Noughts and Crosses with Alpha-Beta Pruning

Problem Statement:

The task is to develop an intelligent version of the classic Tic-Tac-Toe game, also known as Noughts and Crosses, where the AI plays optimally using the Minimax Algorithm and Alpha-Beta Pruning. In this game, Player X is the human player, and Player O is the AI, which uses the Minimax algorithm to decide its moves, improving efficiency with Alpha-Beta pruning.

- **Name: Hemang Singh**
- **Roll No: 202401100400098**
- **Course: Introduction To AI**
- **Branch: CSE AIML**
- **Section: B**
- **Date: 10/03/2025**

INTRODUCTION

Problem Explanation:

Noughts and Crosses (Tic-Tac-Toe) is a 2-player game played on a 3x3 grid. The players take turns marking spaces with 'X' (Player X) and 'O' (Player O). The first player to align three marks in a row, column, or diagonal wins. If the grid is filled without a winner, the game results in a draw.

This project simulates an optimal game where **Player X** and **Player O** play using the **Minimax algorithm**, which evaluates possible moves to select the best one. To improve efficiency, **Alpha-Beta Pruning** is used to eliminate unnecessary calculations.

Methodology

The approach for solving the Noughts and Crosses (Tic-Tac-Toe) problem involves the following key steps:

1. **Game Setup**:
   The game starts with an empty 3x3 grid. Player X is the maximizing player, and Player O is the minimizing player. They take turns marking their moves.
2. **Minimax Algorithm**:
   The Minimax algorithm evaluates all possible moves to select the optimal one for each player. Player X aims to maximize their chances of winning, while Player O minimizes Player X's chances.
3. **Alpha-Beta Pruning**:
   Alpha-Beta pruning enhances the Minimax algorithm by eliminating unimportant branches in the decision tree, improving efficiency and reducing computational time.
4. **Game Over Condition**:
   The game checks for a win after each move. If a player places three marks in a row, column, or

diagonal, they win. If the grid is full without a winner, the game ends in a draw.

5. **Player Interaction**:
   Player X (AI) and Player O (opponent) alternate turns. Both players use the Minimax algorithm to determine their moves.
6. **End of Game**:
   The game concludes when there is a winner or when it ends in a draw. The result is displayed once the game finishes.

Code:

```
import random

# Constants for the game
PLAYER_X = 'X'
PLAYER_O = 'O'
EMPTY = ' '

# Initialize the board
def init_board():
    return [[EMPTY, EMPTY, EMPTY] for _ in range(3)]

# Check if a player has won
def check_win(board, player):
    # Check rows, columns, and diagonals
    for i in range(3):
        if all([board[i][j] == player for j in range(3)]) or 
all([board[j][i] == player for j in range(3)]):
            return True
```

```
    if all([board[i][i] == player for i in range(3)]) or
all([board[i][2-i] == player for i in range(3)]):
        return True
    return False

# Check if the game is over (win or draw)
def game_over(board):
    if check_win(board, PLAYER_X):
        return 1  # Player X wins
    if check_win(board, PLAYER_O):
        return -1  # Player O wins
    if all(board[i][j] != EMPTY for i in range(3) for j in
range(3)):
        return 0  # Draw (no empty spaces)
    return None  # Game is ongoing

# Minimax with Alpha-Beta Pruning
def minimax(board, depth, alpha, beta,
maximizing_player):
    result = game_over(board)
```

```
    if result is not None:
        return result  # 1, -1, or 0 (win, loss, draw)

    if maximizing_player:
        max_eval = -float('inf')
        for i in range(3):
            for j in range(3):
                if board[i][j] == EMPTY:
                    board[i][j] = PLAYER_X
                    eval = minimax(board, depth + 1, alpha,
beta, False)
                    board[i][j] = EMPTY
                    max_eval = max(max_eval, eval)
                    alpha = max(alpha, eval)
                    if beta <= alpha:
                        break  # Beta cut-off
        return max_eval
    else:
        min_eval = float('inf')
        for i in range(3):
```

```
        for j in range(3):
            if board[i][j] == EMPTY:
                board[i][j] = PLAYER_O
                eval = minimax(board, depth + 1, alpha,
beta, True)
                board[i][j] = EMPTY
                min_eval = min(min_eval, eval)
                beta = min(beta, eval)
                if beta <= alpha:
                    break  # Alpha cut-off
    return min_eval

# Find the best move for the current player
def best_move(board, maximizing_player):
    best_val = -float('inf') if maximizing_player else
float('inf')
    move = (-1, -1)
    for i in range(3):
        for j in range(3):
            if board[i][j] == EMPTY:
```

```
            board[i][j] = PLAYER_X if
maximizing_player else PLAYER_O
            move_val = minimax(board, 0, -float('inf'),
float('inf'), not maximizing_player)
            board[i][j] = EMPTY
            if (maximizing_player and move_val >
best_val) or (not maximizing_player and move_val
< best_val):
                best_val = move_val
                move = (i, j)
    return move


# Print the board with dashes after the last line
def print_board(board):
    for i, row in enumerate(board):
        print(' | '.join(row))
        if i < 2:  # Only print dashes between rows, not
after the last one
            print('---------')
    print("\n")  # Adding a newline after the board
for better readability.
```

```
# Example gameplay loop
def play_game():
    board = init_board()
    current_player = PLAYER_X  # Start with "X"

    while game_over(board) is None:
        print_board(board)

        if current_player == PLAYER_X:
            # Player X's turn (user input)
            valid_move = False
            while not valid_move:
                try:
                    i, j = map(int, input("Player X, enter your
move (row and column, e.g. 1 2): ").split())
                    if board[i][j] == EMPTY:
                        valid_move = True
                        board[i][j] = PLAYER_X
                    else:
```

```
                print("Cell already taken! Try again.")
            except (ValueError, IndexError):
                print("Invalid input! Enter row and
column numbers between 0 and 2.")

        else:
            # Player O's turn (AI move)
            print("Player O is making a move...")
            i, j = best_move(board, False)
            board[i][j] = PLAYER_O

        current_player = PLAYER_O if current_player
== PLAYER_X else PLAYER_X

        # Print a message after each move
        print("Next move is being made...\n")

    print_board(board)
    result = game_over(board)
    if result == 1:
```

```
        print("Player X wins!")
    elif result == -1:
        print("Player O wins!")
    else:
        print("It's a draw!")

if __name__ == "__main__":
    play_game()
```

Output:

```
  |   |
---------
  |   |
---------
  |   |


Player X, enter your move (row and column, e.g. 1 2): 2 2
Next move is being made...

  |   |
---------
  |   |
---------
  |   | X


Player O is making a move...
Next move is being made...

  |   |
---------
  | O |
---------
  |   | X
```

```
Player X, enter your move (row and column, e.g. 1 2): 2 1
Next move is being made...

  |   |
---------
  | O |
---------
  | X | X


Player O is making a move...
Next move is being made...

  |   |
---------
  | O |
---------
O | X | X


Player X, enter your move (row and column, e.g. 1 2): 1 2
Next move is being made...

  |   |
---------
  | O | X
---------
O | X | X
```

```
Player O is making a move...
Next move is being made...

  |   | O
---------
  | O | X
---------
O | X | X


Player O wins!
```

Reference:

Minimax Algorithm - For understanding the Minimax algorithm, I referred to various online resources and tutorials:

- **GeeksforGeeks:** "Minimax Algorithm with Alpha-Beta Pruning" (https://www.geeksforgeeks.org/mini-max-algorithm-in-artificial-intelligence/)
- **Alpha-Beta Pruning** - To implement Alpha-Beta Pruning for optimizing the Minimax algorithm:
- **Wikipedia:** "Alpha–beta pruning" (https://en.wikipedia.org/wiki/Alpha%E2%80%93beta_pruning)