|  |  |  |  |
| --- | --- | --- | --- |
|  | 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:**

