Universidade do Minho
Licenciatura em Ciências da Computação

Programação Concorrente
Trabalho Prático
Nova Arena

Hugo Costeira	**João Diogo**	**João Goulart**
A87976	A87939	A82643

Junho 2023

Índice

Capítulo 1

Introdução

Este trabalho tem como objetivo a implementação de um mini-jogo - Nova Arena - onde vários utilizadores podem 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. Este relatório abordará a implementação do mini-jogo.

Capítulo 2

Cliente

O cliente quando é executado instancia um objeto *Espaco*, um objeto *Mouse*, um objeto *Data*, um objeto *Keyboard* e um objeto *ConnectionManager* e cria dois processos, o *Handler* e o *Screen* que vão correr concorrentemente partilhando os *Mouse*, *Espaco*,*Keyboard* e *Data*.

2.1 Espaco

Responsável por armazenar a informação sobre as peças.

2.2 Data

Possui um *ReentrantLock* e duas condições e efetua a comunicação entre o Handler e o Screen. Também é aqui armazenada a informação do cliente (*username* e *password*), o estado da aplicação, o tipo de resposta enviado pelo servidor e a informação sobre a leaderBoard que é atualizada ao longo do jogo.

2.3 Exceptions

Neste processo ocorre o tratamento de erros.

2.4 Keyboard

Responsável pelas ações do teclado.

2.5 Mouse

Responsável pelas ações do rato.

2.6 Piece

Responsável pela informação de uma peça (posição e cor).

2.7 Handler

Possui os objetos ConnectionManager, Mouse, Board e Data e funciona como um intermediário entre o servidor e o Screen.

2.8 Screen

Neste processo é realizada a interface gráfica através da biblioteca de *Java Processing*.

2.9 ConnectionManager

O ConnectionManager é utilizado para comunicar com o servidor.

2.10 Tuple

Este processo é utilizado para guardar dois itens numa variável.

Capítulo 3

Servidor

O servidor quando é inicializado regista um processo principal com o nome do módulo e cria um *socket* de *ConnectionManager* e dois processos, o *acceptor* e o *party*. O servidor vai também ler o ficheiro de registos onde está armazenada a informação dos clientes já registados e guarda a informação num mapa.

3.1 Accounts

O processo *acceptor* utiliza o *socket* de *ConnectionManager* criado pelo servidor para fazer o primeiro contacto com o cliente criando um novo *socket* para o efeito de comunicação com o mesmo. Por cada cliente que chega, o último *acceptor* criado é responsável por criar um novo processo *acceptor*. Este processo vai passar para um processo *client* que recebe os pedidos do cliente e os comunica ao servidor. Caso seja feito um pedido de participação num jogo por parte do cliente, o processo *client* comunica esta informação ao processo *party*. Resumindo, pedidos "burocráticos"são comunicados ao servidor e pedidos de jogo são comunicados à *party*.

3.2 Jogos

O processo *party* tem uma fila de clientes à espera que inicie um jogo. Se o comprimento da fila for 2 ou se receber um *timeout* o processo passa a ser o processo *game* com os jogadores na fila e envia uma mensagem ao servidor que vai criar uma nova *party*. No máximo existem 2 processos *party* a decorrer em simultâneo, sendo que é apenas um criado, não recebendo qualquer informação até que o restante termine.

O processo *game* avisa o servidor e os jogadores que o jogo vai iniciar e inicializa cada jogador, isto é, posições, cores iniciais e velocidades. De seguida, transforma-se no processo *gameTimer*

3.3 Funcionamento jogo

O processo *gameTimer* inicializa um *timer* de *tickrate* de 40 milisegundos que é o tempo que cada jogador tem para comunicar os seus movimentos. De seguida, lida com as colisões entre jogadores e entre jogadores e cristais, através da função auxiliar `handleGame`, gera novos cristais com probabilidade de 1% a cada *tick* através da função `generateCrystals` e envia a informação dos jogadores e dos cristais a cada cliente. Por fim, passa para a função

`gameLoop` que recebe a informação dos movimentos dos jogadores. Caso receba o *timeout*, regressa para a função `gameTimer`, ou remove um determinado jogador caso receba um *leave* desse jogador.

Capítulo 4

Conclusão

Durante a execução deste trabalho, aplicamos os conhecimentos adquiridos nas aulas sobre Programação Concorrente em memória compartilhada. Utilizamos abordagens clássicas baseadas em monitores, bem como a concorrência em sistemas distribuídos por meio da troca de mensagens. Dessa maneira, aprofundamos e fortalecemos o nosso entendimento desses conceitos. Acreditamos que alcançamos os objetivos esperados e sentimo-nos mais experientes e preparados em relação à modelagem de sistemas concorrentes e à escrita de aplicações concorrentes em memória compartilhada, com base na passagem de mensagens. No entanto, o planejamento inadequado da solução resultou em uma complexidade maior do que o desejado. Além disso, a falta de familiaridade com a linguagem Erlang também causou algum atraso no progresso do trabalho.Contudo, não conseguimos implementar o limite de espaço de jogo e emparelhar os jogos por níveis. A elaboração deste projeto também nos proporcionou um melhor conhecimento da ferramenta "Processing", uma biblioteca de animação de baixo nível e interface gráfica desenvolvida em Java. Por fim, podemos afirmar que este trabalho estabeleceu uma base sólida para aproveitarmos melhor o que foi ensinado ao longo desta disciplina no futuro.