Date:24.10.25

TASK:11

GAME PLAYING

CO1, CO2, CO3 S3

PROBLEM STATEMENT : Develop an interactive game-playing system that allows a human player to play against the computer. The system should implement an intelligent strategy for the computer to make optimal moves and ensure a fair, engaging experience.

AIM: To design and implement an intelligent game-playing application that demonstrates the use of Artificial

1. To implement a classic board game (Tic Tac Toe) where the player competes against the computer
2. To use the Minimax algorithm for the computer's move decision-making.
3. To demonstrate the concept of game trees, heuristic evaluation, and optimal play.
4. To provide an interactive interface for user input and real-time game feedback.

Intelligence techniques for decision-making.

OBJECTIVE:

1. To implement a classic board game (Tic Tac Toe) where the player competes against the computer.
2. To use the Minimax algorithm for the computer's move decision-making.
3. To demonstrate the concept of game trees, heuristic evaluation, and optimal play.
4. To provide an interactive interface for user input and real-time game feedback.

DESCRIPTION : Game playing is an important domain in Artificial Intelligence. It involves designing algorithms that can make optimal moves in a competitive environment.

In this project, we implement the Tic Tac Toe game — a simple two-player game played on a 3×3 grid. The AI opponent uses the Minimax algorithm, which evaluates all possible moves recursively and selects the best move that maximizes its chances of winning while minimizing the player's chances.

ALGORITHM:

1. Start the game with an empty 3×3 board.
2. The player and computer take turns alternately.
3. For the computer's move:

- Generate all possible moves.
- For each move, simulate the game recursively using the Minimax function:
- If the computer wins, assign a positive score.
- If the player wins, assign a negative score.
- If it's a draw, assign zero.
- Choose the move with the best score (max for computer, min for player).

4. Repeat until the board is full or a player wins.
5. End the game by displaying the winner or draw message.

PROGRAM :

```
# TIC TAC TOE GAME USING MINIMAX ALGORITHM

import math

player = 'X'

SS   computer

= 'O'

def print_board(board):

    for row in board:

        print('|'.join(row))

    print()

def check_winner(board):

    for row in board:

        if row.count(row[0]) == 3 and row[0] != ' ':
```

```
            return row[0]
    for col in
    range(3):
        if board[0][col] == board[1][col] == board[2][col] != ' ':
            return board[0][col]
    if board[0][0] == board[1][1] == board[2][2] != ' ':
        return board[0][0]
    if board[0][2] == board[1][1] == board[2][0] != ' ':
        return board[0][2]
    return None
def is_full(board):
    return all(cell != ' ' for row in board for cell in row)
def minimax(board, depth, is_maximizing):
    winner = check_winner(board)
    if winner == computer:
        return 1
    elif winner == player:
        return -1
    elif is_full(board):
        return 0
 if is_maximizing:
        best_score = -math.inf
        for i in range(3):
            for j in range(3):
                if board[i][j] == ' ': board[i]
                    [j] = computer
```

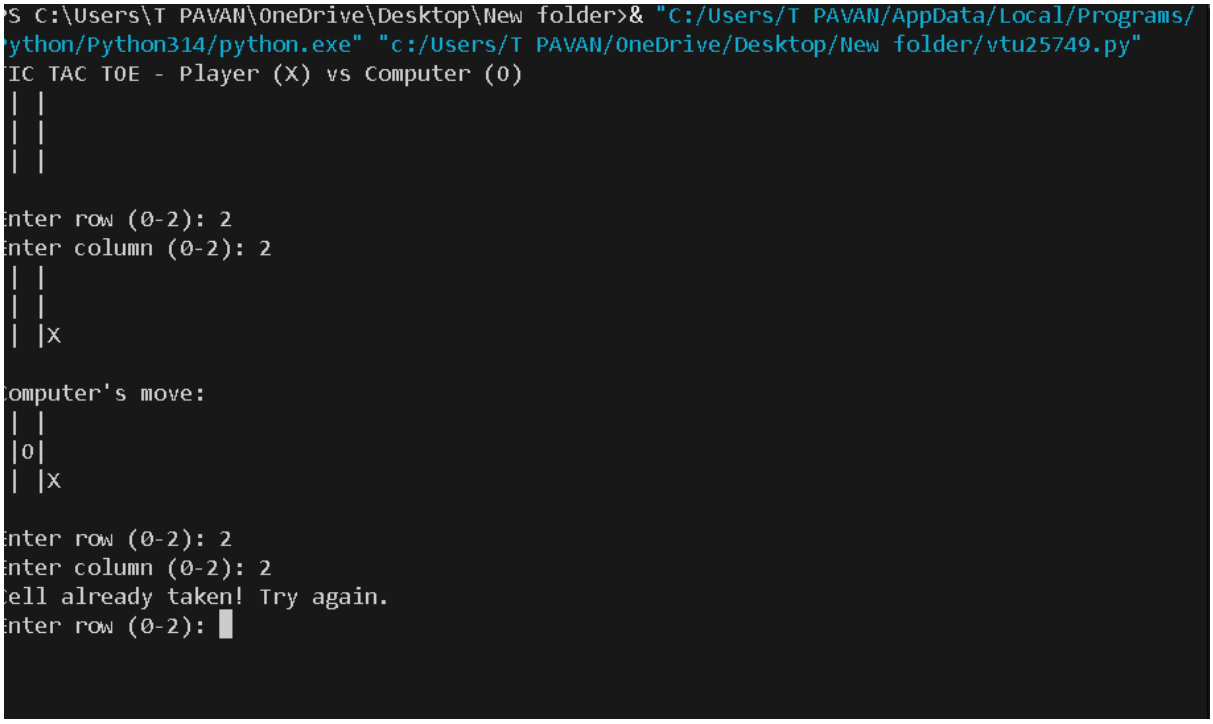
```
            score = minimax(board, depth + 1, False)
```

```
                board[i][j] = ' '
                best_score = max(score, best_score)
        return best_score
    else:
        best_score = math.inf
        for i in range(3):
            for j in range(3):
                if board[i][j] == ' ':
                board[i][j] =
                player
                score = minimax(board, depth + 1, True)
                board[i][j] = ' '
                best_score = min(score, best_score)
        return best_score
def computer_move(board):
    best_score = -math.inf
    move = None
    for i in range(3):
        for j in
        range(3):
            if board[i][j] == ' ':
                board[i][j] = computer
                score = minimax(board, 0, False) board[i]
                [j] = ' '
                if          score          >
                    best_score:
```

```
                best_score = score
                move = (i, j)
    if move:
```

```
        board[move[0]][move[1]] = computer
def main():
    board = [[' ' for _ in range(3)] for _ in range(3)]
    print("TIC TAC TOE - Player (X) vs Computer (O)")
    print_board(board)
 while True:
        row = int(input("Enter row (0-2): "))
        col = int(input("Enter column (0-2):
        ")) if board[row][col] != ' ':
            print("Cell already taken! Try again.")
            continue
             print_board(board)
        if check_winner(board) == player:
            print(" You win!")
            break
        elif is_full(board):
            print("Á It's a draw!")
            break
        computer_move(board)
        print("Computer's move:")
      if check_winner(board) == computer:
            print("z Computer wins!")
                    elif is_full(board):
            print("Á It's a draw!")
            break
       main()
```

OUTPUT :



CONCLUSION: The Game Playing project successfully demonstrates the use of Artificial Intelligence for optimal decision-making.

By implementing the Minimax algorithm, the system ensures that the computer always plays optimally, either winning or forcing a draw.

This project showcases the practical application of AI search strategies in real-world game design.