Bansilal Ramnath Agarwal Charitable Trust's

Vishwakarma Institute of Information Technology

Department of Artificial Intelligence and Data Science

Name: Siddhesh Dilip Khairnar		
Class: TY	Division: B	Roll No: 372028
Semester: V	Academic Year: 2023-2024	
Subject Name & Code: ADUA31201: Artificial Intelligence		
Title of Assignment: Write a program which uses Q-values to iteratively improve the behaviour of learning agent (Reinforcement learning)		
Date of Performance:24-10-2023	Date of Submission: 11-11-2023	

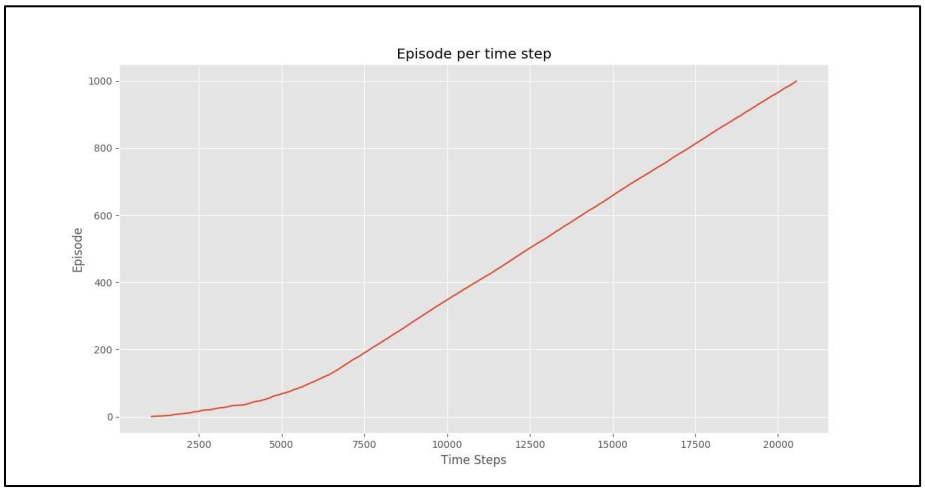
ASSIGNMENT NO. 8

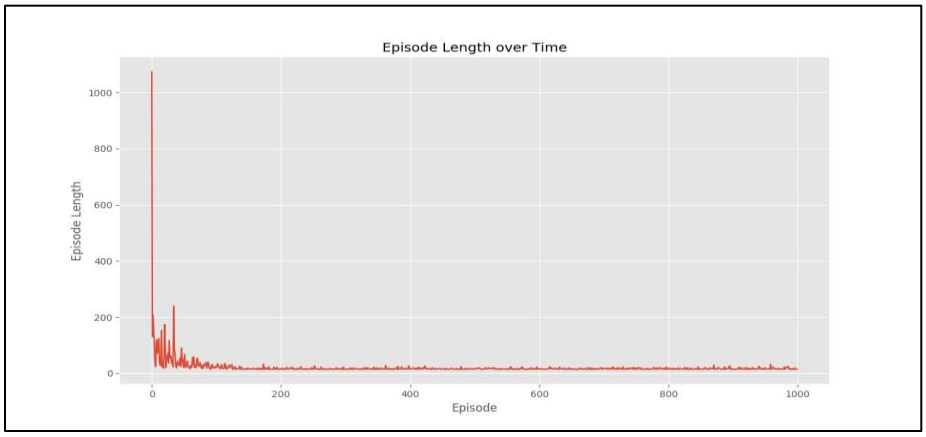
CODE:

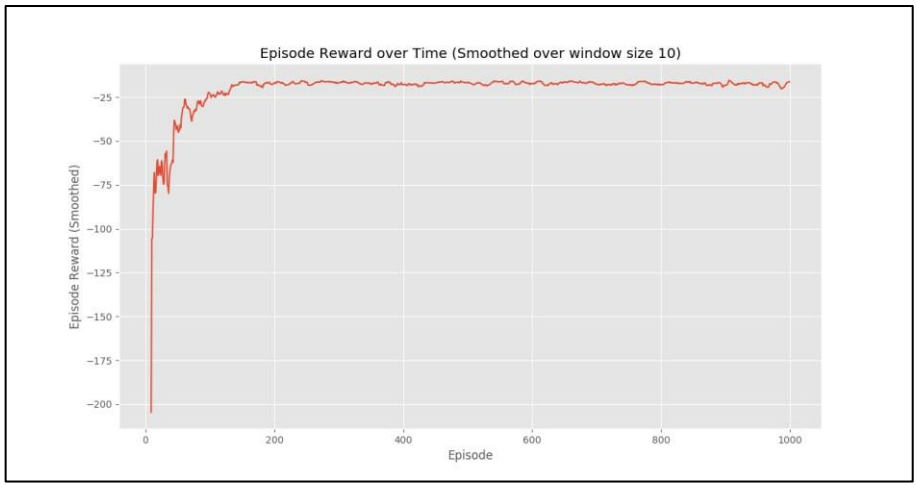
```
import itertools
import matplotlib
import matplotlib.style
import numpy as np
from collections import defaultdict
from plotting import EpisodeStats
from plotting import plot_episode_stats
import plotting
from windy_gridworld import WindyGridworldEnv


matplotlib.style.use('ggplot')
env = WindyGridworldEnv()


def create_epsilon_greedy_policy(Q, epsilon, num_actions):
    def policy_function(state):
        action_probabilities = np.ones(
            num_actions, dtype=float) * epsilon / num_actions
        best_action = np.argmax(Q[state])
        action_probabilities[best_action] += (1.0 - epsilon)
        return action_probabilities

    return policy_function


def q_learning(env, num_episodes, discount_factor=1.0, alpha=0.6,
epsilon=0.1):
    Q = defaultdict(lambda: np.zeros(env.action_space.n))
    stats = plotting.EpisodeStats(
        episode_lengths=np.zeros(num_episodes),
        episode_rewards=np.zeros(num_episodes)
    )
    policy = create_epsilon_greedy_policy(Q, epsilon, env.action_space.n)

    for ith_episode in range(num_episodes):
        state = env.reset()
        for t in itertools.count():
            action_probabilities = policy(state)
            action = np.random.choice(
                np.arange(len(action_probabilities)), p=action_probabilities)
            next_state, reward, done, _ = env.step(action)
            stats.episode_rewards[ith_episode] += reward
            stats.episode_lengths[ith_episode] = t
            best_next_action = np.argmax(Q[next_state])
            td_target = reward + discount_factor * \
```

```
                Q[next_state][best_next_action]
            td_delta = td_target - Q[state][action]
            Q[state][action] += alpha * td_delta
            if done:
                break
            state = next_state

    return Q, stats


Q, stats = q_learning(env, 1000)
plotting.plot_episode_stats(stats)
```

OUTPUT:


Episode per time step
Episode
Time Steps
0
200
400
600
800
1000
2500
5000
7500
10000
12500
15000
17500
20000


Episode Length over Time
Episode Length
Episode
0
200
400
600
800
1000


Episode Reward over Time (Smoothed over window size 10)
Episode Reward (Smoothed)
Episode
−25
−50
−75
−100
−125
−150
−175
−200
0
200
400
600
800
1000