程序报告

学号：2313721　　姓名：许洋

一、问题重述

机器人自动走迷宫：分别使用基础搜索算法和 Deep QLearning 算法，完成机器人自动走迷宫。游戏规则为:从起点开始，通过错综复杂的迷宫，到达目标点(出口)。在任一位置可执行动作包括：向上走 'u'、向右走 'r'、向下走 'd'、向左走 'l'。

- 执行不同的动作后，根据不同的情况会获得不同的奖励，具体而言，有以下几种情况。

- 撞墙

- 走到出口

- 其余情况

- 需要实现基于基础搜索算法和 Deep QLearning 算法的机器人，使机器人自动走到迷宫的出口。

对问题的理解

- 自己实现基础算法和强化学习算法。其中，基础算法主要是搜索算法，强化学习算法则更加侧重于根据智能体与环境的交互来学习。

二、设计思想

2.1 基础算法

- 广度优先搜索

1. 首先以机器人起始位置建立根节点，并入队；
2. 接下来不断重复以下步骤直到判定条件：
    - 将队首节点的位置标记已访问；判断队首是否为目标位置(出口)， 若是则终止循环并记录回溯路径；
    - 判断队首节点是否为叶子节点，若是则拓展该叶子节点
    - 如果队首节点有子节点，则将每个子节点插到队尾
    - 将队首节点出队

- 深度优先搜索

1. 首先将根节点放入stack中。
2. 从stack中取出第一个节点，并检验它是否为目标。
   如果找到目标，则结束搜寻并回传结果。
   否则将它某一个尚未检验过的直接子节点加入stack中。
3. 重复步骤2。
4. 如果不存在未检测过的直接子节点。
   将上一级节点加入stack中。
   重复步骤2。
5. 重复步骤4。
6. 若stack为空，表示整张图都检查过了——亦即图中没有欲搜寻的目标。结束搜寻并回传“找不到目标”。

- A*算法

1. 将起点节点加入开放列表(open list)，并计算其评估值 f = g + h
   (g为起点到当前节点的实际代价，h为当前节点到目标的估计代价)
2. 从开放列表中取出f值最小的节点作为当前节点：
   a. 如果当前节点是目标节点，回溯路径并结束
   b. 否则将当前节点移至关闭列表(closed list)
3. 检查所有当前节点的相邻节点：
   a. 如果相邻节点不可通行或在关闭列表中，则跳过
   b. 如果相邻节点不在开放列表中：
    - 计算其g值和h值
    - 设置其父节点为当前节点
    - 将其加入开放列表
      c. 如果相邻节点已在开放列表中：

- 如果新路径的g值更小，更新其g值和父节点

4. 重复步骤2-3，直到：
   a. 找到目标节点（成功）
   b. 开放列表为空（失败，无解）

2.2 Deep QLearning算法

QLearning 算法要点：

- Q-Learning 算法是值迭代算法。
- Q-Learning 算法在执行过程中会计算每个”状态“或”状态-动作“的 Value，然后在执行动作的时候，会设法最大化这个值。算法（值迭代）的核心在于对每个状态值的准确估计。
- 考虑最大化动作的长期奖励——考虑当前动作带来的奖励 + 考虑动作长远的奖励。

QLearning 算法步骤：

1. 获取机器人所处迷宫位置
2. 对当前状态，检索Q表，如果不存在则添加进入Q表
3. 选择动作

- 为了防止出现 因机器人每次都选择它认为最优的路线而导致的路线固定（缺乏有效探索） 的现象，通常采用 *epsilon-greedy* 算法：
  - 在机器人选择动作的时候，以一部分的概率随机选择动作，以一部分的概率按照最优的 Q 值选择动作。
  - 同时，这个选择随机动作的概率应当随着训练的过程逐步减小。

4. 以给定的动作（移动方向）移动机器人
5. 获取机器人执行动作后所处的位置
6. 对当前 next_state ，检索Q表，如果不存在则添加进入Q表
7. 更新 Q 表 中 Q 值以及其他参数

- Q表（Q_table）：
  - Q-learning 算法将状态和动作构建成一张 Q_table 表来存储 Q 值；

- Q-Learning 算法中，长期奖励记为 Q 值，其中会考虑每个 ”状态-动作“ 的 Q 值，计算公式为：

$$Q(s_t, a) = R_{t+1} + \gamma \times \max_a Q(a, s_{t+1})$$

  - （St，a）：当前的 “ 状态-动作 ”
  - Rt+1：执行动作 a 后的环境奖励
  - maxQ(a，St+1)：执行任意动作能够获得的最大的 Q值
  - γ：折扣因子

- 然而，计算得到新的 Q 值之后，一般会使用更为保守地更新 Q 表的方法，即引入松弛变量 alpha ，按如下的公式进行更新，使得 Q 表的迭代变化更为平缓。

$$Q(s_t, a) = (1 - \alpha) \times Q(s_t, a) + \alpha \times (R_{t+1} + \gamma \times \max_a Q(a, s_{t+1}))$$

三、代码内容

- 基础算法

```
import heapq
def my_search(maze):
    """
    任选深度优先搜索算法、最佳优先搜索（A*)算法实现其中一种
    :param maze: 迷宫对象
    :return :到达目标点的路径 如：["u","u","r",...]
    """
    path = []
    # ----------------请实现你的算法代码-----------------------------
--------
    start = maze.sense_robot()
    goal = maze.destination

    direction_map = {
        'u': (-1, 0),
        'r': (0, 1),
        'd': (1, 0),
        'l': (0, -1)
    }
    def heuristic(pos):
        return abs(pos[0] - goal[0]) + abs(pos[1] - goal[1])
```

```
    open_list = []
    heapq.heappush(open_list, (heuristic(start), 0, start, []))
    visited = set()
    while open_list:
        f, g, current, current_path = heapq.heappop(open_list)
        if current in visited:
            continue
        visited.add(current)
        if current == goal:
            path = current_path
            break
        for action in maze.can_move_actions(current):
            dx, dy = direction_map[action]
            next_pos = (current[0] + dx, current[1] + dy)
            if next_pos in visited:
                continue
            new_path = current_path + [action]
            heapq.heappush(open_list, (g + 1 + heuristic(next_pos),
g + 1, next_pos, new_path))
    # ------------------------------------------------------------
----------
    return path
```

- **Q-learning**

```
import random
from QRobot import QRobot
class Robot(QRobot):
    valid_action = ['u', 'r', 'd', 'l']
    def __init__(self, maze, alpha=0.5, gamma=0.9, epsilon=0.5):
        """
        初始化 Robot 类
        :param maze:迷宫对象
        """
        self.maze = maze
        self.state = None
        self.action = None
        self.alpha = alpha
        self.gamma = gamma
        self.epsilon = epsilon  # 动作随机选择概率
```

```
        self.q_table = {}
        self.maze.reset_robot()  # 重置机器人状态
        self.state = self.maze.sense_robot()  # state为机器人当前状态
        if self.state not in self.q_table:  # 如果当前状态不存在，则为 Q
表添加新列
            self.q_table[self.state] = {a: 0.0 for a in
self.valid_action}
    def train_update(self):
        """
        以训练状态选择动作，并更新相关参数
        :return :action, reward 如："u", -1
        """
        self.state = self.maze.sense_robot()  # 获取机器人当初所处迷宫位
置
        # 检索Q表，如果当前状态不存在则添加进入Q表
        if self.state not in self.q_table:
            self.q_table[self.state] = {a: 0.0 for a in
self.valid_action}
        action = random.choice(self.valid_action) if
random.random() < self.epsilon else max(self.q_table[self.state],
key=self.q_table[self.state].get)  # action为机器人选择的动作
        reward = self.maze.move_robot(action)  # 以给定的方向移动机器
人,reward为迷宫返回的奖励值
        next_state = self.maze.sense_robot()  # 获取机器人执行指令后所处
的位置
        # 检索Q表，如果当前的next_state不存在则添加进入Q表
        if next_state not in self.q_table:
            self.q_table[next_state] = {a: 0.0 for a in
self.valid_action}
        # 更新 Q 值表
        current_r = self.q_table[self.state][action]
        update_r = reward + self.gamma *
float(max(self.q_table[next_state].values()))
        self.q_table[self.state][action] = self.alpha *
self.q_table[self.state][action] +(1 - self.alpha) * (update_r -
current_r)
        self.epsilon *= 0.5  # 衰减随机选择动作的可能性
        return action, reward
    def test_update(self):
        """
        以测试状态选择动作，并更新相关参数
        :return :action, reward 如："u", -1
```

```
        """
        self.state = self.maze.sense_robot()  # 获取机器人现在所处迷宫位
置
        # 检索Q表，如果当前状态不存在则添加进入Q表
        if self.state not in self.q_table:
            self.q_table[self.state] = {a: 0.0 for a in
self.valid_action}
        action =
max(self.q_table[self.state],key=self.q_table[self.state].get)  #
选择动作
        reward = self.maze.move_robot(action)  # 以给定的方向移动机器人
        return action, reward
```

四、实验结果

平台测试结果

测试点	状态	时长	结果
测试基础搜索算法	✓	2s	恭喜, 完成了迷宫
测试强化学习算法(初级)	✓	1s	恭喜, 完成了迷宫
测试强化学习算法(中级)	✓	1s	恭喜, 完成了迷宫
测试强化学习算法(高级)	✓	2s	恭喜, 完成了迷宫

五、总结

在本次实验中，我分别实现了基础搜索算法和强化学习算法来解决机器人自动走迷宫问题。基础算法方面选择了A*搜索算法，通过维护开放列表优先扩展最有希望的节点，结合曼哈顿距离启发式函数，有效缩小搜索范围。实验证明该算法能够在已知地图中高效找到最短路径，但随着迷宫规模增大，存储空间需求成为主要制约因素。

强化学习方面采用了Q-learning算法，设计了包含状态动作值函数的Q表。通过ε-greedy策略平衡探索与利用，在训练过程中动态更新Q值。随着训练轮次增加，算法逐渐收敛到有效策略。但学习效率受参数设置影响显著，过高的ε值导致训练后期路径选择震荡。

实验让我深刻认识到：启发式函数设计对搜索算法性能起关键作用，而强化学习中探索策略的衰减方案直接影响最终性能。未来将尝试深度Q网络处理更大状态空间，并研究多目标寻路问题，进一步探索两种方法的适用边界。