Reinforcement Learning Report

Vasu Soni

October 15, 2023

Contents

1 Complete Python Code for Task 1

Listing 1: Complete Python Code for Task 1

```
#python code for task1
import numpy as np
import argparse
import pulp

def Value_Policy(T, R, gamma, S, Policy):
    Vpi = np.zeros(S)
    Vpi_1 = np.zeros(S)
    eps = 1e-12
    while True:
        Vpi_1 = Vpi.copy()
        for i in range(S):
            Vpi[i] = np.sum(T[i, int(Policy[i])] * (R[i, int(Policy[i])] + gamma *
        if abs(np.max(Vpi - Vpi_1)) < eps and abs(np.max(Vpi_1-Vpi)) < eps:
            break
    Qpi = np.sum(np.multiply(T, (R + gamma * Vpi)), axis=2)
    return Vpi, Qpi


def PolicyIteration(T, R, S, gamma):
    Policy = np.zeros(S)
    Pre_Policy = np.zeros(S)
    V, Q = Value_Policy(T, R, gamma, S, Policy)
    while True:
        Pre_Policy = Policy.copy()
        IA = []
        for i in range(S):
            indices = np.where(Q[i] > V[i])
            indices = list(indices[0])
            IA.append(indices)
        for i in range(S):
            if len(IA[i])>0:
                Policy[i] = np.argmax(Q[i])
        V, Q = Value_Policy(T, R, gamma, S, Policy)
        if np.array_equal(Pre_Policy,Policy):
          break

    return Policy

def LinearProgramming(T,R,gamma,S,A):
    problem = pulp.LpProblem("LP", pulp.LpMinimize)
    num_vars = S
    V = [pulp.LpVariable(f"V{i}") for i in range(num_vars)]
    problem += pulp.lpSum(V[s] for s in range(S))

    for s in range(S):
        for a in range(A):
            problem += V[s] >= pulp.lpSum(T[s][a][s_prime] * (R[s][a][s_prime] + g

    problem.solve(pulp.PULP_CBC_CMD(msg=False))
    optimal_Vs = [V[s].varValue for s in range(S)]
    Vs = np.array(optimal_Vs)
    As = np.argmax(np.sum(T*(R+gamma*Vs),axis=2),axis = 1)
    return Vs,As
```

```
def ValueIteration(T, R, gamma, S, A,endstate):
    Vt = np.zeros(S)
    At = np.zeros(S)
    Vt_1 = np.zeros(S)
    eps = 1e-10
    sum = 0
    while(True):
        Vt_1 = Vt.copy()
        for i in range(S):
            if i not in endstate:
                Vt[i] = np.max(np.sum(T[i]*(R[i]+gamma*Vt_1),axis=1))
                At[i] = np.argmax(np.sum(T[i]*(R[i]+gamma*Vt_1),axis=1))
            else:
                Vt[i] = 0
                At[i] = -1
        if(abs(np.max(Vt-Vt_1))<eps and abs(np.max(Vt_1-Vt))<eps):
            break
    return Vt,At

def ReadFile(mdppath):
    States = 0
    Actions = 0
    T = None
    R = None
    gamma = 0
    mtype = "not-specified"
    endstate = -1

    with open(mdppath, 'r') as file:
        lines = file.readlines()
        for line in lines:
            parts = line.strip().split(' ')
            parts = [i for i in parts if i != '']
            if parts[0] == 'numStates':
                States = int(parts[1])
            elif parts[0] == 'numActions':
                Actions = int(parts[1])
                T = np.zeros((States, Actions, States))
                R = np.zeros(shape = (States, Actions, States), dtype = np.float32
            elif parts[0] == 'transition':
                s1, ac, s2 = map(int, parts[1:4])
                p = float(parts[5])
                r = float(parts[4])
                T[s1,ac,s2] = p
                R[s1,ac,s2] = r
            elif parts[0] == 'mdptype':
                mtype = parts[1]
            elif parts[0] == 'discount':
                gamma = float(parts[1])
            elif parts[0] == 'end':
                endstate = parts[1:]

    return States, Actions, T, R, gamma, mtype, endstate

def ReadPolicy(policypath,States):
    P = np.zeros(States)
    N = []
```

```
    with open(policypath, 'r') as file:
        lines = file.readlines()
        for line in lines:
            parts = line.strip().split(' ')
            parts = [i for i in parts if i != '']
            N.append(int(parts[0]))
    P = np.array(N)
    return P
if __name__ == "__main__":

    parse = argparse.ArgumentParser()
    parse.add_argument("--mdp", type=str, help="MDP file path to be entered")
    parse.add_argument("--algorithm", type=str, help="Algorithm to be executed", c
    parse.add_argument("--policy", type=str, help="Policy for Value Function")
    argument = parse.parse_args()
    mdppath = argument.mdp
    algorithm = argument.algorithm
    policypath = argument.policy
    States, Actions, T, R, gamma, mtype, endstate = ReadFile(mdppath)
    if policypath != None:
        Policy = ReadPolicy(policypath,States)
        Vpi,Qpi = Value_Policy(T,R,gamma,States,Policy)
        for i in range(States):
            print(Vpi[i], " ", Policy[i])
    elif algorithm == 'vi':
        Vt, At = ValueIteration(T,R,gamma,States,Actions,endstate)
        for i in range(States):
            print(Vt[i], " ", int(At[i]))

    elif algorithm == 'hpi':
        Policy = PolicyIteration(T,R,States,gamma)
        Vt,Qt = Value_Policy(T,R,gamma,States,Policy)
        for i in range(States):
            print(Vt[i], " ", int(Policy[i]))

    elif algorithm == 'lp':
        Vt,At = LinearProgramming(T,R,gamma,States,Actions)
        for i in range(States):
            print(Vt[i]," ",At[i])
```

2 Task 1

The code for planner.py is shown above and piecewise explanation of the code for each algorithm is done below. The MDP file given is read and 2 3-D Matrices are formed T and R respectively and the rest information is stored inside variables with there respective names.

2.1 Value Iteration

For this I made a function and inside it I have used 2 1-D arrays to store the current and previous values of the states and another 1-D array for storing Action (Optimal). Initially filled with 0(arbitrary value). Variable eps is used to store precision. Inside the loop firstly the value of current states is copied to arrays storing previous value states. After that for every state the value is calculated using the bellman optimality operator. Since np.argmax() is used for finding optimal action the tie break is taken care by taking the first value which is the maximum.

2.1.1 Algorithm

Listing 2: Complete Python Code for Task 1 (Value Iteration)

```
def ValueIteration(T, R, gamma, S, A,endstate):
    Vt = np.zeros(S)
    At = np.zeros(S)
    Vt_1 = np.zeros(S)
    eps = 1e-10
    sum = 0
    while(True):
        Vt_1 = Vt.copy()
        for i in range(S):
            if i not in endstate:
                Vt[i] = np.max(np.sum(T[i]*(R[i]+gamma*Vt_1),axis=1))
                At[i] = np.argmax(np.sum(T[i]*(R[i]+gamma*Vt_1),axis
                    =1))
            else:
                Vt[i] = 0
                At[i] = -1
        if(abs(np.max(Vt-Vt_1))<eps and abs(np.max(Vt_1-Vt))<eps):
            break
    return Vt,At
```

2.2 Policy Iteration

The code for policy iteration is given below. Here I have used 2 function one for policy iteration and another for finding the value of the policy which is value-policy function. Initially a policy which takes action 0 for every state is assigned and then policy iteration is done where value and action-value for each state is calculated and then new policy is made by taking those actions for which the action-value is maximized for that state.

2.2.1 Algorithm

Listing 3: Complete Python Code for Task 1 (Policy Iteration)

```
    def Value_Policy(T, R, gamma, S, Policy):
    Vpi = np.zeros(S)
    Vpi_1 = np.zeros(S)
    eps = 1e-12
    while True:
        Vpi_1 = Vpi.copy()
        for i in range(S):
            Vpi[i] = np.sum(T[i, int(Policy[i])] * (R[i, int(Policy[i
                ])] + gamma * Vpi_1))
        if abs(np.max(Vpi - Vpi_1)) < eps and abs(np.max(Vpi_1-Vpi))
           < eps:
            break
    Qpi = np.sum(np.multiply(T, (R + gamma * Vpi)), axis=2)
    return Vpi, Qpi


def PolicyIteration(T, R, S, gamma):
    Policy = np.zeros(S)
    Pre_Policy = np.zeros(S)
    V, Q = Value_Policy(T, R, gamma, S, Policy)
    while True:
        Pre_Policy = Policy.copy()
        IA = []
        for i in range(S):
```

```
            indices = np.where(Q[i] > V[i])
            indices = list(indices[0])
            IA.append(indices)
        for i in range(S):
            if len(IA[i])>0:
                Policy[i] = np.argmax(Q[i])
        V, Q = Value_Policy(T, R, gamma, S, Policy)
        if np.array_equal(Pre_Policy,Policy):
          break

    return Policy
```

2.3 Linear Programming

Here as stated I have used pulp library for the calculation of optimal value for each state and for calculating the optimal action I have used the action-value function.

2.3.1 Linear Programming Formulation

Listing 4: Complete Python Code for Task 1 (Policy Iteration)

```
def LinearProgramming(T,R,gamma,S,A):
problem = pulp.LpProblem("LP", pulp.LpMinimize)
num_vars = S
V = [pulp.LpVariable(f"V{i}") for i in range(num_vars)]
problem += pulp.lpSum(V[s] for s in range(S))

for s in range(S):
    for a in range(A):
        problem += V[s] >= pulp.lpSum(T[s][a][s_prime] * (R[s][a
            ][s_prime] + gamma * V[s_prime])for s_prime in range(S
            ))

problem.solve(pulp.PULP_CBC_CMD(msg=False))
optimal_Vs = [V[s].varValue for s in range(S)]
Vs = np.array(optimal_Vs)
As = np.argmax(np.sum(T*(R+gamma*Vs),axis=2),axis = 1)
return Vs,As
```

3 Complete Python Code for Task 2

The code for encoder is shown. Here I have printed all the transitions with non-zero probabilities and for that the code is given below which shown The comments given describe the code completely.

In brief what I have done is to find possible positions of opponent with non-zero probabilities since those are the only valid states for which the probability will be non-zeros than I have independently coded each action starting from 0 till 9 where in for actions 0 - 7 I have first checked the conditions for the which the player goes out of bound or not than I have checked status of ball (i.e. which player has the ball) than after that I have checked for the case tackling if the player with the ball moves otherwise simply put probabilities. For action 8 and 9 similar to above found the next positions of the opponent and than for 8 checked if next position of R comes in line of both the players or not than put the probablities accordinly For action 9 I checked the position of R in the next positions where it was standing infront of the goal and put the probablities if it was not standing in front of the goals than I didn't halved the probabilities

Listing 5: Complete Python Code for Task 2

```
import numpy as np
```

```python
import argparse


def read_probabilities_file(file_path):
    #This function is for reading the opponents file and getting all
        this inside a numpy array.
    # Initialize an empty list to store the data.
    data = []

    # Open the file for reading
    with open(file_path, 'r') as file:
        lines = file.readlines()

        # Iterate over lines, starting from the second line
        for line in lines[1:]:
            # Split the line into fields using space as the delimiter
            fields = line.strip().split(' ')

            #this is the reading part for the probabilities
            state = fields[0]
            p_l = float(fields[1])
            p_r = float(fields[2])
            p_u = float(fields[3])
            p_d = float(fields[4])
            data.append((state, p_l, p_r, p_u, p_d))

    # Convert the list of tuples to a NumPy array
    data_array = np.array(data, dtype=[('State', 'U7'), ('P(L)',
        float), ('P(R)', float), ('P(U)', float), ('P(D)', float)])

    return data_array

def get_R_probabilities(data_array, state_name):
    #This function is for finding the probabilities of R for a given
        state
    filtered_data = data_array[data_array['State'] == state_name]

    # Check if the state was found
    if len(filtered_data) > 0:
        answer = np.array([filtered_data[0][1],filtered_data[0][2],
            filtered_data[0][3],filtered_data[0][4]])
        return answer  # Return the first matching row as a NumPy
            array
    else:
        print("found None")
        return None

def findxy(num):
    #This function is used to find the coordinates of the position of
        player
    #####Note!!!!!!!!! I have used coordinates starting from top left
        corner as (1,1) and bottom right corner as (4,4)
    ##Positive x-axis is towards the right and positive y-axis is
        downwards
    x = num%4
    y = 0
    if(x == 0):
      x = 4
```

```
    if num in [1,2,3,4]:
      y = 1
    elif num in [5,6,7,8]:
      y = 2
    elif num in [9,10,11,12]:
      y = 3
    else:
      y = 4
    return x,y


def give_Positions(State):
    #This function is used to the positions of all the player and the
         state of ball for a given state.
    Positions = np.array([int(State[0:2]),int(State[2:4]),int(State
        [4:6]),int(State[6])])
    return Positions

def MapState():
    #This Dictionary is a mapping from state to integers from 0-8193
        in which 0-8191 are the states and 8192 is the lose state
    #8193 is the win state
    mydict = {}
    s = 0
    P = ['01','02','03','04','05','06','07','08','09','10','11','12',
        '13','14','15','16']
    B = ['1','2']
    for i in P:
      for j in P:
        for k in P:
          for e in B:
            mydict[i+j+k+e] = s
            s += 1
    mydict['lose'] = 8192
    mydict['win'] = 8193
    return mydict

def check_line(x1, y1, x2, y2,x3,y3):
    #This code for checking that whether R is in the line of B1 and
        B2 while passing
    #For this I have checked along the line x+y = c and x-y = c and
        horizontal and vertical line
    #this returns a bool stating whether R is present between them
        while passing or not
    if x1+y1 == x2+y2 == x3+y3 and ((x3>=x1 and x3<=x2) or (x3<=x1
       and x3>=x2)):
        return True
    elif x1-y1 == x2-y2 == x3-y3 and ((x3>=x1 and x3<=x2) or (x3<=x1
       and x3>=x2)):
        return True
    elif x1 == x2 == x3 and ((y3 >= y1 and y3 <= y2) or (y3 <= y1 and
        y3 >= y2)):
        return True
    elif y1 == y2 == y3 and ((x3>=x1 and x3<=x2) or (x3<=x1 and x3>=
       x2)):
        return True
    else:
        return False
```

```
def give_T_and_R(op_policy, p, q):
    #This is the main function for building the MDP
    # T = np.zeros((8194,10,8194))
    Statemap = MapState()
    #Here I am considering the case of game end when player goes out of bounds so in this case these are block positions after
    #taking the action and if a player B1 was in 5 and took action 0 then it will move to 4 in code but in grid it will
    #fall out of it hence these values taken taking care of that.
    LB = np.array([0,4,8,12])          #This is used for checking whether after moving in left a player goes out of bounds or not
    RB = np.array([5,9,13,17])         #Same as above for right
    UB = np.array([-3,-2,-1,0])        #For Up
    DB = np.array([17,18,19,20])       #For Down
    #From here I am starting to calculate Transition probabilities
    for states in op_policy['State']:
        positions = give_Positions(states) #find positions given the state
        R_p = get_R_probabilities(op_policy, states) #get probabilities of R given the state
        R_next = [positions[2] - 1, positions[2] + 1, positions[2] - 4, positions[2] + 4] #Position of in possible next state
        non_zero_indices = np.nonzero(R_p)[0] #removing all the zero probabilities states of R
        R_next = [R_next[i] for i in non_zero_indices] #Finding non-zero next positions of R
        R_prob = [R_p[i] for i in non_zero_indices] #Finding their respective probabilities
        R_next = [str(x).zfill(2) for x in R_next if 0 < x < 17] #Check for the out-bounds case for R
        prob1 = 0
        losing = 0
#For Action 0 this is the calculation of probabilities
        B1_next = str(positions[0]-1).zfill(2) #finding next position of B1 for action 0
        if not int(B1_next) in LB: #checking is B1 goes out of bounds
            if positions[3] == 2: #Since this is movement case checking for ball status if 2 then probabilities are set accordingly
                for v,i in enumerate(R_next): #For all possible next state of R
                    nextS = B1_next + str(positions[1]).zfill(2) + i + '2'
                    prob1 = (1-p)*R_prob[v]
                    losing += p*(R_prob[v])
                    if prob1!=0:
                        print("transition", Statemap[states],0, Statemap[nextS],0,prob1)
            else:
                for v,i in enumerate(R_next):
                    nextS = B1_next + str(positions[1]).zfill(2) + i + '1'
                    #Below condition is for tackling case here I have just checked the final position of R and B1_next
```

```
            #Whether they are same of interchanged these 2
                conditions are considered as tackling
            if B1_next == i or (B1_next == str(positions[2]).
                zfill(2) and i == str(positions[0]).zfill(2)):
                prob1 = (1-2*p)*R_prob[v]/2
                losing += (1-(1-2*p)/2)*R_prob[v]
            else:#condition if no tackling is present
                prob1 = (1-2*p)*R_prob[v]
                losing += 2*p*R_prob[v]
            if prob1!=0:
                print("transition", Statemap[states],0,
                    Statemap[nextS],0,prob1)
else:
    losing = 1

print("transition", Statemap[states],0,Statemap['lose'],0,
    losing)
prob1 = 0
losing = 0
#For Action 1
B1_next = positions[0]+1 #same as above finding the next
    position of player
B2_next = states[2:4] #next position of second player which
    will be same as before for Action 1
if not B1_next in RB:
    if positions[3] == 2: #check of ball status
        for v,i in enumerate(R_next): #Probabilities if
            player doesn't has ball
            nextS = str(B1_next).zfill(2) + B2_next + i + '2'
            prob1 = (1-p)*R_prob[v]
            losing += p*(R_prob[v])
            if prob1!=0:
                print("transition", Statemap[states],1,
                    Statemap[nextS],0,prob1)
    else:#Probability if the player has the ball
        for v,i in enumerate(R_next):
            nextS = str(B1_next).zfill(2) + str(B2_next).
                zfill(2) + i + '1'
            #here tackling is checked
            if str(B1_next).zfill(2) == i or (str(B1_next).
                zfill(2) == str(positions[2]).zfill(2) and i
               == str(positions[0]).zfill(2)):
                prob1 = (1-2*p)*R_prob[v]/2
                losing += (1-(1-2*p)/2)*R_prob[v]
            else:#if no tackling than
                prob1 = (1-2*p)*R_prob[v]
                losing += 2*p*R_prob[v]
            if prob1!=0:
                print("transition", Statemap[states],1,
                    Statemap[nextS],0,prob1)
else:
    losing = 1

print("transition", Statemap[states],1,Statemap['lose'],0,
    losing)
#The same process is repeated is for action from 0-7 as the
    comments mention above for the respective player.
#for Action 2
```

```
prob1 = 0
losing = 0
B1_next = str(positions[0]-4)
if len(B1_next) == 1:
    B1_next = '0' + B1_next
B2_next = states[2:4]
ball_next = states[6]
if not int(B1_next) in UB:
    if positions[3] == 2:
        for v,i in enumerate(R_next):
            nextS = str(B1_next).zfill(2) + B2_next + i + '2'
            prob1 = (1-p)*R_prob[v]
            losing += p*R_prob[v]
            if prob1!=0:
                print("transition", Statemap[states],2,
                    Statemap[nextS],0,prob1)
    else:
        for v,i in enumerate(R_next):
            nextS = str(B1_next).zfill(2) + B2_next + i + '1'
            if B1_next == i or (B1_next == str(positions[2]).
               zfill(2) and i == str(positions[0]).zfill(2)):
                prob1 = (1-2*p)*R_prob[v]/2
                losing += (1-(1-2*p)/2)*R_prob[v]
            else:
                prob1 = (1-2*p)*R_prob[v]
                losing += 2*p*R_prob[v]
            if prob1!=0:
                print("transition", Statemap[states],2,
                    Statemap[nextS],0,prob1)
else:
    losing = 1

print("transition", Statemap[states],2,Statemap['lose'],0,
   losing)
#for Action 3
prob1 = 0
losing = 0
B1_next = str(positions[0]+4).zfill(2)
B2_next = states[2:4]
if not int(B1_next) in DB:
    if positions[3] == 2:
        for v,i in enumerate(R_next):
            nextS = str(B1_next).zfill(2) + B2_next + i + '2'
            prob1 = (1-p)*R_prob[v]
            losing += p*R_prob[v]
            if prob1!=0:
                print("transition", Statemap[states],3,
                    Statemap[nextS],0,prob1)
    else:
        for v,i in enumerate(R_next):
            nextS = B1_next + B2_next + i + '1'
            if B1_next == i or (B1_next == str(positions[2]).
               zfill(2) and i == str(positions[0]).zfill(2)):
                prob1 = (1-2*p)*R_prob[v]/2
                losing += (1-(1-2*p)/2)*R_prob[v]
            else:
                prob1 = (1-2*p)*R_prob[v]
                losing += 2*p*R_prob[v]
```

```
            if prob1!=0:
                print("transition", Statemap[states],3,
                    Statemap[nextS],0,prob1)
else:
    losing = 1

print("transition", Statemap[states],3,Statemap['lose'],0,
   losing)
#for Action 4
prob1 = 0
losing = 0
B2_next = str(positions[1]-1)
if len(B2_next) == 1:
    B2_next = '0' + B2_next
B1_next = states[0:2]
ball_next = states[6]
if not int(B2_next) in LB:
    if positions[3] == 1:
        for v,i in enumerate(R_next):
            nextS = str(B1_next).zfill(2) + B2_next + i + '1'
            prob1 = (1-p)*R_prob[v]
            losing += p*R_prob[v]
            if prob1!=0:
                print("transition", Statemap[states],4,
                    Statemap[nextS],0,prob1)
    else:
        for v,i in enumerate(R_next):
            nextS = str(B1_next).zfill(2) + B2_next + i + '2'
            if B2_next == i or (B2_next == str(positions[2]).
                zfill(2) and i == str(positions[1]).zfill(2)):
                prob1 = (1-2*p)*R_prob[v]/2
                losing += (1-(1-2*p)/2)*R_prob[v]
            else:
                prob1 = (1-2*p)*R_prob[v]
                losing += 2*p*R_prob[v]
            if prob1!=0:
                print("transition", Statemap[states],4,
                    Statemap[nextS],0,prob1)
else:
    losing = 1

print("transition", Statemap[states],4,Statemap['lose'],0,
   losing)
#for Action 5
losing = 0
prob1 = 0
B2_next = str(positions[1]+1)
if len(B2_next) == 1:
    B2_next = '0' + B2_next
B1_next = states[0:2]
ball_next = states[6]
if not int(B2_next) in RB:
    if positions[3] == 1:
        for v,i in enumerate(R_next):
            nextS = str(B1_next).zfill(2) + B2_next + i + '1'
            prob1 = (1-p)*R_prob[v]
            losing += p*R_prob[v]
            if prob1!=0:
```

```
                print("transition", Statemap[states],5,
                    Statemap[nextS],0,prob1)
    else:
        for v,i in enumerate(R_next):
            nextS = str(B1_next).zfill(2) + B2_next + i + '2'
            if B2_next == i or (B2_next == str(positions[2]).
                zfill(2) and i == str(positions[1]).zfill(2)):
                prob1 = (1-2*p)*R_prob[v]/2
                losing += (1-(1-2*p)/2)*R_prob[v]
            else:
                prob1 = (1-2*p)*R_prob[v]
                losing += 2*p*R_prob[v]
            if prob1!=0:
                print("transition", Statemap[states],5,
                    Statemap[nextS],0,prob1)
else:
    losing = 1

print("transition", Statemap[states],5,Statemap['lose'],0,
    losing)
#for Action 6
prob1 = 0
losing = 0
B2_next = str(positions[1]-4)
if len(B2_next) == 1:
    B2_next = '0' + B2_next
B1_next = states[0:2]
ball_next = states[6]
if not int(B2_next) in UB:
    if positions[3] == 1:
        for v,i in enumerate(R_next):
            nextS = str(B1_next).zfill(2) + B2_next + i + '1'
            prob1 = (1-p)*R_prob[v]
            losing += p*R_prob[v]
            if prob1!=0:
                print("transition", Statemap[states],6,
                    Statemap[nextS],0,prob1)
    else:
        for v,i in enumerate(R_next):
            nextS = str(B1_next).zfill(2) + B2_next + i + '2'
            if B2_next == i or (B2_next == str(positions[2]).
                zfill(2) and i == str(positions[1]).zfill(2)):
                prob1 = (1-2*p)*R_prob[v]/2
                losing += (1-(1-2*p)/2)*R_prob[v]
            else:
                prob1 = (1-2*p)*R_prob[v]
                losing += 2*p*R_prob[v]
            if prob1!=0:
                print("transition", Statemap[states],6,
                    Statemap[nextS],0,prob1)
else:
    losing = 1

print("transition", Statemap[states],6,Statemap['lose'],0,
    losing)
#for Action 7
prob1 = 0
losing = 0
```

```
B2_next = str(positions[1]+4).zfill(2)
B1_next = states[0:2]
if not int(B2_next) in DB:
    if positions[3] == 1:
        for v,i in enumerate(R_next):
            nextS = str(B1_next).zfill(2) + B2_next + i + '1'
            prob1 = (1-p)*R_prob[v]
            losing += p*R_prob[v]
            if prob1!=0:
                print("transition", Statemap[states],7,
                  Statemap[nextS],0,prob1)
    else:
        for v,i in enumerate(R_next):
            nextS = str(B1_next).zfill(2) + B2_next + i + '2'
            if B2_next == i or (B2_next == str(positions[2]).
               zfill(2) and i == str(positions[1]).zfill(2)):
                prob1 = (1-2*p)*R_prob[v]/2
                losing += (1-(1-2*p)/2)*R_prob[v]
            else:
                prob1 = (1-2*p)*R_prob[v]
                losing += 2*p*R_prob[v]
            if prob1!=0:
                print("transition", Statemap[states],7,
                  Statemap[nextS],0,prob1)
else:
    losing = 1

print("transition", Statemap[states],7,Statemap['lose'],0,
   losing)

#For action 8 since we require the coordinates hence they are
     calculted using the function shown above
B1x, B1y = findxy(positions[0])
B2x, B2y = findxy(positions[1])
prob1 = 0
losing = 0
#for Action 8
B1_next = str(positions[0]).zfill(2) #next positions of the
   players
B2_next = str(positions[1]).zfill(2)
for v,i in enumerate(R_next):
    B1x, By = findxy(positions[0])
    B2x, B2y = findxy(positions[1])
    Rx, Ry = findxy(int(i))
    if str(positions[3]) == '1':
        nextS = B1_next + B2_next + i + '2'
    if str(positions[3]) == '2':
        nextS = B1_next + B2_next + i + '1'
    if check_line(B1x,B1y,B2x,B2y,Rx,Ry): #Here if R's next
       position is between B1 and B2 then probabilities are
       halved
        prob1 = (q-0.1*max(abs(B1x-B2x),abs(B1y-B2y)))*R_prob
           [v]/2
        losing += (1 - (q-0.1*max(abs(B1x-B2x),abs(B1y-B2y)))
           /2)*R_prob[v]
    else:#If R's next position is not between B1 and B2 then
       probabilities as shown
        prob1 = (q-0.1*max(abs(B1x-B2x),abs(B1y-B2y)))*R_prob
```

```
                [v]
            losing += (1 - (q-0.1*max(abs(B1x-B2x),abs(B1y-B2y))))*R_prob[v]
        if prob1!=0:
            print("transition", Statemap[states],8,Statemap[nextS],0,prob1)

    print("transition", Statemap[states],8,Statemap['lose'],0,losing)
    #for Action 9
    B1_next = str(positions[0]).zfill(2) #next positions of the players
    B2_next = str(positions[1]).zfill(2)
    prob1 = 0
    losing = 0
    # if Statemap[states] == 63:
    for v,i in enumerate(R_next):
        Rx, Ry = findxy(int(i)) #next positions of the opponent
        if Rx == 4 and (Ry == 2 or Ry == 3): #condition if R is present infront of goal post
            if positions[3] == 1: #status of the ball while shooting if B1 or B2
                prob1 += (q - 0.2*(4-B1x))*R_prob[v]/2
                losing += (1 - (q - 0.2*(4-B1x))/2)*R_prob[v]

            else: #shooting done by B2
                prob1 += (q - 0.2*(4-B2x))*R_prob[v]/2
                losing += (1 - (q - 0.2*(4-B2x))/2)*R_prob[v]
        else: #If R is not infront of the goal post than the probabilities of transition are shown below
            if positions[3] == 1: #if Shooting done by B1
                prob1 += (q - 0.2*(4-B1x))*R_prob[v]
                losing += (1 - (q - 0.2*(4-B1x)))*R_prob[v]

            else:#shooting done by B2
                prob1 += (q - 0.2*(4-B2x))*R_prob[v]
                losing += (1 - (q - 0.2*(4-B2x)))*R_prob[v]

    print("transition", Statemap[states],9,Statemap['lose'],0,losing)
    print("transition", Statemap[states],9,Statemap['win'],1,prob1)

 # return T



if __name__ == "__main__":
    #Code for the command line arguments
    parse = argparse.ArgumentParser()
    parse.add_argument("--opponent", type=str, help="Opponent Policy File path")
    parse.add_argument("--p", type=float, help="Value for p")
    parse.add_argument("--q", type=float, help="Policy for Value Function")
    argument = parse.parse_args()
    OPpath =argument.opponent
    p = argument.p
```

```
q = argument.q
op_policy = read_probabilities_file(OPpath) # code to get
    opponent policy
#printing the MDP
print("numStates 8194")
print("numActions 10")
print("end 8192 8193")
give_T_and_R(op_policy,p,q)
print("mdptype episodic")
print("discount 1")
```

3.1 Graphs of Task 2 and Observations

The graphs are shown below:

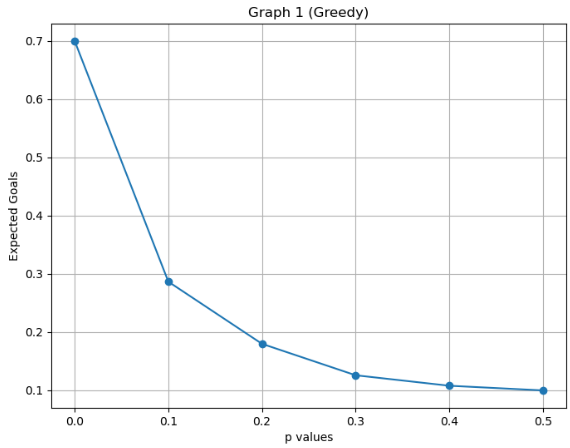

Figure 1: Graph with P varying

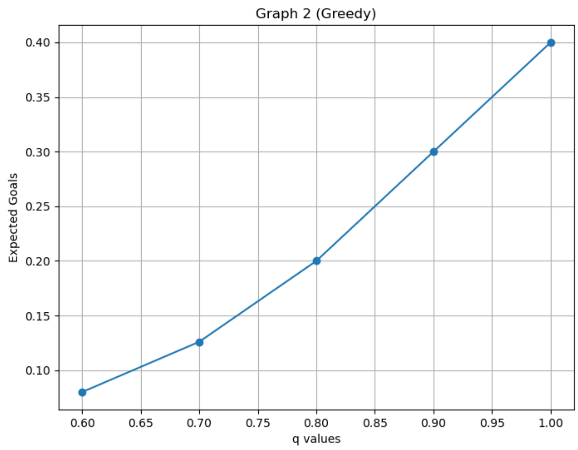

Figure 2: Graph with Q varying

The observation's made are that while keeping q constant and increasing p the expected goals decrease. While keeping p constant and increasing q the expected goals increases as the probability of shooting increases with q. While the expected number of goals are decreasing for keeping q constant and increasing p because increasing p is reducing the movement probability and for p = 0.5 the movement probability is becoming 0 which states that the players whoever has the ball is either passing it or directly of shooting which is decreasing the probability of wining and hence the expected number of goals.