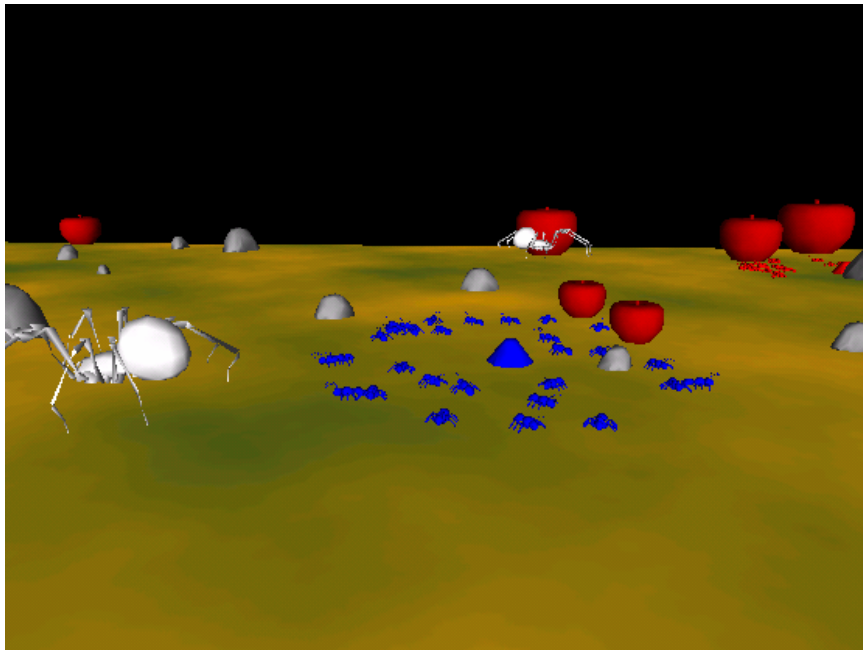


Aymeric Carrez
Sébastien Fournier
Romain Sahut



LO43 : FOURMILIERE EN JAVA

SIMANT PARK

Responsable de l'UV : J-C Creput

Printemps 2004

Sommaire

1	Introduction	- 2 -
2	Spécification des besoins.....	- 3 -
3	Analyse	- 5 -
3.1	Diagrammes de classes.....	- 6 -
3.1.1	UML de l'existant	- 6 -
3.1.2	UML du projet	- 7 -
3.2	Diagrammes dynamiques	- 8 -
3.2.1	Fourmi ouvrière.....	- 8 -
3.2.2	Fourmi Soldat	- 8 -
3.3	Diagrammes d'interactions	- 9 -
3.3.1	Interactions Fourmis – Fourmis et Fourmis - Fourmilière	- 9 -
3.3.2	Interactions Fourmi Ouvrière - Nourriture	- 10 -
4	Conception	- 11 -
4.1	Les objets de la simulation.....	- 11 -
4.2	L'interface graphique	- 12 -
4.3	Diagramme de classes JAVA	- 13 -
5	Mode d'emploi	- 14 -
6	Conclusion	- 16 -
7	Annexes	- 17 -
7.1	Graphiques d'explication.....	- 17 -
7.2	Les sources de la partie réalisée	- 17 -
7.2.1	Classe Objet	- 17 -
7.2.2	Classe ObjetNourriture	- 18 -
7.2.3	Classe ObjetCadavre.....	- 20 -
7.2.4	Classe ObjetPredateur	- 21 -
7.2.5	Classe ObjetObstacle	- 24 -
7.2.6	Classe ObjetFourmi	- 25 -
7.2.7	Classe ObjetFourmiOuvriere	- 28 -
7.2.8	Classe ObjetFourmiSoldat.....	- 35 -
7.2.9	Classe ObjetFourmilière	- 39 -
7.2.10	Classe Simulation	- 42 -
7.2.11	Classe SimAntJBoutons	- 48 -
7.2.12	Classe SimAntGLCanvas	- 54 -
7.2.13	Classe Objet3DASE	- 56 -

1 Introduction

Dans le cadre de l'UV LO43, il nous a été demandé d'améliorer une application JAVA simulant la vie d'une fourmilière et de ses occupantes. Pour ce faire, nous avons un existant (déjà développé par les étudiants les semestres précédents) ; Dans un premier temps nous présenterons les différentes étapes d'analyse, indispensable avant tout développement, le biais de diagrammes d'états, de transitions et UML.

Ensuite, nous nous pencherons sur la conception : Comment nous avons fait, pourquoi avons nous fait ces choix...

Enfin, en annexes, nous proposerons le code source du projet (sans répéter ce qui reste de l'existant afin de ne pas surcharger)

2 Spécification des besoins

Le public auquel doit s'adresser cette simulation mettant en scène des fourmis pourrait être constitué de professionnels tels que des biologistes s'intéressant au comportement des fourmis si la simulation était totalement fidèle à la réalité. Mais dans la mesure où les améliorations que nous avons apportées à ce projet sont assez éloignées d'une réalité biologique (animal errant qui mange attaque fourmis, fourmis guerrières qui s'entretuent, cadavres de fourmis qui se transforment en nourriture...), on considérera que cette animation sera destinée à une utilisation ludique et distrayante.

L'utilisateur (exempt d'arrière pensées biologiques) aura le choix entre les deux principaux modes de visualisation de l'animation à savoir le mode 2D et le mode 3D. Pour se faire, la ligne de commande qui lance le programme dispose d'un certain nombre d'options dont voici la liste exhaustive :

- -2d pour le mode affichage 2d
- -3d pour le mode affichage 3d
- -texte pour un mode affichage texte
- -gl4java pour un mode affichage 3d utilisant gl4java
- -? ou -help pour l'affichage de l'aide
- -version pour connaître la version utilisée

La tâche principale qui doit être accomplie par le programme est de montrer à l'utilisateur l'évolution d'un certain nombre de fourmilières composées d'un certain nombre de fourmis dans un environnement carré parsemé d'obstacles et de sources de nourriture, sachant que les différentes fourmilières seront en concurrence et se livreront à une guerre sans pitié pour survivre. Le nerf de cette guerre étant pour une fois la nourriture et non pas l'argent (on aurait également pu faire ça avec de l'argent, ce qui souligne la diversité de nos compétences...).

C'est l'utilisateur au moyen d'une interface graphique qui déterminera le nombre de fourmis, de fourmilières, d'obstacles, de prédateurs et de sources de nourritures qu'il désirera voir apparaître à l'écran. Nous décrivons l'utilisation précise de cette interface graphique dans la partie mode d'emploi.

Toute **fourmi** doit pouvoir manger, vivre, mourir, contourner les obstacles, et rentrer à la fourmilière.

Notre **fourmi ouvrière** doit pouvoir s'épuiser, rechercher de la nourriture, libérer, suivre ou révoquer des phéromones, ramener la nourriture à la fourmilière, et transmettre ses connaissances aux autres.

Une **fourmi soldat** doit pouvoir s'épuiser, rechercher des ennemis, les poursuivre et les combattre. Son rôle est de protéger la fourmilière de l'attaque des adversaires, elle reste donc à une certaine distance.

Une fourmi qui meurt devient un **cadavre** dont d'autres fourmis peuvent se nourrir. Ce cadavre est situé sur le lieu de mort de la fourmi.

Une **fourmilière** peut générer d'autres fourmis, soldats ou ouvrières ; Elle doit également permettre aux fourmis de poser de la nourriture, et de se nourrir. Elle gère les phéromones de sa tribu.

Une source de **nourriture** s'épuise au fur et à mesure qu'on en prend.

Une piste de phéromones relie la fourmilière à une source de nourriture. Elle n'est détectable que depuis la fourmilière, et uniquement pour la tribu de fourmis qui l'a créée. Elle existe jusqu'à ce qu'elle soit révoquée par une fourmi qui n'aurait pas trouvé de nourriture à cet endroit.

Une source de nourriture (et donc également un cadavre), peut être prise en partie, pour être consommée, ou pour en ramener. Lorsque son stock est nul, cette source disparaît.

Un **obstacle** est un objet fixe qui gêne la progression des fourmis (exemple : un rocher).

Le **terrain** utilisé par les fourmis est un plan.

Tous ses objets interagissent ensemble au sein de la **simulation**.

Cette simulation peut être visualisée par le biais de plusieurs modes : textuel, 2D ou 3D.

3 Analyse

Au premier lancement, nous avons été surpris par le travail déjà accompli, puis la surprise a laissé place au rire devant le comportement des fourmis, dont l'automate était très simplifié, et qui gérait uniquement les collisions avec les murs.

La première étape fut donc une étude de l'existant en établissant un cahier des charges contenant ce qui n'était pas géré ainsi que les idées que nous avons pour rendre la simulation plus réaliste et vivante. A ce stade là ; les connaissances de certains membres en littérature de science fiction fut un avantage certain, les réactions et le comportement des fourmis ont été inspirées par celles de Werber, ainsi que par certains sites sur l'Internet.

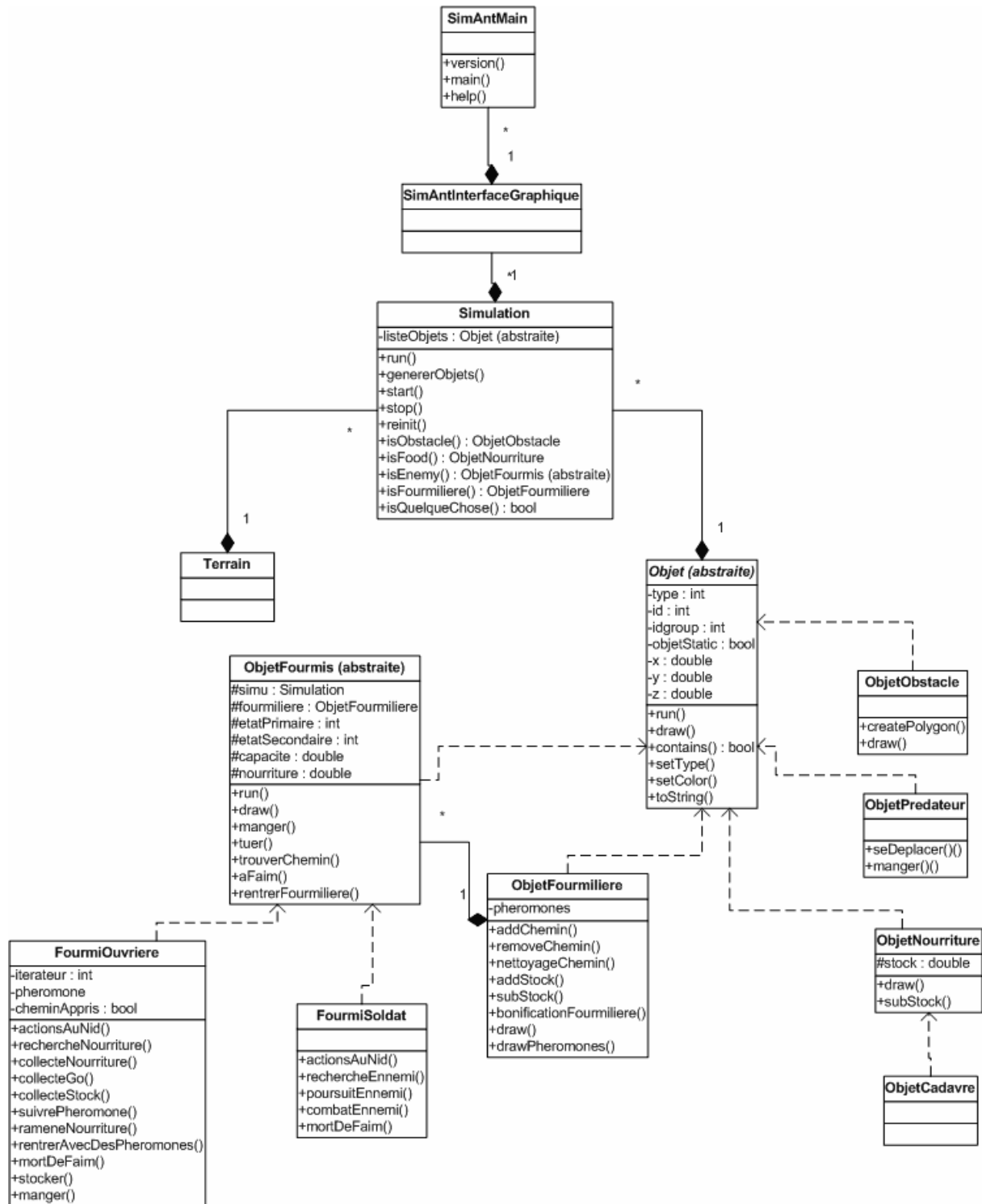
L'étude de l'existant fut rapidement effectuée car le code source était correctement commenté. Après avoir réalisé un modèle UML de l'existant afin de mieux cerner les diverses interactions entre classes, nous avons établi un cahier des charges, qui était en fait une série d'idées sans savoir si nous pourrions toutes les implémenter.

- Evolution de la nourriture
- Gestion ouvrière/guerrière
- Gestion de la mort des fourmis/fourmilières
- Gestion collision/pierres : optimisation du contournement
- Zoom/ bouger Map
- Voir nourriture transportée
- Dépouiller les cadavres
- Destruction de la fourmilière (pied ou feu ou bestiole)
- Optimisation du code
- Changement du mode d'affichage dans le menu sans régénérer
- Détection sur passage des phéromones de la tribu/de toutes les tribu
- Pièges a fourmis

Après avoir réparti les tâches à effectuer, nous avons mis en place le second modèle UML, tel que nous souhaitions l'implémenter. Celui-ci utilisait pleinement l'existant, et les modifications ne furent effectuées que pour prendre en charge de nouveaux comportements, ou utiliser de nouvelles classes.

Il restait à concevoir l'intelligence de la fourmi, et déterminer si nous allions créer deux classes (ouvrières et soldat), ou conserver l'existante en ajoutant de nouvelles fonctionnalités. La première solution a été préférée car elle permet de bien faire la distinction entre l'ouvrière et le soldat, tout en tirant partie de l'héritage des comportements tels que le contournement, qui sont identiques aux deux types de fourmis.

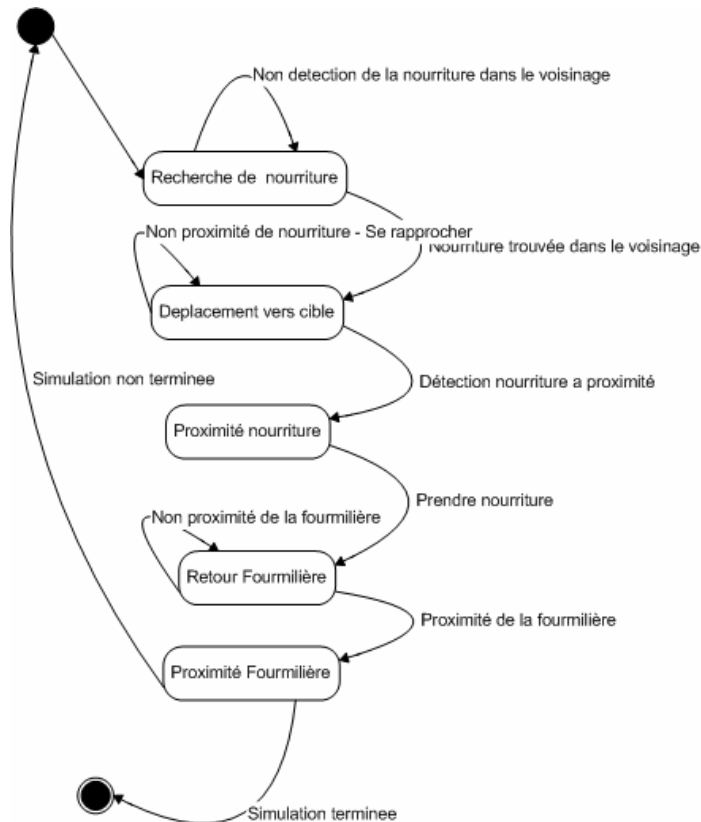
3.1.2 UML du projet



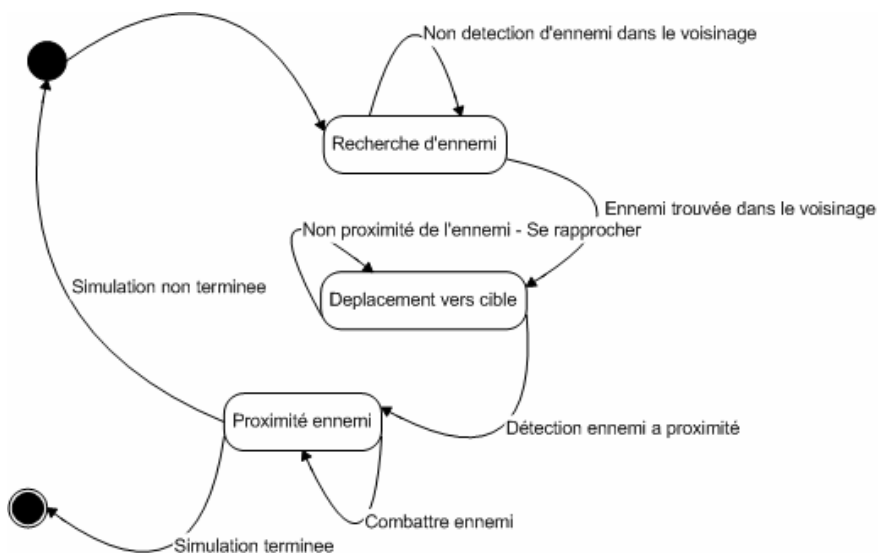
3.2 Diagrammes dynamiques

Ces diagrammes ne sont la que pour proposer une approche de l'automate simplifiée et claire. Pour plus de détails, voir en annexes le code source associé aux diagrammes.

3.2.1 Fourmi ouvrière

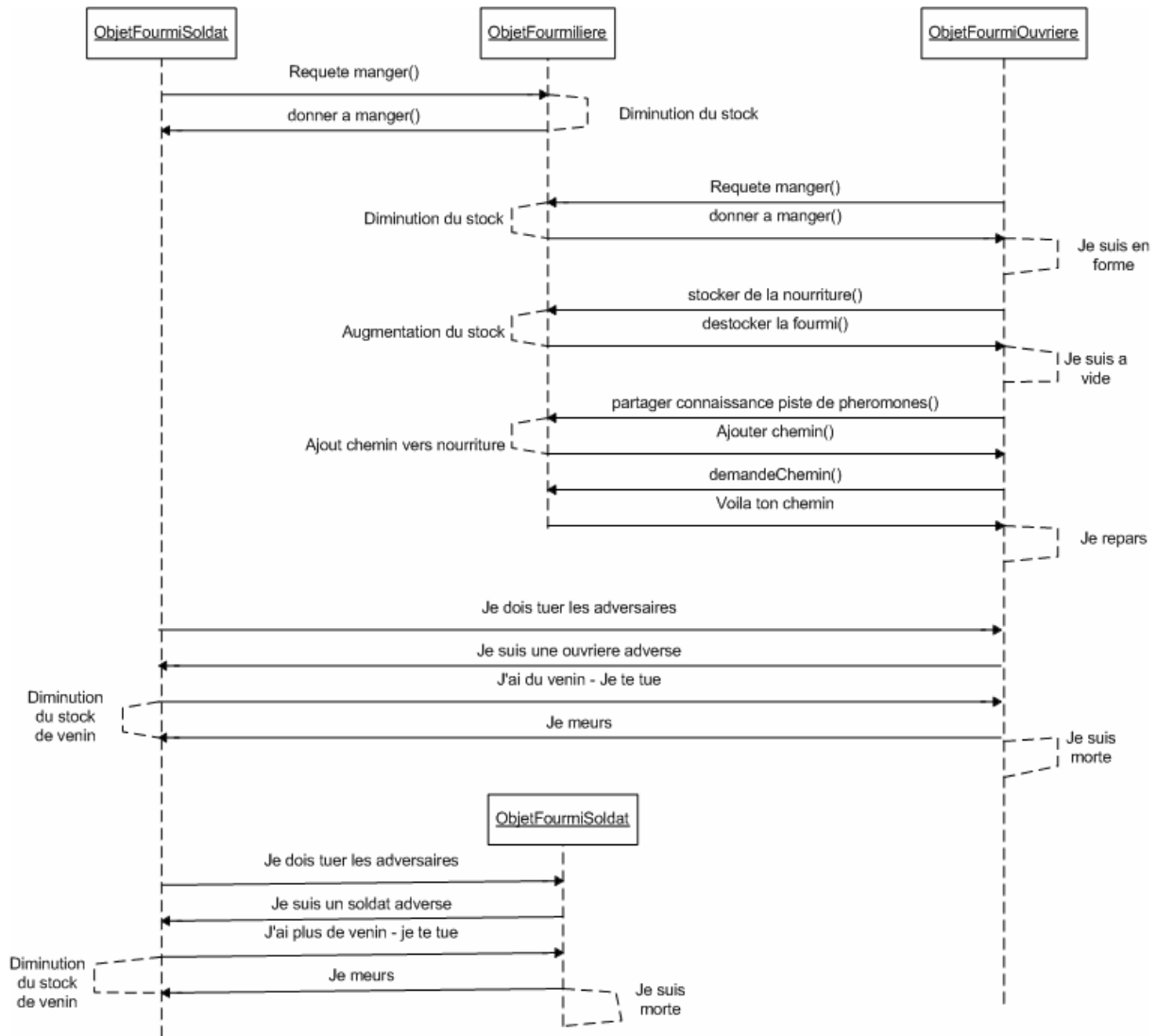


3.2.2 Fourmi Soldat

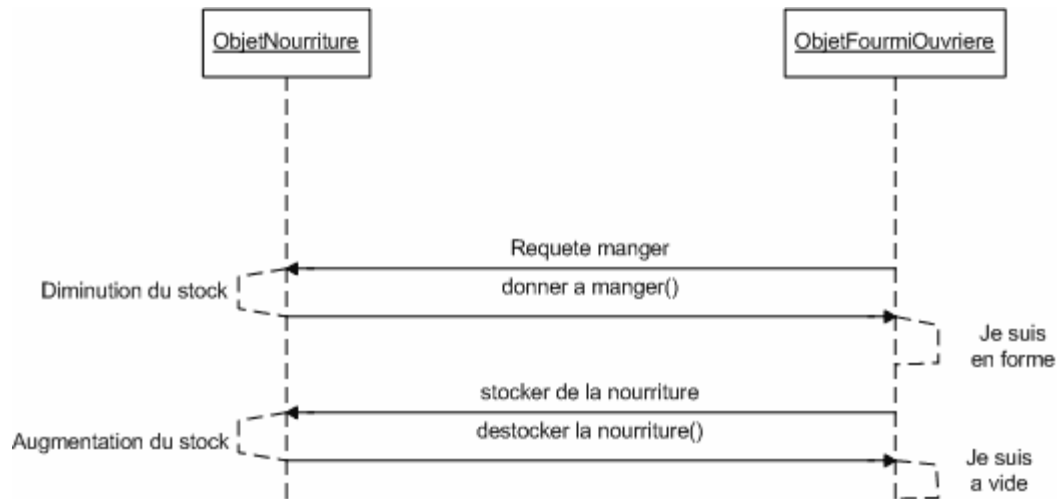


3.3 Diagrammes d'interactions

3.3.1 Interactions Fourmis – Fourmis et Fourmis - Fourmilière



3.3.2 Interactions Fourmi Ouvrière - Nourriture



4 Conception

4.1 Les objets de la simulation

Pour les sources de **nourritures**, nous avons utilisé l'existant en ajoutant la possibilité de disparaître lorsque celle-ci est consommée totalement. Elle a par conséquent un stock initial qui décroît lorsque une fourmi en fait la demande. Pour la représentation 3D, nous avons dû initialiser la variable scale afin de constater visuellement la décroissance du stock.

Pour la représentation 2D, nous avons choisi de modifier la taille de la représentation lorsque celle-ci décroît.

Les **cadavres** héritent des comportements de la nourriture. Leur stock est cependant tout juste suffisant pour nourrir une fourmi et lui donner un peu à ramener à sa fourmilière.

Leur représentation 3D est celle d'une fourmi à moitié couchée dans le sol. Pour la 2D nous avons choisi un carré blanc.

La **fourmilière** a elle aussi un stock, qui augmente lorsqu'une fourmi amène de la nourriture, et qui décroît lorsqu'on en consomme. Elle gère les pistes de phéromones sous forme de liste chaînée.

Les **phéromones** sont représentées par une liste chaînée de points. Le premier point est celui de la nourriture, les autres sont disposés à distance régulière. Une autre manière de procéder aurait été de créer des ObjetsPheromones qui auraient eu un message, une force d'émission (pour la direction à prendre) et appartenir à une colonie. Cependant nous avons jugé ceci trop lourd à gérer sur une grille de 2000 par 2000 et avec le langage Java.

La classe **ObjetFourmi** est devenue abstraite pour ne pas pouvoir instancier de fourmi sans rôle tout en regroupant les parties communes (tel que le contournement, le fait de mourir, d'avoir un stock de nourriture personnel ou le fait de rentrer à la fourmilière pour se nourrir).

Le contournement des obstacles n'a pas été modifié car le système que nous souhaitons implémenter avait des effets de bord qui les faisant rentrer dans un obstacle et elles restaient coincées. Le principe était de choisir dès le départ un orientation préférée (droitier ou gaucher), puis un sens (aller ou retour) et selon le côté de la collision, longer l'obstacle. Le problème venait des bords, et n'a pas pu être résolu.

Les **fourmis ouvrières** recherchent de la nourriture. Si elle en détecte, elle se met dessus (même position), mange (ce qui lui fait retrouver sa force) et stock ce qu'elle peut. Nous avons utilisé une constante de prise de nourriture (une pelletée de nourriture) qui fait rester la fourmi sur l'objet tant qu'elle n'a pas rempli son stock.

Si une fourmi ouvrière a faim et qu'il y a de la nourriture à la fourmilière, elle y va, cependant, si elle en trouve sur son chemin, elle passe en mode collecte de nourriture.

Une fourmi ouvrière qui a pris suffisamment de nourriture, elle rentre à la fourmilière en lâchant des phéromones, pour cela elle calcule la distance entre le point actuel et le point de la phéromone précédente et si cette distance est supérieure au seuil, elle ajoute le point courant à la piste. Si la source est épuisée, elle rentre en effaçant la piste si elle était venue avec.

Sa fatigue est proportionnelle à son action, si elle recherche, elle se fatigue normalement, par contre, si elle transporte de la nourriture, sa fatigue augmente. A chaque tour, le stock personnel de nourriture de la fourmi (qui représente sa santé) décroît de la fatigue.

Une fourmi ouvrière qui est sur le point de mourir peut, si elle a de la nourriture sur le dos, en prendre pour survivre.

Une **fourmi soldat** ne peut se nourrir qu'à sa fourmilière Elle en profite pour recharger sa capacité en venin. Elle recherche des fourmis adverses à combattre. Si elle détecte une fourmi à proximité, elle accélère et la poursuit. Dans le cadre d'un combat contre une fourmi ouvrière, elle gagne. Contre une fourmi soldat, c'est celle qui a le plus de venin qui gagne. Plus elle va vite, plus elle se fatigue.

Une attaque vaut à la survivante de voir son stock de venin décroître. Ceci permet d'éviter qu'une fourmi ne puisse tuer toute une colonie.

La représentation 3D de ces fourmis soldats est plus grosse que les ouvrières ; En 2D, elles sont entourées d'un cercle.

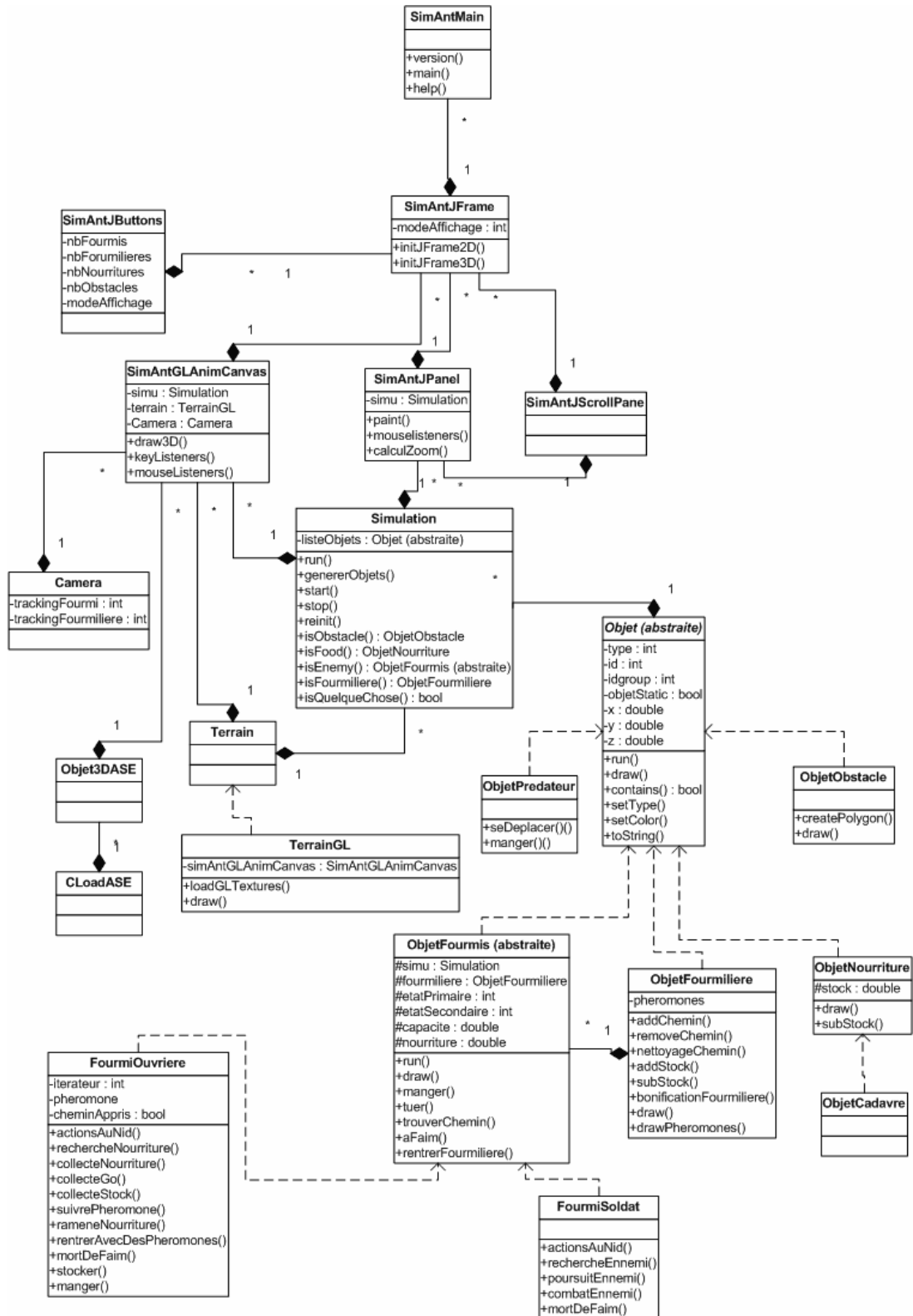
Nous avons créé un **prédateur** qui se nourrit de fourmis. Il agit comme une fourmi soldat (même façon de contourner et de mourir). Il est représenté par un « + » en 2D et par une araignée blanche (une espèce mutante de veuve noire) pour la 3D.

4.2 L'interface graphique

Nous avons décidé de séparer les boutons de l'interface graphique, ceci permettant de ne pas avoir à définir deux fois les mêmes actions et boutons. La classe SimAntJBoutons contient un JPanel dans lequel nous mettons tous les boutons et le code de leurs actions. Elle contient une référence à la simulation en cours pour lier les actions à l'interface.

Le problème est que ceci désactive le KeyListener de la classe SimAntJFrame, nous avons donc ajouté les fonctionnalités liées à celui-ci dans notre menu de boutons.

4.3 Diagramme de classes JAVA



5 Mode d'emploi

L'utilisateur aura le choix entre les deux principaux modes d'exécution que ce programme propose à savoir visualisation en 2D ou visualisation en 3D, pour ce faire il devra lancer l'exécutable avec en utilisant soit l'argument -2D, soit l'argument -3D.

Cette simulation proposera à l'utilisateur d'intervenir sur le nombre d'élément qu'il souhaite voir apparaître à l'écran. Voici la liste des paramètres qui seront modifiables :

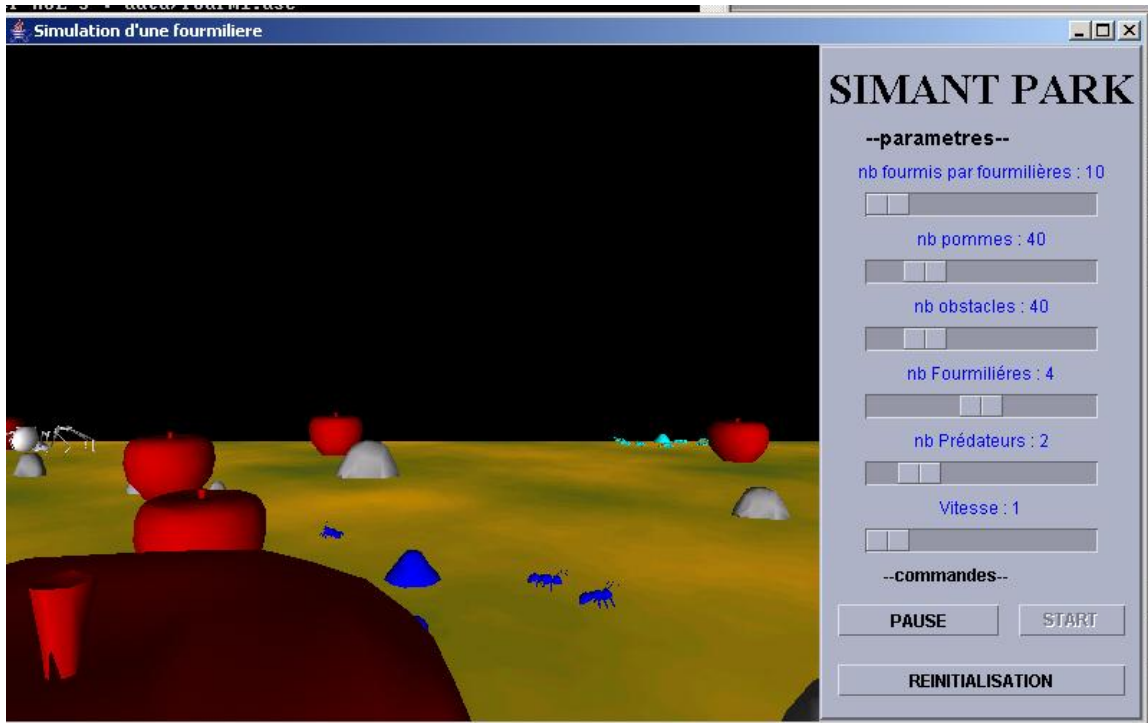
- le nombre de fourmis par fourmilières (variant de 1 à 200 par défaut 10)
- le nombre de fourmis (variant de 0 à 200 par défaut 40)
- le nombre d'obstacles (variant de 0 à 200 par défaut 40)
- le nombre de fourmilières (variant de 1 à 7 par défaut 4)
- le nombre de prédateurs (variant de 1 à 7 par défaut 2)
- la vitesse (multipliant jusqu'à 40 la vitesse d'exécution)

Notre programme disposera également de 3 boutons lui permettant d'interagir avec la simulation :

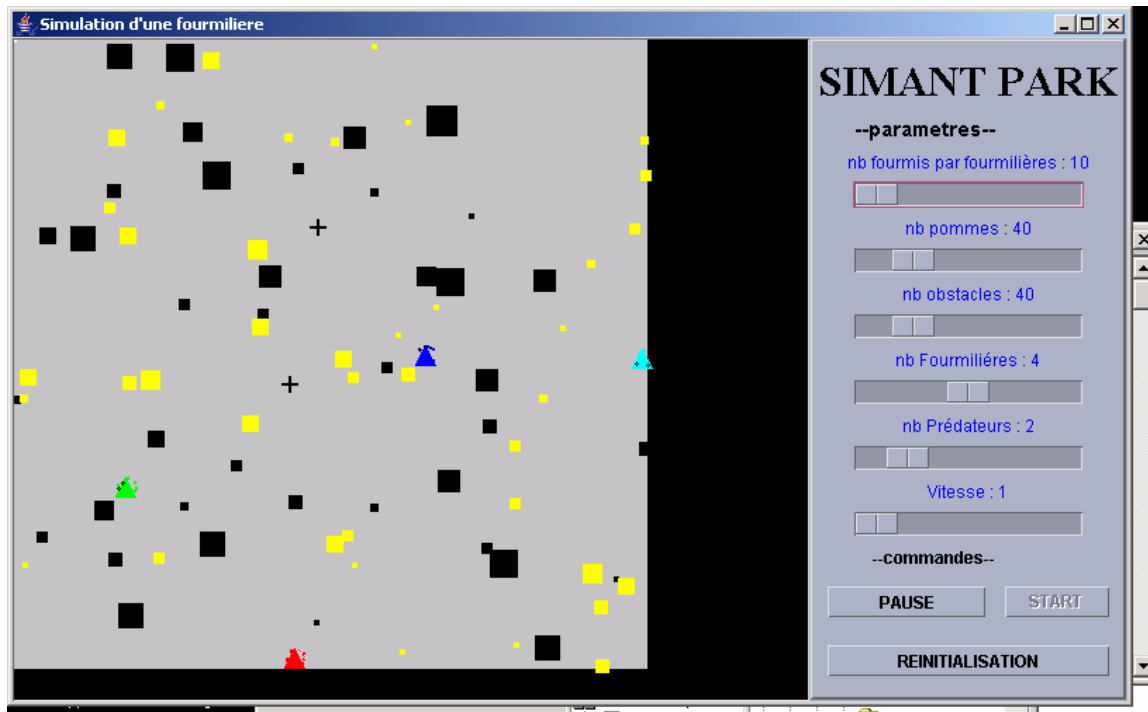
- un bouton « PAUSE » qui arrête l'animation.
- un bouton « START » qui la fait repartir ou qui la démarre.
- un bouton « REINITIALISATION » qui permet de changer la carte en fonction du nombre d'éléments choisi.

Pour ce qui est des choix graphiques à proprement parler, nous avons décidé de placer la fenêtre d'option à droite de l'animation, cela permet à cette dernière de garder une place visuellement prépondérante (puisque le paramétrage des valeurs est facultatif). La modification des paramètres est assurée par des Jsliders, pour plus de convivialité. Quant aux boutons de commandes de l'animation, ils sont situés sous les Jsliders, cela permet de faire comprendre à l'utilisateur qu'il est souhaitable qu'il modifie d'abord ces paramètres avant d'effectuer une réinitialisation (l'inverse étant impossible).

Lors du lancement du programme l'animation se lance automatiquement, ce qui permet à l'utilisateur d'avoir un aperçu immédiat de l'animation. Le fait d'appuyer sur le bouton « REINITIALISATION » créé automatiquement une nouvelle carte avec les derniers paramètres saisis par l'utilisateur, cela à également pour effet de bloquer les Jsliders qui permettent la manipulation des paramètres jusqu'à ce que l'utilisateur lance l'animation, en effet lorsqu' une carte est générée, le nombre d'éléments apparaissant est définitif.



Exemple de la simulation en 3D



Exemple de la simulation en 2D

6 Conclusion

Ce projet nous a vraiment intéressés. Il nous a permis de découvrir de nombreux aspects du langage Java. Ayant repris le sujet en cours, nous avons vu toutes les phases du développement d'un tel projet (analyse de l'existant, rédaction d'un cahier des charges, conception, réalisation).

Par manque de temps, nous n'avons pas fait toutes les optimisations, et quelques stratégies au niveau des ouvrières et des soldats peuvent encore être développées. Cependant, les modifications réalisées constituent une base de travail pour d'autres groupes d'étudiants qui souhaiteraient l'améliorer, et ainsi contribuer au projet commun.

Nous nous sommes beaucoup investis dans ce projet et nous espérons qu'il sera encore amélioré les semestres suivants.

7 Annexes

7.1 Graphiques d'explication

Ces graphiques pourraient s'apparenter à un diagramme de transition d'états, cependant ceux-ci n'en ont pas le formalisme. Ces graphiques seront donc utilisés comme support pour une meilleure compréhension des algorithmes.

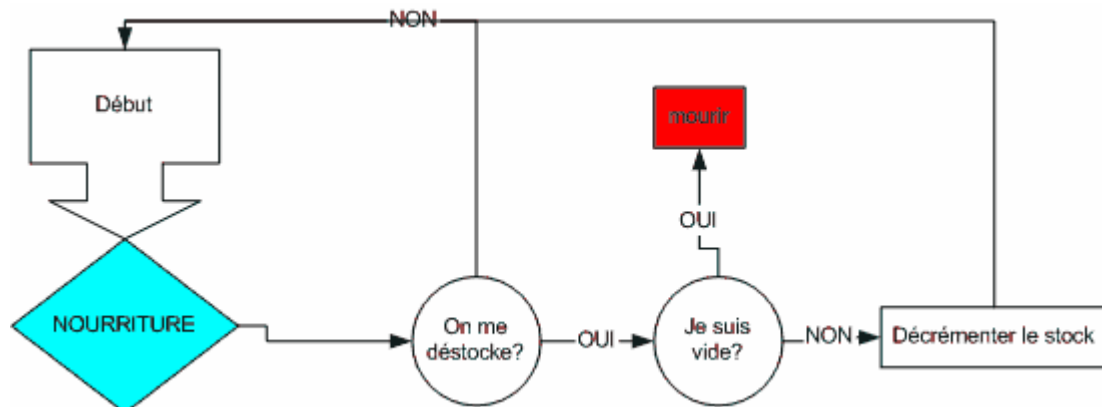
7.2 Les sources de la partie réalisée

On ne mettra que les méthodes modifiées ou ajoutées. Le code non modifié sera symbolisé par une ligne de points

7.2.1 Classe Objet

```
.....  
  
/*  
 *      Methodes de calcul des distances entre 2 objets, ou 2  
points...  
 */  
public double distance(Objet o){  
    return(distance(x,y,(double)o.getX(),(double)o.getY())); }  
public double distance(double xd,double yd){  
    return(distance(x,y,xd,yd )); }  
public double distance(double xs,double ys,double xd,double yd){  
    double dx=xd-xs;  
    double dy=yd-ys;  
    return(Math.sqrt(dy*dy+dx*dx));  
}  
  
.....
```

7.2.2 Classe ObjetNourriture



```

public class ObjetNourriture extends Objet {
    protected int xbase ;
    protected int ybase ;
    protected int largeurbase ;
    protected double stock,oldStock;

    /**
     * Constructeur
     */
    * @param x0      coord x
    * @param y0      coord y
    * @param ry      coord rotation
    * @param s       coord scale
    * @param id0     numero
    * @param idgrp0  numero du groupe
    */
    public ObjetNourriture(int x0, int y0, double ry, double s, int id0, int idgrp0)
    {
        super (x0, y0, ry, s, id0, idgrp0) ;
        type = Objet.NOURRITURE ;
        stock=10000.0+(int)(Math.random()*25000);
        oldStock=stock+1;
        scale=stock/10000.0; //Pour la 3D
        polygon = createPolygon(1.0) ;
    }

    public void run(){
        if(stock<=0){
            enVie=false;
        }
    }

    /*Une fourmi souhaite obtenir "retrait" nourriture... je lui donne si je peux, sinon je donne tout */
    protected double subStock(double retrait){
        if(stock>retrait){
            stock-=retrait;
        }else{
            retrait=stock;
            stock=0;
        }
        scale=stock/10000.0;
        return retrait;
    }
}
  
```

```

protected double getStock(){ return stock;}

protected Polygon createPolygon(double facteurzoom) {
    int xbase = (int)(x*facteurzoom);
    int ybase = (int)(y*facteurzoom);
    double larg=(stock/1000)*facteurzoom ;
    .....

    return poly ;
}

//-----
/**
 * Dessin 2D de l'objet
 */
public void draw(Graphics g, double facteurZoom) {
    if ((polygonZoom==null) || (oldfacteurZoom!=facteurZoom) || (oldStock>stock) ) {
        oldStock=stock;
        oldfacteurZoom = facteurZoom ;
        polygonZoom = createPolygon(facteurZoom) ;
    }
    g.setColor(Color.yellow);
    g.fillPolygon(polygonZoom) ;
}
}

```

7.2.3 Classe ObjetCadavre

```
public class ObjetCadavre extends ObjetNourriture {
    int xbase ;
    int ybase ;
    int largeurbase ;
    double stock,oldStock;

    /**
     * Constructeur
     *
     * @param x0      coord x
     * @param y0      coord y
     * @param ry      coord rotation
     * @param s        coord scale
     * @param id0     numero
     * @param idgrp0  numero du groupe
     */
    public ObjetCadavre(int x0, int y0, double ry, double s, int id0, int idgrp0)
    {
        super (x0, y0, ry, s, id0, idgrp0) ;
        stock=2000;
        oldStock=stock+1;
        polygon = createPolygon(1.0) ;
        scale=stock/10000.0;
    }

    public void run(){
        if(stock<=0){
            enVie=false;
        }
    }

    //-----
    /**
     * Dessin 2D de l'objet
     */
    public void draw(Graphics g, double facteurZoom) {
        if ((polygonZoom==null) || (oldfacteurZoom!=facteurZoom) || (oldStock>stock) ) {
            oldStock=stock;
            oldfacteurZoom = facteurZoom ;
            polygonZoom = createPolygon(facteurZoom) ;
        }
        g.setColor(Color.white);
        g.fillPolygon(polygonZoom) ;
    }
}
```

7.2.4 Classe ObjetPredateur

```
public class ObjetPredateur extends Objet{

    public static final int ETAT_POURSUIT=104;
    public static final int ETAT_RECHERCHE = 101 ;
    public static final int FORCEMAX = 15000;

    public static final double VITESSE1=1.0 ;
    public static final double VITESSE2=1.5*VITESSE1 ;
    public static final double VITESSE3=2.0*VITESSE1 ;

    public static final double FATIGUE_RECH = VITESSE1/2 ;
    public static final double FATIGUE_POURSUIT=FATIGUE_RECH*1.5 ;

    // Etats Secondaires
    public static final int ETAT_CONTOURNEMENT_DROITE = 200 ;
    public static final int ETAT_CONTOURNEMENT_GAUCHE = 201 ;

    private Simulation simu ;
    private int etat, etatSecondaire;
    private double vitesse = VITESSE1 ;

    private int force;

    public ObjetPredateur(Simulation s, int x0, int y0){
        super (x0,y0);
        simu = s;
        setType(PREDATEUR);
        roty=(float)(Math.random()*360.0);
        scale = 1;
        etat = ETAT_RECHERCHE;
        etatSecondaire = 0 ;
        setColor((float)1.0,(float)1.0,(float)1.0);
        id=PREDATEUR;
        idGroup=PREDATEUR;
        force = FORCEMAX;
    }

    protected void actionPredateur(){
        switch(etat){
            case ETAT_RECHERCHE:    rechercheFourmi();    break;
            case ETAT_POURSUIT:    poursuitFourmi();    break;
        }
    }

    private void rechercheFourmi(){
        //chercher me fatigue
        force=FATIGUE_RECH;
        //Si un ennemi est a portee
        if(simu.isEnemy(x,y,-1)!=null){
            etat=ETAT_POURSUIT;
            //On le poursuit en accelerant
            vitesse=VITESSE2;
        }
    }
}
```

```

private void poursuitFourmi(){
    //Poursuivre me fatigue
    force=FATIGUE_POURSUIT;
    ObjetFourmi fourmi;
    //Si l'ennemi est a portee
    if(simu.isEnemy(x,y,idGroup)!=null){
        fourmi=(ObjetFourmi)simu.isEnemy(x,y,idGroup);
        //On le poursuit!
        roty=trouverChemin(fourmi.getX(),fourmi.getY());
        //Si on est tres proche, on accelere et declenche l'attaque
        if(distance(fourmi)<5){
            vitesse=VITESSE3;
            fourmi.tuer();
            force+=300;
            if(force>FORCEMAX) force=FORCEMAX;
        }
    }else{
        etat=ETAT_RECHERCHE;
    }
}
}

```

```

protected double trouverChemin(double x2, double y2)
{
    double dx = x2-x;
    double dy = y2-y;
    double res = 0;
    if (dx==0) res = y<y2?Math.PI*1.5:Math.PI*0.5;
    else
    {
        res = Math.atan(dy/dx);
        if (dx>0)
        {
            if (dy<=0) res = -res;
            else res = Math.PI*2 - res;
        }
        else res = Math.PI - res;
    }
    return 270+simu.CONV_RAD_DEG*res;
}

```

```

public void run() {
    actionPredateur();
    switch (etatSecondaire) {
    case ETAT_CONTOURNEMENT_DROITE :
        roty += (float)45.0 ;
        // roty+/-90+(float)Math.random()*180.0 ;
        break ;
    case ETAT_CONTOURNEMENT_GAUCHE :
        roty -= (float)45.0 ;
        // roty+/-90+(float)Math.random()*180.0 ;
        break ;
    default :
        if (Math.random()<0.1 && etat==ETAT_RECHERCHE) {
            roty += (float)(Math.random()*90.0-45.0);
        }
        break ;
    }
}

double angley = (roty+90)*Simulation.CONV_DEG_RAD ;

```

```

double dx = vitesse*Math.cos(angley) ;
double dy = vitesse*Math.sin(angley) ;

x += dx ;
y -= dy ;

//Detection d'obstacle ou de bord d'ecran
if ( (x<0) || (y<0) || (x>simu.getLargeur()) || (y>simu.getProfondeur()) || (simu.isQuelqueChose(x, y))) {
    x -= dx ;
    y += dy ;
    if (Math.random()<0.5)
        etatSecondaire = ETAT_CONTOURNEMENT_DROITE ;
    else
        etatSecondaire = ETAT_CONTOURNEMENT_GAUCHE ;
}
else etatSecondaire = 0 ;

mortDeFaim();
}

protected void mortDeFaim(){

    if(force<=0){
        enVie=false;
    }
}

public void draw(Graphics g, double facteurZoom) {

    int xbase = (int)(x*facteurZoom) ;
    int ybase = (int)(y*facteurZoom) ;
    int largeurbase = (int)(facteurZoom+2) ;
    g.setColor(new Color (0, 0, 0));
    g.fillRect(xbase-largeurbase/2-5,ybase-largeurbase/2,largeurbase+10,largeurbase);
    g.fillRect(xbase-largeurbase/2,ybase-largeurbase/2-5,largeurbase,largeurbase+10);
}
}

```

7.2.5 Classe ObjetObstacle

```
public class ObjetObstacle extends Objet {
    public final static int TAILLE = 50 ;

    /**
     * Constructeur
     *
     * @param x0      coord x
     * @param y0      coord y
     * @param ry      coord rotation
     * @param s        coord scale
     * @param id0     numero
     * @param idgrp0  numero du groupe
     */
    public ObjetObstacle(int x0, int y0, double ry, double s, int id0, int idgrp0)
    {
        super (x0, y0, ry, s, id0, idgrp0) ;
        type = Objet.OBSTACLE ;
        //le polygon de test de collision (celui la n'est pas affich )
        polygon = createPolygon(1.0) ;
    }

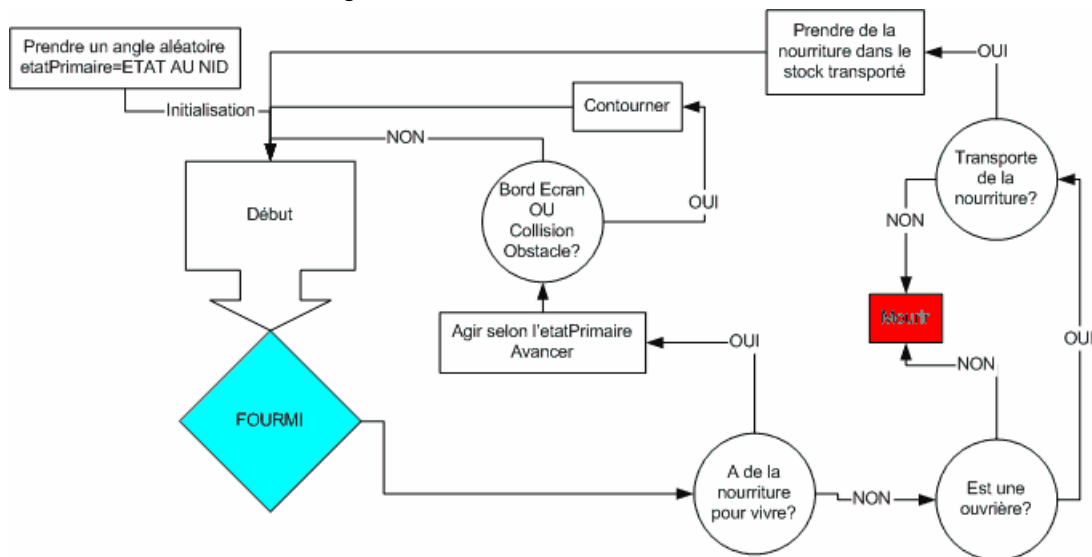
    //-----
    public Polygon createPolygon(double facteurzoom) {
        int xbase = (int)(x*facteurzoom) ;
        int ybase = (int)(y*facteurzoom) ;
        //CORRECTION D'UN BUG (ou oubli)
        //Redimensionnement pour qu'il coincide avec l'objet 3D (dont la propriete scale
        // est definie aleatoirement entre 0.5 et 2.5)
        int largeurbase = (int)(TAILLE*facteurzoom*scale/2.5) ;
        if (largeurbase<1) largeurbase = 1 ;

        Polygon poly = new Polygon() ;
        poly.addPoint(xbase-largeurbase, ybase-largeurbase) ;
        poly.addPoint(xbase+largeurbase, ybase-largeurbase) ;
        poly.addPoint(xbase+largeurbase, ybase+largeurbase) ;
        poly.addPoint(xbase-largeurbase, ybase+largeurbase) ;
        return poly ;
    }

    //-----
    /**
     * Dessin 2D de l'objet
     */
    public void draw(Graphics g, double facteurZoom) {
        if ((polygonZoom==null) || (oldfacteurZoom!=facteurZoom)) {
            oldfacteurZoom = facteurZoom ;

            polygonZoom = createPolygon(facteurZoom) ;
        }
        g.setColor(Color.black);
        g.fillPolygon(polygonZoom) ;
    }
}
```

7.2.6 Classe ObjetFourmi



```

abstract public class ObjetFourmi extends Objet {
    // Etats primaires
    public static final int ETAT_AU_NID = 100 ; //Commun aux soldats et aux ouvrieres
    public static final int ETAT_RECHERCHE = 101 ;

    public static final int NOURRIMAX = 1000 ;

    public static final int ROLE_SOLDAT = 1 ;
    public static final int ROLE_OUVRIERE = 0 ;

    public static final int CAPAMAX_OUVRIERE = 2500 ;

    public static final int ACTION_OUVRIERE=60 ;

    public static final double VITESSE0=0.0 ;
    public static final double VITESSE1=1.0 ;
    public static final double VITESSE2=1.5*VITESSE1 ;
    public static final double VITESSE3=2.0*VITESSE1 ;

    public static final double FATIGUE_RECH = VITESSE1/2 ;

    // Etats Secondaires
    public static final int ETAT_CONTOURNEMENT_DROITE = 200 ;
    public static final int ETAT_CONTOURNEMENT_GAUCHE = 201 ;

    protected int etatPrimaire, etatSecondaire ;
    protected Simulation simu ;
    protected ObjetFourmilere fourmilere ;
    protected int capacite;
    protected double nourriture;

    protected int role; //Soldat: ROLE_SOLDAT et Ouvriere: ROLE_OUVRIERE

    protected double vitesse = VITESSE1 ;

    /**
     * Constructeur
     *

```

```

* @param f          fourmiere de la fourmi
* @param x0         coord x de la fourmi
* @param y0         coord y de la fourmi
* @param id0        numero de la fourmi
* @param idgrp0     numero de la fourmiere (défini la couleur de la fourmi)
*/
public ObjetFourmi(Simulation s, ObjetFourmiere f, int x0, int y0, int id0, int idgrp0){
    super (x0, y0, id0, idgrp0);
    simu = s;
    type = FOURMI;
    fourmiere = f;
    setColor(Objet.RED[idgrp0], Objet.GREEN[idgrp0], Objet.BLUE[idgrp0]);
    roty=(float)(Math.random()*360.0);
    scale = 0.3;
    etatPrimaire = ETAT_AU_NID;
    etatSecondaire = 0;

    nourriture=NOURRIMAX;
}

public int getCapacite(){return capacite;}

abstract void actionFourmi();

//-----
/**
 * simulation de l'objet
 */
public void run() {
    //Fonction redefinie specifiquement au type de fourmi
    actionFourmi();

    //Traitement etat secondaire
    .....

    if(distance(fourmiere)<5 && etatPrimaire!=ETAT_AU_NID && etatPrimaire!=ETAT_RECHERCHE &&
etatPrimaire!=ObjetFourmiOuvriere.ETAT_COLLECTE_NOURRITURE){
        etatPrimaire=ETAT_AU_NID;
    }
    mortDeFaim();
}

protected void tuer(){
    simu.addObject(new ObjetCadavre((int)x,(int)y,roty,scale,-1,-1));
    enVie=false;
}

protected double trouverChemin(double x2, double y2)
{
    double dx = x2-x;
    double dy = y2-y;

```

```

    double res = 0;
    if (dx==0) res = y<y2?Math.PI*1.5:Math.PI*0.5;
    else
    {
        res = Math.atan(dy/dx);
        if (dx>0)
        {
            if (dy<=0) res = -res;
            else res = Math.PI*2 - res;
        }
        else res = Math.PI - res;
    }
    return 270+simu.CONV_RAD_DEG*res;
}

protected void manger(ObjetFourmilere f2){
    if(nourriture<NOURRIMAX)
        nourriture+=f2.subStock(NOURRIMAX-nourriture);
}

protected void aFaim(){
    if(nourriture<NOURRIMAX/4 && fourmilere.getStock(>0){ //Si manque de nourriture
        etatPrimaire=ETAT_AU_NID;
    }
}

abstract void mortDeFaim();

protected void rentrerFourmilere(){
    if(distance(fourmilere)<5){
        x=fourmilere.getX();
        y=fourmilere.getY();
    }
    roty=trouverChemin(fourmilere.getX(),fourmilere.getY());
}

protected int getRole(){
    return role;
}

//-----
/**
 * Dessin 2D de l'objet
 */
abstract public void draw(Graphics g, double facteurZoom);
}

```

7.2.7 Classe ObjetFourmiOuvriere

```
public class ObjetFourmiOuvriere extends ObjetFourmi {

    public static final int ETAT_COLLECTE_NOURRITURE = 102 ;
    public static final int ETAT_RAMENE_NOURRITURE = 103 ;
    public static final int ETAT_RAMENE_ET_EFFACE=106;

    //Les valeurs qui seront decrementees a la nourriture de la bestiole
    public static final double FATIGUE_RAMENE = FATIGUE_RECH*2.0 ;
    public static final double DISTANCE_ENTRE_2_PHEROMONES = 10.0 ;

    private LinkedList pheromone;
    private int iterateur;
    private boolean cheminAppris;

    /**
     * Constructeur
     * @param s          La simulation en cours
     * @param f          fourmiere de la fourmi
     * @param x0         coord x de la fourmi
     * @param y0         coord y de la fourmi
     * @param id0        numero de la fourmi
     * @param idgrp0     numero de la fourmiere (definit la couleur de la fourmi)
     */
    public ObjetFourmiOuvriere(Simulation s, ObjetFourmiere f, int x0, int y0, int id0, int idgrp0){
        super (s,f,x0, y0, id0, idgrp0) ;
        capacite=CAPAMAX_OUVRIERE;

        pheromone=new LinkedList();
        iterateur=0;
        cheminAppris=false;
        role=ROLE_OUVRIERE;
    }

    //-----
    /**
     * simulation de l'objet
     * La fourmi se balade a la recherche de nourriture.
     * Si elle en trouve, elle en prend (nourriture=NOURRIMAX), ce qui reduit sa capacite de transport
     * et aussi le stock de la source de nourriture et rentre a la maison en laissant des Pheromones.
     * Regularierement elle rentre a la maison, et ramene sa nourriture et elle partage son savoir.
     * Concernant la piste de pheromones, si elle n'a rien trouve, elle participe a l'exploitation
     * des sources de nourritures.
     * sinon elle repart en quete de nourriture.
     */

    protected void actionFourmi() {
        //Traitement etat primaire
        //if(fourmiere.getStock(<1.0)System.out.println(etatPrimaire+" "+iterateur+" "+etatSecondaire);
        //if(capacite<CAPAMAX_OUVRIERE&&simu.isFood(x,y)==null)System.out.println(etatPrimaire+" "+iterateur+"
        "+etatSecondaire);
        switch(etatPrimaire){
            case ETAT_AU_NID:
                if(distance(fourmiere)<DISTANCE_ENTRE_2_PHEROMONES/2) //J'y suis
                    actionsAuNid();
                else //J'y vais

```

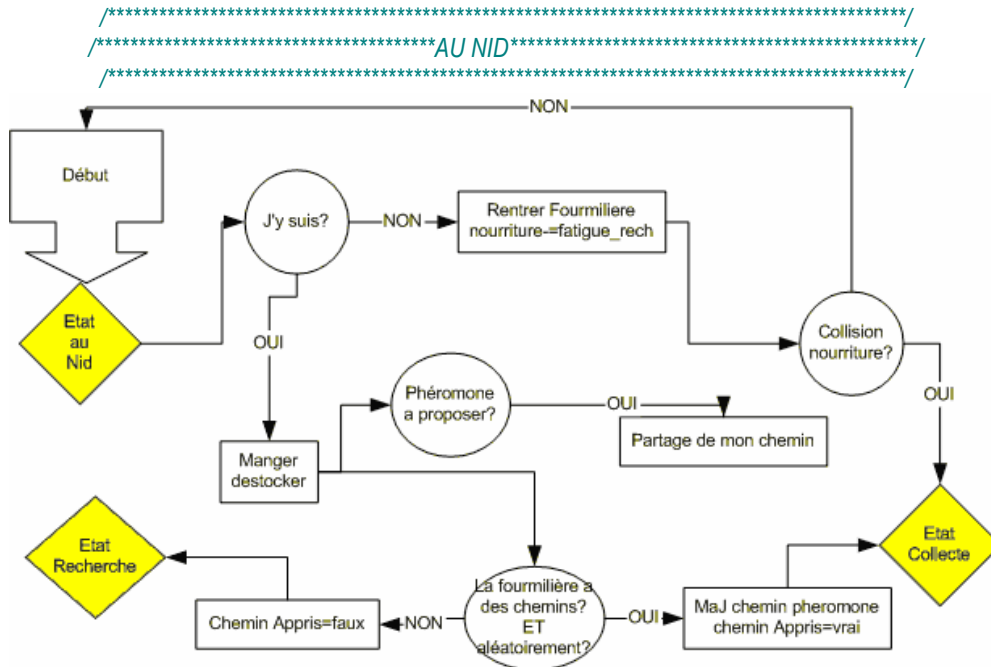
```

    rentrerFourmilier();
    if(simu.isFood(x,y)!=null)
        etatPrimaire=ETAT_COLLECTE_NOURRITURE;
        nourriture-=FATIGUE_RECH;

    break;
    case ETAT_RECHERCHE:    rechercheNourriture();
    case ETAT_COLLECTE_NOURRITURE:    collecteNourriture();
    case ETAT_RAMENE_NOURRITURE:    rameneNourriture();
    case ETAT_RAMENE_ET_EFFACE:    rentrerAvecDesPheromones(false);
}
}

```

break;
break;
break;
break;



```

private void actionsAuNid(){
    if(!cheminAppris && !pheromone.isEmpty()){
        fourmilier.addChemin(pheromone);
    }
    pheromone=new LinkedList();
    iterateur=0;
    fourmilier.addStock(CAPAMAX_OUVRIERE-capacite);
    capacite=CAPAMAX_OUVRIERE;
    manger(fourmilier); //On recharge les batteries
    vitesse=VITESSE1; //On se tient au co
    LinkedList pheromones=(LinkedList) fourmilier.getChemins();
    etatPrimaire=ETAT_RECHERCHE;
    switch(pheromones.size()){ //Soit je pars a la recherche de chemins, soit j'exploite l'existant
        case 0: //Pas encore de chemin
            etatPrimaire=ETAT_RECHERCHE;
            roty=(float)(Math.random()*360.0);
            cheminAppris=false;
            break;
        default:
            if(Math.random()<0.2){
                //Pars chercher d'autres sources de nourriture
                etatPrimaire=ETAT_RECHERCHE;
                roty=(float)(Math.random()*360.0);
                cheminAppris=false;
            }else{

```

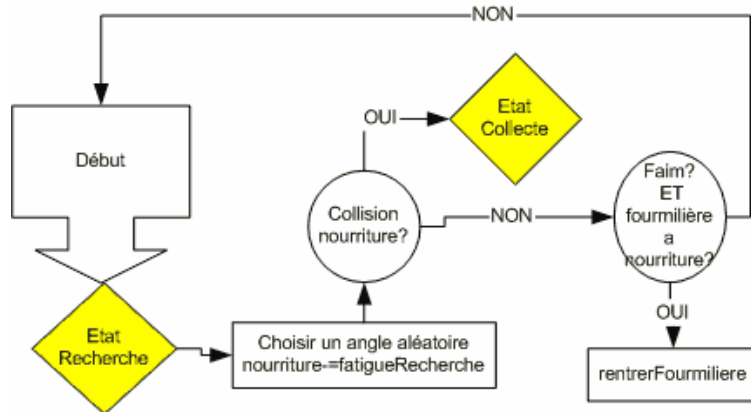
```

//Suivre chemin pheromones
etatPrimaire=ETAT_COLLECTE_NOURRITURE;
int numChemin=(int) (Math.random()*pheromones.size());
cheminAppris=true;
pheromone=(LinkedList) pheromones.get(numChemin);
iterateur=pheromone.size()-1;

break;
}
}
}

/*****RECHERCHE*****/

```



```

/*Recherche la nourriture en se deplacant aleatoirement.
* Si elle trouve, elle passe a l'etat ETAT_COLLECTE_NOURRITURE
*/

```

```

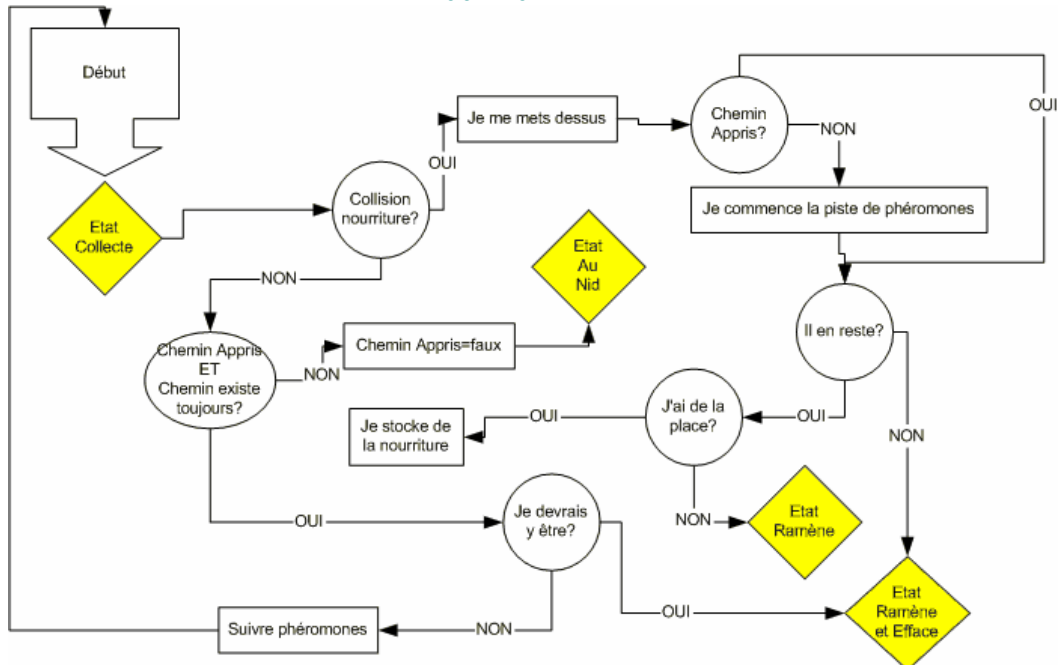
private void rechercheNourriture(){
nourriture=FATIGUE_RECH;
if(simu.isFood(x,y)!=null) //S'il y en a une qq part
etatPrimaire=ETAT_COLLECTE_NOURRITURE;
else
aFaim(); //Si manque de nourriture, je rentre
}

```

```

/*****COLLECTE*****/

```



```

/*Collecte la nourriture.
* Si elle y est deja, elle met en stock

```

```

*      Sinon elle s'y rend, par les pheromones
*/
private void collecteNourriture(){
    ObjetNourriture o=(ObjetNourriture)simu.isFood(x,y);

    if(o==null || iterateur !=0){ //On est sur la piste de pheromones en direction de la nourriture
        collecteGo();
    }else{ //On y est
        collecteStock(o);
    }
}

/*
*Se rendre a une source de nourriture connaissant le chemin qui y mene
*      ATTENTION: une liste de pheromone peut etre en cours d'etre revoquee...
*
*      C'est pour ca qu'on fera bien attention a ne pas la laisser partir
*      n'importe ou.
*/
private void collecteGo(){
    if( cheminAppris && ! pheromone.isEmpty()){
        Point2D.Double p=(Point2D.Double) pheromone.get(0);
        double xd=p.getX(); //On va sur le point precedent de la liste
        double yd=p.getY(); //chainee (le suivant vers la destination)
        if(iterateur==0 && simu.isFood(xd,yd)==null){ //Il n'y a plus rien et je le vois
            etatPrimaire=ETAT_RAMENE_ET_EFFACE;
        }else if(iterateur<pheromone.size()){ //Vers la nourriture
            suivrePheromone(-1);
        }
    }else{

        cheminAppris=false; //On a vide ma liste! je retourne a la recherche
        etatPrimaire=ETAT_AU_NID;
    }
    nourriture-=FATIGUE_RECH;
}

/*
*On est cense etre sur une source de nourriture (elle est en parametre)
*      On se cale au centre de la nourriture (point d'origine des pheromones)
*      On remplit le stock jusqu'a plein, ou nourriture vide.
*      Quand j'en ai fini, selon le cas, je rentre en lachant des pheromones,
*      ou alors je rentre grace aux pheromones, ou alors je les efface si elles
*      ne menent plus a rien.
*/
private void collecteStock(ObjetNourriture n2){
    x=n2.getX();
    y=n2.getY();

    if(pheromone.isEmpty() && !cheminAppris){
        pheromone.add(new Point2D.Double(x,y));
    }
    if(capacite>0 && n2.getStock(>0){
        //On s'arrete et on accumule de la nourriture jusqu'a remplissage ou stock vide
        vitesse=VITESSE0;
        stocker(n2);
        manger(n2);
    }else{
        etatPrimaire=ETAT_RAMENE_NOURRITURE;
    }
}

```

```

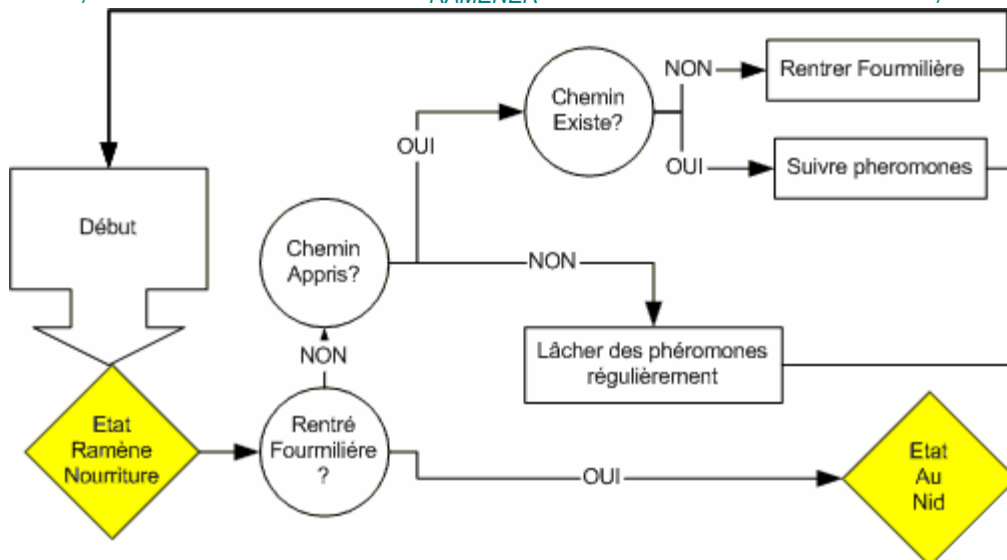
    if(n2.getStock()<=0){ //Il n'y a plus rien
        etatPrimaire=ETAT_RAMENE_ET_EFFACE;
    }
    vitesse=VITESSE1;
    rentrerFourmiliere();
    iterateur=pheromone.size()-1;
}

}

/* Suivre la piste de pheromones
 * Indice 0 represente la nourriture
 * indice size()-1 represente la pheromone la plus proche de la fourmiliere
 */
private void suivrePheromone(int direction){ //1:vers la fourmiliere , -1:vers la nourriture
    if(iterateur<pheromone.size() && iterateur>=0){
        Point2D.Double p=(Point2D.Double) pheromone.get(iterateur);
        double xd=p.getX(); //On va sur le point precedent de la liste
        double yd=p.getY(); //chainee (le suivant vers la destination)
        double d=distance(x,y,xd,yd);
        if(d==0)
            roty=trouverChemin(xd,yd);
        else if(d<DISTANCE_ENTRE_2_PHEROMONES/2) //Si on est assez proche du point de controle, on
            passe au suivant
            iterateur+=direction;
        else if(d>DISTANCE_ENTRE_2_PHEROMONES)
            roty=trouverChemin(xd,yd); //On s'eloigne, donc on recalcule
    }
    if(pheromone==null || pheromone.isEmpty()){
        etatPrimaire=ETAT_AU_NID;
    }
}
}

```

/******RAMENER******/



```

/*
 * Selon que le chemin est appris, ou decouvert, on suit ou crée des traces de pheromones
 * pour ramener la nourriture a la maison
 */

```

```

private void rameneNourriture(){
    //Soit je sais comment je suis venu, soit je dois retrouver
    if(distance(fourmiliere)<DISTANCE_ENTRE_2_PHEROMONES){
        etatPrimaire=ETAT_AU_NID;
    }
}

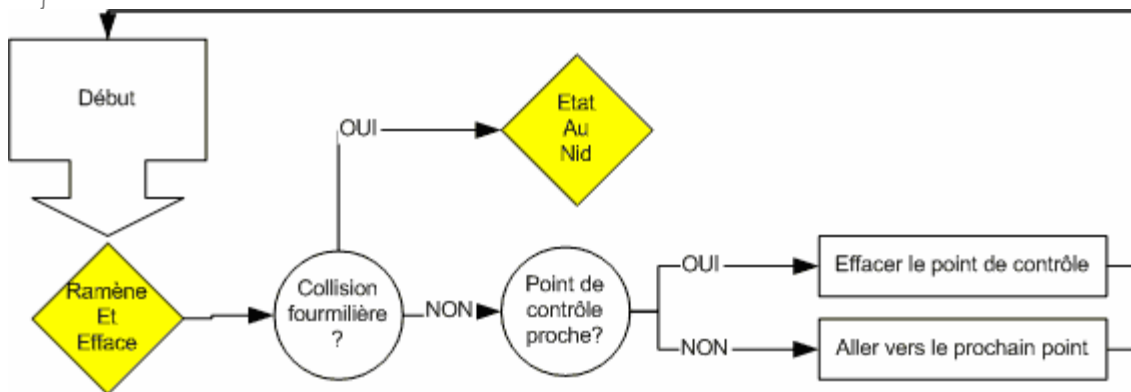
```

```

}else{
    if(cheminAppris){
        if(!pheromone.isEmpty()){

                suivrePheromone(1);
            }else{
                rentrerFourmilier();
            }
        }else{
            rentrerFourmilier();
            rentrerAvecDesPheromones(true);
        }
    }
}
}
}

```



```

/*
 *CREATION: On se dirige vers la fourmiere et si on est loin du precedent on crée un nouveau Checkpoint (indice 0 -
 > taille-1)
 *DESTRUCTION: On se dirige vers la pheromone suivante et si on en est suffisamment proche, on la detruit pour
 passer au suivant. (taille-1 -> 0)
 */

```

```

private void rentrerAvecDesPheromones(boolean creer){ //true pour creer, false pour effacer
    if(distance(fourmilier)>=DISTANCE_ENTRE_2_PHEROMONES && !pheromone.isEmpty()){
        //Je ne suis pas a la fourmiere et que la piste existe encore
        Point2D.Double p;
        if(creer)
            p=(Point2D.Double) pheromone.getLast();
        else
            p=(Point2D.Double) pheromone.getFirst();
        double xd=p.getX(); //On va sur le point suivant de la liste chainee
        double yd=p.getY();
        double d=distance(x,y,xd,yd);

        if(creer){
            if(d==0)rentrerFourmilier(); //Initialisation angle de depart

            if(d>=DISTANCE_ENTRE_2_PHEROMONES){ //Si on est suffisamment loin du dernier
                pheromone.add(new Point2D.Double(x,y)); //Alors on en cree un
                rentrerFourmilier();
                iterateur++;
            }
        }else{
            suivrePheromone(1);
            if(d<DISTANCE_ENTRE_2_PHEROMONES/2){
                pheromone.removeFirst(); //Alors on l'efface
            }
        }
    }
}

```

```

    }
    nourriture-=FATIGUE_RAMENE;
} else { //Me voici rentre a la fourmiere ou ma liste a ete effacee
    etatPrimaire=ETAT_AU_NID;
}
}

protected void mortDeFaim(){
    //Si une des fourmis manque de nourriture, elle meurt, a moins qu'elle transporte de la nourriture
    if(nourriture<0){
        if(capacite==CAPAMAX_OUVRIERE){
            //J'ai rien sur moi, je meurs
            tuer();
            // System.out.println(etatPrimaire);
        } else if (capacite<CAPAMAX_OUVRIERE){
            //j'ai de la nourriture sur le dos, je me sers et ramene le reste
            double dispo=CAPAMAX_OUVRIERE-capacite;
            double requis=(NOURRIMAX-nourriture)/4;
            if(dispo>requis){
                capacite-=requis;
                nourriture+=requis;
            }
            etatPrimaire=ETAT_AU_NID;
        }
    }
}

private void stocker(ObjetNourriture n2){
    if(capacite>ACTION_OUVRIERE){
        capacite-=n2.subStock(ACTION_OUVRIERE);
    } else {
        capacite-=n2.subStock(capacite);
    }
}

private void manger(ObjetNourriture n2){
    if(nourriture>NOURRIMAX)
        nourriture+=n2.subStock(NOURRIMAX-nourriture);
}

//-----
/**
 * Dessin 2D de l'objet
 */
public void draw(Graphics g, double facteurZoom) {

    int xbase = (int)(x*facteurZoom);
    int ybase = (int)(y*facteurZoom);
    int largeurbase = (int)(facteurZoom+2);
    g.setColor(new Color (getRed(), getGreen(), getBlue()));
    g.fillOval(xbase,ybase,largeurbase,largeurbase);
    if(capacite<CAPAMAX_OUVRIERE){
        //Dessine un point jaune si elle transporte de la nourriture.
        g.setColor(Color.yellow);
        g.fillOval(xbase-1+largeurbase/2,ybase-1+largeurbase/2,1,1);
    }
}
}
}

```

7.2.8 Classe ObjetFourmiSoldat

```
public class ObjetFourmiSoldat extends ObjetFourmi {

    public static final int CAPAMAX_SOLDAT = 100 ;

    public static final int ETAT_POURSUIT=104;
    public static final int ETAT_COMBAT=105;

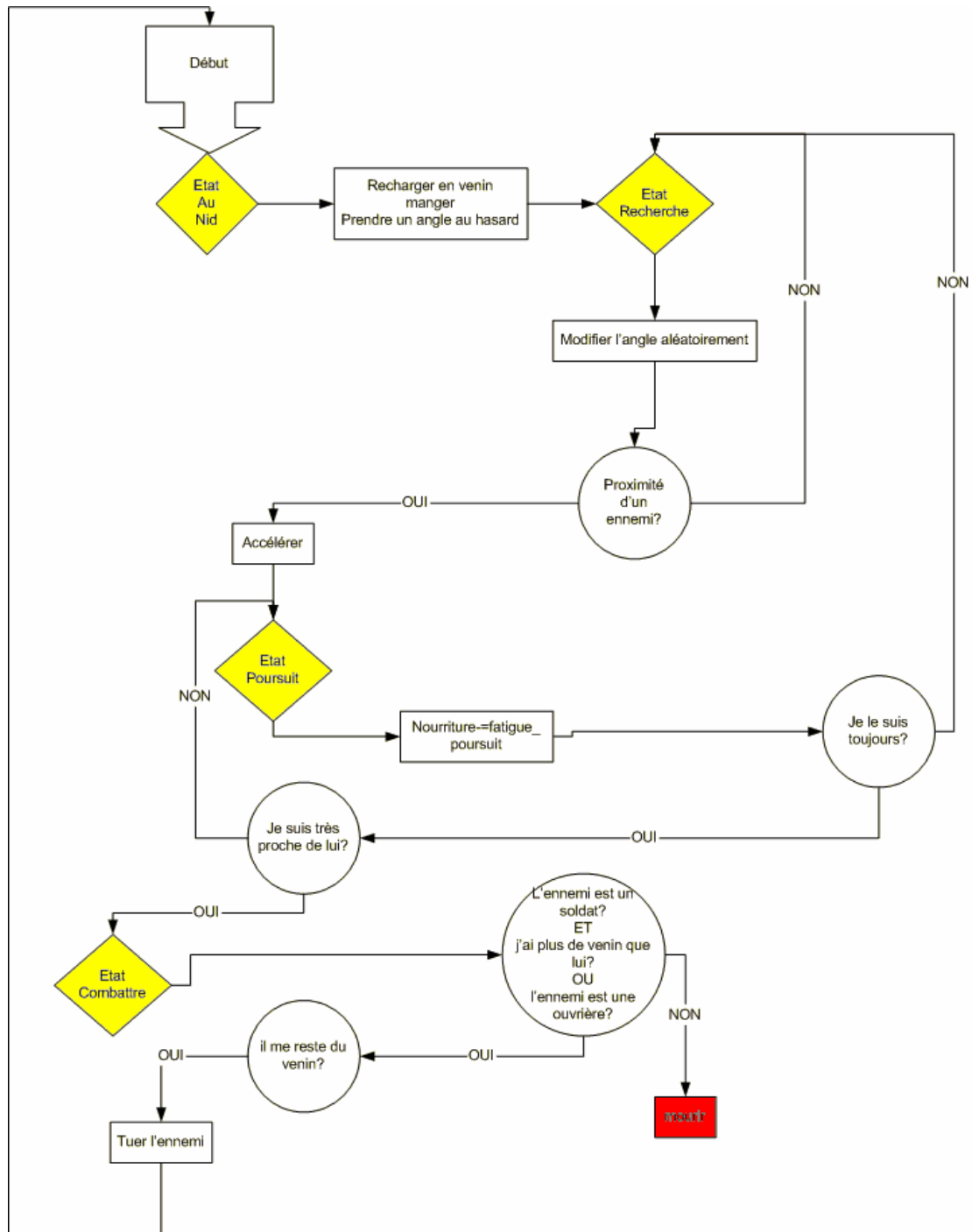
    public static final double FATIGUE_POURSUIT=FATIGUE_RECH*1.5 ;
    public static final double FATIGUE_COMBAT=FATIGUE_RECH*2.0 ;

    public static final int ACTION_SOLDAT=20 ;

    /**
     * Constructeur
     *
     * @param f      fourmiere de la fourmi
     * @param x0     coord x de la fourmi
     * @param y0     coord y de la fourmi
     * @param id0    numero de la fourmi
     * @param idgrp0 numero de la fourmiere (définit la couleur de la fourmi)
     */
    public ObjetFourmiSoldat(Simulation s, ObjetFourmiere f, int x0, int y0, int id0, int idgrp0){
        super (s,f,x0, y0, id0, idgrp0) ;
        capacite=CAPAMAX_SOLDAT;
        role=ROLE_SOLDAT;
        scale=0.4; //+ gros en openGL que la fourmi classique
    }

    //-----
    /**
     * simulation de l'objet
     *
     * Elle se balade, si elle repere un ennemi elle le suit et le combat,
     * si c'est une fourmi classique, elle la tue et perd peu de sa capacite de combat
     * si c'est une fourmi soldat, celle a la plus grande capacite l'emporte
     */

    protected void actionFourmi(){
        //Traitement etat primaire
        switch(etatPrimaire){
            case ETAT_AU_NID:  if(distance(fourmiere)<3) //J'y suis
                               actionsAuNid();
                               //J'y vais
                               rentrerFourmiere();
                               nourriture-=FATIGUE_RECH;
                               break;
            case ETAT_RECHERCHE:  rechercheEnnemi(); break;
            case ETAT_POURSUIT:   poursuitEnnemi(); break;
            case ETAT_COMBAT:     combatEnnemi(); break;
        }
        if(fourmiere.distance(x,y)>250){ //Je dois rester pres de ma fourmiere
            rentrerFourmiere();
        }
    }
}
```



```

/*****
/*****ETAT_AU_NID*****/
/*****/

```

```

private void actionsAuNid(){
    //On recharge en venin
    capacite=CAPAMAX_SOLDAT;
    //On se prepare a retourner chercher la bagarre
    etatPrimaire=ETAT_RECHERCHE;
    vitesse=VITESSE1;
    //On part au hasard
    roty=(float)(Math.random()*360.0);
    //On se remplit la panse si on peut
    manger(fourmilier);
}

```

```

/*****/
/*****ETAT_RECHERCHE*****/
/*****/
private void rechercheEnnemi(){
    //chercher me fatigue
    nourriture-=FATIGUE_RECH;
    //Si un ennemi est a portee
    if(simu.isEnemy(x,y,idGroup)!=null){
        etatPrimaire=ETAT_POURSUIT;
        //On le poursuit en accelerant
        vitesse=VITESSE2;
    }
    //Si manque de nourriture, il retourne a la fourmiere
    aFaim();
}

/*****/
/*****ETAT_POURSUIT*****/
/*****/
private void poursuitEnnemi(){
    //Poursuivre me fatigue
    nourriture-=FATIGUE_POURSUIT;

    ObjetFourmi fourmi;
    //Si l'ennemi est a portee
    if(simu.isEnemy(x,y,idGroup)!=null){
        fourmi=(ObjetFourmi)simu.isEnemy(x,y,idGroup);
        //On le poursuit!
        roty=trouverChemin(fourmi.getX(),fourmi.getY());
        //Si on est tres proche, on accelere et declenche l'attaque
        if(distance(fourmi)<5){
            vitesse=VITESSE3;
            etatPrimaire=ETAT_COMBAT;
        }
    }else //J'ai perdu mon ennemi, je retourne a la recherche d'un nouvel ennemi
        etatPrimaire=ETAT_RECHERCHE;
}

/*****/
/*****ETAT_COMBAT*****/
/*****/
private void combatEnnemi(){
    //Combattre me fatigue
    nourriture-=FATIGUE_COMBAT;

    ObjetFourmi f2=(ObjetFourmi) simu.isEnemy(x,y,idGroup);

    //Si mon ennemi est toujours la
    if(f2!=null){
        //Le gagnant est celui qui a le plus de venin (pour les soldats) ou le soldat (soldat-classique)
        if(f2.getRole()==ROLE_SOLDAT && capacite>f2.getCapacite() || f2.getRole()==ROLE_OUVRIERE){
            //Je gagne

```

```

        if(capacite>0){
            capacite-=ACTION_SOLDAT;
            f2.tuer();
        }
    }else{
        tuer();
    }
    etatPrimaire=ETAT_RECHERCHE;
}
}

```

//Contrairement a la fourmi classique, celle ci n'a jamais de nourriture sur le dos, donc n'a pas de reserve

```

protected void mortDeFaim(){

```

//Si une des fourmis manque de nourriture, elle meurt, a moins qu'elle transporte de la nourriture

```

if(nourriture<0){

```

//J'ai rien sur moi, je meurs

```

    tuer();
}
}

```

```

//-----

```

```

/**

```

** Dessin 2D de l'objet*

```

*/

```

```

public void draw(Graphics g, double facteurZoom) {

```

```

    int xbase = (int)(x*facteurZoom) ;

```

```

    int ybase = (int)(y*facteurZoom) ;

```

```

    int largeurbase = (int)(facteurZoom+2) ;

```

```

    g.setColor(new Color (getRed(), getGreen(), getBlue()));

```

```

    g.fillOval(xbase,ybase,largeurbase,largeurbase) ;

```

```

    g.setColor(new Color (getRed(), getRed(), getRed()));

```

```

    g.drawOval(xbase,ybase,largeurbase,largeurbase) ;
}
}

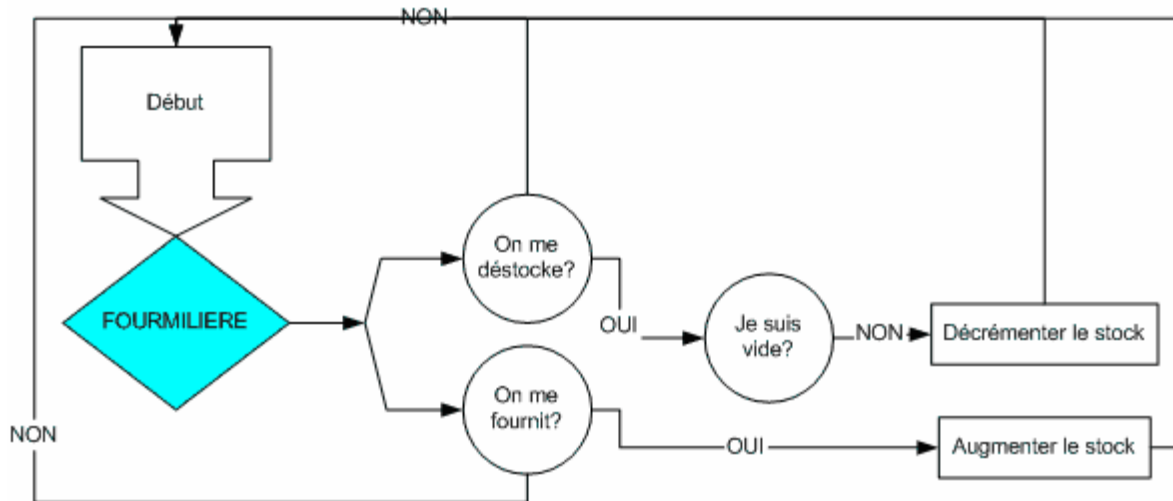
```

```

}

```

7.2.9 Classe ObjetFourmilière



```

public class ObjetFourmiliere extends Objet {
    double oldfacteurZoom=-1;
    double stock; //La stock de nourriture de la fourmiliere
    LinkedList pheromones;
    double oldStock1;
    /**
     * Constructeur
     *
     * @param x0 coord x de la fourmiliere
     * @param y0 coord y de la fourmiliere
     * @param id0 numero de la fourmiliere (définit sa couleur)
     */
    public ObjetFourmiliere(int x0, int y0, int id0){
        super (x0, y0, id0, 0);
        type = FOURMILIERE ;
        stock=10000.0;
        setColor(Objet.RED[id0], Objet.GREEN[id0], Objet.BLUE[id0]);
        scale = 1.0;
        pheromones=new LinkedList();
    }
    //Ajoute le chemin uniquement s'il n'est pas déjà présent dans la liste des pheromones
    public void addChemin(LinkedList l){
        boolean dejaPresent=false;
        Point2D.Double p=(Point2D.Double) l.getFirst();
        for(int j=0;j<pheromones.size();j++){
            LinkedList k=(LinkedList) pheromones.get(j);
            if(k!=null && !k.isEmpty()){
                Point2D.Double q=(Point2D.Double) k.getFirst();
                if(p.distance(q)<2){
                    dejaPresent=true;
                }
            }
        }
        if(!dejaPresent){
            pheromones.add(l);
        }
    }
    //Retire un chemin de pheromone parmi la liste des pheromones de la fourmiliere
    public void removeChemin(int i){if(i>=0 && i<pheromones.size())pheromones.remove(i);}
}

```

```

//Nettoyage de la liste de pheromones (On ne garde que les listes non defaillantes et non nulles)
public void nettoyageChemin(){
    for(int j=0;j<pheromones.size();j++){ //On recherche les pistes de tailles <= 1
        LinkedList k=(LinkedList) pheromones.get(j);
        if(k==null || k.size()<=1){
            pheromones.remove(j);
        }
    }
}

//Retourne la liste des chemins de pheromones
public LinkedList getChemins(){ return pheromones; }

//Ajoute de la nourriture a la fourmiere
public void addStock(double r){stock+=r;}

//Retourne la richesse de denrées gastronomique de la fourmiere
public double getStock(){return stock;}

//Retrait d'une quantité souhaitée qui ne se fera que si le stock est suffisant (sinon on donne ce qu'on peut)
public double subStock(double r){
    if(stock>=r){
        stock-=r;
    }else{
        r=stock;
        stock=0.0;
    }
    //System.out.println("on me mange "+r+" il me reste "+stock );
    return r; //Retourne la quantité consommée
}

//-----
/**
 * simulation de l'objet
 */
public void run() {
    nettoyageChemin();
}

//Choisit si une fourmi doit être créée ou pas: Elle doit avoir un stock suffisant et
//avoir des chemins de pheromones à exploiter et avoir de la chance
public void bonificationFourmiere(Simulation simu){
    if((stock>20000) && (Math.random()<0.01) && !pheromones.isEmpty()){
        simu.ajouterObjetFourmi((int)x,(int)y,id,this);
    }
    scale=stock/10000.0;
}

//Dessine chaque chemin de pheromones de la nourriture vers la fourmiere.
public void drawPheromones(Graphics g, double facteurZoom) {
    g.setColor(Color.orange);
    if(pheromones!=null && !pheromones.isEmpty()){
        for(int i=0;i<pheromones.size();i++){
            LinkedList cheminEnCours=(LinkedList)pheromones.get(i);
            if(cheminEnCours!=null && !cheminEnCours.isEmpty()){
                int xi,yi,xj,yj;
                Point2D.Double p=(Point2D.Double)cheminEnCours.get(0);

```

```

        xi=(int)(p.getX()*facteurZoom);
        yi=(int)(p.getY()*facteurZoom);
        for(int j=0;j<cheminEnCours.size()-1;j++){
            Point2D.Double p2=(Point2D.Double)cheminEnCours.get(j+1);
            xj=(int)(p2.getX()*facteurZoom);
            yj=(int)(p2.getY()*facteurZoom);
            g.drawLine(xi,yi,xj,yj);
            xi=xj;
            yi=yj;
        }
        xj=(int)(x*facteurZoom);
        yj=(int)(y*facteurZoom);
        g.drawLine(xi,yi,xj,yj);
    }
}

//Affichage 2D... on commence a le savoir ;)
public void draw(Graphics g, double facteurZoom) {
    drawPheromones(g,facteurZoom);

    if ((polygon==null) || (oldfacteurZoom!=facteurZoom) || stock!=oldStock1) {
        oldStock1=stock;
        oldfacteurZoom = facteurZoom ;
        polygon = new Polygon() ;
        int xbase = (int)(x*facteurZoom) ;
        int ybase = (int)(y*facteurZoom) ;
        int largeurbase = (int)(6*facteurZoom+6+stock/10000) ;
        // Triangle
        polygon.addPoint(xbase-largeurbase, ybase+largeurbase) ;
        polygon.addPoint(xbase+largeurbase, ybase+largeurbase) ;
        polygon.addPoint(xbase, ybase-largeurbase) ;
    }
    g.setColor(new Color (getRed(), getGreen(), getBlue()));
    g.fillPolygon(polygon) ;
}
}

```

7.2.10 Classe Simulation

```
public class Simulation implements Runnable
{
    public static final double CONV_DEG_RAD = Math.PI/180.0 ;
    public static final double CONV_RAD_DEG = 180/Math.PI ;

    public static final int MAX_FOURMILIERE = 7 ;

    public static final int NBR_OBSTACLES_INIT = 40 ;
    public static final int NBR_NOURRITURES_INIT = 40 ;
    public static final int NBR_FOURMILIERES_INIT = 4 ;
    public static final int nbrFourmilInitial = 10 ; // par fourmilier
    public static final int NBR_PREDATEURS_INIT = 2 ;

    protected float largeur,profondeur ;

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

    private Terrain terrain ;

    private Vector listeObjets ;

    private boolean finished ;
    private boolean exitRun=false ;

    private int attenteSimu=40 ;

    //-----
    /**
     * Constructeur par défaut
     */
    public Simulation(int l, int p) {
        largeur = l ;
        profondeur = p ;
        nbrObstacles = NBR_OBSTACLES_INIT ;
        nbrNourritures = NBR_NOURRITURES_INIT ;
        nbrFourmilieres = NBR_FOURMILIERES_INIT ;
        nbrFourmis = nbrFourmilInitial ;
        nbrPredateurs=NBR_PREDATEURS_INIT ;

        genererObjets() ;
    }

    //-----
    public float getLargeur() { return largeur ;}
    public float getProfondeur() {return profondeur ;}

    //-----
    /**
```

```

* Reinit
*/
public void reinit() {
    genererObjets();
    this.stop();
}

//les 4 méthodes qui suivent récupèrent les nb d'éléments à afficher
public void setNbFourmis(int nb_fourmis){
    nbrFourmis = nb_fourmis;
}
public void setNbObstacles(int nb_Obstacles){
    nbrObstacles = nb_Obstacles;
}
public void setNbFourmilliere(int nb_Fourmilliere){
    nbrFourmillieres = nb_Fourmilliere;
}
public void setNbPommes(int nb_Pommes){
    nbrNourritures = nb_Pommes;
}
public void setNbPredateurs(int nb_Predz){
    nbrPredateurs = nb_Predz;
}
public void setAttente(int attente){
    attenteSimu = attente;
}

//-----
/**
* Generation de l'environnement
*/
public void genererObjets() {
    listeObjets = new Vector();
    int x=0;
    int y=0;
    for (int i=0;i<nbrObstacles;i++){
        x = (int)(Math.random()*largeur);
        y = (int)(Math.random()*profondeur);
        float ry=(float)(Math.random()*360.0);
        float s=(float)(0.5+Math.random()*2.0);
        while(isQuelqueChose(x,y)){ //Non chevauchement des elements
            x = (int)(Math.random()*largeur);
            y = (int)(Math.random()*profondeur);
        }
        listeObjets.add (new ObjetObstacle(x, y, ry, s, i,0));
    }
    for (int i=0;i<nbrNourritures;i++){
        x = (int)(Math.random()*largeur);
        y = (int)(Math.random()*profondeur);

        float ry=(float)(Math.random()*360.0);
        float s=(float)(0.5+Math.random()*0.5);
        while(isQuelqueChose(x,y)){ //Non chevauchement des elements
            x = (int)(Math.random()*largeur);
            y = (int)(Math.random()*profondeur);
        }
        listeObjets.add (new ObjetNourriture(x, y, ry, s, i,0));
    }
    for (int i=0;i<nbrFourmillieres;i++){
        x = (int)(Math.random()*largeur);
        y = (int)(Math.random()*profondeur);
    }
}

```

```

while(isQuelqueChose(x,y)){ //Non chevauchement des elements
    x = (int)(Math.random()*largeur);
    y = (int)(Math.random()*profondeur);
}
if (i==0) {
    // 3D
    x = 1300 ; y = 1000 ;
}

ObjetFourmilier of = new ObjetFourmilier(x, y, i);
listeObjets.add (of);
int nbrSoldat=nbrFourmis/4; //On utilise 1/4 de soldats et 3/4 d'ouvrieres
for (int j=0;j<nbrFourmis-nbrSoldat;j++) {
    ObjetFourmi f = new ObjetFourmiOuvriere(this, of, x, y, j, i);
    listeObjets.add (f);
}
for (int j=nbrFourmis-nbrSoldat;j<nbrFourmilInitial;j++) {
    ObjetFourmi f = new ObjetFourmiSoldat(this, of, x, y, j, i);
    listeObjets.add (f);
}

}

for(int m=0;m<nbrPredateurs;m++){
    x = (int)(Math.random()*largeur);
    y = (int)(Math.random()*profondeur);
    while(isQuelqueChose(x,y)){ //Non chevauchement des elements
        x = (int)(Math.random()*largeur);
        y = (int)(Math.random()*profondeur);
    }

    ObjetPredateur pred = new ObjetPredateur(this, x, y);
    listeObjets.add(pred);
}

}

```

```

.....

//-----
/**
 * run du thread
 */
public void run() {
    exitRun = false ;
    while (!finished) {
        int i = 0 ;
        while (i<getNbrObjets()) {
            Objet obj = getObjet(i);
            if (obj!=null) { //Ajout des nouvelles fourmis
                if(obj.getType()==Objet.FOURMILIERE){
                    ObjetFourmilier of=(ObjetFourmilier)obj;
                    of.bonificationFourmilier(this);
                }
                if(! obj.estEnVie()){ //Ramasse morts - SORTEZ VOS MORTS! (Paris - 1348)
                    removeObjet(i);
                    if(((int)(Math.random()*3))==1){
                        ajouterObjetAleatoire();
                        //Rajoute un Objet Nourriture (une chance sur 3)
                    }
                }
            }
            i++;
        }
    }
}

```

```

        }
        }else{ obj.run(); }
    }
    i++;
}

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

.....

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

//Renvoie le nombre d'objets d'un type specifique
public int getNbrObjets(int type){
    int nb=0;
    int i = 0 ;
    int imax=getNbrObjets();
    while (i<imax) { //Pour chaque objet
        Objet obj = getObjet(i) ;
        if ((obj!=null) && (obj.getType()== type) )
            nb++;
        i++ ;
    }
    return nb ;
}

//Renvoie le nombre d'objets selon le type
public int getNbrObstacles(){ return getNbrObjets(Objet.OBSTACLE); }
public int getNbrNouritures(){ return getNbrObjets(Objet.NOURRITURE); }
public int getNbrFourmis(){ return getNbrObjets(Objet.FOURMI); }

//Renvoie le ieme objet de la listeObjets
public Objet getObjet(int i) {
    if(i<getNbrObjets())
        return (Objet)listeObjets.get(i);
    return null;
}

//Retirer le ieme objet de la listeObjets
public void removeObjet(int i) { listeObjets.remove(i) ;}

//Ajouter un objet a listeObjets
public void addObjet(Objet o) { listeObjets.add(o) ;}

//Retourne l'objet de type demandé s'il correspond aux criteres (x,y et evt idGroup)
public Objet proximite(double x, double y, int type2,int idGroup) {
    int i = 0 ;
    int imax=getNbrObjets();
    while (i<imax) {
        Objet obj = getObjet(i) ;
        boolean cond=false;

```

```

        if ((obj!=null) && (obj.getType()==type2)) {
            switch(type2){
                case Objet.OBSTACLE:
                    cond=obj.contains(x,y);                break;
                case Objet.NOURRITURE:
                    cond=obj.contains(x,y);                break;
                case Objet.FOURMILIERE:
                    if(idGroup>=0)
                        cond=(obj.getId()== idGroup) && obj.contains(x,y) ;
                    else
                        cond=obj.contains(x,y) ;
                    break;
                case Objet.FOURMI:
                    cond=(obj.getIdGroup()!= idGroup) && (obj.distance(x,y)<20);
                    break;
            }
            if(cond)
                return obj ;
        }
        i++;
    }
    return null ;
}

//-----
/**
 * Detection d'objets
 */
public Objet isEnemy(double x, double y,int idGroup) { return proximite(x,y,Objet.FOURMI,idGroup); }
public Objet isObstacle(double x, double y) { return proximite(x,y,Objet.OBSTACLE,-1); }
public Objet isFood(double x, double y) { return proximite(x,y,Objet.NOURRITURE,-1); }
public Objet isFourmilier(double x, double y) { return proximite(x,y,Objet.FOURMILIERE,-1); }
public Objet isFourmilier(double x, double y,int idGroup) { return proximite(x,y,Objet.FOURMILIERE,idGroup); }
public boolean isQuelqueChose(double x, double y) { return (isFood(x,y)!=null || isObstacle(x,y)!=null ||
isFourmilier(x,y)!=null); }

/*Trouve le numero qu'on va donner a la nouvelle fourmi! et qui n'est pas occupé*/
public int trouverIndiceLibre(int idGroup){
    int indice=-1;
    int i = 0 ;
    int imax=getNbrObjets();
    while (i<imax) { //Pour chaque objet
        Objet obj = getObject(i) ;
        if ((obj!=null) && (obj.getType()== Objet.FOURMI) && (obj.getIdGroup()== idGroup) &&
obj.getId()>indice) {
            //Fourmi de la meme fourmilier, on prend le plus grand indice+1 pour la nouvelle fourmi
            indice=obj.getId();
        }
        i++;
    }
    return indice+1 ;
}

//Ajouter une fourmi a une fourmilier aleatoirement (l'ajout, pas la fourmilier)
public void ajouterObjetFourmi(int x, int y, int i,ObjetFourmilier of){
    ObjetFourmi f=null;
    int j=trouverIndiceLibre(i);
    switch((int)(6*Math.random())){// Une chance sur 6 d'avoir un soldat
        case 2:
            f = new ObjetFourmiSoldat(this, of, x, y, j, i) ;
    }
}

```

```

        break;
    default:
        f = new ObjetFourmiOuvriere(this, of, x, y, j, i);
        break;
    }
    listeObjets.add (f) ;
}

//Ajouter une nourriture (2 chances sur 3)
public void ajouterObjetAleatoire(){
    if(((int)(Math.random()*6)>2){
        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);
        while(isQuelqueChose(x,y)){
            x = (int)(Math.random()*largeur) ;
            y = (int)(Math.random()*profondeur) ;
        }
        listeObjets.add(new ObjetNourriture(x, y, ry, s, 0,0));
    }
}
}
}

```

7.2.11 Classe SimAntJBoutons

```
public class SimAntJBoutons extends JFrame{

    public Simulation simu ;

    private JMenuBar panneauBoutons;

    //les intitulés et les sliders
    private JPanel panneau ;
    private JLabel INbFourmis ;
    private JSlider jNbFourmis;
    private JLabel INbFourmilliere ;
    private JSlider jNbFourmilliere;
    private JLabel INbPredateurs ;
    private JSlider jNbPredateurs;
    private JLabel INbPommes ;
    private JSlider jNbPommes;
    private JLabel INbObstacles ;
    private JSlider jNbObstacles;
    //utilisation de l'objet GridBagLayout pour la mise en page
    private GridBagLayout gridbag = new GridBagLayout();
    private GridBagConstraints constraints = new GridBagConstraints();
    //boutons
    private JButton b_stop;
        private JButton b_start;
    //nb_elements au début de l'anim
    private int nbFourmis = 10;
    private int nbFourmilliere = 4;
    private int nbPommes = 40;
    private int nbObstacles = 40;
    private int nbPredateurs = 2;

    // permet de manipuler plus aisément le GridBagLayout
    void buildConstraints(GridBagConstraints gbc, int gx, int gy, int gw, int gh, int wx, int wy)
    {

        gbc.gridx = gx;
        gbc.gridy = gy;
        gbc.gridwidth = gw;
        gbc.gridheight = gh;
        gbc.weightx = wx;
        gbc.weighty = wy;

    }

    //-----
    /**
     * Constructeur
     */
    public SimAntJBoutons(Simulation simu2){
        this.simu=simu2;
        positionneElement();
    }

    void positionneElement(){
        //on affecte le GridBagLayout en tant que calque pour la mise en page
        panneauBoutons = new JMenuBar();
    }
}
```

```

panneauBoutons.setLayout(gridbag);

//mise en place du titre
JLabel titre = new JLabel("SIMANT PARK");
titre.setFont(new Font("Serif",Font.BOLD,30));
titre.setForeground(Color.black);
buildConstraints(constraints, 0,0, 2, 1, 10, 10);
constraints.fill = GridBagConstraints.NONE;
constraints.anchor = GridBagConstraints.CENTER;
gridbag.setConstraints(titre, constraints);
panneauBoutons.add(titre) ;

//mise en place du libellé parametres
JLabel parametres = new JLabel("--parametres--");
parametres.setFont(new Font("Helvetica",Font.BOLD,14));
buildConstraints(constraints, 0,1, 1, 1, 10, 10);
constraints.fill = GridBagConstraints.NONE;
constraints.anchor = GridBagConstraints.EAST;
gridbag.setConstraints(parametres, constraints);
panneauBoutons.add(parametres) ;

//mise en place du libellé nb fourmis
INbFourmis = new JLabel("nb fourmis par fourmilières : "+nbFourmis);
INbFourmis.setForeground(Color.blue);
buildConstraints(constraints, 0,2, 2, 1, 10, 10);
constraints.fill = GridBagConstraints.NONE;
constraints.anchor = GridBagConstraints.CENTER;
gridbag.setConstraints(INbFourmis, constraints);
panneauBoutons.add(INbFourmis) ;

//mise en place du slider nb fourmis
jNbFourmis = new JSlider(10, 200, nbFourmis);
jNbFourmis.setValue(nbFourmis);
buildConstraints(constraints, 0,3, 2, 1, 10, 10);
constraints.fill = GridBagConstraints.NONE;
constraints.anchor = GridBagConstraints.CENTER;
gridbag.setConstraints(jNbFourmis, constraints);
panneauBoutons.add(jNbFourmis) ;
jNbFourmis.addChangeListener(
// écouteur des changements d'états du slider
new ChangeListener(){
    public void stateChanged(ChangeEvent e)
    {
        nbFourmis = jNbFourmis.getValue();
        INbFourmis.setText("nb fourmis par fourmilières : "+nbFourmis);
    }
});

//mise en place du libellé nb pommes
INbPommes = new JLabel("nb pommes : "+nbPommes);
INbPommes.setForeground(Color.blue);
buildConstraints(constraints, 0,4, 2, 1, 10, 10);
constraints.fill = GridBagConstraints.NONE;
constraints.anchor = GridBagConstraints.CENTER;
gridbag.setConstraints(INbPommes, constraints);
panneauBoutons.add(INbPommes) ;

//mise en place du slider nb pommes
jNbPommes = new JSlider(0, 200, nbPommes);
jNbPommes.setValue(nbPommes);
buildConstraints(constraints, 0,5, 2, 1, 10, 10);
constraints.fill = GridBagConstraints.NONE;

```

```

constraints.anchor = GridBagConstraints.CENTER;
gridbag.setConstraints(jNbPommes, constraints);
panneauBoutons.add(jNbPommes);
jNbPommes.addChangeListener(
    new ChangeListener(){
        public void stateChanged(ChangeEvent e)
        {
            nbPommes = jNbPommes.getValue();
            INbPommes.setText("nb pommes : "+nbPommes);
        }
    }
);
//mise en place du libellé nb obstacles
INbObstacles = new JLabel("nb obstacles : "+nbObstacles);
INbObstacles.setForeground(Color.blue);
buildConstraints(constraints, 0, 6, 2, 1, 10, 10);
constraints.fill = GridBagConstraints.NONE;
constraints.anchor = GridBagConstraints.CENTER;
gridbag.setConstraints(INbObstacles, constraints);
panneauBoutons.add(INbObstacles);

//mise en place du slider nb obstacles
jNbObstacles = new JSlider(0, 200, nbObstacles);
jNbObstacles.setValue(nbObstacles);
buildConstraints(constraints, 0, 7, 2, 1, 10, 10);
constraints.fill = GridBagConstraints.NONE;
constraints.anchor = GridBagConstraints.CENTER;
gridbag.setConstraints(jNbObstacles, constraints);
panneauBoutons.add(jNbObstacles);
jNbObstacles.addChangeListener(
    new ChangeListener(){
        public void stateChanged(ChangeEvent e)
        {
            nbObstacles = jNbObstacles.getValue();
            INbObstacles.setText("nb obstacles : "+nbObstacles);
        }
    }
);

//mise en place du libellé nb fourmillières
INbFourmilliere = new JLabel("nb Fourmillières : "+nbFourmilliere);
INbFourmilliere.setForeground(Color.blue);
buildConstraints(constraints, 0, 8, 2, 1, 10, 10);
constraints.fill = GridBagConstraints.NONE;
constraints.anchor = GridBagConstraints.CENTER;
gridbag.setConstraints(INbFourmilliere, constraints);
panneauBoutons.add(INbFourmilliere);

//mise en place du slider nb fourmillières
jNbFourmilliere = new JSlider(1, 7, nbFourmilliere);
jNbFourmilliere.setValue(nbFourmilliere);
buildConstraints(constraints, 0, 9, 2, 1, 10, 10);
constraints.fill = GridBagConstraints.NONE;
constraints.anchor = GridBagConstraints.CENTER;
gridbag.setConstraints(jNbFourmilliere, constraints);
panneauBoutons.add(jNbFourmilliere);
jNbFourmilliere.addChangeListener(
    new ChangeListener(){
        public void stateChanged(ChangeEvent e)
        {
            nbFourmilliere = jNbFourmilliere.getValue();
            INbFourmilliere.setText("nb Fourmillières : "+nbFourmilliere);
        }
    }
);

```

```

//mise en place du libellé nb predateurs
INbPredateurs = new JLabel("nb Prédateurs : "+nbPredateurs);
INbPredateurs.setForeground(Color.blue);
buildConstraints(constraints, 0,10, 2, 1, 10, 10);
constraints.fill = GridBagConstraints.NONE;
constraints.anchor = GridBagConstraints.CENTER;
gridbag.setConstraints(INbPredateurs, constraints);
panneauBoutons.add(INbPredateurs) ;

//mise en place du slider nb predateurs
jNbPredateurs = new JSlider(1, 7, nbPredateurs);
jNbPredateurs.setValue(nbPredateurs);
buildConstraints(constraints, 0,11, 2, 1, 10, 10);
constraints.fill = GridBagConstraints.NONE;
constraints.anchor = GridBagConstraints.CENTER;
gridbag.setConstraints(jNbPredateurs, constraints);
panneauBoutons.add(jNbPredateurs) ;
jNbPredateurs.addChangeListener(
    new ChangeListener(){
        public void stateChanged(ChangeEvent e)
        {
            nbPredateurs = jNbPredateurs.getValue();
            INbPredateurs.setText("nb Prédateurs : "+nbPredateurs);
        }
    });

//mise en place du libellé nb predateurs
IVitesseSimu = new JLabel("Vitesse : "+vitesseSimu);
IVitesseSimu.setForeground(Color.blue);
buildConstraints(constraints, 0,10, 2, 1, 10, 10);
constraints.fill = GridBagConstraints.NONE;
constraints.anchor = GridBagConstraints.CENTER;
gridbag.setConstraints(IVitesseSimu, constraints);
panneauBoutons.add(IVitesseSimu) ;

//mise en place du slider nb predateurs
jVitesseSimu = new JSlider(1, 40, vitesseSimu);
jVitesseSimu.setValue(vitesseSimu);
buildConstraints(constraints, 0,11, 2, 1, 10, 10);
constraints.fill = GridBagConstraints.NONE;
constraints.anchor = GridBagConstraints.CENTER;
gridbag.setConstraints(jVitesseSimu, constraints);
panneauBoutons.add(jVitesseSimu) ;
jVitesseSimu.addChangeListener(
    new ChangeListener(){
        public void stateChanged(ChangeEvent e)
        {
            vitesseSimu = jVitesseSimu.getValue();
            IVitesseSimu.setText("Vitesse: "+vitesseSimu);
            simu.setAttente(41 - jVitesseSimu.getValue());
        }
    });

//mise en place du libellé commandes
JLabel commandes = new JLabel("--commandes--");
commandes.setFont(new Font("Helvetica",Font.BOLD,12));
buildConstraints(constraints, 0,12, 1, 1, 10, 10);
constraints.fill = GridBagConstraints.NONE;
constraints.anchor = GridBagConstraints.EAST;

```

```
gridbag.setConstraints(commandes, constraints);
panneauBoutons.add(commandes);
```

```
//mise en place du bouton stop
b_stop = new JButton("PAUSE");
b_stop.setFont(new Font("Helvetica", Font.BOLD, 12));
buildConstraints(constraints, 0, 13, 1, 1, 10, 10);
constraints.fill = GridBagConstraints.HORIZONTAL;
constraints.anchor = GridBagConstraints.EAST;
gridbag.setConstraints(b_stop, constraints);
panneauBoutons.add(b_stop);
b_stop.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        //stope l anim
        simu.stop();
        //start est utilisable mais pas stop
        b_start.setEnabled(true);
        b_stop.setEnabled(false);
    }
});
```

```
//mise en place du bouton start qui commence inutilisable
b_start = new JButton("START");
b_start.setEnabled(false);
b_start.setFont(new Font("Helvetica", Font.BOLD, 12));
buildConstraints(constraints, 1, 13, 1, 1, 10, 10);
constraints.fill = GridBagConstraints.HORIZONTAL;
constraints.anchor = GridBagConstraints.CENTER;
gridbag.setConstraints(b_start, constraints);
panneauBoutons.add(b_start);
b_start.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        //on envoie a la classe Simulation les différents nb d'éléments à afficher
        simu.start();
        b_start.setEnabled(false);
        b_stop.setEnabled(true);
        jNbFourmis.setEnabled(true);
        jNbPommes.setEnabled(true);
        jNbFourmilliere.setEnabled(true);
        jNbObstacles.setEnabled(true);
        jNbPredateurs.setEnabled(true);
    }
});
```

```
//mise en place du bouton réinitialisation
JButton b_restart = new JButton("REINITIALISATION");
b_restart.setFont(new Font("Helvetica", Font.BOLD, 12));
buildConstraints(constraints, 0, 14, 3, 1, 10, 10);
constraints.fill = GridBagConstraints.HORIZONTAL;
constraints.anchor = GridBagConstraints.WEST;
gridbag.setConstraints(b_restart, constraints);
panneauBoutons.add(b_restart);
b_restart.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        jNbFourmis.setEnabled(false);
        jNbPommes.setEnabled(false);
        jNbFourmilliere.setEnabled(false);
        jNbObstacles.setEnabled(false);
        jNbPredateurs.setEnabled(false);
        simu.setNbFourmis(nbFourmis);
        simu.setNbPommes(nbPommes);
    }
});
```

```

        simu.setNbFourmilliere(nbFourmilliere);
        simu.setNbObstacles(nbObstacles);
        simu.setNbPredateurs(nbPredateurs);
        simu.stop();
        simu.reinit();
        b_start.setEnabled(true);
        b_stop.setEnabled(false);
    }

    try {
        UIManager.setLookAndFeel("com.sun.java.swing.plaf.motif.MotifLookAndFeel");
        SwingUtilities.updateComponentTreeUI((Container) panneauBoutons);
    } catch (Exception ex) { ex.printStackTrace(System.err);
    }

}

public JMenuBar getPanneauBoutons(){
    return(panneauBoutons);
}

}

```

7.2.12 Classe SimAntGLCanvas

class SimAntGLAnimCanvas **extends** GLAnimCanvas **implements** KeyListener, MouseMotionListener

```
{
    .....

//-----
/**
 * Dessin de la scene 3D
 */
public void DrawGLScene() {
    .....

// Dessin des objets 3D
int i = 0 ;
int num=0;
int imax = simu.getNbrObjets() ;
while (i<imax) {
    Objet obj = simu.getObjet(i) ;
    if (obj!=null) {
        if (!obj.isWithColor()) {
            gl.glColor3f(obj.getRed(), obj.getGreen(), obj.getBlue());
            gl.glDisable(gl.GL_TEXTURE_2D); // pour les fourmis
        }
        gl.glPushMatrix();
        if(obj.getId()==-1 && obj.getIdGroup()==-1)
            //c'est un cadavre
            num=5;
        else
            num=obj.getType()-1;

// On inverse Y et Z car les coordonnées x, y de la simu
// correspondent aux axes X et Z de la 3D (Y étant l'altitude).

if(num!=5){           //Objet Nourriture, Fourmi, Obstacle et Fourmiere
            gl.glTranslated(obj.getX(),obj.getZ(),obj.getY());
            gl.glRotatef((float)obj.getRot(),0,1,0);

        }else{         //Pour les cadavres, c'est une fourmi retournée
            gl.glTranslated(obj.getX(),obj.getZ(),obj.getY());
            gl.glRotatef((float)180,1,1,0);           //On la couche sur le coté
        }

        gl.glScaled(obj.getScale(),obj.getScale(),obj.getScale());
        // Appel de la liste d'affichage avec l'index correspondant
        // au type d'objet à afficher :
        // index d'indice 0 pour un obstacle (1)
        // index d'indice 1 pour la nourriture (2)
        // index d'indice 2 pour la fourmiere (3)
        // index d'indice 3 pour la fourmi (4)
        // index d'indice 5 pour la nourriture Cadaverique (2)
        gl.glCallList(Objet3DASE.getIndex(num)); //appel de la liste d'affichage

        gl.glPopMatrix();

    }
    i++;
}
```

}

.....
.....

7.2.13 Classe Objet3DASE

```
//-----  
/**  
  
* Chargement de tous les fichiers ASE et création des listes d'affichage  
*/  
public static void init(GLFunc gl){  
  if (cLoadASE==null) {  
    MAX_ASE = nomFichier.length ;  
    System.out.println ("Chargement des fichiers ASE 3D (MAX_ASE="+MAX_ASE+"") ;  
  
    cLoadASE = new CLoadASE[MAX_ASE] ;  
    t3DModel = new CLoadASE.t3DModel[MAX_ASE] ;  
    index = new int[MAX_ASE] ;  
    for (int i=0;i<MAX_ASE;i++) {  
      // Chargement des fichiers ASE  
      System.out.println ("Chargement du fichier ASE "+i+" : "+nomFichier[i]) ;  
      loadFileASE (i, nomFichier[i]) ;  
  
      // Liste d'affichage  
      switch (i+1) {  
        case Objet.OBSTACLE:  
        case Objet.NOURRITURE:  
          // Création d'une liste d'affichage avec couleurs  
          // provenant du fichier  
  
          genListe(gl, i, true) ;  
          break ;  
        case Objet.FOURMILIERE:  
        case Objet.FOURMI:  
        case Objet.PREDATEUR:  
          // Création d'une liste d'affichage sans les couleurs  
          // du fichier  
          genListe(gl, i, false) ;  
          break ;  
        case 6:// un CADAVRE  
          genListe(gl, i, true) ;  
      }  
    }  
  }  
}
```