Programmation concurrente
Game of Life

Claudio Sousa - David Gonzalez

11 décembre 2016

Image...

1 Introduction

Ce TP de deuxième année consiste à implémenter un Game of Live (jeux de la vie). La particularité de celui-ci est que tous modules doit s'exécuter en parallèle.

Les modules concernés sont :
- gestion du clavier (un thread) ;
- gestion d'affichage (un thread) ;
- gestion de la grille du Game of Life (un ou plusieurs threads).

Le traitement de la grille suit des règles selon 2 paramètres :
- l'état de la cellule : morte ou vivante ;
- le nombre de voisins.

Ainsi, une cellule vivante meure seulement si elle a 0, 1, ou plus de 3 voisins.
Une cellule morte revit seulement si elle a exactement 3 voisins.

2 Development

2.1 Architecture

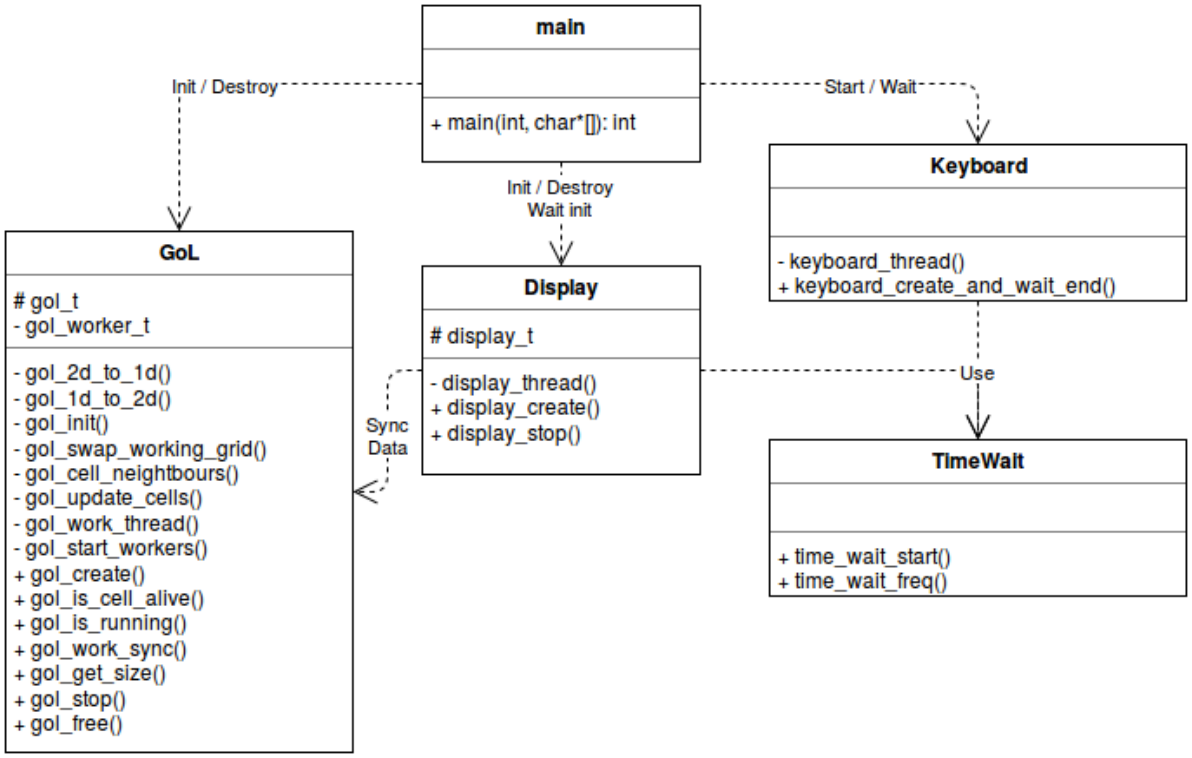

FIGURE 1 – Architecture du Game of Life

L'architecture du Game of Life est divisée en 5 modules.

Le module *main* est le programme principal. Il a pour rôle de :
- récupérer les paramètres de ligne de commande ;
- initialiser et lancer les threads des différents modules ;
- stopper et libérer les modules lorsque demandé.

Le module *time_wait* permet simplement a un thread de se synchroniser avec une fréquence de fonctionnement (en herz).

Le module *display* contient le thread qui s'occupe de l'affichage.
Il a la responsabilité d'initialiser la librairie SDL à l'intérieur du thread. Afin de permettre à d'autre module d'utiliser la SDL, une barrière est utilisée et est joint deux fois :
- une fois par le thread d'affiche après l'initialisation de la SDL ;
- une fois par le thread principal après avoir lancé le thread d'affichage.

Ceci permet d'empêcher le thread principal de continuer avant que la SDL ne soit initialisée.

Le module *keyboard* est également un thread qui a pour seul rôle de bloquer le thread principal tant que la touche *ESC* n'a pas été pressée. Pour cela, le thread principal lance le thread du clavier puis attend sa fin en le joignant immédiatement après l'avoir lancé.

Le module *gol* contient l'ensemble de l'algorithme qui permet de traiter la grille en parallèle...

2.2 Algorithmie

2.2.1 Répartition du travail entre les threads du GoL

2.2.2 Swap

2.3 Concurrence

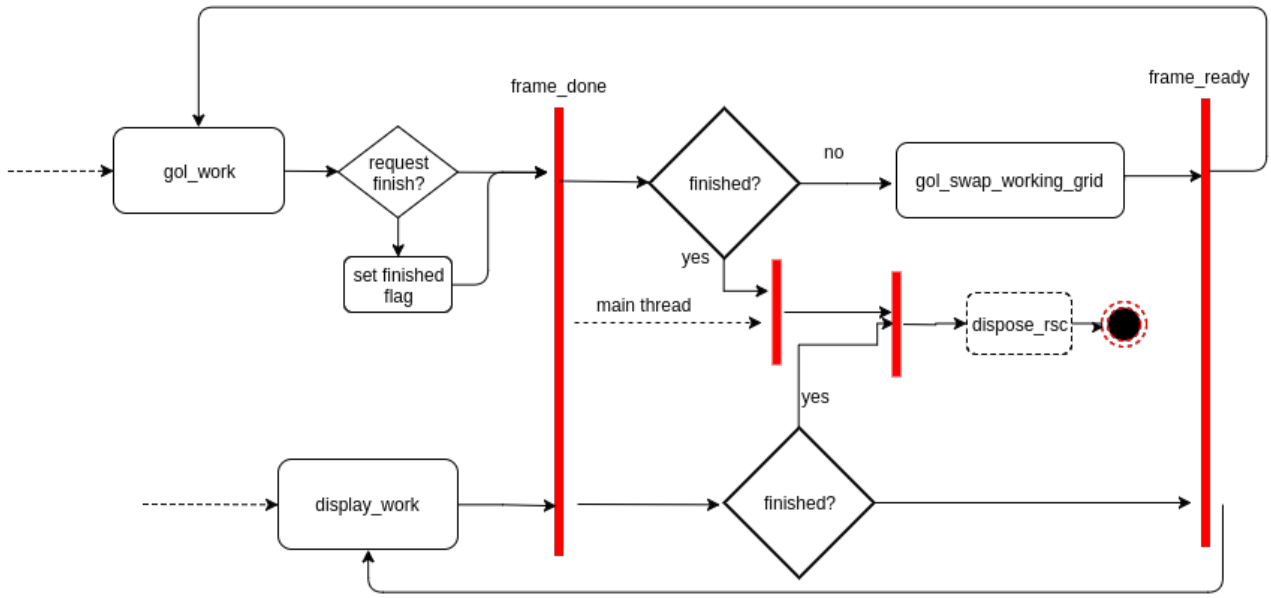

FIGURE 2 – Synchronisation du traitement et de l'affichage

2.3.1 Synchronisation du traitement et de l'affichage

Les flèches à gauche du schéma symbolise les threads, n threads pour le traitement de la grille (haut) et le thread pour l'affichage de cette grille (en bas).

Tous les threads sont synchornisés à deux endroits, symbolisés par deux barres rouges verticales :
- *frame_done* synchronise tous les threads à la fin du traitement de l'état actuel de la grille.
- *frame_ready* s'assure que tous les threads attendent que la grille suivante soit prête à être traitée.

2.3.2 Condition de sortie

Lorsque la touche *ESC* est pressée, le thread du clavier se termine et libère le thread principal. Celui-ci met la variable *request_finish* à *true* et attend que tous les threads se termine.

Les threads du traitement de la grille vérifient cette variable avant la barrière *frame_done* et, le cas échéant, mettent une autre variable *finished* à *true*. Après *frame_done*, chaque thread vérifie l'état de cette variable et se termine s'elle est mise à *true*.

L'utilisaton de deux variables, et surtout la separation entre l'écriture et la lecture de *finish*, permet de s'assurer que sa valeur sera la même pour tous les threads entre *frame_done* et *frame_ready*.

La fin des threads redonne la main au thread principal qui libère les ressources du programme.

2.4 Méthodologie de travail

2.4.1 Répartition du travail