Les automates cellulaires : Le jeu de la vie

1. Introduction

Défini dans les années 70 par le mathématicien américain John Horton Conway, le jeu de la vie appartient à une vaste classe d'algorithmes appelés automates cellulaires. Ces systèmes ont pour but de simuler l'évolution d'une population de cellules vivantes réparties sur une grille. Comme pour les fractales, on peut obtenir des comportements très complexes avec des règles d'évolution très simples.

Malgré son ancienneté, le jeu de la vie fait toujours l'objet d'intenses recherches, avec souvent des résultats surprenants : on a ainsi pu montrer qu'il se comporte comme un ordinateur universel, capable notamment de calculer (et d'écrire) les décimales de π !

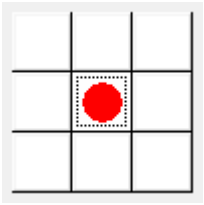
Nous proposons ici une programmation du jeu de la vie en *Crocodile BASIC*. Bien qu'il ne puisse rivaliser avec les logiciels spécialisés tels que Golly, ce programme vous permettra de découvrir les principes du jeu. C'est également un bon test pour le compilateur.

2. Présentation du jeu de la vie

Nous empruntons la description suivante à un ancien article de Frédéric Neuville paru dans la revue « Science et Vie Micro » et disponible sur le site des anciennes revues informatiques.

Qu'est-ce qu'un automate cellulaire ? Ce vocable barbare recouvre en fait une notion assez simple. Un automate cellulaire, c'est, en effet, un ensemble de cases ou "cellules" (par exemple les éléments d'un tableau, les carreaux d'une feuille de papier quadrillé ou les cases d'un damier...). Chaque case peut se trouver dans plusieurs états (occupée ou vide ; blanche, noire ou rouge ; allumée ou éteinte...). L'automate évolue en fonction de règles précises, à chaque étape ou génération. Le contenu de chaque case, lui, est déterminé à partir de son état précédent et de celui de ses proches voisines, un peu comme les valeurs des cases d'un tableur sont calculées les unes à partir des autres.

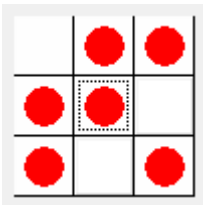
Dans le jeu de la vie, chaque case peut être, soit vivante (ou allumée), soit morte (ou éteinte). A chaque génération, on détermine si une case est vivante ou morte en fonction des règles suivantes. On compte tout d'abord le nombre de voisins que possède la case dans les huit cases contiguës (par côtés et coins). Si la case est vivante mais qu'elle a au moins 4 voisins vivants, elle meurt d'étouffement. Si la case est vivante mais qu'elle n'a que 0 ou l voisin vivant, elle meurt d'isolement. Si elle a exactement 2 ou 3 voisins, elle survit. Une case vide « naît » si elle a exactement 3 voisins. A chaque génération, toutes les modifications (naissances et morts) sont simultanées, il n'y a pas de propagation des changements.



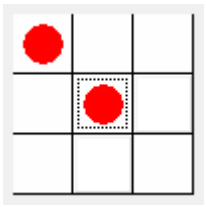
Une cellule et ses 8 plus proches voisins.

- *3 voisins → naissance dans une case vide*
- *0, 1, 4, 5, 6, 7, 8 voisins → mort*
- *2 ou 3 voisins → survie*

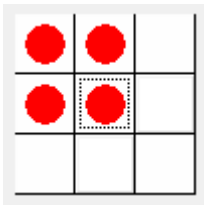
Exemples :



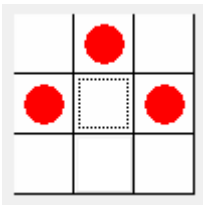
La case centrale a 5 voisins : elle meurt d'étouffement.



La case centrale a 1 voisin : elle meurt d'isolement.



La case centrale a 3 voisins : elle survit.

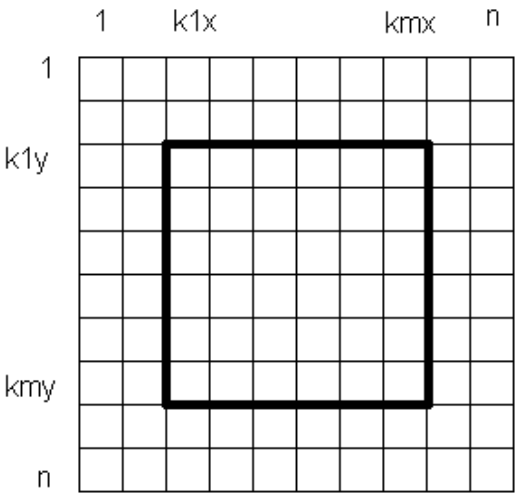


La case centrale a 3 voisins : elle naît.

Le jeu de la vie consiste donc à observer le devenir d'une population donnée, à regarder si elle disparaît au bout d'un certain nombre de générations, si au contraire elle augmente à l'infini, ou encore si elle se stabilise. Les règles du jeu de la vie n'ont pas été choisies au hasard par Conway. Son premier souci était d'éviter deux extrêmes : l'extinction rapide et l'envahissement total, qui auraient privé le jeu de sa richesse. Le savant dosage des naissances et des morts permet de générer des populations changeantes et imprévisibles qui, en général, restent confinées dans une région limitée.

3. Techniques de programmation

3.1. Affichage de la grille



Considérons une grille de jeu carrée, de dimension n (n pair) dont nous allons afficher uniquement la partie centrale sur une étendue de m cases. Soit pour la zone affichée :

- (k_{1x}, k_{1y}) les coordonnées du coin supérieur gauche
- (k_{mx}, k_{my}) les coordonnées du coin inférieur droit

On a les relations :

$k_{1x} = (n - m) / 2 + 1 = k_{1y}$

$k_{mx} = k_{1x} + m - 1 = k_{my}$

Exemple : sur la figure ci-dessus, $n = 10$, $m = 6$: $k_{1x} = k_{1y} = 3$, $k_{mx} = k_{my} = 8$

Pour déplacer la fenêtre sur la grille, il suffira de modifier les valeurs de k_{1x}, k_{1y}, k_{mx}, k_{my}.

3.2. Principe du zoom

Définissons le zoom z par la taille en pixels des cases affichées.

Nous pouvons choisir des valeurs prédéfinies, par exemple $z = (0, 4, 8, 12, 16, 20, 24)$, 0 correspondant à un pixel par case.

Dans ce cas, pour que chaque case apparaisse dans son intégralité, la taille w de la fenêtre graphique doit être égale au plus petit commun multiple (PPCM) des valeurs de z, soit pour notre exemple :

w = PPCM(4, 8, 12, 16, 20, 24) = 240

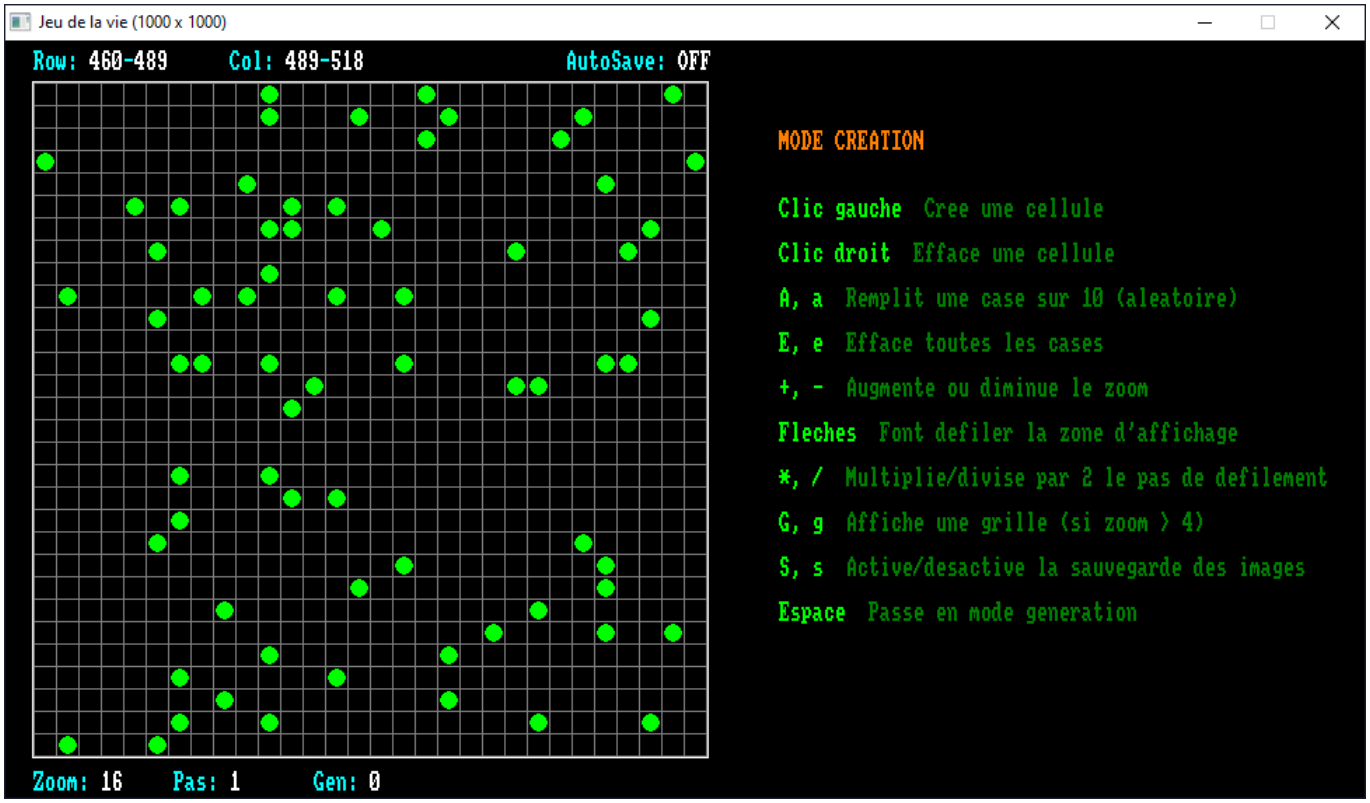
Bien sûr, nous pouvons utiliser un multiple de cette valeur (480, 720, 960…) selon la taille de l'écran.

4. Programmation en « Crocodile BASIC »

4.1. Le programme : `vie.bas`

Ce programme est inclus dans le logiciel *« Crocodile BASIC »* (sous-répertoire **`exemples\autocell`**). La figure suivante représente le fonctionnement du programme en mode « Création ». La fenêtre affiche :

- la grille avec la population
- les coordonnées (Row, Col) des cases affichées
- l'état de la sauvegarde automatique des images (ON / OFF)
- la valeur du zoom
- le pas de défilement de la grille, exprimé en nombre de cases (sous forme 2^n, de 1 à 1024)
- le numéro de la génération affichée



En pressant la barre d'espace nous passons en mode « Génération ». Le programme montre alors l'évolution de la population ; si la sauvegarde automatique est active (AutoSave ON), chaque image est enregistrée dans un fichier PNG ayant un nom de la forme `00000xxx.png` où `xxx` est le numéro de l'image (si des fichiers de mêmes noms sont présents ils seront écrasés). La touche « Echap » permet de quitter le programme.

4.2. Les algorithmes

4.2.1. Taille de la grille et vitesse d'animation

Ces paramètres dépendent de 3 constantes déclarées en début de programme :

`n` : taille de la grille (défaut 1000)

`fg` : taille de la fenêtre graphique en multiple de 240 (défaut 2, soit 480 × 480 pixels, à adapter selon la taille de l'écran)

`lag` : délai entre les images en ms (défaut 50 ; modifier pour accélérer ou ralentir l'affichage ; notez que ce paramètre ne prend pas en compte le temps de calcul, qui augmente avec la taille de la grille).

4.2.2. Codage des populations

Les populations sont mémorisées dans deux tableaux **`a`** et **`b`**, à deux dimensions. Ces tableaux sont de type entier sur un octet (**`dim a%*1(...)`**) Chaque case de tableau contient la valeur 1 ou 0 selon que la cellule correspondante est vivante ou pas.

4.2.3. Les règles d'évolution

L'état futur d'une cellule (survie, naissance ou mort) dépend à la fois de son état présent (**p** = 0 ou 1) et du nombre de ses voisins (**n** = 0 à 8). Il est commode de coder les règles d'évolution dans un tableau **evol** à 2 dimensions, tel que la case **evol(p,n)** contienne l'état futur. Ce tableau est défini en début de programme, les données étant contenues dans des DATA. Il suffit de modifier ces données pour changer les règles du jeu.

4.2.4. Calcul des populations

Le sous-programme **newpop** calcule la nouvelle population (tableau **b**) à partir de l'ancienne (tableau **a**).

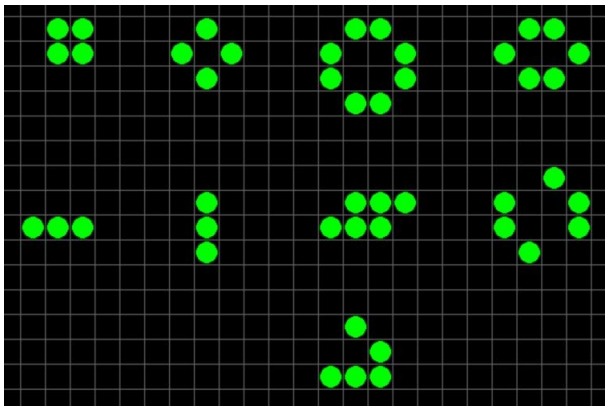
Pour chaque case de coordonnées (**i,j**) nous additionnons les 8 cases qui l'entourent ; cela nous donne le nombre de voisins de la cellule située en (**i,j**), puisqu'une case contient la valeur 1 si elle est occupée par une cellule, 0 sinon.

(i - 1, j - 1)	(i - 1, j)	(i - 1, j + 1)
(i, j - 1)	(i, j)	(i, j + 1)
(i + 1, j - 1)	(i + 1, j)	(i + 1, j + 1)

Le calcul des indices tient compte du fait que la grille se referme sur elle-même, de sorte que par exemple la dernière colonne (**j = ncol**) a pour successeur la première (**j = 1**). On travaille donc sur une surface torique. De cette manière, les cellules qui disparaissent d'un côté réapparaîtront du côté opposé.

5. Exemples de motifs

La figure suivante montre quelques motifs du jeu de la vie parmi les plus simples :



5.1. Les motifs stables

Ces motifs ne changent pas au cours du temps. Ils constituent souvent le terme de l'évolution d'une population. Sur la première ligne de la figure, de gauche à droite : le bloc, le tub, la mare, le nid d'abeilles.

5.2. Les motifs périodiques

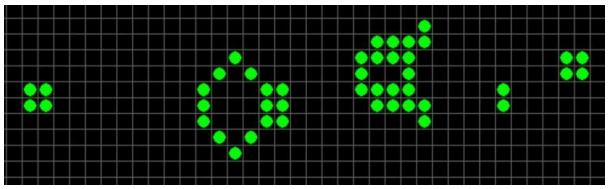
Ces motifs évoluent au cours du temps mais retrouvent leur état initial au bout de n générations. Sur la deuxième ligne de la figure, deux exemples de période $n = 2$, chacun avec ses 2 formes : le clignotant (2 premiers schémas) et le crapaud.

5.3. Le glisseur

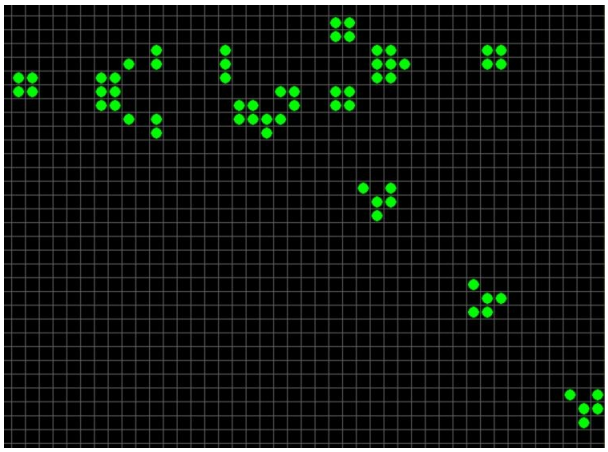
Ce motif représenté sur la troisième ligne de la figure se reproduit toutes les 4 générations tout en se déplaçant d'une case en diagonale.

5.4. Le lance-glisseurs

Ce motif déjà plus complexe émet un glisseur toutes les 30 générations :



Le schéma suivant montre le lance-glisseurs après qu'il ait émit 3 glisseurs :



5.5. Le pentomino

Ce motif très simple donne lieu à une évolution complexe. Il se stabilise à la génération 1103 après avoir émis 6 glisseurs. La figure suivante montre le motif initial et la configuration finale :