

程序报告

学号：2211290

姓名：姚知言

一、问题重述

分别使用基础搜索算法和 Deep QLearning 算法实现机器人，使机器人自动走到迷宫的出口。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。基础搜索算法：选择深度优先算法进行实现。输入迷宫，输出到达目标点的路径。QLearning：与策略迭代（Policy Iteration）算法不同，值迭代算法会计算每个“状态-动作”的值（Value）或是效用（Utility），然后在执行动作的时候，会设法最大化这个值。因此，对每个状态值的准确估计，是值迭代算法的核心。考虑最大化动作的长期奖励，即不仅考虑当前动作带来的奖励，还会考虑动作长远的奖励。

二、设计思想

在深度优先算法实现中，我使用栈的结构，但并没有完全照搬传统的深度优先策略。在访问一个新的结点的时候，我将栈顶弹出之后将这个栈的所有子结点都压入栈。这样做的好处是在保证深度优先算法顺序的前提下，无需频繁的自底向上频繁访问，也无需维护子结点是否已经访问的问题。虽然这样的话没有办法直接存储查找到的路径，但每个结点都存储了父结点的位置，相比于频繁调用的开销，我认为单次的自底向上的开销还是可以接受的。在 QLearning 的实现中，通过控制 α, γ, ϵ 的参数，调整 `maze.reward`，重新定义 `update_parameter`，并在 `train_update` 和 `test_update` 中实现来实现效果较好的 QLearning 算法。在起初的实验中，常常无法通过高级算法，甚至中途在部分修改后，出现无法通过较低级算法的情况（比如当撞墙惩罚过小的时候， ϵ 过大的时候）。最终经过无数次的参与测试，终于顺利通过了所有的测试。

三、代码内容

```
#深度优先算法实现
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):
    """
    拓展叶子节点，即为当前的叶子节点添加执行合法动作后到达的子节点
    :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: 待拓展的叶子节点
    """
    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))
        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):
    """
    选择深度优先算法实现
    :param maze: 迷宫对象
    :return :到达目标点的路径 如: ["u","u","r",...]
    """
    path = []
    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
while True:
    current_node = stack[0]
    is_visit_m[current_node.loc] = 1
    if current_node.loc == maze.destination:
        path = back_propagation(current_node)
        break
    if current_node.is_leaf():
        expand(maze, is_visit_m, current_node)
    stack.pop(0)
    for child in current_node.children:
        stack.insert(0, child)
return path

```

#QLearning 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:

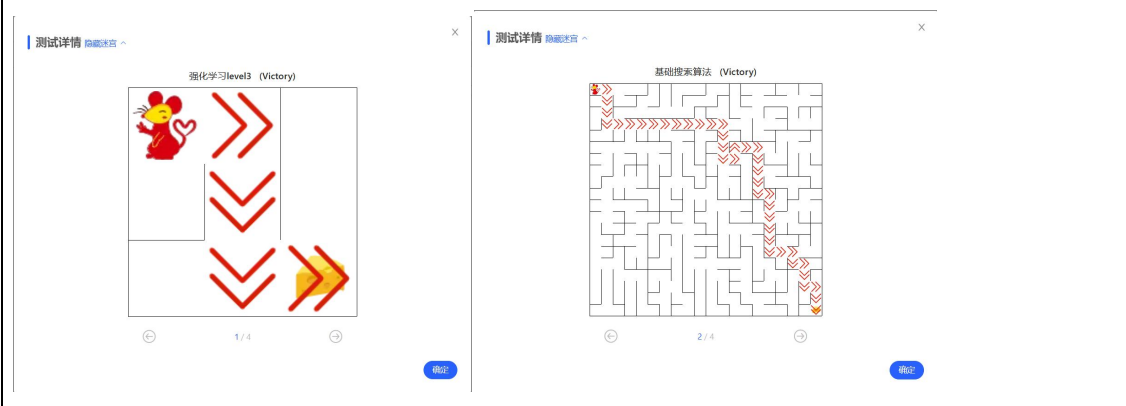
```

        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

```

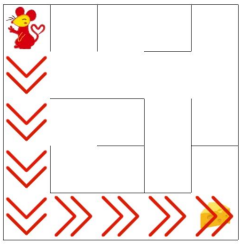
四、实验结果

如图所示，所有迷宫成功完成，用时 1s。



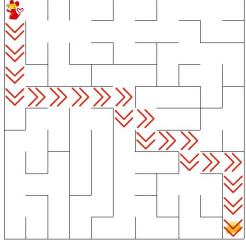
测试详情 隐藏迷宫

强化学习level5 (Victory)



3 / 4

强化学习level11 (Victory)



4 / 4

确定

测试详情 展示迷宫

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

确定

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。