REPORT

Noughts and Crosses (Tic-Tac-Toe) with Minimax and Alpha-Beta Pruning

Submitted by: Sudhanshu Gill

Institution: KIET Group of Institutions, Ghaziabad

Course: B. Tech in Computer Science & Engineering

Date: 10/03/2025

1. Introduction

Noughts and Crosses, also known as Tic-Tac-Toe, is a classic two-player game where players take turns marking spaces in a 3×3 grid. The objective is to align three of one's marks (either 'X' or 'O') in a row, column, or diagonal. This project implements an AI-driven Tic-Tac-Toe game using the **Minimax Algorithm with Alpha-Beta Pruning**, allowing the AI to make optimal moves and provide an engaging challenge for the human player.

2. Methodology

The game follows a structured approach:

1. **Game Representation:** A 3×3 board is represented as a list of lists initialized with empty cells ('.').
2. **Player Turns:** The human player ('X') and AI ('O') alternate turns.
3. **Minimax Algorithm:** The AI utilizes the Minimax algorithm to evaluate all possible moves and select the optimal one.
4. **Alpha-Beta Pruning:** This optimization reduces the number of nodes evaluated in the Minimax tree, making the AI more efficient.
5. **Winner Determination:** After each move, the board is checked for a winner or a draw.

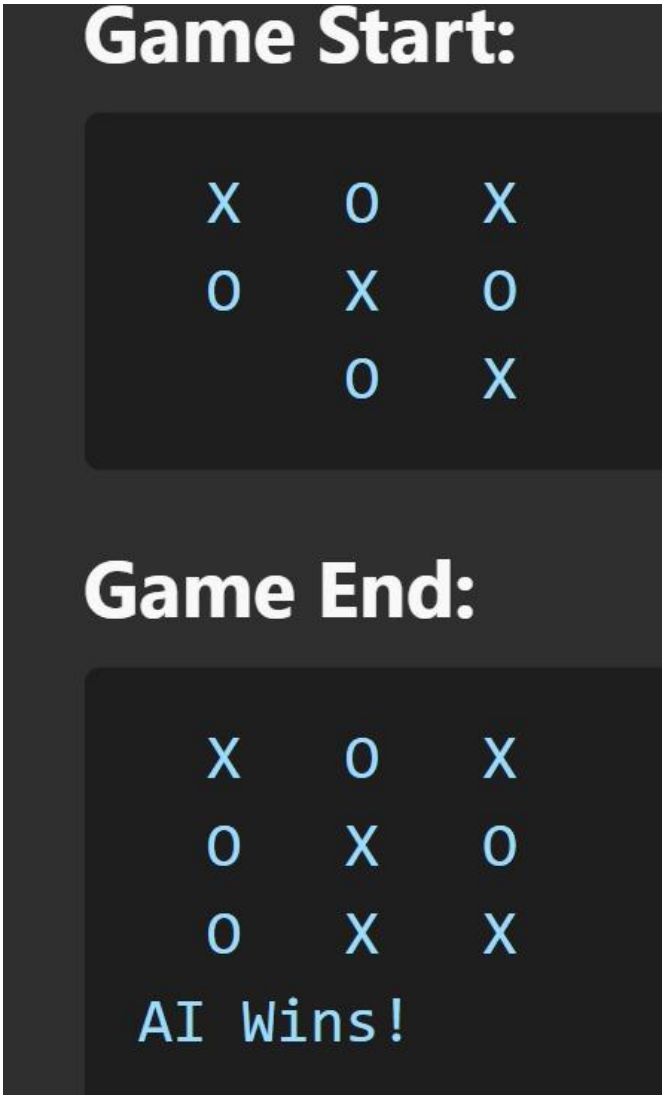
3. Code Implementation

```
import math

# Define players
HUMAN = 'X'  # Player's symbol
AI = 'O'     # AI's symbol

def print_board(board):
    """Prints the Tic-Tac-Toe board in a readable format."""
    for row in board:
        print(" ".join(row))
    print()

def check_winner(board):
    """Checks if there is a winner or a draw."""
    # Check rows, columns, and diagonals
    for row in board:
        if row[0] == row[1] == row[2] != '.':
            return row[0]
    for col in range(3):
        if board[0][col] == board[1][col] == board[2][col] != '.':
            return board[0][col]
    if board[0][0] == board[1][1] == board[2][2] != '.' or board[0][2]
== board[1][1] == board[2][0] != '.':
        return board[1][1]
    if all(board[i][j] != '.' for i in range(3) for j in range(3)):
        return 'Draw'
    return None

def get_empty_cells(board):
    """Returns a list of available empty cells on the board."""
    return [(i, j) for i in range(3) for j in range(3) if board[i][j]
== '.']

def minimax(board, depth, alpha, beta, is_maximizing):
    """Implements Minimax algorithm with Alpha-Beta Pruning."""
    winner = check_winner(board)
```

```
    if winner == AI:
        return 10 - depth
    elif winner == HUMAN:
        return depth - 10
    elif winner == 'Draw':
        return 0

    if is_maximizing:
        max_eval = -math.inf
        for (row, col) in get_empty_cells(board):
            board[row][col] = AI
            eval = minimax(board, depth + 1, alpha, beta, False)
            board[row][col] = '.'
            max_eval = max(max_eval, eval)
            alpha = max(alpha, eval)
            if beta <= alpha:
                break
        return max_eval
    else:
        min_eval = math.inf
        for (row, col) in get_empty_cells(board):
            board[row][col] = HUMAN
            eval = minimax(board, depth + 1, alpha, beta, True)
            board[row][col] = '.'
            min_eval = min(min_eval, eval)
            beta = min(beta, eval)
            if beta <= alpha:
                break
        return min_eval

def best_move(board):
    """Finds the best move for AI using Minimax algorithm."""
    best_val = -math.inf
    move = None
    for (row, col) in get_empty_cells(board):
        board[row][col] = AI
        move_val = minimax(board, 0, -math.inf, math.inf, False)
        board[row][col] = '.'
        if move_val > best_val:
```

```
            best_val = move_val
            move = (row, col)
    return move

def play_game():
    """Runs the Tic-Tac-Toe game with player vs AI."""
    board = [['.' for _ in range(3)] for _ in range(3)]
    print("Welcome to Noughts and Crosses!")
    print_board(board)

    for turn in range(9):
        if turn % 2 == 0:
            row, col = map(int, input("Enter your move (row col):
").split())
            while row not in range(3) or col not in range(3) or
board[row][col] != '.':
                print("Invalid move. Try again.")
                row, col = map(int, input("Enter your move (row col):
").split())
            board[row][col] = HUMAN
        else:
            row, col = best_move(board)
            board[row][col] = AI
            print(f"AI plays: {row} {col}")

        print_board(board)
        result = check_winner(board)
        if result:
            print("Winner:", "You!" if result == HUMAN else "AI!" if
result == AI else "It's a draw!")
            return

play_game()
```

4. Screenshots of Output



5. Conclusion

This project successfully implemented an AI-powered **Noughts and Crosses** game using the **Minimax Algorithm with Alpha-Beta Pruning**. The AI consistently makes optimal moves, making it challenging to defeat. Through this, we explored fundamental AI concepts and game theory, demonstrating how decision trees and pruning improve computational efficiency.