TIC TAC TOE REPORT

Tic Tac Toe

Game Implementation

Prepared by: Vanshika Aggarwal

Roll No. : 202401100400207

Algorithm used : Minimax Algorithm

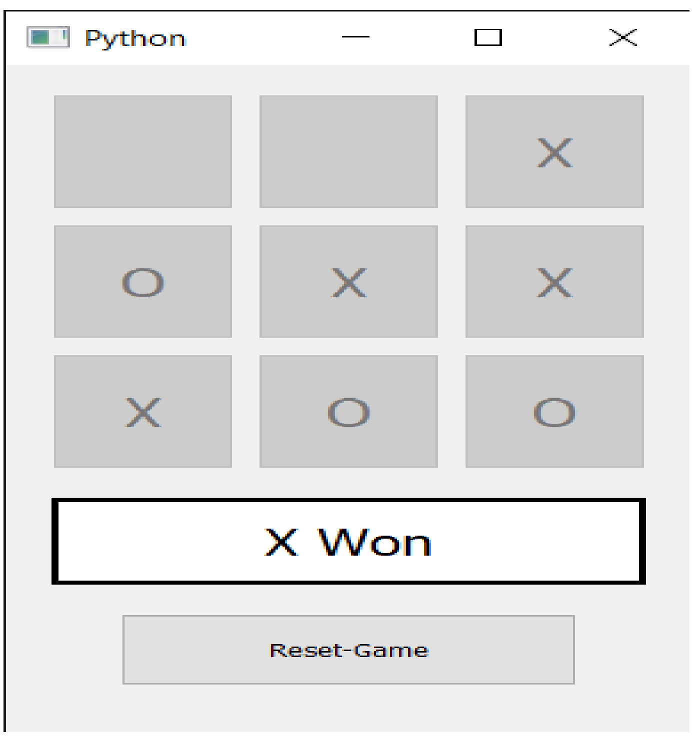
Purpose: This report details the implementation of a Tic Tac Toe game using Python and AI algorithms.

INTRODUCTION

Tic-Tac-Toe is a two-player game where players take turns marking a 3x3 grid with their respective symbols ('O' for the human and 'X' for the AI). The objective is to get three marks in a row, column, or diagonal before the opponent. If all cells are filled without a winner, the game ends in a draw.

This project is a terminal-based Tic-Tac-Toe game, where the AI plays optimally using the Minimax algorithm. The AI always selects the best possible move, ensuring that it never loses. By simulating all potential moves and evaluating their outcomes, the AI strategically plays to win or force a draw if winning is impossible.

This game is an excellent demonstration of game theory, decision-making algorithms, and artificial intelligence in a simple, interactive format. It provides an engaging way to understand how AI can think ahead, anticipate moves, and make optimal choices in competitive settings. Whether you're looking to challenge yourself against a perfect AI opponent or explore the logic behind Minimax, this project serves as a great introduction to AI-powered decision-making in games!



METHODOLOGY

Step 1: Understanding the Game Rules

- The game is played on a **3x3 grid**.
- Two players take turns marking the board with 'O' (human) and 'X' (AI).
- A player wins by placing three of their marks in a row, column, or diagonal.
- If all cells are filled and no one wins, the game ends in a **draw**.

Step 2: Designing the Game Structure

To implement the game, the following core components were identified:

1. **Board Representation** → A 3x3 list to store player moves.
2. **Game Display** → A function to print the board in a readable format.
3. **Move Handling** → Functions to allow user input and AI decision-making.
4. **Win Checking** → A function to determine if a player has won.
5. **AI Decision Making** → Implementation of the **Minimax algorithm**.

Step 3: Implementing the Game Logic

- **Initializing the board** → A 3x3 grid initialized with empty spaces.
- **User input handling** → Validates the user's move and ensures it is within range and not occupied.
- **Win condition checking** → Iterates through rows, columns, and diagonals to determine if a player has won.

Step 4: Implementing the Minimax Algorithm for AI

To create an **unbeatable AI**, the Minimax algorithm was implemented:

1. **Base cases:**
    - If AI wins, return a **positive score**.
    - If the player wins, return a **negative score**.
    - If the board is full, return **zero** (draw).
2. **Recursive exploration:**
    - The AI simulates all possible moves and evaluates their outcomes.
    - It assigns scores based on the **best possible future state**.
    - AI **maximizes** its score, while the human **minimizes** AI's score.

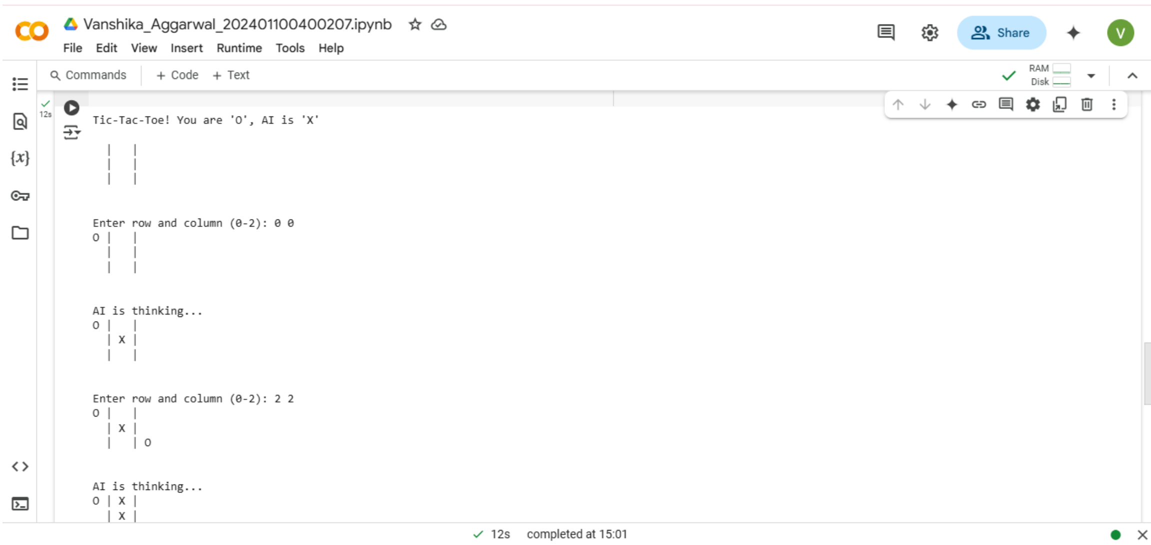
CODE TYPED

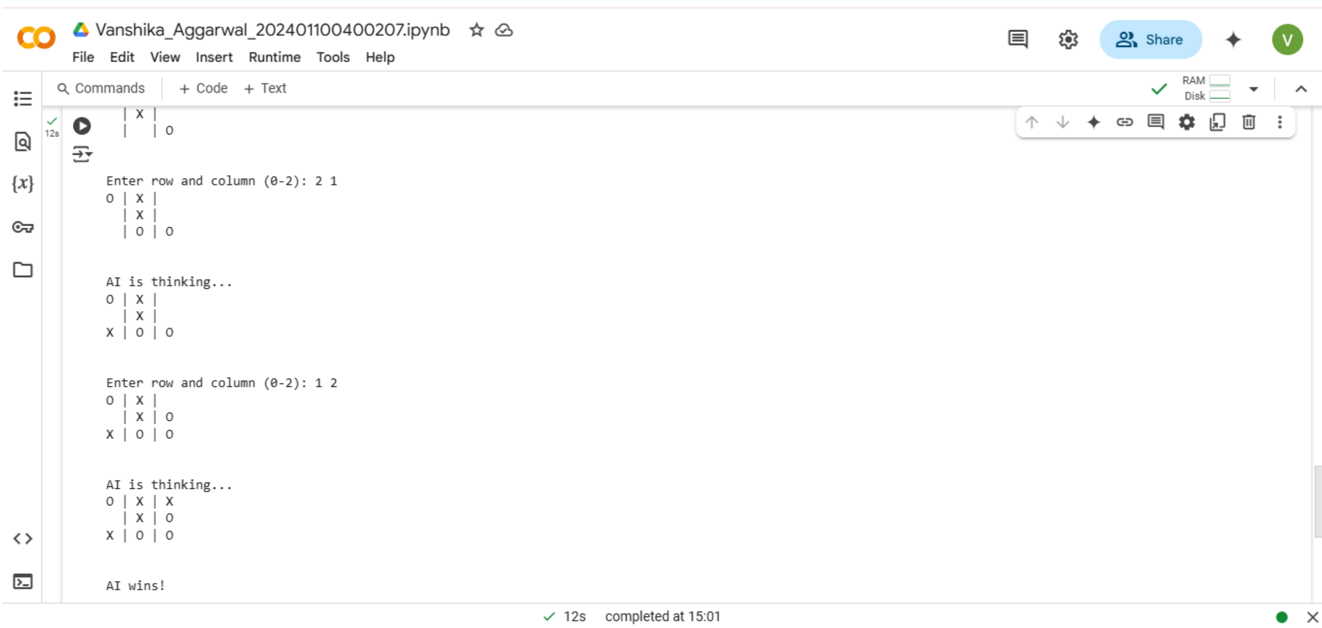
```
import math #math module for infinty value


# Players
user = 'O'
ai = 'X'
empty = ' '

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

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

    # Check columns
    for i in range(3):
        if board[0][i] == player and board[1][i] == player and
board[2][i] == player:
            return True

    # Check main diagonal
    if board[0][0] == player and board[1][1] == player and
board[2][2] == player:
        return True

    # Check secondary diagonal
    if board[0][2] == player and board[1][1] == player and
board[2][0] == player:
        return True

    return False #if no condition becomes true, return false

# Minimax Algorithm
def minimax(board, depth, is_max):
```

```
    if check_winner(board, ai):
      return 10 - depth  # AI wins
    if check_winner(board, user):
      return depth - 10  # Human wins

    draw = True
    for row in board:
        for cell in row:
            if cell == empty:
                draw = False  # Still empty spaces left, game is not a draw

    if draw:
        return 0  # It's a draw

    best_score = -math.inf if is_max else math.inf #if is_max=true then ai turn else user turn
    for i in range(3):
        for j in range(3):
            if board[i][j] == empty:
                board[i][j] = ai if is_max else user
                score = minimax(board, depth + 1, not is_max)
                board[i][j] = empty
                best_score = max(best_score, score) if is_max else min(best_score, score)

    return best_score

# Find the best move for AI
def best_move(board):
    best_score, move = -math.inf, (-1, -1)
    for i in range(3):
        for j in range(3):
            if board[i][j] == empty:
                board[i][j] = ai
                score = minimax(board, 0, False)
                board[i][j] = empty
                if score > best_score:
                    best_score, move = score, (i, j)
    return move

# Game loop
def play_game():
    board = [[empty] * 3 for _ in range(3)]
    print("Tic-Tac-Toe! You are 'O', AI is 'X'\n")
```

```
    print_board(board)

    for turn in range(9):
        if turn % 2 == 0:  # Human move
            while True:
                try:
                    row, col = map(int, input("Enter row and column (0-2): ").split())
                    if board[row][col] == empty:
                        board[row][col] = user
                        break
                    print("Cell occupied! Try again.")
                except:
                    print("Invalid input! Enter numbers between 0 and 2.")
        else:  # AI move
            print("AI is thinking...")
            row, col = best_move(board) #minimax algo for optimal solution
            board[row][col] = ai

        print_board(board)

        if check_winner(board, user): return print("You win! ")
        if check_winner(board, ai): return print("AI wins! ")

    print("It's a draw!")

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

OUTPUT SCREENSHOTS





CONCLUSION

The developmental approach followed an iterative cycle of design, implementation, and testing to create a fully functional **Tic-Tac-Toe AI**. By using the Minimax algorithm, the AI is unbeatable and always makes the best possible move. This project successfully demonstrates AI-driven decision-making, game theory, and strategic planning in a simple yet effective manner.

FUTURE IMPROVEMENTS

- Implement **Alpha-Beta Pruning** to optimize Minimax and improve execution speed.
- Add a Graphical User Interface (GUI) using **Tkinter or Pygame**.
- Extend the game to larger grids (4x4, 5x5) with modified rules.

REFERENCES

1. **Game Theory and Minimax Algorithm:**

- Russell, S., & Norvig, P. (2021). *Artificial Intelligence: A Modern Approach (4th Edition)*. Pearson.

2. **Minimax Algorithm in AI:**

- Michie, D. (1966). *Game-Playing and Artificial Intelligence*. Elsevier.

3. **Online Resources and Documentation:**

- Python Official Documentation: https://docs.python.org/
- Stack Overflow: Discussions on optimizing Minimax in Tic-Tac-Toe.