Faculdade de Engenharia da Universidade do Porto

Universidade do Porto

Faculdade de Engenharia

FEUP

Agentes e Inteligência Artificial Distribuída
Ano Letivo 2015/2016

Relatório Final
T13_2 - Evacuação com Agentes BDI

200900579, João Pedro Milano da Silva Cardoso, ee09063@fe.up.pt

13 de Dezembro de 2015

Índice

Introdução

Cenário

Após a ocorrência de um acidente, é necessária a evacuação de pessoas de um local, o qual possui várias saídas e obstáculos, os quais resultam tanto da configuração normal do local como resultante do acidente.

As pessoas encontram-se distribuídas pelo local e após o acidente procuram atingir uma das saídas o mais rápido possível.

Objetivos

Implementação de um cenário de simulação usando agentes BDI, os quais irão simular o comportamento das pessoas, sendo capaz de ações agressivas como empurrar outros e ações solidárias como ajudar pessoas caídas. Os agentes serão capazes de simular a situação física de uma pessoa (saudável, ferida ou caída).

O local do acidente é editável pelo utilizador, sendo possível alterar a sua configuração, posição das saídas, obstáculos e localização das pessoas.

Adicionalmente o programa será capaz de registar ocorrências de modo a apresentar valores estatísticos para diferentes configurações do local do acidente.

Especificação

Agentes BDI - Arquitetura

O programa tem como base o desenvolvimento e utilização de agentes BDI, os quais são implementados em Java utilizando classes, sendo definidos pela anotação @Agent e tendo obrigatoriamente o seu nome a terminar em BDI. São compostos principalmente de 3 componentes:

Crenças (Beliefs)

Crenças são o que um agente acredita ser verdade sobre o mundo e representam o seu estado de informação. Em Jadex as crenças são variáveis de um agente e a sua definição é feita utilizando a anotação @Belief.

Desejos (Desires)

Desejos representam objetivos ou situações que um agente gostaria de cumprir ou alcançar. No Jadex, os desejos são aproximadamente implementados por Goals, os quais podem ser implementados recorrendo a classes e são definidos pela anotação @Goal. A utilização de goals ajuda na seleção de planos a executar.

Intenções (Intentions)

Intenções representam o estado deliberativo do agente, o que o agente decidiu fazer. No Jadex intenções são aproximadamente implementadas por Plans, que são uma sequência de ações que um agente pode executar de modo a cumprir uma ou mais intenções e são definidos usando a anotação @Plan. Podem ser implementados usando classes. Os planos podem ser executados como resposta a um goal a ser cumprido ou adotados diretamente pelo agente.

Agentes da Simulação

PersonBDI

O agente PersonBDI simula o uma pessoa.

Este agente tem a capacidade de se deslocar no espaço, executar ações agressivas em relação a outros agentes (empurrar e atropelar) e a capacidade de ajudar outros. Estas ações são decididas através da geração de valores aleatórios, tendo em conta o nível de pânico da pessoa.

Consegue também simular a situação física de uma pessoa - capaz de movimento, ferida ou caída no chão - o que afeta a sua velocidade de deslocamento.

Tem como crenças:

- **ContinuousSpace2D space** - O espaço da simulação em que o agente se encontra.
- **SpaceObject myself** - O próprio agente dentro do espaço.
- **double speed** - A velocidade a que o agente se desloca. É afetada pela condição física.
- **int physicalCondition** - A condição física do agente. Influencia a sua velocidade. Abaixo de um determinado valor fica incapaz de se mover e necessita de ajuda. Sempre que é alvo de uma ação agressiva o valor diminui. Se a condição física chegar a zero a pessoa morre.
- **int panicLevel** - Nível de pânico do agente. Afeta a probabilidade de tomar ações agressivas.
- **int levelheadedness** - Personalidade do agente. Multiplicador que afeta a velocidade a que o nível de pânico aumenta.
- **boolean knowledge** - Indica se o agente tem conhecimento da planta do local.
- **boolean down** - Indica se o agente se encontra caído.
- **stunned** - Indica se o agente se encontra atordoado
- **Pair<Integer,Integer> door** - Indica as coordenadas de uma saída não obstruída quando informado por um segurança.

Este agente tem como objetivos:

- **GetOut** - Objetivo tomado por um agente com conhecimento do espaço para procurar uma saída.
- **RandomWalk** - Objetivo tomado por um agente sem conhecimento do espaço para procurar uma saída.

Adicionalmente, utiliza os planos:

- **getOut** - Plano desencadeado pelo objectivo GetOut.
- **randomWalk** - Plano desencadeado pelo objectivo RandomWalk.
- **updateSpeed** - Actualiza a velocidade. É desencadeado quando ocorrem mudanças na condição física.

Este agente é representado no espaço pelo objeto person, que tem as seguintes propriedades:

- **Position** - Posição do objeto num espaço 2D.
- **Speed** - Velocidade do objeto.
- **Alive** - Indica se o agente está vivo.
- **Wounded** - Indica se o agente está ferido.
- **Down** - Indica se o agente está caído.
- **Direction** - Direção em que o agente se dirige.
- **Physical Condition** - Condição física.
- **Collision Radius** - Raio para cálculo de colisões.
- **Stunned** - Indica se o agente está atordoado.
- **Agro Action** - Indica se o agente decidiu ser agressivo numa colisão
- **Agro Value** - Valor para comparar nas colisões.

Fig 1 - Representação do Agente Person

Um agente Person pode ter ou não conhecimento do espaço em que se encontra.

Se tiver conhecimento dirige-se até à saída mais próxima da sua posição na esperança que esta não esteja obstruída. Se estiver desloca-se até à seguinte mais próxima.

Se não tiver conhecimento do espaço escolhe um waypoint aleatório e move-se para ele. Se aí conseguir ver uma porta não obstruída desloca-se para ela.

Ao ser informado da existência de uma saída desloca-se para a mesma se o seu movimento for aleatório.

O agente implementa o serviço IChatService, que permite que o agente Security o informe da existência de uma saída. Utiliza o serviço WorldInformService para informar o agente World quando sai do local ou fica ferido.

SecurityBDI

O agent SecurityBDI simula o comportamento de um segurança.

Um segurança é uma pessoa que tem sempre conhecimento da planta do local, nunca executa ações agressivas e tenta ajudar sempre pessoas ao colidir. Começa por deslocar-se até à saída mais perto da sua posição inicial e se esta não estiver bloqueada informa outros agentes da sua posição. Caso contrário irá procurar outra saída.

Partilha com o agente Person as crenças space, myself, speed, door; o objectivo GetOut e o plano getOut.

Utiliza o serviço ICharService, que serve para comunicar com outros agentes e informar sobre a localização de uma saída.

São representados no espaço pelo objeto Security, com as propriedades position, speed e direction.

Fig 2 - Representação do Agente Security

O agente Security dirige-se até à saída mais próxima da sua posição na esperança que esta não esteja obstruída. Se estiver desloca-se até à seguinte mais próxima. Quando encontrar uma saída não obstruída avisa os outros agentes da sua localização.

O agente utiliza o serviço ICharService para informar agentes Person da localização de uma saída o WorldInformService para informar o agente World quando sai do local.

WorldBDI

O agente WorldBDI serve para reunir informação sobre a simulação e decidir quando esta termina. Adota um único plano, uma rotina que verifica os objetos existentes no espaço para verificar o estado da simulação. A simulação é dada como terminada quando os únicos agentes existentes no espaço estão mortos ou numa condição em que sejam incapazes de se mover. Não tem goals, adotando diretamente o plano WorldMonitorPlan. Regista o número de pessoas vivas, o número de pessoas mortas e o número de vezes que uma pessoa ficou ferida.

Adicionalmente foi implementado um agente WaypointChecker, que não tem impacto sobre a simulação e serve apenas para colocar objetos representativos dos waypoints no espaço para verificação dos mesmos.

Protocolos de Interação

Os agentes Security trocam mensagens com os agentes Person de modo a informá-los quando encontram uma saída, comunicando as coordenadas.

Os agentes Security e Person enviam mensagens para o agente World sempre que saem vivos do local ou ficam feridos.

Os agentes Person interagem entre eles através de 2 tipos de ações: agressivas ou de ajuda.

Quando um agente detecta colisão com outro, existem 2 cenários possíveis:

1. Ambos os agentes estão de pé.
2. Um dos agentes está caído.

Se um agente estiver caído o outro poderá ou não atropelá-lo dependendo se decide tomar uma ação agressiva ou ajudá-lo. Se ambos os agentes estiverem de pé existem 3 situações possíveis:

1. Ambos os agentes decidem tomar ações agressivas. Neste caso é comparada o agroValue dos objetos. O agente vencedor continua o seu percurso enquanto o perdedor fica caído temporariamente e a sua condição física diminui.
2. Nenhum agente executa uma ação agressiva. Neste caso é comparada a sua condição física. O vencedor continua o seu percurso enquanto o perdedor fica temporariamente atordoado mas não cai ou sofre alterações à sua condição física.
3. Um dos agentes decide executar uma ação agressiva. Neste caso ele será sempre o vencedor e o resultado é o mesmo do que na situação 1.

Se um agente estiver caído e for atropelado a sua condição física diminui. Se um agente decidir ajudar um agente caído ele fica de pé, mas atordoado. Se estiver ferido a sua condição física recupera.

Se uma pessoa colidir com um segurança fica sempre atordoada, mas se estiver caída ou ferida é sempre ajudada.

Desenvolvimento

Plataforma de Desenvolvimento - JADEX

Para o desenvolvimento do programa é utilizado Jadex, um motor de raciocínio baseado em BDI (Beliefs, Desires, Intentions), que permite a programação de agentes inteligentes em XML e Java, baseia-se na noção de componentes ativos e providencia um interface, o Jadex Control Center para criação de agentes e lançamento de aplicações.

Entre as suas características são de destacar como relevantes para o projecto:

1. Capacidade de simulação de um espaço configurado em XML, providenciando um interface gráfico em que os agentes se podem deslocar e interagir.
2. Utilização de anotações para facilitar o desenvolvimento de agentes.
3. Utilização de um modelo de concorrência que permite que os componentes possam simular o comportamento de sistemas multi agente.
4. Capacidade de criação de serviços, que os agentes irão utilizar para comunicar.

5. Utilização de objetos no espaço de simulação. Os agentes irão ser representados por objetos que têm um conjunto de propriedades. É através da manipulação destes que a simulação será visualizada.

Ambiente de Desenvolvimento

O projecto foi desenvolvido usando o IDE Eclipse com Java 1.7 em Windows 7 e 8.1 de 64 bits em Java e XML.

Estrutura da Aplicação

No Jadex os agentes são representados por objetos e controlam as propriedades destes, os quais são depois apresentados no interface do Jadex Control Center.

No ficheiro XML de configuração do espaço é possível a declaração de processos a correr antes do interface. É utilizado o processo FloorPlanCreationProcess para ler a configuração do espaço e das variáveis iniciais dos agentes de ficheiros de texto.

Os agentes são posteriormente criados, recolhem as informações iniciais e iniciam os seus objetos, iniciando depois a simulação.

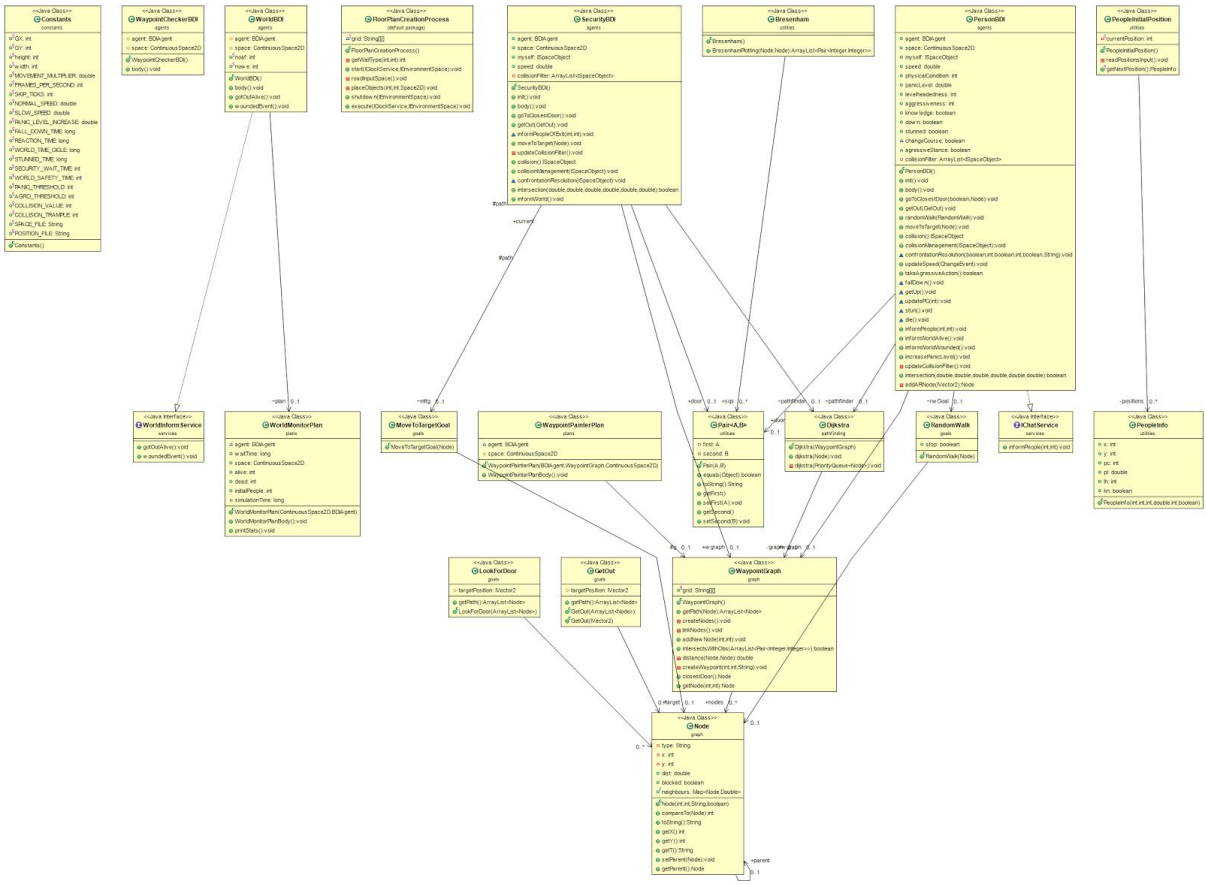

Fig 3 - Diagrama de Classes

O diagrama de classes encontra-se também na pasta do projeto.

Detalhes Adicionais da Implementação

Espaço da Simulação

O Jadex permite a utilização de um interface gráfico para a apresentação de um espaço onde os agentes se podem deslocar. Este espaço está configurado num ficheiro XML que descreve os objetos, as suas propriedades, como estes são representados, configurações e processos iniciais. Estes objetos representam agentes, obstáculos e saídas.

O movimento dos agentes no espaço consegue-se através da manipulação da posição dos objetos.

Para a construção e edição do espaço recorre-se a um ficheiro de texto com a planta do local a evacuar. A planta tem uma tamanho de 30 x 30. As posições e valores iniciais dos agentes estão definidas num ficheiro separado.

O processo FloorPlanCreationProcess lê os ficheiros, cria os objetos necessários e guarda as variáveis iniciais dos agentes.

O ficheiro de espaço utiliza a seguinte configuração:

- Paredes - W
- Obstáculos - O
- Portas - E
- Obstáculos a bloquear portas - 0 (zero)
- Espaço livre - Espaço

Cálculo de Caminhos

O cálculo do percurso é efetuado transformando o espaço num grafo de navegação baseado em waypoints[1].

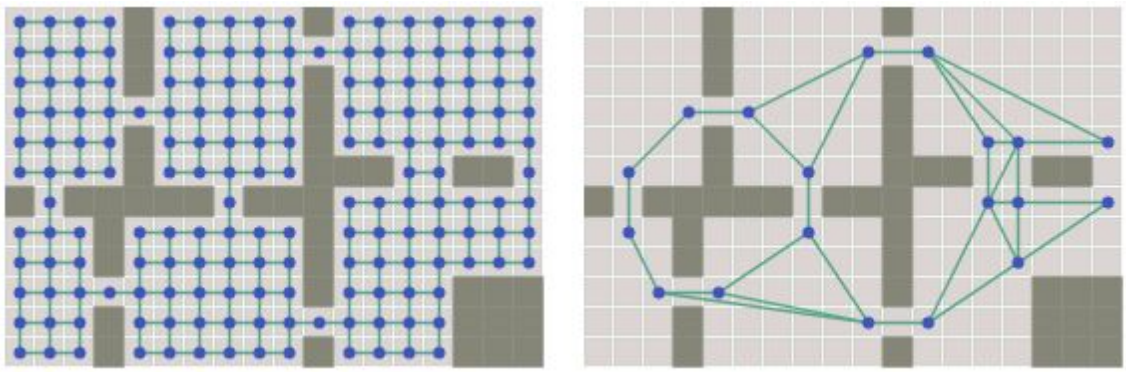

Figura 1 - Grafo da Grelha Inteira vs Grafo com Waypoints

Os waypoints são pontos em que uma pessoa poderá mudar de direção ao deslocar-se no espaço, de modo a tornar o movimento mais eficaz e realista deslocando-se em linha recta e mudando de direção quando necessário em vez de executar movimento em grelha. A utilização de waypoints permite também ter um número muito menor de nós a percorrer.

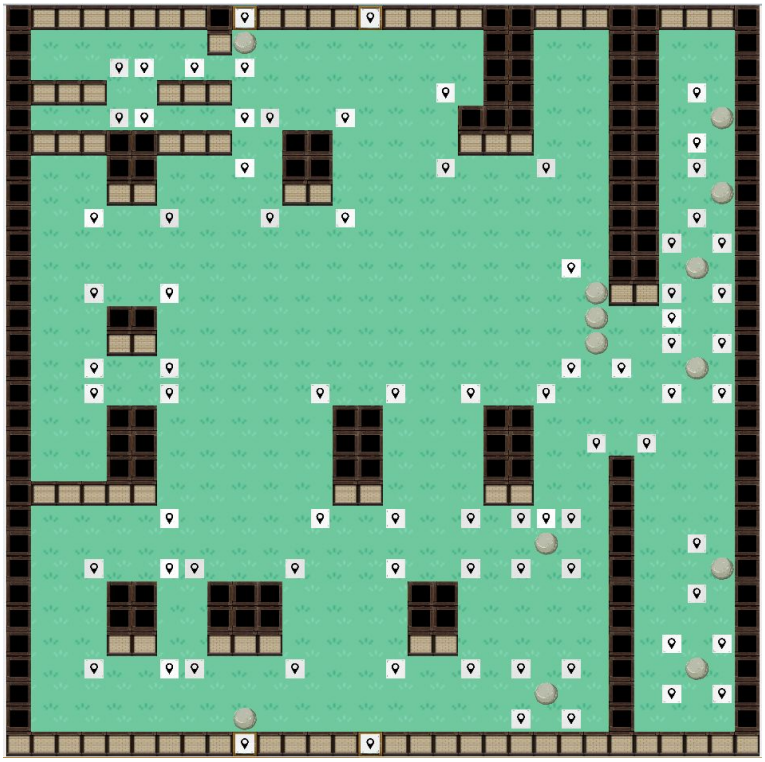

Fig 4 - Espaço da Simulação com Waypoints

A este grafo é aplicado o algoritmo de Dijkstra para construção de caminhos, o que permite ir buscar a porta mais perto depois da sua aplicação. Se um agente encontrar uma porta bloqueada esta é removida do seu grafo e o algoritmo é novamente aplicado.

Obstáculos que se encontrem a bloquear portas não produzem waypoints. Servem apenas para mostrar que a porta está bloqueada. O nó da porta contém informação sobre se a porta está bloqueada e é a essa informação que o agente acede ao encontrar a porta.

Para conectar os waypoints é utilizado o algoritmo de linhas retas de Bresenham[2], as quais são utilizadas para verificar a existência de obstáculos entre waypoints. Se não existir um obstáculo eles são conectados. Isto permite simular também a visão dos agentes, ficando eles com acesso apenas a pontos não obstruídos.

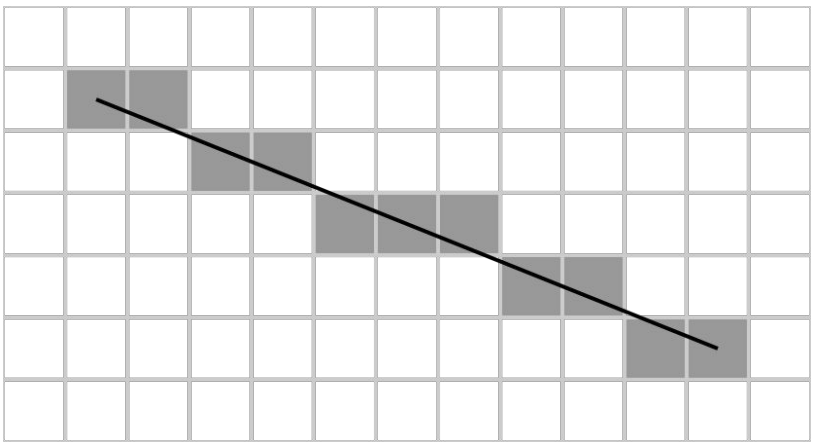

Figura 2 - Aplicação do algoritmo de Bresenham

Colisões

Em cada ciclo de movimento os agentes verificam se existem outros agentes dentro de um determinado raio. Se existir um é iniciada a gestão de colisões, em que cada agente atualiza o seu objeto com os valores necessários e lê os mesmos valores do objeto com que colidiu para decidir o que tem de fazer.

Esse objeto é colocado numa lista, a qual é verificada e atualizada a cada ciclo. Se um agente colidiu com um objeto desta lista então ainda se encontra perto o suficiente para provocar outra colisão e ignora-o. Quando já não colidir com ele retira-o da lista. Isto serve para prevenir que nos ciclos a seguir à resolução da colisão outra seja despoletada até que os agentes estejam longe o suficiente um do outro.

Experiências

Teste 1 - Configuração “Two People”

Ficheiro de espaço - space.txt
Ficheiro de posições - initialPositions.txt

Neste teste estão presentes duas pessoas, uma sem (PSC) e uma com (PCC) conhecimento do espaço, para testar movimento aleatório e movimento não aleatório.

A PCC começa na posição (1,20). A porta mais perto da sua posição está na posição (29,9). A pessoa começa por deslocar-se para essa porta mas ao chegar ao waypoint anterior à mesma verifica que essa está bloqueada por um obstáculo. Começa então a dirigir-se para a porta mais perto da sua posição atual, em (0,9).

A PSC começa na posição (27,27) e irá deslocar-se de um modo aleatório até encontrar um waypoint ligado a uma saída não bloqueada.

O tempo de teste não é previsível devido ao movimento aleatório.

Teste 2 - Configuração “Five People + One Security”

Ficheiro de espaço - space2.txt
Ficheiro de posições - initialPositions2.txt

Neste teste estão presentes 6 pessoas - 1 Segurança, 2 PCC e 3 PSC. O propósito do teste é avaliar o efeito do Segurança em pessoas que se deslocam aleatoriamente.

O Segurança começa por se deslocar para a saída mais próxima e quando encontra uma saída desbloqueada vai avisar todos os agentes que encontrou uma saída.

As duas PCC vão ignorar o aviso e continuar a deslocar-se à procura de uma saída.

As PSC vão parar o seu movimento aleatório e deslocar-se para a saída que o segurança encontrou.

Teste 3 - Configuração “Collision”

Ficheiro de espaço - space3.txt
Ficheiro de posições - initialPositions3.txt

Neste teste estão presentes 3 pessoas, todas elas PCC. O propósito do teste é a verificação da colisão entre pessoas e os seus efeitos.

O espaço foi modificado de modo a afunilar as pessoas para a mesma saída de modo a que elas sejam obrigadas a colidir, originando duas colisões.

Ao manipular as variáveis inciais dos agentes foram criados os vários cenários:

Nenhum agente é agressivo - Ao colocar o nível de pânico dos agentes a zero de modo a que nenhum deles seja agressivo é possível verificar que na primeira colisão um agente fica atordoado enquanto o outro continua o percurso. Na segunda colisão como um agente está atordoado o outro passa por ele para a saída. Não existem alterações de condição física.

Um dos agentes é agressivo - Na primeira colisão um dos agentes é empurrado para o chão, sofrendo 10 pontos na condição física e na segunda colisão é ajudado, recuperando de modo a poder mover-se.

Dois agentes são agressivos - Na primeira colisão um dos agentes é empurrado para o chão, sofrendo 10 pontos na condição física e na segunda colisão é atropelado, sofrendo 20 pontos. Como a sua condição física passa para 0, morre.

Teste 4 - Configuração “Chaos”

Ficheiro de espaço - space4.txt
Ficheiro de posições - initialPositions4.txt

Neste teste estão presentes 10 pessoas, sendo 6 PSC e 4 PCC. O objetivo é testar o comportamento dos agentes com um maior número de pessoas, resultando num cenário bastante aleatório e sendo mais difícil de prever o resultado final.

Teste 5 - Configuração “Chaos + Security”

Ficheiro de espaço - space5.txt
Ficheiro de posições - initialPositions5.txt

Neste teste estão presentes 11 pessoas, as 10 do teste 4 mais um segurança. O objetivo é testar o comportamento dos agentes com um maior número de pessoas e modificar o comportamento das PSC ao incluir um segurança, resultando num cenário aleatório e sendo mais difícil de prever o resultado final.

Conclusões

As experiências permitem concluir que cada componente do projeto funciona como projetada, tanto isolada como combinado com outras componentes.

Todos os requisitos do projeto foram implementados com sucesso, sendo a aplicação capaz de simular o comportamento de indivíduos num incidente, sendo possível simular vários estados, colisões e reunir estatísticas.

Em anos recentes SMA têm sido o método preferencial para simular movimentos de multidões em diferentes cenários visto que aproximam o problema de uma maneira realista porque permitem modelar o comportamento individual de cada pessoa com as suas características únicas e relacioná-la com outras pessoas, recriando interações entre seres humanos. Estes modelos necessitam de conhecimento e dados de interações e regras sociais, e cada pessoa é modelada como um agente autônomo com uma visão limitada do mundo. Adicionalmente a utilização de agentes BDI permite uma melhor aproximação e implementação de comportamento social e processos de tomada de decisão mais complexos.

Melhoramentos

Movimento Aleatório - O movimento sem conhecimento é completamente aleatório. Melhorias possíveis poderiam ser dar a tendência à pessoa para se deslocar numa direção tendo em conta a geometria do espaço (por exemplo, se estivesse num beco sem saída não voltaria para trás) e seguir outras pessoas.

Colisões - As colisões estão implementadas entre duas e duas só pessoas, sendo difícil de prever e avaliar a situação se várias pessoas colidirem umas com as outras num espaço pequeno ou num curto intervalo de tempo. Isto impede também a criação de multidões.

Recursos

Bibliografia

[1]"Grid Pathfinding Optimizations from Red Blob Games." *Grid Pathfinding Optimizations*. - http://www.redblobgames.com/pathfinding/grids/algorithms.html

[2]”The Beauty of Bresenham's Algorithm” - http://members.chello.at/~easyfilter/bresenham.html

Software

Jadex 2.4 - http://sourceforge.net/projects/jadex/files/jadex/2.4/
Eclipse IDE for Java EE Developers com Java 1.7 - https://eclipse.org/downloads/
Windows 7 e 8.1
Plugin ObjectAid para criação do diagrama de classes - http://www.objectaid.com/download

Elementos do Grupo

Trabalho Realizado por João Cardoso

Anexos

Anexo A - Manual de Utilizador

Instalação

Depois de importar o projeto no Eclipse é necessária a adição da biblioteca Jadex 2.4.

De modo a ativar o Jadex Control Center necessita de ser escolhida a classe Starter ao criar uma configuração de lançamento ao correr como Java Application.

Correr a Simulação

Para correr a simulação é necessário escolher dois ficheiros, o ficheiro de espaço e o ficheiro com os valores iniciais dos agentes.

Os testes da secção de Experiências indicam quais as combinações utilizadas, as quais podem ser criadas definindo as variáveis SPACE_FILE e POSITION_FILE na classe constants/Constants.

Para lançar aplicações a pasta onde elas estão deve ser acrescentada no JCC, escolhendo a opção “Add Path”. O ficheiro jar ou a pasta raiz dos packages Java (e.g. “bin”) deve ser acrescentado.