INTRODUCTION TO AI

PERSONAL DETAILS:

Name: Suraj kumar

Branch- CSE AI "D"

University Roll no-2024011100300256

Problem Statement: Tic-Tac-Toe is a two-player game played on a 3x3 grid where players take turns placing X or O. The goal is to get three marks in a row, column, or diagonal. The game ends when a player wins or when the grid is full, resulting in a draw.

INTRODUCTION

Explanation of problem Statement:

Tic-Tac-Toe is a classic two-player game played on a 3x3 grid. The game follows these rules:

1. The game involves two players: **Player 1 (X)** and **Player 2 (O)**.
2. Players take turns placing their respective marks (**X** or **O**) on an empty cell in the 3x3 grid.
3. The objective is to form a horizontal, vertical, or diagonal line of three identical marks.
4. The game ends when:
   - A player achieves three marks in a row, column, or diagonal (that player wins).
   - The grid is completely filled without a winner (resulting in a draw).
5. The game starts with an empty board, and moves are made alternately by the players.

Requirements:

- The game should allow players to input their moves.
- It should check for valid moves (i.e., preventing moves in occupied cells).
- The game should determine the winner or declare a draw.
- It should provide a way to restart or exit.

METHODOLOGY

The approach to solving Tic-Tac-Toe involves:

1. **Game Representation** – Use a 3x3 matrix (list or array) to represent the board.
2. **Player Input Handling** – Allow players to input their moves, ensuring the chosen cell is empty.
3. **Win Condition Checking** – After each move, check rows, columns, and diagonals for three matching marks (**X** or **O**).
4. **Draw Condition Checking** – If the board is full without a winner, declare a draw.
5. **Turn Management** – Alternate turns between players and display the updated board after each move.
6. **Game Loop & Restart Option** – Continue the game until a win/draw occurs, then allow the players to restart or exit.

CODE

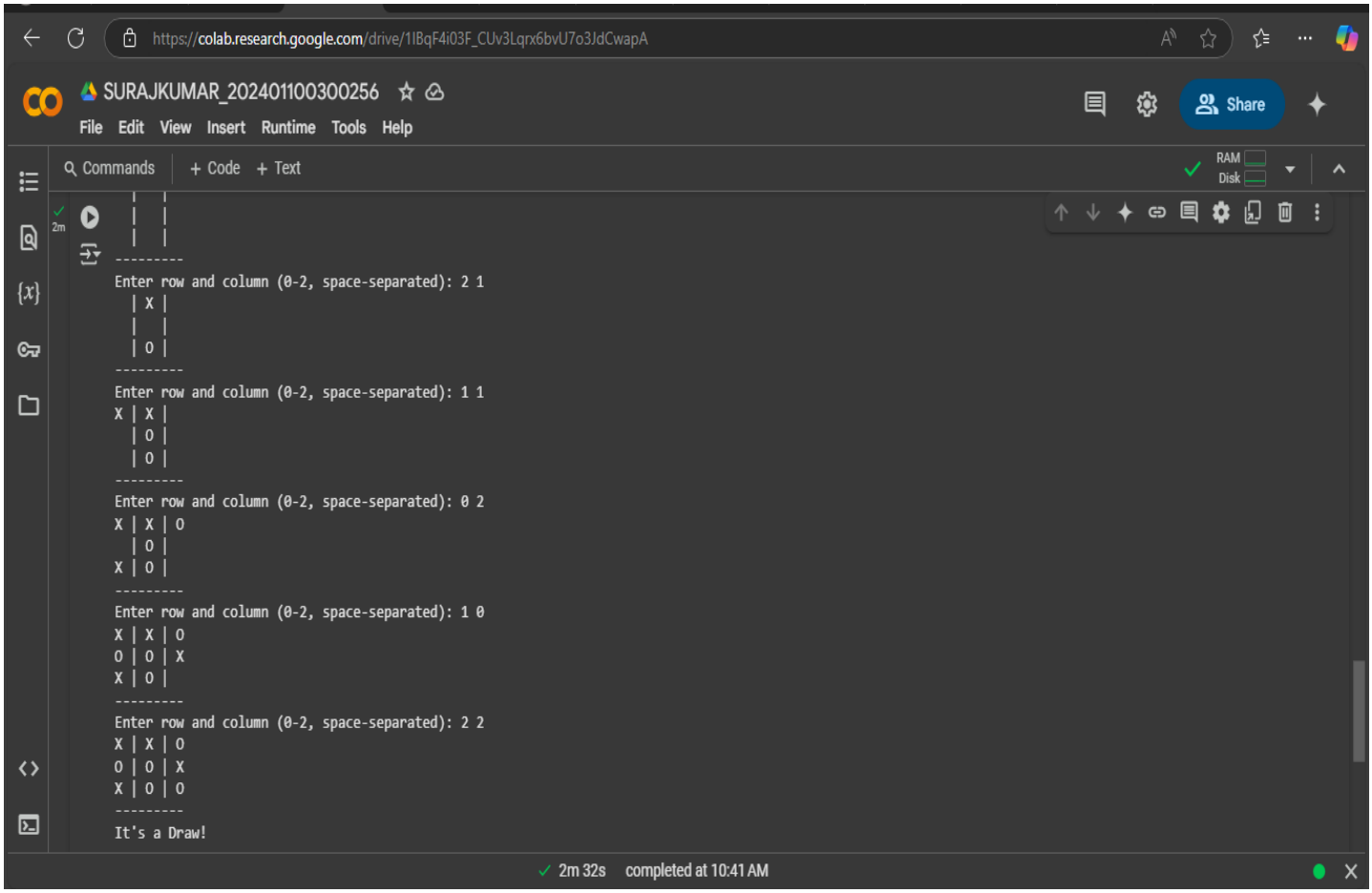
```
import math

# Function to print the current state of the board
def print_board(board):
    for row in board:
        print(" | ".join(row))
    print("-" * 9)

# Function to check if there are moves left on the board
def is_moves_left(board):
    return any(" " in row for row in board)

# Function to evaluate the board and return a score
# +10 if 'X' wins, -10 if 'O' wins, 0 if no winner yet
def evaluate(board):
    # Check rows for a win
    for row in board:
        if row[0] == row[1] == row[2] and row[0] != " ":
            return 10 if row[0] == "X" else -10

    # Check columns for a win
    for col in range(3):
        if board[0][col] == board[1][col] == board[2][col] and
board[0][col] != " ":
            return 10 if board[0][col] == "X" else -10

    # Check diagonals for a win
    if board[0][0] == board[1][1] == board[2][2] and board[0][0] !=
" ":
        return 10 if board[0][0] == "X" else -10

    if board[0][2] == board[1][1] == board[2][0] and board[0][2] !=
" ":
        return 10 if board[0][2] == "X" else -10

    return 0
```

```python
# Minimax function with Alpha-Beta Pruning
def minimax(board, depth, is_max, alpha, beta):
    score = evaluate(board)

    # If the game has been won, return score adjusted for depth
    if score == 10 or score == -10:
        return score - depth if score == 10 else score + depth

    # If no moves left, it's a draw
    if not is_moves_left(board):
        return 0

    if is_max:
        best = -math.inf
        for i in range(3):
            for j in range(3):
                if board[i][j] == " ":
                    board[i][j] = "X"
                    best = max(best, minimax(board, depth + 1,
False, alpha, beta))
                    board[i][j] = " "
                    alpha = max(alpha, best)
                    if beta <= alpha:
                        break
        return best
    else:
        best = math.inf
        for i in range(3):
            for j in range(3):
                if board[i][j] == " ":
                    board[i][j] = "O"
                    best = min(best, minimax(board, depth + 1,
True, alpha, beta))
                    board[i][j] = " "
                    beta = min(beta, best)
                    if beta <= alpha:
                        break
        return best
```

```
# Function to find the best move for AI ('X')
def find_best_move(board):
    best_val = -math.inf
    best_move = (-1, -1)

    for i in range(3):
        for j in range(3):
            if board[i][j] == " ":
                board[i][j] = "X"
                move_val = minimax(board, 0, False, -math.inf,
math.inf)
                board[i][j] = " "

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

    return best_move

# Main function to play the game
def main():
    print("Welcome to Tic-Tac-Toe!")
    board = [[" " for _ in range(3)] for _ in range(3)]

    while is_moves_left(board) and evaluate(board) == 0:
        print_board(board)
        try:
            row, col = map(int, input("Enter row and column (0-2,
space-separated): ").split())
            if 0 <= row < 3 and 0 <= col < 3 and board[row][col] ==
" ":
                board[row][col] = "O"
                if is_moves_left(board) and evaluate(board) == 0:
                    best_move = find_best_move(board)
                    board[best_move[0]][best_move[1]] = "X"
            else:
                print("Invalid move. Try again.")
                continue
        except ValueError:
```

```
            print("Invalid input. Enter two numbers between 0 and
2.")
            continue

    print_board(board)
    score = evaluate(board)
    if score == 10:
        print("AI Wins!")
    elif score == -10:
        print("You Win!")
    else:
        print("It's a Draw!")

# Run the game if the script is executed
if __name__ == "__main__":
    main()
```

SCREENSHOT


https://colab.research.google.com/drive/1IBqF4i03F_CUv3Lqrx6bvU7o3JdCwapA
SURAJKUMAR_202401100300256
File Edit View Insert Runtime Tools Help
Share
Commands + Code + Text
RAM
Disk
| |
| |

Enter row and column (0-2, space-separated): 2 1
| X |
| |
| O |

Enter row and column (0-2, space-separated): 1 1
X | X |
| O |
| O |

Enter row and column (0-2, space-separated): 0 2
X | X | O
| O |
X | O |

Enter row and column (0-2, space-separated): 1 0
X | X | O
O | O | X
X | O |

Enter row and column (0-2, space-separated): 2 2
X | X | O
O | O | X
X | O | O

It's a Draw!
2m 32s completed at 10:41 AM

CREDIT

→Chatgpt