Name : Devkumar Biswas

Class: BE(AI&DS)

Div: B

Subject : ML(CL-I Lab)

Roll no. : BEAD21267

**Assignment No. - 6**

**Problem Statement** : Reinforcement Learning (Any one)

A. Implement Reinforcement Learning using an example of a maze environment that the agent needs to explore.

Code:

def plot\_path(maze, path):

plt.imshow(maze, cmap='gray')

for p in path:

plt.scatter(p[1], p[0], c='red')

plt.show()

plot\_path(maze, path)

maze = np.array([

[0, 0, 1, 0, 0],

[0, 0, 1, 0, 0],

[0, 0, 0, 0, 0],

[0, 1, 1, 1, 0],

[0, 0, 0, 0, 2]

])

# Parameters

alpha = 0.8 # Learning rate

gamma = 0.95 # Discount factor

epsilon = 0.1 # Exploration rate

episodes = 1000 # Number of episodes

# Q-Table

q\_table = np.zeros((maze.shape[0], maze.shape[1], 4)) # 4 possible actions: up, down, left, right

# Define actions

actions = ['up', 'down', 'left', 'right']

def get\_next\_action(state):

if random.uniform(0, 1) < epsilon:

return random.randint(0, 3) # Explore: choose a random action

else:

return np.argmax(q\_table[state[0], state[1]]) # Exploit: choose the action with max Q-value

def get\_next\_state(state, action):

next\_state = state.copy()

if action == 0 and state[0] > 0: # Up

next\_state[0] -= 1

elif action == 1 and state[0] < maze.shape[0] - 1: # Down

next\_state[0] += 1

elif action == 2 and state[1] > 0: # Left

next\_state[1] -= 1

elif action == 3 and state[1] < maze.shape[1] - 1: # Right

next\_state[1] += 1

# Check for obstacles

if maze[next\_state[0], next\_state[1]] == 1:

return state # Return to the original state if an obstacle is hit

return next\_state

def get\_reward(state):

if maze[state[0], state[1]] == 2:

return 100 # Reward for reaching the goal

else:

return -1 # Penalty for each step taken

for episode in range(episodes):

state = [0, 0] # Starting position

while maze[state[0], state[1]] != 2:

action = get\_next\_action(state)

next\_state = get\_next\_state(state, action)

reward = get\_reward(next\_state)

old\_value = q\_table[state[0], state[1], action]

next\_max = np.max(q\_table[next\_state[0], next\_state[1]])

# Update Q-value for the current state-action pair

q\_table[state[0], state[1], action] = old\_value + alpha \* (reward + gamma \* next\_max - old\_value)

state = next\_state # Move to the next state

state = [0, 0]

path = [state]

while maze[state[0], state[1]] != 2:

action = np.argmax(q\_table[state[0], state[1]])

state = get\_next\_state(state, action)

path.append(state)

print("Path taken by the agent:")

print(path)

def plot\_path(maze, path):

plt.imshow(maze, cmap='gray')

for p in path:

plt.scatter(p[1], p[0], c='red')

plt.show()

plot\_path(maze, path)

Output:







