Un jeu de la vie amélioré

Denis Oddoux

1^{er} juin 1999

Résumé

In this report, I am explaining how I implemented a life game, similar to Conway's, on the Pamette. Beginning with the logic I used, I will present some rules that can be chosen and what behaviour they tend to have. I will explain how to use such a design and what can be done to improve it.

Introduction

Il s'agit d'implémenter en hardware une variante de jeu de la vie de Conway. On considère un ensemble de points dans un état initial aléatoire, et on les fait évoluer selon une loi déterminée à l'avance, donnant le nouvel état d'un point à partir de celui de ses voisins immédiats.

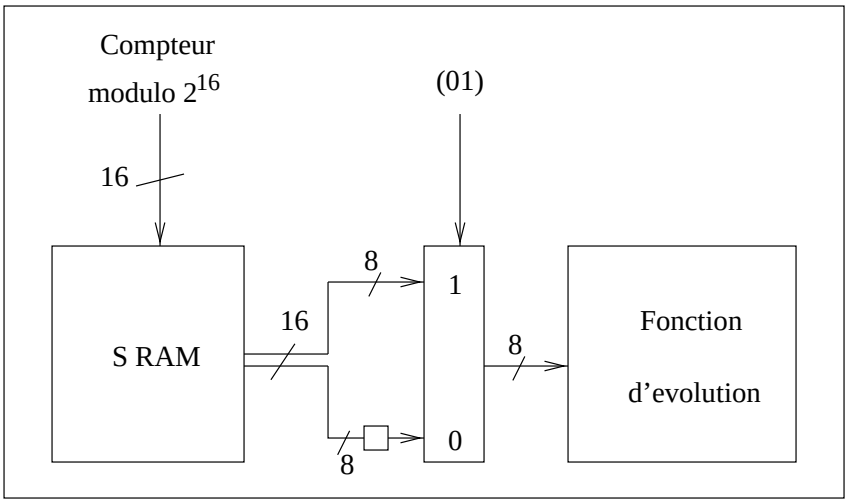
1 Accès aux données

On stocke l'état de tous les points dans la SRAM. Un point est codé sur 4 bits, et on utilise une résolution d'écran de 512×512 pixels, en exploitant ainsi la totalité de la place disponible sur la SRAM : 2^{16} mots sur 16 bits.

À chaque cycle, de façon alternative on lit ou on écrit 16 bits sur la SRAM, et on fait en sorte de voir en permance 8 bits à chaque cycle, en entrée comme en sortie, pour le calcul de la fonction d'évolution. Il faut donc faire évoluer 2 pixels par cycle.

Un grand registre à décalage sur 8 bits permet de stocker la valeur des pixels de 2 lignes consécutives, plus les 4 pixels suivants : on possède alors exactement l'état des 8 voisins respectifs de 2 points donnés.

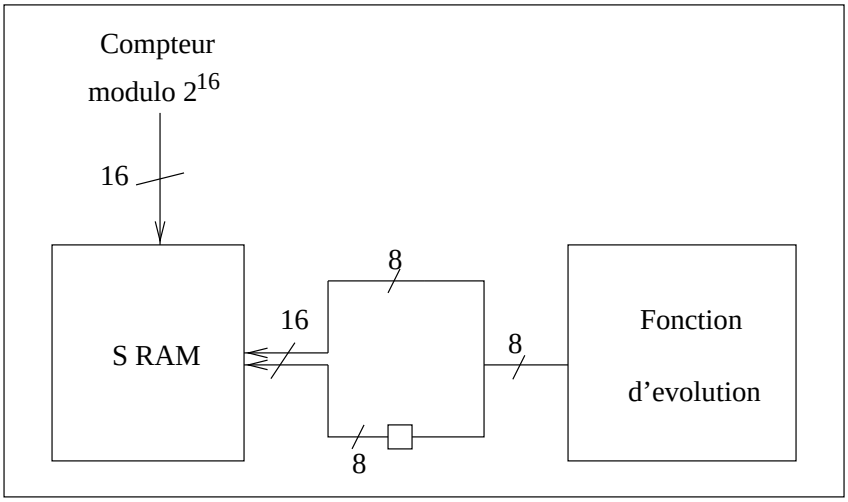
1.1 Lecture



Un compteur modulo 2^{16} donne l'adresse de lecture. L'écriture commençant à l'adresse 0, il faut initialiser ce compteur à l'adresse qui correspond.

La lecture se fait un cycle sur deux, un multiplexeur et un registre sur 8 bits homogénéisent l'entrée : 8 bits par cycle.

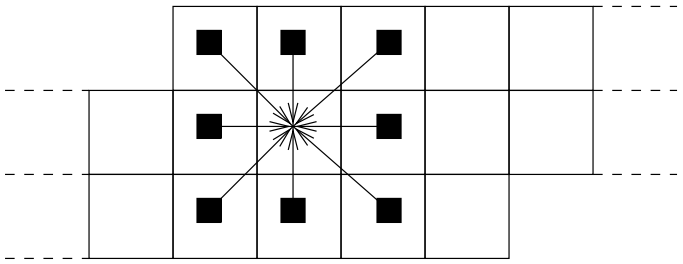
1.2 Écriture



Un compteur modulo 2^{16} donne l'adresse d'écriture, il est initialisé à 0.

L'écriture se fait un cycle sur deux, un registre sur 8 bits permet d'écrire 16 bits à la fois.

1.3 Registre à décalage



Evolution du premier pixel.
Celui qui se situe a sa droite utilise ses 8 propres voisins

L'évolution d'un pixel est déterminée par l'état des 8 pixels qui l'entourent et par le sien. Le stockage de 2 lignes plus 4 pixels fournit les données nécessaires à l'évolution des deux pixels centraux.

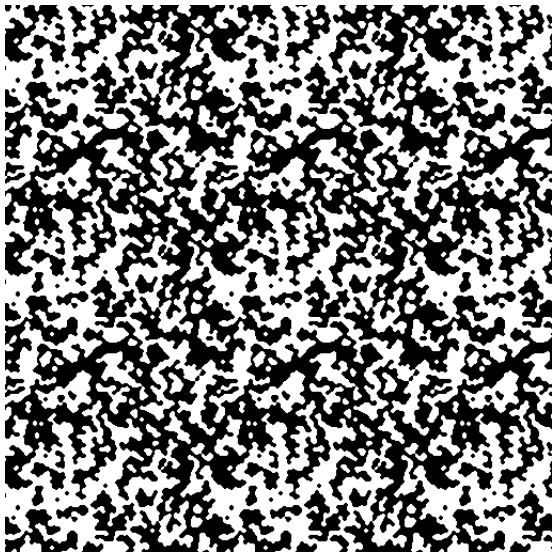
Ce sont ces deux pixels qui sont envoyés en écriture dans la SRAM.

2 Fonctions d'évolution

L'écran initial est constitué de points dont la couleur a été aléatoirement choisie, ce choix étant fait par un logiciel.

On peut imaginer de nombreuses fonctions d'évolution, voici celles que j'ai implémenté sur la Pamette.

2.1 Évolution déterministe (bicolore)



Le pixel que l'on considère prend la couleur majoritaire parmi celles des pixels voisins, lui inclus : il existe 5 voisins de même couleur, et c'est celle-ci que l'on choisit.

Le système évolue, mais arrive très rapidement dans un état bloquant, c'est-à dire une situation où aucun point ne peut plus évoluer (cas de la figure ci-dessus).

2.2 Évolution non détermisiste

Cette fois-ci, on possède un nombre quelconque de couleurs (16 au maximum, un pixel étant codé ici sur 4 bits). La loi d'évolution est alors différente :

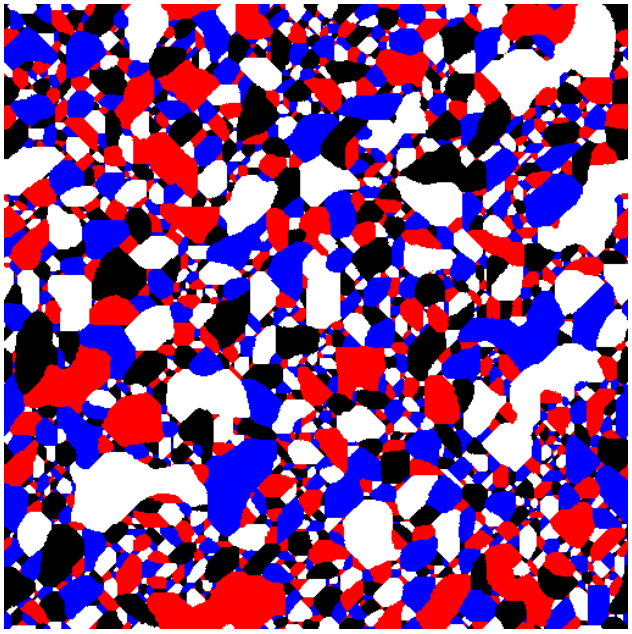
On choisit de maniére aléatoire (en fait, pseudo-aléatoire) une couleur donnée. Ensuite, on compte les voisins du point (en incluant le point lui-même) qui sont de cette couleur. Si ce nombre dépasse un seuil fixé, le pixel prend cette couleur.

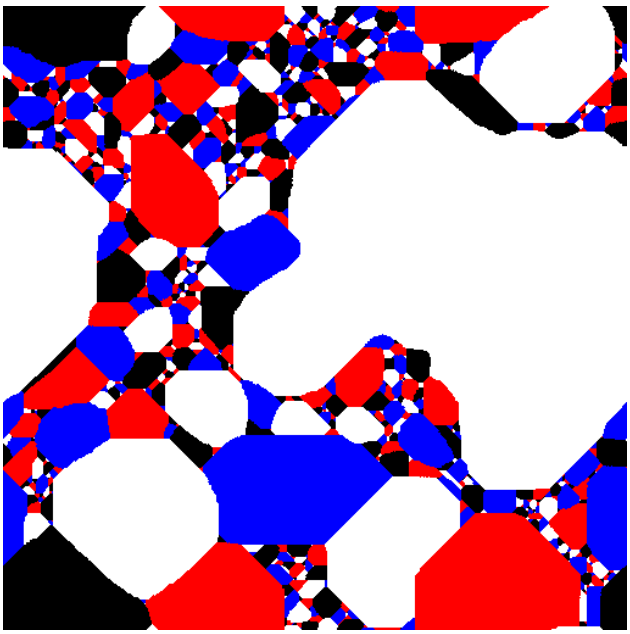
2.2.1 seuil supérieur à 4

On obtient un comportement similaire au precedent. L'interêt est limité, car il est rare que 5 voisins aient la même couleur, dès que le nombre de couleurs dépasse 4.

2.2.2 seuil de 4

Les points tendent à se regrouper par couleur, chaque ensemble prenant au fur et à mesure une forme convexe. On obtient au bout d'un moment une position stable, avec des petits comme des gros amas colorés.

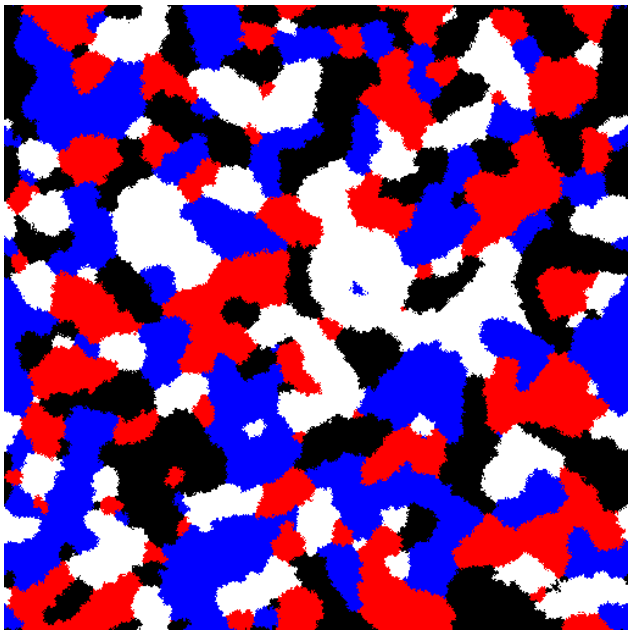


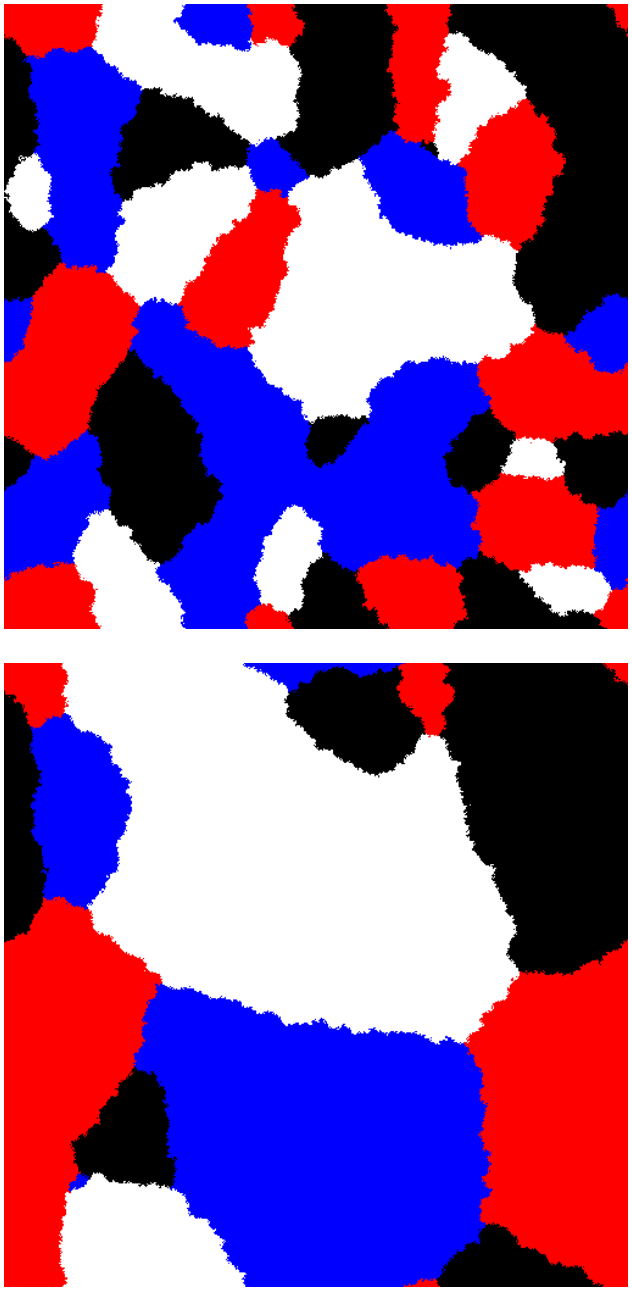


2.2.3 seuil de 3

Les points tendent à se regrouper par groupes monochromes, ceux-ci prenant de plus en plus d'ampleur et s'associant les uns aux autres. Les petits groupes isolés sont voués à disparaître.

L'évolution est quasi-permanente : la seule solution stable est une couleur unique, possibilité qui est envisageable en théorie mais très rarement obtenue en pratique si le tirage aléatoire est correct, c'est-à-dire s'il ne favorise aucune couleur en particulier.



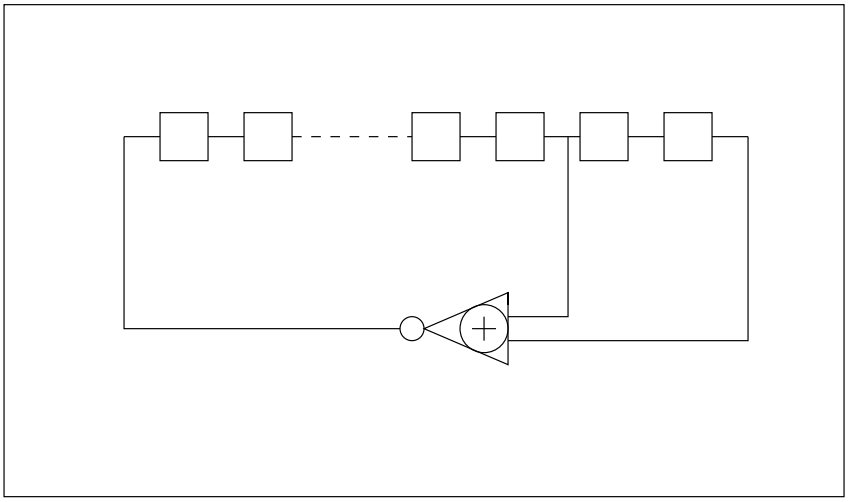


2.2.4 seuil inférieur à 3

Le système est très instable, car il suffit de deux voisins de la même couleur pour que le point central change d'état. Le résultat est donc peu intéressant.

3 Générateur de nombres aléatoires

On peut fabriquer en logique un générateur pseudo-aléatoire de booléens. Voici une méthode possible :

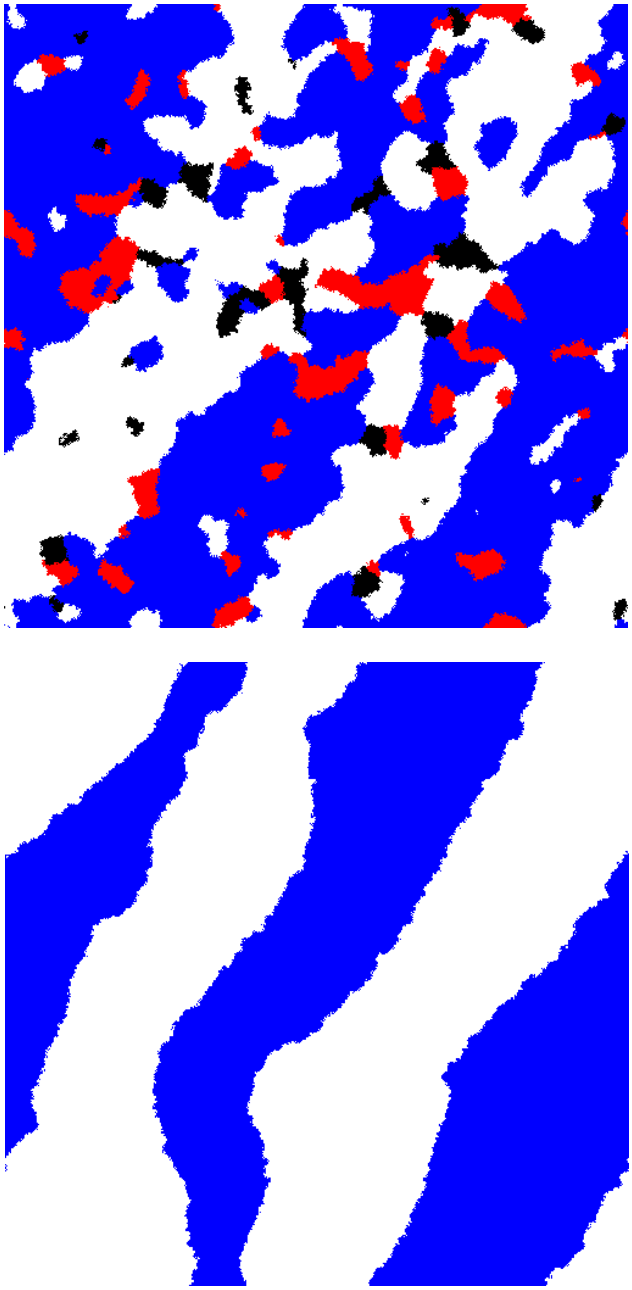


On construit un grand registre à décalage, et on utilise comme entrée le *not* du *xor* des sorties de registres judicieusement choisis. Un choix 'judicieux' permet de générer une suite de bits equi-répartis entre 0 et 1, et de période $2^n - 1$ où n est le nombre de registres utilisés.

Des choix 'judicieux' sont donnés sur

http://www.xilinx.com/xapp/xapp052.pdf

Un exemple de choix non judicieux : certaines couleurs sont privilégiées par rapport aux autres.



4 Utilisation et résultats pratiques

Le design utilise la SRAM liée au chip : on peut donc utiliser lca0 et/ou lca1. Pour faire evoluer une image, on peut opérer ainsi :

1. Transformer l'image en une suite de bits, chaque point de l'image étant codé sur 4 bits, pour une image de 512 x 512 pixels.
2. Charger cette suite de bits dans la SRAM du chip concerné
3. Faire tourner le design pendant le temps souhaité
4. Copier le contenu de la SRAM dans un nouveau fichier
5. Générer un fichier image interprétant l'état de la SRAM
6. Répéter les opérations 3 à 5 pour continuer à faire évoluer la même image

Le design que j'ai créé tourne à 25 MHz, avec uniquement quelques directives très simples de routage. On pourra aisément augmenter la vitesse si le besoin s'en fait sentir.

Il faut 2^{17} cycles pour faire évoluer tous les points, soit 5 ms à 25 MHz : en une seconde, chaque point est examiné 200 fois. Pour information, le même calcul implémenté par Java prend environ dix secondes pour examiner une fois chaque point, pour une résolution de 100×100 pixels.

5 Conclusion

5.1 Intêret du projet

Ce projet m'a permis de découvrir l'utilisation des Pamettes : la programmation en C++ pour créer le design, le placement pour augmenter la fréquence d'exécution et accélérer la compilation, toutes les étapes pour passer du fichier C++ au fichier .lca permettant de cabler la logique souhaitée. En plus j'ai du comprendre comment me servir de la SRAM.

5.2 Améliorations possibles

Le but fondamental de ce projet est de pouvoir observer l'évolution des points sur un écran, en écrivant en temps réel les résultats des calculs sur une carte graphique (projet d'Antoine Miné). Il faudrait pour cela tourner avec une frequence adaptée à celle du bus PCI, et envoyer les résultats des calculs à un autre chip, qui communiquerait avec la carte graphique via le bus.

Une implémentation plus intéressante de ce jeu de la vie consisterait à chosir de façon aléatoire le point qui doit évoluer. Il faudrait générer une adresse sur sur 18 bits pour choisir un pixel, implémentation un peu coûteuse car les générateurs doivent avoir des périodes très grandes et différentes si on veut que le résultat soit homogène.

D'autre part, une RAM pourrait remplacer les registres à décalage. Cette modification ne changerait en rien le résultat du calcul.

Remerciements

Merci à Thomas Deneux et Grégory Miermont pour leur coopération : ce sont eux qui m'ont expliqué les lois d'évolution. En fait ce système modélise la répartition des opinions politiques d'une population : on suppose que si un nombre suffisant de nos amis proches soutient une opinion donnée, on peut être influencé par leur choix.