Programação Concorrente

Trabalho Prático

Vaga Vermelha

Relatório de Desenvolvimento

Joaquim Oliveira (A76958)

Filipe Miranda (A78992)

André Pereira (A79196)

31 de Maio de 2018

Concepção/Desenho da Resolução

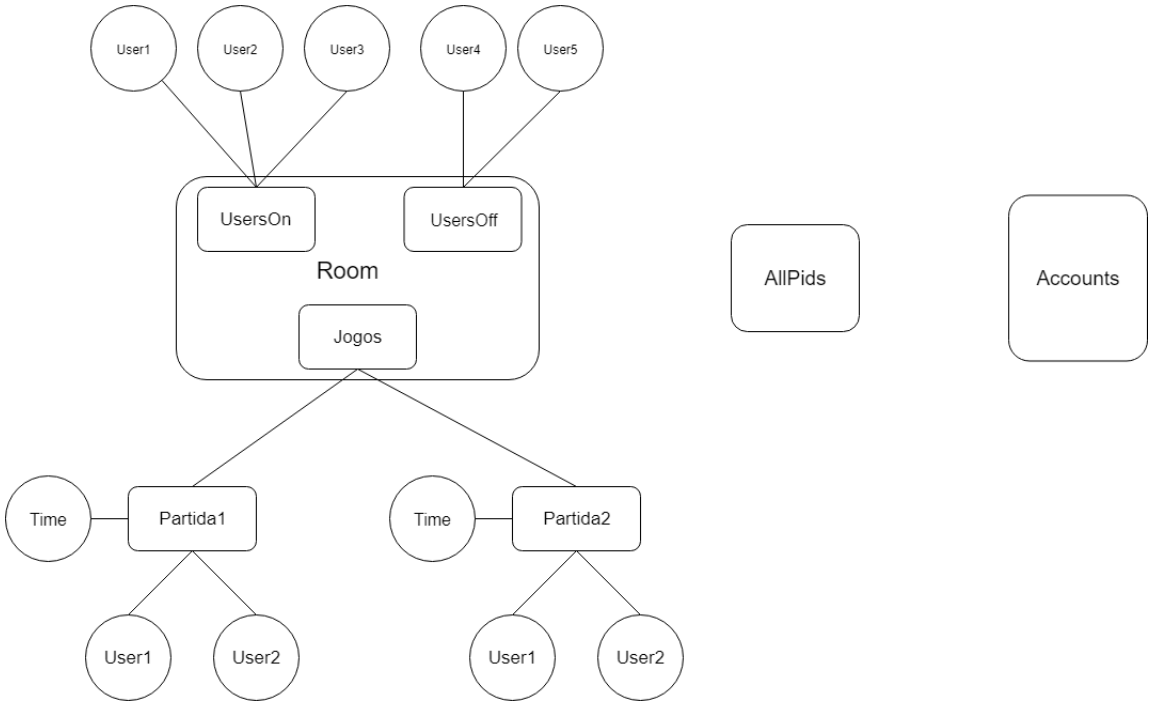

Figura 1: Esquema da concepção do trabalho.

A nossa proposta de resolução para o trabalho prático **Vaga Vermelha** consiste numa aplicação cliente com interface gráfica, escrita em Java, intermediada por um servidor escrito em Erlang. O servidor é constituído por 3 ficheiros erlang (sala.erl, login.erl e motor.erl) e a interface gráfica foi concebida utilizando Processing.

Arquitetura do Servidor

O servidor é composto por um processo principal, denomidado por "Room", que é o responsável por agregar dois tipos de jogadores: os jogadores que fizeram login na sua conta e que não pretendem realizar um jogo de momento e os que fizeram login e tentaram estabelecer uma conexão com outro, ou seja, que pretendem realizar um jogo. Isto facilita a distinção entre os jogadores que pretendem jogar, para que o emparelhamento entre dois jogadores seja mais fácil.

Assim sendo, quando existem dois jogadores disponíveis que sejam compatíveis um com o outro, as condições para uma partida ser realizada estão reunidas, portanto o "Room", cria um novo processo "Partida", que irá ser responsável por gerir toda a informação de um jogo, existindo então vários processos "Partidas", mediante o número de jogos que estão a decorrer.

Cada processo "Partida" é responsável por processar todos os dados relativas ao jogo, comunicando com o "Room"apenas quando o jogo terminada para lhe fornecer os dados importantes relativos ao jogo deccorido, e a pontuaçao, para que seja o "servidor" a guardar toda a informação, visto que este é o processo que estará sempre a ser executado. Este processo, irá tratar do motor do jogo e terá um processo auxiliar que gere o tempo que está a ser decorrido, dizendo ao motor quando é que um novo monstro deve ser introduzido no jogo e o que está a contar a duração do tempo, assim quando o jogo é terminado, este processo termina e diz qual a duração que este teve (o mesmo tempo que o jogo decorreu), atribuindo a pontuação ao jogador que venceu.

Existem dois processos auxiliares, que guardam as informações com maior pormenor, o "allPids" e o Accounts, que trata de guardar todos os dados dos utilizadores (password, username, scores, etc), e os pids que são necessários para que os processos "Room" e "Partida" consigam comunicar corretamente de forma a fazer uma melhor gestão dos jogos. Sendo assim, todo o trabalho computacionalmente mais pesado é distríbuido por "partida"s, não tendo assim o servidor central que processar todos os dados.

Login

A função **initAccounts()** é a função responsável por guardar todas as informações relativas aos utilizadores do jogo (quer existentes ou que ainda vão ser criados) e que invoca a função **accounts(Map, TopPoints)**. A função accounts possui um Map que, a cada utilizador, vai associar uma password, o nível do jogo em que se encontra e as vitórias que possui, e também possui as 5 melhores pontuações que já foram obtidas no jogo Vaga Vermelha.

Ao efetuar login, um utilizador tem que fornecer o seu username e a password. Após introduzir estes dados, averigua-se se o utilizador existe e em caso afirmativo, manda um pid "enter", referindo que se quer entrar na sala de espera. Ao entrar na sala é criado um pid que fica associado a um utilizador, de maneira a que se possa facultar as comunicações via TCP. Caso contrário, dá erro e a função **accounts** é chamada recursivamente. Da mesma forma que se pode fazer login, também se pode fazer logout, que consiste em remover o utilizador do Map da função **accounts**.

Para além do login, também é possível registar um novo utilizador. Ao introduzir os dados, é averiguado se já existe um utilizador com o mesmo nome e, caso exista, é enviado um pid a notificar o utilizador dessa ocorrência. Caso contrário, é enviado um pid confirmando o registo e o Map é atualizado com as informações relativas a este novo utilizador.

Aqui também se incorpora o atomo getInfo que, dado um utilizador, é possível obter as informações a que lhe dizem respeito.

O atomo registVictory corresponde em registar uma vitória. Associa-a ao respetivo jogador, atualiza o nível em que se encontra, adiciona os pontos que obteve nesse jogo à lista dos seus pontos e atualiza a lista das pontuações mais elevadas do jogo, caso a pontuação obtida na partida se encontre entre as 5 melhores do jogo. O Map atualiza as informações dos jogadores consoante o resultado da partida e, posteriormente, a função **accounts** é chamada recursivamente.

Para além das funções e atomos mencionados previamente, fazemos referência que neste ficheiro se encontram alguma funções auxiliares que são invocadas ocasionalmente, tais como adicionar um valor a uma lista, ou um par a uma lista, determinar qual é o valor mínimo de uma lista, etc.

Sala

Este ficheiro é onde se encontram as componentes do TCP e é constituído, essencialmente, por processos do jogo.

Começando pela função **init()**: é escolhida uma porta ("12345", por exemplo), é invocado o **initAccounts()**, cria-se uma sala com o respetivo pid e são registados os pids. Ao abrir uma porta é invocada uma função **acceptor(LSock)**, que se encontra bloqueada até que um novo utilizador emparelhe na mesma porta. Este processo entra em loop: quando um utilizador entra, ele lança outro processo, estando sempre à espera que alguém emparelhe com ele. Ao emparelhar, faz login e transforma-se num utilizador. Basicamente, sempre que alguém emparelhar, esse alguém transforma-se

num utilizador. Ao emparelhar, é obtido um socket, ou seja, já é possível haver comunicação entre dois utilizadores.

As mensagens são recebidas via socket no processo **makeLogin(Sock)**, que se encontra em login.erl. Vai-se partir mensagens, ver o seu conteúdo, etc. Por exemplo, se a string recebida for ”create”, a seguir terá que vir o nome do utilizador e a password. No caso de a string ser para criar uma conta, é feito login a seguir. Enquanto não for feito login, o processo é invocado recursivamente. Quando se faz login, vai mandar um atomo ”enter”para a sala, referindo que se quer entrar na sala. A sala corresponde ao local onde os utilizadores estão à espera para jogar, bem como as pessoas que fizeram login. Ao entrar numa sala, é associado um pid ao utilizador e é através deste pid que ele será identificado.

A sala tem um pid de jogos e os pids dos jogos estão sempre a correr. Existem dois estados na sala: quando se está à procura de jogo e quando não se está.

A partida tem um estado, que é o estado do jogo, dois pids, onde cada um representa um jogador e o tempo, que também é um pid. Ao receber uma line, vê de quem recebe através do pid. São recebidos comandos (frente, tras, direita, esquerda) e interessa saber quem faz esses movimentos, se é o jogador 1 ou o jogador 2. Ao receber uma linha quer dizer que vai mudar o estado. São efetuadas as contas que tem a fazer e calcula-se o novo estado (consoante o pid do jogador).

Quando um jogador sai, normalmente quando sai é por close_tcp. Quando tal acontece, o outro jogador é mandado para a sala, afirmando-o que venceu a partida.

O processo createMonster é o processo que de 10 em 10 segundos, manda um monstro para o jogo. Os monstros verdes têm uma trajetória aleatória e, a cada 3 segundos, a sua trajetória é recalculada.

De 15 em 15 milisegundos, é calculado um novo estado. São calculadas novas trajetórias de maneira a que o jogo prossiga e o estado é convertido para uma string. Aqui são retornados 3 parâmetros: o novo estado, a colisão 1 e a colisão 2 (flags), de maneira a averiguar se algum dos jogadores perdeu, isto é, se saiu fora do cenário ou colidiu com algum dos monstros vermelhos. Caso tal tenha ocorrido com algum dos jogadores, ambos são enviados para a sala, a partida termina, são mandadas as respetivas mensagens e são atualizados as informações dos jogadores. Caso contrário, a partida continua.

Motor

Cada processo ”Partida”, que é o responsável por computar um jogo, como referido na secção anterior. Possui uma estrutura de dados, que denominamos por estado, que guarda as posições das duas naves existentes na partida e as posições de todos os monstros. Este estado é convertido para string, para que possa enviar para o cliente Java, sendo este capaz de o reconhecer, tratar dos dados e renderizar o jogo, permitindo assim que o jogo possa ser demonstrado graficamente.