université PARIS-SACLAY

Travaux Dirigés du projet Termites

Cours de programmation modulaire

—Licence MI/IM - Info 3—

Types pour le projet Termites

Nous allons étudier les types découlant d'une analyse du projet de simulation de colonies de termites. Les questions qui suivent ont pour but de vous aider dans la réalisation du projet. Nous allons considérer les **types** suivants :

- `Coord` pour contenir les coordonnées d'un point de la grille ;
- `Direction` pour contenir un point cardinal ;
- `Termite` pour coder un termite ;
- `Grille` pour contenir l'état du monde à un moment donné.
- `Jeu` pour contenir l'état du jeu à un moment donné.

1 Types pour les termites

L'objectif de ce TD est de faire une analyse ascendante. On cherche la liste des fonctionnalités élémentaires que doivent pouvoir remplir la grille et les termites. Puis, en les combinant, on programme des primitives plus complexes pour aboutir finalement à l'algorithme complet.

1.1 Se repérer sur la grille

Dans tout ce sujet d'aide, on suppose que l'on dispose des types `Coord` et `Direction` que vous avez implémentés lors du dernier TP. Dans ce document nous utiliserons les fonctions `Direction aGauche(Direction)` et `Direction aDroite(Direction)` qui renvoient la direction à gauche et à droite d'une direction donnée. Les méthodes utiles du type `Coord` seront :

- le constructeur `Coord(int x, int y);` qui construit la coordonnée (x, y) ;
- les getters `Coord::getX()` et `Coord::getY()` ;
- les opérateurs d'égalité et d'inégalité pour le type `Coord` ;
- `Coord::devant` qui renvoie les coordonnées de devant les coordonnées courantes dans une direction donnée. Cette méthode lève une exception si la case correspondante n'est pas dans la grille.
  Pour utiliser cette fonction dans le code, il faudra rattraper l'exception.

1.2 La grille

Le jeu se passe sur une grille, on va donc naturellement stocker le monde avec un tableau à deux dimensions (sachant qu'il existe des structures optimisées comme les quad-tree pour rassembler les cases vides). Cependant, si la grille comporte un nombre important de cases ou si la structure `Termite` devient grosse, choisir de stocker une structure `Termite` dans

chaque case de la grille va consommer beaucoup de mémoire. Dans chaque case de la grille, il y a plusieurs informations à enregistrer, il faudra donc utiliser un type `Case` qui code les informations nécessaires.

1. Quelle(s) information(s) doit enregistrer une case du terrain ?
2. Doit-on en faire une classe avec des attributs privée ou bien une structure ?
3. Faut-il mettre un ou des constructeurs pour cette classe ? Faut-il mettre des méthodes ?

Le type `Case` ne sert qu'à la gestion de la grille et n'apparaîtra que dans la déclaration du type concret `Grille` et dans son implémentation. Par conséquent, pour manipuler la grille, il n'y a besoin que de spécifier la grille et les coordonnées de l'endroit de la grille à modifier. Autrement dit, l'utilisateur et le testeur de la grille n'ont pas connaissance du type concret `Case`. Ainsi toutes les fonctions d'interrogation et de manipulation d'une case de la grille devront prendre en paramètre la grille et les coordonnées de la case à interroger ou manipuler.

4. Faire la liste des méthodes utile du type `Grille`.

1.3 Les termites

On représentera les termites par une classe `Termite`. Les termites seront rangés dans un vecteur et chaque termite aura un numéro correspondant à sa position dans le vecteur.

5. Pourquoi choisi-t-on de ranger les termites dans un vecteur plutôt que dans un tableau ?
6. Quelles informations doit contenir un termite ?
7. Quel(s) paramètre(s) passer pour construire un termite ?

Dans la classe Termite, il y a deux types de fonctions : celles qui ne concernent que le termite lui-même, et celles qui concernent également la grille. Nous commençons par celle qui ne concerne que le termite :

- Le contructeur `Termite` crée un animal à partir de son identifiant (un entier), et ses coordonnées.
  Note : Selon comment on stocke la population de `Termite` (en particulier si l'on utilise un tableau), on peut avoir besoin de créer aussi un constructeur par défaut (sans paramètres) pour les `Termite`.
- Accès :
  `toString` convertit l'animal en chaîne de caractères pour l'affichage.
  `directionTermite` renvoie la direction du Termite.
  `devant` renvoie les coordonnées de la case devant le termite
- Prédicats :
  `porteBrindille` renvoie si le termite porte une brindille
- Modification :
  `tourneADroite` fait tourner le termite à droite
  `tourneAGauche` fait tourner le termite à gauche
  `tourneAleat` fait tourner le termite dans un sens aléatoire

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

Il y a ensuite une deuxième série de fonctions qui manipulent non seulement le termite, mais aussi certaines cases de la grille :

`laVoieEstLibre` renvoie si la case devant le termite est libre
`brindilleEnFace` renvoie si la case devant le termite contient une brindille
`voisinsLibre` renvoie le nombre de cases libre autour du termite
`avance` avance le termite
`marcheAleatoire` déplace aléatoirement le termite
`chargeBrindille` le termite prend une brindille
`dechargeBrindille` le termite pose une brindille
`vieTermite` algorithme du termite

9. Donnez les spécifications des méthodes ci-dessus en précisant bien lesquelles sont constantes.
10. En utilisant les fonctions des types `Termites` et `Grille`, écrire les fonctions `laVoieEstLibre`, et `brindilleEnFace`.
11. Écrire la méthode `chargeBrindille`.
12. Écrire la méthode `avanceTermite`.
13. Écrire la méthode `marcheAleatoire`.

2 Intégrité de la modélisation

L'état de la simulation est stocké dans plusieurs structures de données : les tableaux des termites (il peut y en avoir plusieurs s'il y a plusieurs colonies) 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 termite :** chaque termite enregistre son indice dans le tableau. On pourrait avoir stocké par erreur un termite à un autre indice.

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

— **Cohérence termite-grille :** on peut donc avoir un termite qui «pense» être dans une case qui est vide ou qui contient en fait une brindille.

— **Cohérence grille-termite :** ou bien une case qui contient un termite qui n'existe pas ou qui «pense» être dans une autre case.

14. écrire une procédure qui teste que tout est bien cohérent. En cas de problème, cette procédure doit lever une exception qui décrit le problème.
15. Cette procédure pourra être appelée après chaque mouvement de l'ensemble des termites 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.