机器人自动走迷宫
程序报告

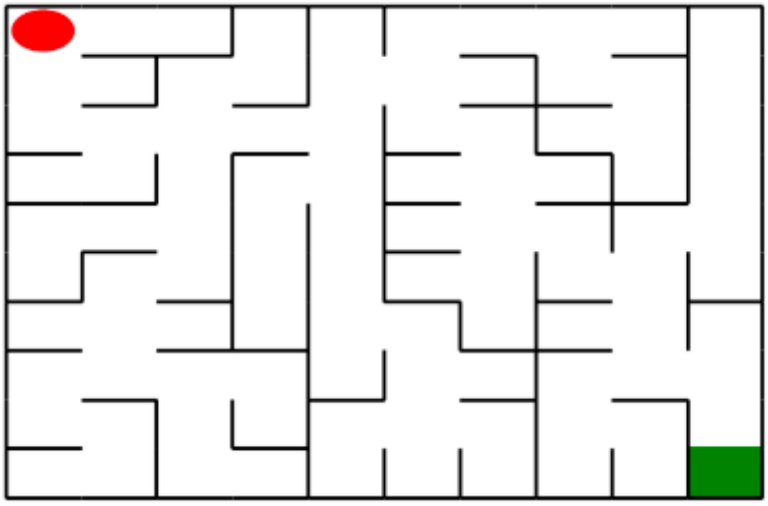
姓名： 牛钟仪　　学号： 3210102328

学院（系）专业：计算机科学与技术

1 算法描述

1.1. 问题描述

在本实验中，要求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。



如上图所示，左上角的红色椭圆既是起点也是机器人的初始位置，右下角的绿色方块是出口。

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

- 需要您分别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。

1.2. 算法展示

1.2.1. 基础搜索算法实现

```
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()
    root = SearchTree(loc=start)
    stack = [root]
    h, w, _ = maze.maze_data.shape
    is_visit_m = np.zeros((h, w), dtype=np.int)  # 标记迷宫的各个位置是否被访问过
    #top = 0
    while True:
        current_node = stack[-1]  # 栈顶元素作为当前节点

        if current_node.loc == maze.destination:  # 到达目标点
            path = back_propagation(current_node)
            break

        if current_node.is_leaf() and is_visit_m[current_node.loc] == 0:  # 如果该点存在叶子节点且未拓展
            is_visit_m[current_node.loc] = 1  # 标记该点已拓展
            expand(maze, is_visit_m, current_node)
            for child in current_node.children:
                stack.append(child)  # 叶子节点入栈
        else:
            stack.pop()  # 如果无路可走则出栈
    # ----------------------------------------------------------------------
    return path
```

1.2.2. Deep QLearning 算法实现

```python
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
    """
    # -----------------请实现你的算法代码--------------------------------------
    self.state = self.maze.sense_robot()  # 获取机器人当初所处迷宫位置
    # 当前状态添加进Q表
    if self.state not in self.q_table:
        self.q_table[self.state] = {a: 0.0 for a in self.valid_action}
    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()
    #当前的next_state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
    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
```

```python
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
    """
    # -----------------请实现你的算法代码--------------------------------------
    self.state = self.maze.sense_robot()  # 获取机器人当初所处迷宫位置
    #当前状态添加进入Q表
    if self.state not in self.q_table:
        self.q_table[self.state] = {a: 0.0 for a in self.valid_action}
    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()
    #当前的next_state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
    current_r = self.q_table[self.state][action]
    update_r = reward + self.gamma * float(max(self.q_table[next_state].values()))
    self.epsilon *= 0.5  # 衰减随机选择动作的可能性
    # ------------------------------------------------------------------------
    return action, reward
```

1.2.3. 代码解释

1.2.3.1. 基础搜索算法

输入：迷宫对象

输出：到达目标点的路径 如：["u","u","r",...]

在基础搜索算法中我选择了深度优先搜索，数据结构使用了栈。核心思路是：先走一条路，一直走到不能

走为止，在回溯到上一个状态选择另一个方向，直到找到出口。

我们在到达每一个节点时，将其未标记的子节点全部压入栈中，但是每次都取栈顶元素作为当前节点，若这个点没有可以走的子节点时将其出栈，这样便实现了先一条路走，再回溯的效果。

1.2.3.2. Deep QLearning 算法

输入：无

输出：根据当前的状态所选取的 action 和这一 action 所对应的 reward

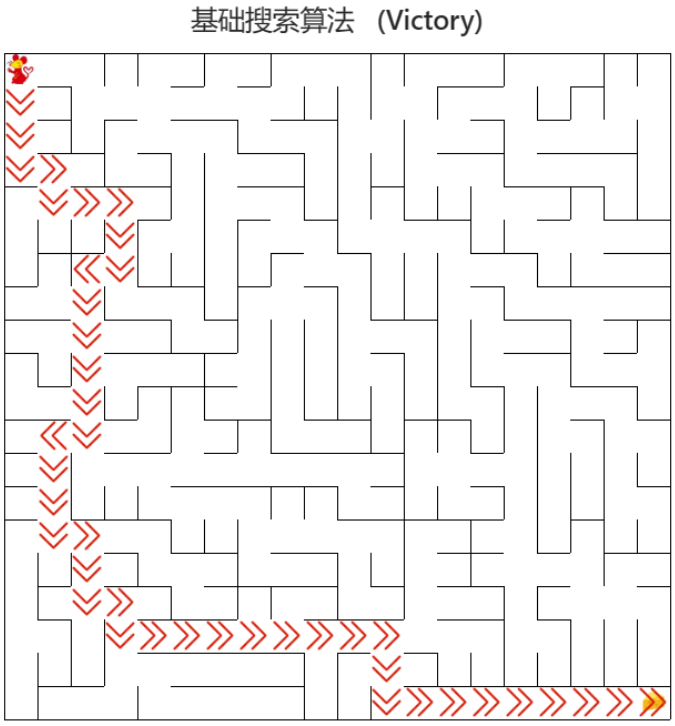
Train_update 函数中，首先检查 Q 表，将当前状态加入 Q 表中，然后选择 action，action 有两种选择，当 random.random() < self.epsilon 时 action 采用随机选取，否则 action 就根据当前状态选取，这样的目的是在训练初期让 action 尽量尝试更多的路线，而在训练一段时间后形成一个相对固定的路线。选取后在 maze 中进行移动，更新位置及 Q 表，然后降低 epsilon

test_update 函数中，只有在 action 确定后，不在 maze 中做真正的移动外，其余与 train_update 函数相同

2 算法实验结果与分析

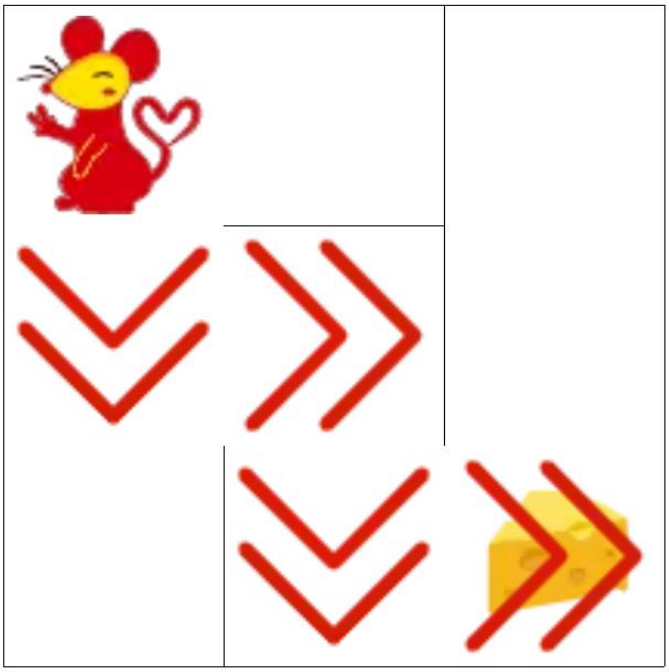
2.1 实验结果

2.1.1 基础算法

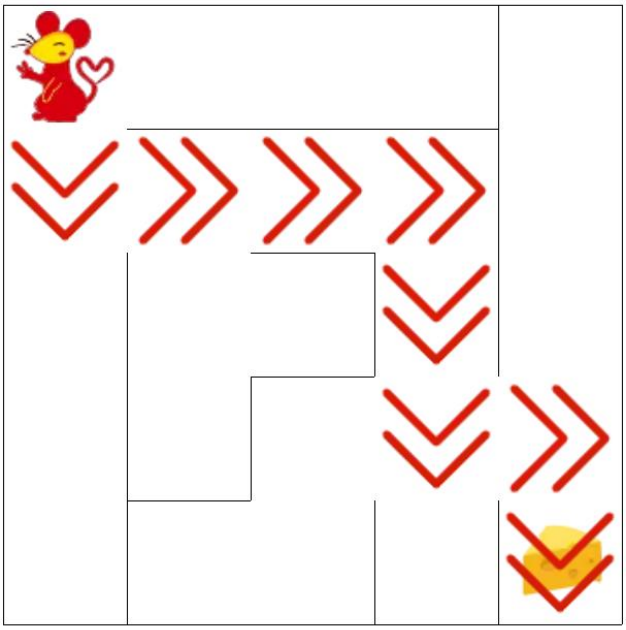


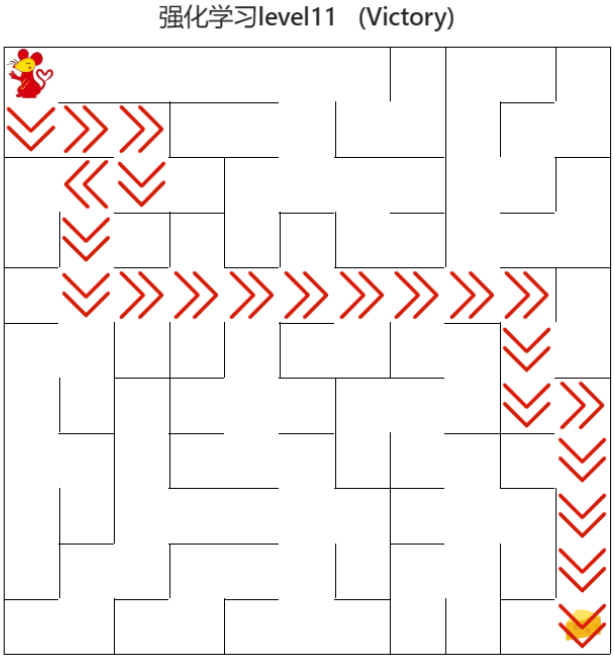
2.1.2 Deep QLearning 算法

强化学习level3 (Victory)



强化学习level5 (Victory)





2.2 结果分析

结果符合预期，算法都成功走完了迷宫

3 算法进一步研究展望

个人认为随机选择路径可以进行改良，可以设置一个参数，这个参数反映了一个路径的尝试次数和程度，在训练的初期，优先选择这个参数较低的 action