**Tic-Tac-Toe Game in Python: A Comprehensive Guide**

This document serves as a comprehensive guide to creating a classic Tic-Tac-Toe game using Python. We'll explore the essential code components, logic behind game functionality, and illustrate how to play the game through program outputs. This guide is suitable for beginner programmers aiming to understand Python fundamentals and learn the logic behind game development.

**Problem Statement**

Our objective is to develop a simple yet functional Tic-Tac-Toe game in Python. The game should adhere to the standard rules, allowing two players to take turns marking their symbols (X or O) on a 3x3 board. The game continues until one player achieves a winning condition by placing three of their symbols in a row, column, or diagonal, or until the board is full, resulting in a draw.

**Introduction**

Tic-Tac-Toe is a classic game of strategy and logic. The game involves two players, each taking turns marking their symbol (X or O) on a 3x3 board. The goal is to be the first player to get three of their symbols in a row, column, or diagonal. If all nine squares are filled without either player achieving a winning condition, the game ends in a draw.

This guide will walk you through the Python code that implements this game, explaining the key concepts and logic behind each component. The code provided is well-commented, making it easier to understand the code flow and logic.

**Technology - Package Used in Project**

The primary technology employed in this Tic-Tac-Toe game is Python, a widely used, versatile, and beginner-friendly programming language. Python's simplicity and readability make it ideal for learning basic programming concepts, and it provides various built-in libraries that simplify game development.

While this project primarily uses core Python, we don't rely on any external libraries or packages. The game's functionality is entirely based on native Python features, making it a great example of building a game from scratch.

**Program Code**

def display\_board(board):

"""Prints the current state of the Tic-Tac-Toe board."""

for row in board:

print("|", " | ".join(str(x) for x in row), "|")

def is\_valid\_move(board, move):

"""Checks if a given move is valid on the current board."""

row, col = move // 3, move % 3

return 0 <= row < 3 and 0 <= col < 3 and board[row][col] == " "

def check\_win(board, player):

"""Checks if the given player has won the game."""

win\_conditions = [[(0, 0), (0, 1), (0, 2)], [(1, 0), (1, 1), (1, 2)], [(2, 0), (2, 1), (2, 2)],

[(0, 0), (1, 0), (2, 0)], [(0, 1), (1, 1), (2, 1)], [(0, 2), (1, 2), (2, 2)],

[(0, 0), (1, 1), (2, 2)], [(0, 2), (1, 1), (2, 0)]]

for condition in win\_conditions:

if all(board[row][col] == player for row, col in condition):

return True

return False

def is\_board\_full(board):

"""Checks if the board is full."""

for row in board:

for cell in row:

if cell == " ":

return False

return True

def main():

board = [[" " for \_ in range(3)] for \_ in range(3)]

current\_player = "X"

while True:

display\_board(board)

print(f"Player {current\_player}'s turn.")

move = int(input("Enter your move (1-9): ")) - 1

if not is\_valid\_move(board, move):

print("Invalid move. Try again.")

continue

row, col = move // 3, move % 3

board[row][col] = current\_player

if check\_win(board, current\_player):

display\_board(board)

print(f"Player {current\_player} wins!")

break

if is\_board\_full(board):

display\_board(board)

print("It's a draw!")

break

current\_player = "O" if current\_player == "X" else "X"

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Program Outputs**

When the code is executed, the game begins. The board is displayed, and players are prompted to enter their moves by typing a number from 1 to 9, corresponding to the position on the board.

Here's a sample gameplay scenario:

``` | | | | |---|---|---| | | | | Player X's turn. Enter your move (1-9): 1 | X | | | |---|---|---| | | | | Player O's turn. Enter your move (1-9): 5 | X | | | |---|---|---| | | O | | ... Player X's turn. Enter your move (1-9): 9 | X | | X | |---|---|---| | | O | X | Player X wins! ```

**Game Logic Explanation**

The program code comprises several functions, each responsible for a specific aspect of the Tic-Tac-Toe game's logic. Let's break down these functions and their roles:

* \*\*display\_board(board):\*\* This function takes the current state of the game board as input and neatly prints it on the screen, displaying the symbols (X, O, or blank spaces) in their respective positions.
* \*\*is\_valid\_move(board, move):\*\* This function verifies if the player's chosen move (a number from 1 to 9) is a valid move based on the current state of the board. It checks if the move is within the bounds of the 3x3 board and if the target square is empty (not already filled with X or O).
* \*\*check\_win(board, player):\*\* This function analyzes the board to determine if the specified player has won the game. It does this by checking all possible winning combinations (rows, columns, and diagonals) to see if the player's symbol occupies all positions within any of those combinations.
* \*\*is\_board\_full(board):\*\* This function checks if the game board is completely filled with symbols. It iterates through all the squares on the board, looking for any empty squares (" "). If all squares are filled, the game is considered a draw.
* \*\*main():\*\* This function serves as the main entry point for the game. It initializes the game board, sets the current player to X, and then enters a loop that continues until either a player wins or the board is full. Inside the loop, it displays the board, prompts the player for their move, validates the move, updates the board with the player's symbol, checks for a win, and switches to the next player.

**Conclusion**

By implementing a Tic-Tac-Toe game in Python, we gain valuable insights into basic game development concepts. This project demonstrates how to handle user input, manage game state, and implement basic game logic using functions. The code is well-structured and commented, providing a solid foundation for learning more complex game development techniques in Python.

This project provides a great starting point for exploring further game development endeavors in Python. You can experiment with different game mechanics, design more intricate game boards, incorporate graphics, and even explore multiplayer capabilities using network communication.

**TEAM**

**KAILESH MURUGAN**

**KALYAN KUMAR R**

**KAMAL BASHEER M**

**KARTHIKEYAN S**