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 allons simuler le jeu de la vie en utilisant le langage de programmation C++. Nous avons choisi ce projet car nous sommes intéressés par les thématiques de l'apprentissage automatique, qui sont de plus en plus rares à traiter avec ce langage de programmation par rapport à Python. En plus, ce projet nous offre l'opportunité de mettre en pratique nos connaissances en C++, notamment en ce qui concerne l'utilisation des pointeurs, ainsi que les compétences acquises dans d'autres matières, comme celles abordées dans le cours d'Introduction au processus au premier semestre.

Le jeu de la vie est un système de simulation de population de cellules créé par John Horton 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. Conway a été inspiré par les travaux de John Leech sur le réseau Leech et 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 » [1]

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 conclusions sur ce projet, en discutant des leçons apprises, des améliorations possibles et des perspectives d'avenir pour ce projet.

1. Martin Gardner : « Mathematical Games »

2 Géneéralités

Le jeu de la vie est considéré comme un "jeu à zéro joueur" car il ne nécessite pas l'interaction d'un joueur pour se dérouler. Comme mentionné précédemment, il a été créé avec l'objectif de décrire une machine capable de s'auto-reproduire. Ce qui distingue cet algorithme de celui proposé par Von Neumann est que, bien que les règles de base soient simples, il est considéré comme Turing-complet.Cela signifie qu'il peut être utilisé pour simuler des systèmes complexes, et c'est pour cette raison qu'il a ouvert un nouveau champ de recherche mathématique dans le domaine des automates cellulaires. [2]

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 cette grille est supposée être infinie. Elle est divisée en cellules, certaines étant considérées comme "mortes" et d'autres comme "vivantes". Les cellules "vivantes" représentent les cellules 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.

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 (on parle alors de naissance d'une nouvelle cellule).

-Une cellule vivante ne change pas d'état si elle a deux ou trois voisins vivants, sinon elle devient morte (on parle alors de mort d'une cellule)

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

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" [3]) : constituée de cellules tel 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.

2. Le terme "Turing complète" désigne une machine qui, avec suffisamment de temps et de mémoire ainsi qu'un minimum d'instructions peut résoudre n'importe quel calcul ou problème mathématique et peu importe sa complexité. [1]

3. John Horton Conway

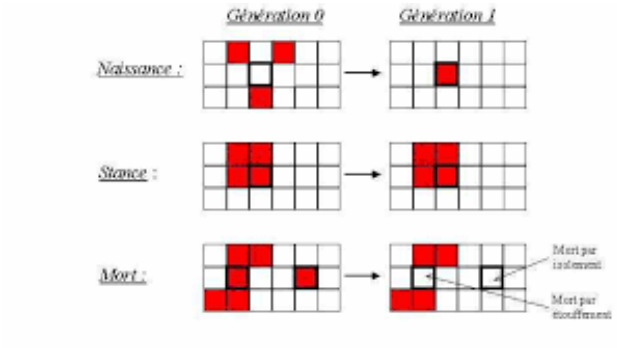

FIGURE 1 – Illustration des différents états possibles d'une cellule donnée [3]

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 ou bien les jardins d'Eden.

2.2 Formalisation

On va définir les différents éléments nécessaires pour créer une simulation du jeu de la vie qui puisse être utilisée dans un espace à trois dimensions. Pour ce faire, on va utiliser les travaux de Carter Bays comme inspiration et poser : L'espace $\widetilde{E}$ comme étant le nombre 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 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 $(\widetilde{E_l}\ \widetilde{E_u}\ \widetilde{F_l}\ \widetilde{F_u})$.

À titre d'exemple la régle de transition dans "Conway's Life" est donnée par $\widetilde{R}$ =(2333). On peut bien sûr utiliser des règles de transitions différentes par exemple $\widetilde{R}$ =(3434) qui générer une " 3-4 life". Par ailleurs, afin que le jeu de la vie simule plus fidélement un processus d'évolution d'une population on peut supposer en plus qu'une cellule vivante meurt après la réalisation d'un nombre de générations, ou élargir son voisinage. Pour complexifier d'avantage l'algorithme on peut aussi penser à un jeu de la vie sur une grille hexagonale à la place d'une grille carrée. Des études ont montré que la régle de Conway est plus simple que celle définie sur un espace à règle de transition hexagonale. Une fois le décor posé on peut définir un jeu de la vie de la manière suivante.

Définition 1 : 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 nous appliquons à plusieurs générations des configurations de type " primordial soup" [4]

-Toutes les configurations de type "primordial soup" lorsqu'elles sont soumises a la règle $\widetilde{R}$ doivent croitre d'une façon

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

limitée. [2]

3 Structure du projet

Comme mentionner dans l'introduction on a beaucoup utilisé la bibliothèque "SDL" dans le programme. En effet cette bibliothèque permet de gérer les graphismes. Le processus consiste donc en 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. On met, alors, 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 principalement. Dans cette partie on vous expliquera l'utilité et la structure de chacune de ces classes.

3.1 Classe Game

La classe 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, on a défini les fonctions suivantes :

• La fonction "newWorld" permet de changer la taille de la grille de cellules affichée.
• 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, on a défini les fonctions suivantes :

• La fonction "drawRect" permet de dessiner un rectangle.

• La fonction "draworld" permet de dessiner les tuiles et les outils d'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 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, on a 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, mais dans quelques cas rares lorsqu'il y a un grand nombre de cellules vivantes, il est nécessaire de renvoyer les voisins proches de deux cellules au lieu d'une. Cela devrait être rare et est probablement dû à une erreur d'indexation que nous n'avons pas encore réussi à résoudre. 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 d'une classe de base du projet. Elle stock l'état d'une cellule (position, vie/mort) et permet de le 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

Dans cette interface, vous pouvez copier un motif en cliquant sur une case de la grille. Pour sélectionner un motif prédéfini, il suffit de cliquer sur une des cases situées à gauche de l'écran. Notez cependant que si le motif est plus grand que la taille de la grille affichée, il faudra utiliser la touche de zoom arrière pour le visualiser correctement. Pour mettre en pause le jeu, utilisez la touche espace. La touche "1" permet de zoomer en arrière, tandis que la touche "2" permet

de zoomer en avant. La touche "t" permet de transposer le motif utilisé, tandis que la touche "n" permet de mettre à jour le jeu d'une génération. La touche "échap" permet de tuer toutes les cases du jeu, tandis que la touche flèche vers le haut"↑" accélère le jeu et que la touche flèche vers le bas "↓" ralentit le jeu.

5 Difficultés

La fonction Get_neighbours de la classe Map est utilisée pour récupérer les cellules voisines proches d'une cellule donnée. Cette fonction est cruciale pour la mise à jour de l'état des cellules dans le jeu de la vie, car elle permet de déterminer les cellules qui doivent être mises à jour en fonction de leur état de vie ou de mort. Cependant, cette fonction renvoie un nombre de voisins plus élevé que nécessaire, ce qui a pour conséquence d'augmenter la complexité de l'algorithme utilisé pour mettre à jour l'état des cellules. En d'autres termes, cela signifie que le programme effectue des calculs inutiles, multipliant ainsi le nombre de calculs par 4.

Cela peut causer des problèmes lorsque le nombre de cellules vivantes est important,peut entraîner des crashes de l'application ou de la bibliothèque SDL utilisée pour gérer les graphismes. Il est probable que cela soit dû à une mauvaise utilisation de cette bibliothèque, qui nécessiterait une analyse plus approfondie pour être corrigé. Il est important de noter que cela est un problème qui n'est pas nouveau et qui a été remarqué lors de la mise au point du programme.

Par ailleurs il y a eu des anciennes versions de ce programme où il était possible de faire planter le programme en cliquant en dehors de la grille de cellules, ce qui indique un manque de sécurité dans les interactions utilisateurs. Cependant, dans la dernière version du code, celle qui accompagne ce rapport , ce problème semble avoir été résolu grâce à des ajustements dans la gestion des interactions utilisateurs.

Enfin, le programme utilise un grand nombre de pointeurs, qui sont des variables permettant de stocker l'adresse d'autres variables. L'utilisation excessive de pointeurs peut rendre le code plus difficile à comprendre et à maintenir, ainsi qu'augmenter la consommation de mémoire et diminuer les performances. Il serait donc possible de réduire l'utilisation de pointeurs dans ce programme, en utilisant des techniques de programmation plus modernes et en optimisant le code pour une meilleure performance. Cela pourrait également améliorer la lisibilité et la compréhension du code.

6 Conclusion

Pour conclure, on pense que La plus grande piste d'amélioration des performances et un meilleur usage de SDL serait de revoir la manière dont la bibliothèque est utilisée dans le programme. Il est probable que des erreurs d'utilisation soient à l'origine des problèmes de stabilité rencontrés avec cette bibliothèque, notamment sur les ordinateurs sous macOS. Il serait donc judicieux de revoir les fonctions utilisées, d'optimiser leur utilisation et de s'assurer qu'elles sont utilisées de manière appropriée. Il pourrait également être envisagé de changer de framework de graphisme, si cela s'avère nécessaire, pour une solution plus stable et adaptée à l'environnement de développement utilisé.

On peut aussi complexifier le problème en essayant de simuler le jeu de la vie dans une grille à trois dimensions. Cela permettra d'utiliser pleinement la définition proposée dans la deuxième partie du rapport. Il est à noter que les travaux de Carter Bays qui propose une démarche théorique pour répondre à cette problèmatique. [5]

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

Références

[1] Coin Academy. https ://coinacademy.fr/lexique/turing-complete.

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

[3] Philippe JEANNET. Le jeu de la vie . https ://cypris.fr/loisirs/le$_j e u_d e_l a_v ie.pdf$.