TP C++

LES TERMITES

Templates et interfaces graphiques

Gauthier Quesnel
quesnel@lil.univ-littoral.fr

9 mai 2006

---

1 Règles de développement

1. Votre projet doit suivre les règles suivantes :
   - Le projet doit fonctionner au minimum sur Linux.
   - Cacher les pointeurs des interfaces publiques des classes que vous développerez.
   - Utiliser un maximum de paramètres constants et de fonctions membres constantes.
   - Les « cast » du C++, *dynamic_cast*, *static_cast*, doivent être utilisés à la place du « cast » du langage C lorsque le « cast » n'est pas sûr.
   - Posséder un fichier *Makefile* fonctionnel.
   - Posséder un fichier *README* avec les problèmes ou les points que vous n'avez pas fini.
   - Les bibliothèques *GTKmm* et *GLIBmm* étant jeunes et libres, des points de bonus seront distribués en cas :
     - de découverte d'un bug non connu - (1 pt).
     - d'acceptation d'un patch pour une correction d'un bug par le mainteneur - (2 pt).
2. Il doit aussi suivre le style suivant :
   - 80x50 : le nombre *maximum* de colonnes et de lignes pour le corps d'une fonction.
   - 6 : le nombre *maximum* de paramètres ou de variables locales dans une fonction.
   - *4 caractères* : la taille des indentations avec un maximum de 3 niveaux d'indentations successives dans une fonction.
   - *« Que fait cette fonction ? »* : la réponse à cette question doit se trouver devant chaque déclaration de fonction dans le fichier de déclaration de la classe. On ne répond jamais à la question *« Comment est faite cette fonction ? »*
   - *Global* : pas de variables globales, remplacez-les par des fonctions statiques ou des classes *singletons*.
   - Les fichiers sources ont pour extension `.cc`, les entêtes `.hh` ou `.h`.

2 Sujet

Ce projet informatique s'inspire du comportement des colonies de termites rassemblant des copeaux de bois en tas. Les termites suivent des règles simples sans communication entre elles ni but explicite mais arrivent tout de même à réaliser un but : rassembler un tas de copeaux.

Le but de ce TP est de simuler l'organisation d'une société de termites sur un terrain, découpé en petites cellules, où sont déposées des brindilles de bois. Les termites sont placées aléatoirement ou non à l'initialisation du système. Elles commencent à chercher un copeau de bois en se déplaçant aléatoirement sur le terrain. Dès qu'une termite rencontre un copeau de bois, elle le prend, et continue de se déplacer au hasard. Si elle rencontre un nouveau copeau de bois dans son entourage, elle recherche une place libre à côté et pose son copeau de bois. Dans ce sujet, une seule termite est autorisée par cellule de terrain.

Afin de compliquer un peu le sujet du TP, plusieurs espèces de termites et de bois existent. Chaque espèce de termites aime une ou plusieurs sortes de bois différents.

En général le nombre de tas décroît avec le temps. La raison en est simplement que des piles peuvent disparaître, lorsque des termites ont emporté tous les copeaux qu'elles contenaient, mais il n'y a aucun moyen qu'une nouvelle pile se forme à partir de rien, car les termites posent toujours leurs copeaux à côté d'un autre.

3 Développement du noyau interne

L'objectif de la simulation est de vérifier que cette organisation aboutira à la formation de tas de brindilles de même espèces. Nous représenterons cette organisation en utilisant une grille, pour le terrain, dont les cases contiendront soit des termites, soit des copeaux de bois, sinon rien.

3.1 Les règles

Les termites, les seules entités activent du programme, suivent des règles très simples dont voici les grandes lignes :

- Les termites se déplacent aléatoirement de case en case dans les huit directions possibles.
- Si une termite sans brindille arrive près d'une brindille, elle la prend sur elle.
- Si une termite avec brindille arrive près d'une brindille elle la dépose dans une case vide **autour** du copeau trouvé.
- Deux termites ne peuvent se trouver sur une même case.
- Une termite renonce à un déplacement si celui-ci doit l'amener en dehors des limites du terrain ou sur une case où se trouve déjà une autre termite.

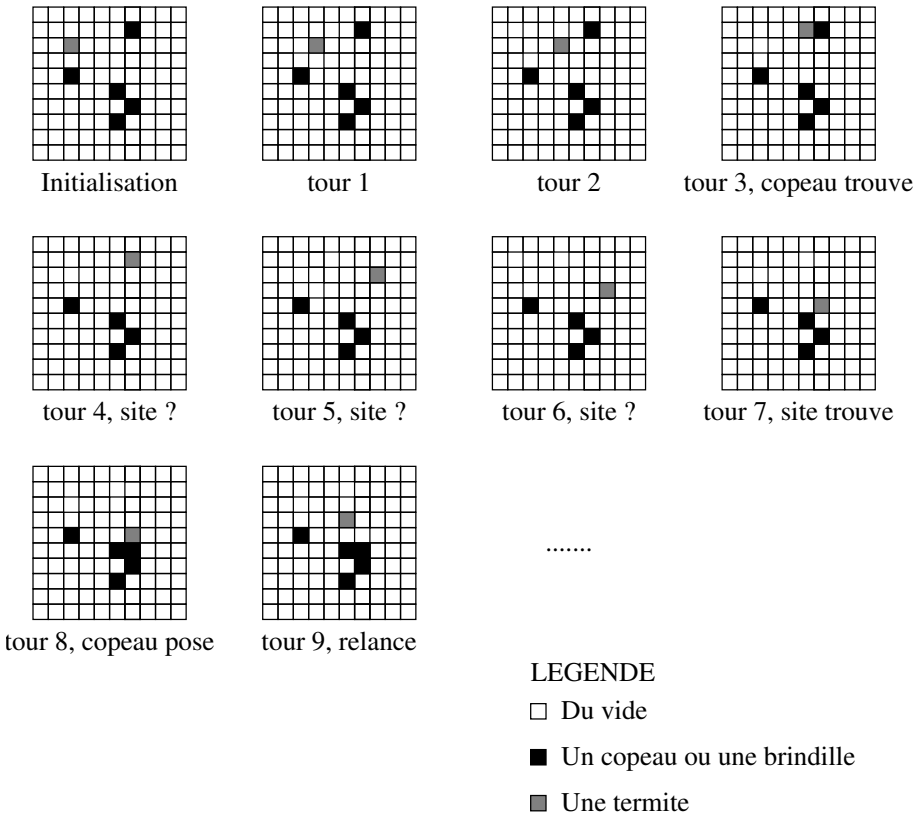

Fig. 1 – Exemple d'une simulation avec une termite, cinq copeaux et une grille de dix cases sur dix pour une durée supérieure à 10 tours. La termite récupère un copeau est le dépose près d'un autre. La ligne de lancement est : *./termites -l 10 -w 10 -t 1 -c 5*

3.2 Description du simulateur

Développer le moteur de simulation du projet. Ce moteur utilise six classes au minimum et doit obligatoirement utiliser les conteneurs de la *STL* (les tableaux du C ne doivent être utilisés que dans de rare cas). Un diagramme UML simplifié, **erroné** et incomplet est fourni avec la figure 2. Une petite description des classes à développer :

- *Espece* décrit les caractéristiques d'une espèce de termite, c'est-à-dire, la liste des bois préférés de cette espèce.
- *EspeceGlobale* contient la liste des espèces de termites. **Une seule instance** de cette classe existe dans le programme.
- *Entite* est la classe virtuelle pure, où les membres ne sont pas définis, utilisée pour gérer le polymorphisme avec les fonctions *est_copeau()* ou *est_termite()* et *print(std::ostream& out)*.
- *Termite* hérite de la classe *Entite* et est associée à une espèce via une référence sur un objet *Espece*. Elle peut posséder une référence sur un objet *Grille* suivant les choix de développement que vous ferez.
- *Copeau* hérite de la classe *Entite* et ne possède pas de fonction spéciale.
- *Grille* implémente un tableau à une dimension avec des pointeurs sur les entités *Termites* ou *Copeaux*, ou *0* lorsque rien n'est présent. La largeur et la longueur de la grille pourront être modifiées en cours de simulation.

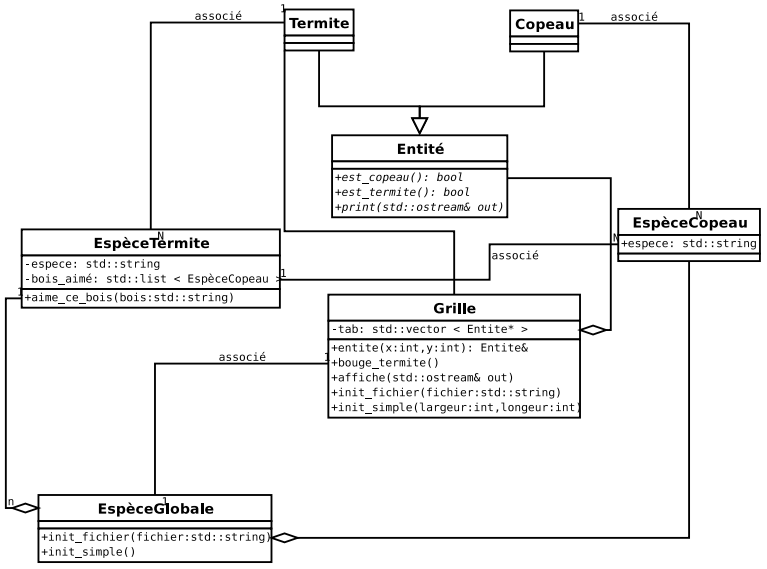

FIG. 2 – Une version simplifiée, éronnée et incomplète du diagramme UML du noyau de simulation.

La boucle générale, du fichier *main.cc*, de votre projet doit ressemble aux lignes suivantes :

```
int main(int argc, char** argv)
{
...                          // analyse des paramètres

Grille* grille;
if (ouverture_par_fichier) { // gestion avec colonie
grille = new Grille(''initialisation.dat'');
} else {                     // gestion sans colonie
grille = new Grille(tailleX, tailleY, nbTermite, nbCopeau);
}

for (int pas = 0; i < finsimulation; ++i) {
grille->bougeTermite();  // deplace toutes les termites.
std::cout << ''Temps : '' << pas << ''\n'';
grille->affiche();
}
```

```
delete grille;
return 0;
}
```

Remarque 1 : Ceci est un exemple d'implémentation possible, si vous pensez à une meilleure solution, décrivez-la dans le fichier *README* de votre projet. Attention, n'oubliez pas que ceci est un noyau de simulation et qu'il faudra l'intégrer plus tard à une interface graphique !

Remarque 2 : Le déplacement aléatoire des termites doit utiliser la classe *Glib::Rand*. Le fonctionnement général des pseudos générateurs de nombres aléatoires génére des nombres en fonction des précédents. Il faut alors initialiser **une seule fois** le générateur. De plus, pour des questions mathématiques, il faut utiliser **un seul générateur** pour une simulation.

3.3 Main

L'utilisation du programme dans une console est le suivant :

```
termites [ options ] fichiers
```

Votre programme principal doit gérer les paramètres décrit ci-dessous. Aidez-vous de la fonction *getopt()* fourni dans le fichier *unistd.h* :

```
int getopt(int argc, char* const argv[], const char* opstring);
```

- `-h` une aide rapide sur les paramètres,
- `-i` informations sur votre projet, noms des binômes. . .,
- `-l x` la largeur de la grille,
- `-h y` la hauteur de la grille,
- `-t z` le nombre de termites,
- `-c a` le nombre de copeaux.
- `-z t` le nombre de pas de simulation à réaliser.

Avec les paramètres précédents, les termites et copeaux n'appartiennent qu'à une seule espèce.

Si le programme est appelé avec un fichier texte en paramètre, celui-ci doit contenir une description complète d'une simulation avec les informations sur la grille, les espèces, etc. Le format, **obligatoire**, est le suivant :

```
################################################################################
# Les lignes commençant par le caractère '#' sont des commentaires.
# Ce format, permet une lecture / écriture très simple via les classes
# std::fstream.
################################################################################
temps: 2000
largeur: 80
hauteur: 50
copeaux: ( boulot chene baobab )
termites: rouge ( boulot ) verte ( boulot chene ) noir ( baobab boulot )
termite: rouge 10 5
termite: rouge 23 34
copeau: boulot 42 23
copeau: baobab 1 3
...
```