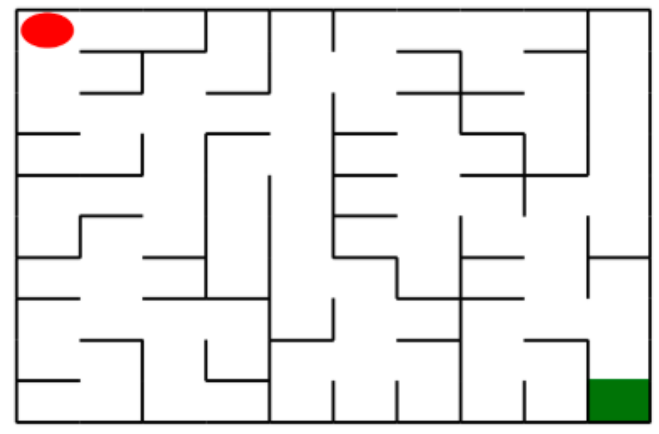
**程序报告**

学号：2212998 姓名：胡博浩

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。

a)撞墙

b)走到出口

c)其余情况

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。

1. **设计思想**

本实验的主要目标是通过强化学习让机器人在一个迷宫环境中找到最短路径，我主要采用了深度优先搜索和深度Q学习的方法。

**深度优先搜索**是一种用于遍历或搜索树或图的算法。它通过尽可能深地搜索树的分支来沿着树的深度遍历节点。当一个节点的所有边都已被探索后，搜索将回溯到发现该节点的那条边的起始节点。这个过程会一直进行，直到找到从源节点可达的所有节点。如果还有未被发现的节点，会选择其中一个作为新的源节点，并重复上述过程。整个进程会反复进行，直到所有节点都被访问过为止。在本实验中，我们使用DFS来指导机器人穿越迷宫，以找到从起点到终点的路径。

**Deep Q-Learning**是一种先进的机器学习技术，它结合了Q学习和深度神经网络的优势。在迷宫环境中，我们利用深度Q学习来训练机器人，使其能够做出最优的决策。在深度Q学习中，我们使用了一个名为DQN的模型。这个模型的目标是学习一个策略，使得机器人能够根据其当前的观察来选择能够最大化未来奖励的行动。在训练过程中，机器人会根据其经验（包括状态、行动和奖励）来更新Q值。通过这种方式，机器人可以逐渐学习到一个优化的策略，从而在迷宫任务中取得更好的表现。



**优化方向：**

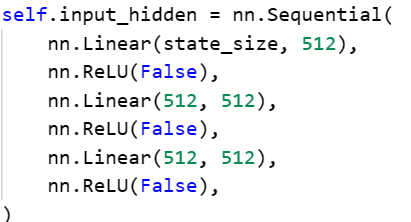
1. 题目一：

对于题目一，我认为在算法层面已经没有进一步优化的空间。深度优先搜索是一种非常成熟的算法，题目要求我们使用这种算法，因此我不会进行优化。

然而，从时间复杂度的角度来看，我们可以采取一些措施来提高算法的效率。具体来说，我们可以利用一个字典来记录已经访问过的节点，以避免重复访问和浪费资源。这样，在最坏的情况下，算法的时间复杂度可以降低到O(n)，其中n是节点的个数。因此，我在这里只进行了一个简单的防止重复访问的优化。

2. 题目二：

这里我主要是扩宽了QNetWork的网络结构，添加了一层Linear，代码如下：



当然我也修改了奖励策略，把撞墙的惩罚调整到了50，从而保证机器人不会撞墙。由于这个实验只要求实现，我没有过多的调参优化。

1. **代码内容**

**main.py**

# 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

from collections import deque

def my\_search(maze):

# 机器人移动方向

move\_map = {

'u': (-1, 0), # 上

'r': (0, 1), # 右

'd': (1, 0), # 下

'l': (0, -1), # 左

}

def dfs(maze):

""" 深度优先搜索 """

start = maze.sense\_robot()

stack = deque([(start, [])]) # 使用deque作为栈，同时记录路径

visited = set() # 记录已访问的位置

while stack:

loc, path = stack.pop()

if loc in visited:

continue

visited.add(loc)

if loc == maze.destination:

return path

for action in maze.can\_move\_actions(loc):

dx, dy = move\_map[action] # 获取移动方向的增量

new\_loc = (loc[0] + dx, loc[1] + dy) # 计算新位置

stack.append((new\_loc, path + [action])) # 将新位置和路径入栈

return dfs(maze)

# 导入相关包

import numpy as np

import torch

from Maze import Maze

from Runner import Runner

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

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": 50.,

"destination": -(maze.maze\_size \*\* 2 \* 10),

"default": 1.,

})

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.train()

def train(self):

# 训练，直到能走出这个迷宫

while True:

self.\_learn(batch=len(self.memory))

self.reset()

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

action, reward = self.test\_update()

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

return

def train\_update(self):

state = self.sense\_state()

action = self.\_choose\_action(state)

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

return action, reward

def test\_update(self):

state = np.array(self.sense\_state(), dtype=np.int16)

state = torch.from\_numpy(state).float().to(self.device)

self.eval\_model.eval()

with torch.no\_grad():

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

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

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

return action, reward

1. **实验结果**

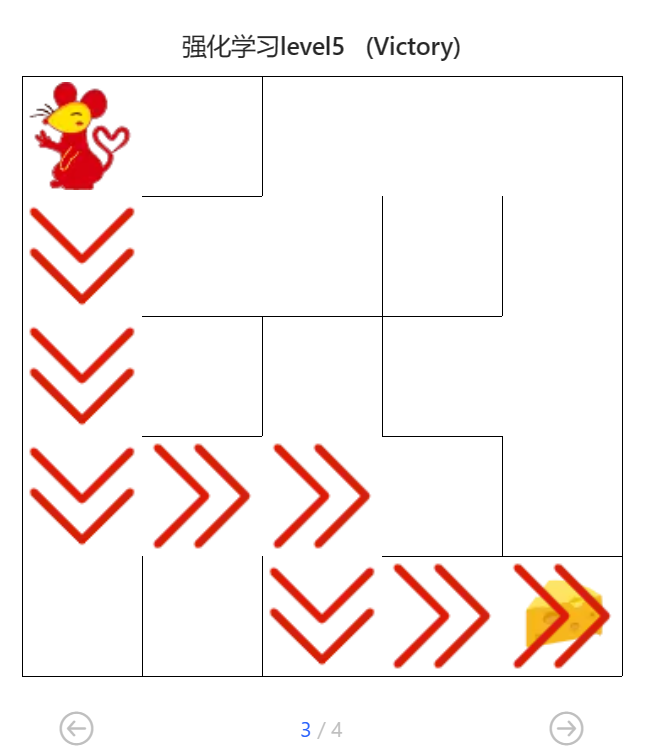
平台测试结果如下，顺利完成了迷宫的考验：

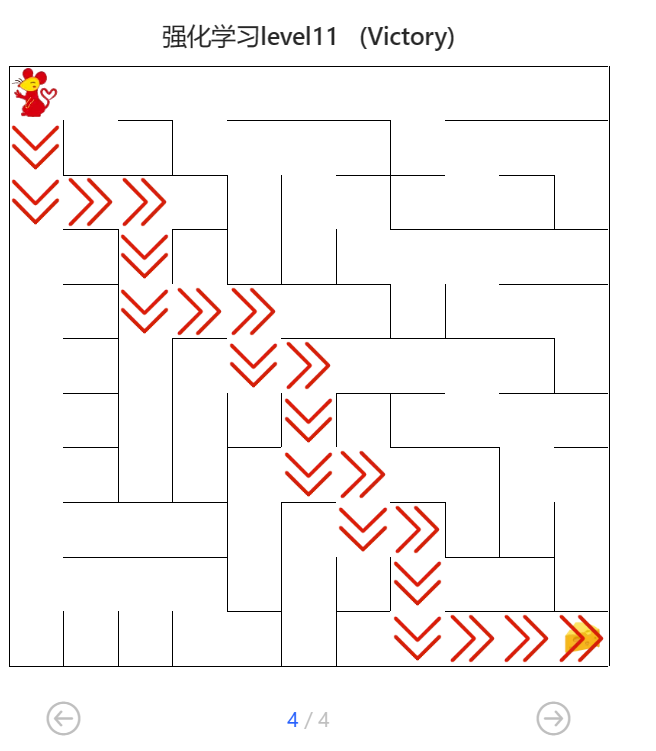


对应的迷宫走法：









1. **总结**

在这次实验中，我成功地应用了深度优先搜索和DQN技术，使机器人能够找到从迷宫入口到出口的路径。尽管取得了一些进展，但仍有许多方面可以改进：

1. 调整训练周期和超参数：通过改变迷宫的大小和复杂度，进一步探索DFS和DQN的性能。

2. 优化奖励机制：引入更复杂的奖励机制，或利用先验知识引导机器人学习，优化训练方法。

3. 引入双DQN：减少Q值过度估计的问题，提高训练稳定性。

4. 考虑多个时间步的回报：提升长远策略学习效果。

在实验过程中，我遇到了一些困难。比如，在保持探索与利用的平衡方面，初期需要较多探索，而后期更倾向于利用。此外，Q值的波动较大，导致策略不稳定，需要通过调节学习率和折扣因子等超参数来稳定训练过程。

通过实现基础的搜索算法和深度Q学习算法，我的编程技巧得到了显著提升，同时对这些算法的理解也更加深入。在实验过程中，我对强化学习和深度学习有了更全面的认识，包括理解探索-利用权衡问题，以及如何通过算法调优和模型优化来提高性能。面对实验中的各种挑战时，我学会了如何分析问题并找到有效的解决方案。这不仅增强了我的问题解决能力，也让更好地理解了算法调优和模型优化的重要性。

总的来说，通过这次实验，我不仅掌握了Q-Learning算法和强化学习的基本概念，还学习了多种优化方法和高级技术。这个过程不仅提高了我的技术能力，也为我未来的学习和研究打下了坚实的基础。