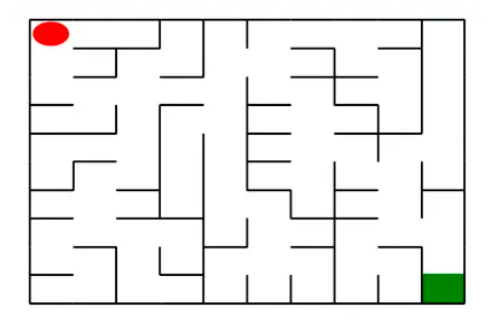
**程序报告**

学号：2213041 姓名：李雅帆

一、**问题重述**

**1.机器人走迷宫介绍**



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：

（1）从起点开始，通过错综复杂的迷宫，到达目标点(出口)。

（2）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'。

（3）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

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

**2.实验要求**

（1）使用基础搜索算法（广度优先搜索、深度优先搜索和最佳优先搜索（A\*)）完成机器人走迷宫。

（2）使用 Deep QLearning 算法完成机器人走迷宫。

（3）算法部分需要自己实现，不能使用现成的包、工具或者接口。

（4）使用Python 语言。

**二、设计思想**

**1.方法选择与设计**

（1）基础搜索算法:

①广度优先搜索 (BFS)

特点：无权图中找到最短路径。

实现思路：使用队列（FIFO）记录节点，按层级顺序扩展节点。

②深度优先搜索 (DFS)

特点：可以更快找到解决方案但未必是最优路径，可能耗费较多时间。

实现思路：使用栈（LIFO）记录节点，尽可能深地搜索节点。

③A\*算法

特点：结合了广度优先搜索和贪婪搜索的优点，使用启发式函数估计路径。

实现思路：使用优先队列（heap）记录节点，按照估计的总成本进行排序。

我在使用基础搜索算法完成机器人走迷宫时采用了最佳优先搜索（A\*)。

（2）Deep Q-Learning算法

①Q-Learning

特点：基于Q值更新策略，机器人通过试探和学习获得最优路径。

实现思路：机器人在迷宫中随机行动，通过获得的奖励更新Q表。

②Deep Q-Learning

特点：使用神经网络近似Q值函数，解决大规模状态空间问题。

实现思路：使用经验回放和目标网络提高学习效果和稳定性。

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

**2.方法改进与优化方向**

（1）广度优先搜索和深度优先搜索

可改进方向：增加节点剪枝策略，减少重复计算。

优化方向：结合启发式方法提升效率。

（2）A\*算法

可改进方向：优化启发式函数设计，提高搜索效率。

优化方向：结合动态规划方法，进一步减少计算量。

（3）Deep Q-Learning

可改进方向：优化神经网络结构和超参数，如学习率、折扣因子、探索率等。

优化方向：引入双重Q-Learning、优先经验回放等技术提升学习效果。

**三、代码内容**

**1.使用基础搜索算法—最佳优先搜索（A\*)完成机器人走迷宫。**

我使用了A\*算法来实现机器人在迷宫中的路径搜索，利用启发式函数和优先队列加速搜索过程，确保找到代价最低的路径。

代码首先定义了一些基础结构和函数。move\_map字典定义了四个基本移动方向。SearchTree类表示搜索树节点，包含节点位置、移动方向、父节点、子节点以及从起点到当前节点的实际成本（g）、从当前节点到目标节点的估计成本（h）和总成本（f）。heuristic函数计算启发式估价函数，这里使用曼哈顿距离。

expand函数用于拓展当前叶子节点，将合法移动后到达的子节点添加到当前节点的子节点列表中。back\_propagation函数通过回溯记录路径，从目标节点回溯到起点。my\_search函数是A\*算法的实现，首先获取迷宫中机器人起点和目标点的位置，并初始化根节点。根节点的估计成本和总成本都等于启发式估价。open\_set是优先队列，存储待处理节点，按总成本排序。is\_visit\_m矩阵记录迷宫中各位置是否被访问。

在搜索过程中，首先从优先队列中取出总成本最小的节点，将其标记为已访问。如果当前节点是目标节点，则通过回溯函数记录路径并结束搜索。如果当前节点是叶子节点，调用expand函数拓展其子节点。然后将子节点按总成本入队。重复这一过程，直到找到路径或所有节点都被处理。

# 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

import heapq

import numpy as np

# 机器人移动方向

move\_map = {

'u': (-1, 0), # up

'r': (0, +1), # right

'd': (+1, 0), # down

'l': (0, -1), # left

}

# 迷宫路径搜索树

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

self.g = 0 # 从起点到当前节点的实际成本

self.h = 0 # 从当前节点到目标的估计成本

self.f = 0 # 总成本

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 \_\_lt\_\_(self, other):

"""

定义节点的比较函数，基于总成本 f

"""

return self.f < other.f

def heuristic(loc, goal):

"""

计算启发式估价函数，这里使用曼哈顿距离

:param loc: 当前节点位置

:param goal: 目标位置

:return: 启发式估价

"""

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

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

"""

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

:param goal: 目标位置

"""

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)

child.g = node.g + 1

child.h = heuristic(new\_loc, goal)

child.f = child.g + child.h

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):

"""

对迷宫进行 A\* 搜索

:param maze: 待搜索的maze对象

"""

start = maze.sense\_robot()

goal = maze.destination

root = SearchTree(loc=start)

root.h = heuristic(start, goal)

root.f = root.h

open\_set = []

heapq.heappush(open\_set, (root.f, root))

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

is\_visit\_m = np.zeros((h, w), dtype=int) # 标记迷宫的各个位置是否被访问过

path = [] # 记录路径

while open\_set:

\_, current\_node = heapq.heappop(open\_set)

is\_visit\_m[current\_node.loc] = 1 # 标记当前节点位置已访问

if current\_node.loc == goal: # 到达目标点

path = back\_propagation(current\_node)

break

if current\_node.is\_leaf():

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

# 入队

for child in current\_node.children:

if not is\_visit\_m[child.loc]:

heapq.heappush(open\_set, (child.f, child))

return path

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

**2.使用 Deep QLearning 算法完成机器人走迷宫。**

我实现了一个机器人在迷宫中使用深度强化学习（Deep Q-Learning）进行路径搜索的算法。

机器人在迷宫中通过反复尝试不同的路径，不断学习和更新每个状态下采取不同动作所获得的奖励。首先，机器人根据一定的概率随机选择动作进行探索，同时根据已经学习到的经验选择最优动作。每次动作后，机器人获取新的状态和相应的奖励，并通过贝尔曼方程更新Q值表。随着训练的进行，机器人逐渐减少随机选择的概率，更倾向于选择Q值最高的动作，从而优化路径。最终，经过多次训练，机器人学会在迷宫中找到从起点到终点的最优路径。这种方法通过反复试验和不断更新策略，使机器人能够适应复杂的迷宫环境，逐步提高路径规划的效率和准确性。

from QRobot import QRobot

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:

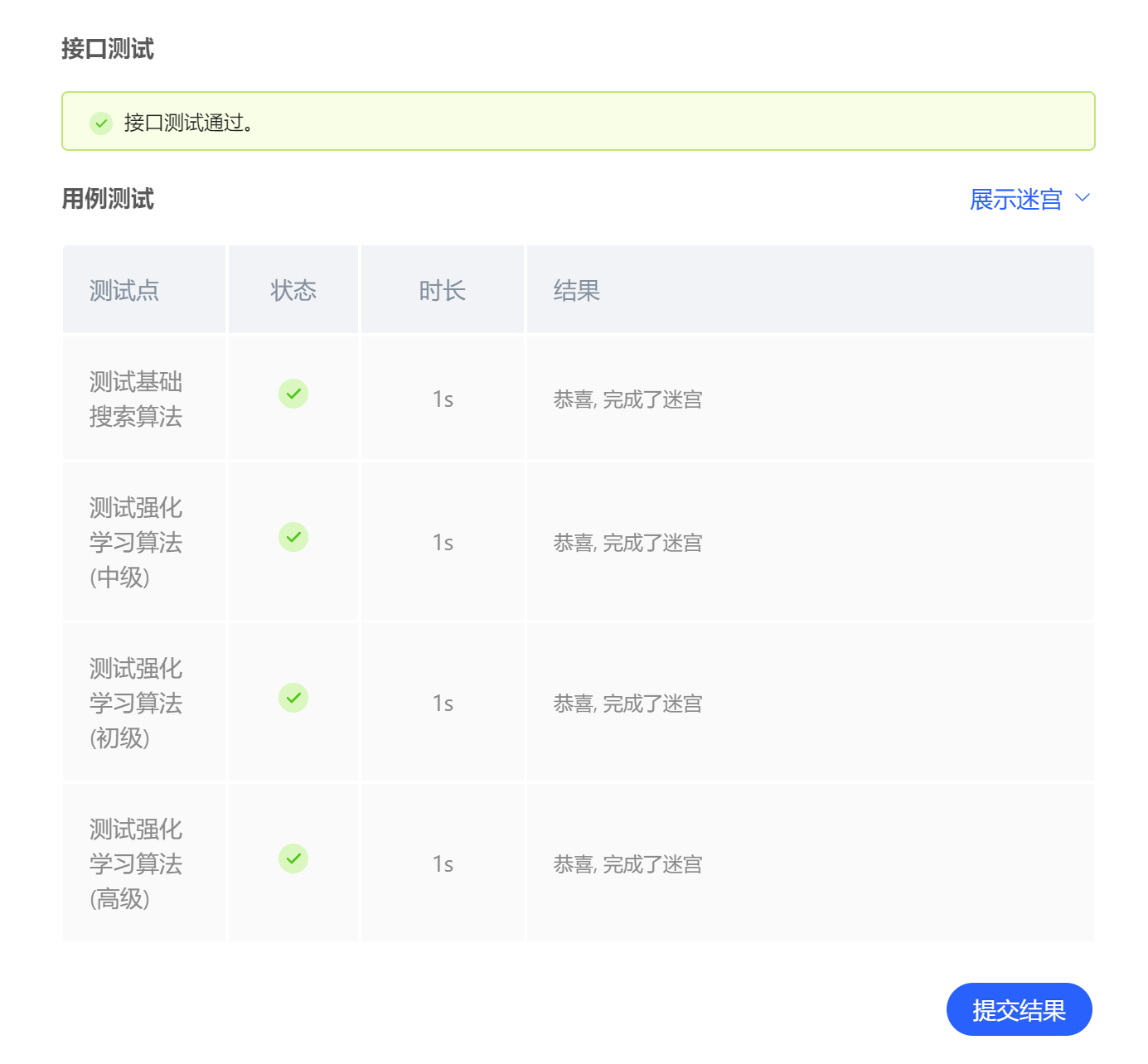
self.q\_table[self.state] = {a: 0.0 for a in self.valid\_action}

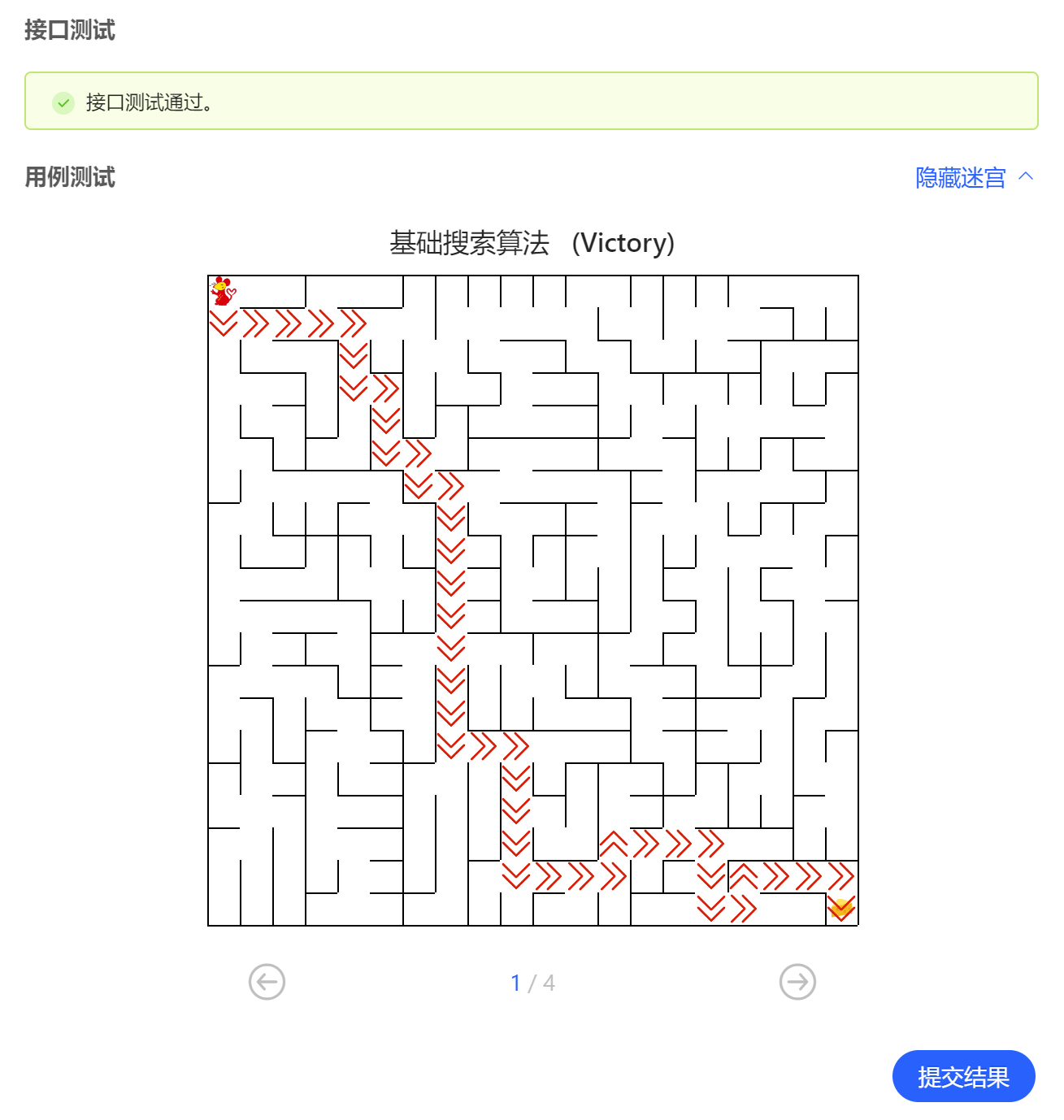
action = max(self.q\_table[self.state],key=self.q\_table[self.state].get) # 选择动作

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

return action, reward

**四、实验结果**





**五、总结**

**1.达到目标预期**

实验目标是让机器人从迷宫的起点通过各种算法到达目标点。通过实现和测试，机器人能够在迷宫中利用广度优先搜索和A\*搜索算法成功找到路径。

此外，使用Deep Q-Learning算法的机器人也能够在训练后有效地找到出口，达到了实验的预期目标。

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

**2. 可能改进的方向**

（1）优化A\*算法：

①对于基础搜索算法，可以进一步优化A\*算法的启发函数，使其更符合实际情况，提升搜索效率。

②引入动态规划技术，减少搜索过程中的重复计算，提升算法效率。

（2）增强Deep Q-Learning算法：

①增加训练次数：更多的训练可以让机器人学习到更优的策略。

②使用经验回放：通过经验回放机制，可以减少样本间的相关性，提高训练效果。

③使用双重Q-learning：可以有效减少过估计问题，提高Q值的准确性。

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

**3.实现过程中遇到的困难**

（1）算法实现复杂度高：实现和调试A\*搜索算法时，如何设计和计算启发式函数是一个挑战。在实现Deep Q-Learning算法时，如何设计奖励机制和状态表示，以及如何有效更新Q值表，都是较为复杂的问题。

（2）超参数调优困难：在Deep Q-Learning算法中，超参数（如学习率、折扣因子、探索率等）的选择对结果影响较大，调优这些参数需要大量的实验和验证。