Name: Tejas Redkar
PRN: 1032210937
Panel-C
Roll No: PC-44

AIES Assignment - 2

* **Aim:** Solve Tic-Tac-Toe using Minimax algorithm

* **Objective:** To study & implement Minimax algorithm for Tic-Tac-Toe

* **Theory:**

- Adverserial Search: It is a method applied to a situation where you are planning while another actor prepares against you. It is used in AI to model a competition between two individuals. Adversarial search is often used in two-person games such as chess, tic-tac-toe, Go, etc. In these games, the players can see the moves of the opponents.

- Tic-Tac-Toe simply involves playing the game strategically to ensure a win or draw.
  Steps for tic-tac-toe
1) Understand the rules of Tic-tac-Toe
2) Focus on making winning moves when possible
3) If winning isn't possible, block your opponent from winning
4) Prioritize center & corner positions for your moves.

5) If you can't win or block, aim for a draw by preventing your opponent from winning.
6) Keep adapting your strategy based on the game's progress.

- Data structures & other details about Minimax algo excluding algorithm.

1) Game Tree : Represent the game state as a tree structure, where nodes are game positions & edges are legal moves.
2) Nodes : Each Node contains current game board state player's turn & level.
3) Evaluation / Heuristic function : Used to estimate the desirability of a game state if its not terminal state. It assigns numerical value to position
4) Alpha-beta pruning
   - α = Maximizing player score
   - β = Minimizing player score
5) Depth limit : To prevent the algorithm from exploring the entire tree.

* Minimax Algorithm

Function Minimax - Decision (state) returns an action
    $v \leftarrow$ Max-value (state)
    return the action in successors (state) with value
    v

classmate
Date :
Page :

function Max-value (state) returns a utility value
if Terminal-test (state) then return Utility (state)
$v \leftarrow \infty$
for a, s in successors (state) do
$v \leftarrow$ Max (v, min-value (s))
return v

function Min-value (state) returns a utility value
if Terminal-Test ~~s~~ (stable) then return utility (stable)
$v \leftarrow \infty$
for a, s in successors (state) do
$v \leftarrow$ Min (v, Max-value (s))
return v

* FAQ's

Q1) Compare informed search & adversarial search

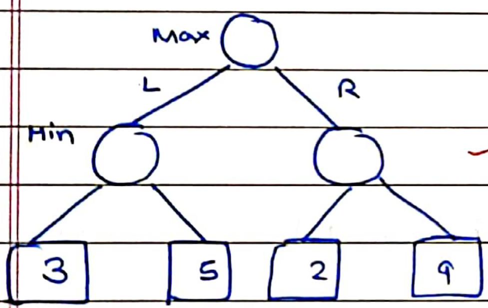
Ans

Informed Search	Adversarial Search
1) Uses Heuristic & domain specific search	Designed for competitive scenarios, like games
2) Aims to find solutions efficiently	Considers opponents moves
3) Applicable in various problem solving domains	Focus on finding optimal strategies
4) Can guarantee optimally with admissible & consistent heuristic	May not guarantee absolute best outcome due to game complexity
5) Ex: A*, Greedy, BFS	Eg: Minimax with Alpha-beta pruning, MCTS, etc

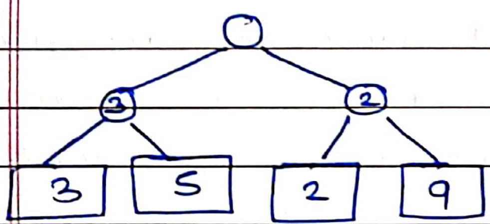
Q2 Explain Minimax algorithm with an example

Ans Minimax is a kind of backtracking algorithm that is used in decision making & game theory to find the optimal move for a player, assuming that your opponent also plays optimally. It is widely used in two player turn-based games such as tic-tac-toe, go, etc.

e.g. Consider a game which 4 final states & paths to reach final state. Assume you are maximizing player & you get 1st chance to move, then which move you would make a maximizing player.



Max goes left : It is minimizers turn
It will choose 3
Max goes right : again minimizes turn it will choose 2



After being Maximizer it will chooses 3 from (3,2).

The tree shows possible scores when maximizer makes left & right

Q3) Explain alpha-Beta pruning

classmate
Date :
Page :

Ans - It is modified version of Minimax algorithm.
- Alpha beta pruning can be applied at any depth of tree, & it not only prune the tree leaves but also entire sub-tree.
- Two parameters can be defined as:
- Alpha : (highest value) choice we have found so far at any point along the path of Maximizer. The initial value of alpha is $-\infty$
- Beta : (lowest value) choice we have found so far at any point along the path of Minimizer. The initial value of beta is $+\infty$

The alpha - beta pruning to a standard minimax algo returns the same move as the standard algorithm does, but it removes all the node, which are not really affecting the final decision but making algorithm slow.

1/12/23

AIES_Assignment_2

```
def isMovesLeft(board):
    for i in range(3):
        for j in range(3):patiadkvnaslvnakvsc
            if board[i][j] == '_':
                return True
    return False

def evaluate(b):
    for row in range(3):
        if b[row][0] == b[row][1] == b[row][2]:
            if b[row][0] == player:
                return 10
            elif b[row][0] == opponent:
                return -10
    for col in range(3):
        if b[0][col] == b[1][col] == b[2][col]:
            if b[0][col] == player:
                return 10
            elif b[0][col] == opponent:
                return -10
    if b[0][0] == b[1][1] == b[2][2]:
        if b[0][0] == player:
            return 10
        elif b[0][0] == opponent:
            return -10
    if b[0][2] == b[1][1] == b[2][0]:
        if b[0][2] == player:
            return 10
        elif b[0][2] == opponent:
            return -10
    return 0

def minimax(board, depth, isMax):
    score = evaluate(board)
    if score == 10:
        return score
    if score == -10:
        return score
    if not isMovesLeft(board):
        return 0

    if isMax:
        best = -1000
        for i in range(3):
            for j in range(3):
                if board[i][j] == '_':
                    board[i][j] = player
                    best = max(best, minimax(board, depth + 1, not
```

```
isMax))
                    board[i][j] = '_'
        return best
    else:
        best = 1000
        for i in range(3):
            for j in range(3):
                if board[i][j] == '_':
                    board[i][j] = opponent
                    best = min(best, minimax(board, depth + 1, not
isMax))
                    board[i][j] = '_'
        return best

def findBestMove(board):
    bestVal = -1000
    bestMove = (-1, -1)
    for i in range(3):
        for j in range(3):
            if board[i][j] == '_':
                board[i][j] = player
                moveVal = minimax(board, 0, False)
                board[i][j] = '_'
                if moveVal > bestVal:
                    bestMove = (i, j)
                    bestVal = moveVal
    return bestMove

player, opponent = 'x', 'o'

board = [
    ['x', 'o', 'o'],
    ['x', 'x', 'o'],
    ['_', '_', '_']
]
bestMove = findBestMove(board)
print("The Optimal Move is :")
print("ROW:", bestMove[0], " COL:", bestMove[1])
```

```
The Optimal Move is :
ROW: 2  COL: 0
```