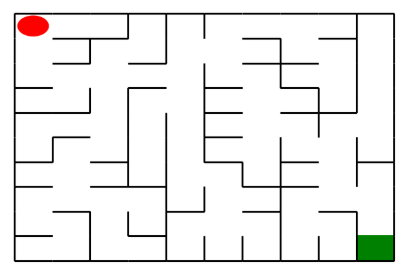
**程序报告**

学号：2113662 姓名：张丛

1. **问题重述**

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。

## **实验要求 ：**

* 使用 Python 语言。
* 使用基础搜索算法完成机器人走迷宫。
* 使用 Deep QLearning 算法完成机器人走迷宫。
* 算法部分需要自己实现，不能使用现成的包、工具或者接口。

1. **设计思想**

**A\*搜索算法：**

一种寻路算法，它在搜索过程中同时考虑到了启发函数（用来评估当前节点到终点的距离）和当前节点到起点的距离，以得出对下一步行动最优的判断。

在实现基础A\*搜索算法的my\_search()函数中，采用以下步骤：

1. 定义辅助函数heuristic用来计算启发函数；

2. 定义辅助函数find\_next\_direction用来根据路径找到下一个移动方向；

3. 复制迷宫对象maze；

4. 使用A\*算法搜索迷宫；

5. 将找到的路径应用于原迷宫对象；

6. 根据路径生成机器人的移动方向列表。

其中open\_list用来保存未处理的节点，closed\_set则保存已处理的节点，通过比较各个节点的F值，从open\_list中选取当前F值最小的节点作为当前节点，不断搜索相邻节点直到找到终点或者open\_list为空。

**基于DQN算法**：

步骤：

1. 初始化机器人（包括迷宫对象、alpha、gamma和epsilon0）；

2. 对训练和测试时更新机器人的状态和Q表；

3. 在训练更新过程中，机器人会根据当前状态选择动作，并获取奖励反馈，再利用Q表更新当前动作的Q值；

4. 在测试更新过程中，机器人会基于Q表中的最优动作，执行动作并获取奖励反馈；

5. 以上步骤重复执行直至到达终点或超时。

重要的函数：

- sense\_robot：获取机器人的当前位置；

- move\_robot：根据方向执行机器人的移动操作，并获取奖励反馈；

- q\_table：机器人的Q表，用于存储状态和每个状态对应的Q值；

- random.choice：随机选择动作的函数；

- max：选择具有最大动作值的动作的函数；

- [a: 0.0 for a in self.valid\_action]：当机器人当前状态不存在于Q表中时，添加当前状态的键，并为所有动作初始化Q值为0.

1. **代码内容**

#A\*基础搜索算法

import copy

def my\_search(maze):

# 辅助函数：计算启发式函数

def heuristic(position):

goal = maze.destination

return abs(position[0] - goal[0]) + abs(position[1] - goal[1])

# 辅助函数：根据路径找到下一个移动方向

def find\_next\_direction(path):

current\_position = maze.sense\_robot()

if len(path) > 0:

next\_position = (current\_position[0] + maze.move\_map[path[0]][0],

current\_position[1] + maze.move\_map[path[0]][1])

if next\_position[0] > current\_position[0]:

return 'd'

elif next\_position[0] < current\_position[0]:

return 'u'

elif next\_position[1] > current\_position[1]:

return 'r'

elif next\_position[1] < current\_position[1]:

return 'l'

return None

# 复制迷宫对象

maze\_copy = copy.deepcopy(maze)

# 使用A\*算法搜索迷宫

path = []

start\_position = maze\_copy.sense\_robot()

open\_list = [(start\_position, [], 0)]

closed\_set = set()

while open\_list:

current\_position, current\_path, current\_cost = min(open\_list, key=lambda x: x[2] + heuristic(x[0]))

open\_list.remove((current\_position, current\_path, current\_cost))

closed\_set.add(current\_position)

if current\_position == maze\_copy.destination:

path = current\_path

break

valid\_actions = maze\_copy.can\_move\_actions(current\_position)

for action in valid\_actions:

next\_x = current\_position[0] + maze\_copy.move\_map[action][0]

next\_y = current\_position[1] + maze\_copy.move\_map[action][1]

next\_position = (next\_x, next\_y)

if next\_position not in closed\_set:

next\_path = current\_path + [action]

next\_cost = current\_cost + 1

open\_list.append((next\_position, next\_path, next\_cost))

# 将找到的路径应用于原迷宫对象

for direction in path:

reward = maze.move\_robot(direction)

if reward == maze.reward['hit\_wall']:

# 如果撞墙，则返回空路径

return []

# 根据路径生成机器人的移动方向列表

path = [find\_next\_direction(path[i:]) for i in range(len(path))]

return path

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

#DQN算法实现Robot类

from QRobot import QRobot

import random

import torch

import torch.nn as nn

import torch.optim as optim

class Robot(QRobot):

def \_\_init\_\_(self, maze,alpha=0.5, gamma=0.9, epsilon0=0.5):

"""

初始化 Robot 类

:param maze:迷宫对象

"""

super(Robot, self).\_\_init\_\_(maze, alpha, gamma, epsilon0)

self.maze = maze

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Deep Q network的相关参数

:return : action, reward 如："u", -1

"""

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

# 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) # 选择具有最大动作值的动作

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:

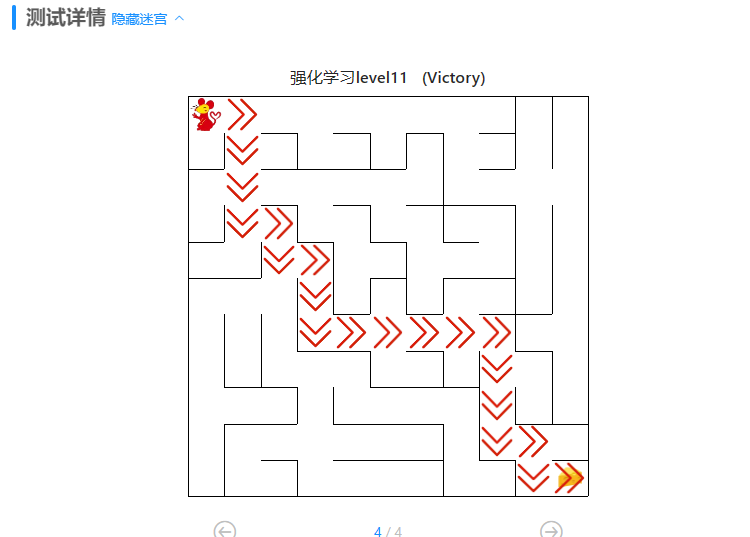
self.q\_table[next\_state] = {a: 0.0 for a in self.valid\_action}

self.state = next\_state # 更新当前状态为下一个状态

return action, reward

1. **实验结果**







1. **总结**

此次实验基本实现了预期：实现了**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。

关于实验可能的优化：

1. 更改机器人的移动策略（比如A\*算法），以更远步距移动机器人，减少对策略的搜索次数。

2. 对机器人的训练数据进行增强处理，如增加数据样本以及采样的方差。

3. 优化机器人的奖励函数，减少出发点到终点的距离，奖励更快到达目标点并且行动次数少的机器人行为。

1. 使用更高级的模型，如RNN，或者更多的神经网络结构来更好地近似机器人的行为。