SISTEMAS PERICIAIS E ROBÓTICA

RELATÓRIO FINAL

SISTEMA MULI-AGENTE SIMULANDO O TRÁFEGO DE UMA CIDADE

António Manuel da Costa Delgado
(ei97019@fe.up.pt)
Vânia Guiomar da Silva Gonçalves
(ei97011@fe.up.pt)

OBJECTIVOS

Este projecto visa desenvolver uma interface que permita a simulação de trânsito numa cidade baseada num sistema multi-agente.

Os agentes são independentes e representam um conjunto de semáforos colocados num cruzamento.

Pretende-se com este projecto comparar três abordagens:

- semáforos de tempo fixo, em que ao fim de um certo tempo, definido pelo utilizador, o semáforo muda de luz;
- agentes reactivos, em que o gestor de semáforos reage a sensores, tais como, número de veículos que passaram, número de veículos chegados ao cruzamento num certo intervalo de tempo, entre outros;
- sistemas multi-agente, em que os vários gestores de semáforos comunicam entre si trocando informações e sugestões.

Com estas abordagens pretende-se realizar testes sobre os tempos de espera máximo e mínimo e sobre a razão do número de veículos saídos sobre os entrados no sistema.

MOTIVAÇÃO

A simulação que foi implementada visa, essencialmente, avaliar o comportamento de agentes reactivos e de sistemas multi-agente.

Consideramos que este projecto pode abranger as duas perspectivas: académica e não académica.

Em termos académicos este projecto expande os nossos conhecimentos de Inteligência Artificial, em particular uma das suas importantes abordagens: Agentes.

No entanto esta concepção poderá ter interesse não académico, pois poderá ser a base de um projecto mais elaborado a ser utilizado por uma autarquia que pretenda melhorar o controlo de tráfego numa cidade.

Como todos sabemos, o trânsito (e em particular os engarrafamentos) é um dos grandes problemas das grandes cidades. Temos também conhecimento que o estudo de formas para melhorar o trânsito nas cidades tem recorrido a simulações, em particular com o uso de Agentes.

- DESCRIÇÃO

FUNCIONALIDADES

Ao utilizador é oferecida a possibilidade de construir as estradas de forma a simular uma cidade. Através de intercepção de duas estradas pode ser criado um cruzamento. Para além do mais, existe ainda a possíbilidade de serem criadas curvas, sem semáforos.

É possível escolher o número de viaturas que se pretende que circulem na cidade.

Antes de iniciar a simulação é possível escolher qual das abordagens se pretende analisar, nomeadamente: semáforos de tempo fixo (com possibilidade de definir o número de segundos entre transições de luz), agentes reactivos e multi-agentes.

É possível interromper a simulação, a qualquer momento, e posteriormente analisar diversas estatísticas.

ESTRUTURA DO PROGRAMA

O presente projecto encontra-se dividido em módulos. Para facilitar a sua análise representam-se de seguida, em forma de esquema. As setas significam transferência de informação.

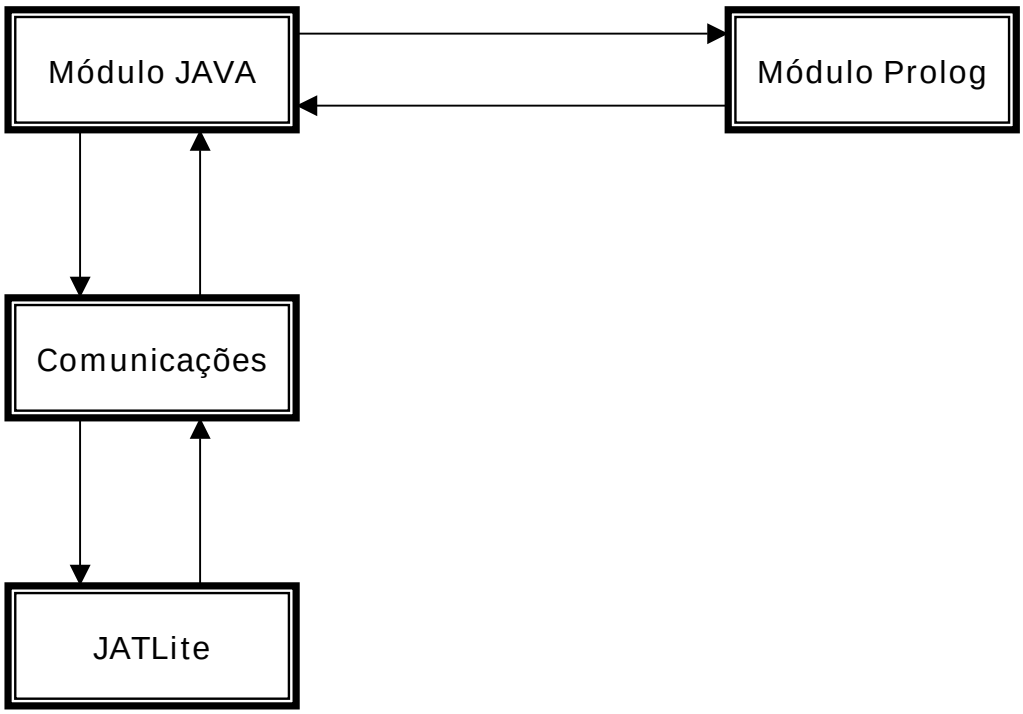

Fig. 1 – Representação esquemática dos módulos

ESQUEMAS DE REPRESENTAÇÃO DE CONHECIMENTO

As regras de decisão vão constituir o módulo do Prolog. Para tal, cada carro está conectado a um `LogicServer`. Deste modo, é possível definir quais as atitudes a tomar, por parte de cada carro, em cada momento.

Em contrapartida, cada semáforo regista as suas alterações em todos os `LogicServer`. Esta informação encontra-se registada no esquema que se segue.

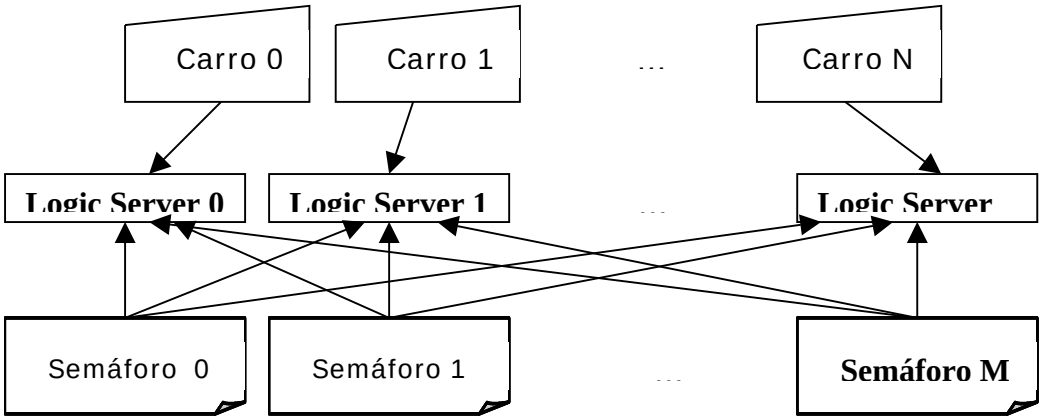

Fig. 2 – Esquema de representação do conhecimento

Dentro de cada `LogicServer` são tomadas diversas decisões que vão influenciar as acções de cada carro presente na simulação. Estas acções passam por ordenar a entrada do carro na cidade, movimentar-se numa das quatro direcções possíveis ou a chegada a um cruzamento. Esta última acção pode ainda compreender várias atitudes: o carro pára, se estiver vermelho; escolhe a nova estrada e avança para ela, se estiver verde.

As vantagens da representação do conhecimento através de um módulo Prolog são notórias. Esta é a linguagem utilizada nas grandes simulações e tem um funcionamento muito característico, que se distingue das restantes linguagens de programação. Daí foi decidido que o módulo referente à representação do conhecimento seria implementado em Prolog.

IMPLEMENTAÇÃO

Ao longo do desenvolvimento deste projecto foi tida grande preocupação em comentar correctamente todo o código produzido. No entanto, e com o intuito de um esclarecimento adicional, é apresentada de seguida uma descrição mais pormenorizada de todas as classes JAVA e código Prolog escritos.

- **TrafficSim.java**

A classe `TrafficSim` define os componentes da interface gráfica. É no método `initComponents` que são definidos os componentes do painel da esquerda, tais como, botões, *radio buttons* e caixas de texto. Do lado direito, está presente um *canvas* que permite a edição do

mapa da cidade. Este *canvas* vai permitir visualizar a simulação e quando for dada a ordem de terminar é também, o local onde são apresentadas as estatísticas.

Nesta classe são também definidos os eventos para os botões que permitem construir estradas. O método `mouse_click()` é o método que processa os eventos originados no *canvas*.

Além disso iniciliza o *thread* principal que vai supervisionar todas as operações.

- **Cidade.java**

Nesta classe são definidas as estruturas que guardam a informação da localização dos semáforos e dos diversos tipos de estrada (horizontal, vertical, curvas). O *canvas* está dividido em *macro pixels* para facilitar a edição das estradas. O construtor desta classe inicializa a matriz `cityArray` que guarda a estrutura que existe em cada macro-pixel. Esta matriz tem uma dimensão igual à da grelha existente para edição da cidade. Adicionalmente é criada uma matriz de caracteres, `estrutura`, onde fica registada a informação relativa ao tipo de estrutura que existe em cada ponto da cidade. Esta matriz é preenchida com o caractere `v` que significa que inicialmente a cidade se encontra vazia.

O método `paint()` faz o *refresh* do *canvas* percorrendo esta matriz e redesenhando toda a cidade.

Nesta classe estão registadas todas as estruturas que compõem a cidade: estradas, semáforos e carros. Existem *array*'s que registam as estruturas presentes na cidade. Para cada tipo de estrutura existe informação pormenorizada, quer relativa ao número máximo que é possível de existir na cidade, quer ao número que, no momento presente, existe.

O método `aprEstatisticas()` é chamado quando o utilizador dá ordem para terminar a simulação. Este método desenha um rectângulo por cima da grelha que representava a cidade. Neste rectângulo vão ser apresentadas as diversas estatísticas relativas à evolução da simulação.

Os métodos `ConvertToMacroCoordinate()` e `ConvertFromMacroCoordinate()` convertem, respectivamente, coordenadas em *pixels* para *macro pixels* e *macro pixels* para coordenadas em *pixels*.

O método `insereEstrutura(int, int, int, int)` permite inserir uma estrada cujas coordenadas de início e fim são passadas como argumento. Este método altera a matriz, que representa as estruturas em cada macro-pixel, inserindo nos pontos respectivos o caracter `e` (estrada).

O método detQuadrante() é utilizado para determinar o quadrante de uma curva. Para melhor ilustrar o conceito de quadrante é apresentada de sequida uma imagem.

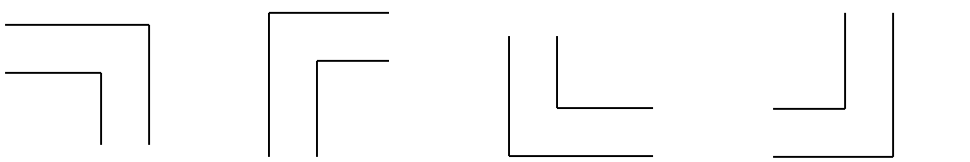

Fig.3 Quadrantes das Curvas

O método `divideEstrada(int, int)` é utilizado quando se pretende uma divisão de uma estrada devido à criação de um cruzamento.

O método `podeInserirEstrutura(int, int, int, int)` permite verificar se é possível ou não inserir uma estrada nas coordenadas apresentadas como argumento.

Por vezes é necessário determinar se existe alguma estrada que inicie ou termine num determinado ponto. Para tal, foi criado o método `pontoTerminal(int, int)` que recebe como argumentos as coordenadas de um dado ponto e retorna verdadeiro ou falso, caso exista definida na cidade uma estrada com início ou fim neste ponto. No entanto, existem situações em que não é indiferente saber se exise alguma estrada que inicie ou termine num dado ponto. Por vezes é exigido mais pormenor. Assim, estão implementados os métodos `estradaComInicioEm(int, int)` e `estradaComFimEm(int, int)` que retornam a estrada que tem o início ou fim, respectivamente, no ponto passado como argumento. Se existir um cruzamento, então existem duas estradas que terminam (e duas que se iniciam) no mesmo ponto. No entanto uma delas é horizontal e a outra é vertical. Para tratar estas situações foram definidos mais dois métodos: `estradaComInicioEm (int, int, boolean)` e `estradaComFimEm(int, int, boolean)`. Nestes métodos o último argumento indica se pretendemos uma estrada horizontal (*true*) ou vertical (*false*).

Para verificar se existe algum semáforo num determinado ponto foi definido o método `existeSemaforo(int, int)`.

Por fim surge o método `posicaoRelativa(int, int)` que determina a posição relativa de dois semáforos. Os argumentos são os identificadores de cada semáforo. Os valores retornados por esta função são os seguintes:

0 - o primeiro está à direita do segundo
1 - o primeiro está por baixo do segundo
2 - o primeiro está à esquerda do segundo
3 - o primeiro está em cima do segundo

Fig.4 Valores de retorno do método `posicaoRelativa(int, int)`

- ## Semaforo.java

Esta classe gere todas as possíveis operações que podem ocorrer num cruzamento controlado por semáforos.

A variável `tempo` é utilizada apenas quando o semáforo oferece tempo constante para todas as ruas. Para saber o modo de funcionamento do semáforo (tempo fixo, reactivo ou troca de mensagens com os vizinhos), opção escolhida pelo utilizador, existe a variável `funcionamento`. Além desta informação o semáforo sabe a `cidade` a que pertence, as estradas adjacentes (`estrada[]`), a sua localização (`posX` e `posY`) e o seu estado cromático (`luz[]`).

O construtor trata de inicializar as variáveis. De referir que `luz[0]` toma o valor *`true`*, o que significa que o semáforo zero está a verde. Todos os restantes semáforos do cruzamento estão vermelhos, ou seja, `luz[1]`, ..., `luz[NR_ESTRADAS-1]` tomam o valor *`false`*.

Os métodos `insereEstradas(Estrada, Estrada, Estrada, Estrada)` e `insereEstradas()` permitem adicionar ao *array* de estradas, aquelas que são adjacentes ao semáforo em questão.

O método `verde()` retorna um valor inteiro que indica qual o semáforo que se encontra verde num determinado momento.

O método `mudaLuz(int)` trata de alterar a indicação luminosa dos semáforos, ou seja, acende o verde no semáforo que é passado como argumento, após ter colocado todos os outros a vermelho.

O método `mudaLuz()` trata de acender o verde no semáforo seguinte. Este método pressupõem que os semáforos acendem verde de uma forma rotativa e sempre pela mesma ordem. Esta ordem coincide com o sentido dos ponteiros do relógio. Caso se pretenda acender um semáforo em particular, é necessário recorrer ao método `mudaLuz(int)` que recebe como argumento o identificador do novo semáforo a acender a luz verde.

Para saber quais os semáforos que são vizinhos a este semáforo, foi escrito o método `vizinhos()`. Este método retorna um *array* contendo os semáforos vizinhos.

Quando estamos perante uma simulação baseada na troca de mensagens entre os semáforos vizinhos é necessário recorrer ao método `processaMsg(int, String, String)`. Este método actualiza a base de dados deste semáforo de acordo com uma mensagem recebida. Recebe como argumentos a identificação da origem da mensagem, a mensagem recebida e o tipo da mensagem (pedido -> *ask* OU resposta -> *ans*). Caso a mensagem a processar seja do tipo *ask* este método cria uma nova mensagem do tipo *ans* que a vai fazer encaminhar até ao semáforo que lhe enviou o pedido. Se a mensagem recebida for do tipo *ans*, então é necessário processar a mensagem. Este processamento procede-se descodificando a mensagem recebida. Um exemplo de uma mensagem recebida é apresentado de seguida.

```
0-4 1-2 2-5 3-10
```

Fig.5 Exemplo de mensagem

Esta mensagem indica que pela estrada 0 poderão circular 4 carros; pela estrada 1, 2 carros; pela estrada 2, 5 carros e pela estrada 3, 10. Para uma melhor compreensão foi desenhado o esquema seguinte que mostra em que posição se encontram as quatro estradas à volta de um semáforo.

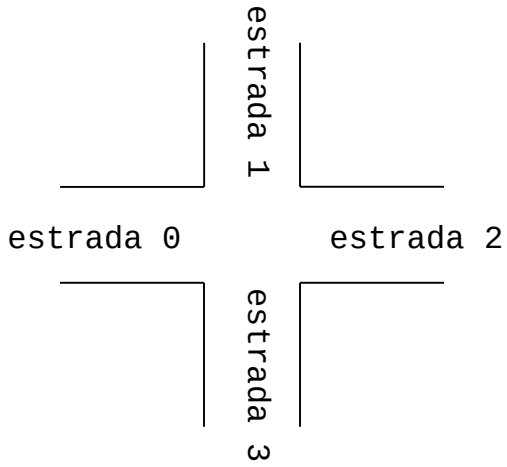

Fig.6 Posição das estradas à volta de um semáforo

O método `detSituacao()` cria uma *string* contendo a informação relativa à situação deste semáforo relativamente ao número de carros que a ele se dirigem. Esta *string* tem uma estrutura semelhante à apresentada na penúltima figura.

- **Estrada.java**

As variáveis globais desta classe registam coordenadas dos extremos da estrada, a orientação da estrada (vertical ou horizontal), o número de carros que circulam em cada direcção e a cidade a que pertence a estrada.

Tal como nas classes já analisadas, existe um construtor que inicializa as variáveis de classe, de acordo com a informação que surge nos argumentos recebidos.

O método `limite()` é utilizado para determinar se esta estrada tem início ou fim nos lados da cidade. Esta validação é necessária pois na cidade tem que existir, pelo menos uma estrada nestas condições para que os carros possam entrar.

Os métodos `entradaX()` e `entradaY()` retornam, respectivamente, a coordenada `x` e `y` de entrada na estrada. Adicionalmente foi definido `sentidoEntrada(int, int)` que permite determinar o sentido se entrarmos nesta estrada pelo lado da cidade.

O método `sentidoEstrada(int, int)` foi desenvolvido para que seja possível obter facilmente o sentido de uma estrada. Este sentido só tem significado quando a estrada se enconta ligada a um cruzamento. O seguinte esquema mostra como se encontra codificado o sentido de uma estrada. Como é possível verificar o sentido corresponde ao ponto cardeal de onde o carro provém.

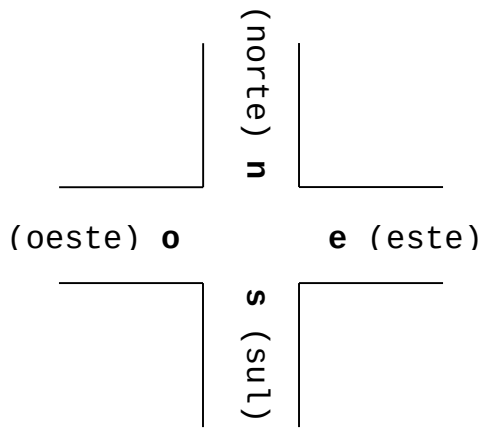

Fig.7 Sentido das estradas num cruzamento

O método `pertence(int, int)` verifica se determinado ponto pertence à estrada.

Para alterar a localização de uma estrada existe o método `alteraPosicao(int, int, int, int)` que recebe como argumentos as novas posições dos pontos inicial e final da estrada.

- **Carro.java**

As variáveis de classe são `INCREMENTO`, que indica o número de pixeis que um carro se desloca de cada vez; a `estrada` onde se encontra o carro; as suas coordenadas em pixeis; o `sentido` de deslocamento do carro; e variáveis estatísticas tais como `distPercorrida` que indica a distância que este carro já percorreu ao longo de toda a simulação, número de `inversoes` de marcha que o carro fez por ter chegado a uma rua sem saída e o `tempoEspera` que indica o tempo de espera em segundos num ao longo de todos os semáforos.

O construtor inicializa as variáveis de classe.

Para entrar na cidade é invocado o método `entraCidade()`. É feita a actualização das variáveis estatísticas e determinada a estrada a percorrer

Quando o carro chega a um cruzamento é necessário invocar `mudaEstrada(Estrada)`, que recebe como argumento a estrada seguinte. No entanto, quando chega ao fim de uma estrada sem saída é utilizado o método `inverteSentido()`.

Por vezes é necessário converter o *char* que representa o sentido do movimento, já apresentado na última figura, para inteiro. Assim, está justificada a existência do métido sentido().

A entrada numa nova estrada é feita com recurso à invocação do método `entraEstrada()`.

Quando o carro se desloca numa estrada, vertical ou horizontal, é necessário determinar a nova posição e actualizar as variáveis estatísticas. A competência destas tarefas fica a cargo dos métodos `moveX()` e `MoveY()`.

Ao chegar a um cruzamento é necessário determinar quais as estradas para onde o carro pode seguir. Para tal existe o método `estradasPossiveis()` que retorna um `Vector` contendo as estradas em questão. Para o preenchimento deste vector é determinado o sentido do movimento do carro. Seguidamente, e com recurso a métodos auxiliares de determinação de estradas com início e fim em determinado ponto, que serão explicadas em pormenor de seguida, é preenchido o vector.

Com muita frequência é necessário verificar em que estrada se encontra o carro. Por isso, esta classe possui os métodos `determinaEstrada(int, int)` e `determinaEstrada()`. Estes métodos percorrem todas as estradas da cidade, com o auxílio de um ciclo `for`, que termina quando o ponto onde se encontra o carro se situar dentro do "rectângulo" da estrada.

O método `estradaComInicioEm(int, int)` retorna uma `Estrada` cujo ponto inicial é o indicado nos argumentos. Para tal, o *array* que contém as estradas todas da cidade é percorrido até ser encontrada aquela cujo valor de `xIni` e `yIni` coincide com os valores passados como argumento.

O método `estradaComFimEm(int, int)` tem um funcionamento semelhante ao apresentado anteriormente. A diferença reside no ponto de comparação, que neste caso interessa que seja o ponto final da estrada.

`mudaEstrada(Estrada)` é invocado quando o carro passa a circular numa nova estrada por ter chegado a um cruzamento.

Por fim, surge o método `escolheEstradaSeguinte()` que retorna, aleatoriamente, uma estrada de entre todas as possíveis no momento, para o carro em questão.

- ## Curva.java

Esta classe é utilizada para definir as quatro variantes de curvas que já foram explicadas ao pormenor aquando da explicação da classe `Cidade`.

Existem três construtores diferentes para serem aplicados nas divesas situações que podem ocorrer. Um deles requere a cidade, e as coordenadas da curva. Neste caso o quadrante ainda não foi determinado e é considerado como sendo –1. O segundo construtor já pede adicionalmente o quadrante, ficando desde já determinado. Por último, está disponível um construtor que recebe como argumentos as duas estradas que convergem na curva. Neste caso o quadrante é determinado automaticamente, de acordo com a informação que as estradas têm associada.

Para esta classe não se justificou a criação de métodos.

- **ThreadInterface.java**

Esta classe gere todos os *thread*'s que estão a correr na simulação. As variáveis de classe são `cidade` (tipo `Cidade`), `Cthread` (tipo `CarroThread[]`), `Sthread` (tipo `SemThread[]`) e `ls` (tipo `LogicServer[]`). Devido ao facto de a distribuição do Amzi! ser apenas de avaliação este apresenta uma janela (que fecha automaticamente de seguida) na primeira vez que é chamado o Prolog. Por este facto consideramos mais realista que esta janela fosse apresentada no momento em que o programa inicia e não quando se ordena o início da simulação. Assim, existe, adicionalmente a variável `dummy` do tipo `LogicServer`. Da inicialização faz parte também o preeenchimento do *array* `ls` com novos objectos do tipo `LogicServer`.

O método `actualizaLS()` possibilita o *assert* das estradas, curvas, semáforos e carros no módulo *prolog*.

O método `connect()` liga os semáforos e põe os carros em movimento. Caso o funcionamento escolhido pelo utilizador for por mensagens, adicionalmente, cria os mensageiros. O funcionamento destes mensageiros serão explicados com todo o pormenor mais à frente, neste relatório.

Quando se pretende terminar a simulação é necessário invocar o método `shutdown()` que pára os carros e desliga os semáforos através da chamada de `terminate()`. Mais uma vez, apenas se estivermos perante uma simulação do tipo transmissão de mensagens, os mensageiros são desligados.

- **SemThread.java**

Esta classe é uma sub-classe de `Thread`. O seu papel é gerir todas as tarefas de um cruzamento / conjunto de semáforos.

O método `terminate()` desliga o semáforo alterando o valor lógico da *flag* `stop`.

Em contrapartida o método `run()` é utilizado para o processamento de acções de um conjunto de semáforos. Para que não surjam problemas de sincronização na actualização da base de conhecimento Prolog, este método teve que ser definido como sendo *synchronized*.

Em primeiro lugar é verificado o tipo de funcionamento do semáforo (fixo, reactivo ou troca de mensagens). No processamento do tipo fixo é invocado o método `mudaLuz()` definido na classe `Semaforo`. De seguida é actualizado toda a base de conhecimento em Prolog através da invocação dos métodos `RetractStr` e `AssertaStr`. Neste momento, é então possível actualizar as luzes dos semáforos na interface gráfica. De acordo com a variável `tempo` da classe `Semaforo` é efectuada uma pausa. Isto porque estamos perante um conjunto de semáforos de temporização fixa. Finda esta pausa, todo o processo referido se repete.

Num processamento do tipo reactivo o processo difere na medida em que o semáforo tem a visão das quatro estradas que para ele convergem. Assim, o semáforo abre o verde naquela estrada que tem mais carros.

Se passarmos para o processamento do tipo transmissão de mensagens a situação torna-se um pouco mais complexa. A solução implementada pode ser mais facilmente percebida se for analisada o esquema seguinte.

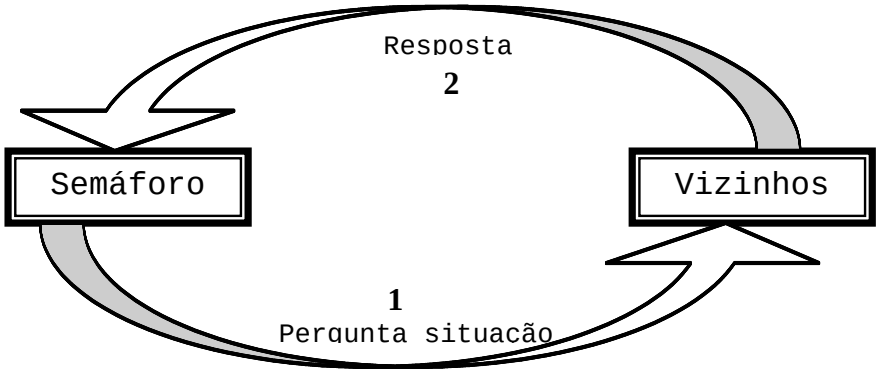

Fig.8 Funcionamento da passagem de mensagens

Quando obtém a resposta de todos os seus vizinhos, o semáforo que efectuou o pedido, tem que decidir qual das estradas vai ser contemplada com o sinal verde. Como pretendemos um agente independente, ele não se pode basear cegamente na informação que os seus vizinhos lhe facultam. Por isso tem ele próprio que efectuar a sua análise. Na figura seguinte, encontram-se representados nas setas os valores fornecidos pelos quatro semáforos vizinhos. São valores que os semáforos vizinhos prevêm que circulem para essa estrada. Nos círculos figuram o número de carros que o semáforo consegue ver pois estão a circular nas estradas adjacentes. Apesar de o nosso vizinho de ESTE apresentar um valor de 13 carros, o nosso semáforo vai abrir verde para a estrada de SUL pois é aquela que apresenta uma soma maior.

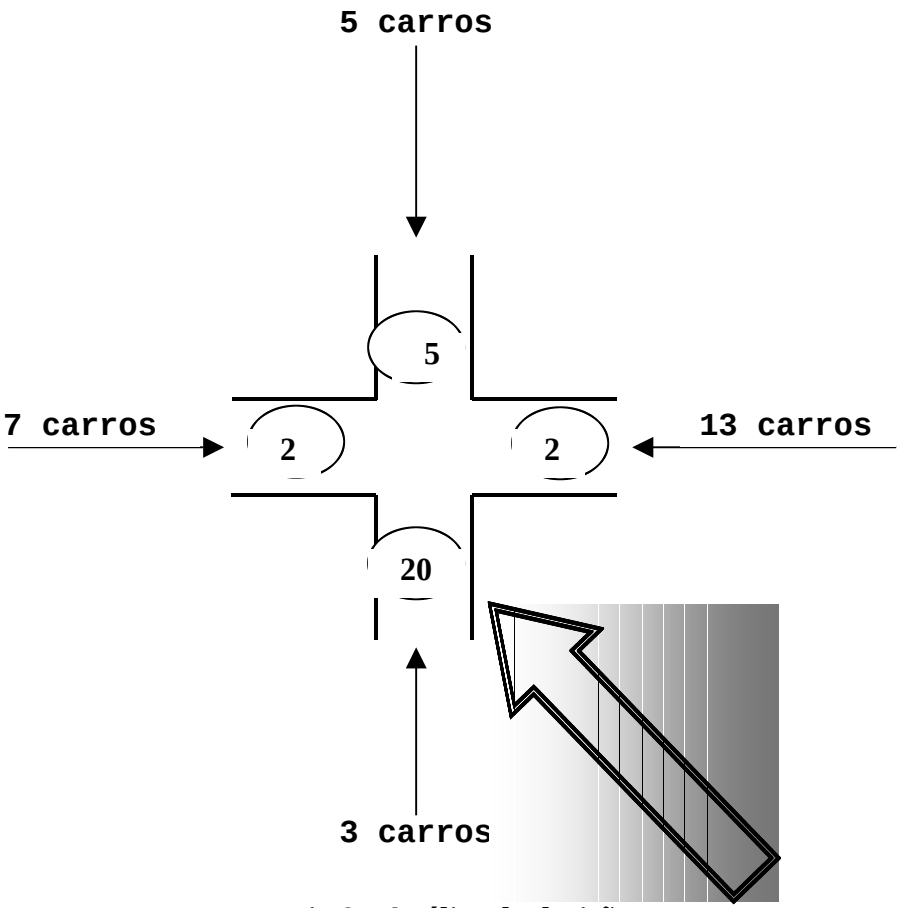

Fig.9 - Análise de decisão

Como apoio para a realização deste algoritmo foi desenvolvido o método `maximo(int, int, int, int)` que determina qual o semáforo a abrir de acordo com a indicação do número de carros que poderão convergir para cada uma das 4 estradas do cruzamento

- **CarroThread.java**

Esta classe também é uma sub-classe de `Thread`. Tem como função o controlo dos movimentos de um carro ao longo das ruas da cidade.

O método `run()` possui um ciclo *while* que só termina quando é invocado o método `terminate()`.

Este ciclo inicia por consultar o Prolog com o intuito de determinar qual a próxima acção que o carro vai tomar. Este procedimento recorre aos métodos `ExecStr` e `GetStrArg` que surgem em `amzi.ls`. Esta consulta tem obrigatoriamente que ser *synchronized* para evitar conflitos entre os vários *thread*'s a correr.

De seguida surge uma sequência de instruções *if – else if* . Dependendo da acção ordenada pelo Prolog são executados os procedimentos respectivos. As acções possíveis são: `entraCidade`, `curva1`, `curva2`, `curva3`, `curva4`, `direita`, `esquerda`, `cima`, `baixo`, `inverte` e `Cruzamento`. Estas acções têm explicação intuitiva, no entanto, encontram-se devidamente comentadas no código que surge em anexo.

Quando estamos perante um cruzamento, é necessário determinar aleatoriamente qual a estrada a seguir. Para tal existe o método `escolheEstradaSeguinte(String)` que recebe a *string* retornada pelo Prolog que corresponde a uma lista de estradas possíveis. O método tem que descartar essa string e escolher a próxima estrada pela qual o carro vai circular.

Por fim é actualizado o Prolog de acordo com a nova posição do carro.

É feita uma pequena pausa e todo o processo se repete até a simulação terminar.

- **Mensageiro.java**

Esta classe é utilizada apenas no caso de o utilizador escolher a opção de agentes comunicativos.

Estão disponíveis dois construtores. Um deles utiliza os valores por omissão da super-classe `KQMLAgentAction` que está implementada no JATLite. O outro construtor permite personalizar as variáveis. Além disso disponibiliza ainda uma matriz de inteiros que permite guardar a situação dos semáforos vizinhos, situação essa quantificada em número de carros a circular nas ruas vizinhas.

Após a criação da tabela de endereços é preenchida a tabela de ligações e criada a fila de mensagens onde serão guardadas todas as transmissões entre os agentes.

O método `processMessage(String, Object)` é uma extensão de outro com o mesmo nome e permite processar uma mensagem de acordo como o seu tipo.

O método `Act(Object)` obtem o objecto existente em `MessageQueue` e converte-o para um objecto KQML. Depois executa a acção apropriada de acordo com o comando recebido.

Para o envio de mensagens foi desenvolvido o método `enviaMsg(int, String, String, String)` que envia uma mensagem para um semáforo.

- **Transito.pro**

Este módulo é a parte central de todo o processamento. É aqui que todas as decisões relativas ao movimento dos carros são tomadas.

A representação dos factos em Prolog é feita como a seguir se exemplifica.

```
semaforo(nrSem, x, y, verde).
estrada(nrEstrada, x0, y0, x1, y1).
carro(nrCarro, x, y, sentido).
curva(nrCurva, x, y, quadrante).
```

Todo o código encontra-se devidamente comentado e assim dispensa qualquer explicação adicional.

A estrutura dos diversos predicados é semelhante e facilmente assimilável. O programa tenta sucessivamente efectuar determinadas acções até conseguir uma delas.

AMBIENTE DE DESENVOLVIMENTO

O presente projecto foi elaborado tendo por base o sistema operativo Windows NT e Windows 98.

Como ferramentas foram utilizadas as que se apresentam de seguida.

- Amzi! Prolog + Logic Server 5.0
  Interactive Development Environment
  http://www.amzi.com

- Java™ 2 SDK, Standard Edition
  Version 1.2.2
  http://java.sun.com

- JATLite
  http://java.stanford.edu

AVALIAÇÃO DO PROGRAMA

Num projecto desta natureza interessa verificar se realmente os agentes cooperam na tarefa que é comum. Se existe algum agente que em algum momento *rema contra a maré*, o projecto não terá a eficácia pretendida. No entanto, todos eles devem ser independentes, apesar de poderem comunicar com outros. Isto significa que caso determinado agente (conjunto de semáforos) receba uma mensagem indicando que se estão a dirigir muitos veículos por uma das ruas, ele só deve alternar para o sinal verde se considerar que o deve fazer. As mensagens recebidas servem apenas como meio de informação e não como ordens de atitude.

Outro ponto que deve ser valorizado é o interface permitir a observação da posição dos carros em movimento. Informação gráfica relativa ao estado dos semáforos deve também constituir ponto a valorizar.

Outro ponto relevante é o facto de como os resultados são mostrados. Uma interface amigável, com uma composição cromática suave e constituída por cores agradáveis é sempre bem-vinda.

Um algoritmo que ofereça um bom estudo estatístico seria o ideal. Melhor ainda seria aquele que permitisse personalizar essa apresentação para satisfazer os interesses de cada utilizador.

O realismo é também outro ponto a valorizar. Ou seja, numa cidade as ruas podem estar distribuídas das mais variadas formas: estradas horizontais, estradas verticais, cruzamentos, entroncamentos, curvas. Uma simulação que contemple a inserção destas diversas formas deverá ser valorizada.

RESULTADOS EXPERIMENTAIS

Foram realizadas diversas experiências, quer ao longo da elaboração do simulador, quer após a sua conclusão.

Vamos salientar um teste-modelo que foi realizado após todas as funcionalidades que nos propusemos terem sido implementadas. Este teste foi realizado com o objectivo de analisar os três tipos de funcionamento dos semáforos quanto à sua eficiência.

Assim, foram criadas três cidades idênticas (com as mesmas ruas, semáforos, curvas e carros) e foi dada ordem de iniciar a simulação. Cinco minutos depois foi terminada a simulação e obtidos os resultados apresentados nas figuras seguintes.

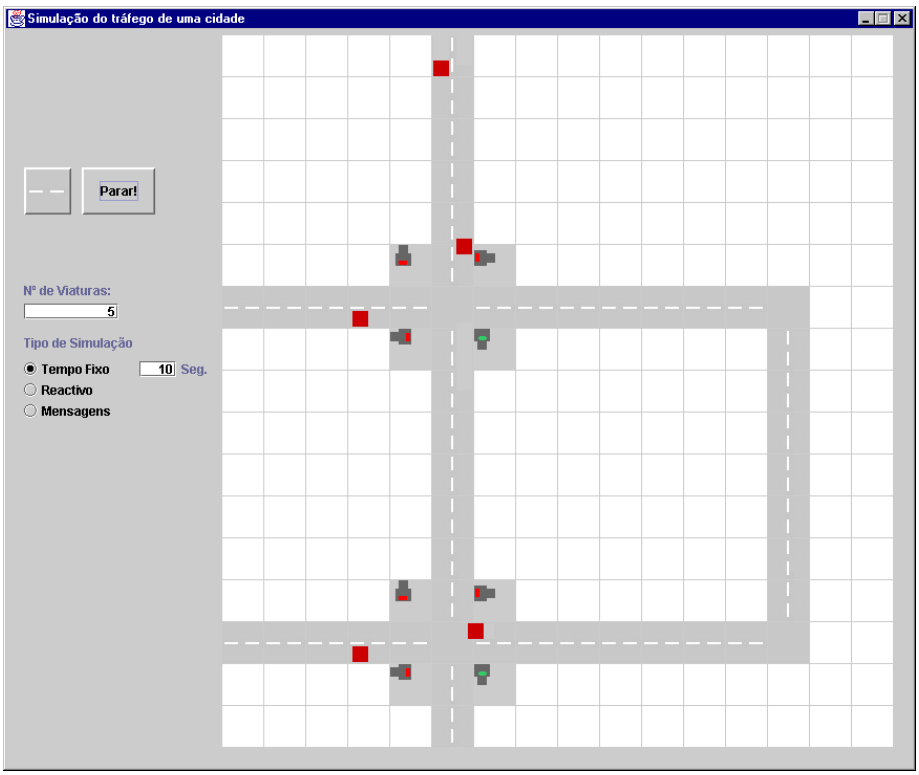

Fig.10 – Cidade exemplo

- Tempo fixo

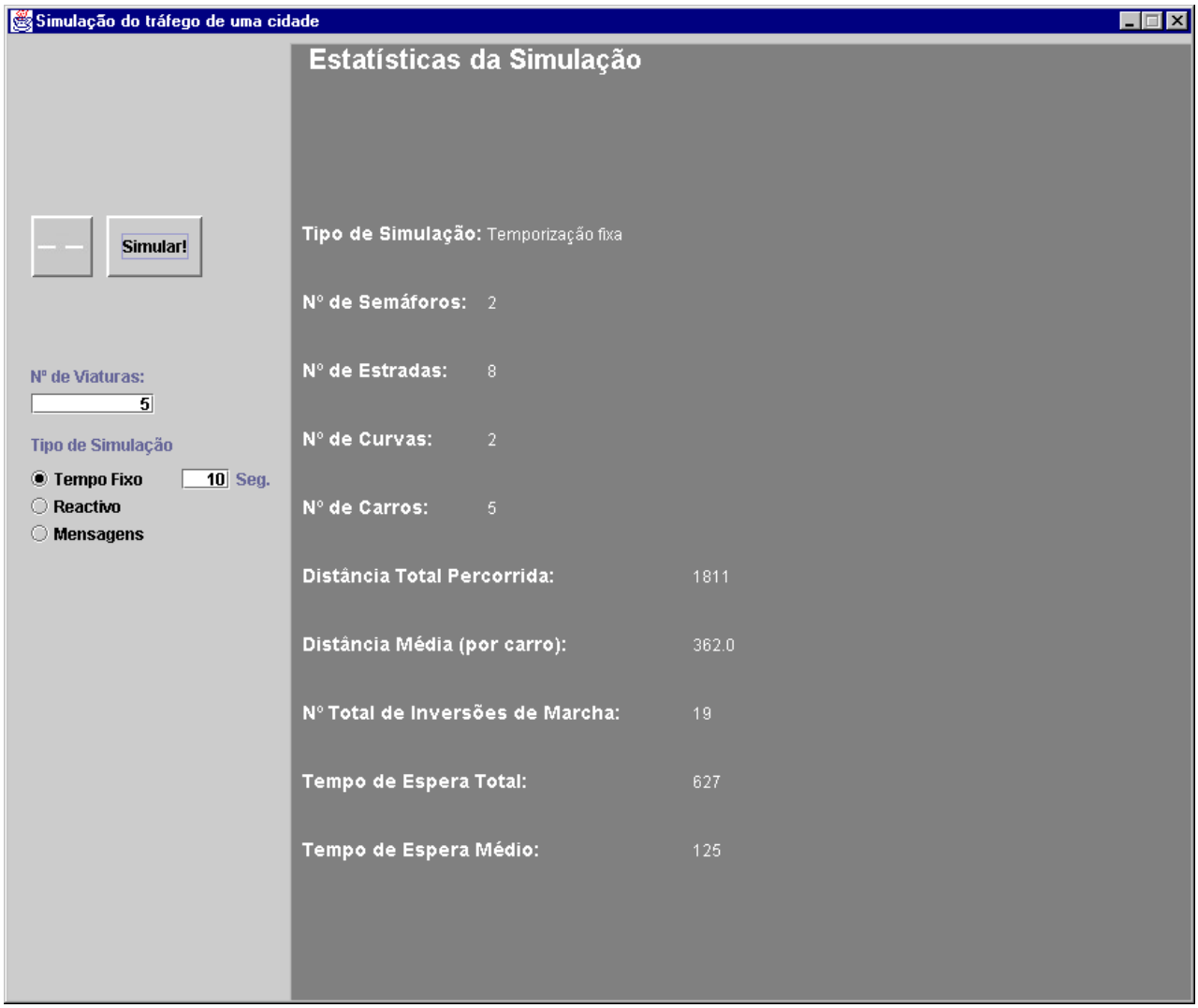

Fig.11 – Resultados da simulação com tempo fixo

- Reactivo

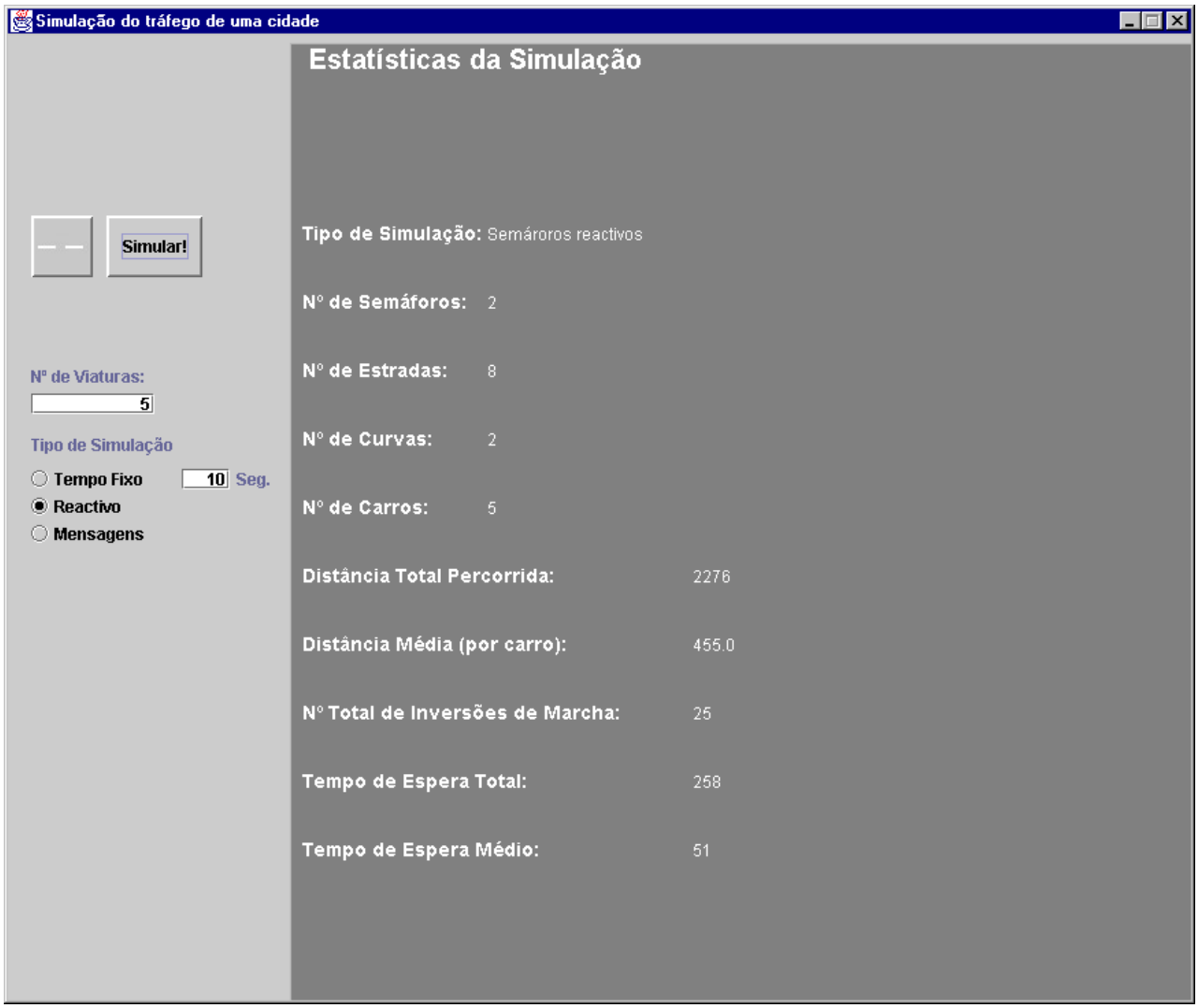

Fig.12 - Resultados da simulação com agentes reactivos

- Mensagens

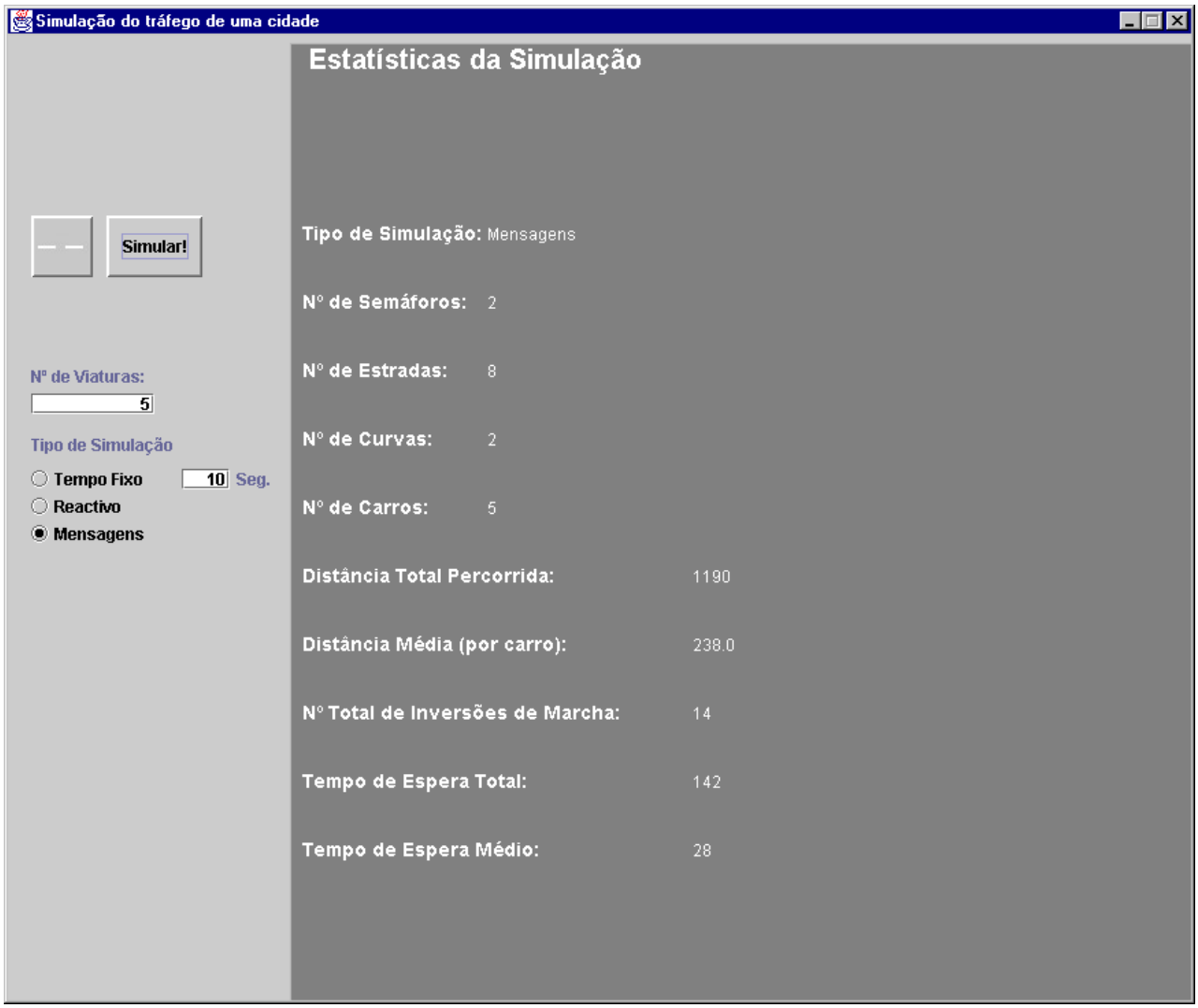

Fig.13 - Resultados da simulação com comunicação

MELHORAMENTOS

Caso tivéssemos tempo para tal, uma das funcionalidades que poderia ser implementada prende-se com o facto de criar estradas com prioridades diferentes. Como é sabido, na realidade existem ruas com maior prioridade do que outras. Outro ponto que poderia ter sido implementado, e está de certa forma relaccionado com as prioridades, é a criação de sinalização de trânsito. Este ponto possibilitaria a definição de estradas com um ou dois sentidos. De referir que na presente simulação existem apenas estradas com dois sentidos.

A existência de peões a cruzarem a via pública é uma realidade que também seria interessante implementar.

Outras estruturas existentes numa rede de estradas, tais como rotundas, viadutos, túneis ou pontes também poderiam ser tidas em conta caso tivéssemos tempo para tal.

Todos estes pontos ficam aqui registados com o intuito de ser implementados numa futura versão do simulador.

BIBLIOGRAFIA

Ao longo deste vasto projecto a equipa de desenvolvimento serviu-se de diversa documentação, sem a qual jamais poderiam ser atingidos os resultados que foram obtidos.

Segue-se uma lista da bibliografia utilizada.

- "Artificial Intelligence: A Modern Approach", Russel and Norvig, Prentice-Hall
- "Multiagent Systems: a modern approach to Distributed Artificial Intelligence"- Ed. G.Weiss, MIT Press, 1999
- Documentação do JATLite existente em http://java.stanford.edu
- Documentação do Amzi! disponível em http://www.amzi.com

CONCLUSÃO

Em termos conclusivos resta referir que estamos perante um projecto que utiliza agentes para apoiar uma tarefa real e evidentemente necessária no controlo do tráfego.

Importa salientar que a utilização de agentes como elemento chave na simulação é verdadeiramente útil. É a forma mais aproximada de como os semáforos na realidade funcionam, ou deveriam funcionar. Apesar de independentes na tomada de decisões, ter uma imagem do que está a acontecer na vizinhança, é importante.

Durante todo o presente projecto verificamos o quão importante pode ser um sistema como este para o controlo do tráfego de uma cidade. Todos sabemos a complexidade deste problema na nossa sociedade. Como tal, uma ferramenta de simulação como esta pode ajudar a detectar e resolver grande parte destes problemas.

Porto e Faculdade de Engenharia, 21 de Dezembro de 2000

APÊNDICES

MANUAL DO UTILIZADOR

Ao desenvolver este simulador foi tido em consideração, desde o início do projecto, a criação de uma interface intuitiva. Em termos gerais, consideramos este objectivo atingido. No entanto existem alguns pontos que consideramos necessário focar.

Criação de Estradas

Clique no ponto inicial e posterior clique no ponto final.

Criação de Cruzamentos

Um cruzamento resulta da intercepção de duas estradas perpendiculares. Assim, basta criar uma estrada e, de seguida, criar outra que a intercepte perpendicularmente.

Criação de curvas

Inicia-se por criar uma estrada. Seguidamente cria-se uma segunda estrada onde um dos seus pontos terminai coincide com um dos pontos terminais da estrada criada inicialmente.

Escolha do tipo de simulação

No painel existente no lado esquerdo da janela está presente um grupo de 3 botões de rádio. Basta seleccionar qual o tipo de simulação que pretendemos: tempo fixo, agentes reactivos ou mensagens. Se for escolhido tempo fixo, o valor existente na caixa de texto, que se encontra ao lado direito do grupo de botões de rádio, funciona como temporizador.

Início e fim da simulação

Existe um botão que alterna a sua indicação entre “Simular” e “Parar”.

LISTAGEM DO CÓDIGO

- **Transito.pro**

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% DETERMINA A ACÇÃO A TOMAR
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

accao(NrCarro, $curva1$) :-
	carro(NrCarro, PosX, PosY, _),
	curva(_, PosX, PosY, 1),
	!.

accao(NrCarro, $curva2$) :-
	carro(NrCarro, PosX, PosY, _),
	curva(_, PosX, PosY, 2),
	!.
accao(NrCarro, $curva3$) :-
	carro(NrCarro, PosX, PosY, _),
	curva(_, PosX, PosY, 3),
	!.
accao(NrCarro, $curva4$) :-
	carro(NrCarro, PosX, PosY, _),
	curva(_, PosX, PosY, 4),
	!.

	% chegou a um cruzamento e escolhe a nova estrada
accao(NrCarro, NovaEstrada) :-
	carro(NrCarro, PosX, PosY, o),
	estrada(NrEstrada, _, _, PosX, PosY),
	estradasPossiveis(NrEstrada, o, EstPoss),		% quais as estradas possíveis
neste cruzamento
	Xmais is PosX + 1,
	semaforo(NrSem, Xmais, PosY, _),
	formata([NrSem|EstPoss], EstradasStr),
	strcat($cruzamento$, EstradasStr, NovaEstrada),
	!.

	% chegou a um cruzamento e escolhe a nova estrada
accao(NrCarro, NovaEstrada) :-
	carro(NrCarro, PosX, PosY, e),
	estrada(NrEstrada, PosX, PosY, _, _),
	estradasPossiveis(NrEstrada, e, EstPoss),		% quais as estradas possíveis
neste cruzamento
	Xmenos is PosX -1,
	semaforo(NrSem, Xmenos, PosY, _),
	formata([NrSem| EstPoss], EstradasStr),
	strcat($cruzamento$, EstradasStr, NovaEstrada),
	!.

	% chegou a um cruzamento e escolhe a nova estrada
accao(NrCarro, NovaEstrada) :-
	carro(NrCarro, PosX, PosY, s),
	estrada(NrEstrada, PosX, PosY, _, _),
	estradasPossiveis(NrEstrada, s, EstPoss),		% quais as estradas possíveis
neste cruzamento
	Ymenos is PosY -1,
	semaforo(NrSem, PosX, Ymenos, _),
	formata([NrSem| EstPoss], EstradasStr),
	strcat($cruzamento$, EstradasStr, NovaEstrada),
	!.

	% chegou a um cruzamento e escolhe a nova estrada
```

```
accao(NrCarro, NovaEstrada) :-
	carro(NrCarro, PosX, PosY, n),
	estrada(NrEstrada, _, _, PosX, PosY),
	estradasPossiveis(NrEstrada, n, EstPoss),		% quais as estradas possíveis
neste cruzamento
	Ymais is PosY +1,
	semaforo(NrSem, PosX, Ymais, _),
	formata([NrSem| EstPoss], EstradasStr),
	strcat($cruzamento$, EstradasStr, NovaEstrada),
	!.

	% chegou a uma estrada sem saída: inverte o sentido
accao(NrCarro, $inverte$) :-
	carro(NrCarro, PosX, PosY, o),
	estrada(_,_,_,	PosX, PosY),
	!.

	% chegou a uma estrada sem saída: inverte o sentido
accao(NrCarro, $inverte$) :-
	carro(NrCarro, PosX, PosY, e),
	estrada(_, PosX, PosY, _, _),
	!.

	% chegou a uma estrada sem saída: inverte o sentido
accao(NrCarro, $inverte$) :-
	carro(NrCarro, PosX, PosY, n),
	estrada(_,_,_,	PosX, PosY),
	!.

	% chegou a uma estrada sem saída: inverte o sentido
accao(NrCarro, $inverte$) :-
	carro(NrCarro, PosX, PosY, s),
	estrada(_,PosX, PosY,_,_),
	!.

accao(NrCarro, $direita$) :-
	carro(NrCarro, _, _, o),
	!.

accao(NrCarro, $esquerda$) :-
	carro(NrCarro, _, _, e),
	!.

accao(NrCarro, $cima$) :-
	carro(NrCarro, _, _, s),
	!.

accao(NrCarro, $baixo$) :-
	carro(NrCarro, _, _, n),
	!.

	% o carro entra na cidade
accao(NrCarro, $entraCidade$) :-
	carro(NrCarro, -1, -1, _),
	!.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% PREDICADOS AUXILIARES
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


estradasPossiveis(EstradaActual, s, [E1, E2, E3]) :-
	estrada(EstradaActual, X0, Y0, _, _),
	Xmenos is X0-1,
	Xmais is X0+1,
```

```
	Ymenos is Y0-1,
	Ymenos2 is Y0-2,
	estrada(E1, _, _, Xmenos, Ymenos),
	estrada(E2, _, _, X0, Ymenos2),
	estrada(E3, Xmais, Ymenos, _, _),
	!.

estradasPossiveis(EstradaActual, n, [E1, E2, E3]) :-
	estrada(EstradaActual, _, _, X1, Y1),
	Xmenos is X1-1,
	Xmais is X1+1,
	Ymais is Y1+1,
	Ymais2 is Y1+2,
	estrada(E1, _, _, Xmenos, Ymais),
	estrada(E2, X1, Ymais2, _, _),
	estrada(E3, Xmais, Ymais, _, _),
	!.

estradasPossiveis(EstradaActual, o, [E1, E2, E3]) :-
	estrada(EstradaActual, _, _, X1, Y1),
	Xmais is X1+1,
	Xmais2 is X1+2,
	Ymenos is Y1-1,
	Ymais is Y1+1,
	estrada(E1, _, _, Xmais, Ymenos),
	estrada(E2, Xmais2, Y1, _, _),
	estrada(E3, Xmais, Ymais, _, _),
	!.

estradasPossiveis(EstradaActual, e, [E1, E2, E3]) :-
	estrada(EstradaActual, X0, Y0, _, _),
	Xmenos is X0-1,
	Xmenos2 is X0-2,
	Ymenos is Y0-1,
	Ymais is Y0+1,
	estrada(E1, _, _, Xmenos, Ymenos),
	estrada(E2, _, _, Xmenos2, Y0),
	estrada(E3, Xmenos, Ymais, _, _),
	!.

maximo(X,X,X) :-
	!.
maximo(X,Y,X) :-
	X>Y,
	!.
maximo(X,Y,Y) :-
	Y>X,
	!.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% TRANSFORMA UMA LISTA DE INTEIROS NUMA STRING SEPARADA POR HIFEN's
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

formata([], $$) :-
	!.

formata([L|Ls], Str) :-
	string_integer(S, L),
	formata(Ls, Str1),
	strcat(S, $-$, Str2),
	strcat(Str2, Str1, Str),
	!.
```

- **TrafficSim.java**

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import java.util.*;
import amzi.ls.*;

public class TrafficSim extends javax.swing.JFrame
{
	protected static TrafficSim ts;
	protected static JPanel panel1;
	protected static Image roadHor;
	protected static Image roadVert;
	protected static Image cross;
	protected static Image curva;
	protected static Image greenup;
	protected static Image greendown;
	protected static Image greenleft;
	protected static Image greenright;
	protected static Image redup;
	protected static Image reddown;
	protected static Image redleft;
	protected static Image redright;
	protected Image imageselec;
	private JLabel jlblCarros;
	private JTextField jtxtCarros;
	protected static JLabel jlblSim;
	private ButtonGroup btSim;
	private JRadioButton jrbFixo;
	private JRadioButton jrbReactivo;
	private JRadioButton jrbMsg;
	private JTextField jtxtTempo;
	private JLabel jlblTempo;
	private JButton jbRoad;
	private JButton jbSimular;
	protected static Cidade cidade;
	protected static ThreadInterface ti;
	private boolean inicioEstrada;
	private int estradaXini, estradaYini;
	boolean simular = true;

	public static void main (String args[])
	{
		ts = new TrafficSim();
		ts.setSize(870,730);
		ts.show();

		try
		{
			ti = new ThreadInterface();
			ti.inicializa(ts.cidade);
		}
		catch(LSException lse)
		{
			System.out.println(lse);
		}
	}

  /**
   * Construtor
   * Define os componentes da interface gráfica
   */
	public TrafficSim()
	{
		initComponents ();
		pack ();
```

```
            inicioEstrada = false;
        }

    /**
     * Inicialização dos componentes necessários à simulação
     */
        private void initComponents ()
        {
            addWindowListener (new java.awt.event.WindowAdapter ()
            {
                public void windowClosing (java.awt.event.WindowEvent evt)
                {
                    System.exit(0);
                }
            }
            );

            setTitle("Simulação do tráfego de uma cidade");
            setSize(870,730);
            setResizable(false);
            panel1 = new JPanel();
            panel1.setLayout(null);
            panel1.setLocation(2,2);
            panel1.setSize(200,700);
            getContentPane().add(panel1);
            cidade = new Cidade (this, this.getSize());
            cidade.setLocation(20, 60);
            cidade.addMouseListener( new MouseAdapter() {
            public void mouseClicked( MouseEvent e) {
            mouse_click(e);
            }});
            getContentPane().add(cidade);
            roadHor = (new ImageIcon("roadhor.jpg")).getImage();
            roadVert = (new ImageIcon("roadvert.jpg")).getImage();
            cross = (new ImageIcon("cross.gif")).getImage();
            curva = (new ImageIcon("curva.gif")).getImage();
            greenup = (new ImageIcon("greenup.gif")).getImage();
            greendown = (new ImageIcon("greendown.gif")).getImage();
            greenleft = (new ImageIcon("greenleft.gif")).getImage();
            greenright = (new ImageIcon("greenright.gif")).getImage();
            redup = (new ImageIcon("redup.gif")).getImage();
            reddown = (new ImageIcon("reddown.gif")).getImage();
            redleft = (new ImageIcon("redleft.gif")).getImage();
            redright = (new ImageIcon("redright.gif")).getImage();
            imageselec = roadHor;
            jbRoad = new JButton (new ImageIcon("roadhor.jpg"));
            jbRoad.setBounds(15, 40, 45, 45);
            jbRoad.setBorder(BorderFactory.createRaisedBevelBorder());
            jbRoad.setSelected(true);
            jbRoad.addMouseListener( new MouseAdapter() {
            public void mouseClicked( MouseEvent e) {
            mouse_click(e);
            }});
            jbSimular = new JButton ("Simular!");
            jbSimular.setBorder(BorderFactory.createRaisedBevelBorder());
            jbSimular.setBounds(30, 130, 70, 45);
            jbSimular.addMouseListener( new MouseAdapter() {
            public void mouseClicked( MouseEvent e) {
            mouse_click(e);
            }});
            jlblCarros = new JLabel("Nº de Viaturas:");
            jlblCarros.setFont(new java.awt.Font("dialog", 1, 12));
            jlblCarros.setBounds(15, 240, 90, 15);
            jtxtCarros = new JTextField("5");
            jtxtCarros.setHorizontalAlignment(JTextField.RIGHT);
            jtxtCarros.setFont(new java.awt.Font("dialog", 1, 12));
```

```
            jtxtCarros.setBounds(15, 260, 90, 15);
            jlblSim = new JLabel("Tipo de Simulação");
            jlblSim.setFont(new java.awt.Font("dialog", 1, 12));
            jlblSim.setBounds(15, 290, 120, 15);
            btSim = new ButtonGroup();
            jrbFixo = new JRadioButton("Tempo Fixo", true);
            jrbFixo.setFont(new java.awt.Font("dialog", 1, 12));
            jrbFixo.setBounds(15, 315, 100, 15);
            jrbReactivo = new JRadioButton("Reactivo", false);
            jrbReactivo.setFont(new java.awt.Font("dialog", 1, 12));
            jrbReactivo.setBounds(15, 335, 120, 15);
            jrbMsg = new JRadioButton("Mensagens", false);
            jrbMsg.setFont(new java.awt.Font("dialog", 1, 12));
            jrbMsg.setBounds(15, 355, 120, 15);
            btSim.add (jrbFixo);
            btSim.add (jrbReactivo);
            btSim.add (jrbMsg);
            jtxtTempo = new JTextField("10");
            jtxtTempo.setHorizontalAlignment(JTextField.RIGHT);
            jtxtTempo.setFont(new java.awt.Font("dialog", 1, 12));
            jtxtTempo.setBounds(125, 315, 35, 15);
            jlblTempo = new JLabel("Seg.");
            jlblTempo.setFont(new java.awt.Font("dialog", 1, 12));
            jlblTempo.setBounds(165, 315, 35, 15);
            panel1.add(jbRoad);
            panel1.add(jbSimular);
            panel1.add(jlblCarros);
            panel1.add(jtxtCarros);
            panel1.add(jlblSim);
            panel1.add(jtxtTempo);
            panel1.add(jlblTempo);
            panel1.add(jrbFixo);
            panel1.add(jrbReactivo);
            panel1.add(jrbMsg);
      }


  /**
   * mouse_click
   * Processa o evento click do rado
   * @param e Um evento do rato
   */
      private void mouse_click(MouseEvent e)
      {
            if (e.getSource() == cidade)
            {
                  int coordX = cidade.ConvertToMacroCoordinate(e.getX());
                  int coordY = cidade.ConvertToMacroCoordinate(e.getY());

                        // estrada
                  if (imageselec.equals(roadHor))
                  {
                        if (inicioEstrada)
                        {
                              cidade.insereEstrutura(estradaXini, coordX,
estradaYini, coordY);
                        }
                        else
                        {
                              estradaXini = coordX;
                              estradaYini = coordY;
                        }
                        inicioEstrada = !inicioEstrada;
                  }

                  cidade.repaint();
```

```
		}
		else if (e.getSource() == jbRoad)
		{
			imageselec = roadHor;
		}
		else if (e.getSource() == jbSimular)
		{
			if (jbSimular.getText().equals("Simular!"))
			{
				simular = true;
				cidade.nrCarro = (new
Integer(jtxtCarros.getText())).intValue();

				if (jrbFixo.isSelected())			// tempo fixo
				{
					cidade.funcionamento = 'F';
					cidade.tempo = (new
Integer(jtxtTempo.getText())).intValue();
				}
				else
				{
					if (jrbReactivo.isSelected())
						cidade.funcionamento = 'R';		//
reactivos
					else
						cidade.funcionamento = 'M';		// troca
de mensagens
				}

				ti.connect();
				jbSimular.setText("Parar!");
			}
			else
			{
				ti.shutdown();
				simular = false;
				jbSimular.setText("Simular!");
			}
		}
	}
}
```

- **Cidade.java**

```
import java.awt.*;
import java.lang.Math.*;

public class Cidade extends Canvas
{
		// variáveis gerais
	int locX, locY, sizeX, sizeY, boxX, boxY, macroX, macroY, NumMacroPixelsX,
NumMacroPixelsY;

		// array de imagens
	Image[][] cityArray;

		// simulador
	TrafficSim ts;

		// dimensões
	Dimension dts, dpn1;

		// tamanho da grelha onde é possível editar a cidade
```

```
	protected static int MACRO_PIXEL = 40;

		// número máximo de semáforos
	final int MAX_SEM = 20;

		// número máximo de carros
	final int MAX_CARROS = 30;

		// número máximo de estradas
	final int MAX_ESTRADAS = 50;

		// número máximo de curvas
	final int MAX_CURVAS = 20;

		// número de semaforos actualmente existentes na cidade
	int nrSemaforos;

		// número de carros actualmente existentes na cidade
	int nrCarro;

		// número de estradas actualmente existentes na cidade
	int nrEstradas;

			// número de curvas actualmente existentes na cidade
	int nrCurvas;

		// temporização dos semaforos(só no modo de tempo fixo)
	int tempo;

		// funcionamento dos semáforos: Fixo, Reactivo ou Mensagens;
	char funcionamento = 'F';

		// matriz das indica a estrutura que existe em cada ponto da cidade
	char [][] estrutura;

		// as estradas que existem na cidade
	Estrada [] estrada;

		// as curvas que existem na cidade
	Curva [] curva;

		// os semaforos que existem na cidade
	Semaforo [] semaforo;

		// os carros que existem na cidade
	Carro [] carro;

/**
 * Construtor
 * Regista todas as informações relativas às diferentes estruturas existentes
 * na simulação.
 * É a representação de uma cidade real
 * @param ts Representa a simulação a que está associada esta cidade
 * @param dts As dimensões da janela
 */
	Cidade (TrafficSim ts, Dimension dts)
	{
		super(); //cria um Canvas
		this.ts = ts;
		this.dts = dts;
		dpn1 = ts.panel1.getSize();
		locX = dpn1.width;
		locY = 0;
		sizeX = this.dts.width - dpn1.width;
		sizeY = this.dts.height;
		boxX = sizeX - 25;
```

```
            boxY = sizeY - 45;
            NumMacroPixelsX = ConvertToMacroCoordinate(boxX);
            NumMacroPixelsY = ConvertToMacroCoordinate(boxY);
            cityArray = new Image[NumMacroPixelsX+1][NumMacroPixelsY+1];

            this.nrSemaforos = 0;
            this.nrCarro = 0;
            this.nrEstradas = 0;
            this.nrCurvas = 0;

            this.semaforo = new Semaforo [MAX_SEM];
            this.estrada = new Estrada[MAX_ESTRADAS];
            this.curva = new Curva[MAX_CURVAS];
            this.carro = new Carro [MAX_CARROS];

            for(int i =1;i <=NumMacroPixelsX; i++)
                  for(int j =1;j <=NumMacroPixelsY; j++)
                        cityArray[i][j] = null;

            this.estrutura = new char [NumMacroPixelsX+1][NumMacroPixelsY+1];
                        // cria uma cidade vazia (sem estradas)
            for(int i =0;i <NumMacroPixelsX; i++)
                  for(int j =0;j <NumMacroPixelsY; j++)
                        this.estrutura[i][j] = 'v';
      }

  /**
   * Permite desenhar todas as estruturas da cidade
   * Deve ser chamado todas as vezes que ocorre uma alteração na cidade
   * e esta deve ser visualizada
   * @param g Os gráficos que estão desenhados na cidade
   */
      public void paint(Graphics g)
      {
            setLocation(locX, locY);
            setSize(sizeX, sizeY);

            macroY = boxY;
            macroX = boxX;

                  // desenho das esctuturas existentes na cidade
            for(int i=0; i<NumMacroPixelsX; i++)
                  for(int j=0;j <NumMacroPixelsY; j++)
                  {
                        macroX = ConvertFromMacroCoordinate(i);
                        macroY = ConvertFromMacroCoordinate(j);
                        try
                        {
                              g.drawImage((Image)cityArray[i][j], macroX+6, macroY+6,
null);
                        }
                        catch (NullPointerException e)
                        {
                              g.setColor(Color.white);
                              g.fillRect(macroX+6, macroY+6, 39, 39);
                              g.setColor(Color.black);
                        }
                  }

                  // desenho dos carros
            for (int i=0; i<nrCarro; i++)
            {
                  if (carro[i].sentido == 'n')
                        g.drawImage((Image)ts.carroImg, carro[i].posX+7,
carro[i].posY, null);
                  else if (carro[i].sentido == 's')
```

```
                        g.drawImage((Image)ts.carroImg, carro[i].posX+29,
carro[i].posY, null);
                else if (carro[i].sentido == 'o')
                        g.drawImage((Image)ts.carroImg, carro[i].posX,
carro[i].posY+29, null);
                else //      if (carro[i].sentido == 'e')
                        g.drawImage((Image)ts.carroImg, carro[i].posX,
carro[i].posY+7, null);
            }


            if(!ts.simular)
            {
                aprEstatisticas(g);
            }
        }

   /**
    * Permite apresentar as estatísticas finais, no final da apresentação
    * @param g Os gráficos
    */
       private void aprEstatisticas(Graphics g)
       {
            g.setColor(Color.gray);
            g.fill3DRect(6,6 , sizeX-6, sizeY-6, true);
            g.setColor(Color.white);

                // título
            g.setFont(new java.awt.Font("dialog", Font.BOLD, 20));
            g.drawString("Estatísticas da Simulação", 20, 25);

                // rubricas
            g.setFont(new java.awt.Font("dialog", Font.BOLD, 14));
            g.drawString("Tipo de Simulação:", 15, 150);
            g.drawString("Nº de Semáforos:", 15, 200);
            g.drawString("Nº de Estradas:", 15, 250);
            g.drawString("Nº de Curvas:", 15, 300);
            g.drawString("Nº de Carros:", 15, 350);
            g.drawString("Distância Total Percorrida:", 15, 400);
            g.drawString("Distância Média (por carro):", 15, 450);
            g.drawString("Nº Total de Inversões de Marcha:", 15, 500);
            g.drawString("Tempo de Espera Total:", 15, 550);
            g.drawString("Tempo de Espera Médio:", 15, 600);

                // resultados
            g.setFont(new java.awt.Font("dialog", Font.PLAIN, 12));
            String tipoSim = "";
            switch(funcionamento)
            {
                case 'F':
                    tipoSim = "Temporização fixa";
                    break;
                case 'R':
                    tipoSim = "Semároros reactivos";
                    break;
                case 'M':
                    tipoSim = "Mensagens";
                    break;
            }
            g.drawString(tipoSim, 150, 150);
            g.drawString((new Integer(nrSemaforos).toString()), 150, 200);
            g.drawString((new Integer(nrEstradas).toString()), 150, 250);
            g.drawString((new Integer(nrCurvas).toString()), 150, 300);
            g.drawString((new Integer(nrCarro).toString()), 150, 350);

            int nr=0;
```

```
            for (int i=0; i<nrCarro; i++)
                  nr += carro[i].distPercorrida;
            g.drawString((new Integer(nr).toString()), 300, 400);

            g.drawString((new Float(nr/nrCarro).toString()), 300, 450);

            nr = 0;
            for (int i=0; i<nrCarro; i++)
                  nr += carro[i].inversoes;
            g.drawString((new Integer(nr).toString()), 300, 500);

            nr = 0;
            for (int i=0; i<nrCarro; i++)
                  nr += carro[i].tempoEspera;
            g.drawString((new Integer(nr).toString()), 300, 550);

            g.drawString((new Integer(nr/nrCarro).toString()), 300, 600);
      }


   /**
    * Converte uma coordenada em pixeis para macro-pixeis
    * @param coord A coordenada em pixeis a converter para macro-pixeis
    */
      public static int ConvertToMacroCoordinate(int coord)
      {
            return (int)java.lang.Math.ceil((double)coord / MACRO_PIXEL)-1;
      }

  /**
    * Converte uma coordenada em macro-pixeis para pixeis
    * @param macroCoord A coordenada em macro-pixeis a converter para pixeis
    */
      public static int ConvertFromMacroCoordinate(int macroCoord)
      {
            return macroCoord * MACRO_PIXEL ;
      }

  /**
    * Instala uma estrada na cidade
    * @param xIni A coordenada X inicial da estrutura a inserir na cidade
    * @param xFim A coordenada X final da estrutura a inserir na cidade
    * @param yIni A coordenada Y inicial da estrutura a inserir na cidade
    * @param yFim A coordenada Y final da estrutura a inserir na cidade
      * @return true se conseguiu inserir correctamente, false em caso contrário
      */
      public boolean insereEstrutura(int xIni, int xFim, int yIni, int yFim)
      {
            int x0, y0, x1, y1, xTmp, yTmp;
            boolean horizontal = (yIni == yFim);
            boolean inseriuCurva = false;
            Image road;

            xTmp = x0 = Math.min(xIni, xFim);
            x1 = Math.max(xIni, xFim);
            yTmp = y0 = Math.min(yIni, yFim);
            y1 = Math.max(yIni, yFim);

            if (horizontal)     // estrada horizontal
                  road = ts.roadHor;
            else                        // estrada vertical
                  road = ts.roadVert;

            if (podeInserirEstrutura(x0, x1, y0, y1))
            {
                  if (pontoTerminal(x0, y0))
```

```
                    {
                          inseriuCurva = true;
                          estrutura[x0][y0] = 'c';
                          cityArray[x0][y0] = ts.curva;
                          curva[nrCurvas++] = new Curva(this, x0, y0);
                          if (horizontal)
                                x0++;
                          else
                                y0++;
                    }
                    else if (pontoTerminal(x1, y1))
                    {
                          inseriuCurva = true;
                          estrutura[x1][y1] = 'c';
                          cityArray[x1][y1] = ts.curva;
                          curva[nrCurvas++] = new Curva(this, x1, y1);
                    }

                    for (int i=x0; i<=x1; i++)
                          for (int j=y0; j<=y1; j++)
                          {
                                if (estrutura[i][j] == 'e')
                                {
                                      estrutura[i][j] = 's';

                                      this.cityArray[i][j] = ts.cross;
                                      this.cityArray[i+1][j+1] = ts.redup;
                                      this.cityArray[i-1][j+1] = ts.redright;
                                      this.cityArray[i-1][j-1] = ts.reddown;
                                      this.cityArray[i+1][j-1] = ts.redleft;

                                      this.semaforo[nrSemaforos++] = new
Semaforo(nrSemaforos, this, i, j);

                                      if (horizontal)
                                      {
                                            estrada[nrEstradas++] = new Estrada(this,
xTmp, yTmp, i-1, j);
                                            divideEstrada(i,j-1);
                                            xTmp = i+1;
                                            yTmp = j;
                                      }
                                      else
                                      {
                                            estrada[nrEstradas++] = new Estrada(this,
xTmp, yTmp, i, j-1);
                                            divideEstrada(i-1, j);
                                            xTmp = i;
                                            yTmp = j+1;
                                      }
                                }
                                else if (estrutura[i][j] != 'c')
                                {
                                      estrutura[i][j] = 'e';
                                      cityArray[i][j] = road;
                                }
                          }
                    estrada[nrEstradas++] = new Estrada(this, xTmp, yTmp, x1, y1);
                    if (inseriuCurva)
                          detQuadrante();

                    return true;
              }
              return false;
        }
```

```
  /**
    * determina os quadrantes de todas as curvas da cidade
      *
      */
      private void detQuadrante()
      {
            for (int i=0; i<nrCurvas; i++)
            {
                  if (curva[i].quadrante == -1)
                  {
                        int x = curva[i].x;
                        int y = curva[i].y;

                        if ( estrutura[x-1][y] == 'e' && estrutura[x][y+1] == 'e')
                              curva[i].quadrante = 1;
                        else   if ( estrutura[x+1][y] == 'e' && estrutura[x][y+1] ==
'e')
                              curva[i].quadrante = 2;
                        else   if ( estrutura[x+1][y] == 'e' && estrutura[x][y-1] ==
'e')
                              curva[i].quadrante = 3;
                        else //if ( estrutura[x-1][y] == 'e' && estrutura[x][y-1] ==
'e')
                              curva[i].quadrante = 4;
                  }
            }

      }

  /**
    * divisão de uma estrada devido à criação de um cruzamento
    * @param x A coordenada X onde será feita a divisão
    * @param y A coordenada Y onde será feita a divisão
      */
      private void divideEstrada(int x, int y)
      {
            boolean flag = true;
            for (int i=0; i<nrEstradas && flag;i++ )
            {
                  if (estrada[i].orientacao && estrada[i].pertence(x, y))    //
horizontal
                  {
                        estrada[nrEstradas++] = new Estrada(this, x+2, y,
estrada[i].xFim, y);
                        estrada[i].xFim = x;
                        flag = false;
                  }
                  else if (estrada[i].pertence(x, y))
      // vertical
                  {
                        estrada[nrEstradas++] = new Estrada(this, x, y+2, x,
estrada[i].yFim);
                        estrada[i].yFim = y;
                        flag = false;
                  }
            }
      }

  /**
    * verifica se estes pontos podem conter uma estrada
    * @param xIni A coordenada X inicial
      * @param xFim A coordenada X final
    * @param yIni A coordenada Y inicial
      * @param yFim A coordenada Y final
      * @return true se for possível inserir a estrutura, false em caso contrário
      */
```

```
	private boolean podeInserirEstrutura(int xIni, int xFim, int yIni, int yFim)
	{
		if(nrEstradas == MAX_ESTRADAS)
			return false;

			// só é possível criar uma estrada de cada vez
		if (xIni != xFim && yIni != yFim)
			return false;

			// verifica se o local está vazio
		for (int i=xIni; i<=xFim; i++)
			for (int j=yIni; j<=yFim; j++)
				if (estrutura[i][j] == 's')
					return false;
		return true;
	}

  /**
    * verifica se existe alguma estrada que inicie ou termine neste ponto
    * @param x A coordenada X
    * @param y A coordenada Y
     * @return true se existir alguma estrada que inicie ou termine em (x,y), flase em
caso contrário
     */
	private boolean pontoTerminal(int x, int y)
	{
		boolean flag=true;

		for (int i=0; i<nrEstradas && flag; i++)
		{
			flag = !(((estrada[i].xIni == x) && (estrada[i].yIni == y)) ||
((estrada[i].xFim == x) && (estrada[i].yFim == y)));
		}

		return !flag;
	}

  /**
    * determina a estrada cujo ponto inicial é (x, y)
    * @param x A coordenada X
    * @param y A coordenada Y
     * @return A estrada que tem o seu início no ponto que é passado como argumento
     */
	public Estrada estradaComInicioEm(int x, int y)
	{
		int i;
		boolean flag = false;

		for (i=0; i<nrEstradas && !flag; i++)
			flag = (estrada[i].xIni == x) && (estrada[i].yIni == y);

		return estrada[i-1];
	}

  /**
    * determina a estrada cujo ponto final é (x, y)
    * @param x A coordenada X
    * @param y A coordenada Y
     * @return A estrada que tem o seu fim no ponto que é passado como argumento
     */
	public Estrada estradaComFimEm(int x, int y)
	{
		int i;
		boolean flag = false;

		for (i=0; i<nrEstradas && !flag; i++)
```

```
                flag = (estrada[i].xFim == x) && (estrada[i].yFim == y);

            return estrada[i-1];
      }

  /**
    * determina a estrada cujo ponto inicial é (x, y) e que possui uma dada orientação
(horizontal ou vertical)
    * @param x A coordenada X
    * @param y A coordenada Y
    * @param orientacao true significa que pretendemos uma estrada horizontal, false se
pretendemos uma estrada vertical
       * @return A estrada que tem o seu inicio no ponto que é passado como argumento
       */
      public Estrada estradaComInicioEm(int x, int y, boolean orientacao)
      {
            int i;
            boolean flag = false;

            for (i=0; i<nrEstradas && !flag; i++)
            {
                  flag = (estrada[i].xIni == x) && (estrada[i].yIni == y) &&
(estrada[i].orientacao == orientacao);
                  System.out.println(" ECiF " + ((estrada[i].xIni == x) &&
(estrada[i].yIni == y) && (estrada[i].orientacao == orientacao)) + "   " + x + "    "+
y);
            }

            return estrada[i-1];
      }

  /**
    * determina a estrada cujo ponto final é (x, y) e que possui uma dada orientação
(horizontal ou vertical)
    * @param x A coordenada X
    * @param y A coordenada Y
    * @param orientacao true significa que pretendemos uma estrada horizontal, false se
pretendemos uma estrada vertical
       * @return A estrada que tem o seu fim no ponto que é passado como argumento
       */
      public Estrada estradaComFimEm(int x, int y, boolean orientacao)
      {
            int i;
            boolean flag = false;

            for (i=0; i<nrEstradas && !flag; i++)
            {
                  flag = (estrada[i].xFim == x) && (estrada[i].yFim == y) &&
(estrada[i].orientacao == orientacao);
                  System.out.println("ECFE   " + ((estrada[i].xFim == x) &&
(estrada[i].yFim == y) && (estrada[i].orientacao == orientacao)) + "   " + x + "    "+
y);
            }

            return estrada[i-1];
      }

  /**
    * verifica se existe algum semáforo num determinado ponto
    * @param x A coordenada X
    * @param y A coordenada Y
       * @return o número do semaforo ou -1 se não existir
       */
      public int existeSemaforo(int x, int y)
      {
            for (int i=0; i<nrSemaforos; i++)
```

```
		{
			if (semaforo[i].posX == x && semaforo[i].posY == y)
				return i;
		}
		return -1;
	}

/**
    * determina a posição relativa de dois semaforos
    * @param sem0 A identificação de um semaforo
    * @param sem1 A identificação do outro semaforo
      * @return um inteiro que representa a posição relativa dos dois semáforos:
      *             0 - o primeiro está à direita do segundo
      *             1 - o primeiro está por baixo do segundo
      *             2 - o primeiro está à esquerda do segundo
      *             3 - o primeiro está em cima do segundo
      */
      public int posicaoRelativa(int sem0, int sem1)
      {
            if (semaforo[sem0].posX > semaforo[sem1].posX)
                  return 0;
            if (semaforo[sem0].posY > semaforo[sem1].posY)
                  return 1;
            if (semaforo[sem0].posX < semaforo[sem1].posX)
                  return 2;
            return 3;
      }
}
```

- **Semaforo.java**

```
import java.util.StringTokenizer;

public class Semaforo
{
            // número de estradas à volta de cada cruzamento/semáforo
      final int NR_ESTRADAS = 4;

            // o número deste semáforo
      int nrSem;

            // a cidade onde se situa este semaforo
      Cidade cidade;

            // as estradas que existem no cruzamento
      Estrada [] estrada;

            // as coordenadas do semáforo
      int posX, posY;

            // true -> verde; false -> vermelho
      boolean [] luz;

            // sistema de gestão de mensagens
      Mensageiro mensageiro;

            // flag que indica se a pergunta deste semáforo já foi respondida
      boolean respondeu;

  /**
   * Construtor
   * Cria uma estrutura que permite representar um semáforo ou cruzamento
   * @param nr Número identificatido deste semaforo
   * @param cidade A cidade a que pertence este semáforo
   * @param posX A coordenada X da posição deste semáforo na grelha de macro-pixeis
   * @param posY A coordenada Y da posição deste semáforo na grelha de macro-pixeis
```

```
 */
    public Semaforo(int nr, Cidade cidade, int posX, int posY)
    {
           nrSem = nr;
           this.posX = posX;
           this.posY = posY;
           this.estrada = new Estrada[NR_ESTRADAS];
           this.luz = new boolean[NR_ESTRADAS];
           this.cidade = cidade;
           luz[0] = true;
           for (int i=1; i<NR_ESTRADAS; i++)
                  luz[i] = false;
           respondeu = false;
    }

/**
 * preenche o array de estradas
 * @param e0 Uma estrada
 * @param e1 Uma estrada
 * @param e2 Uma estrada
 * @param e3 Uma estrada
 */
    public void insereEstradas(Estrada e0, Estrada e1, Estrada e2, Estrada e3)
    {
           estrada[0] = e0;
           estrada[1] = e1;
           estrada[2] = e2;
           estrada[3] = e3;
    }

/**
 * preenche o array de estradas
 */
    public void insereEstradas()
    {
           estrada[0] = cidade.estradaComFimEm(posX-1, posY);
           estrada[1] = cidade.estradaComFimEm(posX, posY-1);
           estrada[2] = cidade.estradaComInicioEm(posX+1, posY);
           estrada[3] = cidade.estradaComInicioEm(posX, posY+1);
    }

/**
 * determina o semáforo que está verde
 * @return O semáforo que está verde
 */
    public int verde()
    {
           if (luz[0])
                  return 0;
           else
           if (luz[1])
                  return 1;
           else
           if (luz[2])
                  return 2;
           else
           //if (luz[3])
                  return 3;
    }


/**
 * Acende o verde noutro semáforo
 * @param novoVerde O número do próximo semáforo a acender verde
 */
    public void mudaLuz(int novoVerde)
```

```
    {
        luz[0] = luz[1] = luz[2] = luz[3] = false;
        luz[novoVerde] = true;
    }

/**
 * Acende o verde noutro semáforo (o seguinte, no sentido dos ponteiros do relógio)
 */
    public void mudaLuz()
    {
        if (luz[0])
        {
            luz[0] = false;
            luz[1] = true;
        }
        else
        if (luz[1])
        {
            luz[1] = false;
            luz[2] = true;
        }
        else
        if (luz[2])
        {
            luz[2] = false;
            luz[3] = true;
        }
        else
        // if (luz[3])
        {
            luz[3] = false;
            luz[0] = true;
        }
    }


 /**
  * Determina os semaforos que são vizinhos a este semaforo
  * @return Um array contendo os semaforos vizinhos
  */
    public Semaforo [] vizinhos()
    {
        Semaforo [] sem = new Semaforo[4];
        int i=0;
        int nrSem;

        if ((nrSem = cidade.existeSemaforo(estrada[0].xIni-1, posY)) != -1)
            sem[i++] = cidade.semaforo[nrSem];

        if ((nrSem = cidade.existeSemaforo(posX, estrada[1].yIni-1)) != -1)
            sem[i++] = cidade.semaforo[nrSem];

        if ((nrSem = cidade.existeSemaforo(estrada[2].xFim+1, posY)) != -1)
            sem[i++] = cidade.semaforo[nrSem];

        if ((nrSem = cidade.existeSemaforo(posX, estrada[3].yFim+1)) != -1)
            sem[i++] = cidade.semaforo[nrSem];

        return sem;
    }

 /**
  * Actualiza a base de dados deste semaforo de acordo
  * com uma mensagem recebida
  * @param emissor A identificação da origem da mensagem
  * @param msg A mensagem recebida
```

```
    * @param tipo O tipo da mensagem (pedido -> ask OU resposta -> ans)
    */
       public void processaMsg(int emissor, String msg, String tipo)
       {
              if (tipo.equals("ask"))
              {
                           // responde ao semaforo que efectuou o pedido
                     mensageiro.enviaMsg(nrSem-1, "R"+emissor, detSituacao(), "ans");
              }
              else // if (tipo.equals("ans"))
              {
                           // processa a resposta de um semáforo
                     int e0, e1, e2, e3;
                     StringTokenizer st = new StringTokenizer(msg);
                     String str;
                     str = (String)st.nextElement();
                     e0 = (new Integer(str.substring(str.indexOf('-')+1))).intValue();

                     str = (String)st.nextElement();
                     e1 = (new Integer(str.substring(str.indexOf('-')+1))).intValue();

                     str = (String)st.nextElement();
                     e2 = (new Integer(str.substring(str.indexOf('-')+1))).intValue();

                     str = (String)st.nextElement();
                     e3 = (new Integer(str.substring(str.indexOf('-')+1,
str.indexOf('"')))).intValue();

                           // determina a posicão relativa entre este semaforo e quem
enviou a mensagem
                     switch(cidade.posicaoRelativa(nrSem-1, emissor))
                     {
                           case 0:      // o emissor está à direita deste semaforo
                                  mensageiro.sitVizinhos[emissor][0] = e1;
                                  mensageiro.sitVizinhos[emissor][1] = e2;
                                  mensageiro.sitVizinhos[emissor][2] = e3;
                                  break;

                           case 1: // o emissor está por baixo deste semaforo
                                  mensageiro.sitVizinhos[emissor][0] = e2;
                                  mensageiro.sitVizinhos[emissor][1] = e3;
                                  mensageiro.sitVizinhos[emissor][2] = e0;
                                  break;

                           case 2: // o emissor está à esquerda deste semaforo
                                  mensageiro.sitVizinhos[emissor][0] = e3;
                                  mensageiro.sitVizinhos[emissor][1] = e0;
                                  mensageiro.sitVizinhos[emissor][2] = e1;
                                  break;

                           case 3: // o emissor está em cima deste semaforo
                                  mensageiro.sitVizinhos[emissor][0] = e0;
                                  mensageiro.sitVizinhos[emissor][1] = e1;
                                  mensageiro.sitVizinhos[emissor][2] = e2;
                                  break;
                     }
                     respondeu = true;
              }
       }

   /**
    * Cria uma string contendo a informação relativa à situação deste semáforo
    * relativamente ao número de carros que a ele se dirigem.
    * @return Uma mensagem formatada para ser enviada ao servidor de mensagens
    */
       private String detSituacao()
```

```
	{
		return "\"" + 0 + "-" + estrada[0].nrCarros1 + " " + 1 + "-" +
		estrada[1].nrCarros1 + " " + 2 + "-" + estrada[2].nrCarros2 + " " + 3 + "-"
		+ estrada[3].nrCarros2 + "\"";
	}
}
```

- **Estrada.java**

```
public class Estrada
{
		// coordenadas
	int xIni, xFim, yIni, yFim;

		// nr de carros em cada direcção
	int nrCarros1, nrCarros2;

		// false é vertical; true é horizontal
	boolean orientacao;

		// a cidada a que esta estrada pertence
	Cidade cidade;

  /**
   * Construtor
   * Cria uma estrutura que representa uma estrada real
   * @param cidade A cidade a que pertence esta estrada
   * @param x0 Coordenada x do ponto inicial
   * @param y0 Coordenada y do ponto inicial
   * @param x1 Coordenada x do ponto final
   * @param y1 Coordenada y do ponto final
   */
	public Estrada(Cidade cidade, int x0, int y0, int x1, int y1)
	{
		this.cidade = cidade;
		this.xIni = x0;
		this.xFim = x1;
		this.yIni = y0;
		this.yFim = y1;
		this.nrCarros1 = this.nrCarros2 = 0;
		orientacao = (x0!=x1);
	}

  /**
   * Determina se esta estrada tem inicio ou fim nos lados da cidade
   * @return true caso a estrada tenha início ou fim num dos lados da cidade, false caso
contrário
   */
	public boolean limite()
	{
		return (yIni == 0) || (xIni == 0) || (yFim == (cidade.NumMacroPixelsY-1))
|| (xFim == (cidade.NumMacroPixelsX-1));
	}

  /**
   * coordenada X da entrada na estrada
   * @return um valor inteiro que representa a coordenada X da entrada na estrada
   */
	public int entradaX()
	{
		if (!orientacao)
			return xIni;
		else if (xIni==0)
			return 0;
```

```
		else
			return xFim;
	}

  /**
   * coordenada Y da entrada na estrada
   * @return um valor inteiro que representa a coordenada Y da entrada na estrada
   */
	public int entradaY()
	{
		if (orientacao)
			return yIni;
		else if (yIni==0)
			return 0;
		else   // estrada vertical
			return yFim;
	}


  /**
   * O sentido se entrarmos nesta estrada pelo lado da cidade
   * @return um char que representa o sentido
   */
	public char sentidoEntrada(int xEntrada, int yEntrada)
	{
		if (xEntrada == 0)
			return 'o';
		else if (yEntrada == 0)
			return 'n';
		else if (cidade.ConvertToMacroCoordinate(xEntrada) ==
(cidade.NumMacroPixelsX-2))
			return 'e';
		else
			return 's';
	}

  /**
   * Determina o sentido de uma estrada
   * @return um char que representa o sentido
   */
	public char sentidoEstrada(int x, int y)
	{
		if(orientacao)            // estrada horizontal
		{
			if(x==xIni)
				return 'o';
			else
				return 'e';
		}
		else                      // estrada vertical
		{
			if(y==yIni)
				return 'n';
			else
				return 's';
		}
	}


  /**
   * Verifica se um ponto petence a esta estrada
   * @return true, se o ponto pertencer à estrada, false caso contrário
   */
	public boolean pertence(int x, int y)
	{
		return (orientacao && x >= xIni && x <= xFim && y == yIni) ||
```

```
                        (!orientacao && y>= yIni && y<=yFim && x == xIni);
      }


  /**
   * Altera a localização desta estrada
   * @return true, se a alteração for bem sucedida, false caso contrário
   */
      public boolean alteraPosicao(int x0, int x1, int y0, int y1)
      {
            boolean flag = false;

            if(this.xIni != x0)
            {
                  this.xIni = x0;
                  flag = true;
            }

            if(this.xFim != x1)
            {
                  this.xFim = x1;
                  flag = true;
            }
            if(this.yIni != y0)
            {
                  this.yIni = y0;
                  flag = true;
            }
            if(this.yFim != x0)
            {
                  this.yFim = y1;
                  flag = true;
            }

            return flag;
      }
}
```

- **Carro.java**

```
import java.util.*;

public class Carro
{
            // movimento do carro, em pixeis
      final int INCREMENTO = 10;

            // cidade a que o carro pertence
      Cidade cidade;

            // a estrada onde se encontra o carro
      Estrada estrada;

            // coordenadas do carro
      int posX, posY;

            // o sentido em que se movimenta o carro(N - norte, S - sul, E - este, O -
oeste)
      char sentido;

            // variáveis estatísticas
      int distPercorrida;
      int inversoes;
```

```
  /**
   * Construtor
   * Representa um carro
   * @param cidade A cidade a que pertence este carro
   */
      public Carro(Cidade c)
      {
            this.cidade = c;
            posX = posY = -1;
            distPercorrida = inversoes = 0;
            sentido='-';
      }

            //
  /**
   * Actualiza as variáveis estatísticas e determina a estrada a percorrer
   */
      public void entraCidade()
      {
            int [] estradas = new int[cidade.nrEstradas];
            int nrLimite=0;

                  // determinação das estradas com inicio ou fim no limite da cidade
            for (int i=0; i<cidade.nrEstradas; i++)
                  if (cidade.estrada[i].limite())
                        estradas[nrLimite++] = i;


            this.estrada = cidade.estrada[estradas[(new Random()).nextInt(nrLimite)]];

            this.posX = cidade.ConvertFromMacroCoordinate(estrada.entradaX());
            this.posY = cidade.ConvertFromMacroCoordinate(estrada.entradaY());

            this.sentido = estrada.sentidoEntrada(posX, posY);

            if (sentido=='o' || sentido == 'n')
                  this.estrada.nrCarros1++;
            else
                  this.estrada.nrCarros2++;

      }

  /**
   * Chegou a um cruzamento
   * @param estradaSeguinte A nova estrada
   */
      public void mudaEstrada(Estrada estradaSeguinte)
      {
            if (sentido=='o' || sentido == 'n')
                  this.estrada.nrCarros1--;
            else
                  this.estrada.nrCarros2--;

            if (Math.abs(cidade.ConvertToMacroCoordinate(posX)-estradaSeguinte.xIni) <
Math.abs(cidade.ConvertToMacroCoordinate(posX)-estradaSeguinte.xFim))
                  posX = cidade.ConvertFromMacroCoordinate(estradaSeguinte.xIni);
            else
                  posX = cidade.ConvertFromMacroCoordinate(estradaSeguinte.xFim);

            if (Math.abs(cidade.ConvertToMacroCoordinate(posY)-estradaSeguinte.yIni) <
Math.abs(cidade.ConvertToMacroCoordinate(posY)-estradaSeguinte.yFim))
                  posY = cidade.ConvertFromMacroCoordinate(estradaSeguinte.yIni);
            else
                  posY = cidade.ConvertFromMacroCoordinate(estradaSeguinte.yFim);
```

```
            this.sentido =
estradaSeguinte.sentidoEstrada(cidade.ConvertToMacroCoordinate(posX)+1,
cidade.ConvertToMacroCoordinate(posY)+1);
            this.estrada = estradaSeguinte;

            if (sentido=='o' || sentido == 'n')
                  this.estrada.nrCarros1++;
            else
                  this.estrada.nrCarros2++;
      }


  /**
   * chegada a uma estrada sem saída
   */
      public void inverteSentido()
      {
            if (sentido=='o' || sentido == 'n')
                  this.estrada.nrCarros1--;
            else
                  this.estrada.nrCarros2--;

            if (sentido=='n')
                  sentido='s';
            else if (sentido=='s')
                  sentido='n';
            else if (sentido=='e')
                  sentido='o';
            else //if (sentido=='o')
                  sentido='e';

            if (sentido=='o' || sentido == 'n')
                  this.estrada.nrCarros1++;
            else
                  this.estrada.nrCarros2++;

      }


  /**
   * converte o sentido em que se desloca o carro para inteiro
   * @return O mapeamento, em int, do char que representa o sentido do movimento do
carro
   */
      public int sentido()
      {
            if (sentido=='n')
                  return 0;
            else if (sentido=='e')
                  return 1;
            else if (sentido=='s')
                  return 2;
            else //             if (sentido=='o')
                  return 3;
      }

  /**
   * movimenta-se na horizontal
   */
      public void moveX()
      {
            if (sentido=='e')
                  posX -= INCREMENTO;
            else
            //    if (sentido=='o')
                  posX += INCREMENTO;
```

```
    }

/**
 * movimenta-se na vertical
 */
    public void moveY()
    {
        if (sentido=='n')
            posY += INCREMENTO;
        else
        //     if (sentido=='s')
            posY -= INCREMENTO;
    }

/**
 * determina quais as estradas para onde o carro pode seguir
 * @return vector que contém todas as estradas possíveis
 */
    private Vector estradasPossiveis()
    {
        Vector v = new Vector(3);

        if (sentido == 'o') // esquerda -> direita
        {
            v.addElement(cidade.estradaComFimEm(posX+1, posY-1));
            v.addElement(cidade.estradaComInicioEm(posX+2, posY));
            v.addElement(cidade.estradaComInicioEm(posX+1, posY+1));
        }
        else if (sentido == 'n')  // cima -> baixo
        {
            v.addElement(cidade.estradaComInicioEm(posX+1, posY+1));
            v.addElement(cidade.estradaComInicioEm(posX, posY+2));
            v.addElement(cidade.estradaComFimEm(posX-1, posY+1));
        }
        else if (sentido == 'e')  // direita -> esquerda
        {
            v.addElement(cidade.estradaComInicioEm(posX-1, posY+1));
            v.addElement(cidade.estradaComFimEm(posX-2, posY));
            v.addElement(cidade.estradaComFimEm(posX-1, posY-1));
        }

        else //if (sentido == 's')// baixo -> cima
        {
            v.addElement(cidade.estradaComFimEm(posX-1, posY-1));
            v.addElement(cidade.estradaComFimEm(posX, posY-2));
            v.addElement(cidade.estradaComInicioEm(posX+1, posY-1));
        }

        return v;
    }


/**
 * Verifica em que estrada se encontra o carro
 * @param x Coordenada x
 * @param y Coordenada y
 * @return A estrada onde se encontra
 */
    private Estrada determinaEstrada(int x, int y)
    {
        int i;
        boolean flag = true;
        Cidade c = this.estrada.cidade;

        for (i=0; i<c.nrEstradas && flag; i++)
        {
```

```
                    if ((c.estrada[i].xIni<=x && c.estrada[i].xFim>=x &&
c.estrada[i].yIni == y) ||
                            (c.estrada[i].yIni<=y && c.estrada[i].yFim>=y &&
c.estrada[i].xIni == x))
                                    flag = false;
              }

              return c.estrada[i];
       }

  /**
   * Verifica em que estrada se encontra o carro
   * @return A estrada onde se encontra
   */
       public Estrada determinaEstrada()
       {
              return determinaEstrada(/*this.estrada.cidade,*/ this.posX, this.posY);
       }

  /**
   * Aleatoriamente decide por que estrada vai seguir
   * @return A estrada para onde irá circular
   */
       public Estrada escolheEstradaSeguinte()
       {
              return (Estrada)(estradasPossiveis()).elementAt((new Random()).nextInt(3));
       }
}
```

- **Curva.java**

```
public class Curva
{
              // a cidade a que pertence esta curva
       Cidade cidade;

              // coordenadas desta curva
       int x, y;

              // quadrante (orientação) da curva
       int quadrante;


  /**
   * Construtor
   * Representa a estrutura 'curva'
   * @param cidade A cidade a que pertence esta curva
   * @param x Coordenada x desta curva
   * @param y Coordenada y desta curva
   */
       public Curva(Cidade c, int x, int y)
       {
              cidade = c;
              this.x = x;
              this.y = y;
              quadrante = -1; // quadrante não determinado
       }

  /**
   * Construtor
   * Representa a estrutura 'curva'
   * @param cidade A cidade a que pertence esta curva
   * @param x Coordenada x desta curva
   * @param y Coordenada y desta curva
```

```
   * @param quadrante O quadrante desta curva
   */
       public Curva(Cidade c, int x, int y, int quadrante)
       {
              cidade = c;
              this.x = x;
              this.y = y;
              this.quadrante = quadrante;
       }

  /**
   * Construtor
   * Representa a estrutura 'curva'
   * @param cidade A cidade a que pertence esta curva
   * @param e1 Uma estrada
   * @param e2 A outra estrada
   */
       public Curva(Estrada e1, Estrada e2)
       {
              cidade = e1.cidade;

              if(e1.xFim == e2.xIni && e1.yFim == e2.yIni && e1.orientacao)
              {
                     quadrante = 1;
                     x = e1.xFim;
                     y = e1.yFim;
              }
              else
              if(e1.xIni == e2.xIni && e1.yIni == e2.yIni && !e1.orientacao)
              {
                     quadrante = 2;
                     x = e1.xIni;
                     y = e1.yIni;
              }
              else
              if(e1.xFim == e2.xIni && e1.yFim == e2.yIni && !e1.orientacao)
              {
                     quadrante = 3;
                     x = e1.xFim;
                     y = e1.yFim;
              }
              else
              if(e1.xFim == e2.xFim && e1.yFim == e2.yFim && e1.orientacao)
              {
                     quadrante = 4;
                     x = e1.xFim;
                     y = e1.yFim;
              }
       }
}
```

- **ThreadInterface.java**

```
import amzi.ls.*;
import java.awt.*;
import java.util.*;

public class ThreadInterface
{
       protected static Cidade cidade;
       protected static CarroThread [] Cthread;
       protected static SemThread [] Sthread;
       public static LogicServer [] ls;
       public static LogicServer dummy;
```

```
	protected static Mensageiro mensageiro;


  /**
   * Permite inicializações dos threads
   * @param cidade A cidade a que pertence este ThreadInterface
   */
	public static void inicializa(Cidade cid) throws LSException
	{
		cidade = cid;
		Cthread = new CarroThread[cidade.MAX_CARROS];
		Sthread = new SemThread[cidade.MAX_SEM];
		ls = new LogicServer[cidade.MAX_CARROS];
		dummy = new LogicServer();
		dummy.Init("");
		dummy.Load("transito.xpl");

		for (int i=0; i<cidade.MAX_CARROS; i++)
			ls[i] = new LogicServer();

	}

  /**
   * Permite actualizações do Logic Server
   */
	private static void actualizaLS()
	{
		for (int i=0; i<cidade.nrCarro; i++)
		{
			try{
				ls[i].Init("");
				ls[i].Load("transito.xpl");
			}catch(LSException lse) {System.out.println("EXCEPçAO ao inicializar
" + lse);}

				// insere as estradas
			for (int k=0; k<cidade.nrEstradas; k++)
				try{
					ls[i].AssertaStr("estrada(" + k + ", " +
cidade.estrada[k].xIni + ", " + cidade.estrada[k].yIni + ", " + cidade.estrada[k].xFim +
", " + cidade.estrada[k].yFim + ")");
					System.out.println("estrada(" + k + ", " +
cidade.estrada[k].xIni + ", " + cidade.estrada[k].yIni + ", " + cidade.estrada[k].xFim +
", " + cidade.estrada[k].yFim + ")");
				}catch(LSException lse) {System.out.println("EXCEPçAO ao
inserir as estradas " + lse);}

				// insere as curvas
			for (int k=0; k<cidade.nrCurvas; k++)
				try{
					ls[i].AssertaStr("curva(" + k + ", " +
cidade.curva[k].x + ", " + cidade.curva[k].y + ", " + cidade.curva[k].quadrante + ")");
					System.out.println("curva(" + k + ", " +
cidade.curva[k].x + ", " + cidade.curva[k].y + ", " + cidade.curva[k].quadrante + ")");
				}catch(LSException lse) {System.out.println("EXCEPçAO ao
inserir as estradas " + lse);}

				// insere os semaforos
			for (int k=0; k<cidade.nrSemaforos; k++)
				try{
					ls[i].AssertaStr("semaforo(" + k + ", " +
cidade.semaforo[k].posX + ", " + cidade.semaforo[k].posY + ", " +
cidade.semaforo[k].verde() + ")");
				}catch(LSException lse) {System.out.println("EXCEPçAO ao
inserir os semáforos " + lse);}
```

```
                        // insere os carros
                for (int k=0; k<cidade.nrCarro; k++)
                        try{
                                cidade.carro[k] = new Carro(cidade);
                                ls[i].AssertaStr("carro(" + k + ", -1, -1, -)");
                        }catch(LSException lse) {System.out.println("EXCEPçAO ao
inserir os carros " + lse);}
            }
        }

  /**
   * Permite arrancar os threads
   */
        public static void connect()
        {
                // actualiza o logic server com as estradas, semaforos e carros
existentes
            actualizaLS();

                // determina as estradas vizinhas a cada semáforo
            for (int i=0; i<cidade.nrSemaforos; i++)
                cidade.semaforo[i].insereEstradas();

            try
            {
                        // se o funcionamento for por mensagens cria os mensageiros
                if (cidade.funcionamento == 'M')
                {
                        for (int i=0; i<cidade.nrSemaforos; i++)
                        {
                                cidade.semaforo[i].mensageiro = new Mensageiro(i,
cidade);

        cidade.semaforo[i].mensageiro.setPriority(Thread.NORM_PRIORITY);
                                cidade.semaforo[i].mensageiro.start();
                        }
                }

                        // liga os semaforos
                for (int i=0; i<cidade.nrSemaforos; i++)
                {
                        Sthread[i] = new SemThread(cidade.semaforo[i], ls);
                        Sthread[i].start();
                }

                        // põe os carros em movimento
                for (int i=0; i<cidade.nrCarro; i++)
                {
                        Cthread[i] = new CarroThread(i, cidade.carro[i], ls[i]);
                                // espera 1 segundo entre cada carro
                        try{
                                java.lang.Thread.sleep(1000);
                        }catch(java.lang.InterruptedException e){}
                        Cthread[i].start();
                }
            } catch(Exception e){System.out.println(e);}
        }

  /**
   * Termina a simulação
   */
        public static void shutdown()
        {
                // pára os carros
            for (int i=0; i<cidade.nrCarro; i++)
            {
```

```
                    Cthread[i].terminate();
             }


             for (int i=0; i<cidade.nrSemaforos; i++)
             {
                          // desliga os semaforos
                    Sthread[i].terminate();
                          // desliga os mensageiros se for o caso
                    if (cidade.funcionamento=='M')
                          cidade.semaforo[i].mensageiro.stop();
             }
      }
}
```

- **SemThread.java**

```
import java.lang.ThreadLocal;
import java.util.*;
import java.awt.*;
import amzi.ls.*;
import KQMLLayer.*;

public class SemThread extends Thread
{
             // semáforo associado a este thread
      Semaforo semaforo;

             // variável de controlo
      boolean stop = false;

             // array de servidores prolog
      public static LogicServer [] ls;


  /**
   * Construtor
   * @param semaforo O semáforo que se encontra associado a este thread
   * @param ls [] O array de servidores prolog
   */
      public SemThread(Semaforo semaforo, LogicServer [] ls)
      {
             this.semaforo = semaforo;
             this.ls = ls;
      }

  /**
   * desliga o semáforo
   */
      public void terminate()
      {
             stop = true;
      }

  /**
   * liga o semáforo
   */
      public synchronized void run()
      {
             if (this.semaforo.cidade.funcionamento == 'F')                    // tempo
fixo
             {
                    while(!stop)
```

```
                    {
                          try
                          {
                                synchronized(this)
                                {
                                      int verde;
                                            // actualiza a base de conhecimento

                                      for (int k=0; k<this.semaforo.cidade.nrCarro;
k++)
                                      {
                                            ls[k].RetractStr("semaforo(" +
(semaforo.nrSem) + ", _)");
                                            ls[k].AssertaStr("semaforo(" +
(semaforo.nrSem) + ", " + semaforo.verde() + ")");
                                      }

                                            // desenha a nova luz do semáforo
                                      verde = semaforo.verde();
                                      if (verde == 0)
                                      {
                                            semaforo.cidade.cityArray[semaforo.posX-
1][semaforo.posY-1] = semaforo.cidade.ts.greendown;
                                            semaforo.cidade.cityArray[semaforo.posX-
1][semaforo.posY+1] = semaforo.cidade.ts.redright;
                                      }
                                      else if (verde == 1)
                                      {

      semaforo.cidade.cityArray[semaforo.posX+1][semaforo.posY-1] =
semaforo.cidade.ts.greenleft;
                                            semaforo.cidade.cityArray[semaforo.posX-
1][semaforo.posY-1] = semaforo.cidade.ts.reddown;
                                      }
                                      else if (verde == 2)
                                      {

      semaforo.cidade.cityArray[semaforo.posX+1][semaforo.posY+1] =
semaforo.cidade.ts.greenup;

      semaforo.cidade.cityArray[semaforo.posX+1][semaforo.posY-1] =
semaforo.cidade.ts.redleft;
                                      }
                                      else if (verde == 3)
                                      {
                                            semaforo.cidade.cityArray[semaforo.posX-
1][semaforo.posY+1] = semaforo.cidade.ts.greenright;

      semaforo.cidade.cityArray[semaforo.posX+1][semaforo.posY+1] =
semaforo.cidade.ts.redup;
                                      }


                                            semaforo.cidade.repaint();
                                }
                          }catch(LSException lse) {}

                                // pausa
                          try{
                                sleep(this.semaforo.cidade.tempo * 1000);
                          }catch(java.lang.InterruptedException e){}

                          semaforo.mudaLuz();
                    }
              }
```

```
		else if (this.semaforo.cidade.funcionamento == 'R')  // reactivo
		{
			while(!stop)
			{
				System.out.println("REACTIVO");
				boolean flag = false;

				try
				{
					synchronized(this)
					{

							// actualiza a base de conhecimento
						for (int k=0; k<this.semaforo.cidade.nrCarro;
k++)
						{
							ls[k].RetractStr("semaforo(" +
(semaforo.nrSem) +", _)");
							ls[k].AssertaStr("semaforo(" +
(semaforo.nrSem) + ", " + semaforo.verde() + ")");
						}
					}
				}catch(LSException lse) {}

				while (!flag)
				{
					if (semaforo.estrada[0].nrCarros1 != 0)
					{
						flag = true;
						semaforo.mudaLuz(3);
						semaforo.cidade.cityArray[semaforo.posX-
1][semaforo.posY+1] = semaforo.cidade.ts.greenright;

	semaforo.cidade.cityArray[semaforo.posX+1][semaforo.posY+1] =
semaforo.cidade.ts.redup;
						semaforo.cidade.cityArray[semaforo.posX-
1][semaforo.posY-1] = semaforo.cidade.ts.reddown;

	semaforo.cidade.cityArray[semaforo.posX+1][semaforo.posY-1] =
semaforo.cidade.ts.redleft;
					}
					else
					if (semaforo.estrada[1].nrCarros1 != 0)
					{
						flag = true;
						semaforo.mudaLuz(0);
						semaforo.cidade.cityArray[semaforo.posX-
1][semaforo.posY-1] = semaforo.cidade.ts.greendown;
						semaforo.cidade.cityArray[semaforo.posX-
1][semaforo.posY+1] = semaforo.cidade.ts.redright;

	semaforo.cidade.cityArray[semaforo.posX+1][semaforo.posY+1] =
semaforo.cidade.ts.redup;

	semaforo.cidade.cityArray[semaforo.posX+1][semaforo.posY-1] =
semaforo.cidade.ts.redleft;
					}
					else
					if (semaforo.estrada[2].nrCarros2 != 0)
					{
						flag = true;
						semaforo.mudaLuz(1);

	semaforo.cidade.cityArray[semaforo.posX+1][semaforo.posY-1] =
semaforo.cidade.ts.greenleft;
```

```
                                        semaforo.cidade.cityArray[semaforo.posX-
1][semaforo.posY-1] = semaforo.cidade.ts.reddown;
                                        semaforo.cidade.cityArray[semaforo.posX-
1][semaforo.posY+1] = semaforo.cidade.ts.redright;

      semaforo.cidade.cityArray[semaforo.posX+1][semaforo.posY+1] =
semaforo.cidade.ts.redup;
                                }
                                else
                                if (semaforo.estrada[3].nrCarros2 != 0)
                                {
                                        flag = true;
                                        semaforo.mudaLuz(2);

      semaforo.cidade.cityArray[semaforo.posX+1][semaforo.posY+1] =
semaforo.cidade.ts.greenup;

      semaforo.cidade.cityArray[semaforo.posX+1][semaforo.posY-1] =
semaforo.cidade.ts.redleft;
                                        semaforo.cidade.cityArray[semaforo.posX-
1][semaforo.posY-1] = semaforo.cidade.ts.reddown;
                                        semaforo.cidade.cityArray[semaforo.posX-
1][semaforo.posY+1] = semaforo.cidade.ts.redright;
                                }
                        }
                                // actualiza o canvas

      semaforo.cidade.repaint(semaforo.cidade.ConvertFromMacroCoordinate(semaforo.posX-
1), semaforo.cidade.ConvertFromMacroCoordinate(semaforo.posY-1),
semaforo.cidade.MACRO_PIXEL * 3, semaforo.cidade.MACRO_PIXEL * 3);

                        try{
                                sleep(10000);
                        }catch(java.lang.InterruptedException e){}
                }
        }

        else //if (this.semaforo.funcionamento == 'M')      // mensagens
        {
                // determina os semáforos com quem vai comunicar
              Semaforo [] semVizinhos = semaforo.vizinhos();
              int novoVerde=0, total0=-1, total1=-1, total2=-1, total3=-1;


              while(!stop)
              {
                      System.out.println("MENSAGENS");

                      try
                      {
                              synchronized(this)
                              {

                                      // actualiza a base de conhecimento
                              for (int k=0; k<this.semaforo.cidade.nrCarro;
k++)
                              {
                                      ls[k].RetractStr("semaforo(" +
(semaforo.nrSem) +", _)");
                                      ls[k].AssertaStr("semaforo(" +
(semaforo.nrSem) + ", " + semaforo.verde() + ")");
                              }
                              }
                      }catch(LSException lse) {}

                              // pergunta aos vizinhos qual a situação deles
```

```
                        for (int i=0; i<4 ;i++ )
                        {
                                try
                                {
                                        semaforo.mensageiro.enviaMsg(semaforo.nrSem-1,
"R" + (semVizinhos[i].nrSem-1), "", "ask");
                                }
                                catch(NullPointerException npe)
                                {
                                        // esta excepção ocorre quando o semáforo
tem menos que 4 vizinhos
                                        i=4;
                                }
                        }
                                // espera a resposta
                        while(!semaforo.respondeu)
                                try{
                                        sleep(50);
                                }catch(java.lang.InterruptedException e){}
                        semaforo.respondeu = false;


                                // decide qual o semaforo a abrir

                        try
                        {
                                total0 =
semaforo.mensageiro.sitVizinhos[semVizinhos[3].nrSem][0] +
semaforo.mensageiro.sitVizinhos[semVizinhos[3].nrSem][1] +
semaforo.mensageiro.sitVizinhos[semVizinhos[3].nrSem][2] +
semaforo.estrada[0].nrCarros1;
                        }catch(NullPointerException npe) {
                                total0=semaforo.estrada[0].nrCarros1;
                        }

                        try{
                                total1 =
semaforo.mensageiro.sitVizinhos[semVizinhos[0].nrSem][0] +
semaforo.mensageiro.sitVizinhos[semVizinhos[0].nrSem][1] +
semaforo.mensageiro.sitVizinhos[semVizinhos[0].nrSem][2] +
semaforo.estrada[1].nrCarros1;
                        }catch(NullPointerException npe)
                        {
                                total1=semaforo.estrada[1].nrCarros1;
                        }

                        try{
                                total2 =
semaforo.mensageiro.sitVizinhos[semVizinhos[1].nrSem][0] +
semaforo.mensageiro.sitVizinhos[semVizinhos[1].nrSem][1] +
semaforo.mensageiro.sitVizinhos[semVizinhos[1].nrSem][2] +
semaforo.estrada[2].nrCarros2;
                        }catch(NullPointerException npe)
                        {
                                total2=semaforo.estrada[2].nrCarros2;
                        }

                        try{
                                total3 =
semaforo.mensageiro.sitVizinhos[semVizinhos[2].nrSem][0] +
semaforo.mensageiro.sitVizinhos[semVizinhos[2].nrSem][1] +
semaforo.mensageiro.sitVizinhos[semVizinhos[2].nrSem][2] +
semaforo.estrada[3].nrCarros2;
                        }catch(NullPointerException npe)
                        {
                                total3=semaforo.estrada[3].nrCarros2;
```

```
                        }

                        System.out.println(semaforo.nrSem + " TOTALES!!  " + total0 +
"  " + total1 + "  " + total2 + "  " + total3 + "  ");
                        //novoVerde = ...

                              // determinação do maior
                        novoVerde = maximo(total0, total1, total2, total3);

                        semaforo.mudaLuz(novoVerde);

                              // actualiza a matriz que representa a cidade
                        if (novoVerde==0)
                        {
                              semaforo.cidade.cityArray[semaforo.posX-
1][semaforo.posY-1] = semaforo.cidade.ts.greendown;
                              semaforo.cidade.cityArray[semaforo.posX-
1][semaforo.posY+1] = semaforo.cidade.ts.redright;

      semaforo.cidade.cityArray[semaforo.posX+1][semaforo.posY+1] =
semaforo.cidade.ts.redup;

      semaforo.cidade.cityArray[semaforo.posX+1][semaforo.posY-1] =
semaforo.cidade.ts.redleft;
                        }
                        else
                        if (novoVerde==1)
                        {

      semaforo.cidade.cityArray[semaforo.posX+1][semaforo.posY-1] =
semaforo.cidade.ts.greenleft;
                              semaforo.cidade.cityArray[semaforo.posX-
1][semaforo.posY-1] = semaforo.cidade.ts.reddown;
                              semaforo.cidade.cityArray[semaforo.posX-
1][semaforo.posY+1] = semaforo.cidade.ts.redright;

      semaforo.cidade.cityArray[semaforo.posX+1][semaforo.posY+1] =
semaforo.cidade.ts.redup;
                        }
                        else
                        if (novoVerde==2)
                        {

      semaforo.cidade.cityArray[semaforo.posX+1][semaforo.posY+1] =
semaforo.cidade.ts.greenup;

      semaforo.cidade.cityArray[semaforo.posX+1][semaforo.posY-1] =
semaforo.cidade.ts.redleft;
                              semaforo.cidade.cityArray[semaforo.posX-
1][semaforo.posY-1] = semaforo.cidade.ts.reddown;
                              semaforo.cidade.cityArray[semaforo.posX-
1][semaforo.posY+1] = semaforo.cidade.ts.redright;
                        }
                        else
                        if (novoVerde==3)
                        {
                              semaforo.cidade.cityArray[semaforo.posX-
1][semaforo.posY+1] = semaforo.cidade.ts.greenright;

      semaforo.cidade.cityArray[semaforo.posX+1][semaforo.posY+1] =
semaforo.cidade.ts.redup;
                              semaforo.cidade.cityArray[semaforo.posX-
1][semaforo.posY-1] = semaforo.cidade.ts.reddown;

      semaforo.cidade.cityArray[semaforo.posX+1][semaforo.posY-1] =
semaforo.cidade.ts.redleft;
```

```
                        }

                              // actualiza o canvas

      semaforo.cidade.repaint(semaforo.cidade.ConvertFromMacroCoordinate(semaforo.posX-
1), semaforo.cidade.ConvertFromMacroCoordinate(semaforo.posY-1),
semaforo.cidade.MACRO_PIXEL * 3, semaforo.cidade.MACRO_PIXEL * 3);

                        try{
                              sleep(10000);
                        }catch(java.lang.InterruptedException e){}
                  }
            }
      }     // fim do método 'run'


  /**
   * Determina qual o semáforo a abrir de acordo com a indicação do número de carros que
poderão
   * convergir para cada uma das 4 estradas do cruzamento
   * @param v0 O número de carros na estrada 0
   * @param v1 O número de carros na estrada 1
   * @param v2 O número de carros na estrada 2
   * @param v3 O número de carros na estrada 3
   */
      private int maximo(int v0, int v1, int v2, int v3)
      {
            if (Math.max(v0, v1) == v0 && Math.max(v0,v2) == v0 && Math.max(v0,v3) ==
v0)
                  return 3;
            if (Math.max(v1, v0) == v1 && Math.max(v1,v2) == v1 && Math.max(v1,v3) ==
v1)
                  return 0;
            if (Math.max(v2, v0) == v2 && Math.max(v2,v1) == v2 && Math.max(v2,v3) ==
v2)
                  return 1;
//          if (Math.max(v3, v0) == v3 && Math.max(v3,v1) == v3 && Math.max(v3,v2) ==
v3)
                  return 2;


      }
}
```

- **CarroThread.java**

```
import java.lang.ThreadLocal;
import java.util.*;
import java.awt.*;
import amzi.ls.*;

public class CarroThread extends Thread
{
            // o carro que este thread comanda
      Carro carro;

            // número de ordem do carro
      int numCarro;

            // variável de controlo
      boolean stop = false;

            // servidor lógico (prolog)
      LogicServer ls;

      int MACRO_PIXEL = 40;
```

```
  /**
   * Construtor
   * @param numCarro O número do carro a que este thread está associado
   * @param c O carro a que este thread está associado
   * @param ls O servidor lógico deste thread
   */
       CarroThread(int numCarro, Carro c, LogicServer ls)
       {
              this.numCarro = numCarro;
              this.carro = c;
              this.ls = ls;
              //MACRO_PIXEL = carro.cidade.MACRO_PIXEL;
       }


  /**
   * Permite terminar a execução deste thread
   */
       public void terminate()
       {
              stop = true;
       }

  /**
   * Início da execução
   */
       public synchronized void run()
       {
              long termo=0;
              String accao="";

              try
              {
                     while(!stop)
                     {
                            synchronized(this)
                            {
                                   termo = ls.ExecStr("accao(" + numCarro + ", Accao)");
                                   accao = ls.GetStrArg(termo, 2);
                                   System.out.println("ACCAO " + accao + "    X= " +
(carro.cidade.ConvertToMacroCoordinate(carro.posX)+1) + "    Y= " +
(carro.cidade.ConvertToMacroCoordinate(carro.posY)+1) + " SENTIDO = " + carro.sentido);
                            }

                                   // o carro entra, por uma estrada, na cidade
                            if (accao.equals("entraCidade"))
                            {
                                          // actualiza as variáveis estatísticas e
determina a estrada a percorrer
                                   carro.entraCidade();
                            }

                                   // curva sem cruzamento/semaforo
                            else if (accao.equals("curva1"))
                            {
                                          // significa que avançou uma posição
                                   carro.distPercorrida++;

                                          // apaga a posição anterior
                                   carro.cidade.getGraphics().clearRect(carro.posX,
carro.posY, 15, 15);

       carro.cidade.repaint(carro.cidade.ConvertFromMacroCoordinate(carro.posX)-
```

```
MACRO_PIXEL, carro.cidade.ConvertFromMacroCoordinate(carro.posY)-MACRO_PIXEL,
MACRO_PIXEL, MACRO_PIXEL);


                                    if (carro.estrada.orientacao)    // horizontal
                                    {
                                          carro.estrada.nrCarros1--;
                                          carro.estrada =
carro.cidade.estradaComInicioEm(carro.cidade.ConvertToMacroCoordinate(carro.posX)+1,
carro.cidade.ConvertToMacroCoordinate(carro.posY)+1, false);
                                          carro.posX =
carro.cidade.ConvertFromMacroCoordinate(carro.cidade.ConvertToMacroCoordinate(carro.posX
) + 1);
                                          carro.posY =
carro.cidade.ConvertFromMacroCoordinate(carro.cidade.ConvertToMacroCoordinate(carro.posY
) + 2);
                                          carro.sentido = 'n';
                                          carro.estrada.nrCarros1++;
                                    }
                                    else                                              //
vertical
                                    {
                                          carro.estrada.nrCarros2--;
                                          carro.estrada =
carro.cidade.estradaComFimEm(carro.cidade.ConvertToMacroCoordinate(carro.posX)+1,
carro.cidade.ConvertToMacroCoordinate(carro.posY)+1, true);
                                          carro.posX =
carro.cidade.ConvertFromMacroCoordinate(carro.cidade.ConvertToMacroCoordinate(carro.posX
));
                                          carro.posY =
carro.cidade.ConvertFromMacroCoordinate(carro.cidade.ConvertToMacroCoordinate(carro.posY
) + 1);
                                          carro.sentido = 'e';
                                          carro.estrada.nrCarros2++;
                                    }
                              }
                              else if (accao.equals("curva2"))
                              {
                                          // significa que avançou uma posição
                                    carro.distPercorrida++;

                                          // apaga a posição anterior
                                    carro.cidade.getGraphics().clearRect(carro.posX,
carro.posY, 15, 15);

      carro.cidade.repaint(carro.cidade.ConvertFromMacroCoordinate(carro.posX)-
MACRO_PIXEL, carro.cidade.ConvertFromMacroCoordinate(carro.posY)-MACRO_PIXEL,
MACRO_PIXEL, MACRO_PIXEL);
                                    if (carro.estrada.orientacao)    // horizontal
                                    {
                                          carro.estrada.nrCarros2--;
                                          carro.estrada =
carro.cidade.estradaComInicioEm(carro.cidade.ConvertToMacroCoordinate(carro.posX)+1,
carro.cidade.ConvertToMacroCoordinate(carro.posY)+1, false);
                                          carro.posX =
carro.cidade.ConvertFromMacroCoordinate(carro.cidade.ConvertToMacroCoordinate(carro.posX
) + 1);
                                          carro.posY =
carro.cidade.ConvertFromMacroCoordinate(carro.cidade.ConvertToMacroCoordinate(carro.posY
) + 2);
                                          carro.sentido = 'n';
                                          carro.estrada.nrCarros1++;
                                    }
                                    else                                              //
vertical
                                    {
```

```
                                        carro.estrada.nrCarros2--;
                                        carro.estrada =
carro.cidade.estradaComInicioEm(carro.cidade.ConvertToMacroCoordinate(carro.posX)+1,
carro.cidade.ConvertToMacroCoordinate(carro.posY)+1, true);
                                        carro.posX =
carro.cidade.ConvertFromMacroCoordinate(carro.cidade.ConvertToMacroCoordinate(carro.posX
) + 2);
                                        carro.posY =
carro.cidade.ConvertFromMacroCoordinate(carro.cidade.ConvertToMacroCoordinate(carro.posY
) + 1);
                                        carro.sentido = 'o';
                                        carro.estrada.nrCarros1++;
                                }
                        }
                        else if (accao.equals("curva3"))
                        {
                                // significa que avançou uma posição
                                carro.distPercorrida++;

                                // apaga a posição anterior
                                carro.cidade.getGraphics().clearRect(carro.posX,
carro.posY, 15, 15);

      carro.cidade.repaint(carro.cidade.ConvertFromMacroCoordinate(carro.posX)-
MACRO_PIXEL, carro.cidade.ConvertFromMacroCoordinate(carro.posY)-MACRO_PIXEL,
MACRO_PIXEL, MACRO_PIXEL);

                                if (carro.estrada.orientacao)    // horizontal
                                {
                                        carro.estrada.nrCarros2--;
                                        carro.estrada =
carro.cidade.estradaComFimEm(carro.cidade.ConvertToMacroCoordinate(carro.posX)+1,
carro.cidade.ConvertToMacroCoordinate(carro.posY)+1, false);
                                        carro.posX =
carro.cidade.ConvertFromMacroCoordinate(carro.cidade.ConvertToMacroCoordinate(carro.posX
) + 1);
                                        carro.posY =
carro.cidade.ConvertFromMacroCoordinate(carro.cidade.ConvertToMacroCoordinate(carro.posY
));
                                        carro.sentido = 's';
                                        carro.estrada.nrCarros2++;
                                }
                                else                                            //
vertical
                                {
                                        carro.estrada.nrCarros2--;
                                        carro.estrada =
carro.cidade.estradaComInicioEm(carro.cidade.ConvertToMacroCoordinate(carro.posX)+1,
carro.cidade.ConvertToMacroCoordinate(carro.posY)+1, true);
                                        carro.posX =
carro.cidade.ConvertFromMacroCoordinate(carro.cidade.ConvertToMacroCoordinate(carro.posX
) + 2);
                                        carro.posY =
carro.cidade.ConvertFromMacroCoordinate(carro.cidade.ConvertToMacroCoordinate(carro.posY
) + 1);
                                        carro.sentido = 'o';
                                        carro.estrada.nrCarros1++;
                                }

                        }

                        else if (accao.equals("curva4"))
                        {
                                // significa que avançou uma posição
                                carro.distPercorrida++;
```

```
                                        // apaga a posição anterior
                                carro.cidade.getGraphics().clearRect(carro.posX,
carro.posY, 15, 15);

      carro.cidade.repaint(carro.cidade.ConvertFromMacroCoordinate(carro.posX)-
MACRO_PIXEL, carro.cidade.ConvertFromMacroCoordinate(carro.posY)-MACRO_PIXEL,
MACRO_PIXEL, MACRO_PIXEL);
                                if (carro.estrada.orientacao)    // horizontal
                                {
                                        carro.estrada.nrCarros1--;
                                        carro.estrada =
carro.cidade.estradaComFimEm(carro.cidade.ConvertToMacroCoordinate(carro.posX)+1,
carro.cidade.ConvertToMacroCoordinate(carro.posY)+1, false);
                                        carro.posX =
carro.cidade.ConvertFromMacroCoordinate(carro.cidade.ConvertToMacroCoordinate(carro.posX
) + 1);
                                        carro.posY =
carro.cidade.ConvertFromMacroCoordinate(carro.cidade.ConvertToMacroCoordinate(carro.posY
) - 1);
                                        carro.sentido = 's';
                                        carro.estrada.nrCarros2++;
                                }
                                else                                            //
vertical
                                {
                                        carro.estrada.nrCarros1--;
                                        carro.estrada =
carro.cidade.estradaComFimEm(carro.cidade.ConvertToMacroCoordinate(carro.posX)+1,
carro.cidade.ConvertToMacroCoordinate(carro.posY)+1, true);
                                        carro.posX =
carro.cidade.ConvertFromMacroCoordinate(carro.cidade.ConvertToMacroCoordinate(carro.posX
) - 1);
                                        carro.posY =
carro.cidade.ConvertFromMacroCoordinate(carro.cidade.ConvertToMacroCoordinate(carro.posY
) + 1);
                                        carro.sentido = 'e';
                                        carro.estrada.nrCarros2++;
                                }
                        }

                        else if (accao.equals("direita"))
                        {
                                        // significa que avançou uma posição
                                carro.distPercorrida++;

                                        // apaga a posição anterior
                                if (carro.sentido == 'o')
                                {
                                        carro.cidade.getGraphics().clearRect(carro.posX,
carro.posY + 29, 15, 15);
                                }
                                else //if (carro.sentido == 'e')
                                {
                                        carro.cidade.getGraphics().clearRect(carro.posX,
carro.posY + 7, 15, 15);
                                }



      carro.cidade.repaint(carro.cidade.ConvertFromMacroCoordinate(carro.posX)-
MACRO_PIXEL, carro.cidade.ConvertFromMacroCoordinate(carro.posY)-MACRO_PIXEL,
MACRO_PIXEL, MACRO_PIXEL);
                                // move-se para a direita
                                carro.posX += carro.INCREMENTO;
                        }
```

```
                        else if (accao.equals("esquerda"))
                        {
                                // significa que avançou uma posição
                              carro.distPercorrida++;

                                // apaga a posição anterior
                              if (carro.sentido == 'o')
                              {
                                      carro.cidade.getGraphics().clearRect(carro.posX,
carro.posY + 29, 15, 15);
                              }
                              else //if (carro.sentido == 'e')
                              {
                                      carro.cidade.getGraphics().clearRect(carro.posX,
carro.posY + 7, 15, 15);
                              }


      carro.cidade.repaint(carro.cidade.ConvertFromMacroCoordinate(carro.posX)-
MACRO_PIXEL, carro.cidade.ConvertFromMacroCoordinate(carro.posY)-MACRO_PIXEL,
MACRO_PIXEL, MACRO_PIXEL);
                                // move-se para a esquerda
                              carro.posX -= carro.INCREMENTO;
                        }

                        else if (accao.equals("cima"))
                        {
                                // significa que avançou uma posição
                              carro.distPercorrida++;

                                // apaga a posição anterior
                              if (carro.sentido == 'n')
                              {
                                      carro.cidade.getGraphics().clearRect(carro.posX
+ 7, carro.posY, 15, 15);
                              }
                              else //if (carro.sentido == 's')
                              {
                                      carro.cidade.getGraphics().clearRect(carro.posX
+ 29, carro.posY, 15, 15);
                              }


      carro.cidade.repaint(carro.cidade.ConvertFromMacroCoordinate(carro.posX)-
MACRO_PIXEL, carro.cidade.ConvertFromMacroCoordinate(carro.posY)-MACRO_PIXEL,
MACRO_PIXEL, MACRO_PIXEL);
                                // move-se para cima
                              carro.posY -= carro.INCREMENTO;
                        }

                        else if (accao.equals("baixo"))
                        {
                                // significa que avançou uma posição
                              carro.distPercorrida++;

                                // apaga a posição anterior
                              if (carro.sentido == 'n')
                              {
                                      carro.cidade.getGraphics().clearRect(carro.posX
+ 7, carro.posY, 15, 15);
                              }
                              else //if (carro.sentido == 's')
                              {
                                      carro.cidade.getGraphics().clearRect(carro.posX
+ 29, carro.posY, 15, 15);
                              }
```

```
	carro.cidade.repaint(carro.cidade.ConvertFromMacroCoordinate(carro.posX)-
MACRO_PIXEL, carro.cidade.ConvertFromMacroCoordinate(carro.posY)-MACRO_PIXEL,
MACRO_PIXEL, MACRO_PIXEL);
					// move-se para baixo
				carro.posY+= carro.INCREMENTO;
			}
					// chegou a uma estrada sem saída
			else if (accao.equals("inverte"))
			{
					// significa que avançou uma posição
				carro.distPercorrida++;
				carro.inversoes++;

					// apaga a posição anterior
				if (carro.sentido == 'n')
				{

	carro.cidade.getGraphics().clearRect(carro.posX+7, carro.posY, 15, 15);

	carro.cidade.repaint(carro.cidade.ConvertFromMacroCoordinate(carro.posX+15),
carro.cidade.ConvertFromMacroCoordinate(carro.posY+10), MACRO_PIXEL, MACRO_PIXEL);

				}
				else
				if (carro.sentido == 's')
				{

	carro.cidade.getGraphics().clearRect(carro.posX+29, carro.posY, 15, 15);

	carro.cidade.repaint(carro.cidade.ConvertFromMacroCoordinate(carro.posX+15),
carro.cidade.ConvertFromMacroCoordinate(carro.posY+10), MACRO_PIXEL, MACRO_PIXEL);

				}
				else
				if (carro.sentido == 'o')
				{
					carro.cidade.getGraphics().clearRect(carro.posX,
carro.posY+29, 15, 15);

	carro.cidade.repaint(carro.cidade.ConvertFromMacroCoordinate(carro.posX+15),
carro.cidade.ConvertFromMacroCoordinate(carro.posY+10), MACRO_PIXEL, MACRO_PIXEL);

				}
				else
				//	if (carro.sentido == 'e')
				{
					carro.cidade.getGraphics().clearRect(carro.posX,
carro.posY+7, 15, 15);

	carro.cidade.repaint(carro.cidade.ConvertFromMacroCoordinate(carro.posX+15),
carro.cidade.ConvertFromMacroCoordinate(carro.posY+10), MACRO_PIXEL, MACRO_PIXEL);

				}



	carro.cidade.repaint(carro.cidade.ConvertFromMacroCoordinate(carro.posX)-
MACRO_PIXEL, carro.cidade.ConvertFromMacroCoordinate(carro.posY)-MACRO_PIXEL,
MACRO_PIXEL, MACRO_PIXEL);
					// inversao do sentido de marcha
				carro.inverteSentido();
			}
				// ex.: Cruzamento-3 indica que chegou ao cruzamento 3
```

```
                        else if (accao.startsWith("cruzamento"))
                        {
                                Estrada estradaSeg;
                                int semaforo;

                                        // indica o número do cruzamento/semaforo a que
chegou
                                semaforo = (new Integer(accao.substring(10,
accao.indexOf('-')))).intValue();

                                        // determina a nova estrada a percorrer
                                estradaSeg = escolheEstradaSeguinte(accao);

                                        // espera pelo verde
                                while
(!carro.estrada.cidade.semaforo[semaforo].luz[carro.sentido()])
                                {
                                        // actualiza a variável estatística
                                carro.tempoEspera++;
                                        // pequena pausa
                                try{
                                        sleep(1000);
                                }catch(java.lang.InterruptedException e){}
                                }

                                        // apaga a posição anterior
                                if (carro.sentido == 'n')
                                {

        carro.cidade.getGraphics().clearRect(carro.posX+7, carro.posY, 15, 15);

        carro.cidade.repaint(carro.cidade.ConvertFromMacroCoordinate(carro.posX+15),
carro.cidade.ConvertFromMacroCoordinate(carro.posY+10), MACRO_PIXEL, MACRO_PIXEL);

                                }
                                else
                                if (carro.sentido == 's')
                                {

        carro.cidade.getGraphics().clearRect(carro.posX+29, carro.posY, 15, 15);

        carro.cidade.repaint(carro.cidade.ConvertFromMacroCoordinate(carro.posX+15),
carro.cidade.ConvertFromMacroCoordinate(carro.posY+10), MACRO_PIXEL, MACRO_PIXEL);

                                }
                                else
                                if (carro.sentido == 'o')
                                {
                                        carro.cidade.getGraphics().clearRect(carro.posX,
carro.posY+29, 15, 15);

        carro.cidade.repaint(carro.cidade.ConvertFromMacroCoordinate(carro.posX+15),
carro.cidade.ConvertFromMacroCoordinate(carro.posY+10), MACRO_PIXEL, MACRO_PIXEL);

                                }
                                else
                                //      if (carro.sentido == 'e')
                                {
                                        carro.cidade.getGraphics().clearRect(carro.posX,
carro.posY+7, 15, 15);

        carro.cidade.repaint(carro.cidade.ConvertFromMacroCoordinate(carro.posX+15),
carro.cidade.ConvertFromMacroCoordinate(carro.posY+10), MACRO_PIXEL, MACRO_PIXEL);

                                }
```

```
                                        // avança para a nova estrada
                                synchronized(this)
                                {
                                        carro.mudaEstrada(estradaSeg);
                                }
                        }

                                // desenha o carro na nova posição
                        if (carro.sentido == 'n')
                        {

      carro.cidade.getGraphics().drawImage((Image)carro.cidade.ts.carroImg,
carro.posX+7, carro.posY, null);
                        }
                        else if (carro.sentido == 's')
                        {

      carro.cidade.getGraphics().drawImage((Image)carro.cidade.ts.carroImg,
carro.posX+29, carro.posY, null);
                        }
                        else if (carro.sentido == 'o')
                        {

      carro.cidade.getGraphics().drawImage((Image)carro.cidade.ts.carroImg, carro.posX,
carro.posY+29, null);
                        }
                        else //if (carro.sentido == 'e')
                        {

      carro.cidade.getGraphics().drawImage((Image)carro.cidade.ts.carroImg, carro.posX,
carro.posY+7, null);
                        }


                        synchronized(this)
                        {
                                ls.RetractStr("carro(" + numCarro + ",_,_,_)");
                                ls.AssertaStr("carro(" + numCarro + ", " +
(carro.cidade.ConvertToMacroCoordinate(carro.posX)+1) + ", " +
(carro.cidade.ConvertToMacroCoordinate(carro.posY)+1) + ", " + carro.sentido + ")");
                        }

                                // pequena pausa
                        try{
                                sleep(400);
                        }catch(java.lang.InterruptedException e){}
                }
        }catch(Exception ee) {System.out.println("EXCEPCAO " + ee);}
      }


  /**
   * Determina qual a proxima estrada a percorrer
   * @param estradas A lista de estradas possíveis retornada pelo prolog
   */
      private Estrada escolheEstradaSeguinte(String estradas)
      {
            int [] nrEstradas = new int[3];

            estradas = estradas.substring(estradas.indexOf('-')+1);
            nrEstradas[0] = (new Integer(estradas.substring(0, estradas.indexOf('-
'')))).intValue();

            estradas = estradas.substring(estradas.indexOf('-')+1);
            nrEstradas[1] = (new Integer(estradas.substring(0, estradas.indexOf('-
'')))).intValue();
```

```
            estradas = estradas.substring(estradas.indexOf('-')+1);
            nrEstradas[2] = (new Integer(estradas.substring(0, estradas.indexOf('-
')))).intValue();

            return carro.cidade.estrada[nrEstradas[(new Random()).nextInt(3)]];
      }

}
```

- **Mensageiro.java**

```
import java.lang.*;
import java.io.*;
import java.util.*;

import Abstract.*;
import BaseLayer.*;
import KQMLLayer.*;

import java.awt.*;

public class Mensageiro extends KQMLAgentAction
{
            // a cidade a que pertence este mensageiro
      Cidade cidade;

            // matriz da situacao dos vizinhos
      int [][] sitVizinhos;


 /**
   * Construtor com os valores por omissão
   */
   public Mensageiro()
      {
            super();
      }

  /**
   * Construtor
   * Lê os endereços da AddressTable e tenta criar todas as ligações
   * Se o ServerThread falhar, é invocada o processMessage para apresentar
   * o erro respectivo.
   * @param id Identificação do semaforo a que pertence este mensageiro
   * @param cidade A cidade a que pertence este mensageiro
   */
      public Mensageiro(int id, Cidade cidade)
      {
            super(id+"");
            this.cidade = cidade;

            sitVizinhos = new int[4][3];

            try
            {
                        // criação da tabela de endereços
                  _addresses = new BAddressTable();

                        // endereço dummy
                  _addresses.addAddress(new Address(id+"",null,-
1,"BRecvThread","Cliente"));

                        // a tabela de endereços _addresses não deve ser nula
                  _security = new KQMLSecurity(_addresses);
```

```
                        // Tabela de Ligações
                _connections = new BConnectionTable();

                        // Fila de mensagens
                _queue = new BMessageBuffer();

                        // terminação de cada mensagem
                _endWith = '\004';

                        // -1 significa que não indica o tempo máximo que o thread
espera a recepção da mensagem
                _durationTime = -1;

                        // actualização dos endereços
                _addresses.addAddress(new Address(id + ",localhost," + (5555 - id) +
",BServThread,(" + id + "'s Address)"));
                if(createServerThread(id+"",Thread.NORM_PRIORITY) == null)
                {
                String args[] = new String[1];
                args[0] = id + "";
                processMessage("InitializeFailed",args);
            }

                        // criação dos receptores
                for (int i=0; i<cidade.nrSemaforos; i++)
                {
                        Address addr = new Address("R" + i + ",localhost," + (5555 +
i) + ",BRecvThread,(N" + i + " 's Address)" );
                        _addresses.addAddress(addr);

                        String type = addr.getType();
                        String name = addr.getID();
                        try
                        {
                                createReceiverThread(id + "", name,
Thread.NORM_PRIORITY);
                }
                        catch (Exception e)
                        {
                                // esta excepção ocorre quando o servidor não está a
correr
                                // pode ser ignorado
                }
                }
            }
            catch (Exception ee)
            {
                        // erro, não é possível iniciar
                String args[] = new String[1];
                args[0] = ee.toString();
                processMessage("InitializeFailed",args);
            }
      }


  /**
   * Extensão do processMessage
   * @param comando Comando a executar
   * @param obj Lista de argumentos
   */
      public void processMessage(String comando,Object obj)
      {
            String[] args = (String[])obj;

            if(comando.equals("UnsupportedType"))
```

```
            {
                  System.out.println(comando + " " + args[0]);
          }
            else if(comando.equals("InitializeFailed"))
            {
                  System.out.println(comando + " " + args[0]);
                  endAction();
            }
            else if(comando.equals("Quit"))
            {
                  endAction();
            }
      }


  /**
   * Obtem o objecto existente em MessageQueue e converte-o
   * para um objecto KQML.
   * Depois executa a acção apropriada de acordo com o comando.
   * @param kqml mensagem KQMLmessage recebida
   * @return true se acção bem sucedida, false em caso contrário
   */
      protected boolean Act(Object o)
      {
            try
            {
                  KQMLmessage kqml = new KQMLmessage((String)o);

                  String perf = kqml.getValue("performative");
                  int emissor = (new Integer(kqml.getValue("sender"))).intValue();
                  int receptor = (new
Integer(kqml.getValue("receiver").substring(1))).intValue();
                  String msg = kqml.getValue("content");

                        // executa a acção
                  cidade.semaforo[receptor].processaMsg(emissor, msg, perf);

                  if(perf.equals("disconnected"))
                  {
                        String args[] = new String[1];
                        args[0] = kqml.getValue("sender");
                        _connections.removeConnection(args[0]);
                  }
            }
            catch (ParseException e)
            {
                  return false;
            }

            return true;
  }

  /**
       * Envia uma mensagem para um semáforo
       * @param emissor Identificação do semáforo que emite a mensagem
       * @param receptor Identificação do semáforo vai receber a mensagem
       * @param msg A mensagem a enviar
       * @param tipo O tipo da mensagem a enviar
       */
      public void enviaMsg(int emissor, String receptor, String msg, String tipo)
      {
            try
            {
                  KQMLmessage kqml = new KQMLmessage();
                  kqml.addFieldValuePair("performative", tipo);
                  kqml.addFieldValuePair("content", msg);
```

```
                    kqml.addFieldValuePair("sender", emissor + "");
                    kqml.addFieldValuePair("receiver", receptor);

                    System.out.println(kqml.getSendString());
                    sendMessage(kqml);
             }
             catch(Exception pe)
             {
                    System.out.println(pe);
             }
      }
}
```