**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](#_Toc73398224)

[**PART I: MAIN GAME IMPLEMENTATION** 4](#_Toc73398225)

[**Flow of joining the game – Normal mode** 4](#_Toc73398226)

[**Queue** 7](#_Toc73398227)

[**In-game moves** 10](#_Toc73398228)

[**In-game logic** 12](#_Toc73398229)

[**PART II: LEADERBOARD** 14](#_Toc73398230)

# **Table of Figures**

[Figure 1: State diagram for JOIN\_QUEUE actions on the client side 4](#_Toc73457769)

[Figure 2: State diagram for JOIN\_QUEUE actions, monitoring queue and finding match on the server side 5](#_Toc73457770)

[Figure 3: State diagram for handling MOVE and LISTEN\_MOVE on the client side 6](#_Toc73457771)

[Figure 4: State diagram for handling MOVE and LISTEN\_MOVE on the server side 6](#_Toc73457772)

[Figure 5: Structure of a Match object that store information of the match 7](#_Toc73457773)

[Figure 6: QueueController class for monitoring queues 8](#_Toc73457774)

[Figure 7: The client waiting for a match, with all buttons disabled for preventing improper behaviors 8](#_Toc73457775)

[Figure 8: Structure of a Move object that stores information of players' moves in a match 12](#_Toc73457776)

[Figure 9: The leaderboard example 15](#_Toc73457777)

# **PART I: MAIN GAME IMPLEMENTATION**

## **Flow of joining the game – Normal mode**

In order to begin the game, player can choose to play as a guest, or login to join different queues: normal or ranked, as desired. The main differences between the normal and ranked queue is that players from the ranked queue can only be registered players. They will be matched based on another matching algorithm to guarantee a proper match is founded based on their ELO, and each ranked match will also affect their ELO. On the contrary, normal queue accepts both guests and registered users, and they also do not have to take care about their ELO before and after the match.

If player decides to have a guest play, he will be immediately sent to the normal queue. Otherwise, he will be taken to the proper queue as his choice at the Game Mode stage.

To join the queue, player will choose the wanted mode, and the client will create a corresponding message to notify the server about the join queue request. Player will be taken out from the hall, where they are after having logged in successfully, and put into the appropriate queue. The server will check to see if there is any potential player, then match them and send back a notification back to the client, which includes information about the new game, to start the match.

Diagram

Description automatically generated

Figure 1: State diagram for JOIN\_QUEUE actions on the client side

Diagram

Description automatically generated

Figure 2: State diagram for JOIN\_QUEUE actions, monitoring queue and finding match on the server side

During the game, users take turns to place an X or an O onto the gameboard. Each move will be recorded and checked on the client side for its state and result before sending to the server to be forwarded to both players. A move is considered to mark the game as final when it creates a consecutive chain of at least 5 Xs or 5 Os, or when the board is fully filled, but there is no proper line of 5 founded.

Diagram

Description automatically generated

Figure 3: State diagram for handling MOVE and LISTEN\_MOVE on the client side

Diagram

Description automatically generated

Figure 4: State diagram for handling MOVE and LISTEN\_MOVE on the server side

## **Queue**

Up to this moment, the application allows players to join the game by logging in and choose normal mode. The client will send a message to the server of form

{

“command\_code”: “JOIN\_QUEUE”,

“info”: {

“session\_id”: “ASKRGBJq308-6”,

“mode”: “normal”,

}

}

The server will base on this information to put user into the queue. The session\_id is used to check in the database for identifying logged user. In the future, this will also be the base for differentiate returning users and guests. The mode field will help the server to put player into the proper queue. There are 2 values of modes: “normal” and “ranked”.

When having successfully sent request to the server, the client will freeze all the buttons to prevents users from sending other queue joining request while waiting for reply. The server after having put player into the queue he demanded, has a controller for always checking if there is a proper opponent for player, then creates a match and pushes it to the list of matches. This controller is set to always running as a background process and is put in a separated thread for not affecting other socket processing and provide ease of managing.

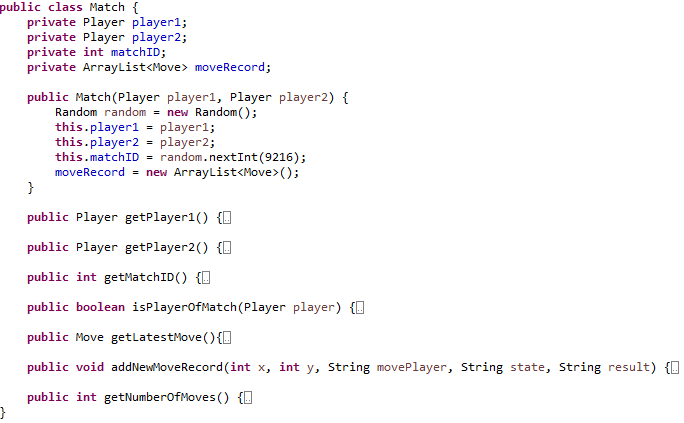


Figure 5: Structure of a Match object that store information of the match

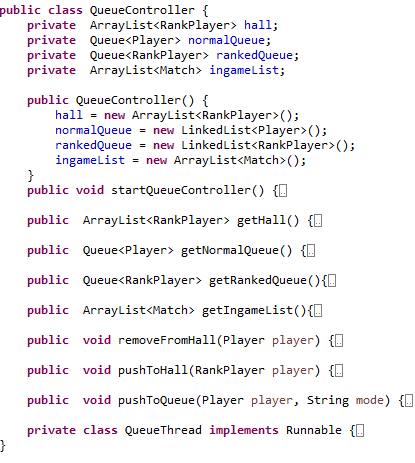


Figure 6: QueueController class for monitoring queues



Figure 7: The client waiting for a match, with all buttons disabled for preventing improper behaviors

We modified the protocol so that the client will receive the message with the same command code, corresponding to the previous sent message, to start the game.

{

“command\_code”: “JOIN\_QUEUE”,

“info”: {

“username”: “duong”,

“session\_id”: “ASKRGBJq308-6”,

“match\_id”: 1234,

“opponent”: “phuc”,

“opponent\_elo”: 1465,

“player\_1”: “duong”,

}

“status\_code”: “success”,

“error”: “”

}

}

The username field is the log in username of player corresponding to his session\_id saved in the database of server. In the future, when the program allows guests to join in the queue, the username field will be filled with a temporary name for displaying on the client, which is of the form “anon\_xxx”, where “xxx” is a random number. Also, session\_id will also be generated for the guest in order for tracking behaviors of him in the game.

The match\_id is automatically generated by the server when it founds a match for 2 players. Some information of the opponent is also included in the message to display on the client side. player\_1 field indicates player that will take an X and goes first in the match.

The status\_code indicates if the finding match process succeeded or failed (corresponding to “success” or “failed” message). The error may raise when the server takes too much time for finding a game, or there is no other player in the queue to match, etc. Detail of the error is stated in the error field.

## **In-game moves**

After having entered into a match, players will take turns to place move onto the boardgame. Each placement will trigger the client to create a message to send to the server, which will be forwarded to both clients.

{

“command\_code”: “MOVE”,

“info”: {

“match\_id”: 1234,

“session\_id”: “ASKRGBJq308-6”,

“move\_player”: “duong”,

“move\_position”: {

“x”: 6,

“y”: 9,

}

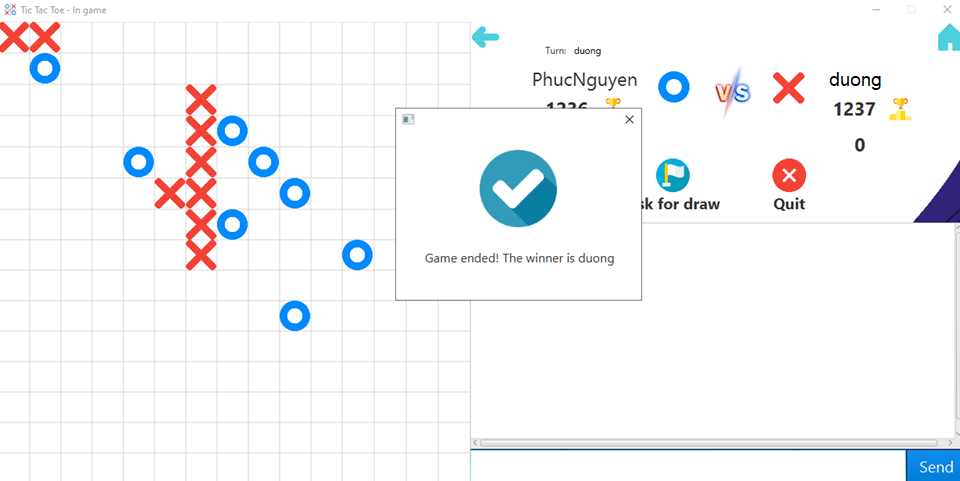
“state”: “valid”,

“result”: “win”

}

}

The match\_id and session\_id is for validating the user is sending information in the right match. The move\_position field indicates coordinates of the move that move\_player has sent. The state of the move can be valid or invalid, according to the game rule. Finally, the result field indicates if the move takes the game to the end. If the player placed a move that create a proper line, then value of “result” will be “win”. If the board is fully filled without any 5-line, the value of result is “draw”. Otherwise, this field is left blank.



## **In-game logic**

After each time the player place a move, the client will call a function to check if the placed move creates a chain of 5 consecutive Xs or Os. The idea of the function is to trace on its horizontal, vertical and diagonal lines and determine the above conditions. It also checks if the board is filled without any lines that satisfies the game’s rule.

If this function returns true, the message created will have the “result” field value of “win”. If the second case stated above occurs, the result field will be “draw”.

The message is then sent to the server and get into listen state. The server will parse these information, save it to a list of structure that contains information about the moves. The other player, who has previously sent the LISTEN\_MOVE message to the server to ask for new changes of the gameboard, will wait for its handler on the server side to check the list of moves return the requested information and display it on the screen.

{

“command\_code”: “LISTEN\_MOVE”,

“info”: {

“username”: “duong”,

“match\_id”: 1234,

}

}

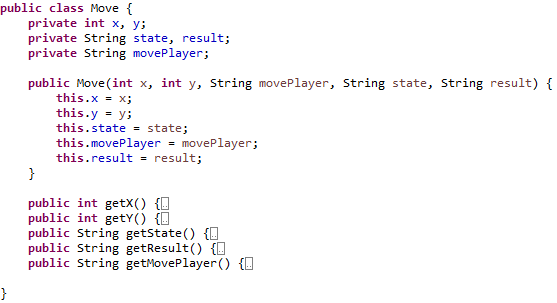


Figure 8: Structure of a Move object that stores information of players' moves in a match

The username field is for indicating who is requesting the information of the game with the match\_id. Both players will continuously switch between sending state and listen state for forwarding and retrieving data of the game.

After receiving the message containing information of the most recently placed move, client will first get the information of the valid field to check if it can be accepted on both sides. Then, it examines the result field to see if the move drives the game to the end. If it does, a notification will show up, the game board is frozen to prevent users from placing any further moves, then the clients will take players back to the Game mode screen. Otherwise, the information about coordinates of the moves is retrieved to display on the board.

On the server side, the message is continuously receiving and forwarding data until the game reach the end. Then it pops both players out of the list of currently in-game players and puts them back into the hall (in case of logged users), or take them back to the home screen (in case of guests)

# **PART II: LEADERBOARD**

The leaderboard shows the top 10 users that has the highest ELO in the game, with information of number of matches played, number of matches that they have won, as well as the win rate.

To get this information, client will send a require message to the server. Server will fetch the desired information in the database and sent it back to the client.

{

"command\_code": "LEADERBOARD",

"info": {

"username": ["duong", "haha", "kuuhaku", "chouchou", “phuc”],

"elo": [1500, 1490, 1480, 1470, 1460],

"rank": [1, 2, 3, 4, 5]

},

"status\_code": "success",

"error": ""

}

{

"command\_code": "LEADERBOARD"

"info": {

"session\_id": "ASKRGBJq308-6",

"username": "duong"

},

}



Figure : The leaderboard example