Software Design Specification

For

Advanced Tic Tac Toe

Submitted to: Dr. Omar Nasr

Table of Contents

1. Introduction

1.1 Purpose

This document provides the Software Design Specification for the Tic Tac Toe game, a cross-platform Tic Tac Toe game developed using C++ and Qt. The design details include the system's architecture, components, database interaction, and user interface. The software supports Player vs Player (PvP) and Player vs AI (PvE) modes, user authentication, and game history tracking.

1.2 Scope

The Tic Tac Toe game is designed for desktop platforms using Qt for GUI development and SQLite for data persistence. It enables users to play games, register/login, and view historical statistics and replays. The software ensures modular design with separate responsibilities for GUI, logic, and data.

1.3 Definitions, Acronyms and Abbreviations.

- **GUI**: Graphical User Interface.
- **AI**: Artificial Intelligence.
- **SQLite**: A lightweight, embeddable database engine.
- **UML**: Unified Modeling Language.
- **Minimax Algorithm**: An algorithm used in decision-making and game theory.
- **SHA-256**: Cryptographic hash function secures fixed length hashed password.
- **PvE**: Player versus Environment
- **PvP**: Player versus Player

2. System Overview

The system is divided into a GUI layer using Qt, game logic in C++, and a persistent backend using SQLite. It provides functionalities such as move validation, AI response, and storing game history.

2.1 High-Level-Description

The Tic Tac Toe game follows a modular design composed of three core layers: the graphical user interface (GUI), the game logic backend, and a data persistence layer. The GUI is built using Qt Designer and manages player interaction and visual feedback. The backend is implemented in C++ and encapsulates core functionality such as game state management, move validation, win/tie detection, and AI decision-making. Communication between the GUI and backend is facilitated through dedicated wrapper files. These wrapper classes expose backend functionality in a GUI-accessible format, enabling smooth integration and event-driven interaction.

This architecture ensures separation of concerns, scalability, and maintainability. The system is engineered for local desktop usage with no network requirements, ensuring lightweight execution.

2.2 Main Features

- **User Authentication**: Enables new users to create accounts securely with password hashing and existing users to log in.
- **Game Modes**: Offers multiple modes including local multiplayer and AI opponents with varying difficulty levels (Easy, Medium, Hard).
- **Game History**: Tracks and stores game outcomes (win, lose, draw) along with timestamps and unique IDs in an SQLite database.
- **Database Integration**: Utilizes SQLite for persistent storage of user profiles and game data.

3. System Architecture

3.1 High-Level Architecture Diagram

The architecture diagram illustrates the structure of the Tic Tac Toe game system, highlighting components such as:

- **Presentation Layer**: Implemented using Qt, this layer manages all visual elements and user interactions. It presents game windows (e.g., login screen, game board) and captures user input.
- **Logic Layer (Backend)**: This contains core C++ classes responsible for handling game mechanics, AI logic, and input validation.
- **Data Access Layer (SQLite + User system)**: A local SQLite database handles all persistent data including user credentials, match history, and game replays. A user system class acts as a bridge to interact with the database.

To bridge the GUI and logic layers, wrapper classes are used. These act as intermediaries, exposing backend functionality in a way that can be cleanly accessed from Qt widgets and signals/slots.

3.2 Class Diagram

The UML class diagram depicts the key classes and their relationships within the system:

- **Main Window**: Main Qt interface class managing UI widgets and flow.
- **GUI Wrapper**: Connects UI events to core game logic and vice versa.
- **Game Controller**: Central manager of game state, move validation, and turn control.
- **AI**: Implements AI strategies (Easy, Medium, Hard) using algorithms like minimax for decision-making.
- **User System:** Handles registration, login, saving game data, and retrieving history.

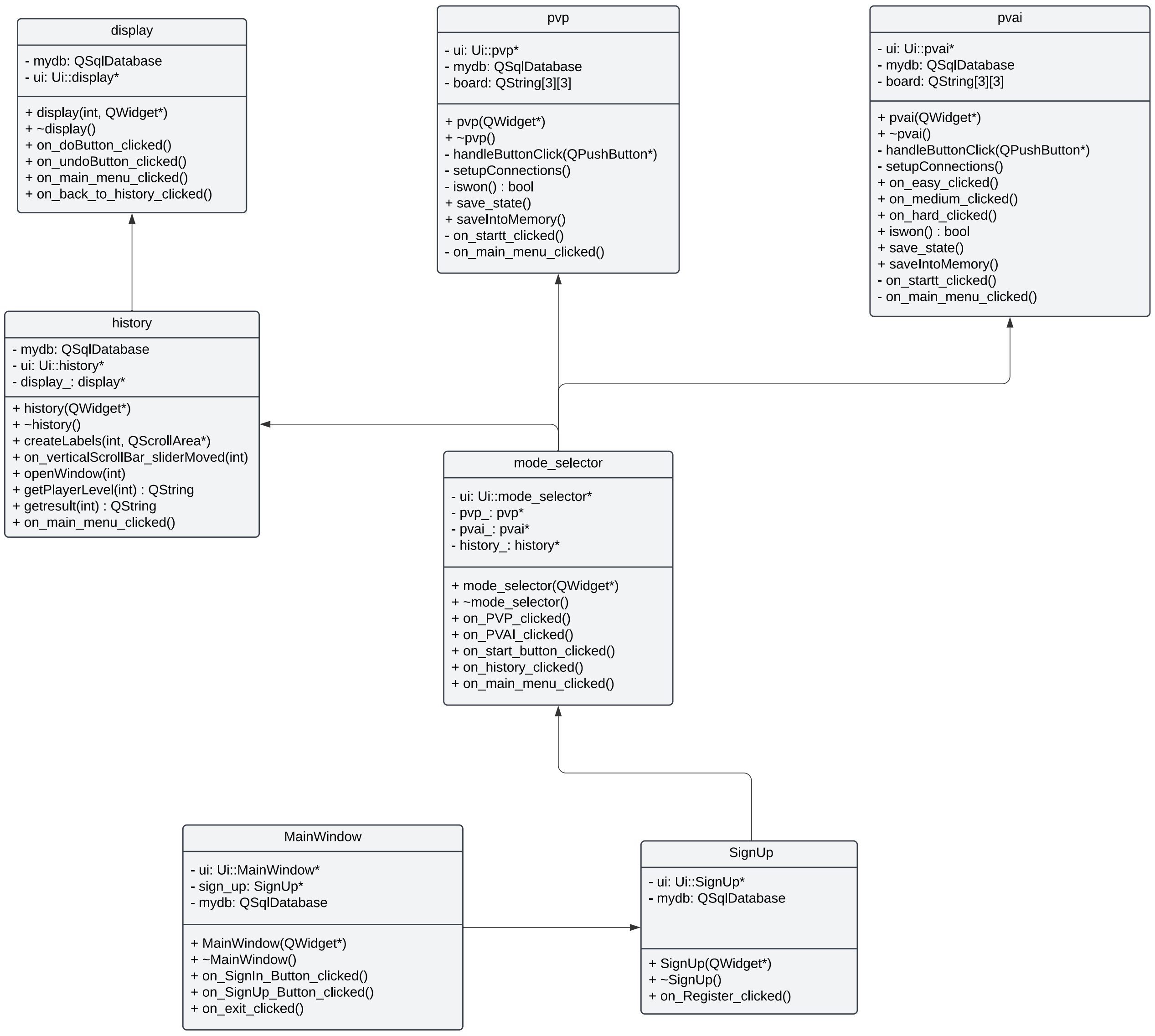

Figure 1: Class Diagram

3.3 Module Descriptions

Main Window (Qt GUI)

- Handles the main visual interface.
- Displays login, mode selection, game grid, statistics, and replay.
- Connects buttons/signals to wrapper functions.

Wrapper Files

- Act as a bridge between Qt and the backend.
- Provide methods callable from UI to invoke Game Controller, User System, and AI logic.

Game Controller

- Manages the overall game state, including board updates, turn switching, and move validation.
- In PvP, it alternates turns between two players, validating each move and checking for win/tie conditions.
- In PvE, it invokes the AI module to generate computer moves on the appropriate turn.
- Triggers game end conditions and interfaces with the UI to display results.

AI Module

- Responsible for generating computer-controlled moves during Player vs Environment (PvE) gameplay.
- Activated by the Game Controller whenever it is the AI's turn to play.
- Supports multiple difficulty levels:
  1. Easy Mode: Uses a random-move strategy by selecting from available cells arbitrarily.
  2. Medium Mode: Implements the Minimax algorithm, which recursively evaluates possible game states to choose the optimal move that either maximizes the AI’s chances of winning or minimizes its chances of losing. It runs with depth two.
  3. Hard Mode: Implements the Minimax algorithm, which recursively evaluates possible game states to choose the optimal move that either maximizes the AI’s chances of winning or minimizes its chances of losing. It runs with depth six.
- Enhanced by alpha-beta pruning technique.
- Generates explanation for each AI move.

User System

- Manages secure user registration/login using SHA-256 hashing.
- Saves and loads game history, including moves for replays.

SQLite Database

- Contains tables for users, games, and moves storage.
- Persistent storage used by User System via SQL queries.

4. Module Descriptions

4.1 Class Descriptions

- User

  Attributes:

  - username
  - hashedPassword
  - winCount, drawCount

  Methods:

  - hash(password: QString) – hashes using QCryptographicHash::Sha256
  - saveToDatabase() – stores user data (e.g., counts, credentials) in SQLite
  - loadState() / saveState() – loads or updates persisted state
- GameLogic

  Attributes:

  - boardState (3×3 array of enums or ints)
  - currentPlayer (enum: Player1, Player2, AI)
  - gameStatus (enum: InProgress, X_Won, O_Won, Draw)

  Methods:

  - updateBoard(position) – applies a move and toggles player
  - checkWin() – returns true if the current state is a win
  - checkDraw() – returns true if the board is full and no winner

- AIPlayer

Attributes:

- o **MoveLog** {row, col, score, depth, explanation, move_number}
- o **difficulty** ("easy", "medium", "hard")

Methods:

- o **LogMove(int row, int col, int score, int depth, const std::string& explanation, int move_number)**

- o **GetLastExplanation()**

- o **Reset()**

- o **Minimax(Board board, bool is_maximizing, char ai_player, char human_player, int depth, int alpha, int beta, int max_depth)**

  core minimax with alpha-beta pruning
- o **EvaluateBoard(const Board& board, char ai_player, char human_player**)

  a scoring function that tells the AI how good or bad a particular board state is, guiding the AI to make intelligent moves.

- o **GenerateExplanation(const Board& board_before, const Board& board_after, char ai_player, char human_player)**

  wrapper that explains each move
- o **GetLegalMoves (const Board& board)**
- o **GetBestMove(const Board& board, char ai_player)**

  wrapper that selects the optimal move

4.2 Data structures

• **Game Board Representation:**

The Tic Tac Toe board is represented using a vector <vector<string>> of size 3×3. Each element corresponds to a cell in the 3×3 grid and stores the current state:

- "X" for Player X
- "O" for Player O
- " " (empty string) for an unoccupied cell

This structure is central to the game logic and is passed between the GUI and backend through wrapper functions.

• **Game Move Format:**

Moves are represented as a vector of pairs:

- pair<int, string>
  - o The int value indicates the index of the move (from 1 to 9).
  - o The comment on the move provided by the AI.

This format enables move-by-move recording and game replay functionality.

• User and Game Data (SQLite Database Schema):

1. **users table**
   Stores user credentials and their statistics.
- username (Credentials
- hashed_password (Text)
- win_count (Integer)
- draw_count (Integer)
- lose_count (Integer)

2. **games table**
   Stores high-level data for each completed game.
- id (Integer, Auto-incremented for every new game)
- player1 (Text)
- player2 (Text)
- winner (Text)

3. **moves table**
   Stores individual moves for each game.
- id (Integer, Auto-incremented for every new game)
- game_id (Integer, Foreign key to games.id)
- move_index (Integer: index from 0 to 8)
- player_symbol (Text: "X" or "O")

4.3 Algorithms

AI Algorithms:

- **Minimax Algorithm**: Used for implementing the hard level AI logic.
- **Blocking and Winning Tactics**: Used for the medium level AI.
- **Random Move Algorithm**: Used for easy level AI.

5. User Interface Design

5.1 Screens and layouts

- **Login/Sign up Windows**: Allows users to sign up for a new account or log in with existing credentials securely.
- **Main Window**: Offers options for Player vs Player, Player vs AI, access to the History Window, and log out
- **AI Window**: Offers two boxes to choose the AI difficulty and choose the Human starting symbol (X or O)
- **Turn Window**: Offers a method to choose Player1 starting symbol (X or O)

- **Game Board Display**: Visual representation of the Tic Tac Toe board with interactive buttons for player moves, previous head-to-head stats between those players and turn indication.
- **End game Window**: Offers the names of the Winner and Loser and allows Rematch or Return to main menu.
- **Game History Table Window**: Offers all previous games for user one either it was player1 or player2.
- **Game Replay Window**: Offers full game replay (Forward and Backward) with the AI comment on each move; AI provide comments on its moves, Human is displayed as human move.

5.2 User Interactions

- **Game Play Interaction**: Users click on empty spaces on the board to place their X or O. Error messages display if a space is already occupied.
- **End Game Interaction**: After a game concludes (win, lose, draw), users can choose to play again, switch game modes, or view their game history.

6. Data Flow and Control Flow

6.1 Data Flow Diagrams

Illustrate the flow of data within the system:

- **User Input**: Sign up/login credentials, move selection.
- **Game Logic**: Updates game state, checks win conditions.
- **Database Interaction**: Saves user profiles, game history.

6.2 Control Flow Diagrams

Show the sequence of operations during gameplay:

- **Player Turn Management**: Switches between players, updates game board.
- **AI Decision Making**: Determines optimal moves based on difficulty level.
- **Game Outcome Check**: Verifies win, lose, or draw conditions.

7. External Interfaces

- **SQLite Database**: Handles persistent storage for user profiles (sign up, login) and game history.
- **GUI Framework (Qt)**: Provides graphical components for the user interface and event handling.

8. Constraints and Assumptions

- **Hardware Requirements**: Runs on standard desktop and laptop computers.
- **Security Assumptions**: Passwords are hashed for storage in the database, and sensitive information (like passwords) is not displayed in plain text.

9. Appendices

- Detailed flowchart illustrating the sequence of actions from sign up/login to gameplay and database interaction

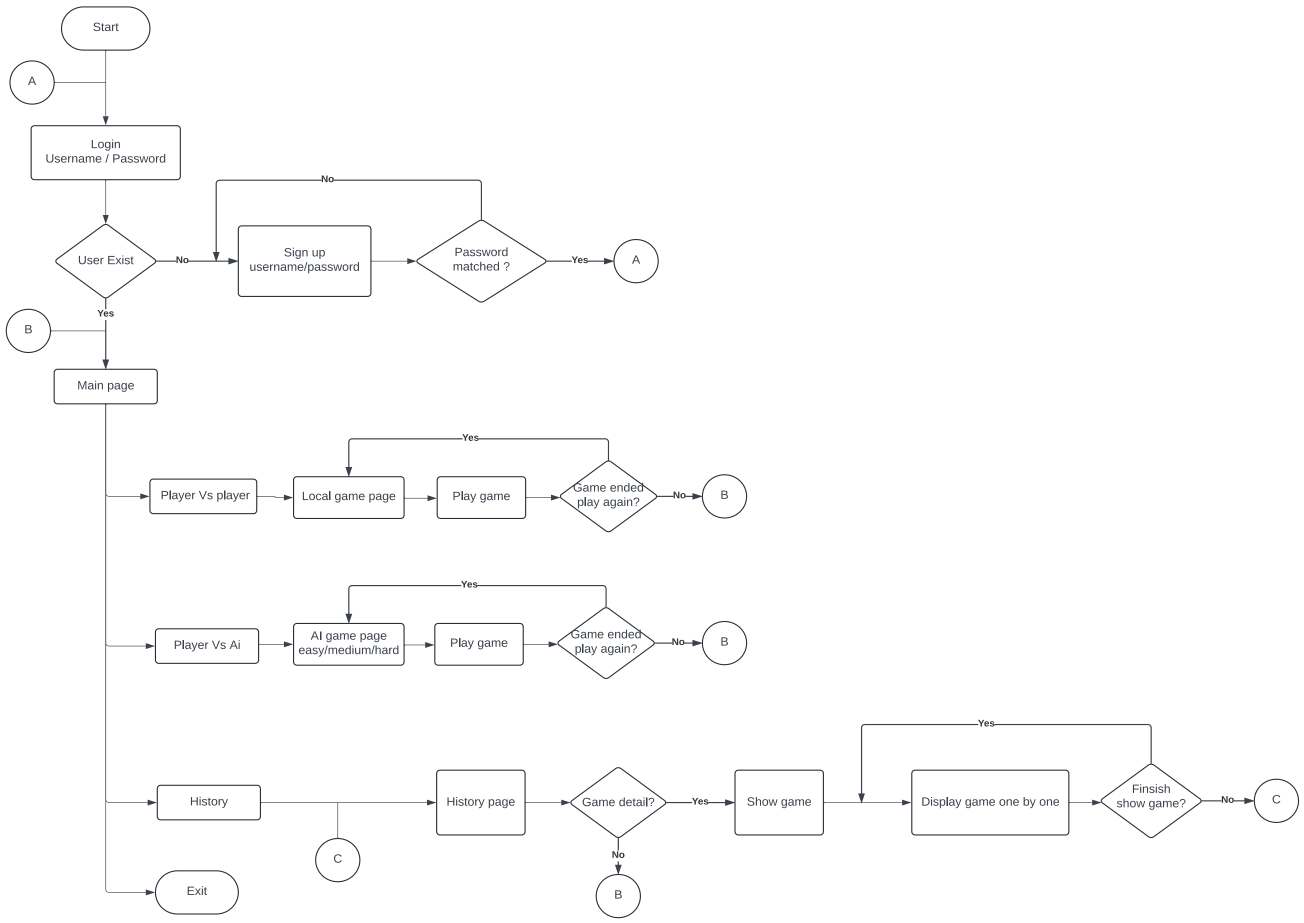

Figure 2: System Flow

Additional Notes: Any supplementary information such as third-party libraries used, development environment setup instructions.