Your Own Application

Final Report

Miguel Pires – up201406989

Nuno Corte-Real – up201405158

Sérgio Salgado – up201406136

Sistemas Distribuídos

3º ano do Mestrado Integrado em Engenharia Informática e Computação

Índice

Introdução

Para o trabalho final da disciplina de Sistemas Distribuídos, o nosso grupo decidiu fazer um jogo que envolve ligações de cliente-servidor, ambos desenvolvidos por nós.

O jogo funciona da seguinte forma: os jogadores, temporários, escolhem um nome de utilizador (pode haver vários utilizadores na mesma máquina, mas com nomes diferentes) e podem criar um lobby ou juntar-se a um lobby previamente criado por outro utilizador. Por cada turno de jogo, irão haver os jogadores que podem jogar e um que não pode, que será o júri. No início de cada turno, o servidor envia para os jogadores do lobby duas palavras pseudoaleatórias da língua portuguesa, com as quais os jogadores terão que escrever uma frase original que posteriormente será avaliada pelo júri. Numa segunda parte da ronda, o júri escolhe qual a frase vencedora, sendo atribuído um ponto ao autor, começando uma nova ronda. No fim de cinco rondas, é determinado um vencedor, terminando assim o jogo.

Para este jogo funcionar, encontra-se por detrás uma infraestrutura sobre a qual incide este relatório, sendo abordada numa primeira parte a arquitetura do projeto, seguido de como esta arquitetura foi implementada, assuntos relevantes sobre a implementação e por fim uma conclusão.

```
||||||||||||||||||||||||||||||||||||||||||||||||||||
||                        teste                     ||
||--------------------------------------------------||
|| CURRENT JUDGE:        teste_1                    ||
||--------------------------------------------------||
||                  ROUND WORDS                     ||
||--------------------------------------------------||
||             olímpia   |          cinzenta        ||
||--------------------------------------------------||
||          PLAYER NAME          |     SCORE        ||
||--------------------------------------------------||
||               teste_1         |        0         ||
||               teste_2         |        0         ||
||               teste_3         |        0         ||
||--------------------------------------------------||
||    TYPE A SENTENCE THAT USES BOTH WORDS ON       ||
||                  THE SCREEN!                     ||
||||||||||||||||||||||||||||||||||||||||||||||||||||
```

Figura 1 - UI do jogador no início da ronda.

Arquitetura

Como foi dito anteriormente, quer o servidor, quer o cliente, foram da autoria do grupo e para tal, o grupo teve que criar uma estrutura que nos permitisse isso. Passaremos então a descrever quais os componentes principais da arquitetura desenvolvida, começando pelo servidor, passando pelo cliente e terminando no protocolo.

Servidor

O servidor utiliza um protocolo SSL para fazer a ligação com os clientes, sendo criada uma nova *thread* no servidor por cada cliente que o acede. Sendo este acesso feito, o servidor guarda um *HashMap* com informação de quem está conectado e qual *thread* é responsável por esse cliente, estrutura de dados que será essencial para um bom funcionamento do programa.

Cada *thread* do servidor é responsável por atender aos pedidos de cada cliente individualmente, verificando a permissão do cliente para efetuar tal comando no servidor, permitindo assim um nível de escalabilidade, consistência e segurança do servidor bastante considerável, tópico que está abordado mais tarde.

Cliente

O cliente acede ao servidor através de uma ligação efetuada por um *SSLSocket*, enviando comandos ao servidor durante quer a sua navegação nos menus, quer na sua navegação dentro do jogo. Todos os comandos do cliente são validados por lógica dentro do cliente e as permissões validadas no servidor, permitindo reduzir o *stress* no servidor (pois nem todos os comandos têm que ser validados no servidor), melhorar o desempenho deste e aumentar os níveis de segurança (pois não permitimos “injeções” no nosso protocolo).

Protocolo

O nosso protocolo de mensagens entre servidor e cliente é relativamente simples e foi inspirado no protocolo realizado no primeiro trabalho, onde há uma mensagem do cliente para o servidor a enviar um comando e o servidor envia um comando de volta para o cliente com a resposta.

Foi criado um protocolo para as mensagens que vão do cliente para o servidor e outro protocolo para as mensagens de retorno, tópico que será abordado mais à frente.

Implementação

Nesta secção, iremos abordar como formam implementadas cada um dos componentes descritos na secção anterior em mais detalhe.

Servidor

O servidor, tal como todos os componentes do nosso programa, foi escrito em Java e não utiliza *frameworks* nem *libraries* de terceiros.

É relevante dizer que o servidor guarda bastantes estruturas de dados para o melhor funcionamento e atendimento a pedidos possível. O servidor guarda informações sobre os utilizadores conectados ao sistema num *HashMap<User, ServerThread>* que é essencial para o bom funcionamento do programa, pois permite que, por exemplo, quando um utilizador entra num *lobby* com outros, todos no *lobby* recebam mensagem para atualizar a informação do *lobby*, sem enviar mensagens desnecessárias para outros jogadores fora do *lobby* (o mesmo se verifica durante o jogo. Só jogadores que se encontram no *lobby* recebem mensagens sobre as jogadas dos outros jogadores do mesmo *lobby*).

Como foi dito na secção da 'Arquitetura', o servidor utiliza um *SSLServerSocket* e cria uma nova *thread* por cada conexão aceite feita da parte dos clientes *(ficheiro Server.java, linhas 54-59)*. Sendo esta conexão estabelecida, a *thread* corre um loop infinito (não bem infinito, mas até ao utilizador decidir dar *logout*), recebendo mensagens e processando-as *(ficheiro ServerThread.java, linhas 49-57 e 67-96)*.

Após o seu processamento, efetuado na função *solve(String s) (ficheiro ServerThread.java, linhas 117-149)*, o servidor resolve o comando, chamando uma função responsável *(várias funções no ficheiro ServerThread.java, linhas 152-365)* que vai verificar com funções auxiliares *(ficheiro ServerThread.java, linhas 388-475)* as permissões que o cliente que pediu aquele comando tem para o efetuar.

Cada uma destas funções chamadas pelo *solve* vai desencadear um conjunto de alterações às devidas estruturas de dados como por exemplo, quando um jogador entra num *lobby*, o servidor recebe a mensagem, processa-a chamando a função *enterLobby(String username, String address, String lobbyID) (ficheiro ServerThread.java, linhas 212-243*), que verifica se o utilizador tem as permissões necessárias para efetuar tal pedido e caso tenha, introduz aquele utilizador na estrutura de dados do devido *lobby*, enviando uma mensagem a todos os outros jogadores para estes atualizarem o *lobby* e ao novo jogador as informações necessárias para 'desenhar' o *lobby* na UI. Caso o utilizador não tenha as permissões ou o lobby estiver cheio, envia uma mensagem para o cliente a negar o acesso àquele *lobby*.

Cliente

O cliente, como foi dito anteriormente, foi escrito inteiramente em Java, sem alguma *framework* ou *library* externa.

Dito isto, é importante referir que estruturas de dados são guardadas no cliente para tornar possível a implementação da arquitetura desejada. Como um cliente só pode ser utilizado por um utilizador (para jogar com vários utilizadores num só computador, terá que abrir mais clientes), o cliente guarda a informação do utilizador que está *logged in* e, no caso de estar dentro de um *lobby*, guarda também a informação do *lobby* onde se encontra. Deste modo, a representação no UI é feita de um modo mais fácil, por podemos passar as respostas do servidor (em formato de *string*) para um objeto, tornando o código mais legível e mais fácil de trabalhar com.

O cliente, tal como o servidor, possui uma função de leitura do *InputStream* do *socket (ficheiro Client.java, linhas 72-80)* que procede a uma resolução do comando enviado pelo servidor na função *solve(String s) (ficheiro Client.java, linhas 82-133)*. Esta função chama de seguida funções responsáveis pelo tratamento da mensagem enviada pelo servidor, dependendo do comando enviado *(ficheiro Client.java, linhas 135-364)*.

Algumas destas funções chamam funções para imprimir na consola o UI *(ficheiro ClientUI.java)* e outras funções para receber inputs do utilizador *(ficheiro Client.java, linhas 366-446)* que verificam se os inputs introduzidos são válidos (validação com ajuda de *regex*) e/ou seguros para o envio para o servidor, de modo a não haverem 'injeções' no protocolo.

Protocolo

Todas as funções faladas na resolução de comandos do lado do servidor *(ficheiro ServerThread.java, linhas 152-365)* e do lado do cliente *(ficheiro Client.java, linhas 135-364)* invocam chamadas à criação de mensagens que obedecem ao protocolo criado pelo grupo *(ficheiros ServerProtocol e ClientProtocol)*.

Este protocolo, como foi dito na arquitetura, funciona de forma idêntica ao protocolo feito para o primeiro trabalho, onde é enviado um comando a efetuar como primeiro 'argumento', tendo a mensagem formatos diferentes dependendo do comando.

Assuntos Relevantes

Nesta secção do relatório, o grupo descreverá que assuntos de relevância foram tomados em conta durante a criação do projeto e como foram implementados. Iremos abordar a segurança, estabilidade e escalabilidade.

Segurança

A nível de segurança, o grupo implementou ligações entre cliente e servidor utilizando protocolo *SSL* de modo a garantir que nenhum cliente 'pirata' sem o certificado consiga aceder ao servidor, prevenindo também a existência de *eavesdroppers* que tentem aceder às mensagens trocadas entre cliente e servidor.

Do lado do cliente, nós efetuamos, com o uso de *regex (ficheiro Cliente.java, linhas 374 e 393)*, verificações de todos os inputs do utilizador de modo a prevenir, como já foi referido anteriormente, 'injeções' no nosso protocolo.

Do lado do servidor, como já foi referido, sempre que é recebido um comando do cliente, é efetuada uma verificação nas estruturas de dados de modo a verificar se o cliente que pediu determinado comando tem permissões para isso. Pensamos que, embora não devam haver preocupações devido ao tratamento cuidado dos comandos, seria uma segunda verificação útil a ser fazer. Embora cause mais *stress* no servidor, o grupo pensou que a troca de mais segurança por um desempenho pior em graus mínimos valeria a pena.

Estabilidade

A criação de uma *thread* por cada ligação de cliente *(ficheiro Server.java, linhas 54-59)* mostrou-se uma mais valia para a estabilidade do sistema, em testes efetuados com vários clientes a jogar em *lobbies* diferentes num mesmo servidor, o desempenho do sistema manteve-se constante e todas as mensagens foram processadas no mesmo intervalo de tempo, independentemente da quantidade de jogos a serem jogados ao mesmo tempo.

O facto de algumas verificações serem feitas do lado do cliente (abordadas na subsecção de 'Segurança') levou também a uma diminuição do *stress* causado no servidor, acabando por contribuir para a sua estabilidade pois são passadas menos mensagens de um lado para o outro e o servidor não utiliza recursos desnecessários para fazer verificações.

Escalabilidade

Em termos de escalabilidade, o nosso servidor mostrou-se capaz de suportar vários clientes em vários *lobbies* a jogarem ao mesmo tempo, provando que a nossa arquitetura mostrou-se uma mais valia em termos de escalabilidade e estabilidade do sistema, pois sendo o sistema estável, em princípio deverá ser mais escalar.

Como foi abordado na subsecção de 'Estabilidade', a criação de uma *thread* por cliente ligado foi essencial para atingir o nosso objetivo: um sistema que aguentasse com vários jogos de vários clientes ao mesmo tempo, sendo a escalabilidade e estabilidade dependentes do hardware da máquina onde o servidor está hospedado. Mesmo assim, independentemente da máquina onde o sistema fosse hospedado, o processamento mostrou-se praticamente idêntico.

Conclusões

O grupo pensa que atingiu os objetivos que foram traçados para este projeto, saindo com uma boa experiência num tema bastante interessante e que, com certeza, adquirimos conhecimentos que serão úteis para toda a vida profissional.

Quanto ao que o grupo gostava de ter feito mais, sentimos que o nosso projeto beneficiaria da implementação de uma interface fora da linha de comandos, mas que, como não era necessário para este trabalho, o grupo sente que implementamos uma UI o mais completa e simples possível.