Licence Informatique : Projet Java Objet

Le but du projet est la simulation d'une surveillance incendie dans un entrepôt matérialisé par un plan de dimension nl × nc. Cet entrepôt contient du papier rangé par blocs séparés entre eux par des allées. La surveillance est assurée par un ensemble de robots pouvant se déplacer dans les allées. Chaque robot est équipé d'un émetteur-récepteur infrarouge et d'un émetteur récepteur à ondes courtes. Chaque robot émet en infrarouge sur une longueur d'onde λ_0 et reçoit sur 3 longueurs d'onde λ_0 λ_1 λ_2 : un incendie émettant dans un spectre large, chaque robot peut faire la différence entre l'émission infrarouge d'un autre robot et un départ de feu (dans le premier cas la réception sera effective uniquement sur λ_0 , dans le second cas la réception interviendra sur les trois longueurs simultanément). La transmission infrarouge se fait uniquement à vue (tout obstacle empêche un robot de "voir" un incendie ou un autre robot). La transmission radio ne présente pas cet inconvénient. Le projet devra simuler en Java ces protocoles de communication.

Principe:

Chaque robot possède 2 états: veille et alerte

Dans le mode veille, chaque robot évolue dans les allées de l'entrepôt; arrivé à un croisement, il se dirige aléatoirement dans une des directions possibles. Dès qu'il aperçoit un autre robot (réception sur la longueur d'onde λ_0), il cherche à éviter ce robot. Ce comportement garantit une dispersion des robots dans l'espace à surveiller.

Le mode alerte est enclenché dès qu'un robot perçoit un incendie au cours des rondes précédemment décrites. Ce robot lance alors un signal radio faisant basculer les autres robots en mode alerte: au lieu de fuir, ils cherchent à rejoindre le robot ayant déclenché l'alerte. En effet, la détection par un seul capteur peut conduire à une erreur, il est donc important de rassembler plusieurs détecteurs (i.e. robots) pour confirmer un départ d'incendie. Ce seuil sera fixé à 3 dans ce projet.

Implémentation:

L'ensemble des éléments de la simulation prend place sur un damier nl × nc. La représentation est orientée multi-agents: les robots, l'incendie et les carrefours sont des agents (i.e. des objets autonomes) ayant chacun une représentation et un comportement adaptés à leur rôle.

```
class Agent{
   int pos-x,pos-y;
   String aff;
   String affichage(int i, int j)
   void cycle()
}
```

- Agents de type **robot** : (représentés par "a" "b" "c" "d" , … sur le plan)
  Ils se déplacent aléatoirement en mode veille, ils contournent les obstacles. Leur représentation peut passer en majuscule lorsqu'ils aperçoivent un autre robot.

```
class Robot extends Agent{
   int dir;
   char etat;
   void veiller()
   void alerter()
}
```

- Agents de type **carrefour** : (non représentés sur le plan)
  ils matérialisent les intersections (caractérisées par leur position dans le plan, leur largeur et leur hauteur), la méthode **`init()`** appelée au tout début génère les voies correspondantes stockées dans la variable globale **`LA`**. Le stock de papier (matérialisé par des "X") se trouve donc là où il n'existe pas de voies. Il existe 5 formes de carrefour, la première correspond à un carrefour à 4 voies, les 4 autres correspondant à une voie allant dans une de ces 4 directions: nord, sud, est et ouest (dans ce cas, la variable **`fin`** fixe la fin de la voie ainsi ouverte).

```
class Carrefour extends Agent{
   int fin;
   int largeur, hauteur;
   boolean voie;
}
```

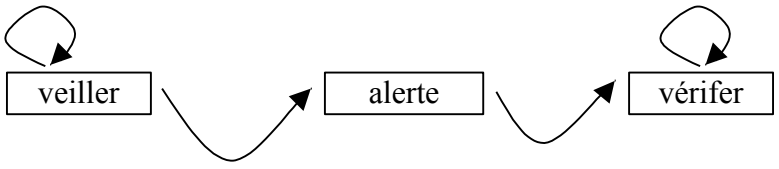
- Agent de type **incendie** : (représenté par "o" ou "O" sur le plan)

  il se déclenche aléatoirement ou après un nombre fixé de cycles.

```
class Incendie extends Agent{
   int log-cycle-dec;
}
```

La représentation du temps:
L'enchaînement des actions est assuré par l'appel de la méthode **`cycle()`**. Un cycle représente une unité de temps, toutes les mesures temporelles sont établies en nombre de cycles (ex. le nombre de cycles se déroulant avant le départ de l'incendie – donné par la variable **`long-cycle-dec`**). Tout agent possède une variable **`horloge-i`** (pour horloge interne).
La gestion des cycles :
La méthode **`cycle()`** est envoyée à tous les agents. La méthode correspondante incrémente l'horloge interne de l'agent et active sa méthode courante. Le comportement d'un agent est donc décrit par un ensemble de méthodes implémentant la graphe des étapes de la vie de l'agent; chacune de ces méthodes renseigne sur le nom de la prochaine méthode à activer (elle est stockée dans la variable **`etat`**).

La vie d'un agent est une succession d'étapes (à chaque étape correspond une méthode); voici l'exemple de l'agent robot:



La méthode **`init()`:**
ATTENTION! Il n'existe pas de matrice nl × nc dont les éléments sont les agents. Il existe seulement une liste **`LA`** des agents (carrefours, robots, incendie), les voies de circulations sont des extensions des carrefours (ce sont donc des agents "carrefour" dont la variable booléenne **`voie`** est **`true`** et ils sont générés par la méthode **`init()`**)
La méthode **`affiche()`** :
Elle provoque l'affichage du plan, le point de coordonnées 1×1 étant en haut et à gauche. Pour cela, on applique successivement la méthode **`caractereImprimable(i,j)`** dans le cadre d'un double balayage ligne / colonne.

La méthode **`caractereImprimable(i,j)`** retourne une chaîne de longueur 1 (ou 3) correspondant à l'occupation de la case i × j :

robot : **a, b, c, …**
incendie : o
papier : X
voies: " "

Remarques:
Par défaut, la largeur et la hauteur d'un carrefour sont égaux à 1, les voies issues de ce carrefour ont donc une largeur de 1 (les robots ne peuvent donc pas se croiser).
Dans l'état "veille", un robot évite les autres robots dès qu'ils sont visibles. Deux solutions se présentent: soit le robot voit uniquement ce qui est devant lui (variable **dir**), soit le robot voit à 360°.

Interface graphique :
Nous utiliserons les composants du package **swing** (composants à importer en début de programme). Principe: associer une classe graphique à l'objet devant être visualisé. Ex.

Manager ⇔ ManagerGUI

Le constructeur de la classe **ManagerGUI** construit l'interface, la classe propose des méthodes pour interagir avec elle (via l'instance de **ManagerGUI** créée).

Ex.

```
class Manager extends Agent{
   static JFrame Affichage;
   Manager(){
      ManagerGUI mGUI = new ManagerGUI();
      mGUI.setVisible(true);
   }
}
class ManagerGUI implements ActionListener{
  ManagerGUI(int n){
       init();
       JButton bInit = new JButton("Init");
       JButton bPaP = new JButton("Pas à Pas");
       JButton bResol = new JButton("Resolution");
       GridLayout d = new GridLayout(n+1,n);
       JFrame fenetre = new JFrame("Projet Prolog L3");
       JPanel affichage = new JPanel();
       JPanel commande = new JPanel();
       Container c = fenetre.getContentPane();
       c.setLayout(d);
       commande.add(bPaP);
       commande.add(bInit);
       commande.add(bresol);
       c.add(commande);
       bInit.addActionListener(this);
       bPaP.addActionListener(this);
       affichage.setLayout(d);
       c.add(affichage);
       Manager.Affichage = fenetre;
    }

  public void actionPerformed(ActionEvent e){
       String s = e.getActionCommand();
       if (s.equals("Init")){
          System.out.println("Init");
       }
   }
}
```

La construction d'une interface fait appel aux classes `JFrame`, `JPanel`, `JButton`, `JTxtField`, `GridLayout`, `GridBagConstraints`. Le principe est basé sur l'utilisation d'un conteneur: un objet fenêtre (instance de `JFrame`) contient des objets graphiques accessibles grâce à la méthode

`getContentPane()` de la classe `JFrame`. Cet objet (instance de la classe `Container`) peut être enrichi de composants graphiques avec la méthode `add()`. La disposition des objets dans le containeur est assurée par un gestionnaire de disposition; il en existe deux types selon qu'il est instance de `FlowLayout` ou `GridLayout`. L'utilisation du second type dispose les objets dans une grille virtuelle dont les dimensions sont déclarées lors de l'instanciation du gestionnaire.

Une des façons d'assurer l'interactivité est de faire correspondre une action aux boutons de l'interface. C'est le rôle du gestionnaire d'action (`Actionlistener`), la classe précédente doit alors implémenter l'interface `ActionListener` et la méthode `actionPerformed(ActionEvent e)` doit être redéfinie. La méthode `addActionListener` de la classe `JButton` permet d'ajouter ce gestionnaire d'action. L'importation du package `java.awt.*` est nécessaire.

Il vous est demandé de compléter le comportement des agents de type robot de telle sorte qu'un départ d'incendie soit annoncé dans un délai inférieur à `max(nl,nc)` cycles (avec max(nl,nc) = 2 × nl × nc). Le plan d'évolution des robots sera d'au moins 30 × 30. On pourra réaliser une interface graphique avec 3 boutons: "Init", "Pas à Pas", "Detection" (le programme s'arrête à la détection de l'incendie).

Annexe

```
package projet_16_16_Incendiev1;
import java.util.*;

public class TestIncendie {
   public static void main(String[] args) {
      Robot r1 = new Robot(4,1,'o',"_a_");
      //Robot r2 = new Robot(8,4,'e',"_b_");
      Carrefour c1 = new Carrefour(4,4,'v',0);
      System.out.println(Agent.LA);
      Carrefour.init();
      System.out.println(Agent.LA);
      Agent.affiche();
   }
}

abstract class Agent{
   static int nl=6;
   static int nc=6;
   static List LA = new LinkedList();
   int pos_x;
   int pos_y;
   String aff;

   abstract void cycle();
   static void affiche(){
      for(int i=1;i<=nl;i++){
         System.out.print(i+" ");
         for(int j=1;j<=nc;j++)
            System.out.print(caractereImprimable(i,j));
         System.out.println();
      }
   }
   static String caractereImprimable(int i,int j){
      String r="_X_";
      Agent a;
      for(int k=0;k<LA.size();k++){
         a=(Agent)LA.get(k);
         if(a.pos_x==i && a.pos_y==j)return a.aff;
      }
      return r;
   }
   public String toString(){
      return getClass().getSimpleName()+" "+pos_x+" "+pos_y+" / ";
   }
}
class Robot extends Agent{
   char etat;
   char dir;
   void cycle(){

   }
   Robot(int i,int j,char dir,String n){
      pos_x=i;pos_y=j;
```

```
      this.dir=dir;
      aff=n;
      LA.add(this);
   }
}
class Carrefour extends Agent{
   int larg;
   int haut;
   int fin;
   char type;
   boolean voie;
   void cycle(){}
   Carrefour(int i,int j,char t,int f){
      aff="   ";
      pos_x=i;pos_y=j;
      larg=1;haut=1;
      type=t;
      fin=f;
      voie=false;
      LA.add(this);
   }

   static void init(){
      Carrefour c,cc;
      int s = LA.size();
      for(int i=0;i<s;i++){
         if(LA.get(i) instanceof Carrefour){
            c=(Carrefour)LA.get(i);
            if(c.type=='v'){//cas d'un carrefour 4 voies
               for(int k=c.pos_x+c.larg;k<=nc;k++){
                  cc=new Carrefour(c.pos_x,k,'v',0);cc.voie=true;
               }
               for(int k=c.pos_x-1;k>0;k--){
                  cc=new Carrefour(c.pos_x,k,'v',0);cc.voie=true;
               }
               for(int k=c.pos_y+c.haut;k<=nl;k++){
                  cc=new Carrefour(k,c.pos_y,'v',0);cc.voie=true;
               }
               for(int k=c.pos_y-1;k>0;k--){
                  cc=new Carrefour(k,c.pos_y,'v',0);cc.voie=true;
               }
            }
         }
      }
   }
}
```