Software Design Specification

for Advanced Tic Tac Toe

Prepared for Dr. Omar Nasr

Contents

1 Introduction

1.1 Introduction

ThisdocumentprovidesadetailedsoftwaredesignspecificationfortheAdvanced Tic Tac Toe application, developed by 404wins. It serves as a comprehensive guide for developers and maintainers to understand, implement, and extend the system. The design addresses all requirements outlined in the Software Requirements Specification (SRS), detailing the implementation of game modes, user authentication, AI, and persistent storage.

1.2 System Overview

The Advanced Tic Tac Toe application is a C++-based game built using the Qt framework for the graphical user interface and SQLite for data persistence. It supports two gameplay modes: Classic Tic Tac Toe (3x3 grid) and Mega Tic Tac Toe (3x3 grid of 3x3 subgrids). Players can engage in two-player matches or play against an AI with three difficulty levels (Easy, Medium, Hard). The system includes user authentication, game history tracking, and replay functionality, all managed through a SQLite database. The AI leverages the Minimax algorithm for strategic move selection.

2 Design Considerations

2.1 Assumptions and Dependencies

- Single-user session at a time.
- Local file access for SQLite database (`tictac.db`).
- Qt framework and SQLite libraries are functional on the host operating sys-tem.
- The bcrypt library is available for password hashing.

2.2 General Constraints

- Developed in C++ with Qt for the UI and SQLite for the database.
- No external network access; all data is stored locally.
- Classic mode uses a 3x3 grid; Mega mode uses a 3x3 grid of 3x3 subgrids.• Passwords must be at least 6 characters; usernames at least 4 characters.

2.3 Goals and Guidelines

- Provide an intuitive and responsive user interface using Qt Designer.
- Ensure secure storage of user credentials with bcrypt hashing.
- Support seamless switching between Classic and Mega game modes.
- Enable persistent storage of game history and moves for replay functional-ity.

- Implement an efficient AI using the Minimax algorithm.

2.4 Development Methods

- Modularobject-orienteddesignfollowingtheModel-View-Controller(MVC)pattern.
- GUI designed using Qt Designer with signals and slots for event handling.
- SQLite for local database management.
- Manual testing for UI interactions and game logic; unit tests for AI anddatabase operations.

3 Architectural Strategies

3.1 Use of Frameworks and Libraries

- **Qt**: Used for building the GUI, leveraging QMainWindow, QDialog, and signals/slots for event-driven interactions.
- **SQLite**: Provideslightweight, embeddedstorageforuseraccountsandgame history.
- **bcrypt**: Ensures secure password hashing for user authentication.

3.2 AI Strategy

The AI is implemented in the `Ai` class (`src/ai.cpp`) using the Minimax algorithm without Alpha-Beta pruning (noted as a potential improvement). The AI supports three difficulty levels:

- **Easy**: Random moves selected from available cells.
- **Medium**: Combines random moves (50% chance) with Minimax-based optimal moves.
- **Hard**: Uses full Minimax evaluation for optimal move selection.

TheMinimaxalgorithmevaluatesgamestatesrecursively, assigningscoresbased on win (+100/depth), loss (-100/depth), or draw (0). The `Tree` class manages the game tree, and the `State` class stores board states and their evaluations.

4 System Architecture

4.1 Architecture

The application adopts the MVC pattern:

- **Model**: Manages data, including game state (`Board`, `megaBoard`), user accounts, and game history (`ReplayManager`, SQLite database).
- **View**: Qt-basedUIcomponents(`MainWindow`, `loginwindow`, `signupwindow`, etc.) for user interaction.
- **Controller**: Handles user inputs, game logic, AI decisions, and database operations, implemented in `MainWindow` slots and game logic classes.

4.2 Subsystem Architecture

- **UI Subsystem**: Manages windows for login, signup, game modes, gameplay, and history (`mainwindow.cpp`, `loginwindow.cpp`, etc.).
- **GameEngineSubsystem**: Handles game logic for Classic and Mega modes (`gameStructure.cpp`).
- **AI Subsystem**: Implements Minimax-based AI (`ai.cpp`).
- **DatabaseSubsystem**: Managesuserandgamedatastorage(`database.cpp`).

5 Policies and Tactics

- **ErrorHandling**: Displays user-friendly messages via `QMessageBox` for invalid inputs, login failures, and game errors.
- **Data Integrity**: Game results and moves are saved to the SQLite database immediately after game completion.
- **Security**: Passwords are hashed using bcrypt; usernames and passwords are validated for minimum length.

6 Detailed System Design

6.1 Classification

Key classes include:

- `MainWindow`: Central UI controller.
- `Board`, `megaBoard`: Manage game state for Classic and Mega modes.
- `Game`: Orchestrates gameplay logic.
- `Ai`, `Tree`, `State`: Implement AI logic.
- `ReplayManager`: Tracks and replays game moves.

6.2 Definition & Responsibilities

- `MainWindow`(`src/mainwindow.cpp`): ManagesnavigationbetweenUIscreens and initializes the application.
- `Board` (`src/gameStructure.cpp`): Represents a 3x3 Classic Tic Tac Toe board with move and win detection.
- `megaBoard` (`src/gameStructure.cpp`): Manages a 3x3x9 Mega Tic Tac Toe board with subgrid win detection.
- `Game` (`src/gameStructure.cpp`): Controls game flow for both modes and AI play.

- `Ai` (`src/ai.cpp`): Implements Minimax-based AI with difficulty levels.
- `ReplayManager` (`src/database.cpp`): Stores and replays game moves, interfacing with the database.

6.3 Constraints

- UI classes inherit from `QMainWindow` or `QDialog`.
- SQLite database (`tictac.db`) stores users, game history, and moves.
- AI limited to local computation without external dependencies.

6.4 Composition

Subcomponents include UI windows, game logic modules, AI decision trees, and database operations, all integrated via signals and slots.

6.5 Uses/Interactions

- UI signals trigger controller slots (e.g., `on_pushButton_login_clicked`).
- `ReplayManager` interacts with `database.cpp` for game history storage.
- `Game` class calls `Ai` for AI moves in single-player mode.

6.6 Resources

- SQLite database (`tictac.db`) for persistent storage.
- Qt resources for UI elements (images, styles).

6.7 Processing

- Player moves update the board and trigger win/draw checks.
- AI moves are computed via `Ai::moveAi` and applied to the board.
- Game results are saved to the database upon completion.

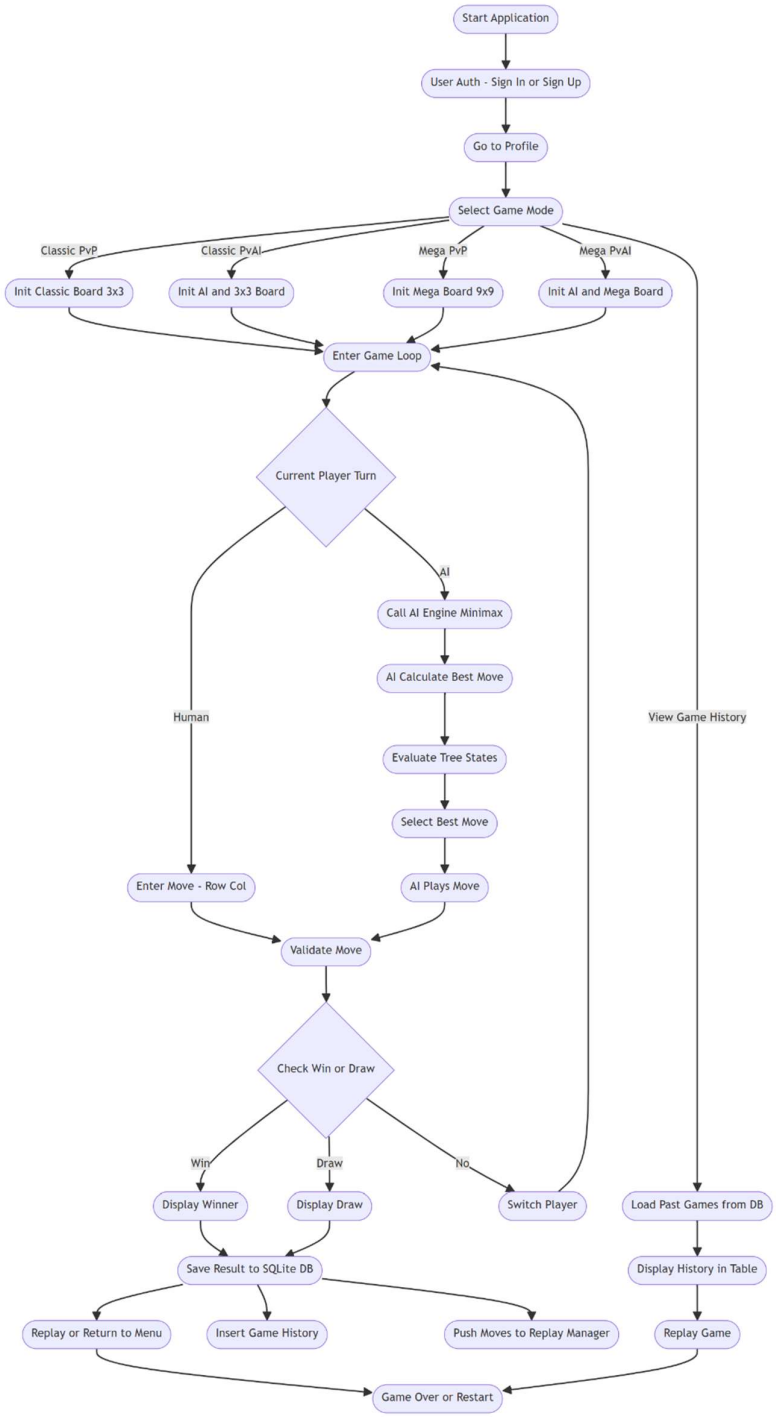
6.8 Interface/Exports

- No external APIs; all interactions occur through the Qt UI.
- Database operations are encapsulated in `database.cpp`.

7 Diagrams

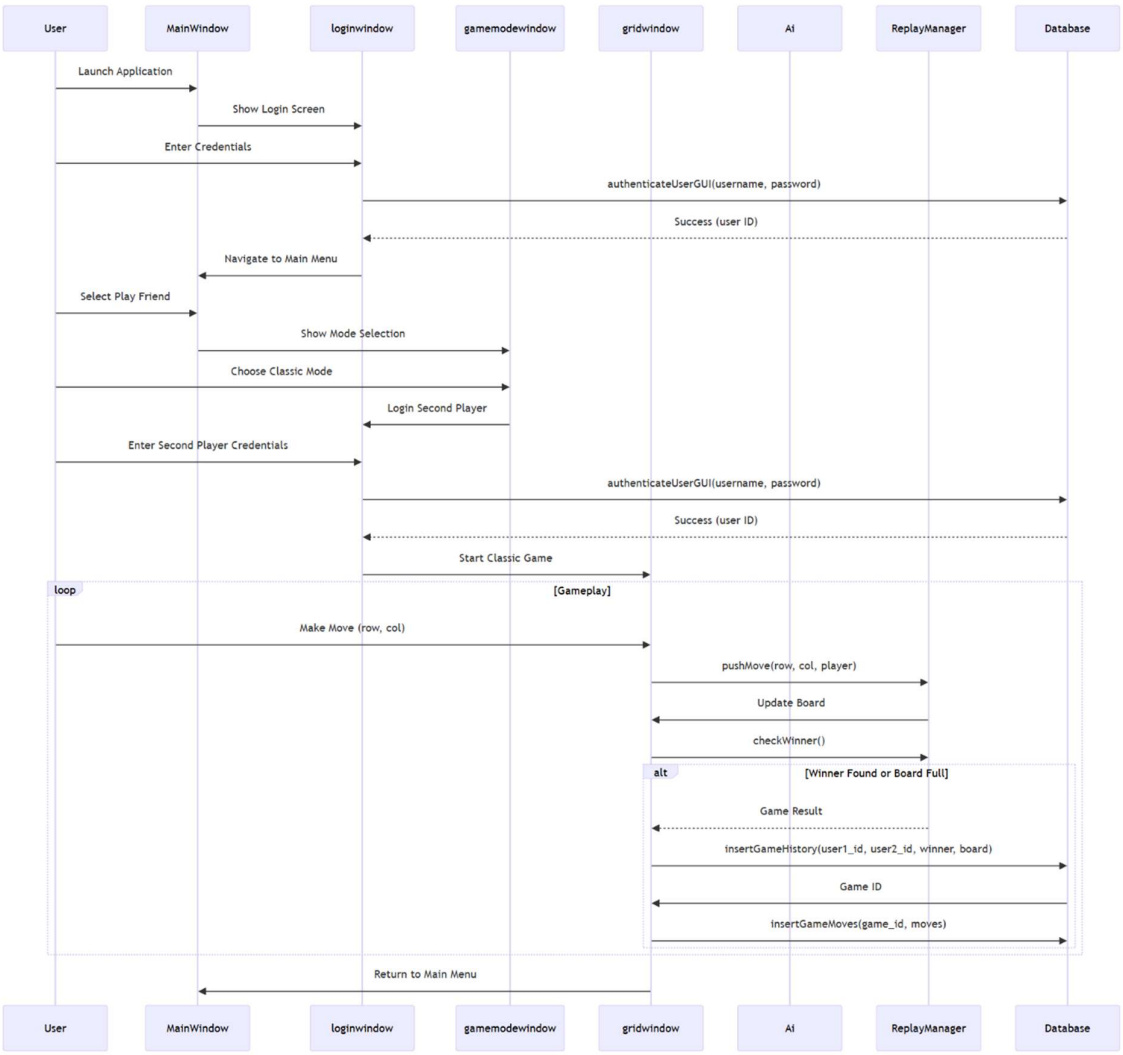
7.1 System Flow Diagram

The system flow diagram (Figure 1) illustrates the MVC architecture, showing interactions between UI components, controller logic, and the data model.



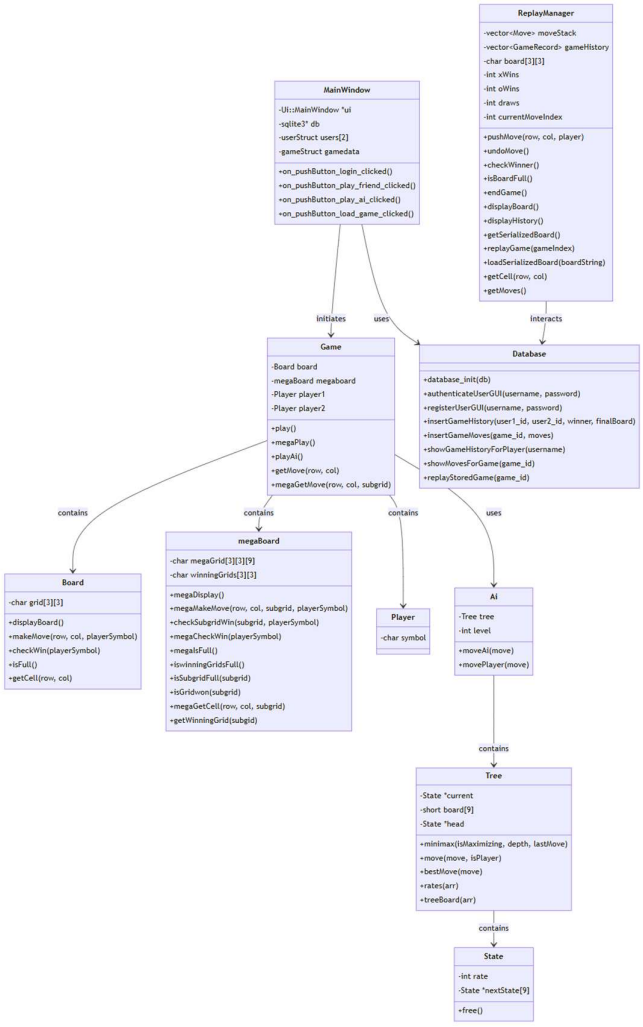
7.2 Sequence Diagram

The sequence diagram (Figure 2) depicts a user session from login to gameplay, highlighting interactions between `MainWindow`, `loginwindow`, `gameStructure`, and `database`.



7.3 Class Diagram

The class diagram (Figure 3) shows relationships between `MainWindow`, `Board`, `megaBoard`, `Game`, `Ai`, and `ReplayManager`.



8 User Interface Screenshots

Key UI screens include:

- **Main Window**: Entry point with login, signup, and exit options.
- **Sign Up Screen**: Form for user registration with username and password.
- **Sign In Screen**: Login form with credential validation.
- **Game Mode Selection**: Choose between Classic and Mega modes.
- **Game Window**: Displays the game board and handles moves. • **Game History View**: Table of past games with replay options.

9 Glossary

- **Minimax**: Algorithm for optimal move selection in turn-based games.
- **SQLite**: Embedded database for local storage.
- **Qt**: Framework for GUI development.
- **bcrypt**: Library for secure password hashing.

10 Bibliography

- Qt Framework Documentation: `https://doc.qt.io/`
- SQLite Documentation: https://www.sqlite.org/docs.html
- bcrypt Documentation: https://github.com/pyca/bcrypt