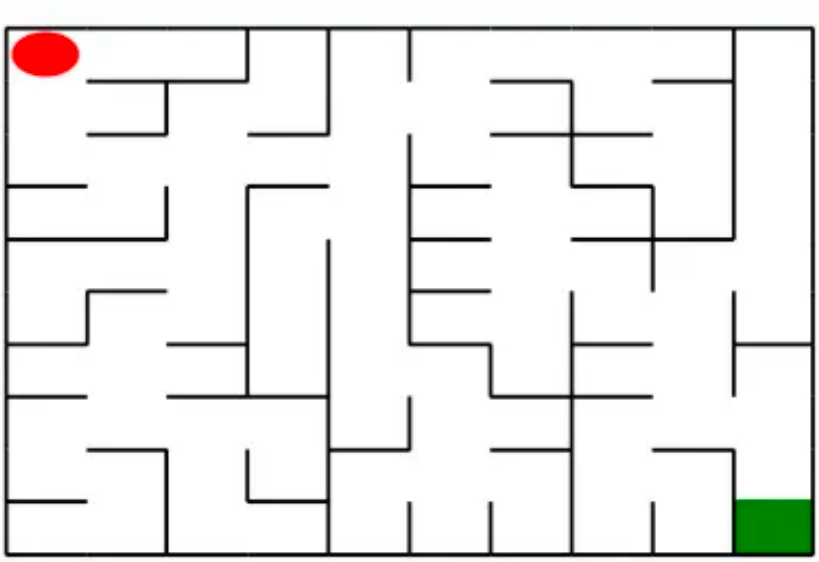
2- 机器人自动走迷宫 - 程序报告

1 实验介绍

1.1 实验内容

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。

1.2 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。

1.3 实验环境

使用 Python 实现基础算法的实现， 使用PyTorch等框架实现 Deep QLearning 算法。

2 实验内容

2.1 基础搜索算法

我选择了深搜，递归形式

```
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",...]
    """

    path = []
    # ----------------请实现你的算法代码--------------------------------------
    visited = {}
    path_t = []
    direction = ['u', 'r', 'd', 'l']


    def reverse(d):
        idx = direction.index(d)
        return direction[(idx + 2) % 4]


    def dfs():
        nonlocal path
        if path:
            return
        location = maze.sense_robot()
        if location == maze.destination:
            path = [_ for _ in path_t]

        visited[location] = True
        for d in maze.can_move_actions(location):
            maze.move_robot(d)
            location = maze.sense_robot()
            if location not in visited:
                path_t.append(d)
                dfs()
                del path_t[-1]
            maze.move_robot(reverse(d))
```

```
    dfs()
    # ----------------------------------------------------------------------
    return path
```

```
搜索出的路径： ['d', 'd', 'r', 'r', 'd', 'r', 'd', 'd', 'r', 'r', 'd', 'r', 'd', 'r', 'd', 'd', 'r', 'r']
恭喜你，到达了目标点
```

2.2 DQN

```
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": 10.,
            "destination": -maze.maze_size ** 2 * 4.,
            "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.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)
            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):
        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
```

```
Training time: 1.58 s
```