Software Design Specification

project name:	Tic-Tac-Toe
Document Version:	1.0
Creation Date:	June 19, 2025
Author:	Embedded Team
Status:	Final

Table of Contents

1. Purpose

The purpose of this Software Design Specification (SDS) is to outline the architecture and design of the Tic Tac Toe system — a Qt-based application that includes secure user authentication, multiple gameplay modes (Player vs Player and Player vs AI with adjustable difficulty), and a complete history tracking mechanism.

This document serves as a clear and structured reference for both the development team and stakeholders, supporting effective implementation through the use of UML diagrams, flowcharts, and modular component design.

2. System Overview

The Tic Tac Toe system is a feature-rich desktop application developed using the Qt framework (C++), designed to offer an engaging and secure gameplay experience.

Key features of the system include:

- **User Management**: Users can register and log in securely using SHA-256 hashed credentials.
- **Game Modes**: Offers both Player vs Player (PvP) and Player vs AI (PvAI) modes.
- **AI Intelligence**: AI opponents are available in three difficulty levels (Easy, Medium, Hard), each utilizing progressively advanced decision-making algorithms.
- **Game History**: Automatically tracks and displays each user's match history with win/loss/draw statistics.
- **User Interface**: Built with a responsive and intuitive Qt-based graphical interface for smooth user experience.

The system follows a modular architecture that emphasizes maintainability, scalability, and data security. All user-related and game-specific data is stored in a structured JSON file, ensuring persistence, easy access, and portability.

3. Architectural Design

The system architecture follows a refined **Model-View-Controller (MVC)** pattern, ensuring a clean separation of concerns across components. Each layer is responsible for specific functionalities as outlined below:

Model Layer

- **UserManager**: Handles user authentication, data persistence, and game history storage.
- **Security**: Manages password hashing using SHA-256 for secure credential handling.
- **GameLogic** *(implicit inside GameWindow)*: Controls game rules, board state, and AI decision-making.

View Layer

- **MainWindow**: Entry point for user login and navigation.
- **RegisterWindow**: Dedicated interface for new user registration.
- **HomeWindow**: User dashboard after successful login.
- **GameWindow**: The main game interface (PvP / PvAI) with clickable board.
- **HistoryWindow**: Displays match history with win/loss statistics.

Controller Layer

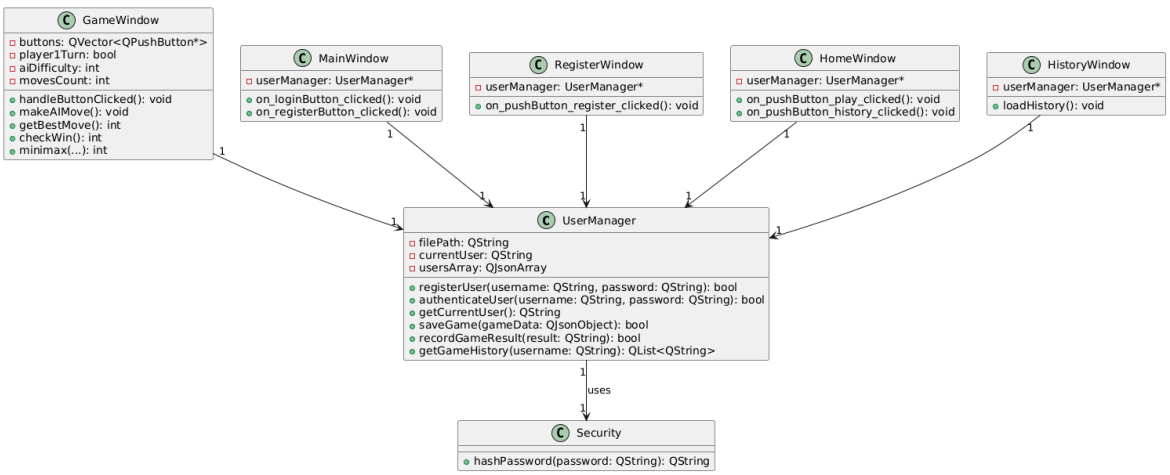
- Event-driven logic using **Qt slots and signals** to handle UI actions.
- Game flow control (move handling, turn switching, AI integration).
- Input validation and error-handling across all user interactions.

4. UML Class Diagram

The **class diagram** offers a clear and structured overview of the system's architectural components, highlighting class responsibilities, relationships, and internal behaviors.

This enhanced diagram adheres to standard UML conventions and includes:

- **Visibility indicators** (+ public, - private, # protected) for all attributes and methods.
- **Well-defined attributes** with appropriate data types for each class.
- **Method signatures** that display parameter types and return types for better understanding of functionality.
- **Relationships** between components, such as associations, dependencies, and inheritance, clearly illustrated.
- **Multiplicity indicators** to show how many instances of one class relate to another (e.g.1, *..1, 1..0).



5. UML Sequence Diagrams

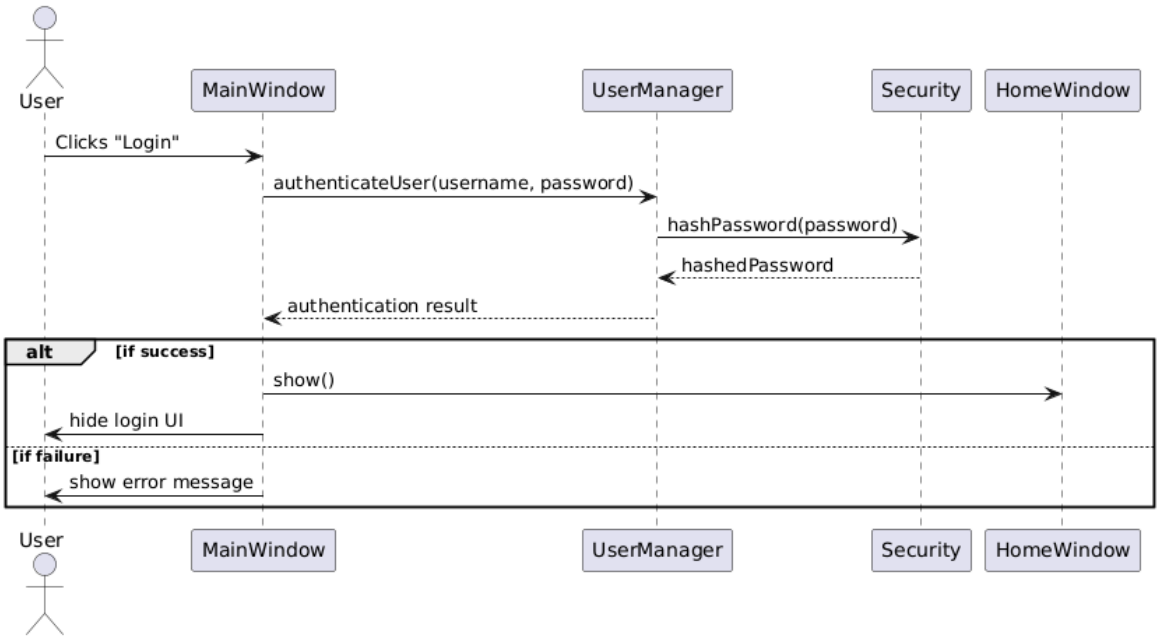
5.1 Login Flow Sequence Diagram

This sequence diagram illustrates the **user authentication process** from the point of interaction in the `MainWindow` interface through to validation by the system components.

Process Overview :

1. The **user** initiates the login process by clicking the **"Login"** button in the `MainWindow`.
2. `MainWindow` retrieves the input **username** and **password** from the corresponding input fields.
3. The credentials are passed to the `UserManager` via the `authenticateUser()` method.
4. Inside `UserManager`, the password is securely hashed using the `Security::hashPassword()` method, which applies SHA-256 encryption.
5. The hashed password is compared with the stored password from the `users.json` file.
6. A boolean result is returned to the `MainWindow`:
    - If authentication **succeeds**, `MainWindow` creates and displays the `HomeWindow`, and hides itself.
    - If authentication **fails**, an error message is displayed using `QMessageBox` indicating invalid credentials.

This sequence ensures secure login handling and user feedback through proper component interaction and input validation.



5.2 Start Game (Player vs AI) Sequence Diagram

This sequence diagram demonstrates the process of initializing the game and executing AI interactions in **Player vs AI (PvAI)** mode. It outlines the flow starting from user input up to the AI decision logic.

Process Overview :

1. The **user** initiates gameplay by clicking the **"Play"** button in the `HomeWindow` interface.

2. `HomeWindow` creates a new instance of the `GameWindow`, passing in the selected **game mode** (PvAI), **difficulty level**, and **current user context**.

3. Inside the `GameWindow`, the constructor retrieves the current user by calling `userManager->getCurrentUser()`.

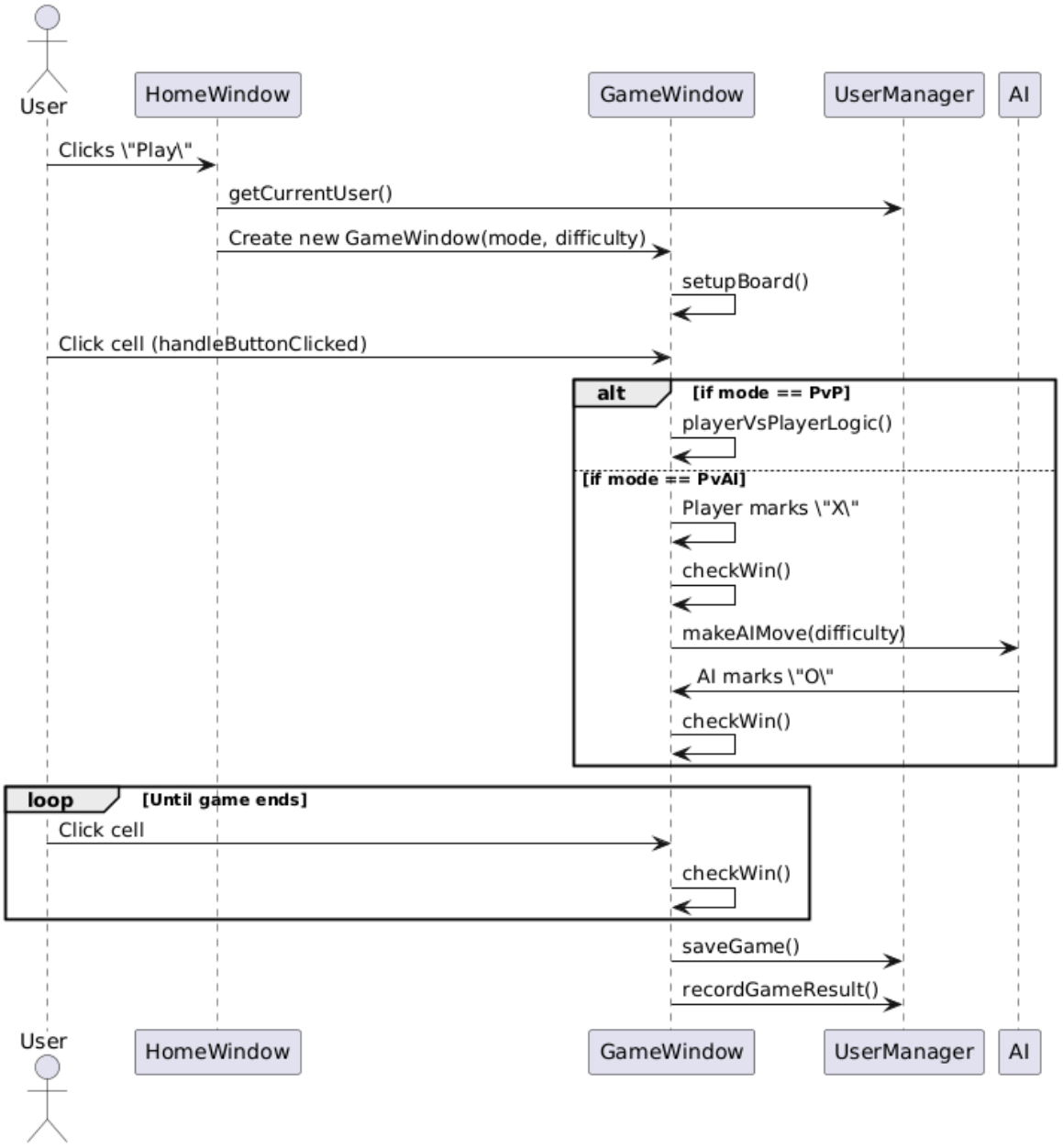
4. The game board UI is initialized through `setupBoard()` which resets the buttons and turn labels.

5. The **user interacts** with the board by clicking on a cell, which triggers

`handleButtonClicked()`.

6. Upon a valid move, the AI's turn is initiated automatically by calling `makeAIMove()`, which in turn calls either:
    - `getRandomMove()` (for easy difficulty),
    - or `getBestMove()` → `minimax()` (for medium and hard).
7. The game state is updated accordingly (win/draw check), the board UI is refreshed, and the current turn is displayed.

This flow ensures a responsive and intelligent gameplay experience, with each component handling its respective responsibility through clear method calls and UI updates.



6. System Flowchart

The comprehensive system flowchart provides a high-level visual representation of the full user journey throughout the Tic Tac Toe desktop application. It highlights major decision points, logical operations, and system responses in a clear and structured manner.

Key Flow Elements :

- **Application Launch**:
  The system starts by launching the Qt application and displaying the `MainWindow`, which serves as the entry point for both login and registration.

- **Authentication Branch**:

The user chooses to either log in or register.

  - If **registering**, the system opens the `RegisterWindow`, validates inputs, and attempts to register the user.
  - If **logging in**, credentials are verified through `UserManager::authenticateUser()`, with appropriate feedback for incorrect input.

- **Post-Login Navigation**:
  Upon successful authentication, the system navigates to the `HomeWindow`, presenting core features of the application.

- **Game Mode Selection**:
  The user selects either **Player vs Player (PvP)** or **Player vs AI (PvAI)** mode.

- **Difficulty Selection (PvAI only)**:
  If PvAI is selected, the user chooses the AI difficulty level:
  - Easy (random moves),
  - Medium (mix of random and optimal),
  - Hard (Minimax algorithm).

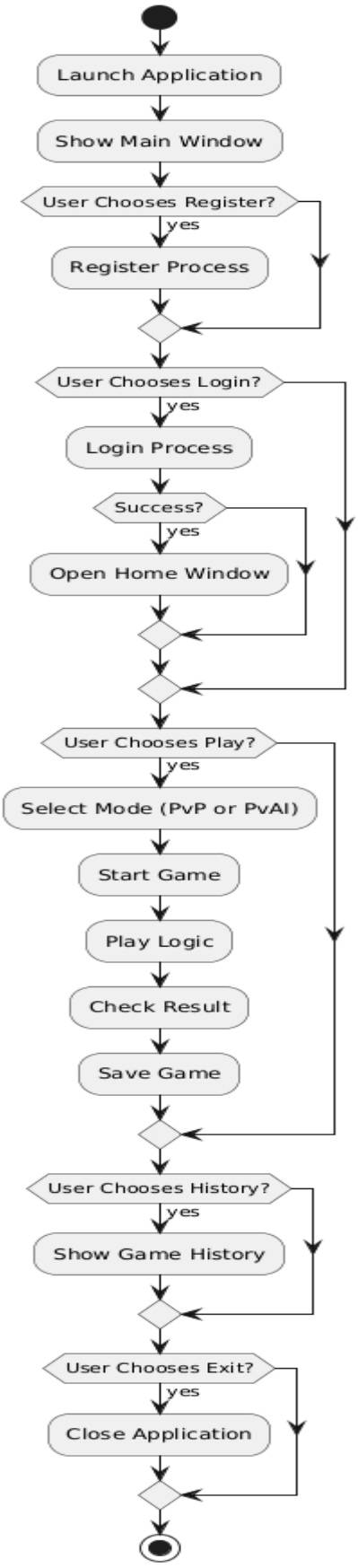
- **Game Execution**:
  The `GameWindow` is launched. The user plays interactively on a 3x3 board, with turns tracked, input validated, and UI updated.

- **AI Move Logic**:
  For AI modes, the system calculates the best move based on the selected difficulty and executes it automatically.

- **Game Completion**:
  When the game ends (win or draw), results are displayed via `QMessageBox`. Outcome is logged.

- **History Management**:
  The `UserManager` records both a detailed JSON game log and a summary result (win/loss/draw) for history tracking.

- **User Options**:
  After game completion, the user can either restart the game, return to the home screen, view game history via `HistoryWindow`, or exit the application.


Launch Application
Show Main Window
User Chooses Register?
yes
Register Process
User Chooses Login?
yes
Login Process
Success?
yes
Open Home Window
User Chooses Play?
yes
Select Mode (PvP or PvAI)
Start Game
Play Logic
Check Result
Save Game
User Chooses History?
yes
Show Game History
User Chooses Exit?
yes
Close Application

7. AI Logic Design

7.1 Difficulty Levels

Easy Mode:

• Randomly selects a valid empty cell without any prediction

• Performs no strategic evaluation or minimax analysis

• Executes the move immediately upon player action

Medium Mode:

• Uses basic heuristics to select moves with higher potential

• Detects and responds to immediate win/loss threats

• Applies shallow prediction with limited depth (1–2 moves ahead)

Hard Mode:

• Implements advanced Minimax algorithm with alpha-beta pruning

• Evaluates complete game tree to identify the optimal move

• Uses depth-based scoring to favor quicker wins and delay losses

• Ensures flawless decision-making with no avoidable defeats

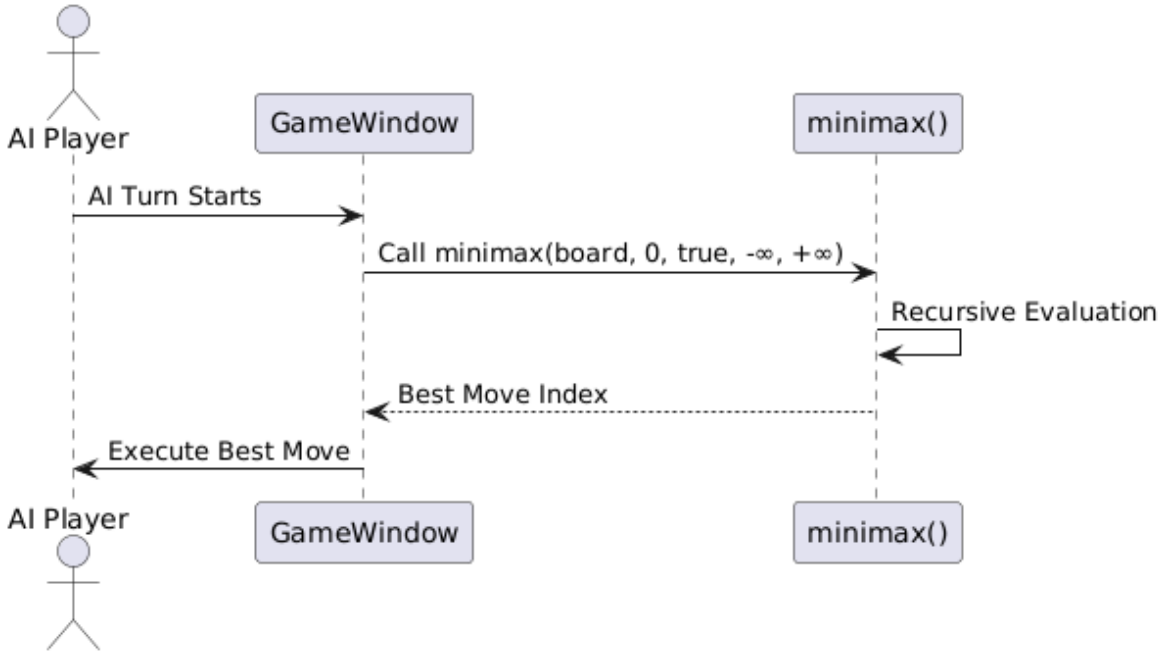
7.2 Algorithm Implementation

The AI decision engine at Hard difficulty leverages an advanced minimax algorithm with alpha-beta pruning.

The sequence diagram below depicts the interaction between the AI logic and game components during an AI turn at Hard difficulty:

- When it's the AI's turn, `GameWindow` triggers the decision-making process.
- The `minimax()` algorithm is called to recursively analyze all possible game states.
- Once the best move is identified, it is applied to the game board.

This ensures intelligent and non-random decision-making that guarantees optimal or near-optimal gameplay.



8. Data Storage Design

8.1 JSON Structure

The application uses a structured JSON format for data persistence:

```
{
  "users": [
    {
      "username": "ali",
      "password": "5e88489...hashed",
      "games": [
        {
          "opponent": "AI",
          "result": "win",
          "date": "2025-06-12",
```

```
            "moves": ["X1", "O2", "X5", ...]

          }

        ]

      }

    ]

}
```

8.2 Data Security

The application implements essential security measures to protect user data and maintain system integrity:

- **Password Security**:
  All user passwords are hashed using **SHA-256** encryption through `QCryptographicHash`. This ensures passwords are stored in a non-reversible format, enhancing protection against unauthorized access.

- **Input Validation**:
  Basic validation is implemented to ensure all required fields (e.g., username and password) are filled. Empty inputs are rejected at the UI level using `QMessageBox` warnings.

- **File Error Handling**:
  During loading and saving of user data from the `users.json` file, the application verifies file access success (e.g., `file.open()` checks). However, deeper recovery mechanisms for corrupted files are not implemented.

- **Data Persistence**:
  All user and game history data are saved to a persistent **JSON file**, which ensures that records are not lost between sessions.

9. Conclusion

This Software Design Specification (SDS) outlines a structured, maintainable, and secure approach to building a robust **Tic Tac Toe desktop application** using the Qt framework. Through the inclusion of detailed UML diagrams, system flowcharts, and well-defined architectural layers, this document serves as a reliable reference for all phases of the software development lifecycle.

By adopting a modular architecture and a systematic design methodology, the system ensures clarity, reusability, and ease of future enhancement. This documentation supports developers, testers, and maintainers with a clear understanding of the system's functionality and technical foundation.

Key Strengths of This Design:

- **Comprehensive UML Documentation**
  Industry-standard class and sequence diagrams that clarify component interactions and responsibilities.

- **Robust Security**
  SHA-256 password hashing and strong input validation to ensure data integrity and privacy.

- **Scalable & Modular Architecture**
  Clearly separated components that support maintainability and future scalability.

- **User-Centric Interface**
  A friendly, intuitive UI that includes personal game history tracking for each user.

- **Advanced AI Integration**
  Multiple difficulty levels including Minimax-based hard mode for intelligent and challenging gameplay.

- **Complete Documentation**
  Full system flowcharts, design guidelines, and implementation structure suitable for professional deployment and evaluation.