# Software Requirements Specification (SRS) for Advanced Tic Tac Toe Game

## 1. Introduction

### 1.1 Purpose

This document specifies the functional and non-functional requirements for the Advanced Tic Tac Toe game. It aims to provide a clear understanding of the system's behavior, features, and constraints.

### 1.2 Scope

The Advanced Tic Tac Toe game is a C++ application with a graphical user interface (GUI) that allows users to play Tic Tac Toe against another player or an AI opponent. It includes user authentication, personalized game history, and an intelligent AI.

### 1.3 Definitions, Acronyms, and Abbreviations

- **AI:** Artificial Intelligence
- **GUI:** Graphical User Interface
- **SRS:** Software Requirements Specification
- **SDS:** Software Design Specification
- **CI/CD:** Continuous Integration/Continuous Deployment
- **Qt:** A cross-platform application development framework.
- **Minimax:** A decision rule used in artificial intelligence, decision theory, game theory, and statistics for minimizing the possible loss for a worst case (maximum loss) scenario.
- **Alpha-Beta Pruning:** A search algorithm that seeks to decrease the number of nodes that are evaluated by the minimax algorithm in its search tree.

## 2. Overall Description

### 2.1 Product Perspective

The Advanced Tic Tac Toe game is a standalone desktop application. It interacts with a local file system for user data and game history storage.

### 2.2 Product Functions

The system provides the following main functions: * User Authentication (Sign Up, Sign In, Sign Out) * Player vs. Player Game Mode * Player vs. AI Game Mode with adjustable difficulty * Game History Tracking and Replay * GUI for user interaction

### 2.4 General Constraints

- **Programming Language:** C++
- **GUI Framework:** Qt
- **Testing Framework:** Google Test (for unit and integration testing)
- **Version Control:** Git with GitHub
- **CI/CD:** GitHub Actions
- **Data Storage:** SQLite or customized file storage (currently implemented as customized file storage using JSON).
- **Security:** Password hashing (SHA-256 implemented).

## 3. Specific Requirements

### 3.1 Functional Requirements

#### 3.1.1 User Authentication

- **REQ-AUTH-001:** The system SHALL allow new users to sign up with a unique username and a password.
- **REQ-AUTH-002:** The system SHALL enforce password complexity rules: minimum 8 characters, at least one uppercase letter, and at least one special character.
- **REQ-AUTH-003:** The system SHALL allow existing users to sign in with their username and password.
- **REQ-AUTH-004:** The system SHALL hash passwords using SHA-256 for secure storage.
- **REQ-AUTH-005:** The system SHALL maintain a logged-in session for the current user.
- **REQ-AUTH-006:** The system SHALL allow logged-in users to sign out.
- **REQ-AUTH-007:** The system SHALL store user credentials and game history in a local file (users.json).

#### 3.1.2 Game Modes

- **REQ-GAME-001:** The system SHALL provide a Player vs. Player game mode.
- **REQ-GAME-002:** The system SHALL provide a Player vs. AI game mode.

- **REQ-GAME-003:** The system SHALL allow users to select AI difficulty levels (Easy, Medium, Hard, Unbeatable).

### 3.1.3 Game Logic

- **REQ-LOGIC-001:** The system SHALL implement a 3x3 Tic Tac Toe board.
- **REQ-LOGIC-002:** The system SHALL allow players to take turns marking cells with 'X' or 'O'.
- **REQ-LOGIC-003:** The system SHALL detect a win condition (three in a row, column, or diagonal).
- **REQ-LOGIC-004:** The system SHALL detect a tie condition (board full with no winner).
- **REQ-LOGIC-005:** The system SHALL prevent moves on already occupied cells.
- **REQ-LOGIC-006:** The AI opponent SHALL use the Minimax algorithm with Alpha-Beta Pruning for strategic moves, especially at higher difficulty levels.
- **REQ-LOGIC-007:** The AI's decision-making SHALL be influenced by the selected difficulty level, with higher difficulties leading to more optimal moves.

### 3.1.4 Game History

- **REQ-HISTORY-001:** The system SHALL save details of each completed game (date, result, vs AI/Player, difficulty if AI, moves) to the logged-in user's history.
- **REQ-HISTORY-002:** The system SHALL allow logged-in users to view their game history.
- **REQ-HISTORY-003:** The system SHALL allow users to replay past games move by move.

### 3.1.5 User Interface

- **REQ-UI-001:** The system SHALL provide a graphical user interface (GUI) for all interactions.
- **REQ-UI-002:** The GUI SHALL include distinct pages for Login, Sign Up, Main Menu, Game Mode Selection, Game Play, and Game History.
- **REQ-UI-003:** The game board SHALL be visually represented with clickable cells.
- **REQ-UI-004:** The GUI SHALL display the current game status (e.g.,

current player's turn, win/tie messages). * **REQ-UI-005:** The game history interface SHALL display a list of past games and provide controls for replaying selected games.

## 3.2 Non-Functional Requirements

### 3.2.1 Performance

- **NFR-PERF-001:** The game SHALL respond to user input within 100ms.
- **NFR-PERF-002:** AI moves SHALL be calculated and displayed within 500ms, even at the highest difficulty.
- **NFR-PERF-003:** Game history loading and saving SHALL not cause noticeable delays (e.g., within 1 second for typical history sizes).

### 3.2.2 Security

- **NFR-SEC-001:** User passwords SHALL be stored securely using hashing (SHA-256).
- **NFR-SEC-002:** The system SHALL protect against unauthorized access to user accounts and game history data.
- **NFR-SEC-003:** Input fields SHALL be sanitized to prevent common vulnerabilities (e.g., injection attacks).

### 3.2.3 Usability

- **NFR-USAB-001:** The GUI SHALL be intuitive and easy to navigate for new users.
- **NFR-USAB-002:** Error messages SHALL be clear, concise, and provide actionable guidance.
- **NFR-USAB-003:** The game SHALL provide visual feedback for user actions (e.g., cell clicks, game outcomes).

### 3.2.4 Reliability

- **NFR-REL-001:** The system SHALL handle unexpected user inputs gracefully without crashing.
- **NFR-REL-002:** Game state and user data SHALL be preserved in case of unexpected application termination.
- **NFR-REL-003:** The game logic SHALL consistently apply Tic Tac Toe rules and AI algorithms.

### 3.2.5 Maintainability

- **NFR-MAINT-001:** The codebase SHALL follow Google C++ Style Guidelines.
- **NFR-MAINT-002:** The code SHALL be modular and well-commented to facilitate future modifications and extensions.
- **NFR-MAINT-003:** The system SHALL be designed to allow for easy integration of new features (e.g., new game modes, AI algorithms).

#### 3.2.6 Portability

- **NFR-PORT-001:** The application SHALL be runnable on common desktop operating systems (e.g., Windows, macOS, Linux) due to the use of Qt framework.

#### 3.2.7 Testability

- **NFR-TEST-001:** The system SHALL be designed to allow for comprehensive unit and integration testing using Google Test.
- **NFR-TEST-002:** Key components (e.g., game logic, user authentication) SHALL be testable in isolation.

#### 3.2.8 Deployment

- **NFR-DEPLOY-001:** The project SHALL include CI/CD configurations (GitHub Actions) for automated testing and deployment.

## 4. Data Requirements

### 4.1 User Data

- **Username:** Unique string, alphanumeric with underscores.
- **Password Hash:** SHA-256 hash of the user's password.
- **Game History:** A list of game records, each containing:
  - Date and Time of game
  - Game Result (Win/Loss/Tie)
  - Game Mode (Player vs. Player / Player vs. AI)
  - AI Difficulty (if vs AI)
  - Sequence of moves (row, column, player)

### 4.2 Game State Data

- **Board State:** 3x3 matrix representing the Tic Tac Toe board, with each cell storing the player mark (X, O, or None).
- **Current Player:** Indicates whose turn it is (X or O).
- **Game Over Status:** Boolean indicating if the game has ended.
- **Winner:** Indicates the winning player (X, O, or None for a tie).
- **AI Difficulty:** Current difficulty setting for AI games.

## 5. System Architecture (High-Level)

### 5.1 Components

- **User Interface (GUI):** Handles all user interactions and displays game state.
- **User Authentication Module:** Manages user registration, login, and session.
- **Game Logic Module:** Implements core Tic Tac Toe rules, move validation, and win/tie detection.
- **AI Module:** Implements the Minimax algorithm with Alpha-Beta Pruning for AI moves.
- **Data Storage Module:** Manages saving and loading user data and game history.

### 5.2 Interactions

- The GUI interacts with the User Authentication module for login/signup and with the Game Logic module for game play.
- The Game Logic module interacts with the AI module for AI moves and with the Data Storage module to save game history.
- The User Authentication module interacts with the Data Storage module to save and load user credentials and history.

## 6. Future Enhancements

- Online multiplayer mode.
- More advanced AI algorithms.
- Customizable board sizes.
- User profiles with statistics.