



UNIVERSIDADE DO MINHO

LICENCIATURA EM ENGENHARIA INFORMÁTICA

Sistemas Distribuídos

Trabalho Prático - Gestão de Frota

Grupo 9

Ana Rita Poças (A97284) Inês Ferreira (A97372)
Joana Branco (A96584) João Pedro Braga (A97368)

9 de janeiro de 2023

Introdução

Este trabalho prático surge no âmbito da Unidade Curricular, em que nos foi proposto a implementação de uma plataforma de gestão de uma frota de trotinetes elétricas, sob a forma de um par cliente-servidor em Java através do uso de *sockets* e *threads*.

Os objetivos definidos para este projeto baseiam-se nas funcionalidades a implementar descritas no enunciado. Estas focam-se principalmente na interação entre o cliente e o servidor que tem como finalidade apresentar as trotinetes disponíveis para o destino pretendido.

Num estado inicial deve ser possível a autenticação e registo do utilizador, permitindo-lhe efetuar login ou criar conta. Devem também ser consideradas as trotinetes existentes no mapa e se estas continuam disponíveis ou não. Adicionalmente, é gerado um sistema de recompensas que tem por base avaliar a distribuição das trotinetes disponíveis, ou seja, é gerada uma recompensa quando uma trotinete é movimentada de um local para outro. Por outro lado, deve ser registada a reserva de uma trotinete disponível, retornando o código de reserva e o seu local. O custo da viagem também deve ser calculado de acordo com o tempo passado desde a reserva e a distância percorrida pela trotinete. Numa fase final, devem ser implementadas as notificações que destinam-se a informar o utilizador de recompensas com origem a menos de uma distância fixa.

No contexto do nosso programa, definimos uma grelha 20x20 para o mapa, e as localizações são representadas por um par de inteiros (x,y).

Funcionalidades Implementadas

A aplicação distribuída que concebemos foi capaz de implementar as seguintes funcionalidades:

Autenticação e registo de utilizador

Um utilizador, da nossa aplicação, consegue comunicar com o servidor. Um utilizador tem, associado a si um username e uma password. Caso seja a sua primeira vez no programa, o utilizador deve registar a conta nova introduzindo os dados que pretende que fiquem associados a si. Depois de validados, os dados do utilizador são armazenados (através da funcionalidade de serialização - que permite que caso haja falha no servidor, os dados fiquem armazenados). A conexão entre o utilizador e o servidor é assegurada pela classe **Desmultiplexer** e pela classe **Connection**.

```
TrotIUM
Que operação pretende efetuar?
1-> Iniciar sessão
2-> Registrar uma nova conta
0-> Sair
Insira o número corresponde à operação: 2

Registrar Nova Conta
Introduza um username: Grupo9
Introduza uma palavra-passe: Grupo9
Até breve...

Nova conta Registada

TrotIUM
Que operação pretende efetuar?
1-> Iniciar sessão
2-> Registrar uma nova conta
0-> Sair
Insira o número corresponde à operação: 1

Iniciar Sessão
Introduza o seu username: Grupo9
Introduza a sua password: Grupo9

Sessão iniciada!
```

Figura 1: Menu de registo de conta e inicio de sessão

A classe **Connection** é uma intermediária entre os utilizadores e os servidores, uma vez que consegue enviar uma mensagem recebida através de um socket que lhe está associado ou receber dados por esse mesmo socket, transformando-os num formato que o utilizador e o servidor consigam ler, através do uso de `DataInputStreams` e `DataOutputStreams`.

A classe **Demultiplexer**, baseia-se na ligação entre o utilizador e a `Connection`. Uma vez que existem vários threads **Client**, que podem ter de esperar por uma mensagem, o **Demultiplexer**, garante que estes threads "adormeçam" enquanto esperam pela mensagem e sejam "acordados" assim que ela chegue. A existência de um desmultiplexador no nosso projeto faz com que consigamos receber dados de um canal e distribuir por vários canais. Para cada tag é criado um buffer de mensagens .

Listagem dos locais onde existem trotinetes livres

Depois de ter iniciado sessão, o programa pede ao utilizador que introduza a sua localização atual. Esta localização é enviada para o servidor, que se encarrega de verificar as trotinetes livres num raio de $D=2$. Para tal, foi necessário implementarmos as classes **Positions** e **Trotinetes**.

A classe **Positions** encarrega-se de guardar a informação relativa às coordenadas existentes no nosso programa. A classe **Trotinetes**, inicializa as trotinetes existentes no nosso mapa e implementa todos os métodos relativos à manipulação das trotinetes na nossa aplicação. Assim, de forma a conseguirmos verificar as trotinetes que estão num raio de $D=2$ da posição do cliente, recorreremos à função `getClosestTrotinetes` que utiliza a distância de Manhattan para efetuar o cálculo. Essa informação é depois enviada para o cliente, sendo imprimida no ecrã.

```

 Bem vindo, Grupo9
 Por favor, insira as coordenadas da sua localização (x,y):
 (1,2)
 -----
 As seguintes trotinetes estão no máximo a D=2 de si:
 (1,2);(1,2);(2,2);
 -----

```

Figura 2: Menu de listagem das trotinetes livres num raio de D=2

Listagem das Recompensas

Depois da apresentação no ecrã do locais das trotinetes que se encontram livres, procedemos, também, à apresentação no ecrã da listagens das origem-destino que dão direito às recompensas. O servidor, caso existam recompensas, envia as listas de origens (posições que possuem mais do que uma trotinete) e destinos (posições sem nenhuma trotinete num raio D=2) elegíveis para recompensas.

```

 (1,2)
 -----
 As seguintes trotinetes estão no máximo a D=2 de si:
 (1,2);(1,2);(2,2);
 -----
 Os pares origem-destino com recompensas no nosso mapa são :
 Origens acessíveis a no máximo D=2 de si:
 (1,2);(1,2);
 Destinos elegíveis para recompensa:
 (0,0);(0,4);(0,5);(0,6);(0,7);(0,8);(0,9);(0,10);(0,11);(0,12);(0,13);(0,14);(0,15);(0,16);(0,17);(0,18);(0,19);(1,5);(1,6);(1,7);(1,8);
 (1,9);(1,10);(1,11);(1,12);(1,13);(1,14);(1,15);(1,16);(1,17);(1,18);(1,19);(2,6);(2,7);(2,8);(2,9);(2,10);(2,11);(2,12);(2,13);(2,14);
 (2,15);(2,16);(2,17);(2,18);(2,19);(3,0);(3,7);(3,8);(3,9);(3,10);(3,11);(3,12);(3,13);(3,14);(3,15);(3,16);(3,17);(3,18);(3,19);(4,0);
 (4,2);(4,6);(4,7);(4,8);(4,9);(4,10);(4,11);(4,12);(4,13);(4,14);(4,15);(4,16);(4,17);(4,18);(4,19);(5,0);(5,1);(5,2);(5,6);(5,7);(5,8);
 (5,9);(5,10);(5,11);(5,12);(5,13);(5,14);(5,15);(5,16);(5,17);(5,18);(5,19);(6,0);(6,1);(6,7);(6,8);(6,9);(6,10);(6,11);(6,12);(6,14);(6,15);
 (6,16);(6,17);(6,18);(6,19);(7,0);(7,1);(7,2);(7,6);(7,7);(7,11);(7,15);(7,16);(7,18);(7,19);(8,0);(8,1);(8,3);(8,5);(8,6);(8,9);(8,10);(8,11);
 (8,12);(8,13);(8,14);(8,15);(8,16);(8,17);(8,18);(8,19);(9,0);(9,1);(9,2);(9,3);(9,4);(9,5);(9,6);(9,7);(9,8);(9,9);(9,10);(9,11);(9,12);(9,13);(9,14);(9,15);(9,16);(9,17);(9,18);(9,19);(10,0);(10,1);(10,2);(10,3);(10,4);(10,5);(10,6);(10,7);(10,8);(10,9);(10,10);(10,11);(10,12);(10,13);(10,14);(10,15);(10,16);(10,17);(10,18);(10,19);(11,0);(11,1);(11,2);(11,3);(11,4);(11,5);(11,6);(11,7);(11,8);(11,9);(11,10);(11,11);(11,12);(11,13);(11,14);(11,15);(11,16);(11,17);(11,18);(11,19);(12,0);(12,1);(12,2);(12,3);(12,4);(12,5);(12,6);(12,7);(12,8);(12,9);(12,10);(12,11);(12,12);(12,13);(12,14);(12,15);(12,16);(12,17);(12,18);(12,19);(13,0);(13,1);(13,2);(13,3);(13,4);(13,5);(13,6);(13,7);(13,8);(13,9);(13,10);(13,11);(13,12);(13,13);(13,14);(13,15);(13,16);(13,17);(13,18);(13,19);(14,0);(14,1);(14,2);(14,3);(14,4);(14,5);(14,6);(14,7);(14,8);(14,9);(14,10);(14,11);(14,12);(14,13);(14,14);(14,15);(14,16);(14,17);(14,18);(14,19);(15,0);(15,1);(15,2);(15,3);(15,4);(15,5);(15,6);(15,7);(15,8);(15,9);(15,10);(15,11);(15,12);(15,13);(15,14);(15,15);(15,16);(15,17);(15,18);(15,19);(16,0);(16,1);(16,2);(16,3);(16,4);(16,5);(16,6);(16,7);(16,8);(16,9);(16,10);(16,11);(16,12);(16,13);(16,14);(16,15);(16,16);(16,17);(16,18);(16,19);(17,0);(17,1);(17,2);(17,3);(17,4);(17,5);(17,6);(17,7);(17,8);(17,9);(17,10);(17,11);(17,12);(17,13);(17,14);(17,15);(17,16);(17,17);(17,18);(17,19);(18,0);(18,1);(18,2);(18,3);(18,4);(18,5);(18,6);(18,7);(18,8);(18,9);(18,10);(18,11);(18,12);(18,13);(18,14);(18,15);(18,16);(18,17);(18,18);(18,19);(19,0);(19,1);(19,2);(19,3);(19,4);(19,5);(19,6);(19,7);(19,8);(19,9);(19,10);(19,11);(19,12);(19,13);(19,14);(19,15);(19,16);(19,17);(19,18);(19,19);
 Insira as coordenadas da trotinete (x,y) que pretende reservar:

```

Figura 3: Menu de Listagem das Recompensas

Reserva de trotinetes livres

Depois de ter recebido a listagem das trotinetes livres num raio D=2, o programa pede ao cliente que escolha a que pretende reservar. O cliente deve seleccionar a trotinete que pretende e esta informação será enviada para o servidor. O servidor, irá guardar este valor como a origem do percurso do cliente e terá a atenção de remover esta trotinete da lista de trotinetes livres, uma vez que está agora a ser usada por este cliente.

```

.....
Insira as coordenadas da trotinete (x,y) que pretende reservar:
(1,2)
.....

A viagem com origem em (1,2) tem código de reserva 1
.....
Insira o código de reserva que lhe foi atribuido e o local estacionamento da trotinete (x,y):

```

Figura 4: Menu de reserva de trotinete livre

Estacionamento de uma trotinete

Para estacionar uma trotinete, o utilizador deve fornecer o código de reserva que lhe foi atribuído anteriormente e a posição onde pretende estacionar a sua trotinete. O cliente fornece esta informação ao servidor, que calculará o custo da viagem, com base na distância percorrida entre a origem e o destino (local de estacionamento da trotinete) e o tempo percorrido desde a reserva. O servidor, calculará também a recompensa, caso a posição onde o utilizador pretende estacionar a trotinete seja apta para recompensa. Desta forma é apresentado ao utilizador o valor do custo da viagem e a o valor da recompensa (0, caso não exista).

```

.....

A viagem tem o custo de 4
.....

Ganhou a seguinte recompensa 0
.....

```

Figura 5: Menu de estacionamento de uma trotinete

Observações

Mecanismos de Exclusão Mútua no nosso programa

Sendo o trabalho desenvolvido uma aplicação distribuída, um dos principais objetivos é que suporte a programação concorrente. Para tal, foram adotados os seguintes mecanismos para garantir que não temos problemas de exclusão mútua ocorre no nosso programa:

- Tivemos que ter atenção a fazer um locks na trotinete, para as conseguir reservar;
- Usamos locks nas classes Users, Trotineres, Recompensa, CodUser e nas estruturas de dados do Servidor;
- Para garantirmos exclusão mútua nas contas usamos ReadWriteLocks pois fazemos várias leituras e poucas escritas e assim melhoramos o desempenho do nosso programa;
- Utilizamos condition() e await() e signal(), na classe das Recompensas;

Especificação das Frames

A comunicação entre o utilizador e o servidor, recorreremos à representação das mensagens trocadas entre eles por frames. Uma frame é constituída por uma *tag*- valor numérico, identificador do cliente - usada pelo servidor para saber qual o cliente que comunicou com ele e uma mensagem- array de bytes. Na tabela abaixo temos a representação de cada valor de *tag*:

Tag	Tipo de mensagem
0	Solicitação de início de sessão
1	Solicitação de novo registo de conta
2	Inserir coordenadas do utilizador no mapa
3	Pedido de reserva de trotinete
4	Inserir o código de reserva coordenadas do destino do utilizador no mapa
5	Listar as recompensas presentes no mapa

Conclusão

Por fim, acreditamos que fomos capazes de desenvolver as funcionalidades básicas propostas e consolidar os conhecimentos obtidos no desenvolver da Unidade Curricular. O projeto também permitiu também aprimorar as nossas noções relativos a threads, sockets, etc.

Apesar de não termos sido capazes de implementar a funcionalidade de "Um cliente pedir para ser notificado quando apareçam recompensas com origem a menos de uma distância fixa D de determinado local", uma vez que todas as outras funcionalidades estão completamente implementadas, acreditamos que o programa está totalmente de acordo com as nossas expectativas, pois achamos que fizemos um bom desempenho e conseguimos apresentar a implementação da autenticação e registo de um utilizador, listar os locais onde há trotinetes disponíveis e as respetivas recompensas.