```
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