Assignment 2 - Part 2
Machine, Data and Learning

Kushagra Agarwal - 2018113012
Shreeya Pahune - 2018113011

Code Implementation using Snippets

Value iteration algorithm - the algorithm iterates through the list of utilities, and keeps updating till the utilities converge sufficiently.

Initialisations:

```
import numpy as np

#Creating a directory called outputs in the current folder
import os
os.mkdir("./ouputs")

stamina = [0, 50, 100]
health = [0, 25, 50, 75, 100]
ACTIONS = ["SHOOT", "DODGE", "RECHARGE"]
arrow = [0, 1, 2, 3]
team = 2  # team number
step_cost = -20
bellman = 1e-3
states = [(h, a, s) for h in range( 5 ) for a in range( 4 ) for s in range( 3 )]
discount = 0.99
state_v = np.zeros((5, 4, 3))
```

Made a new folder to store the output files of the code. Also initialised all the values of bellman, stamina, health, arrow.
Also the team number is 2 so the step cost is -20

Print Functions:

```
################################### PRINT FUNCTION FOR TASK1 IS HERE ###############################################################################

def Print1(itr, action_index):

    f=open("./ouputs/task_1_trace.txt","a")
    f.write(('iteration=' + str(itr)+'\n'))
    for h in range(5):
        for a in range(4):
            for s in range(3):
                string = ACTIONS[action_index[h][a][s]]
                if(h==0):
                    string=-1 #because the game is over
                finalstring= "("+ str(h) + "," + str(a) + "," + str(s) + "):" + str(string) + "=[" + str(round(state_v[h,a,s],3)) + "]\n"
                f.write(finalstring)
                finalstring=""
    f.write( "\n\n" )

def Print2(itr, action_index):

    f=open("./ouputs/task_2_part_1_trace.txt","a")
    f.write(('iteration=' + str(itr)+'\n'))
    for h in range(5):
        for a in range(4):
            for s in range(3):
                string = ACTIONS[action_index[h][a][s]]
                if(h==0):
                    string=-1 #because the game is over
                finalstring= "("+ str(h) + "," + str(a) + "," + str(s) + "):" + str(string) + "=[" + str(round(state_v[h,a,s],3)) + "]\n"
                f.write(finalstring)
                finalstring=""
    f.write( "\n\n" )

def Print3(itr, action_index):

    f=open("./ouputs/task_2_part_2_trace.txt","a")
    f.write(('iteration=' + str(itr)+'\n'))
    for h in range(5):
```

The print functions have been written for different tasks. We open the respective file and then make a string of each line and write it to the file opened.

Utility Function:

Utility Function for action=”SHOOT”

```
def Utility(s, a, h, action):

    #Default value of utility is very high so that if none of the allowd cases occur this can take place and not get selected during the max of v
    utility=-100000
    #for action=="SHOOT"
    if action==0:

        #To shoot, conditions are as follows:
        #Number of arrows >0

        if(a > 0):

            #To shoot, also the stamina must be either 50 or 100, so

            if(s>0):

                #The opponents heatlth can either be 25 or >25 so we have 2 cases( The case where h==0 is handled in the task functions itself)

                if(h==1):

                    #The opponent has health=25 so if this arrow hits then the game is over with a reward of 10 extra
                    prob=0.5
                    utility= (1-prob)* state_v[h,a-1,s-1] + (prob)*10

                else:

                    #The opponent cannot die hence getting reward is out of question

                    prob=0.5
                    utility=(1-prob)*state_v[h,a-1,s-1] + prob*state_v[h-1,a-1,s-1]
```

The comments are self explanatory here.

Utility function for action=”DODGE” or “RECHARGE”

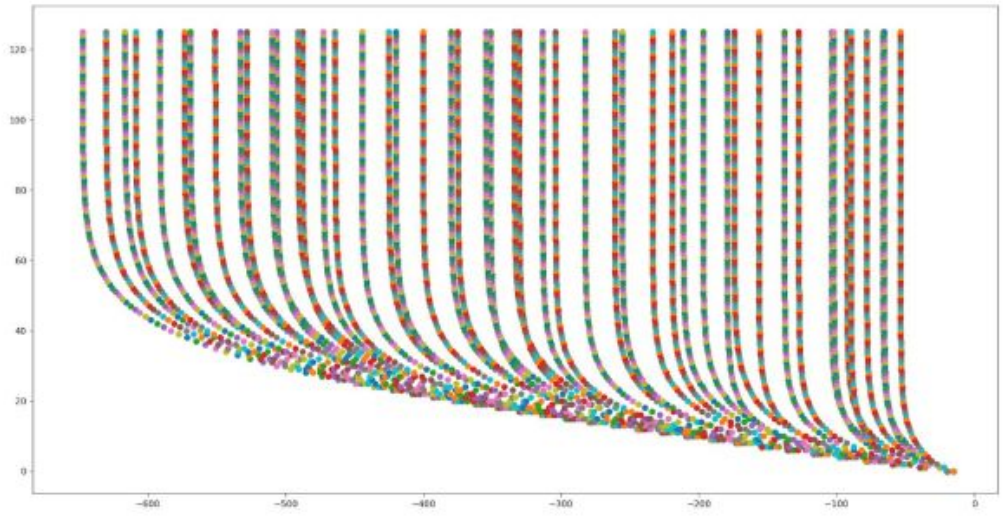
```
if action==1:

    #To dodge the conditions are as follows:
    #if Stamina==50, then it decreases by 50
    #if Stamina==100, then it decrease by 50 or 100 with prob=0.5

    if(s>0):

        if(s==1 and a==3):
            prob=1
            utility= prob*state_v[h,a,0]
        elif(s==1 and a!=3):
            prob=0.8
            utility= prob*state_v[h,a+1,0]+(1-prob)*state_v[h,a,0]
        elif(s!=1 and a==3):
            prob1=0.8
            prob2=1
            utility=prob1*prob2*state_v[h,a,s-1]+(1-prob1)*prob2*state_v[h,a,s-2]
        elif(s!=1 and a!=3):
            prob1=0.8
            prob2=0.8
            utility=prob1*prob2*state_v[h,a,s-1]+(1-prob1)*prob2*state_v[h,a,s-2]+prob1*(1-prob2)*state_v[h,a+1,s-1]+(1-prob1)*(1-prob2)*state_v[h,a+1,s-2]

if action==2:

    #To recharge the stamina increased with prob=0.8, and if the state is 100 already, then no increase in the stamina
    if(s!=2):
        prob=0.8
        utility= prob*state_v[h,a,s+1] + (1-prob)*state_v[h,a,s]
    else:
        utility=state_v[h,a,s]
```

The comments are self explanatory in this piece of code too.

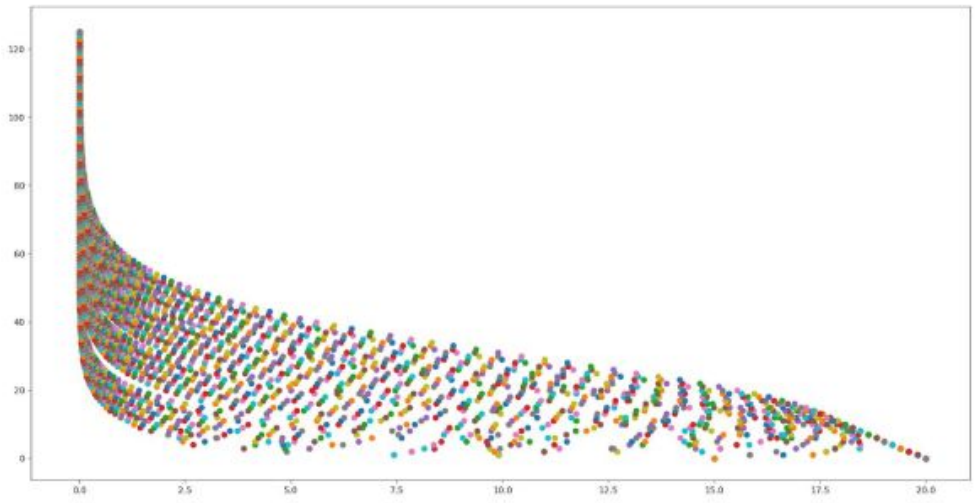
Task1

Results

Utility values of the states vs Number of iterations



Maximum difference between U(t) and U(t+1) vs Iteration number for each tasks and subtasks

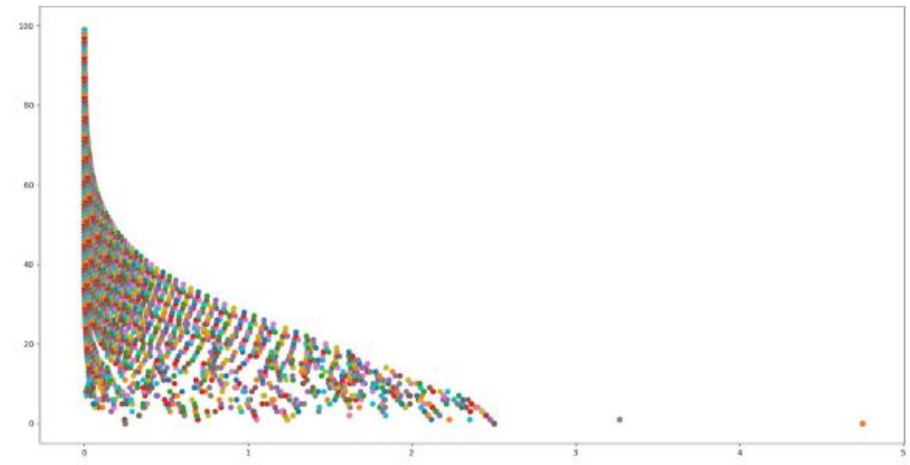


Inferences

The convergence is seen as the we move from the initial confusion to converging values. The algorithm prefers to have more stamina than more arrows i.e. it plays safe. Recharge is chosen whenever stamina becomes 0 and dodge whenever stamina is less or no arrows are left. Shoots only if there are enough arrows along with low health for the MD.

Task2-Part1

Maximum difference between U(t) and U(t+1) vs Iteration number for each tasks and subtasks



What changes do you observe in the policy and why?

It is clearly visible that the values rapidly converge, much faster in case of task 2 part 1 than in task 1, this is due to the reduction in step cost for shooting, as well as the reduction in step costs for the rest of the actions. Hence more preference for shooting which implies MD's health becomes 0 faster.

Task2-Part2

Explain Lero's weird behaviour.

The future rewards are so trivial due to the extremely low value of gamma which causes the user to take random actions as all actions become almost equally likely since rewards are trivial anyways.

Task2-Part3

What changes do you observe and why?

The gamma is equally low in this case however the ***change*** is that the iterations increase due to the decrease in delta which should imply an increase in precision. However, the increase in required precision does nothing to change the high inaccuracy caused due to the decrease in future weights.