Project 2 Report

Overview

This report details the implementation of a simple blackjack game environment as part of a reinforcement learning project. The environment is designed to mimic the traditional blackjack card game with simplified rules.

Blackjack Game Environment

1. *Deck*: The game uses a standard deck of 52 cards. Numerical cards count as their face value, J, Q, and K are considered 10, and Aces can be either 1 or 11.
2. *Gameplay*: Players compete against a dealer. Players can "twist" to draw a card or "stick" to hold their current total. The goal is to achieve a card total as close to 21 as possible without exceeding it.
3. *Dealer Rules*: The dealer draws cards until reaching a total greater than 17.
4. *Ace Handling*: Aces are flexible, counted as 11 unless such a value would cause the player's score to exceed 21, in which case they are counted as 1.

Differences from Gymnasium's Blackjack

1. *Total*: If the total exceeds 21, it is automatically set to 22 to simplify handling of player busts.
2. *State* Representation: Each state in the game is represented as a tuple containing the player's total, the dealer's showing card, and a boolean indicating whether the player holds an ace card.

Methods Implemented

To address the problem posed by the blackjack environment, four methods were implemented: Monte Carlo, Sarsa, Q-Learning, and Deep Q-Learning. Each method employs an epsilon-greedy policy for action selection with an initial default value set to zero.

Monte Carlo

The Monte Carlo method uses a first-visit approach to estimate the value functions through repeated simulated episodes of the game.

Sarsa

An on-policy TD control algorithm.

Update Formula: $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]$ where s' is the next state, a' is the next action, r is the reward, α is the learning rate, and γ is the discount factor.

Q-Learning

An off-policy TD control algorithm. Update Formula: $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$ where s' is the next state, a' is the next action, r is the reward, α is the learning rate, and γ is the discount factor.

Deep Q-Learning

Replaces the Q-value table with an artificial neural network that approximates the Q-values for each state-action pair. The network is capable of generalizing from observed states to unseen states, providing a robust method for handling diverse game situations.

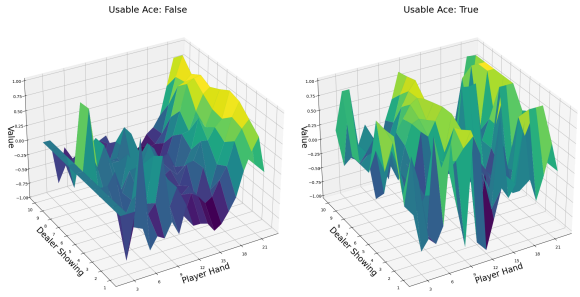
Performance Comparison and Visual Analysis

A comprehensive comparison was conducted after training each method for varying numbers of episodes. The aim was to evaluate the efficiency and effectiveness of each algorithm in learning optimal policies for blackjack.

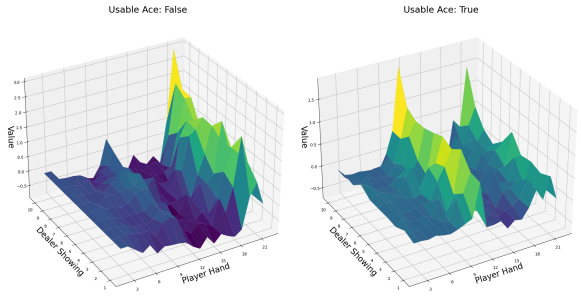
Result Overview

The performance of each method was quantified by the average rewards accumulated over a set number of episodes. This metric reflects the ability of each method to maximize returns based on the learned policies.

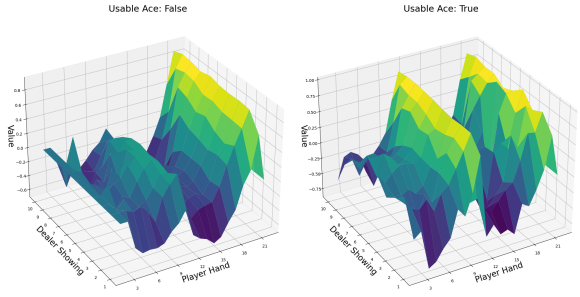
Figures



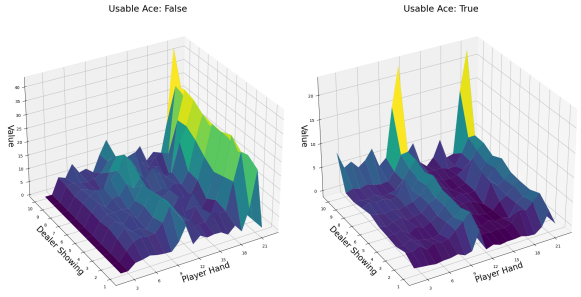
(a) Monte Carlo with 10000 Episodes



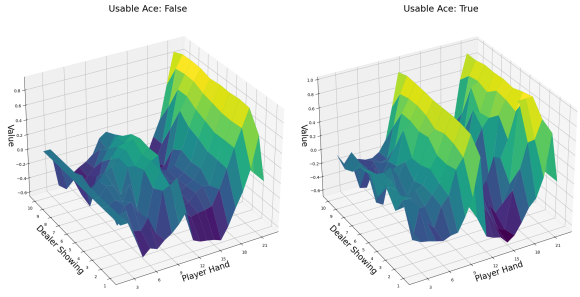
(b) Sarsa with 10000 Episodes



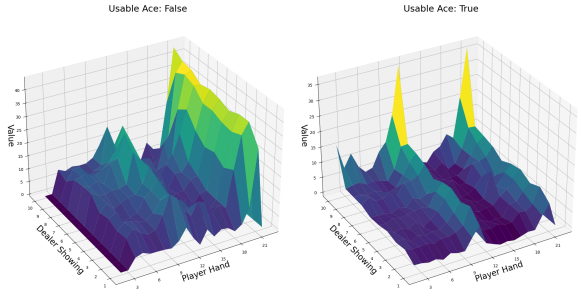
(c) Monte Carlo with 100000 Episodes



(d) Sarsa with 100000 Episodes

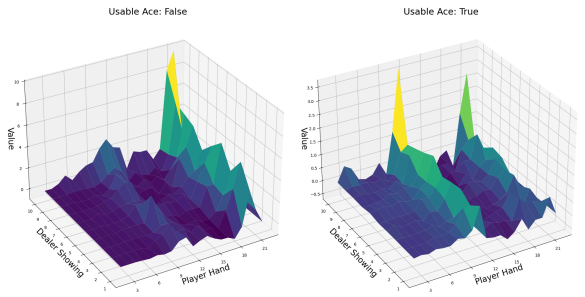


(e) Monte Carlo with 200000 Episodes

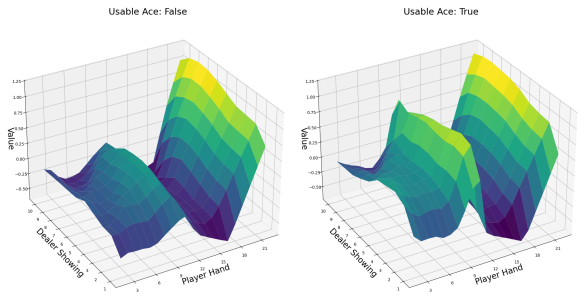


(f) Sarsa with 200000 Episodes

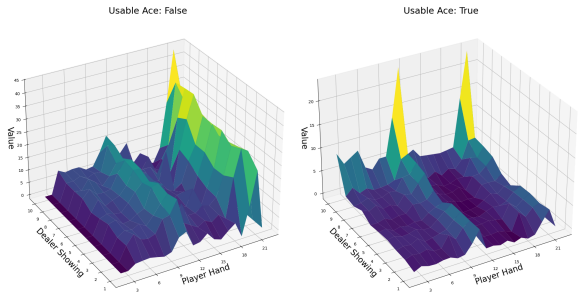
Figure 1: Value Functions of Monte Carlo and Sarsa



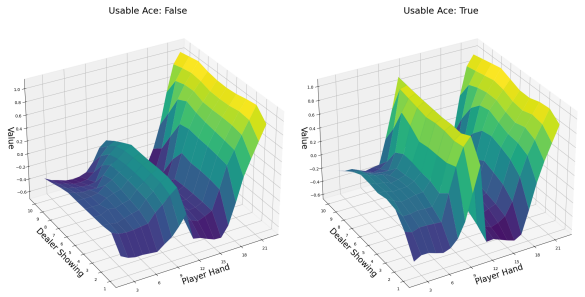
(a) Q-Learning with 10000 Episodes



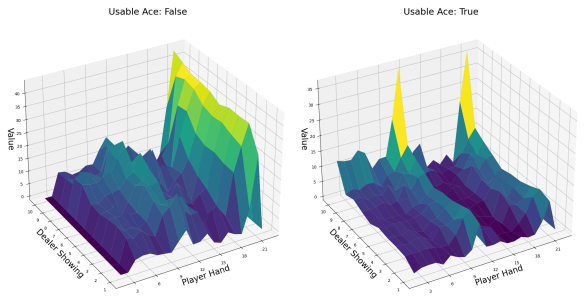
(b) Deep Q-Learning with 10000 Episodes



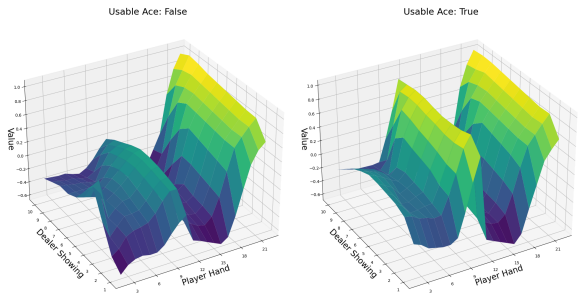
(c) Q-Learning with 100000 Episodes



(d) Deep Q-Learning with 100000 Episodes



(e) Q-Learning with 200000 Episodes



(f) Deep Q-Learning with 200000 Episodes

Figure 2: Value Functions of Q-Learning and Deep Q-Learning

Visual Analysis

The value functions for different methods were visualized, demonstrating how each method approaches learning the optimal strategy for blackjack. Particularly, the Deep Q-Learning method exhibited smoother generalization across different states compared to others.

Performance Table

A detailed performance comparison is presented for 50,000 episodes across different training configurations. This analysis highlights the learning progression and stability of each method over extended periods of training

Training Episodes	Monte Carlo	Sarsa	Q-Learning	Deep Q-Learning
10000	−0.08376	−0.10722	−0.09782	−0.05004
50000	−0.07138	−0.15614	−0.16946	−0.04928
100000	−0.06492	−0.17498	−0.17292	−0.05002
200000	−0.06220	−0.17822	−0.15516	−0.04538

Conclusion

All implemented methods were capable of effectively learning strategies for the simplified blackjack environment. Deep Q-Learning showed particular promise due to its ability to generalize well from the training data, offering smoother value function approximations and potentially more robust strategies in variable game scenarios. The comparative analysis underscores the importance of choosing appropriate learning algorithms based on the specifics of the task and the desired characteristics of the solution, such as learning speed and policy smoothness.

How to Execute the Code

Prerequisites

The code is implemented using Python and depends on several libraries including NumPy (version 1.26.4), Matplotlib (version 0.3.90), Gymnasium (version 0.29.1), and PyTorch (version 2.3.1). It is essential to have these libraries installed prior to running the code.

Execution Instructions

To execute the program:

- Run `python main.py` or `python main.py train` to train the models and save them. This process will also generate visualizations and save them in the `pics` directory.
- Run `python main.py` to load existing models.

Both training and loading configurations are designed to automatically generate visual outputs and save them in the designated pics folder.