2025-2026-1 学期强化学习课程 - 第二次作业

1120231863　左逸龙

October, 28 2025

1 通勤北理工

(i) 既然我们已经知晓了最优的 Q^* 表，那么每一状态下的最优策略满足：

$$\pi^*(a|s) = \begin{cases} 1 \text{ if } a = \text{argmax}_{a \in A(s)} Q^*(s,a) \\ 0 \text{ otherwise} \end{cases}$$

因此最优策略为：

- 在状态 S_1 下，乘坐班车
- 在状态 S_2 下，乘坐班车
- 在状态 S_3 下，乘坐地铁

(ii) 此时最优策略为：

- 在状态 S_{12} 下，乘坐班车
- 在状态 S_3 下，乘坐班车

显然，此时得到的最优策略与使用真实三状态表示时得到的最优策略不同，关键区别在于原来状态 S_3 的最优策略为**乘坐地铁**，而现在状态 S_3 的最优策略为**乘坐班车**。

之所以会出现这样的变化，是因为**状态聚合**导致智能体 Agent 无法区分S_1与S_2的**未来价值**，使得决策依据退化为**即时奖励**。分析如下：

- Q-learning 算法的更新公式为：

$$Q(s,a) = Q(s,a) + \alpha\left[R + \gamma \max_{a \in A(s')} Q(s',a)\right]$$

其中α为学习率，R为即时奖励，γ为折扣因子。由公式可以看出，决定 Q 值的不仅仅只有**即时奖励**，还有**未来价值的预期**。

- 在原始的三状态模型中，Agent 可以精确地知道每个动作会导向哪个具体的状态。从 S_3 出发，乘坐地铁会到达 S_2，而 S_2 的长期价值 ($V^*(S_2) = 1.95$) 远高于乘坐班车所到达的 S_1 的长期价值 ($V^*(S_1) = 1.65$)。尽管坐地铁的即时奖励更低，但为了追求 S_2 带来的更高未来收益，最优策略是选择乘坐地铁。

- 在聚合后的二状态模型中，S_1 和 S_2 被合并为宏状态 S_{12}。此时，无论从 S_3 出发选择乘坐班车（到达 S_1）还是乘坐地铁（到达 S_2），在 Agent 看来，下一个状态都是**同一个** S_{12}。因此，这两个动作所带来的未来价值预期是完全相同的（都等于 $\gamma V^*(S_{12})$）。
- 当两个动作的**未来价值预期相同时**，决策的优劣就完全取决于**即时奖励**。根据表 1，$R(S_3, \text{班车}) = -0.5$，而 $R(S_3, \text{地铁}) = -0.7$。由于 $-0.5 > -0.7$，选择乘坐班车能获得更好的即时奖励。因此，在这种信息受限的情况下，最优策略从乘坐地铁转变为乘坐班车。

综上，导致这种策略上变化的原因是状态表示的粒度变粗后，Q-learning 算法泛化（或平均化）了不同状态的价值，导致决策依据从**长远未来价值**退化为**即时奖励**。

2 Frozenlake 小游戏

(i) 以下是 `policy_evaluation`, `policy_improvement`, `policy_iteration` 的代码实现:

- `policy_evaluation`: 实现了策略评估，即给定一个策略，计算其价值函数。

```python
def policy_evaluation(
    P: PType,
    nS: int,
    nA: int,
    policy: np.ndarray,
    gamma: float = 0.9,
    tol: float = 1e-3
) -> np.ndarray:

    value_function = np.zeros(nS)
    while True:
        delta = 0 # 用于记录价值函数的变化量，小于tol时认为收敛，停止迭代
        for s in range(nS):
            v = value_function[s]
            a = policy[s]

            new_v = 0

            # 状态价值函数更新公式：
            # V_{k+1}(s) = sum_(a in A) pi(a|s) * (sum_(s' in S')
            # P(s'|s,a) * (R(s'|s,a) + gamma * V_{k}(s')))
            # 此处由于policy已经确定，所以 A(s) = {a}，pi(a|s) = 1，
            # 因此原式化可以简化为：
            # V_{k+1}(s) = sum_(s' in S') P(s'|s,a) * (R(s'|s,a)+
            # gamma * V_{k}(s'))
            for prob, next_state, reward, terminal in P[s][a]:
                new_v += prob * (reward + gamma * value_function[next_state])
            value_function[s] = new_v
            delta = max(delta, abs(v - value_function[s]))

        if delta < tol:
```

```
            break

    return value_function
```

- `policy_improvement`：实现了策略提升，即给定价值函数，计算其最优策略。

```
def policy_improvement(
    P: PType,
    nS: int,
    nA: int,
    value_from_policy: np.ndarray,
    policy: np.ndarray,
    gamma: float = 0.9
) -> np.ndarray:

    new_policy = np.zeros(nS, dtype="int")
    for s in range(nS):
        q_values = np.zeros(nA)
        for a in range(nA):
            # 动作价值函数更新公式：
            # Q(s,a) = sum_(s' in S) P(s'|s,a) * (R(s'|s,a) +
            # gamma * V(s'))
            for prob, next_state, reward, terminal in P[s][a]:
                q_values[a] += prob * (reward + gamma * value_from_policy[next_state])

        # 贪婪策略，选择 Q 值最大的动作
        new_policy[s] = np.argmax(q_values)

    return new_policy
```

- `policy_iteration`：实现了策略迭代，交替进行策略评估与提升，直到收敛。

```
def policy_iteration(
    P: PType,
    nS: int,
    nA: int,
    gamma: float = 0.9,
    tol: float = 1e-3
) -> Tuple[np.ndarray, np.ndarray]:

    value_function = np.zeros(nS)
    policy = np.zeros(nS, dtype=int)

    iterations = 0
    while True:
        value_function = policy_evaluation(P, nS, nA, policy, gamma, tol)

        new_policy = policy_improvement(P, nS, nA, value_function, policy, gamma)

        iterations += 1

        # 如果新旧策略相同，则认为策略收敛，停止迭代
        if np.array_equal(policy, new_policy):
            break

```

```
        policy = new_policy

    print(f"Policy Iteration converged in {iterations} iterations")

    return value_function, policy
```

以下是实验的关键配置，其中 `env.is_slippery = False` 表示确定性环境：

```yaml
# config.yaml
env:
  map_size: 4
  frozen_prob: 0.8
  seed: 20241022
  is_slippery: False
policy_iteration:
  gamma: 0.9
  tol: 1e-3
render:
  max_steps: 100
algorithm: policy_iteration
```

以下是实验结果，有修改 `run.py` 以记录更多数据：

```
Policy Iteration converged in 8 iterations
Training completed in 0.0012 seconds

Test results (100 episodes):
Total reward: 100.00
Success rate: 100.00%
Average steps (successful episodes): 5.00
```

可以看到，策略迭代在 8 次迭代后收敛，测试结果表明智能体在 100 次测试中成功到达目标状态 100 次，成功率为 100%，平均步数为 5 步，均为实际最优策略。可以看出策略迭代算法取得了十分理想的效果。

(ii) 以下是 `Q-Learning` 的代码实现：

```python
import gymnasium
from datetime import datetime

def QLearning(
    env:gymnasium.Env,
    num_episodes=2000,
    gamma=0.9,
    lr=0.1,
    epsilon=0.8,
    epsilon_decay=0.99
) -> np.ndarray:

    nS:int = env.observation_space.n
    nA:int = env.action_space.n
    Q = np.zeros((nS, nA))

    # 用于监控训练进度
```

```python
    total_rewards = []
    success_count = []
    start_time = datetime.now()

    max_steps_per_episode = 100  # 防止无限循环

    for episode in range(num_episodes):
        state, info = env.reset()
        done = False
        episode_reward = 0
        steps = 0

        while not done and steps < max_steps_per_episode:
            # 实现了 epsilon-greedy 策略：
            # 小于 epsilon 时随机选择动作，否则选择 Q 值最大的动作
            if np.random.random() < epsilon:
                action = env.action_space.sample()
            else:
                action = np.argmax(Q[state])

            next_state, reward, terminated, truncated, info = env.step(action)
            done = terminated or truncated
            episode_reward += reward

            # Q-learning 更新公式：
            # Q(s,a) ← Q(s,a) + lr * [R(s'|s,a) +
            # gamma * max_a Q(s',a) - Q(s,a)]
            best_next_action = np.argmax(Q[next_state])
            td_target = reward + gamma * Q[next_state, best_next_action]
            td_error = td_target - Q[state, action]
            Q[state, action] += lr * td_error

            state = next_state
            steps += 1

        # 衰减 epsilon，设置下界保持一定程度的探索
        epsilon = max(0.1, epsilon * epsilon_decay)

        # 打印数据用
        total_rewards.append(episode_reward)
        success_count.append(1 if episode_reward > 0 else 0)

        # 每 500 个 episode 打印一次进度
        if (episode + 1) % 500 == 0:
            end_time = datetime.now()
            elapsed_time = (end_time - start_time).total_seconds()
            avg_reward = np.mean(total_rewards[-500:])
            success_rate = np.mean(success_count[-500:]) * 100
            print(f"Episode {episode + 1}/{num_episodes}, "
                  f"Avg Reward (last 500): {avg_reward:.3f}, "
                  f"Success Rate (last 500): {success_rate:.3f}%, "
                  f"Time: {elapsed_time:.2f}s, "
                  f"Epsilon: {epsilon:.3f}")

    return Q
```

以下是实验的关键配置，其中 `env.is_slippery = False` 表示确定性环境：

```yaml
# config.yaml
env:
  map_size: 4
  frozen_prob: 0.8
  seed: 20241022
  is_slippery: False
qlearning:
  num_episodes: 5000
  gamma: 0.9
  learning_rate: 0.1
  epsilon: 1
  epsilon_decay: 0.9995
render:
  max_steps: 100
algorithm: QLearning
```

以下是实验结果，有修改 `run.py` 以记录更多数据：

```
Episode 500/5000, Avg Reward (last 500): 0.080, Success Rate (last 500): 8.000%, Time: 0.08s, Epsilon: 0.779
Episode 1000/5000, Avg Reward (last 500): 0.254, Success Rate (last 500): 25.400%, Time: 0.16s, Epsilon: 0.606
Episode 1500/5000, Avg Reward (last 500): 0.436, Success Rate (last 500): 43.600%, Time: 0.23s, Epsilon: 0.472
Episode 2000/5000, Avg Reward (last 500): 0.552, Success Rate (last 500): 55.200%, Time: 0.30s, Epsilon: 0.368
Episode 2500/5000, Avg Reward (last 500): 0.718, Success Rate (last 500): 71.800%, Time: 0.36s, Epsilon: 0.286
Episode 3000/5000, Avg Reward (last 500): 0.776, Success Rate (last 500): 77.600%, Time: 0.43s, Epsilon: 0.223
Episode 3500/5000, Avg Reward (last 500): 0.828, Success Rate (last 500): 82.800%, Time: 0.49s, Epsilon: 0.174
Episode 4000/5000, Avg Reward (last 500): 0.866, Success Rate (last 500): 86.600%, Time: 0.54s, Epsilon: 0.135
Episode 4500/5000, Avg Reward (last 500): 0.878, Success Rate (last 500): 87.800%, Time: 0.60s, Epsilon: 0.105
Episode 5000/5000, Avg Reward (last 500): 0.908, Success Rate (last 500): 90.800%, Time: 0.65s, Epsilon: 0.100

Training completed in 0.65 seconds

Test results (100 episodes):
Total reward: 100.00
Success rate: 100.00%
Average steps (successful episodes): 5.00
```

可以看到，Q-Learning 在训练了 0.65 秒后收敛，训练过程当中 `Avg Reward` 和 `Success Rate` 均在稳步提升，表明策略正在逐渐收敛到最优策略。测试结果表明智能体在 100 次测试中成功到达目标状态 100 次，成功率为 100%，平均步数为 5 步，均为实际最优策略。可以看出 Q-Learning 算法取得了十分理想的效果。

(iii) 保持其他配置不变，仅修改 `env.is_slippery = True`，两种算法的实验结果如下：

- Policy Iteration:

```
Policy Iteration converged in 5 iterations
Training completed in 0.0041 seconds

Test results (100 episodes):
Total reward: 100.00
Success rate: 100.00%
Average steps (successful episodes): 35.38
```

- Q-Learning:

```
Episode 500/5000, Avg Reward (last 500): 0.020, Success Rate (last 500): 2.000%, Time: 0.06s, Epsilon: 0.779
Episode 1000/5000, Avg Reward (last 500): 0.024, Success Rate (last 500): 2.400%, Time: 0.13s, Epsilon: 0.606
Episode 1500/5000, Avg Reward (last 500): 0.076, Success Rate (last 500): 7.600%, Time: 0.22s, Epsilon: 0.472
Episode 2000/5000, Avg Reward (last 500): 0.100, Success Rate (last 500): 10.000%, Time: 0.32s, Epsilon: 0.368
Episode 2500/5000, Avg Reward (last 500): 0.164, Success Rate (last 500): 16.400%, Time: 0.44s, Epsilon: 0.286
Episode 3000/5000, Avg Reward (last 500): 0.224, Success Rate (last 500): 22.400%, Time: 0.57s, Epsilon: 0.223
Episode 3500/5000, Avg Reward (last 500): 0.282, Success Rate (last 500): 28.200%, Time: 0.72s, Epsilon: 0.174
Episode 4000/5000, Avg Reward (last 500): 0.360, Success Rate (last 500): 36.000%, Time: 0.88s, Epsilon: 0.135
Episode 4500/5000, Avg Reward (last 500): 0.332, Success Rate (last 500): 33.200%, Time: 1.04s, Epsilon: 0.105
Episode 5000/5000, Avg Reward (last 500): 0.438, Success Rate (last 500): 43.800%, Time: 1.23s, Epsilon: 0.100

Training completed in 1.23 seconds

Test results (100 episodes):
The agent didn't reach a terminal state in 100 steps.
Total reward: 99.00
Success rate: 99.00%
Average steps (successful episodes): 38.23
```

直接观察结果，可以发现相较于确定性环境，两种算法有如下变化：

- Policy Iteration:
  - 收敛速度变慢，时间从 0.0012s 增加到 0.0041s，即便迭代次数从 8 次降低到了 5 次；
  - 成功率保持 100% 不变，但平均步数从 5 步增加到了 35.38 步；倘若我们提高要求，进一步限制最大步数的话，成功率可能会降低；
- Q-Learning:
  - 训练时间从 0.65s 增加到 1.23s，表明随机环境需要更多采样步骤；

- 训练曲线不稳定，训练过程当中的最终成功率显著降低至 43.8%，相比确定性环境的 90.8% 下降了一半多；
- 然而测试结果的成功率依然高达 99%，这是因为测试时使用的是贪婪策略（选择 Q 值最大的动作），而训练时使用 epsilon-greedy 策略（有 10% 的探索概率），因此测试性能能够反映已学习到的最优策略，而训练过程中的成功率受探索行为影响而较低；

对比两种算法在随机性环境下的表现，可以发现：

- **Policy Iteration 相较于 Q-Learning 收敛速度更快，运行时间更短**，原因在于：
  - Policy Iteration 是基于模型的（Model-based）算法，直接利用环境的转移概率矩阵 P 进行动态规划，通过迭代计算贝尔曼方程即可收敛，不需要采样；
  - Q-Learning 是无模型的（Model-free）算法，需要通过与环境大量交互来估计 Q 值，随机环境中相同的状态-动作对会产生不同的结果，需要更多样本才能准确估计；
- **Policy Iteration 的鲁棒性显著优于 Q-Learning**，原因在于：
  - Policy Iteration 在策略评估阶段会计算所有可能转移的**期望值**，公式为 $V(s) = \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V(s')]$，这种期望值计算能够很好地适应随机性环境；
  - Q-Learning 通过单次采样更新 Q 值，公式为 $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$，单次采样无法反映转移的真实概率分布，导致学习过程噪声大、不稳定；
  - 从实验数据看，Policy Iteration 测试成功率 100%，而 Q-Learning 测试成功率虽然达到 99%，但训练过程显示其学习曲线波动较大，最终训练成功率仅 43.8%，说明策略质量不如 Policy Iteration 稳定；

(iv) 保持其他配置不变，仅修改 `map_size = 6`，两种算法的实验结果如下：

```
[2025-10-28 07:30:40,403][HYDRA]       #0 : algorithm=policy_iteration env.is_slippery=False env.render_mode=ansi

--------------------------------------------------
Beginning POLICY_ITERATION
--------------------------------------------------
Policy Iteration converged in 13 iterations
Training completed in 0.0056 seconds

Test results (100 episodes):
Total reward: 100.00
Success rate: 100.00%
Average steps (successful episodes): 9.00
[2025-10-28 07:30:40,520][HYDRA]       #1 : algorithm=policy_iteration env.is_slippery=True env.render_mode=ansi

--------------------------------------------------
Beginning POLICY_ITERATION
--------------------------------------------------
```

```
Policy Iteration converged in 7 iterations
Training completed in 0.0081 seconds

Test results (100 episodes):
Total reward: 5.00
Success rate: 5.00%
Average steps (successful episodes): 415.60
[2025-10-28 07:30:40,649][HYDRA]        #2 : algorithm=QLearning env.is_slippery=False env.render_mode=ansi

--------------------------------------------------
Beginning QLEARNING
--------------------------------------------------
Episode 500/5000, Avg Reward (last 500): 0.000, Success Rate (last 500): 0.000%, Time: 0.05s, Epsilon: 0.779
Episode 1000/5000, Avg Reward (last 500): 0.000, Success Rate (last 500): 0.000%, Time: 0.11s, Epsilon: 0.606
Episode 1500/5000, Avg Reward (last 500): 0.000, Success Rate (last 500): 0.000%, Time: 0.18s, Epsilon: 0.472
Episode 2000/5000, Avg Reward (last 500): 0.000, Success Rate (last 500): 0.000%, Time: 0.26s, Epsilon: 0.368
Episode 2500/5000, Avg Reward (last 500): 0.000, Success Rate (last 500): 0.000%, Time: 0.36s, Epsilon: 0.286
Episode 3000/5000, Avg Reward (last 500): 0.000, Success Rate (last 500): 0.000%, Time: 0.49s, Epsilon: 0.223
Episode 3500/5000, Avg Reward (last 500): 0.000, Success Rate (last 500): 0.000%, Time: 0.65s, Epsilon: 0.174
Episode 4000/5000, Avg Reward (last 500): 0.000, Success Rate (last 500): 0.000%, Time: 0.86s, Epsilon: 0.135
Episode 4500/5000, Avg Reward (last 500): 0.000, Success Rate (last 500): 0.000%, Time: 1.11s, Epsilon: 0.105
Episode 5000/5000, Avg Reward (last 500): 0.000, Success Rate (last 500): 0.000%, Time: 1.40s, Epsilon: 0.100

Training completed in 1.40 seconds

Test results (100 episodes):
Total reward: 0.00
Success rate: 0.00%
Average steps (successful episodes): 0.00
[2025-10-28 07:30:42,268][HYDRA]        #3 : algorithm=QLearning env.is_slippery=True env.render_mode=ansi

--------------------------------------------------
Beginning QLEARNING
--------------------------------------------------
Episode 500/5000, Avg Reward (last 500): 0.000, Success Rate (last 500): 0.000%, Time: 0.06s, Epsilon: 0.779
Episode 1000/5000, Avg Reward (last 500): 0.000, Success Rate (last 500): 0.000%, Time: 0.10s, Epsilon: 0.606
Episode 1500/5000, Avg Reward (last 500): 0.000, Success Rate (last 500): 0.000%, Time: 0.14s, Epsilon: 0.472
Episode 2000/5000, Avg Reward (last 500): 0.000, Success Rate (last 500): 0.000%, Time: 0.18s, Epsilon: 0.368
Episode 2500/5000, Avg Reward (last 500): 0.000, Success Rate (last 500): 0.000%, Time: 0.21s, Epsilon: 0.286
Episode 3000/5000, Avg Reward (last 500): 0.000, Success Rate (last 500): 0.000%, Time: 0.24s, Epsilon: 0.223
Episode 3500/5000, Avg Reward (last 500): 0.000, Success Rate (last 500): 0.000%, Time: 0.27s, Epsilon: 0.174
```

```
Episode 4000/5000, Avg Reward (last 500): 0.000, Success Rate (last 500): 0.000%, Time: 0.30s, Epsilon: 0.135
Episode 4500/5000, Avg Reward (last 500): 0.000, Success Rate (last 500): 0.000%, Time: 0.33s, Epsilon: 0.105
Episode 5000/5000, Avg Reward (last 500): 0.000, Success Rate (last 500): 0.000%, Time: 0.36s, Epsilon: 0.100

Training completed in 0.36 seconds

Test results (100 episodes):
Total reward: 0.00
Success rate: 0.00%
Average steps (successful episodes): 0.00
```

保持其他配置不变，仅修改 `map_size = 8`，两种算法的实验结果如下：

```
[2025-10-28 07:33:13,230][HYDRA]        #0 : algorithm=policy_iteration env.is_slippery=False env.render_mode=ansi

--------------------------------------------------
Beginning POLICY_ITERATION
--------------------------------------------------
Policy Iteration converged in 15 iterations
Training completed in 0.0140 seconds

Test results (100 episodes):
Total reward: 100.00
Success rate: 100.00%
Average steps (successful episodes): 13.00
[2025-10-28 07:33:13,369][HYDRA]        #1 : algorithm=policy_iteration env.is_slippery=True env.render_mode=ansi

--------------------------------------------------
Beginning POLICY_ITERATION
--------------------------------------------------
Policy Iteration converged in 10 iterations
Training completed in 0.0210 seconds

Test results (100 episodes):
The agent didn't reach a terminal state in 100 steps.
The agent didn't reach a terminal state in 100 steps.
The agent didn't reach a terminal state in 100 steps.
The agent didn't reach a terminal state in 100 steps.
The agent didn't reach a terminal state in 100 steps.
The agent didn't reach a terminal state in 100 steps.
The agent didn't reach a terminal state in 100 steps.
The agent didn't reach a terminal state in 100 steps.
The agent didn't reach a terminal state in 100 steps.
The agent didn't reach a terminal state in 100 steps.
Total reward: 12.00
Success rate: 12.00%
Average steps (successful episodes): 303.92
[2025-10-28 07:33:13,549][HYDRA]        #2 : algorithm=QLearning env.is_slippery=False env.render_mode=ansi

--------------------------------------------------
Beginning QLEARNING
--------------------------------------------------
```

```
Episode 500/5000, Avg Reward (last 500): 0.000, Success Rate (last 500): 0.000%, Time: 0.05s, Epsilon: 0.779
Episode 1000/5000, Avg Reward (last 500): 0.000, Success Rate (last 500): 0.000%, Time: 0.10s, Epsilon: 0.606
Episode 1500/5000, Avg Reward (last 500): 0.000, Success Rate (last 500): 0.000%, Time: 0.17s, Epsilon: 0.472
Episode 2000/5000, Avg Reward (last 500): 0.000, Success Rate (last 500): 0.000%, Time: 0.26s, Epsilon: 0.368
Episode 2500/5000, Avg Reward (last 500): 0.000, Success Rate (last 500): 0.000%, Time: 0.36s, Epsilon: 0.286
Episode 3000/5000, Avg Reward (last 500): 0.000, Success Rate (last 500): 0.000%, Time: 0.49s, Epsilon: 0.223
Episode 3500/5000, Avg Reward (last 500): 0.000, Success Rate (last 500): 0.000%, Time: 0.64s, Epsilon: 0.174
Episode 4000/5000, Avg Reward (last 500): 0.000, Success Rate (last 500): 0.000%, Time: 0.84s, Epsilon: 0.135
Episode 4500/5000, Avg Reward (last 500): 0.000, Success Rate (last 500): 0.000%, Time: 1.10s, Epsilon: 0.105
Episode 5000/5000, Avg Reward (last 500): 0.000, Success Rate (last 500): 0.000%, Time: 1.37s, Epsilon: 0.100

Training completed in 1.37 seconds

Test results (100 episodes):
Total reward: 0.00
Success rate: 0.00%
Average steps (successful episodes): 0.00
[2025-10-28 07:33:15,156][HYDRA]        #3 : algorithm=QLearning env.is_slippery=True env.render_mode=ansi

--------------------------------------------------
Beginning QLEARNING
--------------------------------------------------
Episode 500/5000, Avg Reward (last 500): 0.000, Success Rate (last 500): 0.000%, Time: 0.04s, Epsilon: 0.779
Episode 1000/5000, Avg Reward (last 500): 0.000, Success Rate (last 500): 0.000%, Time: 0.08s, Epsilon: 0.606
Episode 1500/5000, Avg Reward (last 500): 0.000, Success Rate (last 500): 0.000%, Time: 0.11s, Epsilon: 0.472
Episode 2000/5000, Avg Reward (last 500): 0.000, Success Rate (last 500): 0.000%, Time: 0.14s, Epsilon: 0.368
Episode 2500/5000, Avg Reward (last 500): 0.000, Success Rate (last 500): 0.000%, Time: 0.17s, Epsilon: 0.286
Episode 3000/5000, Avg Reward (last 500): 0.000, Success Rate (last 500): 0.000%, Time: 0.21s, Epsilon: 0.223
Episode 3500/5000, Avg Reward (last 500): 0.000, Success Rate (last 500): 0.000%, Time: 0.24s, Epsilon: 0.174
Episode 4000/5000, Avg Reward (last 500): 0.000, Success Rate (last 500): 0.000%, Time: 0.26s, Epsilon: 0.135
Episode 4500/5000, Avg Reward (last 500): 0.000, Success Rate (last 500): 0.000%, Time: 0.29s, Epsilon: 0.105
Episode 5000/5000, Avg Reward (last 500): 0.000, Success Rate (last 500): 0.000%, Time: 0.32s, Epsilon: 0.100

Training completed in 0.32 seconds

Test results (100 episodes):
Total reward: 0.00
Success rate: 0.00%
Average steps (successful episodes): 0.00
```

可以看到，Policy Iteration 在面对确定性环境时总是能够找到最优策略，而即便是面对随机性环境，也有机会找到策略，使智能体成功到达目标状态，但是平均步数显著增加。以上结果表明 Policy Iteration 具有较好的鲁棒性。

而 Q-Learning 在两种情况下均表现不佳，训练与测试结果均有明显异常(全 0)，我推测这是因为状态数与地图边长呈平方关系，状态空间爆炸，原有的参数设置不足以使智能体充分探索状态空间。我认为可以通过调整相关的参数设置，例如增加 episodes 数量，或者调整 epsilon 等参数，来改进 Q-Learning 的训练过程，使其能够更好地适应更大的状态空间。这也反应了 Q-Learning 算法对于参数的敏感性，没有 Policy Iteration 那么鲁棒。