**Mathis Scheffler**
**Hammouda Ilyes**

**ENSAE 2ème année**
***Rapport project C++***
***Année scolaire 2022 - 2023***


# Jeu de la vie

**Professeure : Roxana DUMITRESCU**

# Table des matières

# 1 Introduction

Dans ce projet, nous avons travaillé sur le jeu de la vie en utilisant le langage de programmation C++. Nous avons manifesté un grand intérêt pour ce projet, ce jeu étant l'un des automates cellulaires les plus importants. Il est souvent utilisé pour étudier la théorie de la complexité, comme le montre bien l'article "Algorithmic Specified Complexity in the Game of Life" [7].En effet, il illustre comment un problème d'informatique complexe peut émerger d'un problème simple. Ainsi, certaines méthodes de résolution numérique s'inspirent de ce jeu, comme la méthode d'Optimisation par Essaim Particulaire (OPS). [1]

Le jeu de la vie est un système de simulation de population de cellules créé par le génial John Conway en 1970. Il s'agit d'un automate cellulaire, c'est-à-dire un système informatique qui simule l'évolution de populations de cellules. Ce dernier a été fortement inspiré par les travaux de John Leech ainsi que par le problème proposé par John Von Neumann, qui cherchait à décrire une machine capable de s'auto-reproduire. Il a voulu, en particulier, simplifier l'algorithme de Neumann tout en proposant une solution efficace. Le jeu de la vie de Conway est considéré comme l'un des premiers exemples d'automates cellulaires et « ouvrit aussi un nouveau champ de recherche mathematique, celui des automates cellulaires » [2]

Dans ce rapport, nous allons d'abord présenter le jeu de la vie de manière théorique, en détaillant les règles de base et les fondements mathématiques qui le sous-tendent. Nous allons ensuite décrire la structure globale de notre projet, en expliquant les différentes classes et fonctions que nous avons utilisées pour simuler le jeu. Nous allons également décrire les difficultés que nous avons rencontrées lors de la réalisation de ce projet, notamment les problèmes liés à l'utilisation de la bibliothèque SDL et les difficultés liées à l'optimisation des performances. Enfin, nous allons présenter nos propositions pour améliorer notre projet.

1. Cette méthode est apparue en 1995. PSO s'inspire de la dynamique d'animaux se déplaçant en groupes compacts (essaims d'abeilles, vols groupés d'oiseaux, bancs de poissons). Les particules d'un même essaim communiquent entre elles tout au long de la recherche pour construire une solution au problème posé, et ce en s'appuyant sur leur expérience collective. [4]
2. Martin Gardner : « Mathematical Games »

# 2 Généralités

Le jeu de la vie est un "jeu à zéro joueur" car il ne nécessite pas l'interaction avec un acteur extérieur pour se dérouler. Comme mentionné précédemment, il a été créé avec l'objectif de décrire une machine capable de s'autoproduire. Ce qui distingue cet algorithme de celui proposé par Von Neumann, c'est qu'il est considéré comme Turing-complet, bien que les règles de base soient simples. [3] [3]

La simulation et la visualisation du jeu de la vie se font généralement en utilisant une grille, qui est représentée par une matrice dans le code informatique. Dans les travaux théoriques de Conway, la taille de celle-là est supposée être infinie. Elle est divisée en cellules, certaines étant considérées comme "mortes" et d'autres comme "vivantes". Les cellules "vivantes" sont celles qui sont considérées comme étant actives dans le système, alors que les cellules "mortes" sont celles qui sont considérées comme inactives. Les règles du jeu de la vie déterminent comment les cellules vivantes et mortes évoluent au cours du temps en fonction de leur état et de celui de leurs voisines. [8]

## 2.1 Règles

Le jeu de la vie est un processus qui permet à des cellules de s'interagir entre elles pour créer de nouvelles cellules "vivantes" ou pour mettre fin au cycle de vie d'une cellule en la transformant en cellule "morte". Ce processus est régi par les deux règles simples suivantes :

- Une cellule morte se transforme en cellule vivante si et seulement si elle possède exactement trois voisines vivantes.

-Une cellule vivante ne change pas d'état si elle a deux ou trois voisins vivants, sinon elle meurt. [5]

Ces règles simples, en combinaison avec les interactions entre les cellules, permettent de générer des comportements complexes et évolutifs dans la grille de cellules. [5]

La succession de ces règles permet de générer des formes imprévisibles qui peuvent être comparées à des formes de vie artificielle. Elles peuvent être classées en quatre catégories principales :

- Structure stable ( "nature morte" [4]) : constituée de cellules telles qu'aucune évolution n'est possible
- Structure oscillante : constituée de cellules tel que l'évolution s'effectue de manière cyclique
- Les vaisseaux : constituée de cellules qui peuvent générer une copie d'elle-même après un nombre fini de générations dans un nouvel emplacement de la grille du jeu
- Mathusalems : est constitué de cellules tel qu'ils peuvent aboutir à un état stationnaire après un nombre fini de réalisations.

En raison de la complexité des interactions entre les cellules, le jeu de la vie est considéré comme un système de simulation de variables aléatoires, ce qui peut entraîner l'apparition de formes inattendues, notamment les puffeurs. [8]

3. Une machine de Turing est un mécanisme abstrait, destiné à effectuer des calculs. Les fonctions calculables par une machine de Turing équivaudraient aux fonctions récursives. D'après la thèse de Church-Turing, cet ensemble de fonctions contient toutes les fonctions effectivement calculables [2]
4. John Horton Conway

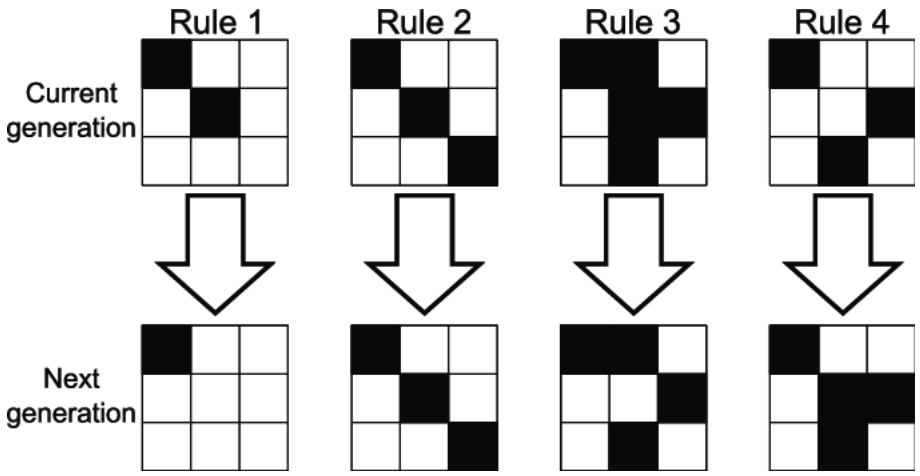


FIGURE 1 – Illustration des différents états possibles d'une cellule donnée [6]

## 2.2 Formalisation

Nous allons définir les différents éléments nécessaires pour modéliser le jeu de la vie. Cette définition peut s'avérer utile pour étendre le jeu à trois dimensions voir plus. Nous allons utiliser les travaux de Carter Bays comme inspiration. On commence par poser $\widetilde{E}$ comme étant l'ensemble des nombres des voisins vivants nécessaires pour permettre à une cellule de continuer à vivre tel que $\widetilde{E_l} \leq \widetilde{E} \leq \widetilde{E_u}$. Soit $\widetilde{F}$ la fertilité, c'est-à-dire le nombre nécessaire de voisins pour générer une nouvelle cellule vivante, tel que $\widetilde{F_l} \leq \widetilde{F} \leq \widetilde{F_u}$. Finalement, on pose la règle de transition $\widetilde{R}$, le quadruplet $(\widetilde{E_l}, \widetilde{E_u}, \widetilde{F_l}, \widetilde{F_u})$.

La règle de transition dans "Conway's Life" est donnée par $\widetilde{R}$ =(2, 3, 3, 3). On peut bien sûr utiliser des règles de transitions différentes, par exemple $\widetilde{R}$ =(3, 4, 3, 4) qui générer une " 3-4 life". Par ailleurs, afin que le jeu de la vie simule plus fidèlement un processus d'évolution d'une population, on peut rajouter des règles supplémentaires comme la mort d'une cellule vivante après la réalisation d'un nombre fixé de générations. Une fois le décor posé, on peut définir un jeu de la vie de la manière suivante :

Une règle $\widetilde{R} = (\widetilde{E_l}, \widetilde{E_u}, \widetilde{F_l}, \widetilde{F_u})$ définie un jeu de la vie si et seulement si elle satisfait ses 2 conditions :
-" A glider" doit exister et doit se produire "naturellement" si nous appliquons à plusieurs générations des configurations de type " primordial soup" [5]
-Toutes les configurations de type "primordial soup" lorsqu'elles sont soumises à la règle $\widetilde{R}$ doivent croître d'une façon limitée. [1]

Le jeu de la vie de Conway, appelé jeu de la vie et bien un cas particulier de jeu de la vie. C'est ce jeu de la vie que nous avons codé.

5. primordial soup est ici définie comme toute masse finie de cellules vivantes arbitrairement denses dispersées au hasard.

# 3 Structure du projet

Comme mentionné dans l'introduction, nous avons utilisé la bibliothèque SDL dans le programme. Cette bibliothèque permet de gérer les graphismes. Le code commence par une initialisation du programme, suivie d'une boucle qui met à jour les composants, analyse les entrées de l'utilisateur et affiche les graphismes. En ce qui concerne le jeu de la vie, la complexité de la mise à jour est linéaire en fonction du nombre de cellules vivantes. Nous mettons, à jour les cellules vivantes ainsi que leurs voisins proches, ce qui permet d'éviter de mettre à jour toute la grille.

Le projet est structuré en 7 classes. Dans cette partie, nous vous expliquerons l'utilité et la structure de chacune de ces classes.

## 3.1 Classe Game

La classe Game est conçue pour gérer tous les objets du code. Elle s'occupe d'initialiser ces objets, de les mettre à jour, de gérer les entrées de l'utilisateur et d'afficher les graphismes. Les fonctions définies dans cette classe sont :

• La fonction "init" s'occupe d'initialiser les composants, notamment SDL.
• La fonction "handle_event" est utilisée pour gérer les entrées du joueur.
• La fonction "update" met à jour les objets.
• La fonction "draw" s'occupe d'afficher les objets.
• La fonction "clean" termine toutes les cellules du jeu.

## 3.2 Classe World

Cette classe contient une Map, qui est une grille de cellules. Elle est utilisée pour mettre à jour l'état des cellules (mort ou vivant) en fonction de leurs conditions. Dans cette classe, nous avons défini les fonctions suivantes :

• La fonction "newWorld" permet de changer la taille de la grille de cellule.
• La fonction "Clear" tue toutes les cellules.
• La fonction "Update" met à jour l'état des cellules.

## 3.3 Classe Painter

Cette classe a pour but d'afficher les divers objets du code. Dans cette classe, nous avons défini les fonctions suivantes :

• La fonction "drawRect" permet de dessiner un rectangle.
• La fonction "draworld" permet de dessiner les tuiles et les aides à l'édition de l'environnement.
• La fonction "drawDrawer" s'occupe de dessiner les cellules sur la gauche pour sélectionner les motifs à dessiner.
• La fonction "drawitem" permet de dessiner l'un des motifs du Drawer.

• La variable "xmargin" correspond à la largeur de la marge.

### 3.4 Classe Map

Cette classe contient et gère un damier de cellules. Dans cette classe, nous avons défini les fonctions suivantes : • La fonction "extend" permet d'agrandir le damier.

• La fonction "clear" permet de tuer toutes les cellules.

• La fonction "Get_neighbours" renvoie les voisins proches d'une cellule. Le programme devrait fonctionner en renvoyant uniquement les voisins les plus proches (au nombre de 8), mais dans quelques rares cas, lorsqu'il y a un grand nombre de cellules vivantes, il est nécessaire de renvoyer les 24 voisins les plus proches. Cela est probablement dû à une erreur d'indexation que nous n'avons pas encore réussi à trouver. Cette méthode nous permet de faire fonctionner notre code, mais multiplie la complexité par quatre.

• La fonction "neighbours" renvoie le nombre de voisins proches d'une cellule vivante.

### 3.5 Classe Tile

Il s'agit de la classe de base du projet. Elle stocke l'état d'une cellule (position, état) et permet de la modifier. Elle stocke également le moment où cet état a été modifié, ce qui est utile lors de la mise à jour des autres cellules.

### 3.6 Classe Drawer

La classe Drawer gère les éléments affichés dans la marge du programme. Ces éléments sont des carrés correspondant à des motifs utilisés pour éditer la grille.

### 3.7 La classe Item

Cette classe décrit un motif utilisé par la classe Drawer. Le polymorphisme est utilisé pour manipuler ces différents motifs de manière uniforme.

# 4 Mode d'emploi

Cliquer dans la grille permet de copier son motif dans celle-ci. Cliquer sur l'une des cases sur la gauche de votre écran permet de sélectionner un motif. Si aucun motif n'est sélectionné, le motif de base (une case seule) sera sélectionné. Attention, si votre motif est plus grand que la taille de la grille affichée à l'écran, il faudra zoomer en arrière pour pouvoir l'afficher sur la grille.

Voici les différentes touches du jeu :

Touche espace : met en pause le jeu

Touche 1 : zoom en arrière

Touche 2 : zoom en avant

Touche t : transpose le motif utilisé.

Touche n : met à jour le jeu une fois

Touche Échap : tue toutes les cases du jeu

Touche flèche vers le haut : accélère le jeu

Touche flèche vers le bas : ralenti le jeu

**Attention avec le zoom ! Si vous dézoomez trop, les cases vont devenir plus petite que la taille d'un pixel et ne seront plus affichées à l'écran. Si votre écran et tout noir et que vous n'arrivez pas à poser de tuille... Zoomez ! Les mêmes problèmes arrivent si vous zoomez et que la case d'une case dépasse la taille de la fenêtre.**

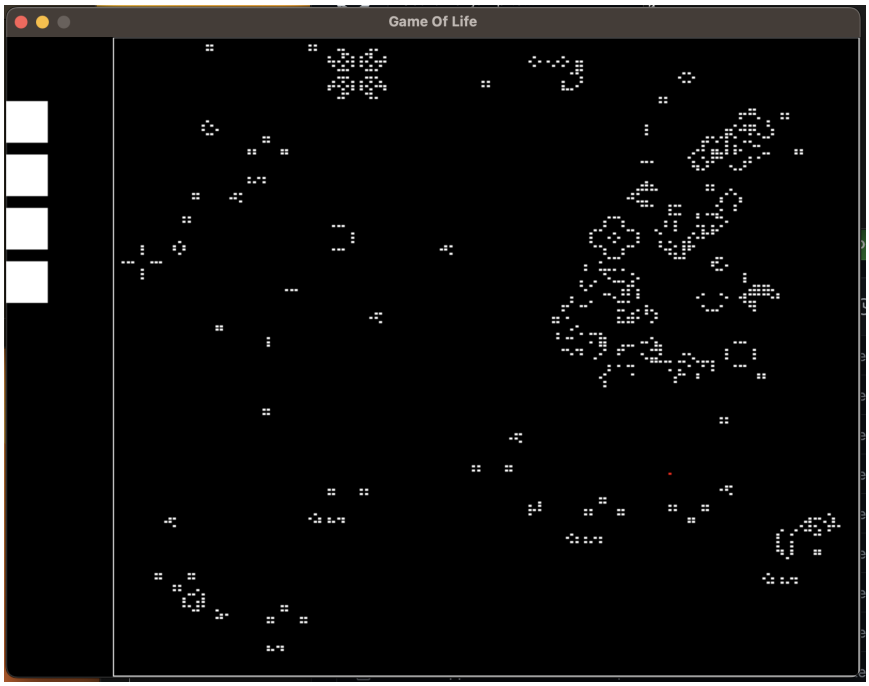


FIGURE 2 – Screen du projet

# 5 Difficultés et remarques

Comme évoqué précédemment, la fonction Get_neighbours de la classe Map nous a et nous pose toujours un problème. Nous suspectons un problème d'indexage quelque part dans le code, mais nous ne l'avons toujours pas trouvé. En attendant, la complexité de la mise à jour de la grille est multipliée par 4 environ.

En plus de cette difficulté vient s'ajouter des difficultés à manipuler SDL. En effet, lorsque le nombre de case en vie est trop grand, soit de l'ordre de 700 à 800, le jeu va crasher à cause d'une erreur SDL. Nous ne savons pas si cette erreur est liée à l'utilisation de SDL sur Mac M1 pro ou s'il s'agit d'une mauvaise utilisation de la librairie. Quoi qu'il en soit, ce problème n'a pas encore pu être résolu.

Par ailleurs, il y a eu d'anciennes versions de ce programme où il était possible de faire planter le programme en cliquant en dehors de la grille de cellules. Ce problème semble avoir été résolu dans les dernières versions du code.

Enfin, le programme utilise énormément de pointeur dû à l'habitude de l'un des membres du groupe à travailler en C# qui est un langage qui utilise beaucoup de pointeurs. Il doit être possible de limiter l'usage de pointeur dans ce programme si cela était nécessaire.

# 6 Améliorations possibles du code

En complexité :

En dehors de la résolution du problème autour de la fonction Get_neihgbours de la classe world, la plus grande piste d'amélioration des performances et un meilleur usage de SDL voir l'utilisation d'un autre framework de graphisme, SDL s'étant révélé peu stable sur Mac M1 Pro. En ce qui concerne la complexité de notre mise à jour de la grille de jeu, notre code est bien plus rapide que la plupart des codes sur internet qui mettent à jour chaque case à chaque itération. Il doit cependant encore être possible de l'accélérer, bien que nous n'ayons pas encore trouvé comment.

GUI :

Nous n'avons pas cherché à développer une belle interface utilisateur, le développement d'un beau et ergonomique GUI étant une longue tache ne permettant pas de mettre en valeur nos compétences en c++. Il va alors sans dire que nous pourrions grandement améliorer notre GUI.

Le Jeu :

Nous aimerions bien améliorer notre jeu en donnant la possibité à l'utilisateur de zoomer dans le jeu à l'aide de sa molette plutôt que de ne pouvoir que doubler ou diviser par deux la taille de la grille.

# Références


[1] Carter Bays. Candidates for the Game of Life in Three Dimensions . http ://wpmedia.wolfram.com/uploads/sites/13/2018/02/01-3-1.pdf.

[2] Bernardo BOLAÑOS. LE DROIT. MACHINE DE TURING, RAPPORT DE FORCES OU ÉQUILIBRE BAYESIEN ? https ://sopha.univ-paris1.fr/fichiers/pdf/2003/02_bolanos.pdf.

[3] Robert A. Bosch. Integer Programming and Conway's Game of Life. https ://epubs.siam.org/doi/pdf/10.1137/S0036144598338252, 1999.

[4] Abbas El Dor. Perfectionnement des algorithmes d'optimisation par essaim particulaire : applications en segmentation d'images et en électronique. https ://theses.hal.science/tel-00788961/document, 2013.

[5] Dariusz Kotula Krzysztof Pomorski. Thermodynamics in Stochastic Conway's Game of Life. https ://arxiv.org/pdf/2301.03195.pdf, 2023.

[6] Tetsuo Sawaragi Takayuki Hirose. Extended FRAM model based on cellular automaton to clarify complexity of socio-technical systems and improve their safety . https ://www.researchgate.net/publication/339605473_Extended_FRAM_model_based_on_cellular_automaton_to_clarify_complex technical_systems_and_improve_their_safety, 2020.

[7] Robert J. Mark Winston Ewert, William Dembski. Algorithmic Specified Complexity in the Game of Life . https ://robertmarks.org/REPRINTS/2015_AlgorithmicSpecifiedComplexityInTheGameOfLife.pdf, 2015.

[8] Francis J. Wrigh. MAS336 Computational Problem Solving Problem 5 : Conway's "Game of Life". https ://neuro.bstu.by/ai/automaton-life-2.pdf, 2007.