Architecture des ordinateurs

TALAB Stéphane

Rapport sur le jeu de la vie

Licence Informatique - 2e année

Année universitaire 2024-2025

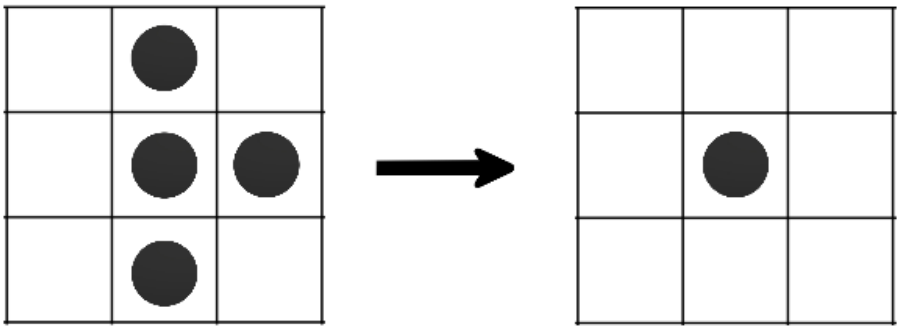
Table des matières :

Introduction

Le "Jeu de la vie", conçu par le mathématicien John Horton Conway en 1970, est un automate cellulaire largement étudié pour ses propriétés émergentes. Ce projet repose sur l'implémentation de ce modèle en assembleur 68k, un langage de programmation qui permet de travailler à un niveau très proche du matériel.

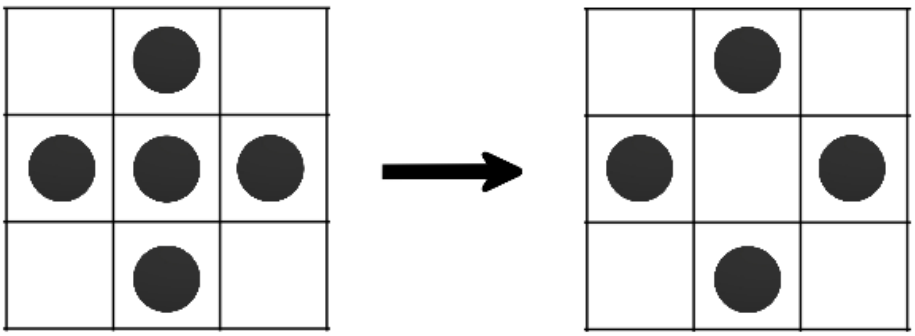
Voici les règles simples de ce jeu :

Survie
Une cellule vivante reste en vie à la génération suivante si elle est entourée exactement de deux ou trois cellules voisines.

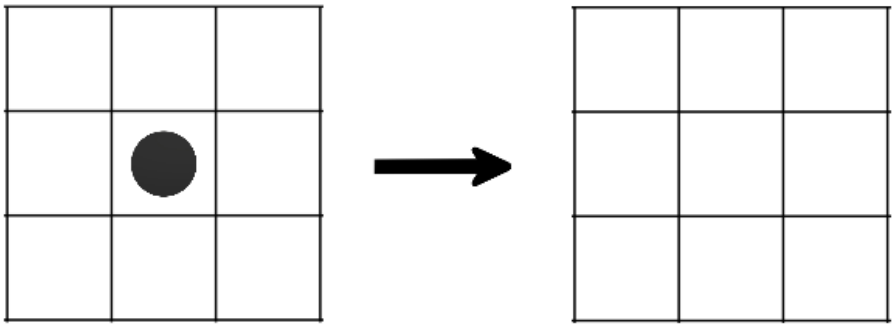


Mort
Une cellule meurt pour l'une des deux raisons suivantes :

- Elle est entourée de quatre cellules voisines ou plus, ce qui provoque sa mort par surpopulation.

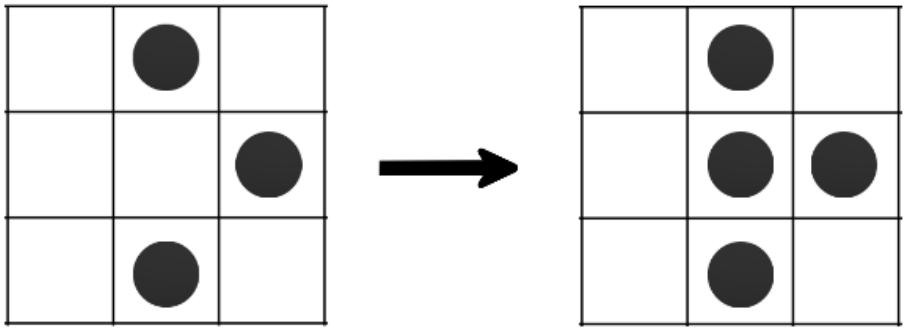


- Elle est isolée ou n’a qu’une seule voisine, entraînant sa mort par manque de connexions.



-

Naissance
Une nouvelle cellule apparaît dans une case vide si celle-ci est entourée exactement de trois cellules voisines vivantes.



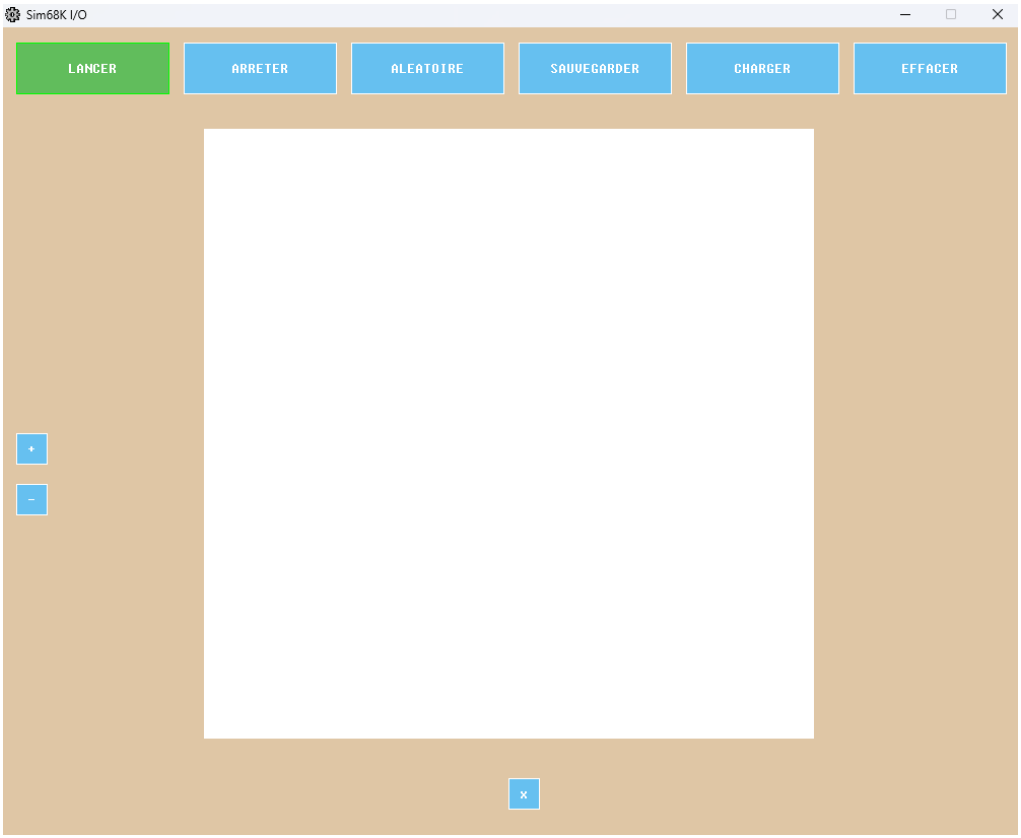
Remarques importantes

- Les cases peuvent être dans l’un des deux états suivants : **vivante** ou **morte**.
- L’état d’une cellule à la génération suivante dépend de l’état des **huit cellules voisines** qui l’entourent.

Fonctionnalités

Avant de lancer le jeu on doit placer des cellules dans la grille, pour placer une cellule il suffit de faire un clic gauche sur une case morte, et clic droit sur une case vivante pour la tuer, on peut faire glisser tout en maintenant pour placer plusieurs cellules à la fois (respectivement supprimer).

Le jeu comporte 7 fonctionnalités principales, nous allons détailler cela en dessous, voici une capture d'écran du jeu après lancement :



Comme nous pouvons voir il y a 6 boutons principaux en haut, ainsi que deux sur la gauche et un en bas.

Le bouton lancer :

Comme son nom l'indique il sert à lancer la simulation, quand il est pressé il bloque la possibilité de presser tous les boutons hormis le bouton « Arrêter » qui devient actif, cela prévient des erreurs potentielles qu'aurait pu causer la manipulation des données dans les registres utilisés par plusieurs fractions de codes.

Le bouton arrêter :

Similaire au bouton « Lancer » il sert à arrêter la simulation dans le temps pour pouvoir placer ou supprimer les cellules, enregistrer la simulation actuelle, ou simplement recommencer une nouvelle simulation.

Le bouton aléatoire :

Le dispositif de génération pseudo-aléatoire utilise trois fonctions principales. La fonction DEF_GRAINE initialise la graine à partir du temps écoulé depuis minuit, ou à 1 si le temps est égal à 0, et la stocke dans la variable SEED. La fonction DEF_RANDINT génère une valeur binaire aléatoire (0 ou 1) en manipulant la graine, qui est mise à jour à chaque appel pour produire une nouvelle séquence aléatoire. Enfin, DEF_GRILLE_ALEA initialise une grille de taille spécifiée, où elle stocke les valeurs générées par DEF_RANDINT dans chaque cellule.

Le bouton sauvegarder :

Ce bouton permet à l'utilisateur d'enregistrer l'état actuel de la simulation dans un fichier. Lorsqu'il est activé, il ouvre une interface pour choisir un emplacement et un nom de fichier. Ensuite, les données de la grille sont écrites dans ce fichier avec l'extension .schema. Cela permet de conserver les résultats et de pouvoir les recharger ultérieurement pour reprendre la simulation ou analyser les données enregistrées. Si l'utilisateur ne saisi pas l'extension .schema, le jeu se charge de l'ajouter afin de garantir l'utilisateur de retrouver sa sauvegarde.

Le bouton charger :

De la même manière que le bouton sauvegarder, ce bouton sert à charger les données d'une simulation enregistrée précédemment contenue dans un fichier .schema via l'explorateur de fichier. Quelques fichiers de démonstration ont été préalablement créés et peuvent être chargés.

Le bouton effacer :

Ce bouton sert tout simplement à vider le contenu de la grille en remettant toutes les cases à 0.

Les boutons « + » et « - » :

Ces boutons situés au milieu à gauche de la fenêtre servent à ralentir ou accélérer la simulation, le bouton « + » augmente le délai permettant ainsi de mieux voir les mises à jour de la grille, de la même sorte le bouton « - » sert lui à accélérer le jeu. En réalité le jeu est par défaut ralenti en accélérant on peut atteindre au mieux la vitesse de base, à noter aussi qu'on peut ralentir le jeu jusqu'à un certain délais au maximum prédéfinit pour éviter une latence exagérée.

Le bouton « x » :

C'est le dernier boutons utilisable situé en bas au milieu de la fenêtre, il sert à arrêter la simulation de manière définitive. Il quitte le programme.

Implémentation en assembleur 68k :

Le code principal (Main) commence par l'initialisation de la grille, en définissant la taille de la grille et des cellules. Des registres sont utilisés pour stocker ces informations. Par la suite, des « fonctions » sont utilisées pour dessiner l'interface graphique, notamment les boutons permettant à l'utilisateur d'interagir avec la simulation.

```
; --- Initialisation ---

    CLR.L   D5                      ; Reinitialise le registre D5 a 0
    CLR.L   D6                      ; Reinitialise le registre D6 a 0
    CLR.L   D7                      ; Reinitialise le registre D7 a 0
    MOVE.W  T_CEL, D5               ; Charge la taille d'une cellule dans D5
    MOVE.W  LARG, D6                ; Charge la largeur de la grille dans D6
    MOVE.W  HAUT, D7                ; Charge la hauteur de la grille dans D7

    ; Configuration de la graine pour la generation aleatoire
    JSR     DEF_GRAINE
    MOVE.L  RES, D1                 ; Charge la resolution d'ecran dans D1
    JSR     RESOLUTION

    ; --- Definir la couleur de fond ---
    MOVE.L  #$a5c6df, D1            ; Definit la couleur de fond en beige (BBVVRR)
    MOVE.L  RES, D2                 ; Charge les dimensions de resolution dans D2
    JSR     REMPLIR_BG              ; Applique la couleur blue ciel en fond sur l'ecran

    ; --- Definir les couleurs initiales du stylo et du remplissage ---
    MOVE.L  #$00FF00, D1            ; Definit la couleur du stylo en vert (BBVVRR)
    JSR     SET_PEN_COLOR

    MOVE.L  #$5cbd61, D1            ; Definit la couleur de remplissage du bouton lancer en vert
    JSR     SET_FILL_COLOR

    MOVE.L  #BOUTON_LANCER, A1
    JSR     DESSINE_BOUTON          ; Dessine le bouton "LANCER"

    ; --- Dessiner les boutons ---
    MOVE.L  #$f0c066, D1            ; Definit la couleur de remplissage des autres boutons en bleu
    JSR     SET_FILL_COLOR

    MOVE.L  #$FFFFFF, D1            ; Definit la couleur du stylo en blanc
    JSR     SET_PEN_COLOR

    MOVE.L  #BOUTON_ARRETER, A1
    JSR     DESSINE_BOUTON          ; Dessine le bouton "ARRETER"
    MOVE.L  #BOUTON_EFFACER, A1
```

Extrait du début du code principal

L’initialisation, la réinitialisation et la mise à jour de la grille se font en fonction des règles du jeu. Une fonction initialise les cellules de la grille, mettant à zéro chaque case avant d'insérer une valeur spécifiée à une position donnée. Une fonction appelée RESET permet de réinitialiser la grille en remettant toutes les cases à zéro, parcourant la grille de manière systématique pour redémarrer la simulation avec une grille vide. La fonction principale, MAJ, gère l'évolution de la grille selon les règles du jeu de la vie en parcourant chaque cellule de la grille et en calculant le nombre de voisins vivants dans un rayon de 3x3 autour de chaque cellule. En fonction du nombre de voisins les cellules évoluent : une cellule vivante reste vivante si elle a 2 ou 3 voisins, sinon elle meurt, tandis qu'une cellule morte devient vivante si elle a exactement 3 voisins vivants.

Après chaque évaluation des cellules, la grille est mise à jour, et les états des cellules vivantes ou mortes sont transférés dans une deuxième grille ce qui permet de garantir que les changements d'état ne se produisent qu'après avoir évalué toutes les cellules.

```
MAJ:                                ; Debut de la fonction MAJ
    MOVE.L  A5, PTR_TAB1            ; PTR_TAB1 = A5, sauvegarde l'adresse de la grille 1
    MOVE.L  A6, PTR_TAB2            ; PTR_TAB2 = A6, sauvegarde l'adresse de la grille 2
    ADD.L   D6, A6                  ; A6 += LARG, on ajuste A6 en fonction de la largeur
    ADD.L   #3, A6                  ; A6 += 3, ajustement supplementaire pour les calculs de cases
    CLR.L   I                       ; i = 0, on initialise l'indice de la ligne

ATTENTE:
    MOVE.L   NB_RAL, D1             ; Charger la valeur de NB_RAL dans D0
    CMP.L    RAL_VAR, D1            ; Comparer la valeur de D5 avec la valeur ï¿½ l'adresse de RAL_VAR
    BEQ      FIN_ATTENTE            ; Si les deux sont Ã©gaux, sortir de la boucle
    ADD.L    #1, RAL_VAR            ; Sinon, incrÃ©menter RAL_VAR
    BRA      ATTENTE                ; Revenir au dÃ©but de la boucle

FIN_ATTENTE:
    MOVE.L  #0, RAL_VAR             ; RÃ©initialiser RAL_VAR Ã  0

TQ_I_MAJ:                           ; Debut de la boucle sur les lignes (i)
    CMP.L   I, D7                   ; Si (i == HAUT), on sort de la boucle
    BEQ     FIN_TQ_I_MAJ            ; Si i == HAUT, on termine la boucle des lignes
    CLR.L   J                       ; j = 0, on initialise l'indice de la colonne

TQ_J_MAJ:                           ; Debut de la boucle sur les colonnes (j)
    CMP.L   J, D6                   ; Si (j == LARG), on sort de la boucle des colonnes
    BEQ     FIN_TQ_J_MAJ            ; Si j == LARG, on termine la boucle des colonnes
    CLR.B   VAR                     ; VAR = 0, initialisation de la somme des voisins vivants
    CLR.L   K                       ; k = 0, on initialise l'indice pour parcourir les voisins

TQ_K_MAJ:                           ; Debut de la boucle sur les voisins (k)
    CMP.L   #3, K                   ; Si (k == 3), on sort de la boucle des voisins
    BEQ     FIN_TQ_K_MAJ            ; Si k == 3, on termine la boucle des voisins
    CLR.L   L                       ; l = 0, on initialise l'indice pour les cases voisines

TQ_L_MAJ:                           ; Debut de la boucle pour chaque voisin (l)
    CMP.L   #3, L                   ; Si (l == 3), on sort de la boucle des voisins
    BEQ     FIN_TQ_L_MAJ            ; Si l == 3, on termine la boucle des voisins

    MOVE.B  (A5)+, D0               ; D0 = *(A5++), on charge la valeur de la case actuelle dans D0
    ADD.B   D0, VAR                 ; VAR += D0, on ajoute cette valeur a la somme des voisins
```

Extrait de la fonction MAJ

Ensuite, la fonction PRINT_GRILLE gère l’affichage de la grille à l'écran, mettant à jour l'état de chaque cellule selon la nouvelle configuration calculée. La simulation fonctionne en boucle, vérifiant constamment les actions de l'utilisateur, tout en mettant à jour la grille à chaque itération. L’utilisateur peut interagir avec la simulation en cliquant sur les boutons pour lancer, arrêter ou réinitialiser le jeu.

Les défis

Les principaux défis rencontrés lors du projet sont liés à la conception du système lui-même. L'un des plus importants défis était de concevoir un système utilisant deux grilles différentes pour la gestion de l'état du jeu et sa mise à jour. Il fallait penser à la façon d'alterner entre les grilles tout en restructurant la gestion de l'état des cellules. L'autre important défi était de gérer les boutons et la souris. Cela impliquait de savoir interagir avec l'interface mais aussi d'utiliser les boutons pour démarrer, arrêter, et contrôler l'animation mais également de décider comment la souris interagirait avec les cellules sous la forme de la grille.

Un autre point majeur était de créer un système pseudo-aléatoire permettant de simuler les cellules vivantes lors de la mise en route du jeu. Cela nécessitait de construire une distribution aléatoire tout en respectant les conditions des algorithmes s'appuyant sur des nombres déterministes notamment avec GET_TIME.

Par ailleurs il fallait vérifier les cellules dans un voisinage de cellules d'une cellule, ce qui nécessitait de construire puis de contrôler les mécanismes de vérification permettant de s'assurer que chaque cellule prenne en compte ses voisines lors de sa mise à jour. Enfin, un autre point était la gestion des fichiers externes sur lesquels ont reposait la sauvegarde des grilles permettant d'enregistrer et de lire leurs états.

Tous ces défis ont été surmontés grâce aux différentes étapes qui ont menés à ces réflexions, au début les concepts ne ressemblaient pas réellement au code final, tout ça montre les chemins de réflexions qu'on a quand on code.

Améliorations

Une fonctionnalité qui n'a pas finalement pas été intégrée est la possibilité d'afficher dynamiquement à chaque tour le nombre de cycle de vie depuis le début de la simulation courante. Ou bien d'afficher la valeur de la variable de ralentissement pour que l'utilisateur comprenne qu'il a atteint le minimum ou le maximum.

Une implémentation simplement visuelle aurait pu être d'ajouter des traits afin de diviser la grille de manière graphique pour l'utilisateur, mais cela s'est avéré par très utile et donc n'a pas été implémenté.

Conclusion

Malgré les difficultés rencontrées lors du développement de ce jeu en assembleur, qui pouvaient sembler insurmontables justement à cause du langage, nous sommes arrivés à trouver des astuces et manières pour faire tout ce qui était nécessaire au bon fonctionnement au jeu, on peut noter aussi que cette expérience était d'autre part enrichissante grâce au cheminement de réflexion qu'il y a eu entre le début de la conceptualisation et le code final.

Merci sincèrement pour le temps que vous avez accordé à lire ce rapport.