Documentation: Tic-Tac-Toe by Azhar

This document provides a complete, A-to-Z breakdown of the Tic-Tac-Toe game project. It covers everything from project setup and installation to a detailed explanation of every line of code.

Part 1: Project Setup & Installation

Before running the Python script (.py), you need to install its dependencies.

1. Install Python

If you don't have it, download and install Python from python.org.

2. Install Necessary Imports (Pygame)

This project relies on one main library: Pygame. You can install it using pip (Python's package installer) from your command prompt or terminal.

pip install pygame

3. Running the Game

Once Pygame is installed, you can run the game by navigating to the project folder in your terminal and executing the script:

python TicTacToe.py

(Assuming your file is named TicTacToe.py)

Part 2: Creating a Standalone .exe File

To share your game with others who don't have Python installed, you can bundle it into a single .exe file using **PyInstaller**.

1. Install PyInstaller

First, install the PyInstaller library:

pip install pyinstaller

2. Run the PyInstaller Command

This is the most critical step. Pygame loads your images (.png files) at runtime, so PyInstaller can't find them on its own. You must tell PyInstaller to include them.

Open your command prompt, cd to your project directory (where your .py and .png files are), and run the following command:

pyinstaller --onefile --windowed --add-data "tic tac opening.png;." --add-data "x.png;." --add-data "o.png;." TicTacToe.py

Command Breakdown:

- --onefile: Bundles everything (Python, Pygame, your code, your images) into a single .exe file.
- --windowed: Hides the black command prompt window when the game runs.
- --add-data "file;.": This is the magic. It tells PyInstaller to "add this file, and place it in the root (.) folder of the virtual environment." You must do this for all three image files.
- TicTacToe.py: The name of your main script.

3. Find Your Game

After the command finishes, look for a new folder named dist. Inside dist, you will find your TicTacToe.exe. This single file is your complete, shareable game.

Part 3: Code Documentation (A to Z)

This is a detailed breakdown of how your game's code is structured and what each part does.

A. Imports & Config (Lines 1-15)

- **pygame as pg**: The core library for all graphics, sound, and input.
- **sys**: Used for sys.exit(), the cleanest way to close the application.
- **pygame.locals**: Imports common constants (like QUIT, KEYDOWN) to make event handling easier to read.
- **Config (Lines 5-15)**: You define constants for screen dimensions, FPS (Frames Per Second), and all your COLORS. Using constants (e.g., WHITE) instead of (255, 255, 255) makes your code much cleaner and easier to modify.

B. Global Variables (Lines 18-28)

These variables track the "state" of the entire game.

- XO: Tracks whose turn it is.
- winner, draw: Start as None or False and are changed when the game ends.
- TTT: The "brain" of the game. This 2D list (a list of lists) is your internal 3x3 game board. None represents an empty cell.
- mode: Remembers if the user chose "PVP" or "PVC".
- p1_name, p2_name: Store the names entered by the users.
- finish_snapshot, win_line: Used for the game-over screen. win_line stores the coordinates for the red winning line, and finish_snapshot stores a full-screen screenshot of the final board.

C. Pygame Init (Lines 31-36)

- pg.init(): Initializes all Pygame modules.
- screen = pg.display.set_mode(): Creates the 600x600 window.
- pg.scrap.init(): **Critical for your clipboard features.** This initializes Pygame's module for accessing the computer's copy/paste clipboard.
- pg.mouse.set_visible(True): Forces the mouse cursor to be visible.
- CLOCK: Creates a Clock object, which is used in the main loop to ensure the game runs at a steady 60 FPS.

D. Asset Loading (Lines 39-61)

- This section loads your background image (opening.png) and your X/O images (x.png, o.png).
- **try...except Blocks**: This is excellent error handling. If an image file is missing, the except block *creates a placeholder* (either a gray surface or a text "X"/"O"). This prevents the game from crashing.
- **Fonts**: You load four different font sizes (title, large, med, small) to be used for all text in the game.

E. UI Helpers (Button Class) (Lines 64-74)

This is a small piece of Object-Oriented Programming (OOP) that makes your code reusable.

- Instead of writing new code for every button, you create a Button "blueprint."
- __init__(): The setup function. It just stores the button's position (rect), text (label), and colors.
- draw(): A method that draws the button onto the screen. It's smart enough to check if the mouse is hovering over it (is_hover) and change its color.
- clicked(): A method that checks if a MOUSEBUTTONDOWN event happened inside the button's boundaries.

F. Board & Status (Lines 77-94)

- draw_board(): A simple function that fills the background with WHITE and draws the four LINEs that make the 3x3 grid.
- draw_status(): Draws the top 72-pixel header. It dynamically updates the text to show whose turn it is ("X's Turn") or the game-over message ("Computer won!").

G. Win/Draw Logic (Lines 97-128)

- check_win(): This function is the core game logic. It runs after every single move.
  1. It checks all 3 **rows** for a 3-in-a-row.
  2. If no win, it checks all 3 **columns**.
  3. If no win, it checks the 2 **diagonals**.
  4. If any check passes, it sets the winner variable ('x' or 'o') and **sets the win_line variable** with the screen coordinates for drawing the red winning line.
  5. If all 8 checks fail, it checks if the board is full. If so, it sets draw = True.

H. Game Moves (Lines 131-160)

- drawXO(row, col): This function doesn't draw anything. It updates the *data*. It places an 'x' or 'o' into the TTT list (e.g., TTT[0][1] = 'x'). It also switches the turn from 'x' to 'o' or vice-versa.
- user_click(): This is the click handler. It converts the mouse's (x, y) pixel position into a grid (row,

col). It then checks if that grid cell is empty (is None). If it is, it calls drawXO() and check_win().

I. Minimax AI (Lines 163-200)

This is the "unbeatable" AI for the Player vs. Computer mode.

- **Concept**: Minimax is an algorithm that explores every possible future move. The AI ('o') is the "maximizer" (a win is +1). The player ('x') is the "minimizer" (a player win is -1). A draw is 0.
- empty_cells(): A helper function that finds all available moves.
- evaluate_state(): A simple function that scores a *finished* board (+1, -1, or 0).
- minimax(state, maximizing): The recursive "brain." It simulates a move, then calls itself for the *other* player. It "backtracks" up the chain, with the maximizer always picking the highest score and the minimizer always picking the lowest. This allows it to find the path that *guarantees* a win or a draw.
- ai_move_if_needed(): This is called by the game_loop. It runs the minimax algorithm to get the best_move and then makes it.

J. Screen Management (Lines 203-294)

Your game isn't just one screen; it's a "state machine." These functions are the different states.

- title_menu(): The main menu screen. It has its own event loop, draws the background and buttons, and waits for a click. It returns the user's choice ("PVP" or "PVC").
- text_input_screen(): A reusable screen for getting player names. It has a KEYDOWN event loop to build a text string.
    - **Clipboard Feature**: This is where your pg.scrap code is used. It checks for KMOD_CTRL (Ctrl key). If Ctrl+V is pressed, it pg.scrap.get()s text from the clipboard. If Ctrl+C/X is pressed, it pg.scrap.put()s the text.
    - **Visual Feedback**: It highlights the text box in blue if all_selected is True (from Ctrl+A).
- collect_names(): A simple manager that calls text_input_screen once or twice based on the mode.
- announce_roles_then_start(): A simple "Get Ready" screen that pauses for 900ms.
- result_menu_overlay(snapshot): The polished game-over screen.
    - It accepts a snapshot (an image of the final board).
    - In its loop, it **draws the snapshot first**, then draws a semi-transparent black dim surface, then draws the "Play Again" / "Menu" buttons. This creates the professional overlay effect.

K. Main Flow (Lines 297-364)

These functions control the high-level application flow.

- reset_board(): A simple helper that resets all global variables to their defaults, readying the game for a new match.
- render_all(): The main drawing function *during gameplay*. It calls draw_board() and draw_status(). Then, it loops through the TTT list and draws the X/O images. Finally, if win_line has been set, it draws the red winning line.
- game_loop(): The heart of the *gameplay* state.
    1. It handles MOUSEBUTTONDOWN events, calling user_click() and ai_move_if_needed().
    2. It checks if winner is None and not draw: If True, the game is in progress, so it calls render_all().

3. If False, the game is **over**.
    - It takes the finish_snapshot (if it's the first frame of the game being over).
    - It then calls result_menu_overlay(finish_snapshot), which "pauses" the game_loop until the user makes a choice.
    - It handles the choice ("AGAIN" or "MENU").

- main(): The top-level function that holds the entire application together. It's a while True loop that calls the other "state" functions in order: title_menu -> collect_names -> game_loop. If game_loop returns (from "Return to Menu"), the main loop simply restarts, showing the title_menu again.
- if __name__ == "__main__":: This is the standard entry point for a Python program. It tells the script: "Start by running the main() function."