**作业3：MC\_off\_policy\_赛道问题实验报告**

**人工智能91 卢佳源 2191121196**

1. **实验目的：**
   1. 理解蒙特卡罗方法原理，以及同轨、离轨策略的过程；
   2. 代码实现蒙特卡洛离轨策略算法，解决赛道问题。
2. **问题重述：赛道问题**
   1. 离散赛道、离散速度，速度分为水平和垂直方向；
   2. 每一步（时刻）：
      1. 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；
      2. 速度变化量取-1，1或0；
      3. 每步收益为-1；
      4. 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）；
   3. 约束（每一幕）：
      1. 赛车接触终点线时一幕结束；
      2. 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；
      3. 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；
   4. 动作：
      1. 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；
   5. 状态：
      1. 赛车位于赛道的位置（水平+垂直）；
      2. 赛车当前的速度分量（水平+垂直）；
   6. 目的：
      1. 使用蒙特卡洛控制方法；
      2. 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；
      3. 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；
      4. 不考虑随机噪声。
3. **实验环境：**
   1. IDE：VSCode，Python-3.9.7；
   2. 编程语言：Python；
   3. 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；
4. **实验原理和思路：**
   1. 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；
   2. 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）

* 1. 从动作集合中选择当前状态下可以选择的动作：
     1. 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；
  2. 对于每一幕：（无限循环）
     1. 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：
        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：
        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；
     2. 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：
        2. 当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：
     3. 按照上述算法得到的最优策略绘制轨迹图。

1. **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=int)

policy2=np.zeros((rows\_1,cols\_1,velocity\_limit,velocity\_limit),dtype=int)

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\_act[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()

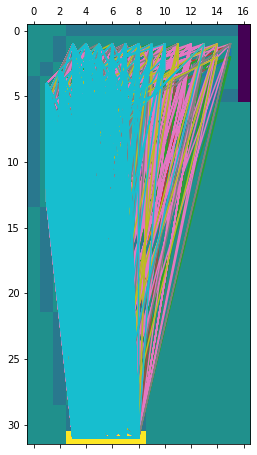
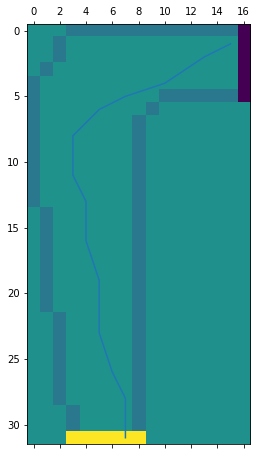
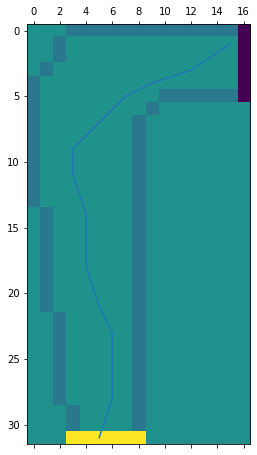
            reward.append(reward\_step\*T)

   track\_para()

    plt.show()

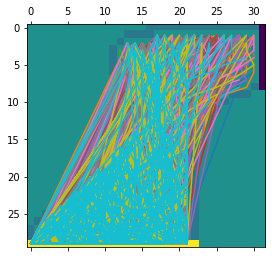
1. **实验结果：**
   1. 实验结果：
      1. 赛道1：

每一幕所有状态的轨迹图： 其中几条最优策略轨迹图：

* + 1. 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：

* 1. 实验结果分析：
     1. 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；
     2. 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；
     3. 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。
     4. 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）；

1. **实验反思：**
   1. 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；
   2. 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。