Software Requirements Specification (SRS)

Advanced Tic Tac Toe Game

Version: 1.0

Date: April 6, 2025

Prepared by: [Your Name]

Table of Contents

1. Introduction

1.1 Purpose

This Software Requirements Specification (SRS) document provides a detailed description of the requirements for the Advanced Tic Tac Toe game. It outlines the functional and non-functional requirements, constraints, assumptions, and dependencies that need to be addressed during the development process. This document serves as a foundation for the design, implementation, and testing phases of the project.

1.2 Document Conventions

The following conventions are used in this document:

- "SHALL" and "MUST" indicate mandatory requirements
- "SHOULD" indicates recommended but not mandatory requirements
- "MAY" indicates optional requirements
- Use case descriptions follow the format: [Actor] [Action] [Object]

1.3 Intended Audience and Reading Suggestions

This document is intended for:

- Project developers who will implement the system
- Testers who will verify the implementation meets requirements
- Instructors evaluating the project

- Future maintainers of the system

Developers should focus on Sections 3, 4, and 5, which detail system features and technical requirements. Testers should focus on Sections 3 and 6, which describe system features and testing requirements.

1.4 Project Scope

The Advanced Tic Tac Toe game is an enhanced version of the traditional Tic Tac Toe game that incorporates user authentication, game history tracking, and an intelligent AI opponent. The system will allow users to create accounts, play against other users or an AI, save their game history, and review past games. The project's scope includes:

- Development of a graphical user interface for game interaction
- Implementation of user authentication and profile management
- Creation of an AI opponent using minimax algorithm with alpha-beta pruning
- Game history tracking and replay functionality
- Comprehensive testing and continuous integration

The system is intended to be a standalone desktop application.

1.5 References

1. Project Assignment Document (Dr. Omar Nasr, April 2025)
2. IEEE Standard 830-1998, IEEE Recommended Practice for Software Requirements Specifications
3. Google C++ Style Guide (https://google.github.io/styleguide/cppguide.html)
4. Qt Framework Documentation (https://doc.qt.io/)

2. Overall Description

2.1 Product Perspective

The Advanced Tic Tac Toe game is a self-contained product that doesn't depend on other systems. It builds upon the classic Tic Tac Toe game by adding user authentication, game history, and an intelligent AI opponent. The system will be developed as a desktop application using C++ and the Qt framework for the GUI.

2.2 Product Functions

The main functions of the Advanced Tic Tac Toe game include:

- User registration and authentication
- Player vs. Player gameplay

- Player vs. AI gameplay with adjustable difficulty
- Game state tracking and validation
- Game history storage and retrieval
- Game replay functionality
- User profile management
- Performance metrics tracking

2.3 User Classes and Characteristics

The system will serve the following user classes:

1. **Registered Users**
   - Can log in with username and password
   - Can play against other users or AI
   - Can access and review their game history
   - Can update their profile information
2. **Guest Users**
   - Can play the game without logging in
   - Cannot save game history
   - Limited access to features
3. **Administrators**
   - Can manage user accounts
   - Can access system metrics and logs
   - Can configure system parameters

2.4 Operating Environment

The system shall operate in the following environment:

- Operating Systems: Windows 10/11, macOS 12+, Ubuntu 22.04+
- Hardware: Standard desktop or laptop computers with keyboard and mouse
- Screen Resolution: Minimum 1024x768
- Required Libraries: Qt Framework 6.2+, SQLite 3.36+

2.5 Design and Implementation Constraints

The following constraints shall apply to the project:

- Programming Language: C++17 or later
- GUI Framework: Qt
- Database: SQLite or custom file storage
- Version Control: Git with GitHub
- Testing Framework: Google Test
- CI/CD: GitHub Actions
- Code Style: Google C++ Style Guide
- Team Size: Up to 5 students
- Deadline: One week before the first final exam

2.6 User Documentation

The following user documentation shall be provided:

- Installation guide
- User manual
- Developer documentation
- API documentation for future extensions

2.7 Assumptions and Dependencies

The following assumptions have been made:

- Users have basic computer literacy
- Users are familiar with the rules of Tic Tac Toe
- The system will be used on desktop/laptop computers, not mobile devices
- Users have sufficient system privileges to install and run the application

Dependencies include:

- Qt Framework availability
- SQLite or similar database system
- Availability of C++ compiler compatible with the target platforms

3. System Features

3.1 Game Logic

3.1.1 Description

The game logic module implements the core rules and mechanics of Tic Tac Toe, allowing two players to take turns marking a 3x3 grid and determining the outcome of the game.

3.1.2 Functional Requirements

1. The system SHALL implement a 3x3 grid for the Tic Tac Toe board.
2. The system SHALL allow two players to take turns placing their marks (X or O) on an empty cell.
3. The system SHALL validate each move to ensure it is placed on an empty cell.
4. The system SHALL check for win conditions after each move (three marks in a row, column, or diagonal).
5. The system SHALL detect and declare a tie when all cells are filled without a winner.
6. The system SHALL track whose turn it is during gameplay.
7. The system SHALL provide an option to start a new game after completion.
8. The system MAY allow customization of player symbols beyond traditional X and O.

3.2 AI Opponent

3.2.1 Description

The AI opponent module implements an intelligent computer player that can make strategic moves in the game using algorithms such as minimax with alpha-beta pruning.

3.2.2 Functional Requirements

1. The system SHALL implement an AI player using the minimax algorithm with alpha-beta pruning.
2. The system SHALL allow players to play against the AI opponent.
3. The system SHALL ensure the AI makes valid moves according to game rules.
4. The system SHALL allow configuration of AI difficulty levels:
   - Easy: AI occasionally makes suboptimal moves
   - Medium: AI makes mostly optimal moves but may miss complex strategies
   - Hard: AI plays optimally using full minimax algorithm
5. The system SHALL ensure AI response time is reasonable (< 2 seconds per move).
6. The system MAY implement machine learning approaches to enhance AI behavior over time.

3.3 Graphical User Interface

3.3.1 Description

The GUI module provides a user-friendly interface for interacting with the game, displaying the game board, and accessing other system features.

3.3.2 Functional Requirements

1. The system SHALL display a visual representation of the 3x3 Tic Tac Toe board.
2. The system SHALL allow players to make moves by clicking on empty cells.
3. The system SHALL provide visual feedback when a cell is selected.
4. The system SHALL display whose turn it is during gameplay.
5. The system SHALL display the outcome of the game (win, loss, or tie).
6. The system SHALL provide navigation elements for accessing different sections of the application (game board, user profile, game history, etc.).
7. The system SHALL include forms for user registration and login.
8. The system SHALL provide a view for game history and replay functionality.
9. The system SHALL display relevant error messages for invalid actions.
10. The system SHALL provide visual indicators for the current game state.

3.4 User Authentication and Management

3.4.1 Description

The user authentication and management module handles user registration, login, session management, and profile updates.

3.4.2 Functional Requirements

1. The system SHALL allow users to register with a unique username and password.
2. The system SHALL securely store user credentials using password hashing.
3. The system SHALL authenticate users during login by comparing credentials with stored data.
4. The system SHALL manage user sessions to keep users logged in.
5. The system SHALL allow users to log out of their accounts.
6. The system SHALL allow users to update their profile information.
7. The system SHALL validate user input during registration and profile updates.
8. The system SHALL implement password reset functionality.
9. The system SHALL enforce strong password policies (minimum length, complexity).
10. The system MAY provide options for remembering login credentials.

3.5 Game History

3.5.1 Description

The game history module records, stores, and provides access to users' past games, allowing them to review and replay previous matches.

3.5.2 Functional Requirements

1. The system SHALL record the sequence of moves in each game.
2. The system SHALL store game outcomes (win, loss, tie) for each completed game.
3. The system SHALL associate game records with user profiles.
4. The system SHALL allow users to view their game history.
5. The system SHALL provide filtering and sorting options for game history.
6. The system SHALL enable users to replay previous games step by step.
7. The system SHALL display statistics based on game history (win rate, number of games played, etc.).
8. The system SHALL maintain game history data between user sessions.
9. The system MAY allow users to add notes or tags to saved games.
10. The system MAY provide an option to export game history data.

4. External Interface Requirements

4.1 User Interfaces

The user interface shall include the following screens and components:

1. **Login/Registration Screen**
   - Username and password fields
   - Login and register buttons
   - Password reset option
2. **Main Menu Screen**
   - Play game (PvP or PvAI) options
   - User profile access
   - Game history access
   - Settings option
   - Logout option
3. **Game Board Screen**

- 3x3 grid for gameplay
- Player turn indicator
- Game status display
- Options for new game, quit, save

4. **User Profile Screen**
   - Display user information
   - Edit profile option
   - Statistics summary
5. **Game History Screen**
   - List of past games with dates, opponents, and outcomes
   - Filtering and sorting options
   - Game replay option
6. **Settings Screen**
   - AI difficulty settings
   - UI preferences
   - Sound settings

The UI shall follow these design principles:

- Consistent color scheme and layout
- Clear visual feedback for user actions
- Intuitive navigation between screens
- Responsive design adapting to different window sizes
- Accessibility considerations (contrast, text size)

4.2 Hardware Interfaces

The system shall interface with the following hardware components:

1. **Display**
   - Support standard computer monitors
   - Minimum resolution of 1024x768
2. **Input Devices**
   - Mouse for point-and-click interaction
   - Keyboard for text input and shortcuts

3. **Storage**
   - Local disk storage for application data and user information
4. **Processing**
   - CPU for game logic and AI calculations

4.3 Software Interfaces

The system shall interface with the following software components:

1. **Operating System**
   - Windows 10/11, macOS 12+, Ubuntu 22.04+
   - Access to file system for data storage
2. **Qt Framework**
   - Version 6.2+
   - Used for GUI implementation and cross-platform compatibility
3. **Database System**
   - SQLite 3.36+ or custom file storage
   - Used for storing user data and game history
4. **C++ Standard Library**
   - Used for core functionality
   - Version conforming to C++17 or later

4.4 Communications Interfaces

As this is a standalone desktop application, no external communication interfaces are required for core functionality. However, the system shall:

1. Support future expansion to include network functionality
2. Implement file I/O for data persistence
3. Use appropriate error handling for all I/O operations

5. Non-functional Requirements

5.1 Performance Requirements

1. The system SHALL respond to user interactions within 100ms.
2. The AI opponent SHALL calculate its move within 2 seconds, even at the highest difficulty level.
3. The system SHALL load user data and initialize the application within 3 seconds of startup.

4. The system SHALL support saving and loading game history with minimal delay (< 1 second).
5. The system SHALL handle up to 1000 saved games per user without significant performance degradation.
6. The system SHALL efficiently use system resources (less than 200MB of RAM during normal operation).

5.2 Safety Requirements

1. The system SHALL handle unexpected errors gracefully without crashing.
2. The system SHALL save user progress automatically to prevent data loss in case of crash.
3. The system SHALL validate all user inputs to prevent unintended behavior.
4. The system SHALL confirm destructive actions (such as deleting saved games) before execution.

5.3 Security Requirements

1. The system SHALL store user passwords using secure hashing algorithms (e.g., bcrypt, Argon2).
2. The system SHALL encrypt sensitive data stored locally.
3. The system SHALL implement session timeout for inactive users.
4. The system SHALL validate all inputs to prevent injection attacks on data storage.
5. The system SHALL implement proper error handling to avoid leaking system information.
6. The system SHALL enforce secure password policies.

5.4 Software Quality Attributes

1. **Reliability**
   - The system SHALL maintain a mean time between failures (MTBF) of at least 100 hours of operation.
   - The system SHALL recover from crashes with minimal data loss.
2. **Usability**
   - New users SHALL be able to learn the game interface within 5 minutes without assistance.
   - The system SHALL provide clear feedback for all user actions.
   - The system SHALL follow user interface design best practices for clarity and ease of use.
3. **Maintainability**
   - The system SHALL follow object-oriented design principles.
   - The system SHALL use proper documentation for all classes and functions.
   - The system SHALL implement modular architecture to facilitate future enhancements.

4. **Portability**
    - The system SHALL run on Windows, macOS, and Linux without code modifications.
    - The system SHALL store data in a format that can be migrated between platforms.
5. **Testability**
    - The system SHALL be designed to facilitate automated testing.
    - The system SHALL achieve at least 80% code coverage in unit tests.

6. Other Requirements

6.1 Data Storage

1. The system SHALL store the following data:
    - User profiles (username, hashed password, creation date, last login)
    - Game records (date, players, sequence of moves, outcome)
    - User preferences and settings
    - Performance metrics
2. The system SHALL use either SQLite or a custom file format for data storage.
3. The system SHALL implement data integrity checks to prevent corruption.
4. The system SHALL backup user data periodically to prevent loss.

6.2 Testing Requirements

1. The system SHALL be tested using Google Test framework.
2. Unit tests SHALL cover all major components of the system.
3. Integration tests SHALL verify proper interaction between components.
4. UI testing SHALL verify the proper functioning of all interface elements.
5. Performance testing SHALL verify the system meets performance requirements.
6. Continuous integration SHALL run tests automatically on code updates.
7. The testing process SHALL achieve at least 80% code coverage.
8. Regression testing SHALL be performed after significant code changes.