Cairo University, Faculty of Engineering
Electronics and Electrical Communications
Department (EECE)

Title	Software Design Specification (SDS)
Project name	Tic Tac Toe game
Version	1.0
Team	Blaze
Date	27/6/2024

I .Purpose

The purpose of this document is to provide a detailed description of the software architecture and design for the Tic Tac Toe game. It includes UML diagrams such as class diagrams and sequence diagrams to ensure a well-structured implementation.

II .System overview

System Context:

The Tic Tac Toe game is a simple two-player game implemented as a software application. Players take turns marking spaces in a 3×3 grid, aiming to place three marks in a row, column, or diagonal to win the game.

System Architecture:

The system consists of a user interface, game logic, and data management components. These interact to provide the functionality required for playing and managing the Tic Tac Toe game.

III . Architectural Design

1. High level architecture:

The Tic Tac Toe game architecture includes the following components:

- GameBoard: Manages the state of the game board.
- GameLogic: Implements the game rules and win/draw conditions.
- Player: Represents the players in the game.
- GameController: Coordinates interactions between components.

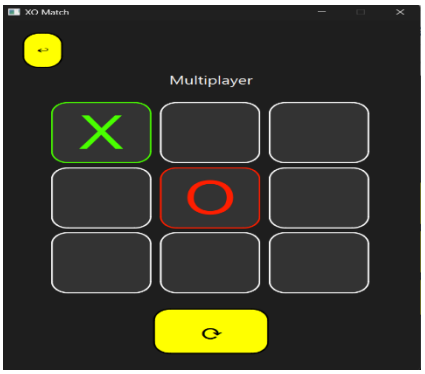
2. Modules and Components:

2.1 .Algorithms:

Minimax algorithm: The minimax algorithm in tic-tac-toe is a decision-making algorithm used to choose the optimal move by simulating all possible moves and their outcomes. It ensures that the maximizing player (X) selects the move with the maximum score, while the minimizing player (O) selects the move with the minimum score, leading to the best possible outcome for both players.

2.2 .Ui:

2.2.1 .Game screen:



Cell button: class inherits from `QPushButton` and represents a button in the Tic Tac Toe game grid. It is designed to interact with the main game screen, which is represented by the `GameScreen` class.

void setCurrentPlayer(int player),int getCurrentPlayer(),void onClicked():These slots are intended for managing the current player and handling button click events.

void cellClicked(int value): This signal can be emitted when a cell is clicked, passing an integer value to represent the event.

void setMode(const int &mode): Sets the game mode (e.g., player vs player, player vs AI).

void setUsername(QString &username): Sets the username of the current player.

enum class Player { X, O }:Enum to represent the current player (X or O).

char gameData[3][3]: 2D array to store the game state.

Database db: Database object for managing game data and user information.

void handleResetButtonClicked():Handles the reset button click event to reset the game.

void handleReturnClicked():Handles the return button click event to return to the main menu.

void handleButtonClick(int row, int col, QPushButton *button): Handles the click event for a game grid cell.

bool checkDraw():Checks if the game has ended in a draw.

bool checkWinner():Checks if there is a winner in the game.

void disableGameButtons():Disables all game buttons, preventing further moves.

void enableGameButtons():Enables all game buttons, allowing moves.

void btnsToArray():Converts the state of the buttons to the game data array.

void makeMoveEasy():Makes a move for the AI in easy mode.

void makeMoveNormal():Makes a move for the AI in normal mode.

void makeMoveHard():Makes a move for the AI in hard mode using advanced algorithms.

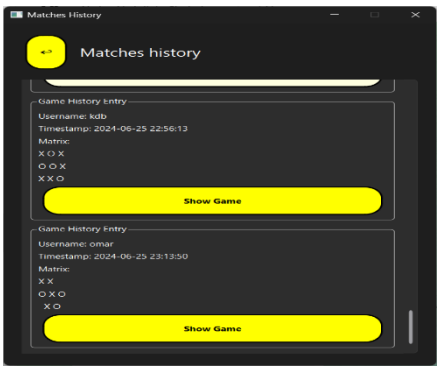
int minimax(int depth, bool isMax, int maxDepth): Implements the minimax algorithm for the AI to evaluate moves.

int evaluateBoard():Evaluates the current game board for the minimax algorithm.

bool isMovesLeft():Checks if there are any moves left on the game board.

void makeBestMove(int maxDepth): Determines the best move for the AI using the minimax algorithm.

2.2.2 .History screen:

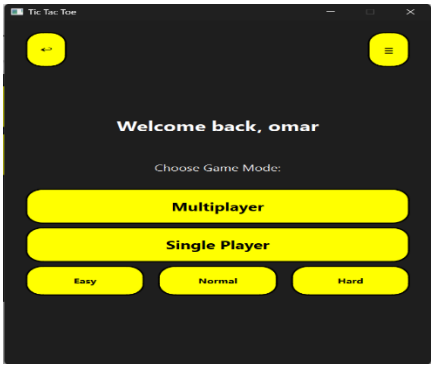


void handleGoBack():Handles the event to go back to the previous screen or main menu.

QString username: Stores the username of the current user to retrieve and display their game history.

Database db: Database object for managing and retrieving game history data.

2.2.3 .Main screen:



void setUserName(QString&username): Sets the username of the current user.

void handleLogout(): Handles logging out the current user.

void handleHistory(): Navigates to the game history screen.

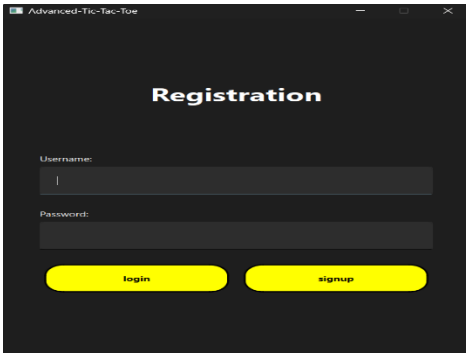
void handleMultiplayer(): Starts a multiplayer game.

void handleSingleplayerEasy(): Starts a single-player game in easy mode.

void handleSingleplayerNormal(): Starts a single-player game in normal mode.

void handleSingleplayerHard(): Starts a single-player game in hard mode.

2.2.4 .Sign screen:



*Public Slots:

signup_pressed: Handles the sign-up button press event.

login_pressed: Handles the login button press event.

*Private Members:

Layouts (main_layout, row1, row2, row3): Manage the layout of the screen components.

Labels (user_name, password): Labels for the input fields.

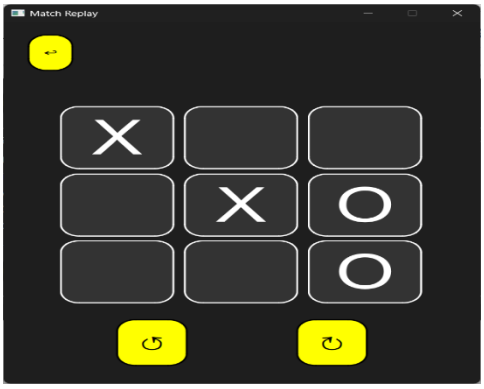
Line Edits (user_name_edit, password_edit): Input fields for user credentials.

Message Box (msgBox): Displays notifications and error messages.

Buttons (login, signup): Custom buttons for login and sign-up actions.

Database (db): Manages user authentication and registration.

2.2.5 .Reply screen:



ShowButtons: Displays the buttons on the screen.

redo: Redoes the last undone move.

undo: Undoes the last move.

Player: An enumeration for the players (X and O).

cellButtons: A 3x3 array of `CellButtonR` pointers representing the game board buttons.

gameData: A 3x3 array representing the game data.

gameEnded: A boolean indicating if the game has ended.

mode: The mode of the game.

username: The username of the player.

modeLabel: A label to display the mode.

entry: The game history entry.

movIndex: The current move index.

gameId: The game ID.

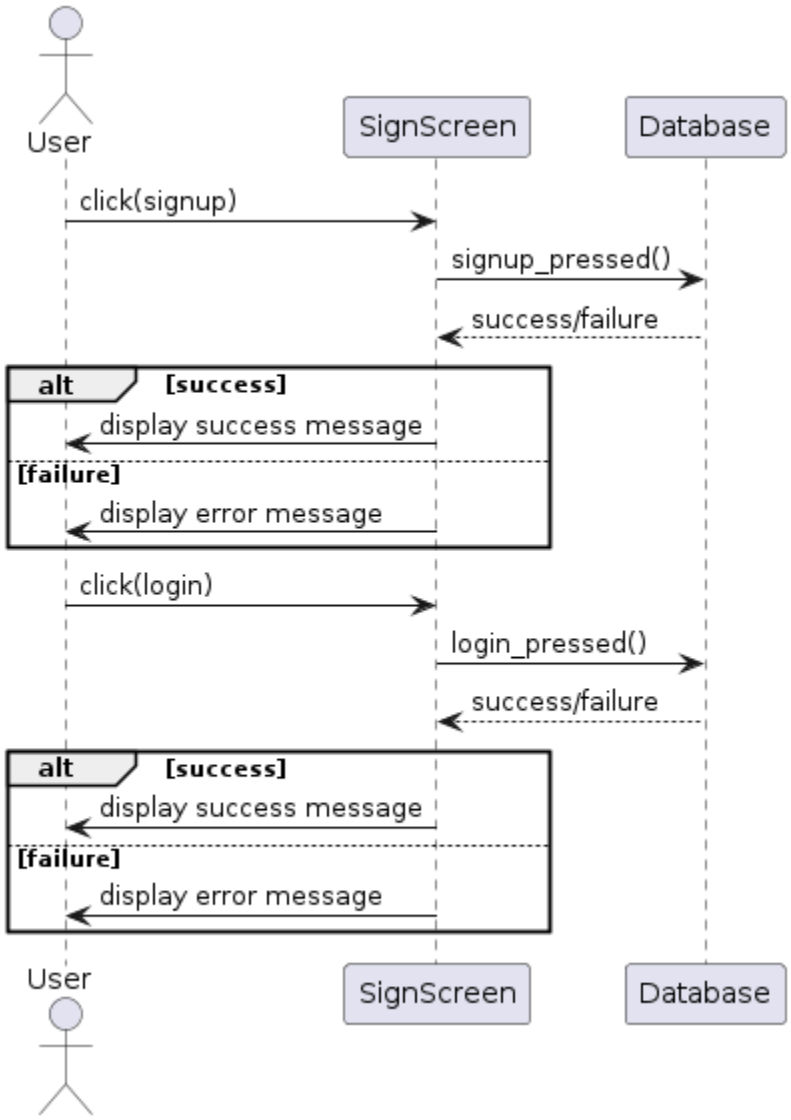
void handleBackClicked(): Handles the back button click event.

void UpdateButtons(): Updates the state of the buttons.

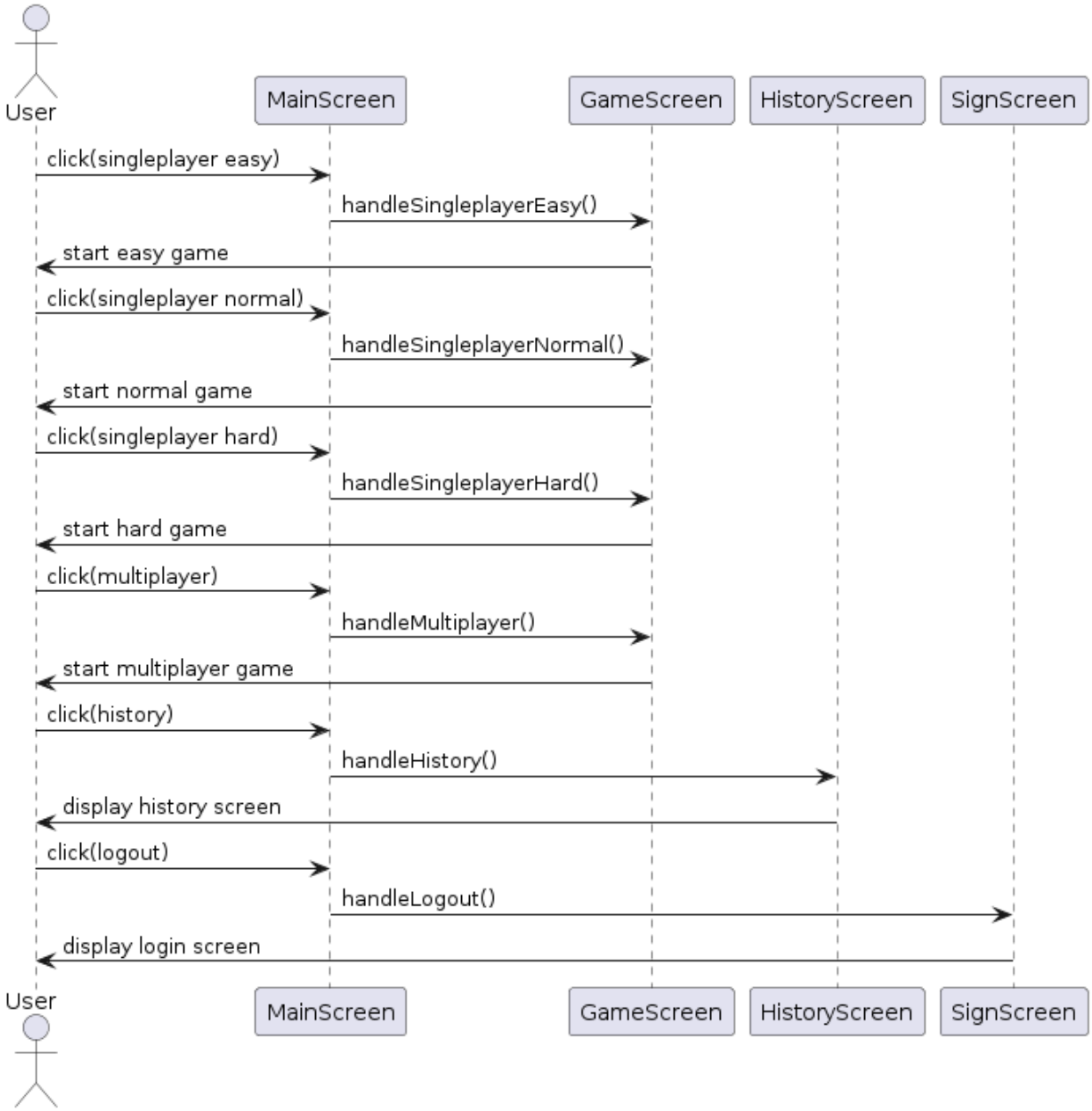
IV .Detailed design

1.Sequence diagrams:

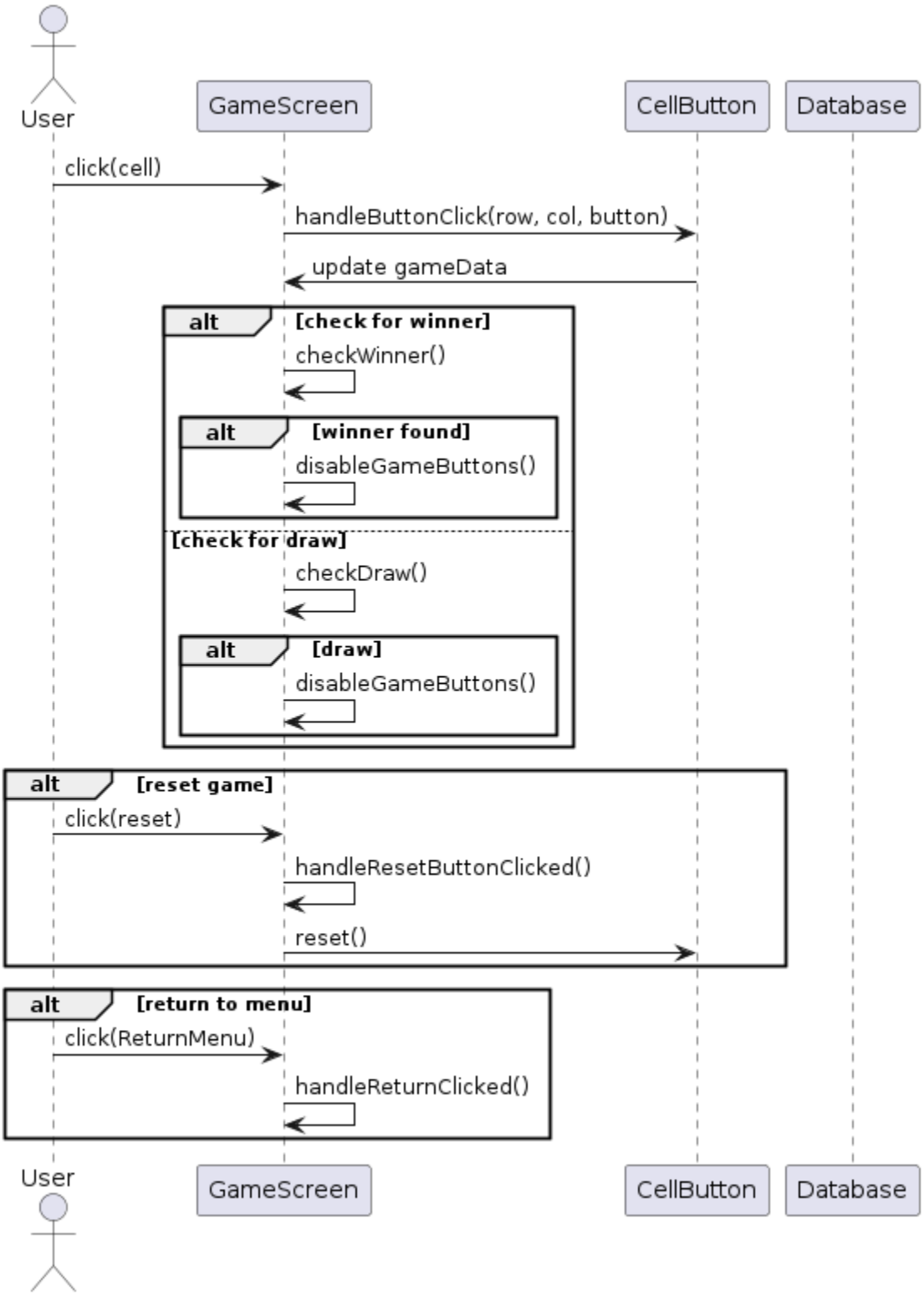
1.1 .Sign screen



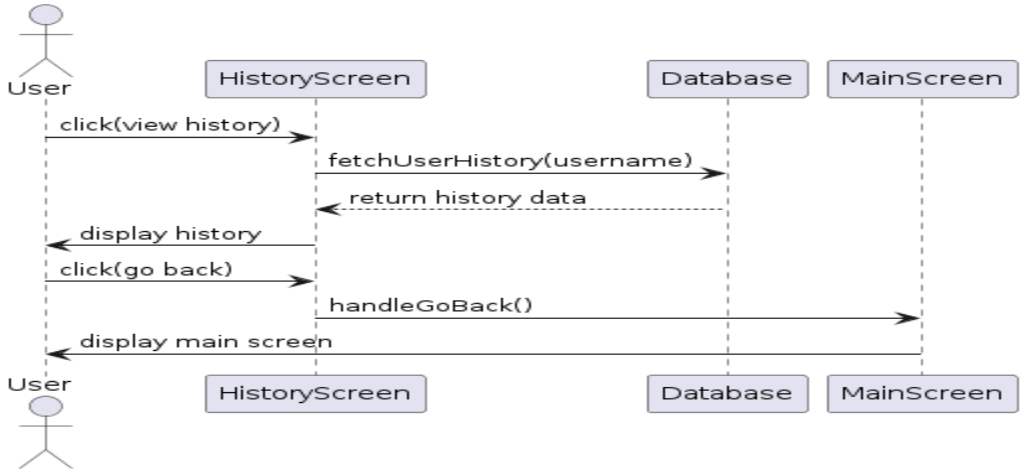
1.2 .Main screen



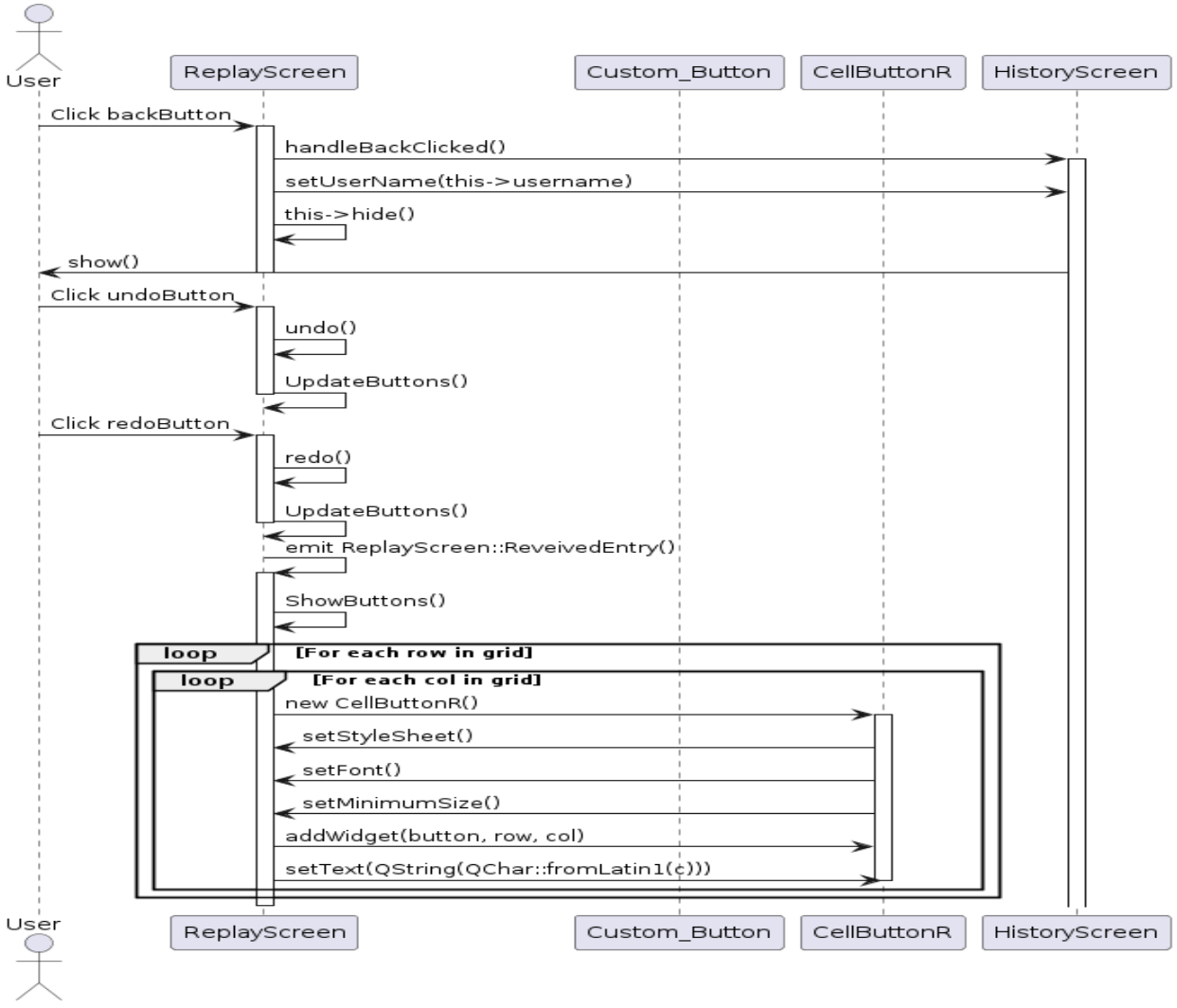
1.3 .Game screen



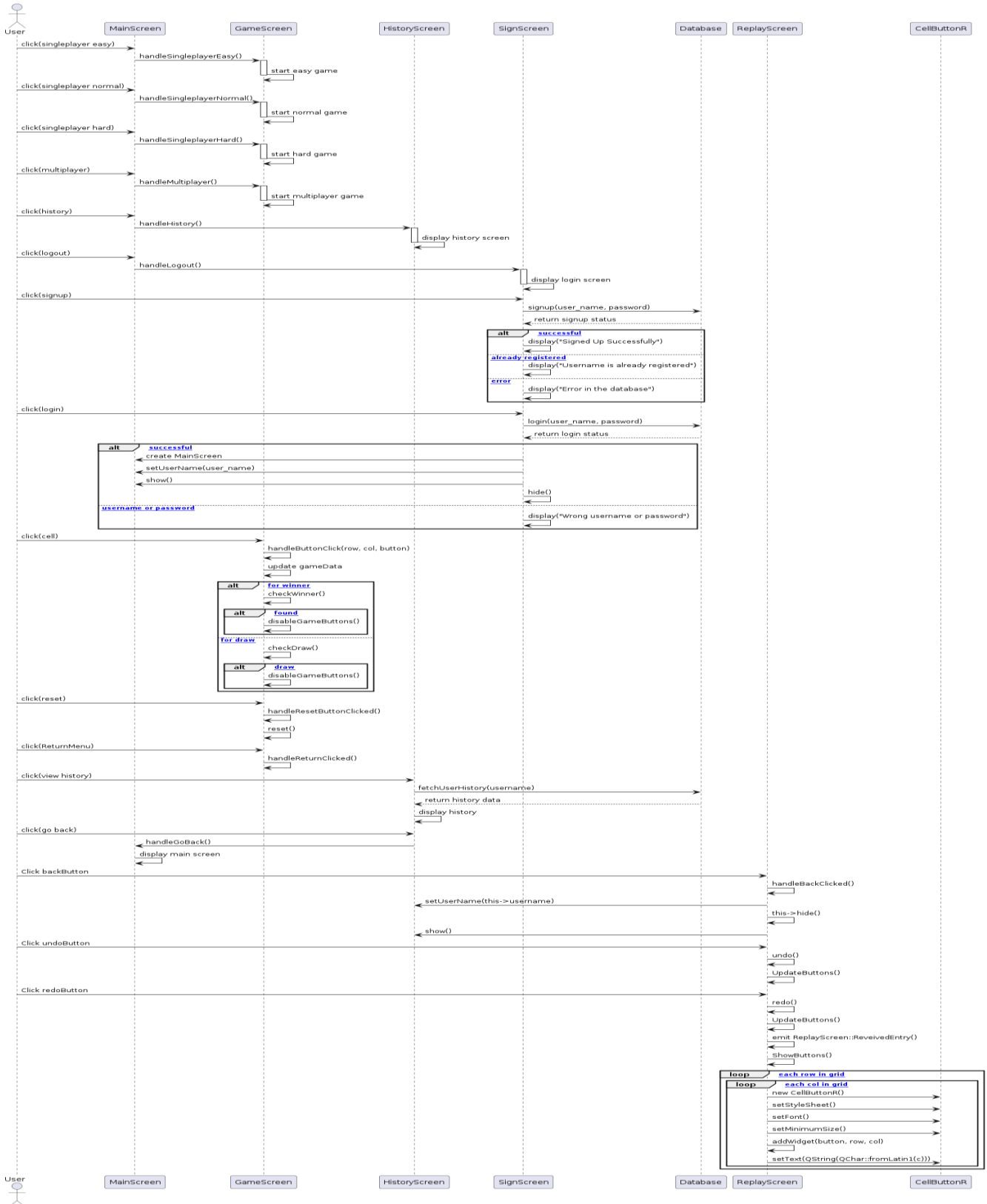
1.4 .History screen



1.5 .Reply screen



1.6 .Application



2 .Class diagrams: