RAPPORT

Sujet : Conception d’un automate cellulaire

Auteurs : NEONAKIS Ionas et TOULARHMINE Samir

Langage utilisé : C

Outils utilisés :

- → Versionnage : Git
- → Gestion de la compilation : Makefile
- → Gestion de la documentation : Doxygen
- → Gestion de la mémoire : Valgrind
- → Répartition des différentes tâches à réaliser : Trello

I – Choix d’implémentation

Tout d’abord, avant de répartir le travail à effectuer, nous avons établi ensemble la structure du projet.
En effet, nous avons jugé cette étape comme une des plus importantes car elle représente en quelques sortes la fondation du projet. Si cette dernière s’avère mauvaise, le développement du projet sera une tâche beaucoup plus compliquée.

Pour débuter, nous avons commencé par représenter l’automate cellulaire. Pour ce faire nous avons décidé d’utiliser une structure dont voici la signature :

SIGNATURE AUTOMATE

Afin d’implémenter les générations de cellule, nous avons préféré utiliser un tableau à deux dimensions où :
→ chaque case contient une cellule
→ chaque ligne représente une génération de cellule
Grâce à ça, nous pouvons facilement accéder à une génération donnée.

Pour représenter la règle de l’automate, nous avons utilisé une structure règle. Cette implémentation nous permet de définir très facilement un type de règle parmi ceux déjà définis ou par ceux que l’utilisateur décide de créer.

Nous pouvons remarquer la présence d’une fonction affichage dans la structure. Cela nous permet de définir un affichage générique, en plus d’être modulable, ce système nous permet de choisir facilement entre un affichage console, une génération d’image, etc...

SIGNATURE CELLULE

Afin de pouvoir gérer facilement une cellule, nous avons décidé de modéliser la structure de la manière suivante :

→ L’état courant de la cellule, il dépendra de la règle
→ Le voisin gauche de la cellule
→ Le voisin droit de la cellule

SIGNATURE REGLE

Chaque règle est représentée par :
→ Une chaine de caractère correspondant à une suite de chiffres compris entre 0 et le nombre d’états maximum de la règle
→ Une fonction générique s’appliquant sur une cellule, elle applique la règle de transition
→ le nombre d’états possible pour les cellules
→ la taille de la règle : 8 pour la règle Wolfram, 10 pour la règle somme par exemple

II – Utilisation du programme

Tout d'abord, c'est dans la fonction main que nous créons l'automate ainsi qu'une règle. En effet, si l'utilisateur décide d'utiliser une règle personnalisée, ce dernier devra créer une fonction de transition de règle et une fonction d'affichage de l'automate. Il devra ensuite préciser tout cela dans le main en passant ses fonctions ainsi que la taille de sa règle aux setters déjà présents. Sans modification du main, la règle personnalisée est définie sur la règle binaire afin de servir d'exemple, mais l'utilisateur pourra choisir une autre règle si il le souhaite.

Remarque : Nous avons établi la convention de définir les fonctions d'affichages dans le fichier affichage.c et leur signature dans le fichier affichage.h. De même, les fonctions de transition pour la règle personnalisée sont définie dans le fichier regle.c et leur signature dans le fichier regle.h.

En fonction de l'argument de démarrage utilisé, le programme va pouvoir recueillir toutes les informations nécessaires à la création de l'automate. En effet, chaque argument de démarrage propose à l'utilisateur une façon différente de créer son automate :

Les différents arguments disponibles

→ avec l'argument **-a** ou **--args**, l'utilisateur peut créer son automate en passant après cet arguments, les différentes informations de son automate, elles sont dans cet ordre :

NOMBRE_ITERATIONS DIMENSION_MAX CONFIGURATION_INITIALE REGLE TYPE_REGLE [TYPE_AFFICHAGE | NB_ETATS]

Remarque : type_affichage et nb_etats sont optionnels car dans le cas d'une règle personnalisée, il faut préciser le nombre d'états mais pas le type d'affichage car il est obligatoire pour l'utilisateur de concevoir sa propre fonction d'affichage pour sa règle. En revanche, dans le cas d'un type de règle déjà pris en charge par le programme (Wolfram et somme), l'utilisateur peut, si il le souhaite, utiliser une fonction d'affichage personnalisée tout de même ou alors utiliser celle déjà présente correspondant ainsi à son type de règle : il a donc le choix.

→ avec l'argument **-f NOM_FICHIER**, l'utilisateur utilise un fichier, qui doit être présent dans le dossier cfg, pour créer son automate. Ce fichier doit respecter les modèles déjà présents dans ce dossier. La remarque précédente s'applique également pour cette lecture.

Remarque : L'ordre des instructions n'importe pas, seulement si les conditions suivantes sont respectées :
→ L a dimension max doit être précisée avant la configuration initiale
→ Le type de règle doit être précisé avant le nombre d'états, si règle personnalisée il y a
→ Le nombre d'état doit être précisé avant la configuration initiale et la règle binaire

Remarque : Afin de sécuriser les entrées utilisateur, des pattern regex ont été utilisés pour être sur que l'utilisateur ne peut pas entrer n'importe quelle valeur. Afin d'appuyer cette approche, beaucoup de tests ont été effectués.

→ sans arguments, l'utilisateur utilise alors la **lecture runtime**. Cette fonction permet de rentrer les informations au fur et à mesure pendant l'exécution du programme.

Remarque : Afin de sécuriser les entrées utilisateur, nous avons utilisé un pointeur char* d'une taille de 1024 * sizeof(char). Ce pointeur nous permet de réceptionner l'entrée utilisateur, puis d'effectuer tous les tests nécessaire avant de stocker cette entrée dans une variable définitive. Ce pointeur est donc réutilisé pour chaque entrée.

Ce programme propose donc 3 manières différentes de créer son automate.

Remarque : Il existe un argument supplémentaire, **--help** ou **-h**, permettant à l'utilisateur d'afficher le manuel de l'automate.