Battle Royale
Unidade Curricular de Programação Concorrente

Licenciatura em Ciências da Computação
Universidade do Minho

Bruno Jardim (A91680) | Inês Presa (A90355) | Tiago Carriço (A91695) | Tiago Leite (A91693)

19 de junho de 2022

Índice

Capítulo 1

Introdução

No âmbito da unidade curricular de Programação Concorrente foi proposta pelo docente Paulo Sérgio Soares Almeida a implementação de um mini-jogo denominado *Battle Royale*, com o intuito de consolidar a aprendizagem dos dois paradigmas abordados na UC, consistindo numa aplicação distribuída com cliente e servidor.

Assim, o mini-jogo deve permitir que vários utilizadores interajam utilizando uma aplicação cliente com interface gráfica, escrita em *Java*, através da biblioteca *Processing*, 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 efetuada pelo servidor.

Neste documento será apresentada a solução do problema, descrevendo de forma sucinta as decisões tomadas na implementação da aplicação cliente e do servidor.

Capítulo 2

Cliente

O cliente quando é executado instancia um objeto *Board*, um objeto *Mouse*, um objeto *Data* e um objeto *TCP* e cria dois processos, o *Postman* e o *Screen* que vão correr concorrentemente partilhando os *Mouse*, *Board* e *Data*.

2.1 Postman

O *Postman* possui os objetos *TCP*, *Mouse*, *Board* e *Data* e funciona como um intermediário entre o servidor e o *Screen*.

2.2 Screen

O *Screen* possui os objetos *Mouse*, *Board* e *Data* e responsável por receber os *inputs* do utilizador.

2.3 TCP

O *TCP* é utilizado pelo *Postman* para comunicar com o servidor.

2.4 Data

O *Data* possui um *ReentrantLock* e duas condições, e é utilizado para fazer a comunicação entre o *Postman* e o *Screen*. No *Data* é também 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 vai sendo atualizada ao longo do jogo.

2.5 Mouse e Board

A troca de informação entre a posição do rato e posição dos jogadores e cristais é realizada através do *Mouse* e *Board* sendo que estes dois objetos possuem os seus métodos como *synchronized*. O *Screen* lê a posição do rato e guarda no *Mouse* que depois é utilizado pelo *Postman* para enviar ao servidor. O servidor envia a informação dos jogadores e dos cristais para o *Postman* e este armazena esta informação no *Board*.

2.6 Interface Gráfica

A interface gráfica é realizada no processo *Screen* através da biblioteca de *Java Processing*.

Capítulo 3

Servidor

O servidor quando é inicializado regista um processo principal com o nome do módulo e cria um *socket* de *TCP* 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 Gerir clientes

O processo *acceptor* utiliza o *socket* de *TCP* 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 Gerir *parties*

O processo *party* tem uma fila de clientes à espera que inicie um jogo. Se o comprimento da fila for 8 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 5 processos *party* a decorrer em simultâneo, sendo que o quinto é apenas criado, não recebendo qualquer informação até que um dos quatro restantes termine.

Se o número de jogadores na fila for 3, é instanciado um processo que envia um *timeout* ao fim de 10 segundos. Caso um cliente tenha desistido de jogar antes da partida iniciar, é recebida uma mensagem de desistência. Se porventura com essa desistência o número de jogadores na fila ficar inferior a 3, o processo fica á espera até consumir o *timeout* que tinha sido inicializado, continuado com os jogadores que estavam na fila.

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

3.3 Funcionamento do 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.

3.4 Gerir Física

Para o cálculo das colisões entre jogadores e entre jogadores e cristais, existe um módulo auxiliar denominado *geometry* que utiliza a área do círculo para verificar se dois círculos se intersetam e se um dos quatro vértices de um losango se encontra dentro do círculo.

Caso dois jogadores colidam, é calculado um vetor que parte do centro de um para o centro do outro e é aplicado ao movimento de ambos os círculos em sentido oposto.

Capítulo 4

Conclusão

Durante a realização deste trabalho aplicamos todos os conhecimentos adquiridos durante as aulas sobre programação concorrente em memória partilhada, com a abordagem clássica baseada em monitores, e a concorrência em sistemas distribuídos via troca de mensagens, com ênfase no modelo dos atores e dessa forma aprofundamos e consolidamos esses conhecimentos.

Consideramos que conseguimos alcançar os objetivos esperados, sendo que nos sentimos mais experientes e prontos no que toca à modelação de sistemas concorrentes e escrita de aplicações concorrentes em memória partilhada baseadas em passagem de mensagens.

No entanto, o mau planeamento da solução levou a um aumento da complexidade superior ao desejável. O facto de não estarmos muito habituados a utilizar o Erlang também atrasou um pouco o andamento do trabalho.

A elaboração deste projeto levou-nos também a conhecer melhor a ferramenta `Processing`, uma biblioteca de animação "low level"e Interface Gráfica construída em Java.

Por fim, podemos garantir que este trabalho fundamentou as nossas bases para que futuramente consigamos tirar um maior proveito do que foi lecionado durante toda esta unidade curricular.