作业 3：MC_off_policy_赛道问题实验报告

人工智能 91 卢佳源 2191121196

一、 实验目的：

a) 理解蒙特卡罗方法原理，以及同轨、离轨策略的过程；
b) 代码实现蒙特卡洛离轨策略算法，解决赛道问题。

二、 问题重述：赛道问题

a) 离散赛道、离散速度，速度分为水平和垂直方向；
b) 每一步（时刻）：
   i. 速度分量取 0~4 范围内的整数值，起点线两个速度分量为 0，其他位置两个速度分量不能同时为 0；
   ii. 速度变化量取-1，1 或 0；
   iii. 每步收益为-1；
   iv. 不管预期的速度增量是多少，每步中两个方向上的速度增量有 0.1 的概率可能同时为 0（随机噪声）；
c) 约束（每一幕）：
   i. 赛车接触终点线时一幕结束；
   ii. 赛车在接触终点线之前触碰到赛道边缘，则会被重置到起点线的一个随机位置，两个速度分量置零，且该幕继续；
   iii. 更新赛车位置前，先判断赛车预计的路径是否与赛道边缘相交，然后按照上述两条约束进行状态更新；
d) 动作：
   i. 每一步（时刻）两个速度分量的变化量（包括-1，+1 和 0），一共 9 种动作；
e) 状态：
   i. 赛车位于赛道的位置（水平+垂直）；
   ii. 赛车当前的速度分量（水平+垂直）；
f) 目的：
   i. 使用蒙特卡洛控制方法；
   ii. 任务：赛车从起点线到终点线，不触碰除终点线外的其他赛道边缘；
   iii. 计算赛道任务中的最优策略并可视化展示最优策略的一些轨迹；
   iv. 不考虑随机噪声。

三、 实验环境：

a) IDE：VSCode，Python-3.9.7；
b) 编程语言：Python；
c) 文件路径：C:\Users\jiayuan lu\OneDrive - MSRA\桌面\大三下\RL\作业 3 MC_5_12_race\MC.ipynb 和 MC2.ipynb；

四、 实验原理和思路：

a) 初始化需要用到的变量：状态、策略、动作、状态价值、权值累加和等；

b) 描述赛道形状：

赛道组成	内部	边缘	起点线	终点线	外部
赋值	1	-10	50	-50	0

（赋值仅作为标记，与 MC 算法中的参数无关）

c) 从动作集合中选择当前状态下可以选择的动作：

i. 由于题目有速度大小的限制：0~4，且除起点线外两个速度分量不能同时为 0，因此我们需要过滤掉动作集合中的一些不符合要求的动作；

d) 对于每一幕：（无限循环）

i. 生成软性策略和一幕数据：

1. 根据下述软性策略的定义公式生成每一幕的初始策略：

$$\pi(\mathrm{a}|S_t) \leftarrow \begin{cases} 1-\varepsilon+\dfrac{\varepsilon}{|A(S_t)|} & if\ a = A^* \\ \dfrac{\varepsilon}{|A(S_t)|} & if\ a \neq A^* \end{cases}$$

2. 保存生成的软性策略数组 P（代码中表示为字母 A，此处为避免与动作 A 混淆，用字母 P 代替），以及对应的概率数组 B；
3. 根据“问题重述”中的约束条件更新速度分量和状态；
4. 上述过程根据软性策略 P 生成了一幕数据：$S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$
5. 令两个离轨策略的参数：汇报序列 G 为 0，随机权重 W 为 0；

ii. 对上述生成的一幕中的每一时刻循环，t 从 T-1 时刻逆推到 0 时刻：

1. 根据以下公式进行参数的更新：

$$\mathrm{G} \leftarrow \Upsilon \mathrm{G} + R_{t+1}$$

$$\mathrm{C}(S_t, A_t) \leftarrow \mathrm{C}(S_t, A_t) + W$$

$$\mathrm{Q}(S_t, A_t) \leftarrow \mathrm{Q}(S_t, A_t) + \frac{W}{\mathrm{C}(S_t, A_t)}[G - \mathrm{Q}(S_t, A_t)]$$

$$\pi(S_t) \leftarrow argmax_a \mathrm{Q}(S_t, \mathrm{a})$$

2. 当$A_T \neq \pi(S_t)$时退出每一幕的循环，开始处理下一幕的数据；
3. 并按照如下公式更新权重 W：

$$\mathrm{W} \leftarrow \frac{W}{\mathrm{p}(A_t|S_t)}$$

iii. 按照上述算法得到的最优策略绘制轨迹图。

五、 实验代码：

```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import random


rows_1=32
cols_1=17
rows_2=30
cols_2=32
epsilon=0.1
gamma=1
reward_step=-1
action=[[-1,-1],[-1,0],[-1,1],[0,-1],[0,0],[0,1],[1,-1],[1,0],[1,1]]
velocity_limit=5
Q=np.zeros((rows_1,cols_1,velocity_limit,velocity_limit,len(action)))
policy1=np.zeros((rows_1,cols_1,velocity_limit,velocity_limit),dtype=in
t)
policy2=np.zeros((rows_1,cols_1,velocity_limit,velocity_limit),dtype=in
t)
policy_1={}
policy_2={}
for i in range(rows_1):
    for j in range(cols_1):
        for m in range(velocity_limit):
            for n in range(velocity_limit):
                policy_1[i,j,m,n]=[policy1[i,j,m,n],policy2[i,j,m,n]]
                p=[]
                for k in range(len(action)):
                    p.append(Q[i,j,m,n,k])
                policy_1[i,j,m,n]=action[np.argmax(p)]
C=np.zeros(Q.shape,dtype=float)
b=np.zeros(Q.shape,dtype=int)
state={}
A={}
B={}


race_track1=np.zeros((rows_1,cols_1),dtype=int)
race_track1[0,3:]=1
race_track1[0,3:]=-10  #boundary
race_track1[1:3,2:]=1
```

```
race_track1[1:3,2]=-10
race_track1[3,1:]=1
race_track1[3,1]=-10
race_track1[4:6,]=1
race_track1[4:6,0]=-10
race_track1[5,10:]=-10
race_track1[6,:10]=1
race_track1[6,0]=-10
race_track1[6,9]=-10
race_track1[7:14,:9]=1
race_track1[7:14,0]=-10
race_track1[14:22,1:9]=1
race_track1[14:22,1]=-10
race_track1[22:29,2:9]=1
race_track1[22:29,2]=-10
race_track1[29:,3:9]=1
race_track1[29:32,3]=-10
race_track1[7:,8]=-10
race_track1[31,3:9]=50
start=race_track1[31,3:9]
race_track1[0:6,16]=-50
finish=race_track1[0:6,16]


def valid_action(t,state):
    valid=[]
    for i in range(-velocity_limit+1,velocity_limit):
        for j in range(-velocity_limit+1,velocity_limit):
            if 0<=i+state[t][2]<=4 and 0<=j+state[t][3]<=4 and not
(i+state[t][2]==0 and j+state[t][3]==0):
                if -1<=i<=1 and -1<=j<=1:
                    valid.append([i,j])
    return valid


def policy_maker1():
    t=0
    state[t]=[]
    start_row=31
    start_col=np.random.randint(3,9)
    start_vel_row=1
    start_vel_col=1
    state[t].append(start_row)
    state[t].append(start_col)
```

```
    state[t].append(start_vel_row)
    state[t].append(start_vel_col)
    while True:
        s=state[t]
        c=[]
        val_act=valid_action(t,state)
        for i in range (0,len(val_act)):
            c.append(i)
        pr=[]
        for i in range(len(val_act)):
            pr.append(1/len(val_act))
        act_num=len(val_act)
        pi=policy_1[state[t][0],state[t][1],state[t][2],state[t][3]]
        if np.random.random()>=epsilon:
            if pi in val_act:
                a=pi
                b=1-epsilon+epsilon/act_num
            else:
                a=val_act[(np.random.choice(c, p=pr))]
                b=1/act_num
        else:
            a=val_act[(np.random.choice(c, p=pr))]
            if pi in val_act:
                b=epsilon/act_num
            else:
                b=1/act_num
        A[t]=a
        B[t]=b
        act_choose=a
        vel_next=[state[t][2]+act_choose[0],state[t][3]+act_choose[1]]
        if 0<=state[t][0]-vel_next[1]-1<rows_1 and
0<=state[t][1]+vel_next[0]-1<17 and not(vel_next[0]==0 and
vel_next[1]==0) and 0<=vel_next[0]<=4 and 0<=vel_next[1]<=4:
            next=[state[t][0]-vel_next[1]-1,state[t][1]+vel_next[0]-
1,vel_next[0],vel_next[1]]
            if race_track1[next[0],next[1]]==1:
                state_next=next
                s=state_next
            elif race_track1[next[0],next[1]]==-50:
                return t,state
            elif race_track1[next[0],next[1]]==-10:
                start_col=np.random.randint(3,9)
                state_next=[start_row,start_col,start_vel_row,start_vel_
col]
```

```
                s=state_next
            elif race_track1[next[0],next[1]]==0:
                start_col=np.random.randint(3,9)
                state_next=[start_row,start_col,start_vel_row,start_vel_
col]
                s=state_next
            else:
                state_next=s
                s=state_next
        else:
            state_next=[state[t][0],state[t][1],1,1]
            s=state_next
        t+=1
        state[t]=[]
        state[t].append(s[0])
        state[t].append(s[1])
        state[t].append(s[2])
        state[t].append(s[3])


def for_each_episode(T,state):
    G=0.0
    W=1.0
    x=[]
    y=[]
    for t in range(T-1,-1,-1):
        G=gamma*G+reward_step
        C[state[t][0],state[t][1],state[t][2],state[t][3],A[t]]+=W
        Q[state[t][0],state[t][1],state[t][2],state[t][3],A[t]]+=W*(G-
Q[state[t][0],state[t][1],state[t][2],state[t][3],A[t]])/C[state[t][0],
state[t][1],state[t][2],state[t][3],A[t]]
        val_act=valid_action(t,state)
        policy_1[state[t][0],state[t][1],state[t][2],state[t][3]]=val_ac
t[np.argmax(Q[state[t][0],state[t][1],state[t][2],state[t][3],:])]
        x.append(state[t][0])
        y.append(state[t][1])
        if
A[t]==policy_1[state[t][0],state[t][1],state[t][2],state[t][3]]:
            return t,x,y
        W/=B[t]


def track_para():
    for i in range(2):
```

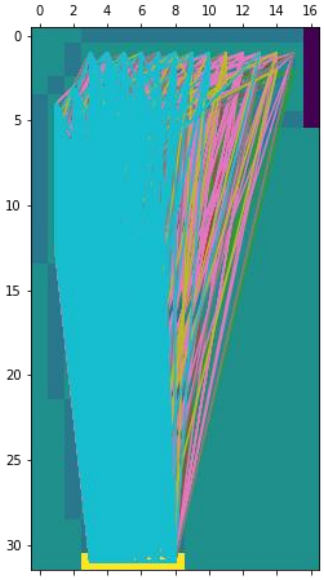
```
        T,state_update=policy_maker1()
        print("trace:")
        for j in range(T):
            print("S:",state_update[j])
            print("A:",A[j])
        print("R:",reward_step*T)


if __name__ == '__main__':
    episode_num=30
    reward=[]
    plt.figure()
    plt.matshow(race_track1)
    for i in range(episode_num):
        x=[]
        y=[]
        T,state_update=policy_maker1()
        t,_,__=for_each_episode(T,state_update)
        for j in range(T-1,t-10,-1):
            x.append(state_update[j][0])
            y.append(state_update[j][1])
        print(t)
        print(x)
        print(y)
        plt.plot(np.transpose(y),np.transpose(x))
        plt.savefig('trace1.png',bbox_inches='tight')
        print("episode"+str(i)+":"+str(T)+","+str(t)+","+str(T-t))
        if i%9==0:
            T,state_update=policy_maker1()
            reward.append(reward_step*T)
   track_para()
    plt.show()
```

六、 实验结果：

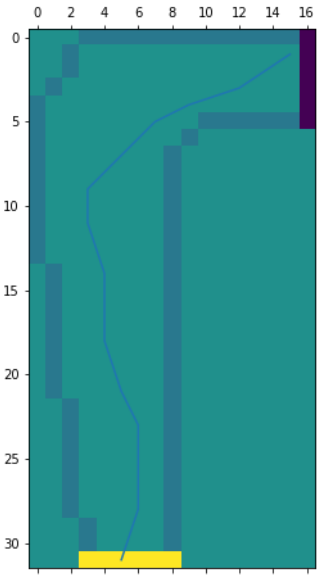
a) 实验结果：

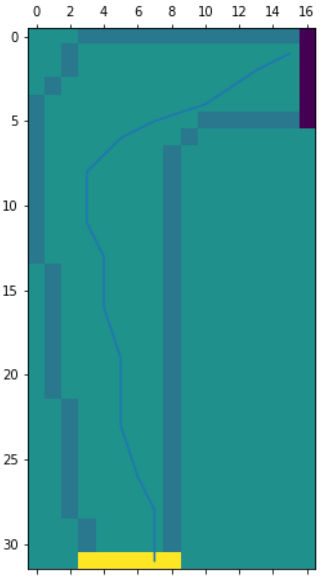
i. 赛道 1：

每一幕所有状态的轨迹图：



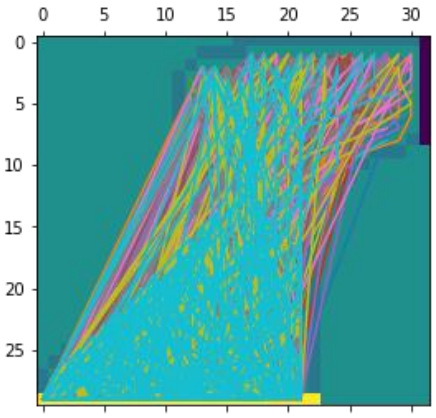
其中几条最优策略轨迹图：



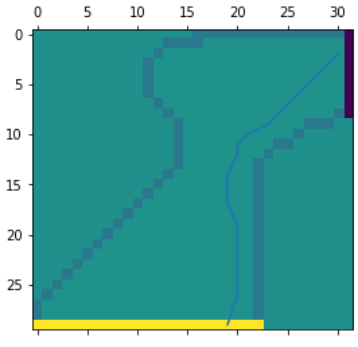


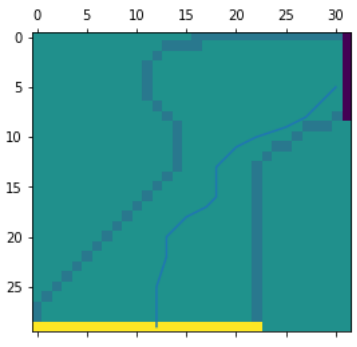
ii. 赛道 2：

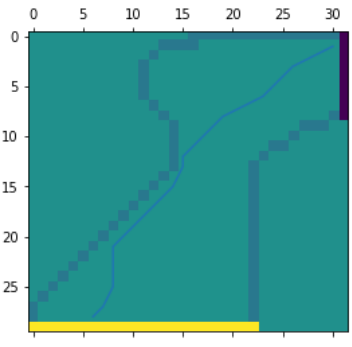
每一幕所有状态的轨迹图：



其中几条最优策略轨迹图：







b) 实验结果分析:

i. 对两种赛道，最优策略都不是唯一的，其中的随机性有以下几点:

1. 起点线的初始位置的随机性;
2. 软性策略的随即性质;
3. 每一步选择动作是在符合要求的动作集合里面随机选的;

ii. 根据软性策略的ε的大小不同, 生成的最优策略轨迹也不相同, 总结其中规律:

1. ε越大，软性越强，最优策略轨迹的弯曲点越多，反之，最优策略轨迹越平滑;

iii. 根据不同的随机起始点，算法得到的轨迹图都是尽量以最小的步数（最小的代价，最高的累积奖励）求得最终的最优策略轨迹。

iv. 根据代码的输出结果, 生成的一幕数据大概需要几百至几千的步数(时刻数);

七、 实验反思:

a) 软性策略的ε会影响最优策略轨迹的平滑程度，ε越小，轨迹曲线越平滑;

b) 生成每一幕的整体轨迹图后很难分离出其中一条最优策略轨迹曲线, 需要在每一幕的其中一段调整时间段长度来截取一条轨迹曲线。