esiea
ECOLE D'INGENIEURS
DU MONDE NUMERIQUE

TP n°3

LAB1413

Jeu de la vie

Arnaud Bannier
Nicolas Bodin
Matthieu Le Berre

Ce jeu est en réalité un automate cellulaire développé dans les années 70. Il n'y a donc pas vraiment d'interactions entre un joueur et sa machine, il s'agit juste de suivre le comportement de cellules sur un damier. Les règles d'évolution sont simples mais la propagation des cellules est remarquable et peut amener de réelles surprises, comme vous pouvez en voir quelques exemples sur la page Wikipedia. Vous pouvez également trouver une application sur cette page. Au niveau scientifique, le but était de trouver des structures de base amenant à des schémas précis, une infinité d'auto-réplication ou au contraire une stabilité de ces structures. Le but de ce TP est de modéliser le jeu de la vie est de le visualiser au travers de la console.

1. Définition des règles

Le jeu se compose d'un damier et chaque case du damier (cellule) est soit morte, soit vivante (pas de chat de Schrödinger ici). Les règles de vie et de mort de chaque cellule sont définies par l'état des cellules directement voisines dans les 8 directions haut, bas, gauche, droite et les diagonales :

- Si une cellule est vivante et qu'elle possède exactement 2 ou 3 voisines vivantes, elle reste en vie.
- Si une cellule est morte et qu'elle possède exactement 3 voisines vivantes, elle devient vivante.
- Dans tous les autres cas, la cellule meurt ou reste morte.

À chaque itération, tout le damier est mis à jour. La mise à jour peut se représenter de la façon suivante : à chaque top d'horloge, un nouveau damier vide est créé, et chaque cellule de ce damier est remplie en fonction des cellules voisines du damier original. Une fois ce nouveau damier rempli, il prend la place de l'ancien pour être affiché.

2. Programmation du TP

Côté programmation, le jeu de la vie sera représenté par un tableau à 2 dimensions dont chaque case devra contenir un espace si la cellule est morte, un caractère affichable (par exemple '`#`') si elle est vivante. Il sera alors nécessaire de demander à l'utilisateur la taille du damier (largeur, hauteur) et son motif de départ puis d'initialiser le tableau. Tant que l'utilisateur n'ordonne pas l'arrêt du jeu, il faudra générer la nouvelle grille et l'afficher à l'écran. Attention, chaque nouvelle grille est une copie de l'ancienne. C'est-à-dire que lors de l'évolution d'une cellule, son nouvel état doit être copié dans une nouvelle grille.

2.1. Les tableaux à double entrée

Comme présenté précédemment, le damier est représenté par un tableau à double entrée. Pour rappel, il s'agit en C d'un tableau contenant des tableaux. Nous avons besoin de demander à l'utilisateur la taille du damier, ce qui implique d'utiliser des tableaux alloués dynamiquement. Pour plus de simplicité lors de l'utilisation de la grille, nous allons utiliser les variables globales suivantes :

- `char **grid` : notre damier,
- `int width` : la largeur de la grille,
- `int height` : la hauteur de la grille.

N'hésitez pas à solliciter l'aide de votre chargé de TP en cas de blocage lors de cette étape. N'oubliez pas également de libérer vos tableaux lorsque vous n'en avez plus besoin.

2.2. Initialisation

Vous devez donc dans un premier temps créer les fonctions suivantes :

- `int InitGame ( void )` : Affiche le menu à l'utilisateur lui permettant de choisir un motif parmi ceux proposés, et récupère la largeur de la grille, sa hauteur (en mettant à jour les variables globales `width` et `height`) et retourne le numéro du motif choisi.

— `int AllocGame ( int motif )` alloue la grille en utilisant les variables globales `width` et `height` et en mettant à jour la variable globale `grid`. La grille est ensuite initialisée avec le motif désigné en paramètres (à placer au centre de la grille). Renvoie 0 ou un nombre négatif en cas d'erreur (motif inconnu par exemple). Afin de ne pas perdre trop de temps avec cette fonction, implémentez uniquement l'initialisation du motif en ligne, les autres seront réalisés s'il vous reste du temps à la fin du projet. Voir ci-dessous pour les motifs disponibles.
— `void DrawGame ( void )` dessine la grille sur la console. Il suffit pour cela de dessiner les bords de la grille avec les symboles `|` et `-`, puis de dessiner les cases. Une case vide peut alors être symbolisée par un espace, une case pleine par un symbole de votre choix (par exemple `#`).

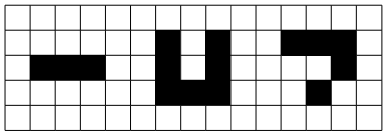

FIGURE 1 – Exemples de motifs de base : ligne, U et vaisseau

Le motif de base doit pouvoir être choisi parmi les suivants (et plus si affinités) :

— Ligne de 3 éléments : cette ligne se transforme en une colonne de trois éléments qui deviendra à nouveau une ligne au prochain tour etc.
— U : un U est dessiné (7 cellules) et son évolution est remarquable.
— Vaisseau : le vaisseau est une structure auto-réplicative de période 4 qui se déplace suivant une diagonale.
— Aléatoire : initialisez la grille de façon aléatoire.
— Aléatoire symétrie verticale : la grille est symétrique suivant un axe vertical et initialisée de façon aléatoire.
— Aléatoire symétrie horizontale : la grille est symétrique suivant un axe horizontal et initialisée de façon aléatoire.

2.3. Déroulement du jeu

Nous avons besoin pour le déroulement du jeu d'analyser les cases voisines à la cellule étudiée. Pour cela, implémentez les fonctions suivantes :

— `int IsInRange (int ligne, int colonne)` : fonction qui renvoie 1 si la case `(ligne,colonne)` appartient à la grille, 0 si elle est hors du plateau. Cette fonction sera appelée par la fonction donnée ci-dessous.
— `int NbCasesAdj (int ligne, int colonne)` : nous donne le nombre de cases adjacentes en vie autour de la cellule en position `(ligne , colonne)` dans la grille. Attention aux cellules qui sont sur les bords de la grille, elles ont moins de voisines que les autres.
— `void UpdateGame ( void )` : met à jour la grille. Utilisez pour cela une grille temporaire qui va stocker les nouveaux états puis recopiez cette grille dans la grille définie en global. Le contenu de cette fonction applique les règles de vie et de mort vues dans la partie 1 et en utilisant les deux fonctions précédemment codées.

2.4. Interactions homme machine

Nous avons enfin un jeu prêt à être déroulé, mais pour le moment aucune instruction pour le faire fonctionner. Cette partie permet de définir l'ensemble des interactions entre l'utilisateur et la machine (via une Interface Homme Machine) à inclure dans une fonction `void PlayGame ( void )`. Nous avons donc besoin de :

- demander à l'utilisateur son motif de base avec la fonction `InitGame()`.
- initialiser la grille puis l'afficher avec les fonctions `AllocGame()` et `DrawGame()`.
- demander à l'utilisateur s'il veut continuer ou non. On considère que pour dérouler le jeu, l'utilisateur doit saisir n'importe quelle touche (par exemple entrée) ou la lettre s pour stopper le jeu.
- tant que l'utilisateur demande de continuer, il faut mettre à jour le jeu et le dessiner avec les fonctions

3. Gestion des paramètres

Au lieu de demander à chaque fois au joueur la taille du plateau ou le motif de départ, nous allons l'écrire dans un fichier que nous allons passer en arguments du programme. La configuration apportera les options et évolutions suivantes :

- Taille du plateau.
- Numéro du motif pour l'initialisation du plateau.
- Gestion des bordures : un nombre (0 ou 1) permettra de savoir si le jeu contient des bords ou si la sortie sur un côté renvoie sur le côté opposé.
- Modification des règles de vie : deux tableaux d'au maximum 9 valeurs permettent de définir les règles de vie et de mort d'une cellule en fonction de son nombre de voisines vivantes. Il faut donc 2 lignes dans le fichier de configuration donnant le nombre exact de voisines vivantes permettant de dire si la cellule naît (`naissance`) ou reste en vie (`survie`).

Le fichier de configuration doit avoir la structure suivante :

```
largeur 50
hauteur 20
motif 5
bordures 1
naissance 3
survie 2 3
```

Vous devez donc dans l'ordre :

- Créer une structure nommée `Config` adaptée au problème.
- Créer une fonction `Config *InitStruct (char *path)` permettant d'allouer et remplir cette structure. Pour cela vous devez lire le fichier de configuration `path` (la fonction `fscanf()` sera la plus adaptée) et remplir votre structure.
- Modifiez votre fonction d'initialisation pour prendre en argument la structure.