AI TIC TAC TOE PROGRAM

VENKATESH VINAY CHANDLE

1BM22CS325

INPUT:

```
import random

PLAYER = 'X'
AI = 'O'
EMPTY = ' '

def print_board(board):
    for row in range(3):
        print('|'.join(board[row]))
        if row < 2:
            print('-' * 5)

def check_winner(board, player):
    # Check rows, columns, and diagonals
    for i in range(3):
        if board[i][0] == board[i][1] == board[i][2] == player:
            return True
        if board[0][i] == board[1][i] == board[2][i] == player:
            return True
    if board[0][0] == board[1][1] == board[2][2] == player:
        return True
    if board[0][2] == board[1][1] == board[2][0] == player:
        return True
    return False

def is_board_full(board):
    for row in board:
```

```
        if EMPTY in row:
            return False
    return True

def minimax(board, depth, is_maximizing):
    if check_winner(board, AI):
        return 10 - depth
    if check_winner(board, PLAYER):
        return depth - 10
    if is_board_full(board):
        return 0

    if is_maximizing:
        best = -float('inf')
        for row in range(3):
            for col in range(3):
                if board[row][col] == EMPTY:
                    board[row][col] = AI
                    best = max(best, minimax(board, depth + 1, False))
                    board[row][col] = EMPTY
        return best
    else:
        best = float('inf')
        for row in range(3):
            for col in range(3):
                if board[row][col] == EMPTY:
                    board[row][col] = PLAYER
                    best = min(best, minimax(board, depth + 1, True))
                    board[row][col] = EMPTY
        return best
```

```
def find_best_move(board):
    best_val = -float('inf')
    best_move = None

    for row in range(3):
        for col in range(3):
            if board[row][col] == EMPTY:
                board[row][col] = AI
                move_val = minimax(board, 0, False)
                board[row][col] = EMPTY

                if move_val > best_val:
                    best_move = (row, col)
                    best_val = move_val
    return best_move

def player_move(board):
    while True:
        try:
            move = int(input("Enter your move (1-9): ")) - 1
            row, col = divmod(move, 3)
            if board[row][col] == EMPTY:
                board[row][col] = PLAYER
                break
            else:
                print("Invalid move. Cell already taken.")
        except (ValueError, IndexError):
            print("Invalid input. Please enter a number between 1 and 9.")

def play_game():
    board = [[EMPTY] * 3 for _ in range(3)]
```

```
    print("Welcome to Tic-Tac-Toe!")
    print("Player is 'X' and Bot is 'O'.")

    while True:
        print_board(board)

        if is_board_full(board):
            print("It's a draw!")
            break



        player_move(board)
        if check_winner(board, PLAYER):
            print_board(board)
            print("Player wins!")
            break

        if is_board_full(board):
            print("It's a draw!")
            break

        print("AI's move:")
        row, col = find_best_move(board)
        board[row][col] = AI
        if check_winner(board, AI):
            print_board(board)
            print("Bot wins!")
            break
print("Name: Venkatesh Vinay Chandle, USN: 1BM22CS325")
if __name__ == "__main__":
    play_game()
```

OUTPUT:

```
Name: Venkatesh Vinay Chandle, USN: 1BM22CS325
Welcome to Tic-Tac-Toe!
Player is 'X' and Bot is 'O'.
 | |
-----
 | |
-----
 | |
Enter your move (1-9): 1
AI's move:
X| |
-----
 |O|
-----
 | |
Enter your move (1-9): 2
AI's move:
X|X|O
-----
 |O|
-----
 | |
Enter your move (1-9): 7
AI's move:
X|X|O
-----
O|O|
-----
X| |
Enter your move (1-9): 6
AI's move:
X|X|O
-----
O|O|X
-----
X|O|
Enter your move (1-9): 9
It's a draw!
```

1 Tic-Tac-Toe implementation using python

Pseudocode

```
function minimax (node, depth, isMaximizingPlayer)
    if node is a terminal state:
        return evaluate(node)
    if isMaximizingPlayer:
        bestValue = -inf
        for each child in node:
            value = minimax(child, depth, false)
            bestValue = max(bestValue, value)
        return bestValue
    else:
        bestValue = +inf
        for each child in node
            value = minimax(child, depth, true)
            bestValue = min(bestValue, value)
        return bestValue
```

R 4/10/24

Output:

Player = 'O' Bot = 'X'

X | _ | _
_ | _ | _
_ | _ | _

-> enter position for O : 2

X | O | _
_ | _ | _
_ | _ | _

X | O | _
X | _ | _
_ | _ | _

-> enter position for O : 7

X | O | _
X | _ | _
O | _ | _

X | O | _
X | X | _
O | _ | O

X | O | _
X | X | X
O | _ | O

Bot wins!

-> enter position for O : 9