université PARIS-SACLAY

Travaux Dirigés du projet Fourmis

Cours de programmation modulaire

—Licence MI/IM/MNSI - Info 3—

Types de base du projet Fourmis

Nous allons étudier les types abstraits découlant d'une analyse du projet de simulation de fourmis. Les questions qui suivent ont pour but de vous aider dans la réalisation du projet.
Nous allons considérer trois types abstraits : un type représentant une fourmi, un type représentant une place (c'est-à-dire une case) et un type représentant la grille. Dans un premier temps, les fourmis de la colonie sont simplement stockées dans un tableau. Il n'y a pas de type modélisant la colonie.

1 Les coordonnées

On suppose que l'on dispose des types abstraits `Coord` et `EnsCoord` que vous implémenterez en TP. Les fonctions utiles au présent TD seront les suivantes :

- le constructeur `Coord(int x, int y);` qui construit la coordonnée (x, y)
- la fonction `EnsCoord voisins(Coord c);` qui renvoie l'ensemble des coordonnées des cases voisines de c.

2 Les fourmis

Le type abstrait `Fourmi` représente une fourmi. Il contient les méthodes suivantes :

- Constructeur : crée une fourmi ne portant pas de sucre, à partir de ses coordonnées sur la grille et de son indice dans le tableau de fourmis.
- Accès :
  - `Fourmi::coord` renvoie les coordonnées de la fourmi
  - `Fourmi::num` renvoie le numéro de la fourmi
- Prédicats :
  - `Fourmi::porteSucre` renvoie vrai si la fourmi rentre au nid en portant du sucre
- Modifications :
  - `Fourmi::prendSucre` ajoute une charge de sucre sur la fourmi
  - `Fourmi::poseSucre` supprime la charge de sucre portée par la fourmi
  - `Fourmi::deplace` déplace une fourmi

1. Donnez les spécifications des contructeurs, fonctions et méthodes ci-dessus en précisant bien lesquelles sont constantes.

2. On suppose donné un ensemble de coordonnées représentant les positions des fourmis au départ. Dans la version simple du jeu, les fourmis seront stockées dans un simple tableau. Chaque fourmi doit connaître son indice dans le tableau. Quelle est l'entête de la fonction `creeTabFourmis` qui crée un tableau de fourmis positionnées aux coordonnées spécifiées par un ensemble de coordonnées ?

3 La grille

Le type abstrait `Place` représente une case de la grille. Il est spécifié de la manière suivante :

- Constructeur : crée une place dont les coordonnées sont fournies en paramètre, ne contenant pas de fourmi, ni de sucre, ni d'élément de nid
- Accès :
  - `Place::coord` renvoie les coordonnées d'une place
  - `Place::pheroSucre` renvoie l'intensité en phéromone de sucre sur la place
  - `Place::pheroNid` renvoie l'intensité en phéromone de nid sur la place
  - `Place::numeroFourmi` renvoie le numéro de la fourmi sur la place, ou -1 si pas de fourmi
- Prédicats sur une place :
  - `Place::contientSucre` renvoie vrai si la place contient du sucre
  - `Place::contientNid` renvoie vrai si la place contient un élément de nid
  - `Place::estSurUnePiste` renvoie vrai si la place se trouve sur une piste vers du sucre (tracée par la présence de phéromones de sucre)
- Modifications sur une place :
  - `Place::poseSucre` pose du sucre sur la place
  - `Place::enleveSucre` enlève du sucre sur la place
  - `Place::poseNid` pose un élément de nid sur la place
  - `Place::poseFourmi` pose la fourmi donnée en paramètre
  - `Place::enleveFourmi` enlève la fourmi dans la place
  - `Place::posePheroNid` pose sur la place une phéromone de nid d'intensité donnée
  - `Place::posePheroSucre` pose une phéromone de sucre d'intensité maximale sur la place
  - `Place::diminuePheroSucre` diminue (par évaporation) l'intensité en phéromone de sucre sur la place
- Modification sur deux places :
  - `deplaceFourmi` déplace la fourmi donnée en 1er paramètre qui est sur la place donnée en 2^{e} paramètre vers la place donnée en 3^{e} paramètre

3. Donnez les spécifications des contructeurs, fonctions et méthodes ci-dessus en précisant bien lesquelles sont constantes.
4. Réalisez un prédicat `estVide` qui prend une place en paramètre et renvoie vrai si la place ne contient ni sucre, ni élément de nid, ni fourmi.
5. Réalisez un prédicat `estPlusProcheNid` qui renvoie vrai si la place donnée en 1er paramètre est plus proche du nid que la place donnée en 2^{e} paramètre.

Le type abstrait `Grille` représente la grille. Voici un extrait de ses spécifications :

- Constructeur : initialise une grille vide.
- Accès et modifications :
  - `Grille::chargePlace` accède à une place p à modifier de la grille, située aux coordonnées indiquées. Renvoie une **Copie** de la place
  - `Grille::rangePlace` range la place dans la grille après l'avoir modifiée ; si on a rangé les coordonnées de la place dans celle-ci, le paramètre `Coord` est inutile.
- Méthodes avancées :
  - `Grille::linearisePheroNid` linéarise l'intensité en phéromones de nid des cases de la grille (voir plus loin).
  - `Grille::diminuePheroSucre` diminue suite à l'évaporation les phéromones de sucre sur toute les cases de la grille.

Attention ! lorsqu'on charge une place et qu'on la modifie, il ne faut pas oublier de la ranger dans la grille, sinon la modification n'est pas enregistrée.

6. Donnez les spécifications des contructeurs, fonctions et méthodes ci-dessus en précisant bien lesquelles sont constantes.
7. Réalisez une procédure `placeNid` qui place des éléments de nid sur la grille passée en premier paramètre, à l'ensemble des coordonnées fournies en 2^{e} paramètre.
   NB : On dispose pour cela de la structure de contrôle suivante pour itérer sur toutes les coordonnées `c` d'un ensemble de coordonnées `ec` :

   ```
   pour chaque c dans ec faire
     ...
   finpour
   ```

   Remarque : une procédure `placeSucre` peut être réalisée suivant le même principe.
8. Réalisez une procédure `placeFourmis` qui prend une grille et un tableau de fourmis en paramètres, et pose les fourmis sur la grille.
9. Réalisez la procédure `initialiseGrille` qui a pour données un tableau de fourmis, l'ensemble des coordonnées des éléments de sucre et l'ensemble des coordonnées des éléments de nid. Cette procédure charge une grille vide, l'initialise en plaçant les fourmis, le sucre et le nid, puis elle linéarise les phéromones de nid et transmet le grille en résultat.
10. Réalisez une procédure `lineariserPheroNid` qui prend une grille en paramètre et linéarise l'intensité en phéromones de nid des cases de la grille (voir la formule dans l'énoncé du projet). On supposera que dans la grille passée en paramètre, l'intensité en phéromones de nid est maximale sur les places contenant un élément du nid et nulle partout ailleurs.
11. Réalisez la méthode `Grille::diminuePheroSucre` qui diminue l'intensité des phéromones de sucre sur la grille pour modéliser leur évaporation.

4 Intégrité de la modélisation

L'état de la simulation est donc stocké dans deux structures de données : le tableau des fourmis et la grille. Ces structures de données sont *redondantes*, c'est-à-dire que *la même information est stockée plusieurs fois*. C'est une manière de programmer dangereuse, car en cas d'erreur de programmation, les deux informations peuvent devenir incohérentes. Voici les redondances :

- **Indice de fourmi :** Chaque fourmi enregistre son indice dans le tableau. On pourrait avoir stocké par erreur une fourmi à un autre indice.
- **Coordonnées des places :** Chaque place enregistre ses coordonnées dans la grille. On pourrait avoir stocké par erreur une place à une autre position.

De plus, chaque fourmi enregistre ses coordonnées dans la grille qui elle-même sait quelle fourmi est dans une case donnée.

- **Cohérence fourmi-grille :** On peut donc avoir une fourmi qui «pense» être dans une case qui est vide.
- **Cohérence grille-fourmi :** Ou bien une case qui contient une fourmi n'existant pas dans le tableau.

13. Écrire une procédure qui teste que tout est bien cohérent. En cas de problème, faire un affichage qui décrit le problème, puis sortir du programme avec une exception `runtime_error` Pour ceci, il faut avoir inclus `exception`.
14. Cette procédure pourra être appelée après chaque mouvement de fourmi pour vérifier la cohérence de la simulation.

Note : tous ces problèmes viennent de l'utilisation des indices et des coordonnées. En général, on évite ces redondances en utilisant des méthodes de programmation plus avancées comme les références ou les pointeurs.

5 Version avancée

Une fois que vous avez fait marcher une version simple avec une seule colonie de fourmis de taille fixe, il faudra modifier ces interfaces (et les types concrets associés) pour rajouter les colonies et faire une classe `Colonie`. Par exemple

- une fourmi devra savoir à quelle colonie elle appartient ;
- la méthode `Place::posePheroNid` devra prendre un paramètre supplémentaire pour dire de quelle colonie il s'agit.