**1. Introduction**

The Tic-Tac-Toe game, also known as Noughts and Crosses, is a classic two-player board game typically played on a 3x3 grid. Each player alternates placing their mark (either "X" or "O") on the grid until one player achieves three marks in a row, column, or diagonal. The game ends when one player wins or all available spaces are filled, resulting in a draw.

This project aims to create a simplified version of the Tic-Tac-Toe game, which is implemented using Python. The game allows two players to interact with the game board through a command-line interface. The goal is to demonstrate basic game logic and user interaction, along with features such as input validation, turn-based gameplay, and detection of winning conditions.

**2. Abstract**

Tic-Tac-Toe is a simple yet engaging game that can be easily implemented in Python. This project involves the development of a text-based Tic-Tac-Toe game where two players take turns choosing a spot on the 3x3 grid. The game detects winning positions by checking for three consecutive marks in rows, columns, or diagonals. Input validation ensures that the players can only select available positions on the grid. Once a player wins or all cells are filled, the game either announces a winner or a draw. This project is an ideal introduction to implementing basic game logic and user interfaces in Python, offering a practical approach to problem-solving.

**3. Problem Statement**

Despite being simple, the Tic-Tac-Toe game presents several challenges for implementation, especially in handling user inputs, ensuring game logic is accurate, and providing clear feedback for the players. Key problems in this project include:

* **Input Validation:** Ensuring that the players enter valid moves within the bounds of the 3x3 grid, and avoiding moves on already filled spots.
* **Detecting Winning Conditions:** Identifying when a player has won the game based on specific winning positions.
* **Managing Turn-based Gameplay:** Alternating turns between players "X" and "O" and displaying the current state of the game after each move.
* **Handling Draws:** Ensuring that a draw is correctly detected if all spots are filled without a winner.

These challenges require a clean, efficient solution to ensure that the game runs smoothly and provides meaningful feedback to the players.

**4. Proposed System**

The proposed system will consist of the following key components:

1. **Game Board:** A 3x3 grid that is represented as a list in Python. Each cell in the grid can either be a number (for an empty cell), an "X", or an "O", representing the players' moves.
2. **Input Handling:** The game will prompt players to enter their moves by specifying a number (1-9) corresponding to the positions on the grid. The system will validate the input to ensure it corresponds to an unoccupied cell.
3. **Game Loop:** The game will alternate between two players, allowing them to input their choices and updating the board after each move. The loop will continue until there is a winner or a draw.
4. **Winning Conditions:** The game will check after every move whether any player has won by matching positions according to pre-defined winning patterns (rows, columns, and diagonals).
5. **Endgame Conditions:** If a player wins, the system will display a message announcing the winner. If all spots are filled without a winner, the game will announce a draw.
6. **User Interface:** A simple command-line interface will be used to interact with the game, displaying the board after each turn and prompting players for their input.

The system will be implemented in Python, with a focus on simplicity and ease of use for two players.

**5. System Requirements**

**Software Requirements:**

* **Programming Language:** Python 3.x
* **IDE/Code Editor:** Any text editor (e.g., Visual Studio Code, Sublime Text, PyCharm) or Python's built-in IDLE.
* **Operating System:** Cross-platform (Windows, macOS, Linux)

**Hardware Requirements:**

* **Processor:** 1 GHz or higher
* **RAM:** 2 GB or higher
* **Storage:** 50 MB free disk space

**Functional Requirements:**

1. **Game Board Display:** The game must display the current state of the grid after every move.
2. **Player Input:** Players must be able to input a number between 1 and 9 to place their mark on the grid.
3. **Turn-based Gameplay:** The game must alternate between two players ("X" and "O").
4. **Input Validation:** The game must ensure that players input valid positions and handle invalid inputs gracefully.
5. **Win Detection:** The game must detect if a player has achieved a winning condition and announce the winner.
6. **Draw Detection:** If the grid is full without a winner, the game must announce a draw.

**Non-Functional Requirements:**

1. **Usability:** The game should have a simple, easy-to-understand interface for players.
2. **Reliability:** The game must function correctly without errors, ensuring smooth gameplay.
3. **Performance:** The game should perform all operations efficiently without noticeable delays.

Code:

# Tuple to store winning positions.

win\_positions = (

(0, 1, 2), (3, 4, 5), (6, 7, 8),

(0, 3, 6), (1, 4, 7), (2, 5, 8),

(0, 4, 8), (2, 4, 6)

)

def game(player):

# diplay current mesh

print("\n", " | ".join(mesh[:3]))

print("---+---+---")

print("", " | ".join(mesh[3:6]))

print("---+---+---")

print("", " | ".join(mesh[6:]))

# Loop until player valid input cell number.

- while True:

try:

ch = int(input(f"Enter player {player}'s choice : "))

if str(ch) not in mesh:

raise ValueError

mesh[ch-1] = player

break

except ValueError:

print("Invalid position number.")

# Return winning positions if player wins, else None.

for wp in win\_positions:

if all(mesh[pos] == player for pos in wp):

return wp

return None

player1 = "X"

player2 = "O"

player = player1

mesh = list("123456789")

for i in range(9):

won = game(player)

if won:

print("\n", " | ".join(mesh[:3]))

print("---+---+---")

print("", " | ".join(mesh[3:6]))

print("---+---+---")

print("", " | ".join(mesh[6:]))

print(f"\*\*\* Player {player} won! \*\*\*")

break

player = player1 if player == player2 else player2

else:

# 9 moves without a win is a draw.

print("Game ends in a draw.")