Universidade do Minho

Escola de Ciências

Licenciatura em Ciências da Computação

Ano Letivo de 2018/2019

Programação Concorrente

Trabalho Prático

Buracos

Grupo 23

Ricardo Jorge Valadares Machado Carneiro Vieira - A81640

Nelson José Dias Teixeira – A80584

João Ribeiro Imperadeiro – A80908

Índice

Introdução

O objetivo deste trabalho prático, elaborado no âmbito da unidade curricular Programação Concorrente, consiste no desenvolvimento de um mini-jogo onde os utilizadores do mesmo possam interagir, usando uma aplicação cliente com interface gráfica (escrita em *Java*) e intermediada por um servidor (escrito em *Erlang*). Este jogo assemelha-se a um já bastante conhecido, o *Agar.io*. Os avatares dos jogadores movimentam-se num espaço *2D* e, para além disso, estes também interagem entre si e com o ambiente que os rodeia, segundo uma simulação efetuada pelo servidor.

Por forma a cumprir este propósito, é necessário garantir que algumas propriedades e funcionalidades inerentes a este projeto sejam satisfeitas. Entre elas destacam-se o registo do utilizador, a criação de partidas e a inicialização do espaço *2D* do jogo.

De salientar que o cliente com interface gráfica deverá suportar as funcionalidades descritas acima e comunicar com o servidor via *sockets TCP*. Para tal, foi adotado pelos elementos deste grupo a utilização da linguagem *Processing* para a construção da componente gráfica deste projeto, dada a sua simplicidade e comodidade. Quanto ao servidor é armazenado em memória a informação relativa à simulação do jogo tratando-se das conexões, dos *inputs* dos clientes e das alterações a serem enviadas à interface gráfica.

Feita a exposição do tema deste projeto, é apresentada de seguida a abordagem tomada pelo nosso grupo.

Trabalho Prático - Buracos

Apresentação da arquitetura

Após uma leitura cuidadosa do enunciado do trabalho prático, demos início ao desenho da arquitetura do nosso jogo. Para tal, idealizámos tanto a parte do servidor como do cliente.

Servidor

Inicialmente decidimos começar por pensar como iríamos estruturar o servidor e o formato das mensagens que serão usadas para comunicar com o cliente, através de *sockets TCP*, visto que este depende de estas decisões estarem tomadas.

O formato das mensagens que optámos por utilizar, consiste em mensagens orientadas à linha, em que os vários campos estão divididos pelo carater ":".

Assim, o servidor da aplicação será responsável por guardar toda a informação necessária para o jogo executar, incluindo informações relativas ao *login* dos utilizadores e às pontuações máximas atingidas desde o seu início, bem como ter capacidade de gerir várias partidas simultâneas, simulando toda a interação dos elementos. Sempre que o cliente pretenda obter alguma informação para disponibilizar ao utilizador ou, no caso de estar uma partida, enviar as teclas premidas em cada momento e receber o estado da partida, terá de enviar um pedido ao servidor e este, por sua vez, responderá de acordo com o seu estado interno.

Para atingir estes objetivos, decidimos que, inicialmente, iríamos ter 4 processos *Erlang* a correr:

- um para aceitar conexões de clientes;
- um para gerir as partidas;
- um para gerir o *login* dos utilizadores;
- um para gerir as pontuações máximas.

Para além destes processos, sempre que um utilizador se conecta, é criado um processo para lidar com ele e, sempre que é iniciada uma partida, é criado um processo para geri-la e outro para enviar a partida aos utilizadores com uma frequência predefinida (30 vezes por segundo).

Com esta arquitetura, os elementos deste grupo acreditam que será possível satisfazer os requisitos impostos sobre o servidor desta aplicação.

Por convenção dos elementos que compõem este grupo, a porta através da qual os clientes comunicam com o servidor é a 1234. Assim, para iniciar o servidor do jogo, procede-se à seguinte instrução em *Erlang:*

server:server().

Cliente

Sendo o cliente uma janela para dentro do servidor, a lógica que tivemos de implementar esteve totalmente relacionada com a apresentação da informação ao utilizador. De forma a satisfazer os requisitos relativos ao cliente da aplicação, foi implementada uma interface gráfica, na linguagem *Processing*. De forma a aplicar os conhecimentos adquiridos na U.C. de Programação Concorrente, a nossa arquitetura contempla o uso de *Threads*. Na prática, criamos uma *Thread* para gerir a interface e outra para atualizar o estado que o servidor vai enviando, utilizando mediadores de concorrência para evitar erros.

Quanto à interface do jogo, são apresentadas as pontuações atuais dos jogadores que estão a participar, as melhores pontuações desde o

momento em que o servidor foi iniciado, o tempo em contagem crescente (2 minutos de jogo), os jogadores (representados por 2 esferas com bordas de cores diferentes), as esferas comestíveis (esferas verdes) e as esferas venenosas (esferas vermelhas). De notar também a presença de alguns menus simples e intuitivos que facilitam a interação dos utilizadores com a aplicação.

Quanto ao estabelecimento de conexão com o servidor, o cliente comunica com este último através de *sockets TCP*.

Implementação da arquitetura apresentada

Nesta secção será explicada mais detalhadamente a implementação do servidor e do cliente da aplicação. De notar que, durante o desenvolvimento do trabalho prático, devido a termos definido a arquitetura da nossa aplicação antes de iniciarmos, pudemos programar o cliente e o servidor em simultâneo, dividindo as tarefas entre os elementos do grupo.

Servidor

No que diz respeito ao servidor, o código deste foi dividido em 4 ficheiros, que representam as unidades lógicas do mesmo.

Um dos ficheiros, *login.erl*, contém todo o código relativo à gestão do registo dos clientes e ao acesso dos mesmos à aplicação, incluindo o início/término da sessão de um utilizador e a criação/eliminação de uma conta de um cliente.

Outro dos ficheiros mencionados é o *scores.erl*, onde são geridas as 5 melhores pontuações obtidas no jogo (desde a inicialização do servidor).

O 3ª ficheiro criado, *user.erl*, trata de todas as interações com o cliente, incluindo a autenticação do mesmo, o envio do estado da partida atual/de qualquer informação por ele pedida e da troca de mensagens entre uma partida e o cliente por ele gerido.

Por último, o 4ª ficheiro, *server.erl*, interliga todos os ficheiros mencionados acima, permitindo não só a gestão de todas as partidas de jogo ativas e das ligações dos utilizadores, como também a criação das esferas intervenientes em cada partida do jogo. Para além disso, é aqui implementado a maioria das funcionalidades da aplicação bem como algumas funcionalidades auxiliares.

Cliente

Quanto ao cliente, criámos 3 classes *Java* em ambiente *Processing* e 3 ficheiros auxiliares.

As classes são:

- *Player*, que tem toda a informação da esfera de um jogador
- *Blob*, que tem toda a informação de um objeto comestível
- *Estado*, que tem toda a informação relevante à interface, incluindo a informação da partida atual.

Quanto aos ficheiros auxiliares, temos: o *Client.pde,* onde são exibidos todos os menus da interface do jogo; o *Login.pde,* que trata da autenticação do utilizador; o *SocketReader.pde*, que trata da atualização o estado durante a partida.

Voltando ao *Estado,* o acesso a este é feito através dos métodos *get()* e *set()* da classe respectiva, de forma síncrona, garantindo a ausência de *race conditions*.

Funcionalidades extra / bónus

Para além das funcionalidades propostas no enunciado do trabalho prático, o nosso grupo tomou a liberdade de adicionar algumas mais por forma a tornar ainda mais apelativo e interessante o resultado final da aplicação desenvolvida. Apresenta-se de seguida as mesmas:

- ➔ Durante uma partida do jogo existe a possibilidade de mostrar/esconder, através da tecla "*s*", as cinco melhores pontuações obtidas desde o momento em que o servidor foi arrancado, sendo sinalizado a azul a pontuação do jogador que se encontra ativo;
- ➔ Existe a possibilidade de associar a todas as partida do jogo um número fixo e arbitrário de jogadores. Para ativar esta funcionalidade basta invocar a seguinte instrução:

  server:server(n)
- ➔ Durante uma partida de jogo existem esferas alaranjadas que, quando comidas por parte de um jogador, tornam o mesmo invisível durante 10 segundos para os restantes jogadores/inimigos.
- ➔ Durante uma partida de jogo existem esferas azuis claras que, quando comidas por parte de um jogador, tornam o mesmo mais rápido durante 10 segundos.

Conclusão

Com a elaboração deste trabalho prático foi possível consolidar conhecimentos a nível de concorrência de processos e, sobretudo, solidificar o trabalho em ambiente *Erlang* e em *Java*. Ao termos a oportunidade de elaborar este trabalho aplicando as duas linguagens, pudemos trabalhar com duas abordagens muito diferentes em termos de concorrência. Por um lado, em *Java*, a concorrência/comunicação é feita através de memória partilhada, utilizando threads; por outro, em *Erlang*, isto é feito através da troca de mensagens, utilizado processos leves, em que na partilha de informação não há problemas quanto à ocorrência de *race conditions*.