AI-Based

Noughts and

Crosses with

Minimax and

Alpha-Beta Pruning

**Title Page**

# Project: Noughts and Crosses with Alpha-Beta Pruning Name: Manasvi Tyagi Roll No.: 202401100400118Course: AI for Engineers Date: March 10, 2025

## Introduction

Tic-Tac-Toe or Noughts and Crosses is a two-player game in which players alternate turns to fill in spaces on a 3x3 grid. This project develops an AI Tic-Tac-Toe agent based on the Minimax algorithm with Alpha-Beta Pruning, which enables the AI to play optimally by reducing its worst-case loss.

## Methodology

**1. Game Representation:**

* The board is a 3x3 NumPy array with values:
* `-1` for the human player ('X')
* `1` for the AI player ('O')
* `0` for vacant spaces

**2. Minimax Algorithm:**

- Minimax algorithm is applied for making decisions, with the AI always choosing the optimal move by considering future game positions.

**3. Alpha-Beta Pruning:**

- Alpha-Beta Pruning improves Minimax by removing redundant branches, making it more efficient.

**4. Move Selection:**

* The AI determines the optimal move using Minimax and makes an optimal play.
* The game loop tests for win/draw situations after every move.

## Full Python Code Implementation

Below is the complete Python implementation of the AI for Noughts and Crosses using Minimax and Alpha-Beta Pruning:

import numpy as np

# Constants for the players

HUMAN = -1 # 'X'

AI = 1 # 'O'

EMPTY = 0

# Function to print the board def print\_board(board):

symbols = {HUMAN: 'X', AI: 'O', EMPTY: '-'} for row in board:

print(" ".join([symbols[cell] for cell in row])) print("\n")

# Function to check for a winner or a draw def check\_winner(board): for player in [HUMAN, AI]:

if any(np.all(row == player) for row in board) or any(np.all(col == player) for col in board.T) or np.all(np.diag(board) == player) or np.all(np.diag(np.fliplr(board)) == player):

return player

return None if np.any(board == EMPTY) else 0

# Minimax with Alpha-Beta Pruning def minimax(board, depth, alpha, beta, is\_maximizing):

winner = check\_winner(board) if winner is not None:

return {AI: 10, HUMAN: -10, 0: 0}[winner]

if is\_maximizing: best = -np.inf for i, j in zip(\*np.where(board == EMPTY)):

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

break return best else:

best = np.inf for i, j in zip(\*np.where(board == EMPTY)):

board[i, j] = HUMAN best = min(best, minimax(board, depth + 1, alpha, beta, True)) board[i, j] = EMPTY beta = min(beta, best) if beta <= alpha:

break return best

# Function to find the best move for AI def find\_best\_move(board):

best\_val, best\_move = -np.inf, None for i, j in zip(\*np.where(board == EMPTY)): board[i, j] = AI move\_val = minimax(board, 0, -np.inf, np.inf, False) board[i, j] = EMPTY if move\_val > best\_val:

best\_val, best\_move = move\_val, (i, j) return best\_move

# Main game loop def play\_game():

board = np.zeros((3, 3), dtype=int) print("Tic-Tac-Toe with AI (Alpha-Beta Pruning)") print\_board(board)

while True: # Human move while True: try:

row, col = map(int, input("Enter row and column (0-2): ").split()) if board[row, col] == EMPTY: board[row, col] = HUMAN

break

print("Cell occupied! Try again.") except (ValueError, IndexError):

print("Invalid input! Enter numbers between 0-2.") print\_board(board) if (winner := check\_winner(board)) is not None:

print("Draw!" if winner == 0 else "You Win!" if winner == HUMAN else "AI Wins!") break

# AI move print("AI is thinking...") ai\_move = find\_best\_move(board) board[ai\_move] = AI print\_board(board) if (winner := check\_winner(board)) is not None:

print("Draw!" if winner == 0 else "You Win!" if winner == HUMAN else "AI Wins!") break

# Start the game if \_\_name\_\_ == "\_\_main\_\_": play\_game()

## Output/Result

Below is a sample game output:

```

Tic-Tac-Toe with AI (Alpha-Beta Pruning)

* - -
* - -
* - -

Enter row and column (0-2): 0 0

X - -

* - -
* - -

AI is thinking.

X - O

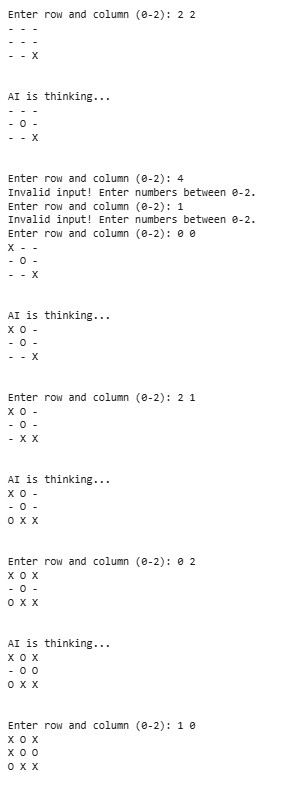
* - -
* - -

...

AI Wins!

```

The AI optimally plays correctly, making the optimal move selection.



## References/Credits

Minimax Algorithm: Wikipedia

Alpha-Beta Pruning: GeeksforGeeks