EXPERIMENT 14

Alpha & Beta pruning

AIM:

To Write the python program to implement Alpha & Beta pruning algorithm for gaming

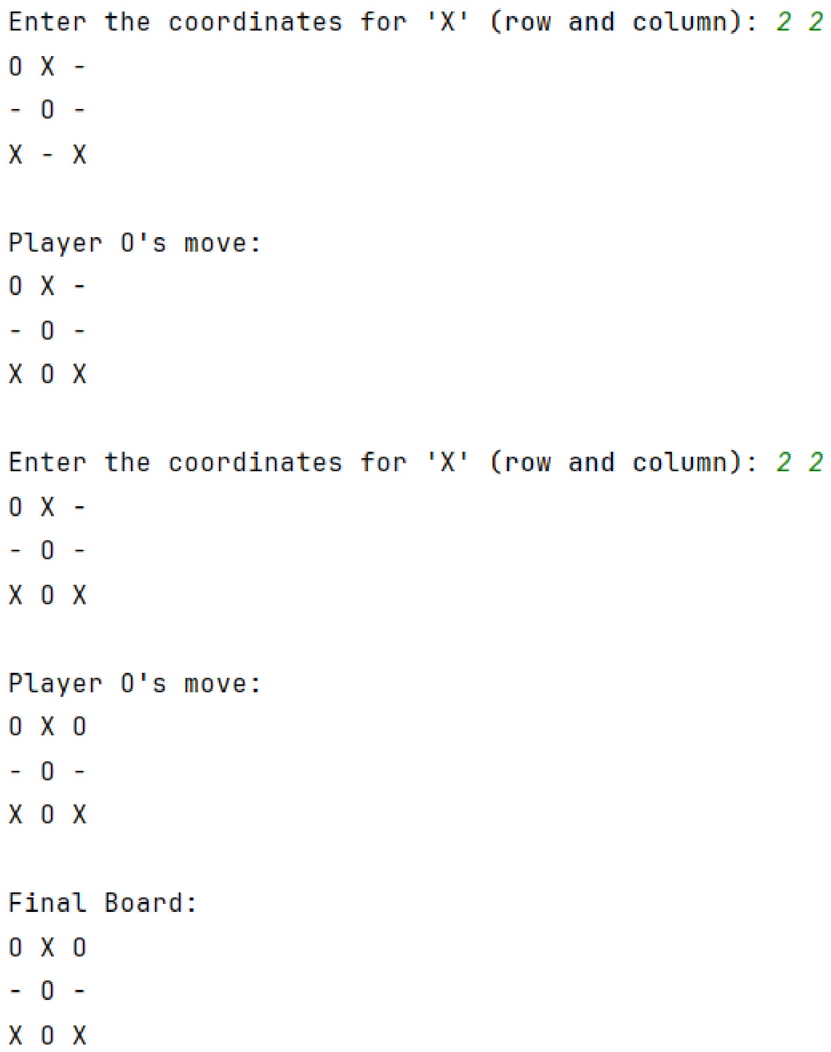
PROCEDURE:

- The code implements the Minimax algorithm with alpha-beta pruning to determine the optimal move for the AI player ('O') in Tic-Tac-Toe.
- The alpha_beta_pruning function recursively evaluates possible moves and prunes branches that cannot lead to a better outcome.
- It assigns scores (-1, 0, or 1) depending on whether 'X' wins, 'O' wins, or the game ends in a draw, respectively.
- It utilizes the alpha-beta pruning algorithm to efficiently narrow down the search space and improve performance.
- The main function orchestrates the game loop, alternating between player ('X') and AI ('O') turns until the game reaches a terminal state.
- The game board is represented as a 3x3 grid using a nested list structure, where empty cells are denoted by '-'.
- Functions such as print_board and is_board_full facilitate board visualization and checking for fullness, respectively.

PROGRAM:

```
import math

def is_winner(board, player):
    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

def is_board_full(board):
    return all(board[i][j] != '-' for i in range(3) for j in range(3))

def evaluate(board):
    if is_winner(board, 'X'): return -1
    elif is_winner(board, 'O'): return 1
    elif is_board_full(board): return 0
    return None

def alpha_beta_pruning(board, depth, alpha, beta, maximizing_player):
    score = evaluate(board)

    if score is not None:
        return score

    if maximizing_player:
        max_eval = float('-inf')
        for i in range(3):
            for j in range(3):
                if board[i][j] == '-':
                    board[i][j] = 'O'
                    eval = alpha_beta_pruning(board, depth + 1, alpha, beta, False)
                    board[i][j] = '-'  # Undo the move
                    max_eval = max(max_eval, eval)
                    alpha = max(alpha, eval)
                    if beta <= alpha:
                        break
        return max_eval
    else:
        min_eval = float('inf')
        for i in range(3):
            for j in range(3):
                if board[i][j] == '-':
                    board[i][j] = 'X'
                    eval = alpha_beta_pruning(board, depth + 1, alpha, beta, True)
                    board[i][j] = '-'  # Undo the move
                    min_eval = min(min_eval, eval)
                    beta = min(beta, eval)
                    if beta <= alpha:
                        break
        return min_eval

def find_best_move(board):
    best_val, best_move = float('-inf'), (-1, -1)
    alpha, beta = float('-inf'), float('inf')

    for i in range(3):
        for j in range(3):
            if board[i][j] == '-':
                board[i][j] = 'O'
                move_val = alpha_beta_pruning(board, 0, alpha, beta, False)
                board[i][j] = '-'  # Undo the move

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

    return best_move

def print_board(board):
    for row in board: print(' '.join(row))
    print()

def main():
    board = [['-' for _ in range(3)] for _ in range(3)]
    print("Initial Board:")
    print_board(board)

    for _ in range(4):
        x, y = map(int, input("Enter the coordinates for 'X' (row and column): ").split())
        board[x][y] = 'X'
        print_board(board)

        print("Player O's move:")
        best_move = find_best_move(board)
        board[best_move[0]][best_move[1]] = 'O'
        print_board(board)

    print("Final Board:")
    print_board(board)

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

OUTPUT:



```
Enter the coordinates for 'X' (row and column): 2 2
O X -
- O -
X - X

Player O's move:
O X -
- O -
X O X

Enter the coordinates for 'X' (row and column): 2 2
O X -
- O -
X O X

Player O's move:
O X O
- O -
X O X

Final Board:
O X O
- O -
X O X
```

RESULT:

Hence the program has been successfully verified.