Universidade do Minho
Escola de Engenharia
Mestrado Integrado em Engenharia Informática

Unidade Curricular de Agentes Inteligentes

Ano Letivo de 2017/2018

Relatório do Trabalho Prático

Bruno Pereira (a75135), Maria Ana de Brito (a73580)

Novembro, 2018

AI

Índice

Índice de Figuras

1. Introdução

O trabalho prático tem por base um Sistema de Partilha de Bicicletas, onde é preciso ter em atenção o equilíbrio do número de bicicletas em cada uma das estações que fazem parte do sistema. Deste modo, o objetivo é que não ocorram grandes desequilíbrios em relação ao armazenamento de bicicletas, ou seja, queremos evitar que algumas estações estejam sempre cheias, enquanto que outras possuam um número pequeno de bicicletas lá armazenadas.

Este relatório visa, então, numa primeira fase um planeamento da resolução deste problema, analisando as várias questões que daí nascem e nomeando soluções para as mesmas. Ainda são apresentadas a arquitetura do trabalho que será levada em conta aquando a fase de programação dos agentes e um esquema do protocolo da comunicação entre os diversos agentes, que explica a necessidade desta troca de mensagens. Além disso, será fornecida uma explicação e justificação dos comportamentos dos três tipos de agentes existentes: Agente Utilizador, Agente Estação e Agente Interface.

Numa segunda fase, será apresentada a implementação do SPB. Os agentes e seus comportamentos serão explicados e justificados recorrendo a fórmulas matemáticas, imagens e excertos de código. Além disso, também serão explicitadas as mensagens trocadas entre os vários agentes do sistema.

Área de Aplicação: Agentes Inteligentes, JADE, JESS

Palavras-Chave: Agentes, Comportamento, Comunicação, Equilíbrio, Bicicletas, Área de Proximidade, Taxa de Ocupação, Estado de Ocupação, Incentivo, Mensagem, Trajeto, Utilizador, Estação, Comportamento

2. Entidades do Sistema de Partilha de Bicicletas

2.1. Utilizador

O utilizador do sistema corresponde a um cliente do SPB que está interessado em requisitar uma bicicleta para completar um determinado trajeto. Deste modo, possui um nome, isto é, um identificador, assim como coordenadas *x* e *y*, que correspondem à sua posição num determinado momento. Estas coordenadas serão atualizadas a cada momento, de modo a ter a posição do utilizador atualizada.

2.2. Estação

O Sistema de Partilha de Bicicletas possui um conjunto de estações espalhadas ao longo do domínio do problema, assim cada estação tem uma coordenada *x* e *y*, que identificam o local onde está situada. De forma a poderem ser distinguidas, possuem, também, um nome identificador. Para poderem definir a sua Área de Proximidade, cada estação tem um raio, pois diferentes estações podem ter APE diferentes. O grupo definiu que todas as áreas teriam a forma de um círculo, tomando a seguinte estrutura:

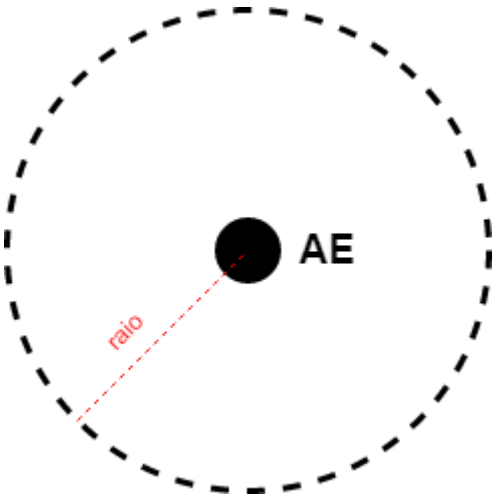


Figura 1 - Área de Proximidade de uma Estação

3. Agentes do Sistema de Partilha de Bicicletas

3.1. Agente Utilizador

Assim que o Agente Utilizador é inicializado, cria-se o seu ficheiro *clp*, cujo nome reflete o identificador do *user*. Deste modo, o nome do ficheiro tomará a seguinte estrutura: <nomedoutilizador>.clp. É neste ficheiro que serão armazenadas as diversas informações relativas ao Utilizador.

O Agente Utilizador começa, então, numa estação aleatória existente no SPB, pois para o domínio do problema só faz sentido considerar o percurso do utilizador dentro dos limites do sistema. Assim, não interessa o percurso prévio do utilizador, pois para o sistema o utilizador só existe nos momentos entre o aluguer da bicicleta e a sua devolução. A estação final para a qual o utilizador se dirige também é selecionada do conjunto de estações do SPB. As estações iniciais e finais não podem ser iguais, pois não faria sentido no contexto em questão, pois era impossível definir um trajeto para o utilizador.

Além disso, cada utilizador, tal como cada pessoa no mundo real, anda de bicicleta a uma velocidade distinta. Enquanto que uns são mais lentos a pedalar, outros podem ser mais rápidos a chegar ao destino. Desta forma, a velocidade de um utilizador é também um número aleatório entre os valores 3 e 5. Graças a este valor e à distância total a ser percorrida (calculada através das coordenadas iniciais e finais), é possível calcular a posição do utilizador a cada momento, fornecendo, então, as suas posições sucessivas atualizadas.

Como a partir de um determinado ponto do seu trajeto, o utilizador vai começar a receber incentivos para devolver a sua bicicleta em várias estações, é preciso, igualmente, definir um valor mínimo para a aceitação do incentivo. Qualquer incentivo abaixo deste limite não será aceite pelo utilizador. No entanto, assim que aceite um incentivo, o utilizador não poderá aceitar outro incentivo. O destino final será a nova estação e o trajeto do utilizador terá de refletir isso, pois agora caminha na direção de uma outra estação. Assim, será calculada uma nova distância a ser percorrida.

Sempre que um AU entra no sistema, terá de interrogar a estação inicial respetiva acerca da disponibilidade para alugar uma bicicleta. Assim, se a estação possuir bicicletas disponíveis, o utilizador poderá alugar uma e prosseguir no seu caminho. Caso contrário, desiste de adquirir uma bicicleta e desaparece do sistema. Esta solução assemelha-se a um caso real, pois se uma pessoa se dirige a uma estação de partilha de bicicletas, não irá esperar por um tempo

indeterminado por uma bicicleta. Arranja, pois, outro meio de transporte para chegar ao destino pretendido.

Após esta fase, o agente irá admitir vários comportamentos, que serão explicados mais adiante.

Por fim, quando este acaba a sua contribuição no sistema, seja porque chegou à estação final ou porque não conseguiu alugar uma bicicleta, o seu ficheiro *clp* é apagado, pois já não faz sentido armazenar o seu ficheiro.

3.2. Agente Estação

O Agente Estação é caracterizado pelo seu estado de ocupação que depende do número de bicicletas que estão armazenadas naquele momento em relação ao número máximo de bicicletas que podem estar lá armazenadas. A ocupação de uma estação é, então, calculada pela seguinte fórmula:

$$Ocupação = \frac{Número\ de\ bicicletas\ armazenadas}{Número\ máximo\ de\ bicicletas\ armazenadas}$$

Dependendo do seu estado, a estação oferecerá um certo incentivo aos utilizadores que se encontrem na sua proximidade. É lógico que se a ocupação for baixa, o incentivo será maior, pois a estação terá como interesse atrair mais bicicletas, enquanto que se a ocupação for alta, o valor do incentivo terá um valor baixo, pois é benéfico que a bicicleta seja armazenada noutra estação qualquer. Assim, o valor do incentivo a ser proposto a um AU será dado pela fórmula:

$$Incentivo = 1 - \frac{Número\ de\ bicicletas\ armazenadas}{Número\ máximo\ de\ bicicletas\ armazenadas}$$

De forma a saber quais os utilizadores que se encontram dentro da sua área de proximidade, é preciso também manter uma lista dos mesmos.

Uma estação de bicicletas, então, possui um número máximo de bicicletas que podem ser lá armazenadas, a sua capacidade. No entanto, quando são inicializadas a sua capacidade não está ao nível máximo, de forma a dar oportunidade aos utilizadores poderem entregar lá a sua bicicleta. Este aspeto surgiu quando o grupo pensou num caso em que as estações seriam inicializadas ao máximo de capacidade e nenhum utilizador requisitasse uma bicicleta de uma determinada estação. Caso um utilizador pretendesse devolver lá a sua bicicleta iria ficar indeterminadamente à espera de ter uma vaga. Assim, resolveu-se criar as estações com menos estações do que o número máximo permite.

De seguida, cada estação trata de se registar no *Directory Facilitator*, de forma a poderem estar identificadas para qualquer agente que pretenda comunicar com elas,

nomeadamente o Agente Interface. O AE terá ainda vários comportamentos de execução que serão explicados na secção seguinte.

Para fins estatísticos, o Agente Estação tem, ainda, de armazenar um conjunto de valores acerca do seu estado geral, tal como o número de alugueres feito naquela estação e o número total de pedidos feitos. Além disso, tem também de conter o número de devoluções aí feitas, assim como o número de entradas de bicicletas na sua área. Outro valor importante para perceber qual o estado da estação é a sua ocupação, ou seja, o número de bicicletas armazenadas na estação em relação à capacidade máxima.

3.3. Agente Interface

De forma a podermos perceber qual o estado de cada estação e se o problema do equilíbrio de distribuição de bicicletas está a ser bem solucionado, é preciso poder analisar os vários valores de cada estação. Desta forma, o Agente Interface é responsável por recolher os diversos dados e mostrá-los em forma de gráficos, que são atualizados intermitentemente. Assim, é possível poder verificar os resultados em tempo real de execução do programa.

Por exemplo, a partir do número de pedidos feitos e de alugueres efetivamente realizados, é possível perceber quantos pedidos são recusados, ou seja, quantas vezes a estação não teve bicicletas que os utilizadores pudessem alugar. Neste caso, os analistas da empresa poderão decidir que será melhor aumentar a capacidade dessa estação. Se o número de entradas de bicicletas na área de proximidade for elevado, poderão considerar colocar mais estações nesta área. Por outro lado, os vários valores de ocupação permitirão tirar conclusões rápidas acerca do estado de cada estação e verificar se o problema em questão está a ser bem tratado.

4. Arquitetura dos Agentes

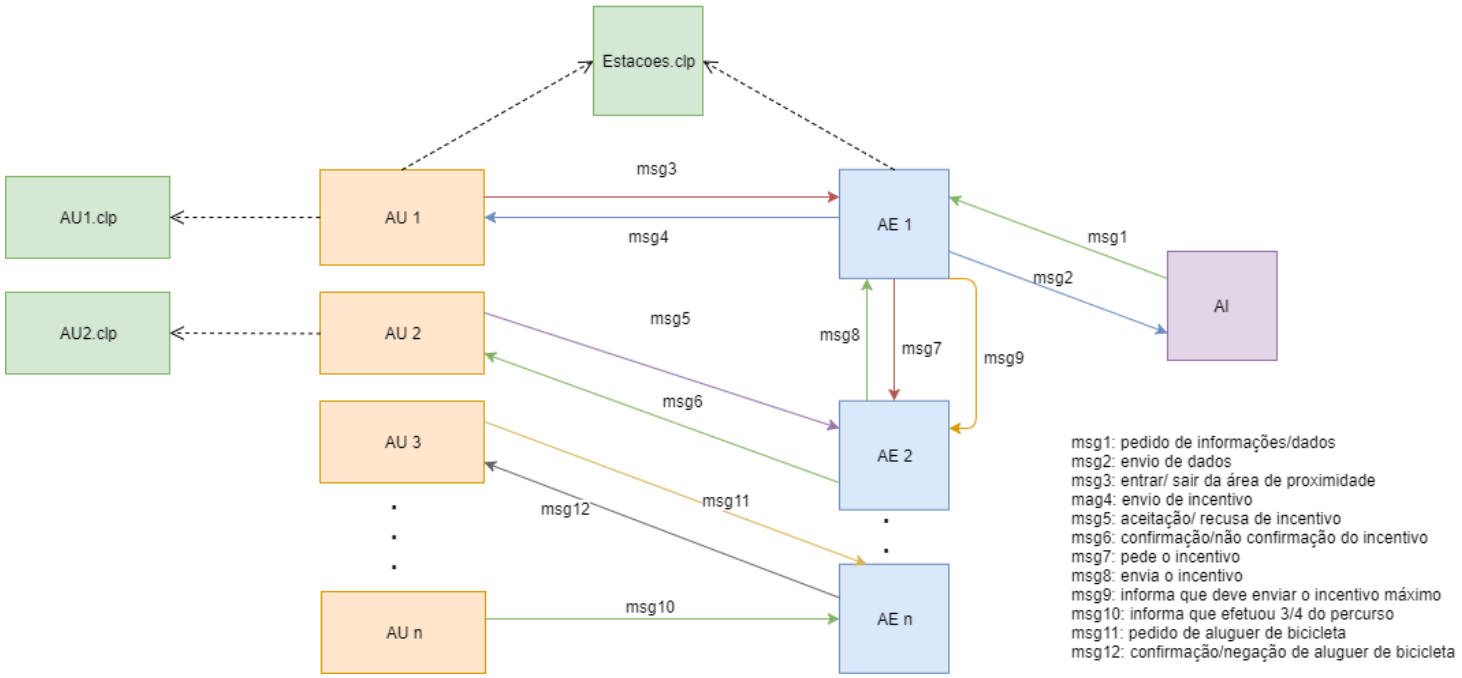


Figura 2 - Arquitetura dos Agentes

Analisando a arquitetura dos agentes, temos representadas as classes identificadoras dos três tipos de agentes: Agente Utilizador (AU), Agente Estação (AE) e Agente Interface (AI), bem como as mensagens que serão transmitidas entre eles. De forma a não sobrecarregar a imagem, esta representação é parcial, por isso assume-se que, embora seja o agente AU1 a enviar a mensagem msg3, qualquer outro Agente Utilizador pode igualmente mandar este tipo de mensagem. Este raciocínio é seguido para as restantes mensagens e agentes.

Deste modo, além das classes e das mensagens, também possuímos ficheiros que apresentam conhecimento sobre as estações e os utilizadores. Cada AU consegue aceder a um ficheiro com os seus factos e regras, enquanto que todos os agentes (à exceção do Agente Interface) têm acesso a um ficheiro com o conhecimento relativo ao nome e coordenadas das estações. As setas a tracejado representam este acesso a um ficheiro.

4.1. Criação dos Agentes

Os primeiros agentes a serem criados são os Agentes Estação. Deste modo, são criados dez agentes com nomes identificadores distintos que correspondem às estações do Sistema de Partilha de Bicicletas.

De seguida, é criado um único Agente Interface. Este agente só é criado após uma pausa de um segundo, pois quando fosse ao *Directory Facilitator* verificar as estações existentes, poderia não encontrar nenhuma. Os Agentes Estações poderiam, então, ser mais lentos que o Agente Interface. No entanto, com esta pausa, quando o AI for ao DF já estarão lá todas as estações do sistema.

Por fim, de 15 em 15 segundos são criados novos Agentes Utilizadores. No início, são criados entre 10 a 15 utilizadores (uma maior carga para o sistema) e em todas as outras vezes são criados utilizadores num número entre 5 e 7.

4.2. Comunicação entre os Agentes

4.2.1 Agente Utilizador

Quando o Agente Utilizador é criado, as suas coordenadas iniciais correspondem a uma estação do SPB. Assim, fazendo a analogia com uma situação real em que um indivíduo se aproxima de uma estação com o intuito de requisitar uma bicicleta, o AU tem de pedir à estação em causa se tem ou não uma bicicleta disponível para ele alugar. Assim, envia à estação inicial uma mensagem (**msg11**) do tipo *REQUEST* com conteúdo “bicicleta”. Está, então, a pedir uma bicicleta à estação. Deste modo, receberá dessa estação uma resposta (**msg12**) positiva ou negativa. Se for positiva, é do tipo *AGREE* e tem permissão para alugar uma bicicleta. Caso seja negativa, é do tipo *REFUSE* e o agente não pode requisitar uma bicicleta, pois não há nenhuma disponível. À semelhança da mensagem enviado pelo AU, o conteúdo destas mensagens é “bicicleta”.

Ao longo da sua viagem, o utilizador tem responsabilidade de avisar as estações por onde passa da sua entrada ou saída das áreas de proximidade. Assim, caso entre numa nova área de proximidade, avisa a estação respetiva com uma mensagem (**msg3**) do tipo *INFORM* com o conteúdo “in”. Caso tenha saído da sua área, o tipo da mensagem (**msg3**) permanece o mesmo, porém o conteúdo passa a ser “out”. Além disso, sempre que atualiza a sua posição, o AU verifica se já completou ¾ do percurso total. Caso já o tenha feito, o AU pode, a partir de agora, receber incentivos das estações. No entanto, precisa, também, de mandar uma mensagem (**msg10**) a informar (tipo *INFORM*) a sua estação final que está a chegar, com o conteúdo “coords” e as coordenadas da sua posição atual.

Quando o AU recebe um incentivo cujo valor está acima do seu limite mínimo, considera aceitá-lo. No entanto, continua a ter de pedir à estação para confirmar se ainda tem

um lugar disponível para ele. Assim, manda-lhe uma mensagem (**msg5**) do tipo *REQUEST* com o conteúdo “incentivo aceite”. Caso a estação tenha espaço livre, responde-lhe com uma mensagem (**msg6**) do tipo *CONFIRM*, caso contrário responde com uma mensagem do tipo *DISCONFIRM*. O conteúdo destas mensagens é “lugar reservado”. Desta forma, o AU tem a confirmação que o estado da estação se alterou ou não, isto é, se houve ou não outras estações que aceitaram o incentivo antes dele e que ocupassem todas as vagas da estação.

Finalmente, quando o AU chega ao final do percurso, precisa de entregar a bicicleta na estação. Antes disso, precisa ainda de confirmar com a estação se tem um lugar vago para ele. Assim, manda à estação destino uma mensagem do tipo *REQUEST* com o conteúdo “entrega”. Caso possa entregar, recebe uma resposta do tipo *AGREE*. Caso contrário, o tipo da mensagem é *REFUSE*. O conteúdo de ambas é “entrega”.

4.2.2 Agente Estação

Quando um AU se encontra na área de proximidade de uma estação, esta envia-lhe incentivos (descontos) conforme o seu estado de ocupação. O utilizador, dependendo da sua posição (se se encontra ou não a ¾ do percurso), pode lê-los ou ignorá-los. O tipo da mensagem (**msg4**) enviada é *INFORM* e tem como conteúdo “incentivo “ e o valor do mesmo.

Para prevenir que o utilizador fique muito tempo à espera de entregar a bicicleta, a estação pode pedir a outra que lhe envie um incentivo impossível de recusar, caso a sua ocupação esteja num nível crítico (acima dos 90%). Deste modo, irá mandar às estações na sua proximidade uma mensagem (**msg7**) a pedir o valor de incentivo que estarão preparadas para mandar ao AU. Esta mensagem é do tipo *REQUEST* com o conteúdo “incentivo”. Assim, as estações que recebam esta mensagem responderão com uma mensagem (**msg8**) do tipo *INFORM* e com a ontologia “pedido de incentivo”, cujo conteúdo é o valor de desconto. Após a recolha de todos os incentivos, o AE escolhe o que possuir o valor mais elevado (pois corresponde à estação que tem a menor taxa de ocupação) e envia a essa estação uma mensagem que significa que a estação escolhida tem de enviar ao AU um valor muito alto de desconto. Esta mensagem (**msg9**) é do tipo *INFORM* e o conteúdo é “incentivo max”.

As restantes mensagens que o Agente Estação envia ao Agente Utilizador já foram explicadas na secção do Agente Utilizador.

Por fim, quando o AE recebe uma mensagem do Agente Interface, a estação manda-lhe uma resposta (**msg2**) do tipo *INFORM* cujo conteúdo corresponde aos valores de certos dados que está a armazenar, tais como o número de alugueres, o número de devoluções, o número de entradas na sua APE, a capacidade atual, a taxa de ocupação e o número de pedidos de aluguer de bicicletas feitos.

4.2.3 Agente Interface

Este agente apenas comunica com o Agente Estação, pois tem como objetivo recolher vários dados estatísticos para os mostrar em forma de gráficos. Deste modo, envia a todas as estações inscritas no *Directory Facilitator* (ou seja, pertencentes ao SPB) uma mensagem (**msg1**) cujo tipo é *REQUEST* e conteúdo é “stats”.

5. Disposição das Estações

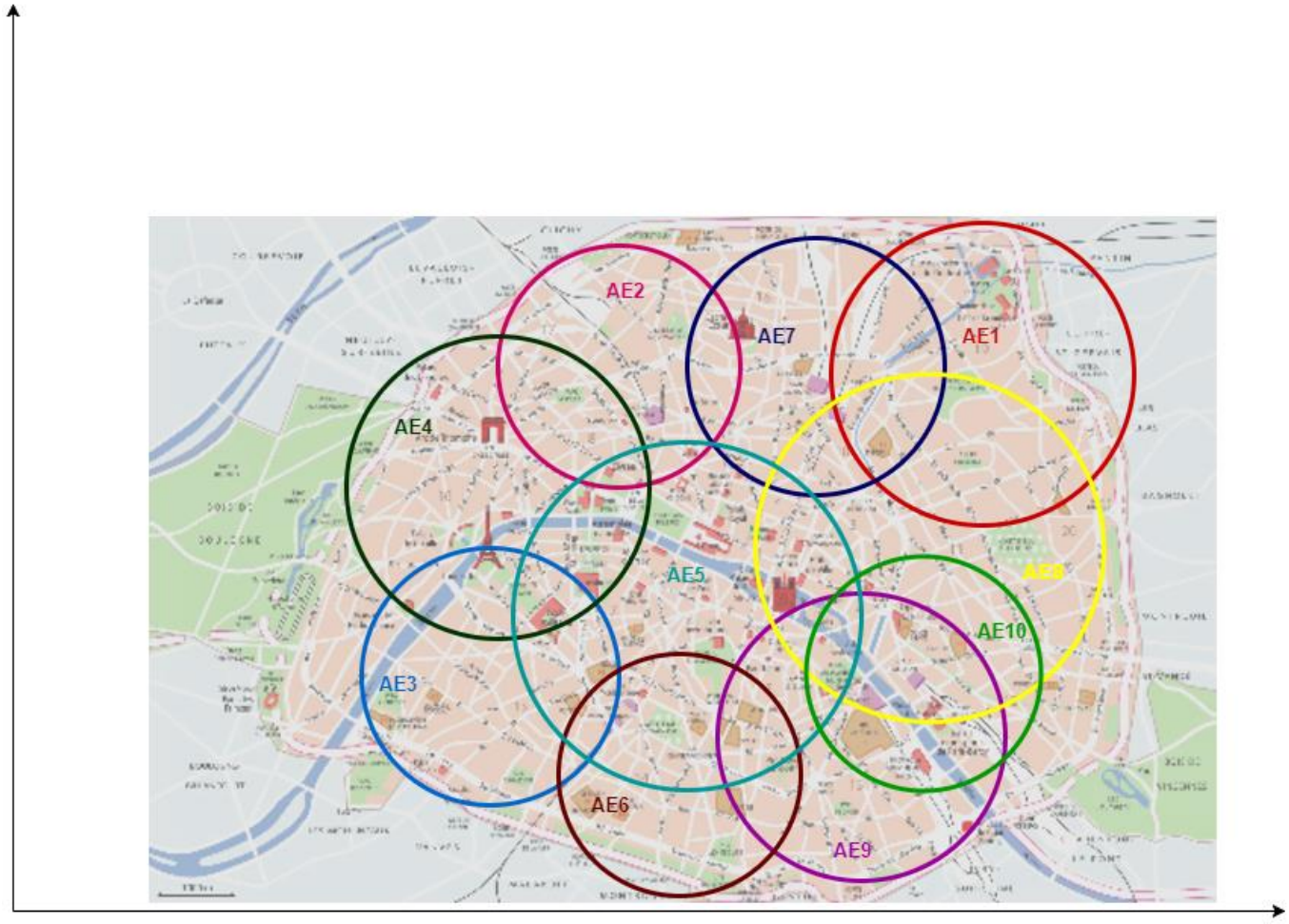


Figura 3 - Disposição das Estações

No Sistema de Partilha de Bicicletas temos um total de 10 estações espalhadas como se mostra no gráfico acima. Assumiu-se que a disposição das estações teria por base uma cidade fictícia e que as estações com maior área se situariam em zonas do centro dessa cidade ou zonas com uma grande afluência. Existem, pois, estações com APE maiores do que outras. Assim, durante o trajeto de um utilizador, existe uma grande probabilidade de ele passar por várias áreas de estações, aumentando, então, a oportunidade de entregar a bicicleta numa estação que não a final.

6. Arquitetura dos ficheiros .clp

Os ficheiros com extensão .clp foram utilizados pelo motor de inferência do JESS, de modo a aceder ao conhecimento que estava representado neles. Os dois tipos de ficheiros existentes são o Estacoes.clp e o AU*.clp (em que * representa o identificador do utilizador, podendo ser definido pela expressão regular [1-9][0-9]*).

6.1. Estacao.clp

O ficheiro Estacao.clp apresenta os dados relativos às estações, estando representado um *template* de como os factos devem estar apresentados, devendo a estação possuir um nome, a sua capacidade máxima, o raio que a sua área atinge (as estações apresentam uma área circular) e as coordenadas do centro da estação.

```
(deftemplate estacoes
	(slot nome)
	(slot capacidadeMax)
	(slot raio)
	(multislot coords)
)
```

Como as especificações das estações não estão dependentes do funcionamento do programa e são definidas antes de este ser executado, podemos adicionar o conhecimento sobre elas através de *deffacts*, que através do comando *(reset)* adiciona todos os factos contidos em *deffacts*.

```
(deffacts factos
	(estacoes (nome AE1)(capacidadeMax 10)(raio 2.5)(coords 16 9))
	(estacoes (nome AE2)(capacidadeMax 10)(raio 2)(coords 10 9))
	(estacoes (nome AE3)(capacidadeMax 10)(raio 2.3)(coords 8 4))
	(estacoes (nome AE4)(capacidadeMax 10)(raio 2.5)(coords 8 7))
	(estacoes (nome AE5)(capacidadeMax 10)(raio 3)(coords 11 5))
	(estacoes (nome AE6)(capacidadeMax 10)(raio 2.5)(coords 11 2))
	(estacoes (nome AE7)(capacidadeMax 10)(raio 2.3)(coords 13 9))
	(estacoes (nome AE8)(capacidadeMax 10)(raio 3)(coords 15 6))
	(estacoes (nome AE9)(capacidadeMax 10)(raio 2.4)(coords 14 3))
	(estacoes (nome AE10)(capacidadeMax 10)(raio 2)(coords 15 4))
)
```

Por fim, o ficheiro contém ainda uma *query* que, quando executada, devolve os factos referentes a todas as estações representadas.

```
(defquery procuraEstacoes
	(estacoes (nome ?n)(raio ?r)(capacidadeMax ?max)(coords ?x ?y))
)
```

6.2. Ficheiro AU*.clp

O sistema de inferência de cada Agente Utilizador irá aceder a um ficheiro AU*.clp, que apresenta conhecimento específico a esse agente utilizador, ou seja: para cada agente utilizador, existe um ficheiro AU*.clp. O conhecimento do utilizador está definido no seguinte *template utilizador*, que apresenta o incentivo mínimo que o utilizador aceitaria, a velocidade a que se desloca, o tempo (constante, serve para determinar a distância percorrida), a distância total entre a estação inicial e final, a distância percorrida nesse momento, o nome da estação final, as coordenadas atuais, e as coordenadas finais (referentes à estação final).

```
(deftemplate utilizador
	(slot incentivoMin)
	(slot velocidade)
	(slot tempo)
	(slot distTotal)
	(slot distPercorrida)
	(slot estacaoFinal)
	(multislot coordsAtuais)
	(multislot coordsFim))
```

Existe também o template incentivo que irá guardar o valor dos incentivos recebidos pelas diversas estações cuja área interseta. Denote-se que há um caso em que o utilizador receberá incentivos de estações que não interseta, mas será visto no futuro.

```
(deftemplate incentivo
	(slot desconto)
	(slot estacao))
```

Para determinar qual estação envia o melhor incentivo, foi criada uma *query procuraIncentivos* que procura todos os factos relativos aos incentivos. Assim, na porção de código JAVA, podemos iterar sobre os resultados da *query* e determinar que incentivo se deve aceitar (caso seja superior ao incentivo mínimo definido pelo utilizador).

```
(defquery procuraIncentivos
	(incentivo (desconto ?d)(estacao ?e))
)
```

A query procuraUtilizadores funciona de modo análogo à procuraIncentivos, retornando, no entanto, os factos referentes ao utilizador. Como há um ficheiro para cada utilizador, esta *query* só retornará um facto, sendo o objetivo principal aceder a uma das *slots* do utilizador em causa.

```
(defquery procuraUtilizadores
```

```
      (utilizador(incentivoMin    ?i)(velocidade    ?v)(tempo    ?t)(distTotal
?dt)(estacaoFinal ?e)(distPercorrida ?dp)(coordsAtuais ?x0 ?y0)(coordsFim ?x1
?y1))
)
```

7. Behaviours

Os *behaviours* (ou comportamentos de um agente) são fundamentais para a correta implementação da solução ao problema proposto. Nesta secção serão discriminados os diferentes *behaviours* que cada agente possui, bem como quando são chamados/executados.

7.1. Agente Utilizador

7.1.1 AtualizaPosicao do tipo TickerBehaviour

Este comportamento é executado a cada segundo e atualiza a posição do Agente Utilizador passado esse segundo.

O primeiro passo necessário será utilizar o motor de inferência do JESS para obter a velocidade média, o tempo que demora a percorrer um troço, as coordenadas atuais do utilizador, as coordenadas da estação final, a distância total a percorrer e a distância percorrida. Estes fatores serão necessários a calcular as novas coordenadas do utilizador. Assim, temos como resultado as seguintes fórmulas:

1. $dis = v \times t$, em que *dis* é a distância percorrida em *t* tempo e *v* de velocidade;
2. $disPercorrida = disPercorrida + dis$, em que *disPercorrida* é a distância percorrida;
3. $novoX = x_0 + \frac{dis \times (x_1 - x_0)}{disTotal - disPercorrida}$, em que x_0, x_1 são as coordenadas *x* da estação atual e da estação final, respetivamente. *novoX* será o novo valor da coordenada *x* após ter percorrido o troço *dis*.
4. $novoY = y_0 + \frac{dis \times (y_1 - y_0)}{disTotal - disPercorrida}$, em que y_0, y_1 são as coordenadas *y* da estação atual e da estação final, respetivamente. *novoY* será o novo valor da coordenada *y* após ter percorrido o troço *dis*.

As variáveis *novoX* e *novoY* são calculadas através da semelhança de triângulos, que pode ser melhor visualizada na seguinte figura.

$$\frac{novoX}{x1-x0} = \frac{dis}{disTotal-disPercorrida}$$

$$\frac{novoY}{y1-y0} = \frac{dis}{disTotal-disPercorrida}$$

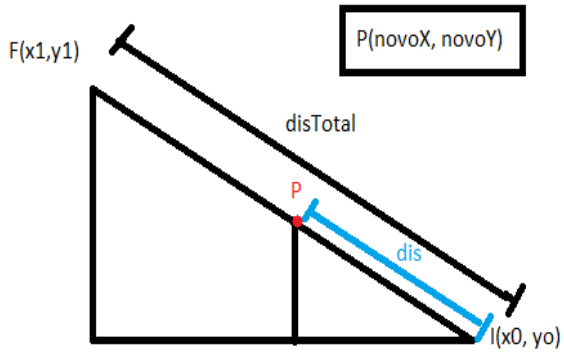
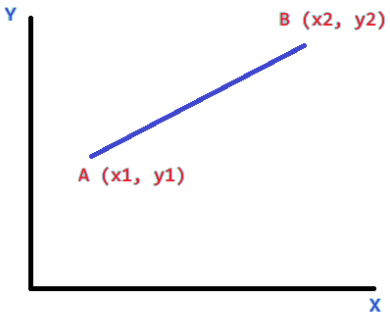


Figura 4 - Exemplificação do cálculo da nova posição do utilizador

Após terem sido efetuados os cálculos, alteram-se as novas coordenadas do utilizador bem como a distância percorrida do facto que constava na base de conhecimento do sistema de inferência. É importante reparar que este programa é uma simulação, logo é possível que tanto as coordenadas do utilizador, como a distância percorrida podem tomar valores após a estação final, daí ser necessário verificar se isso acontece antes de modificar o facto. Se tal acontecer, as coordenadas atuais tornam-se nas finais e a distância percorrida na distância total.

Após esta fase, é necessário ver se, mediante as novas coordenadas, o utilizador entrou na área de alguma estação, se saiu, se já percorreu ¾ do percurso ou se já atingiu o destino final.

A estratégia para determinar se um utilizador alterou a área em que se encontrava, passa por percorrer todas as estações e calcular a distância das coordenadas do utilizador ao centro da estação. A distância entre os dois pontos é dada através da distância entre dois pontos, exemplificada na seguinte imagem:



$$AB = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Figura 5 - Exemplificação do cálculo da nova posição do utilizador

Assim sendo, temos dois casos:

1. **AB > raio da estação:** o utilizador encontra-se fora da área da estação, logo é necessário verificar se esta estação estava previamente armazenada na lista de estações em que o utilizador se encontrava. Caso isto aconteça, remove-se a estação da lista e informa-se a respetiva estação de que se saiu da área dela.

2. **AB ≤ raio da estação:** o utilizador encontra-se dentro da área da estação, logo é necessário adicionar a estação à lista de estações em que o utilizador se encontra (caso não exista) e notifica-se a estação respetiva que se entrou na área dela.

De seguida, há que verificar se o utilizador ultrapassou ¾ do percurso total. Caso o tenha feito e ainda não tenha notificado a estação final (esta verificação serve para não "entupir" a estação final com mensagens de algo que ela já tem conhecimento) é necessário notificar a estação final dizendo que se encontra a ¾ do percurso e enviar-lhe as coordenadas finais.

Finalmente, verificamos se o utilizador atingiu o destino final. Caso seja verdade, removemos o *behaviour* AtualizaPosicao (visto que já não precisa de se deslocar novamente) e adicionamos o *behaviour* FimDoPercurso.

7.1.2 FimDoPercurso do tipo CyclicBehaviour

Este comportamento serve para retratar as ações que o utilizador deve tomar quando chega ao destino. O principal objetivo deste comportamento será notificar a estação final de que o utilizador chegou e que quer entregar a bicicleta.

Assim sendo, o utilizador irá perguntar à estação final se tem capacidade para receber a bicicleta. Após recebida a resposta, caso esta seja afirmativa, o agente morre, caso seja negativa espera 10 segundos e tenta novamente entregar a bicicleta. Como o comportamento é do tipo cíclico, este processo será repetido até conseguir entregar a bicicleta. Denote-se que, através dos incentivos, este cenário torna-se improvável e, de modo a simular melhor o comportamento humano, o agente espera até conseguir entregar a bicicleta.

No entanto, este processo não ocorre, caso o agente tenha aceitado um incentivo. Nesse caso, o agente morre, visto que este caso já está tratado no RecebeIncentivo.

7.1.3 RecebeIncentivo do tipo CyclicBehaviour

Este comportamento encarrega-se de receber os incentivos das estações e decidir se algum desses incentivos é vantajoso. Este comportamento é adicionado ao agente no início da execução do agente utilizador, no entanto só começa a considerar os incentivos após ter atingido ¾ do percurso total. Deste modo, impede que o utilizador aceite um incentivo muito próximo do ponto de partida.

Assim, caso a mensagem recebida do agente estação seja relativa ao incentivo, podemos começar a examinar o conteúdo. Primeiramente, retiramos o conteúdo dessa mensagem (valor do incentivo) e o remetente (necessário para informar a estação se aceita ou não o incentivo). Em segundo lugar, utilizamos o motor de inferência para obter o valor do incentivo mínimo que o utilizador estaria disposto a aceitar. Se o incentivo recebido for superior ao incentivo mínimo, o incentivo é aceite pelo utilizador. Assim, notificamos a estação que enviou o incentivo e esperamos uma de duas respostas:

1. **Rejeita o incentivo previamente enviado.** Esta situação ocorre caso a estação tenha atingido a capacidade máxima entre a troca de mensagens. Neste caso, o utilizador continua o seu percurso, disposto a receber outros valores de incentivos.
2. **Aceita o incentivo previamente enviado.** Através do motor de inferência do JESS, obtemos os dados relativos à nova estação final e modificamos o facto existente de modo a que as novas coordenadas finais correspondam à nova estação final. De seguida, remove-se este comportamento do agente, visto que não está disposto a receber mais nenhum incentivo.

Resumindo, há duas situações que podem ocorrer neste comportamento. Ou o incentivo é rejeitado ou é aceite e o agente continua a executar o comportamento AtualizaPosicao até atingir o destino final.

7.2. Agente Estação

7.2.1 RecebePedido do tipo CyclicBehaviour

O *behaviour* RecebePedido é um comportamento que se encarrega de receber as mensagens do tipo *REQUEST* e trata-las mediante o seu conteúdo. Para filtrar as mensagens desejadas utilizadas, utilizou-se um *MessageTemplate* para corresponder a *performative* à desejada:

```
MessageTemplate mt = MessageTemplate.MatchPerformative(ACLMessage.REQUEST);
```

Se a mensagem recebida não for nula, podemos examinar o conteúdo e efetuar ações sobre ele:

1. **Caso o conteúdo seja "stats":** o agente estação recebe o pedido das suas estatísticas proveniente do agente interface. Assim, o agente estação compõe uma *string* com as variáveis desejadas e envia ao agente estação.
2. **Caso o conteúdo seja "incentivo":** neste caso, o agente estação recebe um pedido de outro agente estação para que lhe envie o seu incentivo. O agente estação calcula o seu valor de incentivo e responde à estação que lhe enviou o pedido. Denote-se como a mensagem a enviar é do tipo *INFORM*, será necessário definir uma ontologia de modo a que o *behaviour* Controlador não capte esta mensagem e que seja captada pelo *behaviour* certo (que será o PedeIncentivoEstacao).
3. **Caso o conteúdo seja "bicicleta"**: o agente estação recebe um pedido do agente utilizador para reservar uma bicicleta. Inicialmente, incrementa-se o número de pedidos de bicicletas (para fins estatísticos). Depois, verifica se existem bicicletas disponíveis. Se não existirem, notifica-se o agente estação que não pode requisitar (*ACLMessage.REFUSE*). Caso existam, antes de notificar o agente que a bicicleta pode ser levantada, devem-se efetuar dois passos. Decrementar o número de bicicletas disponíveis, incrementar o número de alugueres efetuados (para fins estatísticos) e adicionar o utilizador à lista de utilizadores contidos na sua área:
   ```
   utilsArea.add(sender);
   ```
   Finalmente, envia-se a confirmação ao utilizador.
4. **Caso o conteúdo seja "entrega":** existem dois casos possíveis quando o utilizador tenta entregar uma bicicleta. O primeiro acontece quando não dá para

entregar a bicicleta (a estação está na lotação máxima), sendo necessário notificar o utilizador que a entrega não foi bem sucedida (*ACLMessage.REFUSE*). O segundo ocorre quando é possível efetuar a entrega. Incrementamos o número de bicicletas disponíveis, bem como o número de devoluções efetuadas (para fins estatísticos), removendo o utilizador da lista de utilizadores contidos na sua área:

```
utilsArea.remove(sender);
```

Finalmente, notificamos o utilizador que a bicicleta foi entregue (*ACLMessage.AGREE).*

5. **Caso o conteúdo seja "incentivo aceite":** este pedido por parte do utilizador, serve para confirmar o incentivo que acabou de aceitar. Se a estação que enviou o incentivo tiver atingido a lotação máxima entretanto, não pode confirmar o pedido, sendo necessário notificar o utilizador que o incentivo é inválido. Se a estação possuir vagas, confirma o valor do incentivo, remove o utilizador da lista de utilizadores contidos na sua área:

```
utilsArea.remove(sender);
```

incrementa o número de bicicletas disponíveis e o número de devoluções, enviando a confirmação ao agente utilizador.

7.2.2 Controlador do tipo CyclicBehaviour

O *behaviour* Controlador é um comportamento que se encarrega de receber as mensagens do tipo *INFORM* e trata-las mediante o seu conteúdo. Para filtrar as mensagens desejadas utilizadas, utilizou-se um *MessageTemplate* para corresponder a *performative* à desejada:

```
MessageTemplate mt = MessageTemplate.MatchPerformative(ACLMessage.INFORM);
```

Se a mensagem recebida não for nula, podemos examinar o conteúdo e efetuar ações sobre ele:

1. **Conteúdo é "in":** a estação é notificada que um agente entrou na sua área, logo é necessário incrementar o número de entradas (para fins estatísticos), adicionar o utilizador à lista de utilizadores presentes na sua área:

```
utilsArea.add(sender);.
```

2. **Conteúdo é "out":** o utilizador saiu da sua área e será necessário removê-lo da lista de utilizadores presentes na sua área.

```
utilsArea.remove(sender);
```

3. **Conteúdo é "incentivo max":** outra estação notifica a atual que deve enviar o valor máximo de incentivo a um determinado utilizador. Através do método *createReply* sabemos a quem a mensagem deve ser direcionada, só temos de alterar o conteúdo para o valor do incentivo máximo a ser enviado. Como os valores do incentivo dos utilizadores variam no intervalo [0.4,0.7] o valor máximo será 0.75.
4. **Conteúdo é "coords x y", em que x e y são coordenadas:** esta mensagem é recebida quando o utilizador quer informar a estação final que atingiu ¾ do percurso. Após ter sido efetuado o *parse* corretamente, devemos adicionar o utilizador criado à lista de utilizadores que querem acabar o percurso nessa estação:

```
Utilizador u = new Utilizador(sender, x, y);
utilsFinal.put(u.getNome(), u);
```

Finalmente, para resolver a problemática da distribuição das bicicletas, verificamos se a ocupação atual ultrapassa a ocupação crítica (90%), devemos inicializar o behaviour Emergencia.

7.2.3 EnviaIncentivo do tipo TickerBehaviour

Este *behaviour,* iniciado no início da execução do agente estação, é executado a cada segundo e, para todos os utilizadores contidos na sua área, envia o seu valor do incentivo. Assumindo que o incentivo foi calculado previamente, a seguinte porção de código só será executada se a lista de utilizadores contidos na sua área não estiver vazia.

```
ACLMessage acl = new ACLMessage(ACLMessage.INFORM);
acl.setContent(incentivo + "");
for(String nome : utilsArea)
	if(!utilsFinal.containsKey(nome))
		acl.addReceiver(new AID(nome, AID.ISLOCALNAME));

if(incentivo > 0.0f)
	send(acl);
```

Assim, para cada utilizador contido na área, verificamos se não termina o percurso nesta estação, pois não queremos enviar incentivos a utilizadores que já planeiam acabar nesta estação. Caso não acabem, adicionamos o AID do utilizador à mensagem. Finalmente, caso o incentivo seja maior que zero (zero significa que a estação está lotada), a mensagem é enviada.

7.2.4 Emergencia do tipo SimpleBehaviour

Este comportamento é executado quando a estação está lotada e possui utilizadores que lhe pretendem entregar a bicicleta. O objetivo é pedir o valor do incentivo a outras estações, de modo a escolher uma que esteja melhor capacitada para receber o utilizador.

Inicialmente, temos de inicializar três *maps*: um que irá ser preenchido com todas as estações existentes, outro que irá ser preenchido com as estações intermédias e o último que irá ser preenchido com o valor dos incentivos de cada estação.

Assim, temos de preencher o *map* de todas as estações. Utilizamos o motor de inferência JESS para aceder ao ficheiro estacoes.clp (ficheiro que contém os factos sobre todas as estações) e iteramos sobre o resultado, adicionando as estações ao *map* (sendo elas o valor), sendo a chave o nome delas. Para facilitar os cálculos posteriores, quando encontrarmos a estação atual, guardamos as suas coordenadas em variáveis. Esta explicação corresponde ao seguinte excerto de código:

```
while(rs.next())
{
```

```
            e    =    new    Estacao(rs.getString("n"),    rs.getDouble("x"),
      rs.getDouble("y"), rs.getFloat("r"));

            if(e.getEstacao().equals(myAgent.getLocalName()))
      {
            xf = rs.getFloat("x");
            yf = rs.getFloat("y");
      }
      todasEstacoes.put(e.getEstacao(), e);
      }
```

O próximo passo consiste em iterar sobre o *map* que contém todas as estações e verificar quais dessas estações interseta o segmento de reta definido pelo ponto de ¾ do percurso e o centro da estação final. Esta interseção é calculada através do método *intersetaArea* que verifica se um determinado segmento de reta e uma circunferência se intersetam. Caso haja interseção e a estação onde ocorra não seja a final, adicionamos esta estação ao *map* de estações intermédias.

No entanto pode ocorrer um problema, caso não exista nenhuma interseção. A solução encontrada consiste em considerar uma área cujo centro é o da estação final e o raio é a distância do ponto dos ¾ e o centro da estação final e adicionar qualquer estação cujo centro se encontre dentro dessa área ao *map* de estações intermédias. Assim, alargamos o horizonte sem obrigar o utilizador a percorrer distâncias enormes para entregar a bicicleta. Finalmente, removemos a estação final do *map* das estações intermédias. Denote-se que a distância entre o centro da estação final e o centro da estação a considerar em cada iteração é calculada tal como foi calculada previamente para determinar se o utilizador tinha saída de uma área. De seguida, apresenta-se o código referente a este parágrafo.

```
      if(estacoesIntermedias.isEmpty())
      {
            double raio = sqrt(pow(xi - xf, 2)  + pow(yi - yf, 2));
      for(Map.Entry<String, Estacao> entry : todasEstacoes.entrySet())
      {
            double dist = sqrt(pow(xf - entry.getValue().getCoordX(), 2)  +
pow(yf - entry.getValue().getCoordY(), 2));
            if(dist < raio)
            {
                  estacoesIntermedias.put(entry.getKey(),
                entry.getValue());
            }
      }
      }
      estacoesIntermedias.remove(myAgent.getLocalName());
```

Neste momento, já sabemos a que estações temos de pedir o incentivo, só resta contactá-las. De modo a tornar o processo mais paralelo, criou-se um *ParallelBehaviour* (chamado estacaoCheia). A este *behaviour* irão ser adicionados *SubBehaviours*, cuja função é pedir o incentivo às estações contidas no *map* das estações intermédias. Por fim, adiciona-se este comportamento ao agente, como pode ser visto no seguinte excerto de código:

```
for(Map.Entry<String, Estacao> entry : estacoesIntermedias.entrySet())
```

```
{
	estacaoCheia.addSubBehaviour(
		new  PedeIncentivoEstacao(entry.getKey()));
}
myAgent.addBehaviour(estacaoCheia);
```

7.2.5 PedeIncentivoEstacao do tipo SimpleBehaviour

Este *behaviour* é executado pelo *ParallelBehaviour* e tem como objetivo enviar o valor do incentivo à estação que o pediu. De modo a evitar possíveis *deadlocks* que poderiam surgir de duas estações bloqueadas à espera da resposta uma da outra, implementou-se um *timeout* dentro do agente, que a cada execução (desde que ele não tenha terminado) verifica se esse *timeout* foi ultrapassado.

Caso tenha ultrapassado, adiciona-se ao *map* dos incentivos para cada estação o valor 0, para que este não seja contabilizado e termina-se o agente. Caso o *timeout* não tenha sido ultrapassado, envia-se um pedido à estação passada como parâmetro do construtor deste *behaviour* do seu valor de incentivo. O código seguinte representa o que acontece após o envio da mensagem.

```
MessageTemplate mt1 = MessageTemplate.MatchPerformative(ACLMessage.INFORM);
MessageTemplate mt2 = MessageTemplate.MatchOntology("pedido de incentivo");
MessageTemplate mt3 =
		MessageTemplate.MatchSender(new AID(nome, AID.ISLOCALNAME));
MessageTemplate mt4 = MessageTemplate.and(mt1, mt2);
MessageTemplate mt = MessageTemplate.and(mt3, mt4);
ACLMessage acl = receive(mt);
if(acl != null)
{
	float inc = Float.parseFloat(acl.getContent());
	incentivos.put(nome, inc);
	done = true;
}
```

O facto da mensagem a ser recebida ser do tipo *INFORM* poderia causar confusão com o *behaviour* Controlador, sendo necessário captar a ontologia da mensagem como “pedido de incentivo”. Assim, para estas condições e a condição de que a mensagem recebida tinha de surgir da estação a quem foi enviado o incentivo, temos os *templates* definidos para a mensagem. Se recebermos a mensagem corretamente, colocamos no *map* incentivos o nome da estação e o seu incentivo, marcando depois a variável done a *true*, indicando que o *behaviour* terminou e não há mais nada a executar.

7.2.6 estacaoCheia do tipo ParallelBehaviour

O desejo na conceção deste agente passa por saber o valor de incentivo de todas as estações pertences ao *map* das estações intermédias, logo o código que o *ParallelBehaviour* irá executar quando **todos** os seus SubBehaviours terminarem, irá determinar qual o maior

valor presente no *map* dos incentivos, notificando depois a estação a que pertence esse incentivo que deve enviar o valor máximo do incentivo ao utilizador. A estação que recebe esta mensagem saberá quem é o utilizador, pois a mensagem conterá o AID do agente a que a resposta deve ser enviada. Finalmente, removemos o utilizador da lista de utilizadores contidos na área da estação. De seguida, apresenta-se o código do *ParallelBehaviour:*

```
ParallelBehaviour estacaoCheia =
	new ParallelBehaviour(myAgent, ParallelBehaviour.WHEN_ALL)
	{
	public int onEnd()
	{
		float max = 0.0f;
		String nomeMAX="";
		for(Map.Entry<String, Float> entry : incentivos.entrySet())
		{
			if(entry.getValue() > max)
			{
				max = entry.getValue();
				nomeMAX = entry.getKey();
			}
		}

		if(max == 0.0f)
			System.out.println("A estação " + myAgent.getLocalName()
+ " não obteve ajuda de nenhuma estação.");
		else
		{
			System.out.println("Emergência: o melhor incentivo
recebido por "+ myAgent.getLocalName() + " foi " + max + " e pertence à
estação " + nomeMAX);
			ACLMessage acl = new ACLMessage(ACLMessage.INFORM);
				acl.setContent("incentivo max");
			acl.addReceiver(new AID(nomeMAX, AID.ISLOCALNAME));
			acl.addReplyTo(new AID(nome, AID.ISLOCALNAME));
			send(acl);
			utilsFinal.remove(nome);
		}
		return 1;
	}
};
```

7.3. Agente Interface

7.3.1 PedeEstatisticas do tipo TickerBehaviour

Este comportamento é executado a cada 10 segundos e tem como objetivo pedir os dados estatísticos de cada estação para depois mostrar ao utilizador (quem executar o programa).

Este *behaviour* executa um *ParallelBehaviour* em que os *SubBehaviours* (estatisticas) adicionados têm como função pedir as estatísticas a cada estação. Assim, para todos os

agentes registados no *DF*, adicionamos um *SubBehaviour* ao *ParallelBehaviour.* Finalmente, adicionamos este *behaviour* ao agente.

7.3.2 EstatisticasEstacao do tipo SimpleBehaviour

Este comportamento tem como objetivo pedir as estatísticas a uma estação e inserir as estatísticas referidas num *map* cuja chave é o AID da estação e o valor é do tipo InfoEstacao (classe que contém as variáveis que representam as estatísticas). Após ter sido enviado o pedido das estatísticas e ser recebida a resposta, efetua-se o parse da *string* e armazenam-se os valores no *map*. O conteúdo da mensagem referida apresenta a seguinte forma:

```
"alugueres devolucoes entradas bicicletasarmazenadas ocupacao numPedidos"
```

em que as palavras são substituídas pelos valores correspondentes. Estes valores inserem-se na classe InfoEstacao, remove-se o valor prévio existente no map e insere-se o novo valor. O seguinte excerto de código representa a última explicação:

```
String content = msg.getContent();
String [] comp = content.split(" ");
InfoEstacao ie = info.get(aid);

ie.setNumAlugueres(Integer.parseInt(comp[0]));
ie.setNumDevolucoes(Integer.parseInt(comp[1]));
ie.setNumEntradas(Integer.parseInt(comp[2]));
ie.setNumBicicletas(Integer.parseInt(comp[3]));
ie.setOcupacao(Float.parseFloat(comp[4]));
ie.setNumPedidos(Integer.parseInt(comp[5]));

info.remove(aid);
info.put(aid, ie);
```

7.3.3 estatisticas do tipo ParallelBehaviour

Este comportamento é utilizado para mostrar os valores das estatísticas numa interface gráfica. Quando todos os *SubBehaviours* deste comportamento terminarem, é chamado o método estatisticasInterfaceGrafica que irá percorrer o *map* que contém as estatísticas de cada estação e irá ilustrar esses valores no ecrã.

De seguida, apresenta-se um exemplo de um estado do sistema ilustrado pela interface gráfica.

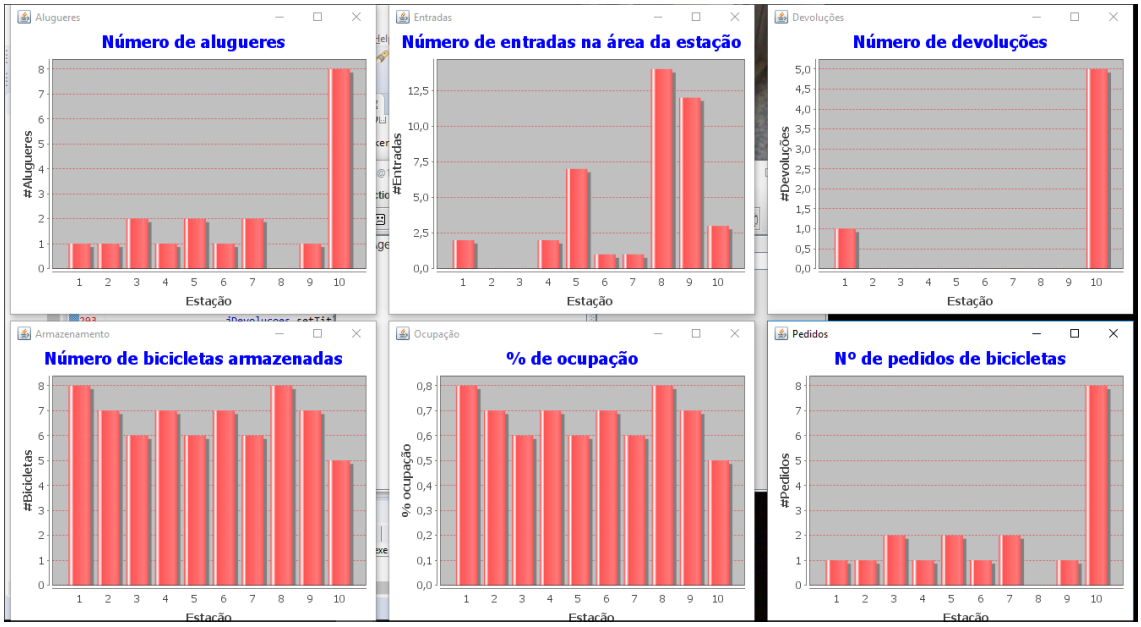


Figura 6 - Exemplo de resultados da execução do programa

8. Conclusões

A primeira fase do projeto permitiu ao grupo analisar todas as vertentes do problema, as questões que poderão surgir ao longo do desenvolvimento do projeto, bem como possíveis situações de conflito entre agentes. Assim, conseguimos arranjar diversas soluções, que serão tidas em conta na fase seguinte, a fase de geração de código.

A arquitetura que será a base do programa foi também concebida, assim como as trocas de mensagens entre os três tipos de agentes. Estabeleceu-se desde já qual o comportamento que cada um dos agentes deve seguir, quais os seus objetivos e qual o seu contexto.

A segunda fase, por outro lado, focou-se na implementação do SPB e seus constituintes: Agente Utilizador, Agente Estação e Agente Interface. Seguiu-se muito proximamente o que foi planeado na fase anterior, pois não foi preciso fazer grandes mudanças do que havia sido planeado previamente. Assim, pensa-se que se conseguiu desenvolver uma boa solução para a temática do trabalho prático, uma vez que o problema da distribuição de bicicletas no sistema foi desenvolvido. A sua eficiência varia, claro, em diferentes execuções do programa, pois como o local de criação e destino dos agentes são atribuídos aleatoriamente, a solução varia a sua prestação. No entanto, nunca se registaram casos em que grandes variações nos estados das estações se verificaram.

Em suma, o planeamento e levantamento de todas as possíveis questões do problema foi muito útil para a parte de conceção de código, pois uma vez planeadas as soluções para os problemas que poderiam surgir, a criação de código para as pôr em prática foi simplificada significativamente.

Lista de Siglas e Acrónimos

SPB	Sistema de Partilha de Bicicletas
PRPB	Problema do Reequilíbrio de Partilha de Bicicletas
APE	Área de Proximidade de uma Estação
AU	Agente Utilizador
AE	Agente Estação
AI	Agente Interface