Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Computer Science and Engineering (Data Science)

Subject: Artificial Intelligence (DJ19DSC502)

AY: 2023-24

Jhanvi Parekh

60009210033

D11

Experiment 7 (Adversarial Search)

Aim: Implement Tic-Tac-Toe game using Minimax algorithm.

Theory:

- Mini-max algorithm is a recursive or backtracking algorithm which is used in decision-making and game theory. It provides an optimal move for the player assuming that opponent is also playing optimally.
- Mini-Max algorithm uses recursion to search through the game-tree.
- Min-Max algorithm is mostly used for game playing in AI. Such as Chess, Checkers, tic-tac-toe, go, and various tow-players game. This Algorithm computes the minimax decision for the current state.
- In this algorithm two players play the game, one is called MAX and other is called MIN.
- Both the players fight it as the opponent player gets the minimum benefit while they get the maximum benefit.
- Both Players of the game are opponent of each other, where MAX will select the maximized value and MIN will select the minimized value.

- The minimax algorithm performs a depth-first search algorithm for the exploration of the complete game tree.
- The minimax algorithm proceeds all the way down to the terminal node of the tree, then backtrack the tree as the recursion.

Psudo Code

```
function minimax(node, depth, maximizingPlayer) is
    if depth ==0 or node is a terminal node then
        return static evaluation of node
    if MaximizingPlayer then      // for Maximizer Player
        maxEva= -infinity
        for each child of node do
            eva= minimax(child, depth-1, false)
            maxEva= max(maxEva,eva)       //gives Maximum of the values
        return maxEva
    else                          // for Minimizer player
        minEva= +infinity
        for each child of node do
            eva= minimax(child, depth-1, true)
            minEva= min(minEva, eva)      //gives minimum of the values
        return minEva
```

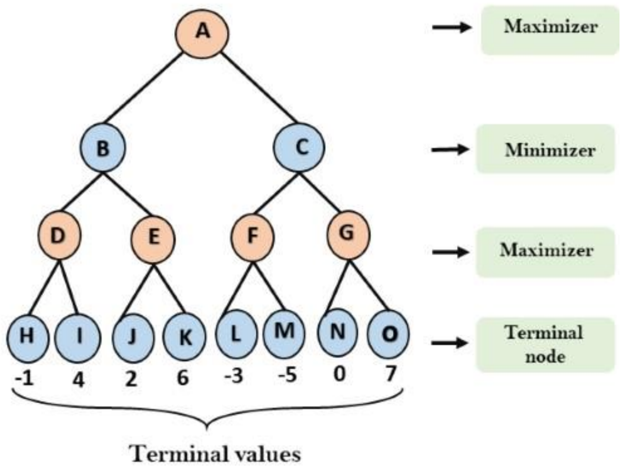
Initial call:

Minimax(node, 3, true)

Working of Min-Max Algorithm: ○ The working of the minimax algorithm can be easily described using an example. Below we have taken an example of game-tree which is representing the two-player game.

- In this example, there are two players one is called Maximizer and other is called Minimizer.
- Maximizer will try to get the Maximum possible score, and Minimizer will try to get the minimum possible score.
- This algorithm applies DFS, so in this game-tree, we have to go all the way through the leaves to reach the terminal nodes. ○ At the terminal node, the terminal values are given so we will compare those value and backtrack the tree until the initial state occurs. Following are the main steps involved in solving the two-player game tree:

Step-1: In the first step, the algorithm generates the entire game-tree and apply the utility function to get the utility values for the terminal states. In the below tree diagram, let's take A is the initial state of the tree. Suppose maximizer takes first turn which has worst-case initial value =- infinity, and minimizer will take next turn which has worst-case initial value = +infinity.

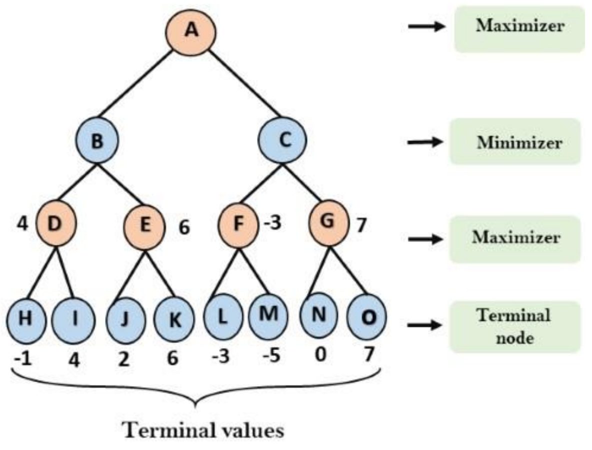


Step 2: Now, first we find the utilities value for the Maximizer, its initial value is -∞, so we will compare each value in terminal state with initial value of Maximizer and determines the higher nodes values. It will find the maximum among the all.

- For node D max(-1,- -∞) => max(-1,4)= 4
  - For Node E max(2, -∞) => max(2, 6)= 6
  - For Node F max(-3, -∞) => max(-3,-5)

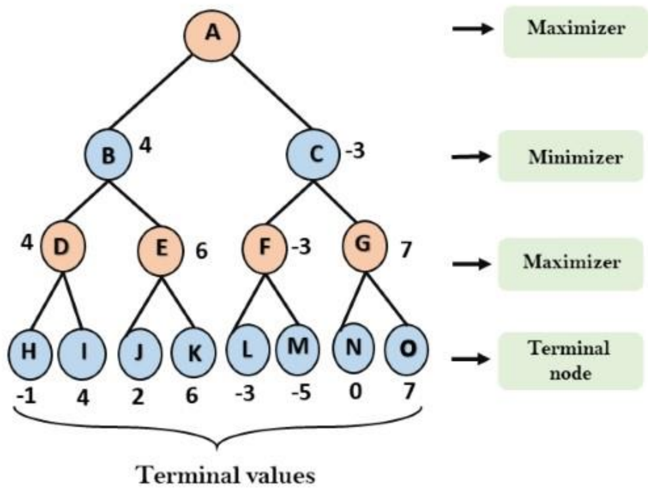
  = -3 ○ For node G max(0, -∞) = max(0,

  7) = 7



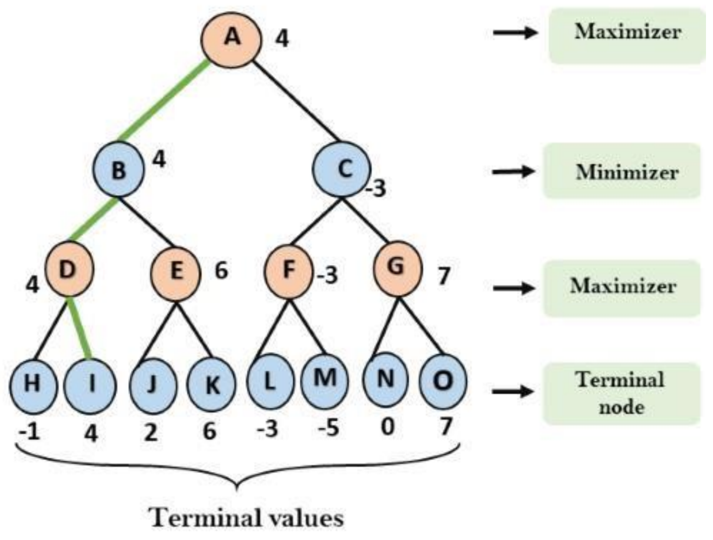
Step 3: In the next step, it's a turn for minimizer, so it will compare all nodes value with +∞, and will find the 3rd layer node values.

- For node B= min(4,6) = 4
- For node C= min (-3, 7) = -3



Step 4: Now it's a turn for Maximizer, and it will again choose the maximum of all nodes value and find the maximum value for the root node. In this game tree, there are only 4 layers, hence we reach immediately to the root node, but in real games, there will be more than 4 layers.

- For node A max(4, -3)= 4



Department of Computer Science and Engineering (Data Science)

Lab Assignment to do:

Implement a two player Tic-Tac-Toe game using Minimax algorithm for Game Playing. One player will be computer itself.

LINK:
https://colab.research.google.com/drive/1GkImd8l6--udLnSrAafmv82FU1sqIBIC?usp=sharing

LINK:

https://colab.research.google.com/drive/1Razpf1y8kmL8FfH0uDh8xJQ_BjBu3XaZ?usp=sharing

```
board=[]
for i in range(9):
  board.append(input("Enter X ,O or . (for blank)"))
print(board)
```

```
Enter X ,O or . (for blank)x
Enter X ,O or . (for blank).
Enter X ,O or . (for blank).
Enter X ,O or . (for blank)o
Enter X ,O or . (for blank)o
Enter X ,O or . (for blank)x
Enter X ,O or . (for blank).
Enter X ,O or . (for blank)o
Enter X ,O or . (for blank)x
['x', '.', '.', 'o', 'o', 'x', '.', 'o', 'x']
```

```
#board=['x', 'o', 'o', '.', 'o', '.', '.', 'x', 'x']
def print_board(board):
    for i in range(0, 9, 3):
        print(" ".join(board[i:i+3]))
print_board(board)
```

```
x . .
o o x
. o x
```

```
def check(borad):
    if (board[0] == board[1] == board[2] != "-") or  (board[3] == board[4] == board[5] != "-") or \
       (board[6] == board[7] == board[8] != "-") or  (board[0] == board[3] == board[6] != "-") or \
       (board[1] == board[4] == board[7] != "-") or  (board[2] == board[5] == board[8] != "-") or \
       (board[0] == board[4] == board[8] != "-") or  (board[2] == board[4] == board[6] != "-"):
     print("win")
     return 1
    else :
     print("loose")
     return -1
```

```
def check_empty(board):
    return "."  in board
check_empty(board)
```

```
True
```

```
def movegen(board, player):
    moves = []
    for i in range(9):
        if board[i] == ".":
            new_board = board.copy()
            new_board[i] = player
            moves.append(new_board)
    return moves

possible_moves = movegen(board, "x")

for move in possible_moves:
    for i in range(0, 9, 3):
        print(" ".join(move[i:i+3]))
    print()
```

```
x x .
o o x
. o x

x . x
o o x
. o x

x . .
o o x
x o x
```

45

```
for board in possible_moves:
  for i in range(0, 9, 3):
        print(" ".join(board[i:i+3]))
  check(board)
  print()
```

```
x x .
o o x
. o x
loose

x . x
o o x
. o x
win

x . .
o o x
x o x
loose
```

MINI-MAX ALGORITHM ON TIC TAC TOE:

```
#Draw the board's current state every time the user turn arrives.
def ConstBoard(board):
    print("Current State Of Board : \n\n");
    for i in range (0,9):
        if((i>0) and (i%3)==0):
            print("\n");
        if(board[i]==0):
            print("- ",end=" ");
        if (board[i]==1):
            print("O ",end=" ");
        if(board[i]==-1):
            print("X ",end=" ");
    print("\n\n");


#Analyses the game.
def analyzeboard(board):
    cb=[[0,1,2],[3,4,5],[6,7,8],[0,3,6],[1,4,7],[2,5,8],[0,4,8],[2,4,6]];
    for i in range(0,8):
        if(board[cb[i][0]] != 0 and #checking that the cell is occupied
           board[cb[i][0]] == board[cb[i][1]] and
           board[cb[i][0]] == board[cb[i][2]]):
            return board[cb[i][2]];
    return 0;


#This function takes the user move as input and make the required changes on the bo
def User1Turn(board):
    pos=int(input("Enter X's position from [1...9]: "));
    if(board[pos-1]!=0):
        print("Cell is already occipied. Try again.");
        exit(0) ;
    board[pos-1]=-1;


def minimax(board, player):
    x = analyzeboard(board)
    if x != 0:
        return x * player
    pos = -1
    value = -2
    for i in range(0, 9):
        if board[i] == 0:
            board[i] = player
            score = -minimax(board, player * -1)  # updating the possible score
            if score > value:
                value = score
                pos = i
            board[i] = 0  # if score is not greater than value, don't update on that position
    if pos == -1:
        return 0

    return value
```

```
#Make the computer's move using minmax algorithm.
def CompTurn(board):
    pos=-1;
    value=-2;
    for i in range(0,9):
        if(board[i]==0):
            board[i]=1;
            score=-minimax(board, -1);
            board[i]=0;
            if(score>value):
                value=score;
                pos=i;
    board[pos]=1;


def main():
    #The broad is considered in the form of a single dimentional array.
    board=[0,0,0,0,0,0,0,0,0];
```

```
player = 1;
for i in range (0,9):
    if(analyzeboard(board)!=0):
        break;
    if((i+player)%2==0):
        CompTurn(board);
    else:
        ConstBoard(board);
        User1Turn(board);
x=analyzeboard(board);
if(x==0):
     ConstBoard(board);
     print("Draw")
if(x==-1):
     ConstBoard(board);
     print("You win. Computer loses")
if(x==1):
     ConstBoard(board);
     print("You lose. Computer wins")
```

```
main()
```

```
Current State Of Board : 


-  -  -  

-  -  -  

-  -  -  


Enter X's position from [1...9]: 1
Current State Of Board : 


X  -  -  

-  O  -  

-  -  -  


Enter X's position from [1...9]: 6
Current State Of Board : 


X  O  -  

-  O  X  

-  -  -  


Enter X's position from [1...9]: 9
Current State Of Board : 


X  O  O  

-  O  X  

-  -  X  
```