TITLE PAGE

1. Project – Tic Tac toe solver.
2. Name – PRAKHAR SINGH.
3. ROLL NO.-2024011000300172.
4. Course name – Introduction to ai
5. Institute name-KIET GROUP OF INSTITUTION.
6. DATE- 11-03-2025

Introduction

Tic-Tac-Toe is a classic two-player game played on a 3x3 grid. Players alternate placing their marks (either an 'X' or an 'O') on the grid, with the goal of getting three of their marks in a row, either horizontally, vertically, or diagonally. The game ends when one player wins, or if all spaces on the grid are filled, resulting in a draw (also known as a "cat's game").

Methodology

Algorithm Overview: Tic-Tac-Toe AI with Minimax

The Tic-Tac-Toe AI is implemented using the Minimax algorithm, which ensures the AI plays optimally. The game consists of the following steps:

____

1. Game Setup

- The game board is a 3x3 grid represented as a 2D list.
- Players:
  - AI (X) → Maximizing Player
  - Human (O) → Minimizing Player
- The board starts empty.

____

2. Game Flow

1. User Move – The player selects a position (1-9), validated before placement.
2. Check for Win/Draw – The board is checked to determine if the game has ended.
3. AI Move – The AI calculates the best possible move using Minimax.
4. Repeat until a player wins or the board is full.

---

3. Win/Draw Conditions

- Winning Check:
  - AI or Human wins if they align three of their marks in a row, column, or diagonal.
- Draw Check:
  - The game is a draw if no empty spaces remain.

---

4. Minimax Algorithm

What is Minimax?

Minimax is a recursive algorithm used in decision-making games. It simulates all possible future moves and assigns a score to each scenario:

- Maximizing Player (AI/X):
  - Tries to get the highest possible score (+1 for AI win).
- Minimizing Player (Human/O):
  - Tries to get the lowest possible score (-1 for Human win).

Minimax Steps

1. Base Cases (Stopping Conditions)
   - If AI wins → return +1.
   - If Human wins → return -1.
   - If Draw → return 0.
2. Recursive Evaluation
   - Maximizing AI (AI's turn):
     - AI places "X" in an empty spot.

□ Calls minimax recursively for the opponent (minimizing step).

□ Chooses the move with the maximum score.

o Minimizing Human (Human's turn):

□ Human places "O" in an empty spot.

□ Calls minimax recursively for the opponent (maximizing step).

□ Chooses the move with the minimum score.

3. Backtracking

o The function undoes the move after evaluation to try another move.

o Ensures all possible moves are explored.

____

5. Finding the Best Move

1. The AI tries each empty spot on the board.
2. Uses the Minimax algorithm to determine the score for each move.
3. Chooses the move with the highest score.
4. Updates the board.

____

6. Complexity and Optimization

- Time Complexity: O(9!) (Factorial growth)

o Worst case: 362,880 states

o Redundant calculations can make it slow.

- Optimization using Alpha-Beta Pruning:

o Cuts off unnecessary branches in the recursion tree.

o Reduces execution time significantly.

Code Typed

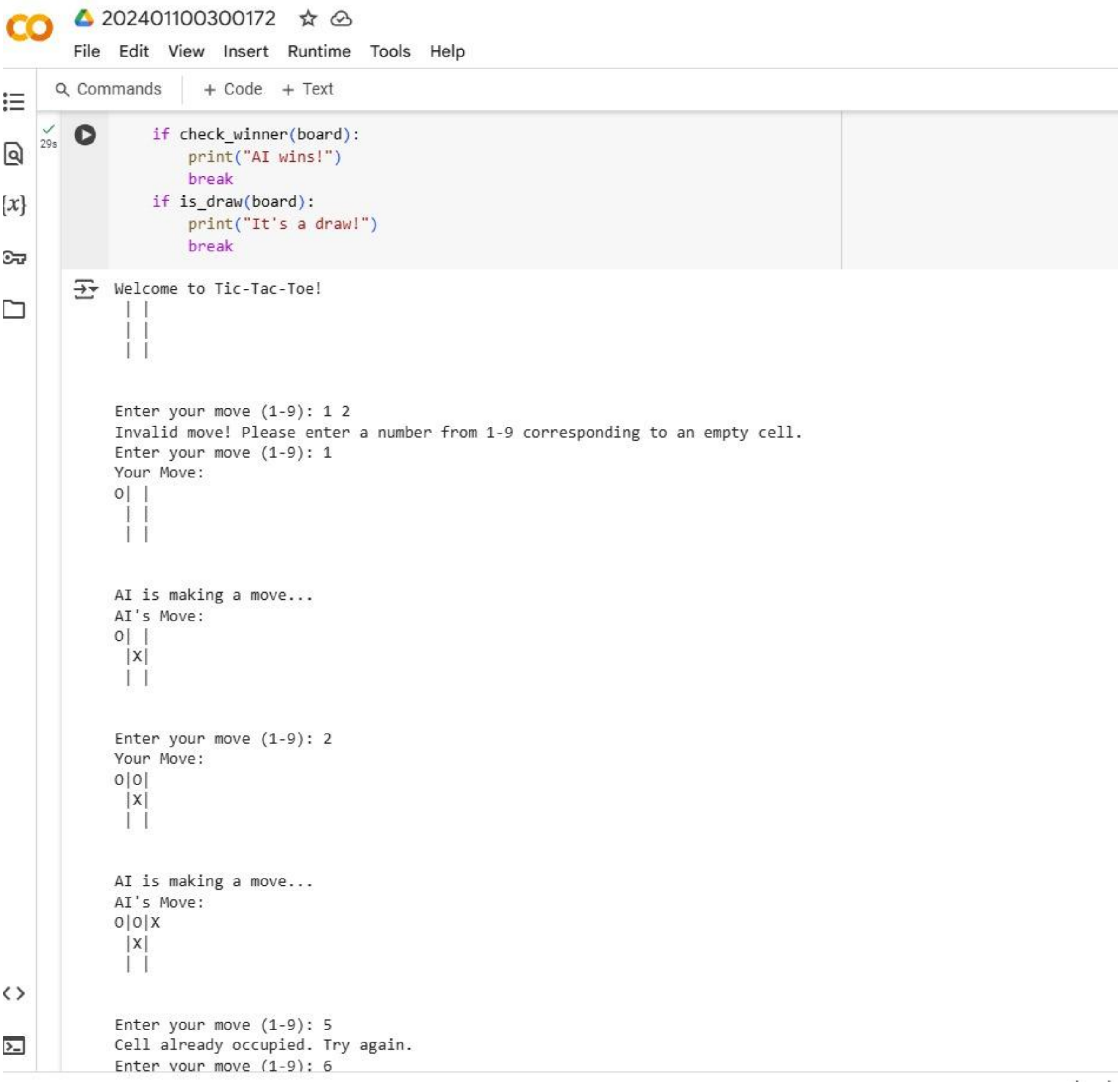
```
import math


# Constants for the players
AI = "X"  # AI is the maximizing player
HUMAN = "O"  # Human is the minimizing player
EMPTY = " "  # Empty cell in the board


# Function to print the Tic-Tac-Toe board
def print_board(board):
    for row in board:
        print("|".join(row))
    print("\n")


# Check if a player has won
def check_winner(board):
    # All possible winning combinations: rows, columns, diagonals
    win_combinations = [
        [board[0][0], board[0][1], board[0][2]],  # Row 1
        [board[1][0], board[1][1], board[1][2]],  # Row 2
        [board[2][0], board[2][1], board[2][2]],  # Row 3
        [board[0][0], board[1][0], board[2][0]],  # Column 1
        [board[0][1], board[1][1], board[2][1]],  # Column 2
        [board[0][2], board[1][2], board[2][2]],  # Column 3
        [board[0][0], board[1][1], board[2][2]],  # Diagonal 1
        [board[0][2], board[1][1], board[2][0]]   # Diagonal 2
    ]
    # Check for a winner
    if [AI, AI, AI] in win_combinations: return AI
    if [HUMAN, HUMAN, HUMAN] in win_combinations: return HUMAN
    return None  # No winner yet

# Check if the game is a draw
def is_draw(board):
    for row in board:
        if EMPTY in row:  # If there are any empty spots
            return False  # Not a draw yet
    return True  # All spots are filled

# MiniMax Algorithm to find the best move
def minimax(board, is_maximizing):
    winner = check_winner(board)
    if winner == AI: return 1  # AI wins, return 1
    if winner == HUMAN: return -1  # Human wins, return -1
    if is_draw(board): return 0  # Draw, return 0

    if is_maximizing:  # AI's turn (maximize score)
        best_score = -math.inf  # Start with a very low score
        for i in range(3):
            for j in range(3):
                if board[i][j] == EMPTY:  # If the cell is empty
                    board[i][j] = AI  # Try placing AI's symbol
                    score = minimax(board, False)  # Recursively evaluate the move
                    board[i][j] = EMPTY  # Undo the move
                    best_score = max(best_score, score)  # Maximize score for AI
        return best_score
    else:  # Human's turn (minimize score)
        best_score = math.inf  # Start with a very high score
        for i in range(3):
            for j in range(3):
                if board[i][j] == EMPTY:  # If the cell is empty
                    board[i][j] = HUMAN  # Try placing Human's symbol
                    score = minimax(board, True)  # Recursively evaluate the move
                    board[i][j] = EMPTY  # Undo the move
                    best_score = min(best_score, score)  # Minimize score for Human
        return best_score

# Find the best move for AI
def find_best_move(board):
    best_score = -math.inf
    best_move = None
    for i in range(3):
        for j in range(3):
            if board[i][j] == EMPTY:  # If the cell is empty
                board[i][j] = AI  # Try placing AI's symbol
                score = minimax(board, False)  # Get the score for the move
                board[i][j] = EMPTY  # Undo the move
                if score > best_score:  # If this move is better than the previous one
                    best_score = score
                    best_move = (i, j)  # Save the best move
    return best_move
```

```
# Function for the user to input their move
def user_move(board):
    while True:
        try:
            move = int(input("Enter your move (1-9): ")) - 1  # User inputs a number from 1-9
            row, col = divmod(move, 3)  # Convert the input to board indices
            if board[row][col] == EMPTY:  # Check if the cell is empty
                board[row][col] = HUMAN  # Place the Human's symbol
                break
            else:
                print("Cell already occupied. Try again.")
        except (ValueError, IndexError):
            print("Invalid move! Please enter a number from 1-9 corresponding to an empty cell.")

# Example Tic-Tac-Toe board (3x3 grid)
board = [
    [" ", " ", " "],
    [" ", " ", " "],
    [" ", " ", " "]
]

# Print the current board
print("Welcome to Tic-Tac-Toe!")
print_board(board)

# Main game loop
while True:
```

```
# Player (Human) move
user_move(board)
print("Your Move:")
print_board(board)

if check_winner(board):
    print("Congratulations, you win!")
    break
if is_draw(board):
    print("It's a draw!")
    break

# AI move (Optimal Move)
print("AI is making a move...")
best_move = find_best_move(board)
if best_move:
    board[best_move[0]][best_move[1]] = AI  # AI makes the move
    print("AI's Move:")
    print_board(board)
else:
    print("Game Over! No moves left.")
    break

if check_winner(board):
    print("AI wins!")
    break
if is_draw(board):
    print("It's a draw!")
```

```
        break
```

Screenshot of Output.



```
    if check_winner(board):
        print("AI wins!")
        break
    if is_draw(board):
        print("It's a draw!")
        break
```

```
Welcome to Tic-Tac-Toe!
 | |
 | |
 | |


Enter your move (1-9): 1 2
Invalid move! Please enter a number from 1-9 corresponding to an empty cell.
Enter your move (1-9): 1
Your Move:
O| |
 | |
 | |


AI is making a move...
AI's Move:
O| |
 |X|
 | |


Enter your move (1-9): 2
Your Move:
O|O|
 |X|
 | |


AI is making a move...
AI's Move:
O|O|X
 |X|
 | |


Enter your move (1-9): 5
Cell already occupied. Try again.
Enter your move (1-9): 6
```