Projet d'algorithmique – L2 2008-09

25 novembre 2008

Simulation de fourmilières

Résumé

Les fourmis Rouges et les fourmis Noires s'affrontent dans le plan. Le but du projet est de simuler cet affrontement à l'aide de la bibliothèque graphique `Mlvlib`. Le projet est à développer par niveaux de difficultés successifs.

Deux groupes de fourmis (Rouges et Noires) sont représentés à l'aide de la bibliothèque graphique. Il y a trois types d'éléments dans chaque groupe :

– les fourmilières qui produisent des reines ou des ouvrières et ne se déplacent pas ;
– les reines qui se déplacent, attaquent ou se transforment en fourmilières
– les ouvrières qui se déplacent et attaquent.

Dans la suite on appellera *agent* les fourmis mobiles.

1. **Mise en place**
   Chaque couleur agit alternativement. A chaque tour, on traite tous les élément d'un groupe puis tous ceux de l'autre groupe.
   Les productions des fourmilières sont traitées, puis le déplacement, d'au plus une case, de chaque agent.
   Initialement, il y a une foumilière, une reine et une ouvrière pour chaque couleur, placées dans des coins opposés.
2. **Production**
   La production d'une fourmilière est choisie et ne peut être changée avant sa réalisation complète. On peut choisir de produire une reine (6 tours) ou une ouvrière (3 tours). Après réalisation d'une production, la fourmilière attend un nouvel ordre de production.
   Chaque fourmilière est reliée à tous les agents qu'elle a produit. Si la fourmilière est détruite tous les agents qu'elle a produit disparaissent immédiatement.

Si une reine se transforme en fourmilière, elle n'est plus attachée à la fourmilière qui l'a produite.

3. **Déplacement**
   Les fourmilières ne se déplacent pas et deux fourmilières ne peuvent pas se trouver sur une même case.
   Chaque agent a une destination.
   Si sa position n'est pas égale à sa destination, il se rapproche d'une case de sa destination.
   Si sa position est égale à sa destination, il attend un nouvel ordre.
   Cet ordre peut être :
   - la destruction de l'agent.
   - une nouvelle destination (en cliquant sur le plateau).
   - immobilité.
     Une ouvrière immobile le reste jusqu'à la fin de la partie ou jusqu'à être détruite.
     Une reine immobile se transforme en fourmilière si elle n'est pas déja sur une fourmilière. Si une reine est sur la même case qu'une fourmilière, elle doit obligatoirement se déplacer.

   Pour repérer aisément une fourmi en attente d'ordre on l'affiche avec une couleur particulière.

4. **Attaque**
   Si un agent veut se placer sur une case occupée par des éléments de couleur différente, un combat a lieu. Il est résolu par tirage aléatoire (fonction `random()`). A l'issu d'un combat, l'un des éléments est détruit. Lorsque plusieurs éléments d'une couleur peuvent être sur une même case, le processus se répète tant qu'une des couleurs n'est pas éliminée. On ordonnera judicieusement les éléments placés sur une même case pour que les ouvrières interviennent en premier, puis les reines. Si une fourmilière est présente, elle n'intervient que dans le dernier combat.
5. **Le plateau de jeu**
   Le plan est considéré comme un quadriage de cases carrées, chaque case possède 8 voisins.
   Les bords du plateau ne sont pas franchissables.
6. **Fin de partie**
   Une partie est terminée lorsque l'une des couleurs ne posséde plus de fourmilière. On peut également choisir d'arrêter le jeu.
7. **Sauvegarde et chargement**
   Au début d'un tour, on peut choisir de sauvegarder la partie dans un

fichier. La couleur devant agir ainsi que tous les éléments devront être mémorisés.
On devra de même pouvoir choisir de récupérer un partie mémorisée dans un fichier.
L'utilisateur entrera le nom du fichier de sauvegarde.

8. **Résultats choisis**
Un mode de contrôle permettant de choisir les résultats des combats devra être disponible.

Implantation et niveaux de jeu

1. **Interface graphique**
L'interface graphique doit respecter les impératifs suivants :
   - La fenêtre graphique est séparée en deux parties.
   - La partie supérieure représente le plateau de jeu. Les éléments occupent des carrés de 10×10 pixels. On pourra représenter une fourmilière par un carré vide et les agents par des carrés pleins de couleurs différentes.
   Le quadrillage du plateau de jeu est apparent.
   - La partie inférieure forme une barre de menu contenant des boutons permettant la gestion des différents ordres à transmettre aux ouvrières, reines et fourmilières ; elle comprend également une partie dans laquelle s'affichera les coordonnées de la fourmi courante, l'action qu'elle est en train d'effectuer. Enfin il doit y avoir une troisième zone dans laquelle se trouve un bouton *sauve*, un bouton *charge*, un bouton *quitter*, un bouton *fin de tour* ainsi que l'indication du numéro du tour et la couleur du joueur en cours.
   Le jeu se joue au tour par tour. A la fin de son tour le joueur doit cliquer sur le bouton *fin de tour* afin de rendre la main à son adversaire.

2. **Structures et représentations des données**
Les fourmilières, reines et ouvrières sont représentées par la même structure `Fourmi`.
En fonction des niveaux réalisés, certains champs peuvent ne pas être utilisés.
On utilisera les constantes symboliques et les types suivants :

```
/* dimension du monde*/
#define X 60
#define Y 40
```

```
/*les deux groupes de fourmi*/
#define ROUGE 'R'
#define NOIR  'N'
/*les genres de fourmi*/
#define OUVRIERE 'o'
#define REINE  'r'
#define FOURMILIERE 'f'
/*les temps de production de fourmi*/
#define TOUVRIERE 3
#define TREINE 6

typedef struct four{
char couleur;/* ROUGE ou NOIRE*/
char genre;/* FOURMILIERE,REINE OUVRIERE*/
int posx,posy;/*position actuelle*/
int destx,desty;/*destination (negatif si immobile)*/
int produit,temps;/*production d'une fourmiliere*/
  struct four *fsuiv,*fprec;/*liste des fourmis liee a une fourmilier*/
  struct four *vsuiv,*vprec;/*liste des voisins*/
}Fourmi,*FListe;

typedef struct
{Fourmi * fourmiliere;/* s'il y a une  fourmiliere sur le case*/
  FListe  habitant;/* les autres occupants*/
}Case;
typedef struct
{Case plateau[X][Y];
  FListe Rouge,Noir;
}Monde;
```

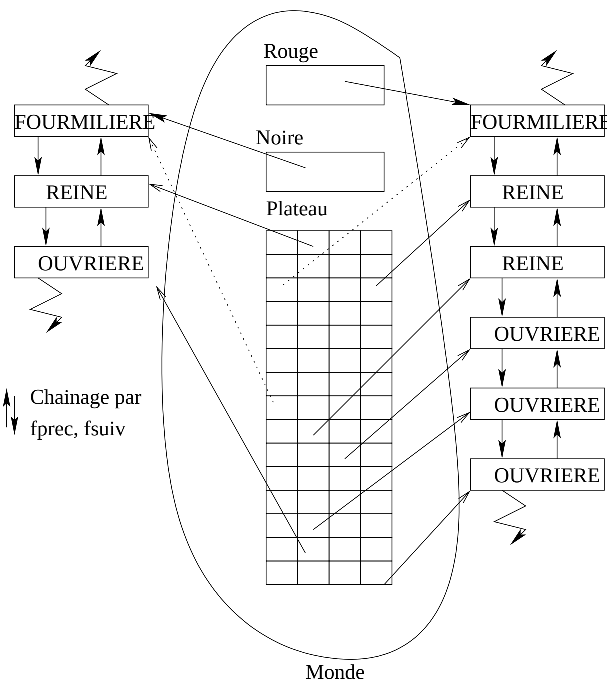
3. **Format de sauvegarde**
   Le format des fichiers de sauvegarde est fixé.
   La première ligne contient la lettre précisant la couleur du groupe devant jouer. Sur chacune des lignes suivantes on trouve les caractéristiques d'un seul élément (sauf bien entendu les valeurs des pointeurs). Une ligne est donc constituée de `2 char` suivis de `6 int` séparés par un seul espace.
   Un agent est relié à la première fourmilière qui le précède.
   Ce formatage doit permettre d'échanger des fichiers de tests.

4. **Les niveaux**

(a) Niveau 1

Dans ce niveau on demande uniquement de gérer les déplacements des agents et la production d'une fourmilière. On doit pouvoir choisir la production, déplacer, immobiliser ou détruire un élément. Deux éléments ne peuvent pas être placés sur la même case du plateau. Un agent produit par une fourmilière est placé sur une case libre voisine de celle-ci.

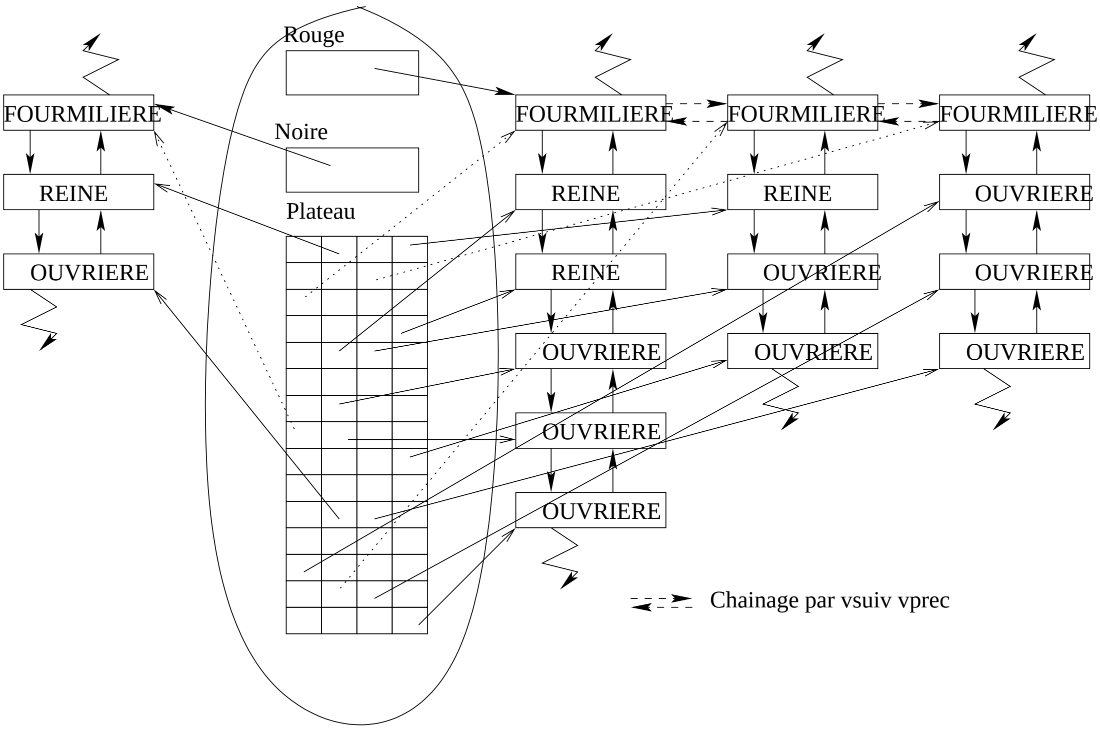
Afin de pouvoir aisément détruire un élément, les éléments d'une couleur sont reliés par une liste doublement chainée. Une case du plateau de jeu pointe sur l'élément situé sur cette case.



Niveau 1

(b) Niveau 2

En plus du Niveau 1 on ajoute la possibilité pour une reine de se transformer en une nouvelle fourmilière. Une reine qui se transforme en fourmilière se retire de la liste des agents. La fourmilière est ajoutée à la liste doublement chaînée des fourmilières de sa couleur grâce aux pointeurs `*vsuiv,*vprec`.



Niveau2

,

(c) Niveau 3
En plus du Niveau 2 on ajoute la gestion des combats entre 2 éléments de couleurs différentes.

(d) Niveau 4
On autorise désormais la possibilité que plusieurs agents soit placés sur la même case. Pour cela la liste doublement chaînée des voisins d'un agent relie tous les agents situés sur la même case. L'utilisation d'une liste circulaire permet de respecter facilement l'ordre d'intervention dans les combats.

Améliorations

De nombreuses améliorations sont possibles. Elles ne seront prises en compte que si tous les niveaux précédents sont totalement réalisés.
On peut par exemple :

- Ajouter un déplacement en mode patrouille : aller-retour entre un point et la fourmilière.
- Ajouter un système de points de vie, fourmilières, reines ou ouvrières ayant un nombre de points de vie différent.
- Relier la survie et la rapidité de production d'une fourmilière au nombre d'ouvrières qui lui sont attachées.
- Ajouter une catégorie de fourmis guerrières et des coefficients de force lors du tirage du résultat.
- ...

Ce qu'il faut faire

Le projet est à effectuer en binôme du même TP, sauf accord des 2 chargés de TP. Une soutenance aura lieu pendant le dernier TP.
Les programmes C **compilables à l'université** devront être clairs et judicieusement commentés.
Un rapport décrivant d'une part le fonctionnement, d'autre part les méthodes utilisées et les choix d'implantation sera rendu. La complexité des principales fonctions devra être expliquée. Le but est de permettre à un utilisateur quelconque d'utiliser le programme, et à un programmeur averti de faire évoluer facilement votre code.
Ne commencer pas à développer un niveau avant que le niveau précédent soit totalement fonctionnel.