Escola de Engenharia
Universidade do Minho

DEPARTAMENTO DE ENGENHARIA INFORMÁTICA
Mestrado Integrado em Engenharia Informática
Sistemas Distribuidos

Trabalho Prático

Matchmaking num jogo online

Grupo 40

Célia Figueiredo
ae3698

Adriana Pereira
a67662

Márcia Costa
a67672

Braga, 3 de Janeiro de 2018

Conteúdo

Resumo

O documento descreve o trabalho efetuado para a realização do projeto, onde foi pedida a implementação de uma aplicação distribuída para matchmaking num jogo online por equipas. A funcionalidade essencial é a formaçao de duas equipas com jogadores com rankings semelhantes, após a inserção numa equipa cada utilizador necessita de escolher um herói para jogar antes de iniciar a partida. Os utilizadores interagem usando um cliente escrito em Java, intermediados por um servidor multi-threaded também escrito em Java, e recorrendo a comunicação via sockets TCP.

1. Introdução

O presente relatório documenta o trabalho prático referente a Unidade Curricular de Sistemas Distribuídos pertencente ao plano de estudos do 3º ano do Mestrado Integrado em Engenharia Informática.

Um servidor pode permitir o acesso concorrente através de Threads, se o projeto não for bem feito, as ações dos clientes podem interferir umas nas outras, essas interferências podem resultar em resultados incorretos dos objetos. Ao longo deste projeto veremos como são tratatos os problemas de concorrencia na implementação de uma aplicação distribuida para *matchmaking* num jogo online por equipas.

2. Descrição geral do projeto

2.1 Implementação

Foram criadas classes que permitem o funcionamento do programa

2.1.1 Client

A classe *Client* é responsável por guardar toda a informação do cliente, permite a conecção deste ao servidor e permite executar todas as funcionalidades pretendidas.

MainClient

A classe *MainClient* executa o cliente.

2.1.2 Server

A classe *Server* é responsável pelo servidor e é esta que está à escuta de novas conecções dos clientes. Quando uma coneção é aceite toda a sua lógica é delegada na classe ServerThread.

ServerThread

A classe *ServerThread* é responsável por tratar os pedidos de cada cliente, existindo portanto uma thread por cliente. Sempre que necessário delega ações na classe GameManager.

MainServer

Esta classe representa o executável que permite manter o servidor ligado enquanto os utilizadores desfrutam da aplicação.

2.1.3 Timeout

A classe *Timeout* é responsável por garantir que a fase de escolha de um herói não ultrapassará um determinado tempo. Esta classe é executada numa Thread à parte, garantimos desta forma que nem o cliente nem o servidor bloqueiam o fluxo normal do programa.

2.1.4 Game

A classe *Game* contém as informações relativas ao Jogo. É aqui que é feita a seleção de herói, alteração do heroi e geração dos resultados do jogo.

2.1.5 Hero

A classe *Hero* contém os herois disponiveis para o jogo.

2.1.6 User

A classe *User* contém todas as informações necessárias relativas a um utilizador que pretende jogar.

2.1.7 GameManager

A classe *GameManager* é a classe responsável por armazenar os dados dos utilizadores, herois, jogos. É nesta classe que a maioria das ações solicitadas pelo servidor são efetuadas entre elas destaca-se a eleição da equipa e a criação do jogo.

3. Comunicação Servidor-Cliente

Para cumprir com os requisitos do enunciado, implementamos um servidor e um cliente que comunicam via sockets (TCP).

A *ServerThread* tem como objectivo receber mensagens por parte do Cliente e enviar mensagens para o Cliente. Do lado do Cliente verifica-se a mesma estrutura, onde o Cliente envia mensagens e recebe mensagens por parte do Servidor. As mensagens neste problema em questão, são orientadas à linha, isto é a comunicação é feita através de linhas de texto (Strings), onde cada linha de texto equivale a uma mensagem. A imagem que se segue clarifica melhor a estruturação:

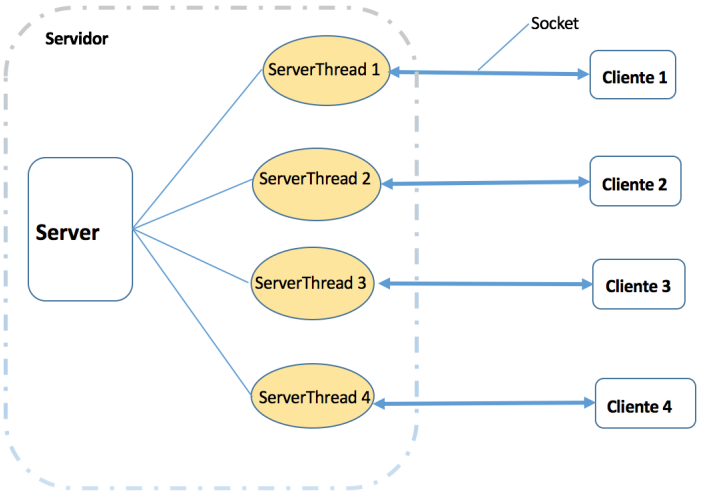

Figura 3.1: Comunicação entre cliente e servidor

4. Controlo de Concorrência

O principal objectivo deste trabalho é controlar o acesso ao conjunto das threads a zonas criticas da aplicação. Foram identificadas três situações onde o controlo de concorrência seria necessário.

4.1 Formação das equipas

Cada thread tenta formar uma equipa com jogadores com o mesmo ranking, superior uma unidade ou inferior uma unidade. Se não houverem os 10 jogadores, os existentes que querem jogar serão adicionados a uma lista e o seu estado alterado para *setWaiting(true)* e ficarão adormecidos numa condição *wantTeamCondition.await()*.

Se a thread conseguir formar equipa, o estado dos elementos desse jogo é alterado por esta Thread e o estado deixa de ser à espera para formar equipa: *setWaiting(false)* (invocando o método *updateWaitingTeamStatus(team))* que muda o estado de toda a equipa) e de seguida são removidos todos os elementos da lista, pois já não estão à espera de equipa. Após efetuar estas ações a thread que conseguiu formar equipa acorda todas as threads que estavam a dormir (*wantTeamCondition.signalAll();*) porque não tinham conseguido formar equipa. De todas as threads que estavam a dormir nesta condição (não tinham conseguido formar equipa) só irão prosseguir para jogo as que o seu estado (*waiting=false*) foi alterado pela thread que conseguiu formar equipa, todas as outras irão voltar a adormecer.

A thread que formou a partida para além do descrito acima cria também o objeto *Game* que contém toda a informação do jogo (p.e. as duas equipas) e invoca a classe *TimeOut* que será a responsável por limitar o tempo de escolha do herói por 30 segundos.

4.1.1 Problemas desta solução

Um problema desta solução é que se um jogador não conseguir uma equipa com jogadores com ranking semelhante ao dele irá ficar infinitamente à espera de equipa.

4.2 Seleção de herói

O objeto Game para além de ter as duas equipas e o estado da partida contém também duas coleções, cada uma delas com todos os heróis. Cada equipa irá usar somente uma destas coleções para posteriormente cada elemento da respetiva equipa selecionar o herói. Cada thread quando inicia o processo de seleção de herói, após pedir ao cliente o herói pretendido a thread irá tentar adquirir um *lock* ao herói na coleção de heróis correspondente à equipa onde está. Após obter o *lock* ao herói pretendido a thread irá usar uma variável de estado (*used*) para saber se o herói já está a ser usado. Se não estiver a ser usado a thread irá alterar o estado

desta variável para que um outro cliente da mesma equipa não possa utilizar esse herói. Para obter o lock ao herói na coleção estamos a utilizar o mecanismo *synchronized* desta forma garantimos que se outro jogador da mesma equipa tentar selecionar um herói que não esteja a ser usado irá consegui faze-lo. Portanto as threads irão conseguir executar em paralelo desde que os heróis selecionados sejam diferentes. Além disso garantimos também que se dois ou mais jogos estiverem a ocorrer em simultâneo, os vários jogadores poderão escolher heróis em comum, desde que os elementos da sua equipa não tenham escolhido esse herói. Tal facto é possivel pois as duas colecoes de heróis apesar de terem os mesmo heróis em todos os jogos, estas coleções são colonadas ou seja teem endereços diferentes e por isso o *lock* de um herói é um *lock* de um endereço diferente.

Após o processo acima descrito se foi possivel alterar a variável de estado do herói, a seleção foi efetuada com sucesso, o cliente irá receber uma mensagem avisando que a escolha foi efetuada com sucesso. Caso contrário o servidor irá informar o cliente que não foi possivel selecionar o herói uma vez que este já estava a ser utilizado por outro utilizador da equipa, e o utilizador deverá escolher um herói diferente.

4.2.1 Alteração de herói

É um processo semelhante ao anterior, com a diferença de que neste caso é necessário adquirir dois *locks*, um ao herói que já estava selecionado e outro ao herói que se pretende alterar.

4.3 Confirmação de herói e Iniciação de jogo

Após o utilizador confirmar o seu herói ele irá validar se todos os outros elementos do jogo tem o herói também confirmado, se nem todos tiverem o herói confirmado e se o tempo limite do *timeOut* não tiver chegado ao fim as threads irão adormcer numa fila de espera especifica para os utilizadores que esperam que toda e equipa tenha os herois selecionados. Caso todos os elementos do jogo tenham herói confirmado e o *timeOut* não tiver sido excedido, esta thread irá alterar o estado do jogo *timeout = false*, gera o resultado aleatório do jogo e acorda todas a threads que estão em fila à espera que todos os elementos da equipa tenham herói confirmado (*this.wantGameCondition.signalAll()*). Apesar de acordarmos todas as threads que estão nesta fila só irão prosseguir para jogo as do jogo em questão, todas as outras irão voltar a adormecer.

4.4 Tempo máximo de seleção de herói

A classe *Timeout* é uma classe que extende Thread e portanto irá ser executada numa outra *thread*, por isso mesmo é que apesar do método *run()* executar um *sleep()* o programa não bloqueia. Após a thread adormecer 30 segundos ela irá tentar alterar o estado do jogo para *timeout==true*, todo o método que tenta alterar o estado do Jogo está dentro do mesmo *ReentrantLock()*), por este motivo a alteração desta variável será executada no máximo por uma thread (sem concorrência) garantimos assim a exclusão mútua nesta seccção critica do código. O estado do Jogo poderá também ser alterado pelo fluxo normal do programa mas o método responsável por efetuar esta acção encontra-se restringido pelo mesmo *ReentrantLock()*) que é usado pelo *Timeout* evitamos assim problemas de concorrência ao aceder/alterar esta variável.

No caso da classe *Timeout* o método que altera a variável de estado do jogo (só o irá fazer se o fluxo normal ainda não tiver atualizado esta variável e vice-versa). Se conseguir alterar o

estado da variável de seguida irá acordar todas as threads que estão adormecidas numa condição em que estão à espera que todos os elementos do jogo tenham o herói confirmado. Desta forma garantimos que quando o tempo limite de escolha de herói é atingido se existirem clientes que estejam adormecidos na fase de escolha de herói são acordados, não necessitamos assim que todos os utilizadores tenham escolhido herói para saberem que o tempo máximo foi excedido.

4.4.1 Problema desta solução

O problema desta solução é que só depois de os utilizadores confirmarem herói é que irão saber se a partida foi abortada ou não. Em suma o utilizador não sabe em tempo real que o timout aconteceu, apesar disso garantimos que os jogos com mais de 30 segundos na escolha do herói são abortados e não temos threads bloqueantes no fluxo normal do programa ou que fazem esperas ativas.

5. Funcionamento da Aplicação

A aplicação apresentada tem uma interface muito simples, uma vez que o seu proprósito é ser utilizado numa linha de comandos.

5.1 Menus

O menu inicial mostra ao utilizador 3 opções: Iniciar sessão, registar na aplicação ou sair.

```
run:
Numero do cliente #5
************* MENU ****************
* 1 - Iniciar Sessao              *
* 2 - Registar                    *
* 0 - Sair                        *
***********************************
```

Figura 5.1: Menu Inicial

Após o inicio de sessão, o utilizador poderá tentar arranjar equipa ou então efetuar logout.

```
1
Username:
celia
password
celia
************* Equipas ************
* 1 - Entrar na Equipa             *
* 0 - Logout                       *
***********************************
```

Figura 5.2: Menu de inicio de sessão

Após selecionar a opção entrar numa equipa, o utilizador ficará à espera até ser encontrado uma equipa que tenha outros jogadores com um ranking semelhante ao dele e após ter sido colocado numa equipa é informado que já tem equipa e é mostrada a constituição da equipa dele e a adversária, seguido do menu para escolher o heroi. Quando escolhe o heroi é apresentada uma lista com 30 herois e o utilizador necessita de escolher um.

```
1
Já tem equipa

A sua equipa é constituida por:
Username: celia, Heroi: Ainda nao esta selecionado | Username: marcia, Heroi: Ainda nao esta selecio
A equipa adversária é constituida por:
Username: celia, Heroi: Ainda nao esta selecionado | Username: adriana, Heroi: Ainda nao esta selec
******************** Menu Jogo *************
* 1 - Escolher Heroi                      *
* 2 - Visualizar Constituiçao da Equipa    *
*********************************************
```

Figura 5.3: Entrar numa equipa

Após selecionar o heroi escolhido o utilizador pode confirmar, escolher outro e ainda visualizar a equipa. De notar que o utilizador apenas tem 30 segundos para efetuar a escolha de heroi.

```
3
************* Menu Jogo ************
* 1 - Confirmar heroi e jogar             *
* 2 - Escolher outro heroi                 *
* 3 - Visualizar Equipa                  *
***********************************
```

Figura 5.4: Menu para a escolha de heroi

Após ser confirmado o herói é gerado um resultado aleatório em que uma das equipas ganha e são atualizados os rakings dos jogadores. Após este acontecimento o jogador poderá tentar jogar outro jogo.

```
1
A sua equipa é constituida por:
Username: luis, Heroi: heroi5 | Username: celia, Heroi: heroi26 |
A equipa adversaria é constituida por:
Username: marcia, Heroi: heroi13 | Username: adriana, Heroi: heroi3 |
Parabens a sua equipa venceu
************* Equipas ************
* 1 - Entrar na Equipa             *
* 0 - Logout                       *
***********************************
'
```

Figura 5.5: Jogo efetuado com sucesso

6. Conclusões

Com a realização deste trabalho foi possivel aplicar os conceitos e o conhecimento adquirido durante as aulas, pondo em prática grande parte daquilo que foi transmitido. Tal como sabemos, os Sistemas Distribuídos têm como principal objetivo otimizar o desempenho de maneira que seja possível dividir tarefas e processá-las separadamente, promovendo assim paralelismo e reduzindo o tempo de execução. Sendo a escalabilidade um dos mais importantes aspetos de um Sistema Distribuído, um dos nossos principais objetivos foi promover a concorrência, usufruindo para isso da implementação de Threads. No entanto, ao mesmo tempo que promovemos a concorrência é necessário assegurar a fiabilidade dos dados através da implementação de exclusão mútua em seccões criticas, utilizando para isso os mecanismos adequados, tal como locks e variáveis de condição.

Consideramos que conseguimos implementar uma solução para o problema que utiliza o conceito de threads mas garante o correto funcionamento do programa, pois nas zonas criticas onde a concorrência não deve existir as técnicas de controlo de concorrência foram utilizadas. Tentamos também garantir que o programa não possibilitaria a ocorrência de *deadLocks* e *starvation*. Por exemplo para garantir que não ocorre *starvation* na fase de formação de equipa tivemos o cuidado de utilizar uma lista para armazenar os jogadores que pretendem formar equipa. Conseguimos desta forma garantir que quando uma thread tenta formar equipa ela irá tentar usar os utilizadores que estão à mais tempo em espera consoante o ranking.

Apesar disso sabemos que não conseguimos atingir uma solução ótima para o tempo de espera na fase de escolha do herói e que na fase de formação de equipa um cliente poderá ficar eternamente à espera por não ter jogadores do seu nivel para jogar.