Sistemas de Partilha de Bicicletas - SPB

André Figueiredo, Luís Ferreira, Pedro Machado, and Rafael Lourenço

Universidade do Minho, Departamento de Informática, 4710-057 Braga, Portugal

Agentes Inteligentes, Trabalho Prático Nº2 - Fase 2

2 de Janeiro de 2021

e-mail: {a84807,a86265,a83719,a86266}@alunos.uminho.pt

Resumo Neste relatório é apresentada uma arquitetura distribuída de um sistema multi-agente para um Sistema de Partilha de Bicicletas.
A criação de sistemas de transportes públicos ou partilhados é algo que tem ganho cada vez mais importância. Esta prática permite o descongestionamento e redução de trânsito (principalmente em áreas de grande densidade populacional), bem como a diminuição de emissões e poluição. Para além disso, estudos afirmam que a utilização de bicicletas como meio de transporte para os locais de trabalho permite, por exemplo, reduzir os níveis de stress e melhorar a produtividade. [1]
Além disso, trata-se de um meio de transporte relativamente barato, que permite a prática de exercício físico, sendo que existem cada vez mais iniciativas a serem criadas pelos governos para tornar este meio de transporte cada vez mais aliciante.

Keywords: Artificial Intelligence · Intelligent Agents · Multi-Agent Systems · Public Bike Share · Bicycle Sharing Systems · Balancing Bicycle Sharing Systems

1 Introdução

No âmbito desta Unidade curricular foi-nos proposta a modelação e criação de um Sistema de Partilha de Bicicletas (`SPB`), que permitisse o seu balanceamento dinâmico.
Um dos maiores problemas de um Sistema de Partilha de Bicicletas é o desequilíbrio de bicicletas em certas estações ao fim de algum tempo, pois, sem intervenção ”inteligente”, algumas estações terão falta de bicicletas, enquanto que outras terão excesso delas, o que impede a ocorrência de devoluções. Portanto, para existir um sistema que agrade aos utilizadores sem que se verifiquem quebras de receitas, devem ser criados mecanismos que permitam o balanceamento do sistema. Assim, é aqui proposta uma modelação de um sistema multi-agente para partilha de bicicletas entre utilizadores, onde é permitido realizar viagens entre as várias estações espalhadas por uma cidade.
Nesta aplicação existirão três tipos de agentes - um agente que representará cada um dos utilizadores do sistema, outro que representará cada uma das estações (de forma a monitorizar o seu estado) e um agente interface responsável pela apresentação de dados estatísticos e por auxiliar os agentes anteriormente referidos a comunicar. Os dois primeiros podem ser vistos como uma espécie de sensores, por exemplo, sensores de `GPS` e de ocupação, enquanto que o último se trata de uma camada de representação/comunicação, de forma a facilitar a interação entre utilizadores e estações, assim como com um administrador.
Desta forma, começámos por fazer uma breve introdução e descrição do problema, seguido da apresentação da arquitetura geral do sistema, descrição dos agentes, comportamento destes e a comunicação/negociação entre os vários agentes.

A modelação deste projeto é baseada em metodologias `Agent UML`, que permitem obter uma descrição detalhada dos protocolos de comunicação, processamento interno e interações entre agentes.
Na segunda parte deste trabalho, passaremos a explicar que decisões foram tomadas e a razão que nos levou a escolhê-las e como foi feita a implementação recorrendo à linguagem `Java` em conjunto com a biblioteca `Jade`. Para a apresentação dos dados e diagramas foi utilizada a biblioteca `jfreechart`.

Considerando a situação atual do nosso país, já foram implementados, pelo menos, seis Sistemas de Partilha de Bicicletas: [2]

- `beÁgueda`, cidade de Águeda em 2011, conta com 4 estações e 20 bicicletas;
- `b→AND`, cidade de Anadia em 2014, conta com 11 bicicletas;
- `Buga`, cidade de Aveiro em 2000, conta com 1 estação e 300 bicicletas;
- `biCas`, cidade de Cascais em 2016, conta com 80 estações e 700 bicicletas;
- `GIRA`, cidade de Lisboa em 2017, conta com 74 estações e 700 bicicletas;
- `biConde`, cidade de Vila do Conde em 2014, conta com 10 estações e 60 bicicletas.

1.1 Descrição do problema

Analisando o enunciado, o principal objetivo deste sistema passa por gerir o `SPB`. Para isso, é necessário levantar certos requisitos, como, por exemplo, as entidades e funcionalidades que devem ser suportadas.
Quanto às entidades (”físicas”), existem dois tipos - estações e utilizadores.
Para permitir a monitorização dos sensores do sistema (`GPS`, entre outros), é necessário criar uma arquitetura que permita a gestão dos vários sensores e a comunicação entre as várias entidades (agentes).

Sendo assim, esta arquitetura pode ser divida em três subsistemas principais:

- **Interface**, responsável pela gestão do sistema, desde o registo de estações até à apresentação de dados relacionados com as estações pertencentes ao sistema, ocupação de cada uma destas, entre outros. Também foi implementado um mapa, que permite emular o movimento e posição de cada agente no sistema, entre outras características;

- **Utilizador e sensores**, onde existem quatro funções principais, sendo estas:

  - **Pedido de Aluguer**, isto é, ter um mecanismo para permitir que seja estabelecida comunicação com o sistema e que possa iniciar o processo de aluguer;
  - **Negociação**, isto é, receber e deliberar sobre as melhores propostas;
  - **Atualização da posição do utilizador**;
  - E, consequentemente, **notificação da sua posição atual** ao sistema.

- **Estação**, onde existem cinco funções principais, sendo estas:

  - **Registo da estação**, para permitir que as estações sejam adicionadas durante a execução do programa;
  - **Negociação**, isto é, decidir que descontos oferecer aos utilizadores;
  - **Receção da informação das restantes estações**;
  - **Receção das posições dos utilizadores**, de forma a saber que utilizadores se encontram na sua `APE` e que poderão ser alvo dos seus descontos;
  - **Notificação do seu estado**, isto é, informar o `AI` e `AE`s da sua disponibilidade.

2 Agentes

2.1 Classes

O nosso sistema é constituído, essencialmente, por três agentes:

- `AI` **(Agente de Interface)**, este agente tem o intuito de gerir o sistema, comportando-se com uma ponte entre os utilizadores e estações. Deste modo, é imperativo que seja capaz de executar algumas tarefas essenciais, como adicionar/remover utilizadores ou estações, entre outros. Este agente é, portanto, responsável pelo registo da informação no sistema e, para tal, precisa de guardar uma lista com os vários utilizadores no sistema, bem como as estações pertencentes.
  Como este agente é o que possui a responsabilidade mais ”global”, revela-se o melhor candidato para armazenar a informação sobre os agentes no sistema e, consequentemente, representá-la graficamente sob a forma de um mapa e vários diagramas;

- `AE` **(Agente de Estação)** é uma extensão da classe auxiliar `InformAE`. Para além do identificador único (`agent`) e das suas coordenadas (`localizacao`), tem de armazenar também o número de bicicletas disponíveis para aluguer (`bicicletas_disponiveis`), bem como o número de lugares reservados (isto é, número de utilizadores que aguardam começar a sua viagem - `reservas` - e utilizadores/bicicletas - `chegadas` - que se dirigem para a estação), sendo que estes dois valores não podem exceder a capacidade máxima (`capacidade`), valor esse também armazenado na classe. O seu número de estação (`num`), valor que também poderá ser utilizado como identificador, é também armazenado.
  É também necessário guardar uma lista com todos os potenciais clientes (isto é, os utilizadores dentro da sua `APE`), sendo que esta é atualizada sempre que um novo utilizador se encontra na `APE` da estação e no último $\frac{1}{4}$ do percurso da viagem. Para isso, é necessário que o `AU` envie a sua posição para o `AE`, para que esta saiba quais os utilizadores dentro da sua `APE` (esta é caracterizada pelo raio de ação de cada estação).
  Além desta informação, é, ainda, necessário guardar a lista de utilizadores que estão a aguardar o início do aluguer e os utilizadores que se estão a dirigir para a estação, assim como, a lista de estações no sistema, de forma a ter acesso ao estado de cada uma destas. De realçar que este agente tem funções para reservar/cancelar reservas de bicicletas e calcular descontos consoante o seu estado;

- `AU` **(Agente de Utilizador)**, caracterizado pelo seu estado, este é a extensão da classe `InformAU`, em que cada utilizador é identificado pelo seu identificador (`agent`), possui um estado que indica se se encontra em deslocamento ou em espera (`em_viagem`) e, ainda, três posições - as coordenadas do agente no início da viagem (`pi`), as coordenadas da estação para a qual se dirige (`pf`) e as coordenadas atuais (`pa`). De realçar que a posição atual é constantemente atualizada ao longo da viagem.
  O `AU` guarda, ainda, o preço da viagem (`preco`), a distância percorrida (`distancia_sem_alteracao`), a velocidade no seu trajeto (`velocidade`) e se já aceitou ou não algum desconto oferecido (`desconto_aceite`).
  Em termos de capacidade, este irá deliberar sobre as várias propostas que lhe são feitas (utilizando valores *random* que caracterizam a predisposição de aceitar/negociar certo tipo de propostas relativamente à ganância de cada utilizador - `ganancia` - e sua a amabilidade - `indice_amigavel` -, bem como regras previamente estipuladas), calcular a percentagem de trajeto já percorrido, atualizar o seu destino na eventualidade de aceitar uma proposta alternativa, entre outros.

Sendo assim, estas três classes, juntamente com as classes auxiliares (definidas no *package* de negócio), irão permitir um funcionamento adequado do sistema.

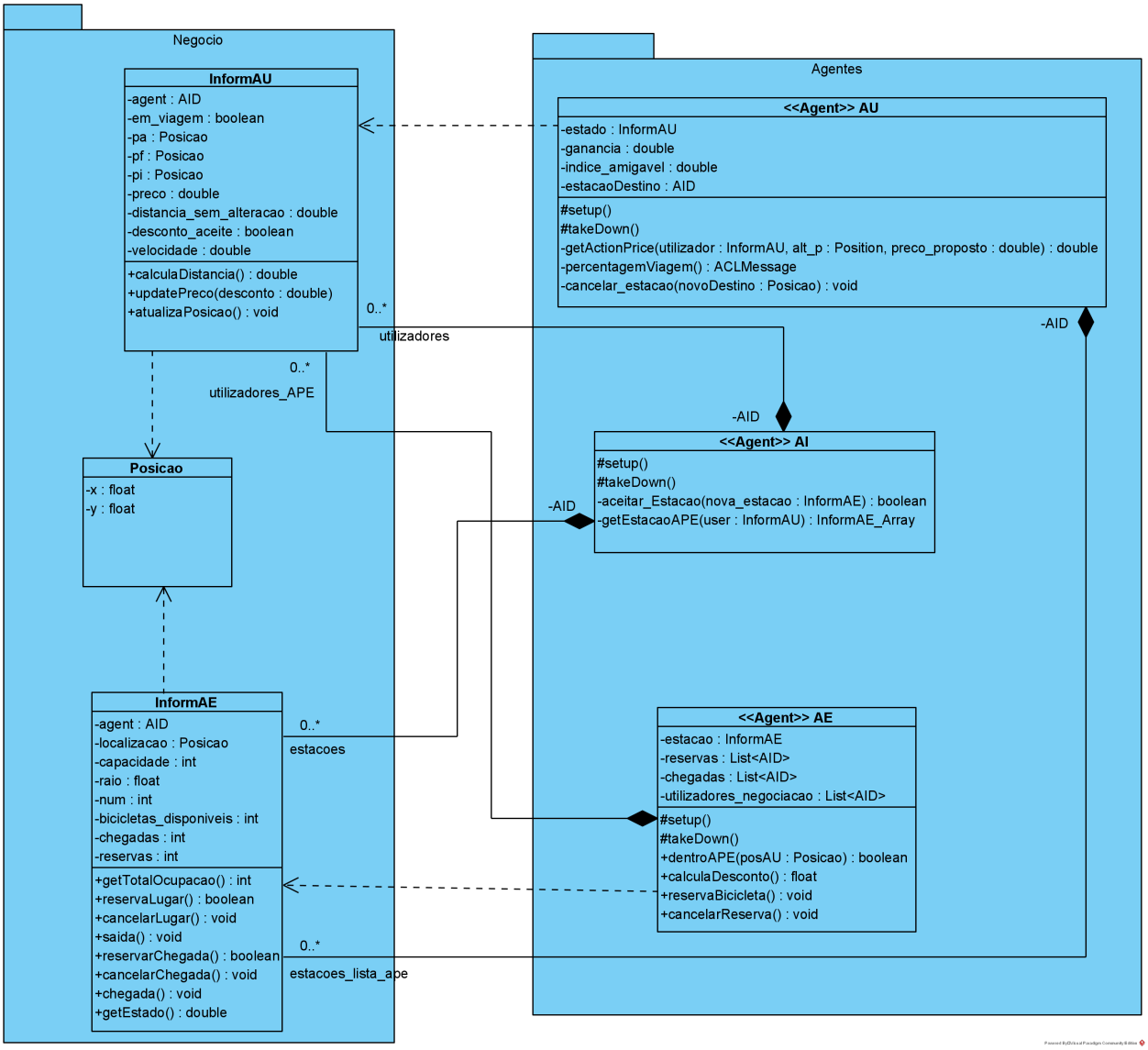

Figura 1. Diagrama de Classes do sistema

Quanto às propostas oferecidas, estas têm associado um desconto, que será um *float* de 0 a 100%, que multiplicado por `price` (variável estática guardada na classe `AE`) e pela distância da viagem, corresponde ao custo total da viagem.

Outra particularidade do nosso sistema prende-se no facto de, quando uma estação é registada, esta não estar sempre na sua capacidade máxima. Isto permite que no início do funcionamento do sistema, exista alguma flexibilidade para os utilizadores circularem entre estações, sendo inicializadas com valores entre 40% a 80% da capacidade de cada estação. O raio da `APE` de cada estação é proporcional à sua capacidade (no caso apresentado, esta toma valores entre 10 e 20), sendo que o raio toma um valor de 2 a 5 unidade de distância.

Por fim, uma particularidade relacionada com o `AU` é facto de este apenas poder mudar de destino uma vez, ou seja, não pode aceitar dois incentivos durante a viagem.

2.2 *Performatives*

Visto o objetivo do trabalho prático ser a simulação de um SPB e posterior balanceamento das bicicletas pelas várias estações, é de esperar que, para os atingir, os vários agentes comuniquem entre si, resultando no bom funcionamento do sistema.

Para representar estas interações recorremos a um Diagrama de Comunicação que reflete o tipo de mensagens que podem ser trocadas dentro do sistema nas diversas situações. As comunicações são baseadas nas normas FIPA, *performatives*, disponíveis na plataforma JADE.

Assim, as comunicações entre as várias entidades utilizam as seguintes *performatives*:

- AI:
  - ACCEPT_PROPOSAL / REJECT_PROPOSAL, para indicar se a estação foi adicionada ao sistema;
  - INFORM, para informar o AU das estações do sistema, quer no início do aluguer, quer na parte final da viagem;
  - CONFIRM, para informar o AU de que foi processado o fim do aluguer.

- AE:
  - SUBSCRIBE, para iniciar o pedido de registo no sistema;
  - INFORM / FAILURE, para indicar a disponibilidade da estação, o seu estado, entre outros;
  - PROPOSE, para enviar os descontos a oferecer aos utilizadores na sua APE, bem como para a proposta inicial;
  - CONFIRM, confirmar que recebeu a bicicleta no fim da viagem;
  - PROPAGATE, para informar os restantes AE no sistema do seu estado.

- AU:
  - REQUEST, para iniciar o processo de aluguer, escolher estação final, entre outros;
  - ACCEPT_PROPOSAL / REJECT_PROPOSAL, para indicar se o preço previsto é aceitável ou não, entre outros;
  - PROPOSE, para apresentar uma contra proposta;
  - INFORM, para informar o AI do início do aluguer e o sistema da sua posição atualizada, entre outros;
  - CONFIRM, para confirmar que o aluguer deve começar e o fim da viagem;
  - CANCEL, para informar uma estação para cancelar a reserva ou chegada.

Quando for recebida uma mensagem que um dos agentes não sabe processar, será enviada uma mensagem com a *performative* UNKNOWN.

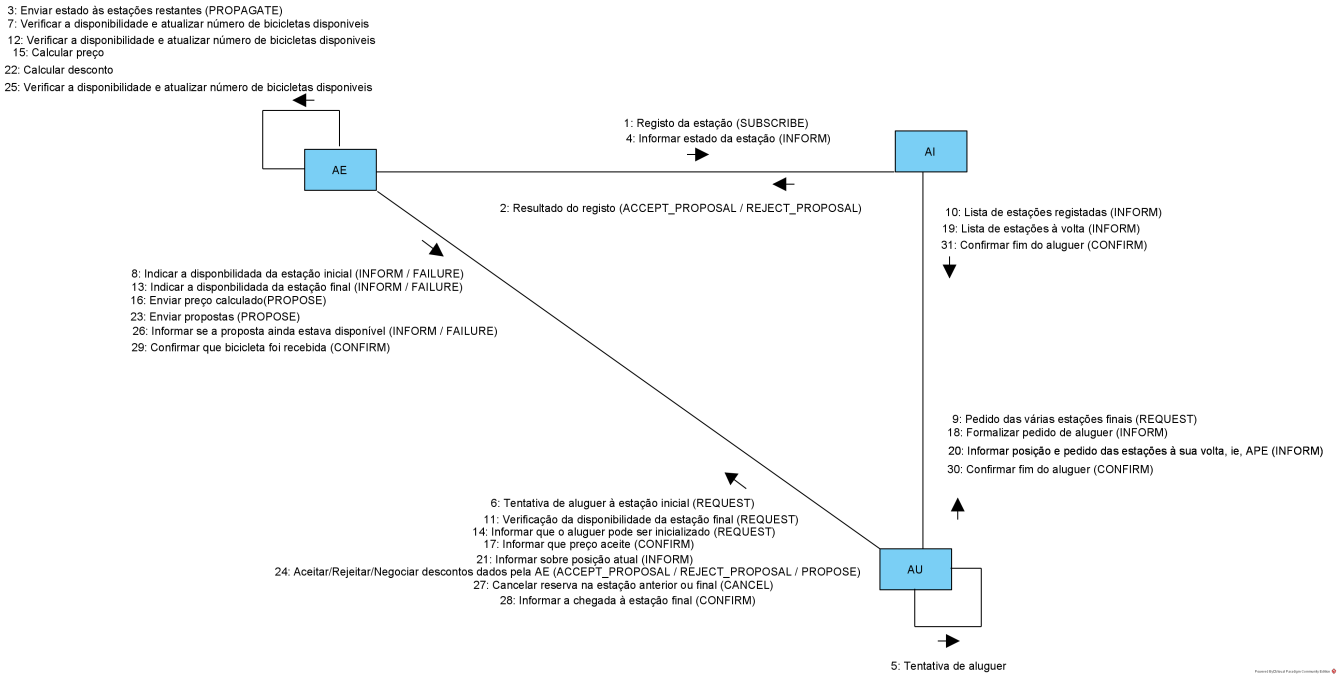

Figura 2. Diagrama de Comunicação do sistema

2.3 *Behaviours*

Na arquitetura deste sistema, são várias as tarefas que terão de ser executadas repetidamente como, por exemplo, o processamento das comunicações do `AI`. Todavia, também existem procedimentos que apenas são executados uma vez como, por exemplo, uma tentativa de registo de estação. Desta forma, será necessário definir alguns *behaviours* para cada agente.

Portanto, para cada agente, terão de ser implementados os seguintes *behaviours*:

- `AI`:
  - `CyclicBehaviour`, para processar as várias mensagens vindas quer de `AU`s quer de `AE`s. Este deve processar continuadamente para responder a todos os pedidos.
- `AE`:
  - `CyclicBehaviour` (a emular um `OneShotBehaviour`[1], pois trata-se de uma ação que apenas deve ser executada uma vez), para o registo da estação;
  - `CyclicBehaviour`, para poder responder aos pedidos de aluguer e atualizar a sua ocupação, bem como manter descontos atualizados consoante a sua ocupação e negociar com os `AU`s;
  - `TickerBehaviour`, para informar o sistema do seu estado.
- `AU`:
  - `CyclicBehaviour` (a emular um `OneShotBehaviour`[1]), para tentar iniciar um pedido de aluguer (visto também se tratar de uma ação que apenas deve ser executada uma vez, caso seja rejeitado, o agente sai do sistema);

[1] Foi implementado desta forma para não haver a necessidade de utilizar ciclos *while* cujo único intuito seria esperar por uma mensagem.

- `CyclicBehaviour`, para comunicar com o sistema (negociar propostas, confirmar/cancelar reservas, entre outros);
- `TickerBehaviour`, para atualizar a sua posição durante a viagem e, consequentemente, informar o sistema.

O seguinte diagrama demonstra o processo interno para os vários agentes pertencentes ao sistema e, como pode ser constatado, grande parte dos estágios dos agentes tratam-se de ações recursivas, isto é, procedimentos repetitivos.

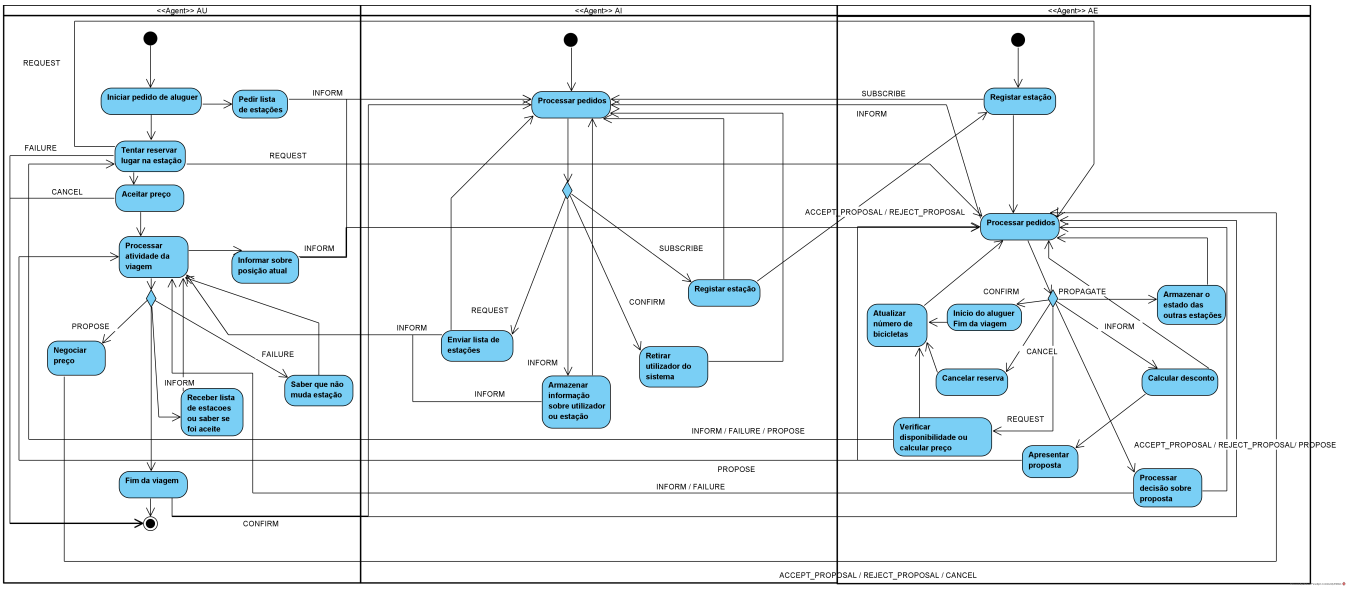

Figura 3. Diagrama de Atividade do sistema

3 Interação/Comunicação entre agentes

3.1 Pedido de aluguer e interação durante a viagem

Para o processo de aluguer, o Agente Utilizador (AU) inicializa a *App* ao lado da estação, de modo a poder utilizar o identificador associado à ela (por exemplo, um *QRCode*), estabelecendo, assim, ligação com o Agente Estação (AE) para iniciar o processo de aluguer, provando que o AU se encontra lá (1).

De seguida, caso não haja bicicletas, o AE informa o AU (2) e este sai do sistema (3), caso contrário o AE reserva uma bicicleta, decrementando o seu número total (4) e informa o utilizador (5).

Depois da interação com o AE inicial, o AU entra em comunicação com o AI para informar do início de aluguer (6). O AI, então, calcula a lista de estações de destino possíveis (7) e envia-a ao AU (8).

Neste momento, o AU escolhe a estação de destino pretendida e comunica a decisão à AE escolhida (9), sendo que esta informa se o aluguer pode ocorrer.
Caso a resposta seja negativa (10), informa a estação inicial para cancelar a reserva da bicicleta (11), o que vai fazer com que esta aumente o número de bicicletas disponíveis (12) e, por consequência, provoca a sua saída do sistema (13).
Caso existam lugares livres na estação de destino, esta reserva um lugar para a bicicleta (14) e informa o utilizador (15).

De seguida, o AU envia uma mensagem a informar o AE final de que o aluguer se irá inicializar (15). Posto isto, a estação final calcula e envia um preço (17/18), sendo que o AU pode decidir entre estas duas opções:

- Recusar o preço apresentado, informando o AE (20), aumentando o seu número de bicicletas disponíveis (21) e, consequentemente, informa o AE inicial para este cancelar a reserva da bicicleta anteriormente realizada (22) e aumentar o número de bicicletas disponíveis (23). Por fim, o AU sai do sistema (24);
- Aceitar o preço (19) e informar o AE inicial de que o aluguer se irá iniciar (25) - que, por sua vez, atualizará a disponibilidade - e o AI de que entrou no sistema (26) e tem um aluguer em curso, adicionando-o, assim, ao *Map* de utilizadores (27).

Durante a viagem, o AU atualiza sempre a sua posição (28), informando o AI desta (29), assim como a percentagem percorrida da viagem.
Quando esta percentagem ultrapassar os 75%, o AU vai receber uma lista de AEs (30) (sendo que este tem de se encontrar dentro das APEs respetivas) e vai passar a enviar-lhes mensagens, regularmente, para se dar a conhecer, iniciando, assim, um processo de negociação entre eles (31/32), que será apresentado noutro diagrama.

Quando finalmente chegar ao destino, informa o respetivo AE de destino (33), levando-o a aumentar o seu número de bicicletas disponíveis (34), sendo-lhe enviada a confirmação da receção (35).

Por fim, informa também o AI, enviando o respetivo pagamento (36). Este, por sua vez, remove o AU do *Map* de utilizadores em viagem (37), informando-o sobre o fim do aluguer (38). O AU sai do sistema (39).

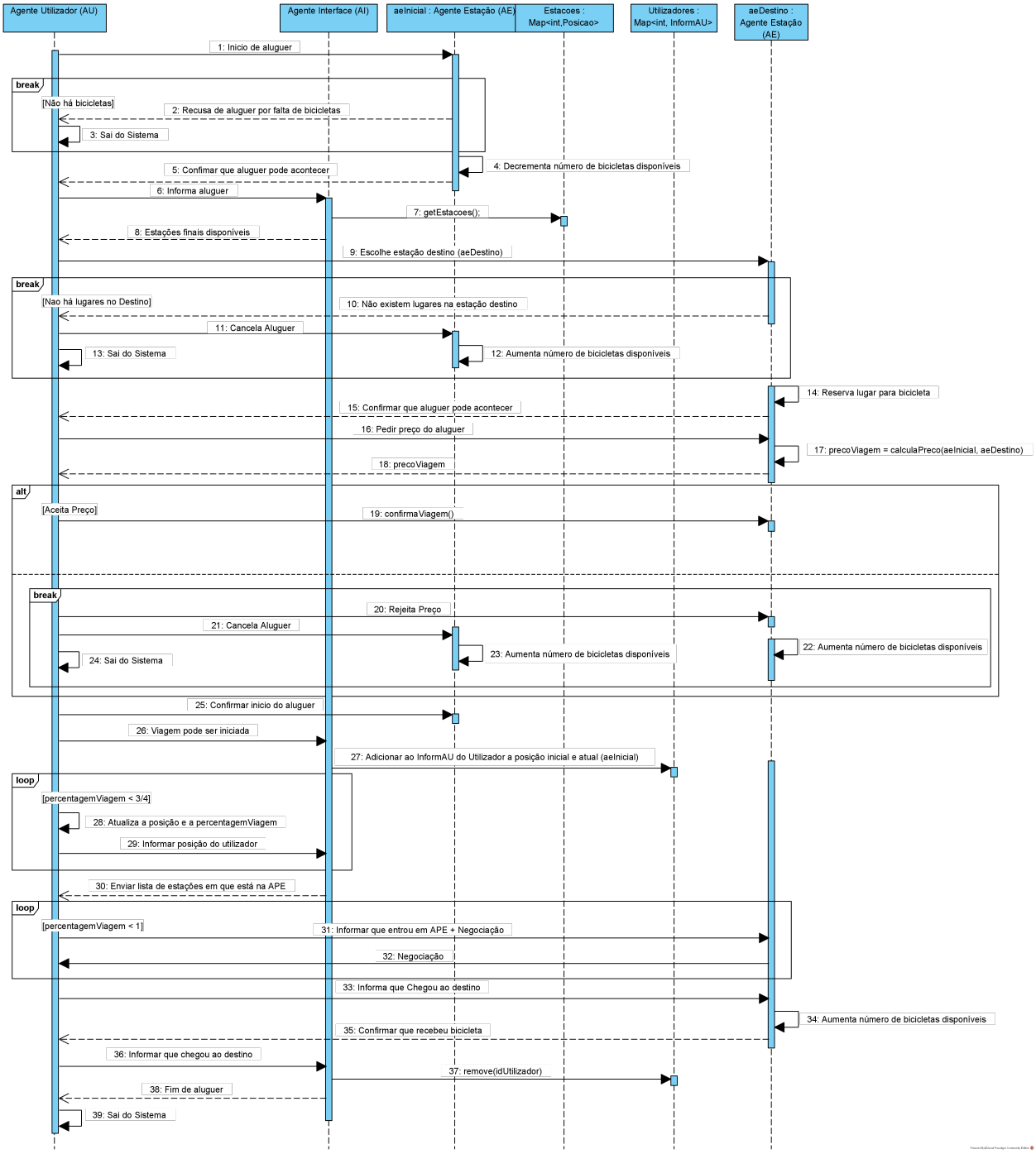

Figura 4. Diagrama de Sequência de um aluguer

3.2 Registo de estação

O processo de registo de uma estação é um processo bem mais simples. Aqui o `AE` apenas procura pelo `AI` nas "páginas amarelas" (`DFService`) e envia um pedido de registo com a informação necessária - capacidade máxima, número de bicicletas atual, entre outros.
De seguida, se o pedido for válido, o `AI` adiciona esta estação à sua lista, caso contrário, rejeita-o. Por fim, informa a estação do resultado do pedido.

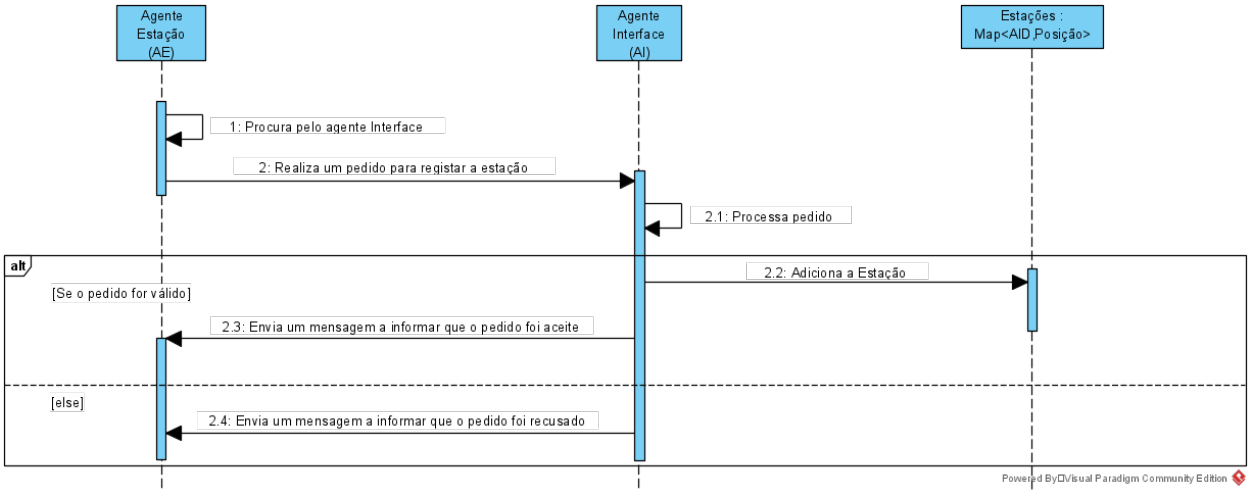

Figura 5. Diagrama de Sequência de um registo de uma estação

3.3 Negociação

Quanto ao processo de negociação, este tem um papel muito relevante na *performance* do sistema. Existe uma certa '*Coopetição*' (estratégia de negócios baseada na Teoria dos Jogos), isto é, os `AE`s terão de competir para garantir o aluguer, quando têm poucas bicicletas disponíveis, apresentando propostas e descontos muito vantajosos que aliciem os clientes. Assim, cada estação tem como objetivo evitar a escassez de recursos (neste caso, bicicletas) de forma a poder continuar a operar, sem provocar a sua sobrelotação. Por outro lado, quando estiver congestionada, esta deverá oferecer propostas menos vantajosas (ou não oferecer propostas de todo) com vista a manter o equilíbrio do número de bicicletas existentes nos sistemas.
Consequentemente, esta competição resulta, de certa forma, em cooperação, pois as estações trabalham para um bem comum - o balanceamento e descongestionamento do sistema. Ainda assim, os `AE` podem ser vistos como agentes competitivos, uma vez que procuram maximizar a sua utilidade (visto terem apenas acesso ao seu estado e não ao estado de outras `AE`s ou até do sistema), tendo, contudo, alguns comportamentos cooperativos.

Outra situação onde se aplica a dualidade 'competição *vs.* cooperação' é na apresentação de descontos ao `AU`, já que um `AU` e `AE` tentarão encontrar propostas que beneficiem ambos. Assim, um `AU` poderá optar por manter o trajeto, com vista a chegar à estação inicialmente pretendida. No entanto, também poderá decidir cooperar com a estação que lhe propõe o desconto, aceitando um preço menor deslocando-se para um novo destino, equilibrando o congestionamento do sistema.

Esta negociação está representada no diagrama seguinte.

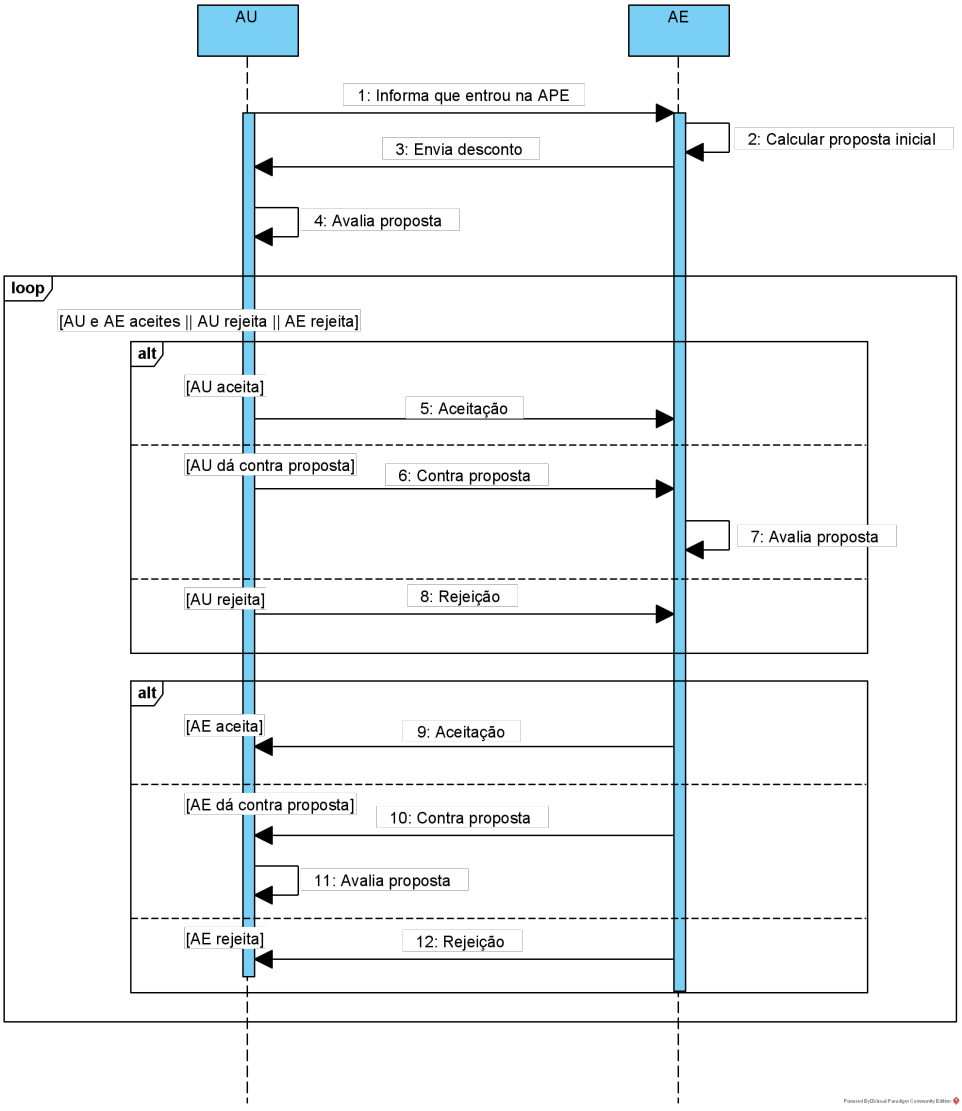

Figura 6. Diagrama de Sequência da negociação entre estação e utilizador

O início de uma negociação é efetuado quando o AU entra na APE de um determinado AE. Aí, o AU informa o respetivo AE que entrou na sua APE (1), o que provoca um cálculo de proposta inicial por parte deste (2), sendo esta enviada ao AU (3) que vai, prontamente, avaliá-la (4).

A partir daqui, inicia-se o processo de negociação propriamente dito, onde o AU tanto pode aceitar (5), rejeitar (8) ou dar uma contra proposta (6), o que vai levar a uma avaliação por parte do AE (7) que, por sua vez, pode aceitar (9), rejeitar (12) ou dar uma contra proposta (10), desencadeando uma nova avaliação por parte do AU, voltando ao início deste processo de negociação. Este é terminado assim que os dois entrem em acordo, ou seja, ambos aceitem a proposta ou, então, logo que uma das partes rejeite uma proposta.

4 Fórmulas e Implementação

Nesta secção iremos apresentar o raciocínio por detrás das várias fórmulas utilizadas para tomar decisões relativamente às propostas, bem como a equação de deslocamento utilizada pelos utilizadores.
Do mesmo modo, também, serão apresentadas algumas particularidades relacionadas com a implementação, problemas e respetivas decisões.

4.1 Negociação de descontos

Na arquitetura aqui em exposição, a negociação de propostas será um ponto fulcral para o bom funcionamento do sistema, pois será o mecanismo que irá permitir que as várias estações equilibrem o seu número de bicicletas.

No lado do utilizador, temos de ter em conta alguns fatores como, cada utilizador ser individual, isto é, diferentes utilizadores na mesma situação não devem reagir da mesma forma. Para isso é necessário utilizar valores *random* que permitam descrever/caracterizar cada utilizador, pelo que escolhemos ter dois valores - um para a ganância (quanto maior, mais elevados serão os descontos procurados) e outro para a amabilidade (quanto maior, mais elevados serão os desvios aceites pelo utilizador).
Além destes fatores, é tida em conta a distância restante para a estação pretendida, isto é, quanto maior a distância, menor a probabilidade de aceitar o desconto. Também é considerada a distância entre a posição atual e a posição da estação que apresentou a proposta (quanto maior este valor, menores as chances de a proposta ser aceite) e a distância entre a estação inicialmente escolhida e a nova estação, que representa a distância a que o utilizador ficará da posição pretendida no fim do aluguer (quanto maior o valor, menores as probabilidades de a proposta ser aceite).
No que toca a valores monetários, quanto menor o preço, maior será a probabilidade de aceitar o incentivo.

Resumidamente, a negociação pode ser explicada através de duas equações, sendo g e a, as variáveis de ganância e amabilidade, respetivamente; d, nd e δ, a distância restante, a nova distância restante e a diferença entre o novo destino e destino anterior, respetivamente; p, pp, L, v e M, o preço inicial, o preço proposto, o desconto máximo, a velocidade do utilizador e a velocidade máxima, respetivamente[2]:

$$1 - g^2 \geq \frac{np}{p} - L$$

$$(a+1) \cdot d \cdot \frac{v}{M} \geq \frac{\delta \cdot nd}{d^2}$$

Após obter o resultado destas equações, o utilizador aceita o preço, se ambas forem verdadeiras. Caso sejam algo abonatórias é feito uma contra proposta pela seguinte fórmula:

$$cp = pp * (0.98 - g^2 * 0.13)$$

Desta forma, um utilizador muito ganancioso, $g = 1$, irá dar um preço que corresponde a 85% do preço proposto, pp, no espetro oposto, enquanto que um utilizador nada ganancioso, irá propor um preço que corresponde a 98% do preço proposto.

Quanto às estações, a sua negociação é bem mais simples, pois apenas têm em conta a sua ocupação, contudo utilizam mais fatores externos.

[2] equivalente à função `getActionPrice` do `AU`.

Para isso, foram criados dois níveis - um de ocupação baixa ($to \leq 35\%$) e outro de ocupação média ($35\% < to \leq 60\%$), sendo que cada um destes tem a sua equação[3]:

$$pp = to^2 \cdot 0.2 \cdot \frac{600}{49} + 0.6$$

$$pp = (to \cdot 0.2 - 0.07) \cdot \frac{8}{5} + 0.9$$

Estas equações permitem que, no estado de baixa ocupação, sejam dados descontos de 40% a 10%, enquanto que no estado de média ocupação a amplitude apenas é de 10% a 2%.

Então, para a primeira equação, o valor máximo de to é 0.35 e, portanto, $to^2 = 0.1225$, que implica que $pp = 0.9$, ou seja, 10% de desconto. Por outro lado, o valor mínimo é 0, o que implica que $pp = 0.6$, ou seja, 40% de desconto.

Relativamente à segunda equação, obtemos um valor na gama de 0.9% ($to = 0.35$) a 0.98% ($to = 0.6$), sendo o valor de desconto, então, de 10% a 2%, tal como referido acima.

A maior diferença entre as duas equações depreende-se no facto de ser utilizado g^2, ou seja, ter um decrescimento quadrático, enquanto que, no outro modo, o decrescimento é apenas linear. Esta diferença deve-se ao facto de uma estação com uma ocupação baixa ter urgência em aliciar novos clientes.

4.2 Decisão sobre propostas e alugueres

Tal como referido na secção anterior, 4.1, a negociação é um dos pontos de maior importância no sistema. Este processo, contudo, engloba não só o cálculo de propostas, como também deve ter "inteligência" para deliberar sobre as propostas apresentadas, "raciocínio" este que será apresentado nesta secção.

Em relação a propostas iniciais, é oferecido o preço calculado contudo, caso a sua ocupação seja superior a 60%, é oferecido um preço sem desconto, a 100%.
Em relação a propostas iniciais de aliciamento por parte do `AE`, caso o preço calculado seja menor que o preço acordado, este não apresentará a sua proposta, contudo, se a sua ocupação for baixa, oferece um desconto de 10% sobre o preço acordado.

Em relação a contra propostas do `AU`, caso seja calculado um preço maior do que o preço oferecido, é escolhido o preço oferecido, ou seja, é aceite a contra proposta, pois é a proposta mais abonatória para o utilizador. Caso o preço oferecido esteja perto do preço calculado, é feita uma contra proposta em que é efetuada uma média pesada entre estes dois preços, 40% para o preço proposto e 60% para o preço calculado. Caso contrário, é rejeitada a proposta.
Outro fator que é tido em conta para as propostas oferecidas, é o estado das estações que partilham território da `APE` da estação final acordada e a estação inicialmente escolhida. Caso uma destas possua uma ocupação baixa, as estações em redor não deverão apresentar propostas ao utilizador.
Outra situação possível é a estação para onde o utilizador se dirige ter uma ocupação muito alta e, desta forma, devem ser apresentadas propostas de desconto de forma a aliciar utilizadores e reduzir o excesso de bicicletas na estação original.

Em relação ao `AU`, quando apresentado com uma proposta inicial, este utiliza a seguinte equação para decidir se esta é aceite ou não, sendo pp, M, g e p_g o preço proposto, preço

[3] equivalente à função `getActionPrice` do `AE`.

máximo, ganância e uma probabilidade (para tornar o processo mais aleatório), respetivamente:

$$0.99 - \frac{pp - 0.6 \cdot M}{0.4 \cdot M} \cdot 0.09 \geq 0.7 + 0.3 \cdot g \cdot p_g$$

Desta forma, caso o preço proposto corresponda ao preço máximo, então apenas será rejeitado no caso do resultado da multiplicação da ganância pelo fator aleatório ser superior a 0.6666 (tendo em conta que os dois fatores variam entre [0, 1], acontecerá, aproximadamente, $1 - \sqrt{0.6666} \approx 18.35\%$ das vezes).
No caso oposto, se o preço proposto corresponder ao preço mínimo, então apenas será rejeitado caso o resultado da ganância a multiplicar pelo fator aleatório seja superior a 0.9666 (tendo em conta que os estes dois fatores variam entre [0, 1], acontecerá, aproximadamente, $1 - \sqrt{0.9666} \approx 1.68\%$ das vezes).

Quando se trata de uma negociação pura, isto é, deliberar sobre contra propostas (se foi apresentada uma proposta inicial e foi feita uma contra proposta), são utilizadas as equações referidas na secção 4.1.

4.3 Deslocamento

Tal como a negociação, o deslocamento é essencial para o funcionamento do sistema, pois, sem este, um aluguer nunca será concluído.

Assim sendo, este foi feito em linha reta, para simplificar o processo, representado pelo seguinte excerto de código:

```
public double move(Position pf, double dist) {
    double d = this.distancia(pf);

    if (d <= dist) {
        this.x = pf.x;
        this.y = pf.y;

        return d;
    }
    else {
        double xtemp = (pf.getX() - this.x) / d;
        double ytemp = (pf.getY() - this.y) / d;

        this.x += xtemp * dist;
        this.y += ytemp * dist;

        return dist;
    }
}
```

Assim, caso o ”salto” a dar seja maior do que a distância restante, o utilizador, simplesmente, passa para as coordenadas da posição final, parâmetro `pf`.

4.4 Inicializar uma estação

Quando uma estação é inicializada, é-lhe passado um inteiro que representa o seu identificador, equivalente às estações já inicializadas no sistema (não é necessariamente equivalente

às estações aceites pelo AI) e são inicializadas as várias variáveis do AE como o raio, a capacidade e a sua posição, com valores *random* (normalmente estes têm uma gama de valores possíveis).

De seguida, envia um pedido ao AI para pertencer ao sistema, todavia, se existir uma estação a menos de um dada distância pré-definida, esta é rejeitada e sai do sistema.

4.5 Inicializar utilizador e escolher estação

Quando o utilizador é inicializado, é-lhe passado um inteiro que representa o número de estações que tentaram subscrever-se no sistema, podendo, assim, contactar diretamente a estação inicial escolhida.
O único problema associado a esta implementação consiste no caso do número escolhido de forma aleatória corresponder a uma estação que foi rejeitada, pois obriga a que o aluguer nunca aconteça. No início do aluguer são também atribuídos os valores às várias variáveis do AU, como velocidade, ganância, amabilidade, entre outros.
Contudo, este problema já não ocorre quando está a ser escolhida a estação final, visto que, nesta etapa, o utilizador tem acesso a uma lista fornecida pelo AI e, desta forma, sabe que estações pertencem ao sistema, nunca escolhendo uma estação que saiu dele.

4.6 Cancelamento de reservas

No que diz respeito ao cancelamento de reservas/chegadas, é necessário ter em consideração algumas situações que podem ocorrer, sendo estas: o AI não enviar a lista de estações; o aluguer ser cancelado durante a tentativa de aluguer (por exemplo, quando a estação final escolhida rejeita o aluguer por sobrelotação ou quando o utilizador rejeita o preço proposto no início do aluguer); ser aceite uma nova proposta, o que traz a necessidade de avisar a estação final anterior de que a bicicleta já não será entregue lá (este caso acontece já durante o decorrer do aluguer).

4.7 Fim de aluguer

Inicialmente, o utilizador enviava mensagem a confirmar que acabou o aluguer e esperava pela confirmação da receção da mensagem por parte do AE e do AI e só depois, saía do sistema, contudo essa abordagem criou alguns problemas no processamento de mensagens por parte do AE e AI (desaparecimento de mensagens e não atualização do estado - aluguer terminado - do utilizador no mapa, assim como ”duplicação” de alugueres), pelo que, foi necessário que o AU saísse logo após acabar o aluguer.

4.8 Aceitar propostas

Outro problema que surgiu posteriormente, foi o aparecimento de bicicletas ”repetidas”, isto é, no fim da execução, o sistema tinha bicicletas que ainda não tinham sido entregues, contudo, a soma das bicicletas de todas as estações correspondia ao número de bicicletas inicial. Este problema devia-se a uma falha de processamento por parte do AU, visto haver uma situação que permitia que este aceitasse mais que uma proposta, caso fizesse duas contra propostas a duas estações diferentes e ambas fossem aceites.

De modo a mitigar este erro, utilizámos a primitiva *synchronized* de forma a garantir que não havia um acesso concorrente a alguma variável do `AU`, bem como colocar mais um caso de processamento no `AU` para as situações em que eram aceites mais que uma proposta, tendo decidido que, nestes casos, seria enviada uma mensagem de cancelamento de reserva à estação que aceitou a segunda proposta, mantendo como destino a primeira estação com a qual foi acordado um incentivo.

4.9 *Pipelining* e implementação

Durante o processamento de mensagens nos `CyclicBehaviours` dos vários agentes, foi necessário utilizar várias formas de diferenciar as diferentes mensagens, por exemplo, visto que o `AE` recebe três tipos de mensagens `REQUEST` - pedido de aluguer à estação inicial ou à estação final ou a informar que já tem acordo com as duas estações - será utilizado o estado das posições do utilizador para ser o fator de diferenciação.
Outro caso, é com as mensagens `CONFIRM` no `AE`, pois, visto que pode receber dois tipos de mensagem - indicar que um utilizador aceitou um preço de aluguer e que este deve ser iniciado ou informar que entregou uma bicicleta na estação final - será verificado se o expedidor da mensagem pertence à lista de reservas ou à de chegadas da estação.

Resumidamente, em todos os `CyclicBehaviours` que processem mensagens, por exemplo `Receiver` do `AE`, `Receiver` do `AI` e `Reply` do `AU`, são necessários mecanismos de diferenciação, isto é, cruzar a *performative* com informação armazenada em `Lists` e `Maps`, pois, apenas, são utilizadas, apenas 10 *performatives* e é necessário diferenciar cerca de 30 tipos de mensagens.

4.10 Problemas e decisões durante a implementação

Por fim, começámos a implementação que foi, sem dúvida, a fase que trouxe mais problemas, alguns derivados a detalhes não tratados aquando da modelação do sistema, outros a erros onde não foram tidos em conta os *standards* de desenvolvimento de código em `JADE`, principalmente os ciclos *while* (estes funcionavam bem, contudo tinham um impacto muito grande na *performance* do sistema).

Outro detalhe que tentámos utilizar foi o facto de apenas serem inicializados certos *behaviours* a partir de um dado ponto e não desde o início, utilizando o método `addBehaviour` dentro de um *behaviour* "inicializador", por exemplo, `myAgent.addBehaviour(new Reply());`.

Outro problema com que nos deparámos foi a pouca predisposição a aceitar alugueres por parte dos utilizadores, o que nos obrigou a fazer algumas alterações nas tomadas de decisão destes agentes. Aliado a este problema, mais do que resolver os casos de perda de alugueres devido a falta de bicicletas numa estação, tornou-se premente resolver o excesso de bicicletas na estação, que era 20 vezes mais frequente. Nessa situação, eram perdidos, aproximadamente, 22.82% dos alugueres tentados, sendo que apenas 1.16% desses deviam-se a falta de bicicletas e os restantes 21.66% a perdas por sobrelotação. Devido a este elevado número de perdas, foi necessário tornar os clientes mais permissivos e recetivos a aceitar incentivos, bem como encontrar uma combinação de capacidade da estação e disponibilidade, para poder obter resultados mais aceitáveis, e raio da estação.

Além das alterações referidas acima, foi também necessário mudar a abordagem da negociação. Inicialmente, tomámos uma abordagem de '*Coopetição*', tal como já foi referido,

contudo os resultados obtidos não eram os melhores e, portanto, optámos por tornar as estações mais conscientes dos estados das estações em redor (como a ocupação) e escolher uma abordagem mais focada na cooperação. Ainda assim, em situações "normais", as decisões das estações são governadas por abordagens competitivas, por exemplo, quando as estações à volta não estão numa situação crítica (quer por excesso, quer por falta de bicicletas). Posto isto, o resultado obtido ainda não foi o melhor, pois a informação que as estações tinham sobre as contíguas era obtida em "segunda mão" e poderia não ser a informação mais atual. Desta forma, ao invés de ser o `AI` a difundir/propagar esta informação, passou a ser o trabalho de cada estação enviar o seu estado a todas as estações do sistema - responsabilidade do `TickerBehaviour` do `AE`.

Por fim, os problemas mais difíceis de resolver, não tanto pela sua complexidade ou dificuldade, mas sim pela limitação do sistema atual, foram as situações em que uma estação ficava demasiado afastada das restantes e, desta forma, era impossível aliciar utilizadores que tinham outra estação como destino ou quando uma estação era escolhida muitas vezes seguidas como estação inicial (que pode acontecer devido a ser uma escolha aleatória), reduzindo muito o seu número de bicicletas, levando à aceitação de todos os alugueres possíveis (todavia, caso nenhum utilizador atravesse a `APE` durante a sua viagem, esta nunca tem a possibilidade de incrementar o seu *stock* de bicicletas). Uma possível solução seria aumentar o raio das estações que se encontram em situações de ocupação baixa, tentando, assim, aumentar a probabilidade de encontrar um utilizador e, consequentemente, negociar com ele.

4.11 Trabalho futuro

Como trabalho futuro, assinala-se a criação de algumas condições na inicialização de agentes, por exemplo, utilizar uma distribuição normal para a escolha das estações e posições e não uma distribuição uniforme ou aleatória, para permitir uma representação mais fiel a uma situação real, pois há zonas que têm mais procura do que outras. Além da utilização de distribuições, ao invés de escolhas aleatórias, também seria interessante utilizar o "tempo" do sistema como variável, pois há horas de ponta, em que certas áreas seriam mais procuradas consoante a altura do dia.

Poderiam ser efetuadas certas otimizações no processo de troca de mensagens entre agentes, principalmente, entre estações, pois não é necessário para uma estação conhecer todas as estações do sistema, apenas a sua vizinhança (e a estação final de todos os alugueres que atravessam a sua `APE` na fase final). Isto permitiria reduzir eventuais custos com a troca de mensagens entre estações que se encontram a grande distância e, adicionalmente, reduzir a carga sobre a rede. Ainda assim, esta particularidade é tida em conta na troca de mensagens entre utilizadores e estações, sendo que estes apenas comunicam caso se encontrem na `APE` da estação.

Outra alteração, talvez mais complexa, seria mudar o trajeto do utilizador durante a viagem, para permitir que este, possivelmente, atravessasse mais `APE`s, ou seja, um trajeto não retilíneo. Esta alteração iria obrigar a calcular um valor aleatório, ω, que representa o desvio do trajeto retilíneo, que teria de ter valores máximos (seriam utilizadas as funções *sen* e *cos*), pois, caso contrário, o utilizador poderia nunca chegar à estação final e, consequentemente, não terminar o aluguer.

5 Interface Gráfica

5.1 Mapa

O mapa é uma representação do funcionamento do sistema desde o início até ao fim do aluguer. Neste, é possível visualizar as ações desempenhadas por dois agentes - o utilizador e as estações.
Cada estação é representada por um círculo colorido, sendo assim possível diferenciar as estações e as suas áreas de proximidade.
Os utilizadores são representados por bicicletas, tendo estas três cores possíveis (azul, vermelho e verde) para caraterizar os três cenários distintos - como já foi dito anteriormente, quando um utilizador realiza 75% da viagem, poderá receber propostas das estações que se encontram na sua área de proximidade, sendo que a bicicleta tem a cor azul nos primeiros 75% da viagem e, a partir deste ponto, passa à cor vermelha; caso o utilizador aceite uma proposta de uma estação, esta muda para a cor verde, caso contrário, mantém-se vermelha.

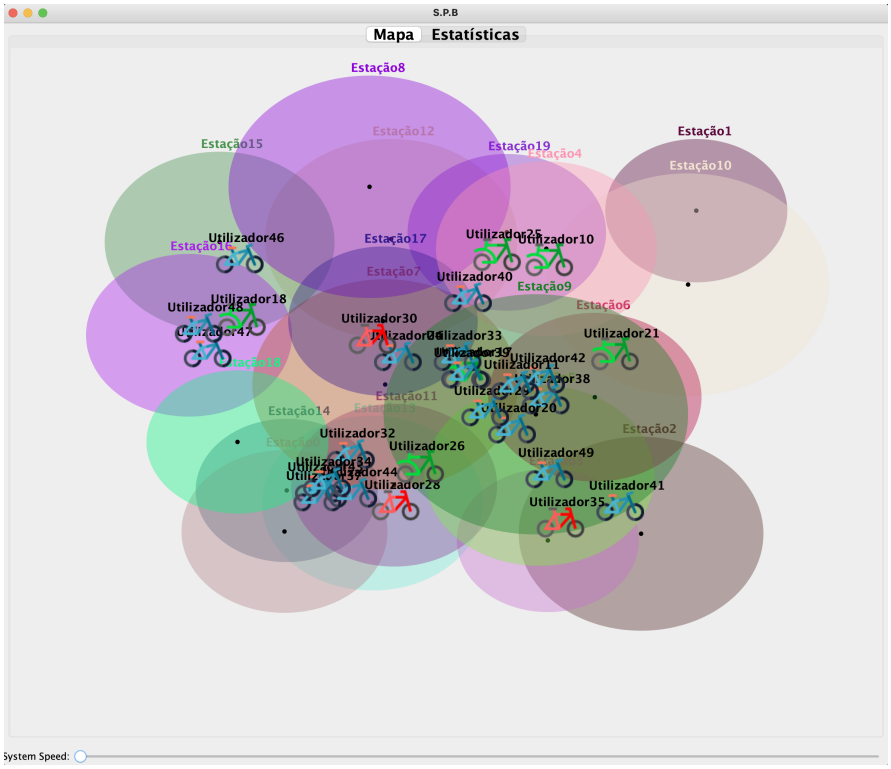

Figura 7. Interface do Mapa

O agente interface, `AI`, é o responsável por desenhar o mapa, isto é, implementa um `TickerBehaviour` que, de 50 em 50 milissegundos, redesenha o mapa apresentado na figura 7, permitindo, assim, observar o estado atual do sistema e as posições atuais de cada utilizador.
Outro componente do mapa é a barra situada na parte inferior da janela, que serve para controlar o tempo do sistema, isto é, quanto mais alto for este valor, mais rápido será o deslocamento do utilizador e, consequentemente, mais curto será o aluguer.
De ressalvar, que os valores da posição e raio foram normalizados, para que, ao alterar o tamanho da janela, o mapa continue visível.

5.2 Estatísticas

Para analisar a performance do sistema são gerados quatro diagramas, através da biblioteca *jfreechart*, que permite desenhá-los na janela.
No primeiro gráfico, é apresentado o número de bicicletas disponíveis de cada estação, sendo assim possível acompanhar o estado de cada estação e ver quais estão em rutura, as mais requisitadas ou em sobrelotação.
No segundo diagrama, temos uma previsão das chegadas a cada estação, que não só permite saber se uma estação irá ter mais bicicletas disponíveis ao longo do tempo, como também permite verificar se deviam ser oferecidos incentivos melhores.
No terceiro gráfico, do tipo circular, observa-se a ocupação de cada estação, ou seja, o número de bicicletas disponíveis, sem contar com as bicicletas reservadas para alugueres, a dividir pela capacidade da estação.
No quarto e último diagrama, é possível analisar o número de alugueres rejeitados devido a falta de bicicletas na estação inicial (isto acontece quando um utilizador chega a uma estação e quer requisitar uma bicicleta, porém não existem bicicletas disponíveis) e o número de alugueres rejeitados devido a excesso de bicicletas na estação final (isto acontece quando um utilizador escolhe uma estação como destino e quer reservar um lugar para entregar bicicleta alugada, porém não existem lugares suficientes para aceitar a bicicleta do aluguer (soma do número de reservas com bicicletas disponíveis iguais à capacidade máxima)). Neste gráfico, é, ainda, possível verificar outro fator (não menos relevante) - o número de alugueres concluídos e o número de incentivos aceites, representado pela barra das mudanças do destino, uma vez que, quando o utilizador aceita uma promoção, o destino final é, necessariamente, diferente do destino inicialmente escolhido.

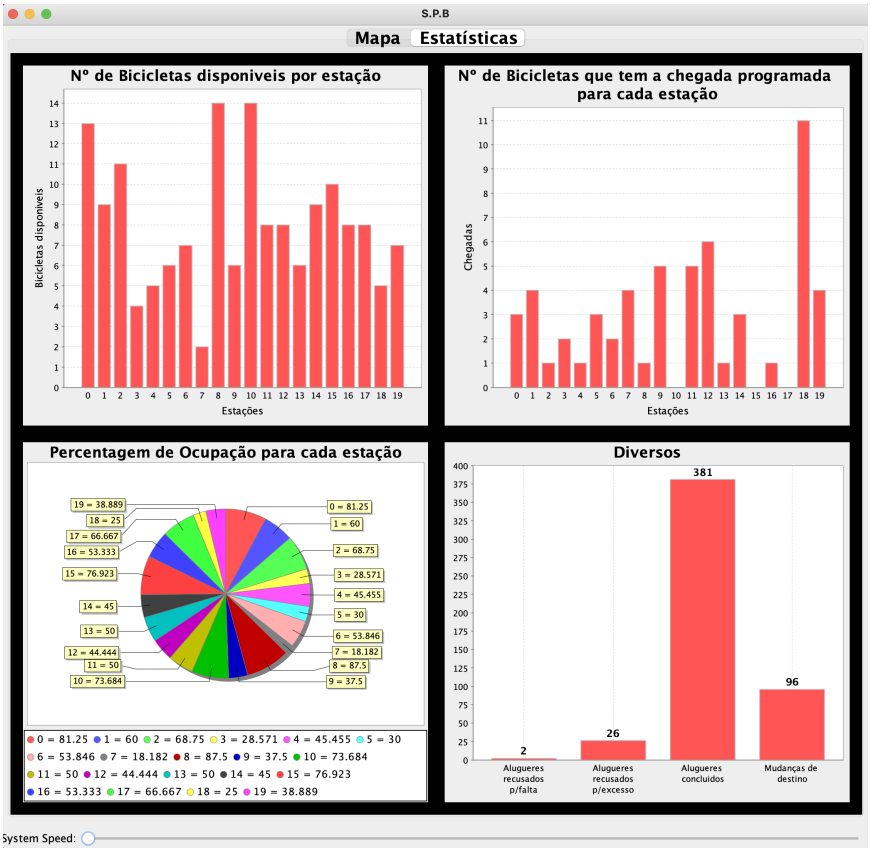

Figura 8. Painel estatístico do sistema

6 Análise de Resultados

Relativamente aos resultados obtidos pelo sistema, é de notar que, normalmente, ocorrem muitas recusas de alugueres por parte da estação destino, devido ao excesso do número total de bicicletas (isto é, quando existe um grande volume de bicicletas que estão em viagem para a estação referida), pois, mesmo que ainda não tenham chegado à estação, o seu lugar já se encontra reservado.
Por exemplo, se a capacidade de uma estação é 10 (tem 4 bicicletas disponíveis e 6 bicicletas a chegar), esta não poderá aceitar mais alugueres, recusando-os por sobrelotação. Em casos extremos e quando esta é a estação inicial, se o número de bicicletas a chegar for igual à capacidade, a estação terá de recusar os pedidos por falta de bicicletas. Ambos os casos são extremamente difíceis de lidar, pois não só estão dependentes do algoritmo de criação de utilizadores (que mesmo sendo *random* pode criar vários pedidos provenientes da mesma estação ou tendo como destino uma só estação), como também da velocidade dos utilizadores, que pode variar de utilizador para utilizador.

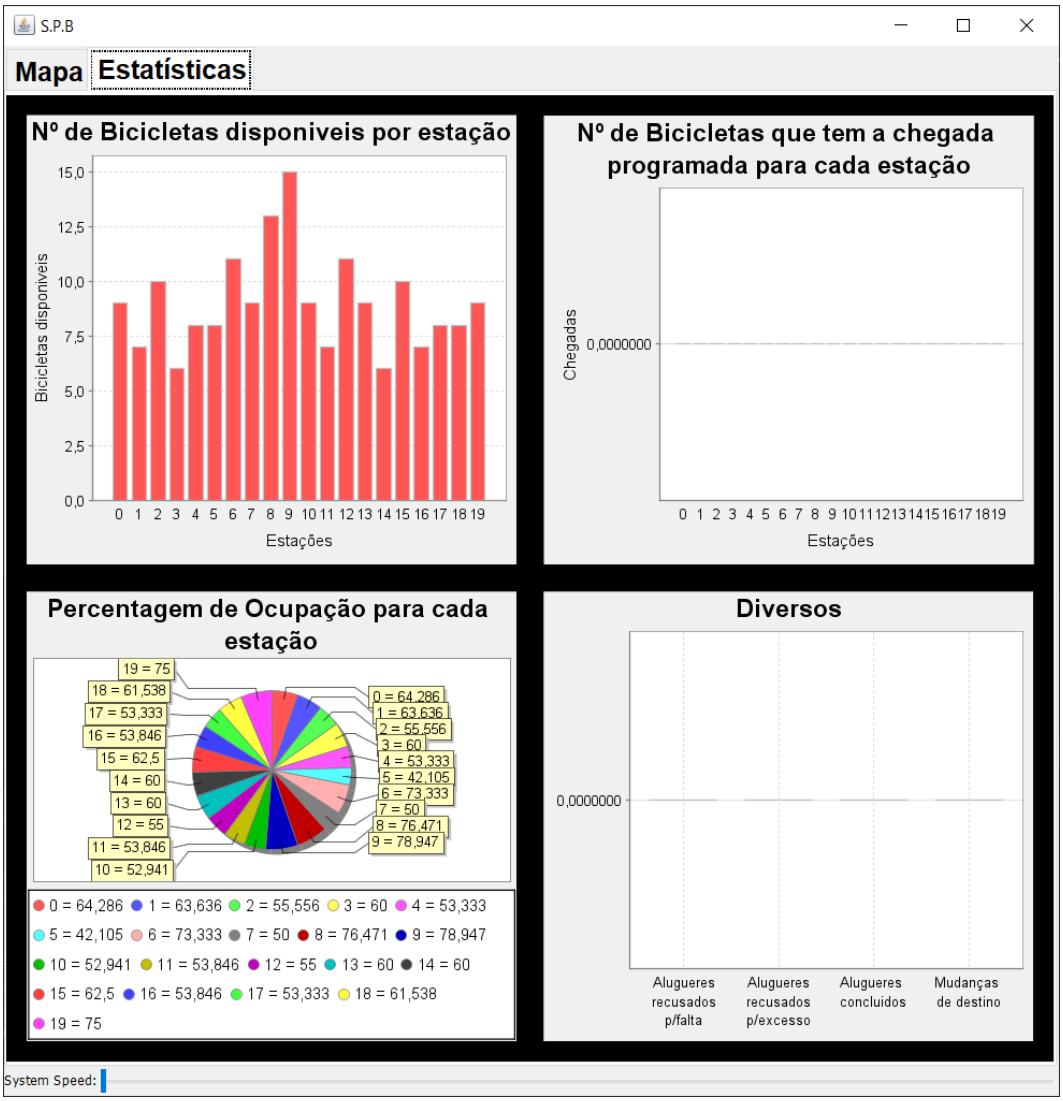

Figura 9. Estatísticas iniciais do sistema

Com o objetivo de simular um dia e testar a mesma situação várias vezes, foram criados testes com 20 estações e 1500 utilizadores. As estações têm as seguintes capacidades {14, 11, 18, 10, 15, 19, 15, 18, 17, 19, 17, 13, 20, 15, 10, 16, 13, 15, 13, 12}, resultando numa capacidade média de 15 bicicletas por estação e um total de 300.
Quanto à disponibilidade, cada estação tem os seguintes valores iniciais {9, 7, 10, 6, 8, 8, 11, 9, 13, 15, 9, 7, 11, 9, 6, 10, 7, 8, 8, 9}, resultando numa média de 9 por estação e um total de 180, o que leva a que a ocupação média tome o valor $\frac{180}{300} = 60\%$. Além de pré-definir a capacidade e disponibilidade, também foram sempre usados o mesmo raio e posição para cada estação.

A representação deste teste será apresentada adiante, na secção 6.1.

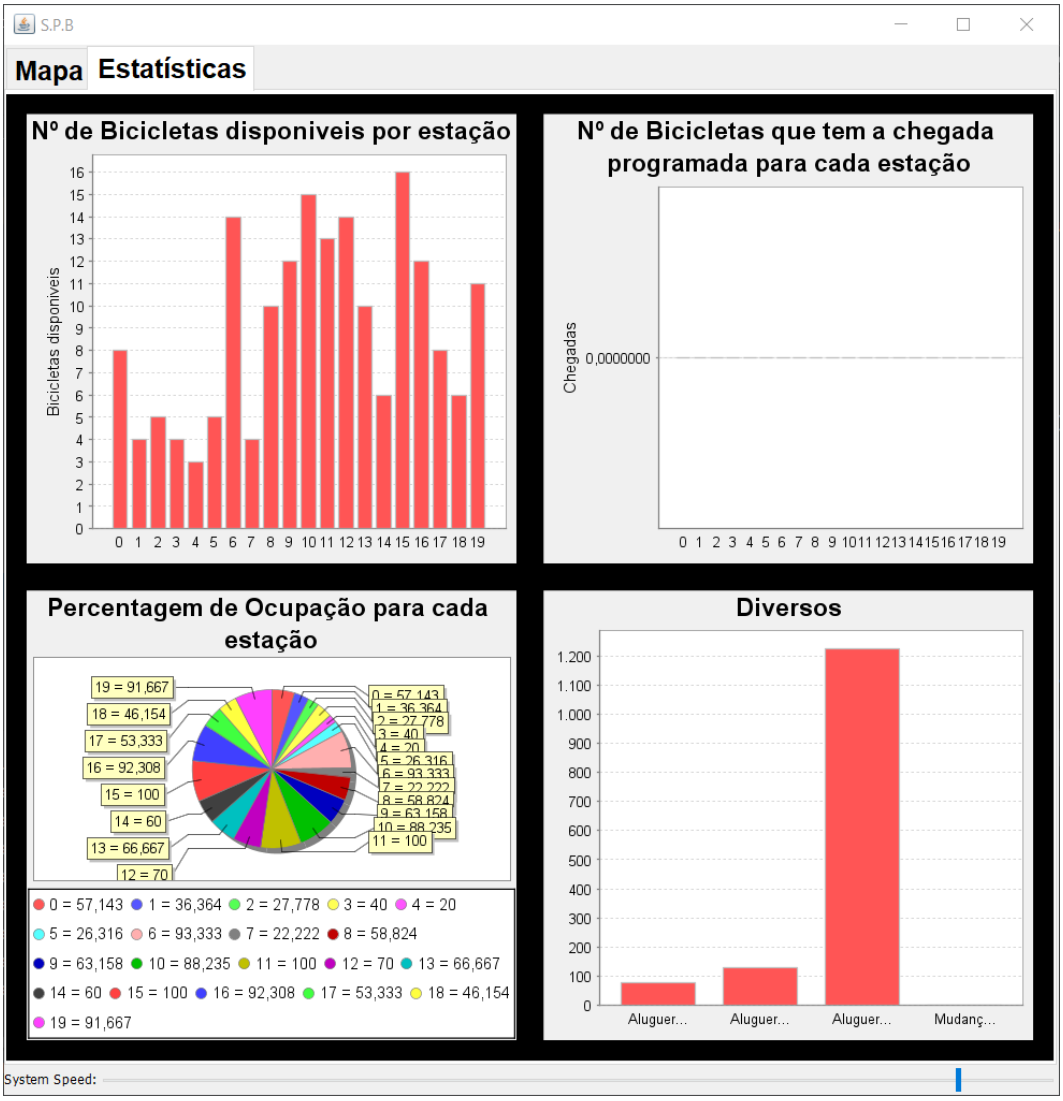

Figura 10. Resultados do sistema sem Negociação

Caso não exista negociação para mudanças de destino, os resultados são pouco satisfatórios. Dos 1500 utilizadores criados apenas 1225 é que aceitaram os preços provenientes das

estações (82%). É de ressalvar que quanto maior é a percentagem de bicicletas numa certa estação destino, maior será o valor monetário apresentado (para evitar a sobrelotação), ou seja, neste caso, 275 utilizadores rejeitaram o montante pedido.
Por outro lado, houve 78 alugueres recusados por falta de bicicletas e 130 por sobrelotação, resultando num total de 208, o que se transcreve numa percentagem de $\frac{208}{1433} \approx 14.5\%$ de alugueres recusados.

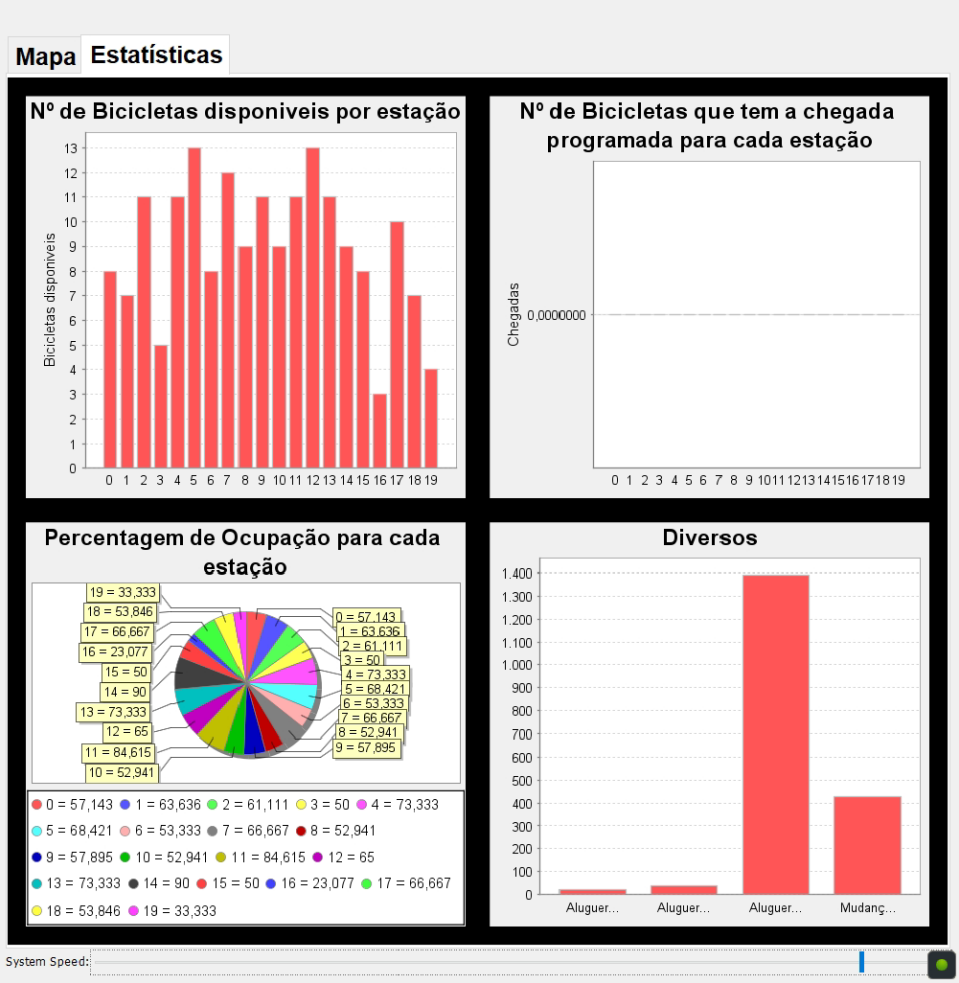

Figura 11. Resultados do sistema com Negociação

Com a nossa implementação de incentivos, já foram obtidos resultados bastante aceitáveis. Dos 1500 utilizadores, 1397 ficaram satisfeitos com os preços apresentados pelas estações (93%). Assim, com os mesmos algoritmos de satisfação dos utilizadores e de cálculo de preço das estações, foram aceites mais alugueres por parte do utilizador do que no caso anterior, o que significa que, em geral, as bicicletas encontram-se melhor distribuídas.
É, ainda, possível constatar uma diminuição de alugueres recusados por parte das estações, tanto por falta, como por excesso de bicicletas, sendo que estes valores diminuíram para 29 e 57, respetivamente, o que resulta num total de 86 alugueres recusados. Este valor, só por si, é um valor inferior, todavia é necessário ter em consideração que, em média, existem mais bicicletas a circular no sistema, o que diminui o número de bicicletas disponíveis em

cada estação. Por fim, a percentagem de alugueres recusados foi de $\frac{86}{1483} \approx 5.8\%$, valor bastante positivo para a demonstração do funcionamento do nosso programa.
É, ainda, de ressalvar que foram realizadas 438 mudanças de destino, representando uma percentagem de $\frac{434}{1397} \approx 31.1\%$ dos alugueres realizados, o que reforça o bom funcionamento do sistema.

Para alcançar este resultado foi necessário alterar os valores limite da ocupação baixa e média para 35% e 60%, respetivamente.

6.1 Teste criado

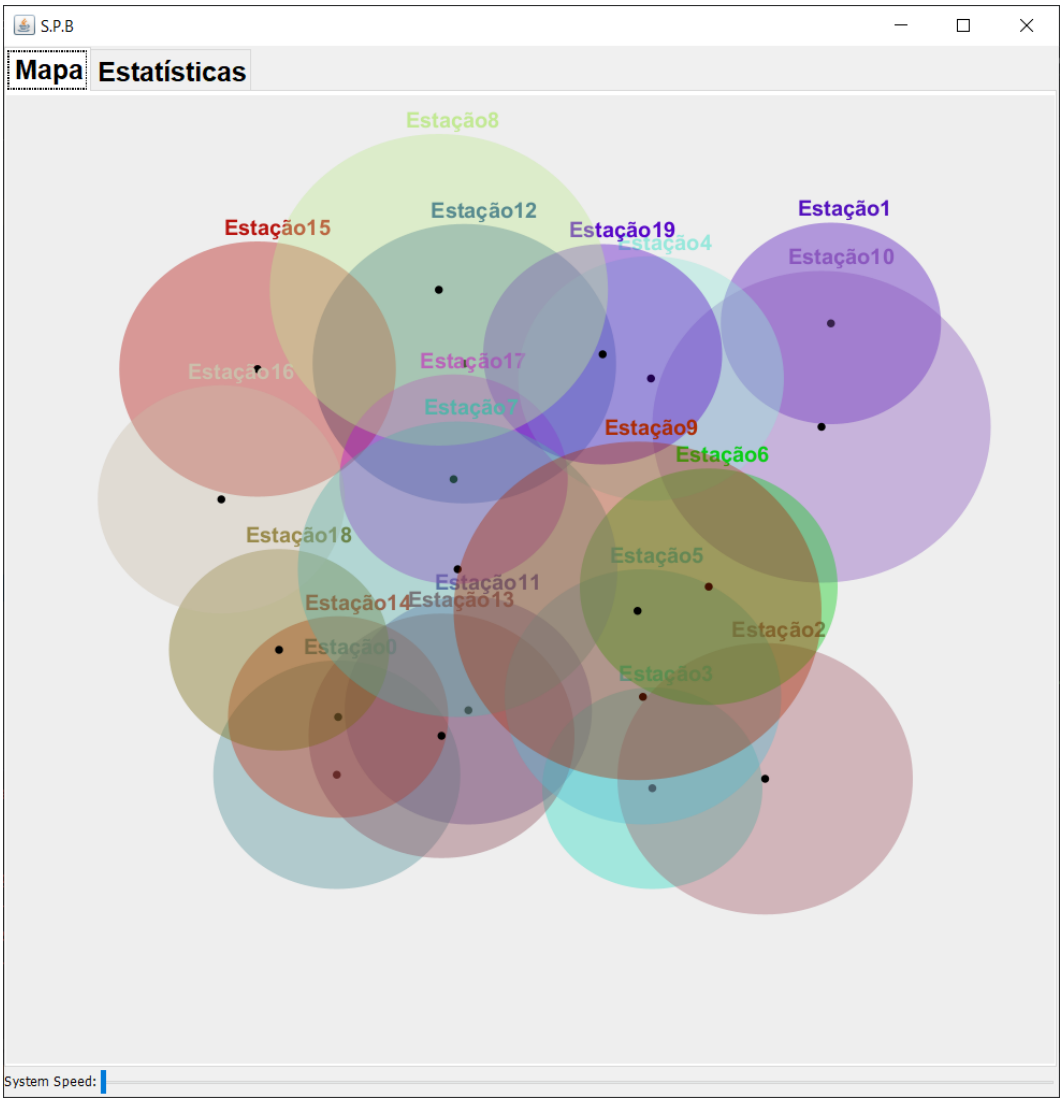

Figura 12. Mapa do teste utilizado

7 Conclusão

Como foi referido ao longo do presente relatório, o objetivo principal é o desenvolvimento e especificação dos vários agentes e da interação e comunicação entre os mesmos. Assim sendo, realizámos, inicialmente, uma descrição geral do problema, onde foi feito um levantamento de requisitos e analisado cada componente útil para o sistema. De seguida, foi feita a descrição de cada agente, das suas funcionalidades e comportamento.
As várias secções estão acompanhadas por diagramas AUML, para permitir uma melhor compreensão de cada etapa elaborada.

O planeamento do sistema multi-agente que visa monitorizar e implementar um serviço SPB é, sem dúvida, uma fase muito importante para o passo de implementação da simulação do sistema.
As tarefas mais importantes são o mecanismo de balanceamento de bicicletas entre as várias estações e o equilíbrio do processamento pelos vários agentes, pois, se o AI for sobrecarregado e tiver demasiadas responsabilidades, pode tornar-se um *bottleneck* para o sistema, pelo que, a decisão da atribuição de descontos deve ser feita pelas AEs.

Para a segunda fase, foi implementado o sistema aqui exposto, desenvolvida a negociação e deliberação de propostas (aspecto fulcral para o bom funcionamento do SPB), entre outros, sendo que cada um dos componentes apresentados compuseram a nossa solução. Além disto, foram apresentados vários problemas com que nos deparámos e respetivas soluções. Por último, foi apresentada uma secção que faz uma breve introdução à interface, bem como uma secção onde são debatidos os resultados obtidos no sistema num caso de teste.

Após alguma investigação, concluímos que este tipo de sistema se trata de um sistema de *Automated stations*, arquitetura esta que consiste no aluguer e receção de bicicletas entre as várias estações, todas pertencentes ao mesmo sistema. As estações necessitam de estar equipadas com um tipo de bloqueio especial, de forma a que apenas libertem bicicletas através do controlo de um computador.

Existem várias vantagens para o meio ambiente num sistema deste tipo, tais como a redução de emissões de poluição e ruído, bem como a melhoria do bem estar da população, reduzindo o congestionamento em áreas urbanas e aumentando a sua mobilidade, sendo, ainda, uma boa forma de praticar exercício físico e um meio de transporte mais saudável.
Todavia, existem algumas desvantagens como a criação de estações (que é, normalmente, feita à custa da redução de lugares de estacionamento disponíveis para automóveis), o risco de vandalismo, acidentes ou roubos sendo, por isso, necessário implementar mecanismos para responsabilizar o utilizador pelos seus atos, bem como a implementação de alguns sistemas de segurança nas estações e bicicletas.
Outra preocupação que deve ser tida em conta é a questão da segurança pessoal, pois é necessário que a estação possa fornecer algum tipo de equipamento, como capacetes, entre outros.

Em suma, para um sistema deste estilo funcionar, é, então, necessário garantir certas condições, como manter o sistema equilibrado (isto é, manter um certo equilíbrio nas quantidades de bicicletas disponíveis em cada estação, de modo a que nenhuma estação fique congestionada com bicicletas e sem clientes ou sem bicicletas para alugar e bastante procura) para permitir uma melhor *performance*. Visto que o sistema a desenvolver deverá ser automatizado, é também necessário garantir a interação entre as várias estações do sistema e consequente colaboração.

Referências

1. BRUTUS, S., JAVADIAN, R. & PANACCIO, A.J. (2017); '**Cycling, car, or public transit: a study of stress and mood upon arrival at work**'. *International Journal of Workplace Health Management*, vol. 10, no. 13.
2. **List of bicycle-sharing systems**. Disponível em: `https://en.wikipedia.org/wiki/List_of_bicycle-sharing_systems`
3. **FIPA** (*Foundation for Intelligent Physical Agents*). Disponível em: `http://www.fipa.org`
4. KLOIMÜLLNER, C., PAPAZEK, P., HU, B. & RAIDL, G. (2014). '**Balancing Bicycle Sharing Systems: An Approach for the Dynamic Case**'. *Evolutionary Computation in Combinatorial Optimisation*.