Software Requirements Specification (SRS)

1. Introduction

The Tic-Tac-Toe game is a graphical user interface (GUI) application designed to provide both single-player (against an AI) and multiplayer (two players) gameplay experiences. The game will include a login and registration system, game history tracking, and a robust AI opponent.

2. Functional Requirements

2.1 User Management

- **Registration:**
  - Users must be able to register with a unique username and password.
  - Passwords must be hashed before storing in the database.
- **Login:**
  - Registered users must be able to log in using their username and password.
  - Successful login must load the user's game history.
- **Authentication:** Verification of User Credentials
  - is a process used to confirm the identity of a user attempting to access a system. In the context of the Tic-Tac-Toe game application, authentication ensures that only registered users can log in and access their game history..

2.2 Game Modes

- **Single Player:**
  - The user plays against the AI.
  - The AI should use the minimax algorithm with alpha-beta pruning to determine its moves.
- **Multiplayer:**
  - Two users play against each other on the same device.

2.3 Gameplay

- **Game Board:**
  - A 3x3 grid where players take turns marking empty cells with 'X' or 'O'.
  - The game board must reset after a game ends.
- **Moves:**
  - Players can make moves by clicking on the buttons corresponding to the cells.
  - The game must check for win/draw conditions after each move.
- **Win/Draw Detection:**
  - The game must detect and announce the winner or if the game is a draw.
  - The game outcome must be recorded in the user's game history.

2.4 Game History

- **Tracking:**
  - The system must record the outcomes of games including players, winner, and game type (single-player/multiplayer).
  - Users must be able to view their game history.

2.5 User Interface

- **Login/Register Screen:**
  - Fields for username and password, with buttons for login and registration.
  - Display appropriate messages for successful/failed login/registration attempts.
- **Main Game Screen:**
  - Display the game board.
  - Display the current player's turn.
  - Options to switch between single-player and multiplayer modes.

3. Non-Functional Requirements

3.1 Performance

- **Responsiveness:** Quick responses to user actions
  - The AI move calculation should not take more than 2 seconds.
  - The game interface should respond to user actions within 100 milliseconds.

3.2 Usability

- **User-Friendly:**
  - The application should be intuitive and easy to navigate.
  - Clear instructions and feedback should be provided.

3.3 Security

- **Password Protection:**
  - Passwords must be stored securely using SHA-256 hashing.
  - The system should prevent SQL injection attacks.

3.4 Reliability

- **Data Integrity:**
  - User data and game history must be stored reliably in the database.
  - The system should handle and recover from errors gracefully.

3.5 Compatibility

- **Platform:**

- The application should run on all major operating systems (Windows, macOS, Linux).

3.6 Maintainability:

- Well-documented and easy-to-update code.

4. System Architecture

4.1 Overview

The system is a desktop application developed using C++ and the Qt framework. It interacts with an SQLite database to manage user data and game history.

4.2 Components

- **User Interface:**
  - Developed using Qt, includes widgets for the game board, login/register forms, and game history display.
- **Game Logic:**
  - Implements the rules of Tic-Tac-Toe, move validation, and win/draw detection.
- **AI Module:**
  - Uses the minimax algorithm with alpha-beta pruning to determine the best move.
- **Database:**
  - SQLite database to store user credentials and game history.

5. System Behavior

5.1 User Registration

1. User enters a username and password.
2. System checks if the username is unique.
3. If unique, the system hashes the password and stores the user data.

5.2 User Login

1. User enters a username and password.
2. System verifies the credentials.
3. If valid, the system retrieves and displays the user's game history.

5.3 Gameplay

1. User selects a game mode (single-player/multiplayer).
2. In single-player mode, the AI makes a move after the user.
3. In multiplayer mode, players take turns making moves.
4. The system checks for a win or draw after each move.
5. The game outcome is recorded and displayed.

5.4 Game History Display

1. Upon successful login, the system retrieves and displays the user's game history.
2. Users can view past game outcomes including the opponents and results.

6 Performance Requirements

Performance requirements specify the expected performance standards of the system to ensure a smooth and efficient user experience. Here are some key performance requirements for a Tic-Tac-Toe game application:

6.1 Response Time:

- **Game Moves:** The game should update the board immediately (within 100 milliseconds) after a player makes a move.
- **AI Move Calculation:** The AI should calculate its move within 1 second to maintain a seamless game flow.

6.2 Login and Registration:

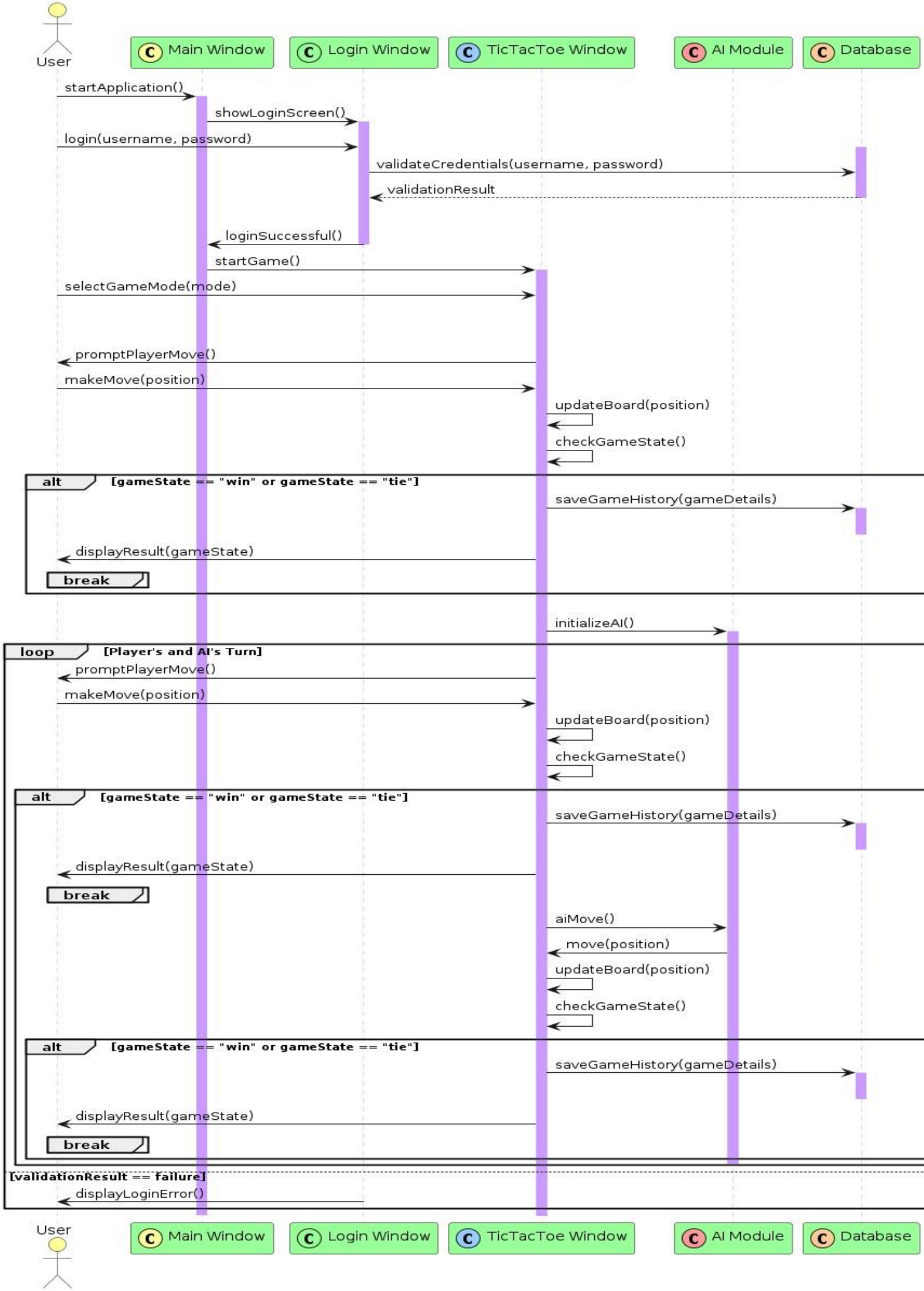
- **Login:** The system should authenticate a user within 2 seconds.
- **Registration:** The user registration process should be completed within 3 seconds.

6.3 Database Operations:

- **Data Retrieval:** Retrieving game history and other user data from the database should take no more than 1 second.
- **Data Storage:** Storing game outcomes and user information should be completed within 2 seconds.

6.4 User Interface:

- **Load Time:** The initial loading of the game interface should take less than 3 seconds.
- **Navigation:** Switching between different sections (e.g., game board, game history) should be instantaneous or take less than 500 milliseconds.


User
Main Window
Login Window
TicTacToe Window
AI Module
Database
startApplication()
showLoginScreen()
login(username, password)
validateCredentials(username, password)
validationResult
loginSuccessful()
startGame()
selectGameMode(mode)
promptPlayerMove()
makeMove(position)
updateBoard(position)
checkGameState()
alt [gameState == "win" or gameState == "tie"]
saveGameHistory(gameDetails)
displayResult(gameState)
break
initializeAI()
loop [Player's and AI's Turn]
promptPlayerMove()
makeMove(position)
updateBoard(position)
checkGameState()
alt [gameState == "win" or gameState == "tie"]
saveGameHistory(gameDetails)
displayResult(gameState)
break
aiMove()
move(position)
updateBoard(position)
checkGameState()
alt [gameState == "win" or gameState == "tie"]
saveGameHistory(gameDetails)
displayResult(gameState)
break
[validationResult == failure]
displayLoginError()
User
Main Window
Login Window
TicTacToe Window
AI Module
Database