Software Design Specification (SDS)

Advanced Tic Tac Toe Game

First of all, the purpose of the System Design Specification is to describe the design and architecture of the game for example

- Ai opponent with different difficulty levels (Easy, Medium, Hard).
- User authentication and sign ups.
- Game History.
- User Interface.

Scope

This SDS includes:

- System Architecture and description.
- Detailed component specifications.
- User Interface Architecture.
- Data Flow Architecture.
- Class Diagram.
- AI Logic

1. System Architecture and description

- **Overall System Structure**

Presentation Layer (UI Tier)

MainWindow: Central UI controller managing all user interactions and view transitions

PlayerChoiceDialog & DifficultyDialog: Specialized input dialogs for game configuration

BoardStateDelegate: Custom rendering component for game history visualization

Qt Designer UI Files: Declarative UI definitions with cyberpunk styling

---

Business Logic Layer (Application Tier)

Game Class: Core game engine handling board state, move validation, win/draw detection, and multi-level AI logic

User Class: Authentication and session management with secure password hashing

GameHistoryModel: Qt model providing structured access to game records with proper MVC integration

---

Data Access Layer (Persistence Tier)

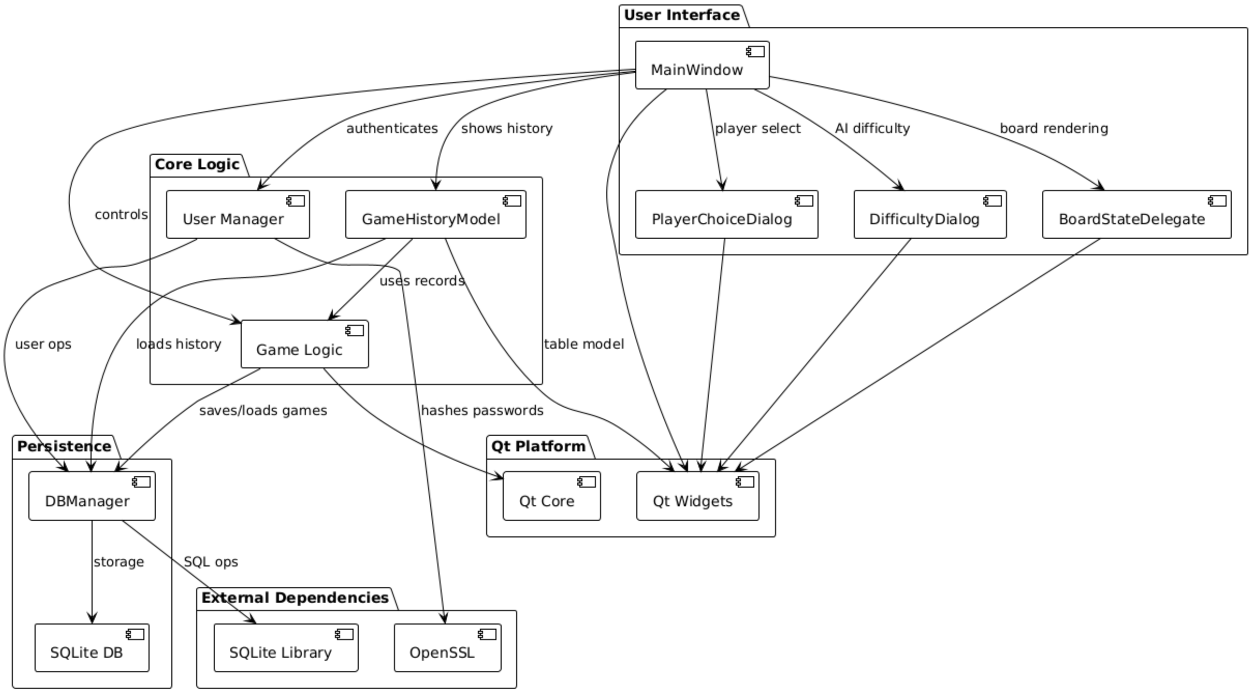
DBManager: Singleton database manager providing centralized SQLite operations

SQLite Database: Persistent storage for user credentials and game history

GameRecord Structure: Data transfer object for game state serialization

---

- **Module Dependency Structure**



- **Architectural Patterns Applied**

Singleton Pattern

DBManager Class: Ensures single database connection instance across the application
Implementation: getInstance() method with static instance and private constructor
Benefits: Centralized database access, connection pooling, and thread safety

Model-View-Controller (MVC)

Model: Game class (business logic), GameHistoryModel (data representation)
View: Qt widgets, custom dialogs, and BoardStateDelegate for rendering
Controller: MainWindow orchestrating user interactions and coordinating between model and view
Benefits: Clear separation of concerns, maintainable code structure

Observer Pattern (Qt Signals/Slots)

Implementation: Extensive use of Qt's signal-slot mechanism for event handling
Examples: Button clicks, menu actions, board updates
Benefits: Loose coupling between UI components and business logic

Strategy Pattern

AI Difficulty System: Different algorithms based on AIDifficulty enum
Strategies: makeEasyAIMove(), makeMediumAIMove(), makeHardAIMove(), makeAIMoveWithTree()
Benefits: Extensible AI behavior without modifying core game logic

Factory Pattern (Implicit)

Dialog Creation: Dynamic instantiation of PlayerChoiceDialog and DifficultyDialog
AI Move Generation: Factory-like selection of AI strategies based on difficulty
Benefits: Flexible object creation and configuration

Delegate Pattern

BoardStateDelegate: Custom rendering for game board states in history view
Implementation: Inherits QStyledItemDelegate for specialized painting
Benefits: Separation of data representation from rendering logic

Data Transfer Object (DTO)

GameRecord Structure: Encapsulates game state data for persistence and transfer
Benefits: Clean data contracts between layers, serialization support

Template Method Pattern

Qt Framework Integration: Overriding virtual methods in QAbstractTableModel, QStyledItemDelegate
Benefits: Framework extension points with consistent behavior

Main Modules and their Responsibilities

Module	Description
Game	Handles Tic Tac Toe mechanics, AI moves, Minimax evaluation, win detection, draw detection, and state management.
MainWindow	Provides the user interface, handles input, displays results, and manages interaction between Game and user.
DBManager	Manages user sign-in, sign-up, and database persistence of game results and history
PlayerChoiceDialog	Dialog for selecting player symbol (X or O).
DifficultyDialog	Dialog for selecting AI difficulty level.
UI Layers	Providing an attractive and responsive experience for users.

2.Detailed Component Specifications

Game Logic

- Maintains game state.
- Supports AI with four difficulty levels:
  - **Easy**: Random moves.
  - **Medium**: Mix of random and smart moves.
  - **Hard**: Smart move with some randomness.
- Provides:

- makeMove(int row, int col): Performs a move for the current player if the cell is empty.

- switchPlayer(): Switches turn between 'X' and 'O'.

- checkWin(): Checks if the current player has won the game.
- checkDraw(): Checks if the board is full and no player has won.
- getBoardValue(int row, int col): Returns the value ('X', 'O', or ' ') at a specific cell.
- getBoardStateAsString(): Returns the board as a flat string (used for database storage).
- getCurrentPlayer(): Returns the current player's character ('X' or 'O').
- setAIDifficulty(AIDifficulty): Sets the current AI difficulty.
- getAIDifficulty(): Returns the currently set AI difficulty.
- makeAIMove(): Dispatches the appropriate AI move logic based on difficulty.
- makeEasyAIMove(): Random move from available positions.
- makeMediumAIMove(): Random move or attempt to win/block.
- makeHardAIMove(): Optimal strategies with slight imperfection.
- makeAIMoveWithTree(): Perfect AI using Minimax tree evaluation.
- evaluateBoard(char[3][3]): Evaluates the given board and returns +1, 0, or -1.
- buildGameTree(): Builds the full tree of possible moves.
- minimaxTree(): Implements the recursive minimax scoring logic.
- findImminentWinOrBlock(): Checks for immediate wins or required blocks.
- deleteTree(): Frees memory of the minimax game tree.
- saveGame(username, result, opponent): Saves a completed game to the database.
- getGameHistory(username): Retrieves all previous games for a user.

MainWindow (UI Logic)

Features

- Provides an interactive 3x3 grid.

- Displays status (player turn, AI difficulty).
- Shows results.
- Enables/Disables buttons based on state.
- Manages user choices:
  - New Game
  - Play AI vs Human
  - View Game History
  - Sign Up / Login
- Displays **Game History** table.

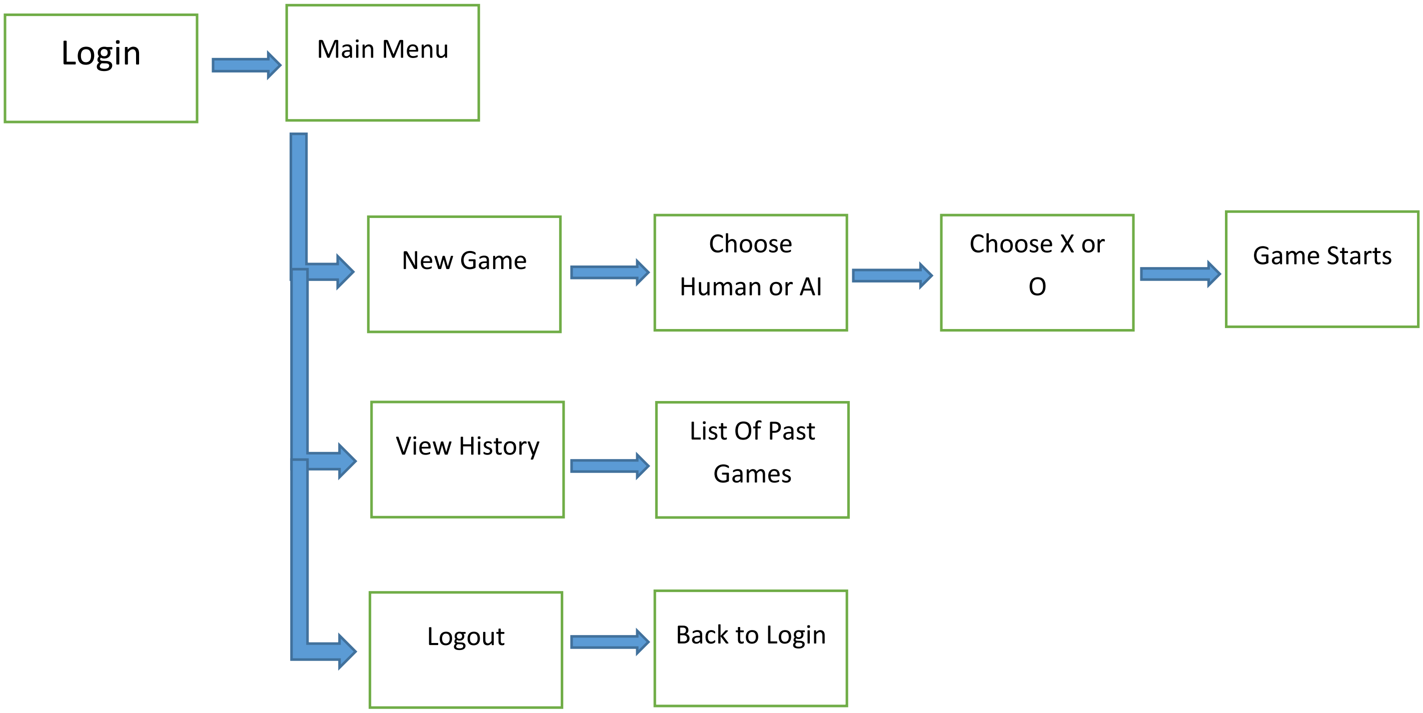
Main Methods

- `processAIMove()`: Triggers AI move after delay.
- `updateGameBoard()`: Updates UI based on `Game` state.
- `displayGameResult()`: Displays win/loss/draw status.
- `showPlayerChoiceSelector()`: Displays X/O choice dialog.
- `showDifficultySelector()`: Displays AI difficulty dialog.

Database Manager (**`DBManager`**)

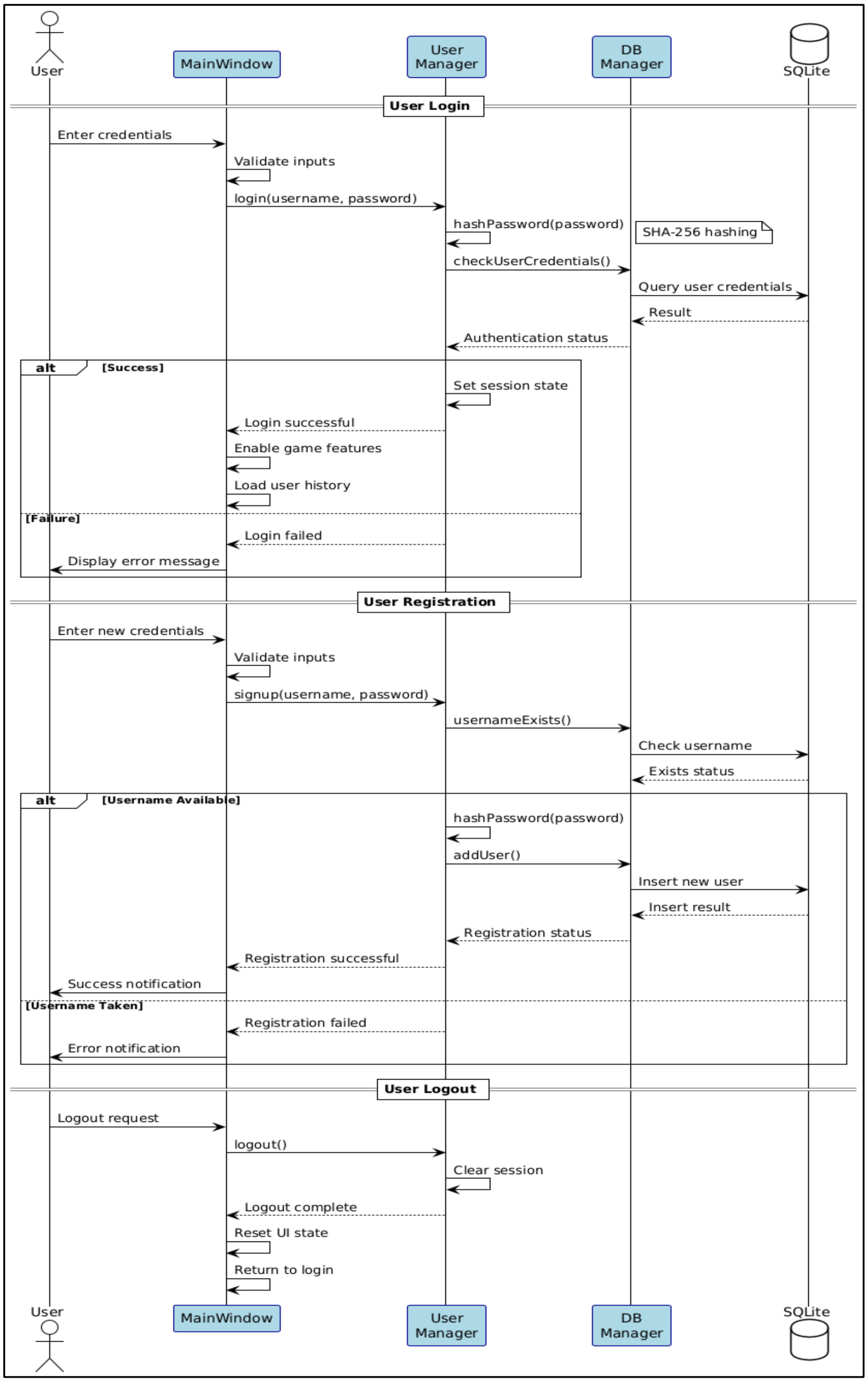
- Provides a **SQLite** database for:
  - User Registration (`addUser()`)
  - User Login (`checkUserCredentials()`)
  - Game History (`saveGameRecord()` / `getGameRecords()`)

3. User Interaction Flow Diagram

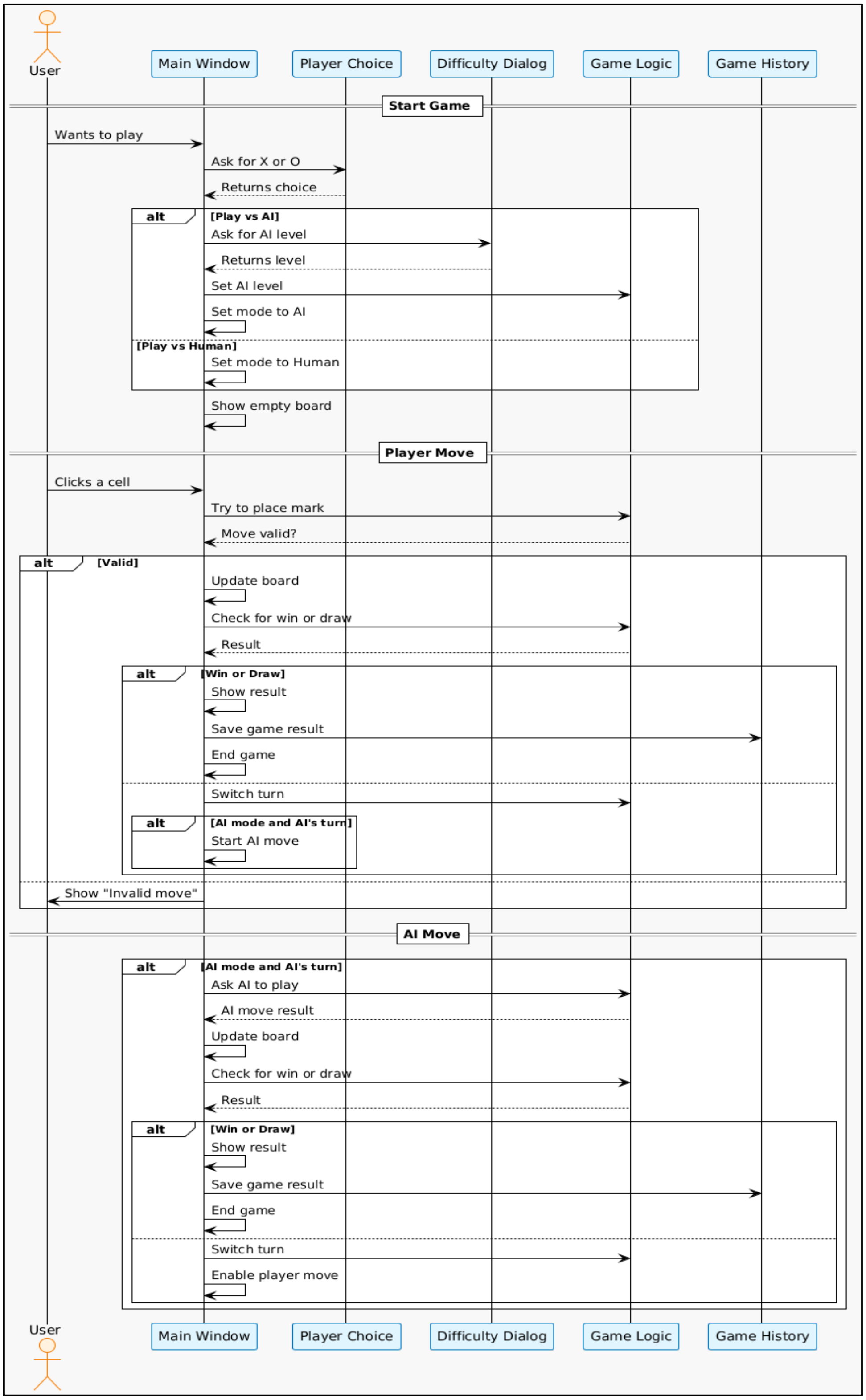


4.Data Flow Architecture

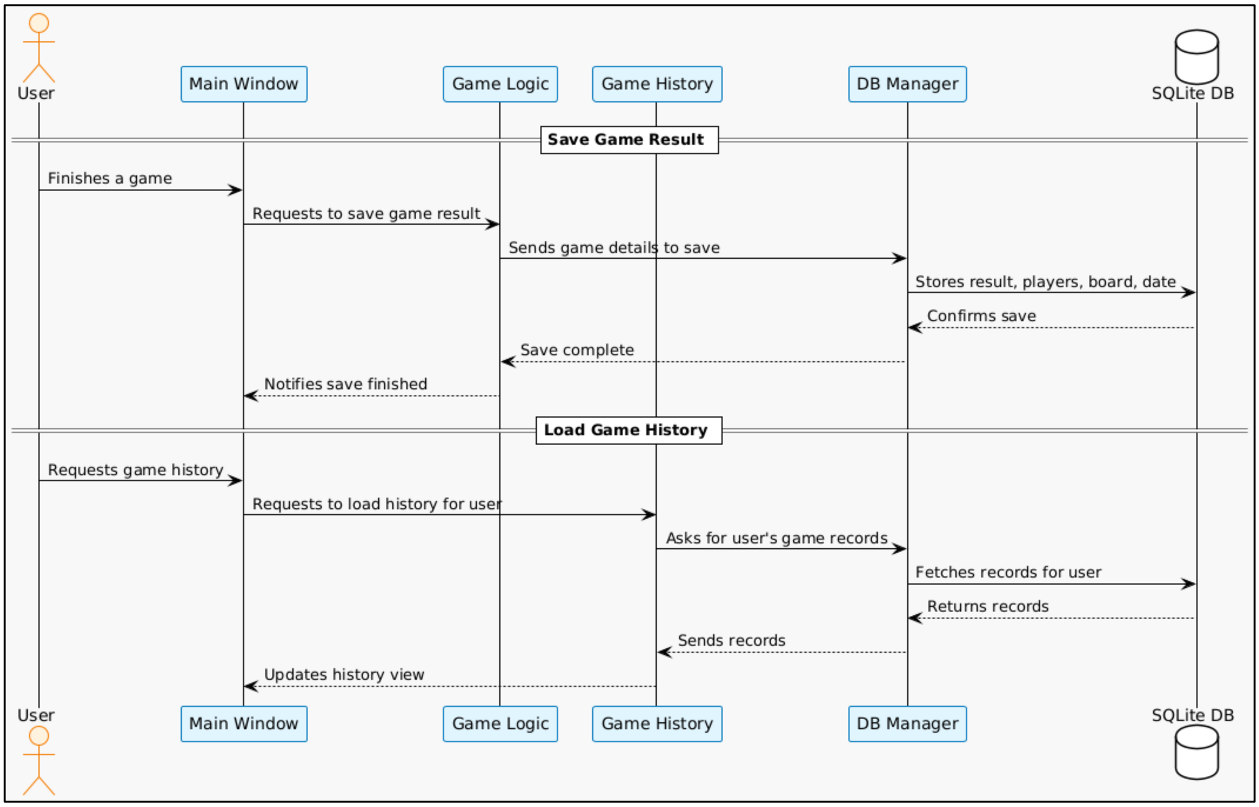
- ## Authentication Flow



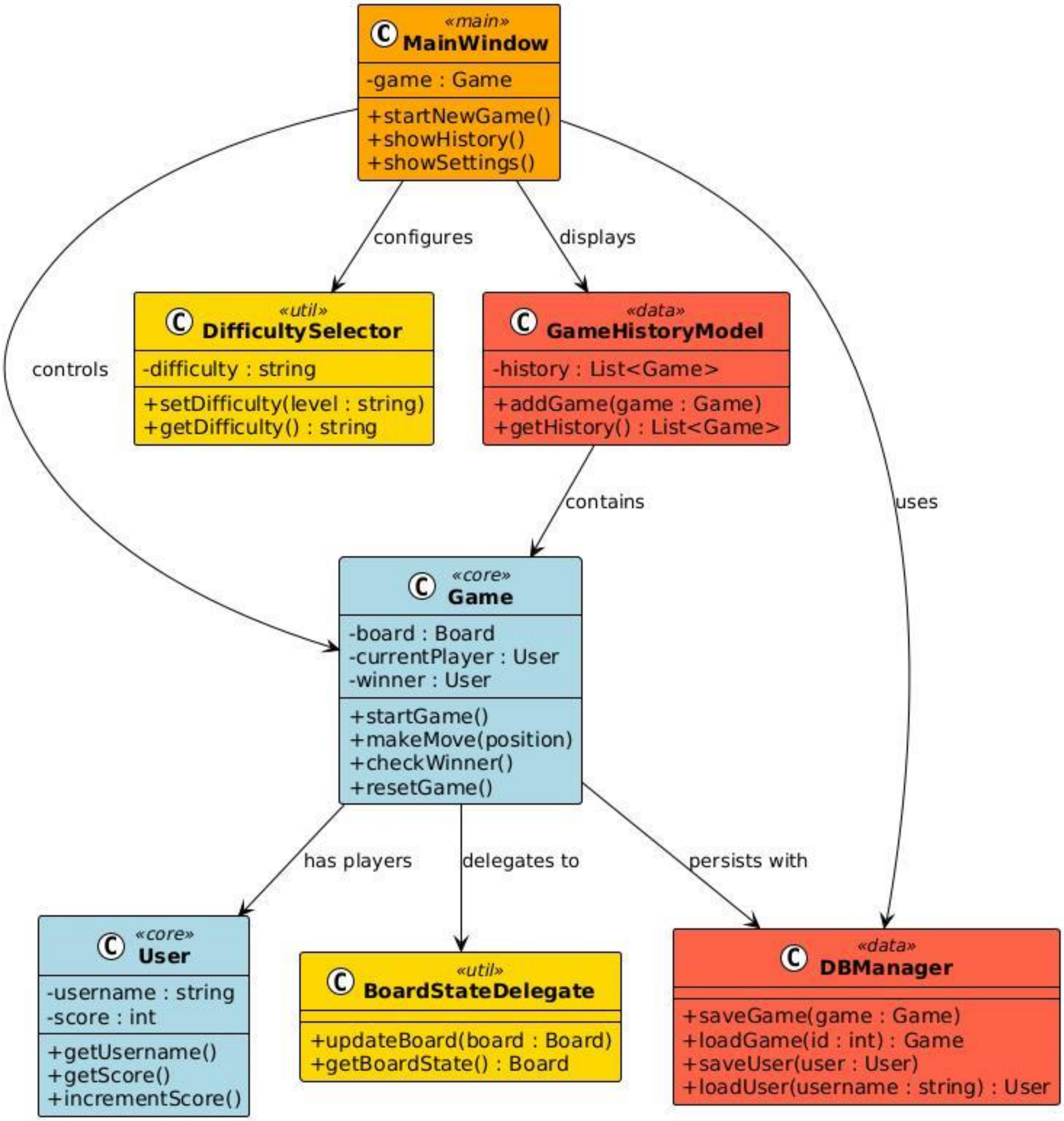
Game Flow


User
Main Window
Player Choice
Difficulty Dialog
Game Logic
Game History
Start Game
Wants to play
Ask for X or O
Returns choice
alt
[Play vs AI]
Ask for AI level
Returns level
Set AI level
Set mode to AI
[Play vs Human]
Set mode to Human
Show empty board
Player Move
Clicks a cell
Try to place mark
Move valid?
alt
[Valid]
Update board
Check for win or draw
Result
alt
[Win or Draw]
Show result
Save game result
End game
Switch turn
alt
[AI mode and AI's turn]
Start AI move
Show "Invalid move"
AI Move
alt
[AI mode and AI's turn]
Ask AI to play
AI move result
Update board
Check for win or draw
Result
alt
[Win or Draw]
Show result
Save game result
End game
Switch turn
Enable player move
User
Main Window
Player Choice
Difficulty Dialog
Game Logic
Game History

Data Management Flow



5.Class Diagram



6.AI Logic

Difficulty	**Logic**
Easy	Random available moves.
Medium	60% Random, 40% Minimax Move.
Hard	20% Random, 80% Minimax Move (and special corner handling).
Impossible	Pure Minimax (unbeatable).