import numpy as np

import matplotlib.pyplot as plt

import random

from collections import defaultdict, deque

class MazeEnv:

def \_\_init\_\_(self, grid, start, goal, step\_reward=-0.04, goal\_reward=1.0, hit\_wall\_reward=-0.2):

self.grid = np.array(grid)

self.start = start

self.goal = goal

self.step\_reward = step\_reward

self.goal\_reward = goal\_reward

self.hit\_wall\_reward = hit\_wall\_reward

self.n\_rows, self.n\_cols = self.grid.shape

self.action\_space = [0,1,2,3]

self.reset()

def reset(self):

self.agent\_pos = tuple(self.start)

return self.agent\_pos

def in\_bounds(self, r, c):

return 0 <= r < self.n\_rows and 0 <= c < self.n\_cols

def is\_free(self, r, c):

return self.in\_bounds(r,c) and self.grid[r,c] == 0

def step(self, action):

r, c = self.agent\_pos

if action == 0:

nr, nc = r-1, c

elif action == 1:

nr, nc = r, c+1

elif action == 2:

nr, nc = r+1, c

elif action == 3:

nr, nc = r, c-1

if not self.is\_free(nr, nc):

reward = self.hit\_wall\_reward

done = False

next\_state = (r,c)

else:

next\_state = (nr,nc)

if next\_state == tuple(self.goal):

reward = self.goal\_reward

done = True

else:

reward = self.step\_reward

done = False

self.agent\_pos = next\_state

return next\_state, reward, done

def render\_text(self, policy=None, q\_table=None):

arrow\_map = {0:'↑',1:'→',2:'↓',3:'←'}

out = ""

for i in range(self.n\_rows):

for j in range(self.n\_cols):

if (i,j) == tuple(self.goal):

out += " G "

elif (i,j) == tuple(self.start):

out += " S "

elif self.grid[i,j] == 1:

out += "###"

else:

if policy and (i,j) in policy:

out += f" {arrow\_map[policy[(i,j)]]} "

elif q\_table and (i,j) in q\_table:

greedy = np.argmax(q\_table[(i,j)])

out += f" {arrow\_map[greedy]} "

else:

out += " . "

out += "\n"

print(out)

def all\_states(self):

for i in range(self.n\_rows):

for j in range(self.n\_cols):

if self.grid[i,j] == 0 or (i,j) == tuple(self.goal) or (i,j) == tuple(self.start):

yield (i,j)

class QLearningAgent:

def \_\_init\_\_(self, actions, alpha=0.5, gamma=0.99, epsilon=0.1):

self.actions = actions

self.alpha = alpha

self.gamma = gamma

self.epsilon = epsilon

self.q = defaultdict(lambda: np.zeros(len(actions)))

def choose\_action(self, state):

if random.random() < self.epsilon:

return random.choice(self.actions)

else:

return int(np.argmax(self.q[state]))

def update(self, state, action, reward, next\_state, done):

qsa = self.q[state][action]

if done:

target = reward

else:

target = reward + self.gamma \* np.max(self.q[next\_state])

self.q[state][action] = qsa + self.alpha \* (target - qsa)

def get\_policy(self):

policy = {}

for s, qvals in self.q.items():

policy[s] = int(np.argmax(qvals))

return policy

def build\_sample\_maze():

grid = [

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

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

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

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

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

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

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

]

start = (0,0)

goal = (6,8)

return np.array(grid), start, goal

def train\_q\_learning(env, episodes=2000, max\_steps=200, alpha=0.5, gamma=0.99, epsilon=0.2, decay\_epsilon=False):

agent = QLearningAgent(actions=env.action\_space, alpha=alpha, gamma=gamma, epsilon=epsilon)

rewards\_history = []

success\_history = deque(maxlen=100)

for ep in range(1, episodes+1):

state = env.reset()

total\_reward = 0.0

done = False

for step in range(max\_steps):

action = agent.choose\_action(state)

next\_state, reward, done = env.step(action)

agent.update(state, action, reward, next\_state, done)

state = next\_state

total\_reward += reward

if done:

break

rewards\_history.append(total\_reward)

success\_history.append(1 if done else 0)

if decay\_epsilon:

agent.epsilon = max(0.01, agent.epsilon \* 0.995)

if ep % 200 == 0 or ep == 1:

recent\_success\_rate = np.mean(list(success\_history)) if len(success\_history) > 0 else 0.0

print(f"Episode {ep}/{episodes} - Reward: {total\_reward:.2f} - Success%: {recent\_success\_rate\*100:.1f}% - Epsilon: {agent.epsilon:.3f}")

return agent, rewards\_history

def plot\_rewards(rewards):

plt.figure(figsize=(10,4))

plt.plot(rewards, label='Episode reward')

window = max(1, len(rewards)//50)

smoothed = np.convolve(rewards, np.ones(window)/window, mode='valid')

plt.plot(range(window-1, window-1+len(smoothed)), smoothed, label='Smoothed', linewidth=2)

plt.xlabel('Episode')

plt.ylabel('Total Reward')

plt.title('Training Rewards per Episode')

plt.legend()

plt.grid(True)

plt.tight\_layout()

plt.show()

def visualize\_policy(env, q\_table):

policy = {}

for s in env.all\_states():

if s in q\_table:

policy[s] = int(np.argmax(q\_table[s]))

print("Final greedy policy:")

env.render\_text(policy=policy)

def run\_episode\_with\_policy(env, agent, max\_steps=200, render=True):

state = env.reset()

path = [state]

for \_ in range(max\_steps):

action = int(np.argmax(agent.q[state]))

next\_state, reward, done = env.step(action)

path.append(next\_state)

state = next\_state

if done:

break

if render:

print("Path taken by greedy policy:")

print(path)

return path, done

def main():

grid, start, goal = build\_sample\_maze()

env = MazeEnv(grid, start, goal, step\_reward=-0.04, goal\_reward=1.0, hit\_wall\_reward=-0.2)

print("Maze layout:")

env.render\_text()

agent, rewards = train\_q\_learning(env, episodes=2500, max\_steps=200,

alpha=0.6, gamma=0.98, epsilon=0.3, decay\_epsilon=True)

plot\_rewards(rewards)

visualize\_policy(env, agent.q)

path, success = run\_episode\_with\_policy(env, agent, render=True)

print("Reached goal?", success)

coords = np.array(path)

fig, ax = plt.subplots(figsize=(6,6))

ax.imshow(env.grid==1, cmap='gray\_r')

ax.plot(coords[:,1], coords[:,0], marker='o')

ax.scatter(start[1], start[0], c='green', s=120, label='Start')

ax.scatter(goal[1], goal[0], c='red', s=120, label='Goal')

ax.set\_title('Path followed by greedy policy')

ax.set\_xlim(-0.5, env.n\_cols-0.5)

ax.set\_ylim(env.n\_rows-0.5, -0.5)

ax.legend()

plt.grid(True)

plt.show()

if \_\_name\_\_ == "\_\_main\_\_":

main()  
  
  
Output:-

Episode 1/2500 - Reward: -18.24 - Success%: 100.0% - Epsilon: 0.298

Episode 200/2500 - Reward: 0.28 - Success%: 100.0% - Epsilon: 0.110

Episode 400/2500 - Reward: -0.12 - Success%: 100.0% - Epsilon: 0.040

Episode 600/2500 - Reward: 0.48 - Success%: 100.0% - Epsilon: 0.015

Episode 800/2500 - Reward: 0.48 - Success%: 100.0% - Epsilon: 0.010

Episode 1000/2500 - Reward: 0.48 - Success%: 100.0% - Epsilon: 0.010

Episode 1200/2500 - Reward: 0.48 - Success%: 100.0% - Epsilon: 0.010

Episode 1400/2500 - Reward: 0.48 - Success%: 100.0% - Epsilon: 0.010

Episode 1600/2500 - Reward: 0.48 - Success%: 100.0% - Epsilon: 0.010

Episode 1800/2500 - Reward: 0.48 - Success%: 100.0% - Epsilon: 0.010

Episode 2000/2500 - Reward: 0.48 - Success%: 100.0% - Epsilon: 0.010

Episode 2200/2500 - Reward: 0.48 - Success%: 100.0% - Epsilon: 0.010

Episode 2400/2500 - Reward: 0.48 - Success%: 100.0% - Epsilon: 0.010



Path taken by greedy policy:

[(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (0, 5), (0, 6), (0, 7), (0, 8), (1, 8), (2, 8), (3, 8), (4, 8), (5, 8), (6, 8)]

Reached goal? True

