University of Minho

Master in Computer Engineering

Distributed Systems

Matchmaking in an Online Game

Authors:

Grupo 36

Césario Perneta (A73883)

Chamnan Leng (e8027)

João Gomes (A74033)

Tiago Fraga (A74092)

January 3, 2018

Contents

Chapter 1

Introduction

This assignment has been proposed with the intent of applying the insights acquired in Distributed Systems during the semester, in which it's asked to implement a program in *Java* capable of managing an Online Game Matchmaking . This matchmaking has phase one and phase two: the former consists in making two teams with ten players (clients) and the former in how each client choose their hero to play the game.

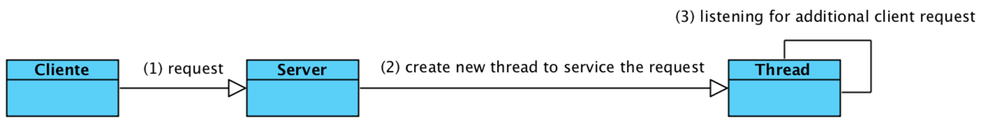
The assignment requires a special care when it comes to managing shared resources - for instance, it's necessary handle how the different clients choose different heroes.

It was necessary that clients can do the register and the *login* in the application.

Chapter 2

Implementation

This application contains two main programs - client and server. The server's main function is to accept incoming client connections and assign a new thread to deal with them.


(3) listening for additional client request
Cliente
(1) request
Server
(2) create new thread to service the request
Thread

The client attempts to connect to the server - if it has success, it communicates through sockets with input and output streams (Strings). This strings make possible that online game matchmaking works with the previously mentioned features.

2.1 Server

The server contains the necessary data structures to store client, hero and game related information.

Class Server is formed by object structures (ex: Client,Game).

2.1.1 Main Server

```
private HashMap <String, Player> players;

private HashMap <String,BufferedWriter> clients;

private HashMap <Integer,Game> games;
private HashMap <Integer,Hero> heroesList;
```

In this structures we store information about the player and client by *username* (Key) and store information about game and hero by *Integer* (Key) which corresponds to the number of the game and hero.

In this class to control competition we use *Java synchronized* methods, so we avoid changing same fields of structures at the same time.

We create *synchronized* methods when threads will change the structures values, for example when create a new game or the client does the register...

Server class still has the *main* and a start method which has the purpose of initialising and running the matchmaking server.

2.1.2 Multi-Threading

We have a class with the name of *Server-Worker* which has the purpose of handle with the client, this is the server creates a new thread running this class (**run()** method implemented in *Server-Worker*) for each client connected to the main server. Because of that each client communicates with the *Server-Worker* and doesn't communicates directly with the server. In this way we don't overload the main server, avoiding that slower clients delay others.

This thread have the main function of exchanging *Client->Server* and *Server->Client* messages, works like a bridge between *Server* and *Client.* This class contains all logic of our program so we allow the server to just listen to new connections with the new clients. Inside of *Server-Worker* we create another thread in the hero's choice phase because of the waiting time, this thread remains listening to the clients choices. During the game there is a phase to exchange messages with the teammates before this situation we create another thread to exchange messages during the stipulated time.

2.2 Client

The *Client* class (program *Client*) contains the methods to interact with the server, and inside of the client we put two classes. This classes have a **run()** method to run a thread when the time comes out. One of them has the purpose of listen the *Server-Worker* messages and print the receive message (***ClientListener* class**), the other has the purpose of read the input of client and send this for the socket of the server (***ClientWriter* class**). This threads are created in the waiting phases and run until has elapsed.

Chapter 3

Discussion of Results

At the end of the implementation of the application, this working group considers that the assignment meets the requirements initially proposed by the statement. It should be noted that there are several aspects that could be improved. Although it is impossible to enumerate in this all the work that should be done to make this application, an application stable in all situations. For example it would be important to test the application with a larger number of clients trying to play, in this way we could detect execution errors that may be hidden by the fact that a test with a major competition has never been done.

It would also be necessary, in the case of server, instead of always being create threads for each client that connects, create a thread pool, avoiding this potential memory depletion risks and would also promote the reuse of threads, whose creation is, as we know, a very costly process. It would also be useful, and by that we mean essential, create data persistence (Users,etc) in a database and also create a graphical interface to improve the experience of using the application.

These are just a few things to improve the application but we could list more aspects.

Chapter 4

Conclusion

As conclusion of this assignment we can mention that the **programming language *Java*** provides a complete **API** for managing competition between threads. The use of the ***synchronized*** on methods and objects gives the programmer concurrent accesses to critical zones.

We use this type of methods to managing competition because it is better when using **TCP Sockets**.

Finally, it should be noted that the working group is satisfied with the work produced, but there was still a feeling that much more could be done, there had been more time to deploy the application.