### **Tic-Tac-Toe Game Development Report**

**Introduction**

In this project, I developed an MVP (Minimum Viable Product) of the classic Tic-Tac-Toe game using basic technologies such as JavaScript, HTML5, and CSS3, with a focus on Object-Oriented Programming (OOP). Throughout the development, various functionalities were implemented, including aesthetic improvements, gameplay optimization, and support for accessibility. This report describes the process, architectural decisions, best practices used, and additional improvements.

### **Solution Design**

#### **Main Classes**

The solution design is based on a clear OOP architecture where each component of the game has its own responsibility. Three main model classes were used:

* **Game**: Controls the game flow, manages turns, restarts the game, and determines the game state (whether there is a winner or a tie).
* **Board**: Represents the game board and manages interactions with the cells.
* **Player**: Represents each player (X or O) and their actions.

The focus on modularity ensures that each class handles a unique responsibility, applying the **Single Responsibility** principle from the **SOLID** principles. As a result, functionalities that could have been included in the model class were moved to a service, ensuring that the object’s responsibility scope remained unchanged, adhering to best practices.

#### **Conceptual Diagram**

* **Game**:
  + Attributes: playerX, playerO, isAI, board, gameHistory.
  + Methods: resetHistory().
* **Board**:
  + Attributes: cells, size (to support NxN boards).
  + Methods: isFull().
* **Player**:
  + Attributes: symbol (X or O).
* **BoardService**:
  + Attributes: Board.
  + Methods: render(), handleCellClick(), markCell(), resetBoard(), checkWinningCombination().
* **GameService**:
  + Attributes: Game, playerX, playerO, board, currentPlayer, boardService, isGameOver.
  + Methods: init(), updateGameStatus(), handleTurn(), switchPlayer(), endGame(), restart(), replayLastGame(), newGame(), aiMove(), minimax(), checkWinner().

This design ensures a flexible and extensible architecture, allowing for the easy addition of features such as AI gameplay or expanding the board to an NxN size​.

### **Development**

#### **Methodology**

An iterative development approach was followed, where the essential functionality was built first (3x3 grid, turn alternation, winner or tie detection), and then additional features were added, such as game reset, a retro arcade visual style, accessibility, and AI gameplay.

#### **Step-by-Step Process**

* **HTML**: A basic structure was created with a container for the board and an indicator of whose turn it was, along with a button to reset the game.
* **CSS**: The board was styled to be a flexible grid with a retro arcade design, using bright colors and pixelated fonts to simulate a classic video game.
* **JavaScript - OOP**:
  + **BoardService** initializes the board and manages interaction with the cells.
  + **GameService** manages the main logic, such as switching turns and checking for a winner.

#### **OOP Principles**

During development, I ensured the application of **SOLID** principles:

* **Single Responsibility**: Each class has a specific function.
* **Open/Closed**: The structure is easy to extend, but the classes don’t need modification to add new features.

These principles ensure that the code is **maintainable** and **scalable**, allowing future additions without compromising the integrity of the solution​.

### **Additional Features**

#### **Retro Arcade Style**

To improve user experience, I implemented a visual style inspired by arcade video games, with a pixelated font ("Press Start 2P"), bright colors, and visual effects like flashing winning cells. Additionally, the game is fully responsive, adapting to small screens without losing its aesthetic​.

#### **AI with Minimax Algorithm**

I implemented the **Minimax algorithm** for AI, ensuring that the machine always chooses the best possible move. The algorithm explores all future combinations and selects the optimal option, maximizing the machine's benefit and minimizing the player’s advantage. This guarantees that the AI is practically unbeatable if both players play optimally​(informe tateti).

#### **NxN Board**

The solution was designed to adapt the board size, no longer limited to 3x3 but to any NxN size defined by the user. This functionality required adjustments to both the **UI** and the validation logic to detect winners on boards of any size, ensuring that the code is reusable and easily adjustable​(informe tateti). The Minimax algorithm, however, was not optimized for larger grids and didn’t work as expected for NxN boards, which remains an area for potential improvement.

### **Testing and Validation**

#### **Manual Testing**

I manually verified:

* **Turn alternation**: Players correctly alternate between "X" and "O".
* **Winner detection**: The game correctly identifies when a player aligns three marks.
* **Tie**: The game declares a tie when the board is full without a winner.

#### **Unit Test Proposal**

To ensure code quality, I proposed and developed **unit tests** using the **Jest** library. The tests focused on:

* Correct winner detection (checkWinner()).
* Turn alternation (switchPlayer()).

### **Accessibility**

I implemented **assistive technologies** to improve the game’s accessibility:

* **Keyboard navigation**: Allowing play without a mouse, using keys such as "Tab", "Enter", and "Space".
* **ARIA attributes**: Attributes like aria-live and aria-label were added to communicate the game state through screen readers.

These improvements make the game accessible to users with motor or visual impairments​.

### **Conclusion**

This project not only meets the requirements of a Tic-Tac-Toe game but has been extended with advanced features, such as the **Minimax algorithm**, support for a **dynamic board size**, and an attractive **visual style**. I applied **best practices** in OOP, maintaining a scalable and accessible architecture, all while ensuring a pleasant and functional user experience.

With all these features, the game is not only fun but also designed to be inclusive, efficient, and extensible.