Software Requirements Specification (SRS) for Tic-Tac-Toe

Table of Contents

Definitions, Acronyms, and Abbreviations

AI: Artificial Intelligence

SRS: Software Requirements Specification

UI: User Interface

UX: User Experience

QSS: Qt Style Sheets

Qt: A cross-platform application development framework.

Minimax: An algorithm used in AI for decision-making in two-player games.

Alpha-Beta Pruning: An optimization technique for the Minimax algorithm.

SQL: Structured Query Language (used for database interaction).

EEST: Eastern European Summer Time.

PVP: Player versus Player.

app.pro: QMake project file for the main application.

tests.pro: QMake project file for the test suite.

mainwindow.ui: XML file defining the user interface layout.

MOC: Meta-Object Compiler, a tool used by Qt to handle C++ extensions for signals and slots.

Uic: Qt User Interface Compiler, a tool that processes .ui files to generate C++ header files.

QSettings: A Qt class for storing application settings persistently.

SQLite: A C-language library that implements a small, fast, self-contained, high-reliability, full-featured, SQL database engine.

SHA-256: Secure Hash Algorithm 256-bit, a cryptographic hash function.

QMake: A build system tool used by Qt to manage project files.

1. Introduction

1.1. Purpose

This Software Requirements Specification (SRS) document details all functional and non-functional requirements for the Tic-Tac-Toe application. It outlines the game rules, system behavior, user interaction flows, and performance considerations to serve as a comprehensive guide for development and testing.

1.2. Scope

The Tic-Tac-Toe application is a desktop game allowing users to play the classic Tic-Tac-Toe game against another human player or against an AI opponent of varying difficulties. It includes user management features such as registration, login, password reset, and the ability to view and replay past game history.

1.3. Definitions, Acronyms, and Abbreviations

AI: Artificial Intelligence SRS: Software Requirements Specification UI: User Interface UX: User Experience QSS: Qt Style Sheets Qt: A cross-platform application development framework. Minimax: An algorithm used in AI for decision-making in two-player games. Alpha-Beta Pruning: An optimization technique for the Minimax algorithm. SQL: Structured Query Language (used for database interaction). EEST: Eastern European Summer Time. PVP: Player versus Player. app.pro: QMake project file for the main application. tests.pro: QMake project file for the test suite. mainwindow.ui: XML file defining the user interface layout. MOC: Meta-Object Compiler, a tool used by Qt to handle C++ extensions for signals and slots. Uic: Qt User Interface Compiler, a tool that processes .ui files to generate C++ header files. QSettings: A Qt class for storing application settings persistently. SQLite: A C-language library that implements a small, fast, self-contained, high-reliability, full-featured, SQL database engine. SHA-256: Secure Hash Algorithm 256-bit, a cryptographic hash function. QMake: A build system tool used by Qt to manage project files.

2. Overall Description

2.1. Product Perspective

The Tic-Tac-Toe application is a standalone desktop game developed using the Qt framework. It integrates a SQLite database for user management and game history persistence. The application aims to provide an intuitive and engaging experience for users to play Tic-Tac-Toe.

2.2. Product Functions

The application provides several core functions. It handles user authentication, allowing users to register, log in, and reset their passwords. Users can choose between game modes: playing against another human player (PvP) or against an AI opponent. For the AI, there are “easy” (random moves) and “hard” (Minimax with Alpha-Beta Pruning) difficulty levels. Players interact with the game board by clicking on empty cells to make moves. The system

manages the game state, tracking the current player, checking for win/draw conditions, and updating the board. A significant feature is game history management, which includes storing and displaying past game data, such as players, results, and move sequences. Users can also replay these past games move by move. Finally, the application maintains a persistent session for auto-login.

2.3. User Classes and Characteristics

The application defines distinct user classes with specific characteristics. A Guest User can navigate the login and signup pages but is restricted from playing games or accessing features that require authentication. A Registered User possesses full access: they can log in with their username and password, play games against either the AI or another player, view and manage their personal game history, update their password, and log out. Within a game, Player X is consistently designated as the first player to make a move, represented by 'X' on the board. Player O is the second player, represented by 'O', and can be either a human opponent or the AI.

2.4. Operating Environment

The application is built as a desktop application using the Qt framework. This design choice ensures its compatibility and ability to run on common desktop operating systems, including Windows, macOS, and Linux. For data storage, the application utilizes SQLite, an embedded database solution, which means it operates independently without the need for an external database server.

3. Functional Requirements

3.1. User Management

3.1.1. User Registration

The system shall allow new users to register by providing a unique username, a password, and optionally their first and last names. For enhanced security, the system shall hash the user's password using SHA-256 before storing it in the database. The system shall prevent the registration of a new user if the chosen username already exists in the database. Upon completion of the registration attempt, the system shall provide clear feedback to the user, indicating success or failure.

3.1.2. User Login

The system shall authenticate users by verifying their provided username and password against the stored hashed passwords in the database. Following a successful authentication, the system shall automatically redirect the user to the main application page. The system shall provide immediate feedback to the user regarding the success or failure of their login attempt. For user convenience, an option to "Show Password" shall be available for the password input fields. Furthermore, the system shall implement an auto-login feature, leveraging QSettings, to automatically log in the last successfully authenticated user upon application startup.

3.1.3. Password Reset

The system shall provide a mechanism for users to reset their password. This process requires the user to input their username, a new password, and confirm the new password by re-entering it. The system shall enforce that the new password and its confirmation input must exactly match. Upon successful validation, the system shall update the stored hashed password for the specified user in the database. Comprehensive feedback shall be provided to the user, informing them of the success or failure of the password reset operation.

3.1.4. User Information Display

Upon accessing the "My Account" page, the system shall display the logged-in user's personal information, specifically their first name, last name, and username.

3.1.5. Logout

The system shall allow a logged-in user to explicitly log out from their session. This action will clear the current user's session data within the application and redirect the user back to the login page. As part of the logout process, the system shall also clear any saved auto-login information to ensure the user is not automatically logged in during subsequent application launches.

3.2. Game Play

3.2.1. Start New Game

The system shall allow users to initiate a new game in "Player Vs AI" mode, providing a choice between "Easy" or "Hard" difficulty settings for the AI opponent. Additionally, the system shall support starting a new game in "Player Vs Player" mode, where two human users can compete. Upon the commencement of any new game, the entire game board shall be reset to an empty state, and Player X will consistently be designated as the starting player.

3.2.2. Make a Move

The system shall enable the current player to make a move by clicking on an available (empty) cell within the 3x3 game board grid. After a successful move, the system shall visually update the selected cell to display the current player's corresponding mark ('X' or 'O'). The system is designed to prevent invalid moves, such as attempting to place a mark on an already occupied cell or selecting a position outside the defined board boundaries. Following a valid move made by a human player, the system shall automatically switch the turn to the next player. If the game is in "Player Vs AI" mode and it becomes Player O's turn, the system shall automatically trigger the AI to calculate and execute its move.

3.2.3. AI Behavior

The Artificial Intelligence (AI) component exhibits distinct behaviors based on the selected difficulty. In "Easy" difficulty, the AI shall select its next move randomly from all available

empty cells on the board. Conversely, in “Hard” difficulty, the AI shall employ the Minimax algorithm, enhanced with Alpha-Beta Pruning, to strategically determine the optimal move that maximizes its chances of winning or minimizes its chances of losing. To provide a more natural user experience and simulate a “thinking” process, the AI shall introduce a small, programmed delay (e.g., 500ms) before executing its chosen move.

3.2.4. Game Outcome Detection

After each valid move is made on the board, the system shall continuously evaluate the game state to detect if a win condition has been met by either Player X or Player O (or the AI). In the absence of a winner, if all cells on the board become occupied, the system shall accurately detect a draw condition. Upon the conclusion of the game, regardless of whether it’s a win or a draw, the system shall clearly display the final result to the user. To prevent any further interaction with the finished game, the system shall disable all interactive elements on the game board once an outcome is determined.

3.2.5. Game Reset

The system shall provide users with the ability to reset the current game board at any point during gameplay, returning it to an empty state. Initiating a game reset shall also automatically revert the current player back to Player X and clear the entire move history accumulated during the current game session.

3.3. Game History Management

3.3.1. Save Game History

Upon the conclusion of each game, whether it results in a win or a draw, the system shall automatically save the game’s outcome and the complete sequence of moves to the persistent database. Each entry in the game history shall comprehensively include the names of Player 1, Player 2 (which will be recorded as “AI” if applicable), the definitive result of the game, and a precise timestamp indicating when the game was completed. The full sequence of moves made throughout the game shall be stored as a single, comma-separated string, formatted to include the row, column, and the character of the player who made each move (e.g., “0:0:X,1:1:O”).

3.3.2. Load Game History

The system shall be capable of retrieving and presenting a comprehensive list of all games in which the currently logged-in user participated, fetching this data from the database. Each individual game history entry shall be displayed within a list widget, clearly indicating the opponent, the final result of the game, and the timestamp of its completion. To ensure the most relevant information is presented first, the history list shall be ordered by timestamp in descending order, showing the most recent games at the top.

3.3.3. Delete Game History

The system shall provide users with the functionality to delete a specific game history entry that they select from the displayed list. Following the deletion attempt, the system shall

provide clear feedback to the user, confirming the successful removal of the history item or indicating any encountered errors.

3.3.4. Replay Game

The system shall enable users to select any game from their stored history and initiate a visual replay of the entire game, move by move, on the main game board. During this replay sequence, the board cells shall be updated sequentially, with a programmed visual delay between each move to allow the user to comfortably follow the game's progression. For the duration of the replay, the game board shall be automatically disabled, preventing any interactive input from the user. Once all moves in the replay sequence have been displayed, the system shall notify the user that the game replay has concluded.

4. Non-Functional Requirements

4.1. Performance Requirements

4.1.1. AI Response Time

The "Hard" difficulty AI, which employs the Minimax algorithm, should consistently determine and return its optimal move within a maximum of 1 second when running on typical desktop hardware.

4.1.2. Database Operations

Key database operations, including user authentication, new user registration, and the saving or loading of game history records, should complete within 500 milliseconds. This performance target applies under normal operating conditions, such as a database containing up to 1000 game entries per user.

4.1.3. Game Replay Speed

The visual animation of game replays should proceed at a controlled pace. This is achieved by having a fixed delay of approximately 800 milliseconds between each displayed move, ensuring that the replay is clear and easy to follow for the user.

4.2. Security Requirements

4.2.1. Password Hashing

All user passwords shall not be stored in plain text. Instead, they shall be securely stored as cryptographic hashes using the SHA-256 algorithm within the database.

4.2.2. SQL Injection Prevention

To safeguard against common database vulnerabilities, all interactions with the database, specifically data queries, shall utilize prepared statements. This practice effectively prevents SQL injection attacks.

4.2.3. Data Access

The system shall implement access controls to ensure that a user's game history and personal information are only accessible by the relevant, authenticated user, thereby preventing unauthorized data access.

4.3. Usability Requirements (UI/UX)

4.3.1. Intuitive Navigation

The application's user interface shall be designed to provide clear, consistent, and intuitive navigation pathways. Users should be able to easily transition between different primary screens, including the login page, main menu, game board, account management, and game history.

4.3.2. Clear Feedback

The system shall provide immediate, clear, and contextually relevant feedback to the user for all their actions. This includes messages indicating the success or failure of login attempts, validity of moves made on the board, and the final outcome of games.

4.3.3. Consistent Styling

The application shall maintain a unified and consistent visual theme across all its screens and components. This consistency is achieved through a modern dark theme with a predefined color palette, ensuring a cohesive and aesthetically pleasing user experience.

4.3.4. Readability

All text elements and visual components on the game board shall be designed to be clearly visible and easily legible to the user, ensuring a comfortable viewing experience.

4.4. Maintainability Requirements

4.4.1. Modularity

The codebase shall be meticulously structured into distinct, logical components. This includes modules for the User Interface (UI), core game logic, Artificial Intelligence (AI) algorithms, database management, and board state representation. Each component shall have well-defined interfaces to minimize interdependencies and facilitate independent development and modification.

4.4.2. Code Readability

The source code shall adhere to consistent coding standards and practices. It shall also be thoroughly commented to explain complex logic, design choices, and functional blocks, thereby enhancing its readability and understanding for other developers.

4.4.3. Testability

Core logic components of the application, specifically the Board representation, AIPlayer behavior, GameLogic rules, and DatabaseManager interactions, shall be designed and implemented in a manner that facilitates comprehensive and automated unit testing.

4.5. Portability Requirements

4.5.1. Cross-Platform Compatibility

Given its development using the Qt framework, the application shall be compilable and fully runnable across major desktop operating systems. This includes Windows, macOS, and Linux, ensuring broad accessibility without platform-specific code changes.

4.6. Reliability Requirements

4.6.1. Error Handling

The system shall be engineered to gracefully handle and recover from anticipated errors. This encompasses scenarios such as invalid user input or potential issues with database connectivity. In such cases, the system will provide informative messages to the user or log detailed error information for debugging purposes.

4.6.2. Data Integrity

The database component of the system shall be designed to maintain high levels of data integrity. This ensures the accuracy, consistency, and reliability of all stored user accounts and game history records.

5. Other Requirements

5.1. Internationalization

The application shall incorporate support for internationalization (I18N). This involves utilizing Qt's robust translation system, allowing the user interface strings to be easily adapted and displayed in multiple languages without requiring modifications to the core source code.