

Concept. et prog. objet


\1 TP n°5 : sim's country

Dépublié

Détails

Écrit par stéphane Domas

Catégorie : M3105 - Concept. et prog, objet avancée (/index.php/menu-cours-s3/menu-mmi3-test)

 Publication : 23 octobre 2014

 Affichages : 695

1°/ Objectif

- Le moteur de jeu des TP's précédent est modifié pour intégrer un réseau de villes, chaque ville abritant une population.
- A chaque tour, une partie des personnes (par famille notamment) de chaque population va migrer d'une ville à une autre.
- Pour cela, on va lire le maillage constitué par les villes dans un fichier texte et constituer ainsi un graphe orienté pondéré.
- Du côté des humains, on va ajouter la notion de famille en permettant à chaque humain de référencer ses ascendants et descendants.

2°/ Exercices

2.1°/ Le graphe des villes

- Créez le fichier `Vertex.java`, à partir du code suivant :

```
1  import java.util.*;
2
3  class Vertex {
4
5      Set<Edge> edgesFrom; // set of edges that come from other vertices
6      Set<Edge> edgesTo; // set of edges that go to other vertices
7      Population pop;
8      String name;
9
10     public Vertex(Population pop, String name) {
11         edgesFrom = new HashSet<Edge>();
12         edgesTo = new HashSet<Edge>();
13         this.pop = pop;
14         this.name = name;
15     }
16
17     public void setPopulation(Population pop) {
18         this.pop = pop;
19     }
20
21     public String toString() {
22         return name+" : "+pop;
23     }
24 }
```

- Créez le fichier `Edge.java` à partir du code suivant :

```
1  class Edge {
2
3      Vertex from;
4      Vertex to;
5      double weight;
6
7      public Edge(double weight) {
8          from = null;
9          to = null;
10         this.weight = weight;
11     }
12
13     public Edge(Vertex from, Vertex to, double weight) {
14         this.from = from;
15         this.to = to;
16         this.weight = weight;
17     }
18
19     public String toString() {
20         return "edge from "+from.name+" to "+to.name;
21     }
22 }
```

- Créez le fichier Graph0P.java à partir du code suivant :

```

1  import java.util.*;
2
3  class GraphOP {
4
5      public List<Vertex> vertices;
6      public int nbVertices;
7      public int nbEdges;
8
9      public GraphOP() {
10         vertices = new ArrayList<Vertex>();
11         nbVertices = 0;
12         nbEdges = 0;
13     }
14
15     public Vertex createVertex(Population pop, String name) {
16         Vertex v = new Vertex(pop, name);
17         vertices.add(v);
18         nbVertices += 1;
19         return v;
20     }
21
22     public void connectVertices(Vertex src, Vertex dest, double weight) {
23         Edge e = new Edge(src,dest,weight);
24         src.edgesTo.add(e);
25         dest.edgesFrom.add(e);
26         nbEdges += 1;
27     }
28
29     public boolean disconnectVertices(Vertex src, Vertex dest) {
30         boolean connOk = false;
31         for( Edge e : src.edgesTo) {
32             if (e.to == dest) {
33                 connOk = true;
34                 src.edgesTo.remove(e);
35                 dest.edgesFrom.remove(e);
36                 break;
37             }
38         }
39         return connOk;
40     }
41
42     public void createFromFile(String fileName) throws IOException {
43         // à compléter
44     }
45 }

```

- La méthode `createFromFile()` (à compléter) permet de créer un graph à partir du contenu d'un fichier texte, contenant par exemple :

1	7
2	1,Bordeaux
3	2,Lyon
4	3,Marseille
5	4,Paris
6	5,Rennes
7	6,Strasbourg
8	7,Toulouse
9	1,2,700
10	1,3,500
11	1,4,750
12	1,7,200
13	2,1,700
14	2,3,300
15	...

- La première ligne contient le nombre de ville N, chacune étant une instance de Vertex dont l'attribut name est le nom de la ville. Lors de l'instanciation, la ville est créée avec une population nulle.
- Les N lignes suivantes sont au format id,nom_ville, sachant que les ids ne sont pas forcément dans l'ordre mais sont tous compris entre 1 et N. **Attention** : la ville idX doit être à l'index X-1 dans l'ArrayList de GraphOP.
- Viennent enfin des lignes au format id1,id2,distance qui donnent le kilométrage (= poids d'une arête) entre la ville d'id1 et celle d'id2.
- Le nombre de ces lignes est indéterminé puisqu'il n'existe pas forcément une connexion entre toutes les villes.

2.2°/ Modification des classes de base

- Pour stocker les ascendants et desendants d'un humain et faciliter la gestion du jeu, on modifie la classe Humain comme suivant :

```

1  abstract class Humain {
2      ...
3      boolean dead;
4
5      // ajouts attributs d'ascendance/descendance
6      protected Humain pere;
7      protected Humain mere;
8      protected Set<humain> enfants;
9      ...
10     // modif méthodes
11     public void vieillir() {
12         age++;
13         if (age > esperanceVie) dead = true;
14     }
15     public boolean isDead() {
16         if ((dead)|| (age > esperanceVie)) return true;
17         return false;
18     }
19
20     // ajout méthodes
21     public void mourrir() {
22         dead = true;
23         // à compléter
24     }
25     public void setParents(Humain pere, Humain mere) {
26         // à compléter
27     }
28
29     public void addChild(Humain enfant) {
30         // à compléter
31     }
32
33     public Set<Humain> getAscendants(int nbLevel) {
34         // à compléter
35     }
36
37     public Set<Humain> getDescendants(int nbLevel) {
38         // à compléter
39     }
40 }

```

- La méthode mourrir() doit permettre de supprimer l'objet courant comme père, mère ou enfants d'autres humains : si l'humain courant a un père/une mère h encore en vie, alors il faut l'enlever de la liste des enfants de h. Ou bien s'il a des enfants, pour chacun, il faut mettre son attributs père/mère (selon le sexe) à null.
- Le paramètre nbLevel des méthodes getAscendants() et getDescendant() permet de regler la profondeur d'exploration des ascendants/descendants. Par exemple, pour un humain h, si nbLevel = 1, le set renvoyé contient respectivement les parents ou les enfants de h. Si nbLevel = 2, le set renvoyé contient les parents+grand-parents ou bien les enfants+petits enfants. Etc.
- D'autre part, il faut modifier les méthodes rencontre() de Homme et Femme : lorsqu'une naissance a lieu, le bebe doit avoir comme parents les 2 humains impliqués dans la rencontre ET ajouter le bebe comme enfant du père et de la mère.

2.3°/ Le moteur du jeu

Copiez/collez le code de TP4.java dans TP5.java et modifiez TP5.java pour que :

- la méthode main() contienne une nouvelle variable GraphOP towns;
- le paramètre args[2] soit pris en compte comme le nom du fichier texte contenant la description du graphe des ville (cf. section 2.1)
- ce fichier permette d'initialiser towns.
- pour chaque ville de towns, on crée une population initiale comme dans les TPs précédents.
- à chaque tour de jeu :
 - pour chaque ville, on organise les rencontres comme dans les TPs précédents.
 - pour chaque ville :
 - on tire aléatoirement un nbMove compris entre 0 et 3.
 - pour i de 0 à nbMove :
 - on choisit aléatoirement un humain h dans la population,
 - on tire aléatoirement un entier nbLevel entre 1 et 3,
 - on récupère les sets des ascendants/descendants de niveau nbLevel de h, puis on met les 2 sets ET h dans un même Set<Humain> famille.
 - on tire aléatoirement une ville de destination parmi les villes **directement** connectées à la ville courante.
 - on déplace famille dans la population de cette ville. Mais comme les routes sont dangereuses, tirez aléatoirement un entier nbMorts entre 0 et min (3,taille famille), et ensuite tirez nbMorts humains dans famille et faites les mourrir (sans oublier de les enlever de famille)
 - on fait vieillir la population.
- Pour implémenter ce moteur de jeu, il est conseillé de définir de nouvelles méthodes. A vous de trouver lesquelles et où les définir.