**Hanoi University of Science and Technology  
School of Information and Communication Technology**



**PROJECT REPORT**

**Project name: Tic Tac Toe Game with Socket Programming**

**Week II**

***Subject: Network Programming***

**Instructor: Ph.D Truong Dieu Linh**

**Group 3:**

|  |  |  |
| --- | --- | --- |
| **Student ID** | **Student name** | **Class** |
| 20176732 | **Tran Thai Duong** | **ICT 03 – K62** |
| 20176845 | **Nguyen Manh Phuc** | **ICT 04 – K62** |
| 20184238 | **Nguyen Thi Minh Chau** | **ICT 01 – K63** |

**Hanoi, 2021**

**Table of Contents**

[**Table of Figures** 3](#_Toc72143814)

[**PART I: APPLICATION MODEL** 4](#_Toc72143815)

[**PART II: DATABASES & LIBRARIES** 5](#_Toc72143816)

[**PART III: STRUCTURE OF THE PROJECT - FUNCTIONS AND METHODS FOR PROCESSING** 6](#_Toc72143817)

[**1. On the client side** 6](#_Toc72143818)

[**2. On the server side** 8](#_Toc72143819)

[**3. Classes for protocols and messages** 9](#_Toc72143820)

[**4. Classes for managing players** 10](#_Toc72143821)

[**PART IV: CODE IMPLEMENTATION FOR NETWORK** 11](#_Toc72143822)

[**1. On the client side** 11](#_Toc72143823)

[**2. On the server side** 14](#_Toc72143824)

[**PART V: INTERGRATION OF NETWORK TO THE APPLICATION & DEMO** 18](#_Toc72143825)

[**1. Integration** 18](#_Toc72143826)

[**2. Demo of the program** 20](#_Toc72143827)

# **Table of Figures**

[Figure 1: Database diagram of the application 4](#_Toc72143735)

[Figure 2: External library configuration for the project 4](#_Toc72143736)

[Figure 3: Structure of the client 5](file:///E:\BKHN\Chous_Projects\networkprogramming_20202\project\report\Report_Team3_TicTacToe_Week2.docx#_Toc72143737)

[Figure 4: Structure of the server 7](#_Toc72143738)

[Figure 5: Classes for protocols and messages 8](#_Toc72143739)

[Figure 6: Classes for managing players 9](#_Toc72143740)

[Figure 7: Function for initialize and getting socket instance on the client side 10](#_Toc72143741)

[Figure 8: Methods for different functions on the client side 10](#_Toc72143742)

[Figure 9: Basic implementation for one function on the client side - Login function 10](#_Toc72143743)

[Figure 10: Implementation for send function on the client side 10](#_Toc72143744)

[Figure 11: Implementation for WriteCompletionHandler on the client side 11](#_Toc72143745)

[Figure 12: Implementation for ReadCompletionHandler on the client side 12](#_Toc72143746)

[Figure 13: Main function of the server 13](file:///E:\BKHN\Chous_Projects\networkprogramming_20202\project\report\Report_Team3_TicTacToe_Week2.docx#_Toc72143747)

[Figure 14: Methods for different functions on the server side 14](#_Toc72143748)

[Figure 15: Basic implementation of one function on the server side - Processing login request from client 14](#_Toc72143749)

[Figure 16: Implementation for send function on the server side 14](#_Toc72143750)

[Figure 18: Implementation of WriteCompletionHandler on the server side 15](#_Toc72143751)

[Figure 17: Implementation of ReadCompletionHandler on the server side 15](file:///E:\BKHN\Chous_Projects\networkprogramming_20202\project\report\Report_Team3_TicTacToe_Week2.docx#_Toc72143752)

[Figure 19: Call to HomeScreenHandler to display the home view and get request from user 17](#_Toc72143753)

[Figure 20: Handler for homescreen 17](#_Toc72143754)

[Figure 21: Controller of each screen is invoked to perform action - LoginFormController has login method called 18](#_Toc72143755)

[Figure 22: Implementation of LoginFormController 18](#_Toc72143756)

[Figure 23: Home screen of the application 19](file:///E:\BKHN\Chous_Projects\networkprogramming_20202\project\report\Report_Team3_TicTacToe_Week2.docx#_Toc72143757)

[Figure 24: Login screen 19](#_Toc72143758)

[Figure 25: Game mode screen 20](#_Toc72143759)

[Figure 26: Main game screen 20](#_Toc72143760)

# **PART I: APPLICATION MODEL**

The application uses **asynchronous socket** in order to create connection and communicate between clients and server. In this model, client or server makes the non-blocking call and start other background tasks instead of waiting for the request to be completed. When the operation is done, a completion callback is generated to finish the I/O operation.

Asynchronous socket communication brings several advantages:

- Allow multiple connection at the same time with good quality: while blocking (in blocking socket calls) prevents the program from completing other operation until the request is returned, and polling (in selecting) consumes computer memory and decrease performance, asynchronous socket allows background works to be continue when the I/O call is being processed, and only notifies when the call is completed.

- Save resources and avoid threading problems: Although creating threads for handling multiple connection at the same time also allow many clients to communicate with the server, this waste the memory since each thread need its own space for saving its metadata. Also, because threads use the same memory space of the program and separated threads may have operation on the same resource at the same time, deadlocks may raise. Asynchronous socket can be implemented on one thread only, which does not require additional memory, prevent mutexes and other synchronizing problems of multithreaded programs.

Clients and server communicate using **TCP protocol**. TCP has mechanisms for guaranteeing transmission of packets, so that the program can be eliminated from having to check if the message is successfully sent. The data stream is not complicated and there is no requirement as for streaming services; therefore, TCP is considered suitable.

Since there is the need of keeping track of each action performed on the client side, the communication is **stateful**, and information of the client is saved in a database on the server side for further references.

# **PART II: DATABASES & LIBRARIES**

In this program, we use SQLite to save database, with the diagram as shown

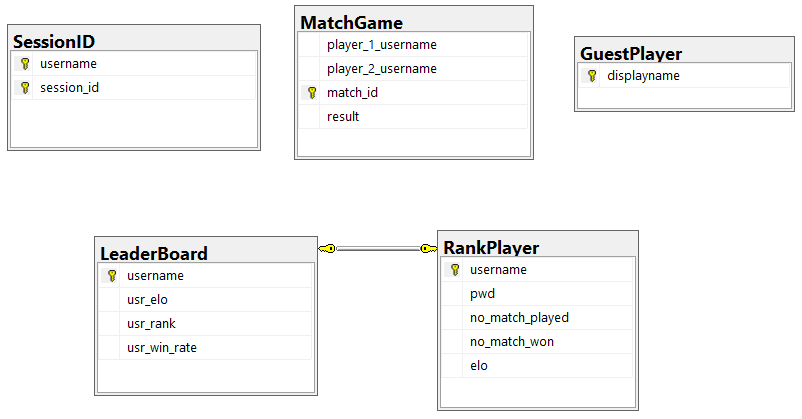


Figure 1: Database diagram of the application

In order to create message, library org.json is used. To connect to the database, the program uses SQLite extensions for Java.

For the project to be built properly, the 2 libraries must be included in the classpath of the project

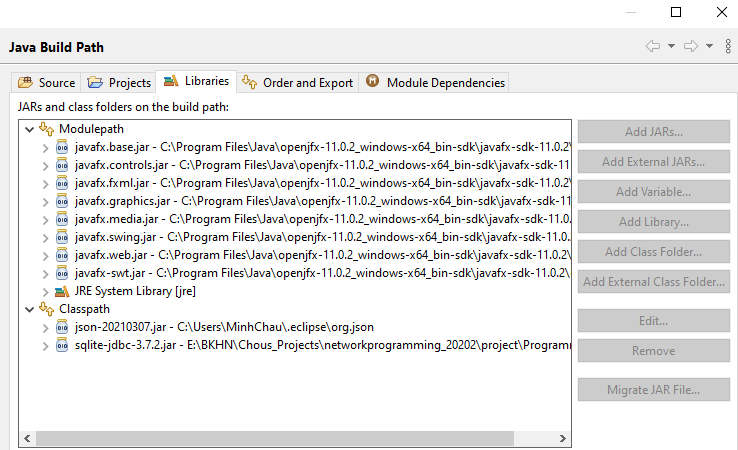
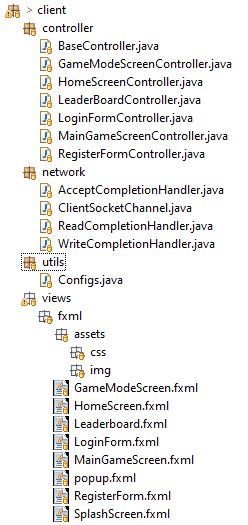
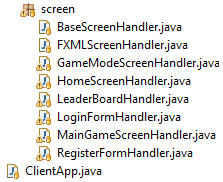


Figure 2: External library configuration for the project

# **PART III: STRUCTURE OF THE PROJECT - FUNCTIONS AND METHODS FOR PROCESSING**

## **1. On the client side**

Figure 3: Structure of the client



The implementation for the client consists of 2 parts:

- The displaying: packages **controller, views** and **screen**;

- The networking: packages **network**

For the displaying part, each view is built with a .fxml document of GUI, a handler for getting events and information from user inputs, and a controller for interacting with the network process.

The networking for each client consists of one asynchronous socket for opening connection and communicating with the server. Behaviors of the client’s outgoing and incoming messages are handled by different handlers corresponding to different classes, as shown in the figures above.

## **2. On the server side**

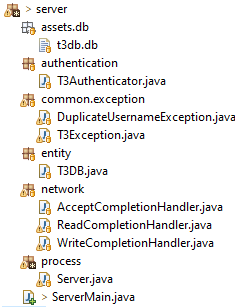


Figure 4: Structure of the server

On the server side, we do not implement the GUI but logging information on the console instead. There are 2 main packages that in charge of the main networking:

- The **network** package: has handlers for receiving and sending message

- The **process** package: for processing information received from the client and creating message for sending, correspondingly.

## **3. Classes for protocols and messages**

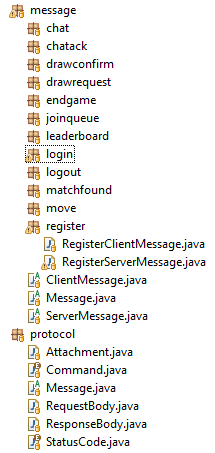


Figure 5: Classes for protocols and messages

For easily creating message for sending, the protocol package and message package are used

- **Protocol** package: has different formats of fields for supporting message creation

- **Message** package: has different constructors of different message types.

## **4. Classes for managing players**

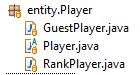


Figure 6: Classes for managing players

Entities considered in the programs are players. There are two types of players: **Guest** and **Rank** (players that have registered to have an account on the server). These classes are used to managing players connecting to the server

# **PART IV: CODE IMPLEMENTATION FOR NETWORK**

## **1. On the client side**

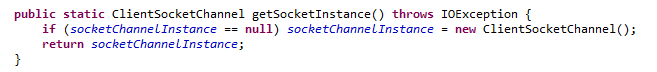


Figure 7: Function for initialize and getting socket instance on the client side

When the client is initiated, a socket is open and connect to the server via predefined IP address and port. The function getSocketInstance() is invoked in the first view of the program to connect to the server. It returns an instance of the socket for further use. Note that to prevent conflicts, there is only 1 socket initialized per each session.

From this returned instance, the program can call to different methods according to different functionality of the game, as shown in the figure

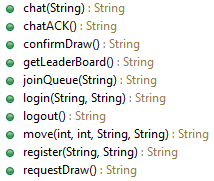


Figure 8: Methods for different functions on the client side

For each function, a request is created with desired parameters, and passed to the function sendRequest() to be sent to the server.

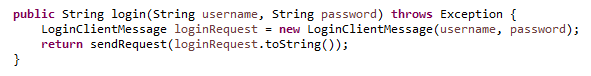


Figure 9: Basic implementation for one function on the client side - Login function

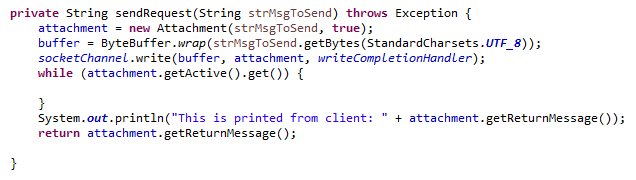


Figure 10: Implementation for send function on the client side

The sendRequest() function then create an attachment, which consists of the message and a Boolean variable for determining the completion of the action. After that, the message is wrapped into a buffer and sent. Using a WriteCompletionHandler with the attachment created, we return when the attachment send is marked as done. That is when the returned message is filled into the attachment and can be taken out for further processing.

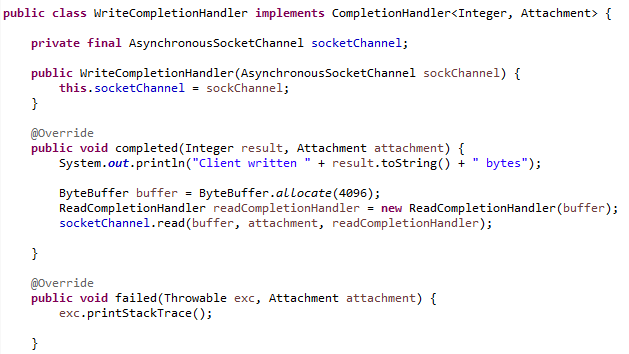


Figure 11: Implementation for WriteCompletionHandler on the client side

The WriteCompletionHandler of the client side wait for the response, using a ReadCompletionHandler. This read handler will put the received message in the attachment, mark the operation as done (by the .getActive().set() function; an inactive attachment is considered to be processed completely), then return it to the write handler, then the main caller.

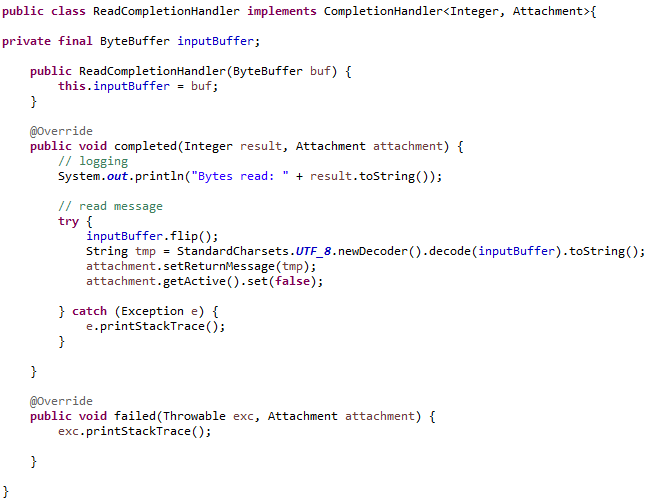
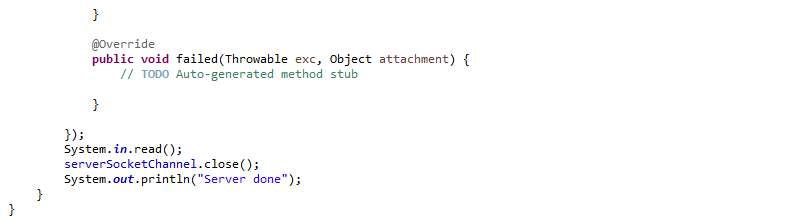


Figure 12: Implementation for ReadCompletionHandler on the client side

## **2. On the server side**

Figure 13: Main function of the server



The server consists of a socket that infinitely listening for connection from the clients. After getting a connection, it creates a ReadCompletionHandler for listening to message from the client. After each received message, it processes (using processReturn function), and returns a proper message for sending to the client via sendResponse() function.

All these operations work separately from the accept function of the main socket, on the same thread (ie. The main socket keeps waiting for new connection while other requests from connected clients are still processed), which is the significant characteristic of asynchronous socket.

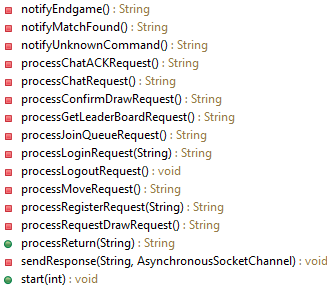


Figure 14: Methods for different functions on the server side

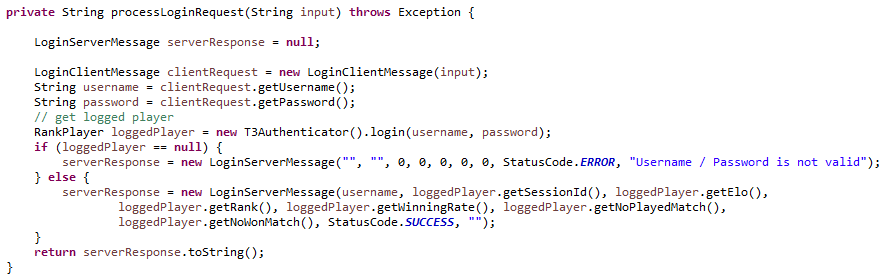


Figure 15: Basic implementation of one function on the server side - Processing login request from client

Each message processing function receive the input from the main caller, then create a corresponding message after having checking conditions and data. It returns a proper string for sending back to the client. The sendResponse() function peforms the same as the sending function of the client side.

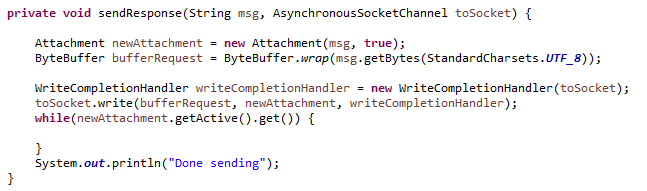


Figure 16: Implementation for send function on the server side

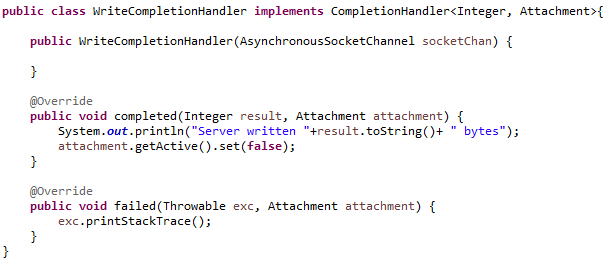
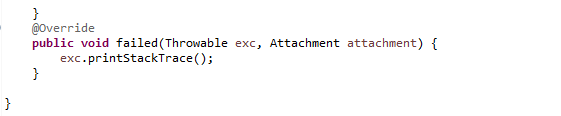
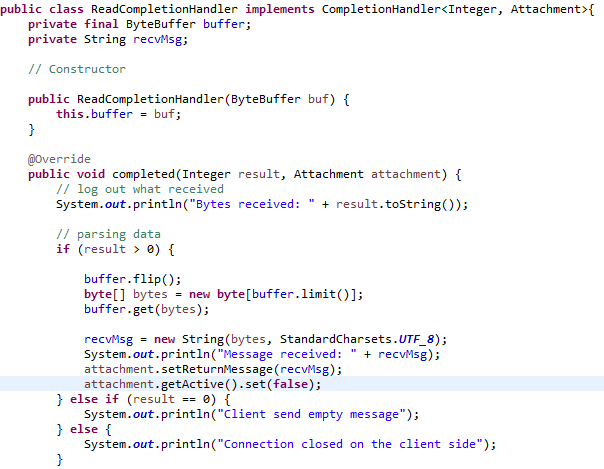


Figure 18: Implementation of WriteCompletionHandler on the server side

Figure 17: Implementation of ReadCompletionHandler on the server side



The handlers for operations on the server side is built similarly to which of the client side. The main difference is that the WriteCompletionHandler does not read right after the operation is done, since the message must be passed to the main controller for analyzing before sending.

# **PART V: INTERGRATION OF NETWORK TO THE APPLICATION & DEMO**

## **1. Integration**

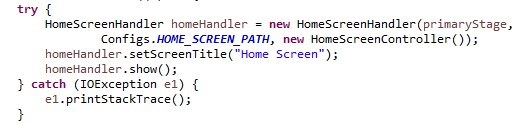


Figure 19: Call to HomeScreenHandler to display the home view and get request from user



Figure 20: Handler for homescreen

The first view of the program has a HomeScreenHandler for catching event from the user. The 3 actions that can be instantiated is *playing as a guest, as a returned user, and registering a new account*. Each of which will invoke another handler of the corresponding screen. Each handler has a controller for getting input from user, sending and receiving data from the server.

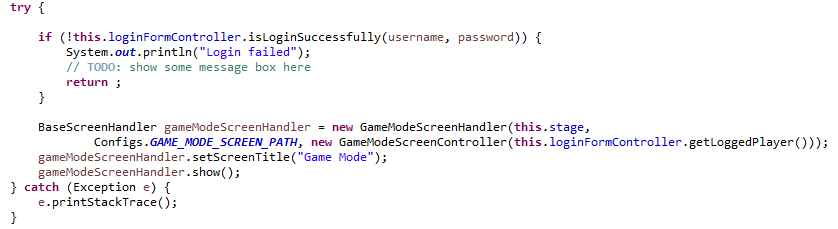


Figure 21: Controller of each screen is invoked to perform action - LoginFormController has login method called

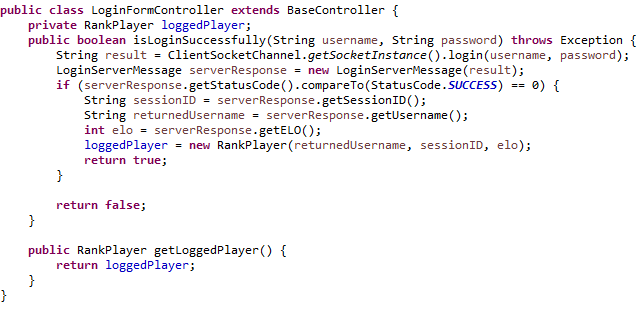


Figure 22: Implementation of LoginFormController

## **2. Demo of the program**

Figure : Home screen of the application

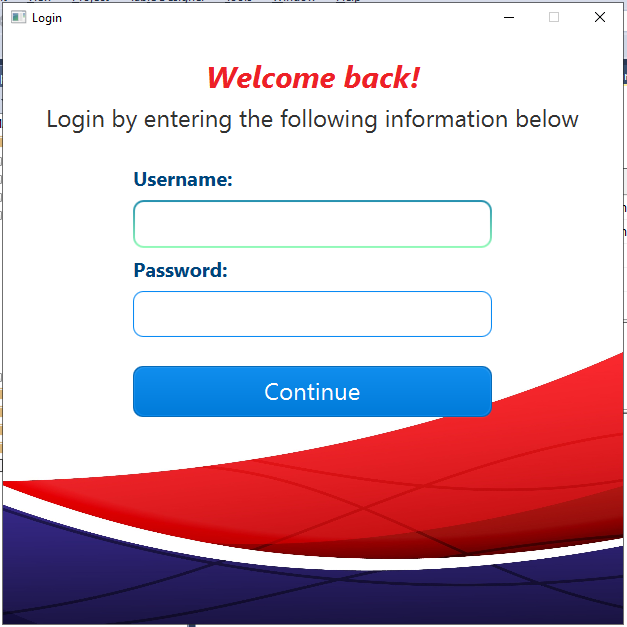


Figure : Login screen

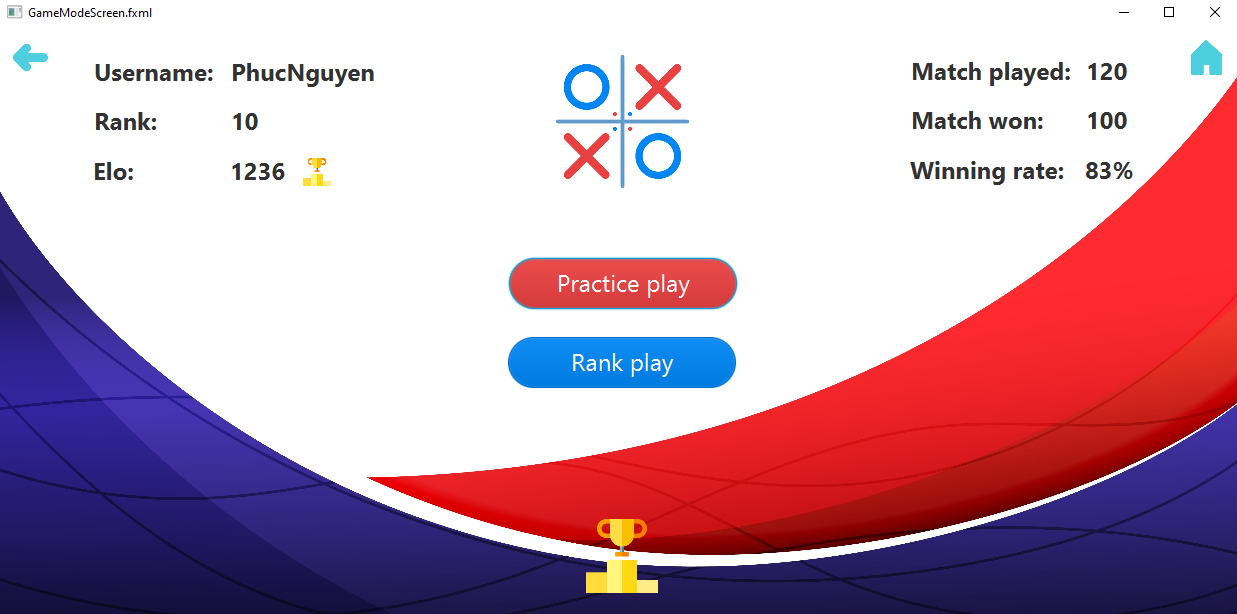


Figure : Game mode screen

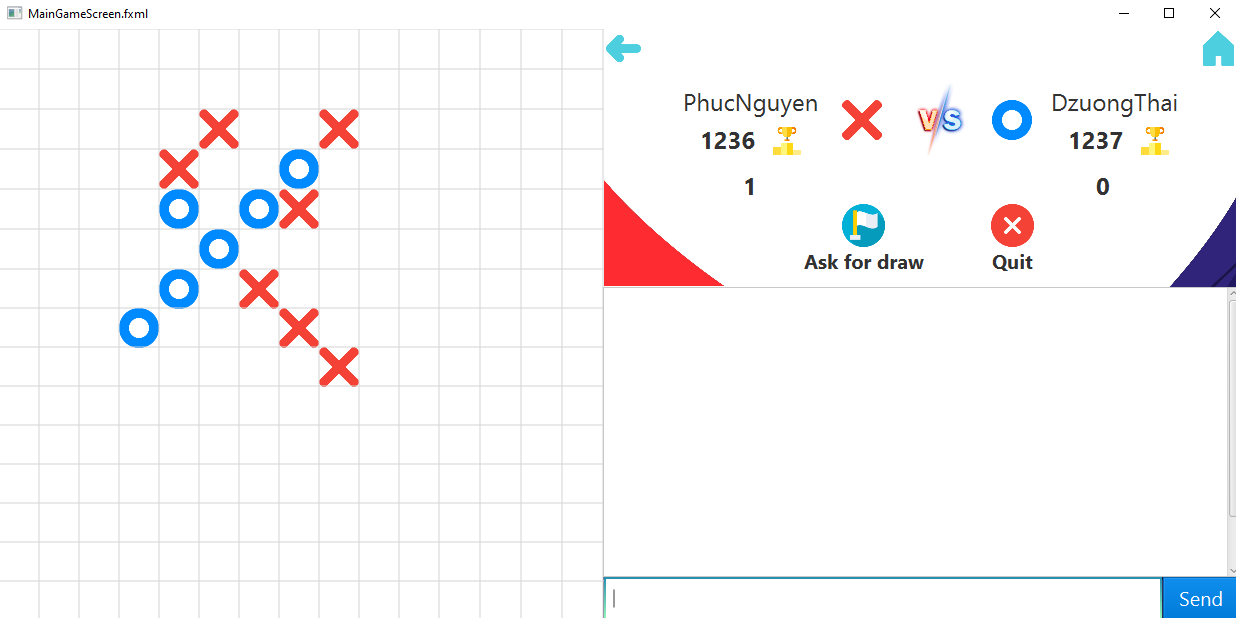


Figure : Main game screen