Flavien GALBEZ

Mini-Rapport : Jeu de la Vie

Mon programme permet de jouer :

- Au jeu classique (avec ou sans visualisation)
- Au jeu avec variante (avec ou sans visualisation)
- Au jeu avec variante à N population (avec ou sans visualisation)

En effet, le jeu avec variante à N population englobe aussi le cas du jeu classique et le cas du jeu avec variante. La seule fonction vraiment différente est la fonction qui permet d'afficher la matrice (expliquer pourquoi ci-dessous). J'ai cependant fait le choix de laisser mes premières fonctions tel que je les avais codées initialement afin de montrer plus aisément le processus que j'ai suivi.

Explication du fonctionnement du jeu :

Jeu classique : On crée tout d'abord une grille d'origine possédant le taux de cellule vivante choisie, puis on peut débuter le jeu. A chaque nouvelle génération on crée une grille temporaire qui est une copie de la grille originale puis on affiche cette dernière. On commence alors à préparer la génération suivante : on modifie les valeurs sur la grille temporaire en appliquant les règles sur la matrice d'origine en prenant de pouvoir différencier ces valeurs aux valeurs non modifiées (donc différents de 0 ou 1) (Cette méthode permet de ne pas lire des « voisins futurs », en effet si on modifie une même grille tout en la testant alors les valeurs testées seront parfois les valeurs modifiées, ce qu'on ne veut pas). On affiche alors la grille temporaire seulement s'il s'agit du jeu avec visualisation puis on met à jour les valeurs dans la grille d'origine (les valeurs correspondant à « va mourir » et « va naître » sont remises respectivement à 0 et 1). Le programme lancera alors une nouvelle génération dès que l'utilisateur lui demande.

Jeu avec variante : Le principe de fonctionnement est le même que celui du jeu classique, il faut juste modifier les règles et adapter quelque fonction. Il faut aussi faire attention à bien différencier les cellules qui vont naitre sous la population 1 et celles qui vont naitre sous la population 2 dans la grille temporaire.

Jeu avec variante à N population : On garde encore le même principe que les deux jeux précédents mais cela doit fonctionner pour n'importe quel nombre de population, il y a donc des changements majeurs de code : Il faut être sûr de rentrer en proportion équivalente chacune des populations dans la grille d'origine. L'affichage est aussi modifié : l'affichage d'une cellule vivante d'une population d'un certain numéro correspond à ce numéro et celui d'une cellule morte qui va naitre sous une population d'un certain numéro correspond à ce numéro en négatif (et enregistrée sous la forme de ce numéro-1). 0 et -1 seront alors préservés pour enregistrer les cellules mortes et les cellules qui vont mourir.

Flavien GALBEZ

Explication du fonctionnement du programme :

Le programme demande tout d'abord les données nécessaires au jeu (nb de ligne et de colonne de la grille et taux de remplissage), en vérifiant bien qu'elles soient cohérentes et en nettoyant la console une fois leur saisie faite.

Remarque : Mon programme pourrait marcher pour des grilles de taille inférieur à 3 (cf. explication de la fonction **`AfficherValeur`**), cependant je les ai volontairement exclus car ces grilles ne présentent plus d'évolution après la génération 1 et sont donc peu intéressantes.

Une fois les données initialisé un menu s'affiche à travers la fonction **`ChoixDuJeu`**. Cette fonction va demander de saisir une lettre correspondant à un jeu tout en vérifiant encore sa cohérence (elle redemandera à l'utilisateur de saisir la lettre si cela n'est pas le cas). Elle retournera l'indice correspondant au jeu (ici un nombre entier entre 1 et 6 inclus).

Le programme après cette initialisation rentre dans sa phase de jeu. (Remarque : pour le jeu à N population, le programme demande aussi le nb de population à l'utilisateur).

Tous les jeux (**JeuClassique**, **JeuVariante**, **JeuVarianteNPopulation**) sont construits sur le même modèle :

1. On crée la matrice d'origine (**`CreationMatriceDeDepartUnePopulation`**, **`CreationMatriceDeDepartDeuxPopulations`**, **`CreationMatriceDeDepartNPopulations`**)

2. On commence ensuite la boucle :
   - On crée une matrice temporaire qui est la copie de la matrice d'origine (**`CreerMatriceTemporaire`**)
   - On modifie cette matrice selon les règles en évaluant sur la matrice d'origine (**`ModificationMatriceTemporaireJeuClassique`**, **`ModificationMatriceTemporaireJeuVariante`**, **`ModificationMatriceTemporaireJeuVarianteNPopulations`**)
   - On affiche la matrice d'origine (**`AfficherMatrice`**, **`AfficherMatriceNPopulations`**)
   - On affiche la matrice temporaire seulement dans le cas où il s'agit du jeu avec visualisation des états futurs (**`AfficherMatrice`**, **`AfficherMatriceNPopulations`**)
   - On affiche le numéro de génération
   - On affiche pour chaque population sa taille (**`CompterValeur`**)
   - On met à jour la matrice d'origine à partir de la matrice temporaire (**`EntrerLesNouvellesValeurs`**, **`EntrerLesNouvellesValeursNPopulations`**)
   - On relance la boucle pour passer à la génération suivante quand l'utilisateur appuie sur « entrer »

3. Le jeu prend fin quand l'utilisateur saisie « fin » puis appuie sur « entrer »

Flavien GALBEZ

Explication des sous fonctions :

`int[,] CreationMatriceMorte (int nbLigne, int nbColonne)` : Permet de créer une matrice composée uniquement de 0 de hauteur nbLigne et de largeur nbColonne. (Remarque : en C# la création d'une telle matrice la remplit directement de 0 mais cela n'est pas le cas dans tous les langages de programmation, il est donc préférable de la reremplir de 0 au cas où)

`int[,] CreationMatriceDeDepartUnePopulation(int nbLigne, int nbColonne, double tauxDeRemplissage)` : A partir de `CreationMatriceMorte (int nbLigne, int nbColonne)`, permet de créer une matrice composée uniquement de 0 et de 1 avec un taux de 1 équivalent au tauxDeRemplissage.

`int[,] CreationMatriceDeDepartDeuxPopulations(int nbLigne, int nbColonne, double tauxDeRemplissage)` : Même principe que la fonction précédente avec des 1 et des 2 qui sont présent en nombre égal, et on pour respectivement pour taux équivalent au tauxDeRemplissage/2

`int[,] CreationMatriceDeDepartNPopulations(int nbDePopulation, int nbLigne, int nbColonne, double tauxDeRemplissage)` : Même principe que la fonction précédente avec des entiers entre 1 & nbDePopulation.

Remarque : ces deux dernières fonctions sont codées de façon légèrement différentes. En effet **`CreationMatriceDeDepartDeuxPopulations`** favorise l'égalité parfaite entre les différentes populations (pouvant alors s'éloigné un peu plus du taux), tandis que **`CreationMatriceDeDepartNPopulations`** favorise la proximité du taux créant parfois un écart de 1 cellule entre l'effectif total de deux populations.

`int ObtenirLaValleurDeLaCase(int ligne, int colonne, int[,] matriceOriginale)` : Permet de retourner la valeur de la case même si cette dernière se trouve hors du tableau, en effet par exemple pour les lignes (mais c'est exactement la même chose pour les colonnes) si indiceLigne est négatif alors l'indice correspondant est nbLigne+indiceLigne (-1 => nbligne-1) à l'inverse si indiceLigne est supérieur à nbLigne alors l'indice correspondant est -nbLigne+indiceLigne (nbLigne => 0).

Remarque 1 : l'utilisation du do … while n'est pas une obligation cependant elle permet les grands dépassements d'indice (dans notre problème le seul cas qui nécessite un do … while est dans le cas d'une grille de dimension 1x1 … que j'avais précédemment exclu pour son intérêt limité).

Remarque 2 : cette fonction revient presque à l'usage d'un modulo en effet soit on ajoute nbLigne ou on soustrait nbLigne jusqu'à rentrer dans l'intervalle [0, nbLigne-1].

`int CompterLesEtatsVoisins(int rang, int ligne, int colonne, int valeurACompter, int[,] matriceOriginale)` : Permet de retourner le nombre de voisins de rang rang à la case [ligne, colonne] qui possèdent pour valeur valeurACompter. Pour cela on parcourt le carré de côté 1+rang et ayant pour centre la case [ligne, colonne] à l'aide de la fonction **`ObtenirLaValleurDeLaCase`**. Lors de ce parcours on prend bien soin de ne pas compter la case du milieu correspondant à la case évaluée.

`int CompterValeur(int valeurACompter, int[,] matriceOriginale)` : Permet de retourner le nombre de case égale à valeurACompter sur l'ensemble de la matrice. Il s'agit d'un simple parcours de la matrice.

`int[,] CreerMatriceTemporaire(int[,] matriceOriginale)` : Retourne une copie de la grille d'origine afin de ne pas modifier cette dernière à chaque opération.

Remarque : il faut rentrer les valeurs une à une, et ne pas mettre une simple égalité entre les deux grilles car cela lira les pointeurs ; ainsi toute modification sur une serait aussi opérée sur l'autre.

`void EntrerLesNouvellesValeurs (int[,] matriceOriginale, int[,] matriceTemporaire)` : Permet de mettre à jour la matrice d'origine d'après la matrice temporaire. Un -1 ou un 0 sur la matrice temporaire corresponds à un 0 sur la matrice d'origine, inversement un -2 ou un 1 sur la matrice temporaire corresponds à un 1 sur la matrice d'origine et un -3 ou un 2 à un 2.

`void EntrerLesNouvellesValeursNPopulations (int[,] matriceOriginale, int[,] matriceTemporaire)` : Même principe que la fonctions précédentes étendus à « l'infini ». Si la case de la matrice temporaire est égal à i<-1 alors cette même case dans la matrice d'origine sera égale à -i-1. Si i>0 alors la case sera alors égale à i. Et enfin si i=-1 ou 0 alors la case sera égale à 0.

`void ModificationMatriceTemporaireJeuClassique(int[,] matriceOriginale, int[,] matriceTemporaire)` : Permet de modifier la matrice temporaire selon les règles du jeu classique. Les voisins sont lus sur la matrice d'origine à l'aide de la fonction **`CompterLesEtatsVoisins`** et les modifications sont enregistrés sur la matrice temporaire.

`void ModificationMatriceTemporaireJeuVariante(int[,] matriceOriginale, int[,] matriceTemporaire)` : Permet de modifier la matrice temporaire selon les règles du jeu avec la variante. Il s'agit exactement du même principe que la fonction précédente avec des règles supplémentaires.

`void ModificationMatriceTemporaireJeuVarianteNPopulations(int[,] matriceOriginale, int[,] matriceTemporaire, int nbDePopulation)` : Permet de modifier la matrice temporaire selon les règles du jeu avec la variante à N populations. Il s'agit exactement du même principe que la fonction précédente étendus pour pouvoir gérer n'importe quel nombre de population.

Remarque : Dans le cas où il y a conflit pour générer une nouvelle cellule vivante entre plusieurs familles, il y a au maximum 2 population rentrant dans ce conflit (car il faut 3 voisin d'une même famille pour donner la vie or il y a seulement 6 cases voisines par case).

`static void AfficherMatrice(int[,] matrice)` : Permet d'afficher la grille sur la console de la même que le propose l'énoncé.

`static void AfficherMatriceNPopulation(int[,] matrice)` : Permet d'afficher la grille sur la console de la même que le propose l'énoncé. Les familles sont représentées selon leur numéro et les cellules mortes par un point. Les cellules devenant vivantes sont représentées par le numéro de leur futur famille précédé par un signe "-"