**程序报告**

学号：2311061 姓名：马淏怡

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）

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

1. 使用基础搜索算法和Deep QLearning算法完成机器人自动走迷宫
2. 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

1. 需要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。
2. **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**）

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

1. 基础搜索算法

核心是利用DFS深度优先搜索算法，首先定义一个根节点root ，然后定义 is\_visit 函变量用于存储bool 值，用于标记迷宫的各个位置是否被访问过。然后循环遍历栈，首先将栈顶元素作为当前节点，如果到达了目标点，就给path 赋值，跳出while 循环。如果节点是叶子节点的话，而且还没有扩展，那么首先对其进行标注已扩展，然后将其进行入栈直到跳出循环。

1. Deep QLearning算法

首先读取机器人当前位置，将当前状态加入Q值表中，如果表中已经存在当前状态则不需重复添加。之后生成机器人的执行动作，并返回地图奖励值、查找机器人现阶段位置。接着再次检查并更新Q值表，最后衰减随机选取动作的可能性。

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）

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

# 导入相关包

import os

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

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--------------------------------------

move\_map = {

'u': (-1, 0), # 表示往上走

'r': (0, +1), # 表示往右走

'd': (+1, 0), # 表示往下走

'l': (0, -1), # 表示往左走

}

class SearchTree(object):

def \_\_init\_\_(self, loc=(), action='', parent=None):

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):

self.children.append(child)

def is\_leaf(self):

return len(self.children) == 0

def expand(maze, is\_visit, node):

child\_number = 0 # 记录叶子节点个数

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[new\_loc]:

child = SearchTree(loc=new\_loc, action=a, parent=node)

node.add\_child(child)

child\_number+=1

return child\_number # 返回叶子节点个数

def back\_propagation(node):

path = []

while node.parent is not None:

path.insert(0, node.to\_this\_action)

node = node.parent

return path

def DFS(maze):

start = maze.sense\_robot()

root = SearchTree(loc=start)

queue = [root] # 节点堆栈，用于层次遍历

h, w, \_ = maze.maze\_data.shape

is\_visit = 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

temp = 0

while True:

current\_node = queue[temp] # 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[current\_node.loc] == 0: # 如果该点存在叶子节点且未拓展

is\_visit[current\_node.loc] = 1 # 标记该点已拓展

child\_number = expand(maze, is\_visit, current\_node)

temp+=child\_number # 开展一些列入栈操作

for child in current\_node.children:

queue.append(child) # 叶子节点入栈

else:

queue.pop(temp) # 如果无路可走则出栈

temp-=1

return path

path = DFS(maze)

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

return path

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

# 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为机器人选择的动作

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)

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)

# 衰减随机选择动作的可能性

self.epsilon \*= 0.5

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

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

"""

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

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

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**）

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

1. 达到目标预期
2. 可能改进的方向：基础搜索算法中，可以不使用DFS改用BFS等其他搜索算法，因为DFS在找到路径后就会停止，有可能找不到最优路径；QLearning算法中，可以去调节衰减率，比如若衰减率过小的话会导致随机性过强，策略短视，容易陷入局部最优影响性能，若衰减率过高的话，过度关注长期奖励，收敛慢，在稀疏奖励环境中表现差，训练不稳定。在调参的同时可以结合ε-greedy 策略、学习率、奖励缩放等超参数优化训练效果。