Projet : Les Fourmis

Les fourmis possèdent un comportement de termites, c'est à dire qu'elles se rassemblent pour former des colonies organisées. Le but de notre projet est donc d'étudier ce comportement. Plus précisément, nous allons poser une grille qui délimite le périmètre autour du nid dans laquelle la fourmi peut se déplacer. Dans cette grille nous allons placer de la nourriture et nous verrons que la fourmi ne se déplace pas tout fait aléatoirement, mais en fonction du niveau de phéromones laissées dans chaque case.

Dans une première version, nous avons considéré le déplacement d'une seule fourmi sans prendre en compte les phéromones qu'elle laisse derrière elle. Tandis que, dans la version 2 lorsque la fourmi passe par une case, elle laissera une phéromone dedans et nous considérerons plusieurs fourmis.

Tout d'abord, nous étudierons la bibliothèque sabrina.h et la fonction d'initialisation, puis les différents algorithmes du code permettant d'étudier le comportement de la fourmi, et les améliorations réalisés dans v2am.

I. Quelques points de repères

A. Les différents types

Avant même d'étudier ces fonctions, expliquons les types qu'on crée :

➔ Le type position est composé de deux champs, deux entiers x et y.

➔ Le type fourmi est composé de quatre champs :

- pos, de type position, qui contient la position courante de la fourmi
- retour, de type entier, qui vaut 0 lorsque la fourmi cherche de la nourriture et 1 sinon
- prec, de type entier, qui indique le mouvement précédent de la fourmi (↑,↓,←,→)↔(0,1,2,3)

➔ Le type **environment** est composé de quatre champs:

- un tableau **Tabfourmis[]** de Nf structures de type fourmi
- un tableau **nourriture[]** de Nnour structures de type position
- un entier **tour** qui correspond au nombre de déplacement que les fourmis ont effectués depuis la sortie du nid
- un tableau d'entiers à 2 dimensions **phéromones[][]** de taille N*N qui contient le nombre de phéromones dans une case[x][y] de la grille

B.Bibliothèque sabrina.h

Pour la clarté du code, j'ai pris la décision de créer une bibliothèque à part répertoriant toutes les fonctions utiles au code. On crée deux fichiers sources sabrina.h et sabrina.c dans le répertoire Projet.

Par souci de concision, nous allons nous allons détailler que les fonctions principales et laisserons soin au lecteur de se renseigner sur le fichier source C où tous les commentaires nécessaires aideront à la compréhension.

1. *La fonction int Rand4Sauf1(int except) renvoie un nombre aléatoire différent de except compris entre 0 et 3.*

Principe :

➔ Il suffit de tirer trois entiers aléatoirement car on doit tirer quatre entiers différents d'un autre.

➔ On tire aléatoirement 0, 1 ou 2 grâce à la fonction rand() de la bibliothèque time.h.

➔ rand()%3 renvoie soit 0,1 ou 2 de manière aléatoire et équiprobable.

➔ On pose donc r=rand()%3.

- Si r>=except alors on renvoie r+1. En effet,r+1 est compris entre 1 et 3 et r+1>except donc r+1!=except.
- Sinon r<except alors r!=except.

2. *La fonction int Rand4Sauf2(int except1, int except2) renvoie un nombre aléatoire différent de except1, except2 compris entre 0 et 3. Nous nous inspirons de ce qui a été fait pour la fonction Rand4Sauf1().*

Principe :

On calcule le minimum et le maximum du couple (except1, except2) grâce aux fonctions fmin() et fmax() de la bibliothèque math.h.

Il suffit de tirer deux entiers aléatoirement car on doit tirer quatre entiers différents de deux autres distincts.

On tire aléatoirement 1 ou 2 grâce à la fonction rand() de la bibliothèque time.h.

rand()%2 renvoie soit 0, soit 1 de manière aléatoire et équiprobable, donc rand()%2+1 renvoie soit 1, soit 2 de la même manière.

On pose donc r=rand()%2+1.

On procède ensuite aux différents tests sur r:

➔ Si r>=max alors on renvoie r+1.

En effet, r>=max imlplique r>=min et donc en renvoyant r+1 (on renvoie soit 2, soit 3) strictement supérieur à max, on renvoie un entier différent de min et max.

➔ Si r<=min alors on renvoie r-1.

En effet, r<=min imlplique r<=max et donc en renvoyant r-1 (on renvoie soit 0, soit 1) strictement inférieur à max, on renvoie un entier différent de min et max.

➔ sinon, r>min et r<max, on renvoie r.

En effet, ces conditions impliquent déjà r!=min et r!=max.

3. *La fonction deplacement_r() prend en paramètres deux struct fourmi rf et f et un entier r, et initialise rf en fonction de f avec l'équivalence mouvement/entier suivante:*
(↑,↓,←,→)↔(0,1,2,3)

C. Fonction d'initialisation

La fonction init_v2() prend en paramètre une struct position nid et renvoie une struct environment où les fourmis sont placées dans le nid, qui place aléatoirement les Nnour nourritures et qui initialise les autres champs à 0.

Principe :

On crée une struct environment ne et on initialise ses champs tour et phéromones[][] à 0.

On place les Nnour nourritures aléatoirement.

On répète ce procédé Nnour fois tant qu'on n'a pas placé la i-ème nourriture

On tire une position au hasard qui ne se trouve pas sur le bord si cette position n'est pas le nid, et ne contient pas encore de nourriture on place la i-ème nourriture dans cette position pour placer la première nourriture, on vérifie seulement que la position n'est pas le nid.

On retourne ne.

II. II. Version 2

A. MoveNourriture()

La fonction MoveNourriture() prend en paramètre une struct fourmi f et renvoie une struct fourmi . Dans cette fonction, la fourmi cherche de la nourriture, elle sort du nid (f.prec=0 et f.retour=0).

Principe :

- On crée une struct fourmi rf
- On crée un tableau d'entiers p[] tel que (p[0],p[1],p[2],p[3]) le niveau de phéromones dans les 4 cases de destination avec l'équivalence mouvement/entier (↑,↓,←,→)↔(0,1,2,3)
- On interdit un des mouvements possibles suivants, selon le mouvement précédent car la fourmi ne peut pas faire demi-tour (grâce à switch_nourriture()).
- S'il y a le même niveau de phéromones dans au moins deux cases (différentes de la case du mouvement interdit), on tire une des trois cases au hasard (grâce à Rand4Sauf1())
- Sinon, on cherche la case (différentes de la case du mouvement interdit) où il y a le moins de phéromones :On tire une case au hasard parmi les deux cases restantes (grâce à Rand4Sauf2())
- Si la fourmi arrive sur une case adjacente au bord (case_bord(rf)==1), on annule le mouvement
- On renvoie rf

B. MoveRetour()

Cette fonction prend et renvoie les mêmes paramètres que la fonction précédente. La fourmi f a trouvé de la nourriture (f.retour=1) et doit maintenant rentrer au nid.

On procède de façon analogue à la fonction précédente, en remplaçant le tableau d'entiers p[] par le tableau d'entiers d[] qui correspond à la distance entre les cases adjacentes à la position de la fourmi et le nid.

Nous remplaçons également le minimum par le maximum car la fourmi ne peut pas aller dans la case qui se trouve la plus loin du nid.

C. La fonction evolve()

Cette fonction prend en paramètre une struct environment et elle retourne une struct environment dans laquelle les champs sont mis à jour selon les règles de la version 2

Principe :

- On place le nid de type de structure position au milieu de la grille de taille N
- On crée une struct environnement ne
- On initialise la struct environment ne grace a init_v2()
- Pour chaque fourmi on procède de la façon suivante
    - Tant Que la fourmi n'a pas trouvé de nourriture, on réitère ce procédé
        - La fourmi cherche de la nourriture grâce à MoveNourriture()(quand position de la fourmi est égale à une case contenant de la nourriture)
        - On affiche le nombre de tours que la fourmi a fait pour trouver de la nourriture une fois la nourriture trouvée
    - Le champ retour de la fourmi est nulle tant que la fourmi n'est pas rentrée c'est à dire tant que la position courante de la fourmi ne correspond pas au nid
    - On réitère le procédé suivant jusqu'à ce que la fourmi soit rentré dans le nid
    - On avance vers le nid d'une case grâce à la fonction MoveRetour()
    - Le niveau de phéromones correspondant à la position de la fourmi et le champ tour sont incrémenté de 1 à chaque tour
- Une fois que la fourmi est rentré on affiche le nombre de tours que la fourmi a pris pour rentrer
- On retourne ne

III. III. Version2 améliorée et compilation

A. Compilation

On se place dans le répertoire courant avec la commande cd 2ME005/Projet

Puis, on compile grâce à la commande gcc -lm -o version2(resp. v2am) * .c * .h

Puis, on fait des test en changeant les valeurs des define et en utilisant la commande ./v2 (resp ./v2am)

B. Les changements

Dans la version améliorée, les fourmis peuvent à présent se déplacer en diagonale et ont une durée de vie limitée pour trouver la nourriture dr, les phéromones disparaissent également au bout d'une certaine durée dp. On considère que dr<dp. Plusieurs nourritures peuvent être placées dans la même case, et lorsqu'une fourmi passe par une case elle enlève une nourriture.

C. L'expression de ces changements sur le code

Les nouvelles valeurs dr et dp sont définies à l'aide de #define. Le type struct environment est modifié : le champ phéromones[][] devient un tableau de struct position de dimension 2, le champ nourriture devient un tableau d'entiers à deux dimensions de taille N*N.

Dans init_v2() on remplace phéromones[i][j] par phéromones[i][j].x on initialise le tableau phéromones[][].y à 0. La fonction est modifiée pour pouvoir placer plusieurs nourritures dans la même case.

Dans MoveNourriture(), on prend en compte les mouvements diagonaux, on crée donc deux nouvelles fonctions switch_nourriture2() et déplacement_r2() adaptés à 8 mouvements.

Pour faire en sorte que la fourmi se déplace aléatoirement, on change totalement de méthode.

On calcule les k tels que d[k] est maximum. On doit en avoir au moins 2. Puis, on tire au hasard parmi ces valeurs.

En réitérant tant que r=rand()%8 réponde aux conditions, cela ne crée pas une boucle infinie car le caractère de la fonction rand() n'est en réalité pas expressément aléatoire.

Dans MoveRetour() on effectue les mêmes changements en replaçant le maximum par le minimum car la fourmi ne va pas dans les cases les plus lointaines du nid.

Dans evolve(), on remplace phéromones[][] par phéromones[][].x et à chaque fois que la fourmi est en position (fx;fy) on incrémente phéromones[fx][fy] de 1. On vérifie à chaque tour que la durée de vie des phéromones de toutes les fourmis n'est pas dépassée si c'est le cas on enlève une phéromone (phéromones[a][b].x--) et on réinitialise phéromones[a][b] à 0. On incrémente de 1 la durée des phéromones à chaque tour. On a pris en compte que les phéromones sont indénombrables dans evolve().

On remplace nourriture[j] par nourriture[fx][fy] avec (fx;fy) la position courante de la fourmi, partout.

En conclusion, nous avons pu étudier le comportement des fourmis dans le cadre de ce projet. Personnellement, j'ai trouvé ce sujet très intéressant car on devait, non seulement être autonome, mais de surcroît on avait possibilité d'étudier un sujet concret. Par ailleurs, les difficultés que j'ai dû surmontées sont que j'étais seule donc contrairement aux autres, je devais tout comprendre et tout traiter par moi-même. J'ai aussi dû commencer à apprendre à programmer sur un nouveau logiciel de programmation (DV C++) car en vue des grèves je n'ai pu travailler sur les ordinateurs de l'université. Je n'ai donc pu tester toutes mes fonctions rencontrant des problèmes avec ce logiciel. J'aurai aimé pouvoir travailler dans de meilleures conditions pour pouvoir traiter le sujet plus profondément.