Programmation Concurrente

Conway's game of life

Travail Pratique

But du projet

Le but de ce travail pratique est de modéliser le jeu de la vie tel qu'imaginé et conçu par le mathématicien John Horton Conway en 1970.

L'univers du jeu de la vie se compose d'une grille à deux dimensions se composant de cellules. Chaque cellule possède deux états possibles : vivante ou morte (aussi dit habité/vide). Chaque cellule interagit avec ses huit voisins (horizontaux, verticaux, diagonaux) comme illustré en Figure 1.

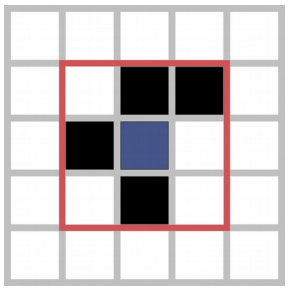

Figure 1 : Le voisinage de la cellule bleue est représenté par la zone en rouge.

A chaque instant temporel t, l'état de chaque cellule de la grille est défini selon les quatre règles ci-dessous :

(1) Chaque cellule vivante possédant moins de deux voisins meurt, comme si causé par une population insuffisante.

(2) Toute cellule vivante possédant deux ou trois voisins reste en vie.

(3) Toute cellule vivante possédant plus de trois voisins meurt, comme si causé par une sur-population.

(4) Toute cellule morte possédant exactement trois voisins devient vivante, comme si causé par reproduction.

Soit t_0 l'état initial du système au temps zéro. Le prochain état du système au temps t_1 ($t_1 = t_0+1$) est calculé en appliquant les quatre règles définies précédemment simultanément à toutes les cellules de la grille. Le même processus est ensuite appliqué à toutes les cellules de la grille pour passer de l'état t_1 à t_2 où $t_2 = t_1+1 = t_0+2$. Tout nouvel état t_i dépend donc exactement et uniquement de l'état précédent t_i-1. Ce processus d'évolution du système se poursuit indéfiniment. En termes d'évolution, le système converge soit vers état stable, ou un état cycle. Ceci est observable en choisissant des conditions initiales différentes.

La Figure 2 illustre l'évolution des cellules pour une grille de 34 colonnes par 22 lignes aux temps t_i, t_i+1, t_i+2.

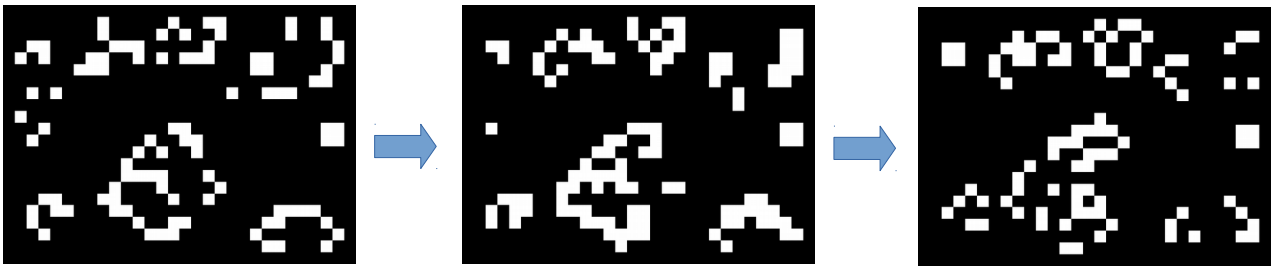

Figure 2 : évolution des cellules à 3 temps consécutifs

L'objectif de ce travail pratique est de réaliser une version multi-threadée du jeu de la vie avec affichage graphique.

Le programme à développer sera nommé `gameoflife` et sa syntaxe est donnée par la Table 1ci-après :

`gameoflife <width> <height> <seed> <p> <freq> <#workers>`

- `width and height are integers specifying the dimensions of the game (>= 4).`
- `seed is an integer used to randomly populate the board.`
- `p is a floating point value (range [0..1]) which is the probability of having a live cell during initialization.`
- `freq is an integer specifying the display frequency in Hz (> 0).`
- `#workers is an integer specifying the number of worker threads (>= 1).`

`Example: gameoflife 240 135 0 0.75 30 8`

Table 1 : arguments et syntaxe du programme à développer

Architcture multi-threadée

Le programme à développer est basé sur une architecture multi-threadée organisée de la manière suivante :

- Le thread principal (fonction `main`) s'occupe seulement de gérer les arguments de la ligne de commande, initialiser la grille, et créer les threads.
- Un thread est dédié à l'affichage.
- Un thread est dédié à la gestion du clavier.
- Un ou plusieurs threads *travailleurs* sont dédiés au calcul du prochain état de la grille.

Cahier des charges

- Votre programme se nommera `gameoflife` et proposera la syntaxe décrite en Table 1.
- La bordure de la grille (première/dernière lignes, première/dernière colonnes) est à initialiser avec des cellules vides (i.e. cellules mortes). Aucun calcul ne sera effectué sur cette bordure : celle-ci sera uniquement considérée dans le calcul des voisins.
- La taille de la grille à créer est donnée par les arguments **`width`** et **`height`** spécifiés sur la ligne de commande.

- L'état initial de la grille (absence/présence de cellules) est calculé aléatoirement selon les deux arguments **`seed`** et **`p`** spécifiés sur la ligne de commande :
  - **`seed`** est la graine à utiliser par le générateur de nombres aléatoires ;
  - chaque cellule de la grille est initialisée avec une probabilité **`p`** indiquant la probabilité d'y trouver une cellule vivante. Une grille initialisée avec **`p`** `= 1` signifie une grille constituée uniquement de cellules vivantes. A l'opposé, une grille initialisée avec **`p`** `= 0` signifie une grille constituée uniquement de cellules mortes.
- Le thread d'affichage affichera la grille à la fréquence spécifiée par l'argument **`freq`** sur la ligne de commande. Attention à ce que l'affichage de la grille ne soit réalisé qu'une fois celle-ci **entièrement** mise à jour !
- L'affichage doit être réalisé avec les fonctions fournies (`gfx_create, gfx_present`, etc.) ; cf. section suivante.
- Le thread responsable de la gestion du clavier testera, à la fréquence de 50 Hz, si une touche a été pressée. Le programme se terminera, **proprement**, une fois la touche d'échappement (ESC) pressée.
- Le nombre de threads travailleurs dédiés au calcul du prochain état de la grille est spécifié par l'argument **`#workers`** sur la ligne de commande**.**
- La division du travail entre les threads travailleurs doit être la plus équitable possible.
- La division du travail ne doit pas être limitée par le nombre de lignes ou colonnes de la grille. Dans le cas d'une grille 10x10 par exemple, nous ne voulons pas être limité à 10 threads au maximum.
- Seules les primitives de synchronisation vues en cours sont autorisées.
- **Toute attente active est prohibée.**
- **Aucune** variable globale n'est autorisée. A noter qu'il est permis d'utiliser des constantes globales (déclarées via la directive `#define` ou le mot-clé `const`).
- Attention : dans les calculs de timings, n'oubliez pas de tenir compte du temps écoulé entre deux mesures successives. Par exemple, si une routine *R* doit être appelée à la fréquence de 1 Hz, réaliser un sleep de 1 seconde est incorrect car il est nécessaire de tenir compte du temps d'exécution de la routine *R*.

Informations utiles

Affichage graphique

Un exemple de code fonctionnel illustrant l'affichage de pixels en mode graphique vous est fourni avec cet énoncé. Celui-ci se trouve dans l'archive `gfx_example.tar.gz` fournie avec ce travail pratique. Vous **devez** utiliser ces fonctions pour l'affichage de votre programme.

Timing

Les mesures de timing peuvent être effectuées avec la fonction `clock_gettime` de la librairie `<time.h>` comme montré ci-dessous :

```
struct timespec start, finish;
clock_gettime(CLOCK_MONOTONIC, &start);

...  // code à mesurer
```

```
clock_gettime(CLOCK_MONOTONIC, &finish);
double seconds_elapsed = finish.tv_sec-start.tv_sec;
seconds_elapsed += (finish.tv_nsec-start.tv_nsec)/1000000000.0;
```

Important : le code ci-dessus requiert la librairie `rt`. Pour ce faire, passez `-lrt` à `gcc` au moment de l'édition des liens.

Travail à rendre

N'oubliez pas de lire attentivement le document « *Consignes Travaux Pratiques.pdf* » disponible sur la page CyberLearn du cours.

Pour ce travail pratique, chaque groupe me rendra une archive contenant les fichiers suivants :

- Les fichiers sources, incluant un makefile pour compiler le projet (et faire le ménage).
- Le rapport au format PDF.

Le code doit respecter les consignes décrites dans le document « *Consignes pour l'écriture du code* » et le rapport doit respecter les consignes décrites dans le document *« Consignes Travaux Pratiques.pdf »*. Ces deux documents se trouvent sur la page CyberLearn du cours.