Tic Tac Toe

For

AI Project(K24MCA18P)
Session (2024-25)

Submitted by

Arun Kumar(202410116100040)
Ayushi Saran Singh (202410116100047)
Akanksha Dwivedi(202410116100012)
Anshu Patel(202410116100034)

Submitted in partial fulfilment of the
Requirements for the Degree of

MASTER OF COMPUTER APPLICATION

Under the Supervision of

Mr. Apoorv Jain

Assistant Professor

Submitted to

Department Of Computer Applications

KIET Group of Institutions, Ghaziabad

Uttar Pradesh-201206

(April- 2025)

Table of Contents

1. Introduction

1.1 Overview of Tic Tac Toe

Tic Tac Toe is a well-known two-player strategy game that is played on a **3x3 grid**. The game consists of two players, where one player marks the board with ‘X’ and the other with ‘O’. The players take alternate turns to place their mark in an empty cell, and the primary objective is to **align three of their marks in a row, column, or diagonal** before the opponent does.

If a player successfully aligns three identical marks in any of these directions, they win the game. However, if all **nine cells are filled** and no player meets the winning condition, the game ends in a **draw**. Tic Tac Toe is one of the simplest games to understand and play but has been widely studied in game theory due to its strategic elements.

Why Tic Tac Toe is an Important Game for AI?

Despite its simple rules, Tic Tac Toe has served as a fundamental example in artificial intelligence and game development. Since the game has a **limited number of possible moves**, it allows researchers and developers to implement various AI techniques for decision-making and strategy optimization.
Tic Tac Toe has the following key properties:

1. **Deterministic Game :** There is no involvement of chance (like dice rolls). The game progresses solely based on the players’ decisions.
2. **Perfect Information :** Both players have full knowledge of the board state at any given time.
3. **Zero-Sum Game :** One player’s gain results in the other player’s loss, making it a competitive game where one wins or it results in a draw.

These properties make Tic Tac Toe an ideal example for **AI-driven decision-making** using algorithms such as **Minimax**, which helps the computer determine the best possible moves in a game.

1.2 Importance of AI in Games

Artificial Intelligence has become an essential aspect of modern gaming, providing players with **challenging and adaptive opponents**. Instead of relying on pre-defined moves, AI-powered opponents can analyze the game environment and make strategic decisions based on logic and prediction.

In **Tic Tac Toe**, AI plays a crucial role by ensuring that the computer always makes the best possible move, eliminating random or weak decisions. This enhances the gaming experience by making the AI a challenging opponent.

How AI Enhances Gaming?

1. **Provides Challenging Gameplay :** AI prevents the game from becoming too easy by making strategic decisions instead of random moves.
2. **Encourages Strategic Thinking :** Players must think ahead and plan their moves since the AI always chooses the optimal move.
3. **Allows Solo Gameplay :** Unlike board games that require two players, AI enables a **single-player mode** where a user can play against the computer.

Real-World Applications of AI in Games

Apart from Tic Tac Toe, AI is used in various modern games to provide intelligent opponents and improve user engagement. Some examples include:

- ➢ **Chess Engines :** AI-driven chess programs like **Stockfish** and **AlphaZero** analyze millions of possible moves per second.
- ➢ **Pathfinding in Video Games :** AI helps characters navigate game environments efficiently in open-world games like **GTA** and **Assassin’s Creed**.
- ➢ **Adaptive Game Difficulty :** AI adjusts the difficulty level based on a player's skill, ensuring a balanced gaming experience.

Thus, AI is a fundamental technology in game development, providing both **engagement and strategic depth** to players.

1.3 Role of Minimax Algorithm in AI-Powered Tic Tac Toe

The Minimax Algorithm is one of the most powerful AI techniques used in turn-based games. It allows the AI to evaluate all possible board states and determine the best move to either win the game or prevent a loss.

How Does Minimax Work?

1. **Game Tree Exploration** : The AI generates all possible future moves and their outcomes.
2. **Score Assignment :**
   - ➢ If the AI wins, it assigns a +10 score.
   - ➢ If the AI loses, it assigns a -10 score.
   - ➢ If it results in a draw, it assigns a 0 score.
3. **Optimal Decision Selection :** The AI chooses the move that minimizes the opponent’s chances of winning and maximizes its own winning potential.

Example of Minimax in Tic Tac Toe

Consider a Tic Tac Toe board where it is the AI's turn :

```
  X | O | X
-------------
  O | X | O
-------------
 __ | __ | __
```

1. The AI can simulate different moves and evaluate whether they lead to a win, loss, or draw.
2. The algorithm calculates the best move that will ensure either a win or force a draw if winning is not possible.

Why is Minimax Important?

- ➢ Ensures optimal decision-making in games.
- ➢ Prevents AI from making weak or random moves.
- ➢ Guarantees that the AI will never lose, making it a perfect challenge for human players.

1.4 Objectives

The main objectives of this project are:

- To develop an AI-powered Tic Tac Toe game that provides an optimal playing experience for human users.
- To implement the Minimax algorithm for decision-making to ensure AI always makes the best possible move.
- To enhance the understanding of AI algorithms through practical game implementation.
- To create an interactive and user-friendly interface using Python and Tkinter.
- To demonstrate the application of game theory in AI-powered decision-making.

1.5 Significance

The importance of this project extends beyond just building a simple game. It highlights the power of AI in:

- **Educational Value :** This project serves as an excellent learning tool for students and developers interested in AI and game development.
- **AI Decision-Making :** It highlights how AI can analyze multiple game scenarios to determine the best course of action.
- **Strategic Thinking :** Playing against AI helps users improve their strategic planning and critical thinking skills.
- **Efficient Problem Solving :** By incorporating Alpha-Beta Pruning, the project demonstrates how optimization techniques improve computational efficiency.
- **Real-World Relevance :** AI-based decision-making is widely used in modern applications, and understanding it through a simple game like Tic Tac Toe makes it more accessible..

2. Methodology

The project follows a structured approach to designing and implementing the AI-based Tic Tac Toe game. The methodology includes game design, AI implementation using the Minimax algorithm, and optimization through Alpha-Beta Pruning.

2.1 Understanding Tic Tac Toe Game Logic

Tic Tac Toe is a simple yet strategically deep game played on a 3x3 grid. The game follows strict rules that dictate how turns are taken and how the game concludes. The rules include:

- Two players take turns placing their respective symbols (‘X’ or ‘O’) on an empty square.
- A player wins when they manage to align three of their marks consecutively, either horizontally, vertically, or diagonally.
- If the grid is completely filled and no player has formed a line, the game ends in a draw.
- Since Tic Tac Toe has a finite number of moves, it is a game that can be solved mathematically, making it an excellent candidate for AI-based strategies.

2.2 Explanation of Minimax Algorithm

The Minimax algorithm is a powerful decision-making tool used in turn-based games to determine the best move for an AI opponent. It works by simulating every possible move, evaluating their outcomes, and selecting the most favorable one.

i) Game Tree Exploration

- The Minimax algorithm generates a decision tree consisting of all possible board configurations that can result from the current game state.
- It expands the tree recursively, considering all potential moves for both the AI and the human player.
- Each node in the tree represents a possible game state, and the algorithm assigns a score to each node :

    - +10 if the AI wins.
    - -10 if the AI loses.
    - 0 if the game results in a draw.
- The AI evaluates all possible paths and selects the move that leads to the best outcome while anticipating the opponent's best responses.

ii) Evaluation Function

- The evaluation function is used to assign numerical values to each possible game state.
- The function checks for winning conditions and assigns scores accordingly.
- By analyzing different board configurations, the AI determines the likelihood of winning, losing, or drawing for each move.

iii) Alpha-Beta Pruning (Optimization)

- Minimax can be computationally expensive, especially in complex games with large decision trees.
- Alpha-Beta Pruning enhances efficiency by eliminating branches of the tree that do not need to be evaluated.
- If one move is already found to be worse than a previously considered move, the algorithm stops evaluating that branch, reducing computation time.
- This optimization allows the AI to make quick decisions without compromising accuracy.

2.3 AI Decision-Making Process

1. **Analyzing the Board :** The AI scans the board to identify available moves.
2. **Generating Possible Moves :** It explores all potential moves and their outcomes.
3. **Evaluating Board States :** The AI assigns scores based on winning or losing conditions.
4. **Selecting the Best Move :** The move with the highest score is chosen to ensure an optimal game strategy.
5. **Executing the Move :** The selected move is applied to the game board, and the cycle repeats until the game ends.

2.4 Game Flow Diagram

Below is the structured flow of the AI Tic Tac Toe game:

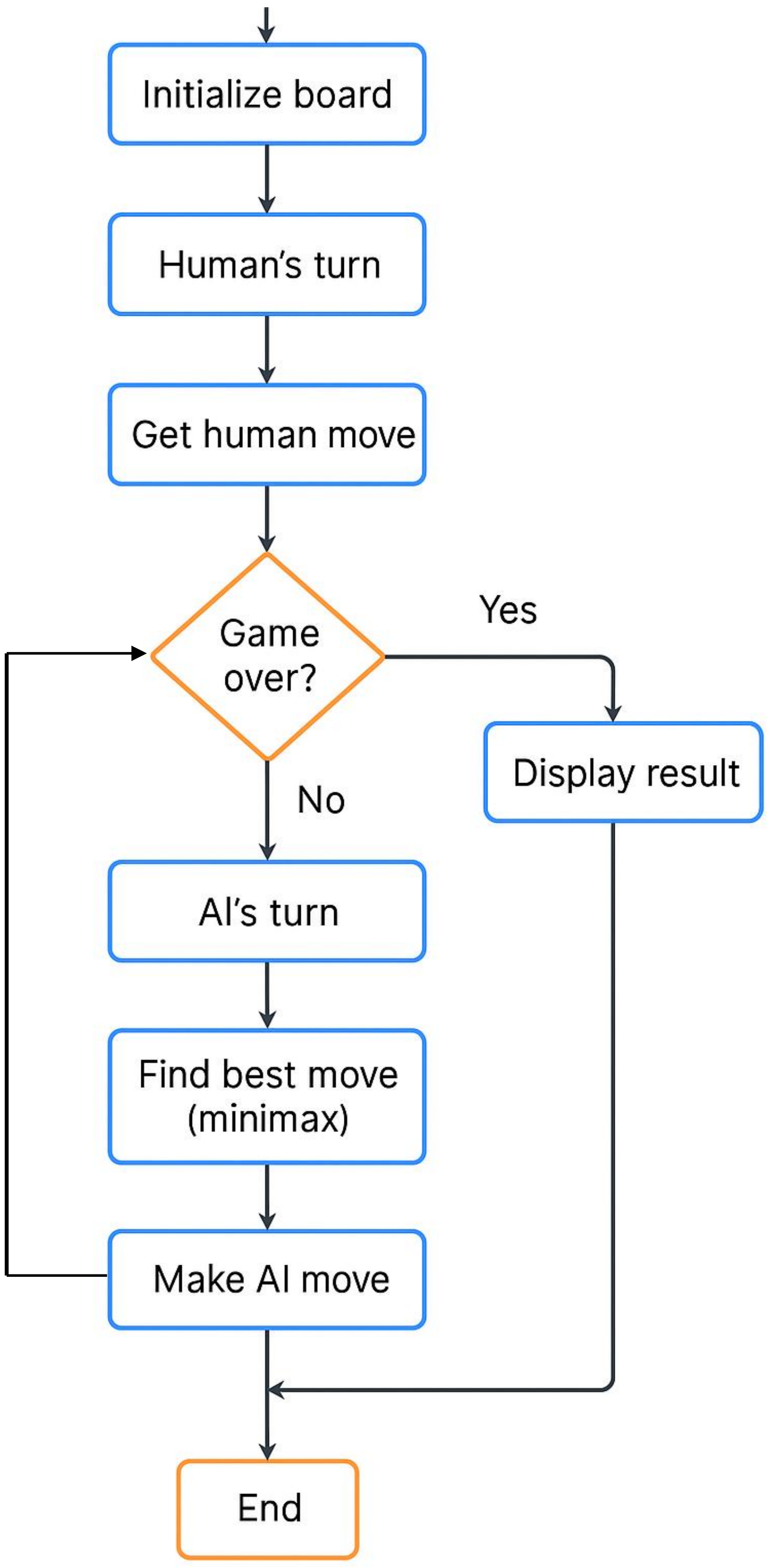
1. **Game Initialization:**
    1. Display game mode selection (Player vs AI or Player vs Player).
    2. Assign symbols (‘X’ or ‘O’) to players.
    3. Create an empty 3x3 board.

2. **Game Loop Execution:**
    1. **Step 1 :** Display the current board state.
    2. **Step 2 :** Check if the game has ended (Win, Loss, or Draw).
    3. **Step 3 :** If the game is still ongoing, the current player makes a move.
        1. If it’s the human player’s turn, take input for row and column selection.
        2. If it’s AI’s turn, determine the best move using Minimax.
        3. Update the board with the selected move.
    4. **Step 4 :** Switch turns to the next player.
    5. **Step 5 :** Repeat the loop until a win or draw condition is met.

3. **Game Termination:**
    1. If a player wins, display the winner.
    2. If all cells are filled without a winner, declare a draw.
    3. Prompt for a new game or exit.

This structured methodology ensures that the AI operates optimally, making Tic Tac Toe both engaging and challenging.

3. Flowchart of AI Tic-Tac-Toe



4. Code Implementation

4.1 Importing Necessary Libraries

The game is implemented in Python, utilizing standard libraries such as math and random. These libraries help in numerical calculations and randomization for assigning player turns.

```
import math

import random
```

4.2 Creating the Tic Tac Toe Board

A 3x3 board is initialized as a nested list where empty spaces are represented by ' ' (spaces). This structure allows efficient storage and manipulation of the game state.

```
def print_board(board):
for row in board:
print(" | ".join(row))
print("\n")
```

4.3 Minimax Algorithm for AI Decision Making

The Minimax algorithm is implemented with recursion to explore all possible game states. It evaluates the best move for the AI by maximizing its chances of winning while minimizing the player's advantage. An evaluation function assigns scores to different board states, and the algorithm selects the move with the best outcome.

```
def minimax(board, depth, is_max):
score = evaluate(board)
if score == 10:
return score - depth # Favor faster wins
if score == -10:
return score + depth # Favor slower wins
```

```
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))
                board[i][j] = ' ‘
    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))
                board[i][j] = ' ‘
    return best
```

4.4 AI Move Implementation

The find_best_move() function identifies the optimal move for the AI by iterating through all available positions and applying the Minimax algorithm. The best move is then executed on the board.

```
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)
            board[i][j] = ' ‘
            if move_val > best_val:
               best_move = (i, j)
               best_val = move_val
   return best_move
```

4.5 GUI Updates and User Interaction

In a text-based version, player moves are taken as input, validated, and executed. If a GUI is integrated using Tkinter, visual board updates are reflected in real-time.

```
def play_game():
   board = [[' ' for _ in range(3)] for _ in range(3)]
   current_player = 0
   while True:
      print_board(board)
      if not is_moves_left(board) or evaluate(board) != 0:
         break
      if current_player == 1:
         ai_move= find_best_move(board)
         board[ai_move[0]][ai_move[1]] = 'X’
      else:
         try:
            row, col = map(int, input("Enter row and column (0-2): ").split())
            if board[row][col] == ' ‘:
               board[row][col] = 'O’
            else:
               print("Invalid move! Try again.")
               continue
         except ValueError:
```

```
print("Invalid input! Please enter numbers between 0 and 2.")
            continue
      current_player = 1 - current_player
   print_board(board)
```

3.6 Checking for a Winner

A function evaluates rows, columns, and diagonals to determine if a player has won or if the game has ended in a draw. The result is displayed accordingly.

```
def evaluate(board):
   for i in range(3):
      if board[i][0] == board[i][1] == board[i][2] != ' ‘:
         return 10 if board[i][0] == 'X' else -10
      if board[0][i] == board[1][i] == board[2][i] != ' ‘:
         return 10
 if board[0][i] == 'X’
 else -10
   if board[0][0] == board[1][1] == board[2][2] != ' ‘:
      return 10
 if board[0][0] == 'X’
 else -10
   if board[0][2] == board[1][1] == board[2][0] != ' ‘:
      return 10
 if board[0][2] == 'X’
 else -10
   return 0This
```

This section provides a structured approach to implementing the AI Tic Tac Toe game.

5. Output Explanation

When the game starts, the user is prompted to choose between two modes:

1.Play against AI

2.Play against another player

5.1 Game Play Vs AI

If AI mode is selected, the user provides their name, and the AI takes the second player's position. If the two-player mode is selected, both players enter their names, and the game randomly assigns 'X' and 'O’

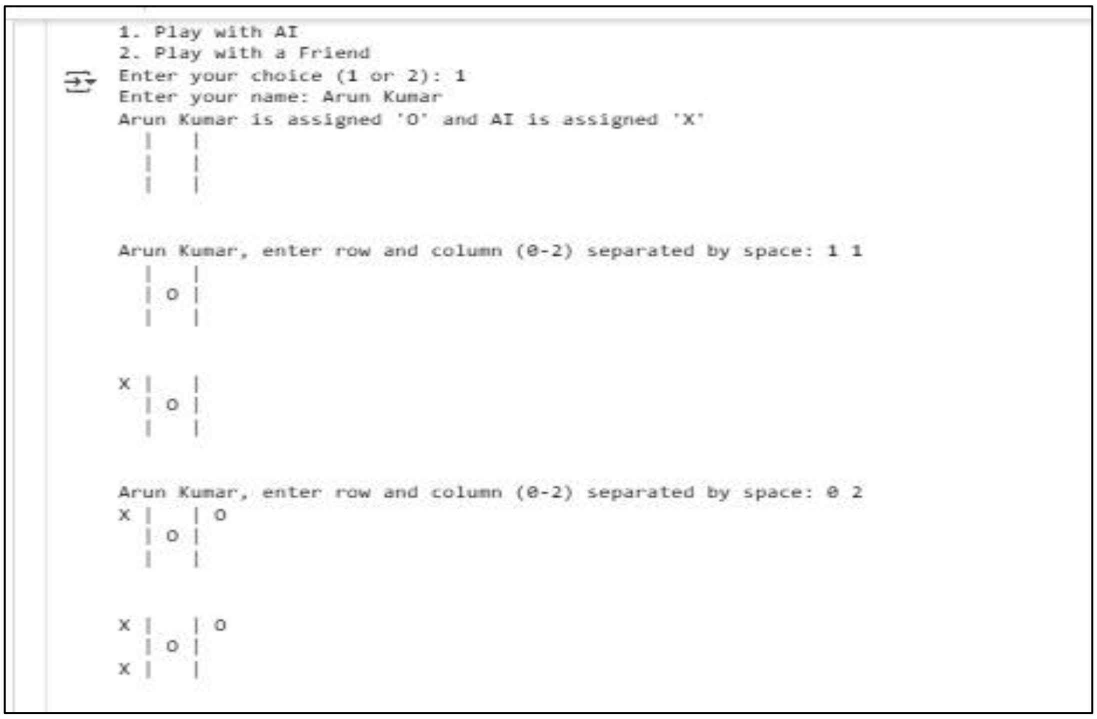
The game displays the 3x3 board after every move. The player and AI take turns making moves, and the board updates accordingly. If Arun Kumar (O) selects row 1, column 1, the board updates: The AI (X) calculates the best move and places its mark , and so on the game continues by calculating the best possible moves.

The game ends under three conditions:

1.A player wins – The game announces the winner if three of their marks align horizontally, vertically, or diagonally.

2.A draw – If no spaces are left and no winner is found, the game declares a draw.

3.Game loop ends – The final board is displayed along with the game result



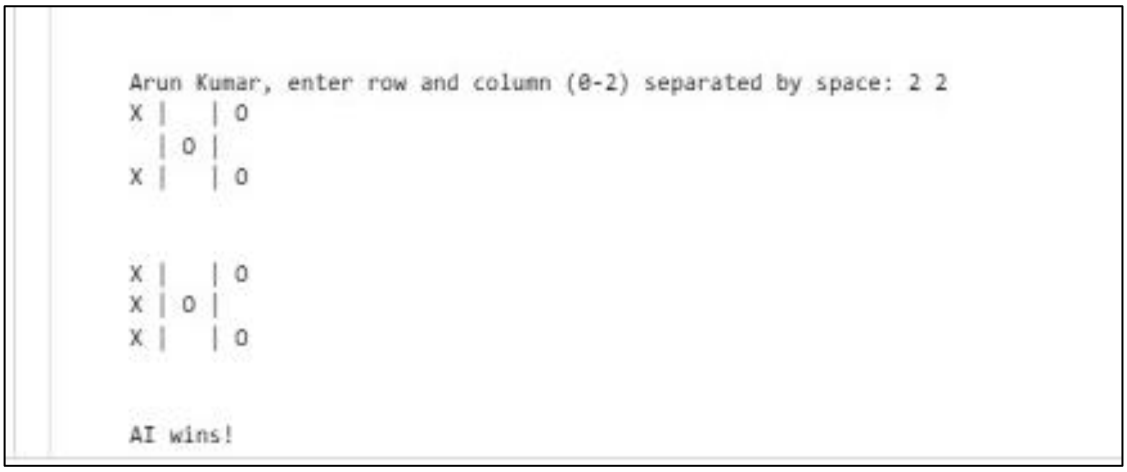
```
1. Play with AI
2. Play with a Friend
Enter your choice (1 or 2): 1
Enter your name: Arun Kumar
Arun Kumar is assigned 'O' and AI is assigned 'X'
  |   |
  |   |
  |   |


Arun Kumar, enter row and column (0-2) separated by space: 1 1
  |   |
  | O |
  |   |


X |   |
  | O |
  |   |


Arun Kumar, enter row and column (0-2) separated by space: 0 2
X |   | O
  | O |
  |   |


X |   | O
  | O |
X |   |
```



```
Arun Kumar, enter row and column (0-2) separated by space: 2 2
X |   | O
  | O |
X |   | O


X |   | O
X | O |
X |   | O


AI wins!
```

5.2 Game Play With A Friend

If AI mode is selected, the user provides their name, and the AI takes the second player's position. If the two-player mode is selected, both players enter their names, and the game randomly assigns 'X' and ‘O’

1. Game Start:

- ➢ The game board is displayed as empty (3x3 grid).
- ➢ Arun Kumar is assigned 'X' and Singh is assigned ‘O’.

2. Players Take Turns:

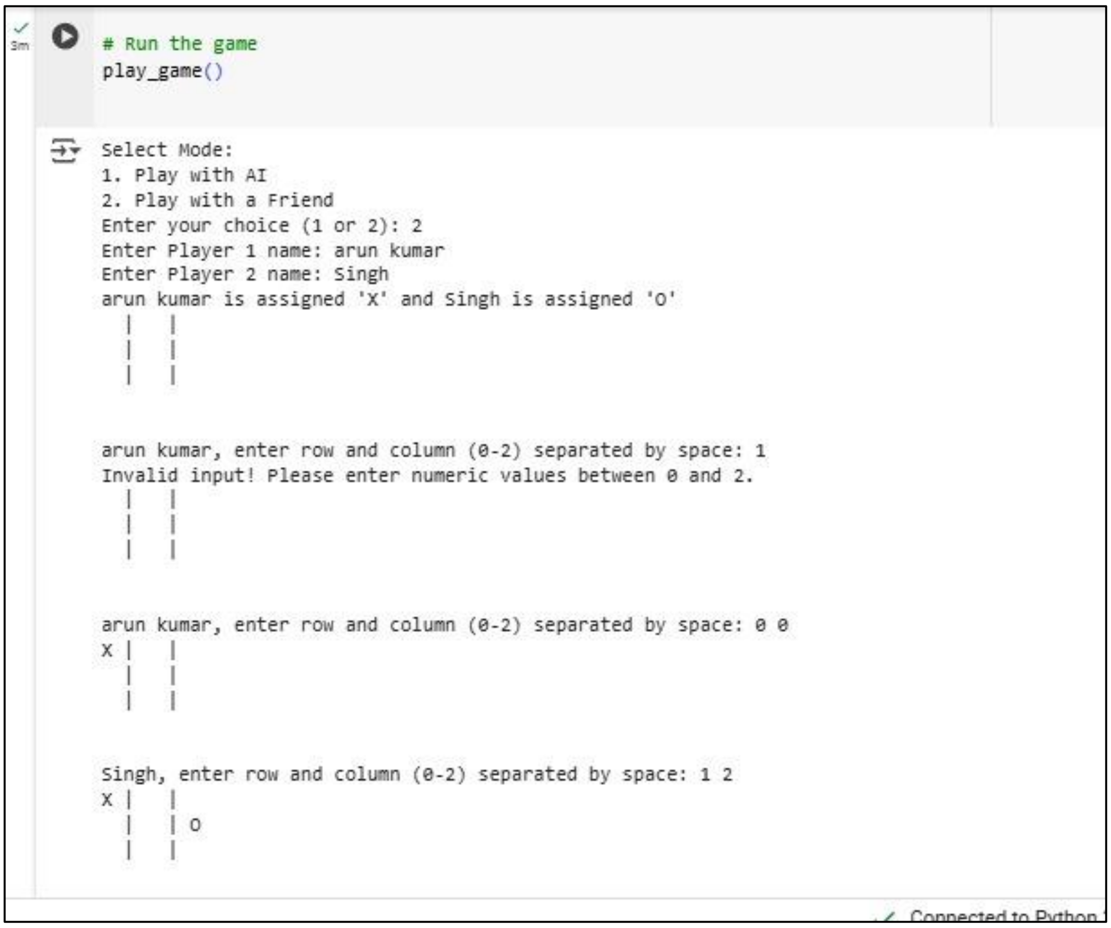
- ➢ Arun Kumar (X) makes the first move by choosing a position (e.g., (0,0)).
- ➢ Singh (O) makes the next move by selecting an empty position(e.g., (1,2)).
- ➢ Players continue alternating moves until a win condition is met or the board is full.

3. Winning Scenarios:

- ➢ If Arun Kumar successfully aligns three 'X' marks in a row, column, or diagonal, he wins.
- ➢ If Singh aligns three 'O' marks in a row, column, or diagonal, he wins.
- ➢ If all spaces are filled without a winner, the game is declared a draw.

4. Game Conclusion:

- ➢ The final board state is displayed.
- ➢ A message is printed to announce the winner (Arun Kumar or Singh) or a draw.



```
# Run the game
play_game()
```

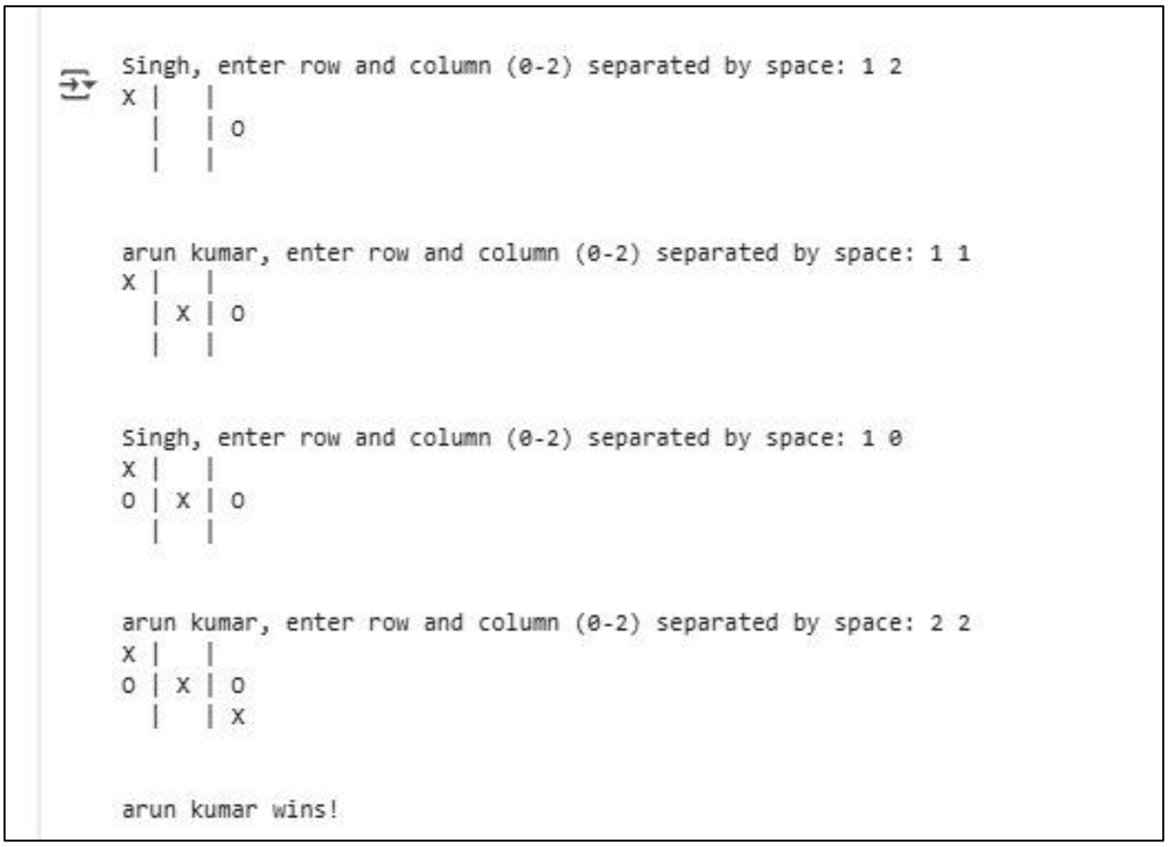
```
Select Mode:
1. Play with AI
2. Play with a Friend
Enter your choice (1 or 2): 2
Enter Player 1 name: arun kumar
Enter Player 2 name: Singh
arun kumar is assigned 'X' and Singh is assigned 'O'
  |   |
  |   |
  |   |


arun kumar, enter row and column (0-2) separated by space: 1
Invalid input! Please enter numeric values between 0 and 2.
  |   |
  |   |
  |   |


arun kumar, enter row and column (0-2) separated by space: 0 0
X |   |
  |   |
  |   |


Singh, enter row and column (0-2) separated by space: 1 2
X |   |
  |   | O
  |   |
```

✓ Connected to Python



```
Singh, enter row and column (0-2) separated by space: 1 2
X |   |
  |   | O
  |   |


arun kumar, enter row and column (0-2) separated by space: 1 1
X |   |
  | X | O
  |   |


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


arun kumar, enter row and column (0-2) separated by space: 2 2
X |   |
O | X | O
  |   | X


arun kumar wins!
```

6. Conclusion

This AI-powered Tic Tac Toe game successfully demonstrates the implementation of the Minimax algorithm in game development. By analyzing all possible board states, the AI effectively determines optimal moves, providing a challenging opponent for human players. The game supports both player-vs-player and player-vs-AI modes, ensuring a versatile and engaging experience.

Strengths:

•Implements an optimal AI opponent using Minimax.
•Provides both AI and multiplayer modes.
•Ensures a fair gameplay experience with logical move selection.
•Simple and efficient implementation with easy-to-understand logic.

Limitations:

•The game is text-based; a graphical interface could enhance user experience.
•The AI currently does not use machine learning, which could further improve strategic adaptability.
•Minimax performance decreases for larger board sizes due to increased computation.

Future Improvements:

•Implement a GUI for better visualization.
•Use reinforcement learning for adaptive AI strategies.
•Extend the game to larger grid sizes for added complexity.
•Add difficulty levels to make the AI adjustable for different skill levels.
•Introduce an online multiplayer feature for remote gameplay.

This project highlights the potential of AI in classic games and lays the foundation for future improvements in game development. By integrating artificial intelligence techniques, it showcases the impact of algorithmic decision-making in interactive applications. Future advancements will continue to enhance the game's adaptability, accessibility, and overall user experience.

7. References

1.Russell, S. J., & Norvig, P. (2020).

Artificial Intelligence: A Modern Approach (4th ed.). Pearson.

– A comprehensive book on AI techniques, including search algorithms and game theory.

2.Nilsson, N. J. (1998).

Artificial Intelligence: A New Synthesis. Morgan Kaufmann.

– Covers foundational AI algorithms and their applications.

3.Rich, E., Knight, K., & Nair, S. B. (2017).

Artificial Intelligence. McGraw Hill Education.

– A popular AI textbook in Indian academics; good for foundational concepts.

4.Knuth, D. E., & Moore, R. W. (1975).

"An analysis of alpha-beta pruning." *Artificial Intelligence*, 6(4), 293–326.

– Focuses on optimization techniques used in Minimax algorithms.

5.Schaeffer, J. (2001).

"A gamut of games." *AI Magazine*, 22(3), 29–46.

– Discusses AI implementations in classic games.

6.GeeksforGeeks. (n.d.).

Minimax Algorithm in Game Theory | Set 1 (Introduction)

– Easy-to-understand explanation of Minimax with code examples.

7.Towards Data Science. (n.d.).

Creating AI for Tic Tac Toe using Minimax

– Practical implementation tutorial for Minimax in Python.