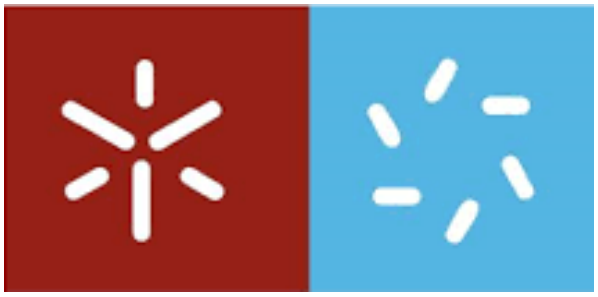

Duelo

Alexis Correia A102495 LCC
João Fonseca A102512 LCC
Moisés Ferreira A97020 LCC
Nuno Machado A68702 LCC
Grupo 6

17 de maio de 2025

Conteúdo

Resumo

Será implementado um mini-jogo onde vários utilizadores poderão interagir através de uma aplicação cliente com interface gráfica, escrita em Java, comunicando com um servidor desenvolvido em Erlang. O avatar de cada jogador movimenta-se num espaço 2D, podendo interagir com o adversário e com o ambiente, segundo uma simulação efetuada pelo servidor. A comunicação será feita via sockets TCP usando a biblioteca Processing. O servidor manterá em memória a informação relevante para a simulação, receção de comandos e envio de dados de atualização para os clientes.

1 Introdução

No âmbito da unidade curricular de Programação Concorrente do curso de Licenciatura em Ciências da Computação, foi-nos proposto como trabalho prático a implementação de um mini-jogo onde vários utilizadores podem interagir através de uma aplicação cliente com interface gráfica, escrita em Java, comunicando com um servidor escrito em Erlang.
Os utilizadores podem registar-se, autenticar-se, jogar e remover a conta.
Após autenticação, acedem a um menu com opções como: procurar jogo, ver ranking, logout e remoção da conta.
O ranking exibe os 10 melhores jogadores, ordenados por nível e número de vitórias consecutivas.
O jogo ocorre num retângulo 2D com colisões nas bordas. Os avatares (circulares) podem disparar projéteis, ganhando pontos ao atingir o adversário.
Colisões com paredes atribuem 2 pontos ao oponente, enquanto disparos certeiros valem 1 ponto.
Em caso de empate, a partida é anulada. O vencedor é determinado por quem tiver mais pontos ao fim de 2 minutos.
Teremos ainda modificadores com várias cores que surgirão aleatoriamente e momentaneamente no campo de jogo que poderão ser colhidos pelos avatares que lhe proporcionará boosts de tempo limitado. Cada cor específica estará diretamente ligada a um tipo de boost específico.
Poderão existir vários jogos a decorrer em simultâneo.

2 Cliente

Interface gráfica onde os utilizadores se registam, fazem login e participam nos jogos.

Estados de interface: cada ecrã é representado por um enum (LOGIN, MENU, GAME, etc). Flags como *readyToStartGame* e *gameEnded* início do jogo e fim de jogo
Thread Listener: Uma thread separada escuta continuamente mensagens do servidor sem bloquear a interface.
Classes principais: `Client.java` (comunicação), `Data.java` (estado), `Display.pde` (renderização e controlo).
Pontuação: a cada colisão com a parede (detectada por comparação com os limites do ecrã) são atribuídos 2 pontos ao adversário. Disparos bem-sucedidos valem 1 ponto.
Concorrência: Várias funções partilham dados do objeto `data`. Para evitar condições de corrida, usamos blocos `synchronized` em secções críticas, garantindo exclusão mútua e consistência dos dados.

Algumas funções importantes:

- `sendPosition()`: Envia a posição do avatar ao servidor.
- `receiveMessage()`: Processa mensagens recebidas como `"opponent_position#x,y"`, `start_game`, entre outras.
- `triggerStartGame()`: Transita o estado para GAME quando ambos os jogadores estão prontos.
- `gameOver()`: Calcula o vencedor e envia o resultado ao servidor via `"game_over#username SCORE"`.

3 Servidor

Responsável por registo, autenticação, lobby, emparelhamento e atualização de nível.

Estrutura: O servidor Erlang organiza utilizadores em mapas e listas. Cada cliente tem um PID associado.
Lobby: Utilizadores que procuram jogos são colocados numa lista. O servidor tenta emparelhar pares com diferença de nível igual ou inferior a 1.
Jogo: Após emparelhamento, os jogadores entram em estado de jogo, e os seus movimentos são sincronizados com mensagens periódicas.
Fim de jogo: O servidor espera pelos dois resultados, determina o vencedor e atualiza os níveis. Vitórias consecutivas aumentam o nível, derrotas consecutivas podem fazê-lo descer, com limite mínimo de nível 1.

Algumas funções importantes:

- `find_match/1`: Procura parceiro de jogo para um jogador.
- `find_pair/5`: Verifica compatibilidade e emparelha jogadores.
- `handle_message/2`: Trata mensagens do cliente como registo, posição, fim de jogo, etc.
- `client/2`: Processo principal de cada cliente ligado; escuta mensagens da socket e delega para `clientInput/4`.
- `clientInput/4`: Interpreta e processa mensagens textuais recebidas do cliente, incluindo autenticação, procura de jogo, envio de posições e resultados.

3.1 server.erl

Contém o ponto de entrada da aplicação, cria o socket de escuta e aceita novas conexões. Cada conexão gera um processo `client/2`, que gere a comunicação individual com o cliente.

3.2 login_manager.erl

Responsável por gerir o estado dos jogadores: mapa de utilizadores autenticados, níveis, vitórias e derrotas. Fornece funções como `register`, `auth` e `get_top10`.

3.3 files.erl

Lida com dados em ficheiros. Implementa leitura e escrita dos utilizadores, níveis, resultados e credenciais. Usa ficheiros texto simples para garantir persistência entre execuções.

4 Implementações

4.1 Emparelhamento de Jogadores

Para garantir competições justas, o emparelhamento é feito com base na diferença de nível entre jogadores (máximo 1). Esta lógica está implementada no módulo `login_manager.erl` usando as funções `find_match/1` e `find_pair/5`.
O lobby verifica ciclicamente se há pares compatíveis e emparelha-os automaticamente.

4.2 Sincronização

Durante o jogo, cada cliente envia periodicamente a sua posição. O servidor reencaminha essa informação para o adversário, garantindo a consistência do estado entre os dois.

4.3 Comunicação Cliente-Servidor

A comunicação baseia-se na troca de mensagens de texto (strings), com comandos como `register#username password`, `join_lobby#username password`, `position#X,Y`, `game_over#username SCORE`, entre outros.
Estas mensagens são processadas pelo servidor, que responde ou reencaminha conforme necessário. A escolha por este protocolo simples facilita o parsing e debugging.

4.4 Escolhas

- **Thread listener no cliente**: Permite receção assíncrona de mensagens do servidor sem bloquear a interface.
- **Estados e flags**: A navegação no cliente é gerida por enums e flags booleanas que asseguram fluidez entre menus.
- **synchronized**: Usado em blocos críticos (como o acesso a dados de jogo), evitando condições de corrida entre threads.

4.5 Imagem jogo

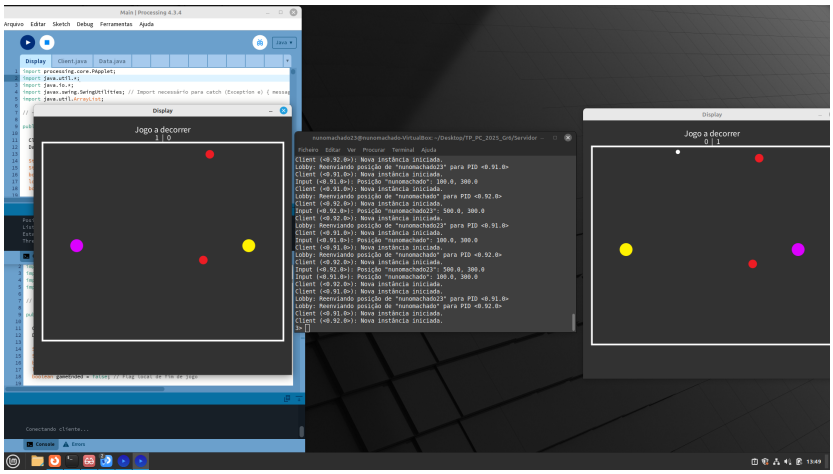

Figura 1: Exemplo de jogo a decorrer c/ 2 jogadores na mesma máquina

5 Conclusão

O projeto permitiu aplicar conceitos de programação concorrente, comunicação entre processos, gestão de estado e design de interfaces gráficas.
A abordagem cliente-servidor revelou-se adequada para este tipo de aplicação competitiva.
O maior desafio foi garantir uma sincronização eficiente entre jogadores, alcançada com o uso de threads, `synchronized` e validação constante do estado do jogo. Assim, garantimos exclusão mútua, ausência de deadlocks e prevenção de starvation.
Apesar de haver margem para melhorias, o sistema encontra-se funcional e estável, cumprindo os objetivos propostos.