机器人自动走迷宫程序报告

问题重述

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。

- 机器人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

设计思想

深度优先搜索算法：

深度优先遍历图的方法是，从图中某顶点 v 出发：

- 访问顶点 v；
- 依次从 v 的未被访问的邻接点出发，对图进行深度优先遍历；直至图中和 v 有路径相 通的顶点都被访问；
- 若此时图中尚有顶点未被访问，则从一个未被访问的顶点出发，重新进行深度优先遍历，直到图中所有顶点均被访问过为止。

广度优先搜索算法：

广度优先搜索使用队列（queue）来实现，整个过程也可以看做一个倒立的树形：

- 把根节点放到队列的末尾。
- 每次从队列的头部取出一个元素，查看这个元素所有的下一级元素，把它们放到队列 的末尾。并把这个元素记为它下一级元素的前驱。
- 找到所要找的元素时结束程序。 4) 如果遍历整个树还没有找到，结束程序

Deep Q Learning:

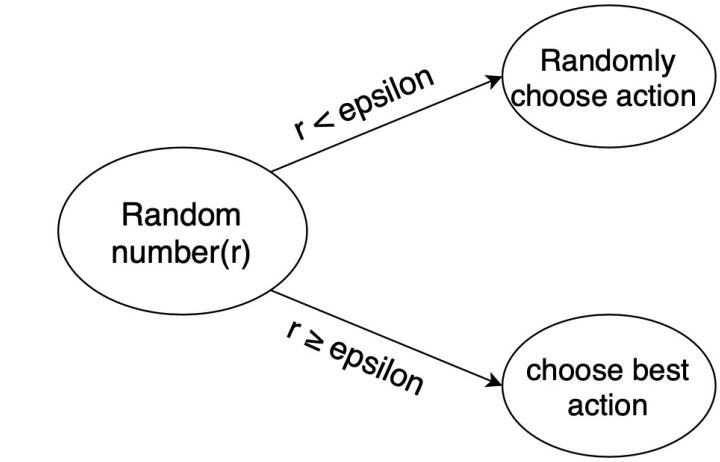
QLearning算法是一个值迭代（Value Iteration）算法。与策略迭代（Policy Iteration）算法不同，值迭代算法会计算每个”状态“或是”状态-动作“的值（Value）或是效用（Utility），然后在执行动作的时候，会设法最大化这个值。

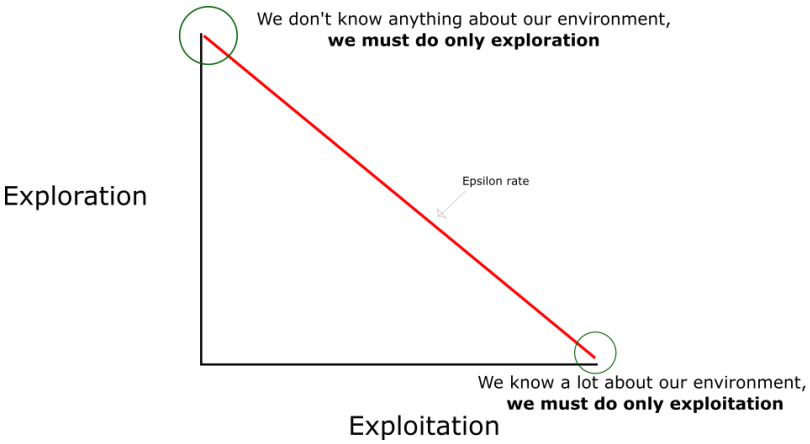
根据QLearning，会尽可能地让机器人在每次选择最优的决策，来最大化长期奖励。但是这样做有如下的弊端：

1. 在初步的学习中，Q 值是不准确的，如果在这个时候都按照 Q 值来选择，那么会造成错误。
2. 学习一段时间后，机器人的路线会相对固定，则机器人无法对环境进行有效的探索。

因此需要一种办法，来解决如上的问题，增加机器人的探索。
通常会使用 **epsilon-greedy** 算法：

1. 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。
2. 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。





代码内容

深度优先

对当前节点各个方向进行遍历，遇到可以前进的情况则前进到下一节点进行遍历，直到到达终点或没有可走的道路。

```
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)
    queue = [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
    path = []  # 记录路径
    while True:
        current_node = queue[0]
        is_visit_m[current_node.loc] = 1  # 标记当前节点位置已访问

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

```
        if current_node.is_leaf():
            expand(maze, is_visit_m, current_node)

        # 入队
        flag=0
        for child in current_node.children:
            if is_visit_m[child.loc]==False:
                queue.append(child)
                flag=1
                break

        if flag==0:
            queue[0]=current_node.parent
        else:
            # 出队
            queue.pop(0)

    # ----------------------------------------------------------------------
-
    return path
```

广度优先

使用队列对所有可走节点存储后进行遍历，每次将队列中所有元素出队，并将下一步可走节点入队。

```
def breadth_first_search(maze):
    """
    对迷宫进行广度优先搜索
    :param maze: 待搜索的maze对象
    """
    start = maze.sense_robot()
    root = SearchTree(loc=start)
    queue = [root]  # 节点队列，用于层次遍历
    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
    path = []  # 记录路径
    while True:
        current_node = queue[0]
        is_visit_m[current_node.loc] = 1  # 标记当前节点位置已访问

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

        if current_node.is_leaf():
            expand(maze, is_visit_m, current_node)

        # 入队
        for child in current_node.children:
            queue.append(child)

        # 出队
        queue.pop(0)

    return path
```

DNQ算法

```
class Robot(QRobot):

    def __init__(self, maze):
        """
        初始化 Robot 类
        :param maze:迷宫对象
        """
        super(Robot, self).__init__(maze)
        self.maze = maze

    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
        """
        action, reward = "r", -1.0

        # -----------------请实现你的算法代码----------------------------------
        self.state = self.sense_state()  # 获取机器人当初所处迷宫位置

        self.create_Qtable_line(self.state)  # 对当前状态，检索Q表，如果不存在则添加进入Q表

        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)  # 选择动作

        reward = self.maze.move_robot(action)  # 以给定的动作（移动方向）移动机器人

        next_state = self.sense_state()  # 获取机器人执行动作后所处的位置

        self.create_Qtable_line(next_state)  # 对当前 next_state ，检索Q表，如果不存在则添加进入Q表

        self.update_Qtable(reward, action, next_state)  # 更新 Q 表 中 Q 值
        self.update_parameter()  # 更新其它参数

        # ----------------------------------------------------------------------

        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
        """
        action, reward = "d", -1.0

```

```
        # -----------------请实现你的算法代码----------------------------------
        self.state = self.sense_state()  # 获取机器人当初所处迷宫位置

        self.create_Qtable_line(self.state)  # 对当前状态，检索Q表，如果不存在则添加进入Q表

        action = max(self.q_table[self.state],
                     key=self.q_table[self.state].get)  # 选择动作

        reward = self.maze.move_robot(action)  # 以给定的动作（移动方向）移动机器人
        # ----------------------------------------------------------------------

        return action, reward
```

实验结果

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)		2s	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	很遗憾, 未能走完迷宫