Machine Data and Learning
SPRING SEMESTER 2020

Team 36
Sartak Periwal (2018101024) and Bhavesh Shukla(2018101036)

ASSIGNMENT 2, March 7

TASK-1

Libraries used :

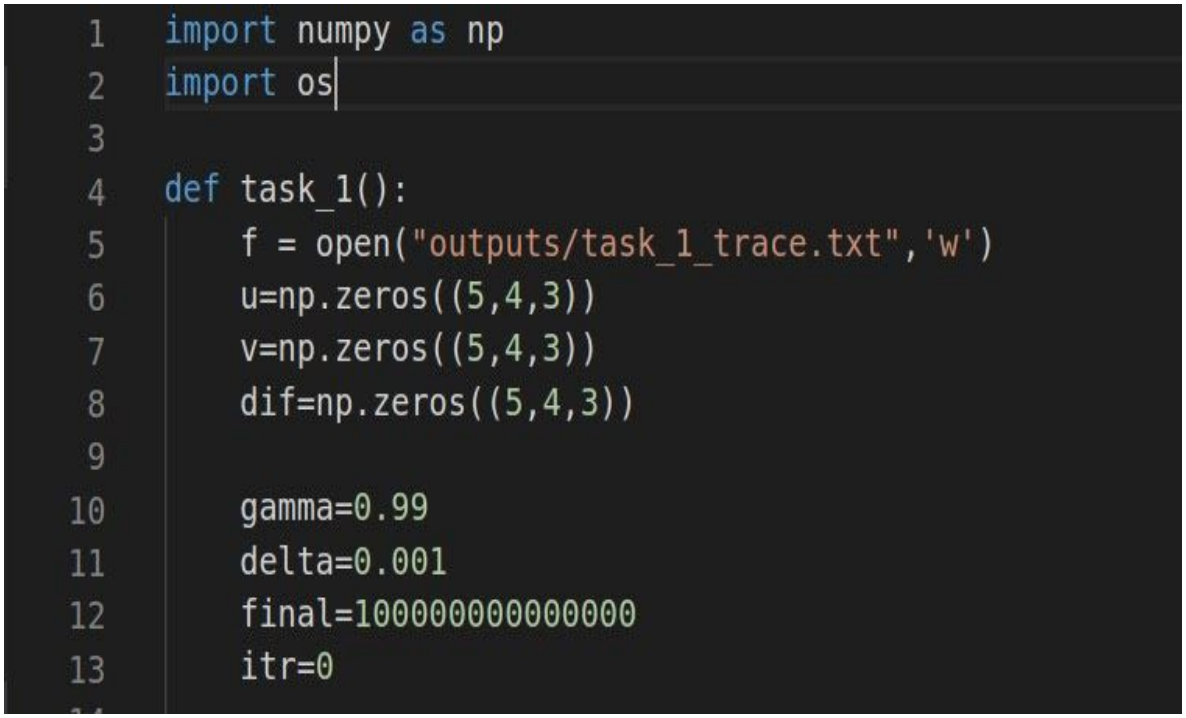
- Numpy
- os

Description of algorithm :

- **Value iteration algorithm**

1: **Procedure** Value_Iteration(S,A,P,R,D)
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: D 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** $\forall s\, |V_k[s] - V_{k-1}[s]| < D$
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:



```
1   import numpy as np
2   import os
3
4   def task_1():
5       f = open("outputs/task_1_trace.txt",'w')
6       u=np.zeros((5,4,3))
7       v=np.zeros((5,4,3))
8       dif=np.zeros((5,4,3))
9
10      gamma=0.99
11      delta=0.001
12      final=100000000000000
13      itr=0
```

F is the file where all the outputs should be kept.it is different for every task.

Gamma,delta are given values.

Final initialized very large to suffice exit condition,itr stores number of iterations.

Step_cost = -20 team(36)

```
while(final > delta):
    # if itr > 5:
    #     break
    print("iteration="+str(itr), file=f)
    for i in range(0,5):
        for j in range(0,4):
            for k in range(0,3):
                ## i=> enemy health , j=> num arrows , k=> stamina
                step_cost=20

                shoot=-100000000
                if j>0 and k>0:
                    if (i-1==0):
                        reward=0.5*(10-step_cost) + 0.5*(-step_cost)
                    else:
                        reward=-step_cost
                    shoot=reward + gamma*(0.5*u[i-1,j-1,k-1] + 0.5*u[i,j-1,k-1])

                recharge=-100000000
                if k==2:
                    reward=-step_cost
                    recharge= reward + gamma*(1*u[i,j,k])
                elif k<2:
                    reward=-step_cost
                    recharge=reward + gamma*(0.8*u[i,j,k+1]+0.2*u[i,j,k])

                dodge=-100000000
                reward=-step_cost
                if k==1:
                    if j==3:
                        dodge=reward + gamma*(u[i,3,0])
                    elif j<3:
                        dodge=reward + gamma*(0.2*u[i,j,0] + 0.8*u[i,j+1,0])
                elif k==2:
                    if j==3:
                        dodge=reward + gamma*(0.2*u[i,3,0] + 0.8*u[i,3,1])
                    elif j<3:
                        dodge=reward + gamma*(0.2*0.2*u[i,j,0] + 0.2*0.8*u[i,j+1,0] + 0.8*0.2*u[i,j,1] + 0.8*0.8*u[i,j+1,1])

                temp=max(shoot,recharge,dodge)
```

$U_t[i,j,k]=u[i,j,k]$

$U_{t+1}[i,j,k]=v[i,j,k]$

I,j,k have mappings as mentioned in the question.

For each action all cases are considered in the conditions

```
                temp=max(shoot,recharge,dodge)

                if i>0:
                    if temp==shoot:
                        # print('(',i,j,k,')',v[i,j,k],'policy:shoot')
                        print("({},{},{}):SHOOT=[{}]".format(i, j, k, round(v[i, j, k], 3)), file=f)
                    elif temp==recharge:
                        print("({},{},{}):RECHARGE=[{}]".format(i, j, k, round(v[i, j, k], 3)), file=f)
                        # print('(',i,j,k,')',v[i,j,k],'policy:recharge')
                    else:
                        print("({},{},{}):DODGE=[{}]".format(i, j, k, round(v[i, j, k], 3)), file=f)
                        # print('(',i,j,k,')',v[i,j,k],'policy:dodge')
                    v[i,j,k]=temp

                elif i==0:
                    print("({},{},{}):-1=[{}]".format(i, j, k, round(v[i, j, k], 3)), file=f)
                    # print('(',i,j,k,')',u[i,j,k],'policy:-1')
                    v[i,j,k]=0
                dif[i,j,k]=np.abs(v[i,j,k]-u[i,j,k])

    u=np.copy(v)
    final=np.max(dif)
    # print(final)
    itr+=1
    print("\n",file =f)
f.close()
```

Final observation:

It's better to have more stamina and arrows, and that the Dragon has less health. 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=125

(0,0,0):-1=[0.0]

(0,0,1):-1=[0.0]

(0,0,2):-1=[0.0]

(0,1,0):-1=[0.0]

(0,1,1):-1=[0.0]

(0,1,2):-1=[0.0]

(0,2,0):-1=[0.0]

(0,2,1):-1=[0.0]

(0,2,2):-1=[0.0]

(0,3,0):-1=[0.0]

(0,3,1):-1=[0.0]

(0,3,2):-1=[0.0]

(1,0,0):RECHARGE=[-179.508]

(1,0,1):DODGE=[-156.522]

(1,0,2):DODGE=[-137.901]

(1,1,0):RECHARGE=[-127.499]

(1,1,1):SHOOT=[-103.856]

(1,1,2):SHOOT=[-92.478]

(1,2,0):RECHARGE=[-102.076]

(1,2,1):SHOOT=[-78.112]

(1,2,2):SHOOT=[-66.409]

(1,3,0):RECHARGE=[-89.648]

(1,3,1):SHOOT=[-65.527]

(1,3,2):SHOOT=[-53.665]

(2,0,0):RECHARGE=[-351.231]

(2,0,1):DODGE=[-330.413]

(2,0,2):DODGE=[-313.549]

(2,1,0):RECHARGE=[-304.128]

(2,1,1):SHOOT=[-282.716]

(2,1,2):SHOOT=[-261.033]

(2,2,0):RECHARGE=[-255.68]

(2,2,1):RECHARGE=[-233.656]

(2,2,2):SHOOT=[-211.353]

(2,3,0):RECHARGE=[-219.569]

(2,3,1):SHOOT=[-197.089]

(2,3,2):SHOOT=[-174.325]

(3,0,0):RECHARGE=[-506.755]

(3,0,1):DODGE=[-487.901]

(3,0,2):DODGE=[-472.628]

(3,1,0):RECHARGE=[-464.096]

(3,1,1):RECHARGE=[-444.703]

(3,1,2):SHOOT=[-425.066]

(3,2,0):RECHARGE=[-420.218]

(3,2,1):SHOOT=[-400.271]

(3,2,2):SHOOT=[-380.072]

(3,3,0):RECHARGE=[-375.086]

(3,3,1):SHOOT=[-354.569]

(3,3,2):SHOOT=[-333.794]

(4,0,0):RECHARGE=[-647.603]

(4,0,1):DODGE=[-630.528]

(4,0,2):DODGE=[-616.696]

(4,1,0):RECHARGE=[-608.969]

(4,1,1):DODGE=[-591.407]

(4,1,2):SHOOT=[-573.622]

(4,2,0):RECHARGE=[-569.232]

(4,2,1):DODGE=[-551.167]

(4,2,2):SHOOT=[-532.874]

(4,3,0):RECHARGE=[-528.358]

(4,3,1):SHOOT=[-509.777]

(4,3,2):SHOOT=[-490.961]

TASK-2

Final Observations

Subtask 1:

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.The convergence is faster because of shooting cost is lower than dodging.

Result of first part:

iteration=99

(0,0,0):-1=[0.0]

(0,0,1):-1=[0.0]

(0,0,2):-1=[0.0]

(0,1,0):-1=[0.0]

(0,1,1):-1=[0.0]

(0,1,2):-1=[0.0]

(0,2,0):-1=[0.0]

(0,2,1):-1=[0.0]

(0,2,2):-1=[0.0]

(0,3,0):-1=[0.0]

(0,3,1):-1=[0.0]

(0,3,2):-1=[0.0]

(1,0,0):RECHARGE=[-10.317]

(1,0,1):DODGE=[-7.291]

(1,0,2):DODGE=[-4.839]

(1,1,0):RECHARGE=[-3.47]

(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=[-22.49]

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

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

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

(2,2,1):SHOOT=[-13.1]

(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=[-39.715]

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

(3,1,2):SHOOT=[-34.4]

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

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

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

(3,3,0):RECHARGE=[-27.72]

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

(3,3,2):SHOOT=[-22.1]

(4,0,0):RECHARGE=[-60.716]

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

(4,0,2):DODGE=[-56.392]

(4,1,0):RECHARGE=[-55.311]

(4,1,1):SHOOT=[-52.854]

(4,1,2):SHOOT=[-50.394]

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

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

(4,2,2):SHOOT=[-44.757]

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

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

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

Subtask 2:

This model converges much more quickly because of very low discount factor thus the value of the reward obtained decreases very quickly such that there is no more great incentive to achieve the final reward.

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.

Result of Second part:

iteration=4

(0,0,0):-1=[0.0]

(0,0,1):-1=[0.0]

(0,0,2):-1=[0.0]

(0,1,0):-1=[0.0]

(0,1,1):-1=[0.0]

(0,1,2):-1=[0.0]

(0,2,0):-1=[0.0]

(0,2,1):-1=[0.0]

(0,2,2):-1=[0.0]

(0,3,0):-1=[0.0]

(0,3,1):-1=[0.0]

(0,3,2):-1=[0.0]

(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=[-2.776]

(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]

Subtask 3:

The model converges slower than part 2 because neither the policy nor the utility changes much in subsequent iterations.

The policy will not change as we sharpen 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 third part:

iteration=11

(0,0,0):-1=[0.0]

(0,0,1):-1=[0.0]

(0,0,2):-1=[0.0]

(0,1,0):-1=[0.0]

(0,1,1):-1=[0.0]

(0,1,2):-1=[0.0]

(0,2,0):-1=[0.0]

(0,2,1):-1=[0.0]

(0,2,2):-1=[0.0]

(0,3,0):-1=[0.0]

(0,3,1):-1=[0.0]

(0,3,2):-1=[0.0]

(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]