LE JEU DE LA VIE

Projet Tableau Algorithme

Par Thomas Carpentier et Guillaume Turcas

Projet Avril-Mai 2018

TD-K, Année 1

Comment fonctionne le programme ?

Notre programme a été codé afin de reproduire le jeu de la vie. Ce dernier est une simulation de vie de certaines populations de cellules, qui simule à la fois leurs vies, mais aussi leurs morts. Les différentes règles sont les suivantes :

R1b si une cellule vivante d'une certaine population est entourée au rang 1 de moins de 2 cellules vivantes de la même population alors elle meurt à la génération suivante (cas de sous-population).

R2b si une cellule vivante d'une certaine population est entourée au rang 1 de plus de 3 cellules vivantes de la même population alors elle meurt à la génération suivante (cas de sur-population).

R3b si une cellule morte est entourée au rang 1 exactement de 3 cellules vivantes d'une seule et même population alors elle naît à la génération suivante en faisant partie de cette population.

R4b si une cellule morte est entourée au rang 1 exactement de 3 cellules vivantes d'une première population et de 3 cellules vivantes de la seconde population alors **si** une population est plus nombreuse au voisinage de rang 2, la cellule naît à la génération suivante en faisant partie de cette population la plus nombreuse, **sinon** (en cas d'égalité des tailles de population au voisinage de rang 2), la cellule naît à la génération suivante en faisant partie de la population *de taille totale* la plus grande ; **et finalement** en cas d'égalité des tailles totales des populations la cellule reste morte.

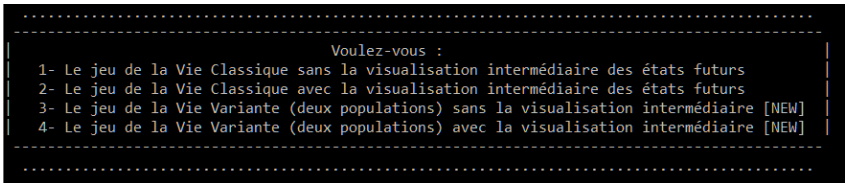
✎ dans tous les autres cas, les cellules restent dans le même état (*mort* ou *vivant*) à la génération suivante.

ϟ L'évolution à la génération suivante (t+1) se fait en même temps pour toutes les cellules (en fonction de la configuration à la génération t et de l'appliaction des règles ci-dessus).

Pour ce faire, nous avons mis l'entièreté du jeu dans une matrice d'entier nommée Grille. Cette grille qui est de taille qui est donnée par l'utilisateur dès le début de la boucle Main. Après avoir demandé le taux aléatoire de remplissage, il récupère ce dernier afin de repartir aléatoirement les cellules vivantes en fonction du taux de cellule vivante choisis. Sachant que c'est une matrice d'entier, la totalité des valeurs à l'origine est de 0, nous avons choisis cette valeur pour dire qu'une cellule est morte. Si la valeur est 1, la cellule est en vie. Si elle est de 2 ou 3, alors la cellule est en train de naitre ou en train de mourir.

```
if (Grille[i, j] == 0) { Console.Write(". "); } //Cellule Morte
if (Grille[i, j] == 1) { Console.Write("# "); } //Cellule Vivante
if (Grille[i, j] == 2) { Console.Write("* "); } //Cellule qui va naitre
if (Grille[i, j] == 3) { Console.Write("- "); } //Cellule qui va mourir
if (Grille[i, j] == 11) { Console.Write("¤ "); } //Cellule Vivante Population 2
```

Après avoir entré ces valeurs, et avant que le jeu commence, on donne un menu à l'utilisateur, où il a quatre choix possibles :



Les deux premiers choix correspondent à l'étape 1, donc à savoir une simulation avec une seule population. Les deux autres correspondant à l'étape 2, sont le même jeu, mais cette fois-ci avec deux populations, et les changements qu'elle accompagne. Chaque étape est proposée avec, ou sans visualisation intermédiaire.

Pour ce qui est du code dans tout cela, nous avons pour cela créer deux matrices différentes (ou trois lorsqu'on demande une visualisation). La première, la principale où se déroule le jeu, et la deuxième qui permet de compter le nombre de cellule entourant une cellule étudiée, pendant que la principale se fait modifier. S'il y'a les visualisations intermédiaires, il y'a alors une troisième matrice, qui permet d'afficher la matrice intermédiaire, sans que la principale soit modifiée pour autant (ce qui était facultatif dans l'étape 1, mais impossible lorsqu'on a deux populations ou plus). Ensuite le jeu affiche la matrice modifiée, et affiche la génération et la taille des populations.

Il s'arrête lorsque le jeu est stabilisé. Donc lorsque la matrice de la génération précédente est exactement la même que la matrice suivante. (Donc quand la matrice provisoire est égale à la matrice principale).

Exemple

grille de départ 3 x 5 cellules, taux de remplissage 20% :

```
.....
.###.
.....
```

Affichage sans visualisation des états futurs :

```
.....      ..#..      .....
.###.      ..#..      .###.
.....  →   ..#..  →   .....  →  etc.

 t0         t1         t2
```

Affichage avec visualisation des états futurs :

```
.....      ..-..      ..#..      ..*..      .....
.###.      .*#*.      ..#..      .-#-.      .###.
.....  →   ..-..  →   ..#..  →   ..*..  →   .....  →  etc.

 t0         t0+        t1         t1+        t2
```

EXPLICATION DES DIFFERENTES FONCTIONS

INITIALISATION

Cette fonction est primordiale pour le programme. Elle prend en compte la matrice, le taux de variation qu'a choisis l'utilisateur dans le Main, et le mode de jeu. Après avoir créé un générateur aléatoire, ce dernier va étudier l'entièreté des cellules de la matrice. Sachant que la matrice d'arrivé est la matrice initiale, donc toutes les valeurs des cellules sont égales à 0. Donc ce sont tous des cellules mortes. Alors la fonction va changer aléatoirement les cellules afin d'en créer des vivantes en fonction de ce taux. Comme vous pouvez le voir ci-dessous, peu importe le mode de jeu, la fonction va fonctionner de la même manière. Le générateur aléatoire

prend une valeur entre 0 et 10. On multiplie le taux de variation T par 10 (pour des raisons pratiques lors de la création du code) afin de permettre de répartir comme il faut. Car nous avons une chance sur T, que la valeur aléatoire soit en dessous de T, donc si c'est le cas, la cellule étudiée à ce moment-là passe de morte à vivante. La petite variante est lorsqu'il y'a deux générations. Pour cela nous avons créé une variable Repart initialisée à 0. Et lorsque nous sommes dans la boucle dédiée aux modes possédant deux populations, alors elle change une fois sur deux afin qu'une fois sur deux, la cellule créée appartient à une population, et une fois sur deux, elle appartient à l'autre (pas d'aléatoire cette fois-ci, pour une répartition équitable).

```
int Repart = 0; //Permet de répartir équitablement au début les deux populations
int L = Grille.GetLength(0);
int C = Grille.GetLength(1);
Random Generateur = new Random();
for (int i = 0; i < L; i++)
{
    for (int j = 0; j < C; j++)
    {
        double Alea = Generateur.Next(0, 10);
        if (Alea < T * 10)
        {
            if (Visualisation == 4 || Visualisation == 3) //Cas si l'utilisateur veut deux pupulations sur la grille
            {
                int Repart1 = Repart; //permet de créer une variable provisoire afin que Repart n'aille pas dans les deux boucles si Repart = 0
                if (Repart == 0)
                {
                    Grille[i, j] = 11; //créé une cellule de la génération 1
                    Repart1++;
                }
                else
                {
                    Grille[i, j] = 1; //créé une cellule de la génération 2
                    Repart1--;
                }
                Repart = Repart1;
            }
            //Ainsi, une fois sur deux, la grille créera une cellule vivante de la population 1, et une fois sur deux, il créera une cellule vivante de la population 2
            if (Visualisation == 1 || Visualisation == 2)
            {
                Grille[i, j] = 1; //Cas si l'utilisateur ne veut qu'une seule population sur la grille
            }
        }
    }
}
```

AFFICHERGRILLE

Cette fonction permet d'afficher la matrice. La matrice appelé Grille, est appelé, et toutes ces cellules sont analysés par la fonction et affiche sur la console, une figure en fonction des règles déjà expliqués précédemment. Vu que la grille de l'affichement provisoire est indépendante est différente de la matrice principale, on peut garder les boucles if qui permettent d'afficher les cellules provisoires qu'importe le mode de jeu choisis. Nous avons aussi mis la possibilité de mettre un cadrillage entourant chaque, que nous avons caché afin d'afficher un maximum de cellule sur l'ensemble de l'écran. La fonction qui suit, le AfficherLigne est complémentaire à AfficherGrille, et sert pour le cadrillage afin de mettre les lignes horizontales du cadrillage.

TAILLE

Cette fonction, prenant en compte la matrice et la population étudiée, permet de compter le nombre de cellule vivante dans cette dernière. Elle retourne donc la taille de la population.

SITUATION

Cette fonction prenant en compte la matrice principale, le mode de jeu, la matrice provisoire des modes intermédiaires, et du numéro de la génération permet de prévenir à quelle génération l'utilisateur est, et le nombre de cellule vivante composantes chaque population.

LEJEUDELAVIE

Cette fonction est la plus importante car elle correspond au fonctionnement du jeu. Toutes celles qui suivent permettent de répartir certaines fonctions afin que tout ne soit pas sur la même. Elle prend en compte uniquement le mode de jeu et la matrice principale. Son fonctionne se fait sur le principe suivant : avant de lancer la boucle Tant Que qui correspondra au jeu, la fonction créé deux matrices provisoires. Une qui sera utilisé pour compter les cellules vivantes autour d'une cellule étudiée pendant que la principale sera modifiée, et une autre qui sera utile afin d'afficher les visualisations intermédiaires (uniquement les valeurs « en train de naitre » et « en train de mourir » sont prises en compte dans cette cellule, en plus des cellules vivantes qui ne sont pas modifiés durant cette génération). La boucle tant que quant à elle, fonctionne tant que le jeu n'est pas terminé par le biais à un booléen. Ce booléen n'est vrai que lorsque la génération précédente est exactement la même que la génération actuelle (cas de stabilisation).

```
static void LeJeuxDeLaVie(int[,] Grille, int Visualisation)
{
    Random Generateur = new Random();
    int G = 0; //Initialise la génération 0
    int L = Grille.GetLength(0);
    int C = Grille.GetLength(1);
    bool FinDuJeu = false;
    int[,] GrilleProv = new int[L,C]; //Création d'une matrice provisoire qui sert à être analyser, sans être modifier en même temps
    int[,] GrilleProv2 = new int[L, C]; //Création d'une matrice provisoire qui conserve la visualisation provisoire
    while(FinDuJeu == false)
    {
        G++;
        int taille = Taille(Grille, 0);
        if (taille == 0 || GrilleProv == Grille) //Fin du jeu si c'est stabilisé
        {
            FinDuJeu = true;
        }
```

Ensuite, le jeu étudie chaque cellule et les modifie en fonction des règles établies au-dessus. Elle prend en compte la règle : « égalité des tailles de population au voisinage de rang 2, alors la cellule naît à la génération suivante en faisant partie de manière aléatoire de l'une ou l'autre des populations ». Pour ce faire, il y'a un générateur aléatoire créer au début de la fonction qui possède une chance sur deux pour donner raison à une population ou à l'autre. Pour ce qui est de compter les cellules vivantes autour de la cellule étudiée, nous avons implémentés quatre compteurs en fonction de la population, et jusqu'à quel rang il était demandé. A la fin de chaque tour, est lancé la fonction Visualisation, et AfficherGrille. Il est aussi utilisé la fonction CopieMatrice, qui permet à chaque tour, de copier la matrice principale, pour mettre à jour les deux matrices provisoires avant leurs utilisations.

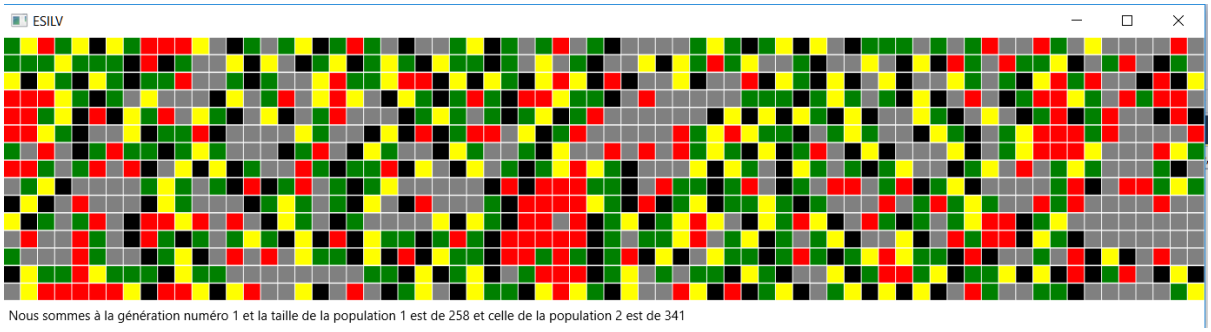
COMPTERAUTOURCELLULE

Cette fonction prend en compte la matrice provisoire qui est utilisé, les coordonnées de la cellule étudiée, de la taille de la matrice, le rang que l'on veut étudier, et aussi la population correspondante à la cellule. Elle compte toutes les cellules vivantes autour et prend en compte lorsque cette dernière se trouve sur une extrémité, et prend l'extrémité opposé pour continuer à compter.

```
{
    int Nombre = 0;
    for (int x = i - Rang; x < i + Rang + 1; x++)
    {
        for (int y = j - Rang; y < j + Rang + 1; y++)
        {
            //Procédons avec les différents cas possibles
            int tempX = x; //au cas où l'on change x pour le remettre comme on l'a eu pour ne pas avoir de problème
            int tempY = y; //au cas où l'on change y pour le remettre comme on l'a eu pour ne pas avoir de problème
            //Si la cellule étudiée se trouve dans l'extrémité de la matrice, alors :
            if (x < 0) { x = L + x; } if (x >= L) { x = x - L; } if (y < 0) { y = C + y; } if (y >= C) { y = y - C; }
            int VOM = GrilleProv[x, y]; //VOM pour vivante ou morte, donc prend la valeure 0 ou 1
            if (VOM == 1 && Popul == 1 && (x != i || y != j)) { Nombre++; } //Cas Population 1
            if (VOM == 11 && Popul == 2 && (x != i || y != j)) { Nombre++; } //Cas Population 2
            x = tempX;
            y = tempY;
        }
    }
    return Nombre;
}
```

L'INTERFACE GRAPHIQUE GUI

Nous avons intégré, dans une deuxième version du programme, l'interface graphique GUI. Nous n'avons que suivis le protocole, et les modifications dans le programme sont assez simples. Remplacer les AfficherGrille par des « gui.Rafraichir(); » , avec cette variable gui en plus, qui correspond à l'interface graphique. Nous n'avons pas rajouté de fonctions supplémentaires, et les seules modifications faites, sont pour faire rentrer l'interface graphique dont vous allez avoir un exemple juste en dessous. Le gris représente les morts, le noir la population 1, le rouge ce qui va mourir, le vert ce qui va naitre et le jaune la population 2. Si nous avons mis des « // » devant les fonctions AfficherLigne, c'est pour si l'utilisateur veut en même temps afficher sur la console, il peut en les enlevant.



Nous vous avons mis à disposition à la fois le fichier avec et sans la visualisation avec l'interface graphique.

En vous remerciant d'avoir consacré du temps à notre travail.