**Rapport de projet**

***Jeu de la vie***

Askar Mohammad, Benaissa Adam, Ripert Alexandre

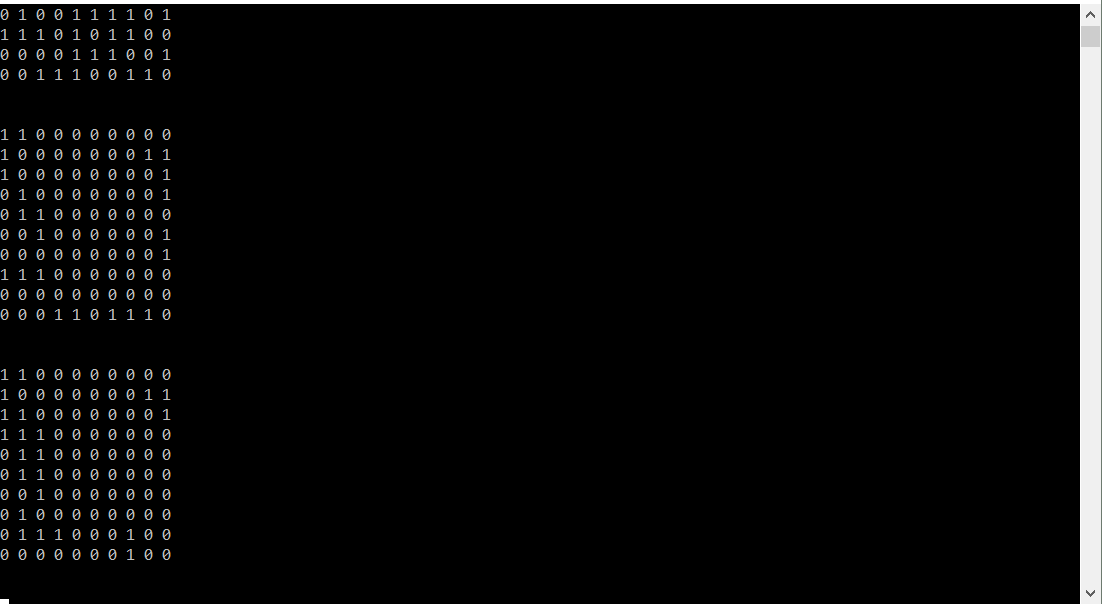


Table des matières

[Présentation du sujet 3](#_Toc534657569)

[Différentes fonctions du projet 5](#_Toc534657570)

[Les fonctionnalités implémentées 5](#_Toc534657571)

[Les fonctionnalités prévues mais non implémentés 5](#_Toc534657572)

[Les raisons des différents échecs 6](#_Toc534657573)

[Difficultés rencontrées 6](#_Toc534657574)

[Répartition du travail 7](#_Toc534657575)

[Architecture du programme 8](#_Toc534657576)

[Le langage C adapté ? 9](#_Toc534657577)

[Améliorations possibles 9](#_Toc534657578)

[Bilan 10](#_Toc534657579)

# Présentation du sujet

Le jeu de la vie est un automate cellulaire, c’est-à-dire un quadrillage contenant des « cellules » définit par un état et qui évoluent au fil du temps, qui a été inventé par John Horton Conway en 1970. Aussi, le jeu de la vie n’est pas un jeu à proprement parlé.

Le principe du jeu de la vie est d’observer l’évolution de « cellules » dans un monde généré aléatoirement. Le monde est représenté par un quadrillage, et les « cellules » par une coloration des cases. Pour cela, il faut définir deux règles : une de vie et une de mort.

**Fonctionnement par itérations**

Le jeu de la vie fonctionne par itérations, une itération étant défini par une répétition des règles du jeu, afin que les cellules évoluent.

**Règles**

Règle de vie :

Afin que les cellules puissent « vivre » et « se reproduire », il faut définir une règle qui le permet. Dans ce projet, la règle de vie est la suivante :

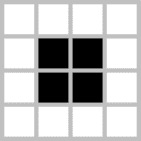
* Si une case vide est entourée de trois cases vivantes, alors elle sera vivante à la prochaine itération.
* Si une case vivante est entourée de deux ou trois cellules, alors elle sera vivante à la prochaine itération.

Règle de mort :

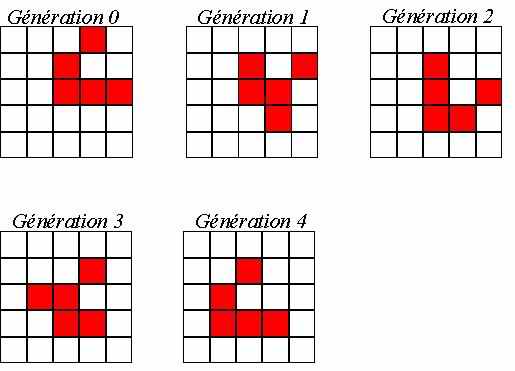
De même que pour la règle de vie, il faut que les cellules puissent « mourir ». La règle de mort est la suivante :

* Si une case vivante est entourée de zéro ou d’une case vivante, alors elle mourra de d’isolement à la prochaine itération.
* Si une case vivante est entourée de plus de trois cases vivantes, alors elle mourra de surpopulation à la prochaine itération.

**Structures**

[](https://www.google.fr/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&cad=rja&uact=8&ved=2ahUKEwibn7aGiM3eAhWjxIUKHU_nAfYQjRx6BAgBEAU&url=https://fr.wikipedia.org/wiki/Bloc_(jeu_de_la_vie)&psig=AOvVaw267cwGbR7c_Ug0GniEu0XS&ust=1542050537967858) On voit que cette structure ne peut plus évoluer. En effet, chaque case vivante est entourée de trois cases, donc elles ne peuvent pas mourir, et aucune case vide n’est entourée de trois cases vivantes, donc aucune cellule ne peut naitre.

On peut aussi au cours de l’évolution des cellules voir apparaitre des « gliders », qui sont des structures qui se répètent toutes les quatre itérations.

[](https://www.google.fr/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&cad=rja&uact=8&ved=2ahUKEwil_Pyjic3eAhWnx4UKHSNRBhoQjRx6BAgBEAU&url=http://cypris.fr/loisirs/le_jeu_de_la_vie/jeu_de_la_vie.htm&psig=AOvVaw1AYsraosD8GxTFEFndKkYR&ust=1542050804200919)

Il est évident possible de voir apparaitre d’autres formes de structures, qu’elles soient stables ou qu’elle se répètent à un certain nombre d’itérations.

# Différentes fonctions du projet

* Une fonction permettant d’initialiser une matrice : init\_matrice.
* Une fonction permettant d’afficher les matrices : aff\_matrice.
* Une fonction permettant de calculer le nombre de voisins d’une cellule : int somme.
* Une fonction permettant de passer à l’itération suivante en appliquant les règles : void pas.
* Une fonction permettant de faire un menu, comprenant le choix de la taille de la matrice, d’afficher les règles et de lancer le jeu : void menu.

# Les fonctionnalités implémentées

* Choix de la taille de la matrice, afin de définir si l’on veut un monde plus ou moins grand dans lequel vont évoluer les cellules.
* Génération aléatoire de cellules vivantes dans le monde généré.
* Définition des règles du jeu, et application de celles-ci à chaque itération.
* Passage d’une itération à une autre en appuyant sur une touche du clavier, ce qui permet de voir l’évolution des cellules itération par itération.
* Création d’un menu permettant de lancer le jeu, lire les règles et choisir la taille de la matrice.

# Les fonctionnalités prévues mais non implémentés

* Choix du pourcentage de cellules dans la matrice générée.
* Lorsqu’une cellule se trouve à une extrémité de la matrice, prise en compte des cellules de l’extrémité opposée lors de l’application des règles.
* Interface graphique à réaliser avec la bibliothèque SDL.

# Les raisons des différents échecs

Les deux premiers échecs ont la même cause. Nous n’avons pas réussi malgré plusieurs tentatives à implémenter ces fonctionnalités (choix du pourcentage et prise en compte de l’extrémité opposée), nous avons donc laissé ces fonctions de côté afin de continuer d’avancer sur le projet. Par la suite, nous avons préféré travailler sur des fonctions que nous jugions plus importantes comme la définition des règles et leur application d’itération en itération. En effet, ce sont des fonctions qui se doivent d’être opérationnelles afin que le prochain puisse tout de même fonctionner.

Le troisième échec, et sans doute le plus gros pour nous, est la non implémentation de l’interface graphique avec la SDL. Nous nous s y sommes pris trop tard et n’avons pas pu réaliser une interface graphique. C’est une erreur majeure, car sans interface graphique, il est difficile de bien observer l’évolution des cellules de manière claire et précise.

Ces échecs pu être évités si nous nous étions plus penchés dessus, le plus gros étant évidemment l’interface graphique, mais les autres laisse eux aussi un goût amer, cela aurait été un avantage si l’on avait pu choisir le pourcentage de remplissage de la matrice et que les cellules ne soient pas bloquées sur les bords.

# Difficultés rencontrées

Une des principales difficultés rencontrées fut une difficulté liée au codage, le calcul de la somme des valeurs des cases autour d'une certaine case, pour déterminer si elle sera vivante ou morte au prochain tour. Le problème venait du fait que si on prenait toutes les valeurs autour d'une certaine case et que l’on était en bord ou en coin de matrice on sortait du tableau définissant celle-ci.

Alors nous avons sollicité le professeur qui nous a donné plusieurs solutions, notamment une qui était créer une sorte de limite pour entourer nos matrices, mais après de nombreux essais, et malgré que ça fonctionnait, nous n’étions pas satisfaits du résultat. Alors nous avons essayé une autre solution qui était de prendre, quand on était tout en bas, les valeurs de tout en haut.

Mais nous ne savions pas comment faire donc nous avons laissé tomber cette idée et nous sommes partit sur un calcul cas par cas, le calcul de la somme dépendra de là où la case se situe.

Nous avons réussi à réaliser cela mais le code était trop long et brouillon donc nous avons donc demandé l'aide du professeur qui nous a aider à simplifier cela en utilisant des return et en simplifiant les conditions.

Une autre difficulté a été de se lancer. Au début, nous ne voyions pas par où commencer. Il a donc fallu un peu de temps avant que les choses se mettent en place, et il a été difficile de commencer à coder sachant que nous ne savions pas comment faire les fonctions que nous voulions. Par la suite, le projet s’est accéléré jusqu’à rencontrer la difficulté évoquée ci-dessus, de même sur la fin du projet, lorsque nous avons été confrontés à la réalisation de l’interface graphique qui n’a pas aboutie.

Une autre difficulté a été le langage C. Mise à part Mohammad, nous n’avions jamais programmé, il a donc été difficile de chercher comment réaliser certaines fonctions. Nous nous sommes pour cela se renseigner sur internet pour trouver par exemple comment afficher un tableau à deux dimensions.

# Répartition du travail

Le travail a été réparti par le chef de projet, Mohammad. Il a réalisé la fonction « aff\_matrice », permettant d’afficher la matrice, et la fonction « somme », la plus compliqué du programme, qui permet d’appliquer les règles du jeu de la vie. Pour cette dernière nous nous y sommes mis à trois, cette fonction étant celle qui nous a donné le plus de fil à retordre.

Adam a réalisé la fonction « menu », qui affiche le menu et renvoie vers le choix fait dans celui-ci, avec l’aide d’Alexandre.

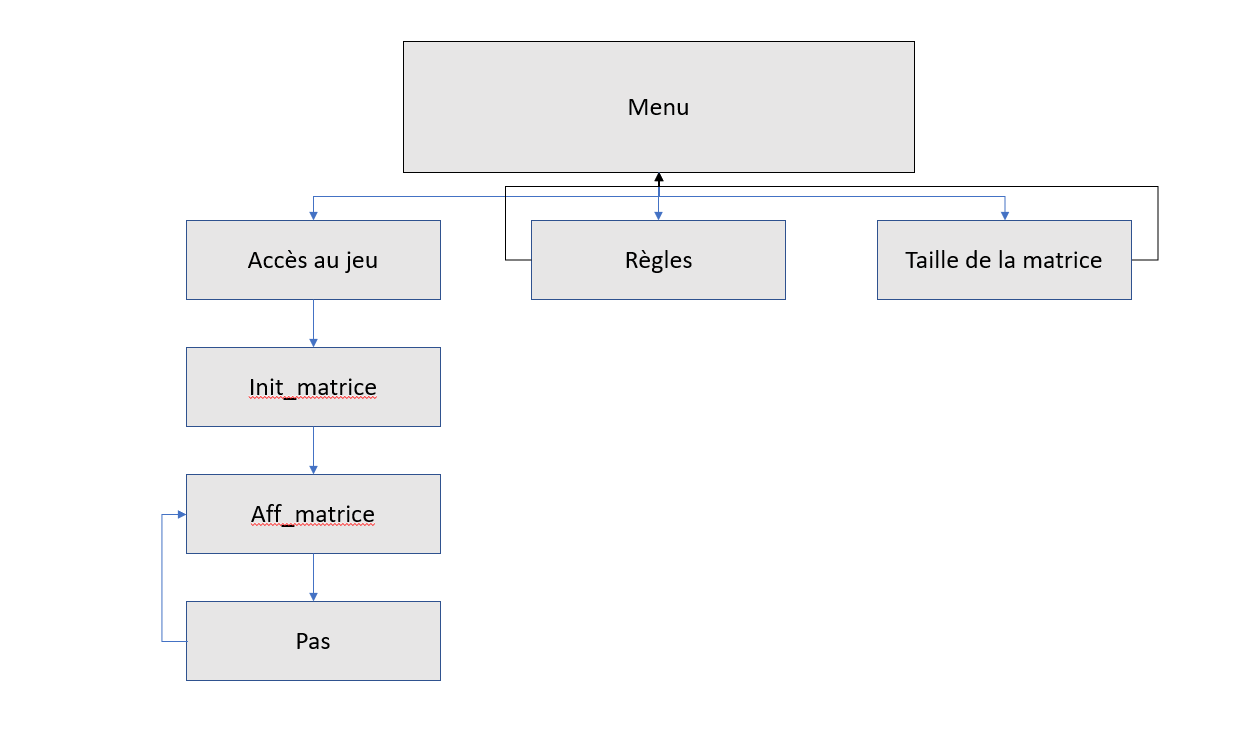
Alexandre a fait la fonction « init\_matrice », qui initialise la matrice, avec l’aide de Mohammad.

La fonction « pas » a été faite par Alexandre et Adam, avec l’aide de Mohammad.

Le travail réalisé n’est pas vraiment en accord avec ce qui a été réalisé au début. Chacun a eu besoin d’aide dans la fonction qu’il devait coder. Mohammad étant celui qui a le plus aider, il a en effet participé au codage de toutes les fonctions. Finalement, chacun a regardé un peu chaque fonction pour essayer d’y apporter son aide.

Pour ce qui a été à faire qui ne concerne pas le codage du projet, comme ce rapport, le travail a été réalisé à trois, nous n’avons pas chacun fais une partie de notre côté afin d’éviter d’éventuels désaccords lors de la rédaction, et que tout le groupe soit au courant de ce qui a été écrit et envoyé.

# Architecture du programme



# Le langage C adapté ?

Le langage C n’a pas spécialement posé de problèmes, nous avons pu réaliser les fonctions voulus, sauf deux, mais la non réalisation de ces fonctions n’est pas liée au langage de programmation. Ce langage semblait approprié afin de réaliser ce projet, il est possible que d’autres permettent de faire la même plus simplement ou plus efficacement. Lors de nos recherches, nous avons trouvé peu d’informations sur un jeu de la vie en C, il était plus question de Python par exemple. Malgré tout il a été tout à fait possible de coder un jeu de la vie en langage C.

# Améliorations possibles

Nous avons pensé à plusieurs améliorations possibles :

* Ajout d’une cellule en cliquant sur une case, voire d’un glider, ce qui permettrai par exemple de casser un équilibre atteint au fil des itérations.
* Création d’une deuxième « civilisation » et ajout de règles en conséquence. Par exemple en prenant une civilisation 1 et une civilisation 2, on peut imaginer qu’une cellule de la civilisation 1 meurt en contact de plusieurs cellules de la civilisation 2, ce qui permettrai de simuler un jeu de la vie avec la concurrence d’une autre civilisation.
* Rajouter un « mode » où les cellules ont une durée de vie, défini par un certain nombre d’itérations. On pourrait imaginer que chaque cellule à un pourcentage de mourir de plus en plus élevé à mesure que les itérations passent, on aurait alors des cellules qui vivent plus ou moins longtemps que d’autres, le tout en couplant cela avec une ou plusieurs autres civilisations présentes dans la matrice.
* Ajouter des « catastrophes », qui ont un certain pourcentage de chance d’intervenir à chaque itération, comme la mort d’un certain nombre de cellules vivantes (comme une épidémie). Possibilité de rajouter des sortes de bonus, comme une modification des règles pendant pour une itération, qui permettrai par exemple aux cellules vide de naitre si elles sont entourées de deux cellules vivantes au lieu de trois.

# Bilan

Bilan d’Alexandre :

Ce projet m’a plu car nous j’ai pu apprendre de nouvelles fonctions du langage C et de comprendre un peu mieux la logique de la programmation. Malgré tout, j’ai toujours du mal à comprendre certaines choses et c’est pour cela que j’ai eu besoin d’aide dans la réalisation des fonctions qui m’étaient assignées. Les erreurs que j’ai faites viennent d’un manque de connaissances du langage pour certaines, et d’un manque de réflexion pour d’autres. Si l’on devait refaire ce projet, je pense que je chercherai plus lorsque je rencontre une difficulté avant de demander de l’aide, je pense que ça permettra de mieux comprendre le langage. Ce dont je suis le plus fier dans ce projet est que nous avons réussi à le faire fonctionner. Au début, on n’avait pas vraiment d’idée de comment réaliser un jeu de la vie, c’est pour ça que le meilleur moment pour moi a été lorsqu’il a fonctionné pour la première fois.

Bilan de Mohammad :

J’ai trouvé ce projet fort amusant à réaliser, être un chef de projet est une chose nouvelle pour moi. Malgré mes connaissances de l’année passée acquise à travers les cours d’ISN, je me suis tout de même confronté à des difficultés notamment en programmant les matrices. Le plus gros problème a été la SDL qu’on a dû abandonner par manque de temps. Cependant je suis très satisfait de l’état actuelle du programme, même s’il peut encore être améliorer il fonctionne parfaitement. Même si au début on a dû recommencer car on a fait l’erreur de commencer à programmer le projet directement sans trop savoir comment passer d’une matrice a l’autre. Si je devais refaire le projet, je me pencherais plus sur le SDL afin de faire l’interface graphique, je pense que c’est presque indispensable pour ce projet. Ce dont je suis le plus fier est le fait que nous avons réussi malgré tout à faire un programme qui fonctionne.

Bilan d’Adam :

J’ai aimé travailler ce projet car il m’a permis d’en apprendre un peu plus le langage C et de découvrir la bibliothèque SDL, que nous n’avons malheureusement pas intégré dans le projet. N’ayant jamais fait d’informatique avant cette année, je n’avais jamais fait de projet en groupe a la différence de mes camarades. J’ai beaucoup apprécié travailler avec mes camarades qui m’ont apporté des connaissances sur différentes fonctions travaillées, notamment Mohammad qui a apporté son aide dans la création de la fonction « pas ». S’il y a une chose que j’aimerais changer si je devais refaire le projet, ce serait d’arriver à faire fonctionner la SDL, afin d’afficher une interface graphique. C’est le seul regret que je garde, mais je suis tout de même fier de ce que nous avons accompli.

# Annexe

**Cahier des charges**

**Le jeu de la vie**

Le jeu de la vie est un automate cellulaire. Il a été imaginé par John Horton Conway en 1970. Il a pour but de simuler un scénario de vie dans lequel des « cellules » sur un quadrillage vont vivre, se reproduire, mourir, selon deux règles très simples.

|  |  |
| --- | --- |
| **Fonctions** | **Critères** |
| Générer un monde aléatoire contenant des cellules | Choisir le nombre de cellule de départ (entre 0% et 100%) |
| Définir des règles | Définir des règles de vie et de mort des cellules |
| Créer une interface au jeu de la vie | Voir l’évolution des cellules au fil des itérations |

Définition des fonctions et de leurs critères :

Générer un monde aléatoire contenant des cellules

Le but est de générer un monde aléatoire contenant des cellules. Il est important que l’on puisse choisir le nombre de cellules de départ, entre 0 % et 100% (entre un monde vide et un monde plein).

Définir des règles

Il faut définir des règles de vie et de mort :

* Règles de vie : si une case vide est entourée de trois cases vivantes, alors à la prochaine itération cette case deviendra une cellule.

Si une case vivante est entourée de deux ou trois cellules, alors elle restera vivante à la prochaine itération.

* Règles de mort : si une case vivante est entourée de zéro ou une case vivante, alors elle meurt de sous-population.

Si une vivante est entourée de plus de trois cases vivantes, alors elle meurt de surpopulation.

Créer une interface au jeu de la vie

Ce qui permettra de voir l’évolution et le comportement des cellules, et les différentes formes que celles-ci peuvent prendre.

Fonctions utilisées :

menu() : Fonction affichant le menu du jeu permettant a l’utilisateur de passer au jeu directement (matrice 10 par 10 par défaut), de visionner les règles du jeu ou bien de modifier la configuration de base (taille de la matrice).

init\_matrice : Fonction d’initialisation de la matrice. La matrice initialisée par défaut est de taille 10\*10. On peut changer cette taille dans le menu.

aff\_matrice : Fonction d’affichage de la matrice, on prend comme donnée en entrée un tableau.

somme : Fonction qui sert à calculer la somme des valeurs des cases voisines. Ce qui va ensuite permettre d’appliquer les règles, car la somme du nombre de voisins permet de savoir combien on a de voisins.

pas : Fonction qui applique les règles à l’aide la fonction somme et rentre les nouvelles valeurs dans une seconde matrice. On ne modifie la matrice d’origine, on applique toujours les règles dans une seconde matrice.