Machine Data and Learning
SPRING SEMESTER 2020

Team 89
Yash Bhansali (2018101068) and Sridhar M (2018111021)

ASSIGNMENT 2, March 7

TASK-1

Libraries used :

- Numpy
- time

Description of algorithm :

- **Value iteration algorithm**

1: **Procedure** Value_Iteration(S,A,P,R,θ)
2: **Inputs**
3: S is the set of all states
4: A is the set of all actions
5: P is state transition function specifying $P(s'|s,a)$
6: R is a reward function $R(s,a,s')$
7: θ a threshold, $\theta>0$
8: **Output**
9: $\pi[S]$ approximately optimal policy
10: $V[S]$ value function
11: **Local**
12: real array $V_k[S]$ is a sequence of value functions
13: action array $\pi[S]$
14: assign $V_0[S]$ arbitrarily
15: $k \leftarrow 0$
16: **repeat**
17: $k \leftarrow k+1$
18: **for each** state s **do**
19: $V_k[s] = max_a \sum_{s'} P(s'|s,a)\,(R(s,a,s')+ \gamma V_{k-1}[s'])$
20: **until** $\forall s\,|V_k[s]-V_{k-1}[s]| < \theta$
21: **for each** state s **do**
22: $\pi[s] = argmax_a \sum_{s'} P(s'|s,a)\,(R(s,a,s')+ \gamma V_k[s'])$
23: **return** π, V_k

- **Markov's decision process :**

```
import numpy as np
import time
states = []
Ut = {}
Ut1 = {}

step_cost  = -20
shoot_step_cost = -20
df=0.99
delta = 0.001
for i in range(5):
    for j in range(4):
        for k in range(3):
            states.append((i,j,k))#(health,arrow,stamina)
            Ut[(i,j,k)]=0
            Ut1[(i,j,k)]=0

itr=0
k=0
```

```
while True:
    print("iteration="+str(itr))
    itr+=1
    for j in states:
        if(j[0]==0):
            Ut1[j]=0
            k+=1
            print(str(j)+':-1=[0.]')
            continue
        #shoot
        shoot=-1000000
        if(j[1]>0 and j[2]>0):
            shoot = shoot_step_cost*0.5 + (shoot_step_cost+10)*0.5 + df*(0.5*Ut[(j[0],j[1]-1,j[2]-1)] + 0.5*Ut[(j[0]-1,j[1]-1,j[2]-1)])

        #recharge
        recharge = -1000000
        if(j[2]==2):
            recharge = step_cost+ df*Ut[j]
        else:
            recharge = step_cost + df*(0.8*Ut[(j[0],j[1],j[2]+1)] + 0.2*Ut[j])

        #dodge
        dodge = -1000000
        if(j[2]>0):
            if(j[2]==2):
                dodge = step_cost + df*(0.8*0.8*Ut[(j[0],min(3,j[1]+1),1)] + 0.8*0.2*Ut[(j[0],j[1],1)] + 0.2*0.8*Ut[(j[0],min(3,j[1]+1),0)] + 0.2*0.2*Ut[(j[0],j[1],0)])
            else:
                dodge = step_cost + df*(0.8*Ut[(j[0],min(3,j[1]+1),0)] + 0.2*Ut[(j[0],j[1],0)])

        mak = max(shoot,recharge,dodge)
        Ut1[j]=mak
        if(Ut[j]-Ut1[j]<delta):
            k+=1

        if(mak==shoot):
            print('(' + str(j[0]) + ',' + str(j[1])+','+str(j[2])+"):SHOOT=[" + str('{0:.8f}'.format(mak)) + ']')
        elif(mak==recharge):
            print('(' + str(j[0]) + ',' + str(j[1])+','+str(j[2])+"):RECHARGE=[" + str('{0:.8f}'.format(mak)) + ']')
        else:
            print('(' + str(j[0]) + ',' + str(j[1])+','+str(j[2])+"):DODGE=[" + str('{0:.8f}'.format(mak)) + ']')
    for key in Ut:
        Ut[key] = Ut1[key]
```

Final observation:

The actions (shown below) are quite close to the obvious greedy policy (e.g. things like - shoot if you have arrows and dragon is close to dying, dodge if you need arrows or if you have lesser stamina and the dragon is still not close to dying and recharge definitely when you have 0 stamina) since the value function of the game is very smooth

iteration=111

(0,0,0):-1=[0.]

(0,0,1):-1=[0.]

(0,0,2):-1=[0.]

(0,1,0):-1=[0.]

(0,1,1):-1=[0.]

(0,1,2):-1=[0.]

(0,2,0):-1=[0.]

(0,2,1):-1=[0.]

(0,2,2):-1=[0.]

(0,3,0):-1=[0.]

(0,3,1):-1=[0.]

(0,3,2):-1=[0.]

(1,0,0):RECHARGE=[-38.016]

(1,0,1):DODGE=[-32.183]

(1,0,2):DODGE=[-27.457]

(1,1,0):RECHARGE=[-24.818]

(1,1,1):SHOOT=[-18.818]

(1,1,2):SHOOT=[-15.930]

(1,2,0):RECHARGE=[-18.366]

(1,2,1):SHOOT=[-12.285]

(1,2,2):SHOOT=[-9.315]

(1,3,0):RECHARGE=[-15.212]

(1,3,1):SHOOT=[-9.091]

(1,3,2):SHOOT=[-6.081]

(2,0,0):RECHARGE=[-81.594]

(2,0,1):DODGE=[-76.311]

(2,0,2):DODGE=[-72.031]

(2,1,0):RECHARGE=[-69.641]

(2,1,1):SHOOT=[-64.207]

(2,1,2):SHOOT=[-58.704]

(2,2,0):RECHARGE=[-57.346]

(2,2,1):SHOOT=[-51.757]

(2,2,2):SHOOT=[-46.097]

(2,3,0):RECHARGE=[-48.182]

(2,3,1):SHOOT=[-42.477]

(2,3,2):SHOOT=[-36.701]

(3,0,0):RECHARGE=[-121.060]

(3,0,1):DODGE=[-116.276]

(3,0,2):DODGE=[-112.400]

(3,1,0):RECHARGE=[-110.235]

(3,1,1):SHOOT=[-105.314]

(3,1,2):SHOOT=[-100.330]

(3,2,0):RECHARGE=[-99.100]

(3,2,1):SHOOT=[-94.038]

(3,2,2):SHOOT=[-88.913]

(3,3,0):RECHARGE=[-87.647]

(3,3,1):SHOOT=[-82.441]

(3,3,2):SHOOT=[-77.168]

(4,0,0):RECHARGE=[-156.798]

(4,0,1):DODGE=[-152.466]

(4,0,2):DODGE=[-148.956]

(4,1,0):RECHARGE=[-146.996]

(4,1,1):SHOOT=[-142.539]

(4,1,2):SHOOT=[-138.027]

(4,2,0):RECHARGE=[-136.912]

(4,2,1):DODGE=[-132.329]

(4,2,2):SHOOT=[-127.687]

(4,3,0):RECHARGE=[-126.541]

(4,3,1):SHOOT=[-121.826]

(4,3,2):SHOOT=[-117.051]

TASK-2

Final Observations

Subtask 1: New Step Rewards

The Convergence of this function is much faster than was before (in task 1), the lowered step costs and the **favored shoot action** allows the model to learn the kill technique much faster, which it later slowly changes to better action combinations involving dodge and recharge, improving the score further. There is more incentive to be greedy and shoot at the beginning.

Subtask 2: The High Future Discount

The **huge discount factor lowers the incentive** to look and try to get the big reward (killing the dragon) in the future. The agent will only look 1 move deep and only try if the dragon has health and it has both arrows and stamina, otherwise it will give up. All moves / futures look relatively equal.

Subtask 3: Complete Convergence

This change has no effect as compared to Subtask 2, since the agent has already almost completely discounted the future. The policy will not change as we hone down more on the utility values as the agent himself does not value anything in the future, changing convergence (δ) to 10^ − 3 to 10^ − 10 are both almost equally good. It just takes a few more iterations to get there.

Decreasing bellman factor only changes number of iterations used without much change in utility value

Result of first part:

iteration=99
(0,0,0):-1=[0.]
(0,0,1):-1=[0.]
(0,0,2):-1=[0.]
(0,1,0):-1=[0.]
(0,1,1):-1=[0.]
(0,1,2):-1=[0.]
(0,2,0):-1=[0.]
(0,2,1):-1=[0.]
(0,2,2):-1=[0.]
(0,3,0):-1=[0.]
(0,3,1):-1=[0.]
(0,3,2):-1=[0.]
(1,0,0):RECHARGE=[-10.317]
(1,0,1):DODGE=[-7.291]
(1,0,2):DODGE=[-4.839]
(1,1,0):RECHARGE=[-3.470]
(1,1,1):SHOOT=[-0.357]
(1,1,2):SHOOT=[1.141]
(1,2,0):RECHARGE=[-0.123]

(1,2,1):SHOOT=[3.032]

(1,2,2):SHOOT=[4.573]

(1,3,0):RECHARGE=[1.514]

(1,3,1):SHOOT=[4.689]

(1,3,2):SHOOT=[6.251]

(2,0,0):RECHARGE=[-28.809]

(2,0,1):DODGE=[-26.016]

(2,0,2):DODGE=[-23.754]

(2,1,0):RECHARGE=[-22.490]

(2,1,1):SHOOT=[-19.617]

(2,1,2):SHOOT=[-16.737]

(2,2,0):RECHARGE=[-16.054]

(2,2,1):SHOOT=[-13.100]

(2,2,2):SHOOT=[-10.137]

(2,3,0):RECHARGE=[-11.272]

(2,3,1):SHOOT=[-8.257]

(2,3,2):SHOOT=[-5.233]

(3,0,0):RECHARGE=[-45.556]

(3,0,1):DODGE=[-42.975]

(3,0,2):DODGE=[-40.884]

(3,1,0):RECHARGE=[-39.716]

(3,1,1):SHOOT=[-37.061]

(3,1,2):SHOOT=[-34.400]

(3,2,0):RECHARGE=[-33.772]

(3,2,1):SHOOT=[-31.042]

(3,2,2):SHOOT=[-28.306]

(3,3,0):RECHARGE=[-27.720]

(3,3,1):SHOOT=[-24.914]

(3,3,2):SHOOT=[-22.100]

(4,0,0):RECHARGE=[-60.717]

(4,0,1):DODGE=[-58.328]

(4,0,2):DODGE=[-56.393]

(4,1,0):RECHARGE=[-55.312]

(4,1,1):SHOOT=[-52.855]

(4,1,2):SHOOT=[-50.395]

(4,2,0):RECHARGE=[-49.815]

(4,2,1):SHOOT=[-47.288]

(4,2,2):SHOOT=[-44.758]

(4,3,0):RECHARGE=[-44.223]

(4,3,1):SHOOT=[-41.625]

(4,3,2):SHOOT=[-39.023]

Result of Second part:

iteration=4

(0,0,0):-1=[0.]

(0,0,1):-1=[0.]

(0,0,2):-1=[0.]

(0,1,0):-1=[0.]

(0,1,1):-1=[0.]

(0,1,2):-1=[0.]

(0,2,0):-1=[0.]

(0,2,1):-1=[0.]

(0,2,2):-1=[0.]

(0,3,0):-1=[0.]

(0,3,1):-1=[0.]

(0,3,2):-1=[0.]

(1,0,0):RECHARGE=[-2.775]

(1,0,1):DODGE=[-2.744]

(1,0,2):DODGE=[-2.442]

(1,1,0):RECHARGE=[-2.358]

(1,1,1):SHOOT=[2.361]

(1,1,2):SHOOT=[2.363]

(1,2,0):RECHARGE=[-2.357]

(1,2,1):SHOOT=[2.382]

(1,2,2):SHOOT=[2.618]

(1,3,0):RECHARGE=[-2.357]

(1,3,1):SHOOT=[2.382]

(1,3,2):SHOOT=[2.619]

(2,0,0):RECHARGE=[-2.778]

(2,0,1):RECHARGE=[-2.778]

(2,0,2):RECHARGE=[-2.778]

(2,1,0):RECHARGE=[-2.778]

(2,1,1):SHOOT=[-2.778]

(2,1,2):SHOOT=[-2.776]

(2,2,0):RECHARGE=[-2.776]

(2,2,1):SHOOT=[-2.757]

(2,2,2):SHOOT=[-2.521]

(2,3,0):RECHARGE=[-2.776]

(2,3,1):SHOOT=[-2.757]

(2,3,2):SHOOT=[-2.519]

(3,0,0):RECHARGE=[-2.778]

(3,0,1):RECHARGE=[-2.778]

(3,0,2):RECHARGE=[-2.778]

(3,1,0):RECHARGE=[-2.778]

(3,1,1):SHOOT=[-2.778]

(3,1,2):SHOOT=[-2.778]

(3,2,0):RECHARGE=[-2.778]

(3,2,1):SHOOT=[-2.778]

(3,2,2):SHOOT=[-2.778]

(3,3,0):RECHARGE=[-2.778]

(3,3,1):SHOOT=[-2.778]

(3,3,2):SHOOT=[-2.777]

(4,0,0):RECHARGE=[-2.778]

(4,0,1):RECHARGE=[-2.778]

(4,0,2):RECHARGE=[-2.778]

(4,1,0):RECHARGE=[-2.778]

(4,1,1):SHOOT=[-2.778]

(4,1,2):SHOOT=[-2.778]

(4,2,0):RECHARGE=[-2.778]

(4,2,1):SHOOT=[-2.778]

(4,2,2):SHOOT=[-2.778]

(4,3,0):RECHARGE=[-2.778]

(4,3,1):SHOOT=[-2.778]

(4,3,2):SHOOT=[-2.778]

Result of Third part:

iteration=11

(0,0,0):-1=[0.]

(0,0,1):-1=[0.]

(0,0,2):-1=[0.]

(0,1,0):-1=[0.]

(0,1,1):-1=[0.]

(0,1,2):-1=[0.]

(0,2,0):-1=[0.]

(0,2,1):-1=[0.]

(0,2,2):-1=[0.]

(0,3,0):-1=[0.]

(0,3,1):-1=[0.]

(0,3,2):-1=[0.]

(1,0,0):RECHARGE=[-2.775]

(1,0,1):DODGE=[-2.744]

(1,0,2):DODGE=[-2.442]

(1,1,0):RECHARGE=[-2.358]

(1,1,1):SHOOT=[2.361]

(1,1,2):SHOOT=[2.363]

(1,2,0):RECHARGE=[-2.357]

(1,2,1):SHOOT=[2.382]

(1,2,2):SHOOT=[2.618]

(1,3,0):RECHARGE=[-2.357]

(1,3,1):SHOOT=[2.382]

(1,3,2):SHOOT=[2.619]

(2,0,0):RECHARGE=[-2.778]

(2,0,1):DODGE=[-2.778]

(2,0,2):DODGE=[-2.778]

(2,1,0):RECHARGE=[-2.778]

(2,1,1):SHOOT=[-2.778]

(2,1,2):SHOOT=[-2.776]

(2,2,0):RECHARGE=[-2.776]

(2,2,1):SHOOT=[-2.757]

(2,2,2):SHOOT=[-2.521]

(2,3,0):RECHARGE=[-2.776]

(2,3,1):SHOOT=[-2.757]

(2,3,2):SHOOT=[-2.519]

(3,0,0):RECHARGE=[-2.778]

(3,0,1):DODGE=[-2.778]

(3,0,2):DODGE=[-2.778]

(3,1,0):RECHARGE=[-2.778]

(3,1,1):SHOOT=[-2.778]

(3,1,2):SHOOT=[-2.778]

(3,2,0):RECHARGE=[-2.778]

(3,2,1):SHOOT=[-2.778]

(3,2,2):SHOOT=[-2.778]

(3,3,0):RECHARGE=[-2.778]

(3,3,1):SHOOT=[-2.778]

(3,3,2):SHOOT=[-2.777]

(4,0,0):RECHARGE=[-2.778]

(4,0,1):RECHARGE=[-2.778]

(4,0,2):RECHARGE=[-2.778]

(4,1,0):RECHARGE=[-2.778]

(4,1,1):SHOOT=[-2.778]

(4,1,2):SHOOT=[-2.778]

(4,2,0):RECHARGE=[-2.778]

(4,2,1):SHOOT=[-2.778]

(4,2,2):SHOOT=[-2.778]

(4,3,0):RECHARGE=[-2.778]

(4,3,1):SHOOT=[-2.778]

(4,3,2):SHOOT=[-2.778]