Dodeman Léna
Leger Bertrand

PROJET INFORMATIQUE
TP4

IMPLÉMENTATION D'UN SIMPLE AUTOMATE DE CELLULE

Table des matières

Table des illustrations

INTRODUCTION :

L’objectif de ce TP est de créer un jeu de la vie au moyen d’un automate de cellule en 2D.

Tout d’abord, le jeu de la vie est une simulation de prolifération de cellules. C’est un jeu qui n’a pas besoin de joueur. Il est représenté par une grille de cellules, chaque cellule a deux états possibles :”morte” ou “vivante”. Elles évoluent dans le temps en fonction de règles qui dépendent du nombre de voisins qu’elles ont. Une cellule est considérée comme ayant au maximum 8 voisins (c’est-à-dire que chaque “case” de la grille a possiblement 8 “cases” voisines).

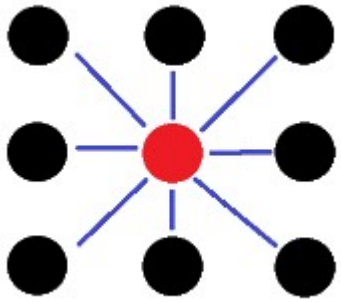


Figure 1 : Schéma voisins

Lorsque que l’on passe d’un instant t à un autre instant t+1 alors :

- Si la cellule est “vivante”:
  - chaque cellule avec au plus 1 voisin meurt (on considérera cela comme mort de solitude)
  - chaque cellule avec 4 voisins ou plus meurt (on considérera cela comme mort par surpopulation)
  - Enfin, chaque cellule avec 2 ou 3 voisins survivent.

- Si la cellule est “morte”:
  - Chaque cellule avec 3 voisins devient vivante.

Pour créer ce jeu de la vie nous avons besoin d’un automate de cellules en 2D. Ce type d’automate a été inventé par John Conway, un mathématicien renommé de Cambridge, et il consiste en une grille de cellules qui ont chacune un nombre fini d’états possibles et dont le prochain état est conditionné par l’état précédent des cellules adjacentes. Il faut ainsi que les règles qui régissent l’automate soient appliquées à toutes les cellules. Conway a ainsi voulu créer un jeu qui agirait de façon imprévisible.

Afin de pouvoir étendre le principe de voisins aux bord de la grille on va considérer que “l’espace” est plié comme un Torus. C'est-à-dire que l'on va plier la grille de façon à ce que les bord soient collés. Ainsi nous allons avoir une expansion de la grille.

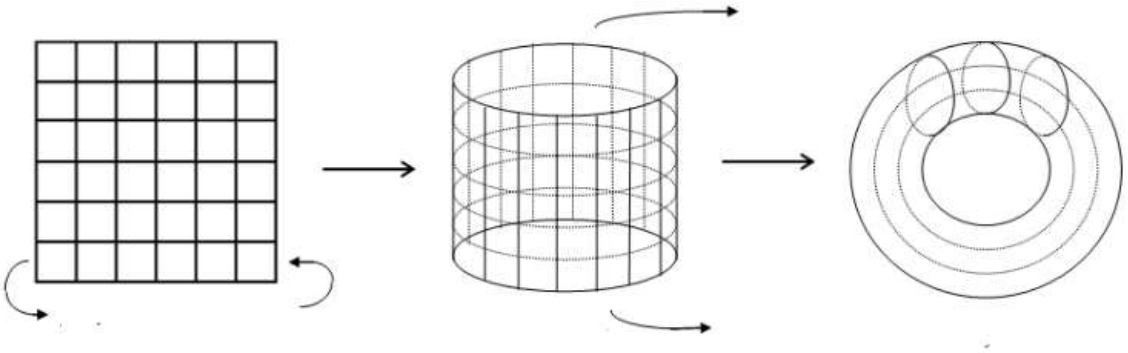


Figure 2 : De non torus à univers torus

Implémentation du jeu de la vie :

Pour cette première question nous allons implémenter le jeu de la vie dont on a expliqué le fonctionnement plus haut. Nous allons faire deux fonctions, l'une où l'on considérera l'espace sans Torus et l'autre où l'on considérera l'espace avec.

Voisinage sans Torus :

Pour la fonction sans torus, nous prenons comme paramètres un tableau avec le glider dedans (notre jeu de la vie à l'instant t) ainsi que sa taille. On initialise tout d'abord un tableau à zéro. Ce tableau sera notre jeu de la vie à l'instant t+1. On va ensuite parcourir chaque case de la grille (à l'instant t) et pour chaque itération on comptera le nombre de voisins vivants grâce à un compteur de vies en fonction de où se situe la case que l'on évalue. Dans ces mêmes itérations en fonction de la valeur du compteur de vie (soit le nombre de voisins) on remplira la case correspondante de la grille "t+1". Enfin on finit notre fonction par copier la grille "t+1" dans la "t" pour que l'on puisse avoir possiblement un "t+2" si l'on appelle la fonction + d'une fois.

Les voisins dans un espace sans Torus :

- Si la case que l'on évalue est au coin de la grille :

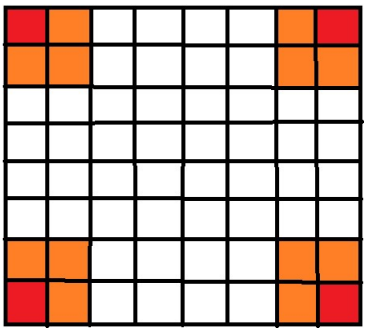


Figure 3 : Case en coin sans torus

Si notre case en grille[i][j] (en rouge) est un coin, nous ne devons évaluer que les 3 voisins (en orange).
D'où dans notre fonction, ce calcul pour le nombre de voisins si la case est celle en haut à gauche par exemple :
compteurDeVies = grille[i][j+1] + grille[i+1][j] + grille[i+1][j+1]

- Si la case que l'on évalue est au bord de la grille et n'est pas un coin :

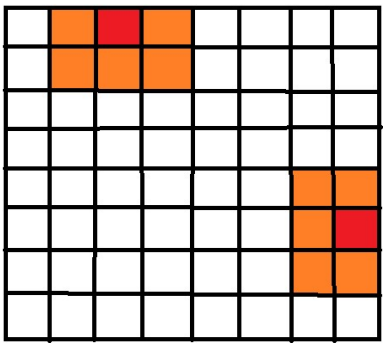


Figure 4 : Case au bord sans torus

Si notre case en grille[i][j] (en rouge) est au bord, alors les voisins à prendre en compte sont les oranges, ici il y en a 5. Notre fonction calcule ainsi le nombre de voisins par exemple si la case est au bord supérieur (sur la première ligne de la grille) :
compteurDeVies = grille[i][j-1] + grille [i+1][j-1] + grille[i+1][j] + grille [i+1][j+1] + grille[i][j+1]

- Si la case que l'on évalue n'est ni dans un coin ni aux extrémités de la grille :

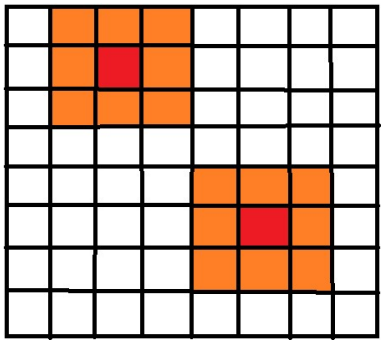


Figure 5 : Case dans la grille sans torus

Si notre case en grille[i][j] (en rouge) est "dans" la grille et ne fait pas partie des 2 cas spéciaux du dessus, elle aura 8 voisins (en orange). Notre faction calcule alors ses voisins comme suit :
compteurDeVies = grille[i-1][j-1] + grille[i-1][j] + grille[i-1][j+1] + grille[i][j-1] + grille[i][j+1] + grille[i+1][j-1] + grille[i+1][j] + grille[i+1][j+1]

Voisinage avec Torus :

Pour la fonction avec torus on fera exactement la même chose que sans torus excepté pour la partie sur le comptage des voisins. En effet puisque l'on change l'espace, les cases aux différents bords de la grille auront plus de voisins possibles et donc on aura plus de résultats par rapport à l'espace sans torus. A noter que nous pouvions aussi implémenter cette fonction avec un modulo en fonction de la taille de notre grille.

Les voisins dans un espace avec Torus :

- Si la case que l'on évalue est au coin de la grille :

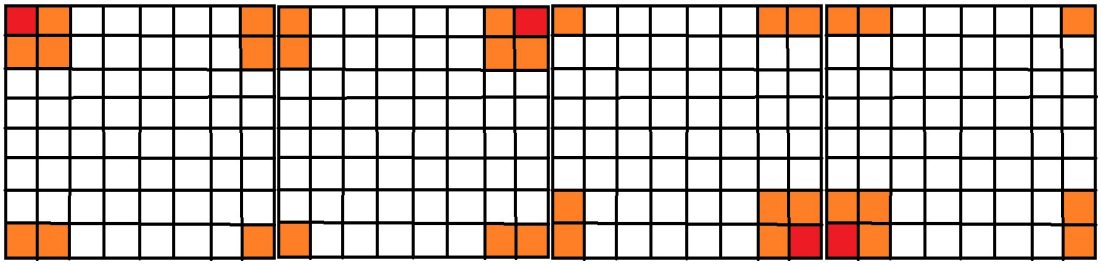


Figure 6 : Différents coins dans un univers avec torus

Lorsque que la case en grille[i][j] est dans l'un de ces 4 cas de figure, nous devons prendre en compte les voisins qui continuent sur les autres côtés de la grille. La case aura toujours 8 voisins.
Ainsi pour le premier cas nous avons ce calcul pour le nombre de voisins :
compteurDeVies = grille[i][j+1] + grille[i+1][j] + grille[i+1][j+1] + grille[taille-1][j] + grille[taille-1][j+1] + grille[i][taille-1] + grille[i+1][taille-1] + grille[taille-1][taille-1]

- Si la case que l'on évalue est au bord de la grille et n'est pas un coin :

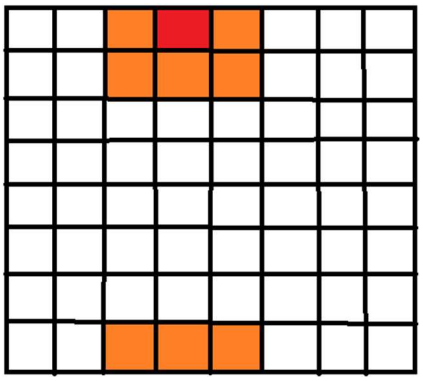
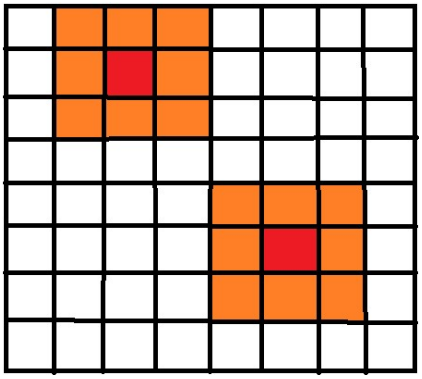


Figure 7 : Case au bord dans un univers avec torus

Si notre case en grille[i][j] est au rebord, alors elle a toujours 8 voisins mais les 3 qu'il manquait à la cellule de l'univers sans Torus se situent en face dans l'univers avec Torus.
Ainsi dans ce cas de figure nous avons ce calcul dans notre fonction :
compteurDeVies = grille[i][j-1] + grille [i+1][j-1] + grille[i+1][j] + grille [i+1][j+1] + grille[i][j+1] + grille[taille-1][j] + grille[taille-1][j-1] + grille[taille-1][j+1]

- Si la case que l'on évalue n'est ni dans un coin ni aux extrémités de la grille :



Dans le cas où la case en grille[i][j] est au centre, rien ne change par rapport à l'univers sans torus.
Le calcul est le même et la case a toujours 8 voisins à prendre en compte.

Figure 8 : Case à l'intérieur de la grille

Ainsi dans notre programme, le compteur de vies est réinitialisé à chaque début d'étape, il y a ensuite une succession de conditions (if, else if) qui évalue le nombre de voisins à chaque case de notre grille.

Remplissage de la grille t+1

Nous avons ensuite une partie qui remplit une grille à t+1 en fonction du nombre de voisins que nous avons calculé plus haut.

```
1.                      if(grille[i][j] == 1)
2.                      {
3.                              if(compteurDeVies == 1 || compteurDeVies == 0 ||
compteurDeVies >= 4)
4.                              {
5.                                      grillePlus1[i][j] = 0;
6.                              }
7.                              else if(compteurDeVies == 2 || compteurDeVies == 3)
8.                              {
9.                                      grillePlus1[i][j] = 1;
10.                             }
11.                     }
12.
13.                     //Si la case est vide
14.                     if(grille[i][j] == 0)
15.                     {
16.                             if(compteurDeVies == 3)
17.                             {
18.                                     grillePlus1[i][j] = 1;
19.                             }
20.                     }
```

Ici les "if" et "else if" qui se situent aux lignes 3,7 et 16 correspondent aux règles du jeu que nous avons spécifiées plus haut.
Ainsi si à la grille au temps t, nous avons une cellule vivante, sa valeur est égale à 1. En fonction de sa valeur du compteur de voisins, elle mourra ou restera en vie et sa nouvelle

valeur 0 (morte) ou 1 (vivante) sera sauvegardée dans le tableau grillePlus1. De même si la cellule en temps t est morte et que sa valeur est donc de 0.

Comment avoir notre nouvelle grille pour la suite du jeu ?

Pour que nous puissions réaliser une étape suivante de notre jeu, il faut utiliser la grille en temps t+1 et évaluer à son tour ses voisins et donc son état en temps t+2. Pour se faire nous mettons les valeurs de notre nouvelle grille maintenant bien initialisée dans l'ancienne grille qui ne nous est plus utile. On repasse donc du tableau grillePlus1 au tableau grille pour que les opérations de comptage de voisins puissent continuer.
Le fonction qui réalise ce transfert est la suivante :

Transfert des deux tableaux :

```
for (i=0; i<taille; i++)
{
        for (j=0; j<taille; j++)
        {
            grille[i][j] = grillePlus1[i][j];
        }
}
```

Réalisation d'un nombre défini d'expériences

Maintenant que nous avons nos deux fonctions fonctionnelles il nous faut des fonctions qui permettent de faire plusieurs itérations de ces grilles. En effet, dans le but de pouvoir faire des observations il faut que l'utilisateur puisse choisir le nombre de "t" qu'il veut étudier. Pour l'expérience sans torus c'est une simple boucle de la fonction mais pour l'expérience avec torus nous avons d'abord besoin de créer la grille et pour se faire nous faisons une randomisation.

Nous avons deux fonctions qui répète le jeu à un nombre d'itérations choisies :

Fonction pour l'espace sans torus :

```
void iterationsSansTorus(int taille, int
grille[][taille], int iterations)
{
        int i;

        for(i = 0 ; i < iterations ; i++)
        {
                jeuSansTorus(taille, grille);
        }
}
```

Fonction pour l'espace avec torus :

```
void iterationsTorus(int taille, int
grille[][taille], int iterations)
{
        int i;

        for(i = 0 ; i < iterations ; i++)
        {
                jeuAvecTorus(taille, grille);
        }
}
```

C’est dans la boucle for que le jeu se répètera, ce qui créera nos différentes étapes pendant la réalisation du jeu de la vie.

Pour un jeu de la vie avec Torus nous avons une fonction nommée initTabAvecTorus ()qui initialise une grille de façon aléatoire. Elle va donner vie à certaines cellules au hasard afin d’avoir une grille de départ.

```
1. void initTabAvecTorus(int taille, int iterations)
2. {
3.       srand( time( NULL ) );
4.       int i,j;
5.       int tab[taille][taille];
6.
7.       for(i=0 ; i < taille ; i++)
8.       {
9.                for(j=0 ; j < taille ; j++)
10.               {
11.                       tab[i][j] = rand() %2;
12.               }
13.      }
14.
15.      printf("Votre grille de départ :\n\n");
16.      afficheTab(taille, tab);
17.      printf("Réalisation du jeu de la vie :\n\n");
18.      iterationsTorus(taille, tab, iterations);
19. }
```

Dans cette fonction, de la ligne 7 à la ligne 13 se réalise la distribution aléatoire de cellules “vivantes” avec un rand qui sortira 0 ou 1, 0 étant notre valeur pour une cellule morte et 1 pour une cellule vivante. Nous utilisons ensuite la fonction d’itérations du jeu “iterationsTorus()” qui va réaliser le jeu à un nombre d’étapes pris en paramètre de la fonction initTabAvecTorus.

Maintenant que nous avons un code fonctionnel nous pouvons commencer à observer différentes réalisations du jeu et ce que l’on obtient selon notre initialisation de départ.

Observations et réalisations du jeu de la vie

Le jeu de la vie peut être utilisé avec une multitude de schémas de base qui évolueront différemment. Il existe des schémas qui restent stables, d’autres qui évoluent pour créer une forme particulière, d’autres encore qui vont même créer des schémas eux-mêmes.

Nous avons pu tester différentes formes dans notre programme pour observer leurs évolutions. Afin d’avoir un rendu plus agréable esthétiquement et efficace, nous avons utilisé ce site internet afin de générer différentes formes. Certaines ont été testées à la main dans notre programme pour vérifier un fonctionnement correspondant à celui du site.

Glider du sujet

Nous avons observé l'évolution du glider dans un espace sans Torus puis dans un espace avec Torus.

Figure 9 : Glider

Dans notre univers sans torus et dans une grille de taille 10, nous voyons qu'à partir de l'étape n°27, les cellules ne vont plus bouger. Le schéma va rester stable à partir de ce moment, il devient un carré de 2x2 (qui fait d'ailleurs partis de l'une des formes stables que nous verrons ci-dessous).

Figure 10 : Fin du glider dans un univers sans torus

Dans le cas d'un univers avec torus, le glider va continuer indéfiniment en suivant un cycle de 4 formes pour revenir à la première.

Figure 11 : Différentes étapes du glider

Jeu en aléatoire

Tout d'abord lorsque l'on teste le jeu de la vie avec une grille aléatoire, nous allons observer des changements à chaque étape. Parmi les mouvements observés, nous pouvons repérer des comportements répétitifs qui finissent par rendre la grille stable et "bloquée" à une certaine itération.

Exemple :

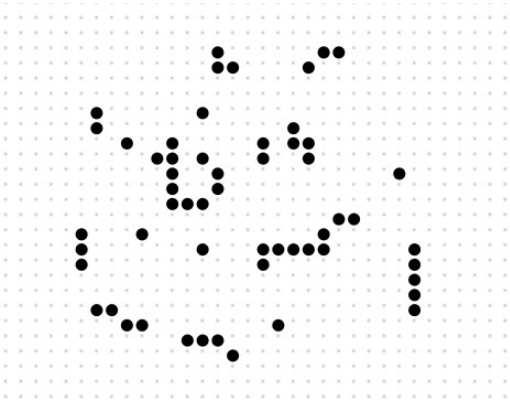


Figure 12 : Grille initialisée aléatoirement

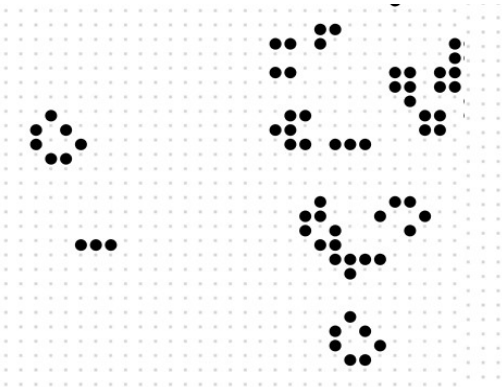


Figure 13 : Grille pendant la réalisation du jeu

Nous pouvons déjà observer deux formes stables à gauche, et le reste qui continue d'évoluer dans la grille.

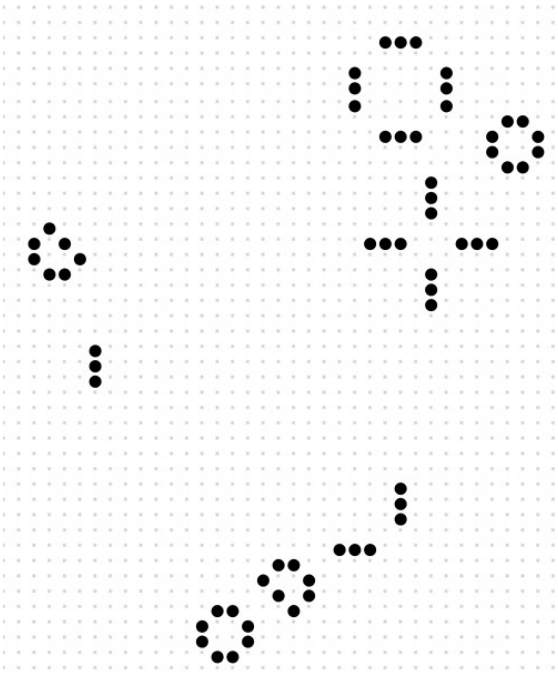


Figure 14 : Grille à la fin du jeu, devenue stable

Finalement, après plusieurs itérations notre jeu va se stabiliser sous cette forme. Certaines des petites formes continuent de bouger, mais elles ne se déplacent plus dans la grille. Nous pouvons alors identifier des formes “stables” qui ne se déplacent pas dans notre grille même si les cellules qui la composent peuvent encore évoluer. Certaines d’entres elles sont même répétitives.

Formes stables

Nous pouvons observer des formes qui n’évolueront jamais, sauf si perturbées par d’autres générations annexes. Une fois initialisées, elles restent stables et ne sont pas modifiées, n’importe le nombre d’itérations effectuées.

Voici 5 exemples pour ces formes :

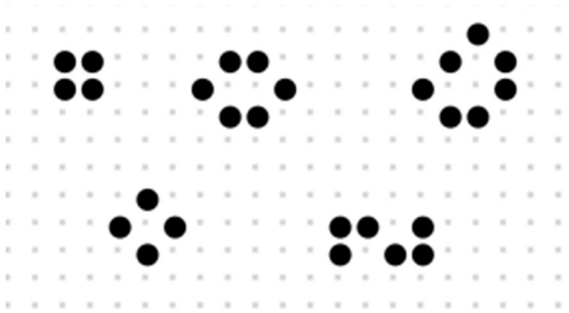


Figure 15 : Exemples de formes stables

Formes répétitives

Nous pouvons observer des formes qui vont évoluer entre plusieurs états, mais qui reviendront toujours à leur état initial. Elles vont répéter une alternance de schéma de la même manière encore et encore.

Quelque exemples :

Figure 16 : Blinker

Figure 17 : Octagone

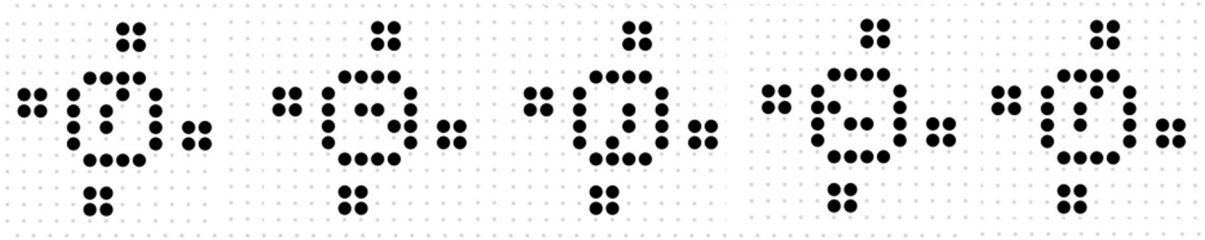


Figure 18 : Clock

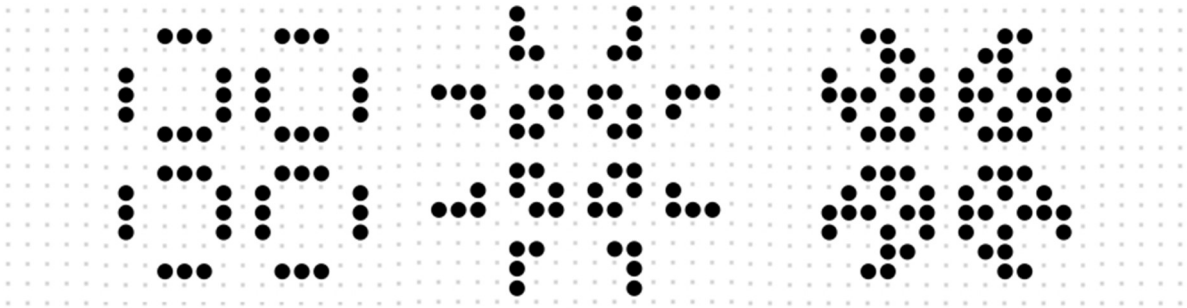


Figure 19 : Pulsar

Ainsi nous pouvons réaliser différents “dessins”, du plus simple au plus complexe. Parmi ces schémas complexes, certains d’entre eux peuvent aussi “dessiner”.

Formes génératrices

Certaines des formes que nous observons peuvent même devenir génératrices d’autres formes. L’évolution des cellules va créer tout un schéma nouveau à partir de l’initialisation de départ.

Voici un exemple connu :

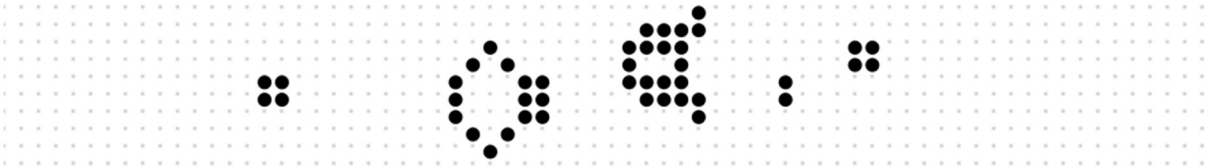


Figure 20 : Gosper Glider Gun

Cette initialisation est le Gosper Glider Gun. Après quelques itérations, le mouvement de ses cellules va créer comme des objets qu’il tire sous lui comme on peut le voir ci-dessous. Les

“blocs” de cellules se déplacent entre les 2 petits carrés, ce qui crée à l’infini le glider vu dans le sujet.

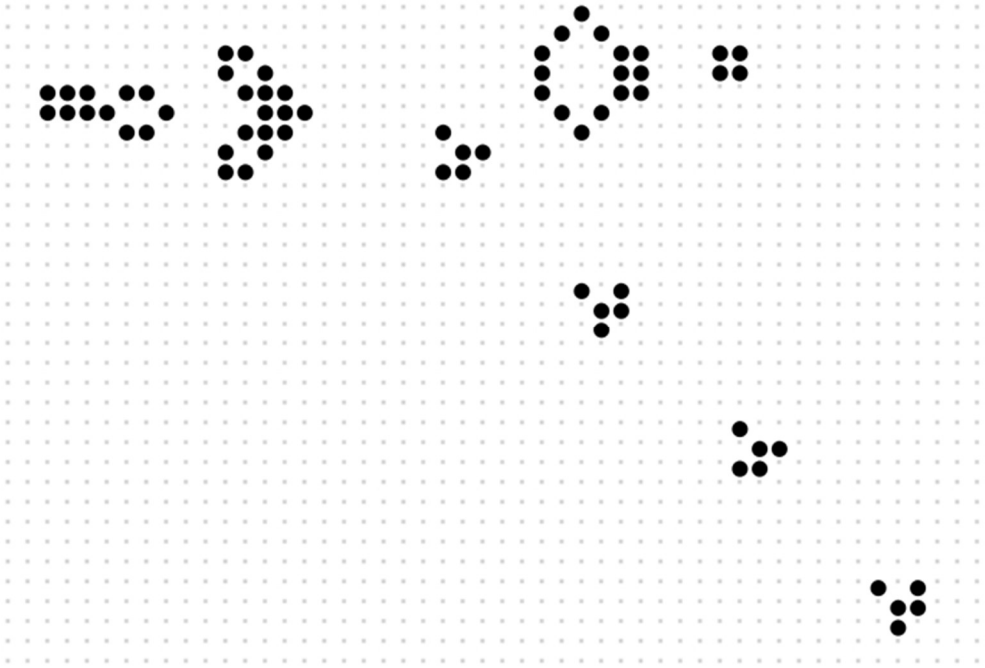


Figure 21 : Gosper Glider Gun en action

Utilisations du principe d’automate cellulaire

Aujourd’hui il existe des simulations qui se basent sur le même fonctionnement que le jeu de la vie. Des modèles d’activité sont ainsi développés afin d’évaluer le comportement possible d’un élément. Plusieurs études sont réalisées en suivant l’évolution d’un automate cellulaire.

- Nous avons par exemple une étude réalisée sur le développement de maladies au sein de notre tube digestif, ici.
- Ici, une étude qui modélise le développement de la moisissure en fonction des conditions météorologiques.
- Ou encore une simulation de la propagation d’un feu de forêt comme vu dans le cours.

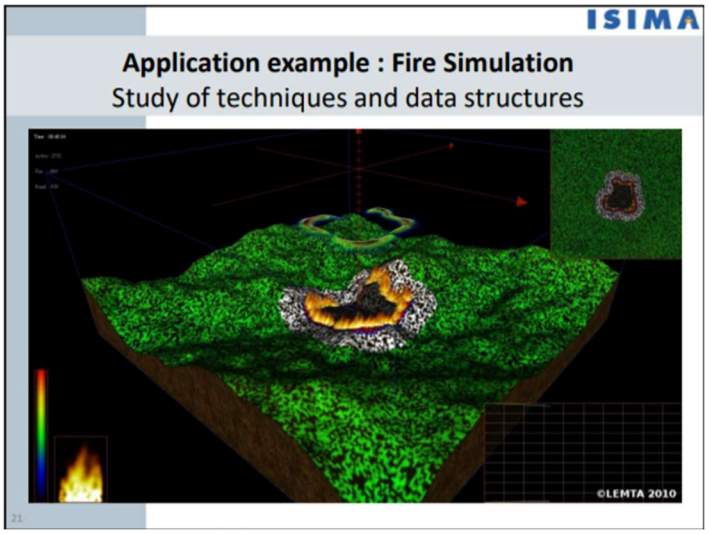


Figure 22 : Slide du cours, propagation du feu

Nous savons aussi qu'il est possible de construire un ordinateur sur ce principe de la Vie sous la forme d'une machine de Turing. Ainsi Paul Rendell a réussi à créer le premier ordinateur fonctionnel en 2000 avec des générations de structures du jeu de la Vie. Cela a servi à pouvoir étudier les ordinateurs d'un point de vue plus mathématique et abstrait.

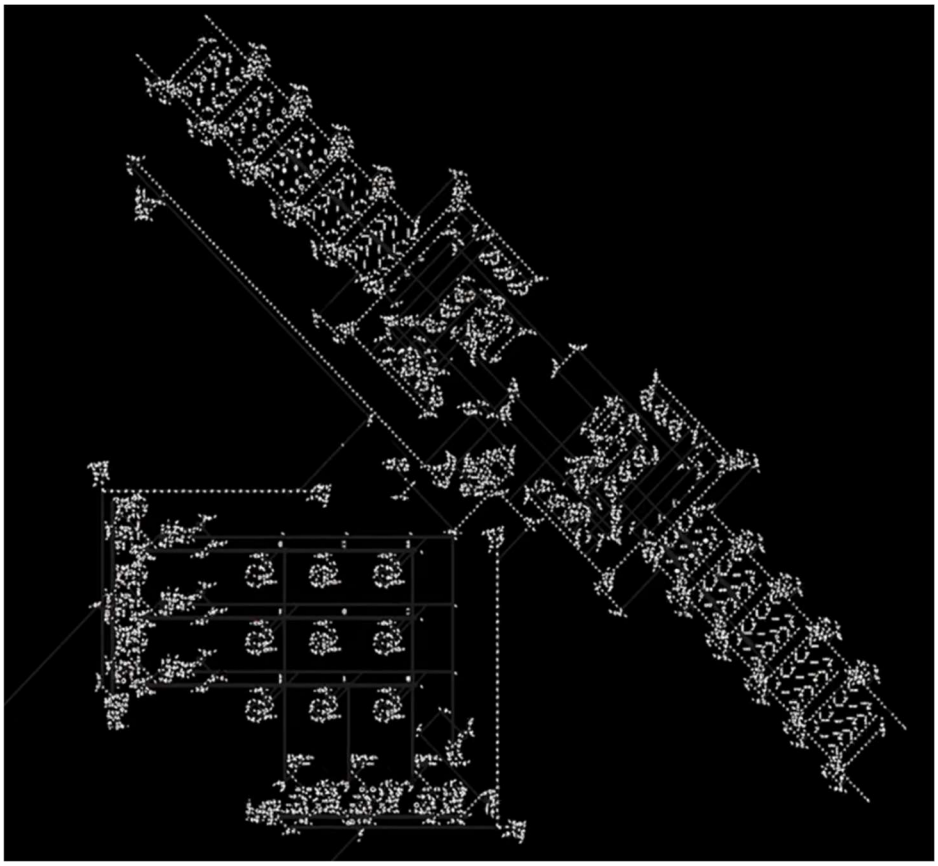


Figure 23 : L'ordinateur de Paul Rendell

Plus récemment, en 2016, Nicolas Loizeau a construit un ordinateur programmable sur 8 bits. C'est une version plus moderne d'un ordinateur, comparé à la version de Rendell qui se rapproche davantage d'une machine de Turing. Il peut ainsi, par exemple, calculer les valeurs de la suite de Fibonacci.

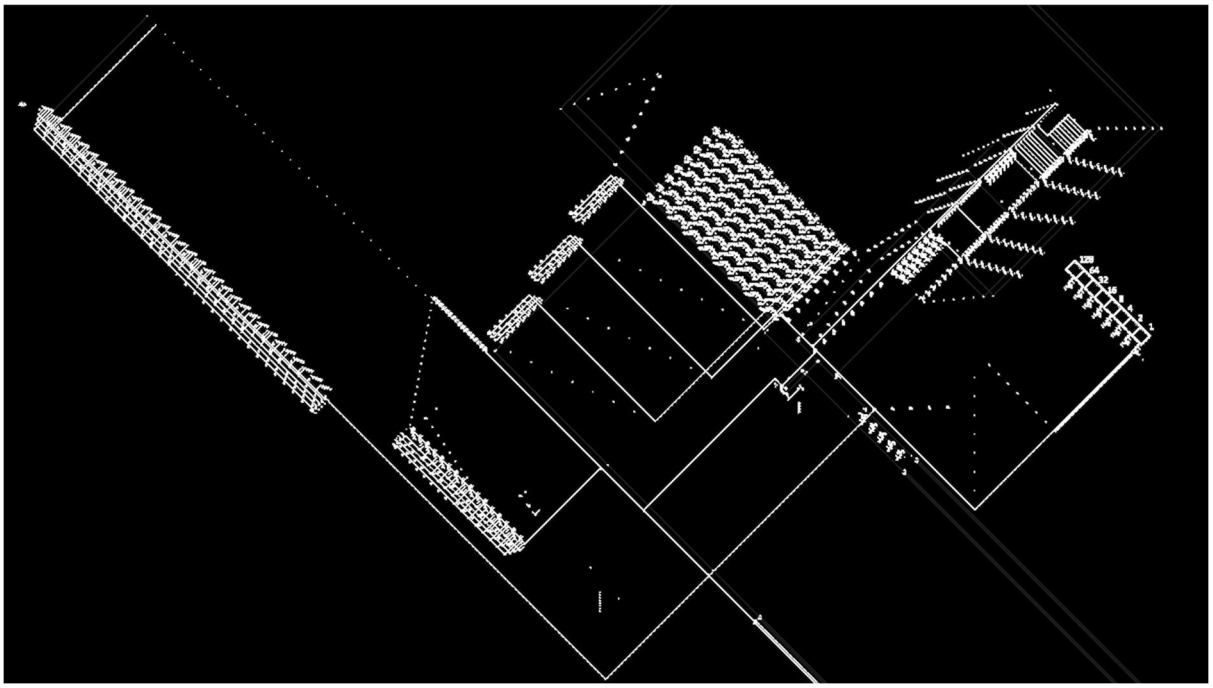


Figure 24 : L'ordinateur de Loizeau

Pour aller plus loin, le jeu de la vie peut être implémenté par le jeu de la vie lui-même. En utilisant des formes spécifiques, ce jeu a été recréé et fonctionne grâce à lui-même. L'ensemble des petites cases créent des cases encore plus grandes, qui créent des formes, qui évoluent à leur tour, et ainsi de suite.

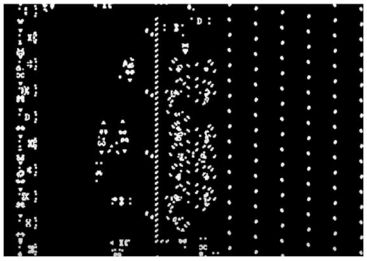


Figure 26 : Zoom sur le bord d'une "grande" case

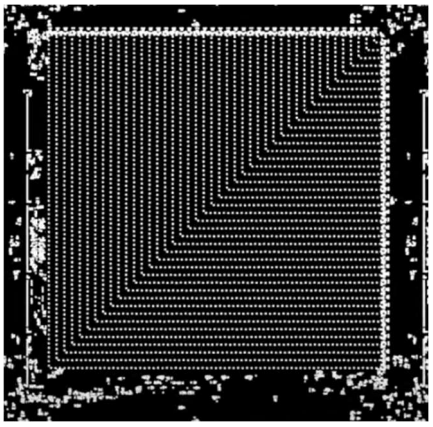


Figure 25 : Case créée par des structures du jeu

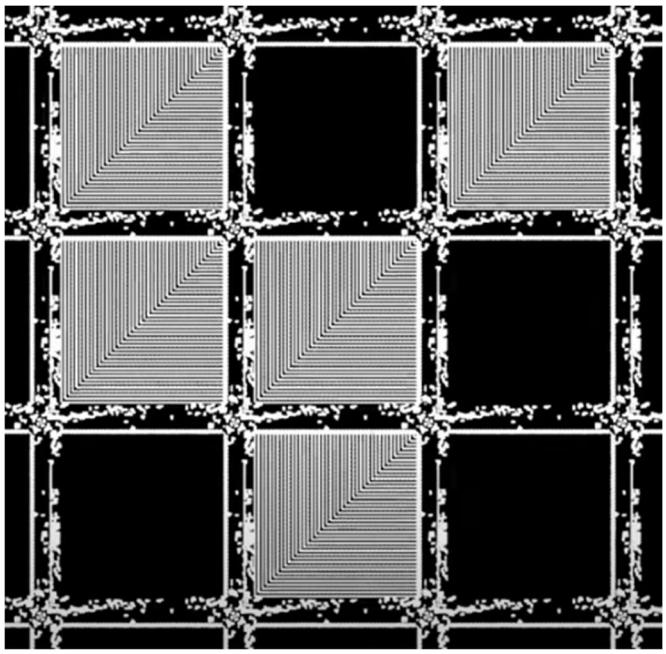


Figure 27 : Le jeu de la vie dans le jeu de la vie

Conclusion

Le jeu de la vie paraît si simple et pourtant il permet de générer des ensembles très complexes. Nous avons vu comment nous pouvions implémenter ce jeu simplement en langage C. Nous avons pu comprendre qu'avec différentes initialisations, le jeu avait un comportement tout aussi différent à chaque fois ce qui nous as montré que l'évolution dépend de deux choses, en tout cas dans cette situation, la situation d'origine (on a utilisé dans ce TP la forme du "glider") ainsi que les règles régissant la mort et la vie des cellules. Nous pouvons imaginer qu'avec une grille assez grande, un certain paterne d'initialisation ainsi qu'une infinité de règles nous simulerions l'humanité.

Références

1. https://www.youtube.com/watch?v=Kk2MH9O4pXY
2. Le cours de Monsieur David Hill