程序报告

学号：2211312　　姓名：贾景顺

一、问题重述

在本实验中，分别采用**基础搜索算法**和**深度强化学习算法**来解决机器人自动走迷宫的问题。迷宫环境中，机器人从起点出发，需要通过复杂的路径找到目标出口。在每个位置，机器人可以选择四个基本移动方向：向上（'u'）、向右（'r'）、向下（'d'）或向左（'l'）。系统会根据不同的移动结果给予相应的奖励反馈：撞墙会获得惩罚性负奖励，成功到达出口会获得高额正奖励，而普通移动则会获得中性或轻微负奖励。

在**基础搜索算法**方面，选择了 **A*算法**进行实现。A*算法是一种经典的启发式搜索算法，它通过综合评估当前路径的实际代价和到目标的预估代价（使用曼哈顿距离作为启发式函数），能够高效地找到最优路径。该算法的输入是迷宫地图信息，输出是从起点到目标点的最优动作序列。

DQN 算法传统的值迭代算法不同，DQN 通过深度神经网络来近似 Q 值函数，不仅考虑即时奖励，还综合考虑长期回报。算法使用经验回放机制来提高数据利用率，并通过目标网络稳定训练过程。DQN 的核心在于建立状态-动作价值的准确估计，并通过梯度下降不断优化网络参数，最终使机器人学会最优的移动策略。

二、设计思想

在 A*算法的设计思想基于启发式搜索，通过结合路径实际代价和启发式估计来高效寻找最优路径。在实现中，move_map 定义了移动方向，manhattan_distance 函数作为启发式函数估算到目标的距离。核心函数 my_search 维护开放优先队列和关闭集合，每次扩展 f 值最小的节点，利用 can_move_actions 获取合法移动方向，通过 is_hit_wall 检测碰撞，最终回溯生成路径。算法通过平衡探索和启发式引导，在保证最优性的同时提高了搜索效率。

DQN 算法采用深度强化学习框架，通过神经网络近似 Q 函数来学习长期最优策略。Robot 类中，_build_network 构建了双网络结构（评估网络和目标网络），_choose_action 实现 ε-greedy 策略平衡探索与利用，_learn 函数使用经验回放和 TD 误差进行网络训练。train_update 处理状态转移并控制学习节奏，test_update 则执行纯策略验证。该设计通过分离目标网络稳定训练，结合经验回放提高数据利用率，使智能体能在复杂迷宫中学习到考虑长期回报的移动策略。实验中通过对奖励函数、alpha、gamma、epsilon0 等参数的调节，最终顺利通过了各检测点。

三、代码内容

```
#A*算法实现
import numpy as np
import heapq
def my_search(maze):
    """
    使用 A*算法搜索迷宫路径
```

```
    :param maze: 迷宫对象
    :return: 到达目标点的路径 如：["u","u","r",...]
    """
    # 机器人移动方向
    move_map = {
        'u': (-1, 0),    # up
        'r': (0, +1),    # right
        'd': (+1, 0),    # down
        'l': (0, -1),    # left
    }

    def manhattan_distance(loc1, loc2):
        return abs(loc1[0] - loc2[0]) + abs(loc1[1] - loc2[1])

    start = maze.sense_robot()
    destination = maze.destination
    open_list = []
    closed_set = set()
    initial_h = manhattan_distance(start, destination)
    heapq.heappush(open_list, (initial_h, 0, start, []))    # (f, g, position, path)
    visited = np.zeros(maze.maze_data.shape[:2], dtype=bool)

    while open_list:
        f, g, current_pos, path = heapq.heappop(open_list)
        if current_pos == destination:
            return path
        if visited[current_pos]:
            continue
        visited[current_pos] = True
        closed_set.add(current_pos)

        # 检查所有可能的移动方向
        for action in maze.can_move_actions(current_pos):
            move = move_map[action]
            new_pos = (current_pos[0] + move[0], current_pos[1] + move[1])

            rows, cols = maze.maze_data.shape[:2]
            if not (0 <= new_pos[0] < rows and 0 <= new_pos[1] < cols):
                continue

            if maze.is_hit_wall(current_pos, action):
                continue

            if visited[new_pos]:
```

```
                continue

            new_g = g + 1
            new_h = manhattan_distance(new_pos, destination)
            new_f = new_g + new_h
            heapq.heappush(open_list, (new_f, new_g, new_pos, path + [action]))


    return []
```

==========

```
#DQN 算法实现
from QRobot import QRobot
import random
class Robot(QRobot):
    def __init__(self, maze,alpha=0.5,gamma=0.95,epsilon0=0.8):
        """
        初始化 Robot 类
        :param maze:迷宫对象
        """
        super(Robot, self).__init__(maze)
        self.maze = maze
        self.maze.reward = {
                "hit_wall": -3,
                "destination": 10,
                "default": -0.1,
        }
        self.alpha = alpha
        self.gamma = gamma
        self.epsilon0 = epsilon0
        self.epsilon = epsilon0
    def update_parameter(self):
        """
        衰减随机选择动作的可能性
        """
        self.t += 1
        if self.epsilon < 0.03:
            self.epsilon = 0.03
        else:
            self.epsilon -= self.t * 0.1
        return self.epsilon
    def train_update(self):
        """
        以训练状态选择动作并更新 Deep Q network 的相关参数
        :return :action, reward 如："u", -1
```

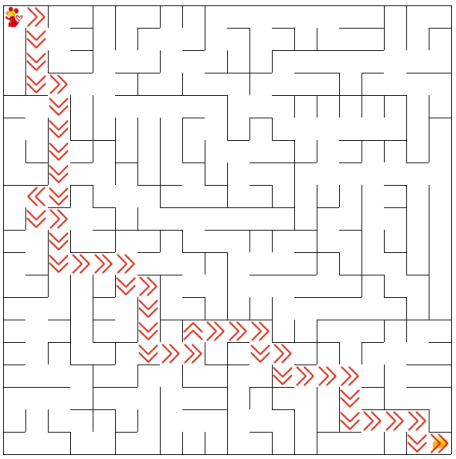
```
        """
        self.state=self.sense_state()
        self.create_Qtable_line(self.state)
        if(random.random()<self.epsilon):
            action=random.choice(self.valid_action)
        else:
            action=max(self.q_table[self.state], key=self.q_table[self.state].get)
        reward = self.maze.move_robot(action)
        next_state = self.sense_state()
        self.create_Qtable_line(next_state)
        self.update_Qtable(reward, action, next_state)
        self.update_parameter()
        return action, reward
    def test_update(self):
        """
        以测试状态选择动作并更新 Deep Q network 的相关参数
        :return : action, reward  如："u", -1
        """
        self.state = self.sense_state()
        self.create_Qtable_line(self.state)
        action = max(self.q_table[self.state],key=self.q_table[self.state].get)
        reward = self.maze.move_robot(action)
        return action, reward
```

四、实验结果

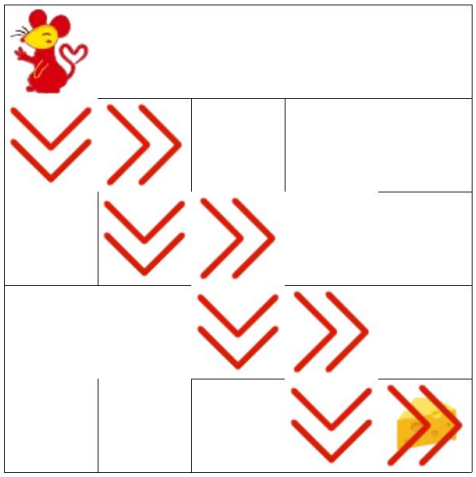
如图所示，A*算法以及 DQN 算法均能通过各测试点

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

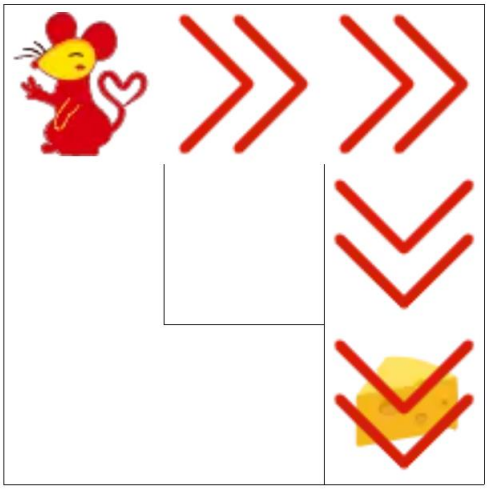
基础搜索算法 (Victory)

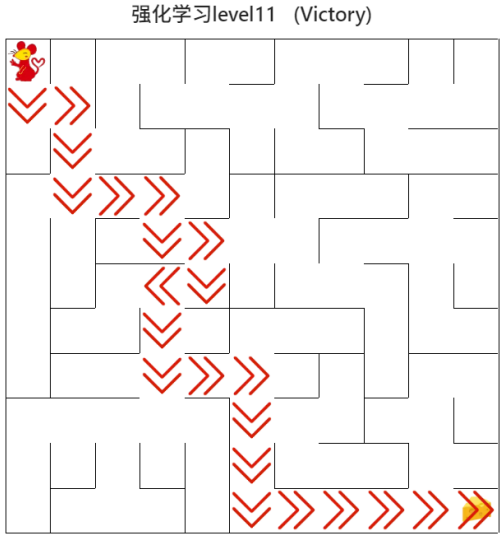


强化学习level5 (Victory)



强化学习level3 (Victory)





五、总结

在本次实验中，通过对机器人走迷宫模型的学习，完成了深度优先搜索算法，并学习了强化学习中的 Q-Learning 算法，成功加以运用在实验中来。强化学习能够广泛应用于解决实际生活中常见的决策问题，作为一种通用的策略学习框架，向人们展示了其强大的能力和应用前景。

在这学期人工智能实验的学习中，我深刻了解了人工智能的各种算法，以及部分常用的模型框架。这为我入门人工智能领域打下了良好的基础，在未来我也会继续努力。