INSTITUTO SUPERIOR TÉCNICO

Programação Orientada por Objetos

2º Semestre 2017/2018

Relatório do Projeto

Grupo 14:

Nº 77028 – Tiago Santos

Nº 81570 – José Correia

Nº 81585 – Pedro Soares

Prof. Alexandra Carvalho

11 de Maio de 2018

Índice

I. Objetivos

Este projeto tem como principal objetivo o desenvolvimento de um simulador que permita encontrar o caminho com o menor custo de um individuo, entre um ponto inicial e um ponto final fornecidos. Deste modo, para obter então o pretendido, existem indivíduos a percorrer uma grelha de acordo com o seu respetivo conforto e, consequentemente, a executar os seus eventos permitidos: morte, reprodução ou movimento.

II. Manual de Utilizador

Para executar o programa desenvolvido deve-se utilizar o terminal e executar o seguinte comando:

java -jar grupo14.jar ficheiroxml.xml

De modo a ter uma completa descrição do comando, salienta-se que:

- Grupo14.jar é o ficheiro em que se encontra todo o código elaborado pelo grupo;
- ficheiroxml.xml é o ficheiro XML de onde vão ser lidar todas as informações para correr o programa.

Caso não seja possível processar as informações do ficheiro XML, o programa não é executado porque a validação do ficheiro *simulation.dtd* falha.

III. Packages, Classes e Interfaces

De forma a tornar o projeto organizado e extensível, optou-se por dividir todo o código por 6 *packages*, cada um deles orientado a determinadas funções especificas, implementadas em classes e interfaces, do programa elaborado. Assim:

1) *Package* events

Trata-se do *package* que inclui as classes alusivas às informações dos eventos do projecto.

a) Classe Death

Trata-se uma extensão da classe Event, de onde são herdados o tempo e o individuo. Nesta classe é implementado o evento Death, responsável pela morte de cada individuo. Na implementação é removido um elemento de uma lista e é devolvido o tempo em que o evento ocorre. Nesta classe foi necessário recorrer ao *package* java.util.*.

b) Classe Event

Trata-se uma classe abstrata. Nesta classe é implementado um evento de forma genérica, sendo depois implementado de uma forma mais especifica em cada um dos eventos. Na implementação é calculado quando ocorre e que evento acontece nesse instante e é devolvida uma *string* com essa informação. Nesta classe foi necessário recorrer ao *package* java.math.*.

c) Interface EventComparator

Trata-se de uma interface que implementa um Comparator (presente no *package* java.util.*). Nesta interface é comparado o tempo de eventos dois eventos. É devolvido um dos valores entre -1, 0 ou 1, consoante o resultado da comparação.

d) Classe Move

Trata-se uma extensão da classe Event, de onde são herdados o tempo e o individuo. Nesta classe é implementado o evento Move, responsável pelo movimento de cada individuo. Na implementação é movido um individuo na grelha, calculado o novo conforto do individuo, calculado o tempo do próximo movimento e adicionado o evento à lista de eventos. É devolvido o tempo em que o evento ocorre. Nesta classe foi necessário recorrer ao *package* java.math.*.

e) Classe Reproduction

Trata-se uma extensão da classe Event, de onde são herdados o tempo e o individuo. Nesta classe é implementado o evento Reproduction, responsável pela reprodução de um individuo, isto é, quando um individuo *parent* gera um novo individuo, *child*, na mesma posição que ele se encontra. Na implementação é identificado o *parent*, calculado o caminho e o conforto do individuo *child* e, consequentemente, proceder à sua criação. É também calculada a próxima reprodução do *parent* e os eventos (morte, reprodução ou move) do *child*. É devolvido o tempo em que o evento ocorre. Nesta classe foi necessário recorrer ao *package* java.util.

2) *Package* grid

Trata-se do *package* que inclui as classes alusivas às informações da grelha do projeto e respetivas informações associadas à mesma.

a) Classe Grid

Nesta classe é implementada a grelha por onde circulam os indivíduos. Na implementação são definidas as dimensões da grelha, a localização, quando existem, de obstáculos e zonas especiais com custo acrescido, como circular na grelha (as várias situações possíveis em que se altera a posição de um individuo) e o custo máximo de toda a grelha entre dois nós. Nesta classe foi necessário recorrer ao *package* java.util.random para gerar um caminho quando o custo para várias posições é semelhante.

b) Classe Node

Nesta classe é implementado um nó da grelha por onde circulam os indivíduos. Na implementação são definidas as coordenadas de uma posição (x,y) e as posições adjacentes com o respetivo custo para transitar para uma qualquer posição adjacente. São devolvidas as coordenadas do nó, o tipo do nó (obstáculo ou não) e os *edges* do nó. Nesta classe foi necessário recorrer ao *package* java.util.*.

3) *Package* individual

Trata-se do *package* que inclui as classes alusivas às informações da população e respetivos indivíduos do projeto.

a) Classe Individual

Nesta classe é implementado um individuo. Cada individuo está associado a um identificador, uma posição, um conforto e uma lista de caminhos. Na implementação é calculado o conforto desse individuo, o custo de um determinado caminho, a distância entre a posição inicial e final, como

adicionar um caminho, bem como a lista de caminhos e o seu tamanho. Nesta classe foi necessário recorrer ao *package* java.math.

b) Classe Population

Trata-se de uma classe que implementa uma população de indivíduos, isto é, um conjunto de indivíduos presente na grelha. Na implementação uma população está associada a um número de indivíduos, e duas listas, uma com os indivíduos vivos e outra com os indivíduos mortos. É ainda implementada a situação *epidemics*, com base numa fila de prioridades da lista de eventos. Nesta classe foi necessário recorrer ao *package* java.util.

c) Interface SortByConfort

Trata-se de uma interface que implementa um Comparator (presente no *package* java.util.*). Nesta interface é comparado o conforto de dois indivíduos. É devolvido um dos valores entre -1, 0 ou 1, consoante o resultado da comparação e, posteriormente, será usado para ordenar os indivíduos pelo seu conforto.

4) *Package* main

Trata-se do *package* que inclui a classe main, alusiva à implementação do projeto.

a) Classe Main

Trata-se da classe principal do projeto onde é implementado o *main*. Na implementação são definidos uma *string* com o nome do argumento com o nome do ficheiro XML com que se corre o programa no terminal. Na execução do simulador apenas foi necessário passar a *string* referida anteriormente para executar o programa.

5) *Package* simulator

Trata-se do *package* que inclui a classe alusiva à simulação do projeto.

a) Classe Simulator

Trata-se de uma classe que implementa um simulador do projeto. Na implementação são definidas todas as variáveis com as informações que são lidas do ficheiro XML, bem como as restantes variáveis que são necessárias para o correto funcionamento do programa ao longo da sua execução. É também criada a população inicial, os indivíduos são ordenados por conforto, são executados os eventos que se realiza e, posteriormente, estes são adicionados à lista de eventos.

6) *Package* utilities

Trata-se do *package* que inclui as classes alusivas às funcionalidades utilitárias ao funcionamento do programa em si.

a) Utils

Trata-se de uma classe de utilitários. É implementação é calculado um parâmetro genérico utilizado nos eventos, uma vez que a fórmula de cálculo em cada evento. É devolvido o resultado do cálculo referido. Nesta classe foi necessário recorrer ao *package* java.math.*.

b) XMLFileParser

Trata-se de uma classe que lê o ficheiro XML indicado no terminal. Na implementação, são lidas todas as variáveis presentes no ficheiro e indexadas num determinado tipo de dados, consoante o caso. Salienta-se que o ficheiro XML é verificado com o ficheiro *simulation.dtd*. Nesta classe foi necessário recorrer ao *package* javax.xml.parsers.DocumentBuilderFactory.

IV. Modo de Funcionamento

Após o comando para correr o programa no terminal são lidos do ficheiro XML indicado nos argumentos, se este estiver de acordo com o ficheiro *simulation*.dtd: o instante final de tempo ***finalinst***; a dimensão da população inicial ***initpop***; o valor máximo de indivíduos da população ***maxpop***; a sensibilidade do conforto ***comfortsens***; a dimensão da grelha, através de ***grid***; as coordenadas da posição (x,y) do ponto inicial do individuo através de ***initialpoint***; as coordenadas da posição (x,y) do ponto final do individuo através de ***finalpoint***; o número de zonas com custo especial e as respetivas coordenadas das suas posições (x,y) através de ***specialcostzones***; o número de obstáculos e as respetivas coordenadas das suas posições (x,y) através de ***obstacles***; os parâmetros de cada um dos eventos (morte, reprodução ou movimento) através de ***events***. Estes valores são inseridos nos respetivos tipos de dados na classe *simulator* e, posteriormente, nos construtores associados ao Individual, à *Grid* ou aos respetivos cálculos intermédios.

A partir da classe *simulator*, é criada uma população inicial, isto é, são criados ***initpop*** indivíduos no ***initialpoint*** e é calculado o seu respetivo conforto. Posteriormente a lista dos indivíduos é ordenada pelo conforto (para facilitar a situação em que se dá uma epidemia) e são processados os eventos com base nos cálculos do conforto e, consequente, nos parâmetros de cada evento e no caminho percorrido por cada individuo.

Durante o programa é suposto ir sendo apresentado o *observation number*, com alguns parâmetros no terminal de acordo com o pedido no enunciado do projeto e, ao chegar à posição final, é apresentado o *Path of the best fit individual*.

V. Conclusões

Ao final este projeto verificou-se que foi conseguida a implementação de praticamente todas as funcionalidades pretendidas. Foi observado que o programa correu como pretendido, mas poder-se-iam ter implementado algumas funcionalidades extra e/ou algumas alterações para melhorar o funcionamento e a eficácia do programa. Uma dessas funcionalidades de forma a melhorar o desempenho do programa é o facto de o caminho poder ser intercetado por si próprio e neste caso poder-se-ia "cortar" a parte do caminho entre a primeira passagem pela interseção e a segunda.

Para testar todo o programa foram testados quer o exemplo fornecido através da página da disciplina quer os ficheiros criados pelo grupo, através dos quais foi possível concluir que o programa apresentou resultados conceptualmente expectáveis e de acordo com o que se esperava ao longo da elaboração do código do programa.

É de salientar também que se tentou garantir que o código do projeto está extensível a novos desenvolvimentos futuros e que se tentou implementar um vasto número de conceitos abordados na disciplina, de forma a enriquecer o conhecimento do grupo em Java.