262 144 façons de jouer au Jeu de la Vie

un projet de lecteur en bref

Brian White (États-Unis)

Le Jeu de la Vie de Conway me fascine depuis longtemps. Suivez-moi dans mon parcours de conversion de ce casse-tête informatique des années 1970 en un spectacle visuel captivant, en utilisant une matrice de LED RVB associée à des méthodes de codage uniques qui nous permettent de simuler toutes les règles imaginables du jeu !

Ce projet est l'histoire de plusieurs pièces, dont certaines m'ont trotté dans la tête pendant 50 ans, qui ont fini par s'assembler pour aboutir à un résultat fort satisfaisant. Il commence avec le Jeu de la Vie de John Horton Conway, dont j'ai entendu parler pour la première fois à la fin des années 1970. À l'époque, je suivais mon premier (et unique) cours de programmation – en langage BASIC sur un terminal DECwriter connecté à un PDP-11 dans mon lycée – et l'écriture du code pour le Jeu de la Vie était l'un des exercices. C'est un exemple amusant de boucles imbriquées (`FOR...NEXT` en BASIC), un excellent exercice d'initiation à la programmation, qui produit des motifs changeants fascinants. Dans les années qui ont suivi, cet exercice a continué à me fasciner alors que l'informatique – et donc le jeu de Conway – passait des terminaux papier aux écrans vidéo, puis aux ordinateurs portables et aux iPads.

Le jeu de la vie

En bref, le Jeu de la Vie de Conway [1] (en anglais « The Game of Life », GoL) est une simulation simple d'automate cellulaire basée sur des règles. Il se joue sur une matrice rectangulaire de taille quelconque. Les pièces du jeu sont des cellules qui occupent chacune une case de la matrice. Dans le jeu, les cellules peuvent survivre, mourir ou naître d'une génération à l'autre. L'état de vie d'une cellule occupant une case donnée dans la génération suivante

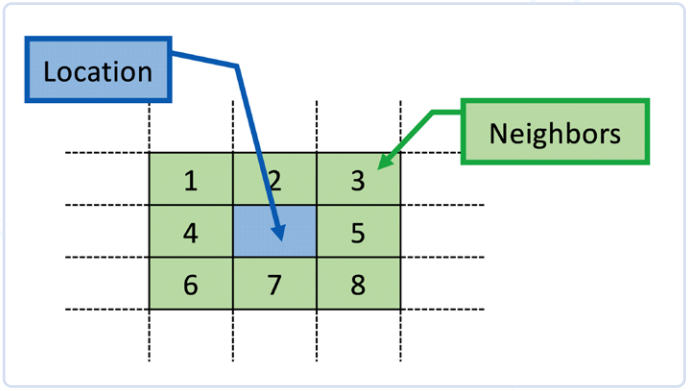


Figure 1. Une case dans le « monde » et ses huit cases voisines.

dépend uniquement du nombre de voisines vivantes qu'elle a dans la génération actuelle. Dans une matrice rectangulaire, chaque cellule a entre zéro et huit voisines (**figure 1**).
Les règles de Conway sont les suivantes :

- Une cellule survit au changement de génération si elle a deux ou trois voisines vivantes. Les cellules qui en ont moins « meurent de solitude » ; celles qui en ont plus « meurent de surpopulation ».
- Une cellule naît dans une case vide à la génération suivante si elle a exactement trois voisines vivantes.

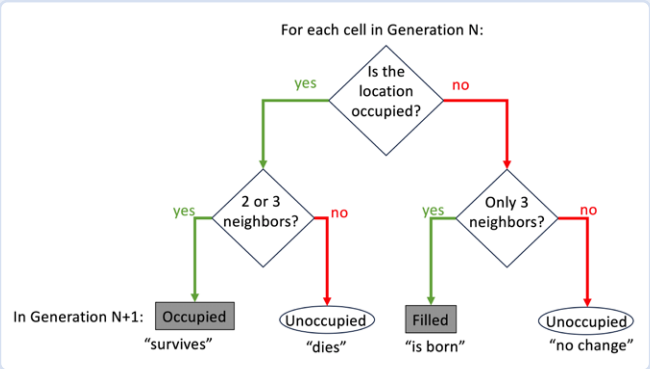
Conway a retenu ces règles après une période d'expérimentation afin de répondre aux critères suivants (tels que décrits dans [2]) :

- Il ne devrait pas y avoir de motif initial pour lequel il existe une preuve simple que la population peut croître sans limite.
- Il devrait y avoir de motifs initiaux qui *semblent* croître sans limite.
- Il devrait y avoir des motifs initiaux simples qui se développent et changent pendant une durée considérable avant de prendre fin de trois manières possibles : en s'éteignant complètement (en raison d'une surpopulation ou d'une trop faible densité), en s'installant dans une configuration stable qui reste inchangée par la suite, ou en entrant dans une phase d'oscillation dans laquelle ils répètent un cycle sans fin de deux périodes ou plus.

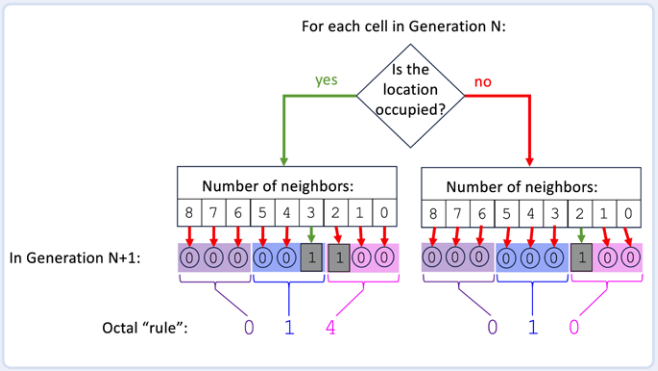
Codage des règles en six chiffres octaux

Les règles originelles du jeu de la vie peuvent être décrites brièvement :

- Une cellule survit si elle a 2 ou 3 voisines, sinon elle meurt.
- Une cellule naîtra dans une case vide avec exactement trois voisines.



Ces mêmes règles peuvent également être représentées de la manière suivante : un « 1 » dans la règle indique une case occupée à la génération suivante (survie ou naissance) pour ce nombre de voisins et un « 0 » dans la règle indique une case vide à la génération suivante (mort ou déjà vide).



Le motif binaire spécifie l'état de la cellule dans la génération suivante ; l'index dans le motif binaire est le nombre de voisins combiné avec le fait que la cellule est actuellement occupée ou non. Le décodage de la règle 256020 est présenté ci-dessous à titre d'exemple :

Tableau 1. Représentation octale du jeu de règles « 256020 ».

Case d'origine	**Occupée**									**Vide**								
Voisines	8	7	6	5	4	3	2	1	0	8	7	6	5	4	3	2	1	0
Génér. suiv.	0	1	0	1	0	1	1	1	0	0	0	0	0	1	0	0	0	0
Octal	2			5			6			0			2			0		

Traduit en texte, cela donnerait

- Actuellement, les cellules vivantes survivent si elles ont 1, 2, 3, 5 ou 7 voisines ; sinon, elles meurent.
- Une cellule naîtra dans chaque case actuellement vide ayant exactement 4 voisines.

Ce format d'encodage permet de spécifier n'importe lequel des 262 244 jeux de règles possibles pour le Jeu de la Vie.

Dans la version originale de 1970, le jeu se jouait avec des pions physiques sur un damier. Peu après, le processus a été informatisé et les innovations se poursuivent encore aujourd'hui [3].

Algorithmes intéressants

La pièce suivante du puzzle a été le livre « A New Kind of Science » de Stephen Wolfram, dans lequel il présente divers algorithmes d'automates cellulaires qui produisent des motifs visuels très intéressants. L'un des éléments clés de l'analyse de Wolfram était le codage des règles de transition d'une génération à l'autre sous forme de nombres binaires. En codant les règles de cette manière, il est possible d'explorer les conséquences de tous les ensembles de règles possibles d'une manière bien définie et reproductible [4]. Cela m'a amené à réfléchir à un moyen d'encoder les règles du GoL afin qu'il soit possible d'explorer tous les ensembles de règles possibles. J'ai compris qu'il était possible d'encoder les règles dans un nombre binaire de 18 bits (voir l'encadré **Encoder les règles en six chiffres octaux** pour plus de détails). Chacun des neuf bits de poids faible (0 à 8) indique l'état de la case dans la génération suivante (1 = occupée ; 0 = vide) pour une case vide avec zéro à huit voisines – le bit 0 indique l'état pour la génération suivante si la case a actuellement zéro voisine ; le bit 1 pour une voisine, etc. De même, les neuf bits de poids fort (9 à 17) donnent l'état de la génération suivante pour une case occupée avec zéro à huit voisines - bit 9 pour zéro voisine ; bit 10 pour une voisine, etc. En utilisant ce codage, il est possible de spécifier n'importe quel ensemble de règles basées sur le nombre de voisines d'une case.

Il est intéressant de noter qu'il n'est pas nécessaire que ces règles aient un « sens biologique ». Par exemple, la règle 256020, selon laquelle les cellules survivent si elles ont 1, 2, 3, 5 ou 7 voisines. Ce jeu de règles donne un comportement intéressant même s'il n'y a aucune justification biologique à ce que 4, 6 et 8 voisines soit un surpeuplement alors que 3, 5 et 7 ne le serait pas.

Étant donné que tous les jeux de règles possibles peuvent être encodés en 18 bits binaires et 2^{18} = 262 144, cela signifie qu'il y a 262 144 façons de jouer au GoL. Il est donc possible de construire un appareil qui permettrait à un utilisateur d'explorer toutes ces possibilités et d'observer comment une population de chaque « espèce » différente se développe au fil du temps.

Le dernier élément de la conception du logiciel est le fruit de

nombreuses conversations avec l'artiste électronique Kelly Heaton [5], que j'ai rencontrée grâce à un article paru dans le magazine *Elektor* [6]. Elle m'a encouragé à sortir des sentiers battus et à réaliser des œuvres qui soient à la fois fidèles à l'algorithme et visuellement attrayantes. Cela m'a amené à modifier le schéma de coloration standard des simulations du GoL, où les cases occupées sont colorées et les cases vides sont noires. Je voulais donner aux motifs une plus forte impression de changement, j'ai donc colorié les cellules nouveau-nées en bleu ; les cellules qui survivent pendant plus d'une génération sont vertes ; et les cellules qui meurent laissent un « fantôme » violet foncé pendant une génération (les « fantômes » ne sont pas comptés comme des voisins). Ce schéma est détaillé dans la **figure 2**.

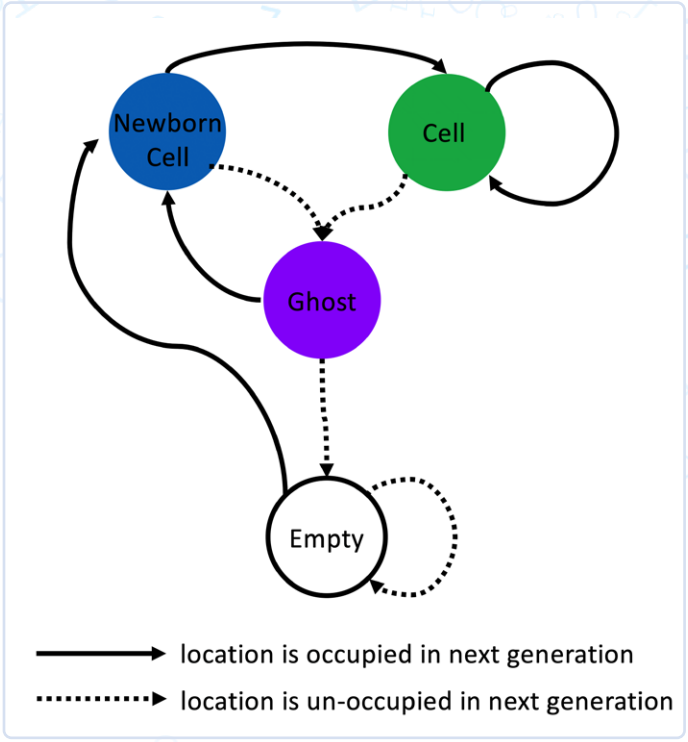


Figure 2. Diagramme d'état pour un motif de cellule multicolore.

Assemblage du matériel

La dernière pièce, le matériel, est un cadeau de Noël de mes fils : un Matrix Portal d'Adafruit [7] et une matrice de 32×64 LED RGB [8]. J'ai ensuite assemblé toutes les pièces dans un boîtier en acrylique pour rendre toutes les parties internes visibles. Le produit final est illustré à la **figure 3**. Sur la gauche se trouvent trois séries de trois roues codeuses de sélection des règles et des paramètres pour la prochaine exécution du programme. Celles du haut et du milieu définissent les règles sous la forme de six chiffres octaux – chaque chiffre représente 3 bits ; multipliés par 6 chiffres, on obtient les 18 bits requis. Les trois chiffres du haut spécifient les règles pour les cases occupées, c'est-à-dire les règles qui déterminent quelles cellules survivent à la génération suivante. Les trois chiffres du milieu spécifient les règles pour les cases vides, c'est-à-dire les règles pour la naissance de nouvelles cellules. À titre d'exemple, les règles canoniques de Conway seraient exprimées sous la forme 014010. Pour plus de détails, lire l'encadré **Codage des règles en six chiffres octaux**. Sous l'écran, de gauche à droite, se trouvent la carte d'extension de port MCP23017, la carte MCU Matrix Portal, le bouton-poussoir de réinitialisation et l'interrupteur à bascule du mode d'affichage.

Chaque exécution commence par une distribution aléatoire de cases occupées et vides, puis se poursuit pendant un nombre fixe de générations avant de redémarrer avec une nouvelle distribution aléatoire.

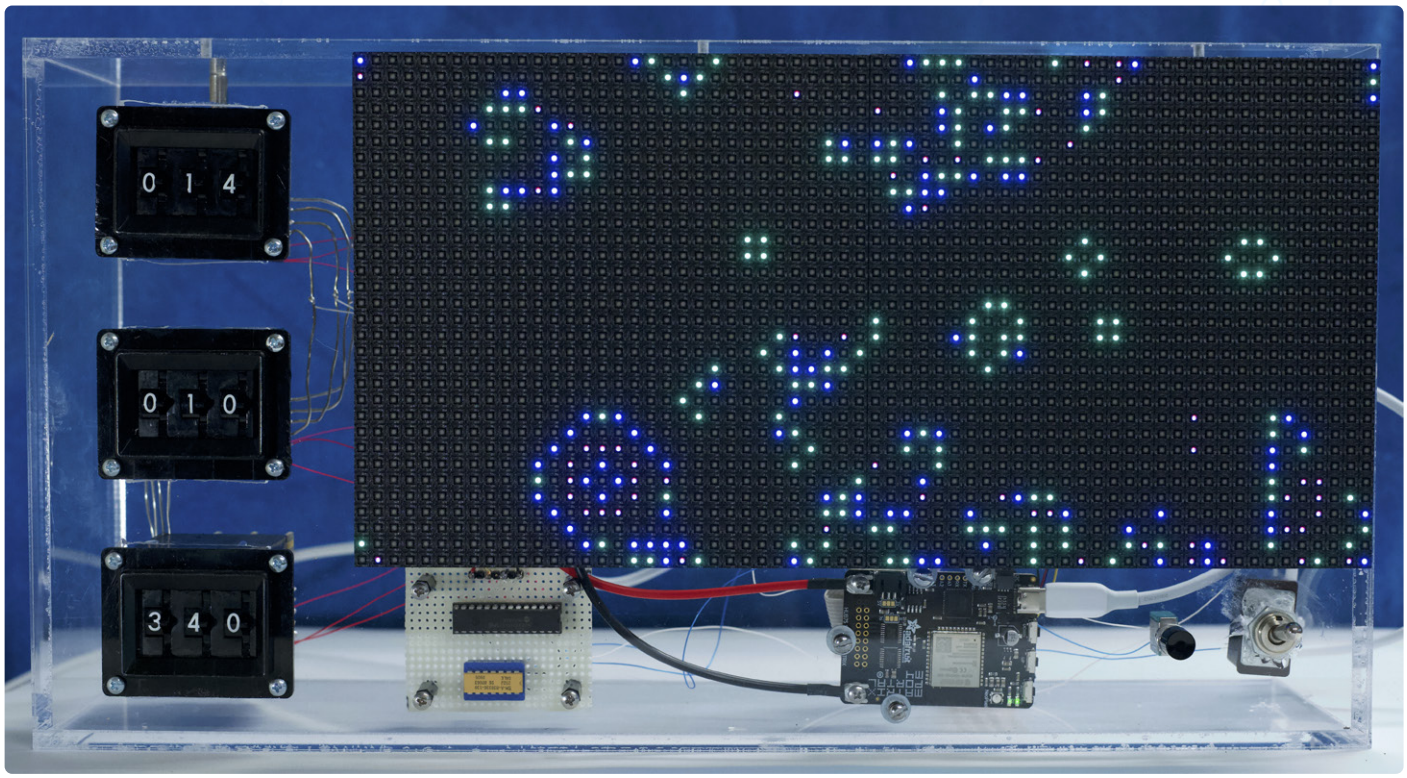


Figure 3. Démonstration du projet assemblé.

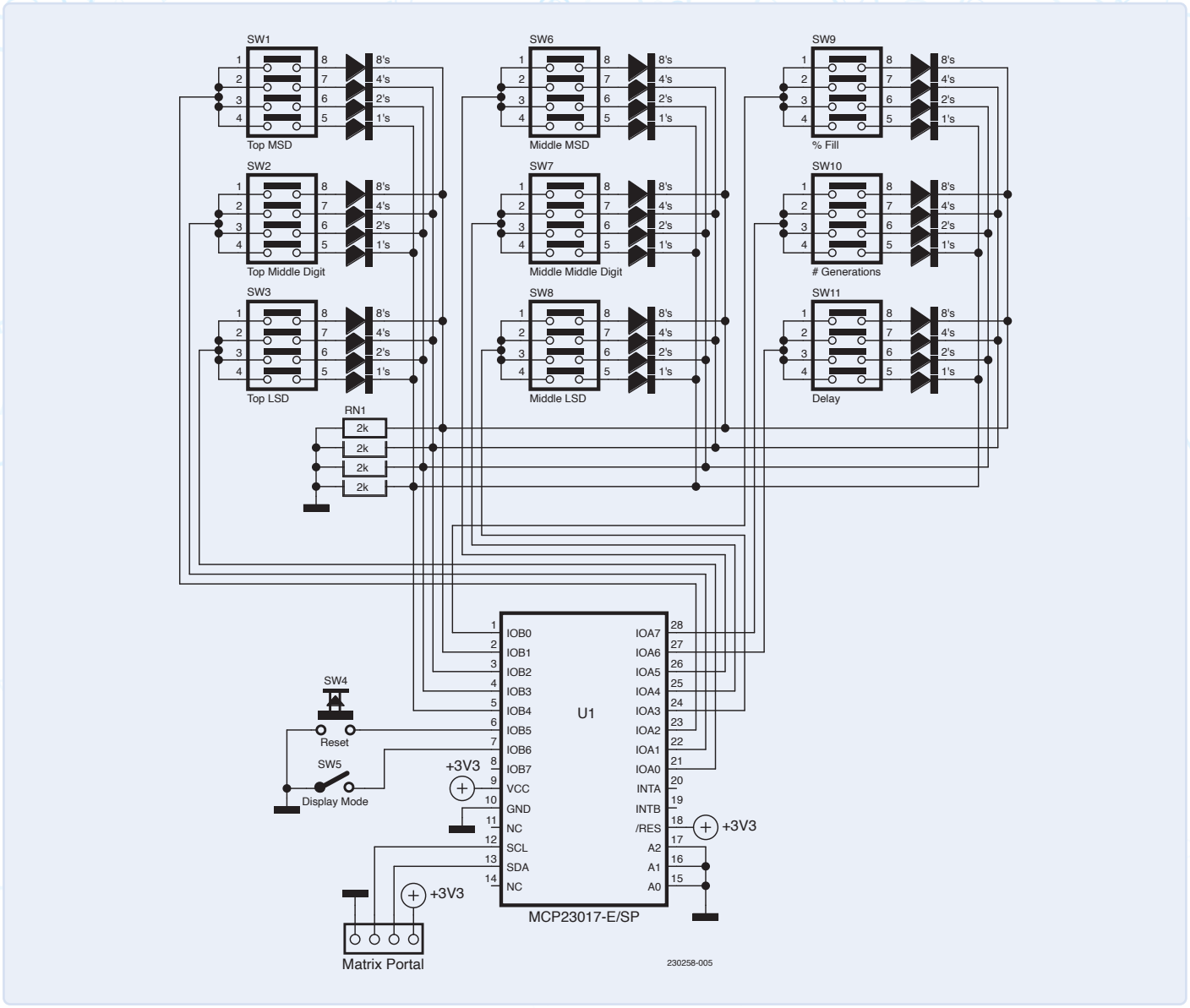


Figure 4. Schéma sans le Portal Matrix, l'afficheur 64×32 et l'alimentation USB.

Les trois chiffres inférieurs précisent les détails de ce processus. Le chiffre de gauche spécifie la densité de cellules vivantes dans la population initiale ; plus précisément, la probabilité qu'une case soit occupée au départ est égale à 10 % de la valeur de ce chiffre. Ainsi, un réglage de «3» signifie qu'au commencement, environ 30 % des cellules sont occupées. Étant donné que certains ensembles de règles donnent des résultats plus intéressants avec plus ou moins de cases occupées, cela permet d'explorer les effets de la variation de la densité cellulaire initiale. Le chiffre central indique le nombre N de générations à exécuter avant de redémarrer. J'ai choisi une échelle exponentielle pour permettre une large gamme de temps d'exécution. Plus précisément, il s'agit de $2\times10^{(N/2)}$ générations ; ainsi, un réglage de 4 correspond à 200 générations. Certains motifs se terminent rapidement, tandis que d'autres ne se terminent jamais ; ce réglage permet de maintenir l'affichage dans un état intéressant. Enfin, le chiffre le plus à droite indique le délai en secondes entre les générations ; ainsi, un réglage de 0 permet d'obtenir un défilement maximal. En ralentissant ce défilement, on peut suivre l'effet des règles en détail. Les deux dernières commandes sont un bouton-poussoir qui redémarre immédiatement l'exécution et randomise à nouveau la population et un interrupteur à bascule qui permet de sélectionner la coloration standard (bleu = occupé ; éteint = vide) ou mon schéma de coloration multicolore des cases dans la matrice.

Le réaliser soi-même

Le schéma de la **figure 4** est très simple. Le Matrix Portal et l'écran sont connectés comme décrit dans les instructions d'Adafruit pour le microcontrôleur et la matrice LED. Les extensions représentées reçoivent leur alimentation +3.3 V et la communication I²C du Portal Matrix via son connecteur Stemma QT. Le gros du travail est effectué par un port d'expansion MCP23017 I²C. Les roues codeuses DCB (SW1 à SW6, SW9 à SW11) sont activées individuellement par les sorties du MCP23017, et les sorties DCB activées de chaque roue sont ensuite lues. Comme le MCP23017 n'a pas d'option de rappel au niveau bas, les lignes DCB sont ramenées à la masse par un réseau de résistances. Les deux dernières entrées du MCP23017 sont utilisées avec les résistances de rappel internes au niveau haut pour lire les commandes de mode de réinitialisation et d'affichage. L'alimentation est fournie par un bloc mural de 5 V, 4 A qui alimente le Matrix Portal via un câble USB-C.

Le code est basé sur le code GoL fourni par Adafruit pour le Portal et l'écran Matrix [9] qui utilise de remarquables astuces de programmation pour exécuter très rapidement le code des boucles

Liste des composants

Adafruit Matrix Portal – Afficheur Internet alimenté par CircuitPython (Adafruit 4745)
64x32 RVB LED Matrix – au pas de 5 mm (Adafruit 2277)
Roue codeuse DCB à 3 chiffres (exemple : Littlefuse 3P-3-23-3-1-0-2)
MCP23017 port d'expansion
4 x 2K résistances de rappel
Bouton-poussoir
Interrupteur à bascule SPST
Bloc d'alimentation USB
Boîtier acrylique

emboîtées typiques du GoL. J'ai modifié le code pour remplacer les règles « standard » par celles basées sur les nombres saisis sur les roues codeuses. En bref, je lis les valeurs binaires des roues et les utilise pour créer une liste de 18 bits à 1 ou à 0 représentant l'état de la case dans la génération suivante, en fonction de l'état présent de la case et du nombre de voisines non vides et non fantômes. Je compte ensuite les voisines et j'utilise ce nombre, auquel j'ajoute 9 si la case est actuellement occupée, comme index dans la liste de règles. J'ai également réalisé le motif multicolore décrit plus haut. Le code est disponible sur la page du projet Elektor Labs [10].

Pour aller plus loin

Les résultats ont été fascinants. J'ai créé une YouTube playlist [11] avec une collection de courtes vidéos montrant les motifs qui émergent avec différents réglages des roues codeuses. Il est remarquable de constater que certaines modifications des règles ont un effet important alors que d'autres n'apportent que des changements subtils dans la manière dont les motifs se développent, changent, se déplacent et s'éteignent. Je n'ai fait que commencer à étudier les possibilités (262 144 est un nombre assez important !), mais voici ce que j'ai remarqué :

- Le réglage des bits de poids faible (bits 0 et 9), qui permet les naissances et/ou les survies avec zéro voisine, entraîne des inversions positives-négatives spectaculaires à chaque génération.
- Le réglage de bits d'ordre supérieur - contrôler les naissances et les survies avec un plus grand nombre de voisines - tend à avoir moins d'effet, peut-être parce que les cases ayant beaucoup de voisines sont plus rares.
- La règle originale, 014010, évolue généralement vers des structures relativement stables ou des figures oscillantes. D'autres règles, comme la 114110, sont les préférées de ma femme car elles semblent ne jamais s'arrêter.
- Vous pouvez créer des règles dans lesquelles les structures sont créées et se développent progressivement de manière intéressante si vous autorisez la survie pour de nombreuses voisines et les naissances pour quelques-unes seulement ; par exemple, 256020.

En ce moment, l'appareil repose sur notre table de cuisine et je constate souvent que quelqu'un de la maisonnée a trouvé un nouvel ensemble de règles qui donne lieu à un nouveau motif fascinant. Depuis que nous vivons avec cet appareil et que nous le partageons avec d'autres, nous avons eu plusieurs idées pour ceux qui voudraient en construire un à leur tour. Tout d'abord, on pourrait ajouter d'autres couleurs au schéma multicolore. Par exemple, les couleurs pourraient passer progressivement à d'autres teintes au fur et à mesure que les cellules « mûrissent ». Kelly Heaton a suggéré que les couleurs se fondent l'une dans l'autre plutôt que de changer brusquement, afin d'obtenir un effet plus fluide. Enfin, les membres de mon foyer qui ignorent l'octal suggèrent de trouver un moyen de montrer plus explicitement la correspondance entre les bits des règles et le nombre de voisines. On pourrait ajouter une rangée de 18 LED pour afficher l'équivalent binaire des paramètres octaux ou, plus simplement encore, utiliser 18 interrupteurs individuels, un pour chaque bit de la règle.
J'espère que des lecteurs de cet article se lanceront dans la réalisation de leur version de ce projet, soit avec un écran et un MCU dédiés, soit entièrement sur ordinateur, et qu'ils partageront les ensembles de règles qu'ils trouvent intéressants. Qui aurait cru que partager des nombres octaux à 6 chiffres pouvait être aussi intéressant ?

VF : Helmut Müller — 230258-04

À propos de l'auteur

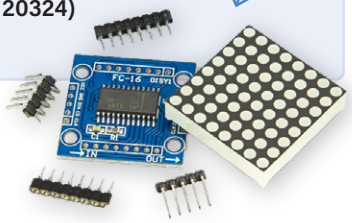
Brian White est professeur au département de biologie de l'université du Massachusetts, à Boston. Il a étudié la biologie au MIT et à Stanford. En tant qu'enseignant, il mène également des recherches sur la biologie et l'enseignement de la biologie et développe des logiciels connexes. Brian est également un passionné d'électronique, un musicien et un radioamateur (KA1TBQ). De plus amples informations sur ses projets électroniques sont disponibles sur son site [12].

Questions ou commentaires ?

Envoyez un courriel à l'auteur (brian.white@umb.edu) ou contactez Elektor (redaction@elektor.fr).

Produits

- **MAX7219 Dot Matrix Module (Set of 8)MAX7219 Module matriciel à points (lot de 8)**
  www.elektor.fr/18422
- **Adafruit Feather RP2040 (SKU 19689)**
  www.elektor.fr/19689
- **ESP32-C3-DevKitM-1 (SKU 20324)**
  www.elektor.fr/20324



LIENS

[1] Wikipédia : Le jeu de la vie de Conway : https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life
[2] Martin Gardner, «MATHEMATICAL GAMES : The fantastic combinations of John Conway's new solitaire game 'life'», Scientific American 223 (octobre 1970) : 120-123 : https://web.stanford.edu/class/sts145/Library/life.pdf
[3] LifeWiki : https://conwaylife.com/wiki
[4] Stephen Wolfram, Un nouveau genre de science, voir en particulier cette section : https://wolframscience.com/nks/p53--more-cellular-automata
[5] Kelly Heaton Studio: https://kellyheatonstudio.com
[6] C.J. Abate, « Faire de l'art avec l'électricité : Q/A avec Kelly Heaton », Elektor Circuits de Vacances 2022 : https://www.elektormagazine.fr/magazine/elektor-264/60889
[7] Matrix Portal - Affichage Internet alimenté par CircuitPython : https://adafruit.com/product/4745
[8] Matrice de 64×32 LED RVB – Au pas de 5 mm : https://adafruit.com/product/2277
[9] Exemple Adafruit : Le « Jeu de la Vie » de Conway : https://learn.adafruit.com/rgb-led-matrices-matrix-panels-with-circuitpython/example-conways-game-of-life
[10] Page du projet sur Elektor Labs : https://elektormagazine.fr/labs/262144-ways-to-play-the-game-of-life
[11] Vidéos du Jeu de la Vie : https://youtube.com/playlist?list=PL2wCLIpn1MkRpBHeZ2svMs-CdLvf2B_9N
[12] Site web de l'auteur : https://brianwhite94.wixsite.com/electronics