Universidade do Porto

Faculdade de Engenharia

FEUP

T02 - Tráfego numa Cidade

Relatório Final

Agentes e Inteligência Artificial Distribuída
4º ano do Mestrado Integrado em Engenharia Informática e Computação

Elementos do grupo (T02_3):

Catarina Ramos - up201406219 - up201406219@fe.up.pt
Inês Gomes - up201405778 - up201405778@fe.up.pt
Mário Fernandes - up201201705 - ec12143@fe.up.pt

10 de Dezembro de 2017

Índice

1. Objetivo

1.1. Descrição do cenário

O trabalho “Tráfego numa cidade” consiste na simulação de um ambiente rodoviário constituído por ruas com pontos de cruzamento com semáforos de temporização fixa. Cada rua pode ter um ou dois sentidos e são utilizadas pelos automóveis. Os automóveis têm um ponto de partida e um ponto de destino, devendo respeitar as leis das direções das estradas, semáforos e não ultrapassarem outros veículos. Os automóveis devem tomar decisões de acordo com a informação que lhes chega através da perceção visual, comunicação com outros automóveis e via rádio.

Nesta simulação existem dois tipos de automóveis: aleatórios (vermelhos) e monitorizado (azul). Os automóveis aleatórios têm origem e destino aleatórios e surgem de acordo com a hora do dia (em horas de maior tráfego existem mais carros). O automóvel azul tem a origem, destino e modo escolhidos pelo utilizador (mais detalhes sobre a utilização poderão ser encontrados no Apêndice).

1.2. Objetivos do trabalho

O objetivo principal deste trabalho é construir a simulação anteriormente descrita com a ajuda das ferramentas Repast Simphony e SAJas. Esta simulação resultará num aglomerado de carros que param em semáforos, podendo causar trânsito. O objetivo será verificar que o carro monitorizado (azul), se desloca de acordo com as informações recebidas (perceção visual, comunicação com outros carros e rádio) optando por caminhos mais eficientes para chegar ao seu destino.

2. Especificação

a. Agentes

i. Automóvel

Arquitetura

O automóvel é o nosso principal agente. Na nossa simulação usamos três tipos de automóveis derivados da class *Car.* O primeiro é o *MonitoredCarAgent* que gera um carro de cor azul e que tem origem e destino definidos pelo utilizador nos parâmetros do repast. O segundo e o terceiro são o *CarNoneLearning* (cor vermelha) e o *CarShortLearning* (cor

preta). Ambos têm origem e destino aleatórios, sendo a maior diferença o seu comportamento. Enquanto o *CarNoneLearning* tem conhecimento total da cidade, o *CarShortLearning* começa sem qualquer conhecimento e avança de acordo com as perceções visuais ou pela comunicação com outros carros. Ambos aplicam o algoritmo A*.

Um agente do tipo Car tem ainda um modo de aprendizagem. Este modo pode ser escolhido entre os 4 seguintes:

Tipo de Aprendizagem	Descrição
SHORT_LEARNING	Aprende a cidade conforme avança e aplica o A* para tentar encontrar o destino. Caso não tenha sucesso, tenta explorar ruas não visitadas.
LEARNING	Aprende a cidade enquanto preenche a tabela dos valores de qualidade do algoritmo QLearning. Também usa o A* para encontrar ruas ainda não visitadas.
APPLYING	Conhece toda a cidade e usa os valores de qualidade da tabela aplicando assim o QLearning para chegar ao seu destino. Estes valores foram preenchidos previamente quando se encontrava ainda no modo LEARNING.
NONE	Conhece toda a cidade e usa o algoritmo A* para chegar ao seu destino

Um *Car* é constituído por:

- posição atual no mapa (*Point* e *Road*);
- última interseção (*Intersection*) pela qual passou;
- destino no mapa (*Point* e nome da rua);
- modo de aprendizagem escolhido para esse carro (short, none, learning ou applying);
- QLearning → objeto que contém todas as informações para conseguir aplicar este algoritmo;
- Knowledge (CarSerializable) → todo o conhecimento que deve ser guardado aquando o término do programa. Assim é possível utilizar o mesmo agente em execuções do programa diferentes;
- Journey → sequência de ruas que o carro deve seguir para chegar a um certo destino.
- Space → Espaço em que o automóvel é representado tendo em conta a sua posição;
- Algumas estatísticas para ajudar a interpretar os resultados (tempo em segundos que demorou a chegar ao seu destino, número de mensagens enviadas e número de vezes que recalcula a sua rota).

É possível verificar a estrutura da classe *Car* no diagrama UML do package *agents* situado no apêndice do presente relatório.

Comportamentos e Estratégias

- Ask Directions

Este comportamento é executado de cinco em cinco segundos e só é introduzido no carro se o modo de aprendizagem deste for *SHORT_LEARNING* ou *LEARNING.* O objectivo principal deste comportamento é pedir informações a outros carros sobre qual o nome da rua de destino ou qual o caminho para chegar ao destino de forma a alargar o seu conhecimento.

Os pormenores sobre o tipo de mensagens e o significado do seu conteúdo encontram-se no ponto Protocolos de Interacção (ponto 2.b).

A imagem abaixo apresentada representa o processo deste comportamento.

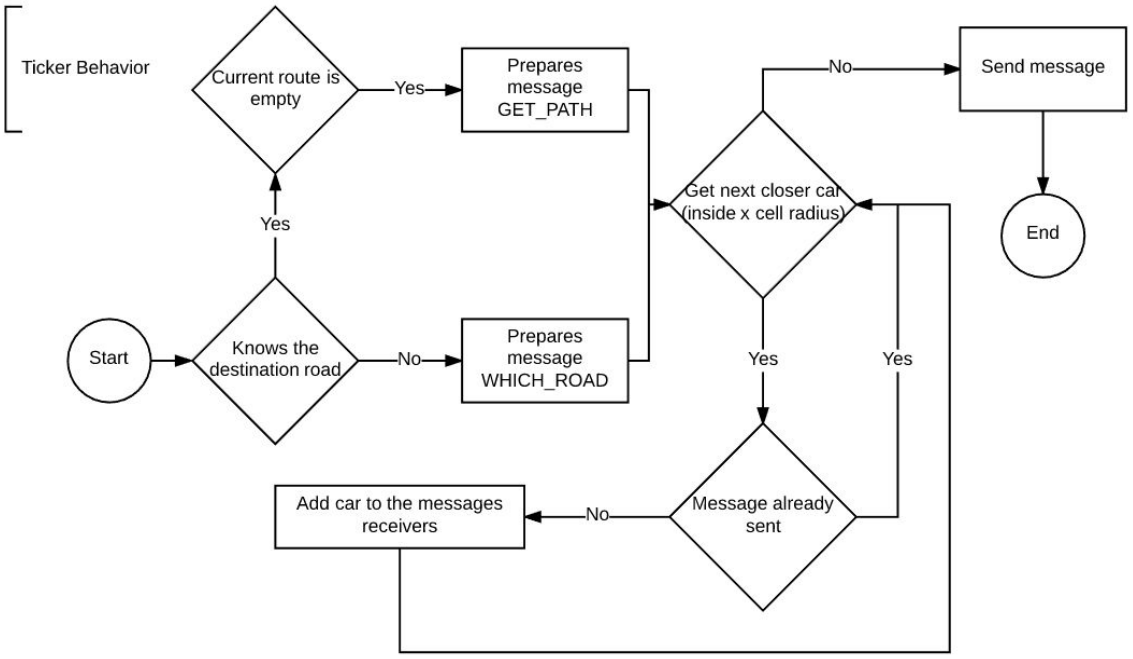

Figura 1 - Máquina de estados do comportamento AskDirections

- Car Messages Receiver

Este comportamento é cíclico e tem como principal objetivo tratar da recepção de mensagens.

A imagem abaixo apresentada representa o processo deste comportamento assim como o tratamento das diversas mensagens recebidas.

Os pormenores sobre o tipo de mensagens e o significado do seu conteúdo encontram-se no ponto Protocolos de Interacção (ponto 2.b).

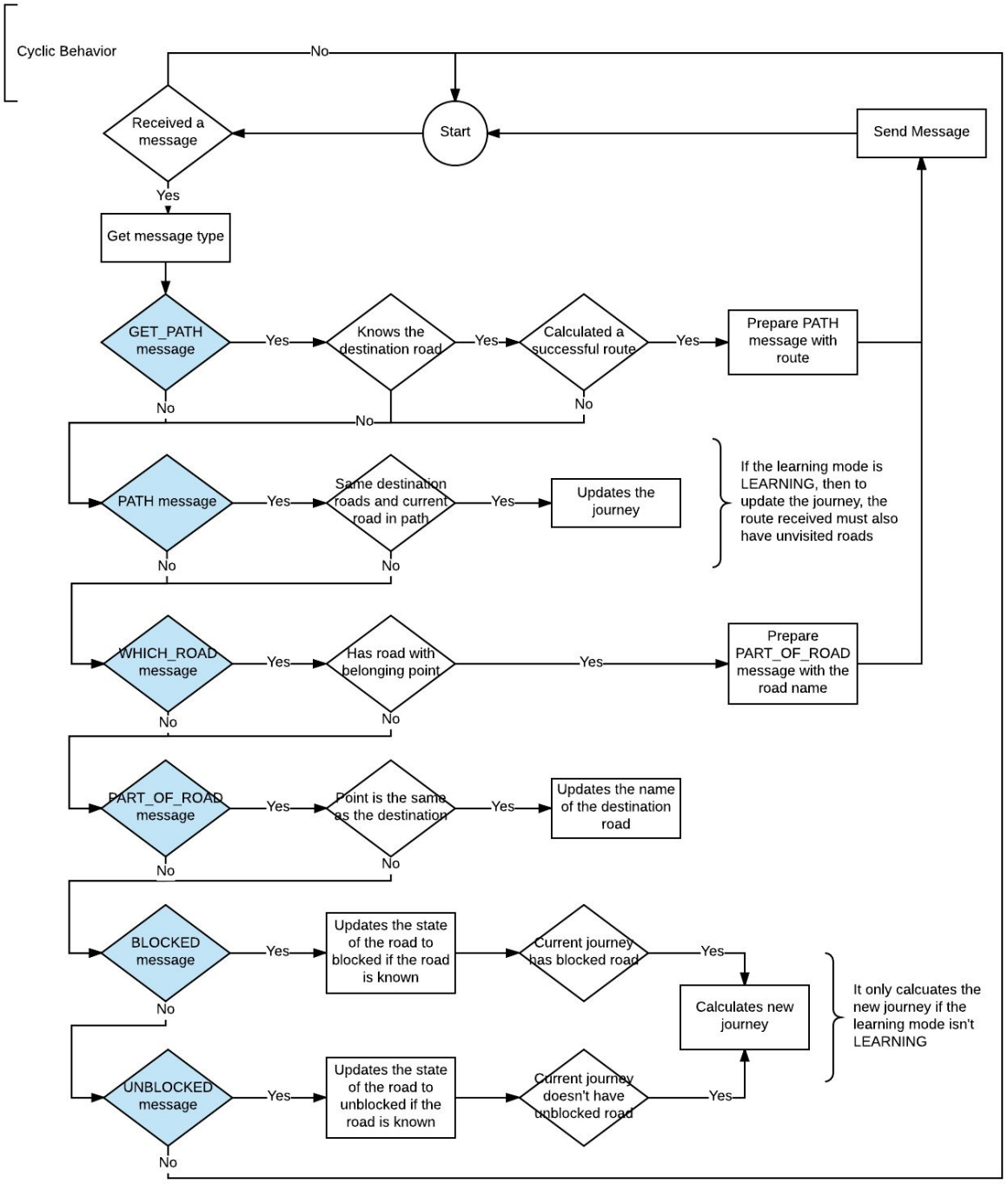

Figura 2 - Máquina de estados do comportamento CarMessagesReceiver

- Learn Map

Este comportamento é cíclico e tem como principal objetivo a actualização do conhecimento que o carro tem sobre a estrutura da cidade (Classe CityMap). Este comportamento é apenas introduzido para agentes que necessitam de ser treinados e que não têm um conhecimento total sobre a cidade, como é o caso dos agentes com aprendizagem do tipo *SHORT_LEARNING* e *LEARNING*.

A imagem abaixo apresentada representa o processo deste comportamento e as estratégias tomadas de forma a introduzir novo conhecimento no agente.

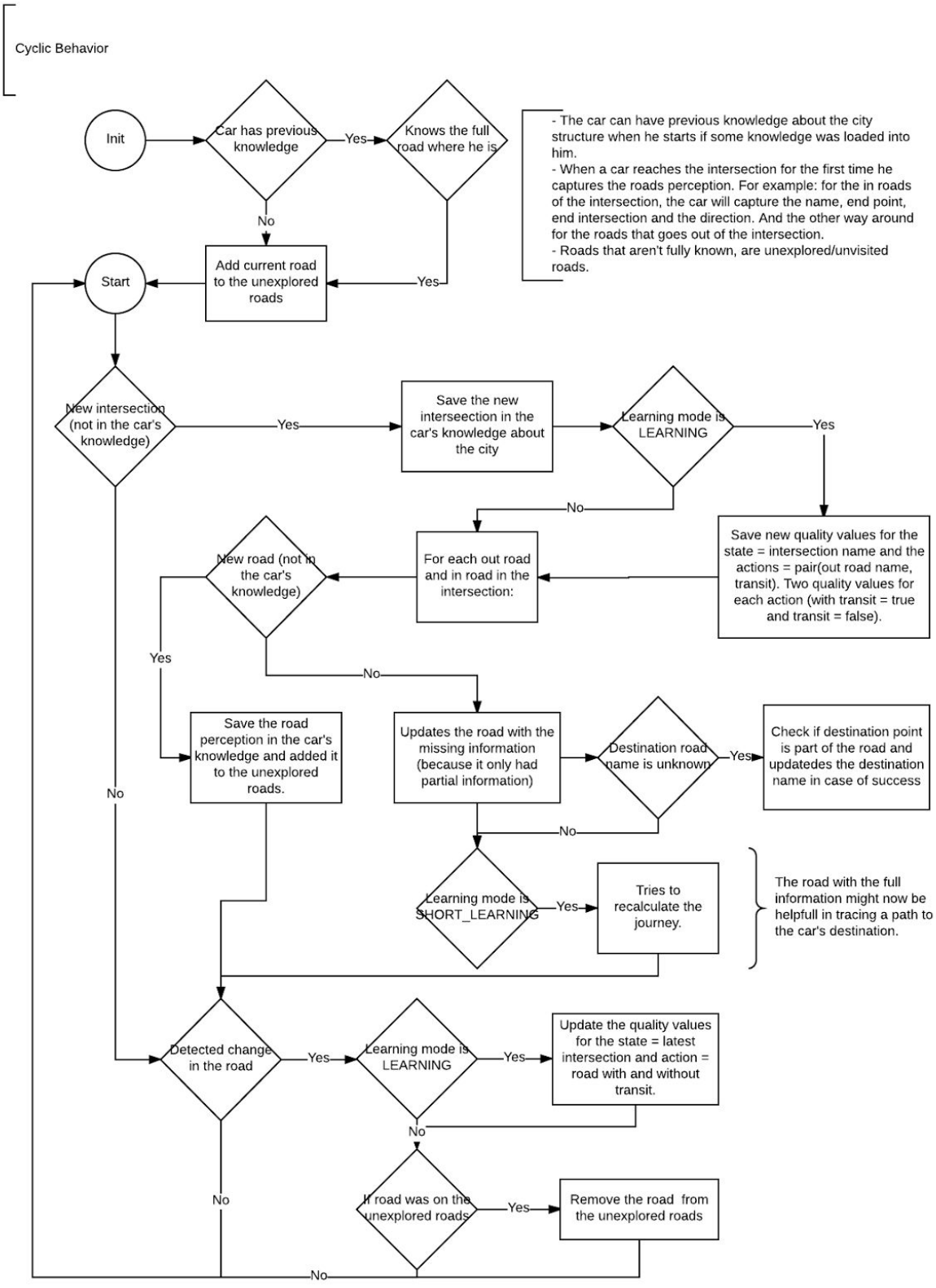

Figura 3 - Máquina de estados do comportamento LearnMap

- Car Movement

Este comportamento é executado a cada 300 milissegundos e tem como principal objetivo efetuar o movimento do carro respeitando outros carros, as direcções das ruas, os semáforos e a rota definida.

A imagem abaixo apresentada representa o processo deste comportamento assim como os vários módulos de decisão (estratégias).

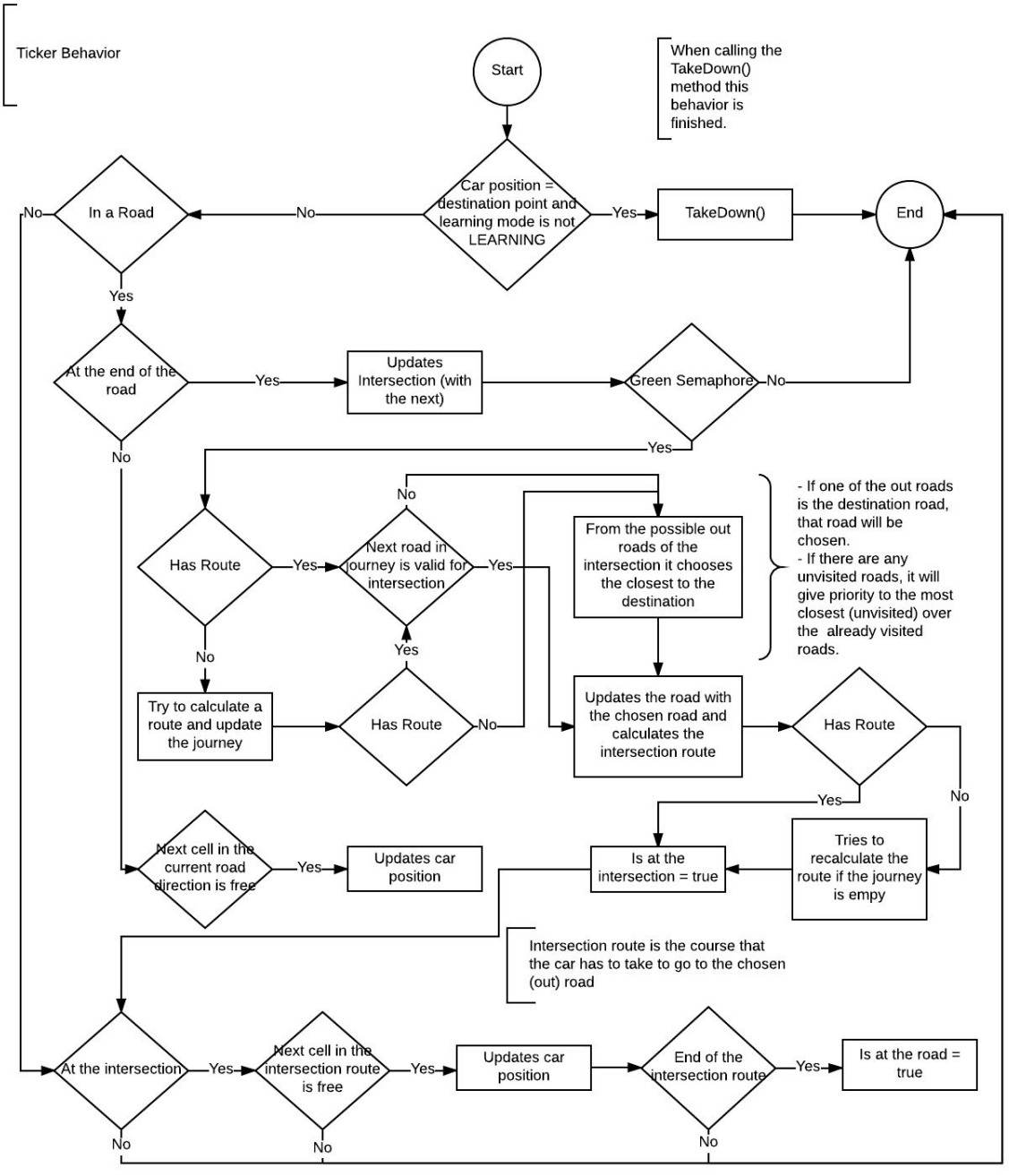

Figura 4 - Máquina de estados do comportamento CarMovement

- Cálculo e actualização da rota

A estratégia para o cálculo e actualização da rota do carro varia tendo em conta o seu modo de aprendizagem. A imagem abaixo apresentada representa o processo de cálculo e atualização do atributo *journey.*

A imagem abaixo apresentada representa o tratamento do cálculo e atualização da rota do carro (por ele próprio).

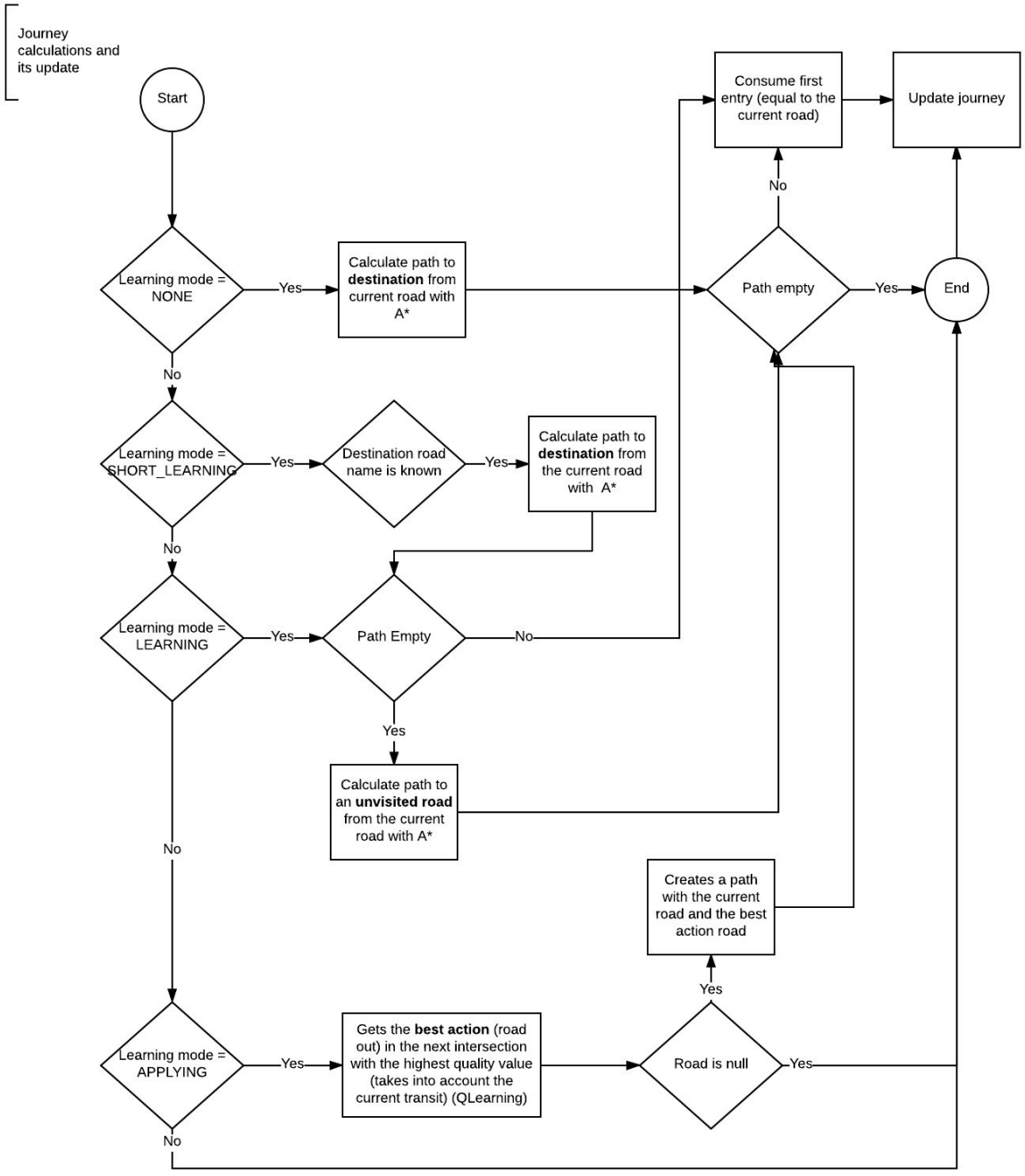

Figura 5 - Máquina de estados para o cálculo e actualização da rota do automóvel

- A* e QLearning (Penalização de trânsito)

A penalização em caso de trânsito representa o pior estado possível numa fila de espera em que o semáforo da rua actual é o último a ser mudado e traduz-se no número de células que um carro conseguiria andar (sem ter que parar) se não estivesse em trânsito.

$$TransitPenalty = (N_{OutRoadsAtEndIntersection} - 1) * (Time_{YellowLight} + Time_{GreenLight}) / Time_{Move1Cell}$$

- QLearning (Recompensa)

A recompensa utilizada no cálculo dos valores de qualidade para um estado (intersecção) e acção (rua de saída) *Road* é calculada pela equação abaixo descrita. A *Importance1* representa a importância da distância ao destino e a *Importance*2 representa a importância de tempo perdido a atravessar uma rua ou até em possível trânsito.

$$Reward = 1.5*(Space_{width}+Space_{height})-distance(Road_{EndIntersection}, Destination)*2*Importance_1-(Road_{length}+TransitPenalty)*2*Importance_2$$

ii. Rádio

Arquiteturas

O rádio (*Radio*) tem como objetivo informar todos os automóveis acerca do trânsito sentido numa certa rua. O rádio é informado do trânsito através dos monitores de rua.

É possível verificar a estrutura da classe *Radio* no diagrama UML do package *agents* situado no apêndice do presente relatório.

Comportamentos e Estratégias

- Radio Service

Este comportamento é acíclico e tem como principal objetivo lidar com as mensagens que lhe são transmitidas sobre mudanças de trânsito e informar todos os carros destas mudanças.

A imagem abaixo apresentada representa a estratégia do tratamento de mensagens recebidas.

Os pormenores sobre o tipo de mensagens e o significado do seu conteúdo encontram-se no ponto Protocolos de Interacção (ponto 2.b).

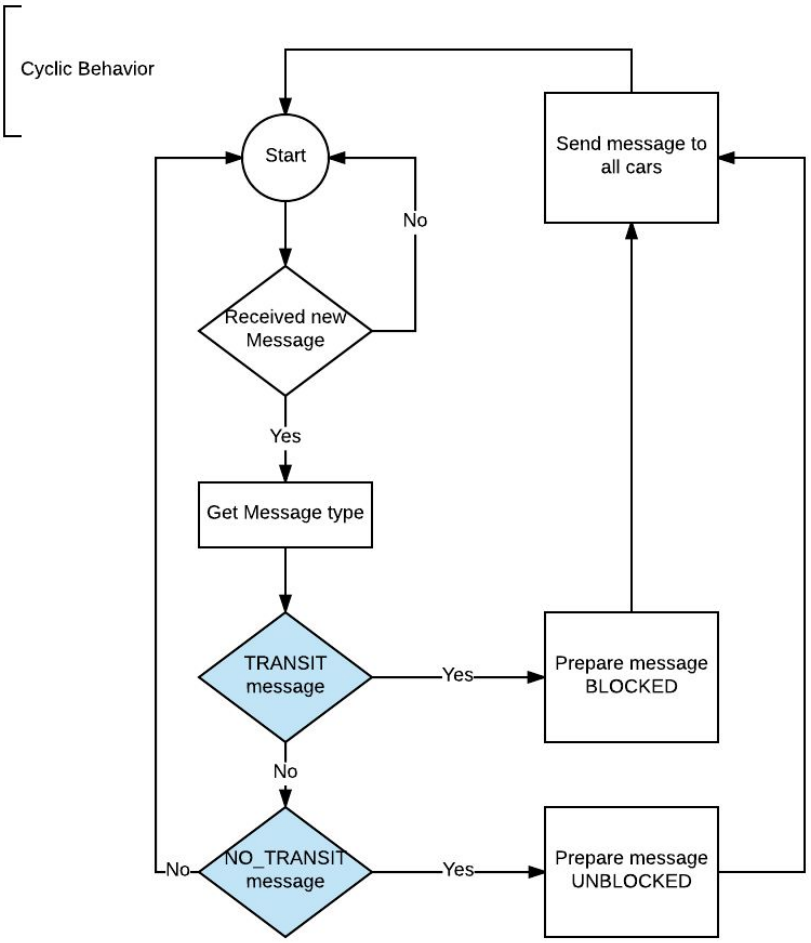

Figura 6 - Máquina de estados do comportamento Radio Service

iii. Gestor de Semáforos

Arquiteturas

O gestor de semáforos (*SemaphoreManager*) é um agente que tem como objetivo a sincronização dos diversos semáforos numa dada interseção, de modo a evitar que haja mais do que uma entrada com verde por cada interseção e consequentemente evitando colisões entre veículos.

A classe *SemaphoreManager* é constituída por:

- Um semáforo amarelo;
- Um semáforo verde;
- Um conjunto de semáforos vermelhos;
- Um booleano que indica se existe um semáforo verde nessa interseção;
- Space → Espaço em que o automóvel é representado tendo em conta a sua posição.

É possível verificar a estrutura da classe *SemaphoreManager* no diagrama UML do package *agents* situado no apêndice do presente relatório.

Comportamentos e Estratégias

- Switch Lights

Este comportamento é executado a cada segundo e tem como principal objetivo efetuar a mudança de luzes nos semáforos de uma intersecção. Como foi necessário a criação dos agentes *REPAST SemaphoreRed*, *SemaphoreGreen* e *SemaphoreYellow* (subclasses de *Semaphore*) para fazer a distinção visual, a alteração das luzes dos semáforos são, na realidade, a alteração das posições entre os semáforos vermelhos, verde e amarelo.

A imagem abaixo apresentada representa o processo e estratégia deste comportamento.

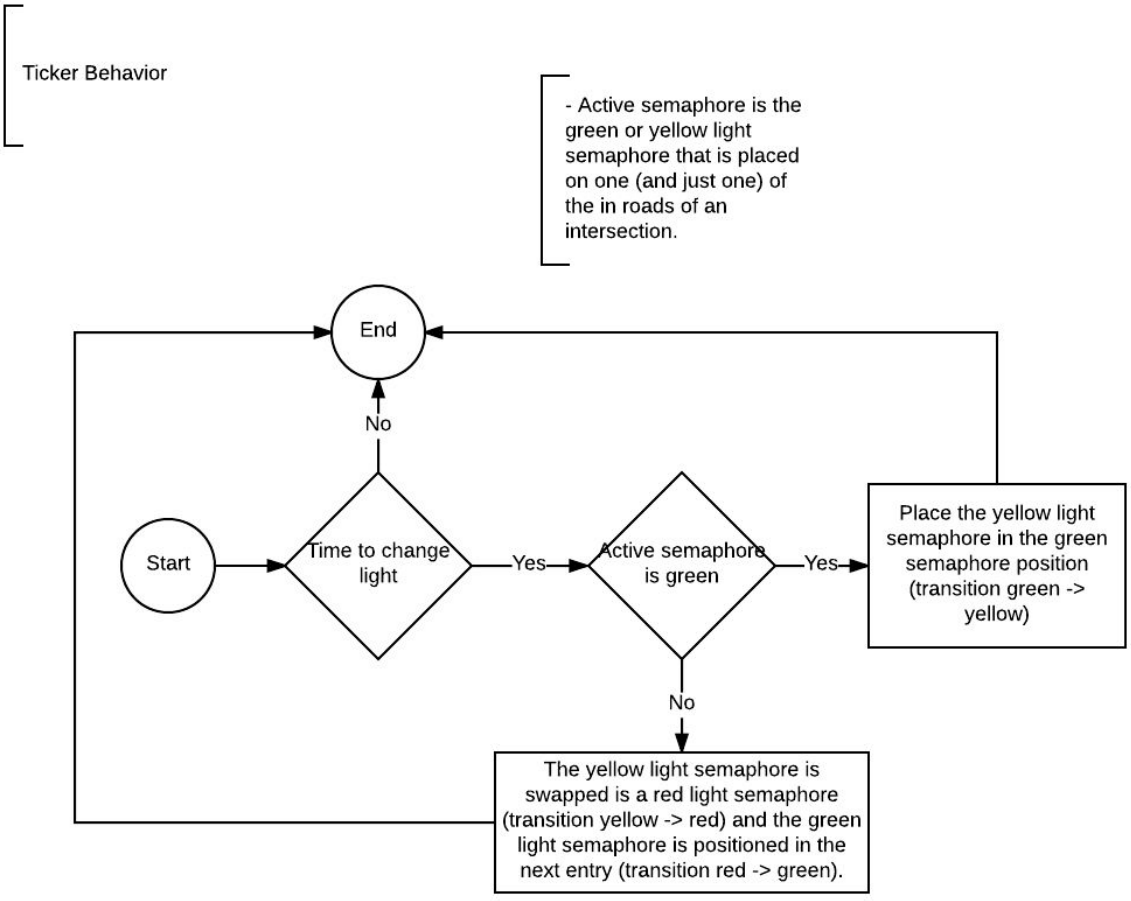

Figura 7 - Máquina de estados do comportamento SwitchLights

iv. Monitor de Rua

Arquiteturas

Os monitores de rua (*RoadMonitor*) têm como função analisar o trânsito na sua rua e informar o rádio em caso afirmativo. O rádio também é informado do momento em que a rua

está novamente sem tráfego. Para um monitor de rua saber se a sua rua tem trânsito efetua os cálculos apresentados na Figura 9 do presente relatório.

Um *RoadMonitor* é constituído por :

- Road → rua que está a monitorizar;
- Radio → rádio do mapa, para poder enviar as informações de trânsito;
- Space → espaço em que o automóvel é representado tendo em conta a sua posição.

É possível verificar a estrutura da class *RoadMonitor* no diagrama UML do package *agents* situado no apêndice do presente relatório.

Comportamentos e Estratégias

- Transit Monitorization

Este comportamento é executado a cada 1500 milissegundos e tem como principal objetivo a monitorização de trânsito na rua a que o monitor de rua é responsável e ver se existem alterações de trânsito. Caso existam, avisar o rádio.

Os pormenores sobre o tipo de mensagens e o significado do seu conteúdo encontram-se no ponto Protocolos de Interacção (ponto 2.b).

A imagem abaixo apresentada representa o processo deste comportamento.

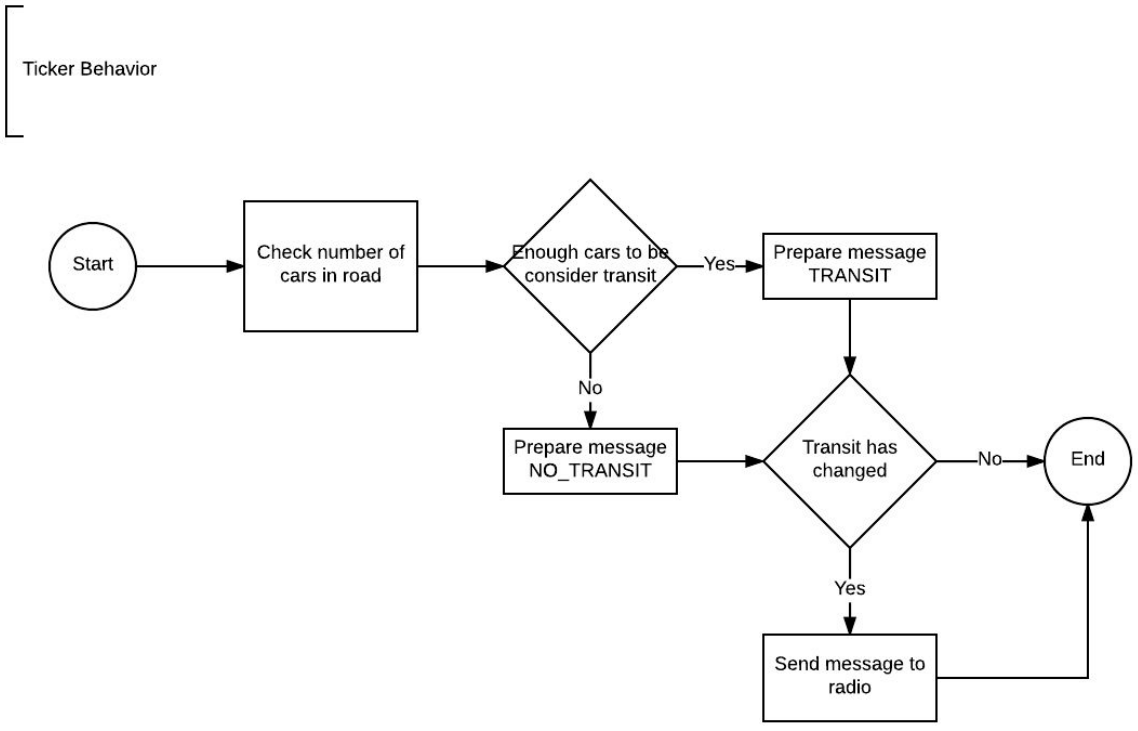

Figura 8 - Máquina de estados do comportamento TransitMonitorization

$$MaxCarsWithoutTransit_{road} = \begin{Bmatrix} length_{road} \leq 3 & 3 \\ length_{road} \leq 5 & length_{road} \\ length_{road} \leq 8 & length_{road} * 0.75 \\ length_{road} > 8 & length_{road} * 0.6 \end{Bmatrix}$$

Figura 9 - Máximo de automóveis permitidos numa rua sem ser considerado trânsito

v. Agentes Repast

Além dos agentes de JADE que representam o agente inteligente, temos os agentes Repast que apenas servem para visualização no mapa. Entre eles temos o semáforo e a cidade.

Um semáforo tem como objetivo regular o trânsito e pode ser uma das três subclasses *SemaphoreGreen*, *SemaphoreRed*, *SemaphoreYellow* de acordo com a cor do mesmo.

Os Agentes *Car* devem obedecer às regras impostas pelos semáforos, nomeadamente:

- Se for verde, pode seguir caminho;
- Se for amarelo ou vermelho deve parar nessa célula.

Um semáforo (*Semaphore*) é constituído por:

- Cor (Light) → Verde, Vermelho ou Amarelo;
- Posição no mapa (Point);
- Space → espaço em que o automóvel é representado tendo em conta a sua posição.

É possível verificar a estrutura da classe *Semaphore* no diagrama UML do package *agents* situado no apêndice do presente relatório.

A classe *City* tem como objetivo desenhar o mapa da cidade e representar a estrutura da mesma. Também é responsável pela geração dos semáforos nas interseções.

Uma cidade (*City*) é constituída por:

- estrutura da cidade (CityMap);
- dimensões da cidade para cálculos de posicionamento;
- Container → suporta os agentes do repast;
- Space → espaço em que o automóvel é representado tendo em conta a sua posição.

É possível verificar a estrutura da classe *City* no diagrama UML do package *agents* situado no apêndice do presente relatório.

b. Protocolos de Interação

A comunicação entre agentes é importante para a transmissão de informação sobre o ambiente.

Devido ao uso da biblioteca JADE, adotamos as normas estabelecidas pelo FIPA-ACL para a comunicação entre agentes, nomeadamente as mensagens do tipo INFORM. Para seguir este protocolo é necessário obedecer aos parâmetros das mensagens mostrados na figura seguinte.

Para este trabalho, foram definidas vários tipos de mensagens, tendo em conta o seu conteúdo.

Abaixo serão descritos alguns pormenores acerca das mensagens utilizadas sobre o seu conteúdo, o tipo de remetente, o tipo de receptor, as condições de envio e as reacções de recepção.

É relevante dizer que, no template e no exemplo apresentado de cada mensagem, as informações que fazem parte do seu conteúdo são separadas por um separador especial, no entanto, por questões de legibilidade, aqui serão separadas por um espaço.

Template	*TRANSIT Nome_da_Rua*
Exemplo	*TRANSIT RH1*
Remetente	Monitores de Rua
Receptor	Rádio
Condições de Envio	Quando a rua ultrapassa o número de automóveis máximos para não ser considerado trânsito numa rua com um determinado comprimento (*Fig x).*
Reacção de Recepção	Enviar uma mensagem do tipo *BLOCKED* para todos os automóveis no espaço.

Template	*NO_TRANSIT Nome_da_Rua*
Exemplo	*NO_TRANSIT RH1*
Remetente	Monitores de Rua
Receptor	Rádio
Condições de Envio	Quando a rua deixa de estar em trânsito (*Fig x).*
Reacção de Recepção	Enviar uma mensagem do tipo *UNBLOCKED* para todos os automóveis no espaço.

Template	*BLOCKED Nome_da_Rua*
Exemplo	*BLOCKED RH1*
Remetente	Rádio
Receptor	Automóvel
Condições de Envio	Quando a mensagem do tipo *TRANSIT* é recebida referente a uma determinada rua com o nome *Nome_da_Rua.*
Reacção de Recepção	Actualização no mapa da cidade (interno) que a rua com nome *Nome_da_Rua* está bloqueada (se for conhecida). Se a rota que o automóvel está a tomar para chegar ao seu destino passar por esta rua a rota é novamente recalculada. Nota: As ruas assinaladas como bloqueadas (em trânsito) têm uma penalização (peso extra) no cálculo do caminho mais curto. Esse peso extra é de 3 vezes mais o cumprimento da rua bloqueada.

Template	*UNBLOCKED Nome_da_Rua*
Exemplo	*UNBLOCKED RH1*
Remetente	Rádio
Receptor	Automóvel
Condições de Envio	Quando a mensagem do tipo *NO_TRANSIT* é recebida referente a uma determinada rua com o nome *Nome_da_Rua.*
Reacção de Recepção	Actualização no mapa da cidade (interno) que a rua com nome *Nome_da_Rua* está desbloqueada (se for conhecida). É calculada uma nova rota para chegar ao destino caso a rua não fizesse parte já da rota actual.

Template	*GET_PATH Nome_da_Rua Ponto_de_Destino*
Exemplo	*GET_PATH RH1 [2,19]*
Remetente	Automóvel
Receptor	Automóvel

Condições de Envio	O automóvel recetor tem que estar num raio de 2 células em torno do remetente. O remetente não tem qualquer nenhuma rota definida para chegar ao seu destino e sabe o nome da rua de destino. A mensagem só é enviada uma vez para cada recetor (por rua). Nota: Esta mensagem é enviada de x em x tempo. Caso o remetente mude de rua, o conteúdo da nova mensagem também muda, pelo que, pode enviar a mensagem para antigos receptores.
Reacção de Recepção	Cálculo da rota que tem como origem a rua de nome *Nome_da_Rua* e destino o ponto *Ponto_de_Destino*, tendo em conta o conhecimento que o automóvel tem da cidade (mapa interno). Se o automóvel tiver sucesso em calcular esta rota, então uma mensagem do tipo *PATH* é enviada para o remetente desta mensagem com a rota.

Template	*PATH Rua_de_Destino Nome_da_Rua1 Nome_da_Rua2 … Rua_de_Destino*
Exemplo	*PATH RH1 RV1 RH6 … RH1*
Remetente	Automóvel
Receptor	Automóvel
Condições de Envio	Quando uma mensagem do tipo *GET_PATH* é recebida e o cálculo da rota é bem sucedido.
Reacção de Recepção	Se a *Rua_de_Destino* contida na mensagem for igual à rua de destino do receptor então: 1. Procura pelo rua corrente no caminho contido na mensagem e começa a guardar o caminho para o destino a partir dessa posição. 2. Actualiza a rota para chegar ao destino se tiver encontrado a rua corrente no caminho contido e se: a. O receptor estiver em modo de aprendizagem do tipo *SHORT_LEARNING.* b. O receptor estiver em modo de aprendizagem do tipo *LEARNING* e o caminho para o destino tiver alguma rua não visitada antes. Nota: Os nomes das ruas estão por ordem das ruas que o automóvel tem que seguir para alcançar o seu destino.

Template	*WHICH_ROAD Ponto*
Exemplo	*WHICH_ROAD [1,59]*
Remetente	Automóvel
Receptor	Automóvel
Condições de Envio	O automóvel recetor tem que estar num raio de 2 células em torno do remetente. O remetente não sabe o nome da rua de destino. A mensagem só é enviada uma vez para cada recetor (por rua). Nota: Esta mensagem é enviada de x em x tempo. Caso o remetente mude de rua, o conteúdo da nova mensagem também muda, pelo que, pode enviar a mensagem para antigos receptores.
Reacção de Recepção	Se o ponto for parte de alguma rua que o automóvel recetor tem em seu conhecimento, então uma mensagem do tipo PART_OF_ROAD é enviada para o emissor da mensagem com a informação sobre a rua.

Template	*PART_OF_ROAD Ponto Nome_da_Rua*
Exemplo	*PART_OF_ROAD [1,59] RH20*
Remetente	Automóvel
Receptor	Automóvel
Condições de Envio	Quando uma mensagem do tipo *WHICH_ROAD* é recebida e o cálculo da rua a que pertence é bem sucedido.
Reacção de Recepção	Se o ponto contido na mensagem for o ponto de destino, então o nome da rua de destino é actualizada.

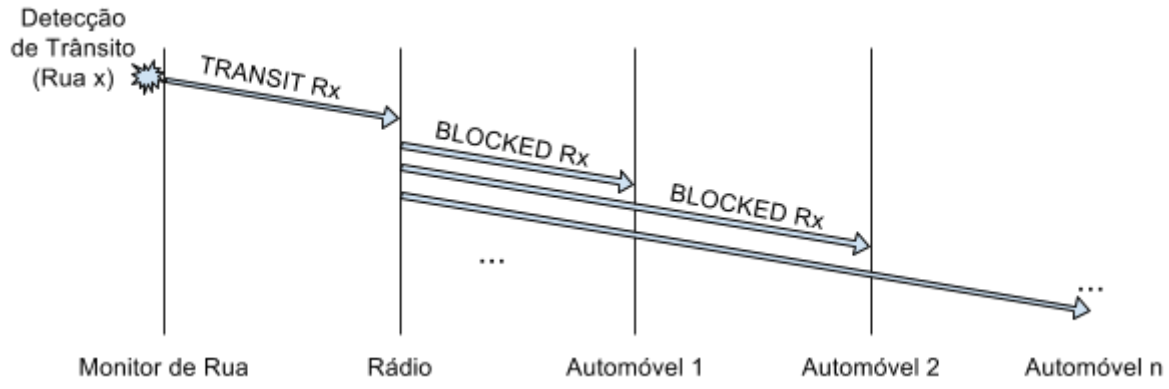

Figura 10 - Exemplo de transmissão de mensagens TRANSIT e BLOCKED.
Rx - The road name where there was a change in the transit flow to “transit”.

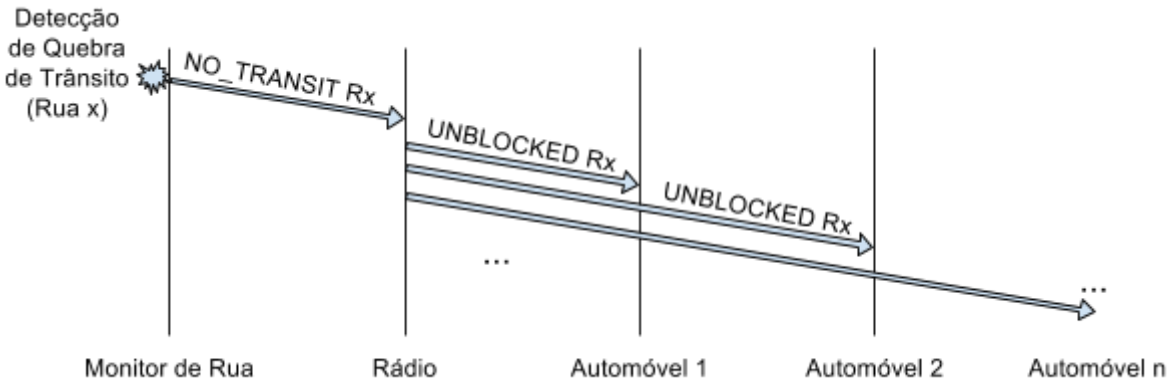

Figura 10 - Exemplo de transmissão de mensagens NO_TRANSIT e UNBLOCKED.
Rx - The road name where there was a change in the transit flow to “no transit”.

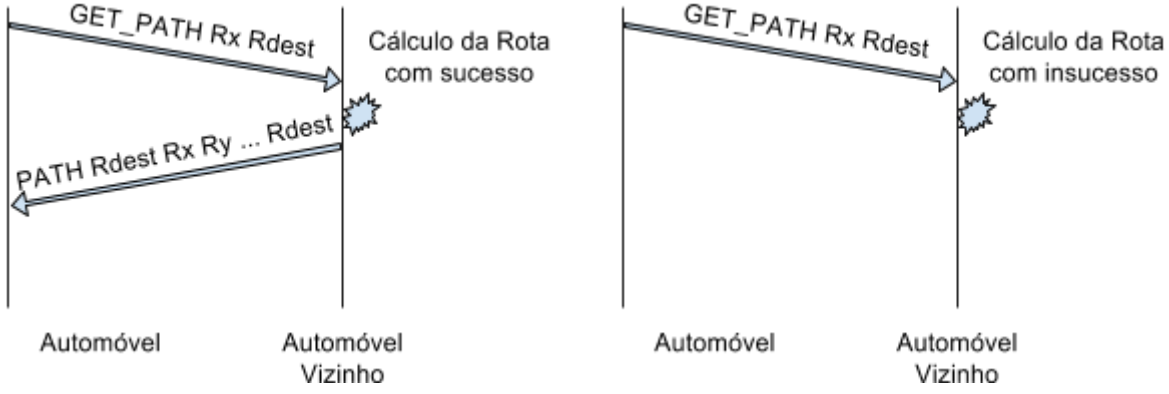

Figura 11 - Exemplo de transmissão de mensagens GET_PATH e PATH.
Rx - The name of the start road; Rdest - The name of the destination road; Rx,Ry - Between roads names.

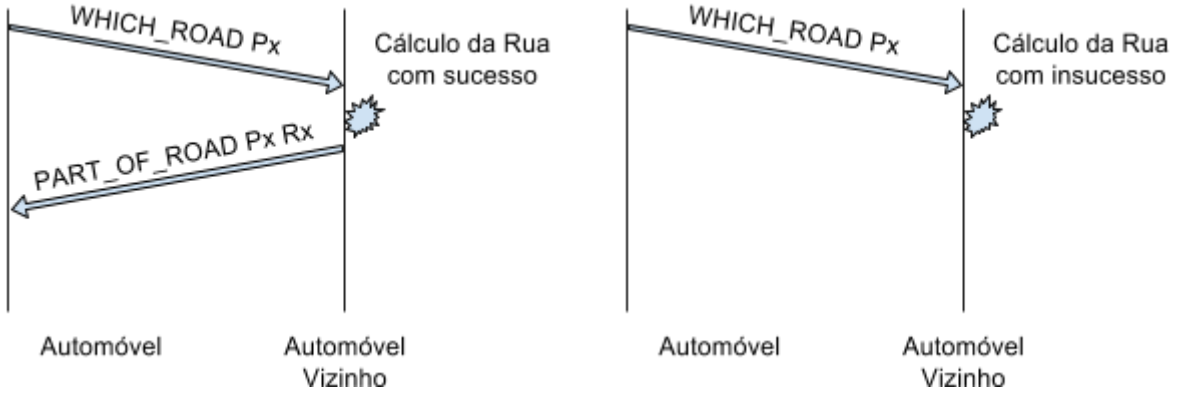

Figura 12 - Exemplo de transmissão de mensagens WHICH_ROAD e PART_OF_ROAD. Px - a point; Rx - the road name that contains Px.

3. Desenvolvimento

a. Plataforma

Para este trabalho serão usadas duas ferramentas: Repast Simphony 2.5 + SAJaS.

O *Repast* (*REcursive Porus Agent Simulation Toolkit*) *Simphony* é uma plataforma que tem como principal objetivo a simulação de sistemas multi-agente (MAS). Para este trabalho vamos usar Repast para a criação do mapa, automóveis e semáforos.

O SAJaS é uma biblioteca que tem como principal objetivo permitir a integração de agentes JADE com plataformas de simulação MAS. O uso da biblioteca SAJaS obriga ao uso da biblioteca JADE (Java Agent DEvelopment Framework). Esta biblioteca permite aceder às inúmeras vantagens do desenvolvimento de agentes com esta framework, incluindo a possibilidade de usar os standards propostos por FIPA para a comunicação entre agentes.

Este trabalho foi realizado e testado no SO Windows10 e no IDE Eclipse Oxygen.

b. Estrutura

O presente trabalho está estruturado em 6 packages: *agents, algorithms, behaviours, cityStructure, cityTraffic e resources*.

CityTraffic

Este package contém a class *CityTrafficBuilder* que contém o builder do repast que inicializa o programa. Aqui são interpretados os parâmetros do repast e é construído o mapa com os respectivos semáforos e automóveis. É possível ver dois exemplos de mapas gerados na secção de Resultados do presente relatório.

É possível observar o Diagrama UML deste package na secção Diagramas UML.

Agents

O package *agents* contém todos os agentes de repast e jade, como descrito na secção 2.a do presente relatório. Além destes, ainda contém a classe *CarSerializale* que contém todo o conhecimento do carro que deve persistir após o término do programa. Aqui são guardadas informações como o conhecimento do mapa, os valores de qualidade do Qlearning e a posição de destino para o QLearning.

É possível observar o Diagrama UML deste package na secção Diagramas UML.

Behaviours

Este package contém todos os comportamentos associados ao agentes JADE. Estes comportamentos também podem ser analisados na secção 2.a do presente relatório.

É possível observar o Diagrama UML deste package na secção Diagramas UML.

CityStructure

O package *CityStructure* foi criado para auxiliar o mapa e a sua construção. O objetivo desta estrutura é assemelhar-se a um grafo, sendo as *Intersections* os nós e as *Roads* os vértices e o CityMap a própria estrutura.

Uma *Road* é constituída por um ponto de início ligado a uma determinada interseção e um ponto de fim ligado a outra. A direcção da rua e o seu cumprimento são calculados automaticamente.

Uma *Intersection* delimitada por uma determinada área tendo um determinado número de pontos de entrada (de ruas) calculado de forma automática. A inserção de ruas é feita quando se associa uma interseção a uma rua num determinado ponto de entrada quando a rua é criada. É possível calcular a rota de pontos necessários ao automóvel passar dada uma rua de entrada e uma rua de saída para sair da intersecção.

Uma Intersection pode ser dividida em duas subclasses diferentes: *SimpleIntersection* e *ComplexIntersection*. A primeira tem apenas, no máximo, 4 pontos de entrada possíveis (um em cada direção). A segunda tem mais de 4 pontos, sendo necessário definir o conjunto de pontos que cobre a área da interseção.

É possível observar o Diagrama UML deste package na secção Diagramas UML.

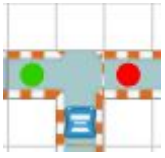

Figura 13 : exemplo de interseção simples

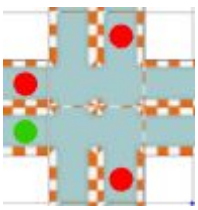

Figura 14: exemplo de interseção complexa

Resources

Este package tem como principal objetivo aglomerar várias classes auxiliares ao desenvolvimento do programa.

Um *Point* representa uma coordenada (x,y) no mapa.

MessageResources é a classe responsável pela criação e descodificação das mensagens enviadas e recebidas entre agentes.

SpaceResources é a classe responsável pela manipulação de objetos no espaço.

Resources é a classe responsável por todos os outros métodos e constantes relacionados com várias classes do projeto.

A classe *Debug* é a responsável pela monitorização de todas as mensagens apresentadas na consola. De acordo com o modo escolhido as mensagens podem variar:

- debugCarMessages (True ou False) quando esta variável está no estado "True", todas as mensagens emitidas pelo carro monitorizado serão visíveis na consola;
- debugQLearning (True ou False): quando esta variável está no estado "True", toda a informação principal referente ao carro monitorizado e à qualidade será visível na consola;
- debugLearningMode (True ou False): quando esta variável está no estado "True", irá ser possível ver o estado de aprendizagem, podendo este variar entre "SHORT_LEARNING", "LEARNING", "APPLYING", "NONE";
- debugRoadMonitor (True ou False): quando esta variável está no estado "True", todas as mensagens emitidas pelos monitores de rua serão visíveis na consola;
- debugRadio (True ou False): quando esta variável está no estado "True", todas as mensagens emitidas pelo rádio serão visíveis na consola;
- debugJourneyUpdate (True ou False): Quando esta variável está no estado "True", a informação referente à viagem do carro monitorizado será visível na consola;
- debugDiscoveries (True ou False): quando esta variável está no estado "True", a informação referente às descobertas do carro monitorizado, nomeadamente novas ruas ou interseções, será visível na consola;
- debugUnvisitedJourney (True ou False): quando esta variável está no estado "True", sempre que o carro monitorizado optar por uma rua desconhecida como o seu próximo destino, será possível visualizar essa escolha na consola.

É possível observar o Diagrama UML deste package na secção Diagramas UML.

Algorithms

Para a execução deste projeto são usados dois algoritmos: A* e QLearning. Estes algoritmos têm como função calcular o melhor caminho para os agentes chegarem ao seu destino.

Nota: O custo em caso de trânsito está explicíto no ponto *"A* e QLearning (Penalização de trânsito)"* (secção 2.a.i Comportamentos e Estratégias).

- A*

O algoritmo A* tem como objetivo calcular o caminho mais curto entre a rua atual e o seu destino, seja ele uma rua ou uma interseção. Para tal, é observado o grafo e são calculados os custos, escolhendo o caminho que tem o custo menor para chegar ao destino. Depois de calculado é construído um caminho, ou seja sequência de ruas, que o agente deve seguir para chegar ao destino.

A heurística utilizada para determinar o custo de uma determinada rua r1 até à rua destino r2, é a distância entre a intersecção final de r1 e a intersecção de início de r2. Quando existe trânsito é calculada uma penalização, como já foi descrito na secção A* e QLearning (Penalização de trânsito) nos Comportamentos e Estratégias do Agente Automóvel.

- QLearning

O algoritmo QLearning é uma técnica de aprendizagem por reforço para encontrar a melhor acção dado um determinado estado.
Neste trabalho, os estados são as intersecções e as acções são as possíveis ruas (com e sem trânsito) de saída relativamente a uma intersecção.

$$State(Intersection_i)$$

$$Action(OutRoad_j, Transit)$$

Para cada intersecção existente há garantidamente um estado dessa intersecção.

$$\forall Intersectioni \exists State(i)$$

Para todas as intersecções existe pelo pelo menos uma rua de saída, e para cada uma, existem dois tipos de acção possível: com e sem trânsito.

$$\forall Intersectioni \exists RoadOutj \therefore Action(RoadOut_i, Transit = true) \wedge Action(RoadOut_i, Transit = false))$$

O cálculo da recompensa está destino na secção QLearning (Recompensa) nos Comportamentos e Estratégias do Agente Automóvel.

É possível observar o Diagrama UML deste package na secção Diagramas UML.

c. Outros Detalhes

Para evitar mapas fixos com o mesmo número de carros e conhecimento, e para promover a maior aprendizagem dos agentes foi criado um mapa com geração aleatória de carros. Como já foi descrito anteriormente na secção sobre o agente automóvel do presente

relatório, existem três tipos de carros. O carro monitorizado é controlado pelo utilizador. Os restantes carros são gerados em posições aleatórias com uma probabilidade aleatória de serem do tipo *CarNoneLearning* ou *CarShortLearning*. Por exemplo, no início do programa é gerada uma probabilidade aleatória de 70%. Isto significa que 70% dos carros gerados serão do tipo *CarShortLearning* e os restantes 30% do tipo *CarNoneLearning*.

O número de carros gerados também é aleatório consoante a hora do dia escolhida. As horas do dia estão divididas em áreas de muito tráfego, moderado ou fraco. Assim é escolhido um número de carros aleatório dentro de um intervalo configurado para a área de tráfego em que essa hora se insere.

Para orientar os carros na cidade são introduzidos semáforos no fim de cada estrada que convirja para uma interseção em que o número de entradas na mesma é maior do que 1. Aquando o load do mapa da cidade, ou seja, na criação da sua estrutura, são lidas as direções das ruas. A criação inicial dos semáforos, força a que haja apenas um verde por conjunto de sinais. O resto é tratado pelo gestor de semáforos.

4. Experiências

a. Experiência A*

A primeira experiência consiste em analisar a evolução média dos tempos, número de mensagens enviadas e número de caminhos recalculados de um agente que aplica o A* para chegar ao seu destino. Esta experiência analisa 3 agentes diferentes e regista os seus tempos durante 3 iterações. Na primeira iteração o agente não conhece o mundo, nas seguintes conhece o mundo registado nas iterações anteriores. A origem e destino são iguais para todos os agentes, bem como o mapa.

Esta experiência será repetida 3 vezes para 3 intensidades de tráfego diferentes.

Experiência 1

Parâmetros

- Mapa : Big
- Origem : [60,32]
- Destino : [28,24]
- Hora : 1h (pouco tráfego)

Resultados

	Iteração 1				**Iteração 2**				**Iteração 3**			
	t	**g**	**w**	**n**	**t**	**g**	**w**	**n**	**t**	**g**	**w**	**n**

Agente 1	204	1	1	3	57	0	0	1	57	0	0	1
Agente 2	88	3	1	2	69	0	0	1	69	0	0	1
Agente 3	233	1	3	4	57	0	0	2	57	0	0	1
Médias	175	1.67	1.67	3	61	0	0	1.33	61	0	0	1

Legenda:

- t - tempo desde que o carro entrou no mapa até que chegou ao seu destino
- g - número de mensagens enviadas do tipo GetPath
- w - número de mensagens enviadas do tipo WhichRoad
- n - número de vezes que o caminho foi recalculado

Conclusões

Por esta experiência é possível ver que o carro diminui o tempo de chegada ao destino ao longo das iterações, pois apenas da primeira vez, desconhece o caminho. A mesma razão leva a interpretar os valores das mensagens enviadas, que passam de uma média de 1.67 para 0 já na segunda iteração. O número de vezes que necessita de recalcular o caminho também diminui com o número de vezes que percorre o mapa.

Numa primeira iteração, devido ao pouco trânsito, o tempo até chegar ao destino é elevado pelo facto de, como ele desconhece completamente a cidade, também não existiram muitos carros que lhe possam transmitir informações úteis.

Experiência 2

Parâmetros

- Mapa : Big
- Origem : [60,32]
- Destino : [28,24]
- Hora : 10h (tráfego moderado)

Resultados

	Iteração 1				**Iteração 2**				**Iteração 3**			
	t	**g**	**w**	**n**	**t**	**g**	**w**	**n**	**t**	**g**	**w**	**n**
Agente 1	87	2	1	2	69	0	0	4	69	0	0	3
Agente 2	69	1	1	1	93	1	4	6	69	0	0	8
Agente 3	95	1	3	2	87	0	0	4	69	0	0	2
Médias	83.67	1.33	1.67	1.67	83	0.33	1.33	4.67	69	0	0	4.33

Legenda:

- t - tempo desde que o carro entrou no mapa até que chegou ao seu destino
- g - número de mensagens enviadas do tipo GetPath
- w - número de mensagens enviadas do tipo WhichRoad
- n - número de vezes que o caminho foi recalculado

Conclusões

Para esta experiência podemos verificar que ao longo do tempo os valores do tempo e das mensagens enviadas, diminui, no entanto o valor do número de caminhos recalculados aumenta. Isto acontece devido ao trânsito sentido. Ao longo do tempo o carro recebe mensagens do rádio com as ruas bloqueadas e o carro deve tomar a decisão de seguir pela rua com trânsito, ou recalcular o seu caminho. Como na primeira experiência ele não conhece as ruas, segue maioritariamente os indicações de outros carros, enquanto nas iterações seguintes faz escolhas a partir do trânsito.

Experiência 3

Parâmetros

- Mapa : Big
- Origem : [60,32]
- Destino : [28,24]
- Hora : 12h (muito tráfego)

Resultados

	Iteração 1				**Iteração 2**				**Iteração 3**			
	t	**g**	**w**	**n**	**t**	**g**	**w**	**n**	**t**	**g**	**w**	**n**
Agente 1	91	1	1	1	71	1	1	1	172	1	1	5
Agente 2	87	2	1	1	75	1	1	1	108	1	1	2
Agente 3	71	1	2	1	58	1	1	1	124	0	1	6
Médias	83	1.33	1.33	1	68	1	1	1	134.7	0.67	1	4.33

Legenda:

- t - tempo desde que o carro entrou no mapa até que chegou ao seu destino
- g - número de mensagens enviadas do tipo GetPath
- w - número de mensagens enviadas do tipo WhichRoad
- n - número de vezes que o caminho foi recalculado

Conclusões

Esta experiência é mais irregular devido ao grande tráfego sentido na simulação. Os tempos sobem na última experiência, mas com eles também sobe o número de caminhos recalculados o que indica que o tráfego era intenso na zona onde o carro se deslocava. No entanto o número de mensagens enviadas também diminui com o tempo, indo de encontro com as experiências anteriores.

Numa primeira iteração o tempo a chegar ao destino não é tão grande como em horas de menos trânsito, pelo simples facto de, com mais trânsito, também haver mais condutores na rua que podem disponibilizar informações úteis.

b. Experiência QLearning vs A*

O objetivo desta experiência é comparar os diversos “modos” de aprendizagem implementados neste projeto. Para tal vamos repetir 3 vezes a mesma experiência para cada modo, e comparar as médias dos resultados alcançados.

A experiência será realizada no mapa mais pequeno, pois no QLearning é necessário aprender todas as ruas e o mapa maior é demasiado grande para este tipo de experiência.

Os 4 modos a comparar são :

- Short (novo) - Agente que aplica A* no entanto é a primeira vez que é introduzido no mapa;
- Short (após 1 iteração) - Agente que aplica A* após já ter conhecido parte do mundo. Neste caso usamos o agente criado após retirar os valores do short (novo);
- None - Agente que conhece o mundo por completo e aplica o A*;
- QLearning - Agente que aplica o algoritmo QLearning, após treino prévio para o destino desejado.

Foi escolhida uma hora com muito trânsito para obrigar os carros a recalcular caminhos consoante o tráfego sentido em cada zona.

Parâmetros

- Mapa : Small
- Origem : [14,12]
- Destino : [4,14]
- Hora : 12h (muito tráfego)

Resultados

	Short (novo)				**Short (após 1 iteração)**				**None**				**QLearning**			
	t	**g**	**w**	**n**	**t**	**g**	**w**	**n**	**t**	**g**	**w**	**n**	**t**	**g**	**w**	**n**
Teste 1	60	2	1	1	34	3	1	2	37	0	0	10	35	0	0	12

Teste 2	59	4	1	1	35	3	1	2	34	0	0	6	25	0	0	12
Teste 3	46	2	4	1	34	2	1	2	38	0	0	15	47	0	0	17
Médias	55	2.67	2	1	34.3	2.67	1	2	36.3	0	0	10.3	35.7	0	0	13.7

Legenda:

- t - tempo desde que o carro entrou no mapa até que chegou ao seu destino
- g - número de mensagens enviadas do tipo GetPath
- w - número de mensagens enviadas do tipo WhichRoad
- n - número de vezes que o caminho foi recalculado

Conclusões

O mapa pequeno não é o melhor para tirar conclusões porque não há muitas ruas alternativas. Em termos de tempos, são muito semelhantes à exceção do short (novo) porque não conhece o mundo. Em termos de mensagens enviadas podemos verificar que quando aplicamos o A* sem conhecer o mundo na totalidade (short) são enviadas mensagens para percebermos a melhor rota para o nosso destino. No caso do QLearning e do A* que conhece todo o mundo (none) enviar este tipo de mensagens não faz sentido, porque sabemos exatamente para onde ir. Em termos de cálculo do caminho, os agentes do tipo Short vs None e QLearning recalculam muito menos o seu caminho, visto que não conhecem quase nada do mundo. No caso do None e QLearning, sempre que recebem, por exemplo, uma mensagem de trânsito deve verificar para as suas ruas se é está ou não no seu caminho de destino.

5. Conclusões

a. Resultados das experiências

Para a Experiência A* foram analisados alguns dos objetivos desta simulação para o algoritmo A*. Analisando as três experiências para três horários diferentes podemos ver que o tempo médio da primeira iteração é menor quando há mais trânsito, enquanto que o número médio de mensagens enviadas não tem alterações significativas. Isto acontece porque a probabilidade de encontrar carros que conheçam o caminho para o destino é maior quanto mais tráfego existir. No entanto para as iterações seguintes os tempos médios para o chegar ao destino são menores quando não há trânsito, o que também vai de encontro com as expectativas visto que para a primeira experiência as ruas estão maioritariamente desocupadas. Por último podemos verificar que o número de vezes que o caminho é recalculado também aumenta com o trânsito, o que vai de encontro com as expectativas, visto que quando há ruas com muito trânsito o agente poderá optar por ruas alternativas.

Para a Experiência A* vs QLearning não foi possível chegar a uma conclusão fiável dado o tamanho do mapa. Neste caso, os agentes apresentam tempos semelhantes, mas podemos observar pelos restantes parâmetros que o None e o QLearning são semelhantes. Na nossa simulação, o agente monitorizado não usa o None (apenas QLearning ou short). Isto significa que o agente será sempre mais eficiente quando usa QLearning dado que conhece todo o mundo e faz as melhores opções de acordo com o mundo atual. As únicas desvantagens de usar este agente é que só tem 1 destino aprendido (se quiser ir para um destino diferente usará o A*), e a aprendizagem pode ser muito longa se o mapa for muito longo.

b. Trabalho

Com o desenvolvimento deste trabalho e usando pela primeira vez um sistema multi agente, reconhecemos a eficácia da utilização deste tipo de sistemas tendo em conta o contexto deste trabalho para a simulação de um ambiente e comportamentos mais realistas.

Uma das conclusões retiradas foi que, a troca e a transmissão de informação acerca do mundo vindo de outros agentes é muito importante para que o agente seja eficaz na concretização dos seus objetivos. Nas experiências efectuadas. verificou-se que, um automóvel que inicialmente não teria nenhuma informação sobre o mundo (ou apenas parcial), mais rapidamente chegaria ao local de destino se houvessem mais automóveis no ambiente. Apesar de nesta situação haver mais congestionamento, também existe mais automóveis que poderão ter informações úteis. A transmissão de informações sobre o trânsito também permite ao automóvel recalcular um melhor caminho, se possível. Para isto é necessário fazer um balanço entre os prós e contras de entre tomar uma rua com trânsito ou ir por uma de maior comprimento.

Este tipo de sistemas permite também a descentralização de tarefas pelos diversos agentes. Por exemplo, a implementação dos monitores de rua permitem que haja atualizações sobre o estado do trânsito. Caso não fossem implementados, uma maneira de determinar trânsito seria o próprio automóvel determinar se está à muito tempo parado e informar os outros. No entanto, essa maneira seria mais custosa em termos de recursos do agente porque ele estaria-se a preocupar com algo que pode ser feito por um outro agente com esse objetivo e no fim, não seria tão eficaz.

Em suma, este tipo de sistemas integrou-se bastante bem no contexto do nosso trabalho e, na nossa opinião, conseguimos aproveitar as vantagens deste tipo de sistemas. Em relação aos objetivos propostos, reportamos que foram todos concretizados.

6. Melhorias

Como em todos os projetos, existe sempre margem para melhorias. Neste caso sugeriríamos a implementação do QLearning para mais do que uma posição de destino. Neste momento o Agente tem apenas uma tabela de valores de qualidade para um certo destino. Um melhoramento seria existir uma lista de tabelas para cada novo destino que o utilizador inseri-se.

Uma outra possibilidade seria definir um ponto de paragem para fazer a transição entre o modo de aprendizagem do tipo LEARNING (treino QLearning) para o tipo APPLYING (aplicação dos valores de qualidade do treino QLearning). Numa das nossas tentativas, definimos que o ponto de transição seria quando o agente automóvel em treino já tivesse conhecimento total sobre o mapa. No entanto, como os valores ainda não se encontravam muito estáveis, existiam valores de qualidade para ruas não tão ótimas, com um valor superior às ruas ótimas. Como consequência, na fase de APPLYING, a escolha da próxima rua nas intersecções, tendo em conta o trânsito, nem sempre era a melhor.

Investir numa melhor maneira para calcular a recompensa no algoritmo QLearning poderia também ser algo a trazer mais eficácia ao trabalho.

A aplicação foi feita em mapas muito simples, apenas ruas horizontais e verticais, sendo que, uma das possíveis melhorias seria lidar com mapas mais complexos e suportar ruas oblíquas.

Também era útil ter um mapa de tamanho intermédio.

7. Recursos

a. Bibliografia

FIPA:

http://www.fipa.org/specs/fipa00026/SC00026H.html

http://www.fipa.org/specs/fipa00037/SC00037J.html#_Toc26729697

Slides disponibilizados:

https://paginas.fe.up.pt/~eol/AIAD/aiad1718_i.html

QLearning:

http://artint.info/html/ArtInt_265.html

A*:

https://en.wikipedia.org/wiki/A*_search_algorithm

Documentação do Repast :

https://repast.github.io/docs/api/repast_simphony/index.html

https://repast.github.io/docs/RepastReference/RepastReference.html

Documentação JADE :

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

b. Software

SAJaS**:**

https://web.fe.up.pt/~hlc/doku.php?id=sajas

Repast Simphony 2.5**:**

https://repast.github.io/repast_simphony.html

JADE:

http://jade.tilab.com/

c. Contribuições

Todos os elementos colaboram equitativamente para o elaboração deste projeto.

8. Apêndice

a. Diagramas UML

Figura 15 : Package cityTraffic

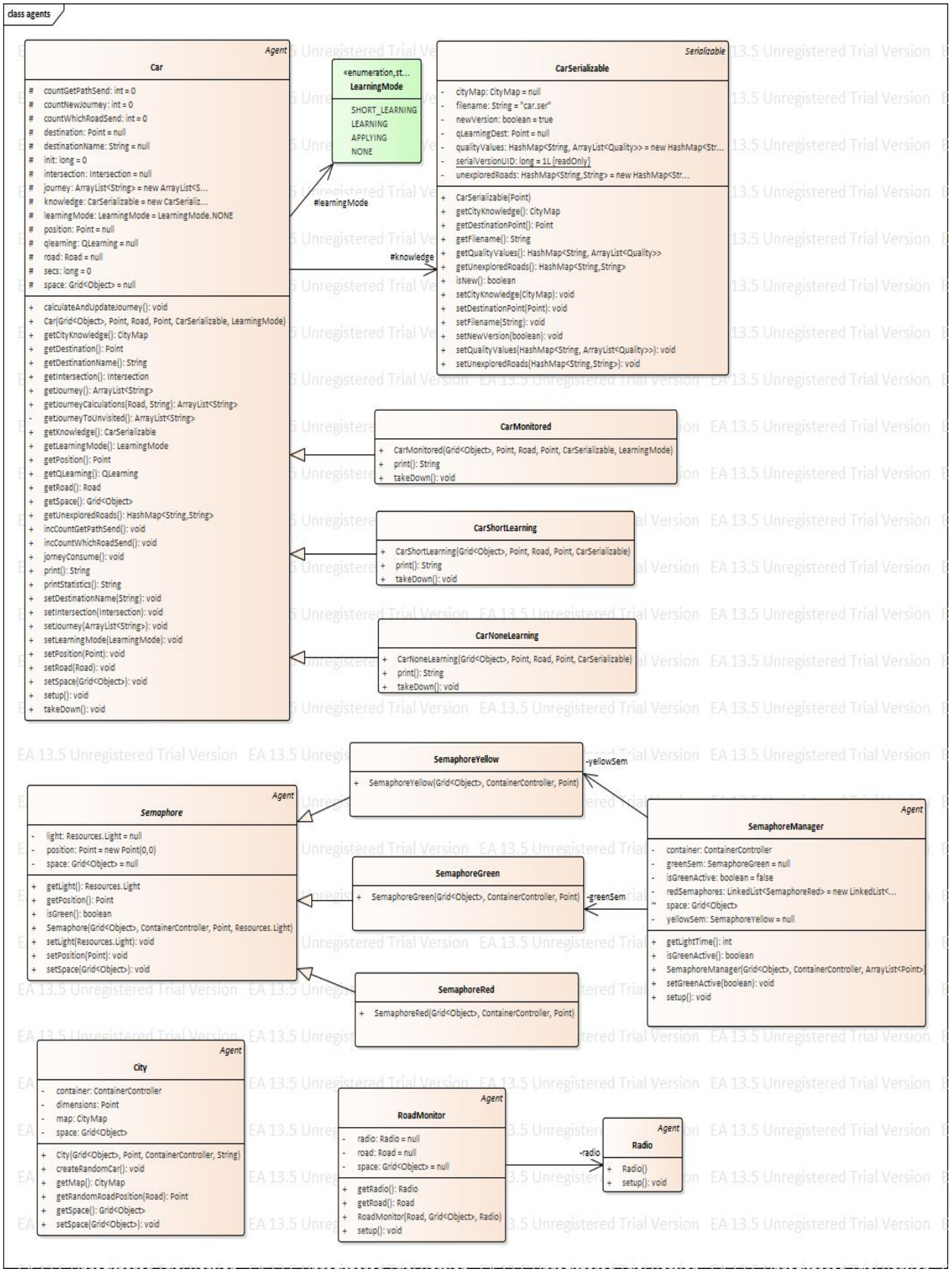

Figura 16 : Package agents

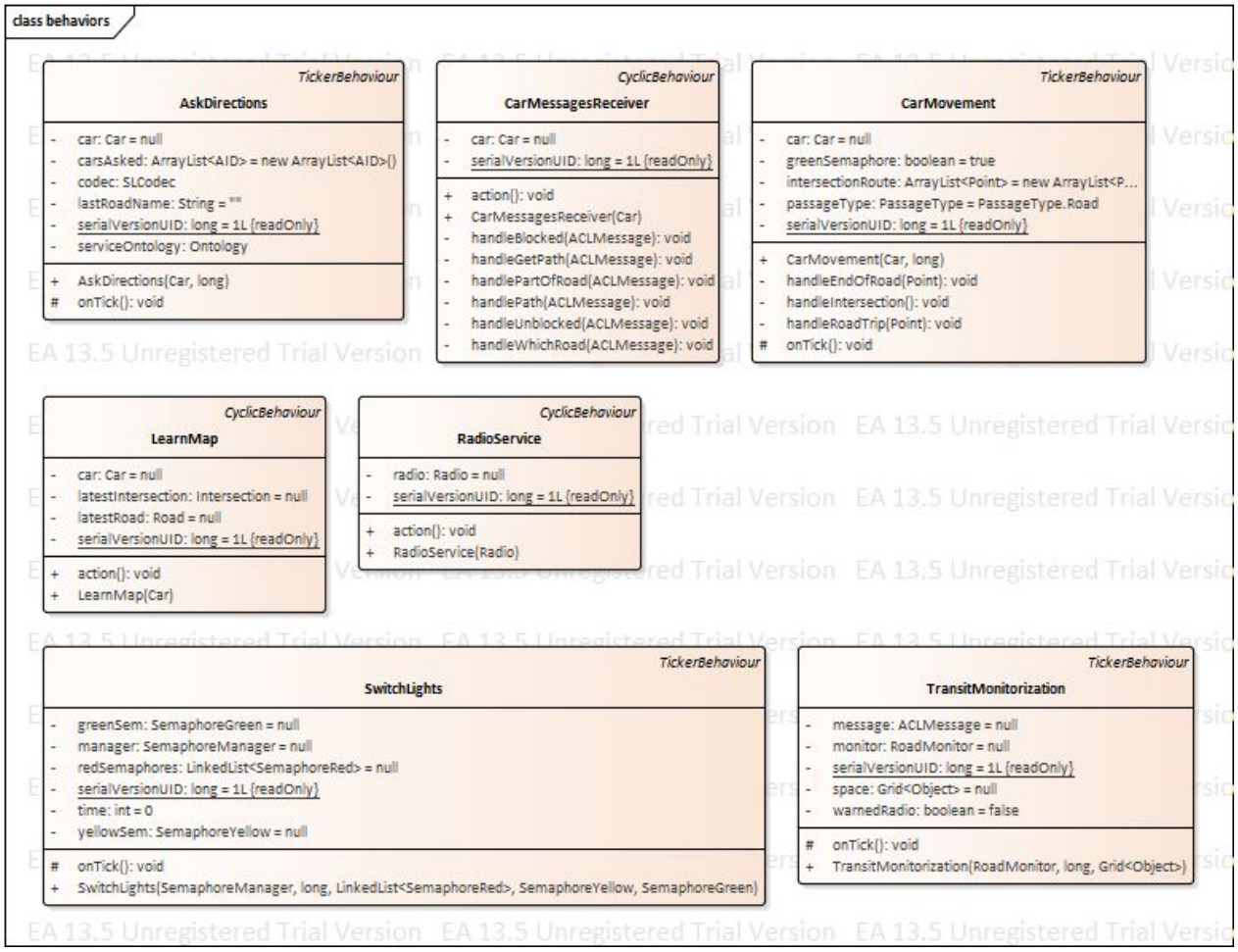

Figura 17 : Package behaviours

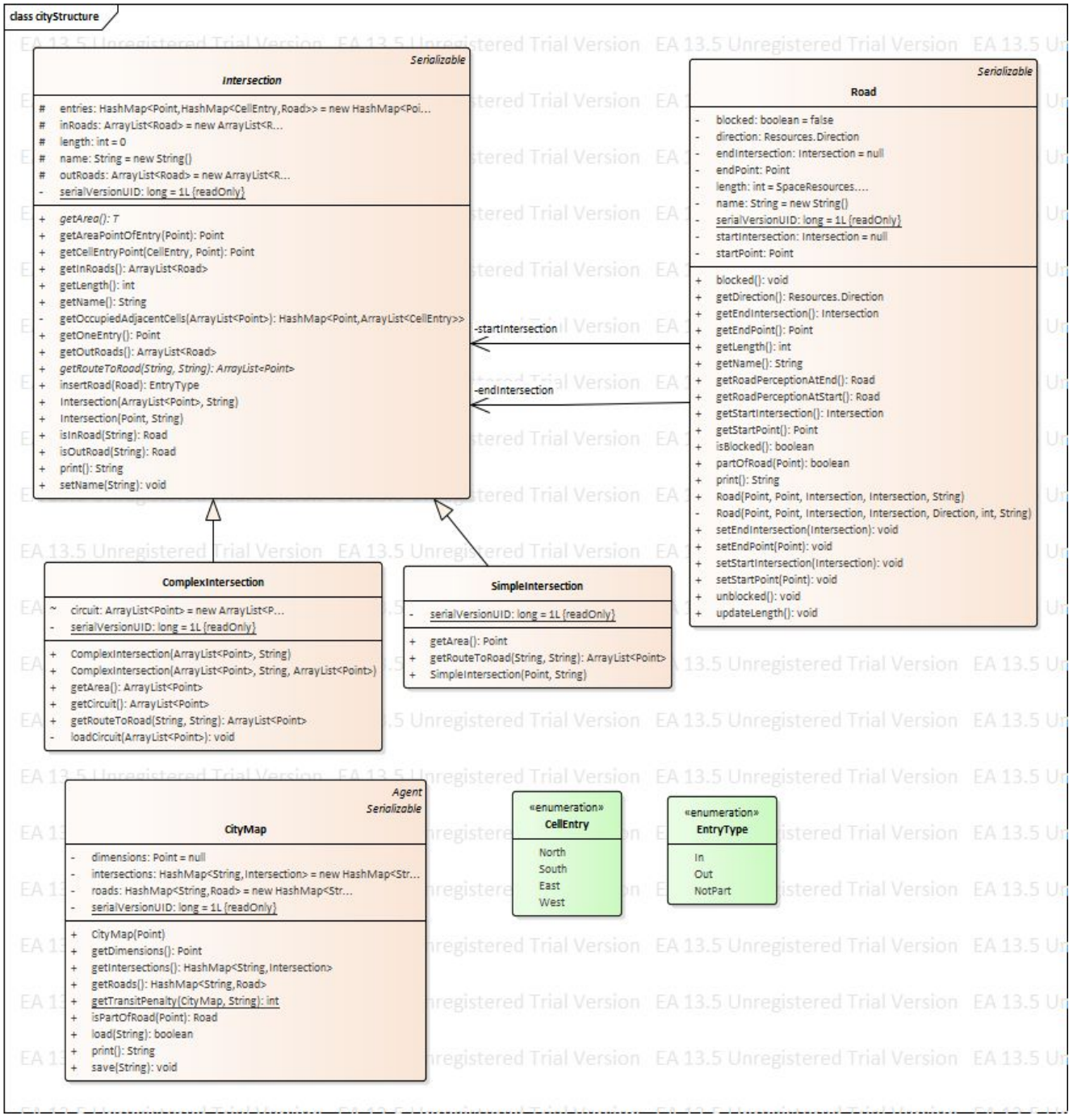

Figura 18 : Package cityStructure

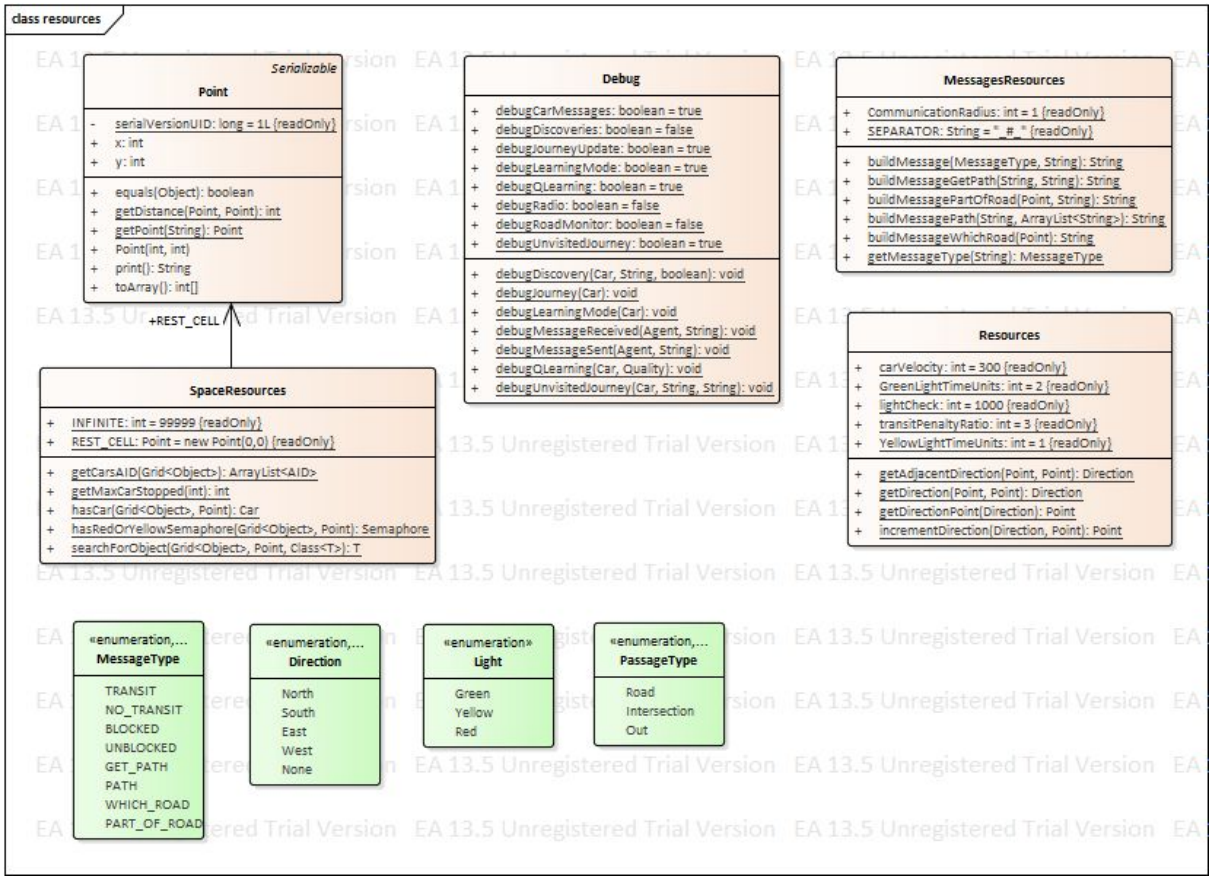

Figura 19 : Package resources

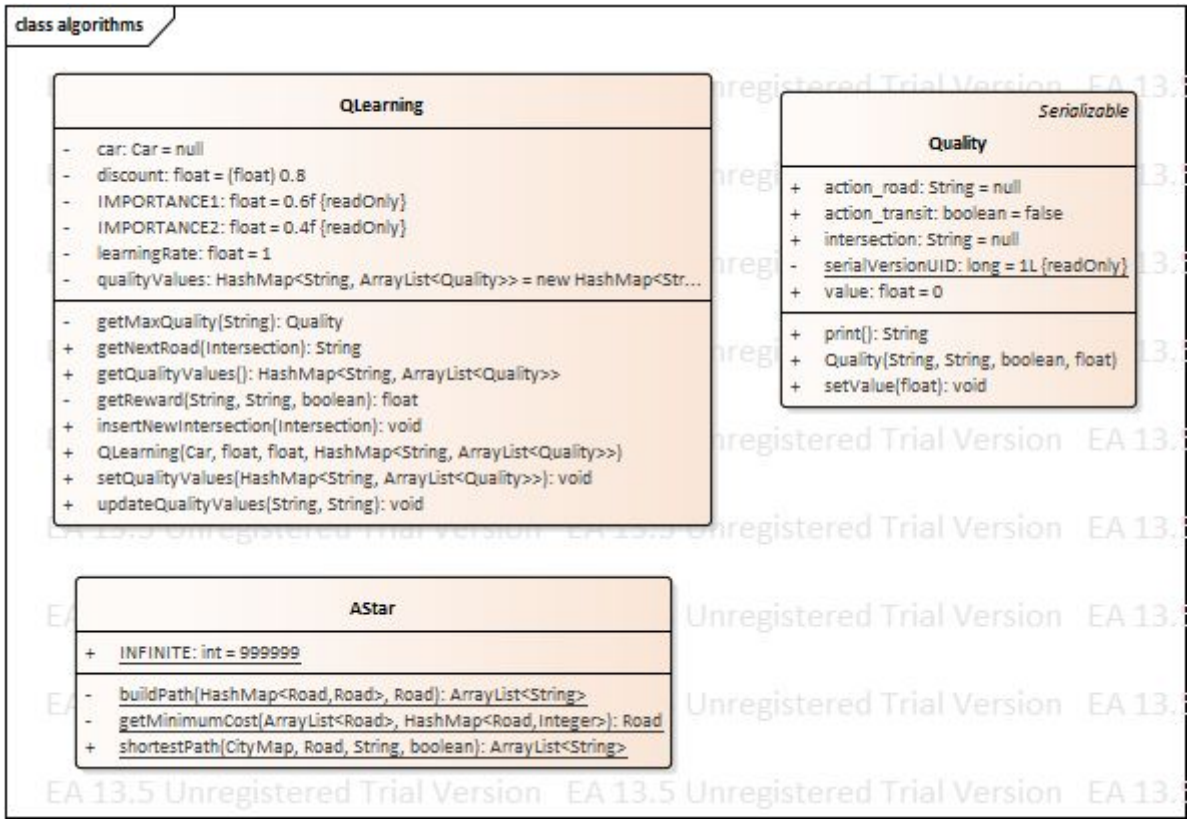

Figura 20 : Package algorithms

b. Manual do Utilizador

Manual do utilizador (sucinto)

Para executar o programa o utilizador deve:

1. Abrir o IDE Eclipse
2. Importar o Projeto (com todas as dependências para Repast + SAJAs)
3. Executar o Modelo

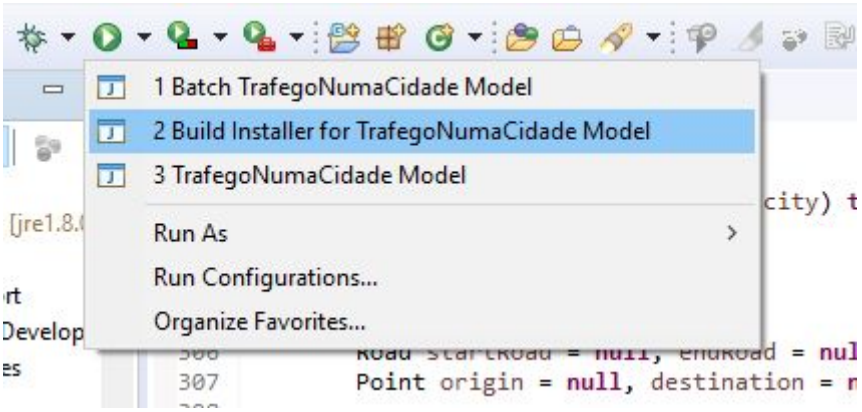

Figura 21 : opção de correr o programa

4. Aqui o utilizador deve selecionar no canto inferior esquerdo o campo “Parameters”. Nesse momento será deparado com a informação representada na imagem seguinte:

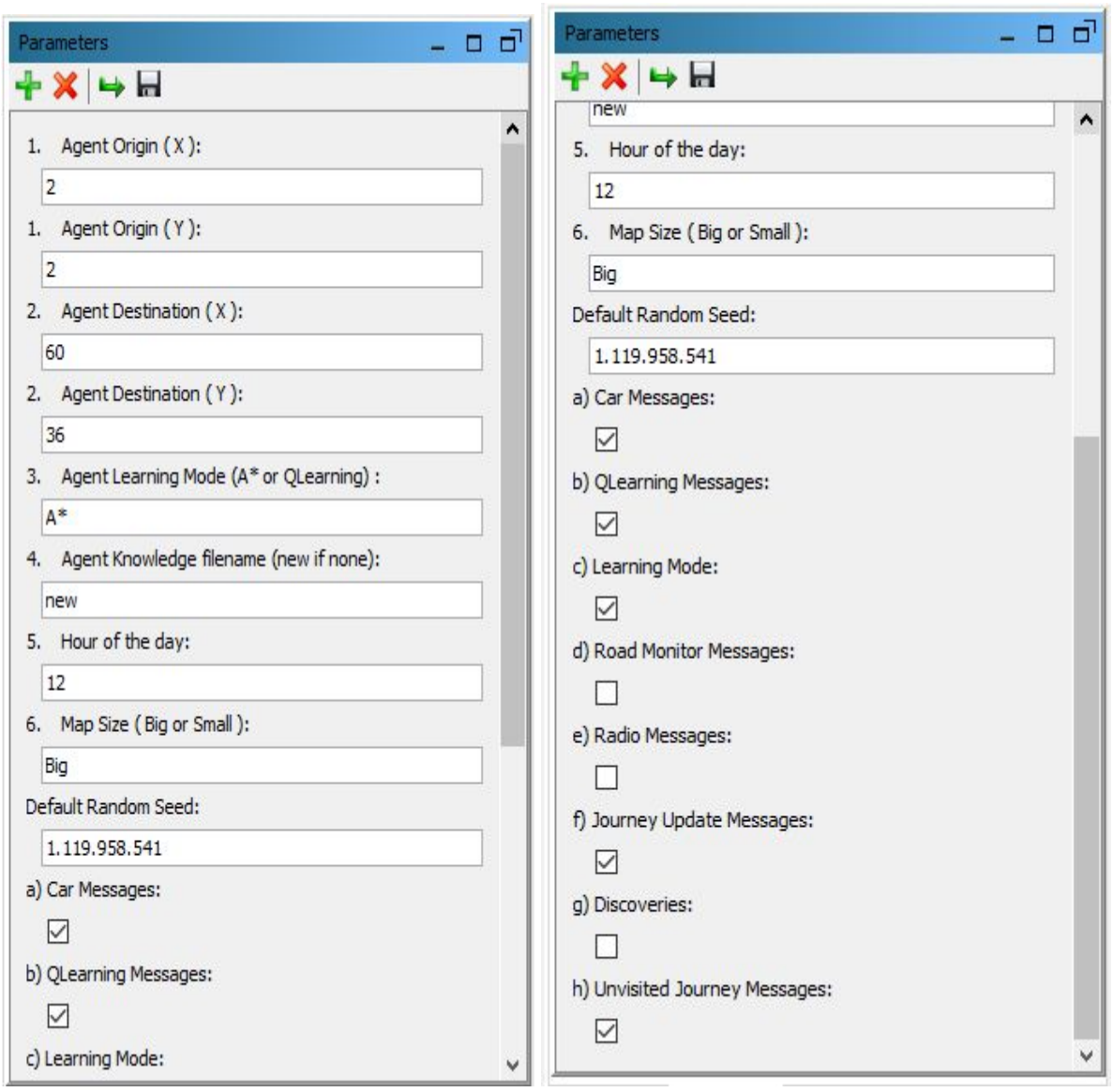

Figura 22: representação dos parâmetros no Repast

- **Agent Origin (x)** e **Agent Origin (y)**
  - Valores : 0-21 se o mapa for pequeno ou 0-61 se o mapa for grande;
  - Descrição : Célula de origem do carro a ser monitorizado. O carro deve ser introduzido numa célula dentro da estrada.

- **Agent Destination (x)** e **Agent Destination (y)**
  - Valores: igual à origem;
  - Descrição: Célula de destino do carro monitorizado. O carro deve ser introduzido numa célula dentro da estrada.

- **Agent Learning Mode (A*, QL or AP)**
  - Valores : A*, QL ou AP;
  - Descrição : O agente monitorizado tem a opção de escolher qual o algoritmo que irá executar para chegar ao seu destino. No caso de escolher o QLearning, é primeiro verificado se esse agente já aprendeu para o destino

escolhido. No caso de ter aprendido para um destino diferente que não o escolhido, será executado o A*. Se não existir nenhuma aprendizagem, o agente entrará em modo treino para esta posição de destino. Se já existe o QLearning para esta posição de destino, simplesmente aplica. Também é possível continuar a aprendizagem

- **Agent Knowledge filename (new if none)**
  - Valores: “new” ou qualquer nome que esteja dentro da pasta “objs”;
  - Descrição: Nome do ficheiro que contém conhecimento prévio do agente. Se o utilizador quiser criar um novo agente, deve escrever “new”. No final será criado um novo ficheiro de acordo com o algoritmo usado. Se A*, cria um ficheiro de nome “astar<timestamp>”. Se Qlearning cria um ficheiro chamado “qlearning_<destino x>_<destino y>”. Quando um agente que só tinha conhecimentos, até então, do algoritmo A* é introduzido o algoritmo qlearning, o nome do ficheiro que guarda o conhecimento passa para “qlearning_<destino x>_<destino y>”.

- **Hour of the day**
  - Valores: Deve ser introduzido um número entre 0 e 24;
  - Descrição : Hora do dia a que a simulação deve ocorrer. sendo:
    - 7h-9h , 12h-14h e 17h-19h há muitos carros em circulação;
    - 10h-11h , 15h-16h e 20h-21h o número de carros em circulação é moderado;
    - restantes horas o número de carros em circulação é baixo.

- **Map Size**
  - Valores: Big ou Small
  - Descrição: Faz load do mapa de acordo com o tamanho dado.
  - Nota: para mudar de mapa é necessária escolher o ícone correto e mudar as dimensões. É possível seguir esses passos aqui.

- **Debug Modes**
  - Valor: é possível escolher os valores na checklist apresentada
  - Descrição: a descrição destes valores pode ser encontrada na secção dos Resources do presente relatório.

- **Default Random seed**
  - Valor aleatório gerado pelo Repast Simphony que não necessita de ser alterado.

5. Após o utilizador configurar os parâmetros descritos, basta clicar no botão de Start do Repast. Quando o carro azul desaparecer, significa que chegou ao destino, e o seu conhecimento foi guardado. Aqui o utilizador poderá terminar a execução do programa.

Nota :
Para melhor execução do algoritmo QLearning as condições favoráveis são mapa ‘Small’.

Resultados

Na figura seguinte está representado o exemplo de uma execução do programa para um mapa pequeno numa hora com tráfego moderado. A azul é possível ver o carro a ser monitorizado, a laranja os carros que conhecem o mapa por completo e a preto os carros que desconhecem o mapa. Os pontos vermelhos, verdes e amarelos representam os semáforos. Os carros podem circular pelas estradas e no sentido em que existe um semáforo(os semáforos estão posicionados no final das ruas).

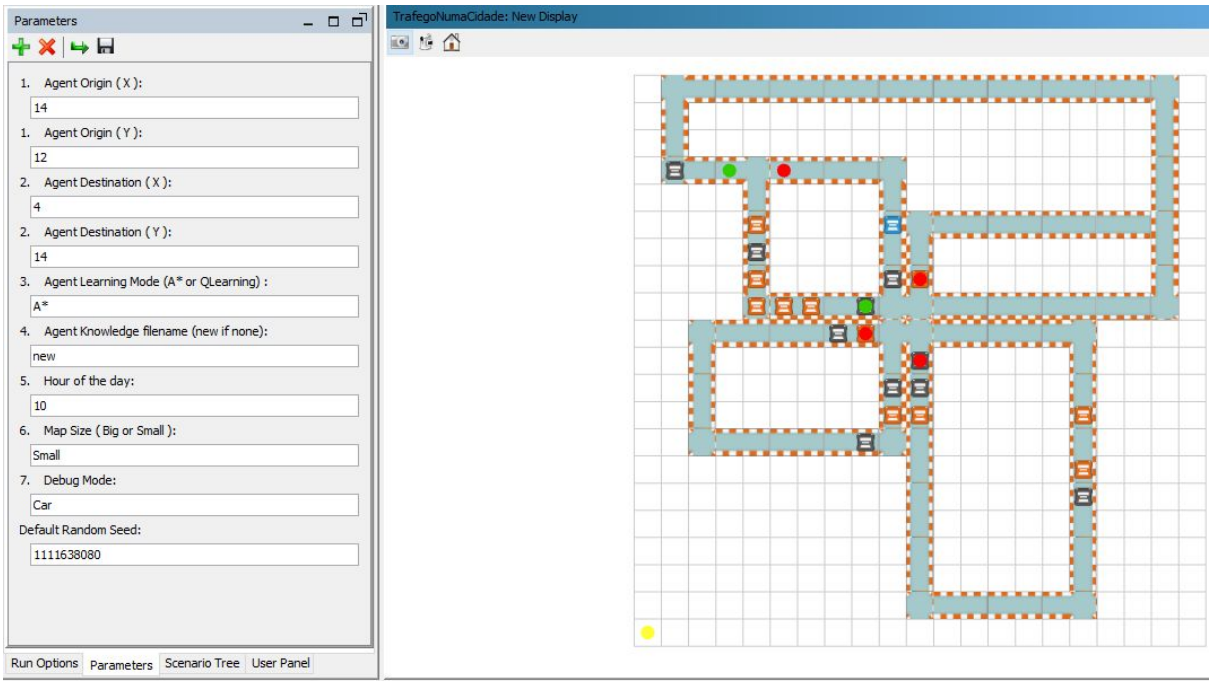

Figura 23 : exemplo da execução para o mapa pequeno

Na figura seguinte está uma representação do mapa grande numa hora de muito tráfego. A descrição do mapa, carros e semáforos mantém-se.

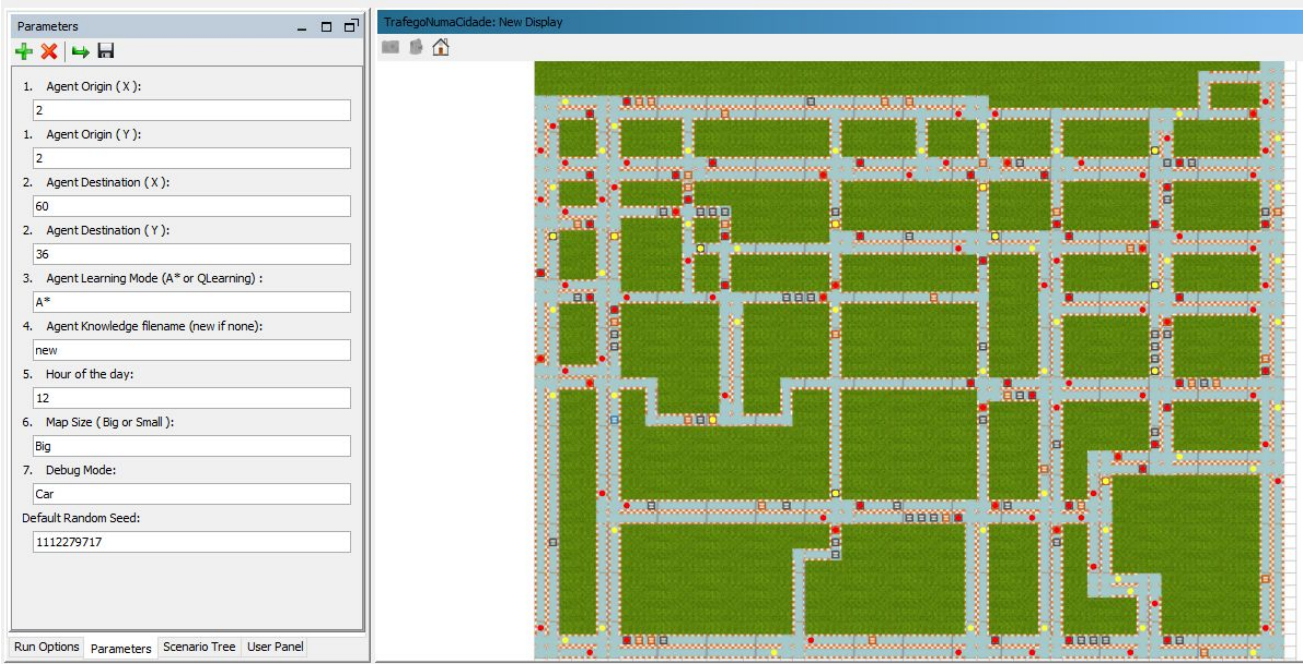

Figura 24 : exemplo da execução para o mapa grande

Mudar Mapa no Repast Simphony

1. Passar para a tab “Scenario Tree“
2. Clicar 2 vezes sobre “New Display”

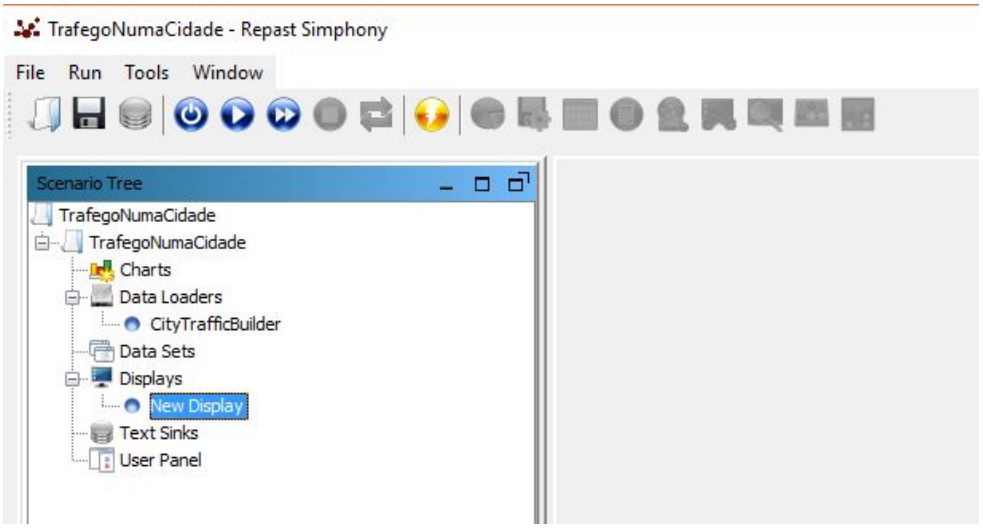

Figura 25 : repast simphony

3. Clicar em “Agent Style”
4. Clicar em “City”
5. Clicar no botão à direita como representado na figura seguinte

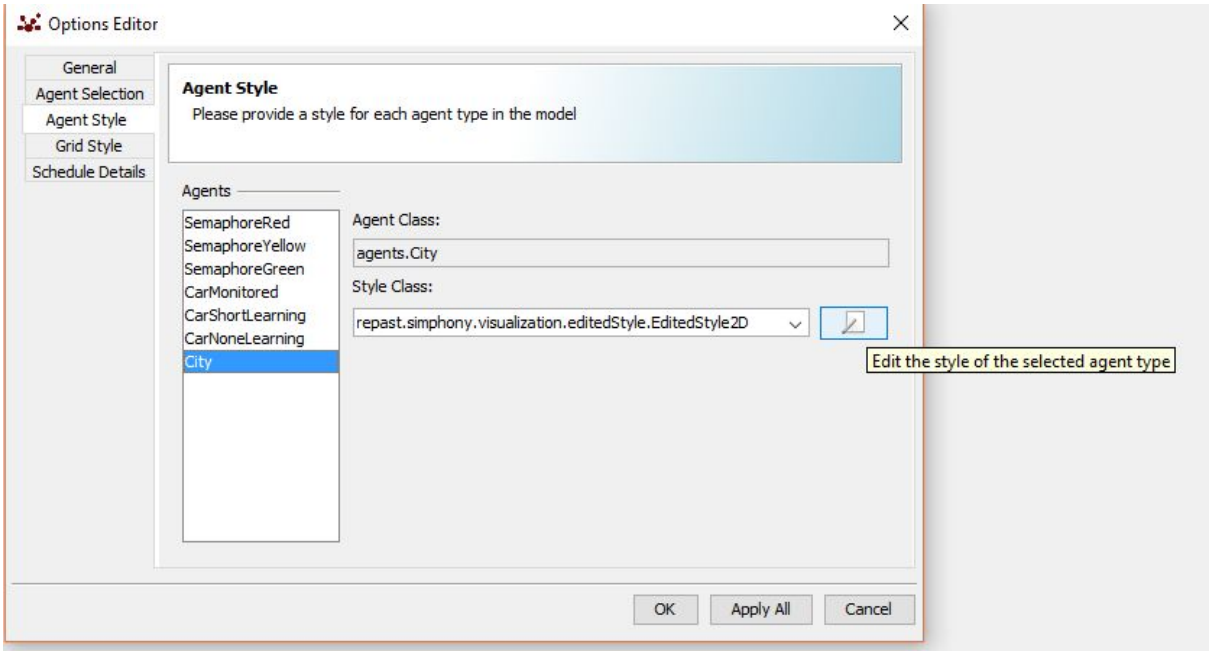

Figura 26 : repast simphony options

6. Escolher o tamanho indicado em “Value” do “Icon Size” como demonstrado na figura seguinte
    a. Small - 630
    b. Big - 1830

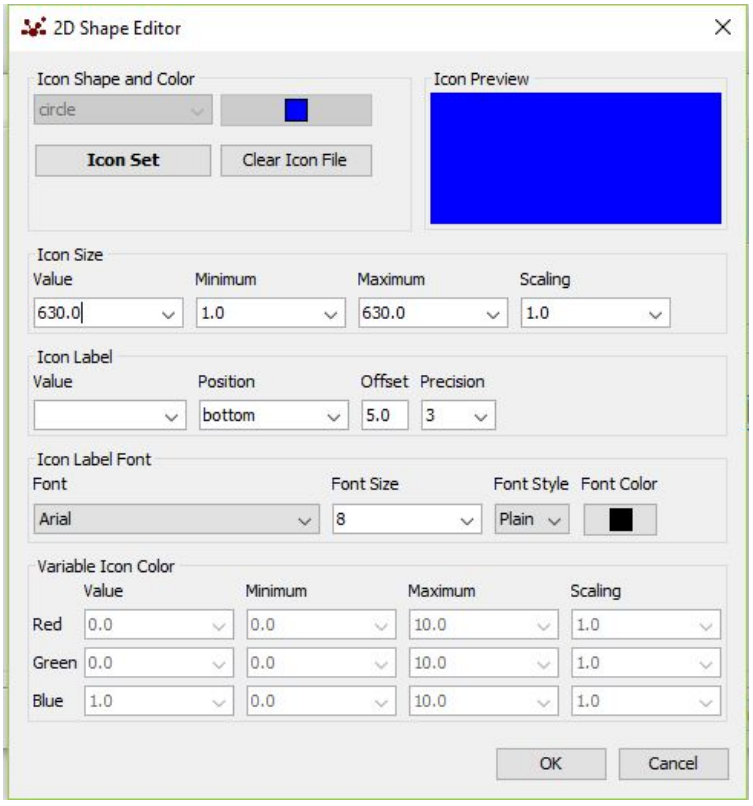

Figura 27: opções do repast simphony

7. Escolher o ícone adequado clicando no botão “Icon Set”

    a. Small : maps → small.png
    b. Big : maps → big.png
8. Clicar “Ok”
9. Clicar “Apply All”