# 强化学习：作业二

史浩男 502024370026

## 一、作业内容

实现Q-learning算法，学习一个移动小人走到出口的策略。

该智能体通过与环境交互来学习最优策略，从而最大化其累积奖励

> 算法关键思想：找到最优Q函数

## 二、实现过程

**首先修改algo.py文件，实现了MyQAgent**

### 1、初始化

- **学习率 (`alpha`)**：用于控制 Q 值更新的幅度，默认为 0.1。
- **折扣因子 (`gamma`)**：用于衡量未来奖励对当前决策的影响程度，默认为 0.9。
- **Q 值表 (`q_table`)**：使用 `defaultdict` 存储 Q 值表，键为状态，值为动作的 Q 值列表。初始 Q 值为 0.1，以鼓励探索。

```
class MyQAgent(QAgent):
    def __init__(self, action_shape=4, observation_shape=2, alpha=0.1,
gamma=0.9):
        super().__init__()
        self.action_shape = action_shape
        self.observation_shape = observation_shape
        self.alpha = alpha
        self.gamma = gamma
        self.q_table = defaultdict(lambda: [0.1 for _ in
range(action_shape)])
```

### 2、动作选择方法 (`select_action`)

```
def select_action(self, state):
    state_str = str(state)
    state_q = self.q_table[state_str]
    max_actions = []
    max_value = state_q[0]
    max_actions.append(0)
    for i in range(1, self.action_shape):
        if state_q[i] > max_value:
            max_actions.clear()
            max_value = state_q[i]
            max_actions.append(i)
        elif state_q[i] == max_value:
            max_actions.append(i)
    return random.choice(max_actions)
```

使用**贪婪策略**选择动作。在当前状态下，从 Q 值表中找到最大 Q 值对应的动作。

- **多重最大值处理**：如果存在多个动作具有相同的最大 Q 值，则随机选择一个，以增加决策的随机性，避免陷入局部最优。
- **状态表示**：状态被转换为字符串形式，以便在 `defaultdict` 中作为键来查找 Q 值列表。

#### 3 、学习方法 (`learn`)

```
def learn(self, state, action, reward, next_state):
    state_str = str(state)
    next_state_str = str(next_state)
    old_q = self.q_table[state_str][action]
    max_future_q = max(self.q_table[next_state_str]) if reward != 100 else 0
    new_q = reward + self.gamma * max_future_q
    self.q_table[state_str][action] += self.alpha * (new_q - old_q)
```

- **当前状态的 Q 值 (`old_q`)**：从 Q 值表中提取当前状态-动作对的 Q 值。
- **最优未来 Q 值 (`max_future_q`)**：计算下一状态的最大 Q 值，表示在最优策略下可以获得的最大未来回报。如果到达终止状态（奖励为 100），则未来 Q 值为 0，以避免非终止状态下的累积影响。

#### 4、修改了main函数中的epsilon部分，增加了ε-greedy策略

为了解决探索不足的问题，ε-贪婪策略在每一步决策时，以一定概率（ε）选择随机动作，以其余概率选择最优动作，这样可以更好地平衡探索与利用。

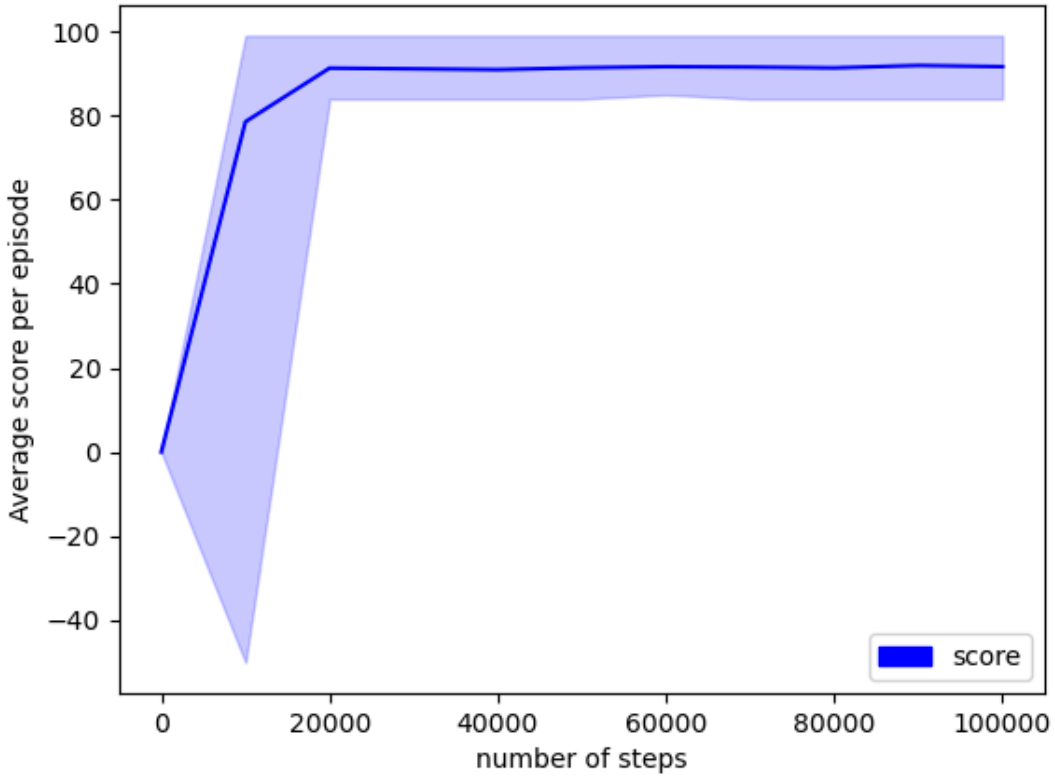
```
for step in range(args.num_steps):
    # Sample actions with epsilon greedy policy
    epsilon = 1 - i / 100.0
```

## 三、复现方式

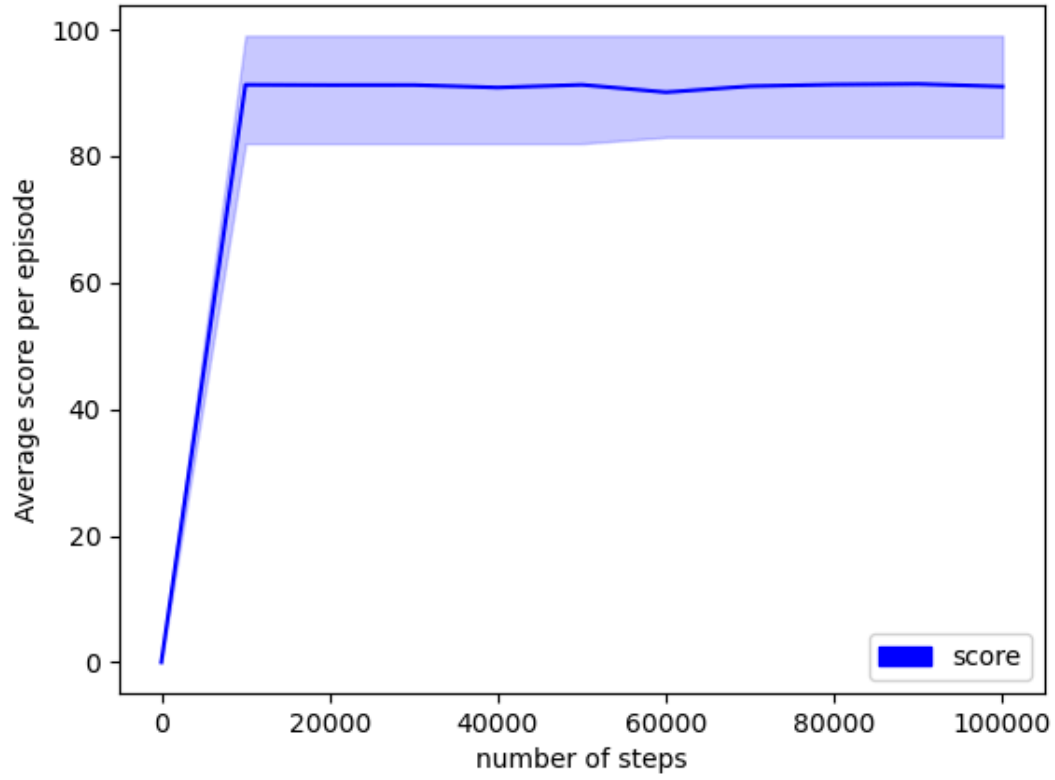
主文件夹下运行 python main.py.

## 四、实验效果

```
参数一：alpha=0.5, gamma=0.9
```




参数二：alpha=0.5, gamma=0.9




对应的分数输出：

```
TIME 00h 00m 01s Updates 99, num timesteps 10000, FPS 9422
 avrage/min/max reward 90.3/84.0/97.0
TIME 00h 00m 02s Updates 199, num timesteps 20000, FPS 7831
 avrage/min/max reward 91.1/84.0/99.0
TIME 00h 00m 03s Updates 299, num timesteps 30000, FPS 7917
 avrage/min/max reward 91.3/85.0/99.0
TIME 00h 00m 04s Updates 399, num timesteps 40000, FPS 8017
 avrage/min/max reward 91.5/85.0/99.0
TIME 00h 00m 06s Updates 499, num timesteps 50000, FPS 8064
 avrage/min/max reward 91.6/85.0/99.0
TIME 00h 00m 07s Updates 599, num timesteps 60000, FPS 8080
 avrage/min/max reward 91.5/85.0/99.0
TIME 00h 00m 08s Updates 699, num timesteps 70000, FPS 8113
 avrage/min/max reward 90.8/85.0/99.0
TIME 00h 00m 10s Updates 799, num timesteps 80000, FPS 7753
 avrage/min/max reward 91.2/85.0/99.0
TIME 00h 00m 11s Updates 899, num timesteps 90000, FPS 7768
 avrage/min/max reward 91.3/85.0/99.0
TIME 00h 00m 12s Updates 999, num timesteps 100000, FPS 7802
 avrage/min/max reward 91.7/85.0/99.0
```

## 五、分析与总结

通过此次代码实践，我成功实现了基于 Q-learning 的强化学习智能体，并深入理解了 Q-learning 算法的原理、贝尔曼方程。

在实验过程中，通过 Q 值表的构建和更新、贪婪策略的动作选择以及终止状态的处理，掌握了如何在有限状态空间中优化智能体的策略。同时，通过分析参数设置（如学习率和折扣因子）对智能体表现的影响，我意识到合理的参数选择对学习效率和收敛性的至关重要。 $\epsilon$-贪婪策略可以增加探索性。

本次实验为后续更复杂环境下的强化学习算法实践奠定了基础。