**程序报告**

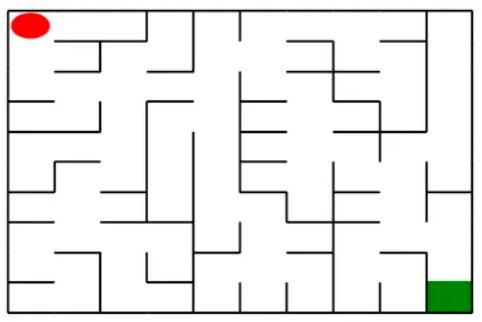
学号： 2013750 姓名：管昀玫

1. **问题重述**

（简单描述对问题的理解，从问题中抓住主干，必填）

====================================================================

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。

1. 撞墙
2. 走到出口
3. 其余情况

需要实现于基础搜索算法和 Deep QLearning 算法的机器人，使机器人自动走到迷宫的出口。

1. **设计思想**

（所采用的方法，有无对方法加以改进，该方法有哪些优化方向（参数调整，框架调整，或者指出方法的局限性和常见问题），伪代码，理论结果验证等… **思考题，非必填**）

====================================================================

**基于基础搜索算法和Deep QLearning算法**

首先通过迷宫类Maze可以随机创建一个迷宫。使用 Maze(maze\_size=size) 来随机生成一个 size \* size 大小的迷宫。机器人可以向上走u，向右走r，向下走d，向左走l。随机移动机器人，并记录下获得的奖励。

**广度优先搜索算法：**

对于迷宫游戏，常见的三种的搜索算法有广度优先搜索、深度优先搜索和最佳优先搜索（A\*)。我们使用广度优先搜索。

算法具体步骤：

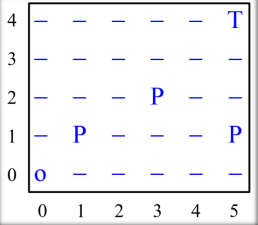
首先以机器人起始位置建立根节点，并入队；接下来不断重复以下步骤直到判定条件:

1. 将队首节点的位置标记已访问；判断队首是否为目标位置(出口)， 是 则终止循环并记录回溯路径
2. 判断队首节点是否为叶子节点，是 则拓展该叶子节点
3. 如果队首节点有子节点，则将每个子节点插到队尾
4. 将队首节点出队

**强化学习算法：Q-Learning算法**

在Q-learning中，我们通过维护一张Q值表， 通过贝尔曼(Bellman)方程对其进行不停的 迭代尝试直至收敛，然后根据Q值表获取 Agent在每个状态下的最优策略。但Q值表 在状态和动作空间都是有限且低维的时候 适用，当状态-动作空间高维且连续时， 维护一张无限庞大的Q值表是不现实的。 因此。DQN提出将Q-Table的更新问题变 成一个函数拟合问题，相近的状态将得到 相近的动作输出，即使用神经网络对动作 -状态的Q值进行建模估计。

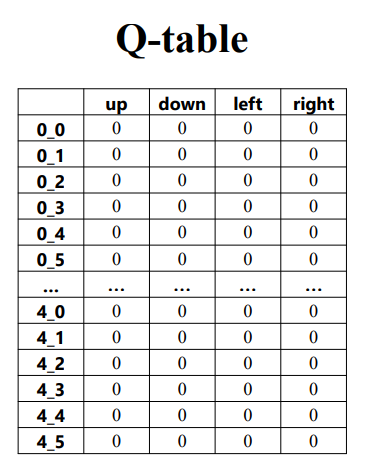
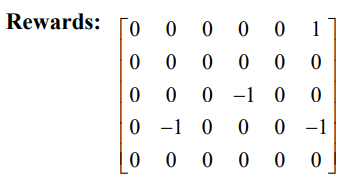
使用强化学习算法设计一个可以在一定大小 的迷宫中找到宝藏的agent(智能体)。迷宫可 以使用一系列字符表示，字符"o"表示agent 的位置，字符"T"表示宝藏的位置，该位置 的收益为1，且为游戏的终结状态，字符"P" 表示陷阱的位置，该位置的收益为-1，且为 游戏的终结状态，字符"\_"表示收益为0的中 间状态。初始情况下，迷宫如下图表示：



每次行动agent可以选择向 不超过迷宫边界的相邻位置 移动一格，通过Q-Learning 算法希望agent最终找到一 条通往宝藏的路径。

States : {(0, 0), (0, 1), …, (4, 4) , (4, 5)}

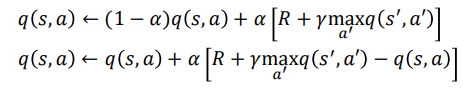
Actions: {(+1, 0), (-1, 0), (0, -1), (0, +1)}



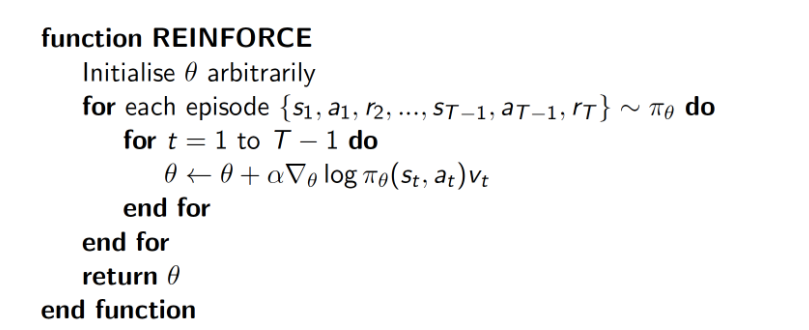
1. 探索与利用

以0.2的概率随机选择策略 以(1-0.2)的概率执行贪婪策略 Q-table中对应状态贪婪策略有多个备 选action时，从备选action中随机选择 action；否则选取最大Q-value的action。

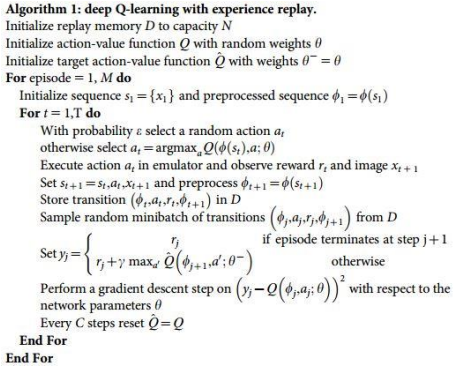
1. 更新Q-table

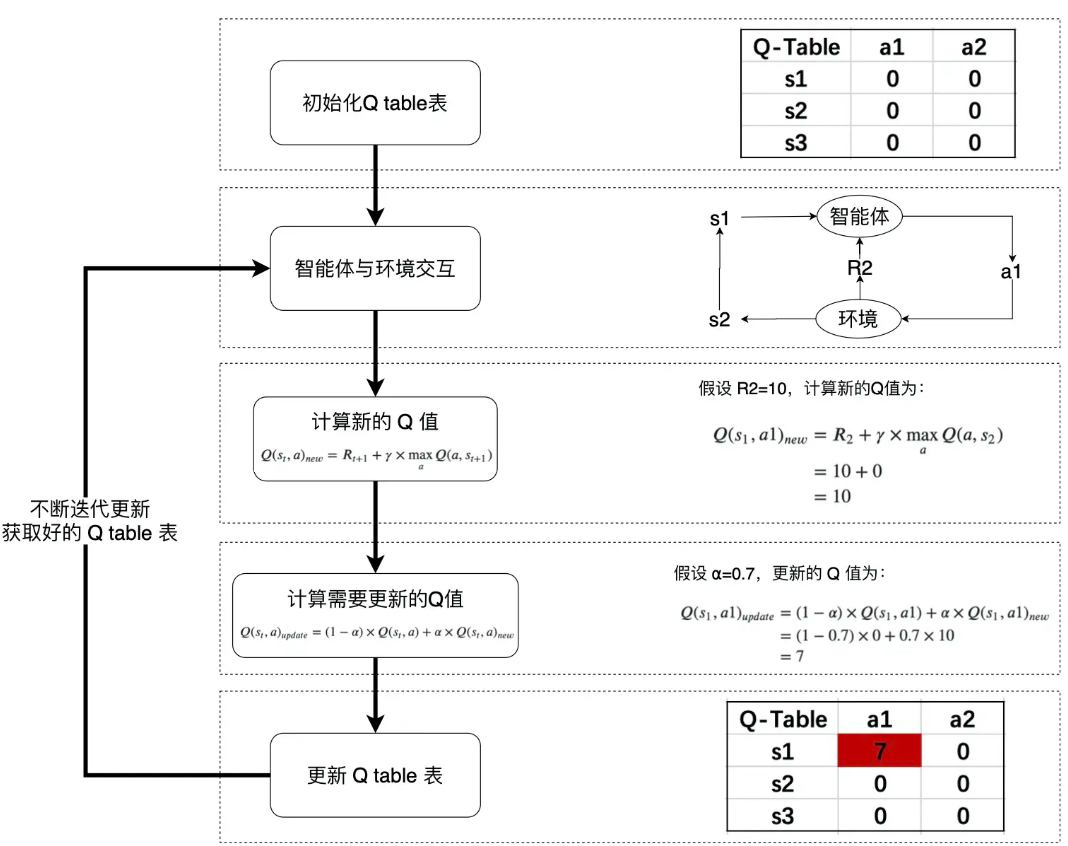


伪代码如下：



整体的伪代码如下：





1. **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）

====================================================================

#广度优先搜索

def dfs(maze,current\_node,is\_visit\_m, path):

is\_visit\_m[current\_node.loc] = 1

if current\_node.loc == maze.destination:

res = back\_propagation(current\_node)

for items in res:

path.append(items)

return

if current\_node.is\_leaf():

expand(maze, is\_visit\_m, current\_node)

for child in current\_node.children:

dfs(maze,child,is\_visit\_m, path)

is\_visit\_m[current\_node.loc] = 0

def depth\_first\_search(maze):

"""

对迷宫进行深度优先搜索

:param maze: 待搜索的maze对象

"""

start = maze.sense\_robot()

root = SearchTree(loc=start)

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

dfs(maze,root,is\_visit\_m,path)

return path

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",...]

"""

print(maze)

path = []

# -----------------请实现你的算法代码--------------------------------------

path = depth\_first\_search(maze)

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

return path

#DQN

import random

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):

"""

初始化 Robot 类

:param maze:迷宫对象

"""

self.maze = maze

self.state = None

self.action = None

self.alpha = alpha

self.gamma = gamma

self.epsilon = epsilon # 动作随机选择概率

self.q\_table = {}

self.maze.reset\_robot() # 重置机器人状态

self.state = self.maze.sense\_robot() # state为机器人当前状态

if self.state not in self.q\_table: # 如果当前状态不存在，则为 Q 表添加新列

self.q\_table[self.state] = {a: 0.0 for a in self.valid\_action}

def train\_update(self):

"""

以训练状态选择动作，并更新相关参数

:return :action, reward 如："u", -1

"""

self.state = self.maze.sense\_robot() # 获取机器人当初所处迷宫位置

# 检索Q表，如果当前状态不存在则添加进入Q表

if self.state not in self.q\_table:

self.q\_table[self.state] = {a: 0.0 for a in self.valid\_action}

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) # action为机器人选择的动作

reward = self.maze.move\_robot(action) # 以给定的方向移动机器人,reward为迷宫返回的奖励值

next\_state = self.maze.sense\_robot() # 获取机器人执行指令后所处的位置

# 检索Q表，如果当前的next\_state不存在则添加进入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 Q 值表

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

def test\_update(self):

"""

以测试状态选择动作，并更新相关参数

:return :action, reward 如："u", -1

"""

self.state = self.maze.sense\_robot() # 获取机器人现在所处迷宫位置

# 检索Q表，如果当前状态不存在则添加进入Q表

if self.state not in self.q\_table:

self.q\_table[self.state] = {a: 0.0 for a in self.valid\_action}

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

reward = self.maze.move\_robot(action) # 以给定的方向移动机器人

return action, reward

在torch\_py文件夹里的MinDQNRobot.py文件里作如下修改：

maze.set\_reward(reward={

"hit\_wall": -10.,

"destination": 50.,

"default": 1.,

})

1. **实验结果**

（实验结果，必填）

====================================================================

1. **总结**

（自评分析（是否达到目标预期，可能改进的方向，实现过程中遇到的困难，从哪些方面可以提升性能，模型的超参数和框架搜索是否合理等），**思考题，非必填**）

====================================================================

虽然能够顺利完成测试，但是耗时较久。

遇到的困难：在编写DQN函数的时候觉得比较困难，但是作业中依靠的是简单的两层全连接神经网络决策动作，也不算太过复杂。