Uma imagem com captura de ecrã, símbolo, Saturação de cores, Gráficos

Descrição gerada automaticamente

Universidade do Minho

Licenciatura de Ciências da Computação

Trabalho prático da Unidade Curricular Programação Concorrente

Realizado por:

Tiago Adriano Moreira (92046)

José Miguel Barbosa (95088)

Leonardo Lordello Fontes (96308)

Uma imagem com pessoa, T-shirt, T-shirt atlética, Cara humana

Descrição gerada automaticamenteUma imagem com Cara humana, pessoa, retrato, sorrir

Descrição gerada automaticamenteUma imagem com Cara humana, pessoa, lábio, retrato

Descrição gerada automaticamenteUma imagem com Cara humana, pessoa, vestuário, Queixo

Descrição gerada automaticamenteMaria Filipa Rodrigues (97536)

**Introdução**

Neste trabalho prático foi nos proposto o desenvolvimento de um mini-jogo onde se deveria construir uma parte gráfica feita em Java e o servidor da aplicação em Erlang. Um objetivo importante deste trabalho era conseguir que este jogo fosse jogado por mais pessoas, ou seja que varias pessoas pudessem ter impacto nas ações da aplicação.

Neste mini-jogo cada jogador é representado e controla um avatar em forma de círculo. Também deverão existir na arena de jogo vários objetos que aparecem aleatoriamente na arena, sendo que estes fazem ações diferentes. Os verdes e os azuis, dão respetivamente bónus de velocidade angular e de aceleração linear. O objeto vermelho, deverá remover os efeitos dados pelos outros dois. A maneira de pontuar é muito simples, quando se atinge um adversário por trás ganha-se um ponto. Ganha o jogador que no fim do tempo tiver mais pontos e em caso de empate, o jogo continua até alguém pontuar. Um jogador sobe para um nível n, quando vencer 2 \* (n-1) partidas. Ou seja, se alguém se encontra no nível 3, para subir para o nível 4, precisa de ganhar 6 jogos.

**Servidor**

O código correspondente à parte do servidor é a parte escrita em Erlang. O servidor está encarregue de gerir a informação relevante. Deve ser capaz de receber conexões do cliente, de autenticar os jogadores, enviar informação para o cliente de modo que seja atualizada a interface gráfica e de simular corretamente o jogo com base nas ações dos clientes.

O código implementado consiste em alguns módulos, que estão interligados. O primeiro módulo a que nos vamos referir é o modulo “server”, que contém a lógica geral do servidor.

O servidor consegue realizar com sucesso a autenticação dos utilizadores da aplicação. É possível, através da função “handle\_authentication” criar uma conta ou fazer login numa já existente. As informações de login são o nome de utilizador e password e a função é capaz de identificar se o utilizador existe e se a password é a correta. Após esta verificação, se esta for a correta ele transmite isto ao cliente.

Após a autenticação do utilizador estes serão redirecionados para o lobby. O servidor gerencia este lobby através da função “handle\_lobby”. A partir do lobby os utilizadores podem escolher começar uma partida, sendo também possível aceder a uma página de configurações. Esta permite ao utilizador sair da sua conta e voltar ao menu principal ou apagar a conta em si. Para apagar uma conta o server, vai ter de verificar se a palavra-passe que o jogador inseriu está correta e se tal for o caso irá remover as informações relevantes que esta tinha, ou seja, nome, senha e número de vitórias. No lobby também existe uma leaderboard. Esta contém os 10 jogadores com mais vitórias. O servidor está encarregue de ir atualizando a tabela quando um jogador entra no top 10, ou quando um jogador deste top ganha mais jogos. Sendo por isso necessário que o server esteja sempre pronto a atualizar esta informação e de a mandar para o cliente, uma vez que isto pode acontecer a qualquer altura.

No que toca ao matchmaking, o servidor irá aguardar que mais um utilizador do mesmo nível também queira inicializar um jogo e quando tal for o caso este irá enviar informação ao cliente para que este comece sendo este o trabalho da função “handle­\_matchmaking”. Através da função “handle\_match”, o servidor irá inicializar o jogo e coordenar a informação das ações que ambos os jogadores realizam. Agora que estamos a falar do jogo propriamente dito, é uma boa altura para falar do outro módulo importante deste trabalho, o “jogofinal”. Esta parte do código do servidor trata a fundo e com mais detalhe daquilo que se passa dentro de uma partida entre dois jogadores.

Para esta parte do código vamos fazer uma breve explicação daquilo que algumas funções importantes estão encarregues de fazer.

A primeira função dessa parte do código, de nome “consome” recebe mensagens dos jogadores para indicar se houve uma colisão com um objeto. Esta função verifica se o item foi consumido e informa o jogo se for o caso.

A função “items” é a função encarregue de gerar os objetos. Ela vai gerar os 3 tipos de objetos (verde, vermelho e azul) aleatoriamente, a cada 30 segundos.

A função “field” recebe informações sobre a posição dos jogadores na arena de jogo e caso algum dos jogadores a abandonar esta vai notificar o sistema e vai dar por encerrado o jogo.

A função “colideAtras” verifica se ocorreu colisão entre os dois jogadores. Se tal for o caso esta irá verificar se este toque foi por trás. Se um utilizador tiver, de facto, colidido com o adversário por trás a função avisa o sistema de que houve uma ação pontuável, se não a função retorna “não”.

A função “pontos” é a função que vai atribuir os pontos aos jogadores em caso de uma colisão ter sido válida de acordo com a função “colideAtras”.

As funções “dot\_product”, “norme”, “vector”, “colide” e “distancia” são pequenas funções auxiliares que vão ajudar em alguns cálculos.

A função “timer” vai definir o tempo regulamentar do jogo, sendo esse de 2 minutos.

A função “timelimit” está encarregue de gerir o tempo do efeito que o consumo de um objeto trouxe ao utilizador, sendo que irá notificar o jogo quando este estiver quase a expirar.

A função “posIni” está encarregue de gerar, de forma aleatória, as posições dos jogadores. Atua no início de cada jogo e sempre que algum jogador pontua e é necessário recolocá-los.

A funções “direcao”, “translate”, “rotate” e “rotateNeg” são usadas para calcular os movimentos e as rotações que os avatares dos jogadores efetuam.

As funções “velocidade” e “velocidadeNeg” calculam a velocidade linear dos jogadores com base na aceleração linear.

Por último, temos a função “match”, que é a função principal desta parte. Esta função está encarregue de receber mensagens provenientes das ações dos jogadores, como movimentações que eles realizam e colisões que ocorram durante o jogo. Com base nestas mensagens que ela recebe, vai gerir o jogo de acordo com a informação que vai obtendo, mandando mensagens aos jogadores e a outros processos sobre tudo o que deve acontecer, atualizando o estado do jogo consoante tudo isto.

Com esta breve explicação de algumas funções e funcionalidades obtidas através do código escrito no módulo “jogofinal”, voltemos agora ao módulo “server”.

Como referimos no início o servidor tem várias funções e uma delas que referimos agora, é na sua ajuda a atualizar a interface gráfica, principalmente durante uma partida, uma vez que é com base na gestão do servidor que os clientes dos dois jogadores são atualizados.

No fim de cada partida ambos os utilizadores irão regressar ao lobby onde se encontravam antes da partida ter começado. O servidor irá notificar o resultado e irá também atualizar o número de vitórias do jogador que venceu. Como sabemos o servidor mantém informação constante sobre os jogadores registados e quantas vitórias estes têm.

Resumindo, como pretendido, o servidor desenvolvido em Erlang para este trabalho prático é capaz de se conectar com clientes e receber os inputs destes, de autenticar usuários e de gerir os seus dados como nome, password e número de vitórias e também de eliminar estes dados se for necessário e é capaz de gerir as partidas entre jogadores e ir atualizando a interface gráfica, de acordo com a simulação que efetua.

**Cliente**

Falamos agora do cliente da aplicação. O cliente é feito em Java e contém a interface gráfica do jogo. Foi um desafio interessante a criação de uma, já que nunca tínhamos feito um trabalho assim, que necessitasse de uma interface gráfica.

Não faz muito sentido andarmos a explicar como as coisas foram desenhadas, uma vez que não é esse o propósito do projeto. Vamos sim explicar e falar daquilo que foi feito no cliente, que nos permite conectá-lo com o servidor.

A classe “ConnectionManager” é responsável por garantir a concorrência ao nível do cliente. Esta utiliza uma estrutura de dados HashMap para armazenar as mensagens recebidas. Cada tipo é colocado na queue correta sendo que desta forma várias mensagens do mesmo tipo podem ser armazenadas à medida que chegam. Nesta classe temos 3 métodos bastante importantes. O primeiro destes métodos é o “send” que é responsável por enviar uma mensagem ao socket. Ele utiliza o sendLock para garantir que apenas uma thread de cada vez envia uma mensagem. O método “receive” é responsável por receber mensagens. Ele utiliza o messagesLock para conseguir aceder à HashMap. Se não houver a thread entra em espera e passado 10 segundos verifica novamente. O método read é chamado para ler mensagens provenientes do socket e para tal utiliza o readLock. O uso de locks diferentes tem como objetivo aumentar e aprimorar a concorrência do programa, seja no momento da leitura, da escrita ou de acesso ao Hashmap.

A “defineConditions” serve para definir as condições de sincronização utilizadas pelo objeto messagesLock da classe. Através da utilização do método newCondition() vão ser criadas várias condições que podem ser utilizadas para bloquear ou desbloquear as threads, permitindo a sincronização e comunicação adequadas entre diferentes partes do código.

A classe foi projetada para lidar com a concorrência de forma segura, garantindo a consistência e realiza uma manipulação segura das mensagens dentro do ambiente cliente-servidor. O uso de um Hashmap para armazenar as mensagens recebidas ajuda a garantir que nenhuma mensagem seja perdida. Além disso, utilizando locks separados para envio, leitura e acesso às mensagens, a classe permite que várias threads realizem operações simultâneas. Nas várias classes java que definimos e que contêm a parte gráfica, várias delas também contêm código que é uma conexão entre o servidor e o cliente e a ConnectionManager está nelas todas. Por exemplo na Main, realiza-se uma criação de threads de leitura e estas são criadas com recurso à classe ConnectionManager.

**Conclusão**

O servidor, desenvolvido para este trabalho prático em Erlang, foi uma boa oportunidade de testar a nossa capacidade de implementar um sistema de simulação de jogos. Fizemos um código capaz de fazer várias coisas como, receber conexões do cliente, autenticar utilizadores, gerir partidas e de atualizar a interface gráfica dos jogadores. O cliente, desenvolvido em Java, foi uma maneira de pôr à prova a nossa capacidade de interligar o servidor e o cliente e de conseguir lidar corretamente com a concorrência, sendo que também tivemos oportunidade de explorar o design de uma interface gráfica.