

Universidade do Minho
Escola de Ciências

Gravidade
Programação Concorrente
Relatório de Desenvolvimento

André Oliveira Barbosa
A91684

Francisco Paulino
A91666

1 de junho de 2024

Conteúdo

1	Introdução	2
2	Cliente	3
2.1	client	3
2.2	state	3
2.3	reader	4
2.4	player	4
2.5	login	4
2.5.1	Interface de login	4
3	Servidor	5
3.1	server	5
3.2	login_manager	5
3.3	state	5
4	Exemplo de um jogo a decorrer	7
5	Conclusão	8

Capítulo 1

Introdução

Área: Programação Concorrente

Este relatório foi elaborado no âmbito do projeto da cadeira de Programação Concorrente, no qual se pretendia criar um mini-jogo onde vários utilizadores pudessem interagir usando uma aplicação cliente com interface gráfica, escrita em Java, intermediados por um servidor escrito em Erlang. O avatar de cada jogador movimenta-se num espaço 2D. Os vários avatares interagem entre si e com o ambiente que os rodeia, segundo uma simulação efectuada pelo servidor.

De uma forma geral, trata-se de um jogo que simula um ambiente espacial, onde os jogadores tentam sobreviver o máximo de tempo possível no Sistema Solar, sem colidir com qualquer planeta ou com o Sol, resistindo à gravidade que os atrai para o Sol, às eventuais colisões com os outros jogadores e ao movimento constante dos planetas. Tudo isto enquanto lida com a perda de combustível nas deslocações.

Capítulo 2

Cliente

O cliente é escrito em java, utilizando o processing para a parte gráfica. Este comunica com o servidor via sockets TCP, permitindo que exista concorrência, fator necessário para este jogo. Para além de lidar com a parte gráfica, o cliente gere as interações do utilizador e atualiza o estado de jogo em resposta às mensagens do servidor.

Decidimos construir o cliente com uma arquitetura modular, de forma a separar responsabilidades entre as várias classes. Esta arquitetura, para além de melhorar a organização do código, que se tornou bastante extenso, permite que este seja facilmente expandido.

De seguida vamos detalhar cada classe e explicar o seu propósito no projeto.

2.1 client

Esta classe é responsável por iniciar a aplicação, estabelecer uma conexão com o servidor e configurar os principais elementos da parte gráfica. A conexão com o servidor é assegurada pelo uso de um objeto *Socket*, para comunicar-se através de TCP. Implementa também o movimento do jogador com as 3 setas do teclado que controlam os 3 propulsores, de forma a existir velocidade angular e velocidade linear. O movimento linear do jogador faz diminuir em 0.05 a quantidade de combustível, enquanto a velocidade angular diminui-a em 0.01.

2.2 state

Esta classe é responsável por gerir o estado atual do jogo, de forma a manter informações sobre os jogadores e os corpos celestes.

É aqui que é desenhada, utilizando o *Processing*, a barra de combustível no canto inferior esquerdo. A barra oferece informação visual ao utilizador sobre o seu próprio combustível por meio de uma barra que é verde quando o combustível é maior que 20% e é vermelha quando é menor, de modo a indicar que o combustível está a acabar. É também mostrada a informação sobre o combustível dos inimigos em percentagem.

2.3 reader

Esta classe, que executa como uma thread separada, lê constantemente as mensagens do servidor, de modo a poder atualizar o estado do jogo. Alguns exemplos de mensagens do servidor são:

- **Comeca** - sinaliza o início do jogo e constroi a configuração inicial do jogo.
- **Vitoria** - Indica que um jogador ganhou a partida.

2.4 player

Tal como o nome indica, esta classe foca-se na representação dos jogadores. Aqui consideramos necessário que os clientes tenham atributos que controlam se o jogador está vivo, a sua posição e combustível. Definimos que o próprio jogador deve-se ver de uma cor diferente dos inimigos, pelo que o próprio user vê-se sempre como azul, enquanto os inimigos estão a laranja.

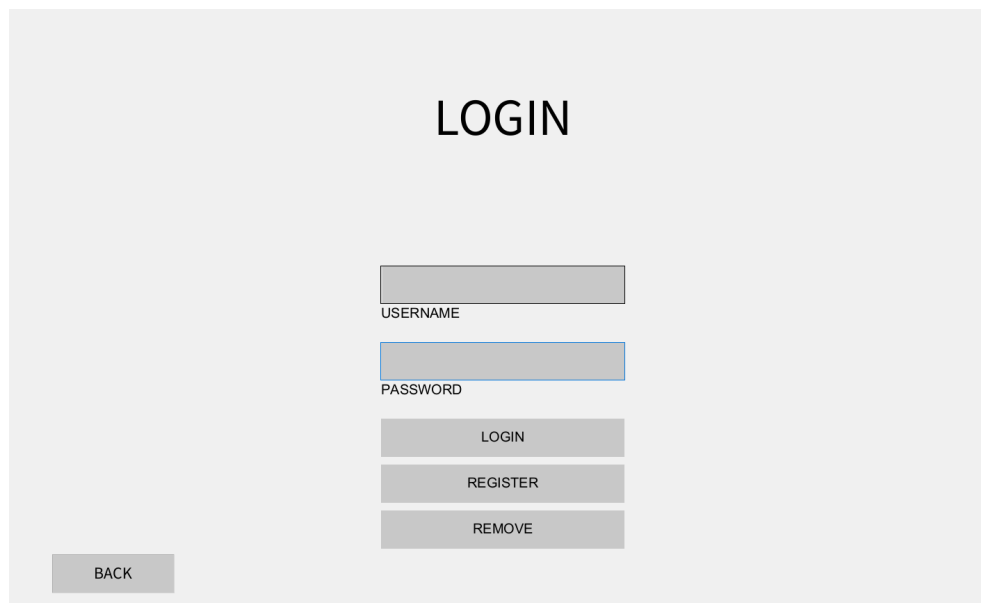
Utilizamos variáveis sincronizadas de modo a evitar conflitos.

2.5 login

Esta classe lida com o registo, login e remoção de contas.

2.5.1 Interface de login

A interface para o login do utilizador é ilustrada na Figura 2.1. O utilizador deve preencher os campos de username e password e depois escolher qual a ação que pretende efetuar. Cada botão é ligado às funcionalidades implementadas na classe *Login*. Após fazer login, o jogador escolhe a sala e entra numa sala de espera. Depois de entrar mais um jogador nessa mesma sala, o jogo começa da forma que é demonstrada na figura 4.1



The diagram illustrates a login interface. At the top center, the word "LOGIN" is displayed in a large, bold, black font. Below this, there are two input fields: the first is labeled "USERNAME" and the second is labeled "PASSWORD". Both labels are in a smaller, black, uppercase font. Below the input fields, there are three buttons stacked vertically: "LOGIN", "REGISTER", and "REMOVE". These buttons are rectangular with a light gray background and black text. To the left of these buttons, there is a separate button labeled "BACK", also rectangular with a light gray background and black text.

Figura 2.1: Menu de login com botões para Login, Registo e Remoção de uma conta.

Capítulo 3

Servidor

Escrito em Erlang, o servidor do nosso jogo é um ponto fundamental para gerir a lógica central, a comunicação entre os jogadores e a manutenção do estado. Este estabelece conexões com o cliente e faz-lhe também chegar informação relevante para a atualização da interface gráfica, de acordo com diferentes cenários, garantindo que todas as interações sejam sincronizadas para todos os jogadores.

O servidor utiliza, tal como o cliente, diferentes módulos para tratar diferentes aspetos do jogo tais como autenticação, gestão de sessões e o controlo de interações dentro do ambiente do jogo.

De seguida, vamos detalhar os diferentes módulos do servidor, de forma a mostrar as suas funções.

3.1 server

Este é o módulo que gere as conexões de rede, dado que ouve de uma porta TCP e aceita as conexões dos clientes. De uma forma geral, ele é responsável por inicializar e coordenar os outros dois componentes do servidor e por rotear mensagens entre os cliente e esses módulos.

3.2 login_manager

Este módulo é usado para gerir todas as operações relacionadas com a autenticação e sessão dos jogadores.

As funções *create_account/2* e *close_acount/2* gerem a criação e exclusão de contas de utilizador, respetivamente. Este mecanismo é desenvolvido adicionando (respetivamente retirando) o nome do utilizador do mapa. Para ambos os casos, tanto o nome como a palavra passe devem estar corretos.

3.3 state

Este é o módulo central na gestão do estado do jogo, já que implementa a sua lógica e coorderna-a com a interação entre jogadores e corpos do jogo.

É aqui que a função *ganhou/2* é chamada quando um jogador vence o jogo, atualizando o estado para declarar a partida como finalizada.

O loop *loop/4* e a função *handle/6* são usados para processar mensagens de forma assíncrona, de forma a manipular diferentes solicitações de forma eficiente.

Ao receber uma solicitação *join_room*, verifica se a sala existe e adiciona-o à sala, ou cria uma se não existir.

Aqui são também definidos o número de corpos celestes, sendo que um deles, o Sol, estará sempre no centro do ecrã com a cor amarela. Os outros corpos (planetas) giram à volta do Sol e têm diferentes tamanhos, cores e velocidades. Todos os corpos ao colidirem com o utilizador fazem imediatamente com que o utilizador perca o jogo. Já os jogadores são inicializados com um tamanho definido, posição inicial aleatória e com o fuel definido como 100.

Os utilizadores estão constantemente a ser atraídos em direção ao Sol, o que aumenta a dificuldade do jogo. Para simular a realidade, a atração do Sol é maior para os jogadores que se encontram mais próximos deste.

A força gravitacional é calculada como:

$$GravityX = \left(\frac{CenterX - X}{Distance} \right) \times \left(\frac{1000}{Distance^2} \right) \times 10$$

$$GravityY = \left(\frac{CenterY - Y}{Distance} \right) \times \left(\frac{1000}{Distance^2} \right) \times 10$$

onde:

- CenterX e CenterY são as coordenadas do centro de gravidade.
- X e Y são as coordenadas atuais do objeto.
- Distance é a distância euclidiana entre o centro e o objeto, calculada por:

$$Distance = \sqrt{(X - CenterX)^2 + (Y - CenterY)^2}$$

Capítulo 4

Exemplo de um jogo a decorrer

De seguida mostramos a figura 4.1 que mostra um exemplo do jogo a decorrer. Neste caso existem dois jogadores. O jogador que fez a captura de ecrã (francisco) tem o combustível a 60%, enquanto o outro utilizador (andre) tem o combustível a 83%. De notar que a barra amarela no utilizador atual indica que o utilizador atual está direcionado para a esquerda.

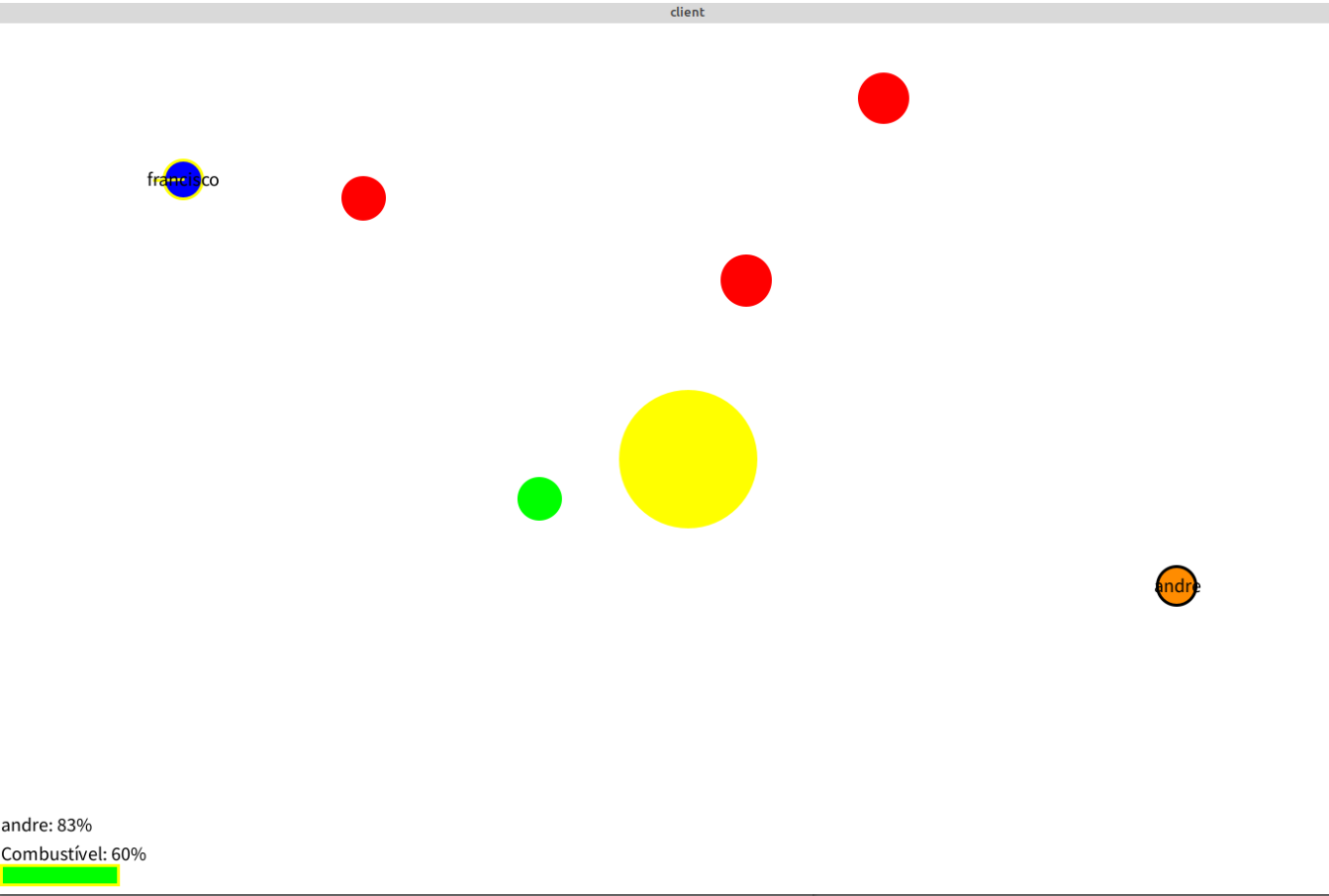


Figura 4.1: Captura de ecrã durante um jogo.

Capítulo 5

Conclusão

Por fim, gostaríamos de realçar tanto os pontos positivos, como os pontos negativos do nosso trabalho, bem como realçar as principais dificuldades que encontramos. Queremos também desde já realçar o facto de considerarmos este projeto bastante positivo para a nossa aprendizagem em Programação Concorrente e em construir um projeto visualmente apelativo com o Processing.

Como pontos positivos, implementamos com sucesso uma grande parte das funcionalidades pedidas no enunciado do projeto, sempre tendo em mente o que foi lecionado na cadeira. As conexões TCP permitem o jogo multiplayer e é também possível ter vários jogos a decorrer em simultâneo.

Pelo lado negativo, o leaderboard não é funcional, e não existem níveis nos jogadores, já que estas funcionalidades não foram implementadas. Isto aconteceu devido a vários fatores, mas principalmente pelo facto de o projeto ser bastante extenso e o tempo muito curto. Assim, focamos as nossas atenções em funcionalidades que consideramos mais importantes e funcionais.

O tempo foi provavelmente o maior entrave no projeto. Já quanto a dificuldades, foi complicado arranjarmos uma fórmula de cálculo da gravidade, fazer a colisão elástica entre jogadores, e também organizar mentalmente as mensagens e a estrutura do código de modo a evitar erros.