Projet de Programmation

Simulation de termites rassemblant des brindilles

.

1 TD Analyse descendante

On cherche à décomposer le problème en sous problèmes plus simples. Il faut analyser le problème pour le préciser en faisant des choix d'implémentation. On fera ces choix de façon à simplifier la programmation finale, tout en permettant la mise en œuvre d'un type abstrait **termite** raisonnablement étoffé.

1.1 Architecture du Programme Principal

Décomposer le programme principal final en deux phases et préciser l'interaction avec l'utilisateur qui a lieu dans la deuxième.

1.2 Sur l'initialisation

- Que faut-il initialiser ?
- Comment programmer où l'on pose des termites et des brindilles ?
- Doit-on définir statiquement un nombre de termites/brindilles donné ?
- Comment procéder pour remplir aléatoirement le terrain de façon simple ?
- A quoi est à peu près égal le nombre de termites final ?
- Une borne sur le nombre de termites est-elle connue statiquement ?
- Doit-on définir un type abstrait **ensemble de termites** ?
- On recommande d'utiliser le moins possible de variables globales pour éviter les effets de bord. Mais, un certain nombre de constantes réputées uniques sur l'ensemble de la simulation, peuvent se déclarer globales pour éviter de les passer en paramètre. Quelles sont ces constantes ?
- Comment fait un termite pour accéder à la case en face ?
- Où sont stockées les coordonnées d'un termite ?
- Ecrire la procédure qui initialise la grille

1.3 2me phase : itération en interaction avec l'utilisateur

- Doit-on itèrer sur le terrain ? oui ou non, et pourquoi ?
- Comment afficher l'évolution des termites sur leur terrain, de façon interactive ?
- Comment facilement demander au simulateur d'afficher la configuration suivante du terrain après une passe sur les termites ?
- Comment faire pour demander au simulateur d'afficher successivement 10 passes ?
- Et si on veut connaître l'état du terrain au bout de 1000 passes ?
- Ecrire le programme principal, en utilisant les réponses aux questions précédentes.

2 Intégrité de la modélisation

L'état de la simulation est donc stocké dans deux structures de données : le tableau des termites 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 :** L'indice d'un termite est sa position dans le tableau des termites. De plus, chaque termite enregistre son indice dans un champ de sa structure. Enfin, la grille contient, dans les cases où il y a des termites, l'indice du termite correspondant.
- **Coordonnée de termite :** La coordonnée d'un termite est sa position dans la grille. De plus, chaque termite enregistre sa coordonnée dans un champ de sa structure.

Ces redondances peuvent induire des problèmes de cohérence :

- **Cohérence termite-termite :** on peut avoir un termite qui est stocké dans le tableau à un autre endroit que ce qu'indique l'indice présent dans le termite.
- **Cohérence termite-grille :** on peut avoir un termite qui «pense» être dans une case qui est en fait vide ou qui contient une brindille ou qui contient un autre termite.
- **Cohérence grille-termite :** on peut avoir une case qui contient un termite qui n'existe pas, ou qui est déjà stocké dans une autre case.

Écrire une procédure qui teste que tout est bien cohérent. En cas de problème, faire un affichage qui décrit le problème.
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.

3 TD Spécification de l'algorithme de rassemblage de brindilles.

Le sujet du projet annonce le but de l'algorithme : regrouper des brindilles. Il explicite que les termites itèrent quatre étapes. Il reste un travail important à faire pour préciser cet algorithme.

3.1 L'algorithme de la primitive rassemblerBrindille.

C'est toujours une primitive du type abstrait **termite** ; la seule qui soit réellement un peu étoffée. Ce sera à vous de tâtonner pour la trouver tout seul. Les questions qui suivent servent à préciser comment utiliser et mettre à jour les variables d'état du termite qui peuvent sembler difficiles.

- Comment et quand un termite renverse-t-il son sablier, le met à jour, teste si le temps est écoulé ?
- Pourquoi faut-il une variable d'état booléenne *tournerSurPlace* ?
- Ecrire l'algorithme qui permet aux termites de rassembler les brindilles. Le corrigé NE SERA PAS donné. Les TDs contiennent suffisamment d'informations pour que cette tâche soit largement faisable, de plus, cela justifie le programme lui même : Il est difficile de voir l'algorithme directement, par contre il est facile de tester différentes façons de faire, d'observer le résultat, puis de corriger graduellement son programme.