Université de Reims Champagne-Ardenne
Département de Mathématiques, Mécanique et Informatique

Info0604 – 2019-2020
Pierre Delisle

Travaux pratiques 5

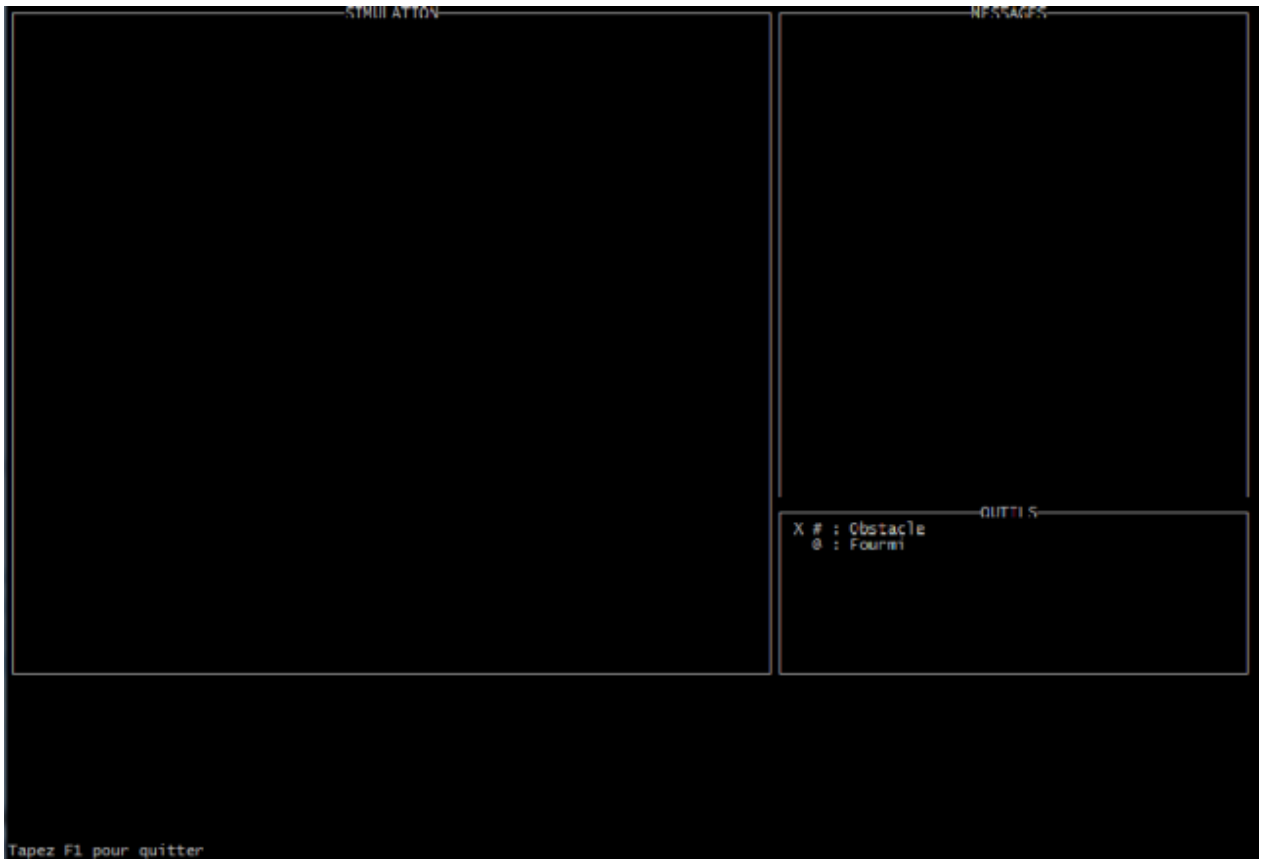
Annulation de threads – L'attaque des fourmis tueuses !

L'objectif du TP est de développer une simulation multi-threadée d'un combat de fourmis tueuses en utilisant la bibliothèque **pthreads**. Pour ce faire, vous vous baserez sur un code source initial que vous pourrez modifier à votre guise tout au long du TP.

Étape 1 : prise en main du code source initial

Vous pouvez télécharger le code source initial proposé "**fourmis.c**" sur la page Moodle du cours.

Compilez et exécutez ce programme. Vous pouvez alors observer l'affichage suivant :



La fenêtre SIMULATION contient la grille de simulation contenant plusieurs cases pouvant soit être vides, soit contenir une fourmi, soit contenir un obstacle. La fenêtre MESSAGES contient une description des événements qui se produisent et la fenêtre OUTILS contient les éléments cliquables que vous pouvez placer dans la simulation en cliquant dans la fenêtre SIMULATION.

Vous remarquerez que pour l'instant, il ne se passe pas grand chose sur la grille SIMULATION quand vous ajoutez une fourmi ou un obstacle : votre tâche consiste à donner vie aux fourmis tueuses en suivant les étapes données ci-dessous. Prenez tout d'abord quelques minutes pour vous familiariser avec ce programme qui est donné et commenté en annexe.

Étape 2 : déplacement aléatoire des fourmis

Chaque fourmi est simulée par un thread différent et leur comportement est donc défini par le code de la routine de thread appelée **routine_fourmi**. Comme vous l'avez sûrement remarqué, cette routine ne contient actuellement pas de code et cela explique le peu de dynamisme des fourmis. Vous devez maintenant programmer cette routine afin de permettre aux fourmis de se déplacer aléatoirement sur la grille. Plus spécifiquement, implémentez cet algorithme simple :

1. La fourmi choisit une case au hasard (haut, bas, droite ou gauche) ;
2. Si la case choisie est vide
   la fourmi se déplace sur cette case (en modifiant la grille de façon appropriée)
   Sinon
   la fourmi ne fait rien ;
3. Actualiser l'affichage.

Testez votre implémentation en vérifiant que toutes vos fourmis se déplacent à intervalles réguliers. Comme les fourmis ne font actuellement que se déplacer, la simulation devrait se poursuivre indéfiniment.

Étape 3 : destruction des obstacles

En vous basant sur l'algorithme de l'étape précédente, vous devez maintenant améliorer vos fourmis en leur injectant une petite dose d'agressivité : lorsque la case choisie contient un obstacle, plutôt que de ne rien faire, la fourmi détruit cet obstacle.

Étape 4 : les fourmis ne rigolent plus !

L'objectif de ce TP étant de réaliser une simulation de combat de fourmis tueuses, il s'agit maintenant de donner aux fourmis des moyens de remporter le combat. Nous leur ajouterons donc une dose supplémentaire d'agressivité en leur permettant de détruire les autres fourmis. Plus spécifiquement, lorsque la case choisie par une fourmi contient une autre fourmi, plutôt que de ne rien faire, la fourmi détruira cet adversaire qui se trouve sur son chemin.

Il est à noter que la destruction d'une fourmi n'implique pas seulement de la faire disparaître de l'affichage : il s'agit aussi d'arrêter l'exécution de son thread. L'annulation doit se faire le plus rapidement possible tout en évitant de perturber le déroulement de la simulation. Cela implique que la fourmi à l'origine de l'agression effectue une requête d'annulation de la fourmi qu'elle élimine. Afin de gérer correctement cette nouvelle agressivité des fourmis, utilisez le mode d'annulation retardé (deferred) et assurez-vous de gérer correctement et proprement cette annulation. La simulation se terminera alors lorsqu'il ne restera qu'une seule fourmi sur la grille.

Exercices supplémentaires pour les rapides et les motivés

- Ajoutez des pièges à la grille : lorsqu'une fourmi est prise dans un piège, elle ne peut plus bouger (pendant un certain temps ou jusqu'à ce qu'elle arrive à détruire le piège) ;
- Ajoutez des pastilles magiques à la grille : lorsqu'une fourmi se déplace sur une case contenant une pastille magique, elle devient invincible pendant un certain temps ;
- Ajoutez un comportement d'agression extrême à vos fourmis : la fourmi regarde toutes les cases autour d'elle et tue toutes les fourmis qui s'y trouvent. Attention aux verrous mortels !

Annexe – fourmis.c (Code source initial à compléter/modifier)

```
#define NB_LIGNES_SIM      40    /* Dimensions des fenetres du programme */
#define NB_COL_SIM         80
#define NB_LIGNES_MSG      29
#define NB_COL_MSG         49
#define NB_LIGNES_OUTILS   9
#define NB_COL_OUTILS      49

#define MAX_FOURMIS        10    /* Nombre maximum de fourmis de la simulation */

#define VIDE               0     /* Identifiants des elements pouvant etre */
#define OBSTACLE           1     /* places sur la grille de simulation */
#define FOURMI             2

typedef struct case_tag {        /* Description d'une case sur la grille de simulation */
   int element;                  /* Ce qui est present sur la case */
   pthread_t *fourmi;            /* Identifiant du thread de la fourmi presente sur la case */
   pthread_mutex_t mutex;        /* Protection de la case */
} case_t;

typedef struct coord_tag {       /* Coordonnees d'une case sur la grille de simulation*/
   int y;
   int x;
} coord_t;

pthread_t *threads_fourmis[MAX_FOURMIS];  /*Identifants des threads des fourmis de la simulation*/

case_t grille[NB_LIGNES_SIM][NB_COL_SIM];   /* Grille de simulation */
WINDOW *fen_sim;                            /* Fenetre de simulation partagee par les fourmis */
WINDOW *fen_msg;                            /* Fenetre de messages partagee par les fourmis*/

void ncurses_initialiser() {
   initscr();                                /* Demarre le mode ncurses */
   cbreak();                                 /* Pour les saisies clavier (desac. mise en buffer) */
   noecho();                                 /* Desactive l'affichage des caracteres saisis */
   keypad(stdscr, TRUE);                     /* Active les touches specifiques */
   refresh();                                /* Met a jour l'affichage */
   curs_set(FALSE);                          /* Masque le curseur */
   mousemask(BUTTON1_CLICKED, NULL);         /* Active le clic gauche de la souris*/
}

void ncurses_stopper() {
   endwin();
}

void simulation_initialiser() {
   int i, j;
   for (i = 0; i < MAX_FOURMIS; i++)         /* Au depart il n'y a aucune fourmi dans la simulation */
      threads_fourmis[i] = NULL;
   for (i = 0; i < NB_LIGNES_SIM; i++) {     /* Initialisation des cases de la simulation */
      for (j = 0; j < NB_COL_SIM; j++) {
         grille[i][j].element = VIDE;
         grille[i][j].fourmi = NULL;
         pthread_mutex_init(&grille[i][j].mutex, NULL);
      }
   }
}
```

```
void simulation_stopper() {
    int i;
    for (i = 0; i < MAX_FOURMIS; i++) {
        if (threads_fourmis[i] != NULL) {
            pthread_cancel(*threads_fourmis[i]);
            threads_fourmis[i] = NULL;
        }
    }
}

WINDOW *creer_fenetre_box_sim() {
/*Creation de la fenetre de contour de la fenetre de simulation */

    WINDOW *fen_box_sim;

    fen_box_sim = newwin(NB_LIGNES_SIM + 2, NB_COL_SIM + 2, 0, 0);
    box(fen_box_sim, 0, 0);
    mvwprintw(fen_box_sim, 0, (NB_COL_SIM + 2) / 2 - 5, "SIMULATION");
    wrefresh(fen_box_sim);

    return fen_box_sim;
}

WINDOW *creer_fenetre_sim() {
/* Creation de la fenetre de simulation dans la fenetre de contour */
/* La simulation est affichee dans cette fenetre */

    WINDOW *fen_sim;

    fen_sim = newwin(NB_LIGNES_SIM, NB_COL_SIM, 1, 1);

    return fen_sim;
}

WINDOW *creer_fenetre_box_msg() {
/* Creation de la fenetre de contour de la fenetre de messages */

    WINDOW *fen_box_msg;

    fen_box_msg = newwin(NB_LIGNES_MSG + 2, NB_COL_MSG + 2, 0, NB_COL_SIM + 2);
    box(fen_box_msg, 0, 0);
    mvwprintw(fen_box_msg, 0, (NB_COL_MSG + 2) / 2 - 4, "MESSAGES");
    wrefresh(fen_box_msg);

    return fen_box_msg;
}

WINDOW *creer_fenetre_msg() {
/* Creation de la fenetre de messages dans la fenetre de contour */
/* Les messages indicatifs des evenements de la simulation et de l'interface */
/* utilisateur sont affiches dans cete fenetre */

    WINDOW *fen_msg;

    fen_msg = newwin(NB_LIGNES_MSG, NB_COL_MSG, 1, NB_COL_SIM + 3);
    scrollok(fen_msg, TRUE);

    return fen_msg;
}
```

```
WINDOW *creer_fenetre_box_outils() {
/* Fenetre de contour de la fenetre des outils */

    WINDOW *fen_box_outils;

    fen_box_outils = newwin(NB_LIGNES_OUTILS + 2, NB_COL_OUTILS + 2, NB_LIGNES_MSG + 2 ,
                                                                                    NB_COL_SIM + 2);
    box(fen_box_outils, 0, 0);
    mvwprintw(fen_box_outils, 0, (NB_COL_OUTILS + 2) / 2 - 3, "OUTILS");
    wrefresh(fen_box_outils);
    return fen_box_outils;
}

WINDOW *creer_fenetre_outils() {
/* Creation de la fenetre des outils a l'interieur de la fenetre de contour */
/* Les outils dans cette fenetre sont clickables, l'outil actif etant indique par un X */

    WINDOW *fen_outils;

    fen_outils = newwin(NB_LIGNES_OUTILS, NB_COL_OUTILS, NB_LIGNES_MSG + 3, NB_COL_SIM + 3);
    mvwprintw(fen_outils, 0, 3, "# : Obstacle\n");
    mvwprintw(fen_outils, 1, 3, "@ : Fourmi");
    mvwprintw(fen_outils, 0, 1, "X");
    wrefresh(fen_outils);
    return fen_outils;
}

void *routine_fourmi(void *arg) {
    coord_t *coord = (coord_t *) arg;

    while (1) {
        /*Que feront les fourmis ?!?!?!*/
        sleep(1);
    }

    free(coord);
    return NULL;
}

int main(int argc, char *argv[]) {
    WINDOW *fen_box_sim, *fen_box_msg, *fen_msg, *fen_box_outils, *fen_outils;
    MEVENT event;
    int ch, i, item_actif = OBSTACLE;
    coord_t *coord;

    ncurses_initialiser();
    simulation_initialiser();

    fen_box_sim = creer_fenetre_box_sim();
    fen_sim = creer_fenetre_sim();
    fen_box_msg = creer_fenetre_box_msg();
    fen_msg = creer_fenetre_msg();
    fen_box_outils = creer_fenetre_box_outils();
    fen_outils = creer_fenetre_outils();

    mvprintw(LINES - 1, 0, "Tapez F2 pour quitter");
    wrefresh(stdscr);
```

```
    while((ch = getch()) != KEY_F(2)) {
        switch(ch) {
            case KEY_MOUSE :
                if (getmouse(&event) == OK) {
                    wprintw(fen_msg, "Clic a la position %d %d de l'ecran\n", event.y, event.x);
                    wrefresh(fen_msg);
                    if (event.y == 32 && event.x >= 82 && event.x <= 98) {
                        item_actif = OBSTACLE;
                        mvwprintw(fen_outils, 0, 1, "X");    mvwprintw(fen_outils, 1, 1, " ");
                        wrefresh(fen_outils);    wprintw(fen_msg, "Outil obstacle active\n");    wrefresh(fen_msg);
                    }
                    else if (event.y == 33 && event.x >=82 && event.x <= 98) {
                        item_actif = FOURMI;
                        mvwprintw(fen_outils, 0, 1, " ");    mvwprintw(fen_outils, 1, 1, "X");
                        wrefresh(fen_outils);    wprintw(fen_msg, "Outil fourmi active\n");    wrefresh(fen_msg);
                    }
                    else if (event.y>0 && event.y<NB_LIGNES_SIM+1 && event.x>0 &&event.x<NB_COL_SIM+1) {
                        switch (item_actif) {
                            case OBSTACLE :
                                pthread_mutex_lock(&grille[event.y - 1][event.x - 1].mutex);
                                if (grille[event.y - 1][event.x - 1].element == VIDE) {
                                    grille[event.y - 1][event.x - 1].element = OBSTACLE;
                                    mvwprintw(fen_sim, event.y - 1, event.x - 1, "#");
                                    wprintw(fen_msg, "Ajout d'un obstacle a la position %d %d\n", event.y - 1, event.x - 1);
                                }
                                wrefresh(fen_sim);    wrefresh(fen_msg);
                                pthread_mutex_unlock(&grille[event.y - 1][event.x - 1].mutex);
                                break;
                            case FOURMI :
                                pthread_mutex_lock(&grille[event.y - 1][event.x - 1].mutex);
                                if (grille[event.y - 1][event.x - 1].element == VIDE) {
                                    i = 0;
                                    while (i < MAX_FOURMIS && threads_fourmis[i] != NULL)
                                        i++;
                                    if (i < MAX_FOURMIS) {
                                        threads_fourmis[i] = (pthread_t *) malloc(sizeof(pthread_t));
                                        grille[event.y - 1][event.x - 1].element = FOURMI;
                                        grille[event.y - 1][event.x - 1].fourmi = threads_fourmis[i];
                                        coord = (coord_t *) malloc(sizeof(coord_t));
                                        coord->y = event.y - 1;
                                        coord->x = event.x - 1;
                                        pthread_create(threads_fourmis[i], NULL, routine_fourmi, (void *) coord);
                                        mvwprintw(fen_sim, event.y - 1, event.x - 1, "@");
                                        wprintw(fen_msg, "Ajout de fourmi a la position %d %d\n", event.y-1, event.x-1);
                                    }
                                    else wprintw(fen_msg, "Nombre maximum de fourmis atteint\n");
                                }
                                wrefresh(fen_sim);    wrefresh(fen_msg);
                                pthread_mutex_unlock(&grille[event.y - 1][event.x - 1].mutex);
                                break;
                        }
                    }
                }
            break;
        }
    }
    delwin(fen_box_sim); delwin(fen_sim);    delwin(fen_box_msg);    delwin(fen_msg);
    delwin(fen_box_outils); delwin(fen_outils);  simulation_stopper(); ncurses_stopper();
    return 0;
}
```