机器人自动走迷宫

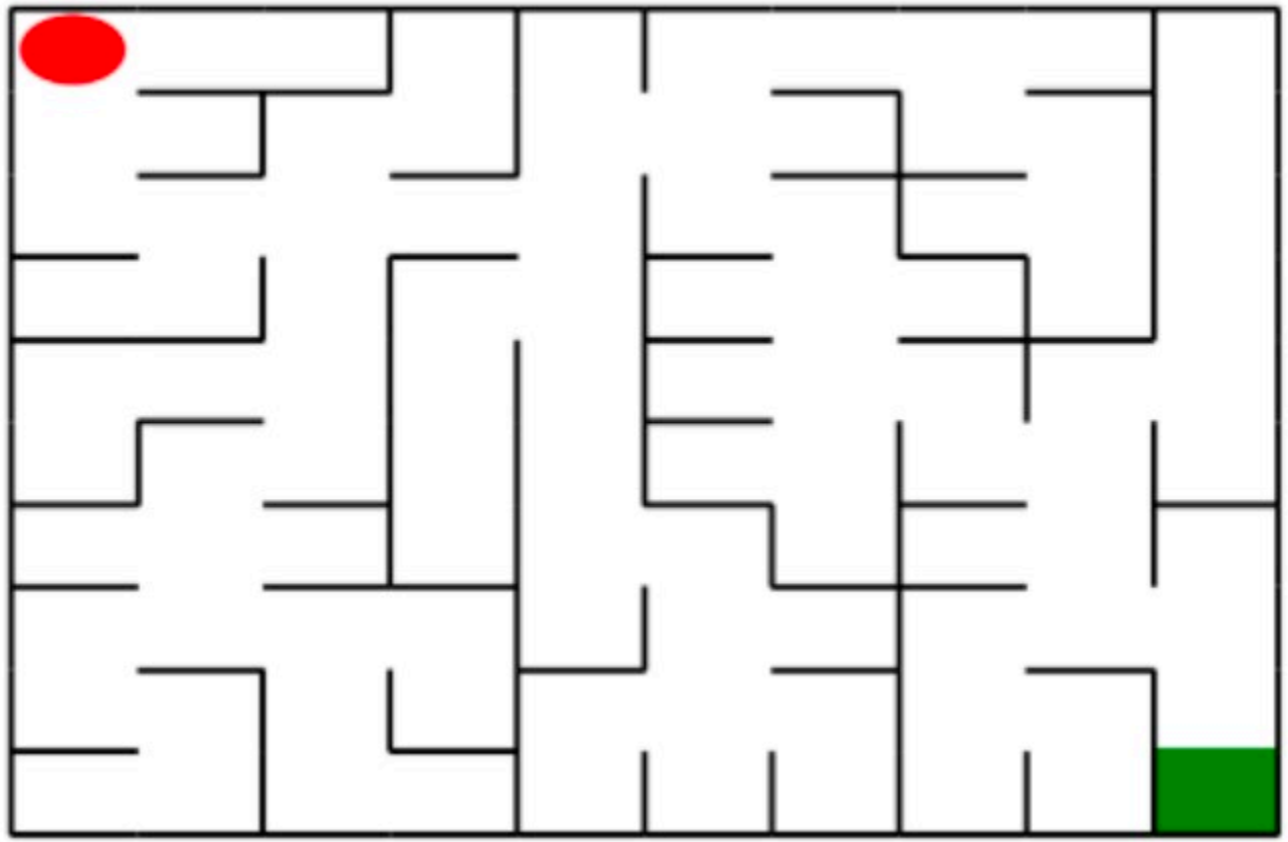
实验时间：2024年5月28日　00:00
截止时间：2024年6月11日 23:55

教师：王亚星　　助教：张鑫

实验思路

The steps of experiment

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

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。

实验思路

The steps of experiment

迷宫类 Maze

1. sense_robot()：获取机器人在迷宫中目前的位置。

   > return：机器人在迷宫中目前的位置。

2. move_robot(direction)：根据输入方向移动默认机器人，若方向不合法则返回错误信息。

   > direction：移动方向, 如:"u", 合法值为： ['u', 'r', 'd', 'l']

   > return：执行动作的奖励值

3. can_move_actions(position)：获取当前机器人可以移动的方向

   > position：迷宫中任一处的坐标点

   > return：该点可执行的动作，如：['u','r','d']

4. is_hit_wall(self, location, direction)：判断该移动方向是否撞墙

   > location, direction：当前位置和要移动的方向，如(0,0) , "u"

   > return：True(撞墙) / False(不撞墙)

5. draw_maze()：画出当前的迷宫

实验思路

The steps of experiment

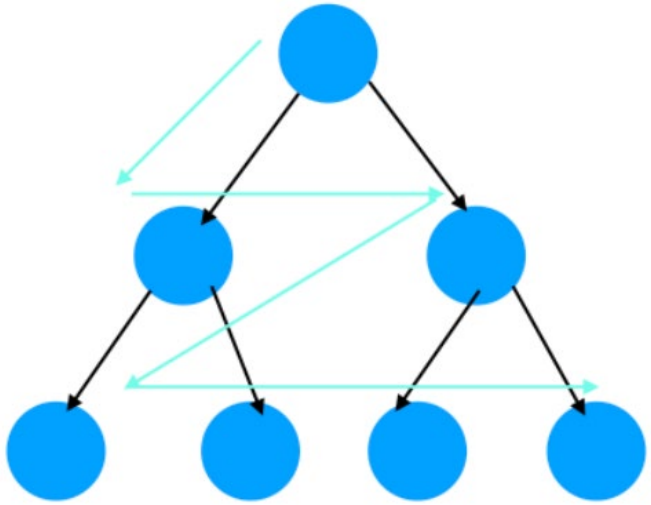
广度优先搜索　　树的层次遍历

`is_visited`　　`queen`

记录节点是否被访问过　　队列

首先以机器人起始位置建立根节点，并入队；接下来不断重复以下步骤直到判定条件:

1. 将队首节点的位置标记已访问；判断队首是否为目标位置(出口)， **是** 则终止循环并记录回溯路径
2. 判断队首节点是否为叶子节点，**是** 则拓展该叶子节点
3. 如果队首节点有子节点，则将每个子节点插到队尾
4. 将队首节点出队



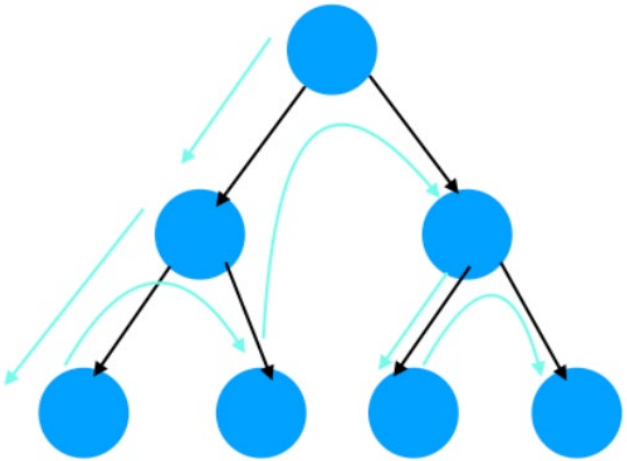
实验思路

The steps of experiment

深度优先搜索　　树的先序遍历

`is_visited`　　`stack`

记录节点是否被访问过　　栈



（1）将起点入栈；

（2）取栈顶元素，求其邻接的未被访问的无障碍结点。求如果有，记其为已访问，并入栈。如果没有则回溯上一结点，具体做法是将当前栈顶元素出栈。

（3）重复步骤（2），直到栈顶元素等于终点（找到一条可行路径）。

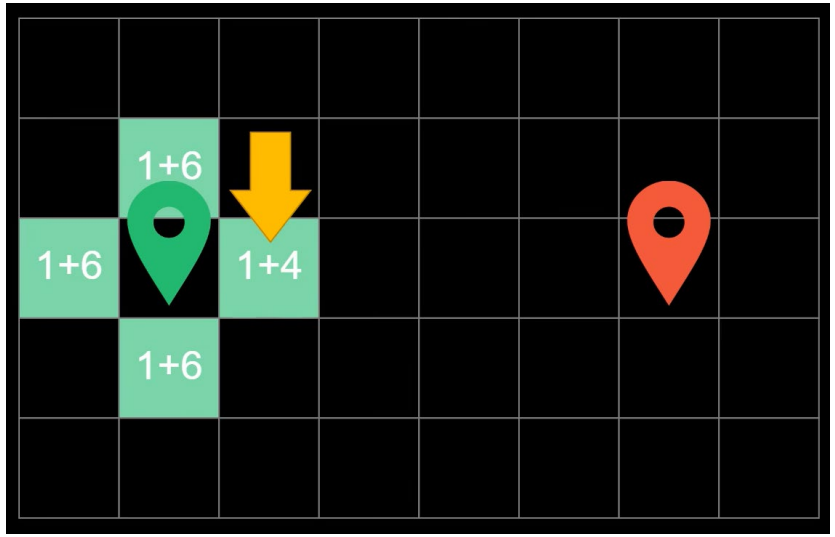
实验思路

The steps of experiment

A*算法

$$f(n) = g(n) + h(n)$$

g(n)为起点到n状态的最短路径代价的估计值，h(n)是n状态到目的状态的最短路径代价的估计值。



实验思路

The steps of experiment

QLearning算法

Q-learning 算法将状态（state）和动作（action）构建成一张 Q_table 表来存储 Q 值，Q 表的行代表状态（state），列代表动作（action）：

Q-Table	a_1	a_2
s_1	$Q(s_1, a_1)$	$Q(s_1, a_2)$
s_2	$Q(s_2, a_1)$	$Q(s_2, a_2)$
s_3	$Q(s_3, a_1)$	$Q(s_3, a_2)$

更新公式 $$Q(s_t, a) = R_{t+1} + \gamma \times \max_a Q(a, s_{t+1})$$

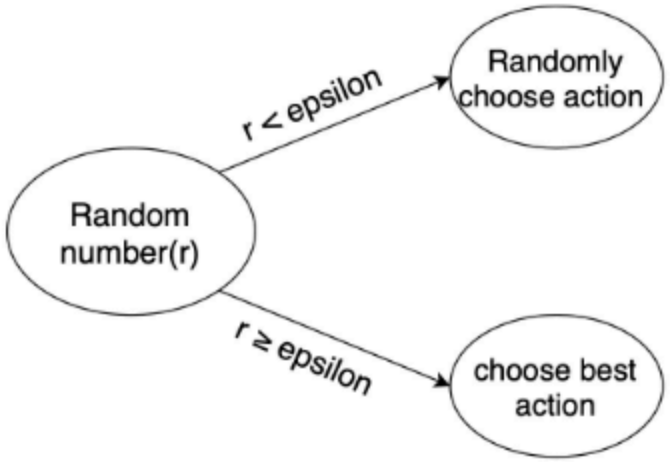
引入松弛变量 $$Q(s_t, a) = (1 - \alpha) \times Q(s_t, a) + \alpha \times (R_{t+1} + \gamma \times \max_a Q(a, s_{t+1}))$$

实验思路

The steps of experiment

Epsilon-greedy策略

解决初始Q值不准确的问题。按一定比例随机选择动作，该比例随着训练过程减小。



实验思路

The steps of experiment

DQN算法

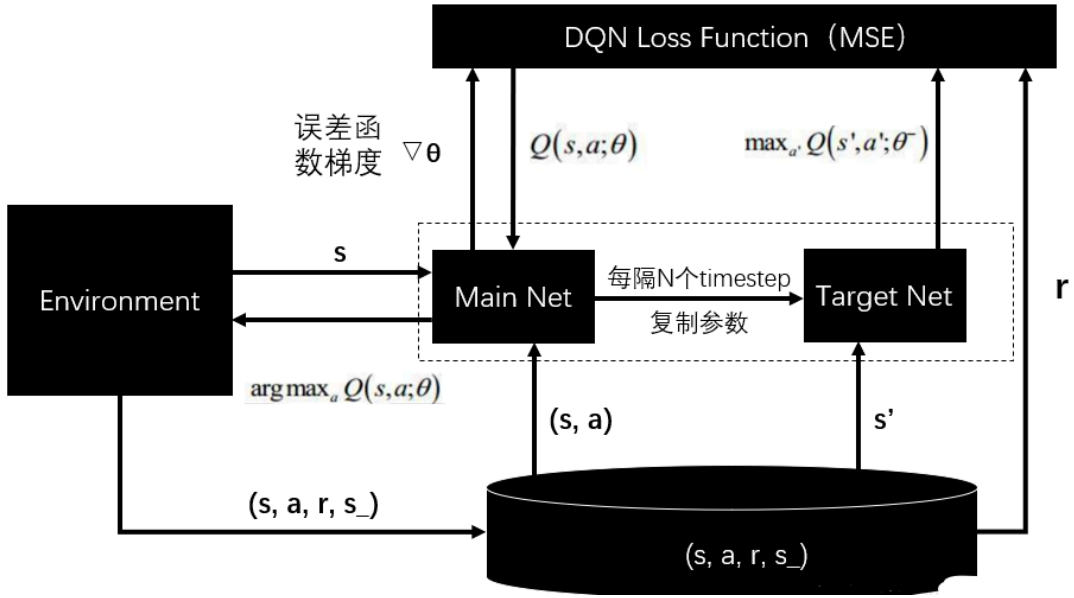
Qlearning算法只针对于离散空间建表

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a' \in A} Q(s',a') - Q(s,a) \right]$$

$$w^* = \arg\min_w \frac{1}{2N} \sum_{i=1}^{N} \left[Q_w(s_i,a_i) - \left(r_i + \gamma \max_{a'} Q_w(s_i',a') \right) \right]^2$$

MSE优化，神经网络输入state，输出action

单个网络可能会有训练不稳定的情况 （交替训练）



实验思路

The steps of experiment

- How to Improve?
- 训练数据利用率不足问题：
  - 采用经验缓冲区技术(Replay)缓解该问题（ReplayDataSet.py）
  - 将transition存储在dataset中，transition:（s,a,s’,r,done)
  - done判断轨迹是否终止
  - 训练过程中每次随机选取dataset中的一个Batch更新网络参数，然后将与环境交互得到的新transition存储到dataset中。
  - 经验缓冲区容量有限，超过限制需要删除对应数据。删除策略？考虑先进先出，队列。有效提高数据利用率。

实验思路

The steps of experiment

QRobot 类的核心成员方法

自己实现的类需要继承自QRbot

1. sense_state()：获取当前机器人所处位置

   > return：机器人所处的位置坐标，如： (0, 0)

2. current_state_valid_actions()：获取当前机器人可以合法移动的动作

   > return：由当前合法动作组成的列表，如： ['u','r']

3. train_update()：以**训练状态**，根据 QLearning 算法策略执行动作

   > return：当前选择的动作，以及执行当前动作获得的回报, 如： 'u', -1

4. test_update()：以**测试状态**，根据 QLearning 算法策略执行动作

   > return：当前选择的动作，以及执行当前动作获得的回报, 如：'u', -1

5. reset()

   > return：重置机器人在迷宫中的位置

实验思路

The steps of experiment

代码实现

```python
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Deep Q network的相关参数
        :return :action, reward 如："u", -1
        """
        action, reward = "u", -1.0

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

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

        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
        """
        action, reward = "u", -1.0

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

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

        return action, reward
```

ReplayDataSet 类的核心成员方法

- add(self, state, action_index, reward, next_state, is_terminal) 添加一条训练数据

  > state: 当前机器人位置
  >
  > action_index: 选择执行动作的索引
  >
  > reward： 执行动作获得的回报
  >
  > next_state：执行动作后机器人的位置
  >
  > is_terminal：机器人是否到达了终止节点（到达终点或者撞墙）

- random_sample(self, batch_size)：从数据集中随机抽取固定batch_size的数据

  > batch_size: 整数，不允许超过数据集中数据的个数

- **build_full_view(self, maze)：开启金手指，获取全图视野**

  > maze: 以 Maze 类实例化的对象

实验思路

The steps of experiment

需要调节哪些参数or结构?

1. QNetwork.py的QNetwork类，其网络结构极为简单，可以调节为相对复杂的结构。增加其表征能力。

2. epoch训练轮数

3. epsilong-greedy中epsilon，平衡探索与利用。

4. 设计的reward，使用maze.set_reward方法。考虑到达终点的reward；在轨迹中的reward，撞墙的reward。Reward Penalty：可以考虑将撞墙的reward调节相对小很多，给model较大的惩罚，使其避免撞墙。

5. reward discount：奖励折扣因子，在0-1之间，考虑长期奖励与短期奖励之间的关系。

实验思路

The steps of experiment

实验提交：

- **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)..

2.6.3 作业测试与提交

- 经过 `2.3` 与 `2.6` 分别测试使用基础算法、DQN算法实现机器人走出迷宫！
- 测试完成之后，点击左侧 `提交作业` 的标签中，把整个 Notebook 目标 cell 转化为 main.py 文件进行 `系统测试` 。
- 平台测试时请记得勾选 main.py 文件需要依赖的其它文件等。
- 通过测试就可以**提交作业**。
- 提交作业时请记得提交勾选 **『程序报告.docx』**或者 **『程序报告.pdf』**。

谢 谢