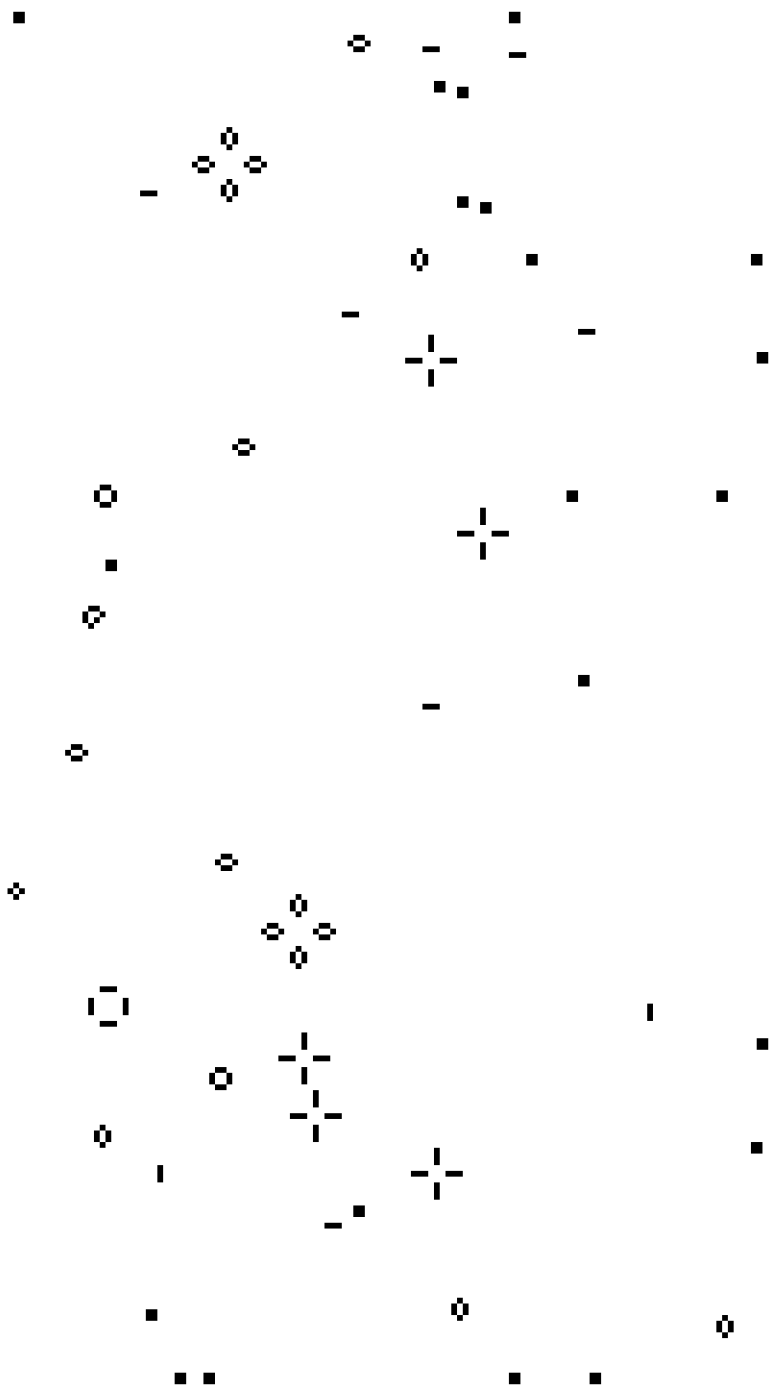
# Programmation concurrente
# Game of Life

David Gonzalez - Claudio Sousa

11 décembre 2016



# 1 Introduction

Ce TP de deuxième année consiste à implémenter le Game of Live (par Conway). La particularité de celui-ci est que tous les modules doivent 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 vivants.

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

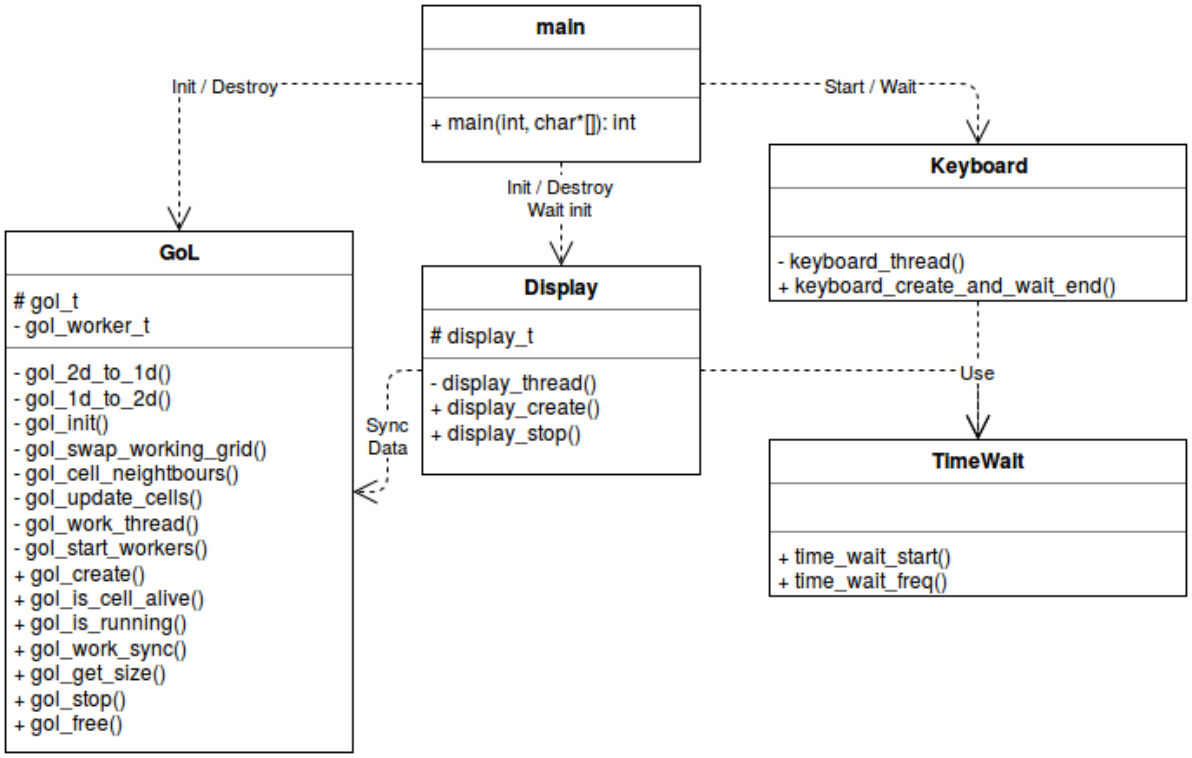



FIGURE 1 – Architecture du Game of Life

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

### 2.1.1 Main

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 les threads et libérer la mémoire des modules lorsque demandé.

### 2.1.2 TimeWait

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

### 2.1.3 Affichage

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'autres modules d'utiliser la SDL, une barrière est utilisée et est jointe deux fois :

- une fois par le thread d'affichage 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.

### 2.1.4 Clavier

Le module *keyboard* contient 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.

### 2.1.5 GameOfLife

Le module *gol* contient l'ensemble de l'algorithme qui permet de traiter la grille en parallèle. En particulier, il :

- initialise la grille originalle en fonction de la graine et la probabilité d'une cellule vivante spécifiés par l'utilisateur ;
- offre les méthodes publiques d'accès au tableau (*gol_is_cell_alive* et *gol_get_size*) ;
- permet de synchroniser l'affichage avec les débuts et fins des traitements de la grille (*gol_work_sync*) ;
- expose l'état de fin de jeu (*gol_is_running*).

**Mémoire**
La grille symbolisant l'état du jeu est un tableau boolean d'une dimension. Un deuxième tableau similaire est aussi alloué comme mémoire de travail temporaire.

**Calcul de l'état suivant**
Dans ce module, plusieurs threads calculent l'état *t+1* à partir de l'état actuel *t*. Simultanement, le thread d'affichage met les pixels *GFX* d'après le tableau actuel.

**Échange des grilles de travail**
Les threads calculant l'état suivant de la grille et le thread d'affichage sont synchronisés une fois leur travail sur le tableau actuel est fini. Après cette synchronization, un worker thread va échanger les pointers des tableaux *t* et *t+1*. Une fois cette opération faite, tous les threads synchronisent une fois de plus et commencer a traiter le nouvel état de la grille.

**Condition de sortie**
La méthode *gol_stop* du module permet à un thread extérieur d'arrêter le travail du *gol*. Cette méthode met un flag interne *request_stop* à *true*, notifiant les threads qu'il est temps d'arrêter et attends que ces threads se terminent avec *pthread_join*.
La synchronisation des threads avec ce flag est expliquée plus en détail dans le chapitre 2.2 Concurrence.

## 2.2 Concurrence

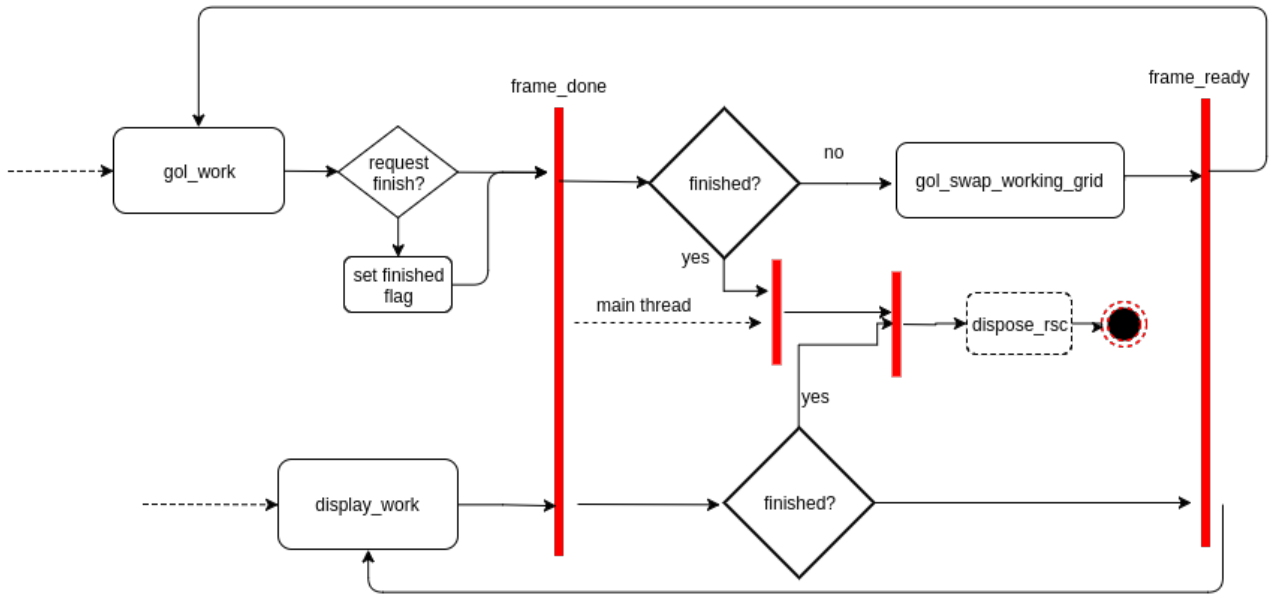



FIGURE 2 – Synchronisation du traitement et de l'affichage

### 2.2.1 Synchronisation du traitement et de l'affichage

Les flèches à gauche du schéma symbolisent 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 synchronisés 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.

### 2.2.2 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 terminent.

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.

# 3 Méthodologie de travail

## 3.1 Répartition du travail

Ce travail a été effectué à deux.
Nous avons commencés par réfléchir sur papier sur deux éléments :

- architecture du programme : modules et interfaces de bases ;
- premier jet de synchronisation entre les différents threads.

Ensuite, le travail a été réparti ainsi :

- le premier collaborateur a fait le module du *GameOfLife*.
- le deuxième collaborateur a fait les modules restant : *Main*, *Clavier*, *Affichage* et *TimeWait*.

Finalement, nous avons mis en commun les modules et finalisés la synchronisation des threads des différents modules.