**程序报告**

学号： 2112060 姓名：孙蕗

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）

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

题目1：

任选深度优先搜索算法、最佳优先搜索 A\* 算法其中一种实现机器人走迷宫。

输入迷宫，输出到达目标点的路径。

深度优先搜索算法（DFS）是一种遍历算法，它通过沿着每条可能的路径尽可能远地搜索，直到到达目标点或无法继续前进。在这个问题中，可以使用递归的方式实现DFS。机器人从起点开始，依次尝试向上、向下、向左、向右四个方向移动，如果可以移动到某个方向，则继续在新的位置上进行DFS搜索，直到找到目标点或所有路径都被探索完。

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

题目2：

编程实现 DQN 算法在机器人自动走迷宫中的应用，输入由 Maze 类实例化的对象 maze，

要求不可更改的成员方法：train\_update()、test\_update()

注：不能修改该方法的输入输出及方法名称，测试评分会调用这两个方法。

补充1：若要自定义的参数变量，在 \_\_init\_\_() 中以 self.xxx = xxx 创建即可

补充2:实现你自己的DQNRobot时，要求继承 QRobot 类，QRobot 类包含了某些固定的方法如reset(重置机器人位置),sense\_state(获取机器人当前位置)..

实现DQN算法在机器人自动走迷宫中的应用。给定一个迷宫对象maze，需要实现两个不可更改的成员方法：train\_update()和test\_update()分别用于训练DQN模型和测试DQN模型在迷宫中的表现。此外，需要定义一个继承自QRobot类的DQNRobot类来实现自己的DQN算法。

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**）

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

题目1：

定义了SearchTree类表示迷宫路径搜索树，包括节点位置、移动方向、父节点和子节点等信息。在depth\_first\_search函数中，使用DFS进行迷宫搜索，从起点开始递归地拓展搜索树并回溯路径。expand函数用于拓展叶子节点，根据合法动作生成子节点，并将子节点添加到搜索树中。back\_propagation函数用于回溯搜索树并记录节点路径。

使用剪枝策略等可以作为优化方向，对于大型迷宫效率可能较低。

题目2：

实现了基于Deep Q-Learning（DQN）算法的机器人走迷宫问题。代码中定义了一个Robot类，继承自TorchRobot类，用于实现DQN算法在机器人走迷宫中的应用。该类在初始化时设置迷宫的奖励值，并调用train()方法进行训练。在train()方法中，使用\_learn()方法进行模型的训练，并将每个训练批次的损失值记录在loss\_list中。然后，在训练过程中，重复执行测试更新test\_update()，直到机器人达到目标点。train\_update()方法用于在训练过程中更新机器人的状态、选择动作和获取奖励值。test\_update()方法用于在测试过程中更新机器人的状态、根据当前状态选择最优动作，并获取奖励值。在测试过程中，通过Runner类的run\_training()方法来运行训练过程，并生成训练过程的GIF动画。

调整参数、网络结构优化等可以作为优化方向，DQN算法在处理复杂的迷宫问题时可能存在局限性和常见问题，如训练过程中的收敛速度较慢、训练样本不充分时可能陷入局部最优等。

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）

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

题目1：

任选深度优先搜索算法、最佳优先搜索 A\* 算法其中一种实现机器人走迷宫。

输入迷宫，输出到达目标点的路径

2.3.1 编写您的基础搜索算法

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

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

"""

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

"""

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)

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 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

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

2.3.2测试您的基础搜索算法

maze = Maze(maze\_size=10) # 从文件生成迷宫

path\_2 = my\_search(maze)

print("搜索出的路径：", path\_2)

for action in path\_2:

maze.move\_robot(action)

if maze.sense\_robot() == maze.destination:

print("恭喜你，到达了目标点")

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

2.6 题目二: 实现 Deep QLearning 算法

（2）实现你自己的 DQNRobot

题目要求: 编程实现 DQN 算法在机器人自动走迷宫中的应用

输入: 由 Maze 类实例化的对象 maze

要求不可更改的成员方法：train\_update()、test\_update() 注：不能修改该方法的输入输出及方法名称，测试评分会调用这两个方法。

补充1:若要自定义的参数变量，在 \_\_init\_\_() 中以 self.xxx = xxx 创建即可

补充2:实现你自己的DQNRobot时，要求继承 QRobot 类，QRobot 类包含了某些固定的方法如reset(重置机器人位置),sense\_state(获取机器人当前位置)..

import random

import numpy as np

import torch

from QRobot import QRobot

from ReplayDataSet import ReplayDataSet

from torch\_py.MinDQNRobot import MinDQNRobot as TorchRobot # PyTorch版本

import matplotlib.pyplot as plt

from Maze import Maze

import time

from Runner import Runner

class Robot(TorchRobot):

def \_\_init\_\_(self, maze):

"""

初始化 Robot 类

:param maze:迷宫对象

"""

super(Robot, self).\_\_init\_\_(maze)

maze.set\_reward(reward={

"hit\_wall": 5.0,

"destination": -maze.maze\_size \*\* 2.0,

"default": 1.0,

})

self.maze = maze

self.epsilon = 0

"""开启金手指，获取全图视野"""

self.memory.build\_full\_view(maze=maze)

self.loss\_list = self.train()

def train(self):

loss\_list = []

batch\_size = len(self.memory)

while True:

loss = self.\_learn(batch=batch\_size)

loss\_list.append(loss)

success = False

self.reset()

for \_ in range(self.maze.maze\_size \*\* 2 - 1):

a, r = self.test\_update()

if r == self.maze.reward["destination"]:

return loss\_list

def train\_update(self):

def state\_train():

state=self.sense\_state()

return state

def action\_train(state):

action=self.\_choose\_action(state)

return action

def reward\_train(action):

reward=self.maze.move\_robot(action)

return reward

state = state\_train()

action = action\_train(state)

reward = reward\_train(action)

return action, reward

def test\_update(self):

def state\_test():

state = torch.from\_numpy(np.array(self.sense\_state(), dtype=np.int16)).float().to(self.device)

return state

state = state\_test()

self.eval\_model.eval()

with torch.no\_grad():

q\_value = self.eval\_model(state).cpu().data.numpy()

def action\_test(q\_value):

action=self.valid\_action[np.argmin(q\_value).item()]

return action

def reward\_test(action):

reward=self.maze.move\_robot(action)

return reward

action = action\_test(q\_value)

reward = reward\_test(action)

return action, reward

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

1. 测试您的 DQN 算法

from QRobot import QRobot

from Maze import Maze

from Runner import Runner

""" Deep Qlearning 算法相关参数： """

epoch = 10 # 训练轮数

maze\_size = 5 # 迷宫size

training\_per\_epoch=int(maze\_size \* maze\_size \* 1.5)

""" 使用 DQN 算法训练 """

g = Maze(maze\_size=maze\_size)

r = Robot(g)

runner = Runner(r)

runner.run\_training(epoch, training\_per\_epoch)

# 生成训练过程的gif图, 建议下载到本地查看；也可以注释该行代码，加快运行速度。

runner.generate\_gif(filename="results/dqn\_size10.gif")

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

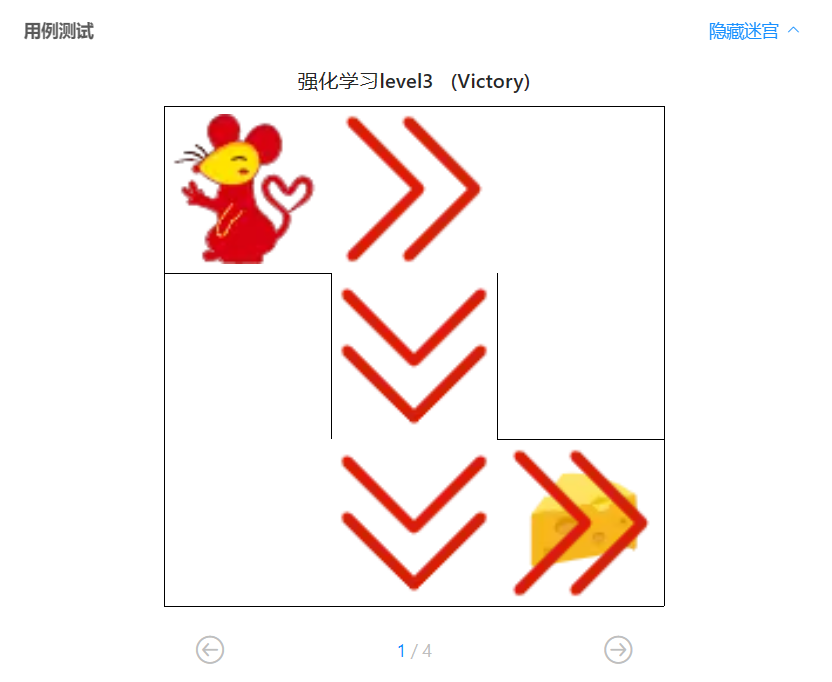
1. **实验结果**

（实验结果，必填）

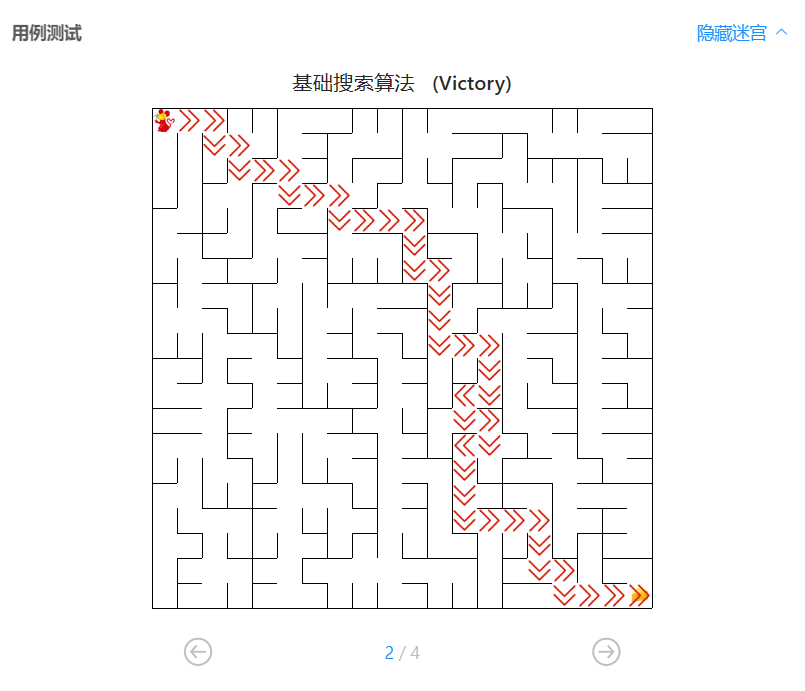
====================================================================



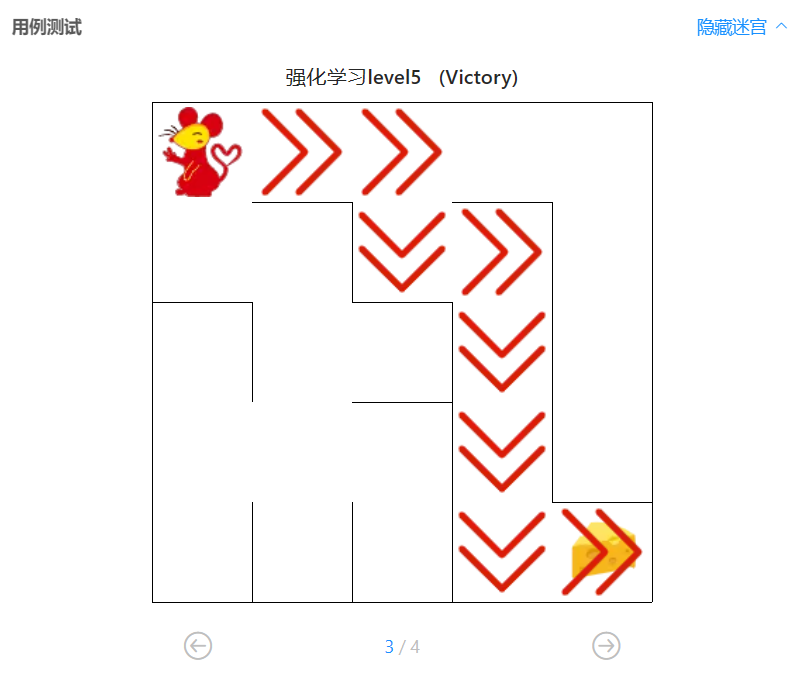
====================================================================



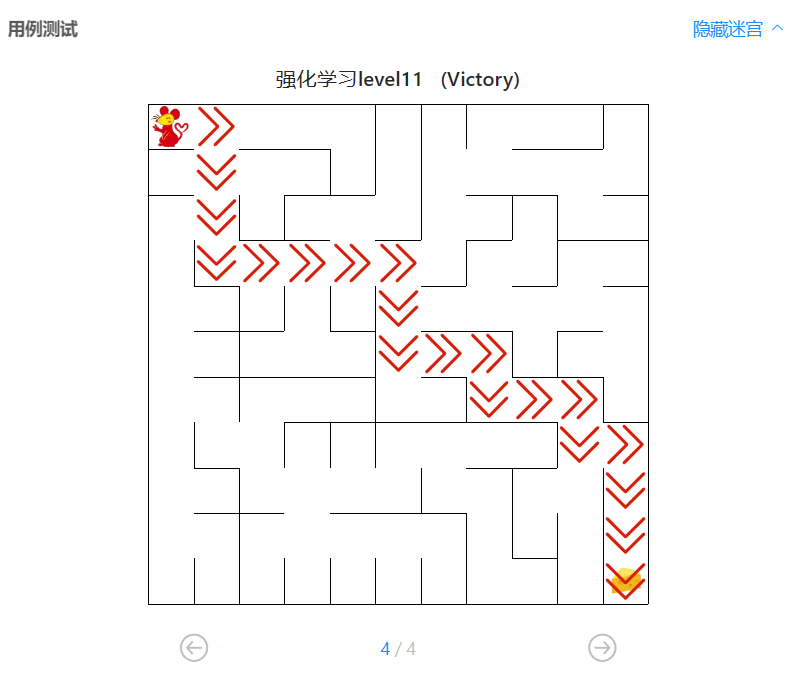
====================================================================



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



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



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**）

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

题目1：

定义了SearchTree类表示迷宫路径搜索树，包括节点位置、移动方向、父节点和子节点等信息。在depth\_first\_search函数中，使用DFS进行迷宫搜索，从起点开始递归地拓展搜索树并回溯路径。expand函数用于拓展叶子节点，根据合法动作生成子节点，并将子节点添加到搜索树中。back\_propagation函数用于回溯搜索树并记录节点路径。

可以进一步优化搜索算法的效率，例如使用启发式函数来改进A\*算法的性能，或者尝试其他优化的搜索算法。

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

题目2：

实现了基于Deep Q-Learning（DQN）算法的机器人走迷宫问题。代码中定义了一个Robot类，继承自TorchRobot类，用于实现DQN算法在机器人走迷宫中的应用。该类在初始化时设置迷宫的奖励值，并调用train()方法进行训练。在train()方法中，使用\_learn()方法进行模型的训练，并将每个训练批次的损失值记录在loss\_list中。然后，在训练过程中，重复执行测试更新test\_update()，直到机器人达到目标点。train\_update()方法用于在训练过程中更新机器人的状态、选择动作和获取奖励值。test\_update()方法用于在测试过程中更新机器人的状态、根据当前状态选择最优动作，并获取奖励值。在测试过程中，通过Runner类的run\_training()方法来运行训练过程，并生成训练过程的GIF动画。

可以尝试调整DQN算法的超参数，如学习率、批次大小、记忆池大小等，以获得更好的训练效果。

可以尝试使用其他改进的DQN算法，如Double DQN、Dueling DQN、Prioritized Experience Replay等，以提升算法的性能和稳定性。

代码中的超参数设置可能需要根据具体问题和实验进行调整，以获得更好的性能。