

Algorithmique et Programmation

TP6 *Fourmis*

FICHIERS À RENDRE :

```
-- main.cpp
-- fourmi.cpp et fourmi.h
-- monde.cpp et monde.h
-- rapport_TP6.pdf
```

0 Présentation du sujet

L'objectif de ce TP est de chercher à reproduire *informatiquement* le comportement de fourmis à la recherche de nourriture.

On peut s'apercevoir, en étudiant le comportement des insectes sociaux (fourmis, termites, etc...) que ceux-ci, bien que doués d'une intelligence très limitée, arrivent à résoudre des problèmes très complexes : construction de structures, recherche optimale de nourriture, etc... De nombreuses expériences montrent la grande efficacité d'une fourmilière dans la recherche de nourriture : par exemple, en laboratoire, si l'on dispose sur le territoire d'une colonie deux sources d'eau sucrée de même dimension, mais de concentration en sucre différentes, on constate, au début, que les deux sources sont exploitées de manière plus ou moins égale, mais que, assez rapidement, l'énorme majorité des fourmis se retrouvent sur la piste menant à la source la plus riche. De la même manière, on peut montrer que si les fourmis peuvent emprunter deux chemins pour aller à une source de nourriture, le chemin le plus court sera celui qui sera, rapidement, le plus emprunté.

Les sociétés d'insectes possèdent un mode de fonctionnement bien différent du modèle humain : un modèle décentralisé, fondé sur la coopération d'unités autonomes au comportement relativement simple et probabiliste, qui sont distribuées dans l'environnement, et qui ne possèdent que des informations locales (elles n'ont aucune représentation ou connaissance explicite de la structure globale où elles évoluent, elles ne possèdent aucun plan, aucune directive globale). Dans ces sociétés, le projet global n'est pas programmé explicitement chez les individus, mais émerge de l'enchaînement d'un grand nombre d'interactions élémentaires entre individus : **il y a intelligence collective à partir de nombreuses simplicités individuelles.**

Le TP a donc pour but de simuler le comportement individuel des fourmis à la recherche de nourriture dans un environnement, afin d'observer le comportement global de la fourmilière. Nous allons construire, petit à petit, un modèle simplifié de ces comportements de fourmis.

1 Question 1 : initialisation et affichage du monde

L'*environnement* (dans un plan) dans lequel évoluent les fourmis est représenté par une structure `t_monde` contenant la taille du monde, la position de la fourmilière ainsi qu'une matrice d'entiers dont les éléments peuvent prendre les valeurs suivantes :

- `OBSTACLE` (constante définie à -1) : pour signifier qu'un obstacle est présent.
- `VIDE` (constante définie à 0) : pour signifier que la case est vide : ni obstacle ni nourriture.
- une valeur de 1 à `NOURRITURE_MAX` (définie à 50) : indique la quantité de nourriture présente.

Les fourmis peuvent évoluer librement dans cet environnement, mais doivent partir de leur fourmilière pour aller chercher de la nourriture. Les fourmis peuvent se diriger dans les 8 directions (haut, bas, gauche, droite et les diagonales), mais ne peuvent marcher sur les obstacles. Une fois la nourriture trouvée, les fourmis doivent prendre un élément de nourriture et revenir à la fourmilière pour le déposer.

Les fourmis peuvent donc se trouver dans deux états : en recherche de nourriture ou en chemin vers la fourmilière. Nous essaierons plusieurs stratégies pour chacun de ces états.

Afin de visualiser le comportement des fourmis, nous proposons des fonctions permettant d'afficher l'environnement dans lequel évoluent les fourmis, les obstacles, la nourriture, ...

Récupérer sur le serveur pédagogique les fichiers suivants :

- `constantes.h` : fichier fournissant des constantes et types utilisés dans le projet
- `affichage.cpp` et `affichage.h` : fonctions d'affichage et leur fichier en-tête
 - `InitAffichage` : fonction permettant d'initialiser l'affichage de la fenêtre représentant le monde dans lequel vont évoluer les fourmis. Elle doit être appelée une seule fois, avant toutes les autres. Elle renvoie la valeur 1 si l'initialisation s'est bien déroulée.
 - `MiseAJourFourmi` : fonction permettant de mettre à jour la position et le mode d'une fourmi (en recherche de nourriture ou de retour à la fourmilière).
 - `MiseAJourEnvironnement` : fonction permettant de mettre à jour l'environnement, c'est à dire les obstacles et la nourriture¹.
 - `MiseAJourPheromone` : fonction permettant de mettre à jour la quantité de phéromone dans l'environnement.
 - `Affichage` : fonction nécessaire après chaque affichage de toutes les fourmis et de l'environnement. Cela permet de faire une pause (avant le prochain cycle d'affichage des fourmis et de l'environnement), et de gérer la fenêtre graphique (réaffichage lorsque la fenêtre est cachée, gestion des touches, etc...). Elle renvoie la valeur 0 si l'utilisateur a pressé la touche ESC et 1 sinon.
- `proba.cpp` `proba.h` : fonction de tirage au sort et son fichier en-tête
 - `nalea` : fonction permettant de tirer au sort un entier (entre 0 et l'entier donné en argument moins 1)
 - `nalea_pondere` : fonction permettant de tirer au sort un entier (entre 0 et 7) pondéré par un vecteur.
- `monde1.dat`, `monde2.dat`, `monde3.dat` : fichiers décrivant un monde :
 - les deux premières valeurs correspondent à la largeur et à la hauteur réelles du monde.
 - les deux valeurs suivantes sont la position (abscisse et ordonnée) de la fourmilière.
 - les valeurs suivantes représentent chaque case (ligne par ligne) et peuvent prendre les valeurs : OBSTACLE, VIDE, ou bien une valeur comprise entre VIDE et NOURRITURE_MAX.

Il est *indispensable* de lire attentivement les fichiers .h qui constituent la référence pour l'utilisation des fonctions et les fichiers .dat pour comprendre le format des données.

1. Il est possible d'accélérer ou diminuer la vitesse de simulation à l'aide des touches "+" et "-" du clavier.

L'algorithme du programme principal prendra la forme suivante :

```

retour <- InitAffichage(largeur,hauteur)
...
TantQue retour=1 Faire
    ...
    Pour <chaque fourmi> Faire
        ...
        MiseAJourFourmi(...)
    FinPour
    ...
    MiseAJourEnvironnement(...)
    retour <- Affichage()
FinTantQue

```

1.1 Travail demandé

1. Écrire l'algorithme et le code d'une fonction (`LireEnvironnement`) dans un fichier `monde.cpp` permettant de constituer un environnement à partir de la lecture d'un fichier monde.
2. à partir de l'algorithme ci-dessus à compléter, écrire la fonction principale dans un fichier `main.cpp` pour afficher un monde. Vous ajouterez la librairie `X11` à votre projet.
3. Définir dans le rapport et dans un fichier `fourmi.h` une structure (`t_fourmi`) caractérisant simplement une fourmi (position, ...).
4. Écrire l'algorithme et le code d'une fonction simple (`DeplaceFourmi`), dans un fichier (`fourmi.cpp`), permettant de déplacer aléatoirement une fourmi, en considérant qu'une fourmi ne peut se déplacer que sur une case adjacente à sa position courante (Utiliser pour cela la fonction `nalea`).
5. Placer une fourmi, la faire déplacer et l'afficher à chaque déplacement.
6. Placer plusieurs fourmis dans le monde.
7. Expliquer le comportement de votre programme lorsque la fourmi sort de la zone réservée par `t_monde` ou marche sur un obstacle.

2 Question 2 : déplacements linéaires avec évitement d'obstacle

Dans cette question, nous réalisons un déplacement de la fourmi plus réaliste pour l'empêcher de marcher sur les obstacles et de sortir du monde. Pour cela, nous examinons tour à tour les 8 déplacements possibles, et nous fixons une probabilité pour chacun d'eux. Il suffit ensuite de faire un tirage au sort, entre ces 8 déplacements, en respectant les probabilités.

Nous allons examiner les déplacements, en considérant une **rotation** par rapport à la direction précédente (et comprise arbitrairement entre -4 et $+4$, de la façon suivante) :

3	2	1
± 4	\rightarrow	0
-3	-2	-1

On utilise la rotation 0 pour garder la même direction, ± 4 pour faire marche arrière, ± 1 pour aller en diagonale en avant. Pour éviter des mouvements de fourmi trop erratiques, la nouvelle direction à suivre est donc choisie en fonction de l'ancienne. Pour cela, on donne une probabilité plus forte au déplacement se situant globalement dans la même direction que celle que la fourmi avait précédemment. Par exemple, nous utilisons les **pondérations** suivantes (en fonction de la rotation à effectuer) pour les changements de direction :

rotation	0	± 1	± 2	± 3	± 4
pondération	12	2	1	1	0

Ce tableau indique que la fourmi a 12 chances sur 20 ($12+2\times 2+2\times 1+2\times 1$) de conserver la même direction que dans le mouvement précédent, et qu'elle ne pourra pas retourner en arrière.

Pour le tirage au sort, la fonction `nalea_pondere` renvoie un nombre tiré au sort (ramené par une opération modulo entre 0 et 7), avec un tirage non équiprobable mais pondéré par un vecteur donné.

La **direction** est donnée par un entier compris entre 0 et 7, correspondant à la rotation modulo 8, qui donne la valeur de l'indice des vecteurs de déplacement (`tdx` et `tdy`). Par exemple, dans l'algorithme de la fonction `DeplaceFourmi` ci-dessous, la rotation -1 est représentée par la direction 7, telle que `tdx[7]=+1` et `tdy[7]=-1`.

Pour chacun des déplacements, nous déterminons donc la pondération finale, en utilisant les critères suivants :

- on tire d'abord au sort le déplacement en favorisant la direction “*tout droit*”, grâce au vecteur de pondération dont les valeurs appartiennent à l'ensemble $\{12, 2, 1, 0\}$.
- si le déplacement n'est pas possible (obstacles ou sortie du monde), la pondération correspondante est annulée.

Le déplacement d'une fourmi se fait donc en 3 étapes :

- i) déterminer la direction à suivre (examen de toutes les directions pour fixer la probabilité pour chacune d'entre elles, puis tirage au sort).
- ii) se déplacer dans cette direction (si c'est possible).
- iii) réaliser éventuellement certaines fonctions après déplacement (prendre la nourriture, etc...).

2.1 Travail demandé

1. Écrire l'algorithme et le code d'une fonction (`PositionPossible`) permettant de savoir si une position donnée peut être occupée par une fourmi. On suppose qu'une position peut être occupée par plusieurs fourmis.
2. Ré-écrire l'algorithme et le code de la fonction de déplacement d'une fourmi (`DeplaceFourmi`) en utilisant les pondérations pour déterminer le déplacement. On peut au besoin compléter la structure de données qui représente une fourmi, afin de conserver la direction (entre 0 et 7).
3. Tester le comportement et tester aussi en changeant exagérément les probabilités (tourner à gauche, à droite, etc...) et commenter les tests dans votre rapport.

On pourra utiliser les algorithmes donnés ci-après :

```
problème:
  ramener un entier de [-7;7] dans l'intervalle [0;7]
principe:
  utilisation de l'opérateur modulo et ré-ajustement, car (x modulo 8) est dans [-7;7]
spécification:
  fonction: {y} <- modulo8(x)
  paramètre: entier x
  résultat: entier y // compris entre 0 et 7
algorithme:
variables
  entier y
début
  y = (x + 8) modulo 8      // car x est compris entre -7 et +7
  retourner y              // et donc x+8 entre 1 et 15
fin
```

```

problème:
    déplacement probabilisé d'une fourmi
principe:
    calcul de la pondération pour chacun des 8 déplacements
    puis déplacement dans la direction donnée par le tirage au sort pondéré
spécification:
    fonction {fourmi} <- DeplaceFourmi(fourmi, environnement)
    paramètre:
        t_fourmi fourmi          // fourmi à déplacer
        t_monde  environnement   // l'environnement
    résultat:
        t_fourmi fourmi          // la fourmi déplacée
algorithme:
    constantes
        vecteur de 8 entiers tdx = {+1,+1,0,-1,-1,-1, 0,+1} // déplacement en x
        vecteur de 8 entiers tdy = {0,+1,+1,+1, 0,-1,-1,-1} // déplacement en y
        vecteur de 8 entiers p_toutdroit={12,2,1,1,0,1,1,2} // coefficients de pondération

    variables
        entier dir,i
        vecteur de 8 entiers pondération // pondération pour le tirage
    début
    // 1) choix d'une direction, en privilégiant la direction "tout droit"
    pour i de 0 à 7 faire
        dir = modulo8(i - fourmi.direction)
        pondération[i] <- p_toutdroit[dir]
    finpour
    // pondération nulle quand le déplacement n'est pas possible
    pour i de 0 à 7 faire
        si non(PositionPossible(fourmi.x+tdx[i], fourmi.y+tdy[i], environnement)) alors
            pondération[i] <- 0
        finsi
    finpour
    // détermination de la direction par tirage au sort suivant la pondération
    fourmi.direction <- nalea_pondere(pondération)
    // 2) déplacement suivant cette direction
    fourmi.x <- fourmi.x + tdx[fourmi.direction];
    fourmi.y <- fourmi.y + tdy[fourmi.direction];
    retourner fourmi
fin

```

3 Question 3 : recherche de nourriture et retour à la fourmilière

Maintenant que nos fourmis sont capables de se déplacer correctement, d'éviter les obstacles, il leur faut aller chercher de la nourriture pour la ramener à la fourmilière.

Les fourmis vont pouvoir être dans deux états, correspondant aux deux modes différents :

- en mode de recherche de nourriture
- en mode de retour à la fourmilière

On complète le calcul de la pondération comme suit, pour prendre cela en compte.

- Pour la fourmi en recherche de nourriture :

i) on choisit une direction :

- on favorise le déplacement *tout droit*,
- si le déplacement amène sur de la nourriture (et qu'il ne s'agit pas de la fourmilière), la pondération est très élevée (par exemple 100000),
- si le déplacement n'est pas possible (obstacles ou sortie du monde), la pondération est nulle,

- ii) on se déplace dans cette direction,
 - iii) si on est arrivé sur de la nourriture, on en prélève (quantité prélevée= 1), et on passe en mode "retour à la fourmilière"
- Pour la fourmi qui cherche à retourner à la fourmilière, le choix de la direction est un peu différent, car la fourmi peut se souvenir de la direction de la fourmilière. On donnera une probabilité plus forte au déplacement se situant dans la direction de la fourmilière.
- Nous utiliserons les pondérations suivantes :

rotation	0	± 1	± 2	± 3	± 4
pondération	200	32	8	2	1

Le déplacement de la fourmi qui retourne à la fourmilière est alors :

- i) on choisit une direction :
 - on favorise le déplacement *en direction de la fourmilière*, avec la pondération $\in \{200, 32, 8, 2, 1\}$,
 - si le déplacement amène sur la fourmilière, la pondération est très élevée (par exemple 100000),
 - si le déplacement n'est pas possible (obstacles ou sortie du monde), la pondération est nulle.
- ii) on se déplace dans cette direction,
- iii) si on est arrivé sur la fourmilière, on dépose la nourriture (on incrémente le tableau environnement à cet endroit), on fait demi-tour (pour repartir dans la bonne direction) et on repasse en mode "recherche de nourriture".

3.1 Travail demandé

1. Compléter l'algorithme et le code de la fonction (`DirFourmiliere` donnée ci-dessous) qui calcule dans quelle direction se trouve la fourmilière (entre 0 et 7) à partir de la position d'une fourmi et de la position de la fourmilière.
2. Modifier l'algorithme et le code de la fonction de déplacement d'une fourmi (`DeplaceFourmi`) pour que les fourmis aillent chercher la nourriture, et la ramènent à la fourmilière.
3. Commenter la performance, c'est-à-dire le nombre d'unités de nourriture ramenées à la fourmilière par nombre d'itérations. Comment pourrait-on l'améliorer ?

```

//fonction: DirFourmiliere
//argument: entiers x,y : position de la fourmi
//           entiers Fx,Fy : position de la fourmilière
//problème: donner la direction de la fourmilière (entre 0 et 7)
//principe: calculer le vecteur direction normé
//           et regarder à quelle direction il se réfère
int DirFourmiliere(int x, int y,int Fx, int Fy)
{
    int resultat;
    int dx = Fx - x;
    int dy = Fy - y;
    float norme = sqrt(dx*dx + dy*dy);
    dx = (int)round(dx/norme); // round retourne un réel qu'il faut transformer en entier
    dy = (int)round(dy/norme);
    for (int i = 0; i <= 7; i++)
    { // vérifier pour les 8 directions possibles celle qui oriente au mieux vers la fourmilière
        if(dx == tdx[i] && dy == tdy[i]) { ... ;}
    }
    return resultat;
}

```

4 Question 4 : les phéromones !!

Afin de nous rapprocher du réel comportement des fourmis pour leur quête de nourriture, nous allons utiliser les même armes qu'elles. Les fourmis qui découvrent de la nourriture ont la particularité de déposer sur leur chemin de retour une hormone (la phéromone) qui permet d'informer les autres fourmis. La phéromone s'évapore naturellement proportionnellement au temps.

On utilise dans cette partie un tableau de type `t_monde` qui représente la concentration (en pourcentage : entier de 0 à 100) de phéromone (pour chaque case du monde). Pour afficher la phéromone, on utilisera la fonction `MiseAJourPheromones(t_monde)` (juste après l'affichage de l'environnement).

Une fourmi de "retour à la fourmilière" dépose de la phéromone (10 unités) sur sa case courante ainsi que 5 unités sur les 8 cases voisines.

Une fourmi en "recherche de nourriture" aura une plus grande probabilité d'aller dans la direction où il y a le plus de phéromone. Le problème dans ce cas est qu'il faut aussi privilégier la direction la plus éloignée de la fourmilière (sinon on pourrait revenir à la fourmilière!!). Soit `directionOpposée` l'écart entre la direction de la fourmilière et la direction de la fourmi, le tout ramené entre -4 et 3 (0 indique que la fourmi se dirige vers la fourmilière et -4 qu'elle se dirige dans la direction opposée). On ajoute alors la pondération : `tauxPheromone*(abs(directionOpposée))` aux pondérations précédemment calculées.

La pondération maximum est alors de : 4 (direction opposée à la fourmilière) * 100 (saturation de phéromone) = 400.

4.1 Travail demandé

1. Écrire une fonction (`EvaporationPheromones`) permettant de modéliser l'"évaporation" de la phéromone. Cette fonction doit être appelée tous les 70 tours, et évaporer 1 unité de phéromone.
2. Modifier le comportement des fourmis en mode "retour à la fourmilière" afin de déposer de la phéromone. Attention, la valeur de la phéromone ne peut dépasser 100 (phénomène de saturation).
3. Modifier le comportement des fourmis en mode "recherche de nourriture" afin de privilégier une direction comportant de la phéromone, opposée à la direction de la fourmilière.
4. Comparer l'efficacité avec la question précédente.