第六次实训程序报告

学号: 2310764 姓名: 王亦辉

1 问题重述

本实验要求实现一个机器人自动走迷宫的程序，分别使用基础搜索算法和 Deep Q-Learning 算法两种方法。机器人从左上角起点出发，目标是到达右下角的出口。在任意位置，机器人可以向上、右、下、左移动。根据移动结果，会得到不同的奖励，包括撞墙、正常移动和到达出口等情况。最终目标是使机器人能成功走出迷宫。

2 设计思想

深度优先搜索（DFS）

DFS 算法通过栈的方式从起点出发不断深入迷宫路径，直到到达终点或无路可走为止。具体实现上，使用了搜索树结构来记录路径信息，每个节点包含当前位置、父节点与到达动作。在搜索过程中，用 is_visit_m 矩阵记录已访问节点，防止重复访问和死循环。

优点：实现简单，能快速探索完整路径。

缺点：容易陷入深层无效路径，缺乏最优性保证，对复杂迷宫效率低。

优化方向：使用启发函数（如 A*）替代纯 DFS 以提升路径质量；引入路径剪枝与回溯优化。

Deep Q-Learning

DQN 算法引入了强化学习思想。通过学习 Q 值函数来估计在当前状态下采取各动作的价值，并通过不断试错（探索与利用）优化策略，使机器人在迷宫中学会高效到达目标。

核心实现：采用 Q 表（Tabular Q-Learning）存储状态-动作值；每次根据 ε-greedy 策略决定动作，兼顾探索与利用；逐步衰减 ε 值，训练后期提高利用程度，提升收敛速度。

存在问题与可改进方向：奖励稀疏，训练早期机器人缺乏有效反馈，导致学习缓慢，可考虑引入 shaping reward（如靠近目标给予正向激励）。

3 代码内容

（能体现解题思路的主要代码，有多个文件或模块可用多个隔开，必填）

深度优先搜索（DFS）

```
# 导入相关包
import os
```

```python
import random
import numpy as np
from Maze import Maze
from Runner import Runner
from QRobot import QRobot
from ReplayDataSet import ReplayDataSet
from torch_py.MinDQNRobot import MinDQNRobot as TorchRobot # PyTorch版本
# from keras_py.MinDQNRobot import MinDQNRobot as KerasRobot # Keras版本
import matplotlib.pyplot as plt



import numpy as np

# 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
}


# 迷宫路径搜索树
class SearchTree(object):

    def __init__(self, loc=(), action='', parent=None):
        """
        初始化搜索树节点对象
        :param loc: 新节点的机器人所处位置
        :param action: 新节点的对应的移动方向
        :param parent: 新节点的父辈节点
        """
        self.loc = loc  # 当前节点位置
        self.to_this_action = action  # 到达当前节点的动作
        self.parent = parent  # 当前节点的父节点
        self.children = []  # 当前节点的子节点

    def add_child(self, child):
        """
        添加子节点
        :param child:待添加的子节点
```

```
        """
        self.children.append(child)

    def is_leaf(self):
        """
        判断当前节点是否是叶子节点
        """
        return len(self.children) == 0


def expand(maze, is_visit_m, node):
    """
    拓展节点，即为当前节点添加执行合法动作后到达的子节点。
    与BFS中的版本相同，关键在于 is_visit_m 的使用。
    :param maze: 迷宫对象
    :param is_visit_m: 记录迷宫每个位置是否访问的矩阵
    :param node: 待拓展的节点
    """
    can_move = maze.can_move_actions(node.loc)
    for a in can_move:
        new_loc = tuple(node.loc[i] + move_map[a][i] for i in range(2))
        # 仅当新位置未被标记为"已处理"时，才创建并添加子节点
        if not is_visit_m[new_loc]:
            child = SearchTree(loc=new_loc, action=a, parent=node)
            node.add_child(child)


def back_propagation(node):
    """
    回溯并记录节点路径
    :param node: 待回溯节点
    :return: 回溯路径
    """
    path = []
    while node.parent is not None:
        path.insert(0, node.to_this_action)
        node = node.parent
    return path


def my_search(maze):
    """
    使用深度优先搜索算法实现迷宫路径查找。
    :param maze: 迷宫对象
```

```
    :return :到达目标点的路径 如: ["u","u","r",...]
    """

    path = [] # 最终路径

    # -----------------请实现你的算法代码--------------------------------
    start_loc = maze.sense_robot()
    destination_loc = maze.destination

    # 创建根节点
    root = SearchTree(loc=start_loc, action='', parent=None)

    # 用于DFS的栈，存储待访问的 SearchTree 节点
    stack = [root]

    # 访问标记矩阵（h，w 是迷宫的高和宽）
    # 假设 maze.maze_data 存在并且可以获取其形状
    h, w, _ = maze.maze_data.shape
    is_visit_m = np.zeros((h, w), dtype=int) # 0 表示未访问（未被处理），1 表示已访问（已被处理）

    while stack: # 当栈不为空时循环
        current_node = stack.pop() # 从栈顶取出一个节点（LIFO）

        # 如果当前节点位置已经被处理过（即is_visit_m中标记为1），则跳过
        # 这是为了处理一个位置可能通过不同路径被多次加入栈中的情况
        if is_visit_m[current_node.loc] == 1:
            continue

        # 标记当前节点位置为已访问（已处理）
        is_visit_m[current_node.loc] = 1

        # 检查是否到达目标点
        if current_node.loc == destination_loc:
            path = back_propagation(current_node) # 回溯路径
            return path # 找到路径，返回

        # 拓展当前节点。
        # expand 函数会检查 is_visit_m，仅为未被标记为1的新位置创建子节点。
        # 子节点会被添加到 current_node.children 列表中。
        expand(maze, is_visit_m, current_node)

```

```
        # 将子节点加入栈中。
        # 为了使得探索顺序与某些递归DFS实现类似（例如，优先探索"向上"的完整路径），
        # 可以将子节点逆序加入栈中。
        for child_node in reversed(current_node.children):
            # expand 函数保证了 child_node.loc 在创建时 is_visit_m[child_node.loc] 是 0。
            # 所以这里直接添加即可。循环开始处的 is_visit_m 检查会处理重复问题。
            stack.append(child_node)

    # ------------------------------------------------------------------
    # 如果栈为空，说明没有找到到达目标点的路径
    return path # 返回空列表


```

Deep Qlearning

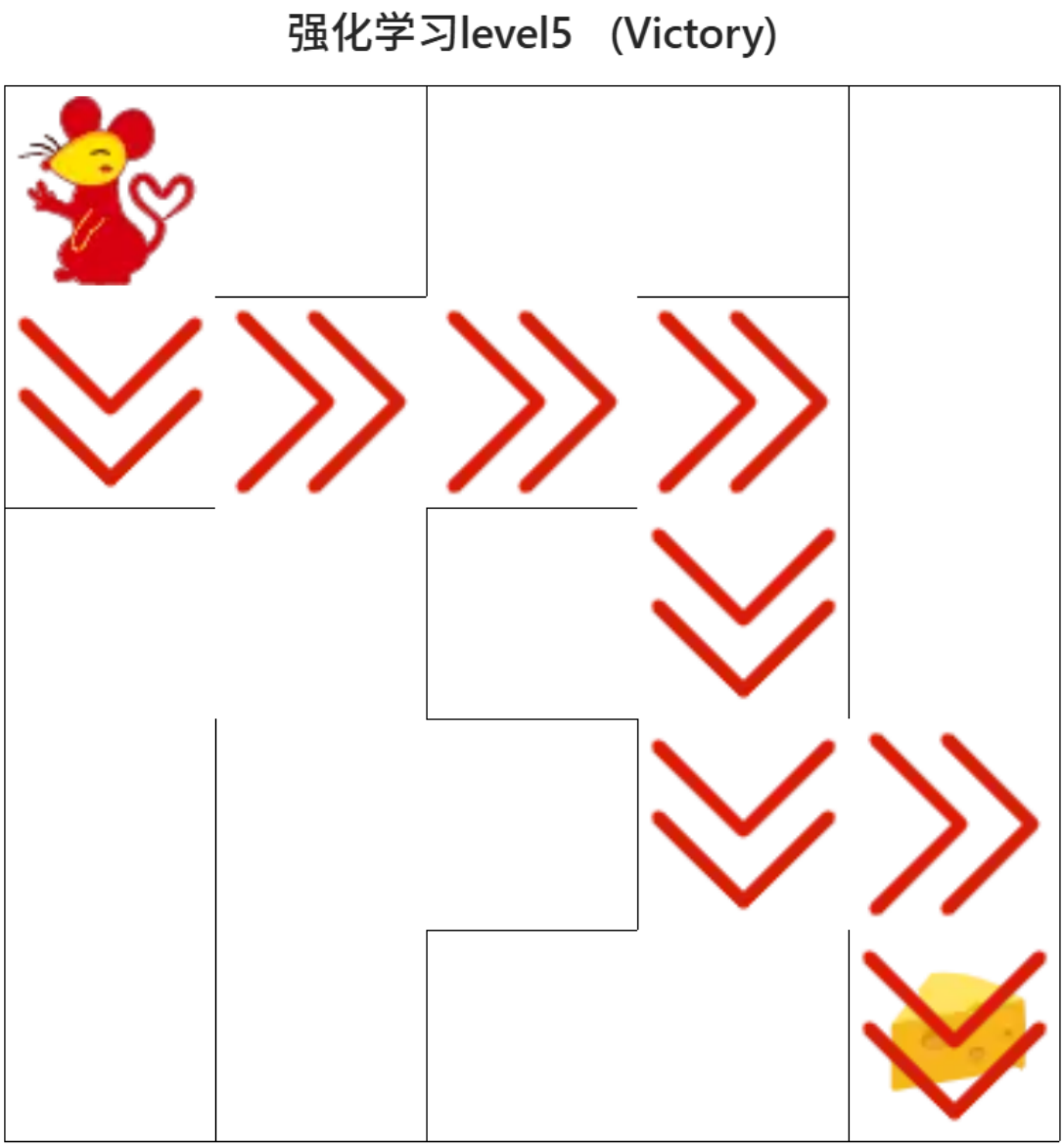
```
from QRobot import QRobot

class Robot(QRobot):

    valid_action = ['u', 'r', 'd', 'l']

    def __init__(self, maze, alpha=0.5, gamma=0.9, epsilon=0.5):
        self.maze = maze
        self.alpha = alpha
        self.gamma = gamma
        self.epsilon = epsilon
        self.q_table = {}
        self.maze.reset_robot()
        self.state = self.maze.sense_robot()
        self._init_state_if_needed(self.state)

    def _init_state_if_needed(self, state):
        if state not in self.q_table:
            self.q_table[state] = {a: 0.0 for a in self.valid_action}

    def train_update(self):
        self.state = self.maze.sense_robot()
        self._init_state_if_needed(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.maze.sense_robot()
        self._init_state_if_needed(next_state)

        current_q = self.q_table[self.state][action]
        max_next_q = max(self.q_table[next_state].values())
        target = reward + self.gamma * max_next_q

        self.q_table[self.state][action] += self.alpha * (target -
current_q)

        self.epsilon = max(0.01, self.epsilon * 0.5)

        return action, reward

    def test_update(self):
        self.state = self.maze.sense_robot()
        self._init_state_if_needed(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

```

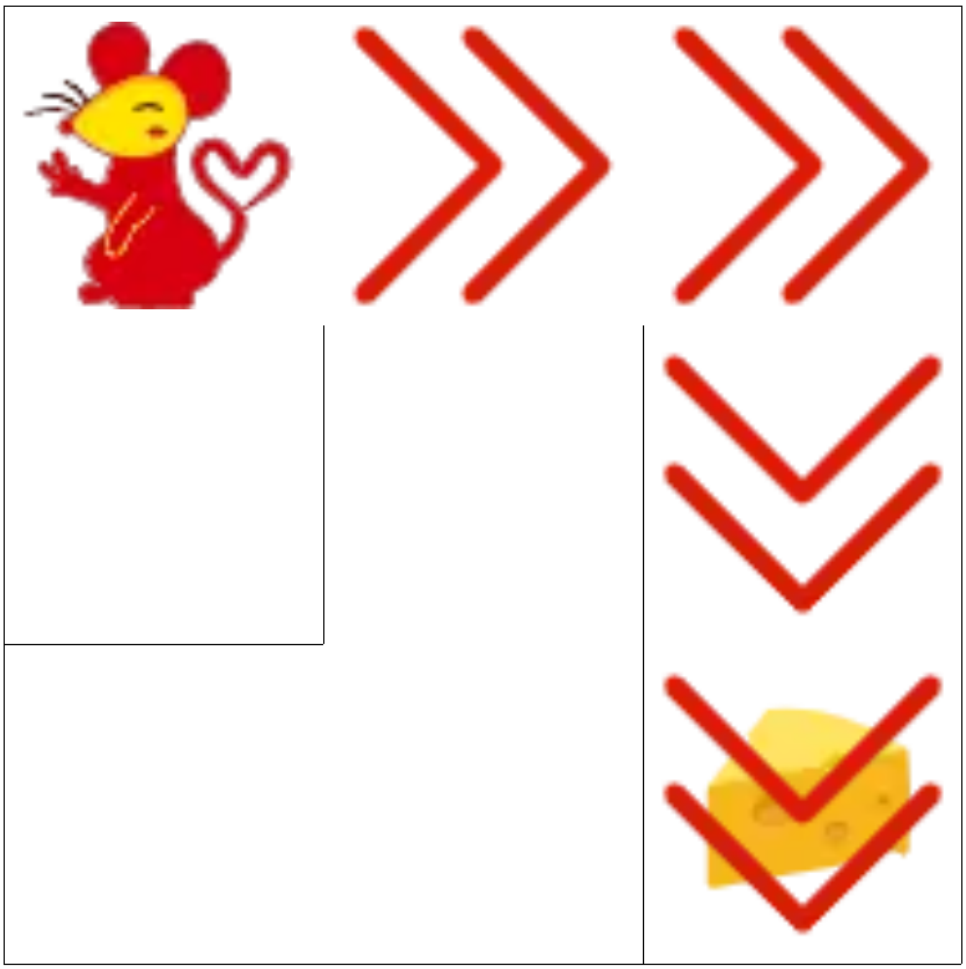
4 实验结果

测试点	状态	时长	结果
测试强化学习算法(中级)	✓	0s	恭喜, 完成了迷宫
测试强化学习算法(初级)	✓	0s	恭喜, 完成了迷宫
测试强化学习算法(高级)	✓	0s	很遗憾, 未能走完迷宫
测试基础搜索算法	✓	0s	恭喜, 完成了迷宫

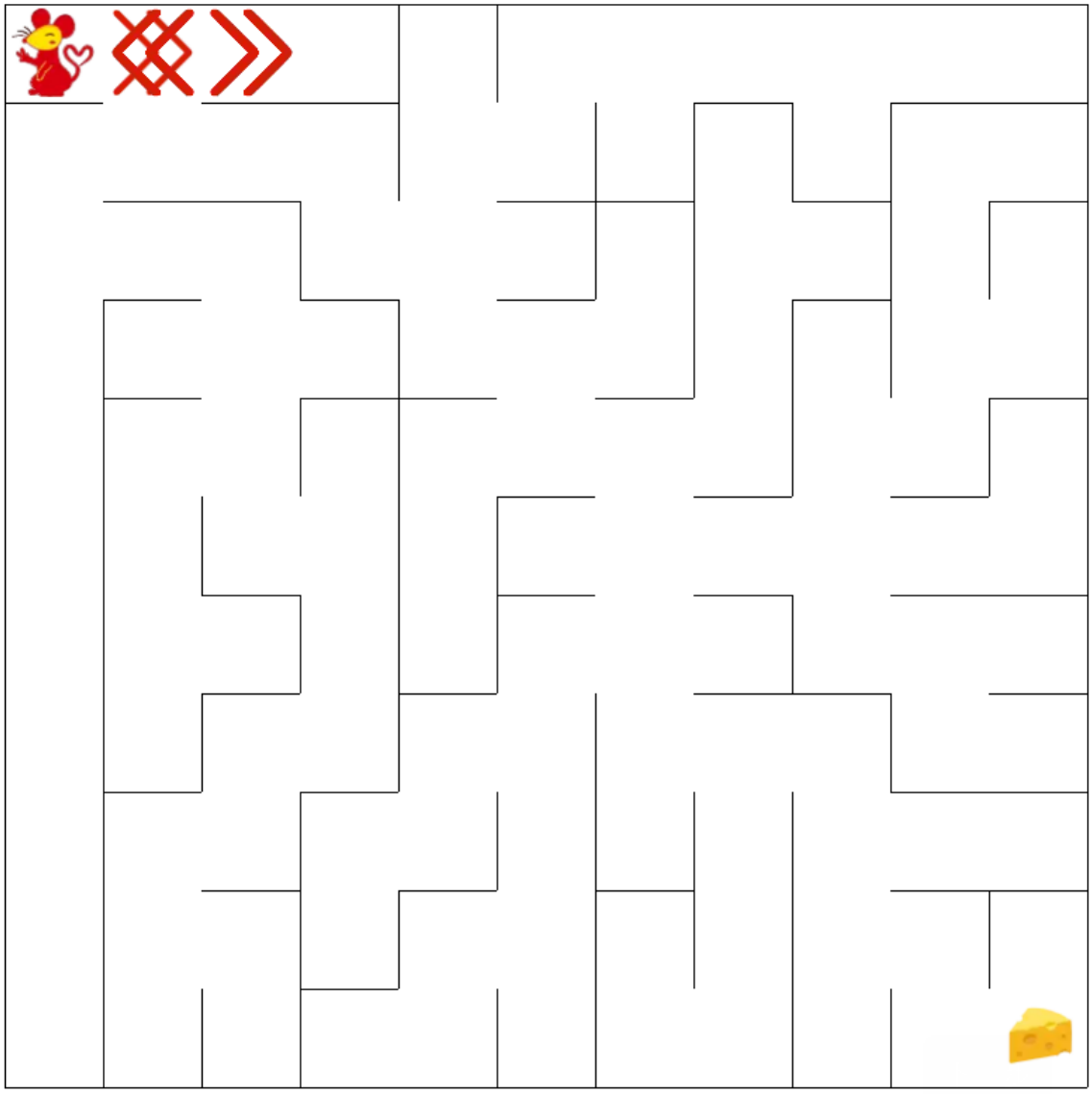
四个迷宫的结果

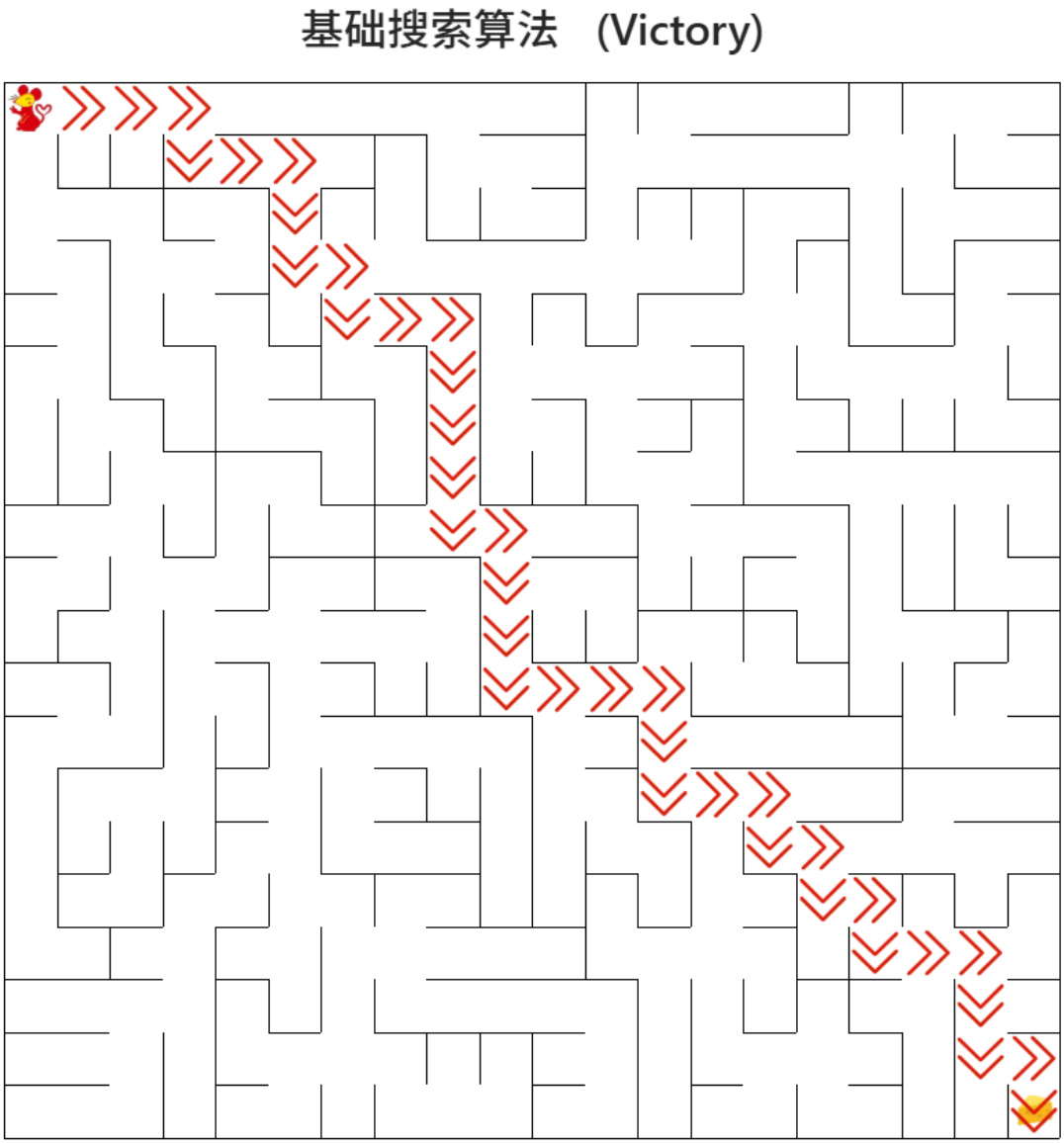


强化学习level3 （Victory）



强化学习level11 (Defeat)





5 总结

本次实验目标是使用基础的广度优先搜索算法与 Deep Q-Learning 实现机器人自动走迷宫。从结果来看，使用 Deep Q-Learning 算法能够较稳定地找到出口，整体达到了实验预期目标。

在实现过程中，主要难点在于对 Q 表的更新逻辑掌握、合理调整参数（如学习率、折扣因子、探索率）以及避免陷入局部最优。特别是在训练初期，机器人容易因为奖励稀疏而学习速度缓慢。为此，通过 ε 衰减策略，使探索逐渐转向利用，提升了最终性能。