Project Report Noughts and Crosses Game using Alpha-Beta Pruning

Submitted by: Jahnavi Singh(57)

Date: [11-03-2025]

Submitted to: Shivansh Prasad

1. Introduction

Noughts and Crosses, also known as Tic-Tac-Toe, is a classic two-player game played on a 3x3 grid. Players take turns marking a cell with their respective symbols ('X' or 'O'), aiming to place three of their marks in a horizontal, vertical, or diagonal row to win. This game serves as a perfect foundation to explore decision-making algorithms in artificial intelligence.

In this project, we implement Noughts and Crosses in Python and explore different play modes including Human vs Human and AI vs Human using the Minimax algorithm enhanced with Alpha-Beta Pruning. The optimization reduces the number of nodes evaluated in the game tree, improving performance without compromising decision quality.

2. Methodology

We approached the game implementation in the following structured way:

- **Game Board Representation**: We used a 3x3 matrix to represent the game board. Each cell stores either 'X', 'O', or None.
- **Game Logic**: Functions were written to evaluate the game board, check for win/draw conditions, print the board, and get available moves.
- **Alpha-Beta Pruning with Minimax**: The AI logic is based on the Minimax algorithm with Alpha-Beta Pruning to optimize decision-making. The 'X' player is set as the maximizing player, while 'O' is the minimizing player.
- **User Interaction**: Input is taken from users to simulate either Human vs Human or Human vs AI game modes. Input
- validation is included to ensure proper format and avoid occupied cells.

3. Code Typed

```
import math
import random
from typing import List, Tuple, Optional

Player = str  # 'X' or 'O'
Board = List[List[Optional[Player]]]

# Initialize empty board
def create_board() -> Board:
    return [[None for _ in range(3)] for _ in range(3)]

# Check for a winner or draw
def check_winner(board: Board) -> Optional[Player]:
    for player in ['X', 'O']:
        # Check rows, columns, and diagonals
        for i in range(3):
            if all(cell == player for cell in board[i]):
                return player
            if all(board[j][i] == player for j in range(3)):
                return player
        if all(board[i][i] == player for i in range(3)) or all(board[i][2-i] ==
player for i in range(3)):
            return player
```

```
    return None

def is_full(board: Board) -> bool:
    return all(cell is not None for row in board for cell in row)

def evaluate(board: Board) -> int:
    winner = check_winner(board)
    if winner == 'X':
        return 1
    elif winner == 'O':
        return -1
    return 0

def get_available_moves(board: Board) -> List[Tuple[int, int]]:
    return [(i, j) for i in range(3) for j in range(3) if board[i][j] is None]

def print_board(board: Board) -> None:
    for row in board:
        print(' | '.join(cell if cell else ' ' for cell in row))
        print('-' * 5)

def get_user_move(board: Board, player: Player) -> Tuple[int, int]:
    while True:
        try:
```

```
            move = input(f"Player {player}, enter your move as row,col (0-2 for each): ")
            i, j = map(int, move.strip().split(','))
            if (i, j) in get_available_moves(board):
                return i, j
            else:
                print("Invalid move. Cell is occupied or out of range.")
        except (ValueError, IndexError):
            print("Invalid input format. Use row,col with values between 0 and 2.")

# Game simulation: Human (X) vs Human (O)
def play_game():
    board = create_board()
    current_player = 'X'

    while not check_winner(board) and not is_full(board):
        print_board(board)
        print(f"{current_player}'s turn")

        move = get_user_move(board, current_player)

        if move:
            board[move[0]][move[1]] = current_player
```

```
            current_player = 'O' if current_player == 'X' else 'X'

        print_board(board)
        winner = check_winner(board)
        if winner:
            print(f"{winner} wins!")
        else:
            print("It's a draw!")

play_game()
```

5. Screenshot Output photo pasted

```
X's turn
Player X, enter your move as row,col (0-2 for each): 1,1
  |   |
-----
  | X |
-----
  |   |
-----
O's turn
Player O, enter your move as row,col (0-2 for each): 1,2
  |   |
-----
  | X | O
-----
  |   |
-----
X's turn
Player X, enter your move as row,col (0-2 for each): 0,1
  | X |
-----
  | X | O
-----
  |   |
-----
O's turn
Player O, enter your move as row,col (0-2 for each): 2,2
  | X |
-----
  | X | O
-----
  |   | O
-----
```

```
X's turn
Player X, enter your move as row,col (0-2 for each): 2,1
  | X |
-----
  | X | O
-----
  | X | O
-----
X wins!
```