**Games4Gamers**

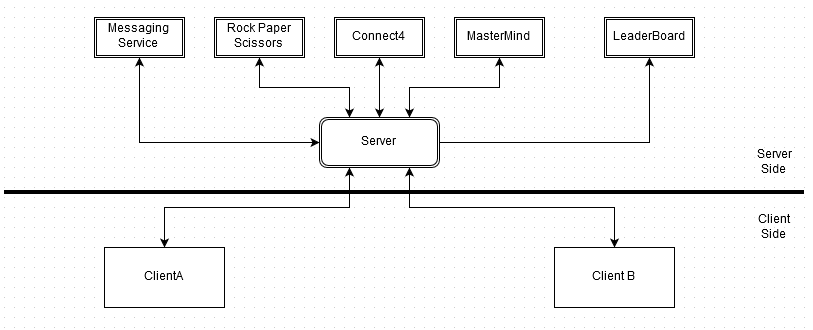
|  |  |  |
| --- | --- | --- |
| Barry Percy  12808148  12808148@ucdconnect.ie | Ciarán O'Neill  12432672  12432672@ucdconnect.ie | Sean Gallagher  12304556  12304556@ucdconnect.ie |

**System Description**

Our project was to design and build a game management system. This game management uses an apache activemq jms server allowing clients to connect and play games against one another. There are currently three games that have been designed for use with our system. These are Connect4, Rock Paper Scissors and Mastermind. The system is robust and can handle other games as well, granted that said games can be adapted to work with our code. Chess and checkers are some games that come to mind that could be designed and used with our system.

A client connects to the server and inputs a username. They are then prompted with a selection of choices such as to get statistics of players current wins and losses, a list of games and there queues, the ability to join a game and an option to quit.

When two people have joined the queue for the same game, a game is started. The two clients will play against one another. When they are finished they are returned to the main menu. If they are waiting, they will be prompted every 10 seconds if they wish to continue waiting or return to the main menu.



*Diagram of system Architecture.*

**System Architecture**

All of the client side code is written in java. We are using an apache activemq web server to handle the clients requests. Clients invoke methods on a messenger object to communicate back and forth with the server. Whether a client is a producer or consumer depends on the method being invoked.

In the beginning, clients will request a connection to the server. Once the connection is accepted, the client will send a username to the server. The server will use this name to generate a unique id to identify the client. This id is sent to the client and stored locally. All exchanges of information between the clients and the server is done in strings. In order to convert objects into strings and back, a serialisation class was created which uses a byte64 encoder and decoder. Each game will be represented by a different class/object. These will be made and destroyed by the server as needed. There will also be a leaderboard showing the rankings for each user and game type.

When a client joins the server, it requests an object for either the leaderboard or gameserverstatus. These objects contain information in relation to how many people are playing and the statistics of their winnings. All of the games are subclasses or the main supergame class. This allows us to change the type of an object the parent class to the child class without any issues. The client will use the subclass objects when they are playing a specific game. The Server will then send a list of games for the client to choose from. The client picks a game and enters a queue. When two clients are in the same queue, a game commences. Clients will send a move to the server and the server will respond with an updated gameboard state represented in ascii. The server then asks the next client for their move and this happens continuously until someone has won

One component that we originally wanted to add to the system was a way for two people playing the same game to message one another. This was later abandoned due to complications. A basic messaging service could be done but the user would have to pick between either sending/receiving a message or sending/receiving their move. If they decided to pick move, they can no longer send or receive messages until their opponent makes a move. A proper system would require two text displays, one for the game and one for messages along with using event listeners. Since this is a prototype project, we decided not to.

At the moment, I know of four bugs in the system. The first is if a client crashes while playing a game or in a queue, all the queues on the server have to reset since that clients info is now stuck in the server. The next is while a user is prompted to continue waiting for game, if another client connects, they will connect straight into the start of a game since the system thinks there are 2 users waiting. If the first client decides not to wait, they will return to main menu, but the second client will be stuck in the game without a way of disconnecting unless they terminate the program. The third issue is that there is no concurrency control on usernames. If two people with the same name join the server, they will interfere with each other. The last one is in connect4 where if there are no longer any possible ways to win due to the arrangement of the counters dropped in, its impossible to exit the game without terminating.

**Problems we encountered**

If a variable used in one of the objects is static, it will only update for the client that created it. If we had of known this, we could've shaved 6 hours off the project easily instead of staring at a screen, feeling our blood boil out of frustration. Another problem encountered was when testing the serialiser class, I serialised the wrong object at the beginning. This led to classcastexception and We were puzzled for a long time.

Out of all the games, rock paper scissors ending up being the most head racking. Connect4 and mastermind have 2 end game conditions. You either win or you lose. However in rock paper scissors, it is possible to draw a game. This meant that we needed to adapt our system to work with games where you have a draw. We came up with the skip move method to skip your enter move step and go straight to see if you've won. The solution makes more sense when your looking at the code

**Future work/improvements**

I already discussed how more games could be added if they are written in the right format.

We could also get around to adding that messenger service using listeners and having multiple text windows.

Updating the leaderboard to show how many games you've played, wins, losses and maybe even have different tallies for each game as well as overall.

Revise the server client communication and change the code to keep track of which users are currently connected to the server instead of whether or not they're in a queue.

**How to run our system**

In order to use our system, you first need to start the server. This can be done by navigating into the games4gamers folder. There are two files with the name RunServerWindowsXXBit.bat where XX refers to what version of windows you are using. Double click the appropriate batch file to start the server. The server can be stopped by using Ctrl-C

There are two ways to run the initialiser and the client. The first is to open the project in eclipse, run InitialiseServerQueues Class to initialise the queues on the server. You then run any number of clients you wish to test our system with.

The other way is to navigate to the games4gamers folder in command prompt. After starting the server, run Re-InitialiseServerQueues.bat followed by running StartClient.bat. You may want to open more than one terminal and run a number of clients in order to efficiently test our system.

connect to http://localhost:8161/admin/ using your web browser to see the queues and topics on the apache activemq server if you are hosting the server locally. The username and password are both admin.