Programação Concorrente

Trabalho Prático

Battle Royale

Trabalho realizado por:

David Matos A87997

Fernando Lobo A87988

Filipe Azevedo A87969

Jorge Freitas A87944

Resumo

O seguinte documento relata o desenvolvimento de um jogo *multiplayer* baseado em troca de mensagens TCP/IP através de sockets.

O projeto consiste na criação de um cliente em Java, recorrendo ao Processing para desenvolver uma interface gráfica, e de um Servidor baseado em Erlang.

Conteúdo

Conteúdo

Capítulo 1

Introdução

Foi-nos proposto o desenvolvimento de um jogo, *Battle Royale*. Cada partida é constituída por um mínimo de 3 jogadores e um máximo de 8. Caso não haja o máximo de jogadores para começar a partida, esta é inicializada passado um determinado período de tempo.

Uma partida é finalizada quando resta apenas 1 jogador sobrevivente, fazendo com que todos os jogadores que são eliminados voltem novamente para o menu, onde se podem juntar a uma *queue,* para entrar noutra partida.

O espaço do jogo consiste num retângulo limitado por bordas, onde são gerados cristais, com posições e cores aleatórias, das quais as cores variam entre vermelho, verde e azul.

Quando um jogador colide com um desses cristais, obtém uma parte da massa do cristal, tal como, a cor associada a esse cristal.

Quando ocorre uma colisão entre dois jogadores, entra em ação uma regra de cores (vermelho ganha a verde, verde ganha a azul, azul ganha a vermelho). O jogador que vencer segundo a regra de cores adquire parte da massa do jogador perdedor, se ambos os jogadores tiverem a mesma cor, ganha o jogador que tiver maior massa. Se os dois tiverem a mesma cor e a mesma massa, ambos perdem uma parte das suas massas. Um jogador é eliminado caso atinja uma massa mínima.

Um jogador movimenta-se em direção à posição do rato. Quanto mais próximo do rato, menor a sua velocidade. Este possui uma velocidade máxima que pode aumentar caso o jogador prima no botão esquerdo do rato. Quanto maior a massa do jogador, menor a sua velocidade base.

Quando um jogador se encontra no menu, consegue criar, eliminar, fazer login e logout de contas.

Capítulo 2

Física do Jogo

2.1 Modelação Base

Começamos por modelar o estado do jogo com um par Erlang, constituído pelos jogadores (map que contem toda a informação dos mesmos) e os cristais (outra map onde cada chave corresponde a um valor, com a sua posição, cor e massa). Quando um jogo é inicializado, é gerado um mapa aleatoriamente com cristais centrados em posições e cores aleatórias.

A informação toda sobre os jogadores é guardada numa map da qual as suas chaves correspondem ao número dos processos dos jogadores e os valores são representados da seguinte forma - {Username, Mouse, Color, Mass, Pos, Max_Vel}.

O servidor calcula o novo estado do jogo após receber uma mensagem proveniente de um processo que coordena os ticks do servidor e atualiza o estado, enviando-o para cada um dos jogadores. Para além disso, o servidor também recebe mensagens de processos *Cliente* com a informação da posição do rato, a qual é usada para atualizar a informação relativa à posição do rato, que será usada no cálculo da velocidade do respetivo jogador.

Para além disso, também é notificado por um processo que tem como objetivo atualizar o número de cristais presentes no mapa, não deixando este ultrapassar um limite máximo.

2.2 Movimentação

A movimentação de um jogador consiste apenas na multiplicação de um valor fixo relativamente baixo (0.005) pela distância entre o jogador e a posição do rato. Caso esta nova velocidade exceda a velocidade base desse jogador, é lhe atribuída a nova posição com base na sua velocidade base.

Outro aspeto importante é a interação entre dois jogadores quando colidem. Neste caso, a velocidade dos dois jogadores é o simétrico das respetivas velocidades, que para dar um efeito de elasticidade, vai-lhes sendo somado um valor no sentido oposto a estas novas velocidades.

O *boost* segue uma lógica semelhante à das colisões entre jogadores. Neste caso, em vez de obtermos o simétrico da velocidade, é multiplicado um valor que permita que a sua velocidade passe da sua velocidade base. Esta vai reduzindo até à velocidade máxima do respetivo jogador, dando assim, uma sensação de propulsão.

2.3 Cristais

O jogo é iniciado com um total de 10 cristais distribuídos aleatoriamente e com cores aleatórias dentro das três referidas anteriormente. Tal como referido, é usado um processo que é acordado de dez em dez segundos para avisar a partida que pretende adicionar novos cristais. Isto só é possível caso haja menos que dez cristais em campo.

2.4 Colisões

Para detetar a ocorrência de colisões entre jogadores e cristais, recorre-se à map que possui todos os cristais e todos os jogadores presentes na partida. Esta colisão é calculada recorrendo à distância entre um jogador e o cristal que pretendemos avaliar. Se esta distância for menor que a soma dos raios do cristal e do jogador, significa que ocorreu uma colisão, e é assim atualizado o jogador com a informação proveniente do cristal.

No caso da colisão entre jogadores, é da mesma maneira calculada a distância entre os dois e feita a comparação com a soma dos raios dos jogadores, seguindo uma lógica semelhante à dos cristais.

Após a deteção da colisão dos jogadores, são feitas as avaliações seguindo as regras das cores e das massas referidas anteriormente, onde são recalculadas as variáveis relativas, quer ao jogador vencedor, quer ao perdedor dessa colisão.

De seguida, apresenta-se a avaliação entre a colisão entre dois jogadores seguindo a regra das cores e das massas, respetivamente.

Capítulo 3

Servidor

O servidor é constituído pelos seguintes processos:

- *login* manager para gerir os registos da ”base de dados”;
- *cliente* que oferece uma interface com o *queue_manager*, *login_manager* e *partida*;
- *queue_manager* que gere os acessos ao jogo, como filas de espera e jogadores ativos na partida;
- *partida* que calcula o estado do jogo e atualiza os jogadores com o novo estado, assim como recebe mensagens diretas do processo cliente;
- Outros processos que servem como controlo dos processos referidos anteriormente.

Quando um cliente se conecta, assumindo que o servidor está operacional, é estabelecida uma conexão via *socket* TCP/IP. De seguida, é iniciado um processo que armazena essa conexão passando para um novo estado que irá permitir a troca de mensagens entre o mesmo e o *login_manager*.

Iniciada a sessão, o jogador tem a possibilidade de entrar numa partida, a qual irá ser inicializada após haver um número suficiente de jogadores dentro do processo *queue_manager*. Este processo cria uma nova partida e é assim estabelecida a conexão entre todos os jogadores e a mesma. Neste momento, cada jogador encontra-se num novo estado, *client,* onde serão realizadas as trocas de mensagens necessárias para atualizar a informação do respetivo *client* na partida que está a decorrer.

Capítulo 4

Cliente

Recorrendo ao processing, é possível criar uma interface gráfica, que permita de forma mais acessível, obter a informação necessária, do utilizador, que será enviada para o servidor.

O aspeto mais importante desta interface são as duas threads que estão permanentemente a trocar e a enviar mensagens com a informação referida.

Só é enviada uma mensagem após se ter obtido a devida informação que se pretende enviar para o servidor.

Para isso, é usada uma barreira que se mantém bloqueada enquanto essa informação ainda não foi obtida e se liberta após receber uma notificação de que a informação está pronta para enviar.

Relativamente à thread de leitura do socket, esta encontra-se permanentemente a escutar os dados provenientes do socket, que serão usados posteriormente para atualizar a informação presente no ecrã do utilizador.

Estas threads são inicializadas no momento da criação do Cliente, sendo esta uma classe principal com a informação necessária para representar a simulação do jogo no ecrã. Recorreu-se ao uso destas threads para atualizar os dados presentes em memória partilhada.

Enquanto um utilizador está a jogar, é recebido o próximo estado a ser desenhado no lado do Cliente, atualizando as variáveis de instância do jogo, estado este, com formato JSON.

Capítulo 5

Conclusão

A criação deste jogo *multiplayer* ajudou a compreender melhor as conexões estabelecidas entre cliente-servidor e as dificuldades associadas a estas.

Inicialmente, o grupo sentiu uma dificuldade relativamente à parte de concorrência do lado do Cliente em Java, mas estas foram rapidamente ultrapassadas.

Gostaríamos também de poder ter tido a possibilidade de uma melhor organização do código Erlang em diferentes módulos.

Relativamente ao servidor, a manipulação da informação e da lógica de jogo causaram bastantes problemas devido a trocas de mensagens serem realizadas entre processos. Após nos sentirmos mais familiarizados com a estrutura e organização dos processos, foi possível a conclusão total do trabalho.