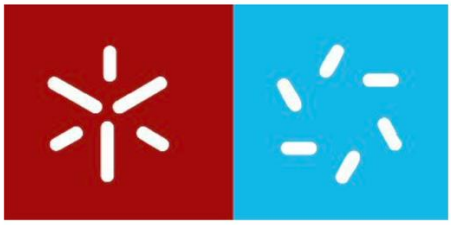


3 DE JUNHO DE 2024

PROGRAMAÇÃO CONCORRENTE
TRABALHO PRÁTICO
GRAVIDADE
2023/2024

LICENCIATURA EM CIÊNCIAS DA COMPUTAÇÃO
UNIVERSIDADE DO MINHO

EDUARDO ANDRÉ SILVA CUNHA A98980
FÁBIO ALEXANDRE MAGALHÃES RIBEIRO A100058
GONÇALO EMANUEL FERREIRA MAGALHÃES A100084

Índice

1. Introdução

O presente relatório refere-se ao projeto desenvolvido ao longo do semestre e visa apresentar e explicar nossa metodologia e implementação. O projeto consiste no desenvolvimento de um jogo multijogador, onde cada jogador controla um avatar representado por um círculo que se movimenta e interage com outros avatares e objetos do ambiente, como o sol e planetas.

O relatório está dividido em duas partes principais. A primeira parte aborda o servidor desenvolvido em Erlang, cuja função é armazenar a informação de cada objeto e o estado do jogo, além de atualizar os clientes com novas informações geradas. O servidor também possui outras funcionalidades, como guardar o estado do jogo em ficheiro. A segunda parte refere-se ao cliente, desenvolvido em Processing (uma extensão de Java), que recebe as informações enviadas pelo servidor e desenha o jogo.

2. Jogo
2.1 Servidor

“Main.erl”

Ao iniciar o servidor, foram criados três processos principais: o match_manager, que funciona como lobby e armazena os jogadores que pretendem jogar antes do início da partida; o accounts_manager, que gere as contas dos utilizadores, verificando se estão autenticados, o seu nível, entre outros; e o file_manager, que tem a função de armazenar em ficheiro as informações dos utilizadores.

Depois disso, é estabelecida a ligação através de socket. Com esta ligação, é chamada uma função que processa as informações recebidas, permitindo ações como criar conta, iniciar sessão, terminar sessão, pedir estatísticas, entre outras.

De seguida, é definida a função matchMan referente ao lobby dos jogadores. Esta função armazena os PIDs dos jogadores que pretendem jogar e inicia uma partida quando encontra dois jogadores com uma diferença de nível inferior a 1. Embora não tenhamos implementado a funcionalidade de iniciar partidas com quatro jogadores, todas as funções do jogo foram definidas de forma a suportar mais de dois jogadores, trabalhando sempre com mapas e listas. Em resumo, quando existem dois jogadores de nível igual ou com uma diferença de até 1, e ambos procuram uma partida, esta é iniciada automaticamente para os dois. Para isso, definimos funções que avaliam o nível dos jogadores e que enviam ao cliente a informação de início de jogo. Também definimos funções que notificam o fim do jogo e que enviam a informação atualizada dos jogadores e objetos da cena para o cliente, para que este possa desenhá-la corretamente (estas funções são gameOver e gameState, respetivamente).

“Game.erl”

Para definir o jogo em Erlang, criámos um ficheiro denominado "game.erl". Este ficheiro contém uma função principal que inicia um jogo com dois jogadores.

São criados dois jogadores, e cada jogador possui um nome, posição (x e y), teclas premidas, velocidade, ângulo, aceleração, combustível e estado de jogo (0 a jogar, 1 perdeu, 2 ganhou). Também é criado um processo "sender" que trata de enviar as informações do jogo ao servidor, para que este possa atualizar o cliente.

Os jogadores têm posições de spawn possíveis, que não podem ser em cima de outro jogador, dentro das bordas do jogo e fora do raio de rotação dos planetas. Cada jogador move-se usando as teclas W, A, D, sendo o W para a frente e A e D para mudar a sua direção. Os jogadores têm um limite de combustível e, quando este se esgota, não podem mover-se mais. A cada momento do jogo, são atraídos para o centro (simulando a força do sol).

Para implementar isto, definimos algumas funções, como o keymanager, que verifica as teclas premidas (enviadas pelo cliente) e aplica as ações correspondentes chamando outras funções auxiliares. Para tratar da atração ao sol, criámos a função

CalculateSunAttraction e a função data_sender, que envia a informação atualizada ao servidor e simula o jogo, mantendo sempre a informação atualizada.

No jogo, foi criado também um sol, fixo no centro do jogo, e planetas que giram ao seu redor. Cada planeta possui uma posição e tamanho, além de outros fatores para o cálculo da sua órbita elíptica.

Durante a simulação do jogo, são calculadas a atração ao sol, as teclas premidas e a nova posição dos itens. Caso não receba uma mensagem a indicar que o jogo terminou, esta informação é enviada ao servidor para ser passada ao cliente.

Definimos ainda colisões. A colisão entre jogadores é elástica, embora nem sempre funcione corretamente (dependendo do angulo de colisão). Para isso, definimos funções como a handleelasticcollision. A colisão entre jogador e planeta faz com que o jogador perca automaticamente o jogo, e o outro jogador ganha se permanecer vivo durante 5 segundos. Sair dos limites do jogo também faz com que o jogador perca.

Quando um jogador perde, é chamada a função surrender, que coloca o seu estado a 1 (perdeu), e é criado um processo que, após 5 segundos, informa a função de simulação que o tempo passou e os restantes jogadores passam a ter o estado a 2 (ganharam). Esta informação é passada ao servidor, que trata de fazer logout dos jogadores em questão.

“User_manager.erl”

Este ficheiro foi criado para gerir os utilizadores e possui funcionalidades como login, logout, registar vitórias, derrotas, pedir o nível do utilizador, entre outras. Para isso, foi criada uma função principal que recebe todos estes pedidos e os processa utilizando funções auxiliares. Este ficheiro trabalha em conjunto com o ficheiro "file_manager", que armazena as informações do jogo em ficheiro.
Possui algumas funções auxiliares, como a create_account, que, dado um utilizador, verifica se já existe nos utilizadores registados. Se não existir, cria uma nova conta com o respetivo nome, password, nível a 1, vitórias e derrotas a 0, e estado como "logged out".
Outras funções auxiliares interessantes são as de registo de vitória e derrota. Ambas verificam se, com uma nova vitória ou derrota, o utilizador mantém o nível ou muda de nível (aumenta/diminui). Caso haja alteração de nível, o número de vitórias e derrotas é colocado a 0, sendo que este número se refere ao número de vitórias e derrotas consecutivas.

“File_manager.erl”

Este ficheiro foi criado para armazenar registos em ficheiro. Foi desenvolvida uma função principal que recebe três tipos de pedidos: estatísticas, escrita no ficheiro ou remoção de uma conta.

Caso receba um pedido de estatísticas, o ficheiro é lido e a sua informação é tratada, ordenando os utilizadores por nível, vitórias e derrotas consecutivas. Se receber um pedido de escrita no ficheiro, a informação recebida é adicionada ao ficheiro conforme o

formato definido. Se receber um pedido para remover uma conta, o ficheiro é aberto e o utilizador em questão é removido do ficheiro.

Este ficheiro de código possui ainda algumas funções úteis para outros módulos, como a função readContent e parser, que, em conjunto, fazem a leitura e interpretação do ficheiro, gerando mapas com a informação lida.

2.2 Cliente

“Game.pde”

O ficheiro "game.pde" é o ficheiro principal do código cliente. Inicialmente, são definidos os vários estados do jogo ("menu", "login", "register", "game", entre outros). Em seguida, é definido o início do jogo na função setup, que estabelece o tamanho da tela, o menu inicial, a porta e o servidor para conexão via socket (de forma a conectar ao servidor). É iniciado um objeto do tipo connectionManager, que terá a função de enviar e receber informações do servidor. Além disso, são iniciados objetos que serão utilizados no jogo.

A seguir, temos a função draw, que, dependendo do estado do jogo, desenha a interface correspondente. Por exemplo, no estado "menu", é desenhado o menu inicial, e no estado "game", é desenhado o jogo em si.

Depois, definimos uma função parser que trata da informação recebida via socket, atualizando a posição dos jogadores (caso seja necessário), atualizando os planetas ou definindo quem ganhou e/ou perdeu a partida.

Em seguida, temos funções como keyPressed e mouseClicked, que definem a interação do jogador com o jogo. Por exemplo, se um utilizador clicar no botão de login no menu inicial, será redirecionado para outra página, e se nessa página inserir um nome de utilizador e palavra-passe válidos, será enviado um pedido de conexão ao servidor e, caso receba uma resposta positiva, o jogo passa para o próximo estado.

Além disso, temos também uma função chamada formataEstatisticas, que, dada uma string de estatísticas recebida do servidor, a organiza de forma a proporcionar uma apresentação correta e agradável para o jogador.

“Player.pde”, “Sol.pde”, “Estrela.pde”

Estas classes foram definidas para representar cada objeto do jogo. A classe Player representa um jogador e tem funções para atualizar a sua posição e fazer o seu desenho. A classe Sol tem como função apenas desenhar um sol no centro do jogo. A classe Estrela desenha um ponto branco de fundo representando uma estrela, tendo apenas um objetivo visual, com o intuito de tornar o jogo mais semelhante ao espaço sideral.

“ConnectionManager.pde”

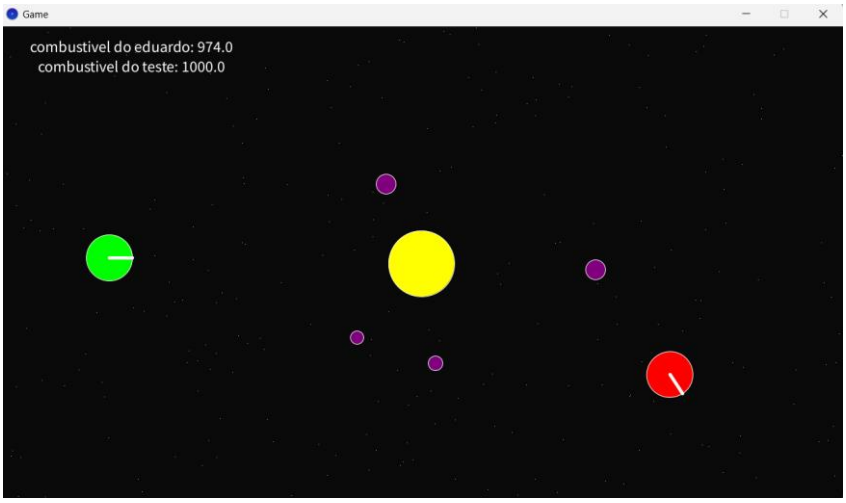
A classe ConnectionManager foi criada para facilitar a conexão entre o utilizador e o servidor. Esta classe guarda variáveis como a socket e os buffers de entrada e saída. Foram definidas funções como receive, que recebe uma mensagem do servidor e a retorna, e funções como registerUser, que, dados um nome de utilizador e uma palavra-passe, envia ao servidor um pedido de registo. Além desta função, existem várias outras, como loginUser, logoutUser, entre outras. É importante destacar as funções sendKey e releaseKey, que enviam ao servidor, durante o jogo, quais teclas foram premidas, para que este possa calcular as novas posições dos jogadores com base nas ações efetuadas.

“InputField.pde”

A classe InputField foi criada com o intuito de facilitar a gestão de campos de texto na interface, permitindo atualização dinâmica de texto, posição, e gerenciamento do estado de ativação para entrada de dados.

Menu do jogo:

Jogo:



3. Conclusão

Como mencionado, neste trabalho criámos um jogo denominado "Gravidade", que consiste em dois jogadores que vagueiam pelo espaço e tentam evitar a colisão com planetas e com o sol.

De forma geral, acreditamos ter alcançado a grande maioria dos objetivos. No entanto, não conseguimos definir uma colisão consistente entre os jogadores e admitimos que o jogo só é jogável uma vez, embora o servidor se mantenha ativo e com capacidade para suportar mais de um jogo. Também sabemos que não implementámos uma funcionalidade importante: permitir que mais de dois jogadores participem na partida. Embora tenhamos definido a maioria das funções de forma a possibilitar isso, usando sempre listas em vez de pares de jogadores, a implementação dessa funcionalidade no início do jogo no servidor foi adiada. Quando tentámos aplicá-la no final, o código já estava demasiado complexo para conseguirmos fazê-lo a tempo.

Consideramos este trabalho bastante extenso e complexo, um verdadeiro desafio! Este projeto foi importante para o aprofundamento do nosso conhecimento, principalmente sobre a linguagem Erlang.