Noughts and Crosses with Alpha-Beta Pruning

NAME- ABHAY PRATAP SINGH

ROLL NO- 202401100400005

BRANCH- CSE(AIML)

SECTION - A

PROBLEM - Noughts and Crosses with Alpha-Beta Pruning

1. Introduction -

Noughts and Crosses (Tic-Tac-Toe) is a two-player game on a 3×3 grid. Players take turns placing ‘X’ or ‘O’, aiming to form a row, column, or diagonal. AI can play optimally using the Minimax Algorithm with Alpha-Beta Pruning.

2. Minimax Algorithm

Minimax is a decision-making algorithm where:

- Maximizer (AI) aims for the highest score.
- Minimizer (Opponent) aims for the lowest score.

How it Works:

1. Generate possible moves.
2. Assign scores: +10 (AI wins), -10 (Opponent wins), 0 (Draw).
3. AI picks the best move; the opponent tries to minimize it.

3. Alpha-Beta Pruning

An optimization that eliminates unnecessary evaluations, improving efficiency.

- Alpha: Best move for Maximizer.
- Beta: Best move for Minimizer.
- If a move is worse than an already explored one, further checks are stopped (pruning).

Benefits:

- Speeds up decision-making.
- Reduces computations without affecting optimality.

4. Methodology

1. Initialize Board – Create a 3×3 grid.
2. Player & AI Turns – Players alternate moves.
3. Evaluate Board – Check for win/loss/draw.
4. Minimax with Alpha-Beta Pruning – AI explores moves, pruning unnecessary ones.
5. Find Best Move – AI selects the optimal move.
6. Game Progress – Repeat until win or draw.
7. Declare Result – Announce winner or draw.

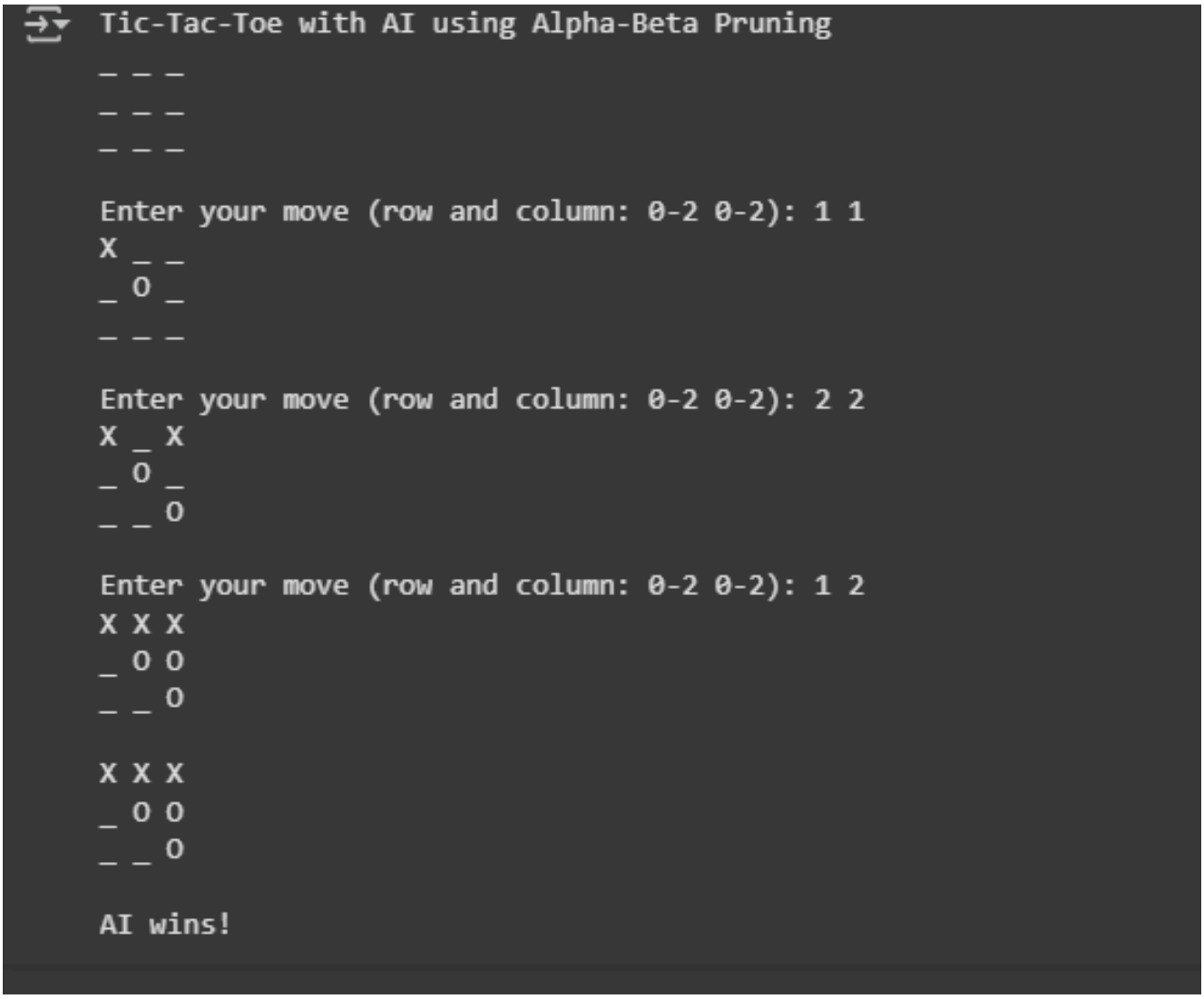
CODE

```
import math

# Function to print the Tic-Tac-Toe board
def print_board(board):
    for row in board:
        print(" ".join(row))
    print()

# Function to check if there are moves left on the board
def is_moves_left(board):
    return any('_' in row for row in board)


# Function to evaluate the board and return a score
def evaluate(board):
    for i in range(3):
        # Check rows and columns for victory
        if board[i][0] == board[i][1] == board[i][2] != '_':
            return 10 if board[i][0] == 'X' else -10
        if board[0][i] == board[1][i] == board[2][i] != '_':
            return 10 if board[0][i] == 'X' else -10

    # Check diagonals for victory
    if board[0][0] == board[1][1] == board[2][2] != '_':
        return 10 if board[0][0] == 'X' else -10
    if board[0][2] == board[1][1] == board[2][0] != '_':
        return 10 if board[0][2] == 'X' else -10

    return 0
```

```
# Minimax algorithm with Alpha-Beta Pruning
def minimax(board, depth, is_max, alpha, beta):
    score = evaluate(board)
    if score in (10, -10):
        return score
    if not is_moves_left(board):
        return 0
    if is_max:
        best = -math.inf
        for i in range(3):
            for j in range(3):
                if board[i][j] == '_':
                    board[i][j] = 'X'
                    best = max(best, minimax(board, depth + 1, False, alpha, beta))
                    board[i][j] = '_'
                    alpha = max(alpha, best)
                    if beta <= alpha:
                        break # Alpha-Beta Pruning
        return best
    else:
        best = math.inf
        for i in range(3):
            for j in range(3):
                if board[i][j] == '_':
                    board[i][j] = 'O'
                    best = min(best, minimax(board, depth + 1, True, alpha, beta))
                    board[i][j] = '_'
                    beta = min(beta, best)
                    if beta <= alpha:
                        break # Alpha-Beta Pruning
        return best
# Function to find the best move for AI (X)
def find_best_move(board):
    best_val = -math.inf
    best_move = (-1, -1)

    for i in range(3):
        for j in range(3):
            if board[i][j] == '_':
                board[i][j] = 'X'
                move_val = minimax(board, 0, False, -math.inf, math.inf)
                board[i][j] = '_'

                if move_val > best_val:
                    best_move = (i, j)
                    best_val = move_val

    return best_move
```

```
# Main function to run the Tic-Tac-Toe game
def main():
    board = [['_', '_', '_'],
             ['_', '_', '_'],
             ['_', '_', '_']]

    print("Tic-Tac-Toe with AI using Alpha-Beta Pruning")
    print_board(board)

    while is_moves_left(board) and evaluate(board) == 0:
        # Get user input for their move
        row, col = map(int, input("Enter your move (row and column: 0-2 0-2): ").split())
        if board[row][col] != '_':
            print("Invalid move! Try again.")
            continue
        board[row][col] = 'O'
        # Check if the game is over after player's move
        if evaluate(board) != 0 or not is_moves_left(board):
            break

        # AI makes a move
        ai_move = find_best_move(board)
        board[ai_move[0]][ai_move[1]] = 'X'

        # Print the updated board after AI's move
        print_board(board)

    # Print final board and declare the result
    print_board(board)
    score = evaluate(board)
    if score == 10:
        print("AI wins!")
    elif score == -10:
        print("You win!")
    else:
        print("It's a draw!")

# Run the game if this script is executed
if __name__ == "__main__":
    main()
```

IMAGE OF OUTPUT



Noughts and Crosses with Alpha-Beta Pruning