CSB 310: Artificial Intelligence

Lab6: Adversarial Search

Submitted By:

Name: **UMANG KUMAR, MOHIT KUMAR**

Roll No: **201210051, 201210030**

Branch: **CSE**

Semester: **5th Sem**

Submitted To: ***Dr. Chandra Prakash***

Release Date: 26/09/2022 **Submitted Date:** 22/10/2022

NATIONAL INSTITUTE OF TECHNOLOGY DELHI

Department of Computer Science and Engineering

2022

PART A: Adversarial Problem [15 Marks]

1. Fastest Multi-Agent Reward Collection [5 marks]

Inputs: Consider the maze given in the figure below. The walled tiles are marked in black and your agent A and B cannot move to or through those positions.

Write a python/C program that takes the maze as a 5x5 matrix input where 0 denotes an empty tile, 1 denotes an obstruction/wall, 2 denotes the start state and 3 denotes the reward. Assume valid actions as L,R,U,D,S,N where L=move_left, R=move_right, U=move_up, D=move_down.

Your code should help the agents collect all the rewards individually and record the steps in doing so. The agent with the minimum number of steps to collect the rewards wins that round of the game. Run this game for 10 rounds/Episodes, the agent with the most number of wins after 10 rounds is declared as the winner.

Hints:

a) To achieve this you can use BFS or DFS.

b) Your program should create the appropriate data structure that can capture problem states, as mentioned in the problem.

c) Once the all the goals are reached (i.e. Reward position), program should terminate.

Outputs: The output should contain the number of tiles visited by each agent and the winner for each round. It should also declare the winner of all the rounds combined as "out_advsearch.txt".

Solution:-

Solution With DFS Approach:-

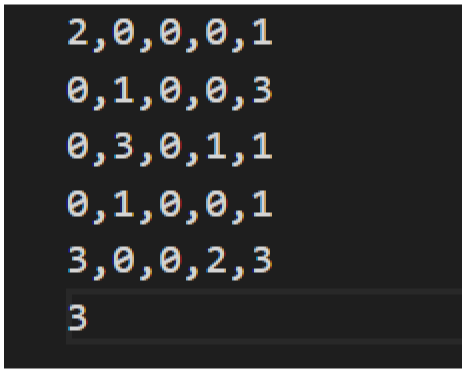
Note:-

Code is also present in Zipped folder:-

Place the input.txt file in the same directory, rest code will take care of it.

Type command :- python TA_6_P1_advsearch.py

Input:



```
2,0,0,0,1
0,1,0,0,3
0,3,0,1,1
0,1,0,0,1
3,0,0,2,3
3
```

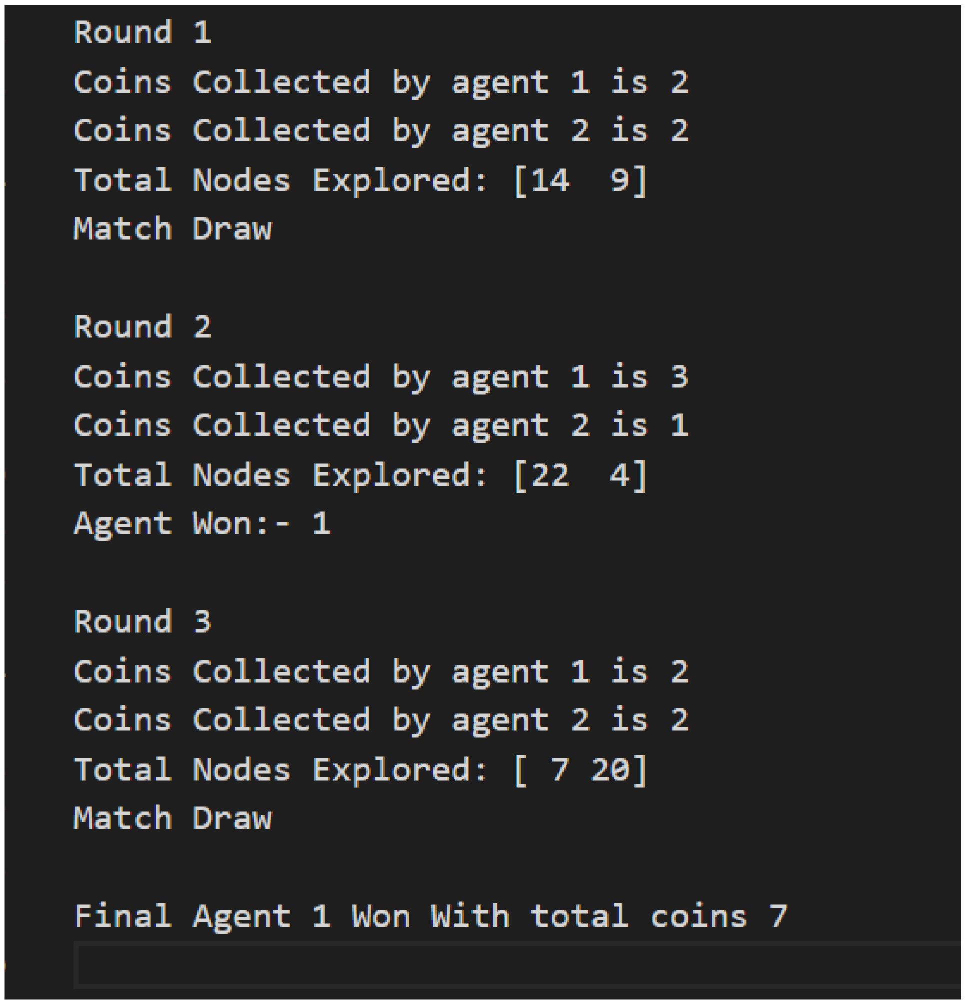
Code:

```python
import sys
import numpy as np
from tkinter import *
from tkinter import ttk
from PIL import Image, ImageTk
import time
import copy
import random
import itertools
from collections import defaultdict
from queue import PriorityQueue
#input and output in file
sys.stdin = open('input.txt', 'r')
sys.stdout = open('out_advsearch.txt', 'w')


# directions to move
directions = [[-1,0],[1,0],[0,1],[0,-1]]
h=80 # height of blocks
indexes=[0,1,2,3]


# dfs function on agent
def dfsOnAgent(agentMaze,dfsPath,agentStartState,coinCount,visitedMaze,n,m,
emptyPath,maze,ep,dt):

    # dfs will run on maze
    x,y = agentStartState
    visitedMaze[x][y]=1
    ep.append(agentStartState)
    # placeGreen(agentStartState)
    dfsPath.append(agentStartState)
    if(agentMaze[x][y]==3):
        return (True,dfsPath)
    agentMaze[x][y]=1

    # trying all directions
    for t in dt:
        a=int(x+directions[t][0])
        b=int(y+directions[t][1])
        newNode=(a,b)
        if(a>=0 and b>=0 and a<n and b<n and (newNode not in emptyPath) and (newNode not
in ep) and  (agentMaze[a][b]==0 or agentMaze[a][b]==3 or agentMaze[a][b]==2)):
            #  print(newNode)
            res,path = dfsOnAgent(agentMaze,dfsPath,newNode,coinCount,visitedMaze,n,m,
emptyPath,maze,ep,dt)
            if(res==True):
                return (True,path )

    agentMaze[x][y]=0
    dfsPath.pop()
    nS=(x,y)
    # placeWhite(nS)
    return (False,dfsPath)




#  agent move function function
def moveAgent(agentNo,maze,n,m,startStates,vMazes,moves,collectedCoins,agentCount,
coinCount,agentCoins,agentsVisitedPath,dt,noOfNodesVisited):

    if(len(collectedCoins)<coinCount):
        dfsPath=[]
        agentMaze=copy.deepcopy(maze)
        vm=np.zeros((int(n),int(m)),dtype=int)
        emptyPath=[]


    #    when coin found to any of agent
        if (maze[startStates[agentNo%agentCount][0]] [startStates[agentNo%agentCount][1]]
==  3 ):
            collectedCoins.append(startStates[agentNo%agentCount])
            agentCoins[agentNo%agentCount].append(startStates[agentNo%agentCount])
            maze[startStates[agentNo%agentCount][0]][startStates[agentNo%agentCount][1]]
= 0
            emptyPath=[]

noOfNodesVisited[agentNo%agentCount]+=len(agentsVisitedPath[agentNo%agentCount])
            agentsVisitedPath[agentNo%agentCount]=[]
```

```
isCoin,pth=dfsOnAgent(agentMaze,dfsPath,startStates[agentNo%agentCount],coinCount,vm,n,m,
agentsVisitedPath[agentNo%agentCount],maze,emptyPath,dt[agentNo%agentCount])
        agentsVisitedPath[agentNo%agentCount].append(startStates[agentNo%agentCount])

        if(len(dfsPath)>1):
           startStates[agentNo%agentCount]=dfsPath[1]
        else:
            agentsVisitedPath[agentNo%agentCount]=[]

        #  again moving next agent

moveAgent((agentNo+1),maze,n,m,startStates,vMazes,moves,collectedCoins,agentCount,coinCou
nt,agentCoins,agentsVisitedPath,dt,noOfNodesVisited)
    else:

        # Deciding each round winner
        totalNodesExplored=np.zeros(int(agentCount),dtype=int)
        def func():
            return 0
        scoreDict2 = defaultdict(func)
        scoreDict={}

        for i in range(0,agentCount):
               scoreDict[i]=len(agentCoins[i])
               scoreDict2[len(agentCoins[i])] = scoreDict2[len(agentCoins[i])]+1
               print("Coins Collected by agent",i+1,"is",len(agentCoins[i]))
               totalNodesExplored[i]+=noOfNodesVisited[i]+1
            #    print(noOfNodesVisited[i])

        print("Total Nodes Explored:",totalNodesExplored)
        maxScore=0
        ag=-1

        for i in scoreDict:
            if(scoreDict[i]>maxScore):
                maxScore=max(scoreDict[i],maxScore)
                ag=i

        if(scoreDict2[maxScore]>1):
            print("Match Draw")
        else :
            print("Agent Won:-",ag+1)

        print("")
```

```
# starting all agent
def agentFunction(maze,n,m,startStates,agentCount,coinCount,dt,totalAgentCoins):

    vMazes=[]
    collectedCoins=[]
    agentCoins=[]
    agentsVisitedPath=[]
    noOfNodesVisited=np.zeros(int(agentCount),dtype=int)
    for i in range(0,agentCount):
         visitedMaze=np.zeros((int(n),int(m)),dtype=int)
         vMazes.append(visitedMaze)
         agentCoins.append([])
         agentsVisitedPath.append([])
        #  noOfNodesVisited.append([])
    moves=random.sample(list(itertools.permutations(indexes)), (agentCount))
    agentNo=0

moveAgent(agentNo,maze,n,m,startStates,vMazes,moves,collectedCoins,agentCount,coinCount,a
gentCoins,agentsVisitedPath,dt,noOfNodesVisited)
    for i in range(0,agentCount):
        totalAgentCoins[i]+= (agentCoins[i])




# main driver function
def driverFunction():

    n=5    # height
    m=5    # width
    maze=np.zeros((int(n),int(m)),dtype=int)
    startStates=[]
    coinCount=0
    # maze formation
    for i in range(0,n):
       a=input().split(',')
       for j in range(0,m):
           if(int(a[j])==2):
            startStates.append((i,j))

           else:
            maze[i][j]=int(a[j])
            if(int(a[j])==3):
                coinCount += 1

    # no of times two agent will run
    testcases=int(input())
    scoreListAgent=[]
    totalAgentCoins=[]
    for s in startStates:
        scoreListAgent.append([])
        totalAgentCoins.append([])


    # randomized direction on move of each agent
    dt=random.sample(list(itertools.permutations(indexes)), 1)
    round=0
    while(testcases>0):
        agentCount=len(startStates)
        round+=1
        print("Round",round)
        Mazes=copy.deepcopy(maze)
        dt=random.sample(list(itertools.permutations(indexes)), agentCount)
        # print(dt)
        agentFunction(Mazes,n,m,startStates,agentCount,coinCount,dt,totalAgentCoins)
            # print("Next Agent")
        testcases -= 1



    # Deciding the final winner
    winner=-1
    maxC=0
    def func():
            return 0
    scoreTotal = defaultdict(func)
    for tc in range(0,len(totalAgentCoins)):
        scoreTotal[len(totalAgentCoins[tc])]+=1
        if(len(totalAgentCoins[tc])>maxC):
            maxC=len(totalAgentCoins[tc])
            winner=tc

    if(scoreTotal[maxC]>1):
        print('Total Series Draw')
    else:
        print("Final Agent",winner+1,"Won With total coins",maxC)


driverFunction()
```

Output:



```
Round 1
Coins Collected by agent 1 is 2
Coins Collected by agent 2 is 2
Total Nodes Explored: [14  9]
Match Draw

Round 2
Coins Collected by agent 1 is 3
Coins Collected by agent 2 is 1
Total Nodes Explored: [22  4]
Agent Won:- 1

Round 3
Coins Collected by agent 1 is 2
Coins Collected by agent 2 is 2
Total Nodes Explored: [ 7 20]
Match Draw

Final Agent 1 Won With total coins 7
```

Observations:

Here we are running both two agents individually , and both the agents will try to get coins . Here we are using the dfs algorithm to run independently. And we need to keep track of the winner and total coins collected by each agent and path followed by each agent. We need to run those agents for more than one round and at each round we need to know the winner of each round. And once all the coins are collected the program will end. For each Agent selection of the next is randomized with random function.

Solution With BFS Approach:-

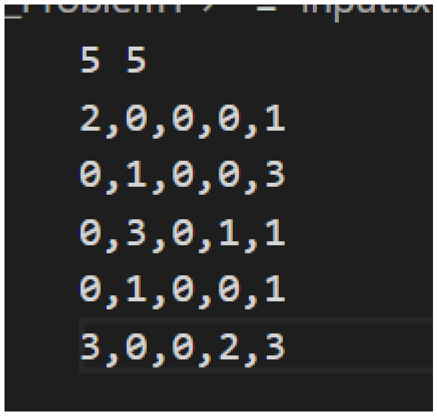
Note:-

Code is also present in Zipped folder:-

Place the input.txt file in the same directory, rest code will take care of it.

Type command :- python lab6_p2_MinMax.py

Input:



```
5 5
2,0,0,0,1
0,1,0,0,3
0,3,0,1,1
0,1,0,0,1
3,0,0,2,3
```

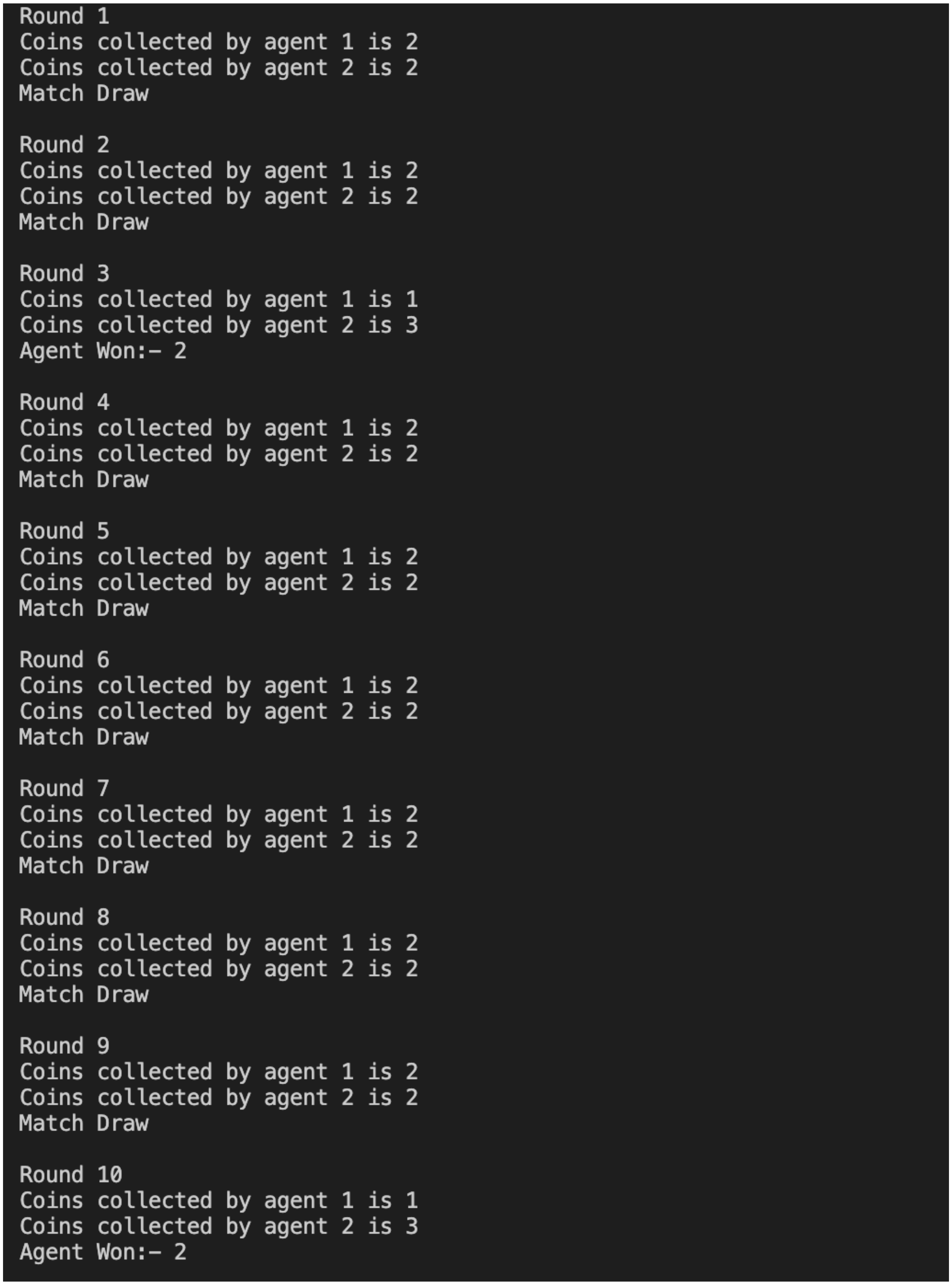
Code:

```
import numpy as np
import random
import sys


sys.stdin = open('input.txt', 'r')
sys.stdout = open('out_advsearch.txt', 'w')

class MultiAgentBfs:

    def __init__(self, pos, world, row, col, n):
        self.depth = np.zeros(n)
        self.row = row
        self.col = col
        self.agentPos = pos
        self.world = world
        self.n = n
        self.initalise()
        self.totalRewardsLeft = (world == 3).sum()
        self.dx = [0, 0, 1, -1]
        self.dy = [1, -1, 0, 0]

        self.simulate()

    def getDirection(self, frm, to):
        if frm == [-1, -1]:
            return ''

        X = to[0] - frm[0]
        Y = to[1] - frm[1]

        if X == 0:
            if Y == 1:
                return 'R'
            else:
                return 'L'
        else:
            if X == 1:
                return 'D'
            else:
                return 'U'

    def initalise(self):
        self.queue = []
        for i in range(self.n):
            self.queue.append([[[[-1, -1], self.agentPos[i]]]])

        self.actions = []
        for i in range(self.n):
            self.actions.append([])

        self.vis = []
        for i in range(self.n):
            self.vis.append(np.zeros((self.row, self.col)))

        self.score = np.zeros(self.n)

        # print(self.vis)

    def isReward(self, pos):
        if(self.world[pos[0]][pos[1]] == 3):
            return True

        return False

    def isValidMove(self, x, y):
        if x >= 0 and y >= 0 and x < self.row and y < self.col:
            return True
```

```python
        return False

    def moveAgent(self, agent):

        # if all the nodes are visited at depth i, then increment depth
        # print(self.queue[agent], self.totalRewardsLeft)
        if len(self.queue[agent][int(self.depth[agent])]) == 0:
            # print("hello\n")
            self.depth[agent] += 1
            self.moveAgent(agent)
            return

        rewards = []

        # find no of rewards nodes
        # for i in self.queue[agent][int(self.depth[agent])]:
        #     if self.isReward(i[1]):
        #         rewards.append(i)

        curNode = []

        if len(rewards):
            # pick random from rewards
            curNode = rewards[random.randrange(len(rewards))]
        else:
            # pick from non-reward Nodes
            l = len(self.queue[agent][int(self.depth[agent])])
            curNode = self.queue[agent][int(
                self.depth[agent])][random.randrange(l)]

        # remove the curNode from the queue
        self.queue[agent][int(self.depth[agent])].remove(curNode)

        # append actions
        self.actions[agent].append(self.getDirection(curNode[0], curNode[1]))
        # if curNode in rewards:
        if self.isReward(curNode[1]):
            self.actions[agent].append('S')
            self.totalRewardsLeft -= 1
            self.score[agent] += 1

        for i in range(4):
            x = curNode[1][0] + self.dx[i]
            y = curNode[1][1] + self.dy[i]

            if self.isValidMove(x, y) and self.vis[agent][x][y] == 0:
                if len(self.queue[agent]) - 1 <= self.depth[agent]:
                    self.queue[agent].append([])

                # print(self.queue)
                self.queue[agent][int(self.depth[agent]) +
                                  1].append([curNode[1], [x, y]])
                self.vis[agent][x][y] = 1

    def simulate(self):
        agent = 0
        while self.totalRewardsLeft:
            agent = agent % self.n
            self.moveAgent(agent)
            agent += 1

        # print(self.actions)
        print(self.score)


if __name__ == "__main__":
    row, col = map(int, input().split())

    world = np.zeros((row, col), dtype=int)
    for i in range(row):
        world[i] = np.array(list(map(int, input().split(','))))

    for i in range(10):
        MultiAgentBfs([[0, 0], [4, 3]], world, 5, 5, 2)
```

Output:

At each line numbers represent the no of coins collected by each agent



```
Round 1
Coins collected by agent 1 is 2
Coins collected by agent 2 is 2
Match Draw

Round 2
Coins collected by agent 1 is 2
Coins collected by agent 2 is 2
Match Draw

Round 3
Coins collected by agent 1 is 1
Coins collected by agent 2 is 3
Agent Won:- 2

Round 4
Coins collected by agent 1 is 2
Coins collected by agent 2 is 2
Match Draw

Round 5
Coins collected by agent 1 is 2
Coins collected by agent 2 is 2
Match Draw

Round 6
Coins collected by agent 1 is 2
Coins collected by agent 2 is 2
Match Draw

Round 7
Coins collected by agent 1 is 2
Coins collected by agent 2 is 2
Match Draw

Round 8
Coins collected by agent 1 is 2
Coins collected by agent 2 is 2
Match Draw

Round 9
Coins collected by agent 1 is 2
Coins collected by agent 2 is 2
Match Draw

Round 10
Coins collected by agent 1 is 1
Coins collected by agent 2 is 3
Agent Won:- 2
```

Explanation:

A normal breadth first search is done from both sides (from agents). The agents will explore the nodes turn by turn. The levels of the nodes are maintained so that the nodes on the higher levels must be explored first. The agent who scores more goals will win the match. The nodes picking from a level are randomized to show variation in the result; otherwise the result would have been the same for all the rounds. The object oriented fashion is adopted to write the code for understandability and maintenance of the code.

2. Fastest Multi-Agent Reward Collection Using Minimax algorithm [10 Marks]

Inputs:

In the above problem, we make a small modification by making the game a turn based one. Agent A will have the first turn, then B and so on till one of them ends up collecting all the rewards.

Use a min-max algorithm to achieve this and declare the winner of the game. You need to do this only for 1 round.

In your output file, include the visiting sequence for each agent and the eventual winner of the game.

Note:-

Code is also present in Zipped folder:-

Place the input.txt file in the same directory, rest code will take care of it.

Type command :- python lab6_p1_advsearch.py

Code:-

```
import random

class Game:
   def __init__(self, world, row, col, posA, posB):
       self.world = world
       self.row = row
       self.col = col
       self.agentAPos = [posA, [-1, -1]]
       self.agentBPos = [posB, [-1, -1]]
       self.dx = [0, 0, 1, -1]
       self.dy = [-1, 1, 0, 0]
       self.agentActions = [[], []]
```

```python
    def getLegalActions(self, agent):
        actions = []
        x = self.agentAPos[0][0]
        y = self.agentAPos[0][1]
        px = self.agentAPos[1][0]
        py = self.agentAPos[1][1]

        if agent == 1:
            x = self.agentBPos[0][0]
            y = self.agentBPos[0][1]
            px = self.agentBPos[1][0]
            py = self.agentBPos[1][1]

        for i in range(4):
            xx = x + self.dx[i]
            yy = y + self.dy[i]
            if xx >= 0 and xx < self.row and yy >= 0 and yy < self.col:
                if [xx, yy] != [px, py] and [xx, yy] != self.agentBPos and [xx, yy] != 
self.agentAPos and self.world[xx][yy] != 1:
                    actions.append(i)

        return actions

    def getMinDistanceFromGoal(self, agent):
        agentPos = self.agentAPos[0]
        if agent == 1:
            agentPos = self.agentBPos[0]

        distance = []
        for i in range(self.row):
            for j in range(self.col):
                if self.world[i][j] == 3:
                    dist = pow(agentPos[0] - i, 2) + pow(agentPos[1] - j, 2)
                    distance.append(dist)

        return distance

    def getGoals(self):
        goals = 0
        for i in range(self.row):
            for j in range(self.col):
                if self.world[i][j] == 3:
                    goals += 1

        return goals

    def moveAgent(self, agent, action):
        if action == -1:
            return

        self.agentActions[agent].append(action)
        if agent == 0:
            self.world[self.agentAPos[0][0]][self.agentAPos[0][1]] = 0
            cur = [self.agentAPos[0][0], self.agentAPos[0][1]]
            self.agentAPos[0][1] = self.agentAPos[0][1] + self.dy[action]
            self.agentAPos[0][0] += self.dx[action]
            # print(self.agentAPos, self.getGoals())
            if self.world[self.agentAPos[0][0]][self.agentAPos[0][1]] == 3:
                self.world[self.agentAPos[0][0]][self.agentAPos[0][1]] = 0
            self.agentAPos[1] = cur
            self.world[self.agentAPos[0][0]][self.agentAPos[0][1]] = 2
            # print(self.agentAPos)/

        else:
            cur = [self.agentBPos[0][0], self.agentBPos[0][1]]
            self.world[self.agentBPos[0][0]][self.agentBPos[0][1]] = 0
            self.agentBPos[0][0] += self.dx[action]
            self.agentBPos[0][1] += self.dy[action]
            # print(self.agentBPos, self.getGoals())
            if self.world[self.agentBPos[0][0]][self.agentBPos[0][1]] == 3:
                self.world[self.agentBPos[0][0]][self.agentBPos[0][1]] = 0

            self.agentBPos[1] = cur
            self.world[self.agentBPos[0][0]][self.agentBPos[0][1]] = 2
            # print(self.agentBPos)

    def clone(self):
        worldd = []
        for i in range(self.row):
            worldd.append([])
            for j in range(self.col):
                worldd[i].append(self.world[i][j])

        agentPosA = [self.agentAPos[0][0], self.agentAPos[0][1]]
        agentPosB = [self.agentBPos[0][0], self.agentBPos[0][1]]

        objClone = Game(worldd, self.row, self.col,
                        agentPosA, agentPosB)

        objClone.agentAPos[1] = self.agentAPos[1]
        objClone.agentBPos[1] = self.agentBPos[1]

        actionss = []
        for i in self.agentActions:
            actionss.append(i)
        objClone.agentActions = actionss

        return objClone


class MinMax:

    def __init__(self, gameState, depth):
        self.depth = depth
        self.gameState = gameState
        self.simulate(gameState, 0)
```

```
def simulate(self, gameState, agent):
    if gameState.getGoals() == 0:
        return

    result = self.minmax(gameState, agent, 1)
    gameState.moveAgent(agent, result[1])
    lst = gameState.world
    for i in lst:
        print(i)
    print()

    self.simulate(gameState, agent ^ 1)

def minmax(self, gameState, agent, depth):
    if depth == self.depth:
        return [self.evaluationFunction(gameState, agent), -1]

    if gameState.getGoals == 0:
        return [self.evaluationFunction(gameState, agent), -1]

    if not gameState.getLegalActions(agent):
        return [self.evaluationFunction(gameState, agent), -1]

    result = []
    for action in gameState.getLegalActions(agent):
        tempGameState = gameState.clone()
        tempGameState.moveAgent(agent, action)

        if not result:
            result = self.minmax(tempGameState, agent ^ 1, depth + 1)
            result[1] = action
        else:
            nextValue = self.minmax(tempGameState, agent ^ 1, depth + 1)
            if agent:
                if result[0] <= nextValue[0]:
                    result[1] = action
                    result[0] = nextValue[0]
            else:
                if result[0] >= nextValue[0]:
                    result[0] = nextValue[0]
                    result[1] = action

def evaluationFunction(self, gameState, agent):

    minA = gameState.getMinDistanceFromGoal(0)
    minB = gameState.getMinDistanceFromGoal(1)

    hueristic = []
    for i in range(len(minA)):
        hueristic.append(minA[i] - minB[i])

    if agent == 0:
            return min(minA)
        else:
            return -min(minB)


if __name__ == "__main__":
    row, col = map(int, input().split())

    world = []
    for i in range(row):
        world.append(list(map(int, input().split(','))))

    posA = []
    posB = []

    for i in range(row):
        for j in range(col):
            if world[i][j] == 2:
                if len(posA) == 0:
                    posA = [i, j]
                else:
                    posB = [i, j]

    gameState = Game(world, row, col, posA, posB)
    play = MinMax(gameState, 2)
```

Output:

We tried our best to make the min-max algorithm work on the grid system. Due to less exposure to the practical knowledge of min-max algorithms we are not able to solve this problem completely. We tried to write the code due to constraint and due to the poor heuristic/evaluation function the agent got stuck in an infinite loop. We tried different heuristic functions but none suits the problem hence the output remains a mystery, at least for us :)

PART B: Exploratory Problem [25 Marks]

3. Multi-Agent Search

This problem allows you to visualize the results of the techniques you implement. Pac-Man provides a challenging problem environment that demands creative solutions of real-world AI problems. In this project, you will design agents for the classic version of Pacman, including ghosts. Along the way, you will implement both minimax and AlphaBeta search and try your hand at evaluation function design.

This lab assignment is inspired by Project 2: Multi-Agent Search, which is a part of a recent offering of CS188 at UC Berkeley[1]. We thank the authors at Berkeley for making their project available to the public.

Aim: Students will implement both minimax and Alpha-Beta search.

The code base has not changed much from the previous project, but please start with a fresh installation, rather than intermingling files from TA5-6.

This project includes an autograder for you to grade your answers on your machine. This can be run on all questions with the command:

```
python autograder.py
```

It can be run for one particular question, such as q2, by:

```
python autograder.py -q q2
```

It can be run for one particular test by commands of the form:

```
python autograder.py -t test_cases/q2/0-small-tree
```

By default, the autograder displays graphics with the -t option, but doesn't with the -q option. You can force graphics by using the --graphics flag, or force no graphics by using the --no-graphics flag.

The code for this assignment is provided to you as TA_6_multiagent zip folder. You can download all the code and supporting files as a Folder/ zip archive from the shared link. The code for this project consists of several Python files, some of which you will need to read and understand in order to complete the assignment, and some of which you can ignore.

Assumption : The projects for this class assume you use Python 3.6.

Files you might want to look at:

pacman.py The main file that runs Pacman games. This file also describes a Pacman GameState type, which you will use extensively in this project.

game.py The logic behind how the Pacman world works. This file describes several supporting types like AgentState, Agent, Direction, and Grid.

util.py Useful data structures for implementing search algorithms. You don't need to use these for this project, but may find other functions defined here to be useful.

Supporting files you can ignore:

graphicsDisplay.py	Graphics for Pacman
graphicsUtils.py	Support for Pacman graphics
textDisplay.py	ASCII graphics for Pacman
ghostAgents.py	Agents to control ghosts
keyboardAgents.py	Keyboard interfaces to control Pacman
layout.py	Code for reading layout files and storing their contents
autograder.py	Project autograder
testParser.py	Parses autograder test and solution files
testClasses.py	General autograding test classes
test_cases/	Directory containing the test cases for each question
multiagentTestClasses.py	Project 2 specific autograding test classes

Files to Edit and Submit: You will fill in portions of multiAgents.py during the assignment.

Welcome to Multi-Agent Pacman

TASK 0: Understanding the Working of Pacman Game (Ungraded)

After downloading the code from the shared link, unzipping it, and changing to the directory, you should be able to :

TASK 0: Play the game of Pacman by typing the following at the command line:

```
python pacman.py
```

Now, run the provided ReflexAgent in multiAgents.py

```
python pacman.py -p ReflexAgent
```

Note that it plays quite poorly even on simple layouts:

```
python pacman.py -p ReflexAgent -l testClassic
```

Inspect its code in multiAgents.py and make sure you understand what it's doing.

Question 1 (4 points) : Reflex Agent

Improve the ReflexAgent inmultiAgents.py to play respectably. The provided reflex agent code provides some helpful examples of methods that query the GameState for information. A capable reflex agent will have to consider both food locations and ghost locations to perform well. Your agent should easily and reliably clear the testClassic layout:

```
python pacman.py -p ReflexAgent -l testClassic
```

Try out your reflex agent on the default mediumClassic layout with one ghost or two (and animation off to speed up the display):

```
python pacman.py --frameTime 0 -p ReflexAgent -k 1
```

```
python pacman.py --frameTime 0 -p ReflexAgent -k 2
```

How does your agent fare? It will likely often die with 2 ghosts on the default board, unless your evaluation function is quite good.

Note: Remember that newFood has the function asList()

Note: As features, try the reciprocal of important values (such as distance to food) rather than just the values themselves.

Note: The evaluation function you're writing is evaluating state-action pairs; in later parts of the project, you'll be evaluating states.

Note: You may find it useful to view the internal contents of various objects for debugging. You can do this by printing the objects' string representations. For example, you can print newGhostStates with print(newGhostStates).

Options: Default ghosts are random; you can also play for fun with slightly smarter directional ghosts using -g DirectionalGhost. If the randomness is preventing you from telling whether your agent is improving, you can use -f to run with a fixed random seed (same random choices every game). You can also play multiple games in a row with -n. Turn off graphics with -q to run lots of games quickly.

Grading: We will run your agent on the openClassic layout 10 times. You will receive 0 points if your agent times out, or never wins. You will receive 1 point if your agent wins at least 5 times, or 2 points if your agent wins all 10 games. You will receive an addition 1 point if your agent's average score is greater than 500, or 2 points if it is greater than 1000. You can try your agent out under these conditions with

```
python autograder.py -q q1
```

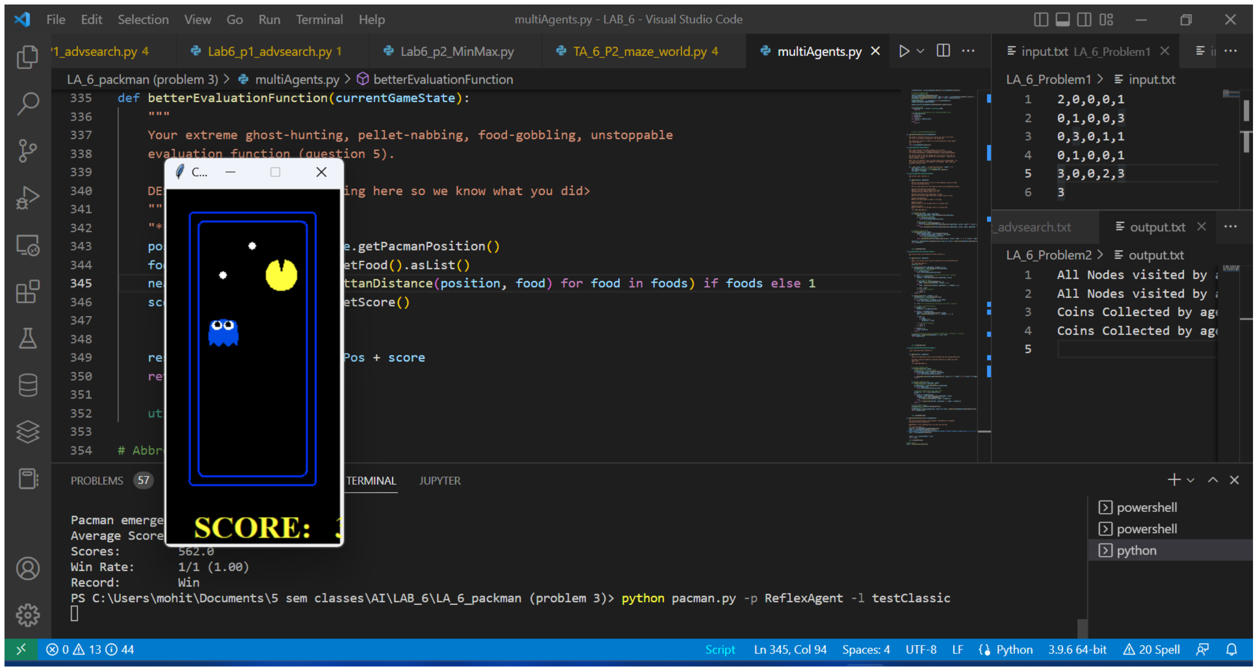
To run it without graphics, use:

```
python autograder.py -q q1 --no-graphics
```

Don't spend too much time on this question, though, as the meat of the project lies ahead.

Output:

```
python pacman.py -p ReflexAgent -l testClassic
```



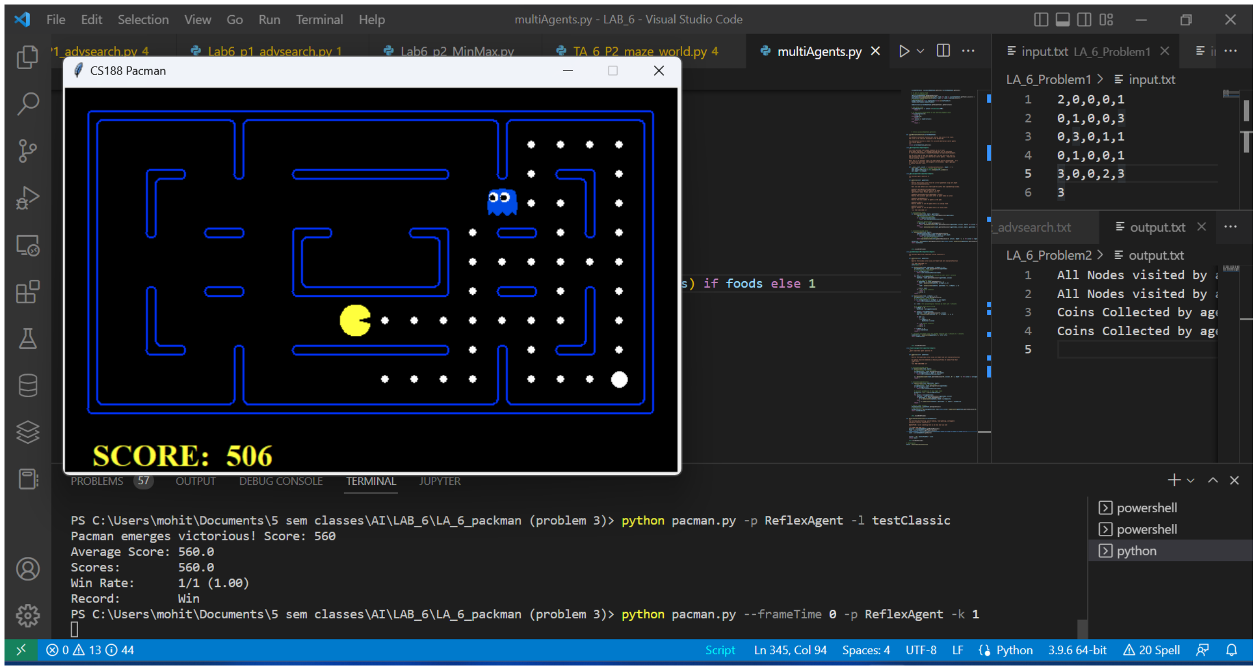
```
def betterEvaluationFunction(currentGameState):
    """
    Your extreme ghost-hunting, pellet-nabbing, food-gobbling, unstoppable
    evaluation function (question 5).

    DE                                    ing here so we know what you did>
    ""
    "*
    po                                    e.getPacmanPosition()
    fo                                    etFood().asList()
    ne                                    ttanDistance(position, food) for food in foods) if foods else 1
    sc                                    etScore()


    re                                    Pos + score
    re

    ut

# Abbr
```

```
2,0,0,0,1
0,1,0,0,3
0,3,0,1,1
0,1,0,0,1
3,0,0,2,3
3
```

```
All Nodes visited by
All Nodes visited by
Coins Collected by ag
Coins Collected by ag
```

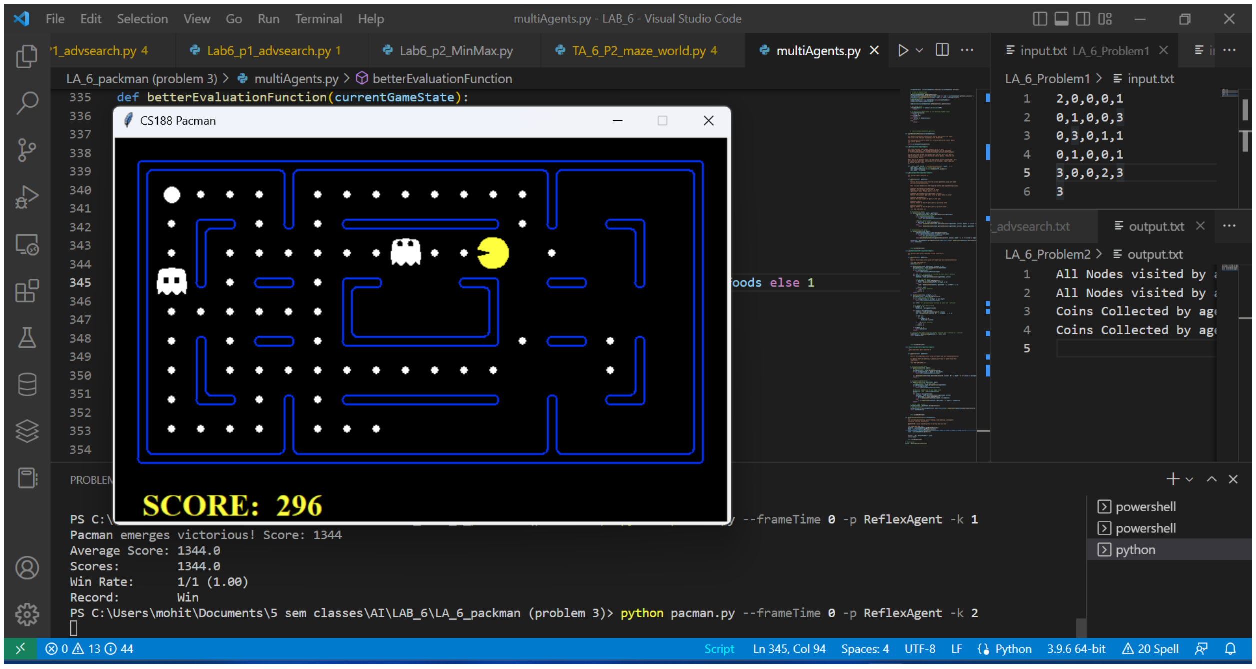
```
Pacman emerge
Average Score
Scores:      562.0
Win Rate:    1/1 (1.00)
Record:      Win
PS C:\Users\mohit\Documents\5 sem classes\AI\LAB_6\LA_6_packman (problem 3)> python pacman.py -p ReflexAgent -l testClassic
```

python pacman.py --frameTime 0 -p ReflexAgent -k 1



```
PS C:\Users\mohit\Documents\5 sem classes\AI\LAB_6\LA_6_packman (problem 3)> python pacman.py -p ReflexAgent -l testClassic
Pacman emerges victorious! Score: 560
Average Score: 560.0
Scores:        560.0
Win Rate:      1/1 (1.00)
Record:        Win
PS C:\Users\mohit\Documents\5 sem classes\AI\LAB_6\LA_6_packman (problem 3)> python pacman.py --frameTime 0 -p ReflexAgent -k 1
```

python pacman.py --frameTime 0 -p ReflexAgent -k 2



```
def betterEvaluationFunction(currentGameState):
```

```
2,0,0,0,1
0,1,0,0,3
0,3,0,1,1
0,1,0,0,1
3,0,0,2,3
3
```

```
All Nodes visited by a
All Nodes visited by a
Coins Collected by age
Coins Collected by age
```

```
PS C:\                                                                        --frameTime 0 -p ReflexAgent -k 1
Pacman emerges victorious! Score: 1344
Average Score: 1344.0
Scores:        1344.0
Win Rate:      1/1 (1.00)
Record:        Win
PS C:\Users\mohit\Documents\5 sem classes\AI\LAB_6\LA_6_packman (problem 3)> python pacman.py --frameTime 0 -p ReflexAgent -k 2
```

Observations:-

Here we have implemented the evaluationFunction for the reflex agent in the multiagent.py file. In that our first priority is to keep the pacman away from the ghosts and also we need to keep track of whether the move will increase the score or not. And then wee need to implement that the we need to move to the first or nearest move .

Question 2 (5 points): Minimax

Now you will write an adversarial search agent in the provided MinimaxAgent class stub in multiAgents.py. Your minimax agent should work with any number of ghosts, so you'll have to write an algorithm that is slightly more general than what you've previously seen in lecture. In particular, your minimax tree will have multiple min layers (one for each ghost) for every max layer.

Your code should also expand the game tree to an arbitrary depth. Score the leaves of your minimax tree with the supplied self.evaluationFunction, which defaults to scoreEvaluationFunction. MinimaxAgent extends MultiAgentSearchAgent, which gives access to self.depth and self.evaluationFunction. Make sure your minimax code makes reference to these two variables where appropriate as these variables are populated in response to command line options.

Important: A single search ply is considered to be one Pacman move and all the ghosts' responses, so depth 2 search will involve Pacman and each ghost moving two times.

Grading: We will be checking your code to determine whether it explores the correct number of game states. This is the only reliable way to detect some very subtle bugs in implementations of minimax. As a result, the autograder will be very picky about how many times you call GameState.generateSuccessor.

If you call it any more or less than necessary, the autograder will complain. To test and debug your code, run

```
python autograder.py -q q2
```

```
python autograder.py -q q2 --no-graphics
```

Hints and Observations:

• Hint: Implement the algorithm recursively using helper function(s).

• The correct implementation of minimax will lead to Pacman losing the game in some tests. This

is not a problem: as it is correct behaviour, it will pass the tests.

• The evaluation function for the Pacman test in this part is already written (self.evaluationFunction).

You shouldn't change this function, but recognize that now we're evaluating states rather than actions, as we were for the reflex agent. Look-ahead agents evaluate future states whereas reflex agents evaluate actions from the current state.

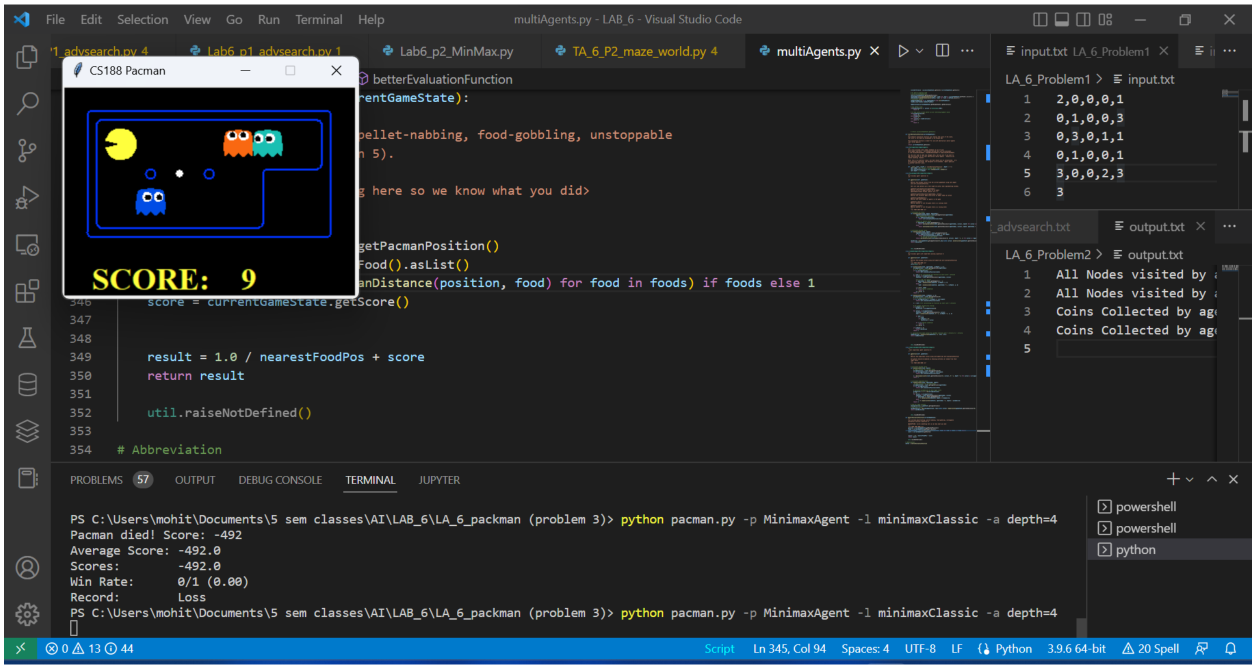
• The minimax values of the initial state in the minimaxClassic layout are 9, 8, 7, -492 for depths 1, 2, 3 and 4 respectively. Note that your minimax agent will often win (665/1000 games for us) despite the dire prediction of depth 4 minimax.

```
python pacman.py -p MinimaxAgent -l minimaxClassic -a depth=4
```

Make sure you understand why Pacman rushes the closest ghost in this case.

Output:-

```
python pacman.py -p MinimaxAgent -l minimaxClassic -a depth=4
```



```
rentGameState):

pellet-nabbing, food-gobbling, unstoppable
n 5).

g here so we know what you did>


getPacmanPosition()
Food().asList()
anDistance(position, food) for food in foods) if foods else 1
    score = currentGameState.getScore()


    result = 1.0 / nearestFoodPos + score
    return result

    util.raiseNotDefined()

# Abbreviation
```

LA_6_Problem1 > input.txt

```
2,0,0,0,1
0,1,0,0,3
0,3,0,1,1
0,1,0,0,1
3,0,0,2,3
3
```

LA_6_Problem2 > output.txt

```
All Nodes visited by a
All Nodes visited by a
Coins Collected by age
Coins Collected by age
```

```
PS C:\Users\mohit\Documents\5 sem classes\AI\LAB_6\LA_6_packman (problem 3)> python pacman.py -p MinimaxAgent -l minimaxClassic -a depth=4
Pacman died! Score: -492
Average Score: -492.0
Scores:        -492.0
Win Rate:      0/1 (0.00)
Record:        Loss
PS C:\Users\mohit\Documents\5 sem classes\AI\LAB_6\LA_6_packman (problem 3)> python pacman.py -p MinimaxAgent -l minimaxClassic -a depth=4
```

Observation:-

In the second part we implemented the min-max of the pacman. We implemented the two minCalculate and maxCalculate function and we keep a track of depth and at each depth decide the we need to calculate the min or max at that level.

Question 3 (5 points): Alpha-Beta Pruning Function

Make a new agent that uses alpha-beta pruning to more efficiently explore the minimax tree, in AlphaBetaAgent. Again, your algorithm will be slightly more general than the pseudocode from the lecture, so part of the challenge is to extend the alpha-beta pruning logic appropriately to multiple minimizer agents.

You should see a speed-up (perhaps depth 3 alpha-beta will run as fast as depth 2 minimax). Ideally, depth 3 on smallClassic should run in just a few seconds per move or faster.

python pacman.py -p AlphaBetaAgent -a depth=3 -l smallClassic

The AlphaBetaAgent minimax values should be identical to the MinimaxAgent minimax values, although the actions it selects can vary because of different tie-breaking behavior. Again, the minimax values of the initial state in the minimaxClassic layout are 9, 8, 7 and -492 for depths 1, 2, 3 and 4 respectively.

Grading: Because we check your code to determine whether it explores the correct number of states, it is important that you perform alpha-beta pruning without reordering children. In other words, successor states should always be processed in the order returned by GameState.getLegalActions. Again, do not call GameState.generateSuccessor more than necessary.

You must not prune on equality in order to match the set of states explored by our autograder. (Indeed, alternatively, but incompatible with our autograder, would be to also allow for pruning on equality and invoke alpha-beta once on each child of the root node, but this will not match the autograder.)

The pseudo-code below represents the algorithm you should implement for this question.

To test and debug your code, run:

```
python autograder.py -q q3
```

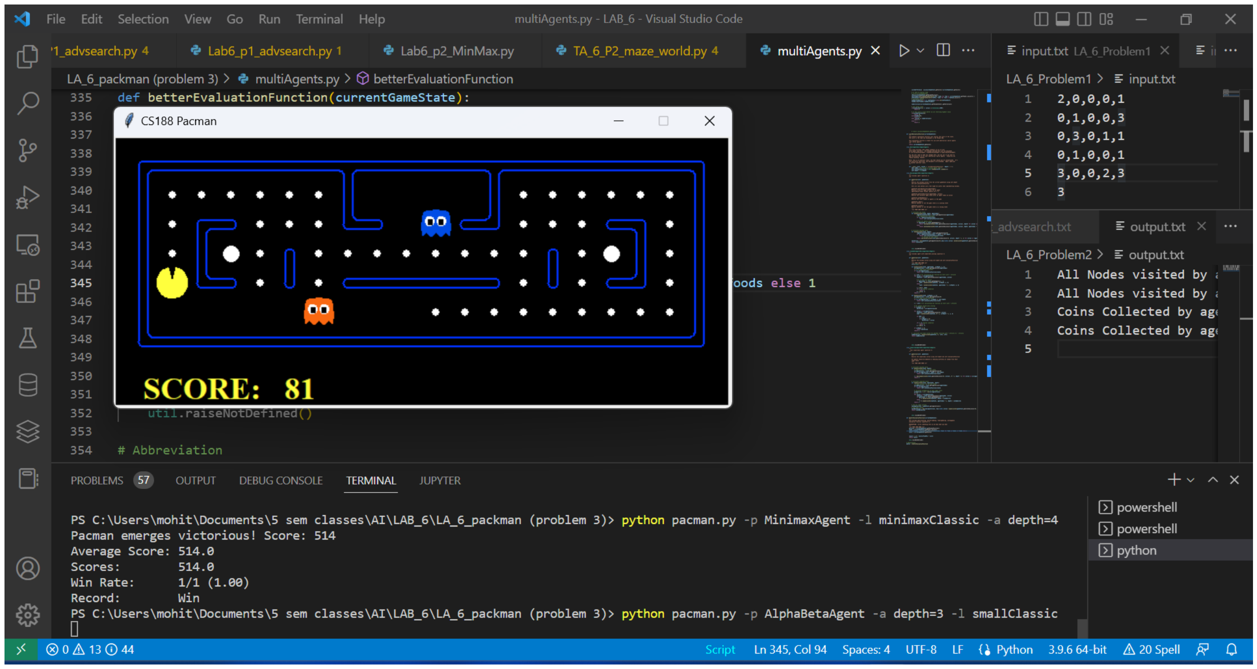
This will show what your algorithm does on a number of small trees, as well as a pacman game. To

run it without graphics, use:

```
python autograder.py -q q3 --no-graphics
```

The correct implementation of alpha-beta pruning will lead to Pacman losing some of the tests. This is not a problem: as it is correct behavior, it will pass the tests.

```
python pacman.py -p AlphaBetaAgent -a depth=3 -l smallClassic
```



```
def betterEvaluationFunction(currentGameState):
    util.raiseNotDefined()

# Abbreviation
```

```
2,0,0,0,1
0,1,0,0,3
0,3,0,1,1
0,1,0,0,1
3,0,0,2,3
3
```

```
All Nodes visited by a
All Nodes visited by a
Coins Collected by age
Coins Collected by age
```

```
PS C:\Users\mohit\Documents\5 sem classes\AI\LAB_6\LA_6_packman (problem 3)> python pacman.py -p MinimaxAgent -l minimaxClassic -a depth=4
Pacman emerges victorious! Score: 514
Average Score: 514.0
Scores:        514.0
Win Rate:      1/1 (1.00)
Record:        Win
PS C:\Users\mohit\Documents\5 sem classes\AI\LAB_6\LA_6_packman (problem 3)> python pacman.py -p AlphaBetaAgent -a depth=3 -l smallClassic
```

Observation :-

Here we have implemented the alpha-beta pruning and more optimized the min-max algorithm and we increase the winning condition of the pacman. Using alpha beta pruning.

The difference between this and above is that we have pruned some of the branches of the search tree. And this increases the efficiency of our algorithm.

Question 4 (5 points): Expectimax

Minimax and alpha-beta are great, but they both assume that you are playing against an adversary who makes optimal decisions. As anyone who has ever won tic-tac-toe can tell you, this is not always

the case. In this question you will implement the ExpectimaxAgent, which is useful for modeling probabilistic behavior of agents who may make suboptimal choices.

As with the search and constraint satisfaction problems covered so far in this class, the beauty of these algorithms is their general applicability. To expedite your own development, we've supplied some test cases based on generic trees. You can debug your implementation on small the game trees using the command:

```
python autograder.py -q q4
```

Debugging on these small and manageable test cases is recommended and will help you to find bugs quickly.

Once your algorithm is working on small trees, you can observe its success in Pacman. Random ghosts are of course not optimal minimax agents, and so modeling them with minimax search may not be appropriate. ExpectimaxAgent, will no longer take the min over all ghost actions, but the expectation according to your agent's model of how the ghosts act. To simplify your code, assume you will only be running against an adversary which chooses amongst their getLegalActions uniformly at random.

To see how the ExpectimaxAgent behaves in Pacman, run:

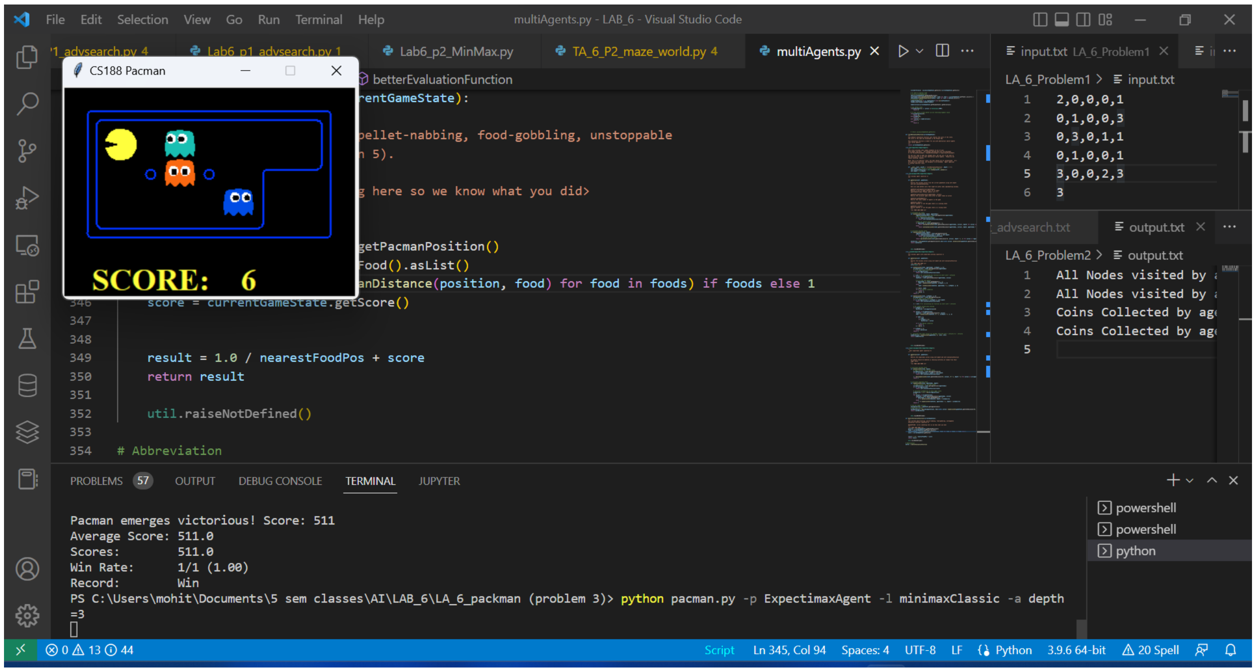
```
python pacman.py -p ExpectimaxAgent -l minimaxClassic -a depth=3
```

You should now observe a more cavalier approach in close quarters with ghosts. In particular, if Pacman perceives that he could be trapped but might escape to grab a few more pieces of food, he'll at least try. Investigate the results of these two scenarios:

```
python pacman.py -p AlphaBetaAgent -l trappedClassic -a depth=3 -q -n 10
```

```
python pacman.py -p ExpectimaxAgent -l trappedClassic -a depth=3 -q -n 10
```

Output:-



```
rentGameState):

pellet-nabbing, food-gobbling, unstoppable
n 5).

g here so we know what you did>


getPacmanPosition()
Food().asList()
anDistance(position, food) for food in foods) if foods else 1
    score = currentGameState.getScore()


    result = 1.0 / nearestFoodPos + score
    return result

    util.raiseNotDefined()

# Abbreviation
```

```
2,0,0,0,1
0,1,0,0,3
0,3,0,1,1
0,1,0,0,1
3,0,0,2,3
3
```

```
All Nodes visited by a
All Nodes visited by a
Coins Collected by age
Coins Collected by age
```

```
Pacman emerges victorious! Score: 511
Average Score: 511.0
Scores:        511.0
Win Rate:      1/1 (1.00)
Record:        Win
PS C:\Users\mohit\Documents\5 sem classes\AI\LAB_6\LA_6_packman (problem 3)> python pacman.py -p ExpectimaxAgent -l minimaxClassic -a depth
=3
```

You should find that your ExpectimaxAgent wins about half the time, while your AlphaBetaAgent always loses. Make sure you understand why the behavior here differs from the minimax case.

The correct implementation of expectimax will lead to Pacman losing some of the tests. This is not a problem: as it is correct behavior, it will pass the tests.

Observation:-

In the expectimax function we will consider the probability of each move an All ghosts should be modeled as choosing uniformly at random from there. We consider probability as the main evaluating function for all legal moves.

Question 5 (6 points): Evaluation Function

Write a better evaluation function for pacman in the provided function betterEvaluationFunction. The evaluation function should evaluate states, rather than actions like your reflex agent evaluation function did. With depth 2 search, your evaluation function should clear the smallClassic layout with one random ghost more than half the time and still run at a reasonable rate (to get full credit,

Pacman should be averaging around 1000 points when he's winning).

Grading: the autograder will run your agent on the smallClassic layout 10 times. We will assign points to your evaluation function in the following way:

• If you win at least once without timing out the autograder, you receive 1 points. Any agent not satisfying these criteria will receive 0 points.

• +1 for winning at least 5 times, +2 for winning all 10 times

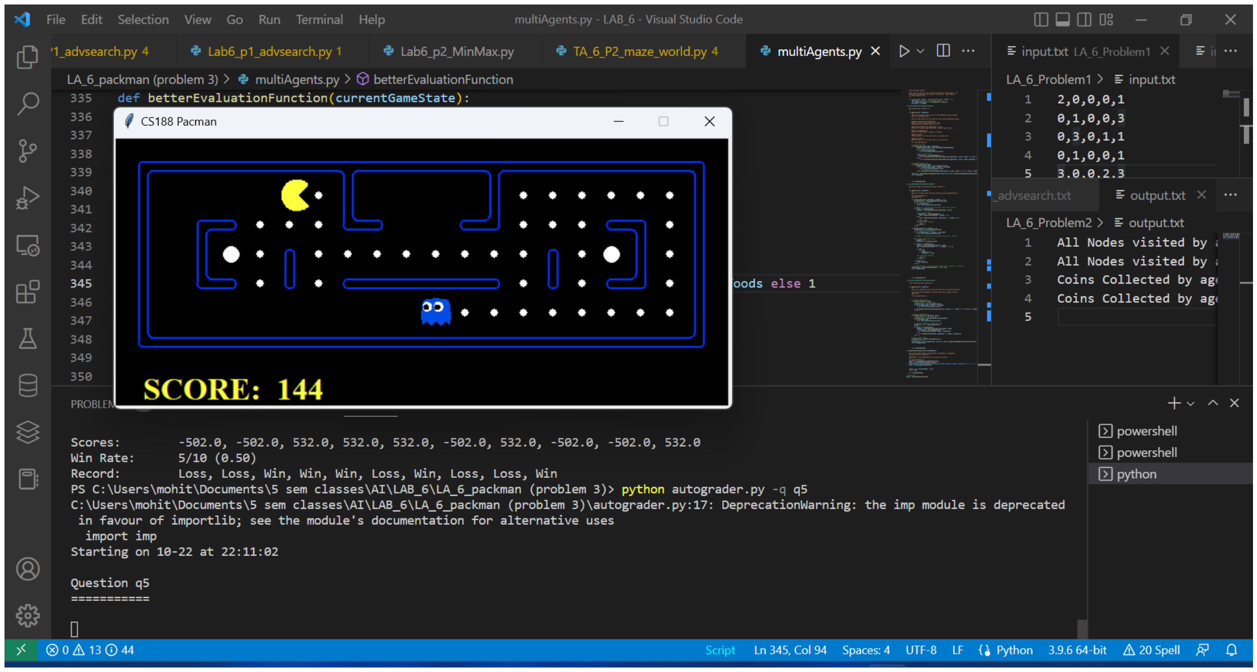
• +1 for an average score of at least 500, +2 for an average score of at least 1000 (including scores on lost games)

• +1 if your games take on average less than 30 seconds on the autograder machine, when run with --no-graphics. The autograder is run on EC2, so this machine will have a fair amount of resources, but your personal computer could be far less performant (netbooks) or far more performant (gaming rigs).

• The additional points for average score and computation time will only be awarded if you win at least 5 times.

• Please do not copy any files from Project 1, as it will not pass the autograder on Gradescope.

You can try your agent out under these conditions with python autograder.py -q q5



```
def betterEvaluationFunction(currentGameState):
```

```
ods else 1
```

```
2,0,0,0,1
0,1,0,0,3
0,3,0,1,1
0,1,0,0,1
3.0.0.2.3
```

```
All Nodes visited by
All Nodes visited by
Coins Collected by ag
Coins Collected by ag
```

```
Scores:        -502.0, -502.0, 532.0, 532.0, 532.0, -502.0, 532.0, -502.0, -502.0, 532.0
Win Rate:      5/10 (0.50)
Record:        Loss, Loss, Win, Win, Win, Loss, Win, Loss, Loss, Win
PS C:\Users\mohit\Documents\5 sem classes\AI\LAB_6\LA_6_packman (problem 3)> python autograder.py -q q5
C:\Users\mohit\Documents\5 sem classes\AI\LAB_6\LA_6_packman (problem 3)\autograder.py:17: DeprecationWarning: the imp module is deprecated
 in favour of importlib; see the module's documentation for alternative uses
  import imp
Starting on 10-22 at 22:11:02

Question q5
===========
```

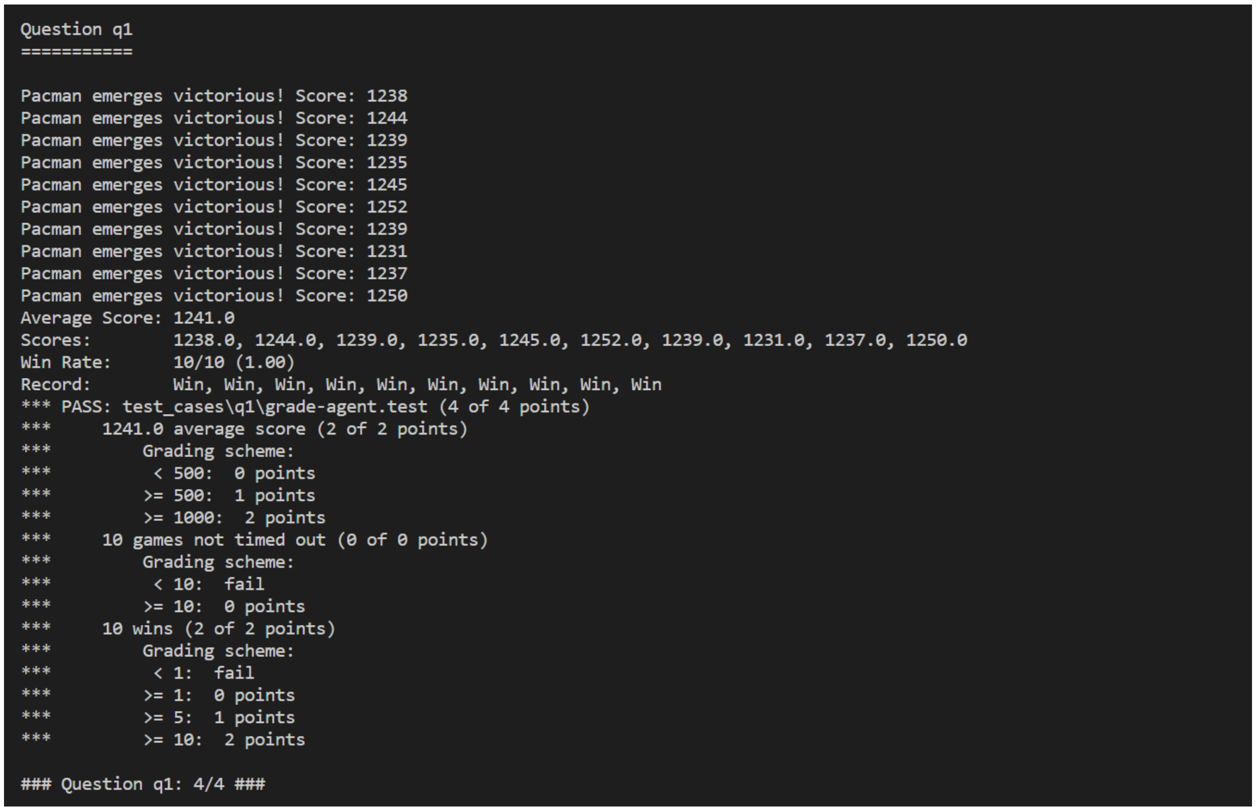
To run it without graphics, use:

python autograder.py -q q5 --no-graphics

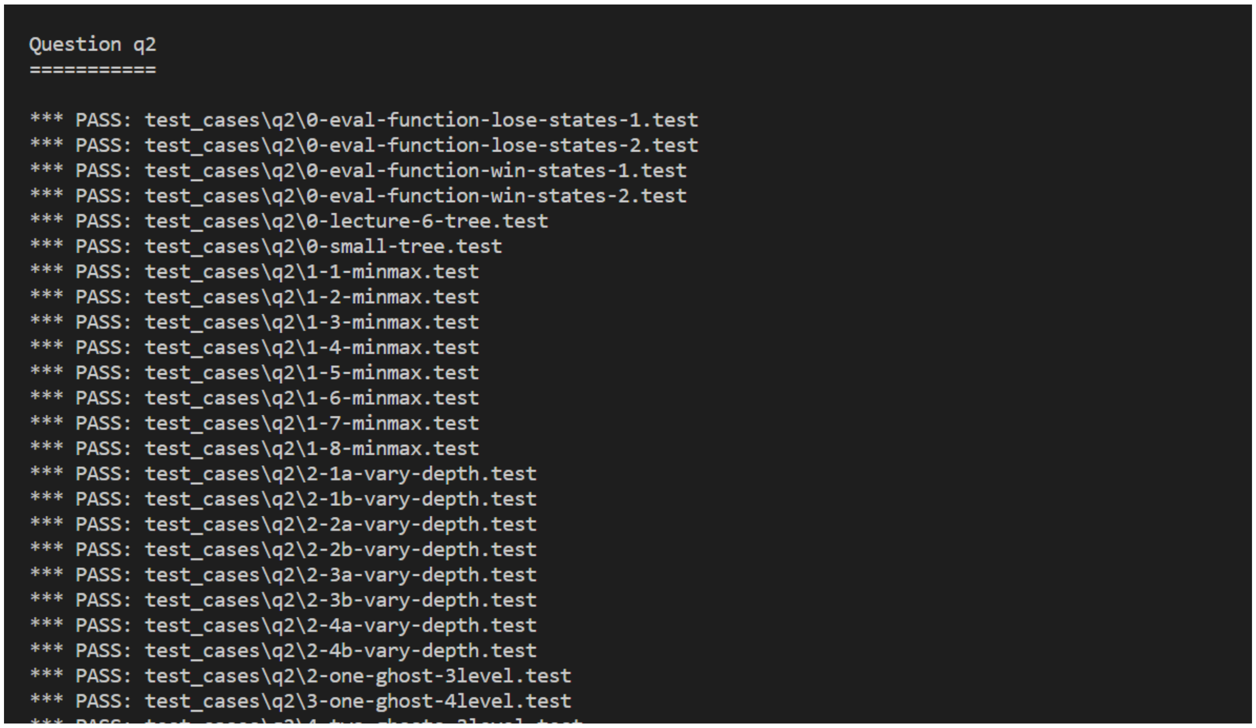
Observation :

In this we only need to complete the evaluation function which will consider the probability of each function for the and return that evaluating value.

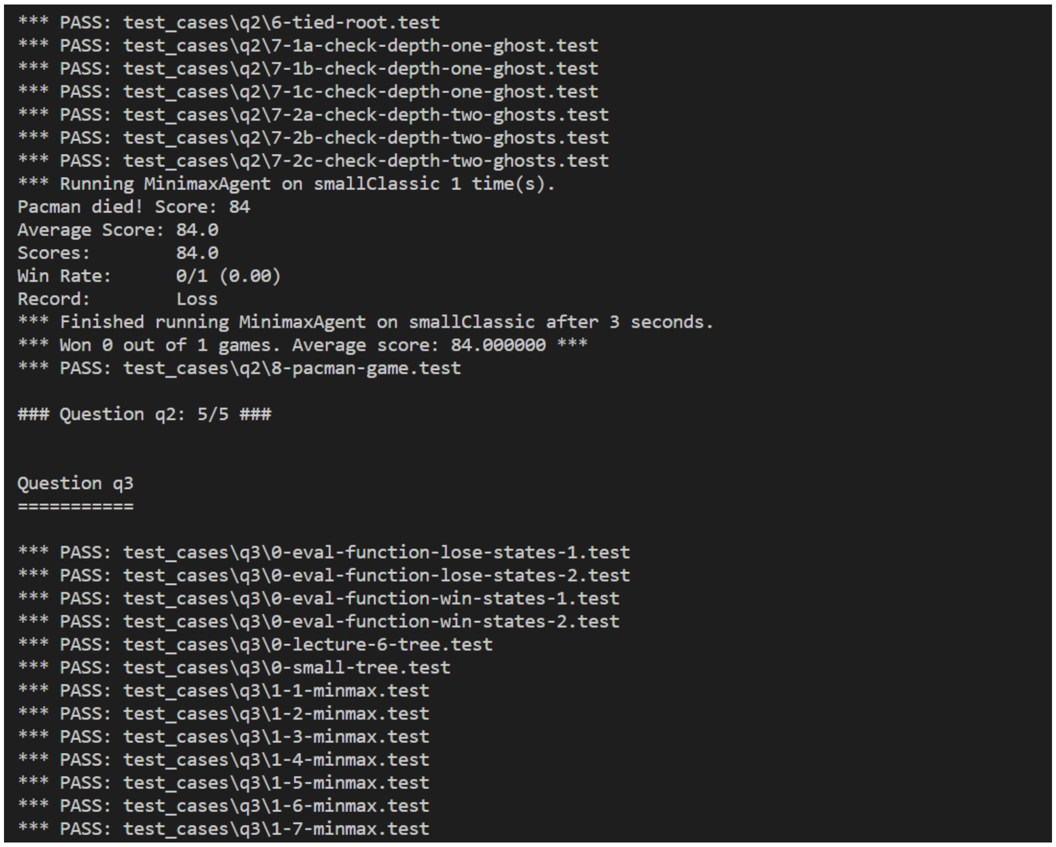
<u>**Output of autograder:**</u>



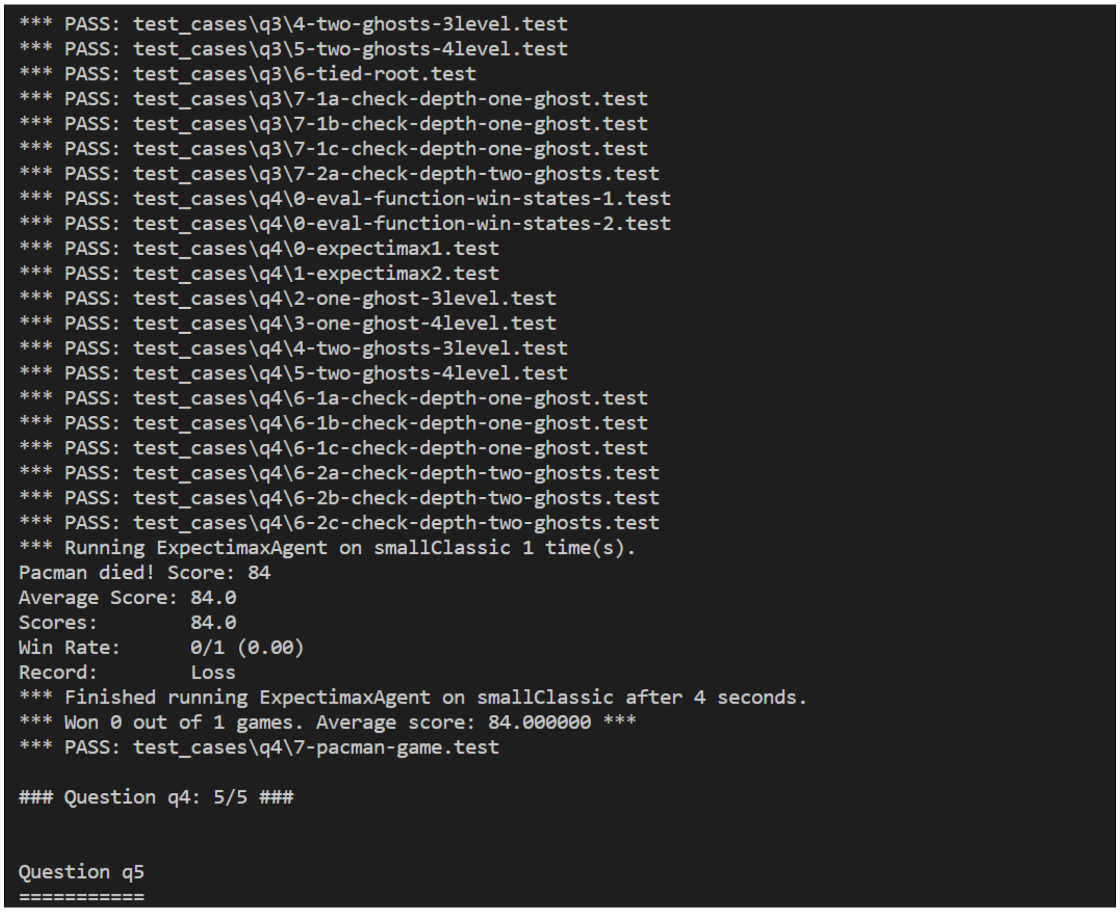
```
Question q1
===========

Pacman emerges victorious! Score: 1238
Pacman emerges victorious! Score: 1244
Pacman emerges victorious! Score: 1239
Pacman emerges victorious! Score: 1235
Pacman emerges victorious! Score: 1245
Pacman emerges victorious! Score: 1252
Pacman emerges victorious! Score: 1239
Pacman emerges victorious! Score: 1231
Pacman emerges victorious! Score: 1237
Pacman emerges victorious! Score: 1250
Average Score: 1241.0
Scores:        1238.0, 1244.0, 1239.0, 1235.0, 1245.0, 1252.0, 1239.0, 1231.0, 1237.0, 1250.0
Win Rate:      10/10 (1.00)
Record:        Win, Win, Win, Win, Win, Win, Win, Win, Win, Win
*** PASS: test_cases\q1\grade-agent.test (4 of 4 points)
***     1241.0 average score (2 of 2 points)
***         Grading scheme:
***          < 500:  0 points
***         >= 500:  1 points
***         >= 1000:  2 points
***     10 games not timed out (0 of 0 points)
***         Grading scheme:
***          < 10:  fail
***         >= 10:  0 points
***     10 wins (2 of 2 points)
***         Grading scheme:
***          < 1:  fail
***         >= 1:  0 points
***         >= 5:  1 points
***         >= 10:  2 points

### Question q1: 4/4 ###
```



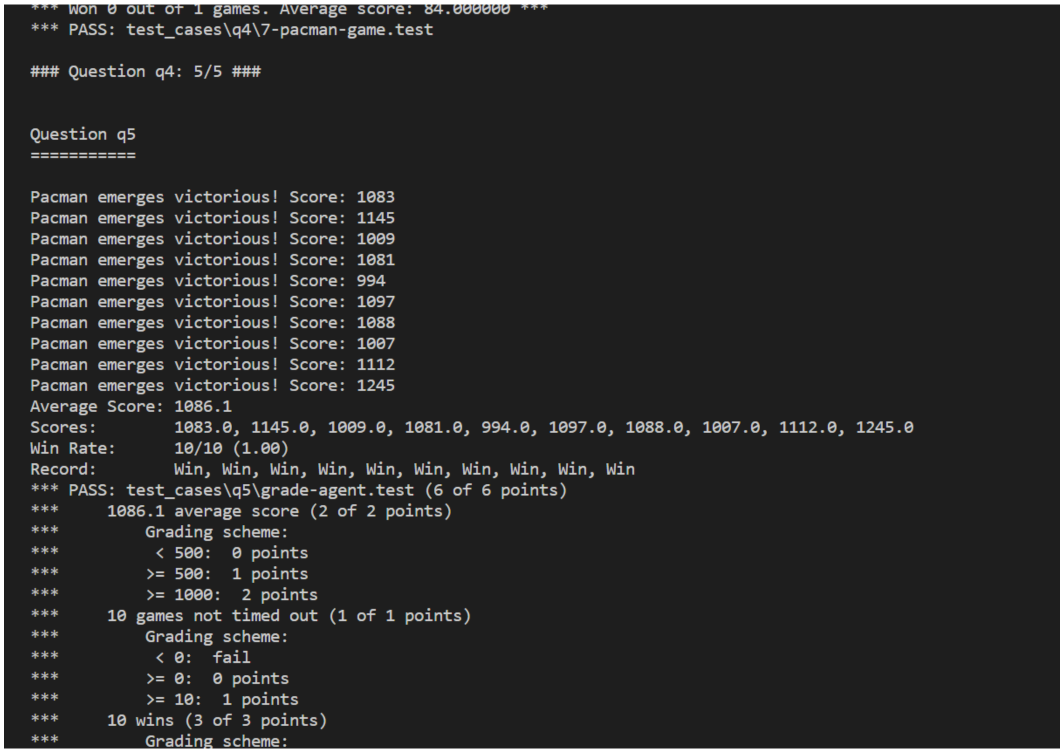
```
Question q2
===========

*** PASS: test_cases\q2\0-eval-function-lose-states-1.test
*** PASS: test_cases\q2\0-eval-function-lose-states-2.test
*** PASS: test_cases\q2\0-eval-function-win-states-1.test
*** PASS: test_cases\q2\0-eval-function-win-states-2.test
*** PASS: test_cases\q2\0-lecture-6-tree.test
*** PASS: test_cases\q2\0-small-tree.test
*** PASS: test_cases\q2\1-1-minmax.test
*** PASS: test_cases\q2\1-2-minmax.test
*** PASS: test_cases\q2\1-3-minmax.test
*** PASS: test_cases\q2\1-4-minmax.test
*** PASS: test_cases\q2\1-5-minmax.test
*** PASS: test_cases\q2\1-6-minmax.test
*** PASS: test_cases\q2\1-7-minmax.test
*** PASS: test_cases\q2\1-8-minmax.test
*** PASS: test_cases\q2\2-1a-vary-depth.test
*** PASS: test_cases\q2\2-1b-vary-depth.test
*** PASS: test_cases\q2\2-2a-vary-depth.test
*** PASS: test_cases\q2\2-2b-vary-depth.test
*** PASS: test_cases\q2\2-3a-vary-depth.test
*** PASS: test_cases\q2\2-3b-vary-depth.test
*** PASS: test_cases\q2\2-4a-vary-depth.test
*** PASS: test_cases\q2\2-4b-vary-depth.test
*** PASS: test_cases\q2\2-one-ghost-3level.test
*** PASS: test_cases\q2\3-one-ghost-4level.test
*** PASS: test_cases\q2\4-two-ghosts-3level.test
```



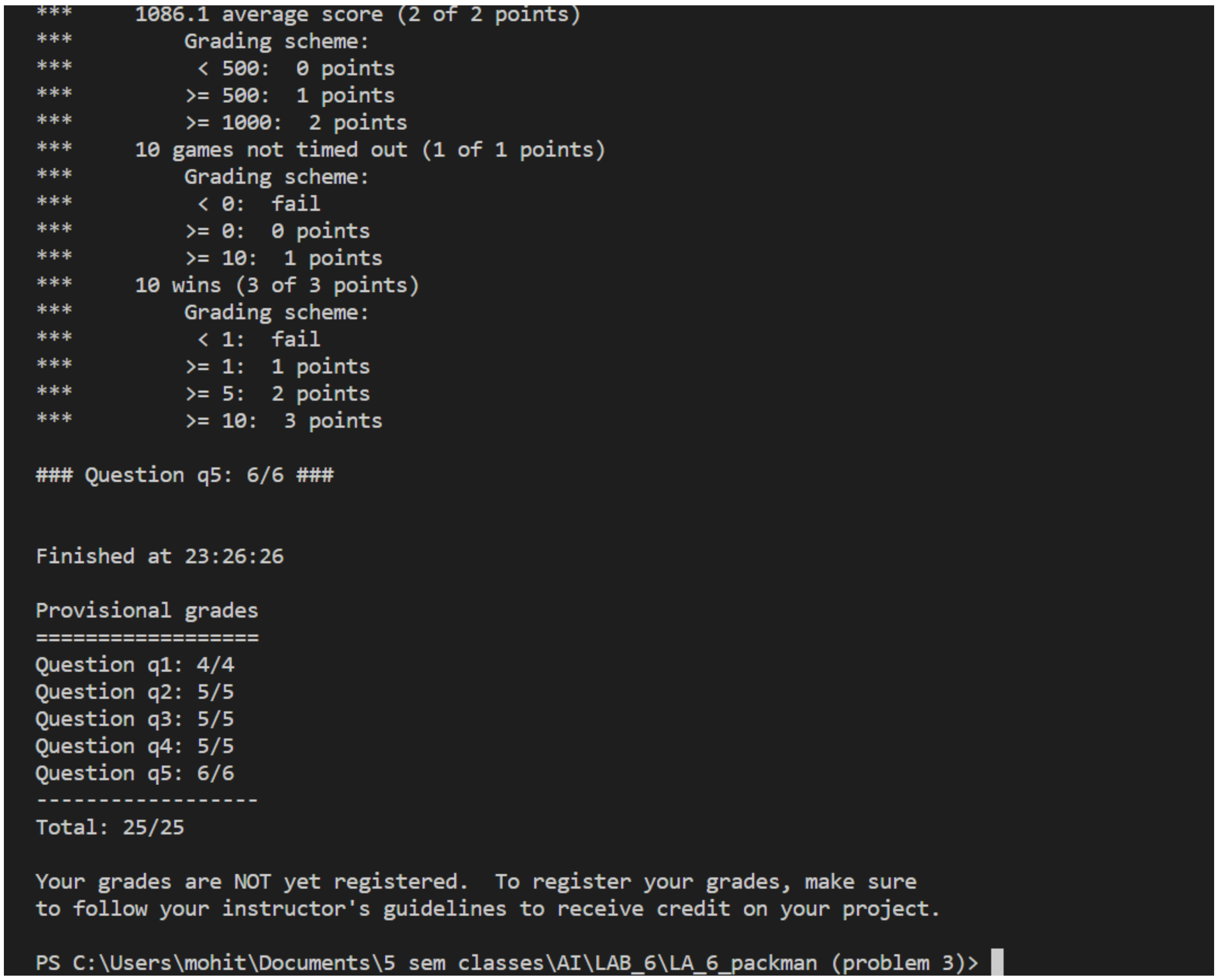
```
*** PASS: test_cases\q2\6-tied-root.test
*** PASS: test_cases\q2\7-1a-check-depth-one-ghost.test
*** PASS: test_cases\q2\7-1b-check-depth-one-ghost.test
*** PASS: test_cases\q2\7-1c-check-depth-one-ghost.test
*** PASS: test_cases\q2\7-2a-check-depth-two-ghosts.test
*** PASS: test_cases\q2\7-2b-check-depth-two-ghosts.test
*** PASS: test_cases\q2\7-2c-check-depth-two-ghosts.test
*** Running MinimaxAgent on smallClassic 1 time(s).
Pacman died! Score: 84
Average Score: 84.0
Scores:        84.0
Win Rate:      0/1 (0.00)
Record:        Loss
*** Finished running MinimaxAgent on smallClassic after 3 seconds.
*** Won 0 out of 1 games. Average score: 84.000000 ***
*** PASS: test_cases\q2\8-pacman-game.test

### Question q2: 5/5 ###


Question q3
===========

*** PASS: test_cases\q3\0-eval-function-lose-states-1.test
*** PASS: test_cases\q3\0-eval-function-lose-states-2.test
*** PASS: test_cases\q3\0-eval-function-win-states-1.test
*** PASS: test_cases\q3\0-eval-function-win-states-2.test
*** PASS: test_cases\q3\0-lecture-6-tree.test
*** PASS: test_cases\q3\0-small-tree.test
*** PASS: test_cases\q3\1-1-minmax.test
*** PASS: test_cases\q3\1-2-minmax.test
*** PASS: test_cases\q3\1-3-minmax.test
*** PASS: test_cases\q3\1-4-minmax.test
*** PASS: test_cases\q3\1-5-minmax.test
*** PASS: test_cases\q3\1-6-minmax.test
*** PASS: test_cases\q3\1-7-minmax.test
```



```
*** PASS: test_cases\q3\4-two-ghosts-3level.test
*** PASS: test_cases\q3\5-two-ghosts-4level.test
*** PASS: test_cases\q3\6-tied-root.test
*** PASS: test_cases\q3\7-1a-check-depth-one-ghost.test
*** PASS: test_cases\q3\7-1b-check-depth-one-ghost.test
*** PASS: test_cases\q3\7-1c-check-depth-one-ghost.test
*** PASS: test_cases\q3\7-2a-check-depth-two-ghosts.test
*** PASS: test_cases\q4\0-eval-function-win-states-1.test
*** PASS: test_cases\q4\0-eval-function-win-states-2.test
*** PASS: test_cases\q4\0-expectimax1.test
*** PASS: test_cases\q4\1-expectimax2.test
*** PASS: test_cases\q4\2-one-ghost-3level.test
*** PASS: test_cases\q4\3-one-ghost-4level.test
*** PASS: test_cases\q4\4-two-ghosts-3level.test
*** PASS: test_cases\q4\5-two-ghosts-4level.test
*** PASS: test_cases\q4\6-1a-check-depth-one-ghost.test
*** PASS: test_cases\q4\6-1b-check-depth-one-ghost.test
*** PASS: test_cases\q4\6-1c-check-depth-one-ghost.test
*** PASS: test_cases\q4\6-2a-check-depth-two-ghosts.test
*** PASS: test_cases\q4\6-2b-check-depth-two-ghosts.test
*** PASS: test_cases\q4\6-2c-check-depth-two-ghosts.test
*** Running ExpectimaxAgent on smallClassic 1 time(s).
Pacman died! Score: 84
Average Score: 84.0
Scores:        84.0
Win Rate:      0/1 (0.00)
Record:        Loss
*** Finished running ExpectimaxAgent on smallClassic after 4 seconds.
*** Won 0 out of 1 games. Average score: 84.000000 ***
*** PASS: test_cases\q4\7-pacman-game.test

### Question q4: 5/5 ###


Question q5
===========
```



```
*** Won 0 out of 1 games. Average score: 84.000000 ***
*** PASS: test_cases\q4\7-pacman-game.test

### Question q4: 5/5 ###


Question q5
===========

Pacman emerges victorious! Score: 1083
Pacman emerges victorious! Score: 1145
Pacman emerges victorious! Score: 1009
Pacman emerges victorious! Score: 1081
Pacman emerges victorious! Score: 994
Pacman emerges victorious! Score: 1097
Pacman emerges victorious! Score: 1088
Pacman emerges victorious! Score: 1007
Pacman emerges victorious! Score: 1112
Pacman emerges victorious! Score: 1245
Average Score: 1086.1
Scores:        1083.0, 1145.0, 1009.0, 1081.0, 994.0, 1097.0, 1088.0, 1007.0, 1112.0, 1245.0
Win Rate:      10/10 (1.00)
Record:        Win, Win, Win, Win, Win, Win, Win, Win, Win, Win
*** PASS: test_cases\q5\grade-agent.test (6 of 6 points)
***     1086.1 average score (2 of 2 points)
***         Grading scheme:
***          < 500:  0 points
***         >= 500:  1 points
***         >= 1000:  2 points
***     10 games not timed out (1 of 1 points)
***         Grading scheme:
***          < 0:  fail
***         >= 0:  0 points
***         >= 10:  1 points
***     10 wins (3 of 3 points)
***         Grading scheme:
```



```
***     1086.1 average score (2 of 2 points)
***         Grading scheme:
***          < 500:  0 points
***         >= 500:  1 points
***         >= 1000:  2 points
***     10 games not timed out (1 of 1 points)
***         Grading scheme:
***          < 0:  fail
***         >= 0:  0 points
***         >= 10:  1 points
***     10 wins (3 of 3 points)
***         Grading scheme:
***          < 1:  fail
***         >= 1:  1 points
***         >= 5:  2 points
***         >= 10:  3 points

### Question q5: 6/6 ###


Finished at 23:26:26

Provisional grades
==================
Question q1: 4/4
Question q2: 5/5
Question q3: 5/5
Question q4: 5/5
Question q5: 6/6
------------------
Total: 25/25

Your grades are NOT yet registered.  To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.

PS C:\Users\mohit\Documents\5 sem classes\AI\LAB_6\LA_6_packman (problem 3)>
```

Observations:
Here we got the score of all the test cases and the autograder ran successfully.

References :

https://www.youtube.com/watch?v=a6PiqvG8ivs
https://www.javatpoint.com/mini-max-algorithm-in-ai
https://www.javatpoint.com/ai-alpha-beta-pruning
https://www.geeksforgeeks.org/expectimax-algorithm-in-game-theory/