Licence Informatique 2e année

Programmation Objet I

Sujet du projet 2017-2018

1 Organisation

Le projet est à réaliser en binôme. En fin de semestre, lors de la dernière séance de TP, les binômes présenteront leur travail aux encadrants de TP. Lors des présentations, chaque binôme rendra un rapport qui présentera le programme développé, les résultats obtenus, les problèmes rencontrés, les solutions apportées et tout élément permettant d'évaluer le travail réalisé. Chaque binôme devra également, au moment des soutenances, envoyer le code des programmes écrits aux responsables de TP (*christine.irastorza@u-picardie.fr* et *frederic.furst@u-picardie.fr*).

La note du projet dépendra en bonne partie du respect des principes de génie logiciel mis en œuvre en programmation objet, en particulier la modularité, l'abstraction, l'encapsulation et la documentation.

2 Sujet

Le projet consiste à réaliser une simulation d'une colonie de fourmis. Le projet doit comporter au moins 3 types de fourmis différentes : les *reines* (qui ne bougent pas et vivent plusieurs années), les *ouvrières* (qui vivent quelques semaines) et les *guerrières* (qui vivent quelques mois). La reine produit des ouvrières et des guerrières, à un certain rythme et dans une certaine proportion. Les ouvrières passent leur temps à chercher de la nourriture ou des matériaux pour consolider le nid. Elles se déplacent aléatoirement pour chercher mais reviennent ensuite directement au nid (à l'aide des phéromones déposées sur le sol) pour rapporter ce qu'elles ont trouvé. Les guerrières se promènent aléatoirement à la recherche de menace (gros animal ou autre fourmilière) et attaquent les intrus. Les animaux menaçants voient leurs forces baisser à chaque attaque d'une guerrière et meurent si ils sont attaqués suffisamment. Le comportement des animaux menaçants (façon de se déplacer, vitesse, etc) dépend du type d'animal.

Les fourmis consomment la nourriture stockée dans le nid. Chaque type de fourmi a une consommation propre (par exemple la reine consomme beaucoup plus de nourriture que les ouvrières ou les guerrières). La quantité de nourriture présente dans la fourmilière doit permettre de nourrir toutes les fourmis, faute de quoi leur espérance de vie décroît fortement. Quand il n'y a pas assez de nourriture, la reine est nourrie en priorité, puis les ouvrières, et les guerrières sont les moins prioritaires. Quand une fourmi n'est pas nourrie pendant un certain temps (qui dépend de son type), elle meurt.

Le terrain est décomposé en petits carrés qui peuvent contenir le nid, de la nourriture (en quantité finie), des matériaux de construction, une menace ou une fourmi. Le contenu des cases peut changer aléatoirement au cours du temps (de la nourriture ou une menace peut apparaître sur une case). Le contenu peut également changer en fonction des actions des fourmis (si une guerrière tue un animal menaçant, le cadavre constitue de la nourriture).

3 Gestion du temps

La simulation se fait en tour par tour, toutes les entités évoluent une fois par tour. Pour temporiser le jeu, on peut utiliser l'instruction *try{Thread.sleep(n);}catch(InterruptedException e){}* qui met le programme en attente *n* millisecondes. On peut aussi utiliser la classe *javax.swing.Timer* qui déclenche l'appel d'une fonction à intervalle régulier.

4 Interface graphique

Pour l'affichage et le contrôle du programme, une classe *GUI_for_Displayable* est fournie. Elle permet d'afficher n'importe quel objet implémentant l'interface *Displayable* fournie également. Cette interface impose que chaque objet affichable donne la forme qui le représente (instance de *java.awt.Shape*), la couleur utilisée pour l'afficher (instance de *java.awt.Color*) et le texte associé avec sa position. Pour avoir un texte sur plusieurs lignes, il faut ajouter des \n dans la chaîne.

La classe *GUI_for_Displayable* offre des constructeurs pour créer une interface avec un fond coloré ou avec une image de fond, des méthodes pour ajouter et retirer un *Displayable* de l'interface, et une méthode *getClic()* qui renvoie une instance de *java.awt.MouseEvent* correspondant au dernier clic non traité de l'utilisateur (les clics sont stockés dans une file d'attente). Un *MouseEvent* décrit un clic avec une position, le bouton appuyé, les touches appuyées au moment du clic (Shift, Alt, ...).

La classe offre également une méthode *setBottomFieldText(String s)* qui permet de spécifier le texte affiché dans le composant textuel en bas de la fenêtre, et une méthode *displayMessage(String s)* qui ouvre une fenêtre de dialogue pour afficher la chaîne de caractères. Il est possible d'afficher ainsi le ratio production/consommation, mais ce ratio peut aussi être affiché à l'aide de courbes par un objet de type *Displayable*.

L'interface peut être modifiée ou améliorée à votre convenance, mais ce n'est pas sur cet aspect du programme que vous serez notés.

5 Pour aller plus loin ...

Les extensions du projet décrites dans cette partie sont optionnelles, mais en traiter au moins une augmentera très sensiblement la note attribuée, SI LA PARTIE OBLIGATOIRE A ÉTÉ TRAITÉE !

Il est possible de simuler la création d'un nouveau nid à partir d'une reine : régulièrement (généralement au printemps), une reine donne naissance à des fourmis sexuées (et ailées pour pouvoir aller fonder de nouveaux nids loin de celui où elles sont nées). Les femelles qui sont fécondées peuvent partir fonder un nouveau nid. Les femelles non fécondées et les mâles meurent en quelques jours.

Il est possible de simuler les comportements coopératifs. Si on considère qu'un tas de nourriture, ou de matériaux, est trop important pour être ramené à la fourmilière en un voyage par une fourmi, elle va solliciter ses collègues. Elle doit pouvoir informer d'autres ouvrières de sa découverte et elles vont suivre la piste des phéromones pour aller récupérer la nourriture ou les matériaux.

Il est possible de simuler la malnutrition : une fourmi qui n'a pas mangé quand elle le devrait se déplace moins vite.

Il est possible de représenter plusieurs nids, et de simuler des guerres entre fourmilières.

Il est possible de simuler des événements catastrophiques (passage d'un fourmilier qui mange une bonne partie des occupantes du nid, inondation du nid suite à une forte pluie, etc) pour voir comment et en combien de temps le nid va se reconstituer (ou non).