UNIVERSIDADE DO MINHO

Gravidade
Programação Concorrente
Grupo nº 5

Catarina Gonçalves	Diogo Costa	João Mirra	Pedro Santos
(A101904)	(A78133)	(A100083)	(A100110)

1 Junho de 2024

Índice

1. Introdução

Na unidade curricular de Programação Concorrente, foi proposto aos alunos o desenvolvimento de um mini jogo no qual vários utilizadores podem interagir utilizando uma aplicação cliente com interface gráfica. No jogo, o avatar de cada jogador movimenta-se num espaço 2D, onde os avatares interagem entre si e com o ambiente circundante, segundo uma simulação efetuada pelo servidor.

Este documento descreve, de forma breve, o trabalho realizado tanto no cliente quanto no servidor para o desenvolvimento deste mini jogo.

2. Servidor

O servidor é responsável por gerir a comunicação entre os clientes e o servidor, oferecendo funcionalidades para iniciar e terminar o servidor, autenticar utilizadores e coordenar as interações entre os jogadores. Uma vez iniciado, o servidor é configurado para ouvir uma determinada porta. Se a inicialização for bem-sucedida, são iniciados os processos necessários para gerir as contas de cada utilizador e as salas de jogo. Em seguida, espera por mensagens para carregar os dados das contas e níveis do jogo, ou para parar o servidor.

O processo de inicialização recebe mensagens com dados carregados das contas de utilizadores e níveis. Se forem bem-sucedidos, os processos responsáveis por gerir as contas de utilizador e as salas do jogo são registados, e um processo é iniciado para lidar com novas conexões de clientes. Se ocorrer algum erro com o carregamento dos dados, é exibida uma mensagem de erro.

Quando uma conexão é estabelecida, um novo processo é iniciado para lidar com ela. Este processo verifica se o utilizador está autenticado. Se sim, aguarda mensagens do

cliente e processa-as de acordo com o contexto do jogo, como criar ou entrar numa sala, alterar configurações da conta ou iniciar o jogo.

Se a conexão com o cliente for perdida ou ocorrer um erro de comunicação, o processo termina e os outros processos relevantes são notificados para lidar com a desconexão ou com o erro de forma adequada.

Os próximos módulos são partes integrantes do servidor. Cada um desses módulos desempenha uma função específica essencial para o funcionamento do servidor como um todo.

2.1. Accounts

Este módulo, escrito em Erlang, foi desenvolvido para atuar como um servidor robusto, gerindo as contas dos utilizadores e respetivos níveis de experiência num ambiente concorrente. Oferece várias funcionalidades, desde a criação e login de contas até à atualização de passwords, remoção de contas, geração de rankings e atualização de níveis.

A função principal deste módulo é iniciar o processo de gestão de contas. Este processo mantém o estado das contas e níveis dos utilizadores e pode responder a diversas mensagens que representam diferentes ações e comandos. Por exemplo, pode guardar o estado das contas, encerrar o servidor preservando os estados e controlar a presença online e offline dos utilizadores.

Uma característica crucial deste módulo é a sua capacidade de lidar com mensagens não reconhecidas de forma adequada. Utilizando o padrão de captura "Other ->", o servidor pode continuar a operar sem interrupções mesmo quando recebe mensagens inesperadas.

2.2. Game_Player

Este módulo desempenha um papel crucial na gestão dos jogadores num ambiente de jogo concorrente. O objetivo principal é controlar o estado individual de cada jogador, como a posição, a velocidade, interações e deteção de colisões. Para isso, o módulo define algumas funcionalidades cruciais para o funcionamento adequado do jogo. O módulo distingue-se pela sua capacidade de receber e processar diversas mensagens, permitindo atualizações em tempo real do estado do jogador. A função ‘*gamePlayer/4’* responde a uma variedade de mensagens que vão desde a atualização do estado do jogador até à deteção e tratamento de colisões com outros objetos no jogo. Além disso, o módulo define constantes importantes que guiam o comportamento dos jogadores, como posições iniciais, limites de velocidade e parâmetros de aceleração. As funções auxiliares ‘*setupPlayerState/2’* e ‘*checkCollision/1’* desempenham papéis fundamentais na configuração inicial do estado do jogador e na verificação contínua de colisões, garantindo uma experiência de jogo fluída.

2.3. Game_sim

O referido módulo atua na gestão do jogo concorrente, controla o estado dos jogadores, interações e deteção de colisões. Começa por definir contantes importantes e exporta a função *‘start/1’* para iniciar o jogo. Esta função configura o ambiente do jogo, inicia os

processos essenciais, como os gestores de planetas e colisões, e começa o ciclo de simulação do jogo.

O ciclo principal de simulação é controlado pela função *'game_sim'*, que recebe e processa mensagens para atualizar o estado dos jogadores, detetar colisões e coordenar outras ações. O módulo é capaz de lidar com várias mensagens, como atualizar o estado dos jogadores, contagem decrescente para o início do jogo, e mensagens de chat entre jogadores.

Além disso, o módulo *'game_sim'* implementa funções auxiliares para gerir o estado dos jogadores, como o *'checkcollision/1'* para verificar colisões, *'send_states/2'* para enviar o estado atualizado de todos os jogadores, e *'countdown/1'* para iniciar a contagem decrescente antes do inicio do jogo.

A gestão dos planetas é realizada pelo processo *'planets_manager'*, que calcula e atualiza o estado dos planetes em cada ciclo de simulação. Ele gera a posição inicial dos planetas de forma aleatória e calcula a próxima posição com base na velocidade e aceleração.

2.4. Lobby

O módulo *Lobby* está presente na gestão do multijogador, onde os participantes podem criar, entrar e sair de salas, além de iniciar o jogo quando estiverem prontos. Este módulo mantém o controlo das salas disponíveis e gere a interação entre jogadores coordenando o início do jogo.

Quando o *Lobby* é iniciado, inicia-se o processo com um conjunto vazio de salas. Estas salas são representadas como um mapa, onde cada chave corresponde ao nome da sala e cada valor é uma tupla contendo o processo de contagem decrescente, o nível de dificuldade e a lista de jogadores na sala.

Os jogadores podem criar uma nova sala ou juntar-se a uma sala já existente, desde que sejam cumpridos os critérios de nível de dificuldade e de capacidade da sala. Quando um jogador entra numa sala, é adicionado à lista de jogadores presentes nessa sala. Se a sala estiver cheia, o jogador é notificado que não pode entrar.

A contagem decrescente é iniciada quando a sala chega a número de jogadores necessários. Assim que a contagem regressiva, termina, o jogo é iniciado.

O *Lobby* lida com desconexões inesperadas de jogadores, removendo-os da sala e notificando os outros jogadores, se necessário. O estado das salas e dos jogadores é atualizado à medida que as interações ocorrem, garantindo que as informações estão sempre atualizadas.

2.5. Offline

Foi criado o módulo *Offline* que controla a funcionalidade de guardar e carregar os dados relativos a contas dos utilizadores e níveis de jogo, permitindo que o jogo funcione mesmo sem conexão à Internet. Ao ser iniciado, o módulo invoca a função '*offline/1'* com o argumento *'false'* indicando que nenhum dado foi guardado ainda. Essa função aguarda mensagens e responde conforme o conteúdo:

Se receber a mensagem *'{load, Pid}'*, lê os ficheiros *accounts.bin* e *levels.bin*, que contém os dados das contas e níveis, respetivamente. Se os ficheiros existirem, os dados

são lidos e enviados de volta ao processo que os solicitou. Caso contrário, dados padrão são criados e enviados.

Se receber a mensagem *'{full_save, File, Accs}'*, guarda os dados das contas do utilizados no ficheiro especificados, tendo o cuidado de os converter antes para formato binário.

Se receber a mensagem *'off'*, verifica se os dados foram guardados. Se sim, o processo termina, caso contrário, continua a aguardar mensagens. Este módulo é fundamental para manter a consistência da experiência de jogo, garantindo assim que o progresso do jogador é preservado mesmo quando está offline.

3. Cliente

3.1. GameState

Esta classe foi projetada para garantir a consistência de dados entre servidor e os clientes, implementado um bloqueio de leitura e escrita. Este mecanismo permite que os dados sejam consultados por várias *threads* ao mesmo tempo para leitura, enquanto apenas uma *thread* de cada vez pode modificar esses dados.

No contexto do jogo, a classe armazena informações essenciais, como a posição dos jogadores, inimigos e planetas, além de estado do cronómetro da contagem decrescente.

O construtor da classe aceita vários parâmetros, incluindo as posições do jogador, informações sobre inimigos e planetas, bem como o estado do cronómetro.

Além disso, a classe *GameState* fornece métodos capazes de alterar o estado do jogo de forma controlada e segura. Por exemplo, *'setCountdown'* é responsável por ativar ou desativar o cronometro.

A classe é uma garantia de consistência e integridade dos dados do jogo, facilitando uma experiência de jogo positiva para os seus utilizadores.

3.2. TCP

A classe TCP estabelece e controla a comunicação TCP entre os clientes e o servidor do jogo. Ela facilita a troca de mensagens entre eles, permitindo a transmissão de dados relevantes para o funcionamento do jogo.

O construtor da classe aceita o endereço do *host* e a porta para estabelecer a conexão TCP com o servidor. Após a inicialização, a classe cria um *'BufferReader'* para ler os dados recebidos do servidor e um 'PrintWriter' para enviar mensagens para o servidor. Além disso, a classe inicializa uma *thread* chamada *'postman'*, que é responsável por receber continuadamente mensagens do servidor em segundo plano.

A classe possível um mapeamento de tarefas (*'taskMap'*) que associa tipo de tarefas a filas de mensagens correspondentes. Isso permite que as mensagens recebidas sejam classificadas e armazenadas em filas específicas com base nos seus tipos.

Quando uma mensagem é recebida, ela é dividida em tipo de tarefa e conteúdo, e então adicionada à fila correspondente no *taskMap*. A fila é sincronizada para garantir o acesso seguro a partir de múltiplas *threads*.

Os métodos *'send'*, *'waiting'* e *'receive'* são utilizados para enviar mensagens para o servidor, aguardar mensagens especificas e receber mensagens do servidor, respetivamente. Todos esses métodos utilizam o *taskMap* para controlar as mensagens e garantir a sincronização entre *threads* quando necessário.

3.3. Worker

A classe *Worke*r apresenta um papel fundamental na atualização do estado do jogo com base nas informações vindas do servidor. A classe *Worker* contém três subclasses: *'PosWorker'*, *'PlanetWorker'*, e *'GameWorker'*, cada uma com uma função especifica no processamento das mensagens recebidas.

'PosWorker' é encarregue de lidar com as atualizações de posição dos inimigos enquanto a *'PlanetWorker'* atualiza as posições dos planetas no jogo. Por outro lado, *'GameWorkers'* tratam de eventos globais do jogo, como, por exemplo, o início ou o fim de uma contagem decrescente.

Cada classe executa as operações de leitura na classe *GameState* utilizado um *'ReadWriteLock'*, garantindo assim a consistência dos dados ao serem consultados por diversas *threads* simultaneamente. Após completar as operações, os *locks* serão desbloqueados, permitindo que outras *threads* possam consultar o estado do jogo.

4. Processing

A interface gráfica do jogo, desenvolvida na plataforma *Processing*, é composta por vários ficheiros que desempenham funções específicas. O ficheiro *Button* é responsável pela criação de botões interativos, enquanto o *Game* controla toda a parte visual do jogo. O *Lobby* gere as interações dos jogadores antes das partidas, enquanto o *Menu* oferece opções como iniciar o jogo e ajustar configurações. Adicionalmente, o *Planet* e o *Player* lidam com a representação visual dos planetas e dos jogadores, respetivamente, enquanto o *Sun* cria a representação do sol. Estes ficheiros trabalham em conjunto para criar uma experiência de jogo envolvente e imersiva.

5. Conclusões

Em resumo, este trabalho proporcionou uma compreensão aprofundada da comunicação cliente-servidor via TCP, bem como do controlo de concorrência em Java. Durante a implementação do servidor, adquirimos um melhor entendimento sobre o funcionamento do sistema de receção e envio de mensagens em Erlang.

O grupo de trabalho considera que o projeto correspondeu às expectativas inicialmente estabelecidas pelo docente da Unidade Curricular.