CSL 303 : Artificial Intelligence

TUTORIAL ASSIGNMENT 6

Adversarial Search

Date Assigned: 17^{th} September, 2021 **Date Submitted :** 26^{th} September, 2021

Submitted by:

Name: S. Bhavyesh

Roll Number: 191210045

Branch: Computer Science and Engineering

Semester: 5^{th}

Submitted to: Dr. Chandra Prakash

Department of Computer Science and Engineering

NATIONAL INSTITUTE OF TECHNOLOGY DELHI
A-7, Institutional Area, near Satyawadi Raja Harish Chandra Hospital, New Delhi, Delhi 110040

Contribution

Udit (**191210051**) and I(**191210045**) collaborated for this Tutorial assignment.

We independently did Q1 in Tutorial 6 (17/09/2021). After that we discussed the algorithms together for the remaining questions.

I did majority of Q2, and Udit added comments and made minor changes.

In Q3 (Pacman), Udit and I independently did Q1, I did Q2, Udit did Q3 but ran into a small problem so I resolved his bug, I did Q4 and Udit did Q5.

PART A : Adversarial Problem [15 Marks]

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

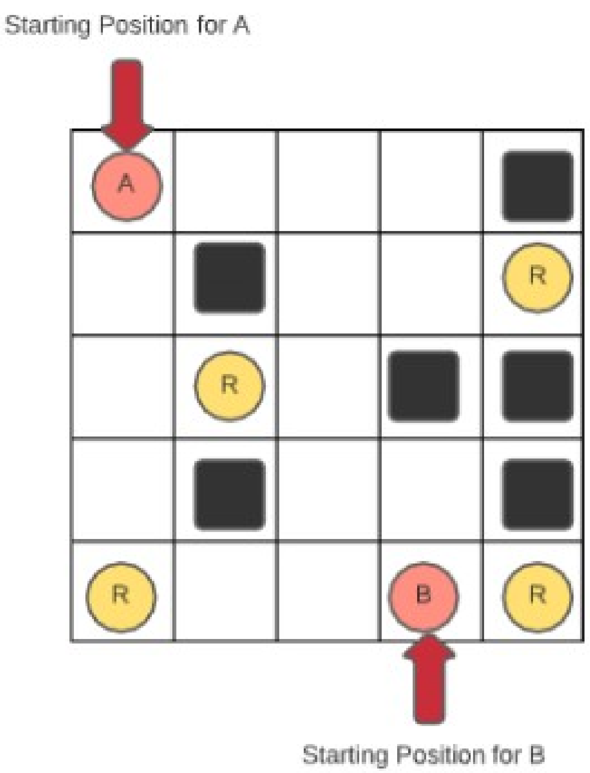
Inputs: Consider the maze given in the figure below. The walled tiles are marked in black and your agent A and B cannot move to or through those positions.

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".



CODE:

In [6]:
```
# check if node is valid: within bounds
# and not already visited

def valid(node,visited):
    if(node[0]<=5 and node[1]<=5 and node[0]>=1 and node[1]>=1
       and node not in visited and node not in obstacles):
        return True
    return False
```

In [7]:
```
# Breadth-First Search

def bfs(start,goal,x):

    # initialize variables
    queue=[start]
    visited=[]
    cost=0

    while True:
        # remove node from fringe and add it to visited list
        curr_node=queue.pop(0)
        visited.append(curr_node)

        # extract co-ordinates for future use
        i,j=curr_node

        # goal found!
        if(curr_node==goal): return cost

        # entertain all possibilities (up,down,left,right)
        else:
            nbrs=[(i-1,j),(i,j-1),(i,j+1),(i+1,j)]
            random.seed(x)
            random.shuffle(nbrs)

            for i in range(4):
                if(valid(nbrs[i],visited)):
                    queue.append(nbrs[i])
                    visited.append(nbrs[i])
        cost+=1
```

In [8]:
```
import random
import numpy as np

# below code snippet is for showing the
# initial state of the maze diagrammatically
maze=np.zeros(25)
maze[4]=maze[6]=maze[13]=maze[14]=maze[16]=maze[19]=1
maze[0]=2
maze[23]=2
maze[9]=maze[11]=maze[20]=maze[24]=3

maze=maze.reshape((5,5))
print(maze)
```

In [9]:

```
#========================================#
#               For Agent A
#========================================#



costA=[]

for i in range(10):

    # tuples defining the respective entities
    # start state, goal state, and obstacles

    start=(1,1)
    goals=[(3,2),(5,1),(2,5),(5,5)]
    obstacles=[(1,5),(2,2),(3,4),(3,5),(4,2),(4,5)]

    cost=0 # total cost to find all goals in the maze

    while(len(goals)>0):
        # find the closest goal using Manhattan Distance
        min_ind=np.argmin(np.sum(np.abs(np.array(goals)-start),axis=1))

        # use bfs on the current state and nearest goal found
        cost+=bfs(start,goals[min_ind],i)

        # repeat the problem with goal state as new start state
        start=goals[min_ind]
        goals.pop(min_ind)

    costA.append(cost)
```

In [10]:

```
#========================================#
#               For Agent B
#========================================#


costB=[]

for i in range(10):

    # tuples defining the respective entities
    # start state, goal state, and obstacles
    start=(5,4)
    goals=[(3,2),(5,1),(2,5),(5,5)]
    obstacles=[(1,5),(2,2),(3,4),(3,5),(4,2),(4,5)]

    cost=0 # total cost to find all goals in the maze

    while(len(goals)>0):
        # find the closest goal using Manhattan Distance
        min_ind=np.argmin(np.sum(np.abs(np.array(goals)-start),axis=1))

        # use bfs on the current state and nearest goal found
        cost+=bfs(start,goals[min_ind],i)

        # repeat the problem with goal state as new start state
        start=goals[min_ind]
        goals.pop(min_ind)

    costB.append(cost)
```

```python
In [11]: # display round-wise scores of each player
print(costA)
print(costB)

# convert to NumPy array for easy comparison
costA=np.array(costA)
costB=np.array(costB)


a_win=np.count_nonzero(costA<costB) # number of games A has won
b_win=np.count_nonzero(costA>costB) # number of games B has won

# check who is the winner
if(a_win>b_win):
    print("A is the winner with " ,a_win," wins")
elif(a_win<b_win):
    print("B is the winner with " ,b_win," wins")
else:
    print("Draw, both have" ,a_win,"wins")
```

OUTPUTS:

```
[[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.]]
```

```
[42, 45, 38, 45, 42, 42, 41, 42, 47, 43]
[34, 37, 30, 37, 35, 34, 34, 34, 38, 34]
B is the winner with  10  wins
```

OBSERVATION/COMMENTS

1. For the purpose of randomization, I am changing the order in which the neighbouring nodes are examined. I am setting a seed value depending on round number, that is common to both players A and B.

2. In spite of the randomization, B is winning rounds with a huge margin due to its proximity to the goals.

3. In this case since actions of agents don't affect the other, we get to see such large margins. Maybe if the agents work in the same maze where the actions of one will affect another, perhaps then the competition will be tight.

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 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.

CODE:

```
In [1]: # check if node is valid: within bounds
        # and not already visited

        def valid(node):
            if(node[0]<=5 and node[1]<=5 and node[0]>=1 and node[1]>=1
            and node not in obstacles):
                return True
            return False
```

```
In [2]: import random
        import numpy as np
        import math

        # below code snippet is for
        # showing the initial state of
        # the maze diagrammatically
        maze=np.zeros(25)
        maze[4]=maze[6]=maze[13]=maze[14]=maze[16]=maze[19]=1
        maze[0]=2
        maze[23]=2
        maze[9]=maze[11]=maze[20]=maze[24]=3

        maze=maze.reshape((5,5))
        print(maze)

        [[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.]]
```

```
In [3]: # evaluation function for leaves of game tree
        # sum of inverse Manhattan and inverse Euclidean distances
        # from each of the remaining goals, bonus of +3 if agent is on goal

        def eval_func(currState,agent,goals):
            i,j=currState[0][agent]
            k,l=currState[0][1-agent]
            score=0
            for goal in currState[1]:
                a,b=goal
                score+=1/(abs(a-i)+abs(b-j))+1/(math.sqrt((a-i)**2+(b-j)**2))
            if(currState[0][agent] in goals):
                score+=3
            return score
```

```
In [4]: # max_part of minimax algorithm

        def max_part(currState,agent):

            temp=[x[:] for x in currState]
            i,j=temp[0][agent]

            #collect neighbours
            nbrs=[(i-1,j),(i,j-1),(i+1,j),(i,j+1)]
            tru_nbrs=[]

            #collect valid neighbours
            for i in nbrs:
                if(valid(i)): tru_nbrs.append(i)
            val=-99999
            scores=[]
            for j in tru_nbrs:
                new_state=[x[:] for x in currState]
                new_state[0][agent]=j

                for i in range(len(new_state[1])):
                    if(new_state[0][agent]==new_state[1][i]):
                        new_state[1].pop(i)
                        break

                #min moves
                scores.append(min_part(new_state,1-agent))

            #return max from min moves
            return tru_nbrs[scores.index(max(scores))]
```

In [5]:

```python
# min part of minimax algo
# since depth is 1, the children of
# min component will be leaves,
# a.k.a the heuristic values of positions

def min_part(currState,agent):

    temp=[x[:] for x in currState]
    i,j=temp[0][agent]

    #collect neighbours
    nbrs=[(i-1,j),(i,j-1),(i+1,j),(i,j+1)]
    tru_nbrs=[]

     #collect valid neighbours
    for i in nbrs:
        if(valid(i)): tru_nbrs.append(i)

    val=99999
    scores=[]
    for j in tru_nbrs:

        new_state=[x[:] for x in currState]
        new_state[0][agent]=j

        for i in range(len(new_state[1])):
            if(new_state[0][agent]==new_state[1][i]):
                new_state[1].pop(i)
                break
        #max moves
        scores.append(eval_func(new_state,1-agent,goals))

    #return min from max moves
    return min(scores)
```

In [6]:

```python
# definition of problem statement
# and various components

goals=[(3,2),(5,1),(2,5),(5,5)]
gameState=[[(1,1),(5,4)],[(3,2),(5,1),(2,5),(5,5)]]
global obstacles
obstacles=[(1,5),(2,2),(3,4),(3,5),(4,2),(4,5)]

global obstacles

player_id={ 0:'A', 1:'B'}

score=[0,0]

while(True):

    for a in range(2):
        gameState[0][a]=max_part(gameState,a)
        print("Player ", player_id[a],"moves to: ", gameState[0][a],'\t', end='')
        for i in range(len(gameState[1])):
            if(gameState[0][a]==gameState[1][i]):
                gameState[1].pop(i)
                goals.pop(i)
                score[a]+=10
                break
        else: score[a]-=1
        if(len(gameState[1])==0): break

    if(len(gameState[1])==0): break
    print("Current scores: A =",score[0],", B =",score[1])

print("\nFinal scores: A =",score[0],", B =",score[1])

if(score[0]<score[1]): print("B is the winner")
elif(score[0]>score[1]): print("A is the winner")
else:  print("Drawn game")
```

OUTPUTS:

```
Player  A moves to:  (2, 1)     Player  B moves to:  (5, 5)     Current scores: A = -1 , B = 10
Player  A moves to:  (3, 1)     Player  B moves to:  (5, 4)     Current scores: A = -2 , B = 9
Player  A moves to:  (3, 2)     Player  B moves to:  (5, 3)     Current scores: A = 8 , B = 8
Player  A moves to:  (3, 1)     Player  B moves to:  (5, 2)     Current scores: A = 7 , B = 7
Player  A moves to:  (3, 2)     Player  B moves to:  (5, 1)     Current scores: A = 6 , B = 17
Player  A moves to:  (3, 3)     Player  B moves to:  (5, 2)     Current scores: A = 5 , B = 16
Player  A moves to:  (2, 3)     Player  B moves to:  (5, 3)     Current scores: A = 4 , B = 15
Player  A moves to:  (2, 4)     Player  B moves to:  (4, 3)     Current scores: A = 3 , B = 14
Player  A moves to:  (2, 5)
Final scores: A = 13 , B = 14
B is the winner
```

OBSERVATION/COMMENTS

1. In contrast to the previous problem, we see that by adding certain scoring constraints and making the agents play in the same maze, we can come up with a closer competition than before.

2. If one does not want to make the entire game tree, then one option is to limit the depth and use an accurate heuristic that is proportional to the utility values of positions.

3. The scoring criteria is as follows:

- move that results in no reward = -1 points
- move that results in reward = +10 points

PART B : Exploratory Problem [25 Marks]

Question 1 (4 points) : Reflex Agent

```
"*** YOUR CODE HERE ***"

score=0
for i in range(len(newFood.asList())):
    temp=manhattanDistance(newPos,newFood.asList()[i])
    if(temp==0):
        temp=1
    score+=1/temp

for i in range(len(successorGameState.getGhostPositions())):
    temp=manhattanDistance(newPos,successorGameState.getGhostPositions()[i])
    if(temp==0):
        temp=0.5
    score-=1/temp

return successorGameState.getScore()+score
```

```
bridges@bridges-CF-C2AHCCZC7: ~/Downloads/T6_multiagent/T

Question q1
===========

Pacman emerges victorious! Score: 1228
Pacman emerges victorious! Score: 1209
Pacman emerges victorious! Score: 1209
Pacman emerges victorious! Score: 1215
Pacman emerges victorious! Score: 1228
Pacman emerges victorious! Score: 1231
Pacman emerges victorious! Score: 1231
Pacman emerges victorious! Score: 1224
Pacman emerges victorious! Score: 1227
Pacman emerges victorious! Score: 1231
Average Score: 1223.3
Scores:        1228.0, 1209.0, 1209.0, 1215.0, 1228.0, 1231.0, 1231.0, 1224.0, 1227.0, 1231.0
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)
***     1223.3 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 ###
```

OBSERVATION/COMMENTS

I am simply adding to the successorGameState.getScore(), as a way of guiding the Pacman, by rewarding food positions, and penalizing ghost positions.

Question 2 (5 points): Minimax

```python
"*** YOUR CODE HERE ***"
def MiniMax(currState, currDepth, currAgent):

    # if PacMan wins or dies
    if(currState.isWin() or currState.isLose()):
        return self.evaluationFunction(currState)

    # if agent is PacMan
    if(currAgent==0):

        # max depth reached->evaluate position
        if(currDepth==self.depth):
            return self.evaluationFunction(currState)

        # check for its moves and implement the min-part
        else:
            val=-99999
            for action in currState.getLegalActions(currAgent):
                v = MiniMax(currState.generateSuccessor(0,action), currDepth, 1)
                val = max(v,val)
            return val
```

```python
    # if agent is a ghost
    elif(currAgent!=0):

        # if this was the last ghost,
        # return to Pacman, and declare the depth complete
        # by traversing the next level
        val=99999
        if(currAgent == currState.getNumAgents()-1):

            for action in currState.getLegalActions(currAgent):
                v = MiniMax(currState.generateSuccessor(currAgent,action), currDepth+1, 0)
                val = min(v,val)
            return val

        # else do the multiagent min-part
        else:

            for action in currState.getLegalActions(currAgent):
                v = MiniMax(currState.generateSuccessor(currAgent,action), currDepth, currAgent+1)
                val = min(v,val)
            return val


# record of scores and the actions that caused it
scores=[]
actions=[]

# check all possible actions from current state
for action in gameState.getLegalActions(0):

    scores.append(MiniMax(gameState.generateSuccessor(0,action), 0, 1))
    actions.append(action)

# return action that predicts max utility
return actions[scores.index(max(scores))]
```

```
*** 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/q2/5-two-ghosts-4level.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 1 seconds.
*** Won 0 out of 1 games. Average score: 84.000000 ***
*** PASS: test_cases/q2/8-pacman-game.test

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

OBSERVATION/COMMENTS

Here covering a unit of depth means taking into account one move of Pacman and one move each of the ghosts in the maze. In fact, we faced some difficulty at first trying to implement this multi-agent minimax algorithm.

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

```python
def alphaBeta(currState, currDepth, currAgent,alpha,beta):

    # if PacMan wins or dies
    if(currState.isWin() or currState.isLose()):
        return self.evaluationFunction(currState)

    # if agent is PacMan
    if(currAgent==0):

        # max depth reached->evaluate position
        if(currDepth==self.depth):
            return self.evaluationFunction(currState)

        # check for its moves and implement the min-part
        else:
            val=MIN
            for action in currState.getLegalActions(currAgent):
                v = alphaBeta(currState.generateSuccessor(0,action), currDepth, 1,alpha,beta)
                val = max(v,val)

                # follow the algo as mentioned in the pdf file
                if val>beta:
                    return val
                alpha = max(alpha,val)
            return val
```

```python
    # if agent is a ghost
    elif(currAgent!=0):

        # if this was the last ghost,
        # return to Pacman, and declare the depth complete
        # by traversing the next level
        val=MAX
        if(currAgent == currState.getNumAgents()-1):

            for action in currState.getLegalActions(currAgent):
                v = alphaBeta(currState.generateSuccessor(currAgent,action), currDepth+1, 0,alpha,beta)
                val = min(v,val)

                # follow the algo as mentioned in the pdf file
                if val<alpha:
                    return val
                beta = min(beta,val)
            return val


        # else do the multiagent min-part
        else:

            for action in currState.getLegalActions(currAgent):
                v = alphaBeta(currState.generateSuccessor(currAgent,action), currDepth, currAgent+1,alpha,beta)
                val = min(v,val)

                # follow the algo as mentioned in the pdf file
                if val<alpha:
                    break
                beta = min(beta,val)
            return val
```

```python
# record of scores and the actions that caused it
scores=[]
actions=[]

# check all possible actions from current state
alpha, beta = -99999, 99999
for action in gameState.getLegalActions(0):

    val = alphaBeta(gameState.generateSuccessor(0,action), 0, 1,alpha,beta)
    scores.append(val)

    # update alpha for topmost node aka PacMan
    alpha = max(alpha, val)
    actions.append(action)

# return action that predicts max utility
return actions[scores.index(max(scores))]
```

```
*** 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/q3/1-8-minmax.test
*** PASS: test_cases/q3/2-1a-vary-depth.test
*** PASS: test_cases/q3/2-1b-vary-depth.test
*** PASS: test_cases/q3/2-2a-vary-depth.test
*** PASS: test_cases/q3/2-2b-vary-depth.test
*** PASS: test_cases/q3/2-3a-vary-depth.test
*** PASS: test_cases/q3/2-3b-vary-depth.test
*** PASS: test_cases/q3/2-4a-vary-depth.test
*** PASS: test_cases/q3/2-4b-vary-depth.test
*** PASS: test_cases/q3/2-one-ghost-3level.test
*** PASS: test_cases/q3/3-one-ghost-4level.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/q3/7-2b-check-depth-two-ghosts.test
*** PASS: test_cases/q3/7-2c-check-depth-two-ghosts.test
*** Running AlphaBetaAgent 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 AlphaBetaAgent on smallClassic after 1 seconds.
*** Won 0 out of 1 games. Average score: 84.000000 ***
*** PASS: test_cases/q3/8-pacman-game.test

### Question q3: 5/5 ###
```

OBSERVATION/COMMENTS

1. Alpha-beta pruning now helps in reducing the number of traversals in the game tree.

2. Earlier, there was a mistake in the outer loop where we keep the scores and actions. We were not accouting for the alpha beta values of the topmost node(PacMan).

Question 4 (5 points): Expectimax

```python
"""
"*** YOUR CODE HERE ***"
def ExpectiMax(currState, currDepth, currAgent):

    # if PacMan wins or dies
    if(currState.isWin() or currState.isLose()):
        return self.evaluationFunction(currState)

    # if agent is PacMan
    if(currAgent==0):

        # max depth reached->evaluate position
        if(currDepth==self.depth):
            return self.evaluationFunction(currState)

        # check for its moves and implement the expecti-part
        else:
            val=-99999
            for action in currState.getLegalActions(currAgent):
                v = ExpectiMax(currState.generateSuccessor(0,action), currDepth, 1)
                val = max(v,val)
            return val
```

```python
    # if agent is a ghost
    elif(currAgent!=0):

        # if this was the last ghost,
        # return to Pacman, and declare the depth complete
        # by traversing the next level
        val=[]
        if(currAgent == currState.getNumAgents()-1):

            for action in currState.getLegalActions(currAgent):
                val.append(ExpectiMax(currState.generateSuccessor(currAgent,action), currDepth+1, 0))

        # else do the multiagent min-part
        else:

            for action in currState.getLegalActions(currAgent):
                val.append(ExpectiMax(currState.generateSuccessor(currAgent,action), currDepth, currAgent+1))

        return sum(val)/len(val)  # average of all moves, expected value(uniform distribution)

# record of scores and the actions that caused it
scores=[]
actions=[]

# check all possible actions from current state
for action in gameState.getLegalActions(0):

    scores.append(ExpectiMax(gameState.generateSuccessor(0,action), 0, 1))
    actions.append(action)

# return action that predicts max utility
return actions[scores.index(max(scores))]
```

```
Question q4
===========

*** PASS: test_cases/q4/0-eval-function-lose-states-1.test
*** PASS: test_cases/q4/0-eval-function-lose-states-2.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 1 seconds.
*** Won 0 out of 1 games. Average score: 84.000000 ***
*** PASS: test_cases/q4/7-pacman-game.test

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

OBSERVATION/COMMENTS

Expectimax is almost similar to normal minimax except here we take the probabilistic score, instead of the min score, in minimax.

Question 5 (6 points): Evaluation Function

```python
    pos = currentGameState.getPacmanPosition()
    ghost = currentGameState.getGhostStates()
    foods = currentGameState.getFood()
    food = foods.asList()
    ghosts = currentGameState.getGhostStates()
    scaredGhosts = []
    activeGhosts = []
    guide = 0


    for ghost in ghosts:
        if ghost.scaredTimer: scaredGhosts.append(ghost)
        else: activeGhosts.append(ghost)

    for item in food:
        temp = manhattanDistance(pos,item)

        if temp ==2: temp = 0.65
        if temp ==1: temp = 0.3
        if temp ==0: temp = 0.01
        guide += 1/temp

    for ghost in activeGhosts:
        temp = manhattanDistance(ghost.getPosition(),pos)

        if temp ==2: temp = 0.3
        if temp ==1: temp = 0.30
        if temp ==0: temp = 0.01
        guide -= 1/temp

    for item in scaredGhosts:
        temp = manhattanDistance(pos,item.getPosition())
        if temp <=10: temp = 0.01
        guide += 1/temp


    return currentGameState.getScore() + guide
```

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

Pacman emerges victorious! Score: 975
Pacman emerges victorious! Score: 1273
Pacman emerges victorious! Score: 1370
Pacman emerges victorious! Score: 1362
Pacman emerges victorious! Score: 1356
Pacman emerges victorious! Score: 1367
Pacman emerges victorious! Score: 1348
Pacman emerges victorious! Score: 1361
Pacman emerges victorious! Score: 1350
Pacman emerges victorious! Score: 1157
Average Score: 1291.9
Scores:        975.0, 1273.0, 1370.0, 1362.0, 1356.0, 1367.0, 1348.0, 1361.0, 1350.0, 1157.0
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)
***     1291.9 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 ###
```

OBSERVATION/COMMENTS

Eval-function is similar to the one in Q1 done by Udit, instead we now consider the scared timings of ghosts, adding another layer of precision to our evaluation.

AUTOGRADER SCORES

```
Finished at 13:02:32

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.
```