Faculty of Engineering

Department of Electronics and Electrical Communications

ELC2080 Project

Tic Tac Toe game

Software Design Specifications (SDS)

Under the supervision of:

Dr. Omar Ahmed Nasr

Contents

1. Introduction

1.1 Purpose

This document defines the software design for the Tic Tac Toe game. It expands upon the requirements outlined in the Software Requirements Specification (SRS) and serves as a reference for developers and maintainers.

1.2 Background

Traditionally, Tic Tac Toe has been played on paper, but the increasing demand for digital and easily accessible entertainment highlights the need for a modernized version. This project addresses that demand by delivering a contemporary, feature-rich application that transforms the classic game into an engaging digital experience.

1.3 Target audience

The Tic Tac Toe desktop application is designed to appeal to a wide and diverse audience, including users of different age groups and technical proficiencies:

- **Families:** the game's simplicity and intuitive interface make it ideal for family-friendly usage, including children.
- **Friends**: it creates an environment full of cheer and happiness while playing. So, it is good to be a fun game that is played with friends.
- **Strategic thinkers**: players interested in challenging the AI to test and improve their tactical decision-making skills.

By catering to this diverse user base, the game aims to deliver an enjoyable experience that balances accessibility with strategic depth.

1.4 Scope

The Tic Tac Toe Game is a C++ application with a graphical user interface (GUI) that allows users to play either against another human or an AI opponent. It includes features such as user authentication, session management, personalized game history, and replay capability. The AI has different levels of difficulty. The system will be tested using Google Test and development will follow version control and CI/CD practices using GitHub and GitHub Actions.

1.5 Definitions, Acronyms, and Abbreviations

- **AI:** Artificial Intelligence.
- **GUI:** Graphical User Interface.
- **CI/CD:** Continuous Integration and Continuous Deployment.
- **SRS:** Software Requirements Specification.
- **Qt:** Cross-platform application development framework for GUI.
- **SQL:** Database engine.
- **UX:** User Experience.
- **UI:** User Interface.
- **OS:** Operating System.
- **UML:** Unified Modeling Language
- **IDE:** Integrated Development Environment,

1.6 Overview

This project aims to make the user experiences the enduring strategy and challenge of Tic Tac Toe in a modern and user-friendly format. This application offers a captivating and convenient way to test your skills, challenge others or AI opponent and experience the enduring fun of Tic Tac Toe.

The rest of the document details a comprehensive blueprint for the design and structure of the Tic Tac Toe game. It defines the system's architecture, core modules, data structures, and interfaces, based on the requirements established in the SRS.

2. System explanation

2.1 System overview

The advanced Tic-Tac-Toe game elevates the traditional version by incorporating modern enhancements. The system is composed of several interrelated components:

- Game logic
- Game mode (player vs player or player vs AI)
- GUI
- User authentication and management
- Personalized game history
- Testing and CI/CD integration

These components interact with each other to provide a seamless gaming experience.

2.2 System description

This Tic Tac Toe game is an interactive desktop application where users can play traditional Tic Tac Toe matches either as registered users or as guests. Registered users benefit from personalized features such as game history tracking and replay options. Upon launching the application, users can choose between two gameplay modes: player vs. player or player vs. AI. The AI opponent offers multiple levels of difficulty to provide a scalable challenge. Every completed match is automatically recorded, including the outcome and move sequence, allowing registered users to revisit and analyze their past games. This comprehensive system ensures both casual play and competitive engagement through intelligent game management and persistent data storage. The system is composed of several interrelated components:

- Model module: it contains the implementation of the game model, which manages the game state.
- Human module: it contains the implementation for the human player.
- AI module: it contains the implementation of the AI player for the Tic Tac Toe game with different difficulty algorithms
- Controller module: it implements the game logic and handles user interaction.
- Main module: this is the entry point of the application.

2.3 Constraints

- **The game is implemented in C++**

  C++ is used for performance and object-oriented design as it is a powerful and versatile programming language, will be the foundation for building the game's logic and functionality. This choice offers precise control over the game's performance.

- **The GUI is developed using the Qt framework**

  Qt provides a cross-platform framework for building responsive and modern user interfaces. It also provides a wide range of pre-built components and tools, streamlining the development process for creating menus, buttons, and other visual elements within the game.

- **User data stored using SQLite**

  SQLite comes into play here as the chosen method for data storage as it is a lightweight database management system, essentially a software tool that organizes and secures information. Also, it is used to store the user’s credentials by secure cryptographic hash functions to protect against unauthorized access and data breaches.

- **CI/CD practices integrated using GitHub Actions**

  CI/CD, which stands for continuous integration and continuous delivery, refers to a set of practices that streamline the process of building, testing, and deploying the game. It specifies using GitHub Actions, a built-in automation feature within the popular code-sharing platform GitHub.

2.4 Assumptions

- **Users are familiar with basic Tic Tac Toe game rules**

  As the won't provide tutorials for the game, it is assumed that users understand the fundamental rules of Tic Tac Toe, including the concept of alternating turns, winning by forming a line "horizontal, vertical or diagonal", and the possibility of a draw.

- **Users interact using a mouse or similar pointing device**

  The application is designed primarily for use with a desktop or laptop computer equipped with a mouse, trackpad, or equivalent input device. It assumes that the user can interact with graphical UI elements by clicking or selecting options on the screen.

2.5 Dependencies

- **Operating system compatibility**

  The game requires a compatible desktop operating system such as Windows or Linux "mainly windows". It relies on OS-level support for GUI libraries, threading, and local file or database access.

- **Display environment**

  The application is designed to run on standard desktop/laptop display environments. Mobile devices, tablets, or small embedded systems are not supported. A minimum screen resolution is assumed for proper GUI rendering.

- **Language support:**

  The application will be developed in English. It is expected that the user can read and interact with English-language menus, messages, and labels and the desktop supports English language.

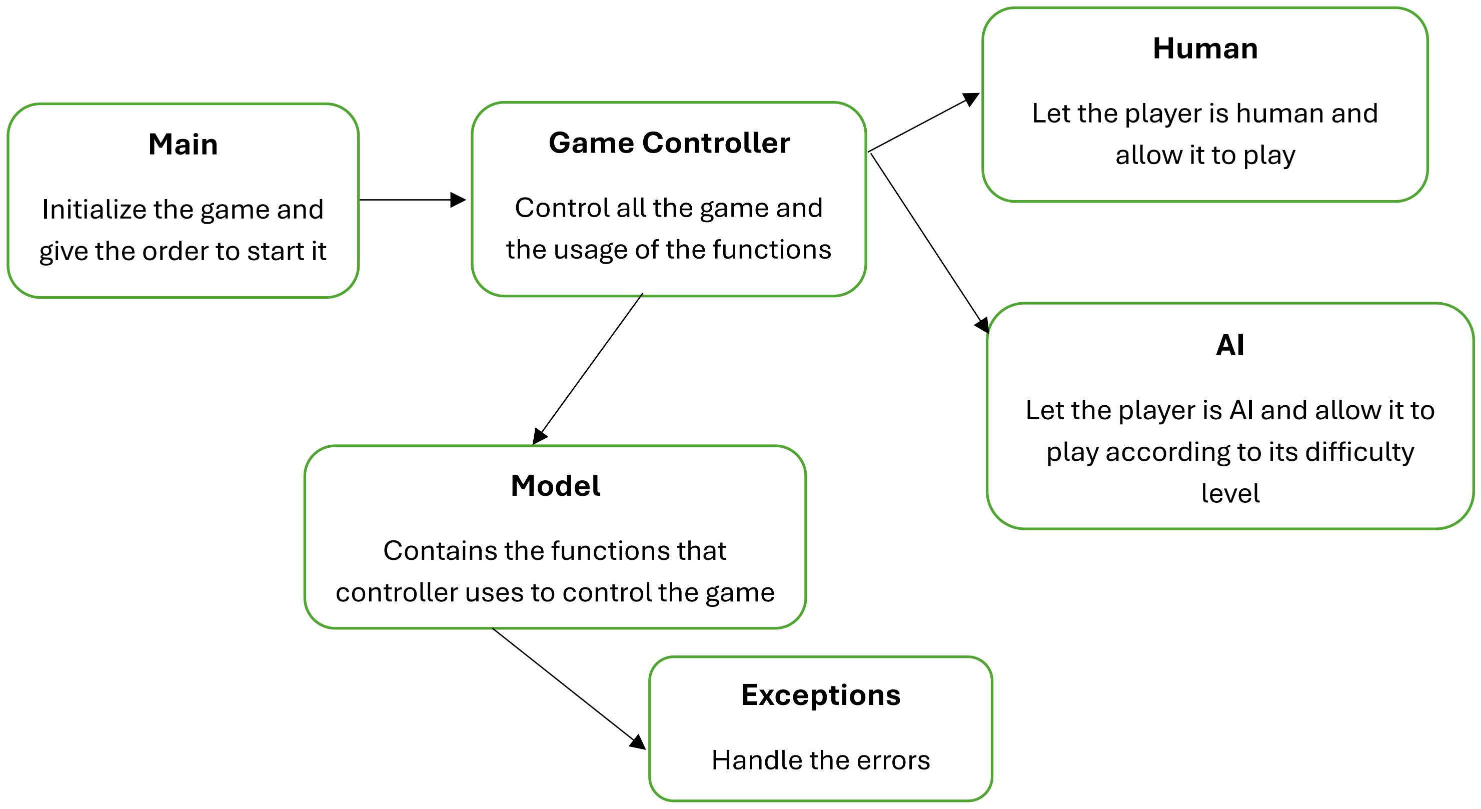
- **Qt framework availability**

  The Qt framework is essential for rendering the GUI and handling user interactions. The system requires Qt runtime libraries to be available on the host machine. These libraries support features such as windows, buttons, text inputs, and game board visualization.

3. Architectural Design

3.1 System Architecture Diagram

We tried to follow the MVC design pattern "Model View Controller". This structure separates the data logic, user interface and user actions. So, the architecture would be like:



Main
Initialize the game and give the order to start it
Game Controller
Control all the game and the usage of the functions
Human
Let the player is human and allow it to play
AI
Let the player is AI and allow it to play according to its difficulty level
Model
Contains the functions that controller uses to control the game
Exceptions
Handle the errors

3.2 Architecture explanation

- **Game Controller**

  The Controller acts as the central brain of the application. It handles the game flow, processes user input (from both human and AI players), and orchestrates interactions between different parts of the system.

- **Model**

  The Model represents the core game state and logic. It is independent of the user interface and contains all the data and rules of the Tic Tac Toe game.

- **Human Player**

  The Human Player module encapsulates the logic for a human participant in the game.

- **AI Player**

  The AI Player module implements the artificial intelligence opponent. It is designed to make strategic moves based on different difficulty levels.

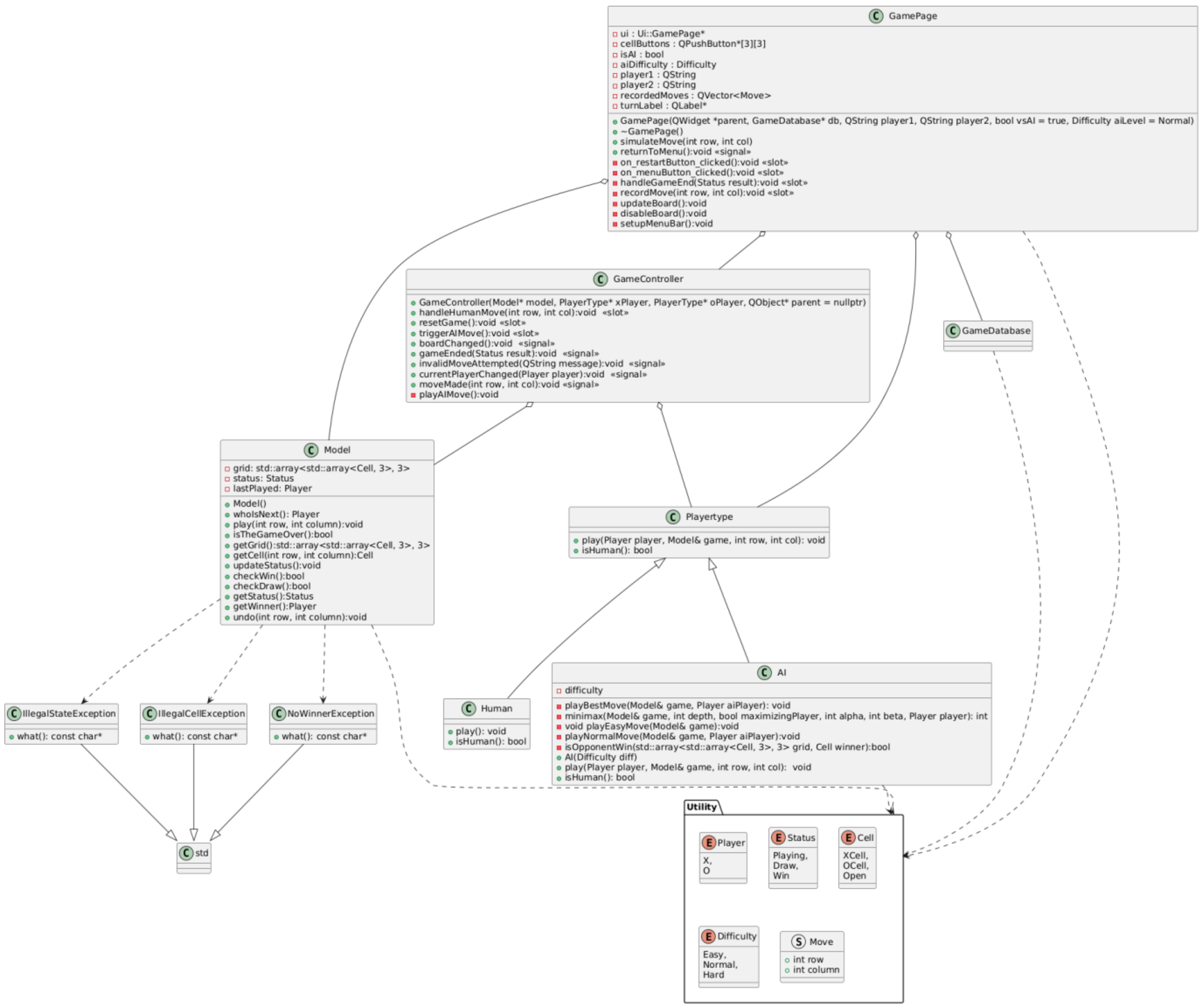
- **Exceptions**

  The Exceptions module defines custom exception classes to handle specific error conditions that can occur during gameplay

- **Main**

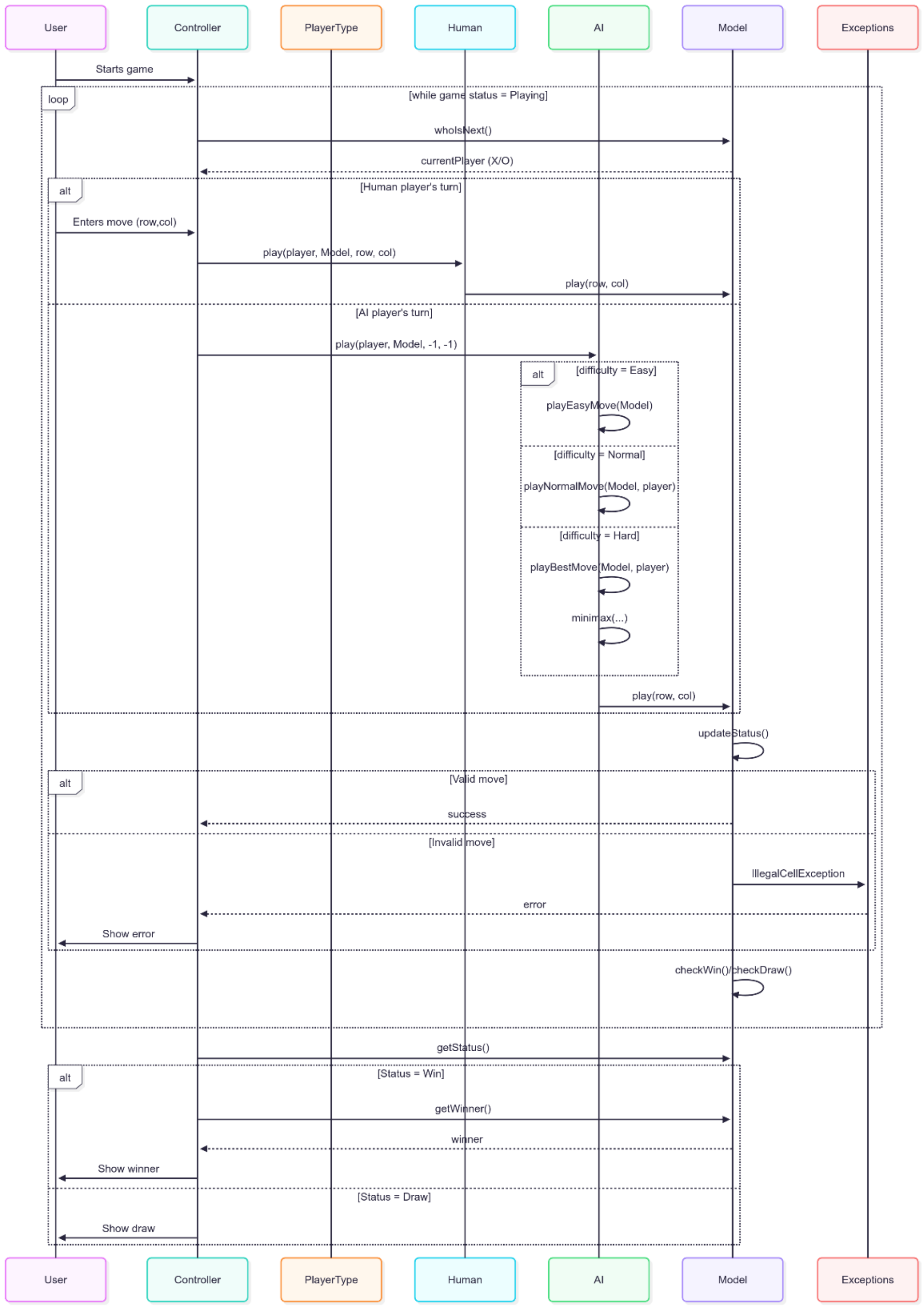
  The Main module serves as the entry point of the entire application. It is responsible for setting up the initial game environment.

3.3 Class diagram

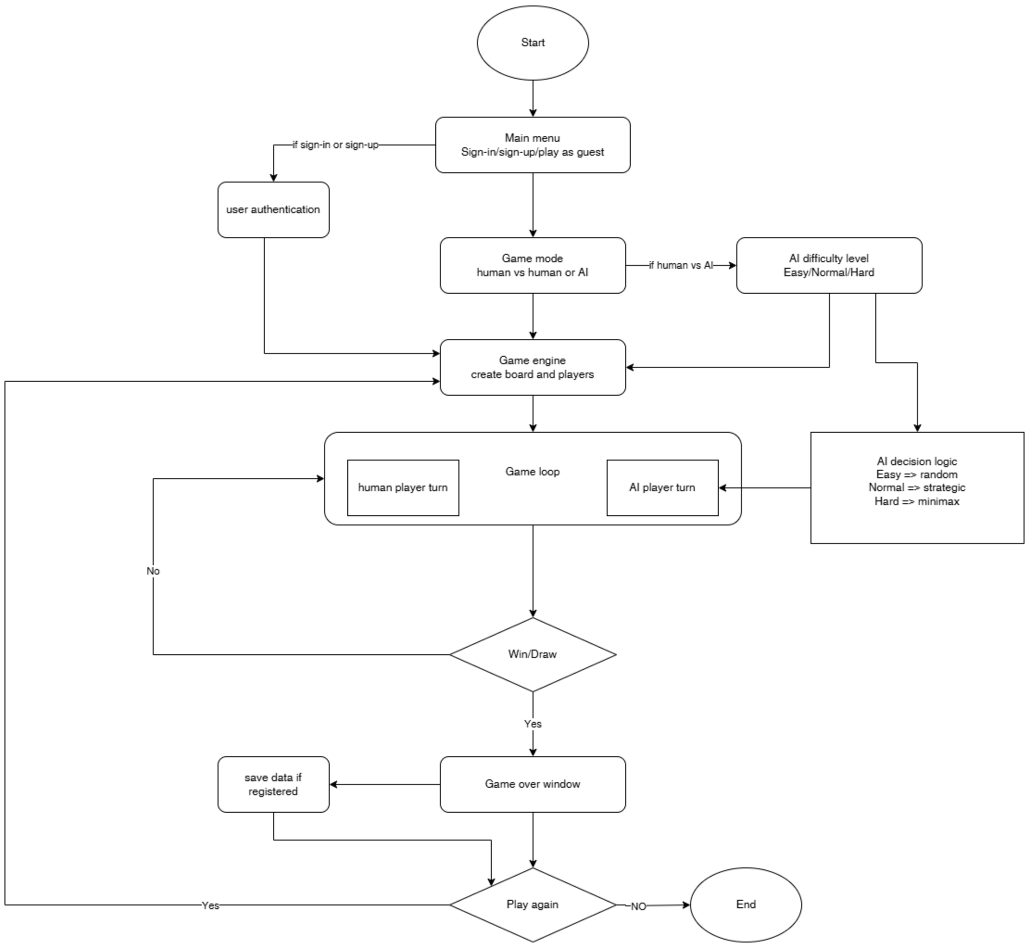

GamePage
ui : Ui::GamePage*
cellButtons : QPushButton*[3][3]
isAI : bool
aiDifficulty : Difficulty
player1 : QString
player2 : QString
recordedMoves : QVector<Move>
turnLabel : QLabel*
GamePage(QWidget *parent, GameDatabase* db, QString player1, QString player2, bool vsAI = true, Difficulty aiLevel = Normal)
~GamePage()
simulateMove(int row, int col)
returnToMenu():void «signal»
on_restartButton_clicked():void «slot»
on_menuButton_clicked():void «slot»
handleGameEnd(Status result):void «slot»
recordMove(int row, int col):void «slot»
updateBoard():void
disableBoard():void
setupMenuBar():void
GameController
GameController(Model* model, PlayerType* xPlayer, PlayerType* oPlayer, QObject* parent = nullptr)
handleHumanMove(int row, int col):void «slot»
resetGame():void «slot»
triggerAIMove():void «slot»
boardChanged():void «signal»
gameEnded(Status result):void «signal»
invalidMoveAttempted(QString message):void «signal»
currentPlayerChanged(Player player):void «signal»
moveMade(int row, int col):void «signal»
playAIMove():void
GameDatabase
Model
grid: std::array<std::array<Cell, 3>, 3>
status: Status
lastPlayed: Player
Model()
whoIsNext(): Player
play(int row, int column):void
isTheGameOver():bool
getGrid():std::array<std::array<Cell, 3>, 3>
getCell(int row, int column):Cell
updateStatus():void
checkWin():bool
checkDraw():bool
getStatus():Status
getWinner():Player
undo(int row, int column):void
Playertype
play(Player player, Model& game, int row, int col): void
isHuman(): bool
AI
difficulty
playBestMove(Model& game, Player aiPlayer): void
minimax(Model& game, int depth, bool maximizingPlayer, int alpha, int beta, Player player): int
void playEasyMove(Model& game):void
playNormalMove(Model& game, Player aiPlayer):void
isOpponentWin(std::array<std::array<Cell, 3>, 3> grid, Cell winner):bool
AI(Difficulty diff)
play(Player player, Model& game, int row, int col): void
isHuman(): bool
Human
play(): void
isHuman(): bool
IllegalStateException
what(): const char*
IllegalCellException
what(): const char*
NoWinnerException
what(): const char*
std
Utility
Player
X,
O
Status
Playing,
Draw,
Win
Cell
XCell,
OCell,
Open
Difficulty
Easy,
Normal,
Hard
Move
int row
int column

3.4 Sequence diagram

The sequence diagram demonstrates a typical user session, starting from launching the application, signing in, selecting game settings, and playing a match.



3.5 Simple game play flowchart



3.6 Data structure

- **Tree:** for AI decision-making via Minimax algorithm in hard level.
- **Struct:** for saving the moves on the grid.
- **Pairs:** to save the salt and hash password in the database.
- **Vectors:** used in the database and it behaves as a queue to save moves of a certain game.
- **Classes:** to determine the play mode human vs human or human vs AI.
- **Enums:** for representing the state of each cell or the game status for example, PLAYER_X and PLAYER_O instead of numeric values to improves code readability and reduces logic errors.

4. Conclusion

This Software Design Specification (SDS) document has provided a comprehensive blueprint for the development of the Tic Tac Toe game. It has detailed the system's architecture, core modules, data structures, and interfaces, building upon the requirements established in the Software Requirements Specification (SRS). By outlining the design choices for components such as the Controller, Model, Human, and AI modules, along with the handling of exceptions and player types, this document serves as a foundational guide for developers. The integration of C++ with the Qt framework for GUI, SQLite for data storage, and GitHub Actions for CI/CD practices underscores a commitment to a robust, modern, and maintainable application. Ultimately, this SDS ensures that the Tic Tac Toe game will deliver an engaging, accessible, and strategically rich digital experience for its diverse target audience.