**Name = Manasvi Tyagi**

**University Roll No: 202401100400118**

**Branch: AIML**

**Section: B**

**Problem Statement: Noughts and Crosses with Alpha-Beta Pruning**

**Noughts and Crosses with Alpha-Beta Pruning**

Noughts and Crosses (Tic-Tac-Toe) is a two-player game where players take turns marking X or O on a 3x3 grid.

The objective is to get three of their marks in a row, column, or diagonal before the opponent does.

This document explains how to implement the game using the Minimax algorithm with Alpha-Beta Pruning, allowing the AI to make optimal moves efficiently.

**Algorithm Overview:**

1. \*\*Board Representation\*\*: The game board is a 3x3 grid initialized with empty spaces.
2. \*\*Game Logic\*\*: Players take turns making moves; the game checks for winners after each move.
3. \*\*Minimax Algorithm\*\*: Evaluates all possible moves and assigns scores based on potentialoutcomes.
4. \*\*Alpha-Beta Pruning\*\*: Optimizes the Minimax algorithm by pruning unnecessary branches,reducing computation time.
5. \*\*AI Move Selection\*\*: The AI uses Minimax with Alpha-Beta Pruning to choose the best move.

**Minimax with Alpha-Beta Pruning:**

* The Minimax function evaluates possible game states and assigns scores.
* Alpha-Beta Pruning optimizes this by stopping the evaluation of branches that won't affect theoutcome.
* This reduces the number of nodes evaluated, making the AI more efficient.

**Python Implementation:**

import math

HUMAN = 'X'

AI = 'O'

EMPTY = ' '

def minimax(board, depth, alpha, beta, is\_maximizing):

winner = check\_winner(board) if winner == AI:

return 10 - depth if winner == HUMAN: return depth - 10 if is\_full(board):

return 0 if is\_maximizing:

max\_eval = -math.inf for i in range(3): for j in range(3): if board[i][j] == EMPTY:

board[i][j] = AI eval = minimax(board, depth + 1, alpha, beta, False) board[i][j] = EMPTY max\_eval = max(max\_eval, eval) alpha = max(alpha, eval) if beta <= alpha:

break return max\_eval

**Conclusion:**

This implementation ensures that the AI always plays optimally, making it unbeatable if played correctly.

Using Alpha-Beta Pruning, we significantly reduce unnecessary computations, making the game more efficient.

OUTPUT:-