Software Requirements Specification

For

Advanced Tic Tac Toe

Submitted to: Dr. Omar Nasr

Date: June 19, 2025

Table of Contents

1. Introduction 4
   1.1 Purpose 4
   1.2 Scope 4
   1.3 Definitions, Acronyms, and Abbreviations 4
   1.4 Overview 4
2. Overall Description 4
   2.1 Product Perspective 4
   2.2 Product Functions 4
   2.3 User Classes and Characteristics 4
   2.4 Operating Environment 5
   2.5 Design and Implementation Constraints 5
   2.6 User Documentation 5
   2.7 Assumptions and Dependencies 5
3. System Features 5
   3.1 Game Logic 3.1.1 Description and Priority 5
      3.1.2 Functional Requirements 5
      3.1.3 Game Flow 6
      3.1.4 Error Handling 6
      3.1.5 Testing 6
   3.2 Artificial Intelligence 6
      3.2.1 Description and Priority 6
      3.2.2 Functional Requirements 6
      3.2.3 External Interfaces 6
      3.2.4 Game Flow 7
      3.2.5 Error Handling 7
      3.2.6 Testing 7
   3.3 Graphical User Interface 8
      3.3.1 Description and Priority 8
      3.3.2 Functional Requirements 8
      3.3.3 External Interfaces 8
      3.3.4 Game Flow 8
      3.3.5 Error Handling 8
      3.3.6 Testing 8
   3.4 User System & Persistence 8
      3.4.1 Description and Priority 8
      3.4.2 Functional Requirements 9
      3.4.3 Interfaces and Data Storage Overview 9

3.4.4 Error Handling 9
3.4.5 Security Considerations 9
3.4.6 User Interaction Flow 9
3.5 QA & DevOps 10
3.5.1 Description and Priority 10
3.5.2 Functional Requirements 10
3.5.3 External Interfaces 10
3.5.4 Game Flow 10
3.5.5 Error Handling 10
3.5.6 Testing 10
4. Non-functional Requirements 10
5. Appendices 10

1. Introduction

1.1 Purpose

This document outlines the requirements for the Advanced Tic Tac Toe Game, providing a clear description of the system's functional and non-functional aspects for both developers and stakeholders.

1.2 Scope

The application is an enhanced version of Tic Tac Toe, supporting player-vs-player and player-vs-AI modes, user authentication, persistent history, and a user-friendly interface.

1.3 Definitions, Acronyms, and Abbreviations

- **AI**: Artificial Intelligence
- **GUI**: Graphical User Interface
- **SQLite**: A lightweight, embeddable database engine.
- **Minimax Algorithm**: An algorithm used in decision-making and game theory.

1.4 Overview

This document details the system's requirements, design constraints, and interfaces, serving as a guide for development and quality assurance.

2. Overall Description

2.1 Product Perspective

The game is a standalone desktop application, using C++ and Qt, and integrates modules for logic, AI, GUI, user management, and testing.

2.2 Product Functions

- Play Tic Tac Toe (PvP and PvE)
- Move validation, win/tie detection
- AI opponent with difficulty settings
- User authentication and history
- Game replay and statistics

2.3 User Classes and Characteristics

- Players: Use the game, log in, play, and review history
- Developers: Maintain and extend the system

2.4 Operating Environment

- Windows/Linux/MacOS
- C++17+
- Qt Framework

2.5 Design and Implementation Constraints

- Must use C++ and Qt
- Modular architecture
- Secure password storage
- SQLite or custom file storage

2.6 User Documentation

- User manual
- In-app help

2.7 Assumptions and Dependencies

- Standard C++ and Qt libraries
- No external dependencies for core logic

3. System Features

3.1 Game Logic

3.1.1 Description and Priority

This module manages the basic rules and flow of the Tic Tac Toe game. It ensures that the board is correctly set up, players take turns, moves are valid, and the outcome of the game (win, draw, or ongoing) is determined and communicated. This is a core component and is essential for the game to function.

3.1.2 Functional Requirements

- **FR1:** Initialize a 3x3 board.
- **FR2:** Alternate turns between X and O.
- **FR3:** Validate moves (in range, not occupied).
- **FR4:** Record move history (position, player, comment).
- **FR5:** Undo last move.
- **FR6:** Detect win (rows, columns, diagonals).

- **FR7:** Detect draw (full board, no winner).
- **FR8:** Provide board state and move history to other modules.

3.1.3 Game Flow

At the start, the board is empty. Players take turns choosing where to place their mark. The system checks if the move is allowed, updates the board, and records the move. After each move, it checks for a win or draw. If the game is not over, play continues. Players can undo their last move, which reverts the board and turn.

3.1.4 Error Handling

Invalid moves (such as choosing an occupied cell) are not accepted. Undo is only possible if there is at least one move in the history.

3.1.5 Testing

- Unit tests for all public methods (move validation, win/draw, undo).
- Edge cases: repeated moves, immediate win, full board.

3.2 Artificial Intelligence

3.2.1 Description and Priority

Implements AI opponent using Minimax with alpha-beta pruning. Supports multiple difficulty levels and provides move explanations. This module provides the computer opponent for the game. It is responsible for analyzing the current board and making moves on behalf of the AI player. The AI can play at different difficulty levels, from random moves to strategic play.

3.2.2 Functional Requirements

- **FR1:** Selects optimal move using Minimax.
- **FR2:** Supports "easy", "medium", "hard" difficulty.
    - On easy, the AI may make random or less optimal moves.
    - On higher difficulties, the AI analyzes the board to make the best possible move.
- **FR3:** Provides explanation for each move.
- **FR4:** Logs AI decisions for transparency.
- **FR5:** Interfaces with game logic for move execution.

3.2.3 External Interfaces

The AI receives the current board and player information and returns its chosen move. It may also provide a brief explanation of its decision.

3.2.4 Game Flow

When it is the AI's turn, it evaluates the board and selects a move based on the chosen difficulty. The move is then executed as if a player had made it. If explanations are enabled, the AI describes its reasoning.

3.2.5 Error Handling

If the AI cannot find a valid move (which should not happen in normal play), it will notify the system.

3.2.6 Testing

The AI's decisions are tested for validity, and its ability to win or block the player is verified at higher difficulty levels.

3.3 Graphical User Interface

3.3.1 Description and Priority

Implements all user interactions using Qt, including board display, move input, login/register forms, and game history visualization.

3.3.2 Functional Requirements

- **FR1:** Display 3x3 board and update on moves.
- **FR2:** Provide buttons for login, register, history, and undo.
- **FR3:** Visual cues for active player, win/tie state.
- **FR4:** Show move comments and AI explanations.
- **FR5:** Integrate with game logic and user system.

3.3.3 External Interfaces

The GUI communicates with the game logic, AI, and user system to update the display and process user actions.

3.3.4 Game Flow

When the game starts, the user sees the board and options. As the user interacts, the GUI updates the display and sends actions to the appropriate modules. Results (like win or draw) are shown immediately.

3.3.5 Error Handling

The GUI provides clear error messages for invalid actions, such as trying to move in an occupied cell or submitting incomplete forms.

3.3.6 Testing

The interface is tested for usability, responsiveness, and correct display of game states.

3.4 User System & Persistence

3.4.1 Description and Priority

This module manages user authentication and persistent game data using an SQLite database. It securely handles user credentials, tracks gameplay history, stores detailed game moves, and computes statistics. This is essential for providing a personalized and competitive experience across sessions.

3.4.2 Functional Requirements

- **FR1:** Allow users to register with a unique username and password.
- **FR2:** Authenticate users using SHA-256 hashed passwords.
- **FR3:** Store each completed game's metadata (players, winner).
- **FR4:** Save individual game moves and optional AI-generated comments.
- **FR5:** Retrieve user-specific game history and detailed move logs.
- **FR6:** Calculate head-to-head statistics and Human vs AI statistics.

3.4.3 Interfaces and Data Storage Overview

The user system interfaces with the following components:

- GUI module for login, registration, and history display.

- Game logic module for storing completed matches.

Persistent storage is handled using SQLite. The database schema includes:

- A 'users' table for storing usernames and password hashes.

- A 'game_history' table for storing match metadata (players and result).

- A 'game_moves' table for recording the sequence of moves and optional comments.

3.4.4 Error Handling

The system shall gracefully handle the following error scenarios:

- Registration fails if the username already exists.

- Login fails if the password hash does not match.

- Any database operation that fails to prepare or execute returns false and aborts the transaction if in progress and for vectors it returns vector of zeros.

3.4.5 Security Considerations

User passwords are never stored in plaintext. Instead, they are hashed using SHA-256 before storage.

The system uses parameterized database queries to mitigate SQL injection attacks.
Sensitive operations are only performed after verifying the session state to prevent unauthorized access.

Session integrity is assumed to be managed at the UI level.

3.4.6 User Interaction Flow

1 - The user opens the application and is prompted to log in or register.

2 - Upon successful login, the user gains access to gameplay and personalized features.

3 - At the end of a match, the game result and all moves are stored persistently.

4 - The user can later review past games from the history screen, including move-by-move playback.

5 - The user may also access statistical summaries (e.g., total wins vs AI, head-to-head records).

3.5 QA & DevOps

3.5.1 Description and Priority

Ensures software quality through automated testing and CI/CD integration.

3.5.2 Functional Requirements

- **FR1:** Unit and integration tests using Google Test.
- **FR2:** Set up CI/CD with GitHub Actions.
- **FR3:** Collect and report code coverage.
- **FR4:** Document test strategy and performance benchmarks.

3.5.3 External Interfaces

This module interacts with the code repository and build systems to run tests and deploy updates.

3.5.4 Game Flow

As developers update the code, automated systems test and build the application. Any issues found are reported for correction before release.

3.5.5 Error Handling

Failed tests or builds are reported with detailed information to help developers fix problems quickly.

3.5.6 Testing

This module is responsible for maintaining and updating the overall test suite and ensuring all parts of the application are tested.

4. Non-functional Requirements

- **Performance:** Move processing < /////delay_time///// ms, fast UI updates.
- **Reliability:** No crashes on invalid input.
- **Usability:** Clear error messages, intuitive GUI.
- **Security:** Passwords hashed, user data protected.
- **Portability:** Runs on Windows, Linux, MacOS.

5. Appendices

- **A. Example Board States**
- **B. Test Cases**
- **C. Glossary**