Software Design Specification for Tic-Tac-Toe Game

Submitted to: Dr. Omar Nasr

1. Introduction

1.1 Purpose

This Software Design Specification (SDS) document describes the technical design and implementation details of the Tic-Tac-Toe desktop application, built using Qt and C++. The SDS translates the requirements from the Software Requirements Specification (SRS) into a detailed design, covering system architecture, component interactions, data structures, and implementation strategies. It serves as a guide for developers and testers to implement and verify the system.

1.2 Scope

The Tic-Tac-Toe game is a standalone desktop application with the following features:

- User account management with registration, login, and guest access.
- Two game modes: Player vs. Player (PvP) and Player vs. AI (PvAI) with Easy, Medium, and Hard difficulty levels.
- Gameplay on a 3x3 grid with win/tie detection and score tracking.
- AI opponent using a minimax algorithm with alpha-beta pruning.
- Game history storage and replay for registered users.
- Qt-based graphical user interface (GUI) with multiple windows.
- Unit tests for core functionalities.

This SDS details the system's architecture, class designs, data flow, and storage mechanisms, based on the provided source files.

1.3 Definitions, Acronyms, and Abbreviations

- **PvP**: Player vs. Player mode.
- **PvAI**: Player vs. AI mode.
- **Qt**: Cross-platform C++ framework for GUI and application development.
- **SDS**: Software Design Specification.
- **SRS**: Software Requirements Specification.
- **GUI**: Graphical User Interface.
- **SHA-256**: Secure Hash Algorithm 256-bit for password hashing.

- **JSON**: JavaScript Object Notation, used for game history storage.
- **Minimax**: Algorithm for AI move selection.
- **Alpha-Beta Pruning**: Optimization for minimax to reduce computation.
- **QMessageBox**: Qt component for alerts.
- **QTimer**: Qt component for timed events (e.g., replay delays).

1.4 References

- **SRS**: SRS_TicTacToe_Full.md (artifact_id: 5c495bfa-f306-434c-bab0-db414690f1fb).
- **Qt Documentation**: https://doc.qt.io/
- **Source Files**: main.cpp, mainwindow.cpp, registerwindow.cpp, gamemodewindow.cpp, playernamewindow.cpp, aidifficultywindow.cpp, symbolwindow.cpp, gameboard.cpp, aiplayer.cpp, gamehistory.cpp, historywindow.cpp, settingswindow.cpp, test_gameboard.cpp.
- **Storage Files**: registered_users.json, (history→username_game_history.json.)

2. System Architecture

2.1 Overview

The Tic-Tac-Toe game follows a layered architecture with three main layers:

- **Presentation Layer**: Qt-based GUI classes (MainWindow, GameModeWindow, etc.) handle user interactions and display.
- **Business Logic Layer**: Core classes (GameBoard, AIPlayer, GameHistory) manage gameplay, AI decisions, and history.
- **Data Layer**: Local file storage (registered_users.txt, JSON history files) for persistent data.

The system uses Qt's signal-slot mechanism for event-driven communication between components. Each window is a QMainWindow or QDialog, managing specific user interactions. The architecture is modular, with separate classes for distinct functionalities.

2.2 Component Diagram

- **MainWindow**: Entry point for login and guest access; interfaces with registerwindow.
- **registerwindow**: Handles user registration with password hashing.
- **GameModeWindow**: Manages game mode selection (PvP, PvAI); links to PlayerNameWindow, AIDifficultyWindow, SymbolWindow.
- **PlayerNameWindow**: Collects player names for PvP.

- **AIDifficultyWindow**: Selects AI difficulty for PvAI.
- **SymbolWindow**: Selects player symbol (X or O) for PvAI.
- **GameBoard**: Core gameplay logic, win/tie detection, and score tracking; interfaces with AIPlayer.
- **AIPlayer**: Implements minimax algorithm for AI moves.
- **GameHistory**: Manages game history storage and retrieval.
- **historywindow**: Displays and replays game history.
- **SettingsWindow**: Provides access to account switching and history.
- **test_gameboard**: Unit tests for win detection and history management.

2.3 Data Flow

1. User Authentication:

- User inputs credentials in MainWindow or registers via registerwindow.
- Credentials are stored in registered_users.txt (username:hashed_password).
- Successful login/guest access opens GameModeWindow.

2. Game Setup:

- GameModeWindow directs to PlayerNameWindow (PvP) or AIDifficultyWindow and SymbolWindow (PvAI).
- Configurations (names, difficulty, symbol) are passed to GameBoard.

3. Gameplay:

- GameBoard manages the 3x3 board, processes user clicks, and updates the GUI.
- In PbAI, GameBoard calls AIPlayer::makeMove for AI moves.
- Outcomes (win, tie) are displayed via QMessageBox and saved via GameHistory.

4. History Management:

- GameHistory saves game details to file (history)→username_game_history
- historywindow loads and displays history, replaying moves using QTimer.

5. **Settings**:
    - SettingsWindow allows account switching (to MainWindow) or history access (to historywindow).

3. Component Design

3.1 Class Descriptions

3.1.1 MainWindow (mainwindow.cpp)

- **Purpose**: Handles user login and guest access.
- **Attributes**:
    - ui: Qt UI object for login form.
    - registeredUsers: QMap<QString, QString> for username:hashed_password pairs.
- **Methods**:
    - hashPassword: Computes SHA-256 hash of a password.
    - addRegisteredUser: Adds a new user to registeredUsers.
    - isUsernameTaken: Checks for duplicate usernames.
    - saveRegisteredUsers/loadRegisteredUsers: Manages registered_users.json.
    - Slots: on_LoginButton_clicked, on_GuestButton_4_clicked, on_RegisterButton_2_clicked, on_clearButton_3_clicked.
- **Signals**: None.

3.1.2 registerwindow (registerwindow.cpp)

- **Purpose**: Manages user registration with password strength checking.
- **Attributes**:
    - ui: Qt UI object for registration form.
- **Methods**:
    - hashPassword: Computes SHA-256 hash.
    - checkPasswordStrength: Evaluates password strength (Weak, Medium, Strong).
    - Slots: on_registerButton_clicked, on_cancelButton_2_clicked, onPasswordTextChanged.

- **Signals**: None.

3.1.3 GameModeWindow (gamemodewindow.cpp)

- **Purpose**: Selects game mode (PvP or PvAI).
- **Attributes**:
  - ui: Qt UI object for mode selection.
  - currentUsername: QString for the logged-in user.
  - Pointers: gameBoard, playerNameWindow, aiDifficultyWindow, settingsWindow.
- **Methods**:
  - Slots: on_playerVsPlayer_clicked, on_playerVsAI_clicked, on_settingButton_clicked, onPlayerNamesEntered, onAIDifficultySelected.
- **Signals**: None.

3.1.4 PlayerNameWindow (playernamewindow.cpp)

- **Purpose**: Collects player names for PvP mode.
- **Attributes**:
  - ui: Qt UI object for name input.
- **Methods**:
  - getPlayer1Name/getPlayer2Name: Retrieves trimmed input names.
  - Slots: on_okButton_2_clicked, on_clearButton_clicked.
- **Signals**: namesEntered(QString, QString).

3.1.5 AIDifficultyWindow (aidifficultywindow.cpp)

- **Purpose**: Selects AI difficulty for PvAI mode.
- **Attributes**:
  - ui: Qt UI object for difficulty selection.
  - selectedDifficulty: QString (Easy, Medium, Hard).
- **Methods**:
  - getDifficulty: Returns selected difficulty.
  - Slots: on_easyButton_clicked, on_mediumButton_clicked, on_hardButton_clicked , on_backButton_2_clicked.

- **Signals**: difficultySelected(const QString &difficulty) , backToGameMode.

3.1.6 SymbolWindow (symbolwindow.cpp)

- **Purpose**: Selects player symbol (X or O) for PvAI mode.
- **Attributes**:
    - ui: Qt UI object for symbol selection.
    - selectedSymbol: QString (X or O).
- **Methods**:
    - getSelectedSymbol: Returns selected symbol.
    - Slots: on_xButton_clicked, on_oButton_2_clicked.
- **Signals**: symbolSelected(QString).

3.1.7 GameBoard (gameboard.cpp)

- **Purpose**: Manages gameplay, board state, and UI updates.
- **Attributes**:
    - ui: Qt UI object for the 3x3 grid, status, and scores.
    - board: int[3][3] (0: empty, 1: Player 1/X, 2: Player 2/O or AI).
    - currentPlayer: int (1 or 2).
    - gameMode: Enum (PlayerVsPlayer, PlayerVsAI).
    - aiDifficulty: QString (Easy, Medium, Hard).
    - aiPlayer: Pointer to AIPlayer.
    - playerSymbol, aiSymbol: char (X or O).
    - player1Name, player2Name: QString for player names.
    - player1Score, player2Score: int for scores.
    - gameHistory: GameHistory object for saving games.
    - currentGameMoves: QList<QPair<int, int>> for move tracking.
- **Methods**:
    - onButtonClicked: Handles cell clicks, updates board, and triggers AI moves.
    - checkWin: Checks for three identical symbols in rows, columns, or diagonals.

- resetBoard: Clears board and resets state.
- updateStatus/updateScore: Updates GUI labels.
- saveGameHistory: Saves game details via GameHistory.
- setGameMode, setAIDifficulty, setPlayerNames, setPlayerSymbol: Configures game settings.
- Slots: on_resetButton_clicked, on_ReturnButton_clicked.

- **Signals**: None.

3.1.8 AIPlayer (aiplayer.cpp)

- **Purpose**: Implements AI logic for PvAI mode.
- **Attributes**:
  - aiSymbol, playerSymbol: char (X or O).
  - difficulty: QString (Easy, Medium, Hard).
- **Methods**:
  - makeMove: Selects AI move based on difficulty (random or minimax).
  - minimax: Recursive algorithm with alpha-beta pruning.
  - evaluateBoard: Scores board (+10 AI win, -10 player win, 0 tie).
  - isBoardFull: Checks if board is full.
  - checkWin: Checks for wins for a given symbol.
- **Signals**: None.

3.1.9 GameHistory (gamehistory.cpp)

- **Purpose**: Manages game history storage and retrieval.
- **Attributes**:
  - historyFilePath: QString for JSON file (history)→username_game_history.
- **Methods**:
  - saveGame: Saves a GameSession to JSON.
  - loadGames: Loads all saved games.
  - clearHistory: Clears the JSON file.
- **Signals**: None.

3.1.10 historywindow (historywindow.cpp)

- **Purpose**: Displays and replays game history.
- **Attributes**:
  - ui: Qt UI object for history list and replay grid.
  - gameHistory: GameHistory object.
  - games: QVector for loaded games.
  - currentReplayIndex, currentMoveIndex: int for replay state.
  - replayTimer: QTimer for 1-second move delays.
- **Methods**:
  - displayHistory: Populates history list.
  - resetReplayBoard: Clears replay grid.
  - replayNextMove: Displays next move during replay.
  - Slots: on_historyList_itemClicked, on_replayButton_clicked, on_clearHistoryButton_2_clicked.
- **Signals**: None.

3.1.11 SettingsWindow (settingswindow.cpp)

- **Purpose**: Provides access to account switching and history.
- **Attributes**:
  - ui: Qt UI object for settings.
  - currentUsername: QString for the logged-in user.
- **Methods**:
  - Slots: on_switchaccountButton_3_clicked, on_historyButton_2_clicked , on_exitButton_clicked , on_backButton_clicked.
- **Signals**: None.

3.1.12 TestGameBoard (test_gameboard.cpp)

- **Purpose**: Unit tests for core functionality.
- **Attributes**: None.

- **Methods**:
    - testCheckWin: Tests win detection for horizontal, vertical, and diagonal cases.
    - testGameHistorySaveLoad: Tests history save/load functionality.
- **Signals**: None.

3.2 Data Structures

3.2.1 Game Board

- **Type**: int[3][3] in GameBoard.
- **Description**: Represents the 3x3 grid.
    - 0: Empty cell.
    - 1: Player 1 or X.
    - 2: Player 2 or O (AI in PvAI mode).
- **Usage**: Updated on player/AI moves, checked for win/tie conditions.

3.2.2 AI Board

- **Type**: QVector<QVector> in AIPlayer.
- **Description**: Temporary 3x3 board for AI computations.
    - ' ': Empty cell.
    - 'X' or 'O': Player or AI symbol.
- **Usage**: Used in makeMove and minimax to simulate moves.

3.2.3 GameSession

- **Type**: Struct in GameHistory.
- **Fields**:
    - player1Name, player2Name: QString for player names.
    - gameMode: QString (Player vs Player, Player vs AI).
    - outcome: QString (e.g., “Alice wins”, “Tie”).
    - timestamp: QDateTime.
    - moves: QList<QPair<int, int>> for move coordinates.
    - winnerSymbol: QString (X, O, or empty for tie).

- **Usage**: Stored in JSON for history.

3.2.4 Registered Users

- **Type**: QMap<QString, QString> in MainWindow.
- **Description**: Maps usernames to SHA-256 hashed passwords.
- **Usage**: Stored in registered_users.json.

3.2.5 JSON History File

- **Format**: QJsonArray of QJsonObject.
- **Fields per Object**:
    - player1Name, player2Name, gameMode, outcome, timestamp, winnerSymbol.
    - moves: QJsonArray of {row, col} objects.
- **File**: tictactoe_<username>_history.json.

4. Implementation Details

4.1 Qt Signal-Slot Mechanism

- **Description**: Qt's signal-slot system connects user actions (e.g., button clicks) to corresponding methods.
- **Examples**:
    - PlayerNameWindow::namesEntered → GameModeWindow::onPlayerNamesEntered.
    - AIDifficultyWindow::difficultySelected → GameModeWindow::onAIDifficultySelected.
    - SymbolWindow::symbolSelected → Lambda in GameModeWindow.
    - Button clicks in GameBoard (e.g., button_0_0) → onButtonClicked.

4.2 AI Algorithm

- **Minimax with Alpha-Beta Pruning**:
    - AIPlayer::makeMove:
        - Easy: Random selection from empty cells using rand().
        - Medium/Hard: Iterates over empty cells, simulates moves, and calls minimax.
    - minimax: Recursive function with depth limit (2 for Medium, 9 for Hard).
        - Evaluates board with evaluateBoard (+10 AI win, -10 player win, 0 tie).

        - Uses alpha-beta pruning to skip branches where beta <= alpha.
    - evaluateBoard: Checks rows, columns, and diagonals for wins.
    - isBoardFull: Checks for tie condition.

4.3 File Storage

- **registered_users.json**:
    - Format: username:hashed_password\n.
    - Read/written by MainWindow::loadRegisteredUsers/saveRegisteredUsers.
- **game_history.json**:
    - Format: JSON array of game sessions.
    - Managed by GameHistory::saveGame, loadGames, clearHistory.
    - Uses Qt's QJsonDocument, QJsonArray, QDir and QJsonObject.

4.4 GUI Implementation

- **Qt Designer**: UI files (e.g., ui_mainwindow.h) define layouts for each window.
- **Widgets**:
    - QPushButton: Used for grid cells, mode selection, difficulty, symbols, etc.
    - QLineEdit: For username, password, and player names.
    - QLabel: For status, scores, and password strength.
    - QListWidget: For history display in historywindow.
    - QMessageBox: For alerts (e.g., win, tie, errors).
- **Styling**: Password strength in registerwindow uses color-coded labels (red: Weak, orange: Medium, green: Strong).

4.5 Testing

- **Framework**: Qt Test (test_gameboard.cpp).
- **Tests**:
    - testCheckWin: Verifies win detection for horizontal, vertical, and diagonal cases.
    - testGameHistorySaveLoad: Verifies saving and loading a game session.
- **Execution**: Uses QTEST_MAIN macro to run tests.

5. Non-Functional Design Considerations

5.1 Performance

- **GUI Response**: Button clicks handled within 0.5 seconds using Qt's event loop.
- **AI Performance**: Minimax with alpha-beta pruning ensures Hard mode moves within 1 second (optimized by depth limits).
- **File I/O**: JSON history loading/saving optimized using Qt's QJsonDocument.

5.2 Security

- **Password Hashing**: SHA-256 via QCryptographicHash in MainWindow and registerwindow.
- **Input Validation**: Username and password fields trimmed to prevent empty or whitespace-only inputs.
- **File Access**: Debug logs (qDebug) for failed file operations.

5.3 Usability

- **Intuitive GUI**: Consistent Qt widget usage (e.g., buttons for actions, labels for feedback).
- **Feedback**: Real-time password strength updates, immediate alerts for errors or game outcomes.
- **Replay**: 1-second delay per move in historywindow for clear visualization.

5.4 Maintainability

- **Modularity**: Separate classes for UI, logic, and data.
- **Debugging**: Extensive qDebug logs for errors (e.g., file access, invalid inputs).
- **Code Structure**: Follows Qt conventions (e.g., ui->setupUi, signal-slot connections).

5.5 Portability

- **Qt Framework**: Ensures cross-platform compatibility (Windows, Linux, macOS).
- **Dependencies**: Minimal, requiring only Qt and C++ standard libraries.

6. Assumptions and Constraints

6.1 Assumptions

- Qt and C++ compiler are installed and configured.
- System has read/write permissions for registered_users.txt and JSON history files.

- Users will not manually edit storage files, which could cause data corruption.
- Hardware supports Qt GUI rendering (e.g., Intel i5, 4GB RAM).

6.2 Constraints

- **Storage**: Limited to local files, no database or cloud support.
- **AI**: Minimax algorithm with depth limits (0, 2, 9) to balance performance.
- **Platform**: Desktop-only, no mobile or web support.
- **Testing**: Limited to unit tests for win detection and history management.
- **Language**: C++ with Qt, limiting portability to Qt-supported environments.

7. Appendices

7.1 Class Diagram (Text-Based)

MainWindow

├── registerwindow

└── GameModeWindow

├── PlayerNameWindow

├── AIDifficultyWindow

│ └── SymbolWindow

├── GameBoard

│ └── AIPlayer

└── SettingsWindow

└── historywindow

└── GameHistory

TestGameBoard (tests GameBoard, GameHistory)

7.2 Sample JSON History File

```
[
 {
  "player1Name": "Alice",
  "player2Name": "AI",
```

```
  "gameMode": "Player vs AI",
  "outcome": "Alice wins",
  "timestamp": "2025-06-20T06:17:00",
  "winnerSymbol": "X",
  "moves": [
   {"row": 0, "col": 0},
   {"row": 1, "col": 1},
   {"row": 0, "col": 1},
   {"row": 2, "col": 2},
   {"row": 0, "col": 2}
  ]
 }
]
```


7.3 UI Layouts (Text-Based)

- **MainWindow**:
  - QLineEdit: Username, Password.
  - QPushButton: Login, Register, Guest, Clear.
- **registerwindow**:
  - QLineEdit: Username, Password, Confirm Password.
  - QLabel: Password strength (color-coded).
  - QPushButton: Register, Cancel.
- **GameModeWindow**:
  - QPushButton: Player vs Player, Player vs AI, Settings.
- **PlayerNameWindow**:
  - QLineEdit: Player 1 Name, Player 2 Name.
  - QPushButton: OK, Clear.
- **AIDifficultyWindow**:

    - QPushButton: Easy, Medium, Hard , Back.
- **SymbolWindow**:
    - QPushButton: X, O.
- **GameBoard**:
    - 3x3 QPushButton grid for gameplay.
    - QLabel: Current player, scores.
    - QPushButton: Reset, Return.
- **SettingsWindow**:
    - QPushButton: Switch Account, History , Exit , Back.
- **historywindow**:
    - QListWidget: Game history list.
    - 3x3 QPushButton grid for replay.
    - QPushButton: Replay, Clear History.

8. Diagrams

8.1. Sequence Diagram

The sequence diagram demonstrates a typical user session, starting from launching the application, registering or logging in, selecting game settings, and playing a match. It shows message flows between objects like **MainWindow, RegisterWindow, GameModeWindow, PlayerNameWindow, SymbolWindow, AIDifficultyWindow, GameBoard, SettingsWindow**, and **HistoryWindow**. This visualization confirms *proper control flow, especially the AI difficulty selection timing and game history saving*.

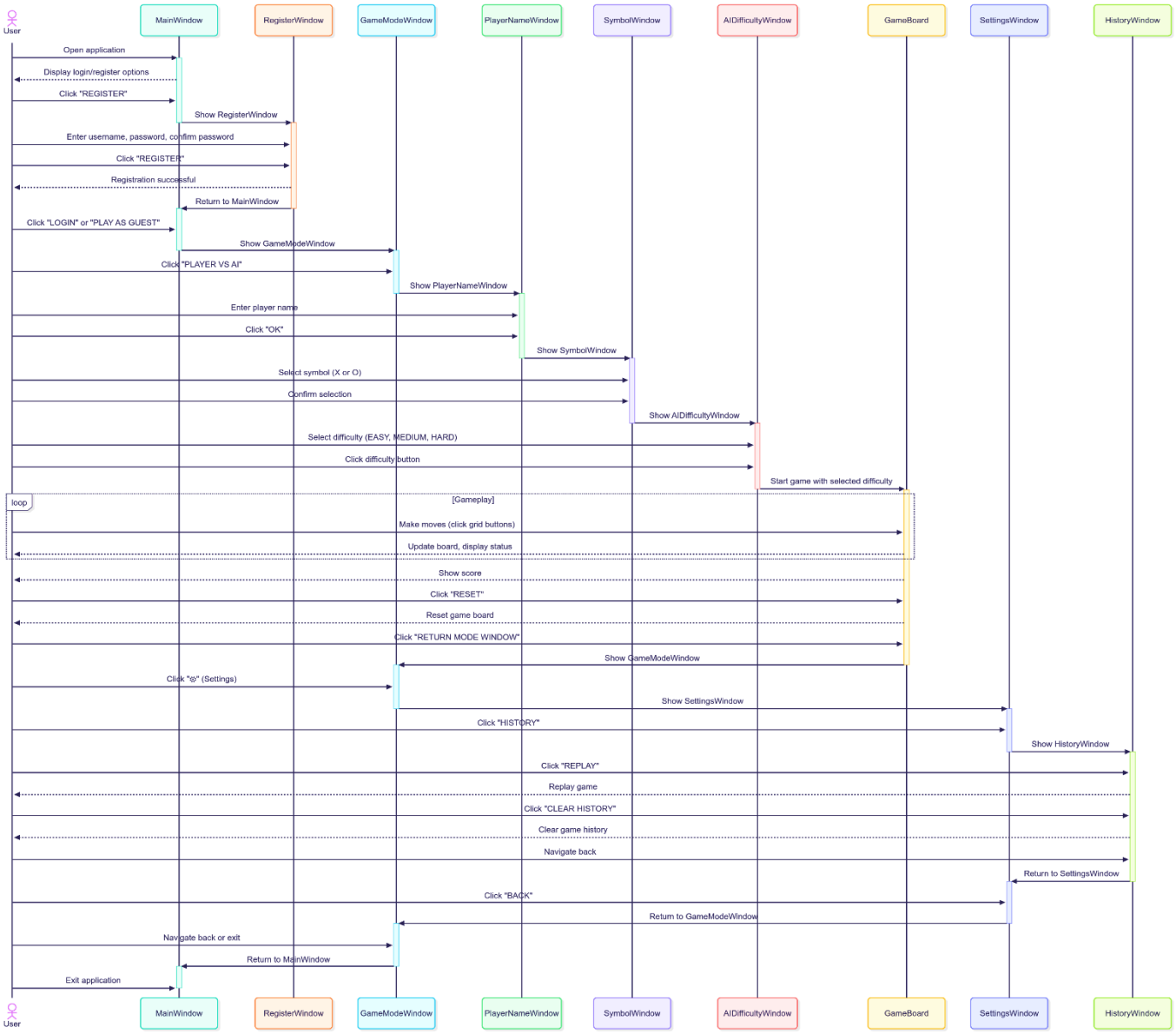


Figure 1: Sequence Diagram

8.2. Class Diagram

The class diagram displays the structure and relationships between main classes: **MainWindow, RegisterWindow, GameModeWindow, PlayerNameWindow, SymbolWindow, AIDifficultyWindow, GameBoard, SettingsWindow, and HistoryWindow**. It includes attributes and methods of each class, emphasizing navigation relationships (e.g**., MainWindow navigates to RegisterWindow) and dependencies (e.g., GameBoard starts from AIDifficultyWindow**). This model supports understanding of the internal logic and codebase structure.

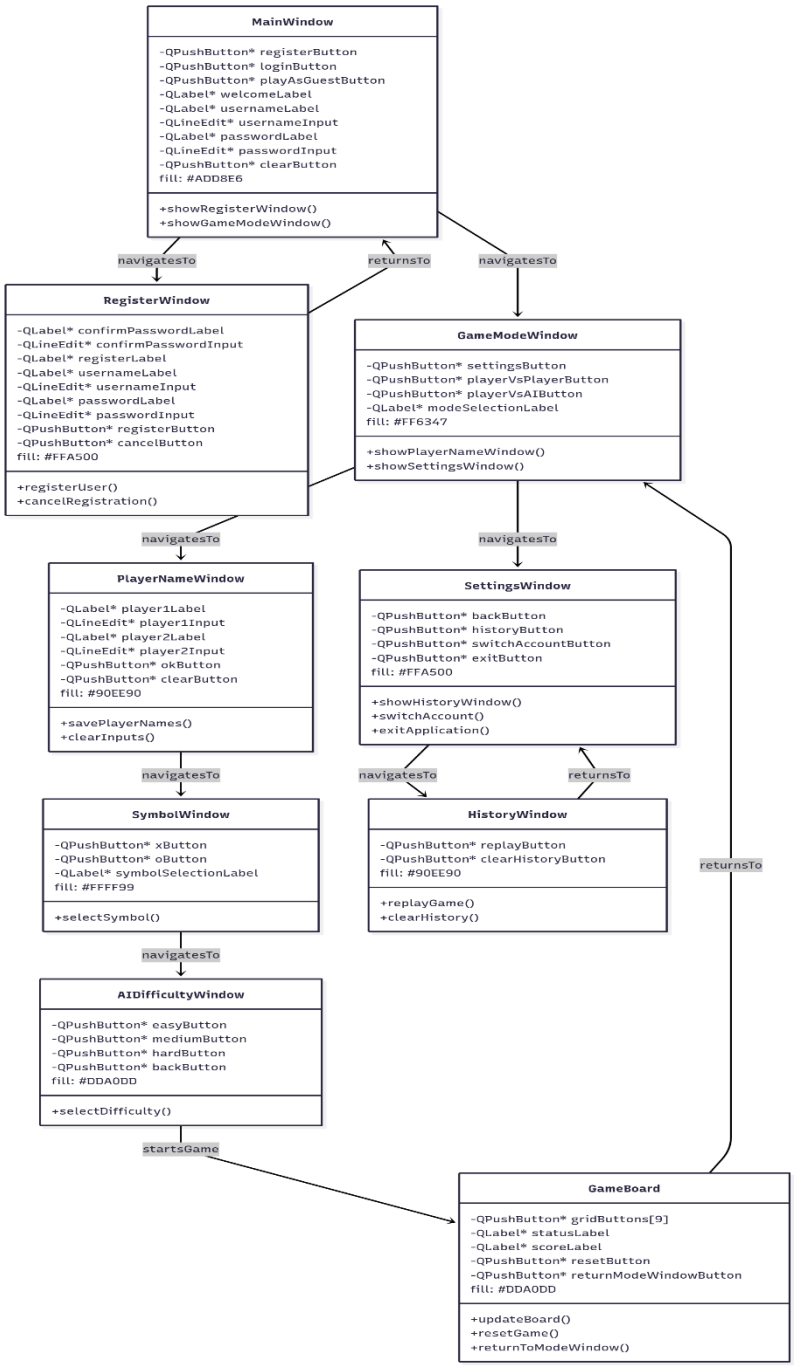


Figure 2: Class Diagram

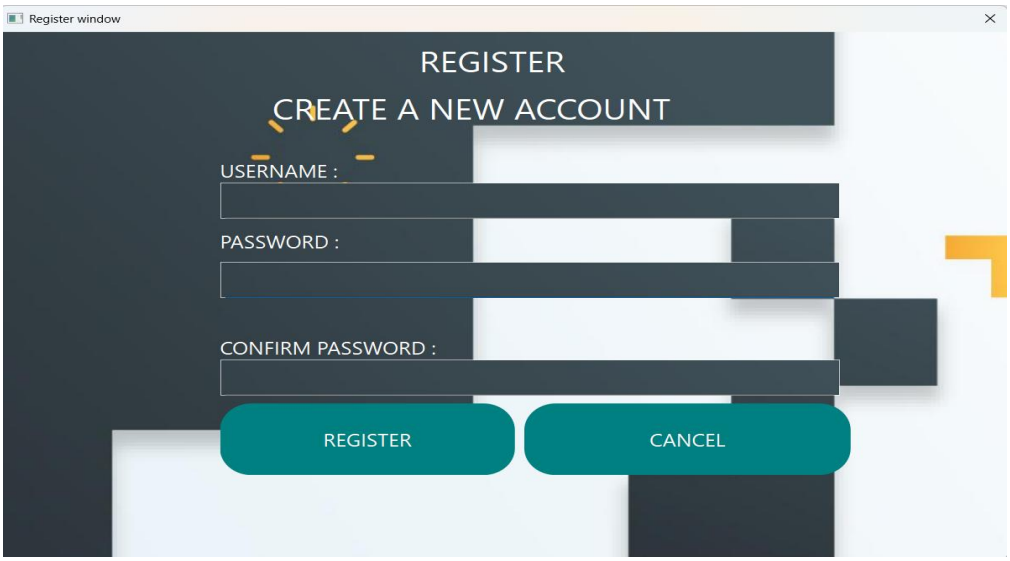
9. User Interface Screenshots

Below are UI screens with explanations to illustrate the application's user experience and flow.

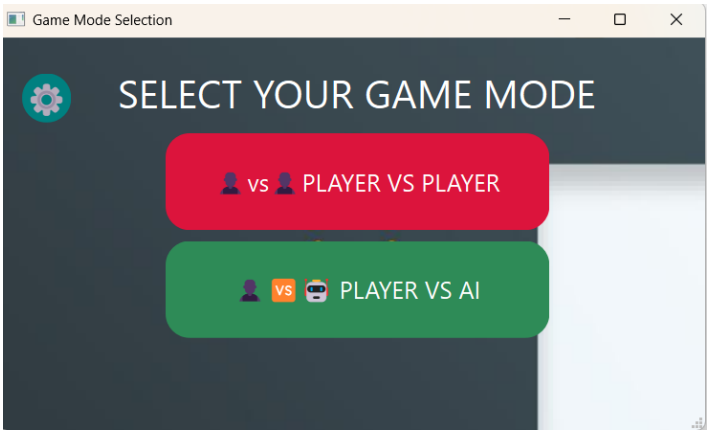
9.1. GameWindows

- **MainWindow :** Main screen showing the TIC TAC TOE tile with WELCOME TO THE GAME and buttons for Login , Clear , Play as Guest and Register.

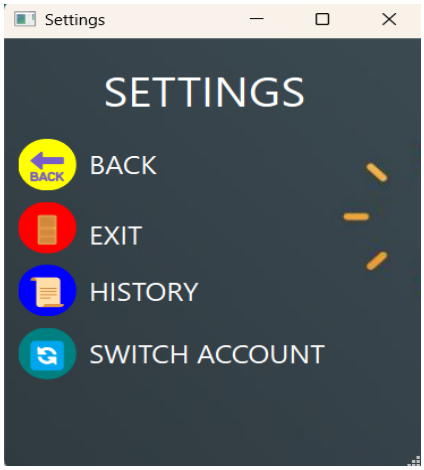
- **Register Window :** Registration form with fields for username, password, and confirm password, and buttons (REGISTER and CANCEL).



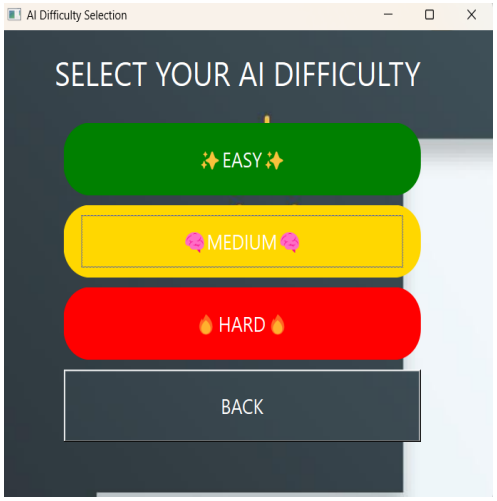
- **Game Mode Window :** Game mode selection screen with options for Player vs Player and Player vs AI, each represented by buttons with corresponding icons.



- **settings:** Settings menu with options including BACK, EXIT, HISTORY, and SWITCH ACCOUNT, each represented by distinct buttons with icons.

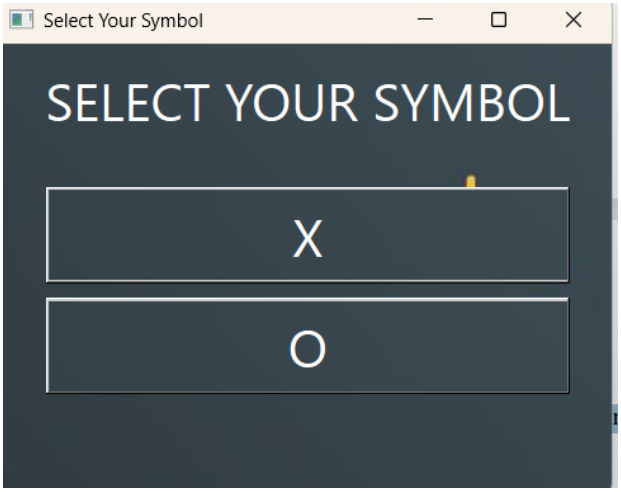


- **AI Difficulty Selection :** AI difficulty selection screen with options for EASY, MEDIUM, and HARD, each represented by buttons with corresponding icons, and a BACK button.

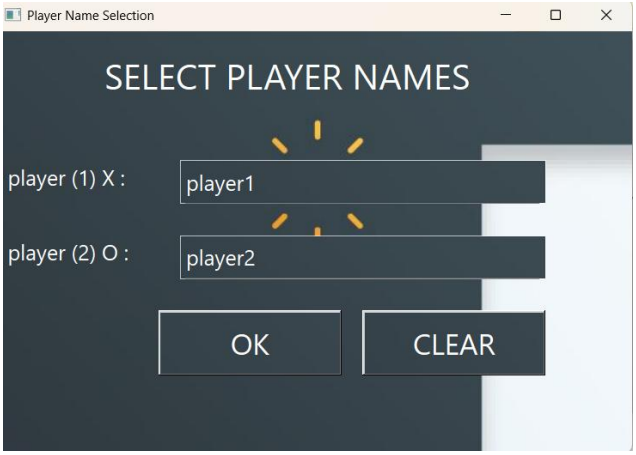


- **Game History :** Game history screen displaying of past game results with Game No. , Playing Mode , Winner, and Date .

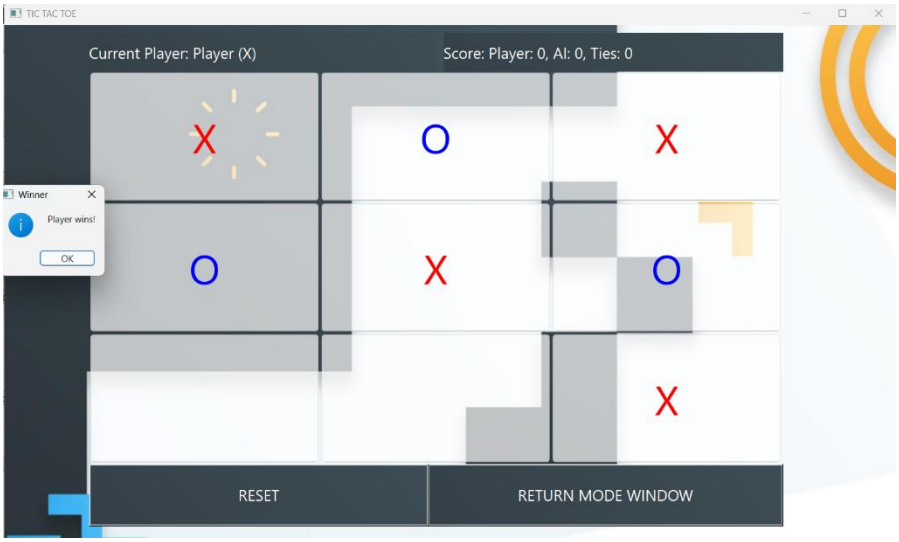
- **Symbol Selection :** Symbol selection screen with options to choose between X and O, each represented by a button.



- **Player Name Selection :** Player name selection screen with fields to enter names for player 1 (X) and player 2 (O).

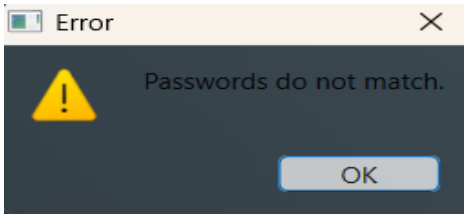


- **Board & Results :** Game board during play and QMessageBox showing the win reslt.

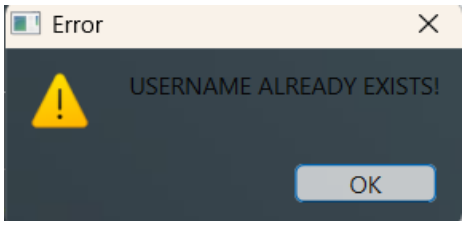


9.2. Error Dialogs

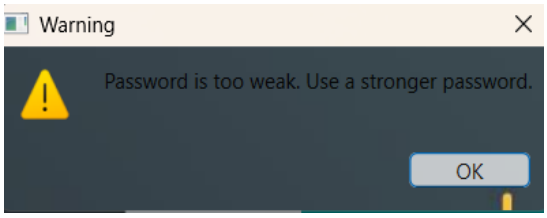
- **Register Fault :** Error screen displayed when the user enters a username, password, and confirm password where the passwords do not match.



- **Register Fault :** Error screen displayed when a username already exists.



- **Register Fault :** Warning screen displayed when a password is too weak, advising to use a stronger password.



- **Login Fault :** Error screen displayed when login with wrong username or password.