Projet LO43 : Fourmilière
Travaux Pratiques

I) Analyse du problème

1) Génération du diagramme de classes à l'aide de jgrasp

- Copier le répertoire "/mnt/gi/LO43/projets_print2004/gfourmi3d" dans votre répertoire local.
- Démarrer l'application "jgrasp" et créer un nouveau projet
- A l'aide de l'explorateur de fichiers de jgrasp ("Browse"), retrouver les fichiers .class du projet. Sélectionner les fichiers ne contenant pas de caractère "$" et les ajouter au projet jgrasp.
- Générer alors le diagramme UML des classes à l'aide du menu UML.
- Les classes en blanc correspondent à des classes externes, qui ne font pas partie des classes du projet. Dans la fenêtre UML, dévalider l'affichage des classes externes et choisir le layout "spring", afin de rendre le diagramme de classes plus lisible.
- Supprimer les classes propre à l'affichage 2D ou 3D : Camera, Objet3DASE, SimAntGLAnimCanvas, SimAntJFrame, SimAntJPanel, SimAntMain, TerrainGL.

2) Analyse du diagramme de classes obtenu

- Vérifier dans le code source que les liens indiqués par le schéma UML simplifié, existent bien.
- Analyser ce schéma UML et proposer éventuellement des améliorations quant aux classes et à leurs relations.

II) Amélioration de la simulation

On souhaite améliorer le comportement des fourmis pour qu'elles détectent les obstacles et la nourriture et qu'elles réagissent en conséquence : contournement pour l'obstacle et collecte pour la nourriture.

1) Modifier la classe Simulation pour ne créer qu'une seule fourmilière, un obstacle et une pomme. La fourmilière ne contiendra qu'une seule fourmi, elle est positionnée par défaut en x = 1300 ; y = 1200 et d'angle de rotation roty=180.

2) La méthode genererObjets() de la classe Simulation permet de positionner les différents objets dans l'environnement. Positionner :
   - l'obstacle aux coordonnées x = 1300 ; y = 1200 et fixer sa taille s = (float)2.0 ;
   - la nourriture aux coordonnées x = 1300 ; y = 800 et fixer sa taille s = (float)2.0 ;

3) Nous allons maintenant créer une classe de fourmi particulière qui avance toujours dans la direction de l'axe Y. De cette façon, elle entrera forcément en contact avec l'obstacle ou la pomme. La méthode qui permet de faire avancer une fourmi est la méthode "void run()". Créer une classe de fourmi particulière, qui s'appellera "ObjetFourmiY", qui a les caractéristiques suivantes :
   - Un constructeur similaire au constructeur de la classe "ObjetFourmi" qui fait progresser la fourmi dans le sens des Y croissants,
   - Un constructeur acceptant en plus des paramètres précédents, un paramètre de type entier permettant de donner un sens positif ou négatif au déplacement de la fourmi selon l'axe Y,
   - Une méthode "run()" qui effectue un déplacement élémentaire de la fourmi selon l'axe des Y.

Le déplacement aura une longueur égale au champs “vitesse”. Si la fourmi détecte un obstacle grâce à la méthode “simu.isObstacle(x, y)”, on inverse le sens de déplacement de la fourmi.
- Modifier les protections de la classe “ObjetFourmi” en fonction de votre nouvelle classe et vérifier que la fourmi “rebondit” bien sur l'obstacle.
- Tester vos modifications. Que constatez-vous ?

4) On veut maintenant que la fourmi puisse détecter et collecter de la nourriture. Pour cela, on va utiliser la méthode “isObstacle(x,y)” qui permet de tester si un obstacle se trouve aux coordonnées (x, y). Pour comprendre le fonctionnement de cette méthode, nous pouvons remarquer que la classe “Objet” contient un attribut “polygon” qui est sensé représenter la zone occupée par l'objet concerné sur le terrain. Or, ce champ “polygon” n'est défini que pour la classe “ObjetObstacle” pour laquelle il a la forme d'un carré. Comme il n'est pas défini (null) pour la classe “ObjetNourriture”, les pommes ne sont pas considérées comme des obstacles et la fourmi passe à travers... Pour corriger cela, réaliser les actions suivantes :
   - Modifier votre diagramme de classe en considérant qu'une pomme est un obstacle, ce qui interdira à la fourmi de la traverser,
   - Modifier la classe “ObjetNourriture” en conséquence, et vérifier que la fourmi “rebondit” sur l'obstacle situé à droite et sur la pomme située à gauche.

5) Pour que la fourmi puisse chercher de la nourriture la collecter et la ramener à la fourmilière, il va falloir modéliser les états possibles de la fourmi. Pour cela, on utilise une variable “etatPrimaire” qui caractérise l'état principal de la fourmi :
   - au nid (constante ETAT_AU_NID),
   - à la recherche de nourriture (constante ETAT_RECHERCHE_NOURRITURE),
   - ramassage de nourriture (constante ETAT_COLLECTE_NOURRITURE),
   - ramener la nourriture au nid (constante ETAT_RAMENE_NOURRITURE).

   Cet état principal est détaillé par la variable “etatSecondaire”. Par exemple, quand la fourmi collecte de la nourriture, l'état secondaire représente l'état d'avancement de la collecte de 0 à 50. Les transitions entre ces états peuvent s'exprimer de la façon suivante :
   - Initialement, la fourmi est au nid. Elle se met alors à rechercher de la nourriture, en partant en ligne droite suivant l'axe des Y.
   - Si elle tombe sur un obstacle, elle inverse le sens de sa progression et part donc en sens opposé.
   - Si elle trouve de la nourriture, elle s'arrête et collecte de la nourriture. A chaque cycle, on incrémente la variable “etatSecondaire”, quand elle arive à 50, la collecte est finie, et la fourmi peut ramener la nourriture au nid. Pendant la phase de collecte, on pourra faire osciller la fourmi légérement de gauche à droite en utilisant “roty” pour simuler le fait qu'elle ramasse de la nourriture.
   - Quand elle ramène la nourriture, elle avance jusqu'à ce qu'elle se trouve dans la fourmilière. Pour vérifier cette condition, on pourra par exemple, tester que la position de la fourmi (x, y) par rapport au centre de la fourmilière (fourmiliere.getX(), fourmiliere.getY()) est à une distance inférieure à sa vitesse de déplacement. Lorsque le test est vérifié, la fourmi dépose sa nourriture, par conséquent la nourriture stockée dans la fourmilière augmente de la même quantité (confirmé par un message à l'écran) et on remet l'état de la fourmi à ETAT_AU_NID.

6) Maintenant, que vous avez réalisé ces différentes étapes, vous pouvez généraliser la démarche précédente et envisager les améliorations suivantes :
   - Revenir à l'environnement initial constitué d'obstacles, de nourriture et de fourmilières placés aléatoirement,
   - Retour au nid : comment calculer la direction de la fourmi quand elle cherche à rentrer au nid, que se passe-t-il si elle rencontre un obstacle sur sa route ?
   - Phéromone : quand la fourmi ramène la nourriture, elle laisse une trace de phéromone derrière elle. Comment modéliser cette phéromone sur l'environnement ? Comment peut-on prendre

en compte la phéromone environnante dans le calcul d'une direction aléatoire pour une fourmi cherchant de la nourriture ? Comment faire évaporer cette phéromone régulièrement ?

- Combat : comment peut-on envisager des interactions entre fourmis de différentes fourmilières ?
- Evolution : comment faire évoluer la population de chaque fourmilière en fonction de la nourriture collectée ?