Parque de Estacionamento

Agentes e Inteligência Artificial Distribuída

4º ano - 1º semestre

Mestrado Integrado em Engenharia Informática e Computação

Bernardo Belchior - up201405381@fe.up.pt
Maria João Mira Paulo - up201403820@fe.up.pt
Nuno Miguel Mendes Ramos - up201405498@fe.up.pt

11 de Dezembro de 2017

Conteúdo

1 Introdução

1.1 Descrição do cenário

Atualmente deparamos-nos nos com uma grande dificuldade de estacionamento em áreas metropolitanas. O elevado fluxo de trânsito e a lotação dos parques de estacionamento geram uma procura exaustiva de estacionamento no centro da cidade. Ao mesmo tempo, parques na periferia encontram-se na sua maioria desocupados.

Com este desequilíbrio surge a ideia de provocar uma maior procura aos parques não concentrados no centro da cidade e, assim, regular a alocação em parques de estacionamento.

Uma solução encontrada é a aplicação de ***Automated Dynamic Pricing (ADP)***, onde o preço de estacionamento durante um certo período de tempo varia conforme o dia e hora da semana, o que, porém, poderá ter efeitos nocivos na sociedade. O lucro gerado pelo parque e o bem-estar social são algumas das métricas que são avaliadas através deste projeto.

1.2 Objetivos do trabalho

Um Sistema Multi-Agente *(SMA)* é um sistema computacional composto por múltiplos agentes. Estes agentes exibem um comportamento autónomo mas ao mesmo tempo interagem com os outros agentes presentes no sistema, trabalhando juntos de forma a desempenhar determinadas tarefas ou satisfazer um conjunto de objetivos.

O objetivo do trabalho baseia-se na implementação de uma simulação multi-agente com o propósito de estudar diferentes formas de gestão dos preços praticados pelos parques de estacionamento.

Desta forma, pretende-se estudar o efeito da utilização de uma tarifa dinâmica ao invés do uso de uma tarifa estática no lucro gerado pelos parques de estacionamento, na competitividade e no bem-estar social.

2 Especificação

2.1 Identificação e caracterização dos agentes

Consideramos dois tipos de agentes, *Parking Facilities* e *Drivers*.

2.1.1 *Drivers*

Arquitetura e Comportamento

Os agentes do tipo *Driver* podem ser *Guided* ou *Explorer* e são ambos agentes **baseados em utilidade**, isto é, preocupam-se em tentar maximizar as suas expectativas e, através de uma função de utilidade estabelecer preferências entre sequências de estados que permitam atingir os mesmos objetivos. Atingir a maior utilidade significa atingir a maior medida de satisfação.

1. ***Guided Drivers*** são caracterizados pelo seu total conhecimento acerca de preços e localização dos parques de estacionamento. São agentes que sabem quais as condições mais satisfatórias e que estão seguros quanto às suas possibilidades. Este agente, exatamente por ter total conhecimento dos preços em vigor, é o que mais rapidamente reage à subida de preços.

   Quando entram no sistema sabem qual o parque que maximiza a sua utilidade e dirigem-se a esse. No caso do parque não ter lugares disponíveis repetem o processo com os restantes. Se a única solução possível resultar numa utilidade negativa para o condutor, o agente sai do sistema.

   - **Medida de desempenho:** Encontrar um determinado parque de estacionamento com lugares disponíveis e com o preço habitual;
   - **Ambiente:** Mapa da Cidade;
   - **Atuadores:** Direcionar-se ao próximo parque que conhece ou estacionar;
   - **Sensores:** Disponibilidade do parque e preço;

Figura 1: Imagem representativa de um *Guided Driver*

2. ***Explorer Drivers*** são caracterizados pela sua ignorância, isto é, por não terem consciência dos preços realizados pelos parques. Consequentemente, procuram apenas um parque de estacionamento livre perto do seu destino e, consequentemente, não reagem tão rapidamente à mudança de tarifa.

   - **Medida de desempenho:** Encontrar um parque de estacionamento disponível, com um preço satisfatório e perto do local de destino de forma a que o condutor não tenha que se deslocar uma grande distância a pé;
   - **Ambiente:** Mapa da Cidade;
   - **Atuadores:** Continuar a procurar ou estacionar;

- **Sensores:** Disponibilidade do parque, proximidade ao local de destino e preço;

Figura 2: Imagem representativa de um *Explorer Driver*

Os agentes *Drivers* têm conhecimento do seu respetivo *ID*, das coordenadas do local de destino pretendido e do tempo de estacionamento.

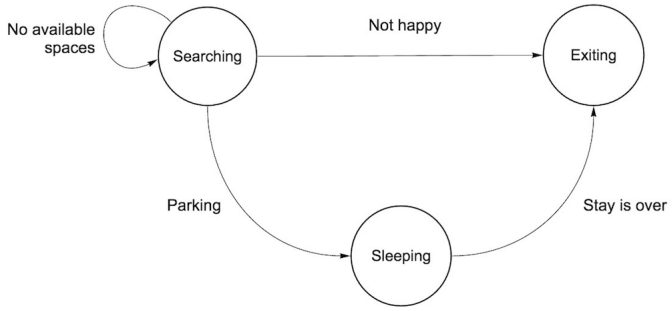

Figura 3: Máquina de estados de um condutor

Estratégias

A função de utilidade é a mesma para os dois tipos de *Drivers*:

$$w_{i,j} = C_i - pr_i * (P_{i,j})^u - w_i * (W_{i,j})^v$$

$$P_{i,j} = \alpha * price_j * duration\_of\_stay_i$$

$$W_{i,j} = \beta * distance\_to\_destination_{i,j}$$

C_i representa a utilidade, ou seja, a satisfação atingida pelo condutor ao chegar ao seu destino sem ter que se deslocar a pé ou pagar pelo estacionamento num parque. Este valor é atribuído aleatoriamente a cada condutor.

$P_{i,j}$ representa o preço a pagar pelo condutor tendo em conta o tempo no qual teve estacionado no parque, multiplicado por uma constante *alpha*.

$W_{i,j}$ representa o esforço dedicado pelo condutor de forma a chegar do parque de estacionamento ao seu destino, multiplicado por uma constante *beta*.

As constantes α e β permitem obter uma comparação mais justa e realista do impacto dos metros e euros na utilidade do condutor.

pr_i, w_i tratam-se de valores gerados aleatoriamente que simbolizam a personalidade do agente i. Isto é, traduzem a utilidade gerada no agente ao pagar a quantia $P_{i,j}$ e ao andar $W_{i,j}$ metros a pé.

Os expoentes u e v pretendem criar não linearidade no impacto do preço e da distância a percorrer a pé. Por exemplo, o acréscimo de 1 € ao custo do parque a um cliente que terá de pagar 2 € terá, obviamente, mais impacto do que num cliente que deve um valor maior, como 20 €.

2.1.2 *Parking Facilities*

Arquitetura e Comportamento

Os agentes *Parking Facilities* podem ser de dois tipos, dependendo do esquema de preços que querem adotar, *Static* ou *Dynamic*. Existem sete esquemas de preços a aplicar, um para cada dia da semana.

Ambos os agentes podem reagir aos comportamentos dos *Drivers*, mas não devem interagir com os mesmos, isto é, não existe negociação de preços. Trata-se, por isso, de agentes reativos simples com um estado interno, ou seja, que mantêm um relógio interno com informação atualizada do tempo de estacionamento e, em função disso, calculam o esquema de preço.

Os agentes *Parking Facilities* têm conhecimento da localização, nome, preço, capacidade do parque, preço por hora, máximo preço por hora, máximo preço por dia, lucro da última semana e o parâmetro de inflação dos preços.

1. *Static*
   Adotam um esquema de preço estático, ou seja, que não varia com fatores do ambiente.

2. *Dynamic*
   Adotam um esquema de preço dinâmico, procurando atingir um maior lucro. O preço que exercem varia em função do dia da semana e da hora. A cada final de semana, é feita uma atualização aos esquemas de preço com o objetivo de maximizar o lucro obtido.

- **Medida de desempenho:** Atingir o maior lucro;
- **Ambiente:** Mapa da Cidade;
- **Atuadores:** Receber condutor, apresentar preço;
- **Sensores:** Disponibilidade do parque, dia da semana e hora;

Estratégias

O preço final de estacionamento é calculado tendo em conta três parâmetros: o preço por minuto, um valor mínimo que deve ser pago por todos os clientes e um valor máximo que, pode ser pago, por exemplo, no caso do condutor usufruir de um dia inteiro estacionado no parque.

Existe, além disso, outro parâmetro que pode causar tanto inflações como deflações no preço por minuto. O valor por defeito deste parâmetro é 1 e, no caso de aumentar ou diminuir durante a otimização, causará variações, hora a hora, no preço final. Por exemplo, no caso do preço por hora ser 1, e a inflação 1.10, o preço irá variar de acordo com tabela a seguir.

O preço final pode depender ainda da capacidade do parque. Assim, caso a disponibilidade do parque varie entre os 30% e os 70%, o preço mantêm-se constante. Caso contrário, o valor oscila linearmente de acordo com o parâmetro a ser atualizado.

Hora	Preço	Preço Final
1	1	1
2	1.1	2.1
3	1.21	3.31
4	1.331	4.641

Tabela 1: Preço ajustado tendo em conta o parâmetro de oscilação

Cada parque de estacionamento deve atualizar os parâmetros referidos anteriormente exclusivamente uma vez por semana e, além disso, cada parâmetro deve ser atualizado cinco vezes antes de atualizar o próximo. Desta forma garante-se uma maior confiança e certeza no impacto de cada um dos parâmetros.

A expressão usada na atualização de parâmetros, no caso de, anteriormente, ter existido uma inflação no preço é:

$$x_i + 1 = x_i + \delta.x_i.(\gamma - 1)$$

Caso contrário, ou seja, caso se tenha verificado uma deflação no preço, deve ter em conta a expressão:

$$x_i + 1 = x_i - \delta.x_i.(\gamma - 1)$$

Sendo que, δ representa a taxa de aprendizagem do agente e, γ, representa a diferença, em percentagem, dos lucros obtidos na semana anterior. Este tipo de aprendizagem permite que o parque aumente os preços para valores considerados justos tanto pelo dono do parque, como pela sociedade.

Tabela 2: *Parking Facilities*

Nome	*Capaci*	*Preço hora*	Preço max p/dia	Localização
Cabergerweg	698	1.43	9.00	(2,46)
Sphinx-terrein	500	2.22	13.00	(35,65)
De griend	351	2.22	13.00	(44,71)
Bassin	407	2.73	25.00	(47,49)
P + R station Maastricht	335	1.89	13.00	(58,73)
Mosae forum	1082	2.73	25.00	(51,55)
Vrijthof	545	3.53	35.00	(61,38)
P + R meerssenerweg	65	1.89	13.00	(73,69)
O.L. vrouweparking	350	2.73	25.00	(79,62)
Plein 1992	449	2.22	13.00	(72,34)
De colonel	297	2.22	13.00	(79,17)
Bonnefantenmuseum	303	1.43	25.00	(88,44)
Brusselse poort	610	1.43	25.00	(90,51)

A Tabela 2 contém os treze parques de estacionamento usados na simulação. Cada linha da tabela representa um parque incluindo o seu nome, capacidade, preço por hora, preço máximo por dia e a sua localização na *grid*.

2.2 Ambiente

O ambiente escolhido simula o centro de uma cidade, uma vez que se trata do local onde existe uma maior procura de estacionamento e, por isso, uma maior oferta de parques de estacionamento.

Este mapa é representado através de uma matriz que inclui os parques de estacionamento e as ruas que lhes dão acesso.

2.3 Protocolos de Interação

Neste projeto, não implementamos nenhum protocolo de interação. Não foi necessário a utilização de mensagens entre agentes. A situação onde inicialmente pensamos que seria necessária era aquando da chegada de um condutor a um parque, o primeiro teria de pergunta ao segundo se existe lugares disponíveis.

No entanto, acabamos por resolver este problema ao criamos um ***Set*** de parques na classe *Launcher* e dentro de cada parque criamos um ***Map*** onde a chave é o *id* do condutor e o valor, o objeto do condutor. Fazendo a capacidade do parque menos o *size* do ***Map*** obtemos os lugares disponíveis.

```
private static Set<ParkingFacilityAgent> parkingFacilities
        = new HashSet<>();
protected Map<String, DriverAgent> parkedCars
        = new HashMap<>();
```

3 Desenvolvimento

3.1 Plataforma e ferramentas utilizadas

Em seguida, segue uma breve descrição do ambiente de desenvolvimento, plataforma e ferramentas utilizadas.

Ambiente de Desenvolvimento

Este trabalho foi realizado no sistema operativo *Windows 10*, usando o *IDE Eclipse Oxygen.1*, na perspetiva *ReLogo*.

Ferramentas Utilizadas

Um dos propósitos da unidade curricular Agentes e Inteligência Artificial Distribuída é a utilização de plataformas capazes de criar agentes de uma forma distribuída e de executar simulações sobre esse agentes.

Desta forma e, com o propósito de atingir o objetivo final do trabalho, recorreremos ao uso de três plataformas distintas: *JADE*, *REPAST* e *SAJaS*.

JADE

JADE é uma *framework open source* implementada em Java que simplifica a implementação de sistemas multi-agente distribuídos de acordo com as especificações da FIPA (*Foundation of Intelligent Physical Agents*).

Uma aplicação baseada em JADE é composta por agentes, associados a um nome único, que executam tarefas de acordo com um comportamento definido e interagem através de mensagens.

A plataforma dispõe de *containers*, onde residem os agentes. Estes agentes podem não estar, necessariamente, na mesma máquina. Cada máquina tem uma JVM (*Java Virtual Machine*) e cada agente tem uma *thread*, atingindo-se, assim, uma plataforma .

As configurações do JADE podem ser controladas através de uma interface. A partir desta interface é possível criar novos agentes, alterar a configuração dos mesmos dinamicamente ou alterar as máquinas onde estão contidos.

REPAST

A *framework REPAST* é uma plataforma *open source* que simplifica a implementação de simulações baseadas em agentes ao disponibilizar ao utilizador o uso de uma biblioteca com uma grande variedade de códigos e exemplos, para criar, executar, apresentar e recolher dados dos modelos desenvolvidos.

Desta forma, ao facilitar a recolha de dados da simulação e a visualização destes através de gráficos ou registos, o *REPAST* possibilita a análise da interação entre os agentes.

SAJaS

A *framework SAJaS* é uma API que faz a ponte entre uma plataforma de desenvolvimento de sistemas multi-agente, JADE, e uma plataforma de simulações multi-agente, REPAST.

A ferramenta *SAJaS* melhora *frameworks* de simulação de sistemas multi-agente, com determinadas características do JADE. Além disso, esta tecnologia

promove, para programadores de JADE, um desenvolvimento mais rápido de modelos de simulação.

3.2 Estrutura da aplicação

A aplicação encontra-se estruturada em cinco *packages* distintos:

- ***parking_lots_simulation***:

  package principal, que contém todos os outros módulos e classes principais:

  - *Launcher*: É a classe responsável por ciar os agentes, a *grid* e por lançar, em seguida, a simulação;
  - *DriverAgent*: É a classe que representa um *driver* e estende a classe *Agents*. Além disso contém toda a lógica associada a um condutor, como, por exemplo, o destino, tempo no parque e cálculo da utilidade;
  - *Class ParkingFacilityAgent*: É a classe que representa um *Parking Facility* e estende a classe *Agents*. Além disso contém toda a lógica associada a um parque, como, por exemplo, capacidade do parque, preço por hora e preço máximo por hora;
  - *Class GodAgent*: É a classe que representa o agente que está responsável pela geração de novos *drivers* e estende a classe *Agents*;
  - *Statistics*: É a classe que se responsabiliza por imprimir estatísticas relevantes durante a execução do programa.
  - *CSVParser*: É a classe responsável por ler dados de um ficheiro CSV parser e, partindo destes, inicializar os parques;

- ***behaviours***: É o módulo onde se encontram os diferentes *behaviours* implementados:

  - *DynamicParkingFacilityBehaviour*: É a classe que simula o comportamento dos agentes *parking facilities* e responsável por atualizar os parâmetros dos parques com preço dinâmico;
  - *ParkRevenueBehaviour*: É a classe que simula o comportamento dos agentes *parking facilities* e responsável por, semanalmente, recolher os lucros do parque e recomeçar o contador do lucro;
  - *ExplorerDriverBehaviour*: É a classe que simula o comportamento dos *explorer drivers*, procurando encontrar o parque mais próximo do destino do agente;
  - *GuidedDriverBehaviour*: É a classe que simula o comportamento dos *guided drivers*, que procuram encontrar o parque que maximiza a sua função de utilidade;
  - *ParkSearchingBehaviour*: É a classe que simula o comportamento dos *driver agents* e responsável por guiar o agente até ao parque que escolheu;
  - *GodBehaviour*: É a classe que simula o comportamento adicionado ao *god agent*. Este agente é responsável pela geração de novos *driver agents*;
  - *SleepBehaviour*: É a classe que representa o comportamento adicionado aos *driver agents* , simulando o tempo que um condutor passa dentro de um parque;

- ***comparators***: Módulo que contém classes que implementam a interface *Comparator*:
  - *DistanceComparator*: É a classe que compara a distância de dois parques a um *driver agent*. Tem como objetivo encontrar o parque mais próximo do condutor;
- ***debug***:
  - *DynamicParkingFacilityAgent* e *StaticParkingFacilityAgent*: São as classes utilizadas para diferenciar os dois tipos de agentes *parking facilities*;
  - *ExplorerDriverAgent* e *GuidedDriverAgent* : São as classes utilizadas para diferenciar os dois tipos de agentes *drivers*;
- ***exceptions***: implementações de exceções personalizadas:
  - *NoValidDestinationException*: É a exceção lançada quando o *driver agent* não encontra um destino válido;
- ***population***: Módulo que contém a lógica da geração dos *drivers agents* ao longo da simulação:
  - *Interface PopulationCalculator*;
  - *WeekDayPopulationCalculator*: É a classe que calcula o número de *driver agents* que deverá existir a uma determinada altura da semana;
  - *WeekendPopulationCalculator*: É a classe que calcula o número de *driver agents* que deverá existir a determinada altura do fim-de-semana;

Diagrama de Classes

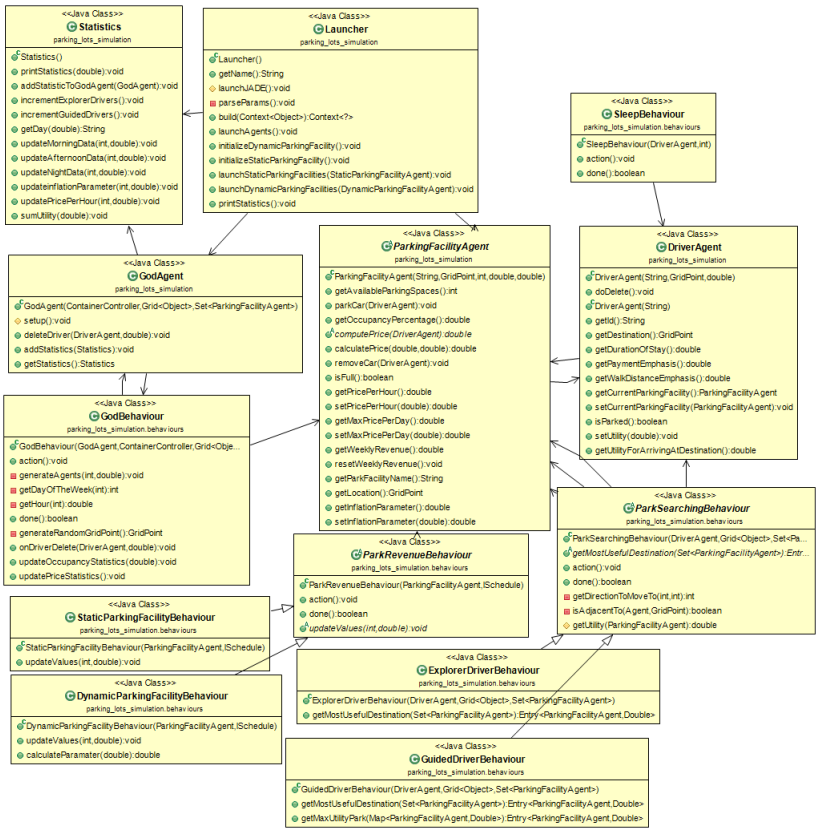

Figura 4: *Diagrama de Classes*

3.3 Detalhes Relevantes da Implementação

3.3.1 Geração de Agentes

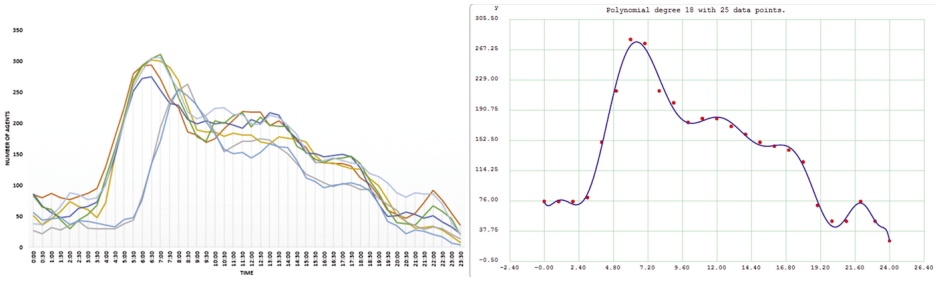

Figura 5

Para gerar os *driver agents* continuamente ao longo da simulação, foi criado um *god agent* cuja função se trata da criação de novos condutores ao longo do tempo.

Foram modeladas duas funções distintas (ver figura 5) com o objetivo de obter o número de condutores: uma função para os dias das semana e outra para

o fim-de-semana. A cada *tick* este agente, que guarda o número de condutores no sistema, verifica se o número de condutores está correto.

Caso queria obter o número de agentes a uma determinada altura da semana o agente utiliza a função *calculate* no ficheiro ***WeekDayPopulationCalculator***, caso seja um dia de fim-de-semana utiliza a mesma função mas pertencente ao ficheiro ***WeekendPopulationCalculator***. Ambos estes ficheiros implementam a interface ***PopulationCalculator*** e encontram-se no *package* ***parking_lots_simulation.population***.

Assim que são gerados os condutores, é gerada, igualmente, a sua posição inicial, o seu destino, o tempo de duração no parque e algumas constantes utilizadas no cálculo da utilidade.

Para a geração do tempo de duração no parque, foi utilizada uma **distribuição normal** com centro nas 8 horas. Esta distribuição normal foi aplicada de forma a tornar a simulação mais realista.

3.3.2 Função de Utilidade

```
protected double getUtility(ParkingFacilityAgent
    parkingFacility) {
double u = 0.9;
double v = 0.9;

double utility = driver.getUtilityForArrivingAtDestination();
double distanceToDestination =
    grid.getDistance(driver.getDestination(),
    grid.getLocation(parkingFacility));

double payment = DriverAgent.alpha *
    parkingFacility.computePrice(driver);
double effort = DriverAgent.beta * distanceToDestination;

return utility - driver.getPaymentEmphasis() *
    Math.pow(payment, u) - driver.getWalkDistanceEmphasis() *
    Math.pow(effort, v);

}
```

A **função de utilidade** utilizada na simulação representa a felicidade atingida pelo condutor ao estacionar num determinado parque.

A utilização dos valores u e v com valor 0.9 resulta numa utilidade não linear. A variável *payment* reflete o preço que o condutor terá que pegar ao estacionar no parque e a variável *effort*, o esforço que terá que fazer para se deslocar a pé do parque ao seu destino.

As variáveis *paymentEmphasis* e *walkDistanceEmphasis*, presentes na classe Driver e, utilizadas no cálculo da utilidade, simulam a personalidade dos agentes. A primeira simboliza a reação do condutor ao ter pagar um determinado preço pelo estacionamento no parque e, a segunda, reflete a empatia do condutor ao ter que se deslocar a pé, desde o parque ao seu destino.

3.3.3 Atualização dos parâmetros de preço

```
public void updateValues(int currentWeek, double
    currentRevenue) {
   gamma = (currentRevenue == 0) ? 0 : currentRevenue /
       lastWeekRevenue;

   if (consecutiveUpdates == NUMBER_OF_CONSECUTIVE_UPDATES) {
         consecutiveUpdates = 0;
         calculationModifier = 1;
         currentParameterIndex++;
   }

   double newValue = 0, oldValue = 0;
   switch (PARAMETERS[currentParameterIndex %
       PARAMETERS.length]) {
   case PRICE_PER_HOUR:
         oldValue = parkingFacility.getPricePerHour();
         newValue=parkingFacility.
        setPricePerHour(calculateParamater(oldValue));
         break;
   case MAX_PRICE_PER_DAY:
         oldValue = parkingFacility.getMaxPricePerDay();
         newValue = parkingFacility.
        setMaxPricePerDay(calculateParamater(oldValue));
         break;
   case INFLATION_RATE:
         oldValue = parkingFacility.getInflationParameter();
         newValue = parkingFacility.
        setInflationParameter(calculateParamater(oldValue));
         break;
   default:
         break;
   }
}
```

A função representada é um excerto da função *updateValues*, responsável por atualizar os parâmetros dos parques dinâmicos. Como se pode ver através da função, estes agentes têm três parâmetros: o preço por hora, o preço máximo que um condutor paga por dia e a taxa de inflação, referida na tabela 1.

No final de cada semana, e antes de recomeçar o contador de lucro, o parque atualiza os seus preços. Assim, calcula o *gamma*, que denota a variação percentual na receita, em comparação com a semana anterior. Além disso, se o parâmetro já foi atualizado cinco semanas consecutivas, passa para o próximo, caso contrário, mantém-se no atual.

Posteriormente, atualizam-se os parâmetros. A fórmula a utilizar depende se na atualização anterior o valor aumentou ou diminui. No caso de um acréscimo no valor:

$$x_i + 1 = x_i + \delta.x_i.(\gamma - 1)$$

No caso de um decréscimo no valor:

$$x_i + 1 = x_i - \delta.x_i.(\gamma - 1)$$

Ao fim de 15 semanas conclui-se o primeiro ciclo de atualização. Ou seja, durante cinco semanas consecutivas foram atualizados três parâmetros.

3.3.4 GUI

A alteração dos parâmetros pode ser feita na janela que abre automaticamente pelo *Repast*, onde corre a simulação.

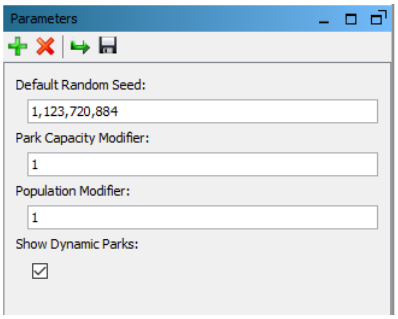

Figura 6: Parâmetros

1. ***Default Random Seed***: *Seed* que permite que a simulação com parques dinâmicos tenha as mesmas condições que a simulação com parques estáticos. A utilização da mesma *seed* permite gerar os condutores exatamente no mesmo ponto de partida. Isto deve-se ao facto do computador gerar números *pseudo-random*. Assim, é gerada a mesma sequência de números quando executados a partir da mesma *seed*.

2. ***Park Capacity Modifier***: Multiplicador da capacidade dos *parking facilities agents*, por exemplo, se tiver a 0.5, a capacidade será metade da normal. Serve para testar diferentes cenários.

3. ***Population Modifier***: Multiplicador do número de *drivers agents* na simulação, por exemplo, se tiver a dois terá o dobro de condutores na simulação. Serve para testar diferentes cenários.

4. ***Show Dynamic Parks***: Variável *booleana* que permite escolher se os parques presentes na simulação são parques dinâmicos ou estáticos. Por defeito, são mostrados os parques dinâmicos.

3.3.5 Estatísticas

Todos os dias da semana são impressas estatísticas na linha de comandos. Com estes dados, é possível aceder a informações como dia da semana, número de agentes *Guided Drivers* e *Explorer Drivers*, ocupação média dos parques nas três importantes fases do dia, 9:00h da manhã, 16:00h da tarde e 20:00h da noite e, por fim, felicidade média dos condutores.

```
--------  New Prices  ----------
Park      |   1   |   2   |   3   |   4   |   5   |   6   |   7   |   8   |   9   |   10  |   11  |   12  |   13 |
Inflation     | 1.04 | 0.96 | 0.98 | 1.12 | 1.03 | 0.99 | 0.91 | 1.02 | 1.02 | 0.98 | 1.12 | 0.83 | 0.93 |
Price per hour| 0.89€ | 1.44€ | 1.18€ | 1.00€ | 1.48€ | 0.85€ | 1.53€ | 1.48€ | 1.21€ | 0.92€ | 1.18€ | 1.11€ | 1.05€ |

Week 14
Day of the week: Monday
Number of Explorer Drives: 208
Number of Guided Drives: 131

Park   |   1   |   2   |   3   |   4   |   5   |   6   |   7   |   8   |   9   |   10  |   11  |   12  |   13 |
9:00h  | 8.25% | 0.83% | 3.20% | 4.62% | 1.76% | 4.44% | 3.71% | 2.46% | 7.41% | 3.11% | 0.85% | 3.34% | 0.60% |
16:00h | 3.63% | 0.18% | 1.60% | 1.54% | 0.92% | 2.29% | 1.14% | 1.72% | 5.05% | 1.97% | 0.28% | 0.89% | 0.00% |
20:00h | 2.31% | 0.18% | 1.60% | 1.54% | 0.55% | 2.87% | 0.86% | 2.70% | 6.06% | 1.48% | 0.57% | 0.89% | 0.30% |

Drivers Average Utility: 7367.515256037301

---------------------------------
```

Figura 7: Estatísticas

Além disso, todas as segundas feiras as estatísticas contém, além das informações atrás descritas, os novos preços a ser aplicados pelos agentes *Dynamic Parking Facilities*. Assim, torna-se possível analisar dados como o parâmetro de inflação e preço por hora para cada um dos parques presentes na simulação.

4 Experiências e Objetivos

4.1 Experiência 1: Preços Dinâmicos vs. Preços Estáticos

Objetivo

A experiência 1 pretende estudar o efeito do esquema de preços dinâmico no lucro atingido pelo parque e no bem-estar dos condutores. Assume-se que os parques fazem parte de uma mesma cadeia de parques de estacionamento e, que por isso, possuem o mesmo dono.

Esta experiência foi testada duas vezes durante 35 semanas, para ambos os tipos de parques de estacionamento.

Os *Static Parks* seguem um esquema de preços estático, representado na Tabela 2. Os *Dynamic Parks* utilizam um esquema de preços dinâmico e dependente da procura. O preço do parque é atualizado segundo as estratégias da Secção 2.1.2.

Para cada teste foi usado o mesmo valor da *seed*, forçando a criação dos mesmos agentes, com a mesma localização inicial, preferências e destino. Desta forma é possível testar exclusivamente o efeito do esquema de preços dinâmico e estático no lucro e felicidade dos condutores.

Resultado

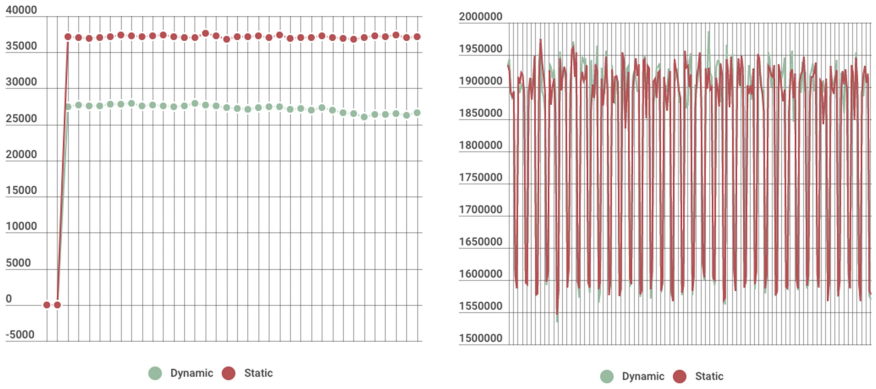

Figura 8: Gráfico de lucro e Gráfico da utilidade ao longo do tempo

A partir dos resultados é possível concluir que o esquema de preços dinâmicos não aumenta o lucro gerado pelo parque de estacionamento. Pelo contrário, um esquema de preços estático aumenta a receita em 10 000 € quando comparado ao lucro gerado pelos parques com tarifa dinâmica.

Quanto ao valor da utilidade dos condutores, pode-se reparar que não há diferenças significativas uma vez que não houve grande mudança a nível de preços a pagar pelos condutores.

Ao analisar a média de ocupação dos parques, concluiu-se que a maioria dos parques possui taxas de ocupação muito baixas, que poderá ter grande impacto nos resultados apresentados acima.

```
--------  New Prices  ----------
Park      |  1   |  2   |  3   |  4   |  5   |  6   |  7   |  8   |  9   |  10  |  11  |  12  |  13 |
Inflation     | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
Price per hour| 1.43€ | 1.43€ | 2.22€ | 2.22€ | 2.73€ | 1.43€ | 2.73€ | 2.22€ | 1.83€ | 2.73€ | 2.73€ | 2.22€ | 1.89€ |

Week 36
Day of the week: Monday
Number of Explorer Drives: 189
Number of Guided Drives: 121

Park   |  1   |  2   |  3   |  4   |  5   |  6   |  7   |  8   |  9   |  10  |  11  |  12  |  13 |
9:00h  | 2.30% | 4.73% | 3.56% | 2.85% | 1.47% | 6.27% | 0.46% | 12.12% | 1.19% | 2.00% | 1.76% | 3.40% | 12.31% |
16:00h | 1.15% | 2.44% | 0.67% | 1.14% | 0.74% | 3.30% | 0.18% | 4.38% | 1.19% | 1.71% | 0.65% | 2.20% | 9.23% |
20:00h | 0.66% | 1.72% | 0.89% | 0.28% | 0.74% | 3.30% | 0.28% | 3.70% | 0.90% | 0.57% | 0.37% | 1.60% | 10.77% |

Drivers Average Utility: 7332.308340009214
```

Figura 9: Ocupação média dos parques

4.2 Experiência 2: Simulação de grande afluência de condutores na cidade

Objetivo

A experiência 2 simula um evento na cidade, o que aumentaria a afluência de condutores e, pretende estudar o efeito desta mudança no lucro e no bem-estar dos condutores.

Esta experiência foi testada três vezes durante 35 semanas, para ambos os tipos de parques de estacionamento.

Para cada teste foi usado o mesmo valor da *seed*, forçando a criação dos mesmos agentes, com a mesma localização inicial, preferências e destino. Desta forma é possível testar exclusivamente o efeito do aumento de condutores na cidade.

Resultado

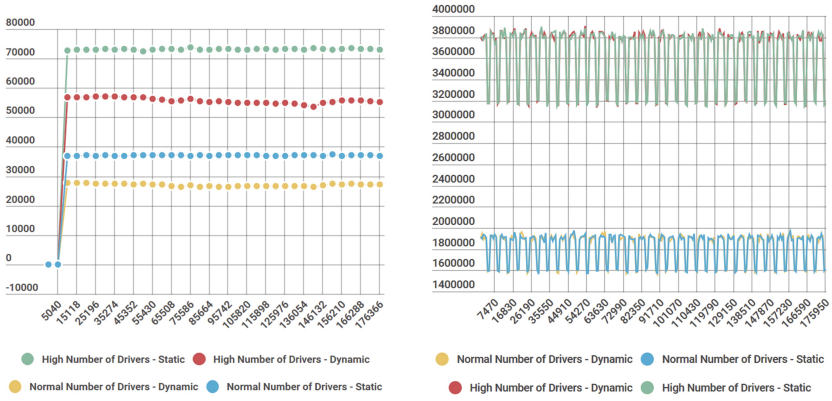

Figura 10: Gráfico de lucro e Gráfico da utilidade ao longo do tempo

A simulação com um maior número de condutores não teve uma visível alteração na utilidade do sistema, podendo concluir que uma maior afluência na cidade não afeta significativamente o bem-estar da sociedade em causa. Isto deve-se maioritariamente ao facto da população a ser simulada não criar uma sobrelotação dos parques, permitindo que estes mantenham os seus preços baixos.

No que toca à receita semanal, é possível verificar que o preço estático oferece um maior lucro à empresa uma vez que pela equação de adaptação do esquema de preço dinâmico, uma menor utilização do parque leva à redução da tarifa a pagar. Como tal, os preços dos parques que utilizam uma tarifa dinâmica são menores que os que taxam de maneira estático, levando a uma menor receita no final da semana.

4.3 Experiência 3: Diminuição da Capacidade dos Parques

Objetivo

A experiência 3 testa o efeito da ocupação máxima dos parques de estacionamento no lucro. Após a análise da experiência 2, concluiu-se que o esquema de preços dinâmico não gera um aumento significativo no lucro. Isto deve-se ao facto de os parques se encontrarem, na sua maioria, desocupados.

Nesta experiência, diminui-se a capacidade dos parques de estacionamento para o que, consecutivamente, leva a um aumento da ocupação média.

Esta experiência foi testada durante 35 semanas, para ambos os tipos de parques de estacionamento.

Para cada teste foi usado o mesmo valor da *seed*, forçando a criação dos mesmos agentes, com a mesma localização inicial, preferências e destino. Desta forma é possível testar exclusivamente o efeito do aumento de condutores na cidade.

Resultado

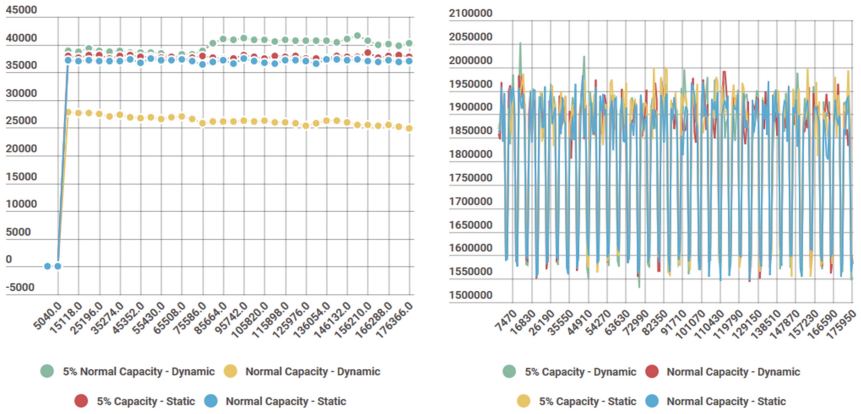

Figura 11: Gráfico de lucro e Gráfico da utilidade ao longo do tempo

A simulação com 5% da capacidade inicial do parque não teve grandes alterações na utilidade tirando alguns picos, alturas onde o parque devia estar lotado. Relativamente ao lucro o parque com uma capacidade menor e preços dinâmicos lucra mais do que com preços estáticos.

A partir do gráfico reparamos que o parque com preços dinâmicos lucrou mais 3 mil €comparativamente ao parque que seguia um esquema de preços estático. Isto deve-se ao facto de uma taxa de ocupação alta conduzir a um ajuste de preços e, neste caso, a um aumento.

Conclui-se, também, que o lucro aumenta mais a partir da 15º semana, devido ao facto de terminar o primeiro ciclo de atualização dos parâmetros, tornando assim o parque mais lucrativo.

5 Conclusões

As experiências 1 e 2 permitem verificar que a tarifa dinâmica não é lucrativa quando existe uma baixa utilização dos recursos a serem disponibilizados e não afeta significativamente o bem-estar social. Neste caso, uma empresa que pretenda maximizar o lucro mesmo quando a sua utilização é baixa deve tentar avaliar qual é a causa dessa baixa utilização. Caso o número de utilizadores na cidade seja reduzido, a melhor abordagem passará por colocar em prática uma tarifa estática ou alterando a fórmula sugerida.

Observando a experiência 3 é possível concluir que uma maior utilização do parque aumenta os lucros da empresa que gere, não alterando significativamente o bem-estar social. Esta experiência mostra que a fórmula utilizada é lucrativa quando utilizada num sistema onde existe uma grande procura pelos recursos disponibilizados.

A utilização de um sistema multi-agente permite simplificar a estrutura da solução, enquanto oferece uma grande flexibilidade de extensão. A separação de conceitos que a arquitetura oferece melhora significativamente a velocidade de desenvolvimento, enquanto o sistema se equipara à estrutura de simulação do cenário.

6 Melhoramentos

Alguns melhoramentos poderiam ser aplicados futuramente ao projeto, tais como a adição de variáveis de ambiente, por exemplo, o tráfego na estrada ou a meteorologia, que influenciariam o número de condutores a estacionar num determinado parque e, por consequente, a utilidade dos condutores.

Futuramente poderia ser criada uma nova função de utilidade mais eficiente e mais realista.

Com mais tempo, seria possível implementar um CSV *Parser* que inicializasse dinamicamente, na simulação, todos os parques com as suas características respetivas.

Além disso, teria sido pertinente a realização de uma experiência com parques independentes, ou seja, não pertencentes à mesma cadeia, e, analisar, também, a competitividade entre eles.

Por fim, seria oportuno um progresso a nível da *GUI* da simulação, usando, por base, o projeto "Repast City"que simula uma cidade e as suas ruas *(https://github.com/nickmalleson/repastcity/tree/master/repastcity3)*.

7 Recursos

7.1 Bibliografia

1. Thomas Vrancken Daniel Tenbrock Sebastian Reick Dejan Bozhinovski Gerhard Weiss Gerasimos Spanakis (2017): Multi-Agent Parking Place Simulation, in Advances in Practical Applications of Cyber-Physical Multi-Agent Systems: The PAAMS Collection: 15th International Conference, PAAMS 2017, Porto, Portugal, June 21-23, 2017, Proceedings, Y. Demazeau, et al., Editors. 2017, Springer International Publishing: Cham. p. 272-283.

2. Jade Documentation - http://jade.tilab.com/doc/tutorials/JADEProgramming-Tutorial-for-beginners.pdf

3. Repast Simphony Documentation - https://repast.github.io/docs/RepastJavaGettingStarted.pdf

4. SAJaS Documentation - https://web.fe.up.pt/ hlc/doku.php?id=sajas

7.2 Software

O projeto foi desenvolvido através do IDE *Eclipse*, com acesso às *frameworks JADE*, *REPAST* e *SAJaS*.

7.3 Contribuição

O trabalho foi realizado pelo grupo segundo as seguintes percentagens:

- Bernardo Belchior - 33.3%
- Maria João Mira Paulo - 33.3%
- Nuno Ramos - 33.3%

8 Apêndice

8.1 Manual do utilizador

Para executar este projeto deve-se importar, no *Eclipse*, e com a perspetiva *ReLogo* instalada, o projeto como um *Repast Simphony project*.

Após a importação do projeto, executa-se o modelo. Surgirá a *GUI* nativa do *Repast Simphony*, que contém uma série de parâmetros e opções de execução da simulação.

É possível pausar ou parar a simulação num determinado *tick* e, além disso, existe a possibilidade de diminuição da velocidade dos *ticks*, tornando mais fácil a visualização da simulação.

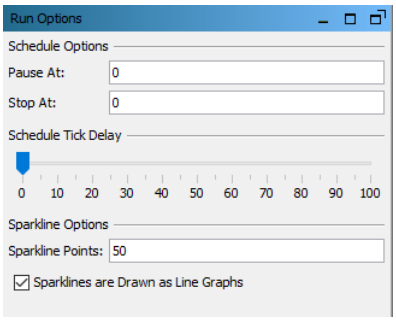

Figura 12: *Run Options*

Por fim, devem-se utilizar os botões do lado superior esquerdo da *GUI* para inicializar a simulação.

Figura 13: *Botões para executar a simulação*

O primeiro botão serve para inicializar a simulação, o segundo para começar a executar e o terceiro serve para executar *step by step*.