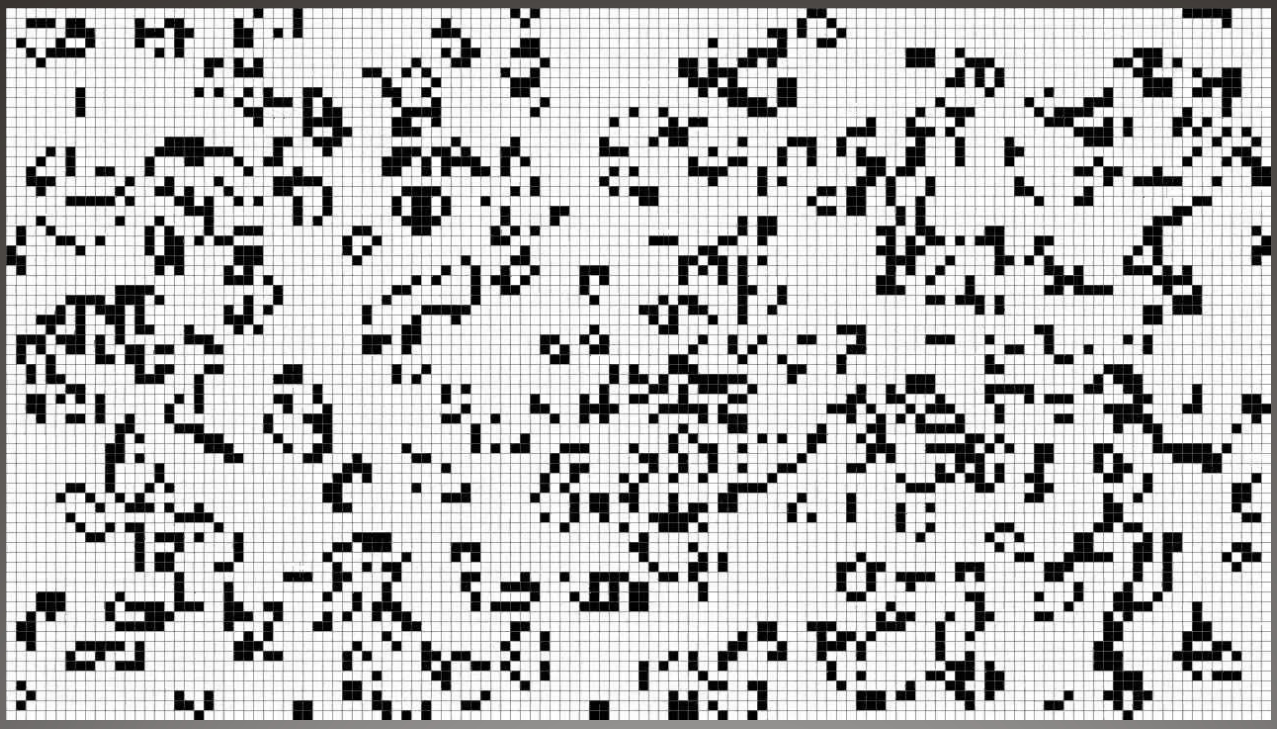

Projet C : Jeu de la vie

Projet réalisé par : Bamouh Mohammed et Adam Saddiki.

Encadré par : M.Hamlaoui

ENSIAS 2016/2017

ENSIAS

Université Mohammed V de Rabat

Sommaire

Remerciements

Avant de commencer la présentation de notre projet, nous profitons de cette opportunité pour remercier toute personne qui a contribué et sacrifier son temps dans la réalisation de ce travail.

Nous remercions notre encadrant M. Mahmoud El Hamlaoui pour ses efforts et ses conseils précieux qui ont bien aidé dans la réalisation et l'amélioration de notre travail.

Nous tenons à remercier : M. Abdellatif EL FAKER professeur du module « Structure de données » pour ses informations et ses conseils durant les séances, M. Ahmed ETTALBI professeur du module « Algorithmique » et M. Mahmoud NASSAR professeur du module « Techniques de programmation » car leurs séances et instructions ont représenté nos premier pas dans la réalisation de ce travail, et aussi nos professeurs du TP M. Mahmoud Elhamlaoui, Mme El Idrissi et Mme Berrada pour consacrer leurs temps durant chaque séance pour améliorer nos compétences.

Finalement, nos sincères remerciements à tous le corps enseignant de l'école Nationale Supérieure d' Informatique et d'Analyse des Systèmes (ENSIAS) qui contribue leur temps et effort pour enseigner chaque génération des étudiants.

Introduction

Dans le cadre de notre première année d'études, à l'Ecole Nationale Supérieure d'Informatique et d'Analyse des Systèmes (ENSIAS), nous avons l'opportunité de réaliser un projet (Projet C) portant sur le célèbre et atypique Jeu de la vie de Conway.

Ce projet est une excellente opportunité pour mettre à l'épreuve nos connaissances en matière d'Algorithmique et Programmation, ainsi qu'en Structures de données en langage C.

Il permet aussi de nous initier au travail de groupe, à la répartition efficace des taches et à la bonne gestion du temps tout en prodiguant un travail de qualité.

Notre rapport est composé de plusieurs parties, tout d'abord, Une partie analyse ou le problème ainsi que ses différents éléments de résolution du problème seront discutés. Une partie Conception ou la structure du programme est expliquée dans les grandes lignes, et finalement la partie Réalisation ou les procédés algorithmiques (langages, structures…) utilisés pour mettre au point le Jeu de la vie seront détaillés.

Table de figures

Analyse

Définition du problème.

Le Jeu de la vie est un automate cellulaire imaginé et conçu par John Horton Conway. Son principe est simple : Dans un tableau rectangulaire a deux dimensions sans bords (la dernière colonne/ligne est adjacente à la première), des cellules (cases) « vivantes » et cellules « mortes » interagissent entre elles selon les règles suivantes : Une cellule morte possédant exactement trois voisines vivantes devient vivante et une cellule vivante possédant deux ou trois voisines vivantes le reste, sinon elle meurt.

Selon les conditions initiales, les cellules forment différents motifs tout au long du jeu. La seule limite aux possibilités offertes par ce jeu est l'imagination humaine.

Cahier de Charges.

A travers cette définition, nous pouvons constater que a Jeu de la vie ne peut être réalisé qu'à travers un programme informatique. Néanmoins, de nombreux obstacles se mettent à travers la conception du programme.

Tout d'abord, le cahier de charges du projet stipule les règles suivantes :

- **Une cellule morte ayant 3 cellules voisines vivantes naît.**
- **Une cellule vivante possédant 2 ou 3 cellules voisines vivantes reste vivante, sinon elle meurt.**
- **La longueur et largeur du tableau doivent être définies par l'utilisateur.**
- **Le résultat doit être stocké dans un fichier texte que l'on peut charger par la suite.**
- **L'utilisateur doit être capable de modifier le tableau durant le jeu.**

Contraintes de programmation.

Le code devra contenir les méthodes suivantes :

-Création d'un tableau a deux dimensions de longueur et largeur imposées par l'utilisateur.

Connaitre l'état actuel de chaque cellule du tableau.

Connaitre le nombre de voisins vivants de chaque case.

A partir des données précédentes, connaitre l'état futur de la cellule.

Modification du tableau en suivant les règles imposées par le Jeu de la vie.

Modification du tableau pendant le jeu.

Affichage du tableau.

Sauvegarde du tableau dans un fichier, afin de pouvoir le charger par la suite.

Création d'une interface graphique pour le projet séparément du programme principal.

Conception

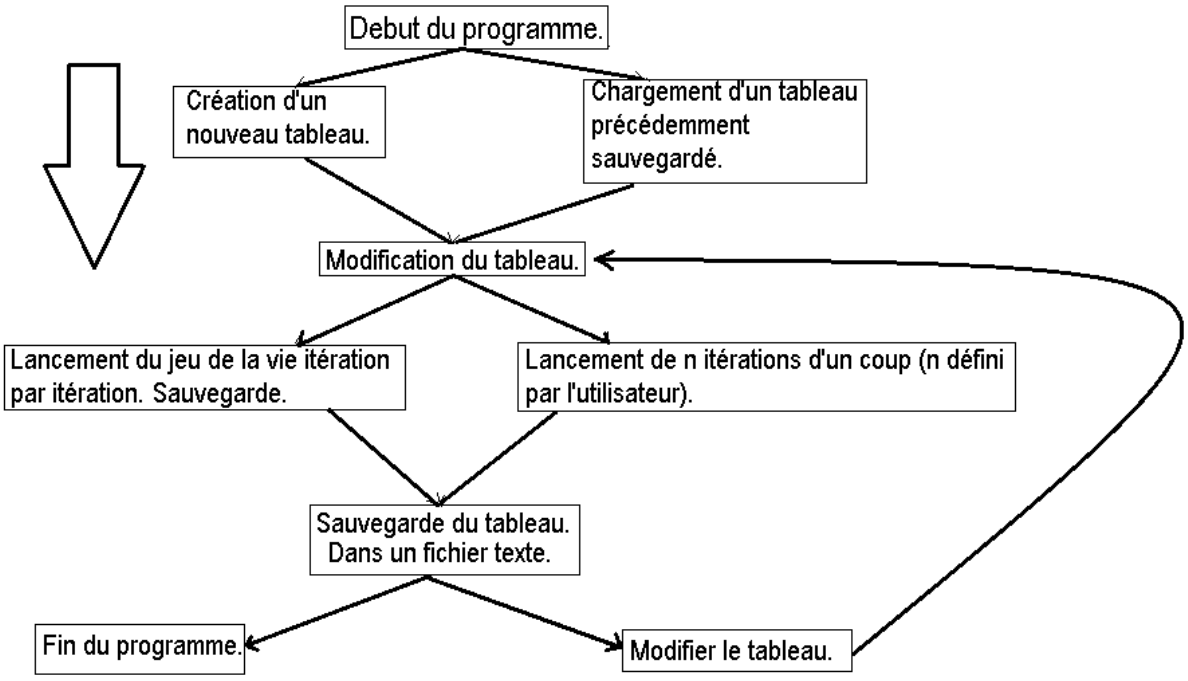

Figure 1 – Vue d'ensemble de l'ordre d'exécution des fonctions du programme principal.

A noter qu'entre chaque étape le tableau est affiché dans la console.

Création d'un nouveau tableau :

L'utilisateur entre la longueur et la largeur du tableau, un tableau longueur*largeur peuplé de cellules mortes est créé.

Chargement d'un tableau précédemment crée :

Le programme reçoit un tableau existant sous forme de fichier texte.

Modification du tableau :

L'utilisateur entre les coordonnées de la cellule qu'il souhaite ranimer ou tuer. Une cellule vivante sélectionnée mourra, et vice versa.

Lancement du jeu de la vie itération par itération :

Le tableau est affiché avec les modifications résultantes des règles du jeu de la vie. Il faut l'autorisation de l'utilisateur pour passer à l'itération suivante.

Lancement de n itérations :

L'utilisateur entre une valeur strictement positive et le programme affiche l'état du tableau après n itérations ainsi que les étapes intermédiaires.

Sauvegarde du tableau :

Un fichier texte est créé contenant le tableau dans son état actuel. On peut y faire des modifications directement. On peut le charger à la prochaine exécution du programme.

Cela concerne le programme principal. Une interface graphique sera créée par la suite, les détails concernant cette étape se trouvent en section Réalisation.

Réalisation :

Conformément au cahier de charges, nous créerons un premier programme en utilisant le langage C pour mettre au point le Jeu de la vie en version console, et un deuxième programme en langage C en utilisant la bibliothèque SDL pour la création de l'interface graphique.

Nous ferons appel aux notions suivantes :

- **Les fichiers.**
- **Les pointeurs.**
- **Les boucles et conditions.**
- **Les tableaux.**
- **Les énumérations.**
- **Les fonctions.**

Les déclarations

Vu qu'une cellule ne peut être qu'à l'état Vivante ou Morte, une énumération représente la meilleure décision pour simplifier les futures instructions.

```
typedef enum Cell Cell;
enum Cell
{
   VIVANTE=1, MORTE=0
};
```

La définition de cette énumération garantit que la cellule ne peut être que dans deux états : **VIVANTE** ou **MORTE**, ce qui est très utile, notamment dans les conditions if… else ou, sans cette énumération, on devrait traiter le cas des nombres autres que 0 et 1.

Les Fonctions en C

l=Longueur.
L=Largeur.

int** JeuDeLaVie(int l,int L);

Cette fonction prend comme entrée la Longueur et la largeur de la matrice, et retourne un pointeur sur un tableau de pointeurs. Chaque élément du tableau est un pointeur qui pointe vers un tableau.

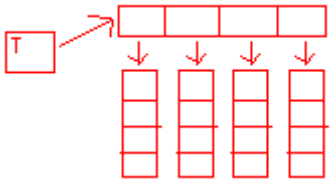

Figure 2 : Double pointeur T.

Toutes les cases de la matrice seront initialisées a MORTE.

int Modulol(int i,int l);

Permet d'éviter d'avoir des coordonnées négatives et de donner les coordonnées des cases de l'autre côté du tableau.
Par exemple, une case située dans la première colonne a, en plus de ses voisins sur la même colonne, une partie de ses voisins dans la 2eme colonne, et une autre dans la dernière colonne.

Int ModuloL(int i,int L) ;

Dans le cas d'un tableau carré, l=L mais dans le cas d'un rectangle il faut prévoir une fonction Modulol pour les extrémités de longueur l et une fonction ModuloL pour les extrémités de largeur L.

int getNbrVoisinsVivants(int** tab,int m,int n,int l,int L);

Additionne les cases avoisinantes de la case en entrée.

int seraVivante(int** tab,int a,int b,int l,int L);

Retourne l'état futur de la cellule en utilisant la fonction précédente.

int** GenerationSuivante(int **tab,int l,int L);

Remplace les cellules dont l'état va changer à la génération suivante. On déclare une nouvelle matrice tab2 via la fonction JeuDeLaVie() qui va recevoir les valeurs de la génération suivante.
On déclare une nouvelle matrice pour éviter le conflit des cellules de l'ancien tableau avec les cellules rendues vivantes ou mortes par cette fonction. En effet, les changements doivent se faire d'un seul coup.

void afficher(int** tab,int l,int L);

Affiche la matrice en parcourant les éléments du tableau et en les affichant selon l'affichage matriciel. La matrice affichée sera la transposée de la matrice déclarée.

int nblignes(FILE* fichier);

Fonction permettant de calculer le nombre de lignes d'une matrice contenue dans un fichier texte en déclarant un compteur qui s'incrémente à chaque fois qu'un caractère autre que le retour à la ligne est lu dans la première ligne du fichier texte.

Ce compteur est divisé par 2 à la sortie de la boucle car les espaces entre cellules sont pris en considération.

Remarque : La fonction retourne le nombre de lignes mais en réalité on calcule le nombre de colonnes de la matrice contenue dans le fichier. C'est parce que cette matrice est la transposée de celle que l'on veut avoir.

int nbcolonnes(FILE* fichier);

Fonction permettant de calculer le nombre de colonnes d'une matrice contenue dans un fichier texte. On utilise la fonction fgets() pour calculer le nombre de lignes de la matrice texte.

Remarque : Même remarque que dans la fonction précédente.

int** charger(FILE* fichier);

Cette fonction permet de lire les valeurs de la matrice dans le fichier texte et les met dans un tableau à deux dimensions déclaré dynamiquement en tant que variable locale. Après remplissage ce tableau est renvoyé.

void sauvegarder(int** tab,int l, int L);

L'inverse de la fonction Charger(), elle copie les données du tableau en console vers un fichier texte de façon à ce que les données soient affichées dans le fichier texte en respectant l'affichage matriciel.

Exécution du programme

Dans cette démonstration, nous allons déclarer une matrice de taille 5*4 :

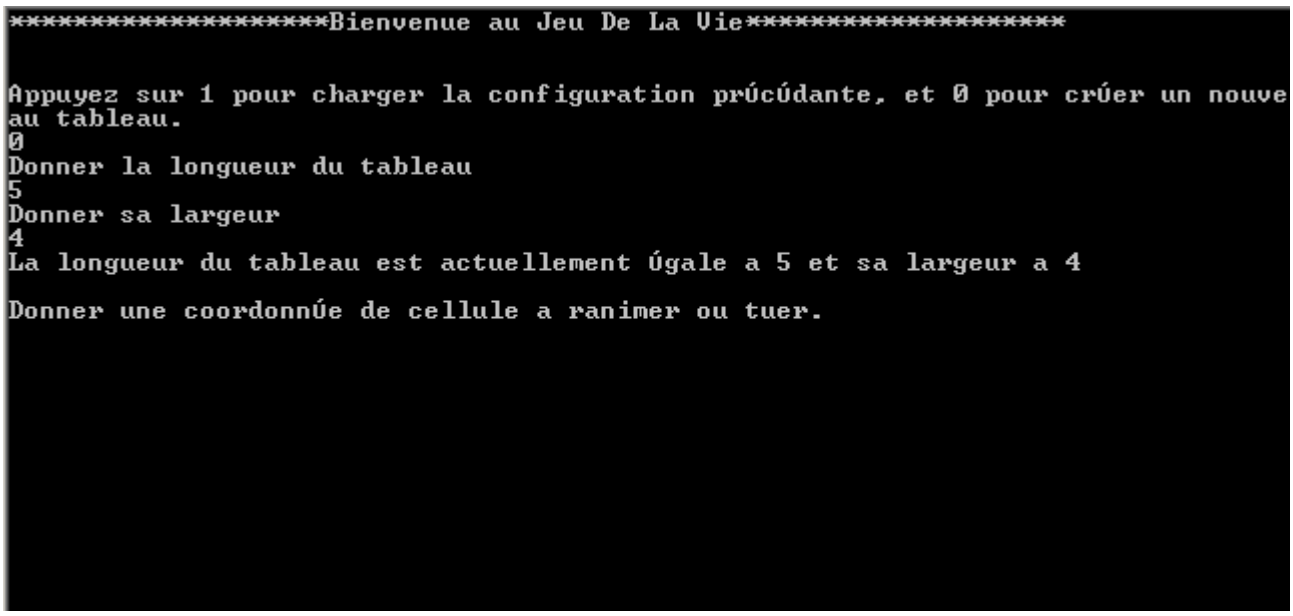

Figure 3 – Création d'une matrice de taille 5*4.

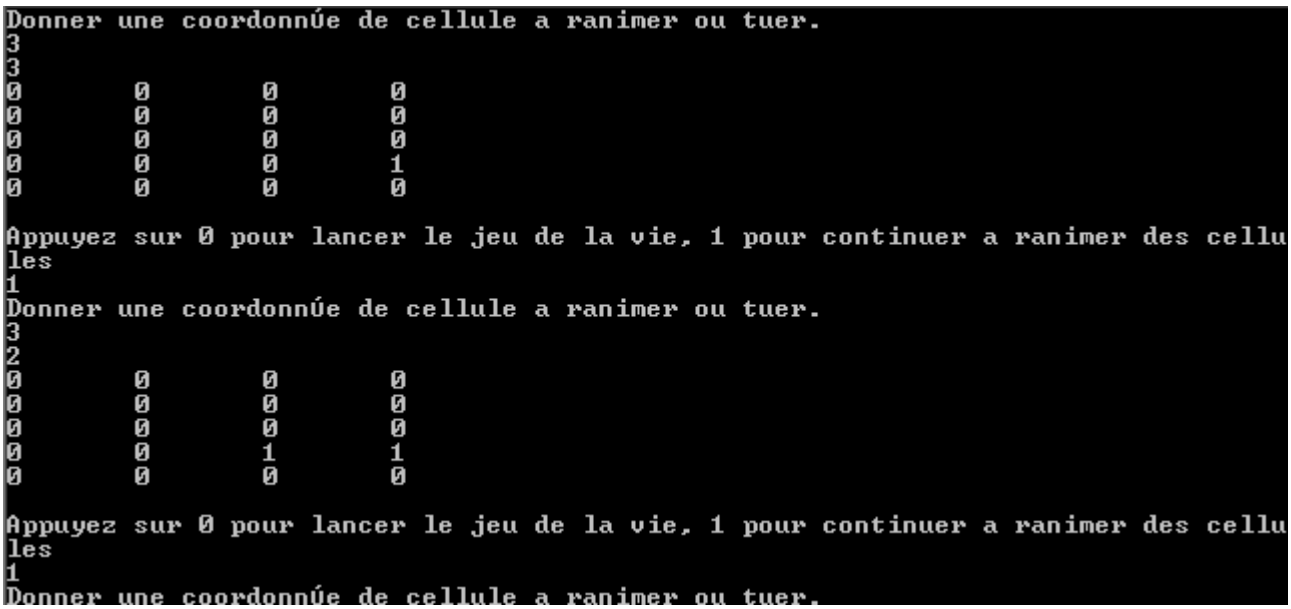

Figure 4- Remplissage de la matrice

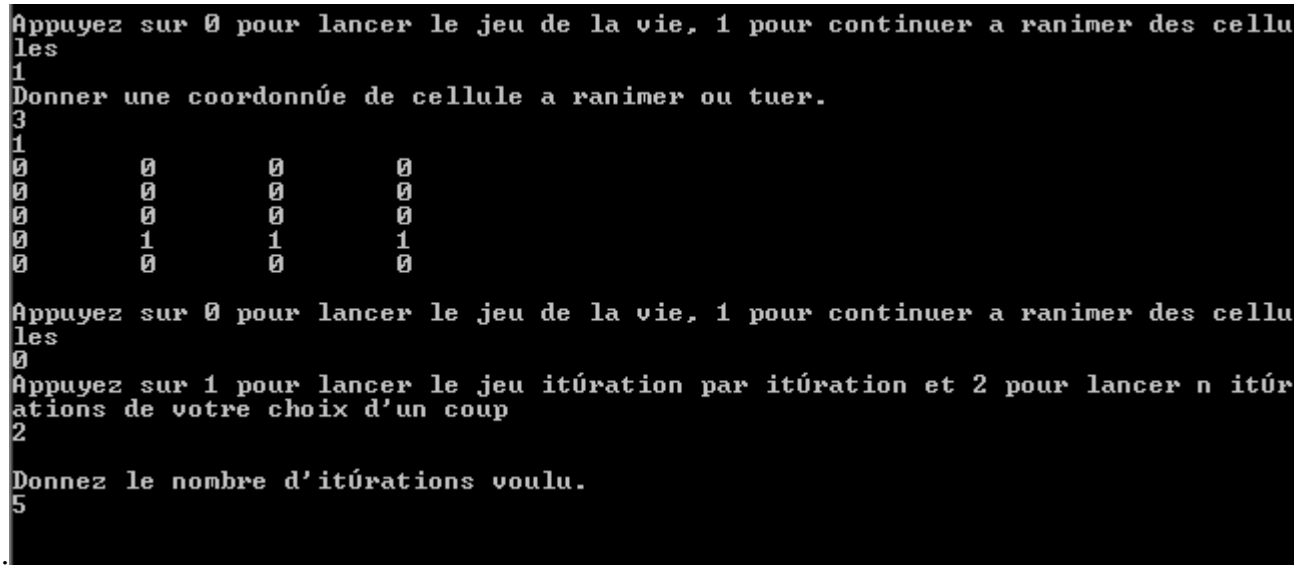

Figure 5- Lancement de 5 itérations d'un coup.

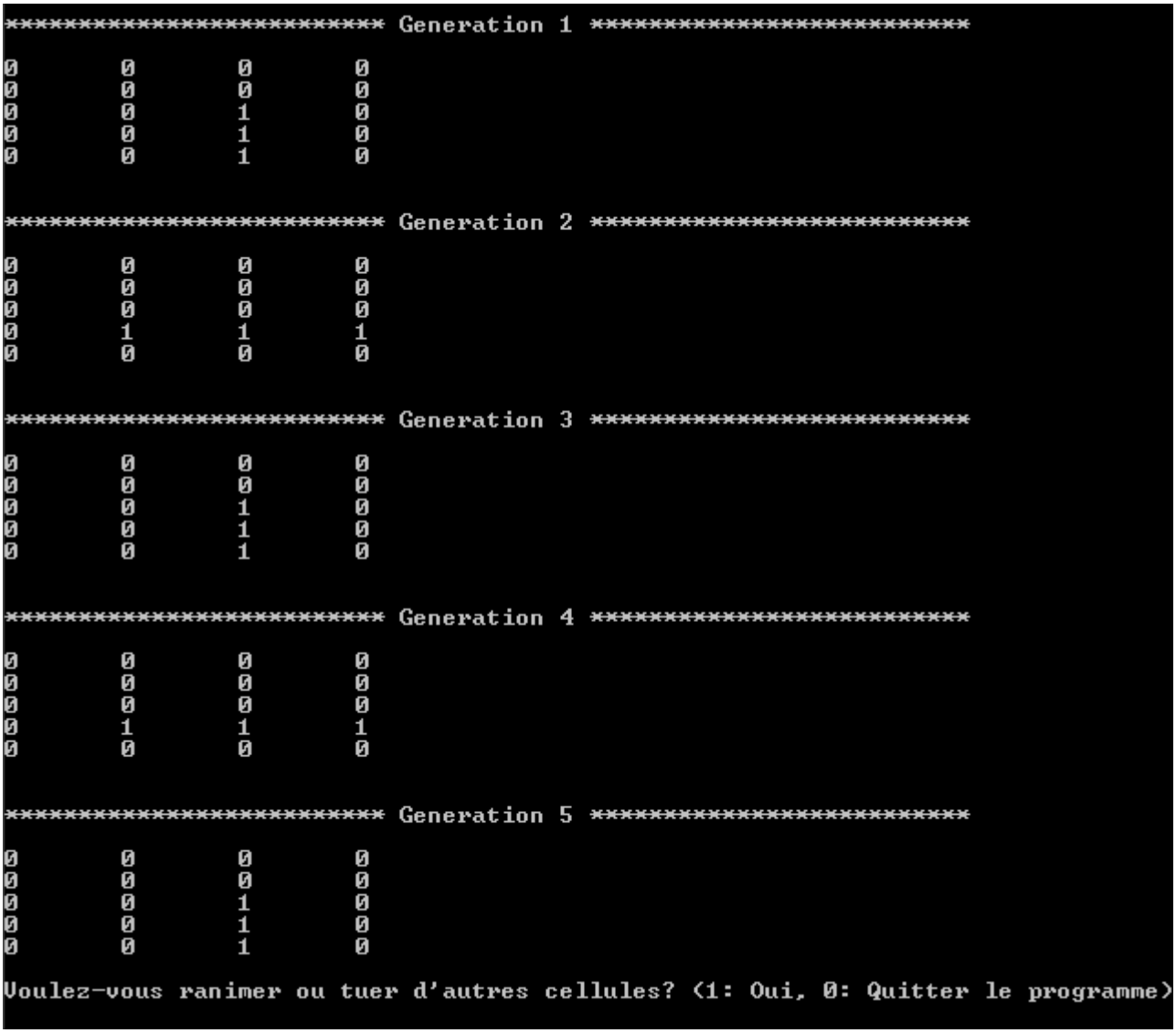

Figure 6- Affichage des 5 itérations.

Maintenant, relançons le programme et chargeons la matrice obtenue.

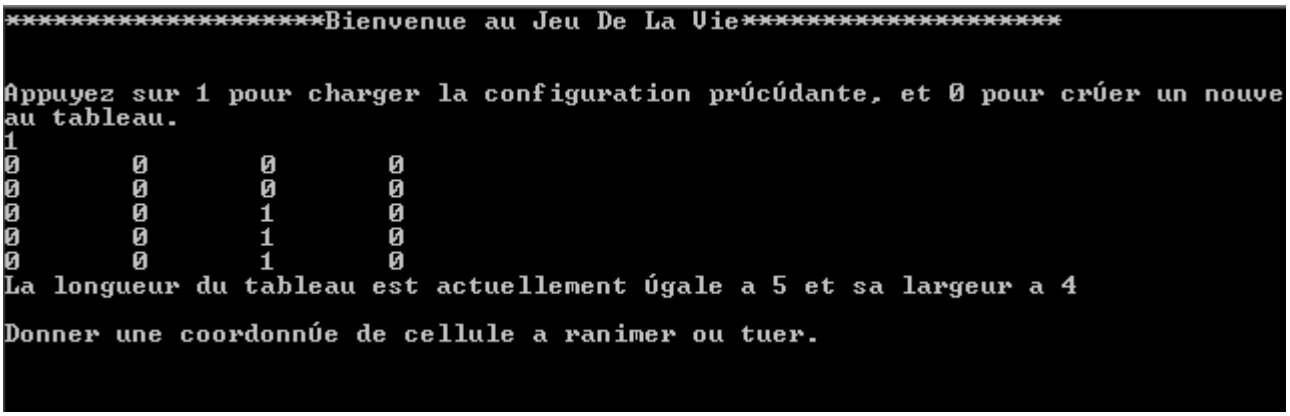

Figure 7- Lancement à nouveau du programme, chargement et affichage de la matrice obtenue précédemment.

Ranimons la cellule de coordonnée (1,1) :

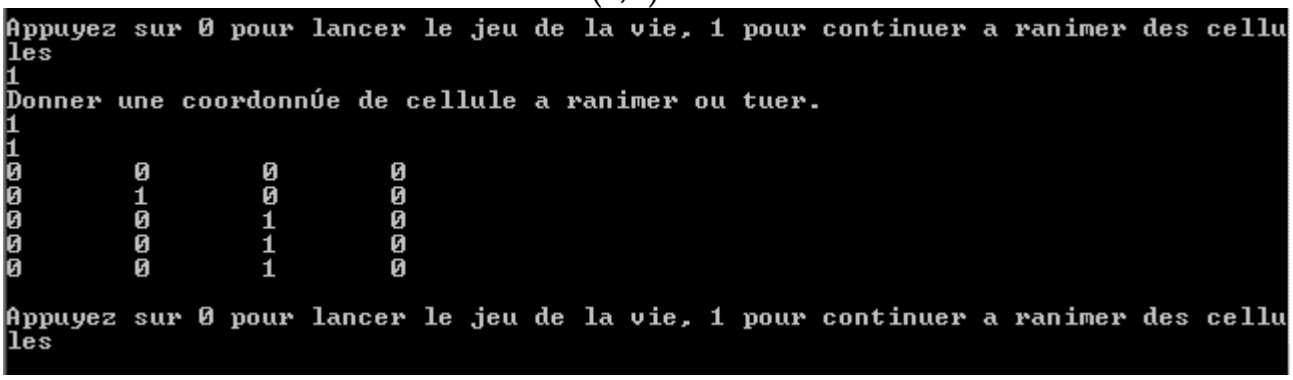

Figure 8- Ranimation de la cellule de coordonnée (1,1)

Lançons cette fois ci 5 itérations une par une :

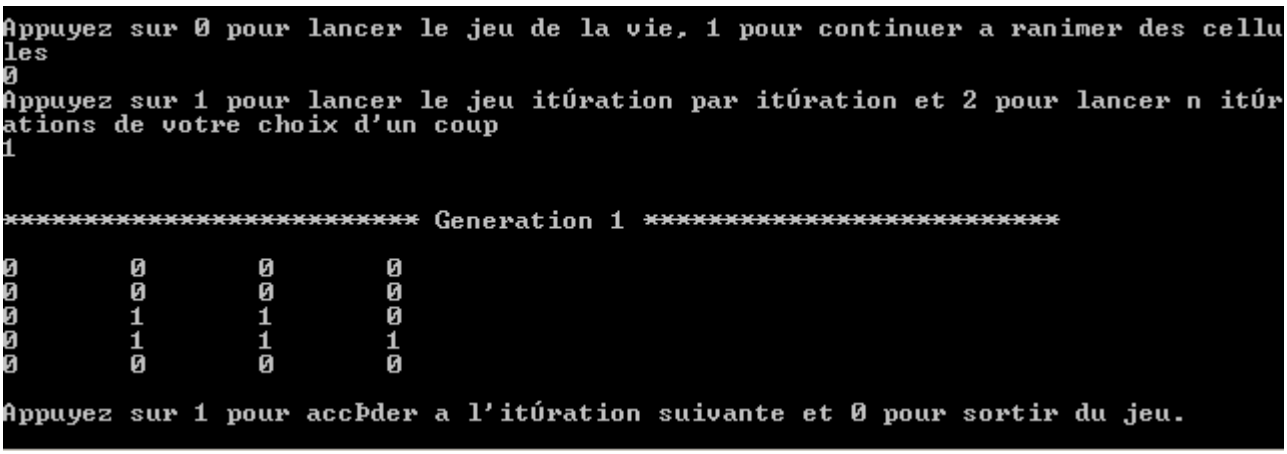

Figure 9- Lancement de la première itération

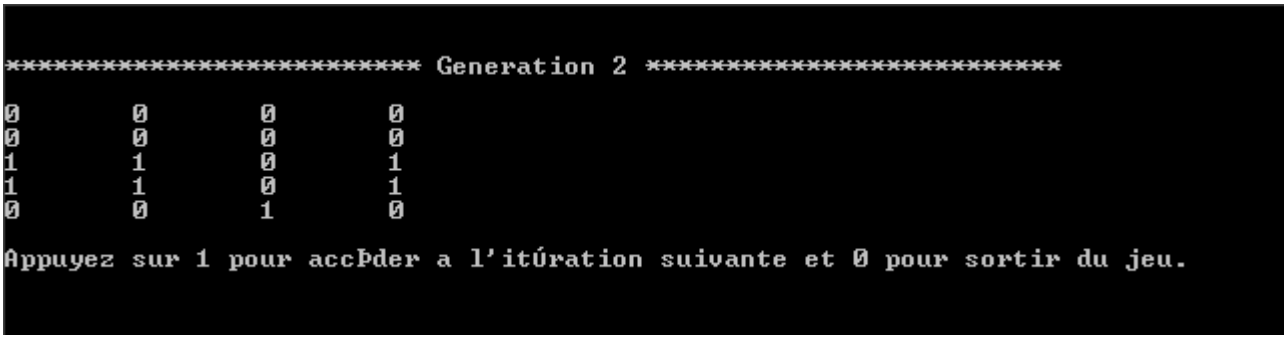

Figure 10- Deuxième itération.

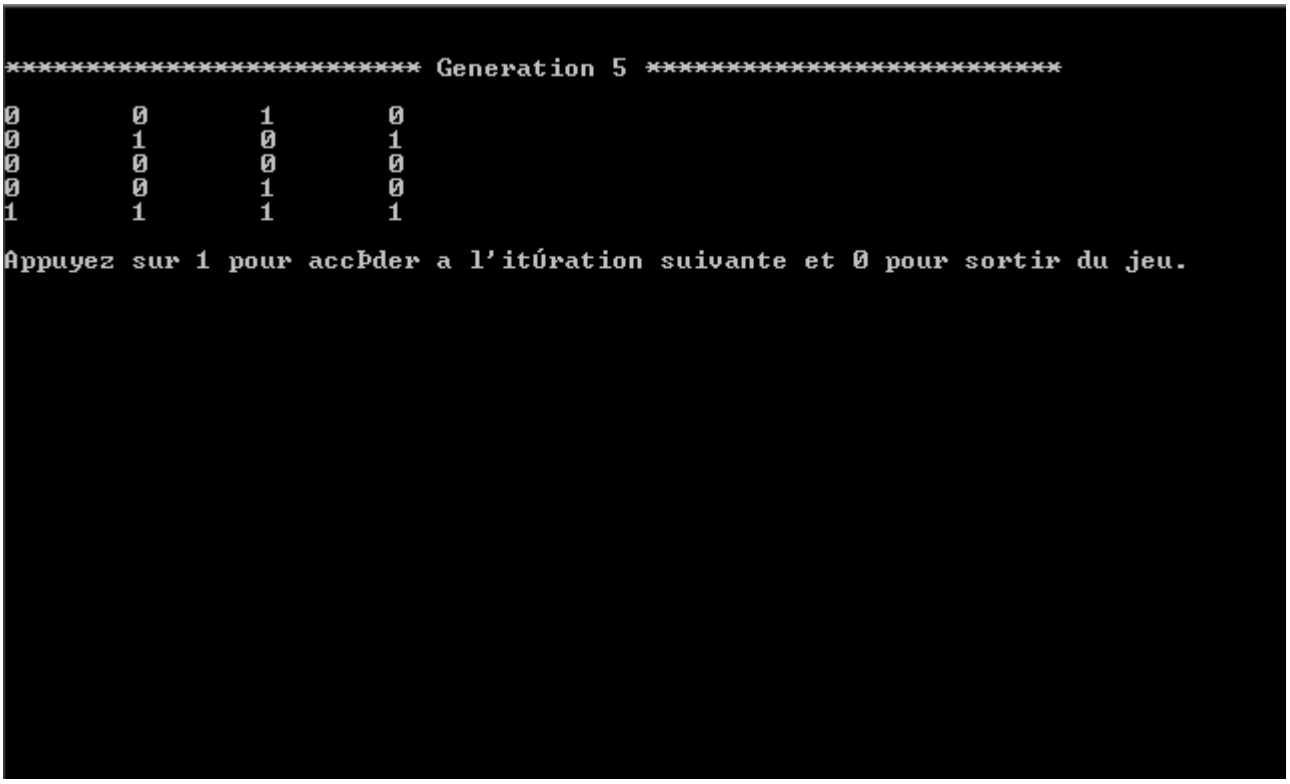

Figure 11- Cinquième génération.

A remarquer que cette fois ci, pour un souci de lisibilité et d'esthétique, tout ce qui précède l'itération est effacé, pour cela, on a utilisé la fonction system("cls") de la bibliothèque windows.h .

L'interface graphique en SDL

Le programme en console donne une présentation sous forme matricielle en vérifiant les règles du « Jeu de la Vie » de Conway. Mais, la version console nécessite que l'utilisateur à chaque fois de remplir la matrice en donnant les cordonnées de chaque cellule à ranimer ou bien tuer ce qui cause la perte d'un grand temps et même l'effort seulement dans le remplissage de la matrice. Pour cela, la version SDL du programme a était créer afin d'éviter ces problème de la version console. D'une côté, elle peut servir à faciliter la tâche à l'utilisateur dans la modification de chaque cellule, et d'une autre côté, elle donne une meilleur lisibilité des cellules à l'utilisateur pour pouvoir facilement suivre le changement de chaque génération.

La figure ci-dessous présente la grille initiale du programme avant le remplissage :

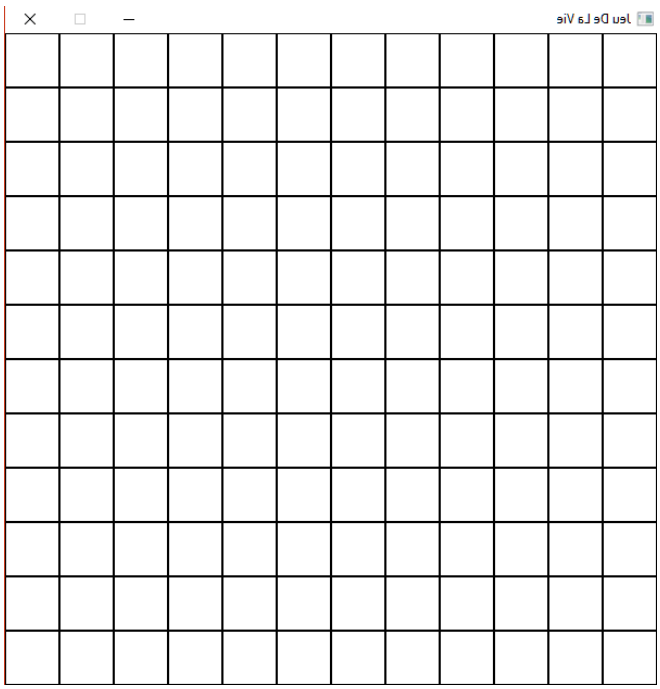

Figure 12 – Tableau avant remplissage.

Cette grille représente la matrice que l'utilisateur doit remplir dans la version console, dans le remplissage, il est suffisant pour l'utilisateur de cliquer par le bouton gauche de la souris sur n'importe quelle cellule de cette grille pour l'animer. Le programme ne donner pas seulement la possibilité à animer les cellules, mais il donne aussi la possibilité de tuer une cellule qui était déjà animer (Exemple : Cas de clic par faute sur une cellule). Après que l'utilisateur fini le remplissage, il suffit de

cliquer sur le bouton droit de la souris pour passer à la génération suivante et répéter cette opération à chaque fois qu'il veut passer à la génération suivante, avec une possibilité d'animer ou tuer des cellules dans chaque génération.

La figure ci-dessous représente la succession de 7 générations de l'exemple fourni dans l'annexe du problème en utilisant la version SDL du programme :

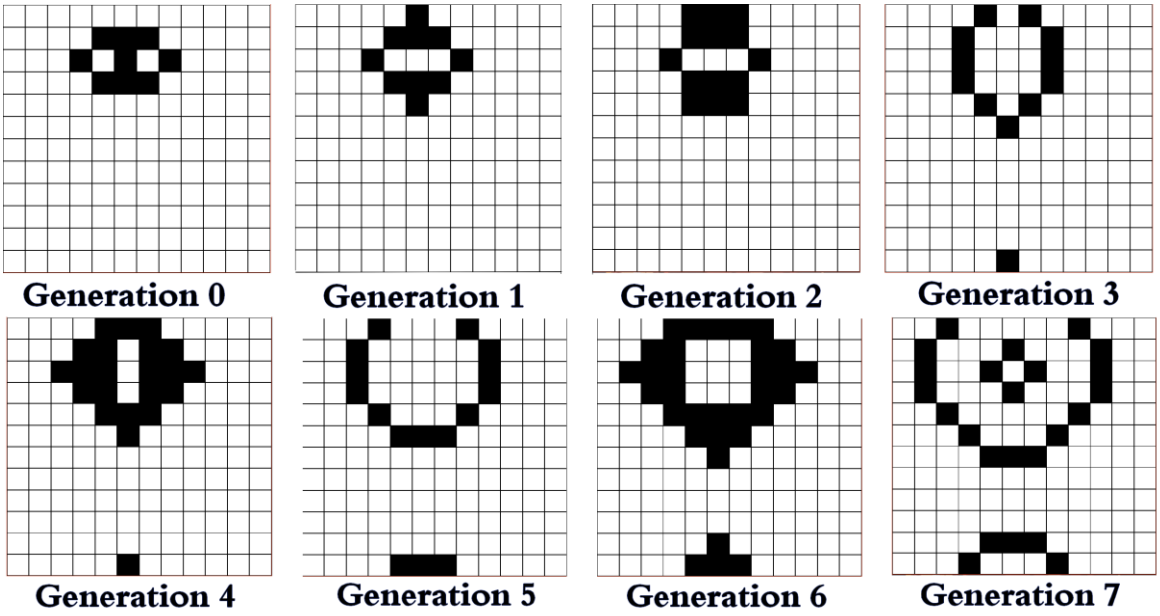

Figure 13- succession de 7 générations de l'exemple fourni dans l'annexe du problème

Conclusion et perspectives de développement

Ce projet a été une expérience nouvelle pour nous, mais malgré notre manque d'expérience en matière de programmation, de travail en équipe et de coordination, nous avons pu tirer de nombreux enseignements qui nous seront définitivement utiles dans nos futurs projets.

Evidemment, rien n'est parfait, et notre Jeu de la vie n'est pas une exception, de nombreuses idées et perspectives d'amélioration du programme n'ont pas pu aboutir à cause de notre inexpérience, notamment la déclaration de matrices prédéfinies, ou de matrices remplies aléatoirement.

Quand a la version SDL du programme, bien qu'elle puisse être considérée plus optimale que la version console car elle facilite la tâche de l'utilisateur à l'entrée et à la sortie des données, elle peut être encore optimisée en prenant en considération les pistes suivantes :

- Donner une grille avec une taille modifiable selon le choix de l'utilisateur.
- Ajouter une variable temps qui contrôle la succession des générations à la place que l'utilisateur clic à chaque fois pour passer à la génération suivante.

Ajouter un bouton « Clear » qui permet à l'utilisateur de refaire la modélisation des générations ou bien tuer tous les cellules.

Néanmoins, l'expérience acquise grâce à ce projet nous servira dans le futur.

Bibliographie

- **La communauté informatique Comment Ça Marche : www.commentcamarche.net**
- **Forum du club des développeurs et IT pro : www.developpez.net/forums**
- **Le site du Zéro : www.openclassrooms.com**
- **Polycopiés de cours de l'ENSIAS.**