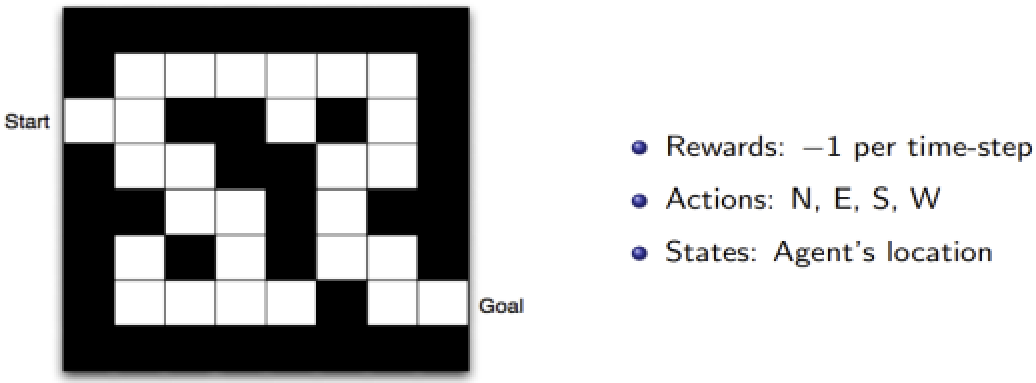
实验报告1

学号	姓名
20337025	崔璨明

1、实验内容

Solve the Maze Problem using Policy Iteration or Value Iteration



2、MDP建模

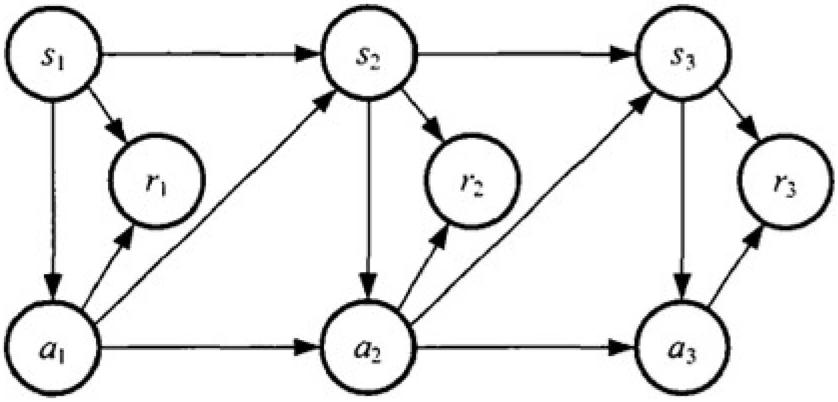
一个马尔可夫决策过程由一个四元组构成(S, A, P_{sa}, R)

- S: 表示状态集（states）
- A:表示一组动作（actions）
- P_{sa}: 表示状态转移概率。表示的是在当前s ∈ S状态下，经过a ∈ A作用后，会转移到的其他状态的概率分布情况。比如，在状态s下执行动作a，转移到s′的概率可以表示为$p(s'|s,a)$
- R: S×A→ℝ，R是回报函数（reward function），回报函数有时也写作状态S的函数（只与S有关），这样的话，R可以简化为R: S→ℝ。

MDP 的动态过程如下：某个智能体(agent)的初始状态为s_0，然后从 A 中挑选一个动作a_0执行，执行后，agent 按P_{sa}概率随机转移到了下一个s_1状态，$s1 \in P_{s_0a_0}$。然后再执行一个动作a_1，就转移到了s_2，接下来再执行a_2...，我们可以用下面的图表示状态转移的过程：

$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} s_3 \xrightarrow{a_3} \ldots$$

如果回报r是根据状态s和动作a得到的，则MDP还可以表示成下图：



设一个状态的价值为$v(s)$，则$v(s)$与当前状态的奖励$R(s)$和此状态即将转移状态的$v(s')$有关，设s转移到s' 的概率为$P(s'|s)$，则bellman equation为：

$$V(s)=R(s)+\gamma\sum_{s'\in S}P(s'|s)V(s')$$

加上action的过程便是MDP，即多了一个动作价值函数$q(s,a)$，价值函数v（s）v(s)v(s)与动作价值函数的关系为：

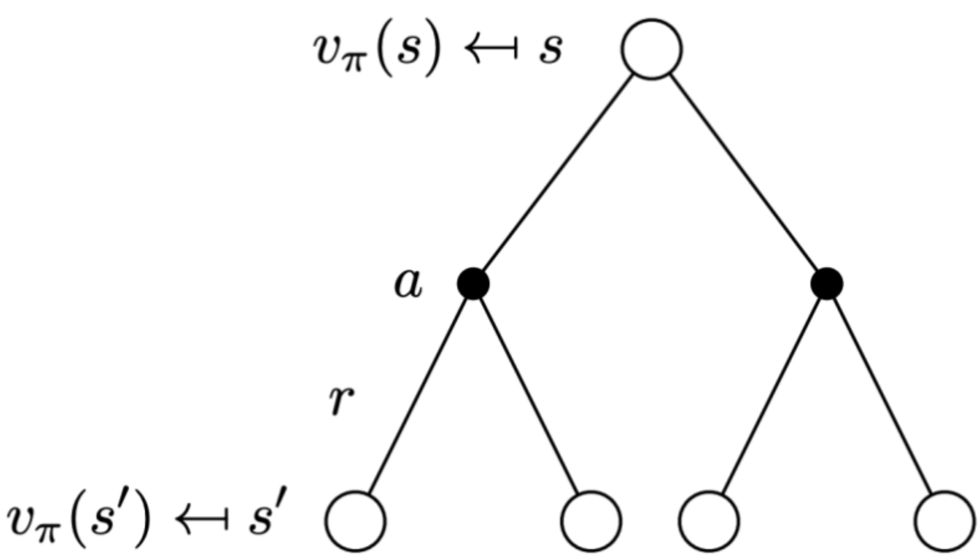
$$V^\pi(s)=\sum_{a\in A}\pi(a|s)q^\pi(s,a)$$

v(s),q(s,a)对应的贝尔曼方程为：

$$V(s)=\sum_{a\in A}\pi(a|s)(R(s)+\gamma\sum_{s'\in S}P(s'|s)V^\pi(s'))$$

$$q^\pi(s,a)=R(s,a)+\gamma\sum_{s'\in S}P(s'|s,a)\sum_{a'\in A}\pi(a'|s')q^\pi(s',a')$$

图形化表示如下：



MDP寻找的是在任意初始条件s下，能够最大化值函数的策略π。*最优策略表示为：*
$$\pi^=argmax_\pi V^\pi(s),(\forall s)

与最优策略$\pi*$对应的状态值函数$V*$与动作值函数\$$Q*$\$之间存在如下关系：

V^=*max_aQ*^(s,a)$$而求解最优策略，则可以用策略迭代（policy iteration）或值迭代（valueiter）的方式，我在本次实验中采用的是值迭代的方式。

3、算法说明

值迭代为了找到最优策略，对每一个当前状态 s ,对每个可能的动作 a 都计算一下采取这个动作后到达的下一个状态的期望价值。看看哪个动作可以到达的状态的期望价值函数最大，就将这个最大的期望价值函数作为当前状态的价值函数 V (s)，循环执行这个步骤，直到价值函数收敛（两次迭代的价值函数不变或变化很小）。

值迭代的算法伪代码如下：

Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
| $\Delta \leftarrow 0$
| Loop for each $s \in \mathcal{S}$:
| $v \leftarrow V(s)$
| $V(s) \leftarrow \max_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
| $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
$\pi(s) = \operatorname{argmax}_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$

4、关键代码展示

根据实验要求和值迭代的算法伪代码编写程序如下，首先建立迷宫的数据结构，具体解析如代码中注释所述：

```
def init():
        #出口，即终止坐标
        terminal_state=(6,8)
        #用numpy数组存储迷宫，1为墙，0为可以到达的地点
        maze = np.array([
            [ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.],
            [ 1.,  0.,  0.,  0.,  0.,  0.,  0.,  1.],
            [ 0.,  0.,  1.,  1.,  0.,  1.,  0.,  1.],
            [ 1.,  0.,  0.,  1.,  1.,  0.,  0.,  1.],
            [ 1.,  1.,  0.,  0.,  1.,  0.,  1.,  1.],
```

```
            [ 1.,  0.,  1.,  0.,  1.,  0.,  0.,  1.],
            [ 1.,  0.,  0.,  0.,  0.,  1.,  0.,  0.],
            [ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.]
        ])
action=[(-1,0),(0,1),(1,0),(0,-1)] #动作，上右下左
num_states=[] #存储所有状态，即可以到达的坐标
#状态转移矩阵的初始化，存储四个动作中可以执行的动作的概率
P_a=np.zeros((dim*dim,4))
    for i in range(dim):
        for j in range(dim):
            if(maze[i][j]==0):
                num_states.append((i,j))
    for i in range(dim*dim):
        x=int(i/dim)
        y=i%dim
        #不能到达的地方就跳过
        if(maze[x][y]==1):
            continue
        sum=0
        for k in range(len(action)):
            if((x+action[k][0])<0 or (x+action[k][0])>=8 or (y+action[k][1])<0 or 
(y+action[k][1])>=8):
                continue
            if (maze[x+action[k][0]][y+action[k][1]]==0):
                P_a[i][k]=1.
                sum+=1.
        for k in range(len(action)):
            if (P_a[i][k]==1):
                P_a[i][k]/=sum
    P_a[55][1]=0.5
    P_a[55][3]=0.5
```

接着是值迭代的算法过程，分为两部分，首先是循环计算状态价值函数，收敛后再进行一次策略的更新，选择转移至状态价值大的状态的动作作为最优策略的动作，终止状态的reward设置为20000，其他的reward都是-1，代码详细解析见注释：

```
def value_iteration(gamma = 0.9, num_of_iterations = 10000):
    N_STATES=len(num_states)#状态数
    N_ACTIONS=len(action)#动作数
    error=0.0001#判断收敛
    values = np.zeros(N_STATES)#状态价值
    rewards=[-1]*N_STATES  #每个动作的回报
    #值迭代的过程，设定最大迭代次数
    for i in range(num_of_iterations):
        values_tmp = values.copy()
```

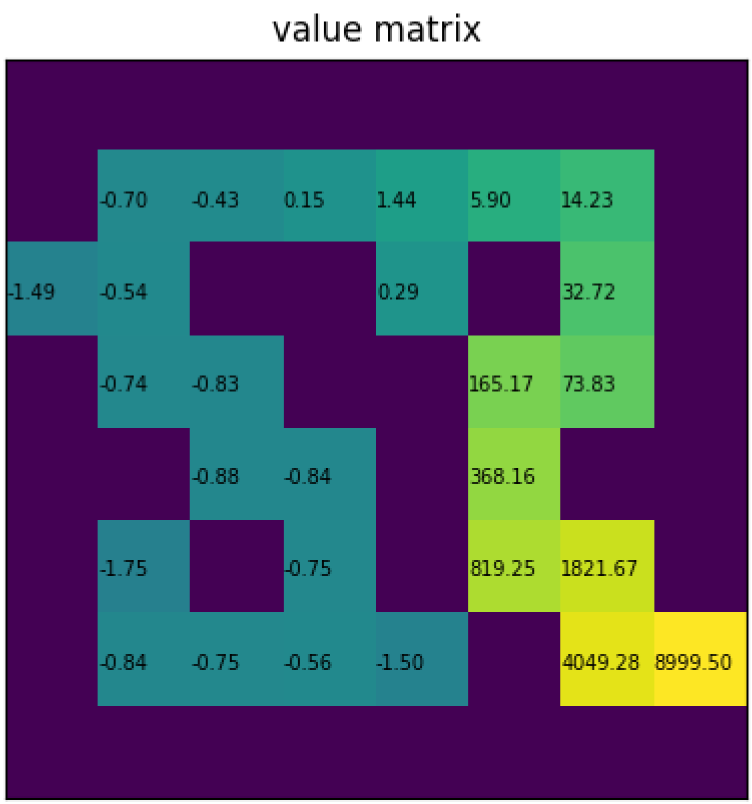
```
        #对每一个状态进行更新
        for idx in range(N_STATES):
            v_a = []
            s=num_states[idx][0]*dim+num_states[idx][1]
                #对每一个动作都计算
            for a in range(N_ACTIONS):
                    #如果是合法动作（即可以到达）
                if(P_a[s][a]!=0):
                    next_x=num_states[idx][0]+action[a][0]
                    next_y=num_states[idx][1]+action[a][1]
                    #终止状态，设置回报为20000
                    if(next_x==terminal_state[0] and next_y==terminal_state[1]):
                        v_a.append(P_a[s][a]*(rewards[idx]+gamma*20000))
                    #其他的回报都是-1
                    else:
                        s1=num_states.index((next_x,next_y))
                        v_a.append(P_a[s][a]*(rewards[idx]+gamma*values_tmp[s1]))
            values[idx]=max(v_a)    #选择最大的进行更新
        #收敛了则退出
        if max([abs(values[s] - values_tmp[s]) for s in range(N_STATES)]) < error:
            break

#p进行策略的更新
    policy = np.zeros([N_STATES])
    for idx in range(N_STATES):
        the_max=-99999.
        cx=num_states[idx][0]
        cy=num_states[idx][1]
        s=num_states[idx][0]*dim+num_states[idx][1]
        flag=-1
        #遍历所有动作，选择可以带来新的状态的状态价值最大的
        for a in range(N_ACTIONS):
            if(P_a[s][a]!=0):
               nx=cx+action[a][0]
               ny=cy+action[a][1]
               if(nx==terminal_state[0] and ny==terminal_state[1]):
                the_max=P_a[s][a]*(rewards[idx]+gamma*20000)
                flag=a
                continue
               s1=num_states.index((nx,ny))
               if(P_a[s][a]*(rewards[idx]+gamma*values[s1])>the_max):
                the_max=P_a[s][a]*(rewards[idx]+gamma*values[s1])
                flag=a
        #设置为最大者
        policy[idx]=flag
```

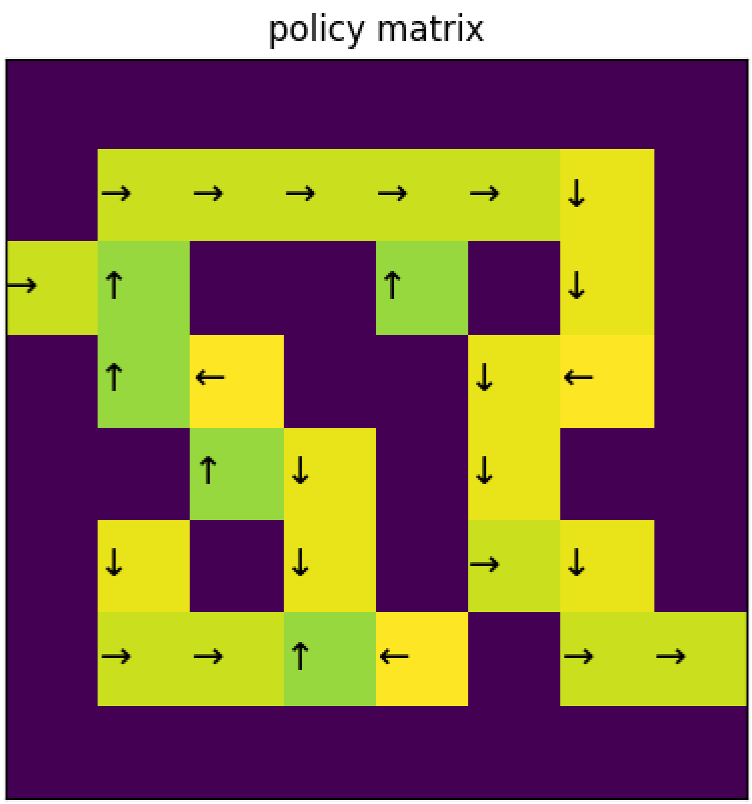
```
    #返回状态价值函数和最优策略
    return values,policy
```

5、实验结果

在实验题目的迷宫中运行程序，最终得到的状态价值函数和策略如下：
状态价值函数：



最终的策略，箭头代表移动方向：




```
PS D:\reforce_learnig\task> & E:/python/python.exe d:/reforce_learnig/task/value_iteration.py
 NAN   NAN   NAN   NAN   NAN   NAN   NAN   NAN
 NAN  -0.70 -0.43 0.15  1.44  5.90  14.23  NAN
-1.49 -0.54  NAN   NAN  0.29   NAN  32.72  NAN
 NAN  -0.74 -0.83  NAN   NAN  165.17 73.83  NAN
 NAN   NAN  -0.88 -0.84  NAN  368.16  NAN   NAN
 NAN  -1.75  NAN  -0.75  NAN  819.25 1821.67  NAN
 NAN  -0.84 -0.75 -0.56 -1.50  NAN  4049.28 8999.50
 NAN   NAN   NAN   NAN   NAN   NAN   NAN   NAN
* * * * * * * *
* → → → → → ↓ *
→ ↑ * * ↑ * ↓ *
* ↑ ← * * ↓ ← *
* * ↑ ↓ * ↓ * *
* ↓ * ↓ * → ↓ *
* → → ↑ ← * → →
* * * * * * * *
```

6、实验心得

通过此次实验我复习了课上所学的内容，并能将理论知识运用到实践中，通过自己编写程序来解决特定问题的这种方式，我巩固了所学内容，并锻炼了自己的实践能力，这让我受益匪浅。