# Tic-Tac-Toe Game: Final Internship Report

**Name:** Immad Baber
**Title:** C++ Developer
**Company:** Velocity Solutions
**Date:** July 19, 2025

## Contents

## 1 Project Summary

During my internship at Velocity Solutions, I developed a Tic-Tac-Toe game using C++ and the SFML library. This project allowed me to apply object-oriented programming (OOP), STL containers, and game development skills to create a fun and interactive game. The game includes a main menu, two modes (Player vs Player and Player vs AI), and visual effects like particles. It was a great opportunity to improve my coding, UI design, and game logic skills.

The game features easy mouse controls, a statistics system to track wins and draws, and an AI with three difficulty levels (Easy, Medium, Hard). This report explains the project goals, code structure, features, challenges faced, and ideas for future improvements.

## 2 Project Objectives

The goals of the Tic-Tac-Toe game project were:

- Create an enjoyable Tic-Tac-Toe game using C++ and SFML.
- Use OOP and STL containers for clean and efficient code.
- Design a user-friendly interface with menus and visual effects.
- Implement an AI opponent with Easy, Medium, and Hard difficulty levels.
- Track game statistics like wins, losses, and draws.
- Learn game development techniques, including input handling and rendering.

## 3 Code Overview

The Tic-Tac-Toe game is built with a modular structure using OOP and STL containers. The main components are:

- **TicTacToeGame Class:** Controls the game loop, states (Menu, Mode Select, Playing, Game Over, Settings), and rendering. It uses SFML for graphics and input.
- **Button Class:** Manages menu buttons with hover and click effects. Uses STL for button properties and SFML for drawing.
- **Particle Class:** Adds visual effects for moves. Uses STL vectors for particle data and SFML for rendering.
- **GameStats Structure:** Tracks wins, losses, and draws using simple data structures. Saves data to a file with STL file streams.
- **AI Logic:** Implements Easy (random moves), Medium (random and strategic moves), and Hard (Minimax algorithm) AI modes.

- **SFML Integration:** Renders the game grid, cells, and text with custom colors and animations for a better visual experience.

The game loop processes inputs, updates objects, and renders the scene. STL containers like arrays manage the game board, and random number generators support AI moves.

## 4 Code Structure

The code is organized in a single file, `game.cpp`, with clear classes and functions:

- **Enums:** Define game states (`MENU`, `MODE_SELECT`, `PLAYING`, etc.), modes (`PLAYER_VS_PLAYER`, `PLAYER_VS_AI`), and cell states (`EMPTY`, `X_PLAYER`, `O_PLAYER`).
- **Button Class:** Handles menu buttons with position, size, text, and gradient effects. Includes `update()` and `isClicked()` for interactions.
- **Particle Class:** Manages particle animations with `position`, `velocity`, and `life` properties.
- **GameStats Structure:** Stores `playerWins`, `aiWins`, `draws`, and `totalGames` for statistics.
- **TicTacToeGame Class:** The core class with functions like `initializeGame()`, `handleInput()`, `update()`, and `render()`. Uses SFML for graphics and STL for data.
- **AI Functions:** Includes `makeAIMove()`, `makeRandomMove()`, `makeStrategicMove()`, and `minimax()` for AI decisions.

## 5 Features and Future Improvements

### 5.1 Current Features

- **Game Modes:** Play against another player or AI (Easy, Medium, Hard).
- **User Interface:** Main menu, mode selection, and game over screens with animated buttons and text.
- **Visual Effects:** Particle animations for moves, with different colors for X and O.
- **Statistics:** Tracks and saves wins, losses, and draws to a file.
- **Grid and Input:** 3x3 grid with mouse-based input for placing X or O.
- **AI Logic:** Easy uses random moves, Medium mixes random and strategic moves, and Hard uses the Minimax algorithm.

### 5.2 Future Improvements

- Add sound effects for moves, wins, and button clicks using SFML Audio.

- Enhance Medium AI with more strategic patterns.
- Include a tutorial mode for new players.
- Add keyboard controls for placing moves.
- Create animations for game start and end screens.
- Implement a leaderboard for high scores.

## 6 Challenges and Learnings

### 6.1 Challenges

- **AI Implementation:** Developing the Minimax algorithm for Hard AI was difficult. I had to ensure it evaluated all moves correctly.
- **Input Handling:** Detecting valid mouse clicks on the grid and preventing invalid moves required careful logic.
- **Visual Effects:** Creating smooth particle animations without slowing the game was challenging. I optimized updates using STL.
- **UI Design:** Making responsive buttons with hover and click effects took time to integrate with SFML.

### 6.2 Learnings

- **OOP and STL:** Using classes and STL containers (like arrays for the board) made code organized and efficient.
- **Game Loop:** Balancing input, updates, and rendering improved my game optimization skills.
- **AI Algorithms:** Implementing Minimax deepened my understanding of decision-making algorithms.
- **UI/UX Design:** Designing clear menus and effects taught me user-focused design.
- **Debugging:** Fixing AI move errors and particle issues improved my debugging skills.

This project was an exciting experience that enhanced my C++ programming, game development, and problem-solving skills. It motivated me to explore more game projects in the future.