Universidade do Minho

Mestrado em Engenharia Informática

2011

Sistemas Inteligentes

Agentes Inteligentes

Detecção de Trajectórias

Ano Lectivo de 2011/2012

pg20189 **André Pimenta Ribeiro**
pg19824 **Cedric Fernandes Pimenta**
pg20978 **Rafael Fernando Pereira Abreu**

26 de Janeiro de 2012

Resumo

A optimização é uma das áreas de investigação mais exploradas e que tem como objectivo a minimização ou maximização dos resultados de determinados problemas.

Será apresentado, neste relatório, um estudo sobre a optimização da colocação de sensores com o objectivo de detectar a trajectória de um objecto numa determinada área, recorrendo ao uso de algumas técnicas do fórum da **Inteligência Artificial**, nomeadamente a utilização de **Agentes Inteligentes**.

Serão apresentados todos os passos e estratégias utilizadas para esta optimização, assim como os resultados obtidos e uma breve análise sobre estes.

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

1.1 Contextualização e Apresentação do Problema

A optimização de recursos é um desafio cada vez mais actual, muitas vezes por questões económicas, outras vezes por questões relacionadas com tempo, situações de risco, entre outras. É nesta linha de pensamento que nos propomos a criar um sistema de optimização de colocação de sensores para detecção de movimento com a finalidade de poder representar da melhor forma o trajecto realizado por um determinado objecto e com o menor número possível de sensores.

Partindo de um movimento a realizar por determinado objecto(seja ele uma pessoa, um carro, etc), num determinado mapa, deverá ser possível representar, através da detecção de movimento do objecto, em pontos estratégicos (*checkpoints*) a trajectória que o objecto realizou no mapa.

Podemos, então, facilmente afirmar que o grande problema será a colocação de sensores de forma a obter as melhores trajectórias possíveis tendo em atenção que será necessário um grande trabalho por detrás disso para garantir a captação de movimentos e a actualização e funcionamento dos sensores, considerando que estes podem ser desligados intencionalmente ou acidentalmente.

É neste contexto que a utilização de agentes inteligentes para a representação de toda a informação necessária para a representação do problema se enquadra. O desafio passará pela utilização de agentes inteligentes através da plataforma *Java Agent DEvelopment Framework* (JADE) e a criação de um ambiente de simulação do problema representando o mais fielmente possível a realidade, ou seja, apenas a captação de movimentos por parte dos sensores será, de certa forma, estimulada com o intuito de se poder resolver o problema.

1.2 Motivação e Objectivos

Enquadrando-se este projecto no âmbito do módulo de **Agentes Inteligentes**, a aplicação destes agentes no domínio do problema em causa, bem como qualquer outro tipo de problema que se enquadre nos mesmos moldes, tem como objectivo principal a utilização desses agentes na resolução dos problemas. Esta pode ser considerada a maior motivação; no entanto, o problema proposto apresenta outro tipo de desafios e motivações que vão para além da utilização de agentes inteligentes.

Tratando-se, então, de um problema de optimização de colocação de sensores para detecção de trajectórias, teremos dois objectivos bem definidos e prioritários. O primeiro passará por tentar conseguir o máximo de precisão possível na detecção de trajectórias, ou seja,

tentar representar um possível caminho detectado pelo conjunto de sensores que represente o caminho realizado com a maior fiabilidade possível.

O segundo objectivo passa pela minimização do número total de sensores necessários para detectar a trajectória a realizar. Este objectivo entra de certa forma em conflito com o primeiro (quantos mais sensores estiverem colocados, melhor será a precisão do possível caminho detectado mas estes são um recurso limitado), e será aqui que trabalharemos na pesquisa pela melhor solução.

Também será tido em atenção que apesar de estarmos a lidar com um ambiente de simulação, tentaremos representar a realidade da melhor forma possível, preparando o projecto para uma possível utilização no mundo real.

1.3 Estrutura

O presente relatório é constituído por 4 capítulos. Sendo este o primeiro, **Introdução**, onde se contextualiza e apresenta o problema, bem como a nossa motivação e objectivos para o trabalho. No capítulo seguinte, denominado **Preliminares**, apresentaremos os conhecimentos que achamos necessários para obter uma melhor compreensão deste documento enquanto no capítulo III (**Desenvolvimento**) é explicado tudo o que foi desenvolvido durante o nosso trabalho e os resultados obtidos. Por fim, no capítulo final (**Conclusão**) são exprimidas as nossas opiniões sobre o projecto e, tendo como base os resultados obtidos, é efectuado o balanço do trabalho.

Capítulo 2

Preliminares

2.1 Domínio do Problema

Neste subcapítulo vamos explicar todos os elementos do domínio do problema e a organização entre eles. Como se trata de um problema de trajetórias temos que definir o que é uma trajetória (*Way*). A trajetória do caminhante (*Walker*) é definida por um caminho aleatório e tendencioso, que tem sempre um início e um fim bem declarados. Este caminho tem de ser realizado num número máximo de passos, podendo o caminhante não chegar ao fim se esse número não for o suficiente.
Como este *software* pretende simular um caso real, o utilizador tem a oportunidade de definir a velocidade (*Speed*) a que o caminhante anda. Esta trajetória é definida em cima de um mapa quadrado (limites: 80x80), em duas dimensões (*Terrain*). Cada posição do mapa pode ser definida com várias cores, cada uma identificando elementos diferentes:

- **Verde** - representa o terreno, em que o caminhante pode andar livremente;
- **Preto** - representa uma parede/obstáculo, obrigando o caminhante a desviar-se;
- **Branco** - representa as posições onde se encontram os sensores (*Sensors*);
- **Vermelho** - representa o raio atingível pelo sensor (*Range*);
- **Azul** - representa o caminho realizado pelo caminhante (*RealWay*);
- **Amarelo** - representa uma suposição do caminho efetuado pelo caminhante (*SupposedWay*);
- **Azul Claro** - representa a suposição do caminho, efetuado pelo caminhante, que estava correcta;
- **Laranja** - representa o ponto inicial do caminhante (*Start*);
- **Cinzento** - representa o ponto final do caminhante (*End*);

Como pretendemos detectar trajetórias, neste caso específico a trajetória de um caminhante, decidimos definir sensores/antenas (*Sensors*) que estarão espalhados pelo mapa. Estes sensores tem um alcance limitado que será definido como sendo o seu raio (*Range*) e que, por definição, estará limitado a 10 posições a toda a volta do sensor. Sempre que o caminhante entrar no raio de um dos sensores, ele será activado. Sempre que o sensor detecta um caminhante, ele informa a central dos sensores (*SensorCommunication*), que é responsável pela

comunicação entre os vários sensores, e no mapa será pintando a vermelho o raio correspondente a esse sensor, indicando ao utilizador do *software* que o sensor detectou o caminhante. Como é previsível, esse mesmo sensor será desactivado quando deixar de detectar o caminhante.

Para além disto, é necessário ter em conta que estes sensores irão custar dinheiro por isso o objectivo é limitar o número máximo de sensores (*Max Sensors*), de forma a tentar descobrir o máximo de caminho com o mínimo de sensores. Para isso, a cada iteração, o software irá simular um caminho diferente para o caminhante, começando por colocar na primeira iteração apenas os sensores iniciais (*Start Sensors*) definidos pelo utilizador. É obrigatório ter pelo menos dois sensores, um no ponto inicial e outro ponto final do caminhante. Se forem definidos mais sensores iniciais esses serão colocados aleatoriamente e, em cada nova iteração, serão acrescentados mais sensores até ao limite máximo de sensores (*Max Sensors* definidos pelo utilizador. Quando chegar a esse limite máximo de sensores, a posição destes, após cada iteração, será reajustada.

O *Supervisor* é o responsável máximo por gerir todo o sistema, ou seja, é o que manda colocar os sensores iniciais, quando o caminhante acaba o seu percurso recebe dele o trajecto realizado (*RealWay*) e solicita o suposto caminho (*SupposedWay*) ao agente *SensorCommunication*. Este suposto caminho é o trajecto que o *Sensor Comunication* prevê como sendo o percurso do caminhante, tendo como base apenas as informações fornecidas pelos sensores.

Com base neste dois caminhos (*RealWay* e *SupposedWay*), o *Supervisor* organiza e reajusta as posições de todos os sensores ao seu dispor, com o intuito de detectar o número máximo de trajetórias que o caminhante poderá percorrer numa nova iteração.

2.2 Diagrama de Classes

Os diagramas de classes fazem uma representação da estrutura e relações das classes que servem de modelo para os objetos do sistema. Cada classe contém os seus atributos e os respetivos métodos. Mostraremos, de seguida, o diagrama de classes onde estão representadas todas as relações entre as classes do nosso sistema:

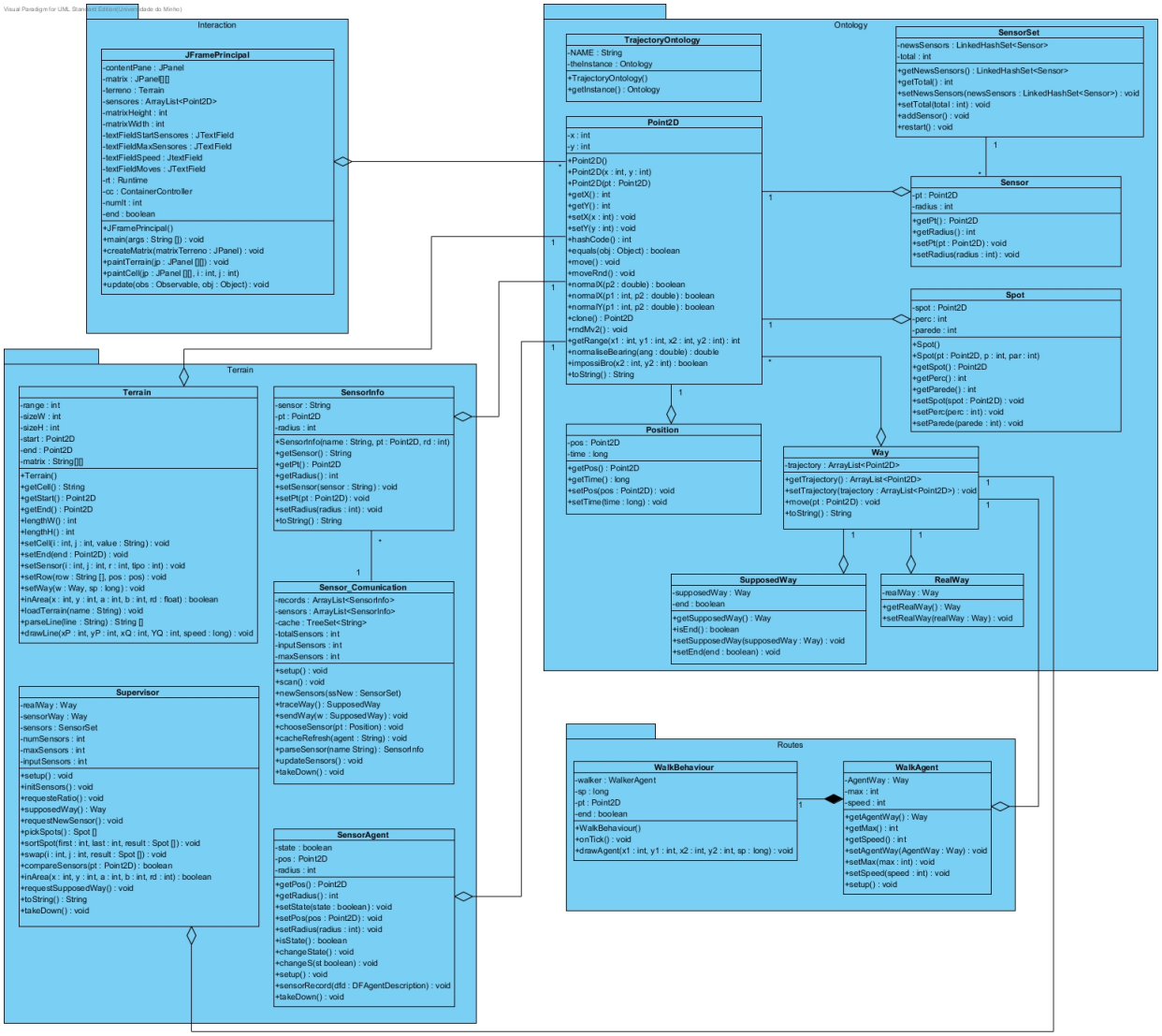

Figura 2.1: Diagrama de Classes

Como é possível verificar, decidimos agrupar as diversas classes do sistema, em quatro *packages*. Mais à frente, daremos uma breve descrição de cada package.

2.2.1 *Interaction*

Este *package* apenas contêm a classe *JFramePrincipal*, que é responsável por: inicializar a plataforma *Jade*, inicializar os vários agentes (Supervisor, Central de Sensores, Caminhante), mostrar a interface ao utilizador e permitir, a este, o uso do software de uma forma amigável.
Assim, temos o *package Interaction*:

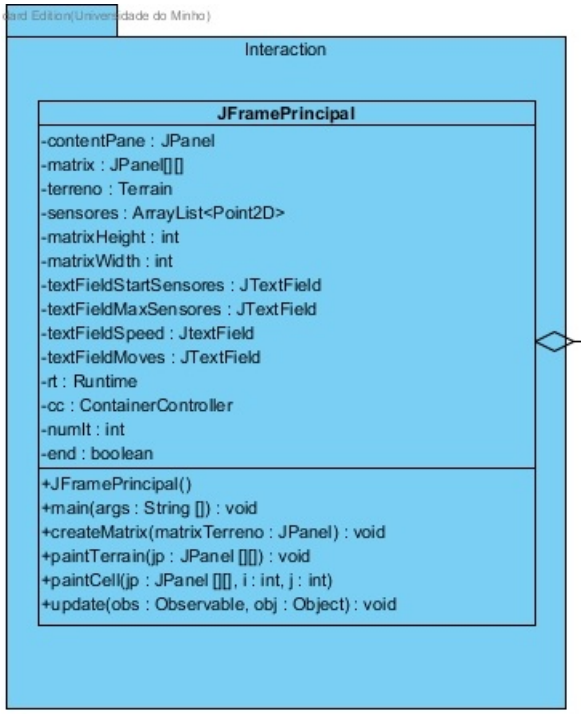

Figura 2.2: *Package Interaction*

2.2.2 *Ontology*

Neste *package* colocámos todas as classes que servem de ontologia, dentro do domínio do problema. A colaboração baseada em ontologias permite modelar o conhecimento dos agentes, facilitando a interação e troca de informações entre eles.
As classes utilizadas como ontologia são:

- *TrajectoryOntology*;
- *Ponto2D*;
- *Position*;
- *Way*;
- *RealWay*;
- *SupposedWay*;
- *Sensor*;
- *SensorSet*;
- *Spot*;

As ontologias utilizados no projecto serão explicadas com mais detalhe numa parte mais avançada deste documento. Por agora, mostramos a organização do *package Ontology*:

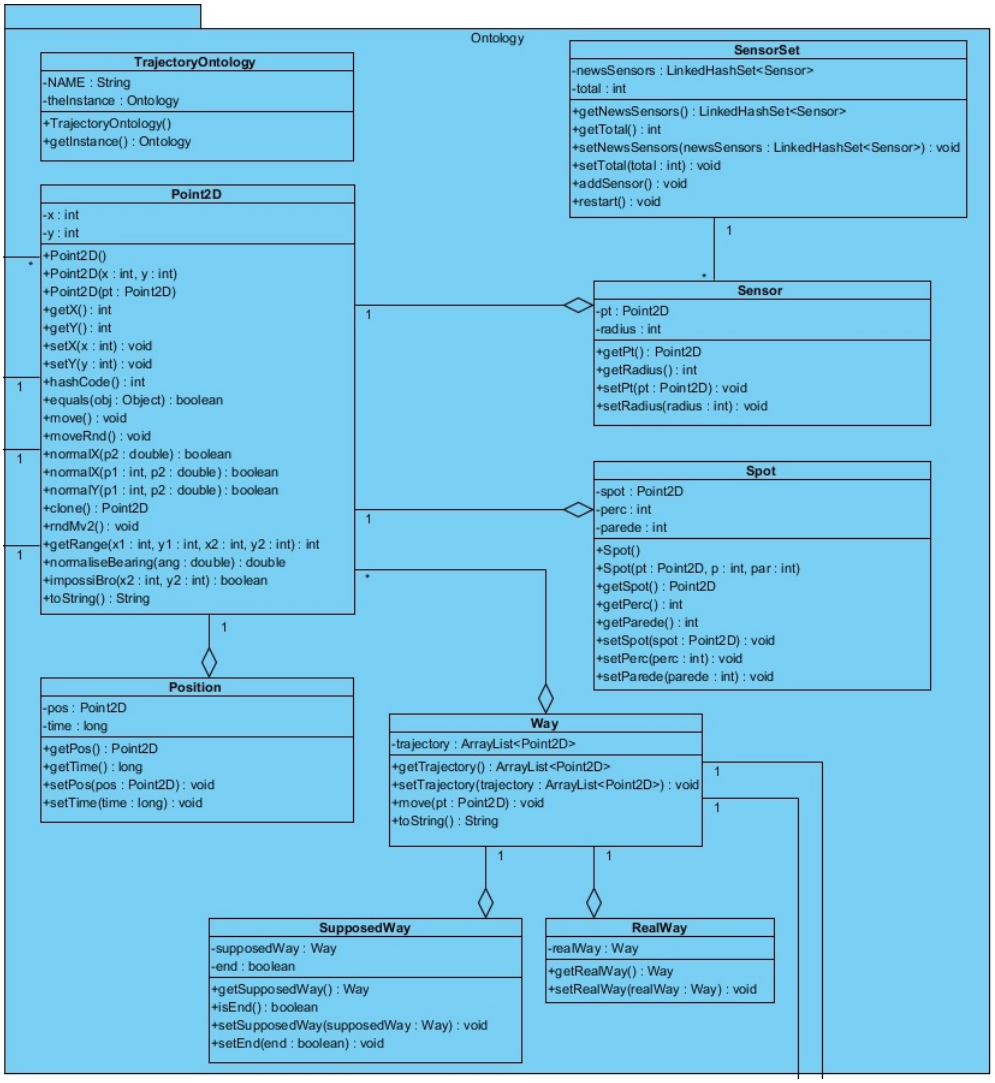

Figura 2.3: *Package Ontology*

2.2.3 *Routes*

Este *package* designado rotas, contêm a classe do agente caminhante (*WalkerAgent*) e, também, a classe *WalkerBehaviour* que detalha os comportamentos possíveis do caminhante em cada instante.

A figura seguinte retrata o *package Routes*:

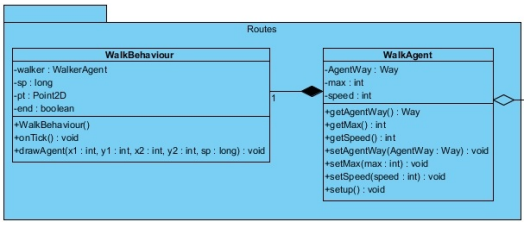

Figura 2.4: *Package Routes*

2.2.4 *Terrain*

Por fim, o *package* do terreno contêm todas as classes que alteram as informações relativas a cada posição do terreno.
As cinco classes deste *package* são:

- *Terrain*;
- *Supervisor*;
- *Sensor_Communication*;
- *SensorAgent*;
- *SensorInfo*;

A figura seguinte retrata o *package Terrain*:

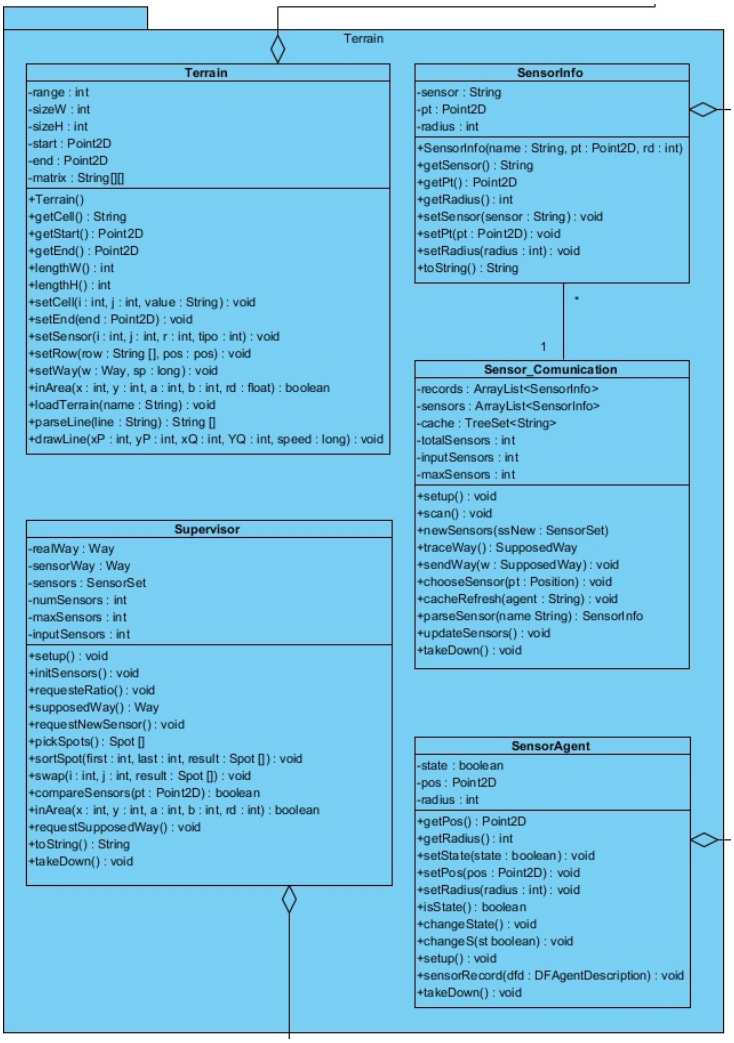

Figura 2.5: *Package Terrain*

2.3 Ferramentas Utilizadas

Neste secção retrataremos as ferramentas usadas tanto na edição do *software* como no auxílio à compreensão de todos os elementos expostos neste documento.
Assim sendo, foram usadas as seguintes ferramentas:

- ***Eclipse*** - o código da aplicação foi todo elaborado em linguagem *JAVA* no *software Open Source Eclipse*;
- ***WADE*** - o objetivo do trabalho era elaborar uma aplicação que explorasse a utilização de agentes inteligentes, utilizando a plataforma *WADE* (*Workflow and Agents Development Environment*). A plataforma *WADE* é uma extensão famosa da popular *Framework Open Source* para desenvolvimento de aplicações orientadas a agentes, designada por *JADE*. As duas principais características da plataforma *WADE*, fluxos de trabalho (*Workflows*) e suporte na administração, são bastantes distintas, tornando possível fazer-se desde simples aplicações em *JADE*, onde algumas das tarefas dos agentes (*Behaviours*) são definidos como *Workflows* ou, então, podendo-se fazer aplicações estruturadas de acordo com a arquitetura *WADE*, explorando todas as características que ela tem para oferecer;
- ***WOLF*** - de forma a utilizar a plataforma baseada em *WADE*, foi necessário usar o *plugin WOLF* do *Eclipse* que agrupa um ambiente gráfico à referida plataforma;
- ***UML*** - Para a elaboração deste documento, e de modo a representar as relações entre as várias entidades de uma forma mais perceptível para o leitor, decidimos utilizar a linguagem de modelação padrão de sistemas, designada *UML* (*Unified Modeling Language*);
- ***AUML*** - Ferramenta utilizada para representar os protocolos de interação entre agentes (*AIP*) através do uso do *AUML* (*Agent UML*). O *AUML* é uma variação/extensão do *UML* para modelação de actividades relativas aos agentes.

Capítulo 3

Descrição do Trabalho e Análise de Resultados

3.1 Ontologias

As ontologias representam o conjunto de conceitos dentro do *domínio do problema* e possuem o objectivo de representar, não só a informação necessária a este, mas também o relacionamento a realizar sobre os objectos desse mesmo domínio.

Aplicando o conceito de ontologias ao nosso problema teremos, essencialmente, de representar a informação necessária para comunicação dos nosso agentes e que permite representar o universo de discurso deste.

Podemos facilmente realçar os conceitos mais importantes para o funcionamento do sistema de simulação, bem como toda a informação que deriva destes. Apresenta-se, de seguida, a lista dos conceitos mais importantes e que originam outros conceitos com menor relevância:

- **Sensores**, responsáveis por detectar os movimentos.
- **Caminhos**, tanto o caminho real, como o caminho hipotético.
- **Posições**, utilizadas para a representação virtual das diferentes localizações ao longo do terreno.

Através destes conceitos conseguimos derivar outros e, desse modo, representar toda a informação necessária. Essa informação passará por conjuntos de sensores e as suas posições, por caminhos *tendenciosos* realizados por um objecto de teste, assim como quaisquer outros que contribuam para a representação de uma trajectória.

Definimos então a ontologia **TrajectoryOntology** que define e representa todos os conceitos já apresentados. Podemos ver de forma mais detalhada a ontologia através do diagrama de classes que se apresenta de seguida:

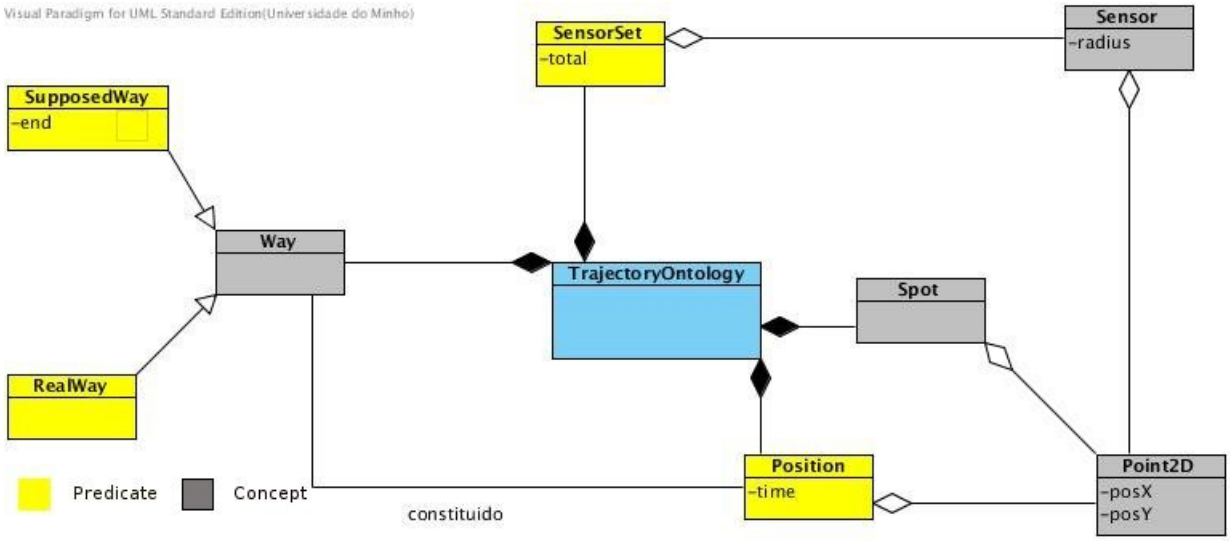

Figura 3.1: Ontologias

3.2 Agentes

Uma vez que iremos, em seguida, explicar o que pretendemos que cada tipo de agente existente no nosso projecto faça, torna-se essencial explicar o que é exactamente um agente.

Apesar de não existir uma definição exacta do que é um agente, todas estão de acordo ao afirmar que se trata de uma componente especial do software que tem autonomia e que providencia uma interface interoperavél para um sistema arbitrário, podendo, ainda, comportar-se como um agente humano, trabalhando para algum cliente em busca dos seus próprios propósitos. Uma vez que trabalhamos sobre um sistema multiagente, é possível modelar sistemas complexos e introduzir a possibilidade de diversos agentes comunicarem entre si.

Com esta pequena introdução sobre o que são os agentes, é possível verificar que para representar, o melhor possível, o nosso problema, a sua utilização é indispensável e consegue exemplificar o tipo de comunicação que teoricamente deveria existir neste caso. Como tal, cada sensor é representado por um agente que comunica com a central de informação desses mesmos sensores (tratando-se, esta central, de outro agente); este último agente comunicará com um outro agente (denominado Supervisor) que pretendemos que represente o *chefe das operações* e que seja responsável pela tomada de decisões com o intuito de melhorar o nosso sistema. Por fim, temos a necessidade de representar um caminhante que execute um caminho segundo determinados padrões; tendo em conta a definição de agente, nada mais natural que criar um agente que represente esse caminhante (assim surge o nome Walker) e que possua o objectivo de executar esse caminho e, uma vez completado, comunicá-lo ao Supervisor.

3.2.1 Supervisor

Todo o processo de optimização necessita de controlo e, uma vez tratando-se de um sistema com diferentes acções que originam diferentes informações e dados, torna-se necessário tratar destes e processá-los de forma coerente.

Para cobrir estas necessidades decidimos criar e utilizar um agente denominado de *Supervisor* que acarretará com a responsabilidade de gerir todo o sistema e tomar as melhores decisões para o mesmo. O *Agente Supervisor* deverá então ser responsável por recolher a informação dos sensores, bem como a informação do objecto que realiza o movimento para, posterior-

mente, ser efectuada uma comparação dos dados obtidos e tomar as decisões que tragam maior benefício ao sistema.

No entanto, para que se possa obter toda a informação necessária, o *Agente Supervisor* necessita comunicar com o restante sistema, mais em concreto com o *Agente SensorCommuication* e com o *Agente Walker*. O *Agente SensorCommuication* é responsável por fornecer toda a informação relacionada com os sensores de movimento, desde localizações e quantidades de sensores até às detecções de posições e consequente caminho de previsão. Já do *Agente Walker*, o *Supervisor* deverá receber o caminho realizado por este, informação que irá influenciar sobremaneira as decisões a tomar.

Tanto para efectuar estas comunicações como para tomar as devidas decisões, este agente é munido dos comportamentos que se apresenta de seguida:

- **InitSensors** (*OneshotBehaviour*). Comportamento responsável por inicializar os primeiros sensores da simulação, tendo em conta todas as configurações do terreno carregado para a dita simulação, assim como o numero mínimo e máximo de sensores pretendido. Este comportamento tem a necessidade de ser invocado apenas quando se inicializa a simulação, logo corresponde ao tipo de comportamento referido.
- **communication** (*CyclicBehaviour*). Comportamento responsável por gerir toda a informação proveniente da comunicação com outros agentes, assim como por enviar todas as "ordens"aos outros agentes. A comunicação ocorre ao longo de toda a simulação por isso optámos por um *CyclicBehaviour*.

Uma vez que o Supervisor é aquele que possui o conhecimento sobre o caminho efectuado pelo *Walker*, ao mesmo tempo que também sabe da localização dos sensores e do hipotético caminho que o *Walker* poderá ter percorrido, então será ele o responsavél por calcular quais as localizações dos próximos sensores, ao fim de cada iteração, e enviá-las ao *SensorCommunication* de modo a optimizar a simulação (o número de posições que é calculado é nos dado por *MaxSensors*-2 sendo apenas enviadas o número de posições correspondentes ao número de sensores disponíveis). Nós tomámos a iniciativa de desenvolver esse algoritmo e ter em conta as diversas especificidades da simulação; como tal, não permitimos que sejam criados sensores nas bordas do terreno (para não desperdiçar o alcance dos sensores, excepção feita aos dois primeiros), não é permitido a sobreposição de sensores (pela mesma razão do anterior e tendo em conta que os sensores são um recurso limitado), as novas posições têm em conta o trajecto executado pelo *Walker* e a posição que mais caminho detectaria. Nesta última condição, caso existam sensores com o mesmo número de passos detectados, então o factor de desempate, na existência de obstáculos no terreno (caso não existam, opta-se pela primeira posição definida), será aquele que possui menos obstáculos no seu raio de alcance. Com isto, apesar de permitirmos a colocação de mais um sensor a cada nova iteração (até ao máximo configurado), pode existir o cenário em que nem todos os sensores disponíveis são colocados pois revelariam-se pouco eficazes.

3.2.2 SensorCommunication

A utilização de vários sensores leva à necessidade de uma central de comunicação capaz de se responsabilizar por todos os sensores, assim como gerir toda a informação que estes fornecem; este agente pode mesmo ser visto como um "gerente"dos sensores.

A par da gestão de sensores, a central de comunicação, representada pelo *Agente SensorCommunication*, ficará responsável por transmitir a informação proveniente dos sensores ao *Agente Supervisor* e por fazer de intérprete entre o *Agente Walker* e os sensores, uma vez que se trata de um ambiente de simulação.

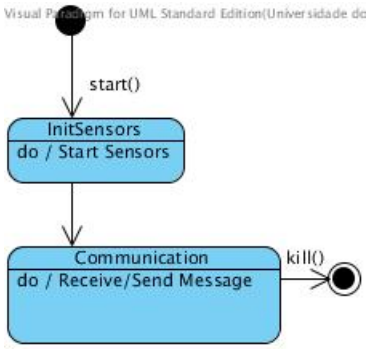

Figura 3.2: Comportamento do *Agente Supervisor*

É, também, função do *Agente SensorCommunication* negociar com o supervisor sobre as posições dos sensores de modo a resolver o problema e a obter a melhor optimização possível. Isto será possível através de troca de informações, entre o *Agente Supervisor* e o *Agente SensorCommunication*, sendo que as "ordens"virão sempre do primeiro.

Para representar todas as funcionalidades necessárias ao funcionamento do *Agente SensorCommunication* apresenta-se os seguintes comportamentos associados a este agente:

- **Scan** (*CyclicBehaviour*). Este comportamento faz a ligação entre o *Agente Walker* e o *Agente Sensor*, ou seja, vai controlar todos os movimentos efectuados e alertar os sensores. Isto deve-se ao facto de se tratar de um ambiente de simulação e não existir uma detecção real por parte dos sensores de movimento.
  Para além disto, registará todas as detecções de movimento e o respectivo sensor para posterior análise de dados, gerando a suposta trajectória e a localização dos novos sensores.
  Devido à necessidade de estar sempre em estado de alerta, este comportamento será do tipo (*CyclicBehaviour).*

- **UpdateSensors** (*TickerBehaviour*) Este é um comportamento que optśmos por introduzir para a aproximação máxima à realidade. Este comportamento é responsável por mandar um sinal a todos os sensores registados no **DF** para garantir o seu bom funcionamento, detectando assim possíveis avarias de sensores ou até actualizando, em caso de término ou no caso da colocação de novos sensores.
  Não sendo necessário verificar a cada momento o bom funcionamento dos sensores decidimos usar o (*TickerBehaviour*) e verificar periodicamente todos os sensores.

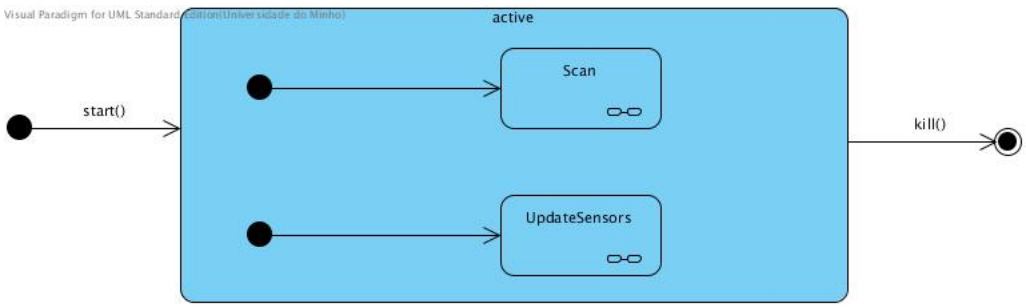

Figura 3.3: Comportamento do *Agente CommunicationCentral.jpg*

3.2.3 Walker

Para a simulação de trajectórias foi utilizado o *Agente Walker* que é responsável por originar as trajectórias e, desta forma, simular um movimento de um qualquer objecto(ex: carro, pessoa, etc).

O movimento a realizar pelo *Agente Walker* será num terreno previamente carregado de um ficheiro e que tem como objectivo representar um terreno real através de uma escala. Este *mapa* terá ainda uma particularidade que será um **início** e um **fim** representados através de coordenadas x e y (Ponto 2D). Pretende-se que desta forma o movimento seja **tendencioso**, ou seja, parta de uma posição inicial e, com um grau de liberdade, atinja uma posição final.

O *Agente Walker* tem a simples tarefa de percorrer uma trajectória entre o dito início e fim, desviando-se de potenciais obstáculos. Para a execução deste agente, existem os seguintes comportamentos:

- **WalkBehaviour** (*TickerBehaviour*). Este comportamento determina que o *Agente Walker* caminhe aos poucos e marca a sua presença no sistema de simulação, provocando possíveis detecções de movimentos por parte dos sensores e, ao mesmo tempo, deixa o seu registo de trajectória.
  Tratando-se de um ambiente de simulação, e para que sejam feitas detecções em tempo real por parte dos sensores, sempre que o *Agente Walker* efectua um movimento (*ticket*) envia uma mensagem com sua posição, provocando assim as mencionadas detecções.
  Este comportamento deverá ser executado periodicamente, enviando a sua posição até atingir o máximo de movimento permitidos para a simulação por parte do agente. O caminho realizado é enviado para o *Agente Supervisor*, facto que levou à escolha de um *TickerBehaviour*.

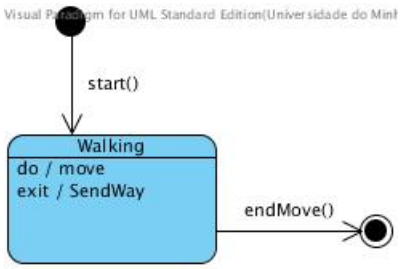

Figura 3.4: Comportamento do *Agente Walker*

Movimento *tendencioso*

O movimento executado por parte do *Agente Walker*, tal como já foi referido, deve ter um início e um fim, e manter um grau de liberdade de movimentos muito elevado.

Inicialmente utilizámos o ***Brownian Motion*** para gerar movimentos totalmente aleatórios ao longo do mapa de simulação, iniciando este a partir da posição definida como sendo o início do mapa. No entanto, obtivemos o problema que é o fim do mapa, ou seja, o objectivo do agente é chegar até este. Para resolver este problema, adaptámos o ***movimento browniano*** introduzindo-lhe uma *tendência*; quer isto dizer que tendo um objectivo, o movimento perde a aleatoriadade que possuía, sendo apenas permitidos os movimentos resultantes do algoritmo para o ***movimento browniano*** que não tenham o vector direcção no sentido da posição inicial no eixo xx e yy ao mesmo tempo.

Como podemos ver na imagem todos os passos a verdes são permitidos pois não infringem a regra do vector director e *tendem* a cumprir o objectivo que é a direcção do ponto final (alvo).

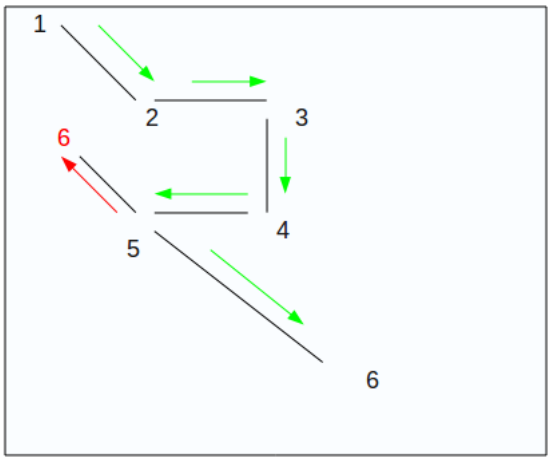

Figura 3.5: Comportamento do *Movimento tendencioso*

Já a vermelho onde se apresenta um possível passo 6 verifica-se que este não é permitido pois o seu vector direcção indica a origem na sua totalidade (eixo xx && yy).

Tendo, então, um *movimento tendencioso*, tivemos em atenção os obstáculos que um terreno possa apresentar, ou seja, existirão movimentos que poderão não ser executados pois existem zonas onde não é permitido mover-se (ex:paredes, muros, etc). Estes intraves são resolvidos através do método *impossibru* que verifica se o passo é realmente possível.

3.2.4 Sensor

A detecção de movimentos será feita através de sensores, sensores estes que contêm um raio de alcance e que deverão ser activados quando detectam a presença de algum movimento no seu raio e, consequentemente, desactivados quando já não detectarem nada.
Para conseguir representar esta mesma informação no ambiente de simulação utilizamos o *Agente Sensor* que deverá ser activado quando algo se encontra no seu raio de acção, usando uma variável binária para representar os dois estados que este pode assumir (**Activado, Desactivado**). A opção pela utilização deste agente vem tentar aproximar o sistema de simulação o mais próximo da realidade, pois é evidente que este agente não será muito preponderante na simulação mas a sua utilização permite cenários como a desactivação de sensores, avarias, entre outros.

- **changeState** (*CyclicBehaviour*) Comportamento responsável por actualizar o estado (**Activado/Desactivado**) dos sensores. Existe a necessidade de os sensores estarem sempre ”à escuta”.

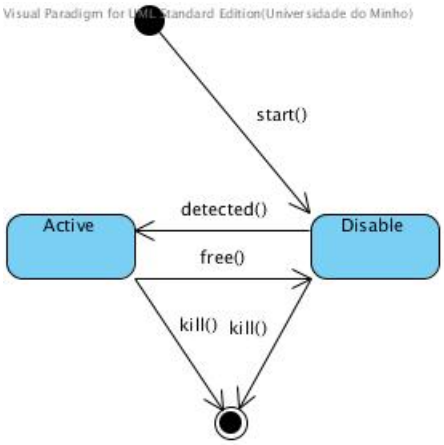

Figura 3.6: Comportamento do *Agente Sensor*

3.3 Comunicação

A comunicação entre os diferentes agentes é fundamental para o funcionamento destes, pois sem ela estes não poderiam potenciar as suas funcionalidades e ser interactivos. No entanto, para que estes possam comunicar, é necessário definir o universo de discurso, que se encontra definido através da ontologia já apresentada (*TrajectoryOntology*).

Apesar de termos a ontologia e agentes definidos, faltam as regras de comunicação regidas pela *Foundation for Intelligent Physical Agents* (**FIPA**). Usando FIPA-ACL como a especificação para as estruturas de mensagens conseguimos gerar a comunicação necessária entre todos os agentes e assim instruir a sua dinâmica.

O seguinte diagrama de sequência apresenta as principais mensagens trocadas entre os diferentes agentes, permitindo assim ter uma visão sobre a comunicação destes, embora esta esteja especificada a um nível muito geral.

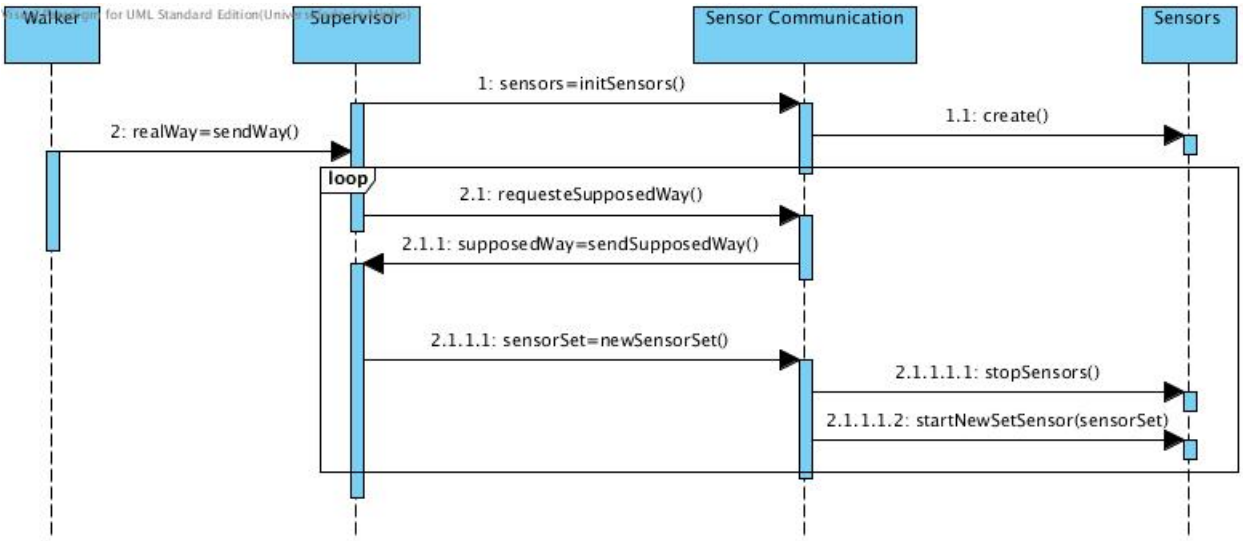

Figura 3.7: Comunicação entre agentes

Como se pode ver no diagrama, a maior parte das mensagens são trocadas entre o *Agente Supervisor* e o *Agente SensorCommunication*, pois tal como referido, estes serão os principais ”gerentes”e negociadores do sistema de simulação.

Já a detecção de movimentos por parte dos sensores será feita por intermédio do *Agente SensorCommunication* pois como nos encontramos num ambiente de simulação, torna-se impossível uma detecção automática por parte dos mesmos sensores. O seguinte diagrama exemplifica esta mesma simulação através da troca de mensagens por parte dos agentes *Walker*, *Sensor* e *SensorCommunication*.

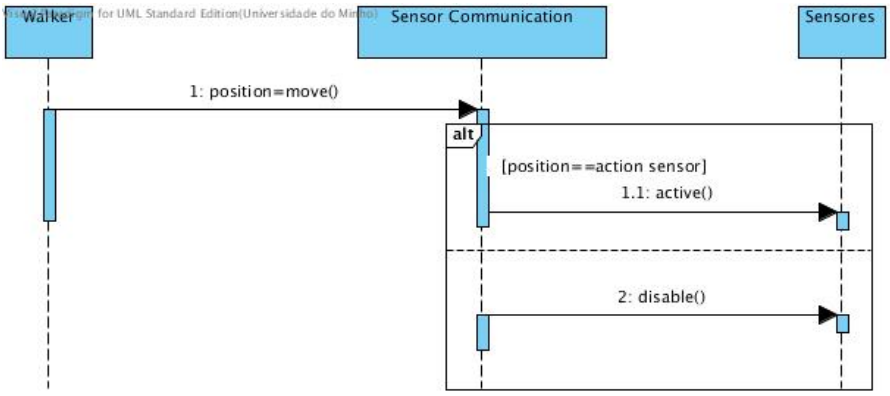

Figura 3.8: Comunicação para simulação de movimentos

Actualização de Sensores de Movimento

Tal como foi referido, é efectuada uma actualização constante de todos os sensores em funcionamento para trazer uma proximidade à realidade. Esta aproximação passa por analisar, periodicamente, quais os sensores em actividade e quais as suas características (*posição e raio de acção*).

Através da utilização do serviço de *Páginas Amarelas*, que o **JADE** implementa através do agente *Directory Facilitator* (**DF**, é feito o registo de todos os sensores de movimento colocados em funcionamento. Sempre que um sensor é inicializado, este regista-se no **DF** em conjunto com as suas propriedades (posição e raio de acção), tornando possível que estes tenham o seu "contacto"disponível. Aquando do seu encerramento, os sensores retiram o seu registo do **DF** para que este se mantenha a informação sempre actualizado.

É através do registo de serviço por parte dos sensores de movimento (*Agente Sensor*) que é efectuada uma actualização do funcionamento destes. Através do *Behaviour UpdateSensors* são recolhidos todos os serviços registados por parte dos sensores sendo, então, feita a posterior actualização destes. Deste modo pretende-se prevenir falhas dos sensores (ex: avarias) ou mesmo deslocamento de posições por parte destes.

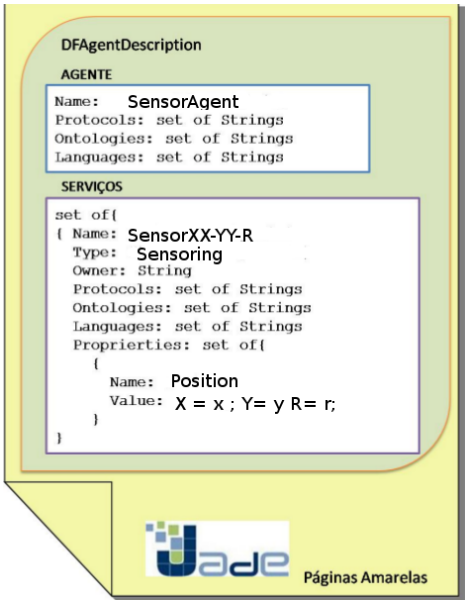

Figura 3.9: Estrutura de uma Entrada no DF

3.4 Funcionalidades

Uma vez criados e definidos os agentes que iriam actuar sobre o nosso ambiente de simulação e uma vez que, nesta fase, estes já contêm os comportamentos necessários para a devida representação, o próximo passo foi a criação de uma interface gráfica intuitiva e agradável ao utilizador comum e que permitisse a esse mesmo utilizador ter um fácil acesso a todas as funcionalidades que o sistema possui. Assim, após diversos aperfeiçoamentos, optámos pela seguinte interface:

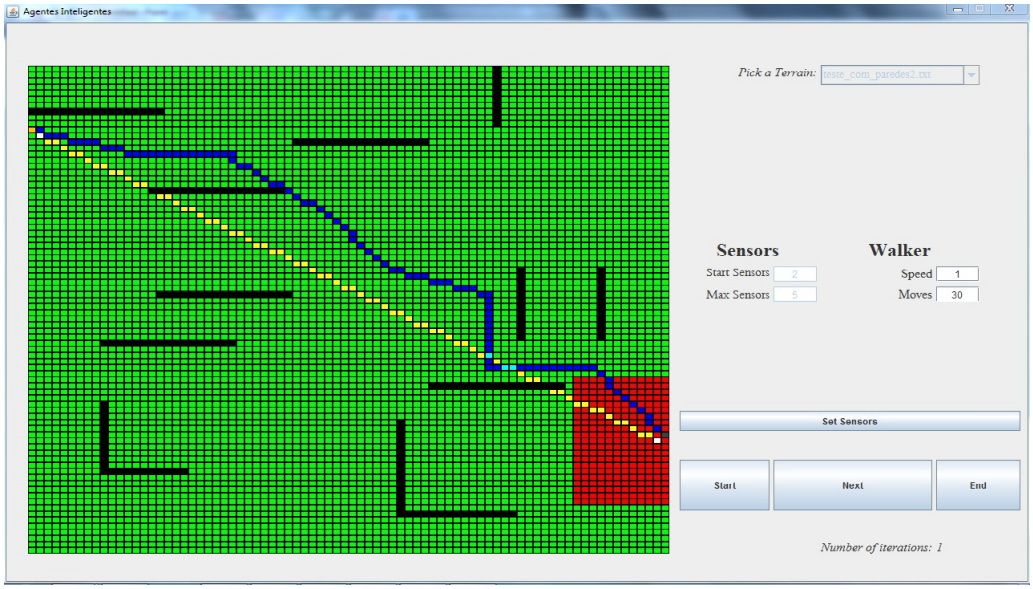

Figura 3.10: Interface do projecto

Uma vez que um dos nossos objectivos passava pela criação de uma interface simples e intuitiva, decidimos que toda a interacção com o utilizador ocorresse sobre a mesma janela, tal como é possível verificar pela imagem acima. Assim, do lado esquerdo temos a matrix (com dimensões 80x80) e que representa o mapa onde o agente *Walker* efetua o seu trajecto e

onde cada um dos agentes Sensor é colocado. Esta matrix encontra-se em conformidade com uma variável pública que instancia a classe *Terrain*, variável esta que possui um *Observer* de modo a reagir a qualquer alteração com a consequente actualização da matrix desenhada na janela.

Por sua vez, do lado direito e começando por cima, temos a escolha do terreno (ou mapa) que o utilizador pretende carregar para o software (após a escolha do terreno, esta "caixa"é bloqueada e apenas é desbloqueada pelo botão *End*). Um pouco mais abaixo existem as opções de configuração tanto dos sensores (número inicial e máximo que pretendemos que sejam colocados no terreno) como do agente Walker (a velocidade a que pretendemos que este se movimente e o número máximo de passos que ele pode efectuar). No canto inferior direito estão colocados os botões que correspondem às possíveis acções do utilizador.

Assim, o botão *Set Sensors* irá ler as configurações sobre os sensores e criar o agente SensorCommunication e o agente Supervisor (passando essas configurações a ambos) e uma vez que as configurações não poderão ser alteradas, optámos por bloqueá-las. Serão, também, colocados no mapa os novos sensores criados pelo SensorCommunication. De salientar que após a primeira iteração sobre um terreno, este botão não volta a criar os agentes mas é responsável por voltar a carregar o mapa original e desenhar os sensores correspondentes à nova iteração.

O botão *Start* apenas é necessário uma vez que é apos a colocação dos sensores no terreno e quando ainda não ocorreu nenhuma iteração. O uso deste botão irá levar à criação do agente Walker (após a leitura das configurações deste, velocidade e número de passos) que automaticamente dá início à sua caminhada.

Por sua vez, o botão *Next* existe com o intuito de encaminhar o utilizador para uma nova iteração. Assim sendo, ele irá reiniciar o agente Walker (matando-o e voltando a criá-lo, com as novas configurações); este processo deverá ocorrer após a primeira iteração e após terem sido colocados os novos sensores.

O último botão, *End*, irá reiniciar todo o programa. Para isso, ele desbloqueia as configurações que estariam bloqueadas (tanto as dos sensores como as do Walker e a escolha do terreno), coloca os valores padrão nas respectivas caixas de texto, reinicia o mapa, coloca o número de iterações a zero e mata todos os agentes que se encontram activos no nosso *Container*.

Falta, apenas, referenciar a legenda que aparece debaixo dos botões e se refere ao número de iterações já efectuadas sobre o mapa em utilização. Ao fim de cada iteração, é apresentado o valor do caminho do agente Walker que foi detectado correctamente e oferecemos a possibilidade ao utilizador de efectuar uma nova iteração caso este não se encontre satisfeito com a percentagem de caminho detectado.

3.5 Análise de Resultados

Após termos concluído a construção dos agentes e da interface, procedemos para a fase de testes na qual realizámos as mais variadas experiências de forma a conseguirmos efectuar uma auto-avaliação ao nosso projecto. Como seria de esperar, no início desta fase, deparámo-nos com alguns *bugs* que foram corrigidos de imediato.

Após esta primeira fase, todos os testes efectuados a seguir mostraram bastante satisfatórios, uma vez que, após cada iteração, o número de antenas se adaptava ao caminho efectuado e, exceptuando os casos em que o agente caminhava de uma forma totalmente díspar à anterior, a previsão do possível caminho mostrou-se bastante precisa.

Outra implementação que nos trouxe bastante satisfação foi a possibilidade de permitirmos criar um terreno com obstáculos (cenário mais próximo da vida real onde existem paredes, corredores, entre outros) e do nosso Walker não conseguir ultrapassar esses obstáculos, tendo

de os contornar.

Mostraremos, de seguida, um gráfico onde será possível visualizar os resultados obtidos após diversas iterações, para um mapa sem obstáculos e outro com obstáculos. Assim:

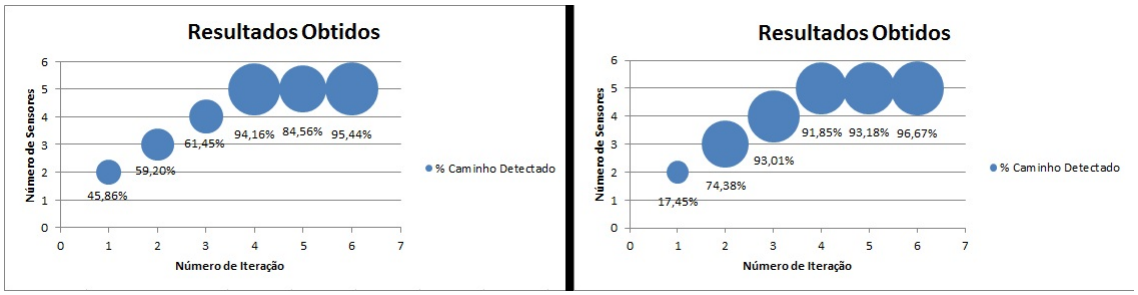

Figura 3.11: Resultados Obtidos com um terreno sem obstáculos (à esquerda) e com obstáculos (à direita)

É fácil de verificar que um maior número de iterações, levou a uma maior percentagem de caminho real detectado (sobretudo pelo facto de serem inseridos mais sensores) embora seja muito improvável obter-se uma detecção completa. Para além disto, os valores apresentados como resultados surgem da média dos diversos testes que foram executados para esta recolha de dados visto que o agente *Walker* tem um caminho bastante aleatório (apesar de ser tendencioso) e a colocação dos novos sensores apenas tem em consideração o último trajecto efectuado por este, o que pode levar à obtenção de resultados com um desvio um pouco elevado (o aperfeiçoamento deste cálculo de posições será referido como trabalho futuro). Importante referir que no mapa com obstáculos obtem-se rapidamente valores elevados uma vez que o *Walker* tem menos liberdade pois tem de contornar os obstáculos existentes.

Embora com algumas *nuances*, os resultados obtidos demonstram que já é possível obter alguma evolução na nossa simulação e estes estão de acordo àquilo que era por nós esperado.

Capítulo 4

Conclusão e Trabalho futuro

Após a conclusão do sistema de simulação, ficamos aptos para analisar o funcionamento do conjunto de agentes, bem como, os efeitos que estes transmitiram sobre o ambiente de simulação.

Pode-se afirmar que a utilização de **Agentes Inteligentes** no ambiente de simulação trouxe bastantes benefícios ao funcionamento deste. Essa utilização permitiu uma enorme injecção de autonomia e independência a todo o funcionamento do sistema, o que de forma objectiva é bastante benéfico pois deixa-o com uma capacidade de adaptação bastante elevada. A sua capacidade de comunicação é também um ponto de destaque, uma vez que permite uma aproximação à realidade bastante relevante, para além do facto de permitir uma colaboração dinâmica entre todos os agents do sistema multi-agente.

Como trabalho futuro fica o aperfeiçoamento dos movimento do *agente Walker*, como por exemplo, a introdução de mais objectivos nos seus movimentos, uma capacidade de reconhecimento de caminhos e do percurso já realizado e a possbilidade de permitir a existência não de um mas de vários agentes, em simultâneo, a efectuar o seu caminho. Também seria bastante útil o armazenamento de todos os resultados e configurações óptimas obtidas após cada iteração de forma a serem posteriormente utilizadas numa melhor análise de resultados (os vários caminhos anteriormentes executados influenciariam a colocação dos novos sensores), assim como, a sua utilização nas configurações utilizadas para a colocação dos sensores de movimento. Um último aspecto que poderia ter sido tratado agora mas, uma vez que não o foi, ficará para trabalho futuro, é representação da previsão do caminho que será efetuado pelo agente Walker ainda antes de ele o efectuar (continuando a ser representado o caminho hipotético que ele executou), isto após a primeira iteração (na iteração zero ainda nao existem dados para auxiliar a previsão).

Assim, efectuando um balanço honesto do projecto, somos da opinião que o resultado final obtido é bastante promissor e que ainda existe bastante motivação na continuação deste pois existem muitas possibilidades para o trabalho futuro, tal como referido.

Bibliografia

[1] Wooldrige M.: *An Introduction to MultiAgent* .
John Wiley & Sons, 2002.

[2] J. Machado, J. Neves e V. Alves.: *Sistemas Multiagente.*
2003.

[3] F. Bellifemine, G. Caire, D. Greenwood.: *Developing multi-agent systems with JADE.*
John Wiley & Sons, 2007.