Batch: C1	**Roll No.: 16010121069**
Experiment / assignment / tutorial No.___3___	
Grade: AA / AB / BB / BC / CC / CD /DD	
Signature of the Staff In-charge with date	

TITLE : 3. To implement Monte Carlo approach

AIM : MC (model-free)

Expected OUTCOME of Experiment: (Mention CO /CO's attained here): CO2

Books/ Journals/ Websites referred:

Richard S. Sutton and Andrew G. Barto, "*Reinforcement Learning:An Introduction",* The MIT Press,Second Edition, 2018

Pre Lab/ Prior Concepts:

Monte Carlo methods learn directly from episodes of experience. MC is model-free. No knowledge of MDP transitions / rewards is needed. MC learns from complete episodes: no bootstrapping is needed. MC uses the simplest possible idea: value = mean return. Hence we can only apply MC to episodic MDPs.

First visit MC evaluation and every visit MC evaluation should be studied before implementation.

Chosen Problem Statement: Black Jack

Policy:

The policy is a strategy that the agent (the player) follows to decide its actions at each state. In Blackjack, the policy defines whether the player should "stick" (stop drawing cards) or "twist" (draw another card) depending on the current state.

For example, a simple policy could be:

- If the player's current sum of cards is less than 17, they should "twist."
- If the sum is 17 or greater, they should "stick."

Reward function:

The reward function defines the reward the agent receives for taking an action in a given state. In Blackjack, the reward is based on the outcome of the game:

- Stick: The player receives:
  - +1 if their hand's total is greater than the dealer's hand and ≤ 21.
  - 0 if their hand equals the dealer's hand.
  - -1 if their hand is less than the dealer's hand.
- Twist:
  - The player receives -1 if their hand exceeds 21 (busts).
  - Otherwise, there is no immediate reward (reward = 0), and the game continues.

Value function:

The value function estimates the expected total reward an agent can receive from a given state or state-action pair, following a particular policy. In Blackjack, the value function reflects the likelihood of winning based on the current sum of cards, the dealer's visible card, and whether the player has a usable ace.

There are two types of value functions:

- State Value Function (V(s)): This is the expected reward starting from state s, following the current policy.
- Action Value Function (Q(s, a)): This is the expected reward from taking action a in state s, and then following the policy thereafter.

Environment:

The environment represents the external system with which the agent interacts. In the Blackjack problem, the environment includes:

- The player's hand (sum of the cards and whether they have a usable ace).
- The dealer's visible card.
- The deck of cards (from which new cards are drawn).
- The rules of the game, which dictate when the dealer must draw cards or stop (e.g., the dealer always sticks on 17 or higher).

The environment takes in the agent's actions and transitions to a new state, providing rewards based on the outcome (win, lose, or draw).

Implementation:

```
import gym
import numpy as np
from collections import defaultdict

# Create Blackjack environment
env = gym.make('Blackjack-v1')

def create_random_policy(nA):
    A = np.ones(nA, dtype=float) / nA
    def policy_fn(observation):
        return A
    return policy_fn

def create_greedy_policy(Q):
    def policy_fn(observation):
        A = np.zeros(len(Q[observation]), dtype=float)
        best_action = np.argmax(Q[observation])
        A[best_action] = 1.0
        return A
```

```
    return policy_fn

def mc_control(env, num_episodes, discount_factor=1.0, epsilon=0.1):
    returns_sum = defaultdict(float)
    returns_count = defaultdict(float)
    Q = defaultdict(lambda: np.zeros(env.action_space.n))

    policy_dict = defaultdict(int)

    for i_episode in range(1, num_episodes + 1):
        episode = []
        state = env.reset()
        for t in range(100):
            probs = create_greedy_policy(Q)(state)
            action = np.random.choice(np.arange(len(probs)), p=probs)
            next_state, reward, done, _ = env.step(action)
            episode.append((state, action, reward))
            if done:
                break
            state = next_state

        sa_in_episode = set([(x[0], x[1]) for x in episode])

        for state, action in sa_in_episode:
            first_occurrence_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_occurrence_idx:])])
            returns_sum[(state, action)] += G
            returns_count[(state, action)] += 1.0
            Q[state][action] = returns_sum[(state, action)] /
returns_count[(state, action)]

        greedy_policy = create_greedy_policy(Q)
        for state in Q.keys():
            policy_dict[state] = np.argmax(Q[state])

    return Q, policy_dict

def run_multiple_iterations(env, num_episodes, num_iterations,
discount_factor=1.0, epsilon=0.1):
    aggregated_Q = defaultdict(lambda: np.zeros(env.action_space.n))
    total_policies = defaultdict(int)

    for i in range(num_iterations):
        Q, policy_dict = mc_control(env, num_episodes, discount_factor,
epsilon)

        for state, actions in Q.items():
```

```
            aggregated_Q[state] += actions

        for state, action in policy_dict.items():
            total_policies[(state, action)] += 1

    for state in aggregated_Q:
        aggregated_Q[state] /= num_iterations

    final_policy_dict = {state: np.argmax(actions) for state, actions in
aggregated_Q.items()}

    return aggregated_Q, final_policy_dict, total_policies

# Parameters
num_episodes = 100
num_iterations = 10

# Run multiple iterations
aggregated_Q, final_policy_dict, total_policies = run_multiple_iterations(env,
num_episodes, num_iterations)

# Display the final learned policy
for state, action in sorted(final_policy_dict.items()):
    print(f"State: {state}, Best Action: {'Stick' if action == 0 else 'Hit'}")
```

```
Policy Table (No Ace):
    1  2  3  4  5  6  7  8  9 10
  ┌──────────────────────────────┐
21 | S  S  S  S  S  S  S  S  S  S | 21
20 | S  S  S  S  S  S  S  S  S  S | 20
19 | S  S  S  S  S  S  S  S  S  S | 19
18 | S  S  S  S  S  S  S  S  S  S | 18
17 | H  S  S  H  H  S  H  S  S  H | 17
16 | H  H  H  H  H  H  H  H  H  H | 16
15 | H  H  H  H  H  H  H  H  H  H | 15
14 | H  H  H  H  H  H  H  S  H  S | 14
13 | H  H  H  H  H  H  H  H  H  H | 13
12 | H  H  H  H  H  H  H  H  H  H | 12
11 | H  H  H  H  H  H  H  H  H  H | 11
10 | H  H  H  H  H  H  H  H  H  H | 10
 9 | H  H  H  H  H  H  H  H  H  H |  9
 8 | H  H  H  H  H  H  H  H  H  H |  8
 7 | H  H  H  H  H  H  H  H  H  H |  7
 6 | H  H  H  H  H  H  H  H  H  H |  6
 5 | H  H  H  H  H  H  H  H  H  H |  5
 4 | H  H  H  H  H  H  H  H  H  H |  4
 3 | S  S  S  S  S  S  S  S  S  S |  3
 2 | S  S  S  S  S  S  S  S  S  S |  2
  └──────────────────────────────┘
    1  2  3  4  5  6  7  8  9 10
```

```
Policy Table (With Ace):
    1  2  3  4  5  6  7  8  9 10
  ┌──────────────────────────────┐
21 | S  S  S  S  S  S  S  S  S  S | 21
20 | S  S  S  S  S  S  S  S  S  S | 20
19 | H  S  S  S  S  H  H  S  S  S | 19
18 | H  H  S  S  H  H  H  S  H  H | 18
17 | H  H  H  H  H  H  H  H  H  H | 17
16 | H  H  H  H  H  S  H  H  H  H | 16
15 | H  H  H  H  H  H  H  H  H  H | 15
14 | H  H  H  H  H  H  H  H  H  H | 14
13 | H  H  H  H  H  H  H  H  H  H | 13
12 | H  H  H  H  H  H  H  H  H  H | 12
11 | S  S  S  S  S  S  S  S  S  S | 11
10 | S  S  S  S  S  S  S  S  S  S | 10
 9 | S  S  S  S  S  S  S  S  S  S |  9
 8 | S  S  S  S  S  S  S  S  S  S |  8
 7 | S  S  S  S  S  S  S  S  S  S |  7
 6 | S  S  S  S  S  S  S  S  S  S |  6
 5 | S  S  S  S  S  S  S  S  S  S |  5
 4 | S  S  S  S  S  S  S  S  S  S |  4
 3 | S  S  S  S  S  S  S  S  S  S |  3
 2 | S  S  S  S  S  S  S  S  S  S |  2
  └──────────────────────────────┘
    1  2  3  4  5  6  7  8  9 10
```

Conclusion:

Successfully implemented the blackjack game using monte carlo learning.

Post Lab Descriptive Questions:

What do you understand by incremental Monte Carlo updates for policy evaluation?

Ans.
Incremental Monte Carlo updates for policy evaluation refer to updating the value function after each episode incrementally, without needing to store all previous episodes. In this approach, the value of each state is updated using a running average of the observed returns. After each episode, the value estimate for a state is adjusted by incorporating the new reward obtained from that episode, gradually improving the estimate. This method allows efficient policy evaluation by updating values on-the-fly.

Date: _____ **Signature of faculty in-charge**