KIET GROUP OF INSTITUTIONS
Connecting Life with Learning
www.kiet.edu
Delhi-NCR, Ghaziabad

NOUGHTS AND CROSSES WITH ALPHA-BETA PRUNING

NAME :- PRIYANKA
BRANCH :- CSE-AI (C)
UNIV. ROLL NO. :- 202401100300184
CLASS ROLL NO. :- 37

INTRODUCTION:

Noughts and Crosses with Alpha-Beta Pruning

Noughts and Crosses (Tic-Tac-Toe) is a simple yet strategic game where two players aim to align three marks in a row, column, or diagonal. While easy for humans, AI requires efficient decision-making to play optimally.

The Minimax algorithm is commonly used for AI in this game, but it can be computationally expensive as it evaluates all possible moves. Alpha-Beta Pruning optimizes Minimax by eliminating unnecessary calculations, making the AI faster and more efficient.

This report explores how Alpha-Beta Pruning improves Minimax in Noughts and Crosses, ensuring optimal gameplay while reducing computational complexity.

METHODOLOGY:

1. Game Representation

The game board is represented as a 3×3 grid, where each cell can be empty (' '), occupied by the human player ('O'), or occupied by the AI ('X').

The game checks for a winner or a draw after each move.

2. Minimax Algorithm Implementation

The Minimax algorithm is used to evaluate all possible moves and select the best one.

It assigns scores to game states:

+10 for an AI win

-10 for a human win

0 for a draw

The AI plays to maximize its score, while the human plays to minimize it.

3. Alpha-Beta Pruning Optimization

Alpha-Beta Pruning is applied to Minimax to eliminate unnecessary branches, improving efficiency.

Two parameters, Alpha (α) and Beta (β), are introduced:

Alpha (α): The best score the maximizer (AI) can achieve.

Beta (β): The best score the minimizer (human) can achieve.

The algorithm prunes (ignores) branches where a move is already proven to be worse than a previously evaluated option.

4. AI Move Selection

The AI iterates through all possible moves and uses the Minimax with Alpha-Beta Pruning to select the move with the highest score.

The AI prioritizes winning moves, blocks the opponent’s winning moves, and chooses the best available option.

5. Human Input and Game Flow

The game runs in a loop, alternating turns between the human player and AI until there is a winner or a draw.

The human provides input by selecting row and column indices for their move.

After each turn, the board is updat

ed, and the game state is checked.

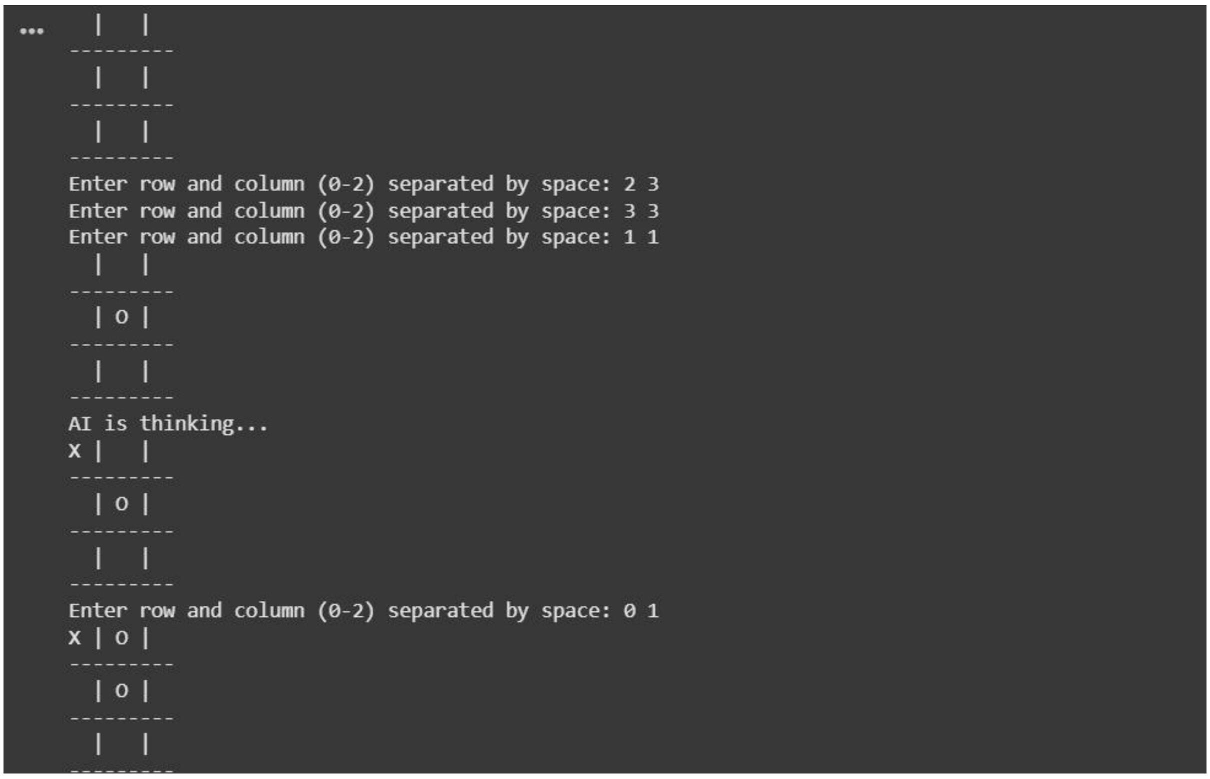
This methodology ensures that the AI plays optimally while improving computational efficiency using Alpha-Beta Pruning.

CODE:-

```
import math

# Constants

PLAYER_X = "X"  # AI

PLAYER_O = "O"  # Human

EMPTY = " "


# Initialize Board

def create_board():

    return [[EMPTY] * 3 for _ in range(3)]


# Check for winner

def check_winner(board):

    for row in board:

        if row[0] == row[1] == row[2] and row[0] != EMPTY:

            return row[0]


    for col in range(3):

        if board[0][col] == board[1][col] == board[2][col] and board[0][col] != EMPTY:

            return board[0][col]


    if board[0][0] == board[1][1] == board[2][2] and board[0][0] != EMPTY:

        return board[0][0]


    if board[0][2] == board[1][1] == board[2][0] and board[0][2] != EMPTY:

        return board[0][2]


    return None


# Check if board is full
```

```
def is_draw(board):
    return all(cell != EMPTY for row in board for cell in row)


# Get available moves
def get_available_moves(board):
    return [(r, c) for r in range(3) for c in range(3) if board[r][c] == EMPTY]


# Minimax with Alpha-Beta Pruning
def minimax(board, depth, alpha, beta, is_maximizing):
    winner = check_winner(board)
    if winner == PLAYER_X:
        return 10 - depth
    if winner == PLAYER_O:
        return depth - 10
    if is_draw(board):
        return 0


    if is_maximizing:
        max_eval = -math.inf
        for (r, c) in get_available_moves(board):
            board[r][c] = PLAYER_X
            eval = minimax(board, depth + 1, alpha, beta, False)
            board[r][c] = EMPTY
            max_eval = max(max_eval, eval)
            alpha = max(alpha, eval)
            if beta <= alpha:
                break  # Alpha-Beta Pruning
        return max_eval
    else:
        min_eval = math.inf
        for (r, c) in get_available_moves(board):
            board[r][c] = PLAYER_O
            eval = minimax(board, depth + 1, alpha, beta, True)
            board[r][c] = EMPTY
            min_eval = min(min_eval, eval)
            beta = min(beta, eval)
            if beta <= alpha:
                break  # Alpha-Beta Pruning
        return min_eval

# Best AI Move
def best_move(board):
    best_score = -math.inf
    move = None
    for (r, c) in get_available_moves(board):
        board[r][c] = PLAYER_X
        score = minimax(board, 0, -math.inf, math.inf, False)
        board[r][c] = EMPTY
        if score > best_score:
            best_score = score
            move = (r, c)
    return move

# Print Board
def print_board(board):
    for row in board:
        print(" | ".join(row))
        print("-" * 9)

# Main Game Loop
def play_game():
    board = create_board()
    human_turn = True

    while True:
        print_board(board)
        winner = check_winner(board)
        if winner:
            print(f"Winner: {winner}!")
            break
        if is_draw(board):
            print("It's a draw!")
            break

        if human_turn:
            move = None
            while move not in get_available_moves(board):
                try:
                    row, col = map(int, input("Enter row and column (0-2) separated by space: ").split())
                    move = (row, col)
                except ValueError:
                    pass
            board[row][col] = PLAYER_O
        else:
            print("AI is thinking...")
            row, col = best_move(board)
            board[row][col] = PLAYER_X

        human_turn = not human_turn

# Run the game
if _name_ == "_main_":
    play_game()
```

SCREENSHOT OF OUTPUT:-



```
...   |   |
    ---------
      |   |
    ---------
      |   |
    ---------
    Enter row and column (0-2) separated by space: 2 3
    Enter row and column (0-2) separated by space: 3 3
    Enter row and column (0-2) separated by space: 1 1
      |   |
    ---------
      | O |
    ---------
      |   |
    ---------
    AI is thinking...
    X |   |
    ---------
      | O |
    ---------
      |   |
    ---------
    Enter row and column (0-2) separated by space: 0 1
    X | O |
    ---------
      | O |
    ---------
      |   |
    ---------
```