Universidade do Minho

Programação Concorrente

Ciências da Computação

Bruno Fernandes (A95972)

David Agra (A95726)

Marta Pereira (A97402)

Tiago Silva (A97450)

31 de maio de 2024

Índice

1 Introdução **2**

2 Servidor **3**
2.1 Comunicação com o cliente (main.erl) 3
2.2 Login manager (login.erl) 3
2.3 Matchmaking service (matchmaking.erl) 4
2.4 Game manager (game.erl) 4

3 Cliente **5**
3.1 GameObject.java 5
3.2 Util.java 5
3.3 Server.java 5
3.4 Screen.java 6

4 Conclusão **7**

Capítulo 1

Introdução

Este relatório foi elaborado no âmbito do trabalho prático da Unidade Curricular de Programação Concorrente, no qual foi proposta a implementação de um jogo, onde vários jogares conseguem interagir através de um programa cliente (escrito em Java), intermediados por um servidor escrito em Erlang. O jogo ocorre num espaço 2D onde os avatares interagem entre si e com o ambiente que os rodeia. A simulação do jogo é totalmente feita pelo servidor e o cliente apenas comunica os eventos ao servidor e a sua interface gráfica apresenta a informação recebida do mesmo.

Capítulo 2

Servidor

2.1 Comunicação com o cliente (main.erl)

No arranque do servidor criamos um socket na porta (...) para receber pedidos dos clientes. Criamos também dois processos utilizando a função *spawn* do Erlang. Estes processos são registados globalmente com os nomes *matchmaking_service* e *login_service* e chamamos a função *accept* que vai ser responsável por aceitar as conexões dos clientes. Quando chega uma conexão de um cliente, criamos um processo que executa a função *handle_client* e chamamos recursivamente a função *accept* para podermos ter vários clientes conectados.

No estado inicial o servidor pode receber quatro opções de pedidos: topN, que lista o ranking com os N jogadores melhor classificados; login, signup e delete, que têm de ser acompanhados de um username e password. Após a receção destes pedidos, é enviado ao processo do Login o Pid do HandleClient e a informação necessária para completar o pedido do cliente. A resposta será posteriormente enviada de volta para o HandleClient atual, e a mesma resposta é enviada ao cliente. No caso do pedido ter sido um login, o nosso HandleClient, para além da informação do socket, passa também a conter a informação do nível e das vitórias do cliente.

Quando o cliente já estiver com a sessão iniciada pode receber quatro pedidos: join, que é a intenção de se juntar a uma partida; logout, para terminar sessão; top, que é igual ao anterior; leave, para abandonar o lobby. Após um join, se o processo MatchMaking encontrar uma partida, enviar-nos-á o Pid do GameManager, e o HandleClient passa conter também o Pid do GameManager. O processo GameManager faz a simulação do jogo.

Estando dentro de uma partida, as instruções que podemos receber são left, right e up, e enviamos ao GameManager. Se este detetar que o cliente ganhou, perdeu ou houve um empate, isso será comunicado ao LoginService para que este atualize a informação do jogador e também ao cliente. É através deste HandleClient que é enviada a informação do jogo para o cliente.

2.2 Login manager (login.erl)

No início do processo do login manager, carregamos do ficheiro *.data* as informações dos registos. Temos um serviço de backups onde é realizado um backup (atualizado o ficheiro *.data*) a cada 5 segundos. Neste serviço temos a informação guardada em dois dicionários (Data e SessionManager) e um conjunto (Usernames). O dicionário Data tem a informação (chave é o Username e Password e os valores são o nivel e vitórias consecutivas) de todos os registos. O dicionário SessionManager tem como chave um Pid e os valores sao Username e Password, e faz o registo dos utilizador que fizeram um login válido e que ainda não fizeram logout. O conjunto Usernames serve para quando há um registo, verificar se não existe outro jogador com o mesmo username. Havendo a receção de uma mensagem pelo LoginManager, este fará a verificação ou atualizará a informação que está em memória e retornará ao Pid de onde provém a mensagem, a conclusão da operação.

2.3 Matchmaking service (matchmaking.erl)

Este serviço tem como função agrupar jogadores para inicializar uma partida. Para tal, recebe dois padrões de mensagens dos processos cliente, join ou leave, ambos acompanhados do Pid do HandleClient e o nível. Essa informação é colocada numa queue (caso seja um join) ou removida (caso seja um leave) e, quando esta contém jogadores suficientes é inicializado um timer de 5 segundos. Após os 5 segundos o timer enviada mensagem ao processo e, se os jogadores continuarem disponíveis, é iniciada uma partida com as lista dos jogadores a participar nela. O timer pode sofrer um reset caso durante a contagem do tempo chegar mais um jogador que possa entrar na partida.No caso de termos 4 jodores nao é necessário um timer e inicializamos a partida de imediato.

2.4 Game manager (game.erl)

Inicialmente são gerados os jogadores e os planetas. Cada jogador é notificado através da mensagem start que o jogo começou, com uma mensagem GamePid, start, Id para que possam comunicar diretamente com o game manager e enviamos as posições de todos os objetos (através da função buildUpdate gera uma lista com a informação dos planetas e dos jogadores) a cada jogador. O game manager tem um serviço de updates onde a cada 15ms é atualizado o estado do jogo (as posições dos planetas e as posições e ângulos dos jogadores) e enviada a informação a cada jogador. Para além disso, a cada update, verificamos se existem colisões entre jogadores e planetas e, desta forma, sabemos quais jogadores estão em jogo e quais já perderam, que estão separados em duas listas distintas na memória do processo. Também verificamos se existem colisões entre dois jogadores e, nesse caso, têm uma colisão elástica (altera o ângulo e a velocidade de cada jogador). Ao receber o evento de uma tecla, provocamos uma aceleração alterando as velocidades angular (no caso de left ou right) ou linear (no caso de up). Este processo termina se já não houver nenhum jogador vivo, e assim empatam todos ou, se o último jogador vivo sobreviver durante 5 segundos.

Capítulo 3

Cliente

3.1 GameObject.java

É um objeto que contém a informação para desenhar uma figura no ambiente gráfico. Este contém a sua posição, o raio, o ângulo e o identificador. Se o identificador for menor que 0, o objeto é um planeta, se for igual a 0 é o Sol e se for maior que 0 é um jogador.

3.2 Util.java

Esta classe é utilizada para intermediar a comunicação com o servidor. Tem vários átomos de Erlang que são enviados para o server.

Na nossa implementação, cada mensagem tem um header de 4 bytes que especifica o tamanho da mensagem.

Esta classe possui três métodos:

- read: Lê o header para determinar o tamanho da mensagem. Lê os dados em binário (usando o header para saber quantos bytes tem de ler) de um InputStream e retorna um OtpInputStream.
- readTuple: Usa a função read para obter um OtpInputStream a partir do InputStream e constrói um OtpErlangTuple usando o OtpInputStream lido.
- send: Cria um OtpOutputStream e codifica o objeto Erlang nele. Converte o OtpOutputStream num array de bytes. Escreve o header, um byte adicional (de acordo com o protocolo) e os dados codificados no OutputStream.

3.3 Server.java

O Server cria a conexão TCP e tem métodos que utilizam o ficheiro Util.java. Estes métodos são utilizados para enviar átomos ou tuplos Erlang ao servidor. Esta classe tem uma thread do tipo GameData que recebe e guarda a informação do jogo.

No caso do método ser um login, é enviado um tuplo Erlang com login, Username, Password. Esperamos pela resposta e, se a resposta for "ok", guardamos o username, o nível e o número de vitórias, que serão exibidas ao utilizador no menu do jogo.

No caso do método ser um findMatch, criámos uma thread que irá enviar o átomo "join"e irá ficar à espera da resposta. Quando a resposta for o átomo "start", o servidor já nos incluiu numa partida. Consequentemente, acordamos a thread do GameData para que esta possa receber e guardar a informação do jogo.

Os restantes métodos são de pergunta-resposta. Não implicam grandes esperas nem grandes atualizações, por isso não fazem uso de threads.

3.4 Screen.java

A interface gráfica foi implementada através da biblioteca Java Processing. Nesta classe temos um estado para cada ecrã possível no jogo. Para cada estado está definida uma função que desenha o ecrã atual de cada utilizador.

Ao inicializar um Screen é estabelecida a conecção com o servidor. Funciona como se cada Screen fosse um cliente do servidor.

Capítulo 4

Conclusão

Apesar de algumas dificuldades que surgiram ao longo do seu desenvolvimento, como por exemplo, definir a comunicação entre processos em Erlang, o parsing das mensagens no cliente e a interface gráfica em Processing, consideramos que o trabalho foi bem conseguido. Concluímos assim que este trabalho permitiu-nos consolidar os nossos conhecimento de Programação Concorrente.