Introduction C - TP 1

Marc Hartley

Janvier 2025

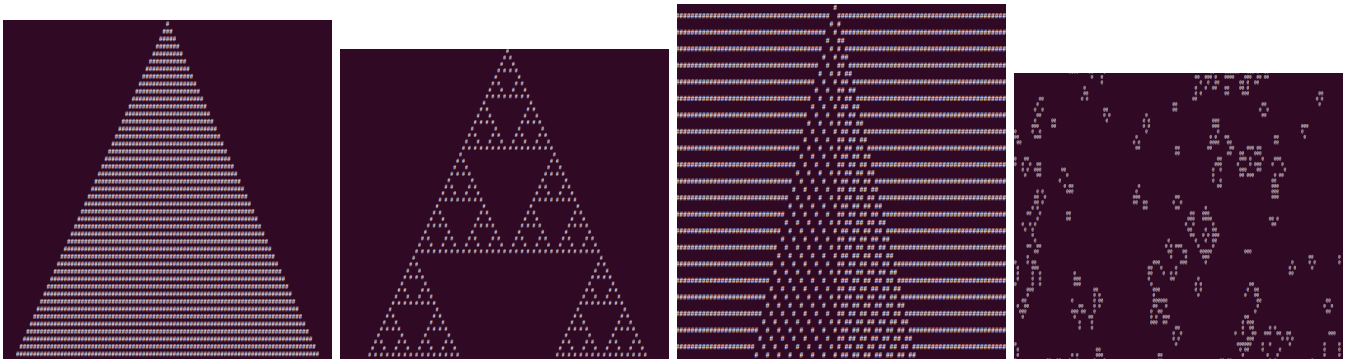


Figure 1: Quelques exemples d'automates cellulaires à coder durant ce TP.

1 Introduction

Les automates cellulaires ont été développés initialement par le mathématicien John von Neumann dans les années 1940 pour modéliser des systèmes biologiques et physiques à l'aide de règles simples. Le concept a été popularisé par John Conway avec son "Jeu de la Vie" dans les années 1970, un exemple de système où des cellules sur une grille évoluent selon des règles fixées basées sur l'état des cellules voisines.

Un automate cellulaire élémentaire est une version simplifiée en une dimension, où chaque cellule (pouvant être active ou inactive) évolue en fonction de l'état de ses deux voisines immédiates. Ces systèmes peuvent produire des motifs complexes et sont utilisés pour étudier des phénomènes dynamiques et les théories du chaos.

Le but de ce TP est de mettre en place un automate cellulaires en utilisant le langage C.

2 Premier programme

Avant de commencer à coder, vous devez vous assurer que votre environnement de développement est prêt.

- Ouvrez un éditeur de texte favori (Visual Studio Code, Sublime Text, ...)
- Copiez et collez le code suivant : Il s'agit d'un simple programme "Hello World" en C.

```c
#include <stdio.h>
#include <stdlib.h>

int main() {
    printf("Hello World!\n");
    return 0;
}
```

- Enregistrez le fichier sous le nom ‘automates.c‘. Assurez-vous de choisir un emplacement facile à retrouver, comme votre dossier de documents ou un dossier spécifique pour ce cours.

2.1 Compilation

Pour compiler le programme, vous allez utiliser un compilateur C. GCC est le plus couramment utilisé sous Linux, tandis que Clang est une alternative populaire. Sous Windows, vous pouvez utiliser MinGW ou Cygwin pour obtenir GCC.

- Ouvrez votre terminal,
- Naviguez jusqu'au répertoire contenant votre fichier,
- Tapez la commande suivante pour compiler votre fichier :

  `gcc automates.c -Wall -o automates`

  Cette commande dit au compilateur GCC de prendre le fichier "automates.c", de le compiler, et de sortir un exécutable nommé "automates" (remplacez par "-o automates.exe" si vous êtes sous Windows). "-Wall" demande au compilateur d'afficher tous les warnings ([W]arnings: [all]). Ce ne sont pas des erreurs, mais ils peuvent être source de mauvais comportement dans l'execution de votre programme.

- Une fois compilé, lancez votre programme :

  `./automates`

  Si tout fonctionne correctement, vous verrez "Hello World!" s'afficher dans votre terminal.

Félicitations, vous êtes maintenant prêts à commencer à travailler sur l'automate cellulaire en utilisant le même processus de compilation pour les programmes plus complexes que nous allons développer par la suite !

3 Création d'un tableau de cellules

Notre automate est un tableau 1D (une liste) de cellules qui peuvent être "vivantes" (1) ou "mortes" (0). Dans un premier temps on va utiliser un tableau "statique" (la taille est connue avant l'execution du programme).

- Créez un tableau d'entiers `cells` de longueur 11.
- Avec une boucle "for", affectez la valeur "0" à toutes les cases du tableau, sauf celle du milieu, qui vaut "1".
- Dans une seconde boucle "for", affichez la valeur de chaque cellule les unes à côté des autres grâce à "printf()".

- Au lieu d'afficher la valeur numérique ("%d"), affichez le caractère "#" ou "_" selon si la cellule est vivante ou morte ("1" ou "0").

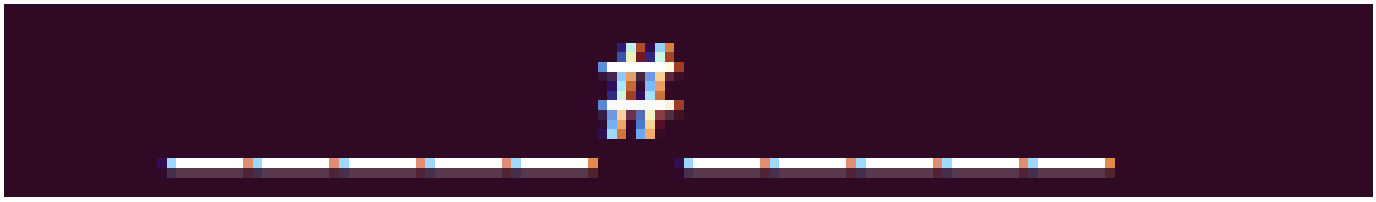


Figure 2: Résultat attendu de la première itération.

3.1 Un tableau dynamique

On souhaiterait pouvoir ajuster la taille de notre tableau pour pouvoir avoir un nombre "variable" de cellules.

- Ajoutez une variable de type entier cosntante (const int) `WIDTH` qui représente maintenant la taille voulue de notre tableau (ex: 11).
- Si vous utilisez cette variable dans la *déclaration* de votre tableau, que se passe-t-il ?
- On va maintenant devoir *allouer dynamiquement* de la mémoire pour notre tableau "cells". Commencez par modifiez la déclaration de votre tableau pour passer de `int cells[11]` à `int* cells`. Que signifie `int*` ?
- Maintenant, on peut allouer la mémoire suffisante pour stocker nos données. On veut un tableau de `WIDTH` cellules, et on sait que chaque cellule est représentée par `sizeof(int)` octets en mémoire. Créez une variable `int size_in_memory = ...` qui contient cette taille nécessaire.
- Allouez la mémoire nécessaire avec la fonction "malloc()". Pour rappel, on utilise cette fonction de cette façon :

  ```
  mon_tableau = (type*) malloc (taille_en_octet);
  ```

- Normalement, le résultat de votre programme est le même qu'avant !

4 Afficher un tableau dans une fonction

- Déplacez les instruction permettant d'afficher les cellules dans une fonction dédiée que vous appellerez "displayCells". Cette fonction ne retourne rien (il y a un type de données pour indiquer "rien") et elle prend en paramètre votre tableau et sa taille.
- Question : Pourquoi doit-on donner la taille du tableau en paramètre ?

5 Lancer l'automate cellulaire

Jusque là nous n'avons fait qu'afficher une ligne, c'est pas super intéressant..! Mettons en place ce principe de "automate".

Un automate fonctionne avec des règles simple : chaque cellule regarde l'état (mort ou vivant) de son voisin de droite, son voisin de gauche, et son propre état. Avec ça, la cellule décide si elle doit être "morte" ou "vivante" au pas de temps suivant.

5.1 Une première règle simple

Commençons par une règle toute simple : si une cellule a son voisin de droite ou son voisin de gauche vivant ou si elle-même est vivante, elle sera vivante à la prochaine étape. Sinon, elle est morte.

On va afficher l'évolution de notre système étape par étape.

- Créez un second tableau `nextStep` de la même taille que `cells`. Il va contenir le résultat de notre automate après une itération.
- Pour chaque cellule de `cells`, on va transférer l'état final de la cellule dans le tableau `nextStep`. Utilisez le pseudo-code suivant :

```
POUR i allant de 1 à WIDTH - 1
    SI cells[i - 1] == 1 OU cells[i] == 1 OU cells[i + 1] == 1 ALORS
        nextStep[i] = 1
    SINON
        nextStep[i] = 0
FIN POUR
```

- Affichez le tableau `cells` avec la fonction `displayCells()`, puis à la ligne en dessous, affichez `nextStep` de la même façon.
- Questions : Que devriez-vous voir ? Pourquoi doit on utiliser un second tableau de cellules ? Quel serait le résultat si on appliquait la transformation des cellules directement dans le tableau `cells` ?
- Créez une fonction `cellRule()` qui prend en paramètre l'état de la cellule de droite, l'état de la cellule courante, et l'état de la cellule de gauche, et qui renvoie "1" si la cellule courante doit devenir vivante, ou "0" sinon. Utilisez cette fonction dans votre boucle "for" à la place du "if/else".

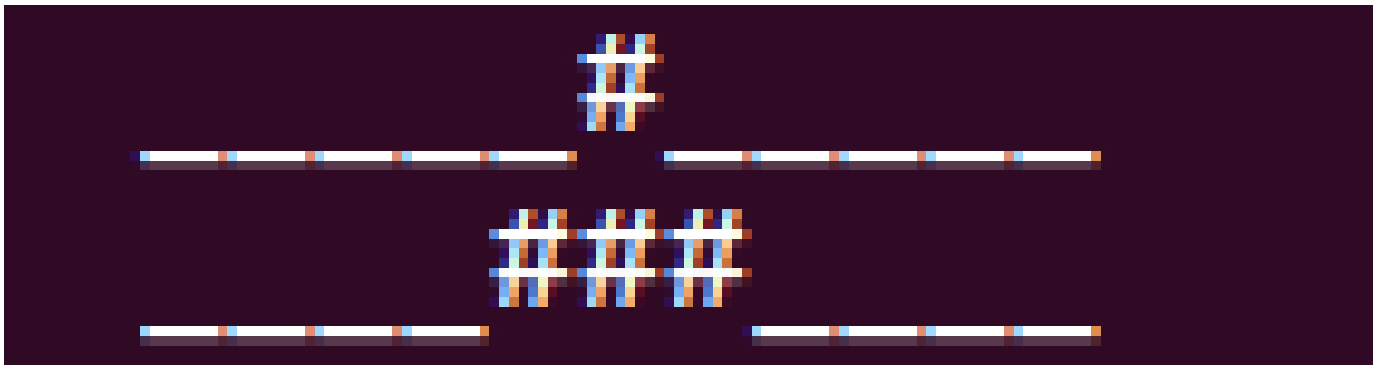


Figure 3: Résultat attendu de la seconde itération.

On a réussi à appliquer une étape de l'automate. Rien de fantastique, alors pourquoi ne pas répéter cette étape ?

- Copiez la valeur de chaque cellule du tableau `nextStep` dans le tableau `cells`.

- De nouveau, appliquez une transformation à chaque cellule en utilisant exactement le même pseudo-code, puis affichez de nouveau le tableau `nextStep` dans une troisième ligne.
- Question : Que devriez-vous voir ?
- Fini les copier-coller, on va maintenant *itérer* sur cet automate à travers une boucle. Suivez ce pseudo-code :

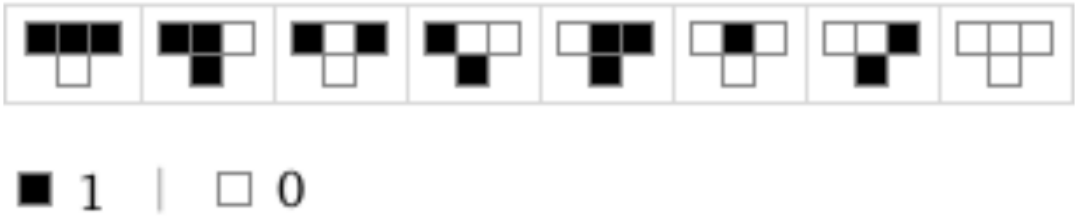
```
cells = un tableau dynamique de taille WIDTH
nextStep = un tableau dynamique de taille WIDTH

POUR iteration allant de 0 à 10
    POUR i allant de 1 à WIDTH - 1
        nextStep[i] = cellRule(cells[i - 1], cells[i], cells[i + 1])
    FIN POUR
    afficher nextStep
    POUR i allant de 0 à WIDTH
        cells[i] = nextStep[i]
    FIN POUR
FIN POUR
```

- Question : Quel est le résultat ?

5.2 Nouvelle règle un peu plus complexe

On va changer le contenu de la fonction `cellRule()` pour appliquer une règle que je trouve très jolie : la règle 90 !

Voici sa définition :



Cette image est une *table de verité* dont les entrées sont représentées sur la ligne du haut, et la sortie sur la ligne du bas.

On va appeler "G" l'état du voisin de gauche, "C" l'état de la cellule courante et "D" l'état de la cellule de droite. Textuellement, voilà ce qu'elle dit :

- Si G, C et D sont vivantes, alors la cellule devient morte.
- Si G et C sont vivantes et D est morte, alors la cellule devient vivante.
- Si G et D sont vivantes, et C est morte, alors la cellule devient morte.
- Si G est vivante, et C et D sont mortes, alors la cellule devient vivante.

- Si C et D sont vivantes, et G est morte, alors la cellule devient vivante.
- Si G et D sont mortes, et C est vivante, alors la cellule devient morte.
- Si G et C sont mortes, et D est vivante, alors la cellule devient vivante.
- Enfin, si G, C et D sont mortes, alors la cellule devient morte.

Modifiez votre fonction `cellRule` pour mettre en application cette nouvelle règle !

Allez à la page de présentation de cette règle sur WolframAlpha (`https://www.wolframalpha.com/input/?i=rule+90`). Allez voir la section "Algebraic form" ou "Boolean form". Ces notations sont des versions beaucoup plus condensées de notre table de vérités! Remplacez le contenu de la fonction `cellRule` par l'une de ces formes (l'opérateur XOR est représenté par un accent circonflexe "∧" en C).

Beaucoup plus simple, non ? Maintenant que vous savez faire, allez voir d'autres règles pour les implémenter ! Essayez les règles 30, 57 et 73.

6 Aller plus loin

À partir de ce même principe d'automate cellulaires, de nombreuses portes s'ouvrent ! En voici notamment 2 très intéressantes à explorer :

- Cherchez un moyen pour avoir un automate à 3 états (chaque cellule peut avoir la valeur 0, 1 ou 2). Affichez les cellules avec des symboles comme " ", "+", "#".

Figure 4: Règle 30 avec 3 états.

- Coder un "Jeu de la vie" : le principe reste à peu près le même mais en 2D. Dessinez une grille 2D de dimension 30x30 dont chaque cellule peut être morte (0) ou vivante (1). À chaque itération et pour chaque cellule, verifiez ces 3 règles :

– Si la cellule courante est vivante et qu'elle a 2 ou 3 voisines vivantes, alors elle reste vivante.
– Si la cellule courante est morte et qu'elle a exactement 3 voisines vivantes, alors elle devient vivante.
– Dans tous les autres cas, elle meurt.

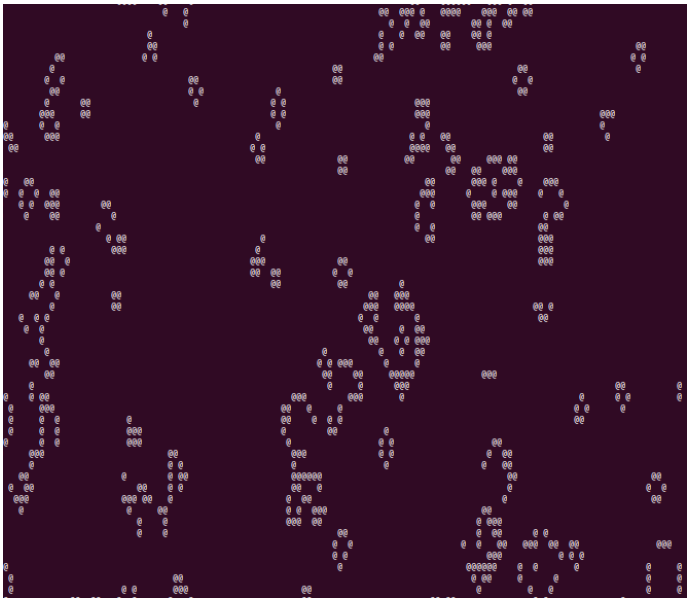


Figure 5: Une itération du Jeu de la Vie de Conway