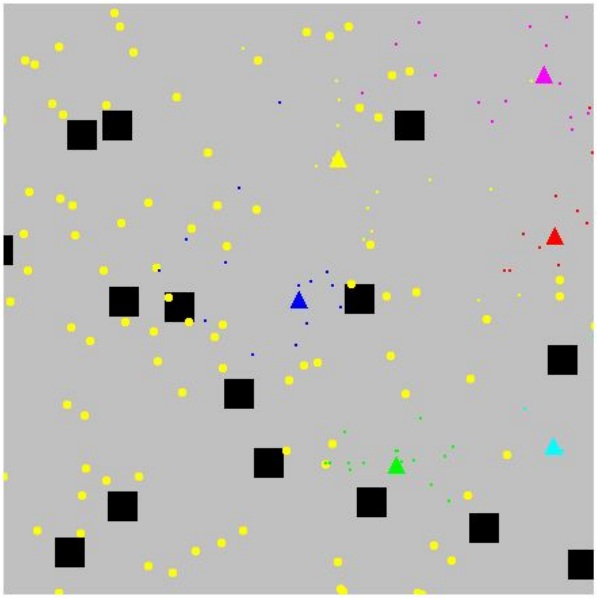
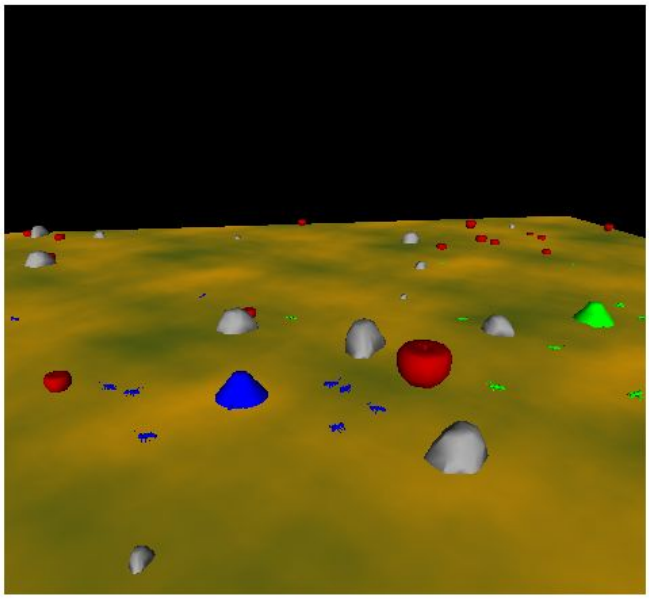
Projet LO43

Simulation de fourmilières





Luet Nicolas
Couvidou Laurent

Rendu le mercredi 16 juin 2004

Plan

Introduction

Ce projet à pour objectif le développement d'une **simulation de fourmilières**, dans un contexte compétitif. Les fourmis issues des différentes fourmilières se déplacent sur un terrain parsemé d'**obstacles** et de **nourriture**. On distinguera quatres types de fourmis différentes :

- **Les ouvrières :** elles doivent ramener à leur nid le maximum de nourriture qu'elles peuvent porter, et indiquer aux autres ouvrières le chemin à suivre vers leurs source à l'aide de phéromones ;

- **Les soldats :** leur but est de trouver les fourmis adverses et de les éliminer ;

- **Les reines :** quand une fourmilière a stocké une certaine quantité de nourriture, sa reine peut donner naissance à une larve ;

- **Les larves :** elles restent à la fourmilière pendant un temps donné (mutation), et se transforment soit en ouvrière soit en soldat, selon ce dont a besoin la fourmilière.

Chaque fourmi qui se déplace doit être capable d'éviter les obstacles et de retrouver sa fourmilière d'origine.

Le programme doit être écrit en **Java**. Naturellement, il doit être conçu de telle manière que des extensions puissent s'ajouter facilement.

Nous disposons d'une base de départ réalisant l'ensemble de l'affichage 2D (directement avec les paquetages Java) et 3D (par le biais d'OpenGL), et disposant des classes de base. On se concentre donc sur la réalisation de la simulation de fourmilières proprement dite.

Ce rapport a pour but de présenter les phases d'**analyse** (le Quoi) et de **conception** (le Comment) menées pour ce projet. Nous précisons aussi le mode d'emploi du programme, et en ajoutons les sources de la partie réalisée en annexe.

1) Analyse

On pose ici les questions **essentielles** du projet. À quel résultat final sommes-nous arrivés ? Quelle structure a-t-il, aussi bien du point de vue objet, que du point de vue fonctionnel ou encore du point de vue dynamique. Comment cette structure a-t-elle évolué par rapport à l'état initial ?

C'est ici que nous exposons la **méthode** suivie pour développer notre projet, méthode que nous avons voulu la plus; logique possible.

a) Spécification des besoins

Avant de décrire l'état final auquel nous sommes arrivés, voyons d'abord quel était l'**état initial** de la simulation lorsque nous avons commençé.

Tel que on nous l'a fournie, la simulation permettait :

- La **création** d'un nombre donné de fourmilières, de fourmis, de nourriture et d'obstacle ;
- La gestion du **déplacement** des fourmis, avec **contournement** des obstacles et des limites du terrain ;
- L'**affichage** 2D et 3D.

Pour obtenir une simulation satisfaisante, nous avons estimé nécessaire de développer les extensions suivantes :

- Création de **différentes types d'objets** : on distingue les objets visibles (fourmilière, fourmi, nourriture et obstacle) des objets invisibles (phéromones). Notons que nous pourrions assimiler tous les objets invisibles à des phéromones. Mais procéder ainsi nous empecherais de créer d'éventuels autres types d'objets invisibles : jets d'acide, odeurs, chaleur/froid, etc.
- Création de **différents types de fourmis** : on distingue les ouvrières, les soldats, les reines et les larves. Leur rôles ont déjà été présentés en introduction.
- Gestion de la **collecte de nourriture** : une ouvrière doit pouvoir détecter la nourriture, et en collecter. Pour cela, l'ouvrière doit pouvoir se charger en nourriture, et la nourriture doit contenir un stock limité à fournir.
- Gestion de la **dispartion de la nourriture** : quand une ouvrière vide un stock durant sa collecte, elle doit repartir et la nourriture doit disparaître du terrain.
- Gestion du **retour au nid** : quand une ouvrière a fini de collecter de la nourriture, elle doit être capable de retourner par le plus court chemin à sa fourmilière, tout en contournant les éventuels obstacles. On suppose, comme précisé dans l'énoncé, qu'une fourmi sait toujours la position de la fourmilière à laquelle elle appartient.

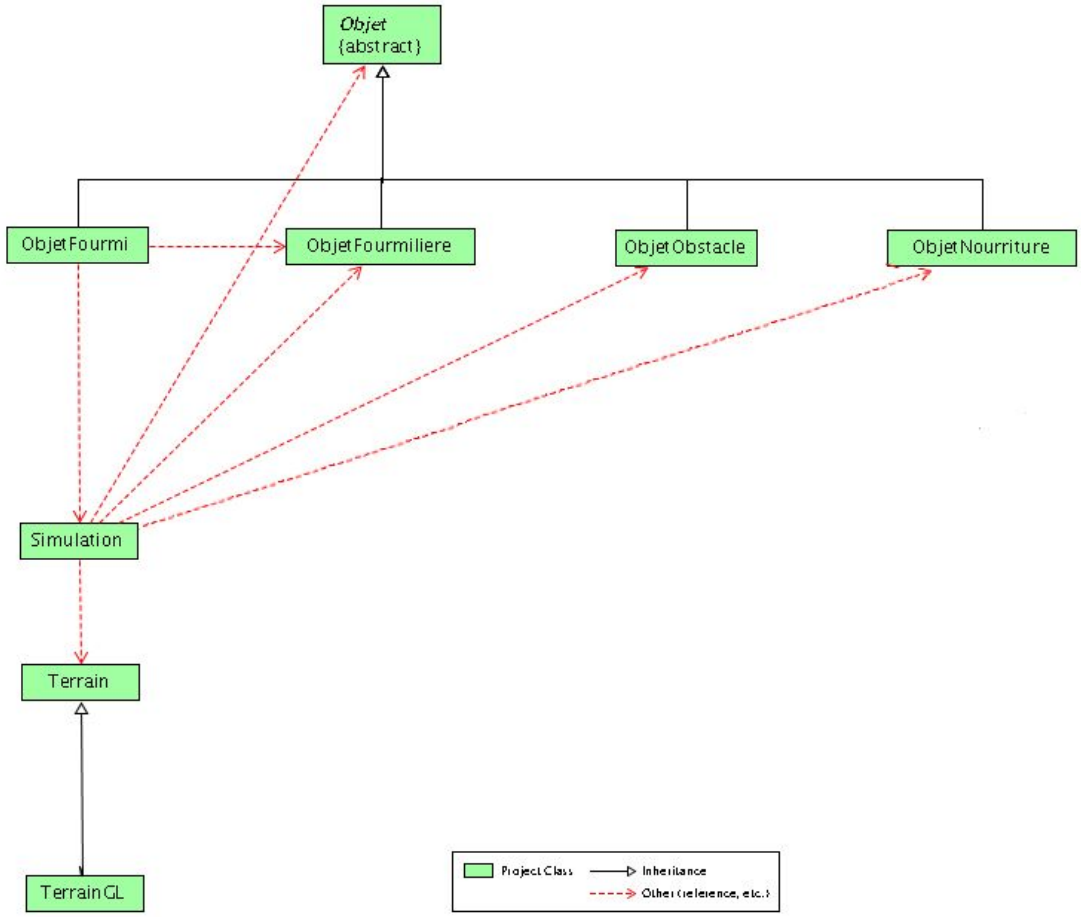
- Gestion des **phéromones** : quand une ouvrière retourne à son nid, elle dépose sur son trajet une trace moléculaire invisible. Ainsi, elle indique aux autres ouvrières l'emplacement de la nourriture qu'elle a trouvé.

- Gestion du **stock des fourmilières** : à l'instar des sources de nourriture, les fourmilières disposent d'un stock. Quand une ouvrière arrive chargée au nid, elle dépose sa charge, attends quelques instants, puis repart à vide.

- Gestion de l'**incubation** : chaque fourmilière est dirigée par une reine, qui peut incuber au maximum trois larves. Elle lance l'incubation d'une larve uniquement si la fourmilière dispose d'assez de nourriture pour cela, et ne manque pas d'enlever au stock la quantité corrrespondante. Une larve prends un certain temps à incuber, et se transforme en ouvrière une fois ce temps passé.

b) Diagramme de classes

On aborde ici le problème de la **modélisation objet** de notre projet. Il nous faut décrire les différentes entités qui le composent et les relations qui les lient. Nous n'avons pas eu à travailler à partir de rien puisque la simulation de départ fournissait déjà une structure efficace. Voici sa représentation UML[1] (en faisant abstraction de l'affichage) :



1 Obtenue avec **jgrasp** (http://www.jgrasp.org/) à partir des sources

Comme l'indique la légende (malheureusement peu lisible et en anglais) :

- Les héritages sont en noir
- Les dépendances sont en rouge

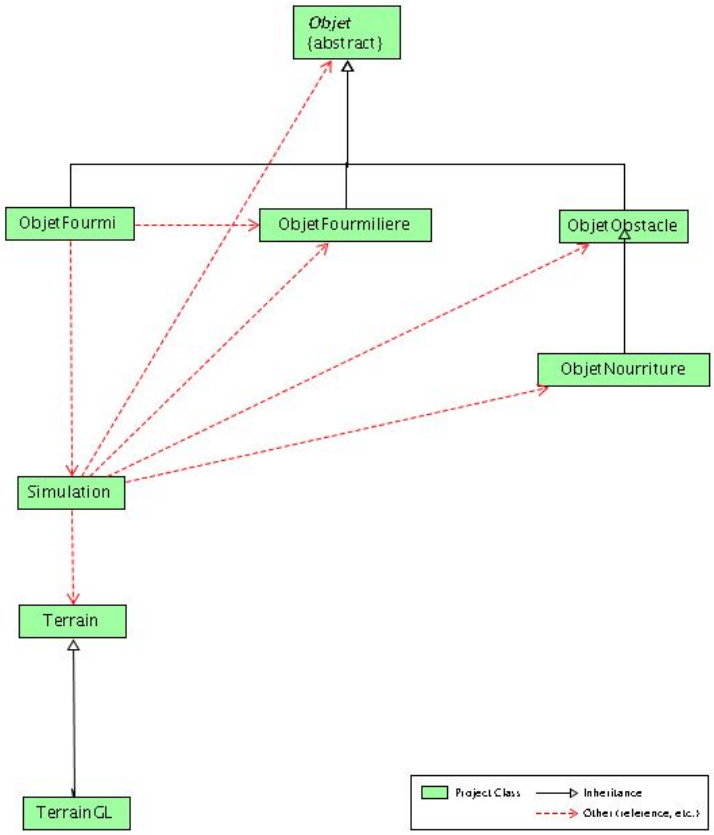
On voit que le centre du projet est la **Simulation**. C'est logique : il s'agit du titre même du sujet ! C'est dans Simulation qu'est geré l'ensemble des interactions. C'est ici que sont instanciés toutes les classes d'objet. C'est aussi ici que le programme doit boucler.

On voit que tous les **éléments** appartenants à la simulation proprement dite sont **regroupés** sous une classe mère **Objet** : celle-ci gère le placement des éléments sur le terrain, et toutes leurs propriétés en général.

Enfin, la simulation fait aussi appel à un **Terrain**, un simple carré. Notons que pour la représentation 3D (avec OpenGL), on a besoin de quelques caractéristiques supplémentaires (texture, etc.), qui sont regroupées dans TerrainGL, classe fille de Terrain.

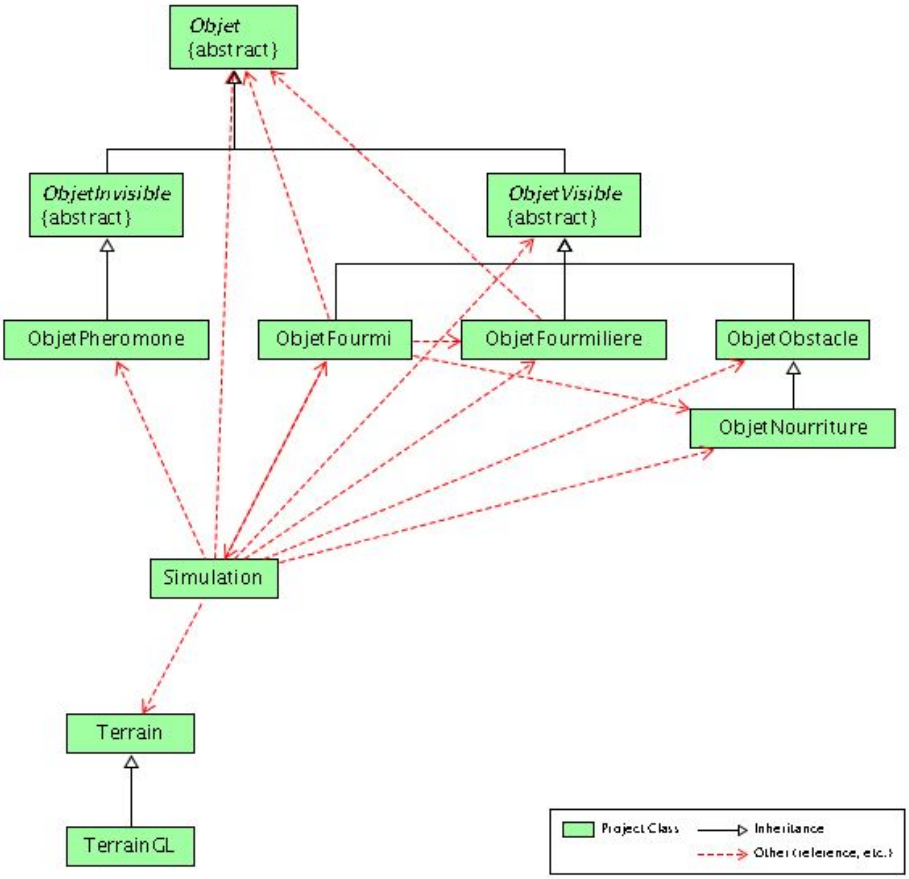
On remarque cependant que, modélisé ainsi, le programme présente un problème : il faudra gérer différemment le contournement des obstacles et celui de la nourriture (une fourmi ne cherchant pas **forcément** de la nourriture). Une manière simple de résoudre le problème est de faire hériter ObjetNouuriture de ObjetObstacle (c'est d'ailleurs celle proposée en TP) : ainsi, on ajoute simplement les attributs et méthodes spécifiques à la nourriture et on gère le reste de la même manière.

On obtient le résultat suivant :



Cette modélisation présente encore le problème de ne pas pouvoir prendre en compte les objets **invisibles** tels que les phéromones. En effet, pour obtenir un affichage cohérent, c'est la classe Objet qui doit **gérer l'affichage** de tous les objets visibles. Impossible donc d'inclure les phéromones à cette modélisation.

On crée donc de classes filles de la classe Objet : ObjetVisible et ObjetInvisible. Ainsi, l'affichage sera géré par ObjetVisible et la classe Phéromone peut hériter d'ObjetInvisible. On obtient la représentation UML suivante :



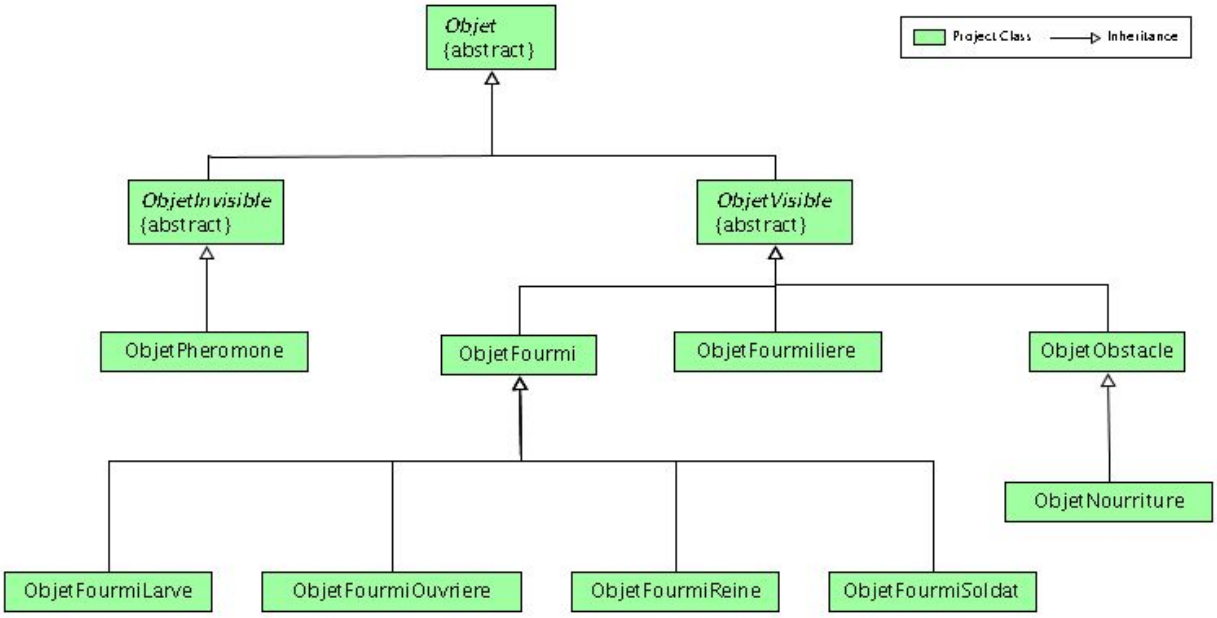
Enfin, il reste à décomposer la classe ObjetFourmi pour obtenir toutes les **spécialisations** possibles : ouvrière, soldat, reine et larve. Sans cette spécialisation, impossible de prétendre obtenir une simulation réaliste : en effet tout le mode fonctionnement des insectes dits sociaux est basé sur ce principe.

Ici, nous faisons un choix de simplification. Dans la réalité, il existe aussi des nourices chargées de prendre en charge la croissance des fourmis. Nous considérerons simplement que la reine peut mettre en incubation jusqu'à trois larves à la fois (en fonction de la nourriture disponible dans la fourmilière), et que cette incubation prends un certain temps. Les trois larves sont créées en même temps que la reine et auront deux états : en attente et en incubation. Une fois l'incubation finie, on peut introduire une nouvelle fourmi dans la simulation.

En fait, il faut savoir s'imposer des **limites** : la simulation doit paraître réaliste, mais il ne s'agit

pas de créer un programme d'intelligence artificielle. Une fourmilière reste un système complexe et on ne peut se permettre d'introduire trop de complications (princes, nourrices, différents types de nourriture, etc.)

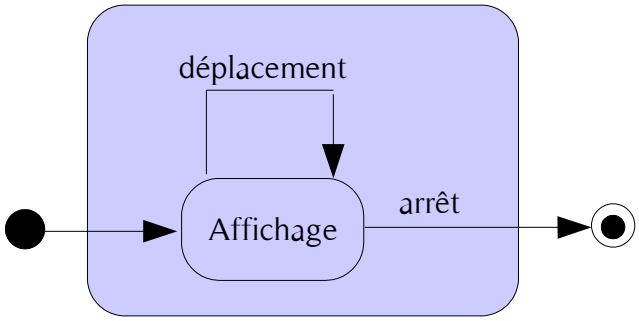
On arrive donc au schéma **final** (ou l'on fait abstraction de la Simulation, du Terrain et des dépendances pour garder une certaine lisibilité) :



On obtient ainsi une structure **efficace** et capable de supporter les éventuelles **évolutions** futures.

c) Diagramme dynamique

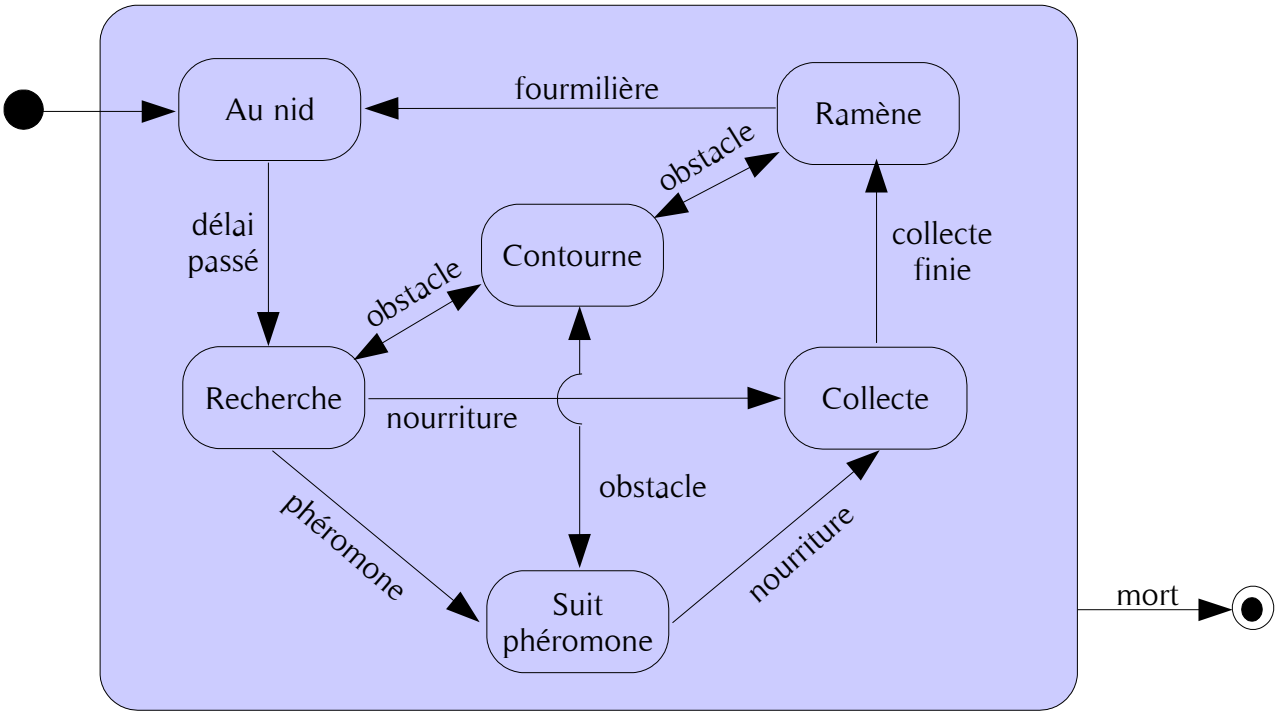
L'interaction avec l'utilisateur se fait de manière **simple**. Il suffit de lancer la simulation, et les seules actions possibles pour l'utilisateur est alors de se déplacer dans l'environnement ou de quitter. C'est logique : la simulation est autonome. Voici le diagramme état-transition correspondant :



Avant d'attaquer la conception, il est aussi nécessaire de connaitre le **comportement** des objets actifs : principalement les différents types de fourmis, mais aussi la nourriture et les

fourmilières. Cependant, la seule réelle difficulté qui se présente est pour les ouvrières. C'est les seuls éléments de la simulation dont le role est réellement complexe.

Voici comment on peut modéliser leur comportement de manière **dynamique** :



Notons que nous simplifions la notation en plaçant des doubles flèches pour le contournement des obstacles. Il faut comprendre que la fourmi commence à coutourner l'obstacle quand elle le rencontre, et poursuit son travail précédent quand il n'est plus dans les parages.

Remarquons par ailleurs que la fourmi peut mourir à n'importe moment (tuée par un soldat, et éventuellement si on améliore la simulation par vieillesse, attrapée par un oiseau, etc.).

Les autres fourmis ont des comportements beaucoup plus simples, que l'on peut expliquer **textuellement** :

- La reine ordonne à une ou plusieurs larves d'entrer en incubation selon le stock ;
- Les larves sont en attente jusqu'à cet ordre, et se transforment en fourmi une fois l'incubation terminée ;
- Les soldats chechent des fourmis d'autres fourmilières et les combattent.

On peut aussi considérer les sources de nourriture comme des objets actifs, leur comportement étant encore une fois très simple : elles se contentent de fournir de la nourriture aux fourmis et de disparaître de la simulation une fois vidées.

2) Conception

On s'est contenté jusqu'ici de présenter la partie simulation du programme, de manière analytique.

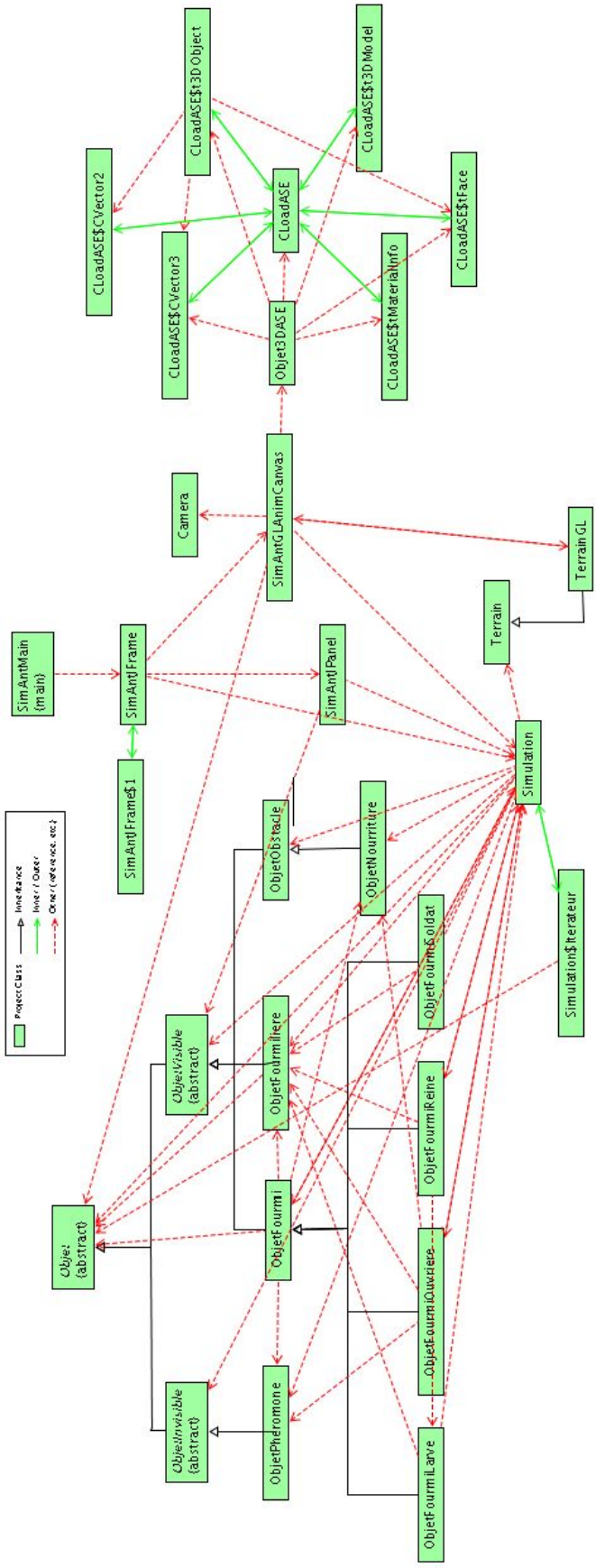
Cette partie simulation se traduit presque directement, du moins au niveau de la structure hiérarchique des classes. Il convient cependant de décrire l'implantation en Java du comportement **dynamique** des fourmis. Les classes de fourmis disposent de trois attributs : etatPrimaire, etatSecondaire et etatTernaire ; ils décrivent son état, en accord avec ceux du diagramme état-transition vu ci-dessus.

Ainsi, par exemple, lorsqu'une ouvrière collecte de la nourriture, son etatPrimaire est ETAT_COLLECTE_NOURRITURE (une constante entière), et son etatSecondaire s'incrémente progressivement de 0 à 50 pour simuler le temps de collecte.

Pour obtenir un programme complet, il faut aussi faire un **affichage**. Celui-ci était déjà présent initialement. Il est composé d'une partie 3D composée de plusieurs classes (ce sont les classes Objet3D, Cload et SimAntGL) dont la classe Camera qui gère les déplacement que l'utilisateur peut effectuer dans la fenètre d'affichage. Il comprend également un affichage 2D contenu dans la classe SimAntJPanel.

Pour lancer le processus, il faut une fonction **main**, celle-ci est contenu dans la classe SimAntMain, qui crée une fenetre (classe SimAntJFrame) dans laquelle s'effectue l'affichage 2D ou 3D selon les paramètres passés en ligne de commande.

Le résultat final est représenté **page suivante**.


CLoadASE$t3DObject
CLoadASE$t3DModel
CLoadASE$CVector2
CLoadASE
CLoadASE$tFace
CLoadASE$CVector3
Objet3DASE
CLoadASE$tMaterialInfo
Camera
SimAntGLAnimCanvas
TerrainGL
Terrain
SimAntMain (main)
SimAntFrame
SimAntPanel
Simulation
SimAntFrame$1
ObjetObstacle
ObjetNourriture
ObjetFourmiSoldat
Simulation$Iterateur
ObjetVisible (abstract)
ObjetFourmiliere
ObjetFourmiReine
Objet (abstract)
ObjetFourmi
ObjetFourmiOuvriere
ObjetInvisible (abstract)
ObjetPheromone
ObjetFourmiLarve

3) Mode d'emploi

On ajoute aux commandes le fait de pouvoir **choisir** le nombre de fourmilières, fourmis, obstacles et nourritures :

```
Usage : ./run [options]

Options : -help -version -texte -2d -3d -gl4java -nfl x -nfm x -nn x -no x
  -help    : affiche l'aide
  -version : affiche la version
  -texte   : affichage texte
  -2d      : affichage 2d
  -3d      : affichage 3d
  -gl4java : affichage 3d (utilisation de gl4java http://www.jausoft.com/)
  -nfl x   : placer x fourmilières (défaut : 2)
  -nfm x   : placer x fourmis par fourmilière (défaut : 5)
  -nn x    : placer x sources de nourriture (défaut : 20)
  -no x    : placer x obstacles (défaut : 20)

-------------------------------------------------------------

Commande claviers :
-----------------

ESC     : fin de l'application
r       : reinitialiser le monde

w       : fil de fer on/off
l       : lumiere on/off
t       : texture on/off

gauche  : déplacer la caméra à gauche
droite  : déplacer la caméra à droite
haut    : déplacer la caméra en avant
droite  : déplacer la caméra en arriere
page up : déplacer la caméra en haut
page dn : déplacer la caméra en bas

f       : mode fly


Mode fly
--------
+       : augmente la vitesse de vol
-       : diminue la vitesse de vol
gauche  : aller à gauche
droite  : aller à droite
haut    : aller en bas
bas     : aller en haut


-------------------------------------------------------------

Commandes souris :
----------------
clique gauche + déplacement souris = déplacement de la caméra
```

Conclusion

Au terme de ce rapport, nous pouvons conclure que le fait d'avoir un support initial nous impose une longue face d'**analyse** pour bien en comprendre le fonctionnement. Ayant passé deux semaines à cette phase inévitable et essentielle, le temps disponible pour la réalisation s'en est trouvé réduit d'autant.

Nous nous sommes donc limités à une simulation de fourmilières purement d'aspect purement **économique** (approvisionnement en nourriture et naissances) et n'avons pas pu aborder la partie militaire de la simulation.

Nous avons tout de même créé la classe ObjetFourmiSoldat dans l'optique d'une **amélioration** future, bien que nous ne nous en servions pas pour le moment.

Pour terminer, nous avons appris que le langage orienté objet tel que Java permet, **si l'analyse est bien menée**, d'ajouter aisément des extensions à un programme existant.

Annexes (Sources importantes)

Reproduire ici en listing l'intégralité des sources n'est pas possible, et serait surtout inutile. Nous nous contenterons donc de reproduire les sources des deux classes les plus transformées : Simulation.java et ObjetFourmiOuvriere.java.

a) Simulation.java

```
import java.util.* ;

 public class Simulation implements Runnable{

   public static final double CONV_DEG_RAD = Math.PI/180.0 ;

   public static final int MAX_FOURMILIERE = 7 ;

   public static final int OBSTACLE = 1 ;
   public static final int NOURRITURE = 2 ;
   public static final int FOURMILIERE = 3 ;
   public static final int FOURMI = 4 ;
   public static final int PHEROMONE = 5 ;

   protected float largeur,profondeur ;

   private int nbrObstacles  ;
   private int nbrNourritures ;
   private int nbrFourmilieres ;
   private int nbrFourmis;

   private Terrain terrain ;

   public Vector listeObjetsVisibles ;
   public Vector listeObjetsInvisibles ;

   private boolean finished ;
   private boolean exitRun=false ;

   private int attenteSimu=40 ;

//-------------------------------------------------------------
/**
* Constructeur par défaut
*/
    public Simulation(int l, int p, int nfl, int nfm, int nn, int no) {

      largeur = l ;
      profondeur = p ;
      nbrObstacles = no ;
      nbrNourritures = nn ;
      nbrFourmilieres = nfl ;
      nbrFourmis = nfm ;
      genererObjets() ;
   }

//-------------------------------------------------------------
    public float getLargeur() {
```

```
      return largeur ;}
    public float getProfondeur() {
      return profondeur ;}

//-------------------------------------------------------------
/**
* Reinit
*/
    public void reinit() {
   //    largeur = profondeur = 2000 ;
   //   nbrObstacles = no ;
   //   nbrNourritures = nn ;
   //   nbrFourmilieres = nfl ;
   //  nbrFourmis = nfm ;
      genererObjets() ;
   }

//-------------------------------------------------------------
/**
* Generation de l'environnement
*/
    public void genererObjets() {
      listeObjetsVisibles = new Vector() ;
      listeObjetsInvisibles = new Vector() ;

         /* Génération des obstacles */
      for (int i=0;i<nbrObstacles;i++) {
         int x = (int)(Math.random()*largeur) ;
         int y = (int)(Math.random()*profondeur) ;
         float ry=(float)(Math.random()*360.0);
         float s=(float)(0.5+Math.random()*2.0);
      // NL+LC
         if (i==0) {
            x = 1200 ; y =1000 ;
            s = (float)2.0 ;
         }
         listeObjetsVisibles.add (new ObjetObstacle(x, y, ry, s, i,0)) ;
      }

         /* Génération de la nourriture */
      for (int i=0;i<nbrNourritures;i++) {
         int x = (int)(Math.random()*largeur) ;
         int y = (int)(Math.random()*profondeur) ;
         float ry=(float)(Math.random()*360.0);
         float s=(float)(0.5+Math.random()*0.5);
         if (i==0) {
            x =1000; y = 1200 ;
            s = (float)2.0 ;
         }
         listeObjetsVisibles.add (new ObjetNourriture(x, y, ry, s, i,0)) ;
      }

         /* Génération des fourmiliÃ¨res */
      for (int i=0;i<nbrFourmilieres;i++) {
         int x = (int)(Math.random()*largeur) ;
         int y = (int)(Math.random()*profondeur) ;
         float ry=(float)(Math.random()*360.0);
         int pop = nbrFourmis;
         if (i==0) {
         // 3D
```

```
            x = 1000; y =1000 ;
            ry = 180 ;
         }
         ObjetFourmiliere of = new ObjetFourmiliere(x, y, ry, i, pop) ;
         listeObjetsVisibles.add (of) ;

         /* Génération de la reine */
         ObjetFourmiReine r = new ObjetFourmiReine(this, of, x, y, 0, i);
         listeObjetsVisibles.add (r);

         /* Génération des ouvrières */
         for (int j=1;j<=nbrFourmis;j++) {
      ObjetFourmiOuvriere f = new ObjetFourmiOuvriere(this, of, x, y, j, i) ;
            listeObjetsVisibles.add (f) ;
         }
      }
   }

//-------------------------------------------------------------
/**
* Detection d'obstacle
*/
    public Objet isObstacle(double x, double y) {
      int i = 0 ;
      int imax = getNbrObjetsVisibles() ;
      while (i<imax) {
         Objet obj = getObjetVisible(i) ;
         if ((obj!=null) && (obj.contains(x,y)) && ((obj.getType() == 1)||
             (obj.getType() == 2))) {
            return obj ;
         }
         i++ ;
      }
      return null ;
   }

//-------------------------------------------------------------
/**
* Detection de nourriture
*/
    public Objet isNourriture(double x, double y) {
      int i = 0 ;
      int imax = getNbrObjetsVisibles() ;
      while (i<imax) {
         Objet obj = getObjetVisible(i) ;
         if ((obj!=null) && (obj.contains(x,y)) &&
            (obj.getType() == NOURRITURE)) {
            return obj ;
         }
         i++ ;
      }
      return null ;
   }

/* Suppression de nourriture */

    public synchronized void suppNourriture(ObjetNourriture obj) {
      //synchronized (obj){
      if (listeObjetsVisibles.indexOf (obj)!=-1){
         listeObjetsVisibles.remove(listeObjetsVisibles.indexOf (obj));
```

```
        }
        //}
    }

 /* Poser une phéromone */

     public void poserPheromone(ObjetFourmi of, ObjetNourriture on) {
       int x = (int)of.getX();
       int y = (int)of.getY();
       int id = of.getId();
       int idg = of.getIdGroup();
       double xn = on.getX();
       double yn = on.getY();
       ObjetPheromone op = new ObjetPheromone(x, y, id, idg, xn, yn) ;
       listeObjetsInvisibles.add (op) ;
    }


 /* Détecter une phéromone */

     public Objet isPheromone(double x, double y) {
       int i = 0 ;
       int imax = getNbrObjetsInvisibles() ;
       while (i<imax) {
          Objet obj = getObjetInvisible(i) ;
          if ((obj!=null) && (Math.abs(x-obj.getX())<2.0) &&
             (Math.abs(y-obj.getY())<2.0) && (obj.getType() == PHEROMONE)) {
             return obj ;
          }
          i++ ;
       }
       return null ;
    }

 /* Créer une fourmi */

public void creerFourmi (ObjetFourmiliere of, int x, int y, int id, int idg) {
  ObjetFourmiOuvriere ofo = new ObjetFourmiOuvriere(this, of, x, y, id, idg) ;
       listeObjetsVisibles.add (ofo) ;
    }



 //-------------------------------------------------------------
 /**
 * init de la simu
 */
     public void start() {
       Thread thread = new Thread(this) ;
       finished = false;
       exitRun = false ;
       thread.start() ;
    }

 //-------------------------------------------------------------
 /**
 * fin de la simu
 */
     public void stop() {
       finished = true ;
```

```
      while (!exitRun) {
         try{ Thread.currentThread().sleep(attenteSimu); }
             catch(Exception e) {}
      }
   }

//-----------------------------------------------------------------
/**
* run du thread
*/
    public void run() {
      exitRun = false ;
      while (!finished) {
         int i = 0 ;
         int imax = getNbrObjetsVisibles() ;
         while (i<imax) {
            Objet obj = getObjetVisible(i) ;
            if (obj!=null) {
               obj.run() ;
            }
            i++ ;
         }

      // Pause dans la simulation
         try{ Thread.currentThread().sleep(attenteSimu); }
             catch(Exception e) {}
      }
      exitRun = true ;
   }

//-----------------------------------------------------------------
/**
* renvoie un iterateur
*/
    public Iterateur getIterateur() {
      return new Iterateur() ;}

//-----------------------------------------------------------------
/**
* Classe interne Iterateur
*/
    public class Iterateur {
      private int index=-1 ;
       public Iterateur() {}
       public boolean hasNext() {
         return (index+1)<listeObjetsVisibles.size() ;}
       public Objet getNext() {
         return (Objet)listeObjetsVisibles.get(++index) ;
      }
   }

//-----------------------------------------------------------------
/**
* renvoie les objets
*/
    public int getNbrObjetsVisibles() {
      return listeObjetsVisibles.size() ;
   }

    public int getNbrObjetsInvisibles() {
```

```
      return listeObjetsInvisibles.size() ;
    }

     public ObjetVisible getObjetVisible(int i) {
      return (ObjetVisible)listeObjetsVisibles.get(i) ;
    }

     public ObjetInvisible getObjetInvisible(int i) {
      return (ObjetInvisible)listeObjetsInvisibles.get(i) ;
    }

     public ObjetNourriture getObjetNourriture(int i) {
      return (ObjetNourriture)listeObjetsVisibles.get(i) ;
    }

 }
```

b) ObjetFourmiOuvriere.java

```
import java.awt.* ;

  public class ObjetFourmiOuvriere extends ObjetFourmi {

    private double vitesse = 2.0 ;

     public ObjetFourmiOuvriere(Simulation s, ObjetFourmiliere f, int x0, int
     y0, int id0, int idgrp0){
      super (s, f, x0, y0, id0, idgrp0);
    }

     public ObjetFourmiOuvriere(Simulation s, ObjetFourmiliere f, int x0, int
     y0, int sens, int id0, int idgrp0){
      super (s, f, x0, y0, sens, id0, idgrp0);
    }

 //-------------------------------------------------------------
 /**
 * simulation de l'objet
 */

     public void run() {

       switch (etatSecondaire) {
          case ETAT_CONTOURNEMENT_DROITE :
             roty += (float)45.0 ;
             break ;
          case ETAT_CONTOURNEMENT_GAUCHE :
             roty -= (float)45.0 ;
             break ;
          case ETAT_TOUT_DROIT :
             break;
          case ETAT_RETOUR :
             roty = (180/Math.PI)*Math.atan((fourmiliere.getX()-x)/
               (fourmiliere.getY()-y));
             if (((fourmiliere.getX()-x)<0) && ((fourmiliere.getY()-y)>0))
               roty+=180;
             if (((fourmiliere.getX()-x)>0) && ((fourmiliere.getY()-y)>0))
               roty+=180;
```

```
            break;
         case ETAT_SUIT_PHEROMONE :
            roty = (180/Math.PI)*Math.atan((pheromone.getXn()-x)/
              (pheromone.getYn()-y));
            if (((pheromone.getXn()-x)<0) && ((pheromone.getYn()-y)>0))
               roty+=180;
            if (((pheromone.getXn()-x)>0) && ((pheromone.getYn()-y)>0))
               roty+=180;
            break;
         default :
            if (Math.random()<0.1) {
               roty += (float)(Math.random()*90.0-45.0);
            }
            break ;
      }

   // Pour faire coincider le modele de simulation avec le
   // modele 3D, qui utilisent le meme angle de rotation roty,
   // il faut calculer l'angle de rotation de la fourmi en
   // ajoutant +90 degrÃ©s Ã  roty (angle utilisÃ© pour la 3D)
   // et retrancher l'incrÃ©ment sur l'axe des y...
      double angley = (roty+90)*Simulation.CONV_DEG_RAD ;
      double dx = vitesse*Math.cos(angley) ;
      double dy = vitesse*Math.sin(angley) ;

      switch (etatPrimaire) {

         case ETAT_AU_NID :

            if (etatSecondaire<10){
               etatSecondaire++;
            }
            else{
               if (charge!=0)
                  fourmiliere.ajouteStock(charge);
               charge = 0;
               etatSecondaire=0;
               roty+=180;
               etatPrimaire = ETAT_RECHERCHE_NOURRITURE;
            }
            break;

         case ETAT_RECHERCHE_NOURRITURE :

            x += dx ;
            y -= dy ;

               // Si je rencontre de la nourriture
            if (((nourriture=(ObjetNourriture)simu.isNourriture(x, y)) !=
              null) && (nourriture.getContenu()>0) ) {
               etatSecondaire = 0;
               etatPrimaire = ETAT_COLLECTE_NOURRITURE;
            }


               // Si je rencontre un obstacle ou les limites du terrain
            else if ( (x<0) || (y<0) || (x>simu.getLargeur()) ||
              (y>simu.getProfondeur()) || (simu.isObstacle(x, y)!=null) ) {
               x -= dx ;
               y += dy ;
```

```
            if (Math.random()<0.5)
               etatSecondaire = ETAT_CONTOURNEMENT_DROITE ;
            else
               etatSecondaire = ETAT_CONTOURNEMENT_GAUCHE ;
         }

            // Si je rencontre une phÃ©romone
         else if ( (pheromone=(ObjetPheromone)simu.isPheromone(x, y)) !=
           null){
            if (((nourriture=(ObjetNourriture)simu.isNourriture
                  (pheromone.getXn(), pheromone.getYn())) != null) &&
                  (nourriture.getContenu()>0) ){
               etatSecondaire = ETAT_SUIT_PHEROMONE;
            }
         }

         else etatSecondaire = 0 ;

         break;

      case ETAT_COLLECTE_NOURRITURE :

         synchronized (nourriture) {
            if (etatSecondaire<50){
               if (nourriture.getContenu()>0){
                  etatSecondaire++;
                  charge++;
                  nourriture.enleveContenu();
               }
               else etatSecondaire=50;
            }
            else{
               etatPrimaire = ETAT_RAMENE_NOURRITURE;
               etatSecondaire = 0;
               etatTernaire = 0;
         }
         break;

      case ETAT_RAMENE_NOURRITURE :

         if ( (Math.abs(x-fourmiliere.getX())<vitesse) && (Math.abs(y-
               fourmiliere.getY())<vitesse) ){
            System.out.println("Fourmi "+getId()+"("+getIdGroup()+") : Je
               suis au nid");
            etatPrimaire = ETAT_AU_NID;
         }
         else{

            simu.poserPheromone (this, nourriture);
            //System.out.println("Fourmi "+getId()+"("+getIdGroup()
                +")"+" : Je pose une phéromone");

            x += dx ;
            y -= dy ;

            if (etatTernaire==0) {
               if ( (x<0) || (y<0) || (x>simu.getLargeur()) ||
                     (y>simu.getProfondeur()) || (simu.isObstacle(x, y)!
                     =null) ) {
                  x -= dx ;
```

```
                        y += dy ;
                        if (Math.random()<0.5)
                           roty+=90;
                        else
                           roty-=90;
                        etatTernaire = 10;

                        etatSecondaire = ETAT_TOUT_DROIT ;
                     }
                     else{
                        etatSecondaire = ETAT_RETOUR;
                     }
                  }
                  else {
                     etatTernaire--;
                  }

                  break;

               }
            }
         }

//-------------------------------------------------------------
/**
* Dessin 2D de l'objet
*/
    public void draw(Graphics g, double facteurZoom) {
      g.setColor(new Color (getRed(), getGreen(), getBlue()));
      g.fillRect((int)(x*facteurZoom), (int)(y*facteurZoom),
            (int)(2*(facteurZoom+1)), (int)(2*(facteurZoom+1))) ;
   }
}
```