Software Requirements Specification

for

Tic Tac Toe Game

Prepared by Team XOXO

<April 2025>

Submitted to: Dr. Omar Nasr

Revision History

Name	Date	Reason For Changes	Version
Team XOXO	April 2025	Initial draft of SRS for project	1.0

Table of Contents

1. Introduction

1.1 Purpose

This Software Requirements Specification (SRS) document outlines the functional and non-functional requirements for the Advanced Tic Tac Toe Game; a software project developed as part of a Spring 2025 embedded systems course. The document specifies the requirements for a C++-based Tic Tac Toe game with user authentication, an AI opponent, a graphical user interface, personalized game history, and CI/CD integration.

1.2 Document Conventions

- **Font Usage**: Bold text is used for section headings, and italic text emphasizes key terms.
- **Priority**: Requirements are labeled as High, Medium, or Low priority. High-priority requirements are critical for core functionality, while Medium and Low priorities indicate optional or secondary features.
- **Requirement IDs**: Each requirement is uniquely identified with a prefix (e.g., REQ-GAME, REQ-AI) followed by a number.

1.3 Intended Audience and Reading Suggestions

This SRS is intended for:

- **Developers**: To understand the system design and implementation requirements.
- **Testers**: To develop unit and integration tests based on specified requirements.
- **Project Managers**: To oversee project progress and ensure alignment with requirements.
- **Instructors**: To evaluate the project against academic standards.

1.4 Product Scope

The Advanced Tic Tac Toe Game is a desktop application that enhances the classic Tic Tac Toe game with user authentication, an AI opponent, and game history tracking. The system aims to provide engaging user experience, secure user management, and a robust development process using modern software engineering practices. The software supports both player-vs-player and player-vs-AI modes, with a focus on performance, reliability, and maintainability.

1.5 References

- Google C++ Style Guide: https://google.github.io/styleguide/cppguide.html
- Qt Documentation: https://doc.qt.io/
- SQLite Documentation: https://www.sqlite.org/docs.html
- GitHub Actions Documentation: https://docs.github.com/en/actions
- Google Test Documentation: https://github.com/google/googletest

2. Overall Description

2.1 Product Perspective

The Advanced Tic Tac Toe Game is a standalone desktop application built using C++ and Qt for GUI. It integrates with a lightweight database (SQLite) for user data and game history storage. The system is self-contained, with no dependencies on external systems beyond the specified tools and libraries.

2.2 Product Functions

- **Game Logic**: Implement a 3x3 Tic Tac Toe game with win/tie detection for player-vs-player and player-vs-AI modes.
- **User Management**: Provide secure registration, login, and profile management.
- **Game History**: Store and display game sessions, allowing users to review and replay past games.
- **AI Opponent**: Use the minimax algorithm with alpha-beta pruning for strategic AI moves.
- **GUI**: Offer an interactive interface for gameplay, user authentication, and history viewing.
- **Testing**: Ensure reliability through unit and integration tests.
- **CI/CD**: Automate testing and deployment using GitHub Actions.

2.3 User Classes and Characteristics

- **Player**: Registers, logs in, plays Tic Tac Toe (vs. another player or AI), and views game history.
- **Administrator**: Optional role for managing user accounts (if implemented).
- **Developer**: Maintains and tests the system, using GitHub for version control.

2.4 Operating Environment

- **Hardware**: Standard desktop or laptop with at least 4GB RAM and a 1GHz processor.
- **Operating System**: Windows 10/11, macOS, or Linux (Ubuntu 20.04 or later).
- **Software**: Qt framework (version 5.15 or later), SQLite (version 3 or later), Google Test, and Git for version control.

2.5 Design and Implementation Constraints

- The system must follow Google C++ Style Guidelines.
- Passwords must be hashed using a secure algorithm (e.g., bcrypt).
- The AI must compute moves within 1 second on standard hardware.
- The GUI must be responsive and compatible with Qt 5 or 6.
- The system must support at least 1,000 user accounts and 10,000 game records.

2.6 User Documentation

- **User Manual**: A guide detailing how to install, log in, play the game, and review game history.
- **Online Help**: Contextual help within the GUI, accessible via a "Help" button.
- **Developer Documentation**: Code comments and a README file in the Git repository explaining the codebase structure and setup.

2.7 Assumptions and Dependencies

- Users have a compatible desktop environment with Qt libraries installed.
- The SQLite database (or file storage) is lightweight and sufficient for data needs.

- GitHub Actions is available for CI/CD setup.
- Players are familiar with basic Tic Tac Toe rules.

3. External Interface Requirements

3.1 User Interfaces

- **Game Board**: A 3x3 grid displaying player moves (X or O) with clickable cells.
- **Login/Registration Forms**: Fields for username, password, and a submit button, with error messages for invalid inputs.
- **Game History View**: A table listing past games with details (e.g., opponent, outcome, date) and a replay option.
- **GUI Standards**: Consistent fonts, colors, and button styles based on Qt's default theme.

3.2 Hardware Interfaces

- The system requires a mouse or touchpad for GUI interaction and a standard display (minimum 1024x768 resolution).
- No specialized hardware is required beyond a standard desktop/laptop configuration.

3.3 Software Interfaces

- **Qt Framework**: Version 5.15 or later for GUI rendering and event handling.
- **SQLite**: Version 3 or later for storing user credentials and game history.
- **Google Test**: For unit and integration testing of game components.
- **Git**: For version control and interfacing with GitHub Actions.

3.4 Communications Interfaces

- No network communication is required, as the system is standalone.
- File-based communication may be used for custom storage solutions if SQLite is not employed.

4. System Features

4.1 Game logic

4.1.1 Description and Priority

The game logic manages the Tic Tac Toe gameplay for both player-vs-player and player-vs-AI modes. It is a High-priority feature, as it forms the core of the system.

4.1.2 Stimulus/Response Sequences

- **Stimulus:** Player clicks a cell on the 3x3 grid.
- **Response:** The system updates the board with the player's mark (X or O) and checks for a win, tie, or continuation.
- **Stimulus:** AI turn in player-vs-AI mode.
- **Response:** The system computes the AI's movement and updates the board.

4.1.3 Functional Requirements

- REQ-GAME-1: The system should maintain a 3x3 grid to store the state of the game (empty, X, or O).
- REQ-GAME-2: The system shall alternate between two players or a player and the AI.
- REQ-GAME-3: The system shall check for a win (three identical marks in a row, column, or diagonal) or a tie (all cells filled with no winner) after each move.
- REQ-GAME-4: The system shall display the game outcome (win, loss, or tie) and prompt the user to start a new game.

4.2 AI Opponent

4.2.1 Description and Priority

The AI opponent provides a challenging gameplay experience using a minimax algorithm with alpha-beta pruning. This is a High-priority feature.

4.2.2 Stimulus/Response Sequences

- **Stimulus**: Player selects player-vs-AI mode and makes a move.
- **Response**: The AI evaluates the board and selects an optimal move within 1 second.

4.2.3 Functional Requirements

- REQ-AI-1: The system shall implement a minimax algorithm with alpha-beta pruning for the AI opponent.
- REQ-AI-2: The AI shall make a move that maximizes its chance of winning or forces a tie.
- REQ-AI-3: The AI's move computation shall complete within 1 second on a standard system.

4.3 User Authentication and Management

4.3.1 Description and Priority

The system allows users to create accounts, log in, and manage sessions securely. This is a High-priority feature.

4.3.2 Stimulus/Response Sequences

- **Stimulus**: User submits registration form with username and password.
- **Response**: The system validates inputs, hashes the password, and stores the user in the database.
- **Stimulus**: User submits login credentials.
- **Response**: The system verifies credentials and grants access to the game.

4.3.3 Functional Requirements

- REQ-AUTH-1: The system shall provide a registration form for users to create accounts with a unique username and password.
- REQ-AUTH-2: The system shall hash passwords using a secure algorithm (e.g., bcrypt) before storage.
- REQ-AUTH-3: The system shall provide a login form to authenticate users.
- REQ-AUTH-4: The system shall maintain user sessions until logout or session timeout (30 minutes of inactivity).

4.4 Personalized Game History

4.4.1 Description and Priority

The system stores and displays game histories for logged-in users, including replay functionality. This is a Medium-priority feature.

4.4.2 Stimulus/Response Sequences

- **Stimulus**: User selects “View History” from the GUI.
- **Response**: The system displays a list of past games with details (e.g., opponent, outcome, date).
- **Stimulus**: User selects a game to replay.
- **Response**: The system animates the game’s moves in sequence.

4.4.3 Functional Requirements

- REQ-HIST-1: The system shall store game details (e.g., moves, opponent, outcome, date) for each completed game.
- REQ-HIST-2: The system shall display a table of game history for the logged-in user.
- REQ-HIST-3: The system shall allow users to replay past games by animating moves in sequence.

4.5 Graphical User Interface (GUI)

4.5.1 Description and Priority

The GUI provides an interactive interface for gameplay and user management. This is a High-priority feature.

4.5.2 Stimulus/Response Sequences

- **Stimulus**: User clicks a grid cell.
- **Response**: The system updates the cell with the player’s mark and refreshes the display.
- **Stimulus**: User interacts with login/registration/history forms.
- **Response**: The system processes inputs and updates the GUI accordingly.

4.5.3 Functional Requirements

- REQ-GUI-1: The system shall display a 3x3 grid with clickable cells for gameplay.
- REQ-GUI-2: The system shall provide forms for login, registration, and game history viewing.
- REQ-GUI-3: The system shall display error messages for invalid inputs (e.g., duplicate username, incorrect password).
- REQ-GUI-4: The system shall use the Qt framework for consistent rendering across platforms.

5. Other Nonfunctional Requirements

5.1 Performance Requirements

- The system shall respond to user inputs (e.g., clicks, form submissions) within 0.5 seconds.
- The AI shall compute moves within 1 second on a system with a 1GHz processor and 4GB RAM.
- The system shall support at least 100 concurrent user accounts without performance degradation.

5.2 Safety Requirements

No safety requirements apply, as the system poses no risk of physical harm.

5.3 Security Requirements

- Passwords shall be hashed using a secure algorithm (e.g., bcrypt) and never stored in plain text.
- User sessions shall be protected against unauthorized access using secure session management.
- The database shall be protected against SQL injection attacks.

5.4 Software Quality Attributes

- **Reliability**: The system shall have 99% uptime with no crashes during normal operation.
- **Maintainability**: Code shall follow Google C++ Style Guidelines for readability and modularity.
- **Usability**: The GUI shall be intuitive, requiring no more than 5 minutes for a new user to learn basic operations.
- **Testability**: All components shall be testable using Google Test, with at least 80% code coverage.

5.5 Business Rules

- Only authenticated users can access game history and player-vs-AI modes.
- The system shall enforce unique usernames for all registered users.

6. Other Requirements

- **Database**: The system shall use SQLite or a custom file-based storage system for user data and game history.
- **CI/CD**: The system shall integrate with GitHub Actions for automated testing and deployment.
- **Code Standards**: All code shall adhere to Google C++ Style Guidelines.
- **Documentation**: All third-party resources used shall be cited in the codebase and documentation.

Appendix A: Glossary

- **AI: Artificial Intelligence, referring to the computer-controlled opponent.**
- **Minimax Algorithm:** A decision-making algorithm used by the AI to choose optimal moves.
- **Alpha-Beta Pruning:** An optimization technique for the minimax algorithm to reduce computation time.
- **Qt:** A C++ framework for building cross-platform GUI applications.
- **SQLite:** A lightweight database for storing user data and game history.

Appendix B: Analysis Models

- **Class Diagram:** To be included in the Software Design Specification (SDS), detailing classes for game logic, AI, user management, and GUI components.
- **Sequence Diagram:** To be included in the SDS, showing interactions between the user, GUI, and backend components.