Faculdade de Ciências e Tecnologia da Universidade de Coimbra

Programação Orientada a Objetos

2014 / 2015

Simulador de Robots Móveis

Trabalho Realizado Por:

André Almeida – 2013152895

Christopher Liu – 2013150914

Introdução

O objetivo do programa é simular o comportamento de robots (agentes), com diferentes comportamentos, num ambiente dinâmico com objetos de diferentes categorias, e guardar em memória os objetos que vê e encontra, mapeando o ambiente.

Os agentes devem mover-se pelo ambiente, escolhendo um objeto dentro do seu campo de visão dependendo da sua estratégia, podendo esta ser a Estratégia Aleatória, onde vai para um objeto aleatório, a Estratégia da Máxima Diferença, onde vai para o objeto mais diferente dos que tem em memória (já visitou) e a Estratégia do Mais Próximo, onde vai para o objeto mais próximo.

Os agentes têm memória para guardar a informação obtida dos objetos visitados e objetos que já estiveram no campo de visão, sendo essa informação os atributos dos objetos e a sua posição numa matriz bidimiensional.

Existe também uma interface para, após a simulação, apresentar a memória de todos os agentes, os seus passos e estatisticas. Esta interface deve permitir a interação com o utilizador

Classes

Simulador.java (main class)

Classe que corre a simulação, lê e escreve em ficheiro.

Atributos

- int compAmbiente;
- int largAmbiente;
- int escolha;
- int id;
- int movimentos;
- ArrayList <Objeto> objetos;
- ArrayList <Agente> agentes;

Construtores

- Public Simulador();
- Public Simulador(int comp, int larg, int mov, ArrayList <Agente> x, ArrayList <Objeto> y);

Métodos

- Public static void main(String[] args);
- Public void executa();
- Public boolean verifica_coord (int [ ] x, int tipo);
- Private void popup_erro(String erro);
- Public void carregarDados();
- Private void escreveResultados();

Entidade.java

Super Classe de Agentes e Objetos. Uma entidade é algo que está no ambiente. Contém os atributos comuns a agentes e objetos, assim como os respectivos getters e setters.

Atributos

- Protected int identificador;
- Protected String cor;
- Protected String forma;
- Protected int [ ] coordenadas;

Métodos

- Setters & Getters

Objeto.java

Sub Classe de Entidade , adiciona o atributo “tipo”, que é o nome do objeto (ex: “cadeira”, “mesa”).

Atributos

- Private String tipo;

Métodos

- Setters & Getters

Agente.java

Sub Classe de Entidade e Super Classe de Aleatorio, Proximo e Diferente, contem atributos e métodos que são comuns aos diferentes tipos de agentes, como os métodos para guardar em memória e cálculo de distância percorrida pelo agente.

Atributos

- Protected int tipo;
- Protected int campovisao;
- Protected int n_passos;
- Protected int [ ] [ ] passos;
- Protected double distancia_percorrida;
- protected ArrayList <Objeto> objetos_apreendidos;
- protected ArrayList <Objeto> objetos_visitados;
- protected ArrayList <Objeto> objetos_campo;

Métodos

- Public void mover(int largAmbiente, int compAmbiente);

- Protected void passos(int [ ] x);
- Protected double distancia(int [ ] x, int [ ] y);
- Public ArrayList<Objeto> objetos_campo_visao(ArrayList <Objeto> objetos);
- Protected ArrayList<Objeto> campo_visao_sem_repeticao();
- Public int Objetos_Diferentes(ArrayList <Objeto> x);
- Public void Adiciona_Apreendidos();
- Setter & Getters;

Aleatório.java

Sub Classe de Agente, contém o método mover() que faz com que o agente escolha um objecto aleatóriamente do seu campo de visão.

Construtores

- Public Aleatorio(int x);

Métodos

- Public void mover(int largAmbiente, int compAmbiente);

Proximo.java

Sub Classe de Agente, contém o método mover() que faz com que o agente escolha o objeto mais próximo de si.

Construtores

- public Proximo(int x);

Métodos

- public void mover(int largAmbiente, int compAmbiente) ;

Diferente.java

Sub Classe de Agente, contém o método mover() que faz com que o agente escolha o objeto de acordo com o cálculo de diferenças atráves da Diferença de Hamming.

Construtores

- Public Diferente(int x);

Métodos

- Public void mover(int largAmbiente, int compAmbiente) ;

Ficheiros de Texto

“dados.txt”

O ficheiro onde se vai buscar as informações para definir o tamanho do ambiente, o número de movimentos que os agentes vão fazer, e as características dos agentes e objetos para os criar no programa.

No inicio do ficheiro tem um modelo de como deve ser configurado, sendo que os modelos do Agente e Objeto não estão preenchidos mas mostram a informação que o programa pede. Se a informação for mal introduzida ou o ficheiro mal configurado o programa não criará os objetos da maneira pretendida.

“Estatisticas.txt”

O ficheiro onde são devolvidos alguns dados relativamente ao percurso de todos os agentes.

“Objetos Visitados.txt”

O ficheiro onde são devolvidos os atributos de todos os objetos visitados por cada agente.

“Objetos Vistos.txt”

O ficheiro onde são devolvidos os atributos de todos os objetos que estiveram no campo de visão de cada agente. Não são devolvidos objetos repetidos por agente.

“Passos.txt”

O ficheiro onde são devolvidos as coordenadas ou “passos” de cada agente.

Considerações

- Quando está a criar agentes e objetos só não permite que objetos sejam criados no mesmo sitio que objetos e agentes no sitio de agentes, é possível que no inicio do programa um agente e um objeto partilhem as mesma coordenadas, uma vez que é algo que vai estar a acontecer durante o programa todo pareceu-nos por bem não limitar no incio.
- Ao atribuir um número identificativo (Entidade.identificador), estamos a começar do 1 e indo incrementando, atribuindo a agentes e objetos intercalados, à medida que são criados.

- Não nos alongámos sobre o tipo de objetos (Objeto.tipo) pois como o único objetivo é a comparação com outros objetos, e o ambiente pode variar, não criámos uma classe Tipo com “cadeiras” ou “mesas”, visto que se o ambiente simulado for a Lua ou um campo de golfe, esses objetos não fazem sentido. Dai uma string para comparar o tipo de objetos pareceu-nos a solução mais indicada.
- A maneira de guardar os passos ao longo da vida dos agentes, não querendo criar uma classe com 2 atributos (int x, int y) ou mesmo 1 (int [ ] x) para depois poder criar um ArrayList em que fossemos adicionando as novas coordenadas, resolvemos criar um Array Bidimensional com 2 linhas e o número de colunas correspondente ao número de movimentações.
- No primeiro UML tinhamos uma Classe Ambiente, que apenas tinha dois atributos. Passámos esses atributos para a Classe Simulador (main) porque a Classe Ambiente não era útil.
- Os nossos agentes só vão uma vez a cada objeto, visto que achámos caso só houvesse 1 ou 2 objetos no campo de visão o agente ficava lá até ao fim da simulação, não percorrendo o resto do ambiente, que é o objetivo da simulação.
- Os nossos agentes só guardam uma vez cada objeto “visto” no campo de visão, pois não vale a pena ter várias vezes o mesmo objeto. Atenção que comparamos o identificador e não os atributos, pois ter o mesmo objeto (identificador) e objetos iguais (atributos iguais), é diferente.
- Nós consideramos que o agente vai direto para os objetos, não se desvia de outros objetos, ou se move primeiro na horizontal e depois na vertical, logo a nossa distância não vai ser sempre um valor certo, dai termos escolhido o tipo double para a distância (Agente.distancia_percorrida).
- O nosso agente do tipo Proximo, caso tenha dois objetos à mesma distância, move-se para o objeto analisado primeiro.
- O nosso agente do tipo Diferente, caso tenha dois objetos com o mesmo número de diferenças, move-se para o objeto analisado primeiro.
- Caso os nossos agentes não tenham nenhum objeto, ou nenhum não visitado, no campo de visão, vão se movimentar 1 casa para qualquer direção, nunca saindo dos limites do ambiente ou colocando-se na posição já ocupada por outro agente.

Funcionamento de Interface

No inicio do programa damos a hipótese ao utilizador de escolher entre carregar dados do ficheiro ou introduzir as definições de ambiente, criar agentes e objetos. O botão só fica ativo, isto é, só é possivel carregar nele após ser escolhida uma das hipóteses - *Figura 1*.

Figura 2 –janela inicial da interface

Assim que for pressionado o botão, caso a escolha seja “Carregar de Ficheiro”, a janela fecha e abre a janela destinada a apresentar a informação dos agentes – *Figura 2.*

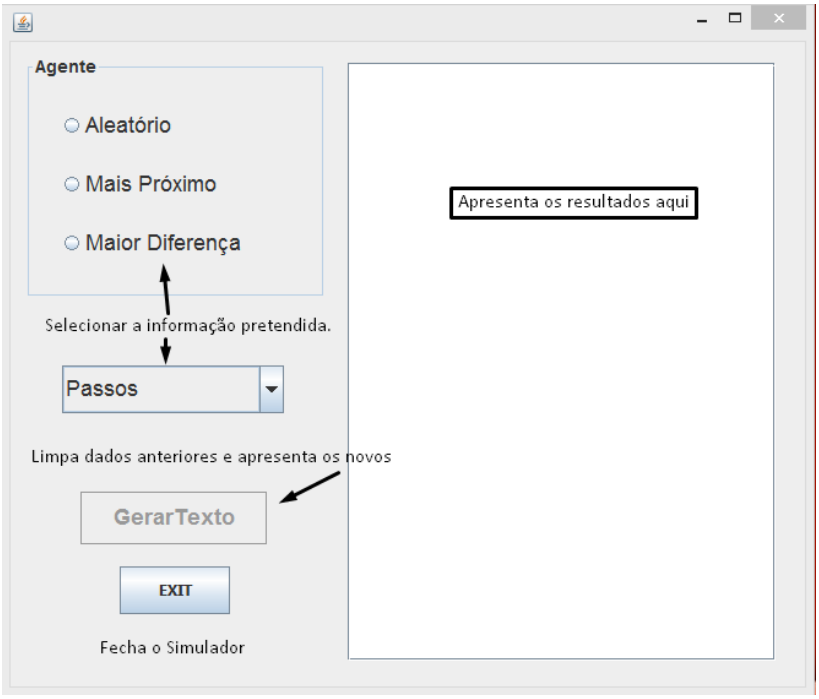


Figura 1 - janela de Resultados

Após selecionar o tipo de Agente que pretende o botão “GeraTexto” ficará ativo. Quando carregar no botão vai aparecer a informação selecionada na ComboBox de todos os agentes do tipo selecionado. Se selecionar um agente diferente, ou um tipo de informação diferente e carregar no botão os resultados serão substituidos pelos novos.

No caso de, na *Figura 1,* escolher “Introduzir Dados”, o botão “Continuar” será desativado e aparecerão novos elementos na janela – *Figura 3*.

Figura 3 - Aparecimento de Novos Elementos

O botão “Seguinte” ficará ativo assim que forem preenchidas os TextField (depois de preencher a última TextField pressione numa das outras). Se apagar uma TextField o botão desativará novamente.

Na janela seguinte poderão criar os agentes e objetos que quiserem simular, tendo que preencher os campos todos para o botão ativar (como na janela anterior,

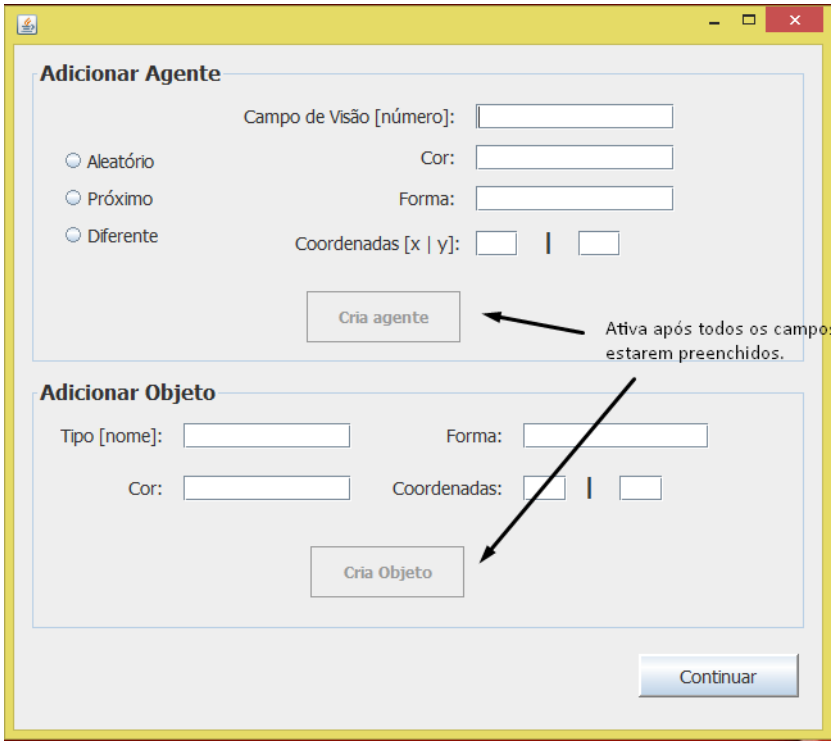


Figura 4 - Adicionar Agentes e Objetos

carregar numa das outras TextField para ativar, e apagando uma delas o botaão desativa) – *Figura 4*.

Se clicar em continuar irá para a *Figura 2* e poderá ver os resultados da sua simulação.

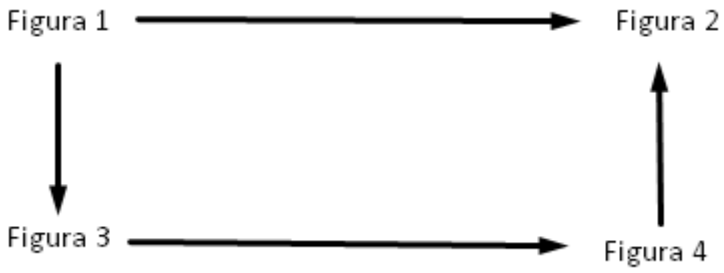
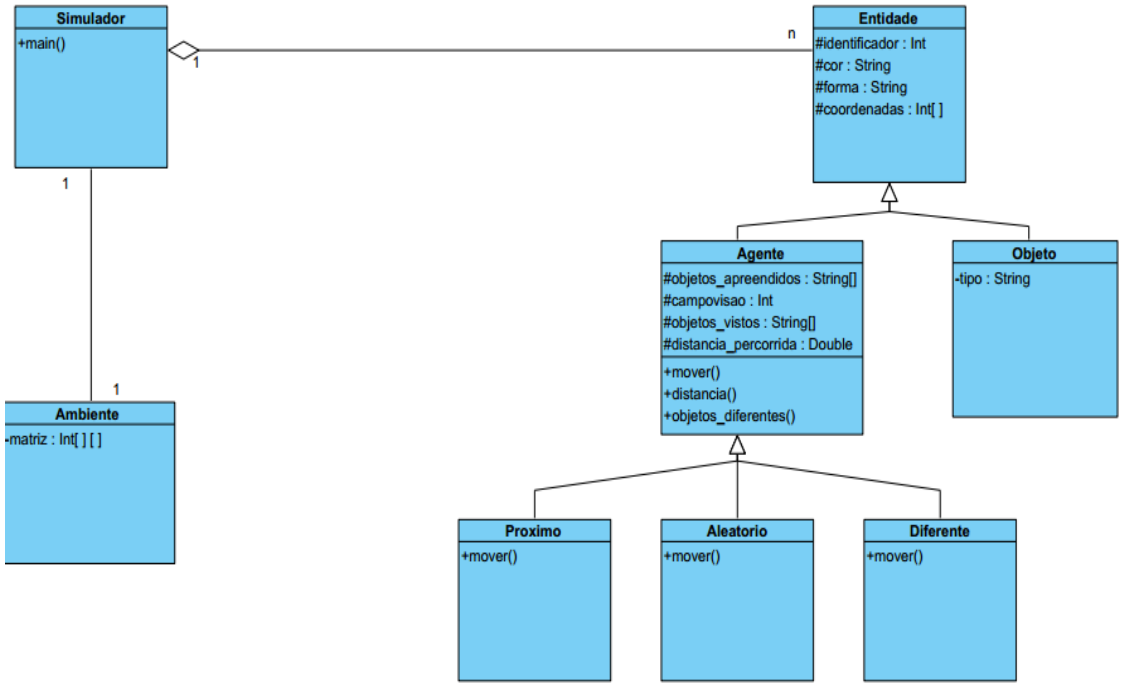


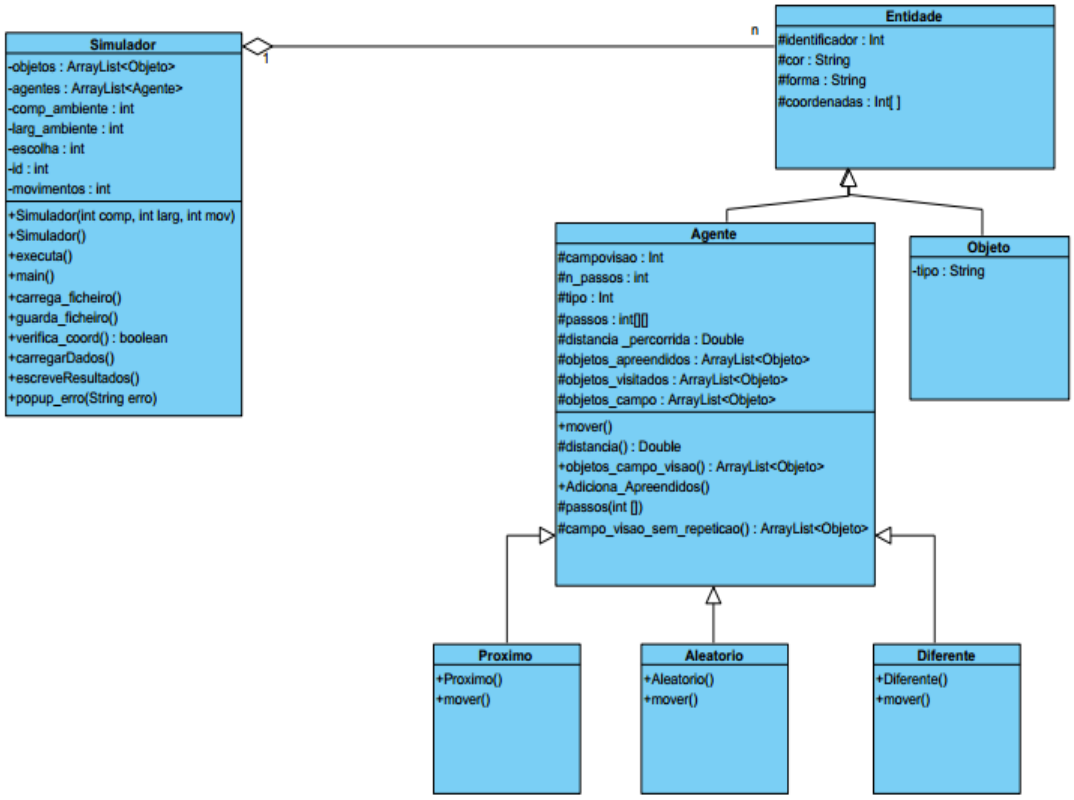
Figura 5 - Esquema Geral Janelas

Diagramas de Classe

Fase Inicial



Fase Final


Simulador
-objetos : ArrayList<Objeto>
-agentes : ArrayList<Agente>
-comp_ambiente : int
-larg_ambiente : int
-escolha : int
-id : int
-movimentos : int
+Simulador(int comp, int larg, int mov)
+Simulador()
+executa()
+main()
+carrega_ficheiro()
+guarda_ficheiro()
+verifica_coord() : boolean
+carregarDados()
+escreveResultados()
+popup_erro(String erro)
1
n
Entidade
#identificador : Int
#cor : String
#forma : String
#coordenadas : Int[ ]
Agente
#campovisao : Int
#n_passos : int
#tipo : Int
#passos : int[][]
#distancia _percorrida : Double
#objetos_apreendidos : ArrayList<Objeto>
#objetos_visitados : ArrayList<Objeto>
#objetos_campo : ArrayList<Objeto>
+mover()
#distancia() : Double
+objetos_campo_visao() : ArrayList<Objeto>
+Adiciona_Apreendidos()
#passos(int [])
#campo_visao_sem_repeticao() : ArrayList<Objeto>
Objeto
-tipo : String
Proximo
+Proximo()
+mover()
Aleatorio
+Aleatorio()
+mover()
Diferente
+Diferente()
+mover()