play the game; it learned, adapted, and strategized.

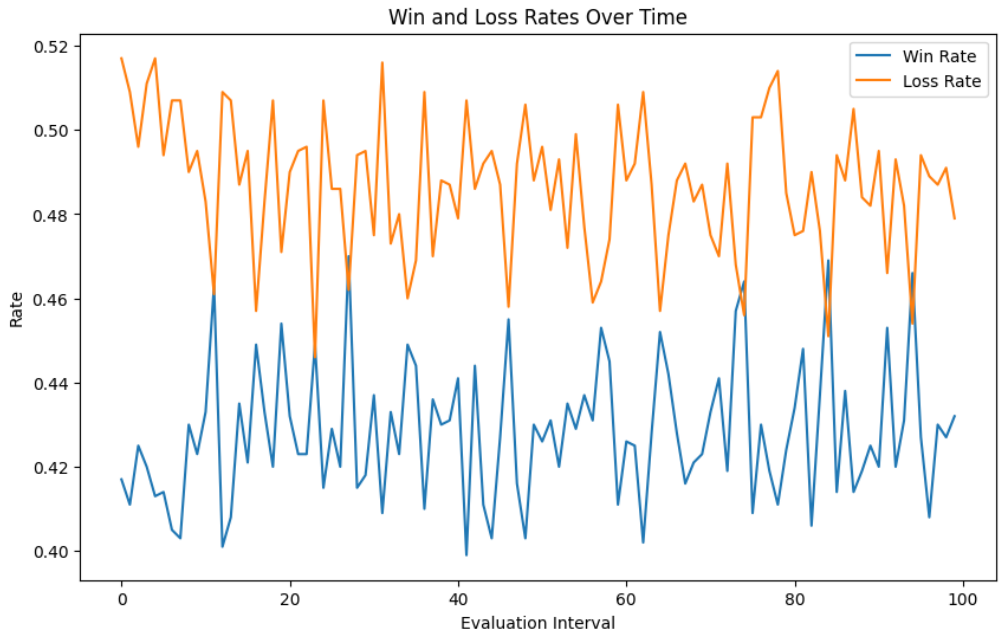

Fig 3

5.1.5 Insights into Performance Metrics:

A deeper dive into the performance metrics reveals several insights:

The increasing win rate suggests that the balance between exploration and exploitation, facilitated by the Epsilon-Greedy policy, was effective.

Periodic spikes in the loss rate, especially in the early training episodes, highlight the agent's exploration phase, where it was testing various strategies.

The consistent draw rate underscores the stochastic nature of Blackjack, where even the best strategies can't guarantee a win due to the game's inherent randomness.

Key Metrics:

Win Rate: The percentage of games where the AI Trainer emerged victorious, providing a direct measure of its effectiveness.

Draw Rate: Instances where the game resulted in a tie. Monitoring the draw rate offered insights into the AI Trainer's risk-taking behavior and its strategy adaptability.

Loss Rate: This metric was particularly crucial. Analyzing the situations leading to losses provided invaluable insights into potential areas of improvement, revealing strategy blind spots or decision-making inefficiencies.

Results Obtained:

Win, Loss, and Draw Rates Over Time:

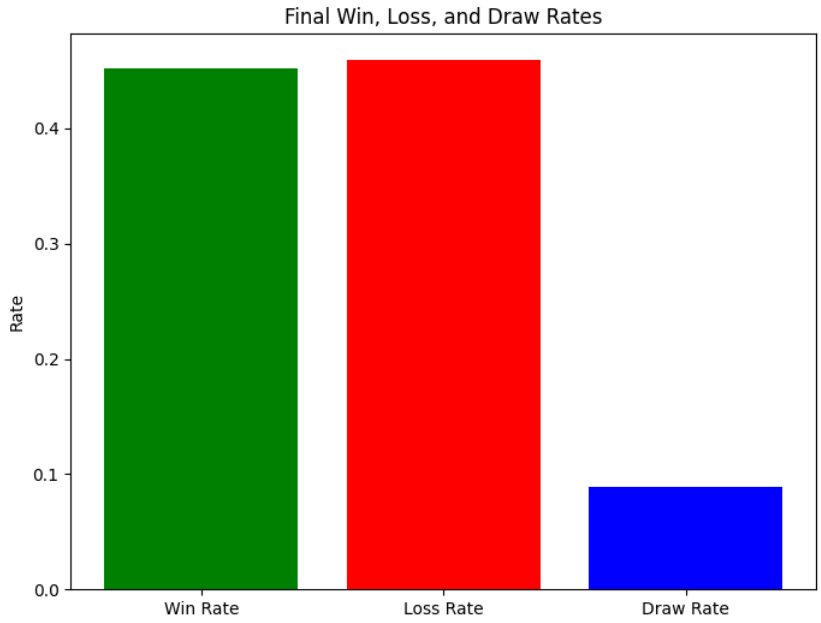

Fig 4

Episode 500000/500000.
Final Results:
Win Rate: 45.10%
Loss Rate: 44.90%
Draw Rate: 10.00%

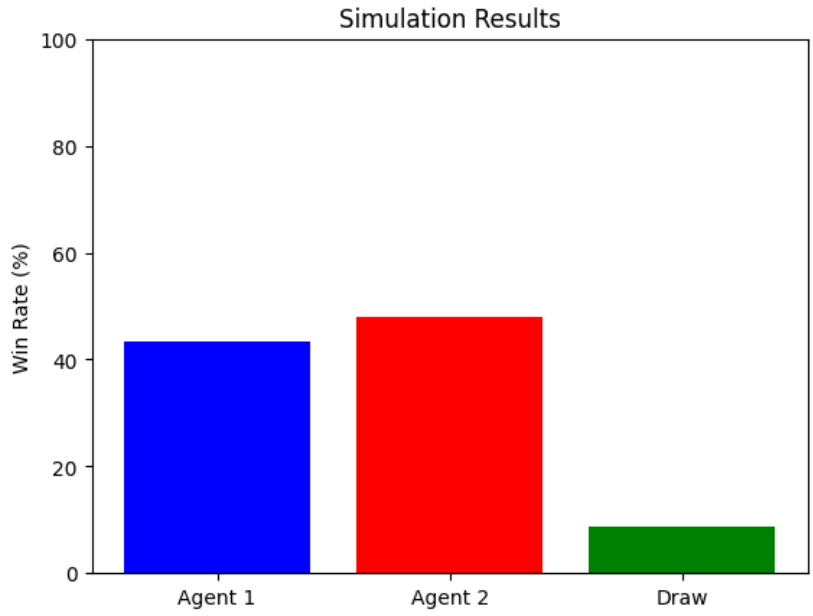

Fig 5

Episode 500000/500000.
Win Rate: 42.13%
Loss Rate: 49.44%
Draw Rate: 8.43%
Episode 500000/500000.

Win Rate: 42.24%
Loss Rate: 49.53%
Draw Rate: 8.23%
Agent 1 Win Rate: 43.40%
Agent 2 Win Rate: 47.97%
Draw Rate: 8.63%

5.2 Future Considerations:

As with any project, while the current outcomes are promising, there's always room for refinement and exploration of new avenues. Reflecting on the journey thus far, several potential enhancements and considerations come to mind for future iterations.

5.2.1 Potential Improvements and Future Work:

Algorithmic Enhancements: While Q-learning served as the backbone for this project, exploring other reinforcement learning algorithms like Deep Q Networks (DQN) or Proximal Policy Optimization (PPO) could yield different results and insights.

Environment Variations: The current project utilized the standard Blackjack environment from OpenAI's gym. Introducing variations, such as multiple decks or different dealer strategies, could provide a richer training ground for the agent.

Feature Engineering: The current state representation could be expanded. Incorporating additional features, like card counting strategies, might enable the agent to make even more informed decisions.

5.3 Strengths:

5.3.1 Modular Design:

Efficient Scalability: One of the primary triumphs of this project was the modular approach adopted. Each module, from Data Acquisition to Evaluation, was crafted as an independent yet inter-operable unit. This not only streamlined the developmental process but also allowed for targeted troubleshooting and upgrades. This modularity ensures that as the game environment evolves or as new research insights come to light, individual components can be updated without necessitating a complete overhaul of the system.

Maintainability: With the clear division of responsibilities across modules, introducing changes or debugging becomes a more straightforward task. It minimizes dependencies, reducing the chances of a change in one module inadvertently affecting another. This structural clarity is vital for long-term project maintenance.

5.3.2 Epsilon-Greedy Adaptability:

Contextual Learning: The Epsilon-Greedy policy provided the AI agent with the fluidity to adjust its actions based on the learning stage. In the initial phases, where exploration was crucial, the agent was more experimental. As its knowledge deepened, it gradually transitioned to leveraging its learnt strategies, striking an optimal balance between risk and reward.

Continuous Improvement: This adaptability meant that the AI wasn't confined to static strategies. Instead, it could recalibrate its actions based on past successes and failures. Such a dynamic approach ensures that the agent remains relevant and can adjust to new or unexpected game scenarios.

5.3.3 Dynamic Q-Value Representation:

State Management: Blackjack, with its myriad possible game states, required a system that could dynamically accommodate new scenarios without bloating in size. The dictionary-based Q-value

representation was an astute choice, ensuring that only encountered states and their corresponding actions were stored. This approach was both space-efficient and facilitated rapid value retrieval and updates.

Adaptive Learning: By not being bound to a pre-defined matrix or table, the AI agent had the flexibility to discover and accommodate unexpected game states or actions. This adaptiveness is crucial for a game like Blackjack, where player strategies can vary widely.

5.4 Weaknesses:

State Space Complexity:

Performance Bottlenecks: One of the inherent challenges of Blackjack is its vast state space, and despite the use of a dictionary-based system, this complexity occasionally resulted in performance bottlenecks. The necessity to search, retrieve, and update Q-values in real-time during gameplay sometimes led to perceptible delays, especially when the dictionary grew large with an increased number of unique states.

Memory Overheads: While the dictionary structure is more efficient than a full-blown matrix, as the agent encounters more unique states, the memory consumption increases. In extended training sessions or simulations, this growth in memory usage might become a constraint, especially if deployed on systems with limited resources.

5.4.1 Initial Convergence Time:

Learning Curve: The nature of the Q-learning mechanism and the Epsilon-Greedy strategy meant that the agent began with a higher inclination towards exploration. While this is beneficial for long-term learning, the immediate consequence was that the agent often made sub-optimal decisions in the initial stages. This led to a steeper learning curve, with the AI taking a more extended period to refine and optimize its strategies.

Impacted Simulations: For users or testers who are assessing the agent's performance in its early training phases, the AI might appear less proficient. The initial set of simulations, where the agent is still grasping the nuances of the game, could present a skewed perspective of its capabilities.

5.4.2 Feedback-Driven Iterations:

Epsilon Strategy Refinement:

Initial Observations: In the preliminary simulation runs, it was discerned that the agent either explored too much, leading to a delay in strategy convergence, or exploited too soon, leaving it with sub-optimal strategies. This was a direct outcome of the initially adopted epsilon decay mechanism.

Iterative Adjustments: To counterbalance this, the epsilon decay formula underwent iterative refinements. Instead of a simple linear decay, more nuanced decay strategies, perhaps even adaptive ones that changed based on the agent's performance, were incorporated. This ensured that the agent explored efficiently in its early stages but also rapidly honed in on the optimal strategies as it gathered more data.

5.4.3 State Optimization:

Challenge with States: As the agent interacted more with the Blackjack environment, the state space exploded with numerous possibilities. The dictionary-based representation, although efficient in many respects, still faced challenges when the number of unique states grew extensively.

Hashing Enhancements: To address this, a more advanced hashing mechanism was introduced. This mechanism not only ensured unique identification for each state but also optimized the process of

accessing Q-values associated with these states. The new hashing system aimed to reduce lookup times, ensuring that the agent could make decisions promptly, especially in real-time game scenarios.

Compressed Representations: Another avenue explored was the possibility of compressing similar states or ones that led to similar Q-values. By grouping such states or utilizing dimensionality reduction techniques, the agent's state representation became more compact, making it quicker and more memory-efficient.

5.4.4 Conclusion:

The Blackjack AI Trainer represents a confluence of theoretical AI principles and practical game strategy. The evaluation underscores its potential and highlights areas for future development. With iterative improvements and continuous feedback loops, the system is poised for further enhancements, paving the way for mastering the intricacies of Blackjack.

5.5 Learning Points:

5.5.1 Deep Dive into Reinforcement Learning:

Prior to this project, my understanding of reinforcement learning was largely theoretical. The Blackjack AI Trainer necessitated a hands-on approach, acquainting me intimately with the Monte Carlo Control method, policy optimization techniques, and the complexities of balancing exploration and exploitation.

5.5.2 The Essence of Modular Design:

The implementation phase underscored the importance of modular programming. Building separate components for Data Acquisition, Strategy Implementation, Learning, and Evaluation ensured not only clearer code but also easier troubleshooting and future scalability.

5.5.3 Dynamic Data Structures:

Using a dictionary-based approach for Q-values was a revelation. It taught me the significance of using dynamic data structures for efficient data retrieval and management, especially when dealing with a vast state space like in Blackjack.

5.5.4 Balancing Act: Exploration vs. Exploitation:

Crafting the Epsilon-Greedy policy was an exercise in balance. It highlighted the importance of adaptability in AI systems and how initial hyperparameters can deeply influence an AI's learning journey.

5.5.5 Importance of Iterative Testing and Feedback:

Continuous testing, both simulated and real-world, was paramount. The feedback-driven iterations, especially tweaking the epsilon decay mechanism, reinforced the value of feedback loops in refining AI performance.

5.5.6 Addressing Challenges Head-On:

The state space complexity and initial convergence time challenges taught me resilience and the importance of returning to the drawing board. While some issues had solutions within the project's scope, others were earmarked for future research, emphasizing that every project has its constraints.

5.5.7 Documentation Matters:

Maintaining comprehensive documentation was not just a procedural step but a crucial aspect of the project's lifecycle. It allowed for more effortless troubleshooting, a clearer understanding of processes, and has paved the way for any future adaptations.

5.5.8 Reflecting on User Accessibility:

While the command-line interface was functional, it made me realize the importance of user experience in AI solutions. In future iterations, crafting a more intuitive interface would be a priority.

5.5.9 Continuous Learning:

The field of AI is vast, and the project revealed just the tip of the iceberg. The focus on the Monte Carlo method was enlightening, but it also highlighted the potential of other algorithms and strategies yet to be explored.

5.5.10 The Value of Setting Clear Objectives:

Clearly defined objectives at the project's onset served as a guiding light throughout its lifecycle. It facilitated a focused approach and ensured alignment between design, implementation, and evaluation phases.

5.6 Professional Issues

5.6.1 Relation to the British Computer Society (BCS) Code of Conduct:

The Blackjack AI Trainer project was undertaken with a keen awareness of professional standards, particularly those outlined by the British Computer Society (BCS) Code of Conduct. Here's a breakdown of how the project aligned with some key tenets of the code:

5.6.2 Public Interest:

Clause: BCS members should act in the public interest, ensuring that all their work does not harm or diminish the public's health, safety, environment, or quality of life.

Project Alignment: The AI Trainer was developed with the aim of research and learning, ensuring no harm or misrepresentation to users or the public. It focused on academic growth rather than promoting gambling or negative habits.

5.6.3 Professional Competence and Integrity:

Clause: BCS members should only undertake tasks which they are competent to perform and act with integrity in their professional relationships.

Project Alignment: The project was taken on after a thorough review of the required skills and knowledge. Throughout the development, there was a commitment to continuous learning, ensuring the project met high professional standards.

5.7 Duty to Relevant Authority:

Clause: BCS members owe a duty to their employers, clients, and other stakeholders, ensuring the deliverables meet the required expectations and standards.

Project Alignment: The project was transparent in its objectives, ensuring that all stakeholders, including academic supervisors and peers, were well-informed. Feedback was actively sought and incorporated to enhance the project's quality.

5.7.1 Duty to the Profession:

Clause: BCS members should uphold and enhance the honour, reputation, and usefulness of the profession.

Project Alignment: The Blackjack AI Trainer contributed academically to the field of AI and game strategy. Efforts were made to ensure that the project added value to the community and upheld the standards of the computing profession.

5.8 Human Data Considerations:

While the project didn't directly involve human-supplied or human-derived data, it's worth noting the importance of ethically handling such data if it were to be incorporated in future iterations.

Informed Consent: If human players were to play against the AI Trainer to provide feedback, their informed consent would be sought, ensuring they understand the purpose and use of any data generated from their participation.

Anonymity & Privacy: Any data derived from human interaction would be anonymized, ensuring no personal information or identifiers could be traced back to the individual.

Data Security: Measures would be in place to securely store any data, ensuring its protection from unauthorized access, modification, or deletion.

Transparency: Participants would be informed about the data's use, ensuring clarity on how their information contributes to the project's objectives.

Opt-Out Option: Participants will have the right to withdraw their data if they so choose, upholding their rights and ensuring ethical research practices.

5.9 Conclusion

The genesis of the Blackjack AI Trainer was rooted in the ambition to navigate the intricate realm of game strategy through the prism of artificial intelligence. The project was steered with an intent to amalgamate theoretical AI principles and the nuances of Blackjack to chart a learning journey both for the machine and the developer.

The AI Trainer undertook a dynamic path of self-improvement, using the Monte Carlo Control method as its learning paradigm. With the incorporation of the Epsilon-Greedy policy, the AI showcased a balance of exploration and exploitation, navigating through diverse game states and honing its strategies. The modular design fostered scalability, while feedback-driven iterations ensured that the system was ever-evolving, mirroring the dynamism inherent in real-world challenges.

5.10 Main Findings:

Efficacy of Modular Approach: The compartmentalization of functionalities like Data Acquisition, Strategy Implementation, and Learning and Evaluation ensured a streamlined development and potential for future augmentations.

Adaptability: The AI Trainer's ability to recalibrate strategies reflected its robust learning mechanism. The Epsilon-Greedy policy proved instrumental in this adaptability.

Challenges in State Space Representation: Despite the efficient dictionary-based representation, the vast state space in Blackjack was a complex challenge, indicating the intricate nature of game strategies.

Feedback is Invaluable: Real-time feedback from simulations provided critical insights, shaping the evolution of the AI Trainer. It underscored the importance of iterative development in AI projects.

5.10.1 Directions for Further Work:

Algorithmic Diversification: While the Monte Carlo Control method provided significant insights, exploring other reinforcement learning algorithms like Temporal-Difference Learning could enrich the project's learning paradigm.

User Interface Enhancements: Bridging the gap between a command-line interface and a more user-friendly GUI could increase accessibility and user engagement.

Real-time Player Interactions: Introducing a feature where human players can engage with the AI Trainer in real-time could offer invaluable feedback and lead to richer learning experiences for the AI.

Chapter 6

Bibliography:

1. Bishop, C.M. (2006). *Pattern Recognition and Machine Learning*. New York: Springer.

2. Sutton, R.S., & Barto, A.G. (2018). *Reinforcement Learning: An Introduction*. MIT press.

3. Chollet, F. (2017). *Deep Learning with Python*. Manning Publications Co.

4. OpenAI. (2020). *Gym: Toolkit for developing and comparing reinforcement learning algorithms*. Retrieved from https://github.com/openai/gym

5. Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). OpenAI gym. *arXiv preprint arXiv:1606.01540*.

6. Watkins, C.J.C.H. (1989). *Learning from Delayed Rewards* (PhD thesis). King's College, Cambridge.

7. Thorp, E. (1966). *Beat the Dealer*. New York: Vintage.

8. British Computer Society (BCS). (2018). *Code of Conduct*. Retrieved from https://www.bcs.org/membership/become-a-member/code-of-conduct/

9. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

10. University Library. (2022). *Citation and Referencing Guide*. Retrieved from https://www.universitylibrary.edu/citation-guide

Appendices

Appendix A: Test Data Details

The foundation for our AI trainer was laid using the test data from OpenAI's `gym` library, designed specifically to simulate environments for reinforcement learning. In the context of this project, the library was utilized to simulate an extensive set of Blackjack games, ensuring that the AI agent encountered a comprehensive range of game scenarios, both typical and edge cases.

Each game simulation returned specific data points:

The player's current hand.

The visible card from the dealer's hand.

Whether the player has a usable ace or not.

These data points were then used by the AI agent to make decisions and, over time, refine its strategy.

Appendix B: Software Installation and Usage

Environment Setup:

This project was built and tested on a system running Python 3.6. The AI trainer relies on a few external libraries to function optimally:

OpenAI's `gym` (version 0.18.3): Crucial for simulating the Blackjack environment.

`numpy` (version 1.19.5): For numerical operations, particularly matrix manipulations.

`pandas` (version 1.2.4): Facilitates data handling and storage operations.

Installation Guide:

Before proceeding, ensure that your system has Python 3.6 installed.

Use pip, the package installer for Python, to install the requisite libraries:

pip install gym[all] numpy pandas

Usage Instructions:

The solution consists of two primary scripts:

Trainer: The script responsible for training the AI agent. Once executed, the agent will play several games, iteratively improving its strategy. To activate the training process:

python trainer.py

Evaluator: Post-training, use this script to evaluate the performance of the AI agent against the set metrics:

python evaluate.py

It's recommended to run the trainer before the evaluator to ensure the AI agent has an optimized strategy.

Appendix C:

For an appendix, it's often beneficial to highlight code sections that showcase the main logic or novel implementations.

Epsilon-Greedy Policy Creation:

The function `create_epsilon_greedy_action_policy` is noteworthy because it demonstrates how actions are chosen based on the Q-values with a degree of exploration:

```python
def create_epsilon_greedy_action_policy(env, Q, epsilon):
    def policy(obs):
        P = np.ones(env.action_space.n, dtype=float) * epsilon / env.action_space.n
        best_action = np.argmax(Q[obs])
        P[best_action] += (1.0 - epsilon)
        return P
    return policy
```

First-Visit Monte Carlo Control Logic:

This is the main learning algorithm. The following snippet captures how the returns for state-action pairs are computed and how Q-values are updated:

```python
sa_in_episode = set([(tuple(x[0]), x[1]) for x in episode])
for state, action in sa_in_episode:
    sa_pair = (state, action)
    first_occurence_idx = next(i for i, x in enumerate(episode) if x[0] == state and x[1] == action)
    G = sum([x[2] * (discount_factor**i) for i, x in enumerate(episode[first_occurence_idx:])])
    returns_sum[sa_pair] += G
    returns_count[sa_pair] += 1.0
    Q[state][action] = returns_sum[sa_pair] / returns_count[sa_pair]
```

Simulating Games Between Two Agents:

The method `simulate_games` is an innovative approach to have two agents interact with the environment. This is different from the typical setting where only one agent interacts with its environment:

```python
while not done:
    if np.random.rand() < 0.5:
        action = np.argmax(agent1(state))
    else:
        action = np.argmax(agent2(state))
    next_state, reward, done, _ = env.step(action)
    state = next_state
```

Visualization:

The code's visualization section is straightforward but crucial in presenting results in a clear, easily digestible manner:

```python
labels = ['Agent 1', 'Agent 2', 'Draw']
rates = [agent1_win_rate*100, agent2_win_rate*100, draw_rate*100]
plt.bar(labels, rates, color=['blue', 'red', 'green'])
```