Programação Concorrente
Licenciatura em Ciências da Computação
Trabalho Prático

Gravidade

João Gomes
A70400

Luís Ventuzelos
A73002

Paulo Barbosa
A70073

10 de Junho de 2017

Conteúdo

Capítulo 1

Introdução

Neste trabalho é nos pedido que implementemos um mini-jogo onde 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 avatares deverão interagir entre si e com o ambiente que os rodeia, segundo uma simulação efetuada pelo servidor. A nossa aplicação teria de permitir o registo e o login no mini-jogo, sendo que o utilizador só pode usufruir do jogo após o login ser verificado com sucesso.

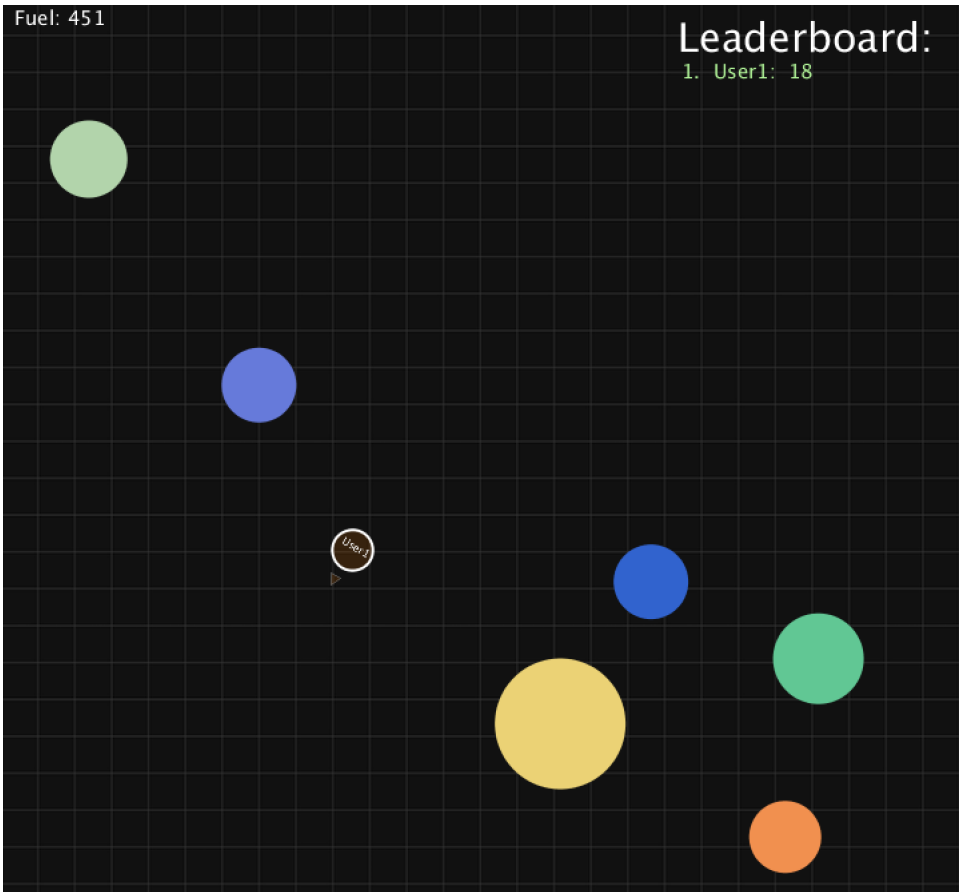

Figura 1.1: Gravity

Capítulo 2

Cliente

De maneira a desenvolvermos uma interface gráfica que nos permita suportar as funcionalidades pretendidas, utilizamos *Processing* e *Java* no desenvolvimento do cliente. Assim, para o desenho do jogo, teremos três classes:

- Cliente;
- Avatar;
- Planeta;

A classe **Cliente**, é responsável por iniciar a comunicação com o servidor, via uma **Socket** TCP e por desenhar os *Planetas* e *Avatares* nas posições calculadas. Infelizmente, não conseguimos utilizar corretamente as várias **Threads** necessárias e por isso, todos os cálculos das posições de cada bola (avatares e planetas) são feitos nas nossas classes *Java*, apesar de os fazermos no servidor, mas pelo motivo descrito anteriormente, não conseguimos utilizar esses resultados para o desenho das bolas.

2.0.1 Classe Avatar

Esta classe, tem todos os métodos necessários para que um avatar se mova corretamente na nossa cena, isto é, construtores da classe, um método para desenhar um avatar ,um método para calcular a posição do avatar caso chegue ao limite da janela estabelecida e um método que deteta colisão entre um planeta e um avatar. Dos requisitos pedidos para um avatar, apenas não implementamos a gravidade e o escudo.

2.0.2 Classe Planeta

Esta classe, tem todos os métodos necessários para que um planeta se mova corretamente na nossa cena, isto é, construtores da classe, um método para desenhar um planeta e um método para calcular a posição do planeta caso chegue ao limite da janela estabelecida.

Capítulo 3

Servidor

Para desenvolvermos o servidor utilizamos Erlang como foi pedido. A nossa função que ativa o servidor é a função *server()*, iniciando a comunicação via *Socket* TCP, através da porta **12345**, em *localhost*. Aqui, iniciamos todos os processos que têm de correr em background, por exemplo: Frames, Gestor_Logins, Gestor_Bolas, Room.

3.0.1 Função acceptor

Nesta função temos que aceitar conexões no socket e criar um novo processo User pois temos um jogador novo. Esta função é recursiva pois tem que estar sempre ativa à espera de novos jogadores.

3.0.2 Função room

Na função room, detetamos a entrada e a saida do cliente no servidor, e as mensagens por si enviadas através da **Socket**.

3.0.3 Função user

Nesta função temos todos os padrões que um processo user poderá receber, e as suas ações para cada padrão recebido, sendo aqui onde conseguimos enviar mensagens para o cliente, através do seguinte *pattern matching*:

```
{line, Data,_} ->
        gen_tcp:send(Sock, Data),
        user(Sock,Gestor_Bolas, Username, Gestor_Logins,Frames,Room);
```

Recebendo um padrão do tipo:{**line**, Msg, Pid} enviamos a mensagem **Msg** ao cliente, através da socket aberta dos dois lados da comunicação.

3.0.4 Função gestor_logins

Esta função é responsável, como o próprio nome indica, pela gestão dos registos e logins efetuados na aplicação. Garante que um registo é único e que não é possível existir dois registos com o mesmo nome. Também garante que, ao fazer login, não é possível entrar no jogo com uma conta que não tenha sido registada anteriormente, não permitindo também que um utilizador que já tenha efetuado o login e esteja online possa efetuar de novo um login.

Após um registo ter sido efetuado com sucesso o nosso servidor vai atualizar o Mapa de utilizadores acrescentando o nosso novo utilizador com o seu Username, a password a si associada e a flag *out* que nos diz que este utilizador ainda não se encontra online.

```
New_Users = maps:put(Username,{out,Password},Users)
```

Posteriormente envia a seguinte mensagem para a função gestor_Bolas.

```
Gestor_Bolas ! {new_avatar, Username, self()},
```

O seu processamento é explicado na secção *Função gestor_Bolas*.
Após um login ter sido efetuado com sucesso o servidor vai atualizar o Mapa de utilizadores modificando a flag , que anteriormente tinha o valor de *out*, para o valor *in* simbolizando assim um utilizador que efetuou um login com sucesso e se encontra online.

```
New_Login = maps:update(Username,{in,Password},Users),
```

Posteriormente envia a seguinte mensagem para a função gestor_Bolas.

```
Gestor_Bolas ! {start_game, Frames, Room},
```

O seu processamento é explicado na secção *Função gestor_Bolas*.

3.0.5 Função gestor_Bolas

No gestor de bolas, podemos receber os seguintes tipos de mensagens:

1. {**new_avatar**, Nome, Pid};
2. {**new_planeta**, Nome, Pid};
3. {**turn_right**, User};
4. {**turn_left**, User};
5. {**faster**,User};
6. {**slower**,User};
7. {**start_game**, Frames, Room}

Se o gestor de bolas receber uma mensagem do tipo 1, vai adicionar um novo avatar ao mapa das Bolas, com a chave **Nome**, que corresponde ao nome do jogador que criou esse avatar.
Com uma mensagem do tipo 2, irá adicionar ao mapa das bolas um planeta com um nome da seguinte forma "Planeta1",...,"PlanetaN".
Mensagens do tipo 3,4,5 e 6 correspondem a um certo avatar a mudar de direção, e nesse caso, a nova posição é calculada, e atualizada no mapa das bolas.
Uma mensagem do tipo 7, inicia o jogo para um determinado jogador, imediatamente após o seu login ser efetuado com sucesso.

3.0.6 Função frames

Esta função apenas serve para se recalcular as novas posições de cada bola, várias vezes, no nosso caso, fazemos em cada 16 milisegundos, de modo a termos 60 frames por segundo. Este tempo é garantido com a função **sleep(16)**.

Capítulo 4

Conclusão

O nosso trabalho neste momento é capaz de garantir a comunicação Cliente-Servidor, faz logins, registos e logouts de utilizadores, e calcula as posições da cada bola no servidor. Neste ultimo ponto, apesar de ter sido feito como pedido, no servidor, não conseguimos enviar corretamente os dado de cada bola para o Cliente a desenhar, e devido a este problema, optámos por efetuar as contas todas no Cliente.
Posto isto, achamos que poderíamos fazer melhor, usando duas **Threads** no Cliente, sendo uma para desenho da cena e para envio de mensagens para o Servidor, e a outra seria uma espera ativa, que estaria sempre pronta a receber mensagens do Servidor. Acreditamos que com estas **Threads** bem implementadas, que já conseguiríamos utilizar os valores calculados no Servidor para desenhas as bolas, no Cliente.

Como nota final gostaríamos apenas de referir que consideramos que este trabalho não está ao nível do que somos capazes e do que seria desejado pelo professor. Temos a perfeita noção que a responsabilidade disso recai sobre nós e que deveríamos ter garantido um trabalho mais consistente, mas devido à elevadíssima carga de trabalho que temos tido, não conseguimos gerir da melhor maneira o nosso tempo, de modo a conciliarmos tudo.