**Title Page Project Title:** Noughts and Crosses with Alpha-Beta Pruning
**Submitted By:** Aditya Mishra
**Roll Number:** 202401100300017
**Course:** BTECH
**Date:** 11-03-2025

---

# Introduction

Noughts and Crosses (Tic-Tac-Toe) is a classic two-player game where players take turns marking X or O in a 3×3 grid. The objective is to form a line of three marks either horizontally, vertically, or diagonally. This project implements an AI opponent using the Minimax algorithm with Alpha-Beta Pruning to make optimal moves. Alpha-Beta Pruning optimizes the Minimax search, reducing the number of nodes evaluated and improving efficiency.

---

# Methodology

1. **Game Representation**: The game board is represented as a 3×3 matrix.
2. **Minimax Algorithm**: The AI recursively evaluates all possible moves to choose the best one.
3. **Alpha-Beta Pruning**: Enhances Minimax by eliminating unnecessary calculations, improving efficiency.
4. **User Input**: The human player selects a move by entering row and column indices.
5. **AI Move**: The AI calculates the best move and updates the board.
6. **Win Condition Check**: The game ends when either player wins or the board is full.

# Code

```
import math

AI = 'X'
HUMAN = 'O'
EMPTY = ' '

# Initialize board
board = [
    [EMPTY, EMPTY, EMPTY],
    [EMPTY, EMPTY, EMPTY],
    [EMPTY, EMPTY, EMPTY]
]

def print_board(board):
    for row in board:
        print('|'.join(row))
        print('-' * 5)

def check_winner(board):
    # Check rows and columns
    for i in range(3):
        if board[i][0] == board[i][1] == board[i][2] != EMPTY:
            return board[i][0]
        if board[0][i] == board[1][i] == board[2][i] != EMPTY:
            return board[0][i]

    # Check diagonals
    if board[0][0] == board[1][1] == board[2][2] != EMPTY:
```

```
        return board[0][0]
    if board[0][2] == board[1][1] == board[2][0] != EMPTY:
        return board[0][2]

    return None

# Check if board is full
def is_full(board):
    return all(cell != EMPTY for row in board for cell in row)

# Minimax with Alpha-Beta Pruning
def minimax(board, depth, is_maximizing, alpha, beta):
    winner = check_winner(board)

    if winner == AI:
        return 10 - depth
    elif winner == HUMAN:
        return depth - 10
    elif 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, False, alpha, beta)
                    board[i][j] = EMPTY
                    max_eval = max(max_eval, eval)
                    alpha = max(alpha, eval)
```

```
                if beta <= alpha:  # Pruning
                    break
        return max_eval
    else:
        min_eval = math.inf
        for i in range(3):
            for j in range(3):
                if board[i][j] == EMPTY:
                    board[i][j] = HUMAN
                    eval = minimax(board, depth + 1, True, alpha, beta)
                    board[i][j] = EMPTY
                    min_eval = min(min_eval, eval)
                    beta = min(beta, eval)
                    if beta <= alpha:  # Pruning
                        break
        return min_eval

# Find the best move for AI
def best_move():
    best_score = -math.inf
    move = None
    for i in range(3):
        for j in range(3):
            if board[i][j] == EMPTY:
                board[i][j] = AI
                score = minimax(board, 0, False, -math.inf, math.inf)
                board[i][j] = EMPTY
                if score > best_score:
                    best_score = score
                    move = (i, j)
    return move
```

```
# Main game loop
def play_game():
    print("Welcome to Tic-Tac-Toe!")
    print_board(board)

    while True:
        # Human move
        row, col = map(int, input("Enter row and column (0-2): ").split())
        if board[row][col] != EMPTY:
            print("Cell already occupied! Try again.")
            continue
        board[row][col] = HUMAN
        print_board(board)

        if check_winner(board) == HUMAN:
            print("You win!")
            break
        elif is_full(board):
            print("It's a draw!")
            break

        # AI move
        ai_move = best_move()
        if ai_move:
            board[ai_move[0]][ai_move[1]] = AI
            print("AI played:")
            print_board(board)

        if check_winner(board) == AI:
            print("AI wins!")
```

```
            break
        elif is_full(board):
            print("It's a draw!")
            break

play_game()
```

---

# Output/Result

```
Enter row and column (0-2): 0 0
O|X|O
-----
 |X|
-----
O|O|X
-----
AI played:
O|X|O
-----
X|X|
-----
O|O|X
-----
Enter row and column (0-2): 1 2
O|X|O
-----
X|X|O
-----
O|O|X
-----
It's a draw!
```

(

---

# References/Credits

**Minimax Algorithm Explanation**: Russell, S. & Norvig, P. (2016). *Artificial Intelligence: A Modern Approach (3rd Edition)*. Pearson.

**Alpha-Beta Pruning Optimization**: Knuth, D. E., & Moore, R. W. (1975). "An analysis of alpha-beta pruning." *Artificial Intelligence*, 6(4), 293-326.

**Tic-Tac-Toe Strategy**: Wikipedia Contributors. (2023). "Tic-Tac-Toe." *Wikipedia, The Free Encyclopedia*. Link