Universidade do Minho

Departamento de Informática

Sistemas Distribuídos

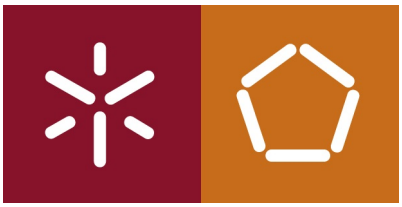
Trabalho Prático - Matchmaking de um jogo online

Grupo 26
Gil Cunha (a77249@alunos.uminho.pt), Nuno Faria (a79742@alunos.uminho.pt)

Resumo

Este relatório abordará o desenvolvimento do projeto prático de Sistemas Distribuídos, que consiste na implementação de um sistema de *matchmaking* para um jogo online.

3 de Janeiro de 2018



Conteúdo

1 Introdução e Objetivos

Este trabalho consiste na implementação de um sistema de matchmaking para um jogo online.

O projeto seguirá uma abordagem cliente-servidor (fazendo uso de *sockets* TCP), sendo que devido à existência de leituras e escritas será necessário proceder-se ao desenvolvimento de um sistema *multithreaded* para suportar toda a lógica. Visto que vários clientes poderão aceder ao servidor ao mesmo tempo, é necessário assegurar a consistência dos dados através do controlo da concorrência.

Visto que esta implementação destina-se a um jogo online, é necessário um bom desempenho por parte do servidor, respondendo aos pedidos dos jogadores em tempo razoável.

2 Desenvolvimento do Sistema

2.1 Jogador

É através da classe `Player` que será armazenada toda a informação de um jogador, desde o seu nome até aos heróis mais utilizados por este.

```
1  private String username; //nome
2  private String email;  //email
3  private String password;  //password
4  private int rank;  //rank atual
5  private int win;  //num. vitorias
6  private int lost; //num. derrotas
7  private HashMap<String,Integer> heroes; //heroi - jogos
8  private String currentTeam; //equipa atual
9  private String currentHero; // heroi atual
10 private boolean loggedOn;  // estado
11 private int numberMVP;  //num. vezes como melhor jogador
```

2.2 Servidor Principal

O servidor principal irá ser o responsável por toda a gestão do sistema. Registará jogadores, confirmará as autenticações, procurará jogos e construirá equipas.

Inicialmente, é criada uma *thread* para receber as ligações dos clientes. A cada nova conexão são criadas duas novas *threads*, uma para ler do cliente e outra para enviar-lhe mensagens.

2.2.1 Login e Registo

O *login* e o registo são bastantes triviais, sendo que será apenas feita a comparação com *username/password* (no *login*) e verificação de dados repetidos (no registo). Será guardado um booleano contendo a informação sobre a autenticidade deste jogador, para impedir que exista mais que dois clientes com a mesma conta.

2.2.2 Logout

O *logout* será necessário para permitir que a desconexão de um cliente não cause problemas no sistema, tratando de retirar o jogador da fila de espera (caso esteja nela) e para remover os `locks/buffers/conditions` associadas à sua conta.

2.2.3 MatchMaking

O *matchmaking* assentará em duas vertentes: a pequisa de um jogo e a escolha das equipas.

Visto que um jogo só será realizado entre jogadores com diferença máxima no *rank* de uma unidade, será necessário implementar um algoritmo (mais eficiente possível) para tratar desta procura. A forma como foi implementado foi a seguinte (sendo P o jogador que inciou a procura):

- É verificado se existem pelo menos nove jogadores do *P.rank*;
- Caso existam, são removidos da fila de espera e colocados num conjunto de jogadores que irão jogar, juntamente com P;
- Caso contrário, será verificado se existem pelo menos nove jogadores na soma dos do *P.rank* mais os do *P.rank +1*
- Caso exista será efetuado o passo 2. Caso contrário, será feito o mesmo para o *P.rank-1*;
- Se não encontrar outros 9 jogadores será colocado na fila de espera.

Um jogo é procurado sempre que um jogador manifesta a sua intenção para jogar.

Após serem encontrados jogadores para jogar uma partida, será feita a divisão entre equipas. Visto que o objetivo deste trabalho não era necessariamente gerar equipas perfeitamente balanceadas, optamos por algo simples, atribuindo a equipa 'A' para jogadores cujo índice num *array* é par e 'B' caso contrário. Se quisemos ser mais rigorosos, poderíamos implementar programação inteira de modo a dividir um conjunto em dois subconjuntos cuja soma dos *ranks* tenha uma diferença miníma.

Encontrados os jogadores e feitas as respetivas equipas, inicia-se um novo jogo.

2.2.4 Estatísticas

Para além das funcionalidades descritas atrás, serão também envidas estatísticas sobre o jogador autenticado e sobre o *top* de melhores jogadores. Estas estatísticas contem informação sobre o *rank*, o número de vezes jogadas, a quantidade de derrotas/vitórias e os heróis mais utilizados.

2.2.5 Estrutura

De modo a suportar a lógica do servidor, será necessário guardar os dados de cada jogador, uma fila de espera (um `Map` de `Lists` indexado pelo *rank*), variáveis para o controlo de escritas (*locks*, *buffers* e *conditions*) e todas as portas disponíveis.

```
1 private HashMap<String, Player> players;
2 private HashMap<Integer,ArrayList<Player>> queue;
3 private HashMap<String, Vector<String>> buffers;
4 private HashMap<String, ReentrantLock> locks;
5 private HashMap<String, Condition> writeConditions;
6 private HashMap<Integer, Boolean> ports;
```

2.3 Servidor de Jogo

É criada uma nova *thread* que instancia um servidor de jogo. Caberá a esta receber as ligações dos dez jogadores e gerir a lógica do jogo em si. Tal como no servidor principal, será criada uma *thread* para cada jogador, que irá processar os pedidos do cliente. Contudo, não será criada uma *thread* para escrita, sendo que será usada a que já foi criada no servidor principal (abordaremos isso mais adiante em "Comunicação").

Cada jogador poderá escolher um dos 30 heróis desde que não tenha sido escolhido por um colega de equipa. Por isso, teremos que controlar a concorrência de forma a não existir inconsistências. Teremos um método `synchronized` que tratará de verificar se o herói que um jogador está a tentar aceder encontra-se disponível. Se sim, é atribuído-lhe o herói pretendido, retirando-lhe o anterior caso tivesse selecionado algum.

Para auxiliar na construção de um *bot* foi implementado um método que atribuirá um herói aleatório, dentro dos disponíveis.

Voltando à *thread* principal, depois de todos os jogadores se tiverem conectado, está irá bloquear durante 30 segundos. Caso durantes esse intervalo algum jogador saia do jogo, esta é acordada e terminará prematuramente a partida. Caso no fim desse intervalo existam jogadores sem heróis, a partida é terminada sem sucesso. Caso ninguém saia do jogo e todos tenham escolhido um jogador, são efetuados os seguintes passos:

- São gerados três números aleatórios, sendo dois deles para a pontuação das equipas e outro para decidir o melhor jogador;
- Consoante os valores gerados, é atribuído a equipa vencedora e serão atualizados os dados de cada jogador;
- A porta atual é libertada e o servidor de jogo termina.

Quando ao envio de informação, no início é enviada a constituição das equipas; durante é enviada a informação sobre os heróis de cada jogador (mensagens diferentes para cada equipa); no fim é dito quem ganhou/perdeu.

Relativamente à estrutura, esta será semelhante à do servidor principal, destacando-se dois `Maps` que guardaram a informação dos heróis usados de cada equipa.

2.4 Cliente

O cliente será constituído por duas *threads*, uma de escrita e outra de leitura.

A *thread* de escrita não faz parte da classe `Client`, pertencendo a outro processo (por exemplo, da aplicação gráfica), usando métodos nesta classe para enviar informação para o servidor principal / de jogo. Quanto à *thread* de leitura, esta irá ler e processar todas as mensagens recebidas, fazendo o *parse* de informação e alterando variáveis de controlo. A comunicação entre a classe `Client` e a classe que a faz uso é feito através da implementação de `Observer/Observable`, sendo por isso necessário as tais variáveis de controlo para poder ser indicado o que foi alterado.

2.5 Comunicação

Visto que esta é uma parte essencial para o funcionamento do sistema, decidimos dedicar-lhe especial importância.

A comunicação funciona bilateralmente, não sendo à base de pedido → resposta, o que leva a que existam duas *threads* (escrita e leitura) para o cliente e servido. Existindo uma *thread* especificamente para escritas no lado do servidor, esta tem que bloquear quando

não existe nada para fazer, sendo por isso necessário a existência de **condições**. De todas as vezes que a *thread* de leitura do servidor principal ou de jogo querem escrever para o cliente, guardam as mensagens num `Vector` e sinalizam a *thread* de escrita para enviar essa informação. Visto que é apenas uma que faz essa escrita sistema fica mais simples (sendo ainda assegurado que no servidor uma *thread* escreve apenas para um *socket*).

2.5.1 Protocolo

A comunicação é feita à base da linha de texto. Criamos o seguinte protocolo para suportar a troca de mensagens:

Servidor Principal → Cliente	
`Wrong username/password`	Dados incorretos
`User already registered`	Dados repetidos
`Successfully registered`	Registado com sucesso
`Authenticated`	Login efetuado
`Added to queue`	Adicionado à fila de espera
`Info ...`	Estatísticas do jogador
`Tops ...`	Top ranks globais
`Game_Starting [ip] [port]`	Jogo está a começar
Cliente → Servidor Principal	
`Login [name] [password]`	Autenticação
`Register [name] [email] [password]`	Registo
`Retrieve Info`	Pedido de informação
`Play`	Jogar uma partida
`Cancel`	Cancelar o pedido de jogar
`Close/Exit`	Abandonar o jogo/aplicação
Servidor de Jogo → Cliente	
`Team [team]`	Equipa a que pertence
`Player [name] [team]`	Infomação dos outros jogadores
`Selected [hero]`	Confirma a selação de um herói
`Hero already used`	Herói já foi escolhido
`Players_Heroes ...`	Herois escolhidos pela equipa
`Chat::[player]:[message]`	Mensagem de chat
`Remove [player]`	Jogador saiu da partida
`Output [message]`	Resultado do jogo
Cliente → Servidor de Jogo	
`Select [hero]`	Selecionar um herói
`Get Unused Hero`	Selecionar um herói aleatório
`Chat::[message]`	Enviar uma mensagem de chat
`Close/Exit`	Abandonar partida

Tabela 1: Protocolo usado na comunicação

2.6 Bot

Para testar o funcionamento do sistema foi implementado um *bot* que irá simular um jogador (fazendo uso da classe `Client`). Este funcionará da seguinte forma:

- Registar uma conta (caso sejam passados três argumentos);
- Autenticar-se na conta registada;
- Caso não tenha sucesso o processo termina. Caso contrário, pedirá para jogar uma partida;
- Depois de iniciada uma partida, tentará escolher sempre um herói especifico (que será também escolhido pelos outros *bots*)[1];
- Caso consiga escolher o herói específico irá esperar até que o jogo termine. Caso contrário, dentro de um intervalo de 0 a 20 segundos irá requisitar ao servidor de jogo um herói que ainda não tenha sido usado;
- No final da partida, tentará começar uma nova.

3 Interface

Tendo a lógica do sistema toda implementada, decidimos construir uma interface gráfica de modo a ser possível que um cliente interaja com o programa.

A primeira vez que o programa é inicializado, é apresentado o menu de *login*, sendo possível abrir uma outra janela para efetuar o registo de uma conta.

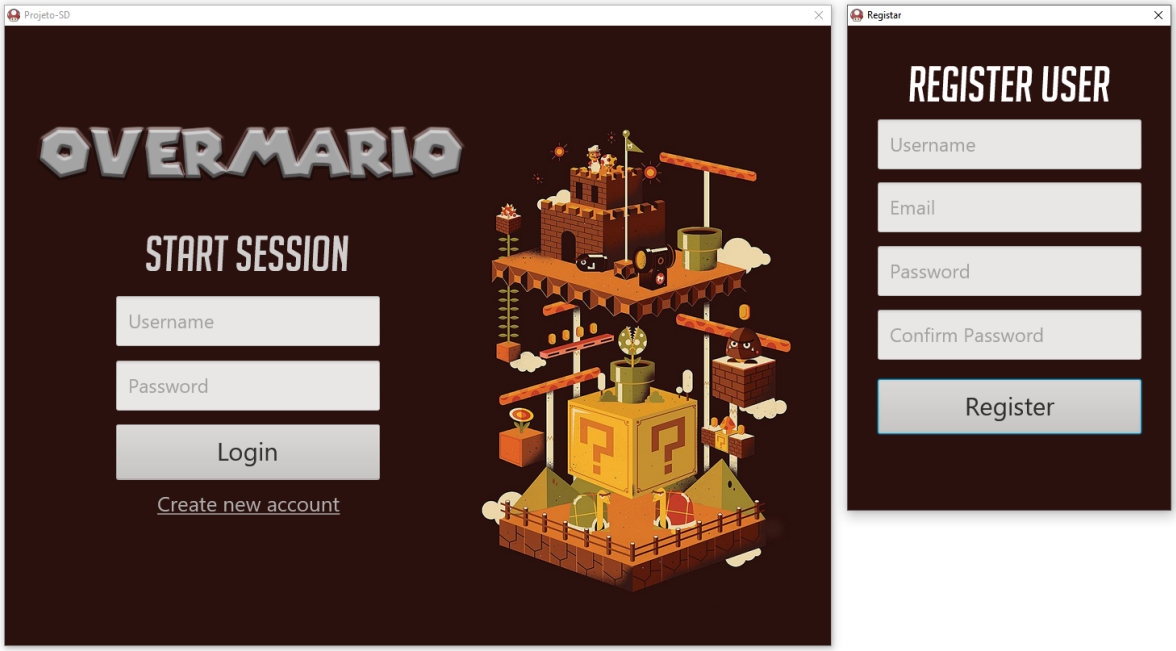


Figura 1: Menu de login e registo

Após a sua autenticação, é apresentado o respetivo menu de utilizador, onde poderá encontrar diversas informações e a possibilidade de inicial uma partida.

[1] Isto irá servir para testar o acesso concorrente ao mesmo herói

– imagem menu utilizador –

Quando inicia uma partida, é apresentado-lhe o menu de jogo, onde poderá escolher um herói, consultar a informação sobre os jogadores que estão a jogar e enviar mensagens a outros através do *chat*.

– imagem menu jogo –

4 Conclusão