MONTE CARLO CONTROL ALGORITHM

Name: Meetha Prabhu

Register Number: 212222240065

AIM

To implement Monte Carlo prediction to evaluate an optimal policy in a grid-based environment using Gym's SlipperyWalkFive-v0.

PROBLEM STATEMENT

The task involves evaluating the effectiveness of a policy in a grid-based environment using Monte Carlo methods. The environment consists of states and actions, where the goal is to navigate the agent to a terminal state while maximizing rewards. The system needs to determine the action-value and state-value functions for the policy and analyze the policy's performance in terms of success probability and average return.

MONTE CARLO CONTROL ALGORITHM

Step 1:

Initialize Parameters Set up the environment, policy, and initialize the action-value (Q) and state-value (V) functions.

Step 2:

Generate Episodes: Simulate episodes by starting at random states and following the policy until reaching a terminal state.

Step 3:

Compute Returns: Calculate cumulative returns for each state-action pair from the rewards in the episode.

Step 4:

Update Action-Value Function: Average the returns for each state-action pair over

Update Action-Value Function: Average the returns for each state-action pair over multiple episodes to update the action-value function.

Step 5:

Estimate State-Value Function: Derive the state-value function from the action-value function by selecting the best action for each state.

Step 6:

Policy Evaluation: Calculate the success rate and mean return of the policy based on the computed values.

Step 7:

Output: Display the action-value, state-value functions, and the performance metrics for the policy.

MONTE CARLO CONTROL FUNCTION

Import the necessary packages:

```
import warnings ; warnings.filterwarnings('ignore')

import gym, gym_walk
import numpy as np

import random
import warnings

warnings.filterwarnings('ignore', category=DeprecationWarning)
np.set_printoptions(suppress=True)
random.seed(123); np.random.seed(123)
```

Define the printing functions:

```
    print(title)
    arrs = {k:v for k,v in enumerate(action_symbols)}
    for s in range(len(P)):
        a = pi(s)
        print("| ", end="")
        if np.all([done for action in P[s].values() for _, _, _, done in
action]):
            print("".rjust(9), end=" ")
        else:
            print(str(s).zfill(2), arrs[a].rjust(6), end=" ")
```

```
        if (s + 1) % n_cols == 0: print("|")


def print_state_value_function(V, P, n_cols=4, prec=3, title='State-value
function:'):
    print(title)
    for s in range(len(P)):
        v = V[s]
        print("| ", end="")
        if np.all([done for action in P[s].values() for _, _, _, done in
action]):
            print("".rjust(9), end=" ")
        else:
            print(str(s).zfill(2), '{}'.format(np.round(v, prec)).rjust(6),
end=" ")
        if (s + 1) % n_cols == 0: print("|")

def probability_success(env, pi, goal_state, n_episodes=100, max_steps=200):
    random.seed(123); np.random.seed(123) ; env.seed(123)
    results = []
    for _ in range(n_episodes):
        state, done, steps = env.reset(), False, 0
        while not done and steps < max_steps:
            state, _, done, h = env.step(pi(state))
            steps += 1
        results.append(state == goal_state)
    return np.sum(results)/len(results)
def mean_return(env, pi, n_episodes=100, max_steps=200):
    random.seed(123); np.random.seed(123) ; env.seed(123)
    results = []
    for _ in range(n_episodes):
        state, done, steps = env.reset(), False, 0
        results.append(0.0)
        while not done and steps < max_steps:
            state, reward, done, _ = env.step(pi(state))
            results[-1] += reward
            steps += 1
    return np.mean(results)
```

Define the environment

```
env = gym.make('FrozenLake-v1')
P = env.env.P
init_state = env.reset()
goal_state = 15
LEFT, RIGHT = range(2)
P
```

Decay Schedule Function:

```
import numpy as np
def decay_schedule(init_value, min_value,decay_ratio, max_steps,
log_start=-2, log_base=10):
  decay_steps = int(max_steps * decay_ratio)
  rem_steps = max_steps - decay_steps
#This function allows you to calculate all the values for alpha for the full
training process.
#(2) First, calculate the number of steps to decay the values using the
decay_ratio variable. (3) Then, calculate the actual values as an inverse
log curve. Notice we then normalize between 0 and 1, and finally transform
the points to lay between init_value and min_value.
  values = np.logspace (log_start, 0, decay_steps,base=log_base,
endpoint=True) [::-1]
  values =(values - values.min()) / (values.max()-values.min())
  values = (init_value - min_value) * values + min_value
  values = np.pad(values, (0, rem_steps), 'edge')
  return values
```

Generate Trajectory:

```
def generate_trajectory(select_action, Q, epsilon,env, max_steps=200): #
Corrected order of arguments
  done, trajectory = False, []
  while not done:
    state = env.reset()
    for t in count():
      action = select_action(state, Q, epsilon)
      next_state, reward, done, _ = env.step(action)
      experience = (state, action, reward, next_state, done)
      trajectory.append(experience)
      if done:
        break
      if t >= max_steps - 1:
        trajectory = []
        break
      state = next_state
  return np.array(trajectory, dtype=object)
```

Monte Carlo Control Function:

```
import numpy as np
from tqdm import tqdm

def mc_control (env, gamma = 1.0,
                init_alpha = 0.5,min_alpha = 0.01, alpha_decay_ratio = 0.5,
                init_epsilon = 1.0, min_epsilon = 0.1, epsilon_decay_ratio =
```

```
                init_epsilon = 1.0, min_epsilon = 0.1, epsilon_decay_ratio =
0.9,
                n_episodes = 3000, max_steps = 200, first_visit = True):
  nS, nA = env.observation_space.n, env.action_space.n

  #Write your code here
  discounts=np.logspace(0,max_steps,num=max_steps, base=gamma,
endpoint=False)
  alphas = decay_schedule(init_alpha, min_alpha,
alpha_decay_ratio,n_episodes)
  epsilons=decay_schedule(init_epsilon, min_epsilon,
epsilon_decay_ratio,n_episodes)
  pi_track=[]
  Q = np.zeros((nS, nA),dtype=np.float64)
  Q_track = np.zeros((n_episodes,nS,nA),dtype=np.float64 )
  select_action = lambda state, Q, epsilon : np.argmax(Q[state]) if
np.random.random()> epsilon else np.random.randint(len(Q[state]))

  for e in tqdm(range(n_episodes),leave=False):
    trajectory = generate_trajectory(select_action,Q, epsilons[e],env,
max_steps)
    visited = np.zeros((nS, nA), dtype=bool)
    for t, (state, action, reward,_,_) in enumerate(trajectory):
      if visited[state][action] and first_visit:
        continue
      visited[state][action]=True
      n_steps=len(trajectory[t:])
      G=np.sum(discounts[:n_steps] * trajectory[t:,2])
      Q[state][action] = Q[state][action] + alphas[e] * (G-Q[state][action])
    Q_track[e]=Q
    pi_track.append(np.argmax(Q,axis=1))
  V=np.max(Q, axis=1)
  pi=lambda s:{s:a for s, a in enumerate(np.argmax(Q, axis=1))} [s]

  # return Q, V, pi, Q_track, pi_track
  return Q, V, pi
```

Print the optimal Value Funtion

```
optimal_Q, optimal_V, optimal_pi = mc_control (env,n_episodes = 3000)
print('Name: Meetha Prabhu     Register Number: 212222240065        ')
print_state_value_function(optimal_Q, P, n_cols=4, prec=2, title='Action-
value function:')
print_state_value_function(optimal_V, P, n_cols=4, prec=2, title='State-
value function:')
print_policy(optimal_pi,P)
```

Probability of Success:

```
# Find the probability of success and the mean return of you your policy
print('Name: Meetha Prabhu     Register Number: 212222240065        ')
print('Reaches goal {:.2f}%. Obtains an average undiscounted return of
{:.4f}.'.format(
    probability_success(env, optimal_pi, goal_state=goal_state)*100,
    mean_return(env, optimal_pi)))
```

OUTPUT:

```
                                                     Name: Meetha Prabhu     Register Number: 212222240065
Action-value function:
| 00 [0.11 0.09 0.09 0.07] | 01 [0.02 0.01 0.05 0.08] | 02 [0.05 0.08 0.02 0.01] | 03 [0.06 0.01 0.   0.  ] |
| 04 [0.12 0.05 0.06 0.05] |           | 06 [0.01 0.05 0.11 0.  ] |           |
| 08 [0.06 0.07 0.06 0.13] | 09 [0.05 0.09 0.16 0.09] | 10 [0.2  0.29 0.18 0.02] |           |
|           | 13 [0.01 0.04 0.15 0.22] | 14 [0.14 0.29 0.57 0.3 ] |           |
State-value function:
| 00   0.11 | 01   0.08 | 02   0.08 | 03   0.06 |
| 04   0.12 |           | 06   0.11 |           |
| 08   0.13 | 09   0.16 | 10   0.29 |           |
|           | 13   0.22 | 14   0.57 |           |
Policy:
| 00      < | 01      ^ | 02      v | 03      < |
| 04      < |           | 06      > |           |
| 08      ^ | 09      > | 10      v |           |
|           | 13      ^ | 14      > |           |
```

```
Name: Meetha Prabhu     Register Number: 212222240065
Reaches goal 16.00%. Obtains an average undiscounted return of 0.1600.
```

Mention the Action value function, optimal value function, optimal policy, and success rate for the optimal policy.

RESULT:

Thus the program to implement Monte Carlo control for a given environment is implemented sucessfully.