Building a Tic-Tac-Toe Game with Pygame

- ***Sathvik N Shendige***

Introduction:

The provided code is a simple implementation of the classic game Tic-Tac-Toe using the Pygame library. In this case study, we will examine the project in detail, discussing its structure, functionality, and the important aspects of its implementation.

Project Overview:

Tic-Tac-Toe is a two-player game played on a 3x3 grid where the goal is to form a line of three of your own marks (either 'X' or 'O') horizontally, vertically, or diagonally. This project provides a graphical interface for playing Tic-Tac-Toe with a friend on the same computer.

Key Components and Concepts:

1. Pygame:

→ Pygame is a popular Python library for creating 2D games and multimedia applications. It simplifies tasks like handling user input, graphics rendering, and event management.

2. Constants and Colors:

→ The project defines various constants for configuring the game's appearance, such as the window size, line width, board size, and colors for the grid, 'X,' and 'O.'

3. Game Board:

→ The game board is represented as a 2D list named 'board,' where each cell can have one of three values: 'X,' 'O,' or ' ' (empty).

4. Functions:

- The project defines several functions:
- `draw_grid()`: Draws the grid lines on the game board.
- `draw_board()`: Renders the 'X' and 'O' marks on the board.
- `check_win(player)`: Checks if the specified player has won by examining rows, columns, and diagonals.

5. Main Loop:

The core functionality of the game is in the `main()` function, which handles player turns, mouse clicks, and updates the screen.

Detailed Explanation:

1. Initialization:

   → Pygame is initialized with `pygame.init()`, and various constants and colors are defined.

2. Screen and Board Initialization:

   → The game window is created with a specified size, and the board is initialized as a 2D list filled with empty cells.

3. Drawing Functions:

   → `draw_grid()` and `draw_board()` functions handle rendering the game's grid and marks on the screen.

4. Checking for a Win:

   → The `check_win(player)` function checks if the specified player has won the game by examining rows, columns, and diagonals. If a win is detected, the game ends.

5. Main Game Loop:

   → The `main()` function manages the game loop. It handles player turns, mouse clicks, and updates the screen.

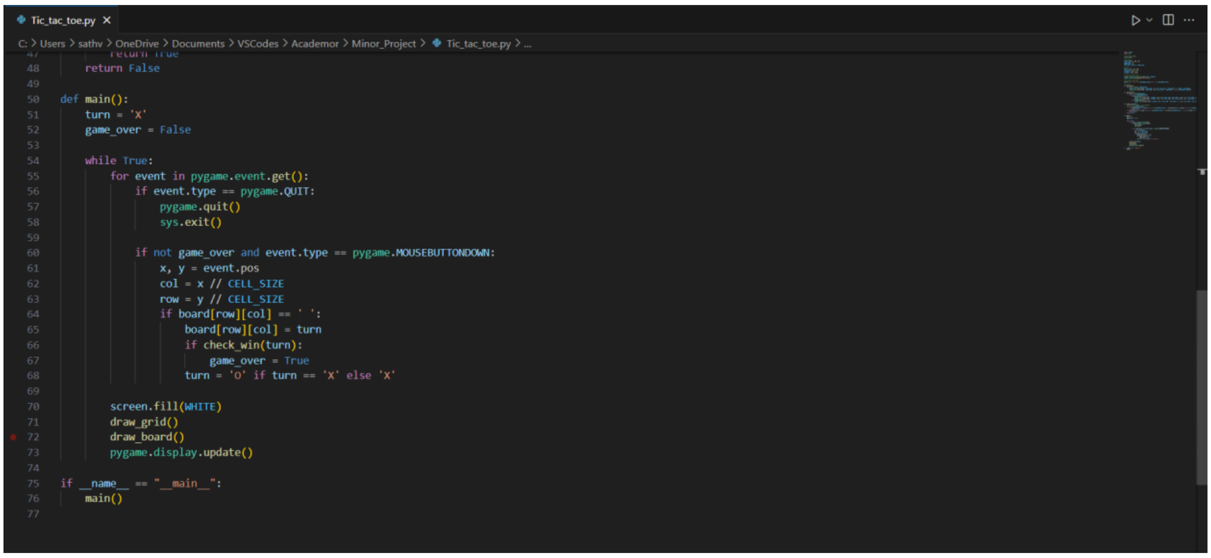
6. Event Handling:

   → The game captures user events, including quitting the game and mouse clicks. If a player clicks on an empty cell, their mark ('X' or 'O') is placed, and the game checks for a win condition.

7. Display Updates:

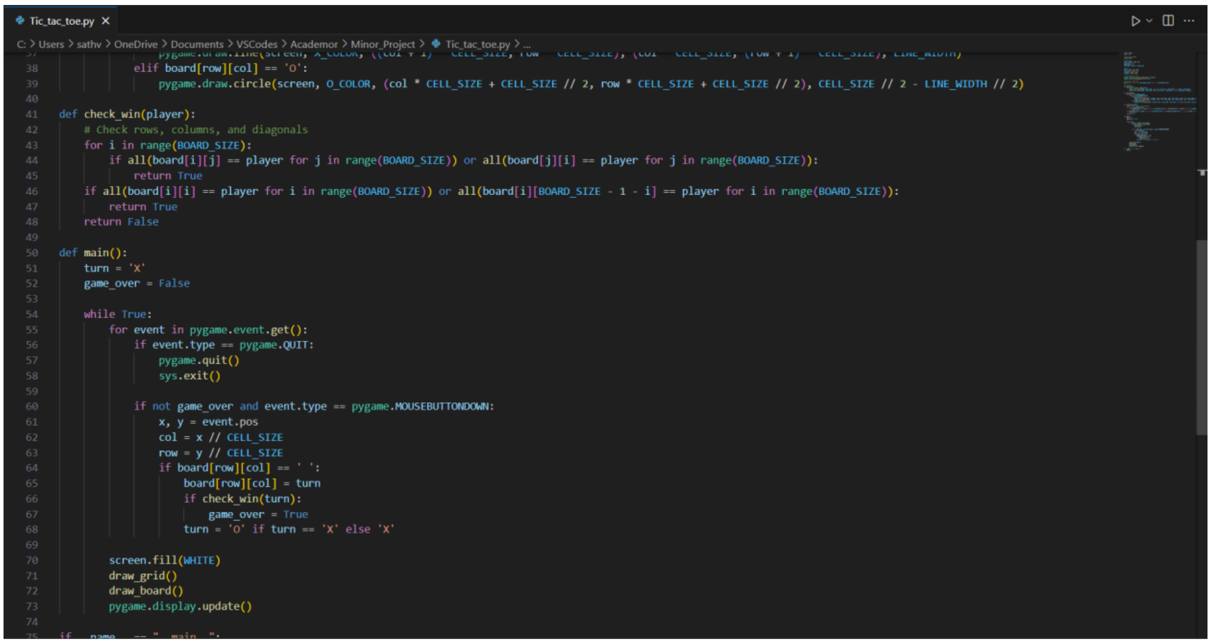
   → The screen is filled with a white background, and the grid and board are drawn on it. The display is updated to reflect changes.

Screen Shots of the Code and Input:



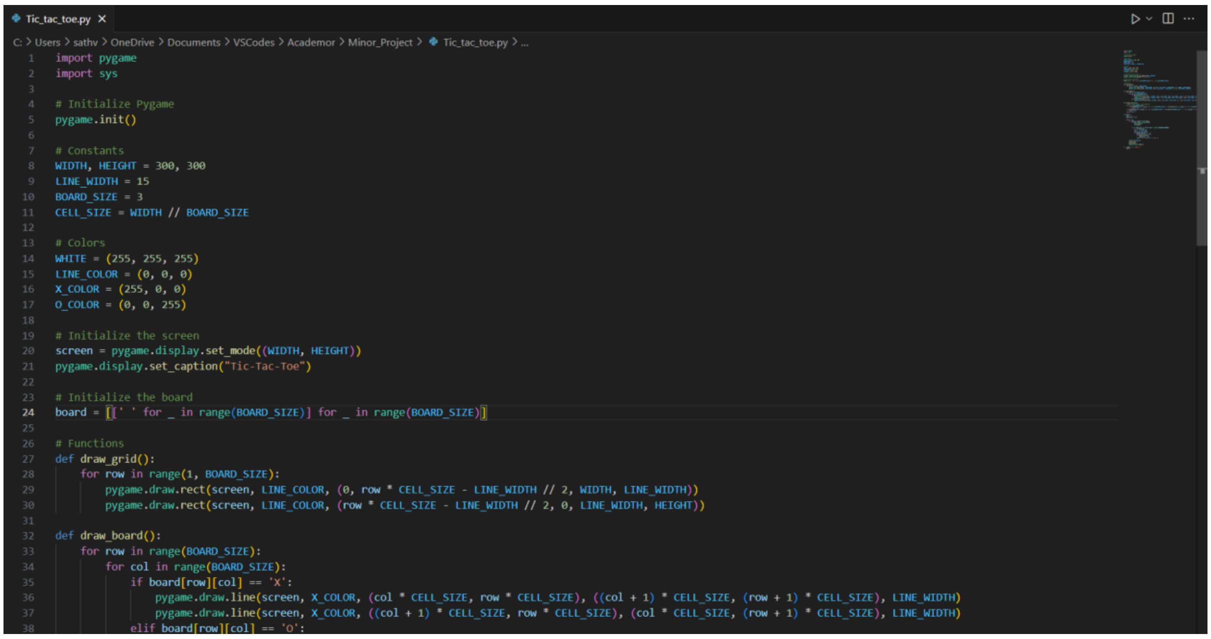
Tic_tac_toe.py
C: > Users > sathv > OneDrive > Documents > VSCodes > Academor > Minor_Project > Tic_tac_toe.py > ...

```python
            return True
        return False

def main():
    turn = 'X'
    game_over = False

    while True:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()

            if not game_over and event.type == pygame.MOUSEBUTTONDOWN:
                x, y = event.pos
                col = x // CELL_SIZE
                row = y // CELL_SIZE
                if board[row][col] == ' ':
                    board[row][col] = turn
                    if check_win(turn):
                        game_over = True
                    turn = 'O' if turn == 'X' else 'X'

        screen.fill(WHITE)
        draw_grid()
        draw_board()
        pygame.display.update()

if __name__ == "__main__":
    main()

```



Tic_tac_toe.py
C: > Users > sathv > OneDrive > Documents > VSCodes > Academor > Minor_Project > Tic_tac_toe.py > ...

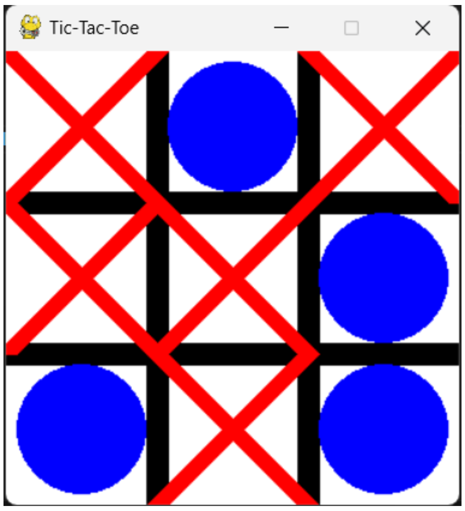
```python
                pygame.draw.line(screen, X_COLOR, ((col + 1) * CELL_SIZE, row * CELL_SIZE), (col * CELL_SIZE, (row + 1) * CELL_SIZE), LINE_WIDTH)
            elif board[row][col] == 'O':
                pygame.draw.circle(screen, O_COLOR, (col * CELL_SIZE + CELL_SIZE // 2, row * CELL_SIZE + CELL_SIZE // 2), CELL_SIZE // 2 - LINE_WIDTH // 2)

def check_win(player):
    # Check rows, columns, and diagonals
    for i in range(BOARD_SIZE):
        if all(board[i][j] == player for j in range(BOARD_SIZE)) or all(board[j][i] == player for j in range(BOARD_SIZE)):
            return True
    if all(board[i][i] == player for i in range(BOARD_SIZE)) or all(board[i][BOARD_SIZE - 1 - i] == player for i in range(BOARD_SIZE)):
        return True
    return False

def main():
    turn = 'X'
    game_over = False

    while True:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()

            if not game_over and event.type == pygame.MOUSEBUTTONDOWN:
                x, y = event.pos
                col = x // CELL_SIZE
                row = y // CELL_SIZE
                if board[row][col] == ' ':
                    board[row][col] = turn
                    if check_win(turn):
                        game_over = True
                    turn = 'O' if turn == 'X' else 'X'

        screen.fill(WHITE)
        draw_grid()
        draw_board()
        pygame.display.update()

if __name__ == "__main__":
```

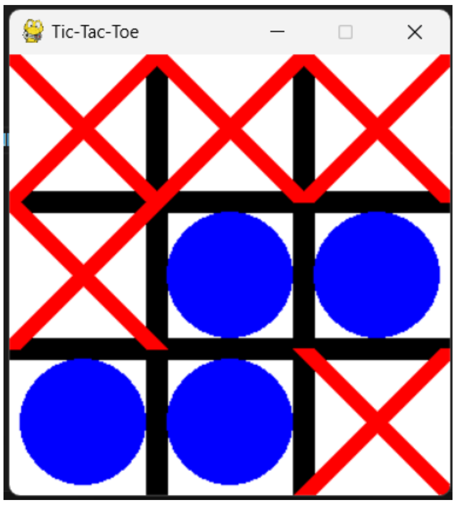


Tic_tac_toe.py

C: > Users > sathv > OneDrive > Documents > VSCodes > Academor > Minor_Project > Tic_tac_toe.py > ...

```python
import pygame
import sys

# Initialize Pygame
pygame.init()

# Constants
WIDTH, HEIGHT = 300, 300
LINE_WIDTH = 15
BOARD_SIZE = 3
CELL_SIZE = WIDTH // BOARD_SIZE

# Colors
WHITE = (255, 255, 255)
LINE_COLOR = (0, 0, 0)
X_COLOR = (255, 0, 0)
O_COLOR = (0, 0, 255)

# Initialize the screen
screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Tic-Tac-Toe")

# Initialize the board
board = [[' ' for _ in range(BOARD_SIZE)] for _ in range(BOARD_SIZE)]

# Functions
def draw_grid():
    for row in range(1, BOARD_SIZE):
        pygame.draw.rect(screen, LINE_COLOR, (0, row * CELL_SIZE - LINE_WIDTH // 2, WIDTH, LINE_WIDTH))
        pygame.draw.rect(screen, LINE_COLOR, (row * CELL_SIZE - LINE_WIDTH // 2, 0, LINE_WIDTH, HEIGHT))

def draw_board():
    for row in range(BOARD_SIZE):
        for col in range(BOARD_SIZE):
            if board[row][col] == 'X':
                pygame.draw.line(screen, X_COLOR, (col * CELL_SIZE, row * CELL_SIZE), ((col + 1) * CELL_SIZE, (row + 1) * CELL_SIZE), LINE_WIDTH)
                pygame.draw.line(screen, X_COLOR, ((col + 1) * CELL_SIZE, row * CELL_SIZE), (col * CELL_SIZE, (row + 1) * CELL_SIZE), LINE_WIDTH)
            elif board[row][col] == 'O':
```





Conclusion:

This Tic-Tac-Toe project showcases a practical implementation of a classic game using the Pygame library. It demonstrates how to handle user input, draw graphics, and manage game logic. It provides a foundation for building more complex games and interactive applications in Python. By following this case study, you can learn the fundamentals of developing simple 2D games using Pygame.