Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Computer Science and Engineering (Data Science)

Subject: Reinforcement Learning

AY: 2022 - 23

Experiment 6

Aim: To solve the Blackjack Game using Monte Carlo methods.

Theory:

Monte Carlo method is used for estimating value functions and discovering optimal policies. Unlike the other methods, we do not assume complete knowledge of the environment. Monte Carlo methods require only experience—sample sequences of states, actions, and rewards from actual or simulated interaction with an environment.

The term “Monte Carlo” is often used more broadly for any estimation method whose operation involves a significant random component. Monte Carlo methods are ways of solving the reinforcement learning problem based on averaging sample returns. To ensure that well-defined returns, Monte Carlo methods are defined only for episodic tasks. That is, we assume experience is divided into episodes, and that all episodes eventually terminate no matter what actions are selected. Only on the completion of an episode are value estimates and policies changed. Monte Carlo methods are thus incremental in an episode-by-episode sense.

Blackjack:

The object of the popular casino card game of Blackjack is to obtain cards the sum of whose numerical values is as great as possible without exceeding 21. All face cards count as 10, and an ace can count either as 1 or as 11.

We consider the version in which each player competes independently against the dealer. The game begins with two cards dealt to both dealer and player. One of the dealer’s cards is face up and the other is face down. If the player has 21 immediately (an ace and a 10-card), it is called a natural. He then wins unless the dealer also has a natural, in which case the game is a draw. If the player does not have a natural, then he can request additional cards, one by one (hits), until he either stops (sticks) or exceeds 21 (goes bust). If he goes bust, he loses; if he sticks, then it becomes the dealer’s turn.

The dealer hits or sticks according to a fixed strategy without choice: he sticks on any sum of 17 or greater, and hits otherwise. If the dealer goes bust, then the player wins; otherwise, the outcome—win, lose, or draw—is determined by whose final sum is closer to 21.

Points to remember for solving the game using Monte Carlo:

- Each game of blackjack is an episode.
- Rewards of +1, −1, and 0 are given for winning, losing, and drawing, respectively.
- All rewards within a game are zero, and we do not discount (γ = 1).
- The player's actions are to hit or to stick.
- The states depend on the player's cards and the dealer's showing card.
- We assume that cards are dealt from an infinite deck (i.e., with replacement).
- If the player holds an ace that he could count as 11 without going bust, then the ace is said to be usable. In this case it is always counted as 11 because counting it as 1 would make the sum 11 or less, in which case there is no decision to be made because, obviously, the player should always hit.
- Thus, the player makes decisions on the basis of three variables: his current sum (12–21), the dealer's one showing card (ace–10), and whether or not he holds a usable ace. This makes for a total of 200 states.

Algorithm:

Initialize:
    $\pi \leftarrow$ policy to be evaluated
    $V \leftarrow$ an arbitrary state-value function
    $Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Repeat forever:
    Generate an episode using π
    For each state s appearing in the episode:
        $G \leftarrow$ return following the first occurrence of s
        Append G to $Returns(s)$
        $V(s) \leftarrow \text{average}(Returns(s))$

Lab Assignment to do:

1. Develop a Blackjack Environment and its variables.
2. Implement the monte Carlo methods for 10000 and 500000 episodes for usable and no usable ace cards.
3. Show the results graphically (3D graphs) for all four combinations. (10000 episodes with and without usable ace and 500000 episodes with and without usable ace)

NAME: Sarvagya Singh

SAP ID: 60009200030

BATCH: K - K1

SOLVING BLACKJACK PROBLEM USING FIRST-VISIT MONTE CARLO

NECESSARY LIBRARIES

In [ ]:

```
import random
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = (15, 7)
```

In [ ]:

```
import itertools
from typing import *
```

In [ ]:

```
def toHit(state):
    total, *_ = state
    if total < 20:
        return True
    return False
```

In [ ]:

```
class BlackJack:
    def __init__(self, defaultPolicy):
        self.pi = defaultPolicy

        self.current_sum = range(12, 22)
        self.showing_card = range(1, 11)
        self.usable_ace = [True, False]
        self.values = dict()
        self.states = list(itertools.product(self.current_sum, self.showing_card, self.u
sable_ace))


    def initialise(self, pair):
        current_sum = pair.sum()
        usable_ace = False

        if (pair == 1).any():
            current_sum += 10
            usable_ace = True

        while current_sum < 11:
            deal = np.random.choice(self.cards)
            current_sum += deal
            # WITH USABLE ACE, SCORE += 11
            # WITHOUT, SCORE += 1
            if deal == 1:
```

```
                current_sum += 10
                usable_ace = True

        if current_sum == 11:
            current_sum += np.random.choice(self.cards)

        return current_sum, usable_ace

    def hit(self, current_sum, usable_ace):
        current_sum += np.random.choice(self.cards)

        if current_sum > 21 and usable_ace:
            current_sum -= 10
            usable_ace = False

        return current_sum, usable_ace

    def play(self):
        # STATES -> (P_TOTAL, DEAL_UP, USABLE_ACE)
        states = []
        actions = []
        rewards = []

        self.cards = np.minimum(np.arange(1, 14), 10)
        '''
        INITIAL CONDITION:
        - PLAYER -> 2 CARDS (BOTH UP)
        - DEALER -> 2 CARDS (1 UP AND OTHER DOWN)
        '''
        self.D_up, self.D_down = self.cards_D = np.random.choice(self.cards, size=2)
        self.cards_P = np.random.choice(self.cards, size=2)

        if set(self.cards_P) == {1, 10}:
            states.append((21, self.D_up, True)) # NATURAL WIN
            actions.append('stick')
            if set(self.cards_D) == {1, 10}:
                rewards.append(0)
            else:
                rewards.append(1)

            return states, actions, rewards

        current_sum_P, usable_ace_P = self.initialise(self.cards_P)
        states.append((current_sum_P, self.D_up, usable_ace_P))

        while True:
            current_sum_P, _, usable_ace_P = state = states[-1]

            if self.pi(state):
                actions.append('HIT')
                current_sum_P, usable_ace_P = self.hit(current_sum_P, usable_ace_P)

                if current_sum_P > 21:
                    rewards.append(-1)
                    break
                # ITS PLAYER'S TURN AGAIN
                else:
                    rewards.append(0)
                    states.append((current_sum_P, self.D_up, usable_ace_P))

            else:
                actions.append('STICK')
                # NOW, IT WILL BE DEALER'S TURN
                current_sum_D, usable_ace_D = self.initialise(self.cards_D)

                while current_sum_D < 17:
                    current_sum_D, usable_ace_D = self.hit(current_sum_D, usable_ace_D)

                if current_sum_D > 21:
                    rewards.append(1)

                elif current_sum_D > current_sum_P:
```

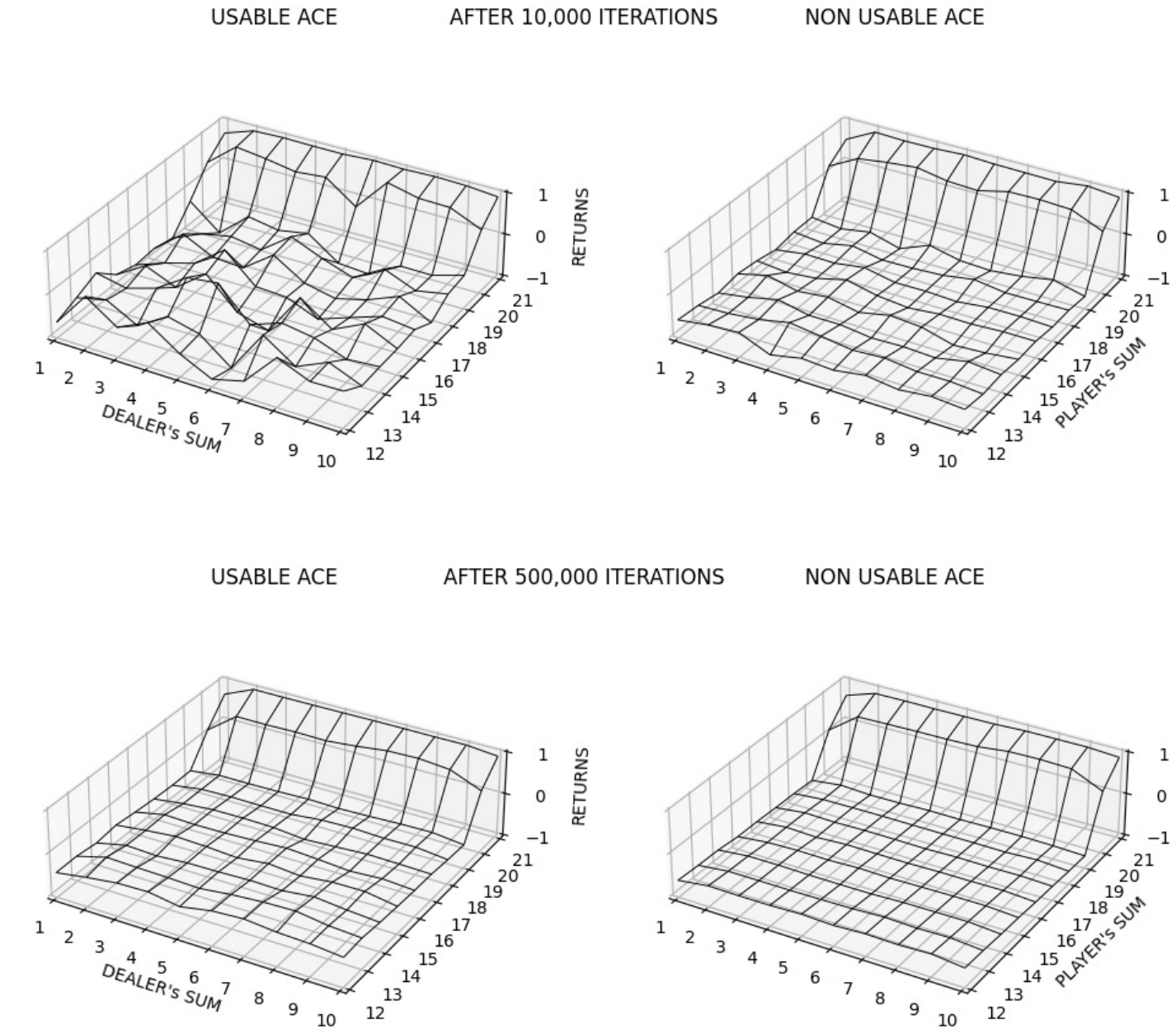
```
                rewards.append(-1)

            elif current_sum_D == current_sum_P:
                rewards.append(0)

            else:
                assert current_sum_D < current_sum_P
                rewards.append(1)

            break

    return states, actions, rewards

def prediction(self, gamma, num_episodes:List[int]):
    for num_episode in num_episodes:
        print(f'NUMBER OF EPISODES: {num_episode},000')

        num_states = len(self.states)
        V = dict(zip(self.states, np.random.normal(size=num_states)))
        returns = {state: [] for state in self.states}
        # print(returns)

        for episode in range(num_episode * 1000):
            episode = self.play()
            S_0Tm1, A_0Tm1, R_1T = episode
            G = 0
            for t, (St, At, Rtp1) in list(enumerate(zip(*episode)))[::-1]:
                G = gamma * G + Rtp1
                # G = SUM(gamma**t * Rt)
                if St not in S_0Tm1[:t]:
                    returns[St].append(G)
                    V[St] = np.mean(returns[St])

        self.values[num_episode] = V

def plot(self):
    for k in self.values.keys():
        fig = plt.figure(figsize=(12, 5))
        plt.title(f'AFTER {k},000 ITERATIONS')
        plt.axis("off")
        val = self.values[k]
        for i, ace in zip(range(1, 3), [True, False]):
            ax = fig.add_subplot(1, 2, i, projection='3d')
            if ace:
                ax.set_title('USABLE ACE')
            else:
                ax.set_title('NON USABLE ACE')

            X, Y = np.meshgrid(self.showing_card, self.current_sum)
            Z = np.zeros_like(X).astype('float')

            for (cs, sc, ua) in val:
                if ua == ace:
                    # TO RESET THE INDEX FOR Z
                    Z[cs-12, sc-1] = val[cs, sc, ua]

            ax.plot_wireframe(X, Y, Z, linewidth=0.7, color='k')

            if i == 1:
                ax.set_xlabel('DEALER\'s SUM')
            else:
                ax.set_ylabel('PLAYER\'s SUM')

            ax.set_zlabel('RETURNS')

            ax.set_xlim(1, 10)
            ax.set_ylim(12, 21)
            ax.set_zlim(-1, 1)

            ax.set_zticks([-1, 0, 1])
            ax.set_box_aspect([1, 1, 0.3])
```

```
            plt.show()
            print()
```

In [ ]:

```
BJ = BlackJack(toHit)
gamma = 1
num_episodes = [10, 500]
BJ.prediction(gamma, num_episodes)
BJ.plot()
```

```
NUMBER OF EPISODES: 10,000
NUMBER OF EPISODES: 500,000
```



In [ ]:

```

```

In [ ]:

```
!jupyter nbconvert --to html '/content/ML_III_PRACT6_K1_60009200032.ipynb'
```

```
[NbConvertApp] Converting notebook /content/ML_III_PRACT6_K1_60009200032.ipynb to html
[NbConvertApp] Writing 1078073 bytes to /content/ML_III_PRACT6_K1_60009200032.html
```