# Universidade do Minho

Relatório do Trabalho Prático - 2016/17

Universidade do Minho
Escola de Engenharia

Sistemas Distribuídos

Docente: Carlos Miguel Ferraz Baquero Moreno

Mestrado Integrado em Engenharia de Telecomunicações e Informática – 3ºAno

14 de junho de 2017

# Constituição do Grupo

Francisco Peixoto Silva, A68491

a68491@alunos.uminho.pt

Luís Pedro Lobo de Araújo, A73232

a73232@alunos.uminho.pt

Nuno Frederico Oliveira Tavares, A73985

a73985@alunos.uminho.pt

# Índice



# Índice de Figuras



# Índice de Tabelas



# 1. Introdução

Este relatório documenta todo o trabalho desenvolvido para resolver um problema proposto no âmbito da Unidade Curricular de Sistemas Distribuídos.

O projeto consiste no desenvolvimento de uma solução que permita desenvolver um servidor do jogo "Pedra, Papel, Tesoura", tendo como suporte uma aplicação implementada em Java, que inclui um Servidor *multi-thread* que interage com vários clientes.

Com este projeto pretende-se que sejam aplicados conceitos de programação com *sockets*, ou seja, comunicação entre os clientes e o servidor e a programação concorrente, no sentido em que podem estar a ocorrer vários jogos ao mesmo tempo que o servidor continua a responder devidamente.

Será ainda definido que a máquina cliente sabe o IP e a Porta do servidor e que este possa servir concorrentemente múltiplos pares de jogadores.

# 2. Estudo do problema

Pretende-se desenvolver um servidor do jogo "Pedra, Papel, Tesoura" que deve receber ligações de jogadores e estabelecer sessões de jogo entre pares de jogadores.

Em cada sessão são emparelhados dois jogadores que interagem entre si, por via do servidor, indicando em cada jogada, a sua escolha e o servidor indica, para cada jogada, quem ganhou, se houve empate, ou se alguém abandonou o jogo.

A Tabela 1, em baixo ilustrada, representa todas as jogadas e resultados possíveis do jogo.

| Jogada do Jogador 1 | Jogada do Jogador 2 | Resultado |
|---|---|---|
| Pedra | Pedra | Empate |
| Pedra | Papel | Vencedor Jogador 2 |
| Pedra | Tesoura | Vencedor Jogador 1 |
| Papel | Pedra | Vencedor Jogador 1 |
| Papel | Papel | Empate |
| Papel | Tesoura | Vencedor Jogador 2 |
| Tesoura | Pedra | Vencedor Jogador 2 |
| Tesoura | Papel | Vencedor Jogador 1 |
| Tesoura | Tesoura | Empate |

Tabela 1 - Jogadas e Resultados Possíveis

É pedido que cada máquina cliente saiba o IP e Porto do servidor para se poder estabelecer uma sessão de jogo entre cada par de jogadores. O servidor deverá ser capaz de servir diversas sessões de jogo para os diversos jogadores que poderão estar disponíveis.

O jogo deverá cumprir os seguintes requisitos:

- Programação com *sockets* onde será feita toda a comunicação entre o Cliente e Servidor;
- Programação concorrente para a execução simultânea das diversas tarefas do jogo;
- Garantia que as transações do jogo sejam executadas de uma forma segura e haja deteção de que um jogador abandonou o jogo.

# 3. Projeto

## 3.1. Modelo Cliente - Servidor

O servidor inicializa um *socket*, conhecido a todos os clientes, no porto 50516 onde será possível a troca de mensagens com o cliente. Este aguarda que se conectem dois clientes para iniciar uma sessão de jogo.

Após a conexão de dois jogadores, o servido lança uma *Thread* da classe *GameControl* para inicializar o jogo. Após o inicio do jogo, o servidor fica a aguardar pelos pedidos dos clientes dentro desta mesma *Thread*, neste caso as jogadas de cada jogador. A seguir a cada jogada, o servidor manda o resultado para cada jogador. Na figura abaixo, podemos encontrar o fluxograma com a descrição do funcionamento do servidor. A classe utilizada para o servidor denomina-se *RPSServer*.

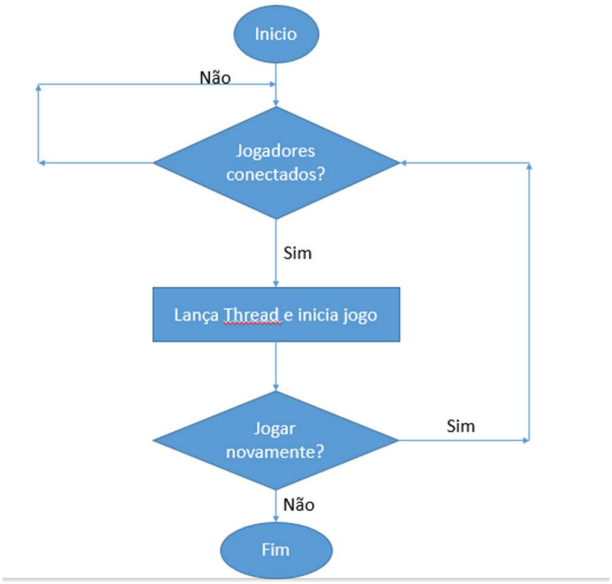


Figura 1 - Funcionamento do Servidor

O cliente, por sua vez, só se terá de preocupar em se conectar a uma sessão de jogo através do *socket* criado. Posteriormente inicializado cada jogo, o cliente apenas terá que enviar as suas jogadas e ficar a aguardar as respostas do servidor para as suas jogadas. A classe implementada em Java para o cliente denomina-se *RPSClient*.

## 3.2. Protocolo

Para a comunicação entre o Servidor e o Cliente, o grupo definiu um protocolo que define quais as mensagens que estes podem trocar entre si, a Figura 2 representa a classe em Java utilizada com os valores identificadores dos vários tipos de mensagens existentes neste protocolo.

```
package RPS;

public final class RPSProtocol
{
        public static final int PORT=50516;

        public static final String MSG_SEP = ":";
        public static final int START = 0; //server Response - game start
        public static final int BET = 1; //client Request - bet Rock,Scissor or Paper
        public static final int RESULT= 2; //server Response - result of game
        public static final int END = 3; //server Response - end of game

        public static final int TIE = 0;
        public static final int WIN=1;
        public static final int LOSE=2;

        public static final int SCISSOR=1;
        public static final int PAPER=2;
        public static final int ROCK=3;
}
```

Figura 2 - *RPS Protocol*

Para este protocolo foi decidido que existem 4 tipos de mensagens que o servidor e o cliente poderão trocar. Estas mensagens têm o formato "Type_Message:Message", em que o carater ":" separa o tipo de mensagem do conteúdo da mensagem. Em baixo podemos ver a explicação para os vários tipos de mensagens:

- START (0): Anúncio aos clientes que o jogo vai começar;
- BET (1): Pedido de aposta por parte de um cliente;
- RESULT (2): Anúncio ao cliente do resultado da sua aposta;

- END (3): Anúncio ao cliente que o seu adversário abandonou o jogo.

Os conteúdos das mensagens foram definidos pelo protocolo para as mensagens do tipo BET enviadas pelos clientes e para as mensagens do tipo RESULT enviadas pelo servidor. No caso do tipo RESULT temos os conteúdos listados abaixo:

- TIE ("3:0"): os dois jogadores empataram;
- WIN ("3:1"): o jogador em questão ganhou;
- LOSE ("3:2"): o jogador em questão perdeu.

Já para o tipo de mensagens do tipo BET podemos ver em baixo o tipo de conteúdo que estas podem apresentar:

- SCISSOR ("1:1"): o jogador aposta tesoura;
- PAPER ("1:2"): o jogador aposta papel;
- ROCK ("1:3"): o jogador aposta pedra.

Para facilitar a compreensão do código nas classes *RPSClient* e *RPSServer* é utilizada esta classe *RPSProtocol*.

## 3.3. Lógica de jogo

O grupo utilizou um algoritmo para definir qual o resultado para cada jogador, para utilizar este algoritmo é chamado um método da classe *Game Control*, este método recebe como parâmetros o valor das apostas dos dois jogadores, sendo que o primeiro valor (*Player1*) corresponde ao valor da aposta do jogador que se pretende devolver o resultado e o segundo valor (*Player2*) ao do seu adversário. Este método é chamado duas vezes invertendo a ordem dos parâmetros dependendo do jogador que se pretende enviar o resultado.

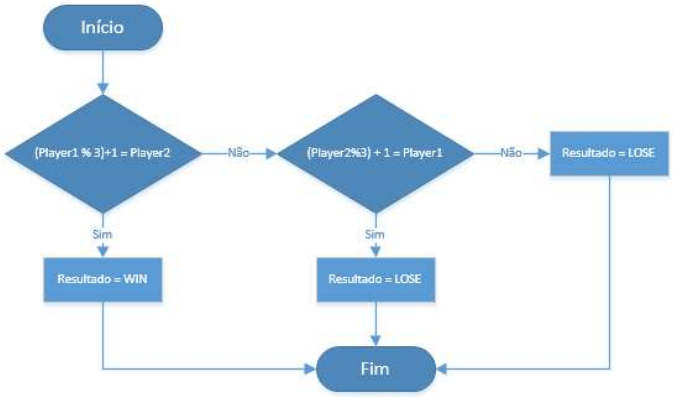


Figura 3 - Fluxograma do método *checkPlay()*

Cada jogo é controlado pela classe *GameControl*. Mal termine um jogo com sucesso, ou um dos jogadores abandona o jogo as ligações estabelecidas com os *sockets* dos clientes são fechadas. Se um jogador abandonar o jogo, é enviada uma mensagem por parte do servidor para o outro jogador informando-o que o jogo foi encerrado porque o seu adversário abandonou o jogo. Quando um jogo acaba, independentemente da forma como termina, a aplicação Cliente pergunta ao jogador se este pretende continuar a jogar, se este responder afirmativamente é estabelecida uma nova ligação através de *sockets* com o servidor, e o mesmo processo repete-se.

# 4. Conclusão

Este relatório serviu o propósito de redigir os passos executados ao longo do desenvolvimento da solução apresentada, com base nos conhecimentos adquiridos na UC de Sistemas Distribuídos, para um exemplo de servidor para o jogo "Pedra, Papel, Tesoura".

Foram utilizadas várias primitivas de programação durante o desenvolvimento da solução para este jogo. Para a interação entre jogadores foi utilizado as primitivas de programação concorrente para que fosse possível aos dois, se emparelharem e jogar simultaneamente. Utilizou-se a programação com *sockets* para a troca de pedidos entre o Cliente e o Servidor e também para a conexão dos jogadores.

Este projeto foi importante porque permitiu ao grupo consolidar os conhecimentos sobre a funcionalidade e uso de *sockets* bem como trabalhar com tarefas *multi-thread*. O grupo considera que o objetivo deste trabalho prático foi concluído com sucesso uma vez que se cumpriram com todas as tarefas propostas.