Software Requirements Specification

for

< Advanced Tic Tac Toe >

Version 1.0 approved

Prepared by <Team>

<CUFE-EECE>

<20/6/2025>

Table of Contents

Revision History

Name	Date	Reason For Changes	Version

1. Introduction

1.1 Purpose

This Software Requirements Specification (SRS) outlines the functional and non-functional requirements for the Advanced Tic Tac Toe application, a C++ desktop game developed for the "Data Structures - Embedded Systems" course (Spring 2025). The application, built using the Qt framework, enables users to play Tic Tac Toe against an AI opponent with adjustable difficulty levels or another human player, with secure user authentication and personalized game history tracking. This SRS defines the system's behavior, interfaces, and constraints for developers, testers, instructors, and team members.

1.2 Document Conventions

- ***Font:*** *Normal text uses standard font; bold denotes section headings and key terms.*
- ***Requirement IDs:*** *Functional requirements are prefixed with "REQ-" (e.g., REQ-AUTH-1), and non-functional requirements use specific prefixes (e.g., PERF-1 for performance).*
- ***Priority:*** *Each system feature includes a priority (High, Medium, Low). High-priority features are critical for core functionality (e.g., game play, authentication).*
- ***Terminology:*** *Terms like "user" and "AI" are defined in Appendix A: Glossary.*
- ***TBD:*** *Used for placeholders where information is pending; tracked in Appendix C.*

1.3 Intended Audience and Reading Suggestions

This SRS is intended for:

- ***Developers:*** *To understand the system's requirements and implement the codebase. Start with Section 4: System Features and Section 3: External Interface Requirements.*
- ***Testers:*** *To design test cases for functional and non-functional requirements. Focus on Section 4: System Features and Section 5: Other Nonfunctional Requirements.*
- ***Instructors:*** *To evaluate the project's alignment with course objectives. Begin with Section 1.4: Product Scope and Section 2: Overall Description.*
- ***Team Members:*** *To ensure all members understand all deliverables, as required by the project guidelines. Read sequentially from Section 1 to Section 6.*

1.4 Product Scope

The Advanced Tic Tac Toe application is a desktop game that enhances the classic 3x3 Tic Tac Toe with:

- ***User Authentication:*** *Secure login and registration with SHA-256 password hashing.*
- ***Game Modes:*** *Single-player (vs. AI with Easy, Medium, Hard, Impossible difficulties) and two-player (human vs. human) modes.*
- ***AI Opponent:*** *Strategic AI using a minimax algorithm for optimal play in Impossible mode.*
- ***Game History:*** *Persistent storage and display of game outcomes (winner, opponent, date, board state).*
- ***GUI:*** *A Qt-based interface with login, game, and history screens.*

Objectives:

- *Provide an engaging, user-friendly gaming experience.*
- *Ensure secure user management and data storage using SQLite.*
- *Meet course requirements for object-oriented programming, data structures (e.g., trees for minimax), and software engineering practices (e.g., testing, CI/CD).*

Benefits:

- *Enhances player engagement with challenging AI and personalized history.*
- *Demonstrates proficiency in C++, Qt, SQLite, and secure coding practices.*
- *Aligns with academic goals of rigorous software development and documentation*.

1.5 References

1.5 References

- ***Qt Documentation:*** *https://doc.qt.io/ (Version 6.9.0, accessed April 2025).*
- ***SQLite Documentation:*** *https://www.sqlite.org/docs.html (Version 3.x, accessed April 2025).*
- ***OpenSSL Documentation:*** *https://www.openssl.org/docs/ (Version 1.x, accessed April 2025).*
- ***IEEE SRS Standard:*** *IEEE Std 830-1998, "Recommended Practice for Software Requirements Specifications."*
- ***Google C++ Style Guide:*** *https://google.github.io/styleguide/cppguide.html (for coding standards).*

2. Overall Description

2.1 Product Perspective

The Advanced Tic Tac Toe application is a new, self-contained desktop application developed for Windows. It is not part of an existing product family but is designed as a standalone system for the course project. The system integrates:

- ***Game Logic:*** *A C++ module for Tic Tac Toe mechanics and AI (minimax algorithm).*
- ***Database:*** *SQLite for storing user accounts and game history.*
- ***Security:*** *OpenSSL for SHA-256 password hashing.*
- ***GUI:*** *Qt framework for a responsive, widget-based interface*

  The system operates locally, with no external system dependencies beyond the Windows environment and specified libraries..

2.2 Product Functions

- ***User Authentication:*** *Register, log in, and log out with secure credentials.*
- ***Game Modes:*** *Play against an AI (with selectable difficulty) or another human.*
- ***Game Logic:*** *Manage a 3x3 board, validate moves, detect wins/draws, and alternate players.*
- ***AI Opponent:*** *Provide strategic moves using random, heuristic, or minimax-based algorithms.*
- ***Game History:*** *Save and display game outcomes, including date, result, opponent, and board state.*
- ***GUI Navigation:*** *Switch between login, game, and history screens with menus and toolbars.*

2.3 User Classes and Characteristics

- ***Casual Players (Primary, High Importance****):*
  - ***Characteristics:*** *General users with basic computer literacy, seeking an engaging game.*
  - ***Usage:*** *Frequent use of game play, authentication, and history features.*
  - ***Requirements:*** *Intuitive GUI, clear feedback (e.g., error messages), adjustable AI difficulty.*
- ***Developers/Testers (Secondary, Medium Importance):***

- ***Characteristics:*** *Team members or instructors with C++, Qt, and SQLite knowledge.*
- ***Usage:*** *Maintain code, test functionality, and access the database for debugging.*
- ***Requirements:*** *Modular code, error logging, and documented database schema*.

2.4 Operating Environment

- ***Hardware:*** *Windows PC with ≥4GB RAM, ≥500MB disk space, and ≥800x600 display resolution.*
- ***Operating System:*** *Windows 10/11 (64-bit).*
- ***Software:***
  - ***Qt 6.9.0:*** *Core, gui, widgets, and sql modules.*
  - ***SQLite:*** *Amalgamation source included in the project.*
  - ***OpenSSL:*** *Crypto and ssl libraries (installed at C:/msys64/mingw64/).*
  - ***MinGW:*** *64-bit compiler for building the application.*

  The system coexists with standard Windows applications and requires no additional runtime environments.

2.5 Design and Implementation Constraints

- ***Language:*** *C++11, adhering to Google C++ Style Guidelines.*
- ***GUI Framework:*** *Qt 6.9.0, restricting the interface to widget-based design.*
- ***Database:*** *SQLite, using a single file (tictactoe.db) for local storage.*
- ***Security:*** *Passwords must be hashed with SHA-256 using OpenSSL's EVP API.*
- ***Data Structures:*** *Trees for the minimax algorithm; vectors for the game board.*
- ***Singleton Pattern:*** *DBManager class ensures a single SQLite connection.*
- ***Build Environment:*** *MinGW 64-bit, with specific library paths.*

2.6 User Documentation

- ***Inline GUI Feedback:*** *Error messages (e.g., "Invalid username or password") and status updates (e.g., "Player X's turn").*
- ***No Separate Manual:*** *The GUI is self-explanatory, with intuitive controls (e.g., clickable board buttons, menu options).*

- ***Developer Documentation:*** *Code comments and this SRS provide implementation details.*

2.7 Assumptions and Dependencies

- ***Assumptions:***
  - *Users have basic computer skills to navigate the GUI.*
  - *The SQLite database file (tictactoe.db) is not modified externally.*
  - *All team members understand the codebase, as required for the oral presentation.*
- ***Dependencies:***
  - *Correct installation of Qt 6.9.0, SQLite, and OpenSSL.*
  - *Availability of MinGW 64-bit for compilation.*
  - *GitHub for version control and CI/CD (via GitHub Action).*

3. External Interface Requirements

3.1 User Interfaces

- ***Login Screen:***
  - ***Fields:*** *Username (text), password (masked).*
  - ***Buttons:*** *“Login” (blue), “Sign Up” (green).*
  - ***Styling:*** *White form background, blue title (“Advanced Tic Tac Toe”), rounded corners.*
  - ***Feedback:*** *Error messages for empty/invalid credentials.*
- ***Game Screen:***
  - ***Board:*** *3x3 grid of buttons (120x120 pixels, gradient background: blue for ‘X’, red for ‘O’).*
  - ***Labels:*** *Status (“Player X’s turn”), difficulty (e.g., “AI Difficulty: Hard”).*
  - ***Navigation:*** *Menu bar (“Game,” “View,” “Account”) and toolbar (“New Game,” “View History,” “Logout”).*
- ***History Screen:***
  - ***Table:*** *Columns for Date, Result, Opponent, Board State; alternating row colors, blue header.*
  - ***Button:*** *“Back to Game” (blue, centered).*

- ***Dialogs:***
    - ***Difficulty Dialog:*** *Combo box (Easy, Medium, Hard, Impossible), OK/Cancel buttons.*
    - ***Message Boxes:*** *For game results (e.g., "AI wins!") and errors, styled with Qt stylesheets.*
    - ***Player Choose Dialog:*** *For the player to choose X OR O*

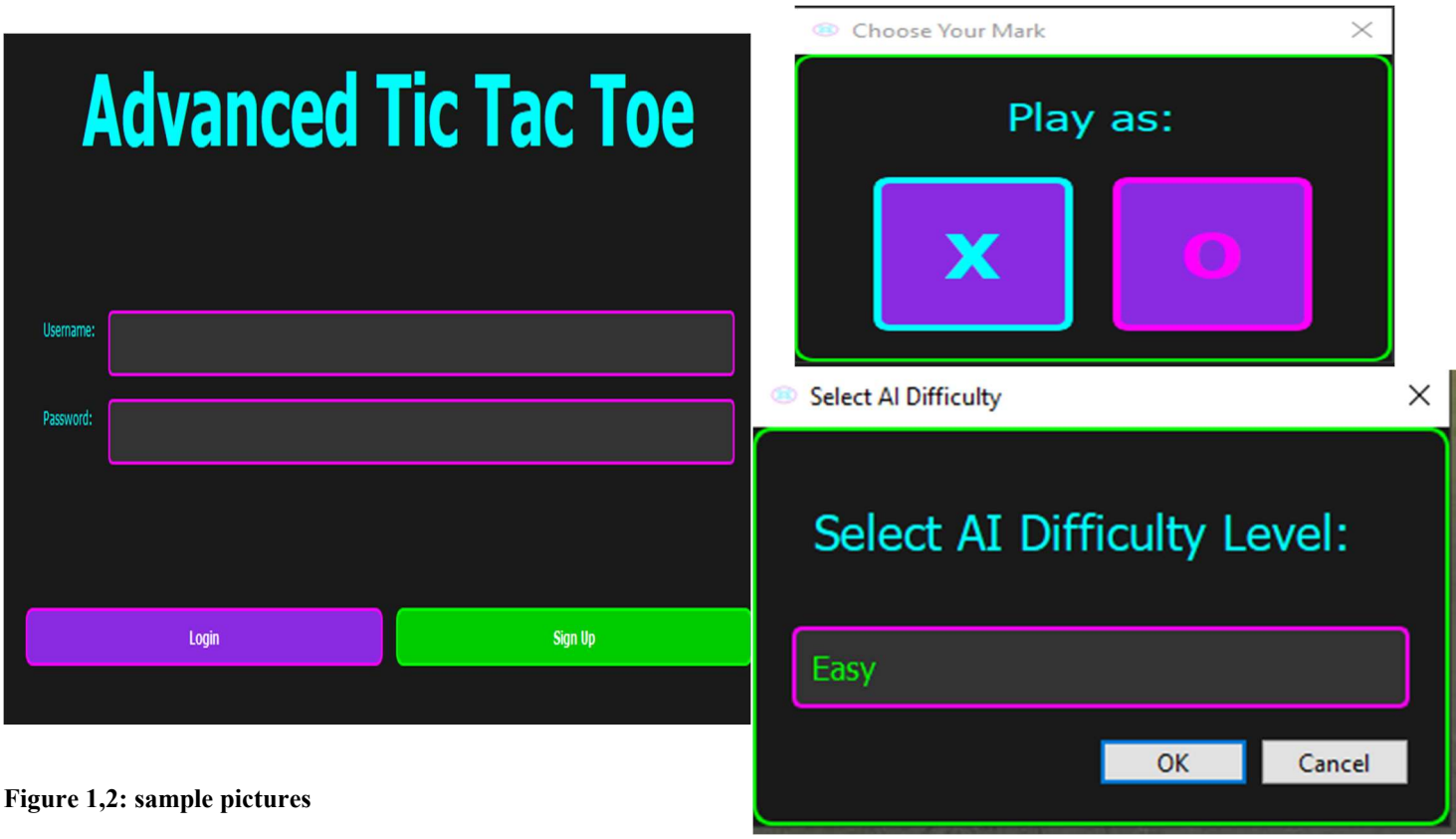

Figure 1,2: sample pictures

3.2 Hardware Interfaces

- ***Input:*** *Mouse for clicking buttons and keyboard for text input (username/password).*
- ***Output:*** *Display for rendering the Qt GUI (minimum 800x600 resolution).*

3.3 Software Interfaces

- ***Qt 6.9.0:***
    - ***Purpose:*** *Provides GUI (QMainWindow, QStackedWidget, QPushButton) and SQL module for SQLite access.*
    - ***Version:*** *6.9.0.*
    - ***Data:*** *Widgets handle user inputs; SQL module queries tictactoe.db.*

- ***SQLite:***
    - ***Purpose:*** *Stores user accounts (users table: username, hashed password, timestamp) and game history (game_history table: id, username, result, opponent, date, board state).*
    - ***Version****: Amalgamation source (included in project).*
    - ***Data:*** *Parameterized queries for CRUD operations.*
- ***OpenSSL:***
    - ***Purpose:*** *Implements SHA-256 hashing for passwords via EVP API.*
    - ***Version:*** *1.x*
    - ***Data:*** *Converts plaintext passwords to 32-byte hashes.*
- ***MinGW:***
    - ***Purpose:*** *Compiles the application, linking Qt, SQLite, and OpenSSL.*
    - ***Version:*** *64-bit.*

3.4 Communications Interfaces

- *None, as the application operates locally without network requirements.*

4. System Features

4.1 User Authentication

4.1.1 Description and Priority

- ***Description:*** *Allows users to register, log in, and log out, managing secure access to personalized game history and settings.*
- *Priority: High (core feature for user management and history tracking).*

4.1.2 Stimulus/Response Sequences

- ***Register:***
    - ***Stimulus:*** *User enters username/password and clicks “Sign Up.”*

- ***Response:*** *System validates inputs, hashes password, stores user in database, and shows "Account created" message.*

- ***Login:***
  - ***Stimulus:*** *User enters username/password and clicks "Login."*
  - ***Response:*** *System verifies credentials, switches to game screen, and enables menu/toolbar options.*
- ***Logout:***
  - ***Stimulus:*** *User selects "Logout" from menu/toolbar.*
  - ***Response:*** *System clears session, returns to login screen, and disables menu/toolbar options.*

4.1.3 Functional Requirements

- ***REQ-AUTH-1:*** *The system shall allow users to register with a unique username (string, 1–50 characters) and password (string, 1–50 characters).*
- ***REQ-AUTH-2:*** *The system shall hash passwords using SHA-256 and store them in the users table.*
- ***REQ-AUTH-3:*** *The system shall prevent registration if the username exists, displaying "Username already exists."*
- ***REQ-AUTH-4:*** *The system shall authenticate users by comparing entered credentials with stored hashed passwords.*
- ***REQ-AUTH-5:*** *The system shall display "Username and password cannot be empty" for empty login/register inputs.*
- ***REQ-AUTH-6:*** *The system shall display "Invalid username or password" for incorrect login credentials.*
- ***REQ-AUTH-7:*** *The system shall allow logged-in users to log out, clearing the session and returning to the login screen.*

4.2 Game Initialization

4.2.1 Description and Priority

- ***Description:*** *Sets up a new Tic Tac Toe game, allowing users to choose between single-player (vs. AI) or two-player (vs. human) modes and configure AI difficulty.*
- ***Priority:*** *High (essential for starting gameplay).*

4.2.2 Functional Requirements

- ***Select Mode:***
    - ***Stimulus:*** *User selects "Play vs AI" or "Play vs Human" from the menu.*
    - ***Response:*** *System initializes a new game with the chosen mode.*
- ***Select AI Difficulty:***
    - ***Stimulus:*** *User selects "Play vs AI" and chooses a difficulty (Easy, Medium, Hard, Impossible).*
    - ***Response:*** *System sets the AI difficulty and displays it (e.g., "AI Difficulty: Hard").*
- ***New Game:***
    - ***Stimulus:*** *User selects "New Game" from menu/toolbar.*
    - ***Response:*** *System resets the board to empty and sets 'X' as the first player.*

4.2.3 Functional Requirements

- ***REQ-INIT-1:*** *The system shall provide menu options for "Play vs AI" and "Play vs Human."*
- ***REQ-INIT-2:*** *For single-player mode, the system shall display a dialog to select AI difficulty (Easy, Medium, Hard, Impossible).*
- ***REQ-INIT-3:*** *The system shall initialize a 3x3 game board with all cells empty.*
- ***REQ-INIT-4:*** *The system shall set 'X' as the first player for all new games.*
- ***REQ-INIT-5:*** *The system shall reset the board and status label ("Player X's turn") when "New Game" is selected.*

4.3 Gameplay

4.3.1 Description and Priority

- ***Description:*** *Manages the core Tic Tac Toe gameplay, including move placement, validation, turn alternation, and win/draw detection.*
- ***Priority:*** *High (core gameplay feature).*

4.3.2 Stimulus/Response Sequences

- ***Player Move:***

- ***Stimulus:*** *User clicks an empty board button.*
- ***Response:*** *System places the current player's symbol ('X' or 'O'), updates the* ***board, and checks for win/draw.***

- ***Win/Draw:***
  - ***Stimulus:*** *A move results in three identical symbols in a row/column/diagonal or a full board.*
  - ***Response:*** *System displays the result (e.g., "Player X wins!"), disables the board, and saves the game.*
- ***Turn Switch:***
  - ***Stimulus:*** *A valid move is made.*
  - ***Response:*** *System switches to the next player and updates the status label.*

4.3.3 Functional Requirements

- ***REQ-PLAY-1:*** *The system shall allow the current player to place their symbol ('X' or 'O') in an empty cell by clicking a button.*
- ***REQ-PLAY-2:*** *The system shall reject moves in occupied cells, maintaining the current state.*
- ***REQ-PLAY-3:*** *The system shall alternate turns between 'X' and 'O' after each valid move.*
- ***REQ-PLAY-4:*** *The system shall detect a win when three identical symbols appear in a row, column, or diagonal.*
- ***REQ-PLAY-5:*** *The system shall detect a draw when all nine cells are filled without a winner.*
- ***REQ-PLAY-6:*** *The system shall display the game result in a message box and status label (e.g., "Player X wins!", "It's a draw!").*
- ***REQ-PLAY-7****: The system shall disable all board buttons after a win or draw.*
- ***REQ-PLAY-8:*** *The system shall update the board's appearance (e.g., blue 'X', red 'O') after each move.*

4.4 AI opponent

4.4.1 Description and Priority

- ***Description:*** *Provides a computer opponent for single-player mode, with four difficulty levels (Easy, Medium, Hard, Impossible) using random, heuristic, or minimax-based strategies.*

- ***Priority:*** *High (key feature for engaging gameplay).*

4.4.2 Stimulus/Response Sequences

- ***AI Move:***
    - ***Stimulus:*** *It's the AI's turn ('O') in single-player mode.*
    - ***Response:*** *System computes and places an AI move, updates the board, and checks for win/draw after a 500ms delay.*
- ***Difficulty Selection:***
    - ***Stimulus:*** *User selects an AI difficulty.*
    - ***Response:*** *System configures the AI's behavior (e.g., random for Easy, minimax for Impossible).*

4.4.3 Functional Requirements

- ***REQ-AI-1:*** *The system shall play as 'O' in single-player mode, with the user as 'X'.*
- ***REQ-AI-2:*** *For Easy difficulty, the AI shall select a random empty cell.*
- ***REQ-AI-3:*** *For Medium difficulty, the AI shall make a random move 60% of the time and an optimal move (win or block) 40% of the time.*
- ***REQ-AI-4:*** *For Hard difficulty, the AI shall prioritize center/corners and make optimal moves 80% of the time.*
- ***REQ-AI-5:*** *For Impossible difficulty, the AI shall use a minimax algorithm to select optimal moves, ensuring a win or draw.*
- ***REQ-AI-6:*** *The system shall display AI moves with a 500ms delay to simulate thinking.*
- ***REQ-AI-7:*** *The system shall update the difficulty label (e.g., "AI Difficulty: Impossible") when playing against AI.*

4.5 Game history

4.5.1 Description and Priority

- ***Description:*** *Saves and displays a user's game history, including date, result, opponent, and final board state.*
- ***Priority:*** *Medium (enhances personalization but not critical for gameplay).*

4.5.2 Stimulus/Response Sequences

- ***Save Game:***
    - ***Stimulus:*** *A game ends (win or draw).*
    - ***Response:*** *System saves the outcome to the database and updates the history table.*
- ***View History:***
    - ***Stimulus:*** *User selects "View History" from menu/toolbar.*
    - ***Response:*** *System displays the history screen with a table of past games.*

4.5.3 Functional Requirements

- ***REQ-HIST-1:*** *The system shall save each game's outcome to the game_history table, including username, result (e.g., "username won," "AI won," "Draw"), opponent (AI or Human), date (YYYY-MM-DD HH:MM:SS), and board state (9-character string).*
- ***REQ-HIST-2:*** *The system shall display a user's game history in a table with columns: Date, Result, Opponent, Board State.*
- ***REQ-HIST-3:*** *The system shall refresh the history table after each game and when the history screen is accessed.*
- ***REQ-HIST-4:*** *The system shall allow users to return to the game screen from the history screen via a "Back to Game" button.*

4.6 User interface Navigation

4.6.1 Description and Priority

- ***Description:*** *Provides seamless navigation between login, game, and history screens, with menu and toolbar controls.*
- ***Priority:*** *High (essential for user experience).*

4.6.2 Stimulus/Response Sequences

- ***Screen Switch:***
    - ***Stimulus:*** *User logs in, selects "View History," or clicks "Back to Game."*
    - ***Response:*** *System switches to the corresponding screen (login, game, history).*
- ***Menu/Toolbar Action:***

- ***Stimulus:*** *User selects an option (e.g., "New Game," "Logout").*
- ***Response:*** *System performs the action (e.g., resets board, clears session).*

4.6.3 Functional Requirements

- ***REQ-NAV-1:*** *The system shall use a QStackedWidget to switch between login, game, and history screens.*
- ***REQ-NAV-2:*** *The system shall provide a menu bar with options: "New Game," "Play vs AI," "Play vs Human," "View History," "Logout."*
- ***REQ-NAV-3:*** *The system shall provide a toolbar with buttons: "New Game," "View History," "Logout."*
- ***REQ-NAV-4:*** *The system shall disable game-related menu/toolbar options (e.g., "New Game") until a user logs in.*
- ***REQ-NAV-5:*** *The system shall enable all menu/toolbar options after successful login.*

5. Other Nonfunctional Requirements

5.1 Performance Requirements

- ***PERF-1:*** *The system shall process user inputs (e.g., button clicks) within 100ms, excluding AI move delays.*
- ***PERF-2:*** *The minimax algorithm for Impossible difficulty shall compute a move within 1 second.*
- ***PERF-3:*** *Database operations (e.g., save game, load history) shall complete within 200ms.*
- ***PERF-4:*** *Screen transitions (e.g., login to game) shall render within 500ms.*

5.2 Safety Requirements

- ***SAFE-1:*** *The system shall not cause data loss or corruption in the SQLite database.*
- ***SAFE-2:*** *The system shall handle invalid inputs (e.g., empty username) gracefully, preventing crashes.*

5.3 Security Requirements

- ***SEC-1:*** *The system shall store passwords as SHA-256 hashes in the users table, not plaintext.*
- ***SEC-2:*** *The system shall use parameterized SQLite queries to prevent SQL injection.*
- ***SEC-3:*** *The system shall restrict access to game and history features to authenticated users.*

5.4 Software Quality Attributes

- ***QUAL-1: Maintainability****: The codebase shall be modular (e.g., separate Game, DBManager, User, MainWindow classes) with comments for key functions.*
- ***QUAL-2: Usability:*** *The GUI shall use a consistent color scheme (blue for 'X', red for 'O', grey for neutral) and readable fonts (≥14px).*
- ***QUAL-3: Reliability:*** *The system shall log database errors to the console and display user-friendly messages, avoiding crashes.*
- ***QUAL-4: Portability:*** *The system shall compile and run on Windows 10/11 with Qt 6.9.0 and MinGW 64-bit.*
- ***QUAL-5: Testability:*** *The system shall support unit testing with Google Test for game logic, authentication, and database operations (to be implemented).*

5.5 Business Rules

- ***BR-1:*** *Only authenticated users can play games or view history.*
- ***BR-2:*** *The system shall treat 'X' as the first player in all games.*
- ***BR-3:*** *In single-player mode, the user plays as 'X', and the AI plays as 'O'.*

6. Other Requirements

- ***OTHER-1:*** *The system shall support up to 1000 user accounts and 10,000 game records without performance degradation.*
- ***OTHER-2:*** *The system shall comply with open-source licenses for Qt, SQLite, and OpenSSL.*
- ***OTHER-3:*** *The codebase shall be hosted in a GitHub repository with CI/CD configured via GitHub Actions (to be implemented).*

- ***OTHER-4:*** *The system shall use object-oriented programming principles, including encapsulation (e.g., private members in Game, User) and inheritance (e.g., GameHistoryModel from QAbstractTableModel).*

Appendix A: Glossary

- ***AI:*** *Artificial Intelligence, the computer opponent in single-player mode.*
- ***GUI:*** *Graphical User Interface, implemented with Qt widgets.*
- ***Minimax Algorithm:*** *A decision-making algorithm used for the AI's Impossible difficulty, evaluating all possible game outcomes.*
- ***Qt:*** *A C++ framework for GUI and database integration (Version 6.9.0).*
- ***SHA-256:*** *A cryptographic hash function used for password security.*
- ***SQLite:*** *A lightweight database for storing user and game data.*
- ***Tic Tac Toe:*** *A 3x3 grid game where players alternate placing 'X' or 'O' to achieve three in a row, column, or diagonal.*