Tic Tac Toe Game

This document provides comprehensive documentation for the Tic Tac Toe web application, covering its structure, functionality, and how to set it up and run it.

1. Introduction

This is a simple web-based Tic Tac Toe game implemented using Flask for the backend and HTML, CSS, and JavaScript for the frontend. It allows two players to enter their names, play the game, and track their scores for the current session.

Features:

- **Player Name Input:** Players enter their names before starting a game.
- **Interactive Game Board:** A 3x3 grid where players take turns.
- **Current Player Display:** Shows whose turn it is.
- **Game Outcome Display:** Announces win or draw.
- **Move History:** Tracks all moves made in the current game.
- **Session-based Player Stats:** Tracks wins for each player, draws, and total matches *within the current game session*. These stats reset if the players start a new session (e.g., by going back to the home page).
- **Persistent Global Leaderboard (Backend Only):** While the frontend displays only session-specific stats, the backend leaderboard.json file *does* keep a persistent, global record of wins, losses, draws, and total matches for each player across all sessions. This data is not directly visible on the game.html page but is updated.
- **Reset Board Functionality:** Clears the board for a new round while keeping session stats.
- **Start New Game Functionality:** Allows players to go back to the name input screen, resetting all session stats.
- **Basic Styling:** Clean and responsive UI.
- **Sound Effects:** Simple sounds for moves and wins.
- **Flash Messages:** Provides user feedback for various actions and errors.

2. Project Structure

The project follows a standard Flask application structure:

```
tic-tac-toe/

 ├── app.py

 ├── leaderboard.json  (created automatically on first run)
├── static/
│   ├── css/
│   │   └── style.css
│   ├── js/
│   │   └── script.js
│   └── sounds/
│       ├── move.mp3
│       └── win.mp3
└── templates/
    ├── game.html
    └── index.html
```

- app.py: The main Flask application file, handling routing, game logic, and data persistence.
- leaderboard.json: A JSON file used to store persistent player statistics (wins, losses, draws, total matches) across different application runs.
- static/: Contains static files (CSS, JavaScript, sounds).
    - css/style.css: Defines the visual styles of the application.
    - js/script.js: Contains the frontend JavaScript logic for game interaction, AJAX calls, and UI updates.
    - sounds/: Directory for game sound effects.
- templates/: Contains Jinja2 HTML templates rendered by Flask.
    - index.html: The landing page for entering player names.
    - game.html: The main game board page where Tic Tac Toe is played.

3. Setup and Installation

To run this application, you need Python and Flask installed.

Prerequisites:

- Python 3.x
- pip (Python package installer)

Steps:

1. **Clone the repository (or create files manually):** If you have the files provided, ensure they are organized in the structure mentioned above.
2. **Navigate to the project directory:**

Bash

```
cd tic-tac-toe
```

3. **Create a virtual environment (recommended):**

Bash

```
python -m venv venv
```

4. **Activate the virtual environment:**
   - **On Windows:**

Bash

```
.\venv\Scripts\activate
```

   - **On macOS/Linux:**

Bash

```
source venv/bin/activate
```

5. **Install Flask:**

Bash

```
pip install Flask
```

6. **Set a secret key for Flask (important for session security):** It's recommended to set this as an environment variable.
   - **On Windows (Command Prompt):**

Bash

```
set FLASK_SECRET_KEY=your_super_secret_key_here
```

   - **On Windows (PowerShell):**

Bash

```
$env:FLASK_SECRET_KEY="your_super_secret_key_here"
```

   - **On macOS/Linux:**

Bash

```
export FLASK_SECRET_KEY='your_super_secret_key_here'
```

7. Replace your_super_secret_key_here with a long, random string. For development, os.urandom(24) in app.py provides a fallback, but a dedicated environment variable is best practice for production.
8. Run the Flask application:

Bash

```
python app.py
```

9. **Access the application:** Open your web browser and navigate to http://127.0.0.1:5000/.

4. Code Explanation

4.1. app.py (Flask Backend)

This file contains the core server-side logic.

- **Imports:** Flask, render_template, request, redirect, url_for, session, jsonify, flash, os, json.
- **App Initialization:** app = Flask(__name__) and app.secret_key for session management.
- **LEADERBOARD_FILE:** Constant for the JSON file storing persistent leaderboard data.

Leaderboard Management Functions:

- load_leaderboard(): Reads and returns the contents of leaderboard.json. Handles file not found or JSON decode errors by returning an empty dictionary.
- save_leaderboard(leaderboard_data): Writes the provided leaderboard_data dictionary to leaderboard.json.
- get_or_create_player_stats(leaderboard_data, player_name): Ensures a player entry exists in the leaderboard_data. If not, it initializes their stats to zero before returning their entry.

Game Logic Functions:

- initialize_game_state():
    - Crucially, this function is responsible for setting up a *new game session*.
    - It defines the initial state of the Tic Tac Toe board, current player, game status, and **resets all session-specific counts (wins for X, wins for O, draws, and total matches)** to zero.
    - This ensures that every new game session started from the / or /start_game route begins with fresh stats on the game.html page.
- check_win(board): Checks for a winning condition on the given board.
- check_draw(board): Checks if the game is a draw (all cells filled, no winner).

Routes:

- @app.route('/'):
    - **GET:** Renders index.html.
    - **Action:** Clears any existing game_state and players from the Flask session. This is vital to ensure that visiting the home page effectively starts a clean new session, leading to a full stat reset on the game board page when a new game is initiated.
- @app.route('/start_game', methods=['POST']):
    - **POST:** Handles player name submission from index.html.
    - **Input Validation:** Checks if both names are provided, are different, and within character limits. Uses flash messages for feedback.
    - **Session Initialization:** Stores player names in session['players'] and calls initialize_game_state() to set up a new game, **resetting all session-specific game stats.**
    - **Redirect:** Redirects to the /game route.
- @app.route('/game'):
    - **GET:** Renders game.html.
    - **Pre-check:** Ensures game_state and players exist in the session. If not, redirects to the home page with an info message.
    - **Session Stats for Display:** Gathers only the session_x_wins, session_o_wins, session_draws, and session_total_matches from the current game_state to be passed to the template. **It explicitly does NOT load global stats from leaderboard.json for display on this page.**
- @app.route('/make_move', methods=['POST']):
    - **POST (AJAX):** Handles a player's move.
    - **Input:** Expects cell_index from the frontend JSON payload.
    - **Game Logic:**
        - Updates the board with the current player's mark.
        - Appends the move to move_history.
        - Calls check_win and check_draw.
        - **Global Leaderboard Update:** *Crucially*, it loads leaderboard.json, updates the *global* wins/losses/draws/total matches for the players involved, and saves leaderboard.json. This ensures persistence of overall player records even if not displayed on game.html.

        - **Session Stats Update:** Updates the session_x_wins, session_o_wins, session_draws, and session_total_matches within the game_state dictionary in the Flask session. These are the values displayed on the frontend.
        - Switches current_player if the game is not over.
    - **Response:** Returns JSON with the updated game state, players information, and current_game_stats (session-specific) to the frontend.
- @app.route('/reset_board', methods=['POST']):
    - **POST (AJAX):** Resets the game board for a new round *within the current session*.
    - **Session Preservation:** It creates a *new board state* but **preserves the existing session_x_wins, session_o_wins, session_draws, and session_total_matches**. This allows players to play multiple rounds in a session and see cumulative session-specific stats.
    - **Response:** Returns JSON with the new game state and preserved current_game_stats.

4.2. templates/ (Jinja2 HTML Templates)

- **index.html**:
    - A simple form to collect Player X and Player O names.
    - Uses Flask's url_for to link to style.css and the start_game route.
    - Includes a flash-messages div to display server-side messages.
- **game.html**:
    - The main game interface.
    - **Left Panel (left-panel):** Displays "Player Stats".
        - It now explicitly shows current_game_stats['session_x_wins'] and current_game_stats['session_o_wins'], along with session_draws and session_total_matches.
        - The "Start New Game" button has been moved from here.
    - **Center Panel (center-panel):**
        - Displays the current players, current turn, and game result.
        - The Tic Tac Toe board is rendered dynamically using a for loop over game.board.
        - Contains the "Reset Board" and "Start New Game" buttons, placed side-by-side.

    - **Right Panel (right-panel):** Displays the move_history as an unordered list.
    - Includes audio elements for move and win sounds.
    - Links to style.css and script.js.

4.3. static/css/style.css

- **CSS Variables:** Defines a color palette using :root variables for easy theme management. This includes the restored green primary color.
- **Layout:** Uses Flexbox for the overall game-container layout, allowing for responsive stacking of panels.
- **Styling:** Provides comprehensive styling for:
    - Body and main containers.
    - Form elements on index.html.
    - Game panels (left, center, right).
    - Headings, paragraphs, and lists.
    - The Tic Tac Toe grid (game-board) and individual cells.
    - Buttons (start-button, reset-button, new-game-button).
    - Flash messages (success, error, info).
    - Responsive adjustments using media queries for various screen sizes, ensuring a good experience on mobile and desktop.

4.4. static/js/script.js

- **DOM Element Selection:** Selects various HTML elements using their IDs or classes.
- **showFlashMessage():** A helper function to display dynamic messages on the frontend, mimicking Flask's flash behavior.
- **Event Listeners:**
    - Attaches click listeners to each cell on the game board to trigger the makeMove function.
    - Attaches a click listener to the reset-board-button to trigger the resetBoard function.
- **updateUI(game, players, current_game_stats):**
    - This is the central function for updating the frontend after an AJAX call.
    - It iterates through the game.board to update cell content and classes.
    - Updates the current player display, game result message, and move_history.

- **Crucially, it updates the "Player Stats" panel using current_game_stats.session_x_wins, current_game_stats.session_o_wins, current_game_stats.session_draws, and current_game_stats.session_total_matches, ensuring only session-specific data is displayed.**
- Plays move and win sounds.

- **makeMove(cellIndex):**
  - An async function that sends an AJAX POST request to the /make_move endpoint with the clicked cell's index.
  - Parses the JSON response from the server.
  - If successful, calls updateUI to refresh the game state and stats on the page.
  - Handles errors and displays flash messages.
- **resetBoard():**
  - An async function that sends an AJAX POST request to the /reset_board endpoint.
  - Parses the JSON response.
  - If successful, calls updateUI to reset the board visually while maintaining session stats.
  - Handles errors.

5. Usage

1. **Start the game:** Run app.py.
2. **Enter Player Names:** On the home page, enter names for Player X and Player O.
3. **Play:** Click on cells to make moves.
4. **Reset Board:** Click the "Reset Board" button to clear the board and start a new round with the same players, accumulating session stats.
5. **Start New Game:** Click the "Start New Game" button to go back to the name input page. This will **reset all session stats** for the next game.

6. Future Enhancements

- **View Global Leaderboard:** Add a new page/route to display the persistent leaderboard data stored in leaderboard.json.
- **Player Profiles:** Allow players to log in or select from existing profiles.
- **AI Opponent:** Implement a simple AI to play against.

- **Undo Last Move:** Add functionality to undo the last move.
- **More Advanced UI/UX:** Animations, more refined design elements.
- **Player Settings:** Options for sound, theme, etc.
- **Error Handling:** More robust error handling and logging.