Morgane Sjk
ESILV A1
2018

RAPPORT PROJET FINAL
JEU DE LA VIE

FONCTION PRINCIPALE COMMUNE AUX 2 ETAPES : VOID JEUDELAVIE()

Au lancement du programme, plusieurs valeurs sont demandées à l'utilisateur : le nombre de lignes et de colonnes de la grille, le taux de remplissage de cellules vivantes, ainsi que la taille d'une cellule (car j'utilise la GUI). Ensuite, un menu s'affiche proposant 4 types de Jeu de la vie. Selon l'entier (entre 1 et 4) rentré par l'utilisateur, une des 4 fonctions JeuDeLaVie1, JeuDeLaVie2, JeuDeLaVie3 ou JeuDeLaVie4 est appelée.

```
static void JeuDeLaVie()
{
    int nombreLignes = SaisirNombreLignes();
    int nombreColonnes = SaisirNombreColonnes();
    double tauxRemplissage = SaisirTauxRemplissage();
    int nombreCellulesVivantesAuDemarrage = Convert.ToInt32(tauxRemplissage * nombreLignes * nombreColonnes);
    int taillecellule = SaisirTailleCellule();
    Console.WriteLine("Vous voulez:");
    Console.WriteLine("1) un jeu DLV classique SANS visualisation intermédiaire des états futurs (taper 1)");
    Console.WriteLine("2) un jeu DLV classique AVEC visualisation intermédiaire des états futurs (taper 2)");
    Console.WriteLine("3) un jeu DLV variante (2 populations) SANS visualisation des états futurs (taper 3)");
    Console.WriteLine("4) un jeu DLV variante (2 populations) AVEC visualisation des états futurs (taper 4)");
    int type = Convert.ToInt32(Console.ReadLine());
    if (type == 1)
    {
        JeuDeLaVie1(nombreLignes, nombreColonnes, nombreCellulesVivantesAuDemarrage, taillecellule);
    }
    if (type == 2)
    {
        JeuDeLaVie2(nombreLignes, nombreColonnes, nombreCellulesVivantesAuDemarrage, taillecellule);
    }
    if (type == 3)
    {
        JeuDeLaVie3(nombreLignes, nombreColonnes, nombreCellulesVivantesAuDemarrage, taillecellule);
    }
    if (type==4)
    {
        JeuDeLaVie4(nombreLignes, nombreColonnes, nombreCellulesVivantesAuDemarrage, taillecellule);
    }
}
```

Description des différentes sous-fonctions utilisées :

- **int SaisirNombreLignes() :**

Cette fonction renvoyant un entier (le nombre de lignes de la grille) permet de demander à l'utilisateur le nombre de lignes qu'il veut avoir dans la grille.

-Le « try » permet de faire un test : convertir en entier ce qui est saisi par l'utilisateur. Si cette conversion ne peut pas se faire, alors

-Le "catch" permet de redemander la saisie en précisant qu'il faut que la saisie soit un entier supérieur à 0.

-Le « while » recommence le « do » constitué du « try » et du « catch » tant que la saisie au clavier n'est pas un nombre positif.

```
static int SaisirNombreLignes()
{
    int nombreLignes = 0;
    do
    {
        Console.WriteLine("Quel est le nombre de lignes de la grille ? ");
        try
        {
            nombreLignes = Convert.ToInt32(Console.ReadLine());  //teste si on peut convertir le nombre saisi en entier
        }
        catch
        {
            Console.WriteLine("Merci de rentrer un nombre entier > 0");
        }
    } while (nombreLignes <= 0);
    return nombreLignes;
}
```

- **int SaisirNombreColonnes() :**

Cette fonction renvoyant un entier (le nombre de colonnes de la grille) permet de demander à l'utilisateur le nombre de colonnes qu'il veut avoir dans la grille. Son fonctionnement est le même que pour la fonction int SaisirNombreLignes.

```
static int SaisirNombreColonnes()
{
    int nombreColonnes = 0;
    do
    {
        Console.WriteLine("Quel est le nombre de colonnes de la grille ? ");
        try
        {
            nombreColonnes = Convert.ToInt32(Console.ReadLine());  //teste si on peut convertir le nombre saisi en entier
        }
        catch
        {
            Console.WriteLine("Merci de rentrer un nombre entier > 0");
        }
    } while (nombreColonnes <= 0);
    return nombreColonnes;
}
```

- **double SaisirTauxRemplissage() :**

Cette fonction renvoyant une valeur décimale permet à l'utilisateur de demander un taux de remplissage de cellules vivantes dans la grille. Le taux de remplissage est compris entre 0,1 et 0,9 car il s'agit d'un pourcentage.

Le « try » teste si l'on peut convertir en double ce qui a été saisi par l'utilisateur. Si ce n'est pas le cas alors le « catch » permet de redemander la saisie du nombre à l'utilisateur. Le « while », une fois que la conversion a pu se faire, s'assure que le nombre est bien compris entre 0,1 et 0,9.

```
static double SaisirTauxRemplissage()
{
    double tauxRemplissage = 0;
    do
    {
        Console.WriteLine("Quel est le taux de remplissage de cellules vivantes au départ ? (donner un réel compris entre 0,1 et 0,9)");
        try
        {
            tauxRemplissage = Convert.ToDouble(Console.ReadLine()); //teste si on peut convertir le nombre saisi en double
        }
        catch
        {
            Console.WriteLine("Merci de rentrer un nombre compris entre 0,1 et 0,9");
        }
    } while ((tauxRemplissage < 0.1) || (tauxRemplissage > 0.9));
    return tauxRemplissage;
}
```

- **int SaisirTailleCellule() :**

Cette fonction renvoyant un entier (la taille d'une cellule) permet à l'utilisateur de saisir la taille d'une cellule (en pixels) grâce à un nombre entier donné. Cette fonction est utilisée car la GUI a été rajoutée à mon programme. Son fonctionnement est le même que pour la fonction int SaisirNombreLignes.

```
static int SaisirTailleCellule()
{
    int taillecellule = 0;
    do
    {
        Console.WriteLine("Quelle est la taille d'une cellule ? (en pixels)");
        try
        {
            taillecellule = Convert.ToInt32(Console.ReadLine());  //teste si on peut convertir le nombre saisi en entier
        }
        catch
        {
            Console.WriteLine("Merci de rentrer un nombre entier > 0");
        }
    } while (taillecellule <= 0);
    return taillecellule;
}
```

ETAPE 1 : UNE POPULATION (FONCTIONS JEUDELAVIE1 ET JEUDELAVIE2)

Description des 2 fonctions principales :

- **void JeuDeLaVie1(int nombreLignes, int nombreColonnes, int nombreCellulesVivantesAuDemarrage, int taillecellule) :**

Cette fonction est utilisée dans le cas où l'utilisateur souhaite un Jeu DLV classique sans visualisation intermédiaire des états futurs (1). Elle fait appel à plusieurs sous-fonctions.

-La grille est créée et elle est affichée avec le nom de la génération (à savoir 0) et le nombre de cellules vivantes.

-Une boucle for permet de passer aux générations suivantes. Pour chaque génération sont affichés la grille, le numéro de la génération ainsi que la taille de la population (nombre de cellules vivantes).

Une grille "intermédiaire" qui ne sera pas affichée permet de savoir si une cellule va mourir (3) ou devenir vivante (2) sans interférer dans le comptage des cellules vivantes à la génération actuelle. Cette grille sera par ailleurs affichée si l'utilisateur demande la visualisation intermédiaire.

Ensuite, cette grille intermédiaire composée de 0,1,2 et 3 est transformée en grille de 0 et de 1 correspondant à la génération suivante. Le 2 (cellule à naître) devient un 1 (cellule vivante) et le 3 (cellule à mourir) devient un 0 (cellule morte). La nouvelle génération est alors atteinte.

```
static void JeuDeLaVie1(int nombreLignes, int nombreColonnes, int nombreCellulesVivantesAuDemarrage, int taillecellule)

{
    int[,] grille = CreationGrille(nombreLignes, nombreColonnes, nombreCellulesVivantesAuDemarrage);
    Fenetre gui = new Fenetre(grille, taillecellule, 0, 0, "Génération : 0");
    gui.RafraichirTout();
    gui.changerMessage("Génération : 0 " + "  Nombre de cellules vivantes : " + TaillePopulation(grille));
    Console.WriteLine("Génération : 0");
    AfficherGrille(grille);
    Console.WriteLine("Nombre de cellules vivantes : " + TaillePopulation(grille));
    Console.ReadLine();
    Console.WriteLine();
    for (int k = 1; k < 20; k++)
    {
        Console.WriteLine("Génération : " + k);
        for (int i = 0; i < nombreLignes; i++)
        {
            for (int j = 0; j < nombreColonnes; j++)
            {
                if ((CelluleDevientVivante(grille, i, j)) && (grille[i, j] == 0))
                {
                    // une cellule morte => une cellule à naître
                    grille[i, j] = 2;
                }
                if ((!CelluleDevientVivante(grille, i, j)) && (grille[i, j] == 1))
                {
                    // une cellule vivante => une cellule à mourir
                    grille[i, j] = 3;
                }
            }
        }
        for (int i = 0; i < nombreLignes; i++)
        {
            for (int j = 0; j < nombreColonnes; j++)
            {
                if (grille[i, j] == 2)
                {
                    // une cellule à naître => cellule vivante
                    grille[i, j] = 1;
                }
                if (grille[i, j] == 3)
                {
                    // une cellule à mourir => cellule morte
                    grille[i, j] = 0;
                }
            }
        }
        AfficherGrille(grille);
        gui.RafraichirTout();
        gui.changerMessage("Génération : " + k + "  Nombre de cellules vivantes : " + TaillePopulation(grille));

        Console.WriteLine("Nombre de cellules vivantes : " + TaillePopulation(grille));
        Console.ReadLine();
        Console.WriteLine();
    }
}
```

- **void JeuDeLaVie2(int nombreLignes, int nombreColonnes, int nombreCellulesVivantesAuDemarrage, int taillecellule) :**

Cette fonction est utilisée dans le cas où l'utilisateur souhaite un Jeu DLV classique avec visualisation intermédiaire des états futurs. Elle a un comportement quasiment similaire à la fonction void JeuDeLaVie1.

La boucle commence dès la génération 0. A chaque tour, la grille à la génération k est affichée ainsi que le numéro de cette génération et la taille de la population. Ensuite, la grille intermédiaire dont le rôle est expliqué ci-dessus est affiché avec l'inscription "génération k+". Enfin la grille de la génération suivante est affichée et ainsi de suite...

```
static void JeuDeLaVie2(int nombreLignes, int nombreColonnes, int nombreCellulesVivantesAuDemarrage, int taillecellule)
{
    int[,] grille = CreationGrille(nombreLignes, nombreColonnes, nombreCellulesVivantesAuDemarrage);
    Fenetre gui = new Fenetre(grille, taillecellule, 0, 0, "Génération : 0");

    for (int k = 0; k < 20; k++)
    {
        gui.RafraichirTout();
        gui.changerMessage("Génération : " + k + "  Nombre de cellules vivantes : " + TaillePopulation(grille));
        Console.WriteLine("Génération : " + k);
        AfficherGrille(grille);
        Console.WriteLine("Nombre de cellules vivantes : " + TaillePopulation(grille));
        Console.ReadLine();
        for (int i = 0; i < nombreLignes; i++)
        {
            for (int j = 0; j < nombreColonnes; j++)
            {
                if ((CelluleDevientVivante(grille, i, j)) && (grille[i, j] == 0))
                {
                    // une cellule morte => une cellule à naître
                    grille[i, j] = 2;
                }
                if ((!CelluleDevientVivante(grille, i, j)) && (grille[i, j] == 1))
                {
                    // une cellule vivante => une cellule à mourir
                    grille[i, j] = 3;
                }
            }
        }
        Console.WriteLine("Génération : " + k + "+");
        AfficherGrille(grille);
        gui.RafraichirTout();
        gui.changerMessage("Génération : " + k + " + ");

        for (int i = 0; i < nombreLignes; i++)
        {
            for (int j = 0; j < nombreColonnes; j++)
            {
                if (grille[i, j] == 2)
                {
                    // cellule à naître =>  cellule vivante
                    grille[i, j] = 1;
                }
                if (grille[i, j] == 3)
                {
                    // cellule à mourir => cellule morte
                    grille[i, j] = 0;
                }
            }
        }
        Console.ReadLine();
        Console.WriteLine();
    }
}
```

Description des différentes sous-fonctions utilisées :

- **int[,] CreationGrille(int nombreLignes, int nombreColonnes, int nombreCellulesVivantesAuDemarrage) :**

Cette fonction, qui prend en paramètres le nombre de lignes, le nombre de colonnes et le nombre de cellules vivantes au démarrage, retourne la grille correspondant à la génération 0. Elle permet de générer une grille d'entiers (contenant des 0 ou 1), où "0" représente une cellule morte et "1" représente une cellule vivante. Les cellules vivantes, dont le nombre est déterminé grâce au taux de remplissage de cellules vivantes au démarrage demandé à l'utilisateur, sont placées aléatoirement dans la grille.

-On crée un tableau (int [] positionCellulesVivantesAuDemarrage) qui contiendra les positions des cellules vivantes à la génération 0. Le tableau est initialisé avec des –1. Si les valeurs n'avaient pas été initialisées, la position 0 pour les cellules vivantes n'aurait jamais pu être utilisée car par défaut le tableau est initialisé avec des 0.

```
static int[,] CreationGrille(int nombreLignes, int nombreColonnes, int nombreCellulesVivantesAuDemarrage)
{
    Random generateur = new Random(); //générateur aléatoire
    int[,] grille = new int[nombreLignes, nombreColonnes];
    int[] positionCellulesVivantes = new int[nombreCellulesVivantesAuDemarrage]; //tableau qui va contenir les positions des cellules vivantes
    int position;
    int positionCourante = 0;
    bool celluleVivante;
    bool celluleOccupee;

    // Initialisation du tableau avec des -1
    for (int i = 0; i < nombreCellulesVivantesAuDemarrage; i++)
    {
        positionCellulesVivantes[i] = -1;
    }
```

-On génère aléatoirement les positions des cellules vivantes. A chaque génération aléatoire d'une position, on regarde si cette position n'a pas déjà été générée grâce à la variable bool celluleOccupee. Si la cellule va déjà être occupée par une cellule vivante car la position a déjà été générée, alors on fait i-1 pour refaire un tour de boucle pour avoir une autre position.

```
// Génération aléatoire des positions des cellules vivantes
for (int i = 0; i < nombreCellulesVivantesAuDemarrage; i++)
{
    position = generateur.Next(1, (nombreLignes * nombreColonnes) + 1); //position appartient à [1,*]
    position--; //on aura donc une position qui appartient à l'intervalle [0,*-1]
    // Verifier que cette position n'est pas déjà occupée par une cellule vivante
    celluleOccupee = false;
    for (int j = 0; j < nombreCellulesVivantesAuDemarrage; j++)
    {
        if (positionCellulesVivantes[j] == position)
        {
            celluleOccupee = true;
            break;
        }
    }

    if (!celluleOccupee)
    {
        positionCellulesVivantes[i] = position;
    }
    else
    {
        // refaire un tour supplémentaire car position déjà   occupée par une cellule vivante
        i--;
    }
}
```

-On fait le tour de la grille en attribuant la valeur 1 à une case dont la position correspond à une position de cellule vivante (bool celluleVivante=true) et 0 sinon. Pour que les positions générées correspondent à des positions d'une grille, on passe par l'intermédiaire de la variable int positionCourante.

```
    for (int i = 0; i < grille.GetLength(0); i++)
    {
        for (int j = 0; j < grille.GetLength(1); j++)
        {
            // Est-ce que ma position courante est une cellule vivante ?
            celluleVivante = false;
            for (int k = 0; k < nombreCellulesVivantesAuDemarrage; k++)
            {
                if (positionCellulesVivantes[k] == positionCourante)
                {
                    celluleVivante = true;
                    break;
                }
            }
            if (celluleVivante)
            {
                grille[i, j] = 1;
            }
            else
            {
                grille[i, j] = 0;
            }

            positionCourante = positionCourante + 1;
        }
    }
    return grille;
}
```

- **void AfficherGrille(int[,] grille) :**

Cette fonction, qui prend en paramètre la grille, permet d'afficher celle-ci avec des caractères et non des nombres entiers. Cette fonction est utilisée dans l'étape 1 et 2.

Partie utilisée dans l'étape 1 :

-une cellule morte (de la population 1) est affichée par (’.’)

-une cellule vivante (de la population 1) est affichée par (’#’)

-une cellule à naître (de la population 1) est affichée par (’-’)

-une cellule à mourir (de la population 1) est affichée par (’*’)

Utilisée dans l'étape 2 :

-une cellule vivante (de la population 2) est affichée par (’@’)

-une cellule à naître (de la population 2) est affichée par (’_’)

-une cellule à mourir (de la population 2) est affichée par (’°’)

```
static void AfficherGrille(int[,] grille)
{
    if (grille == null)
    {
        Console.WriteLine("grille null");
    }
    else
    {
        if ((grille.GetLength(0) == 0) || (grille.GetLength(1) == 0))
        {
            Console.WriteLine("grille vide");
        }
        else
        {
            for (int i = 0; i < grille.GetLength(0); i++)
            {
                for (int j = 0; j < grille.GetLength(1); j++)
                {
                    if (grille[i, j] == 0)
                    {
                        Console.Write(" . ");
                    }
                    if (grille[i, j] == 1)
                    {
                        Console.Write(" # ");
                    }
                    if (grille[i, j] == 2)
                    {
                        Console.Write(" - ");
                    }
                    if (grille[i, j] == 3)
                    {
                        Console.Write(" * ");
                    }
                    if (grille[i, j] == 4)
                    {
                        Console.Write(" @ ");
                    }
                    if (grille[i, j] == 5)
                    {
                        Console.Write("_");
                    }
                    if (grille[i, j] == 6)
                    {
                        Console.Write(" ° ");
                    }
                }
                Console.WriteLine();
            }
        }
    }
}
```

- **int CelluleVivanteInt(int[,] grille, int i, int j) :**

Cette fonction, qui prend en paramètres la grille, un numéro de ligne et un numéro de colonne, retourne 1 si la cellule regardée est vivante à la génération actuelle et 0 si la cellule est morte. Cette fonction va être utilisée pour compter le nombre de cellules vivantes autour d'une cellule (grille[i,j]).

-Tout d'abord, on rend la grille circulaire car c'est dans cette fonction que pourront être donnés en paramètres pour i et j des valeurs telles que –1 ou grille.GetLength(0).

```
static int CelluleVivanteInt(int[,] grille, int i, int j)
{
    int cellulevivanteInt = 0;
    int nbLignes = grille.GetLength(0);
    int nbColonnes = grille.GetLength(1);

    // Tester si on doit prendre la case circulaire
    if (i < 0)
    {
        i = nbLignes - 1;
    }
    if (j < 0)
    {
        j = nbColonnes - 1;
    }
    if (i >= nbLignes)
    {
        i = 0;
    }
    if (j >= nbColonnes)
    {
        j = 0;
    }
```

-Si la cellule est vivante (1) ou va mourir à la génération suivante (3), c'est qu'elle est vivante à la génération actuelle, donc on lui attribue la valeur 1. Si elle est morte (0) ou qu'elle va naître à la génération suivante (2), c'est qu'elle est morte à la génération actuelle, donc on lui attribue la valeur 0.

```
    // cellule vivante ou à mourir
    if ((grille[i, j] == 1) || (grille[i, j] == 3))
    {
        cellulevivanteInt = 1;
    }
    return cellulevivanteInt;
}
```

- **bool CelluleDevientVivante(int[,] grille, int i, int j) :**

Cette fonction, qui prend en paramètres la grille, un numéro de ligne et un numéro de colonne, retourne true si la cellule regardée est amenée à devenir ou rester vivante à la génération suivante, et retourne false sinon (cellule va rester ou devenir morte).

-On compte le nombre de cellules vivantes autour de la cellule regardée. Pour cela, on fait la somme des valeurs retournées par la fonction CelluleVivanteInt() dont les valeurs entrées en paramètres correspondent aux 8 cases autour de la cellule regardée. On obtiendra un nombre appartenant à [0,8].

```
static bool CelluleDevientVivante(int[,] grille, int i, int j) //Fonction qui déci
{
    int nombrecellulesvivantesautour = 0;
    bool celluledevientvivante = false;
    nombrecellulesvivantesautour = CelluleVivanteInt(grille, i - 1, j - 1) +
                                   CelluleVivanteInt(grille, i - 1, j) +
                                   CelluleVivanteInt(grille, i - 1, j + 1) +
                                   CelluleVivanteInt(grille, i, j - 1) +
                                   CelluleVivanteInt(grille, i, j + 1) +
                                   CelluleVivanteInt(grille, i + 1, j - 1) +
                                   CelluleVivanteInt(grille, i + 1, j) +
                                   CelluleVivanteInt(grille, i + 1, j + 1);
```

-On s'intéresse au cas où la cellule regardée est vivante. Si le nombre de cellules vivantes autour est 2 ou 3, alors elle reste vivante, la fonction retournera true. Sinon, elle retournera false (nombre de cellules vivantes autour<2 ou >3).

```
if (CelluleVivanteInt(grille, i, j) == 1) //cellule [i,j] vivante
{
    if ((nombrecellulesvivantesautour == 2) || (nombrecellulesvivantesautour == 3))
    {
        celluledevientvivante = true; //la cellule reste vivante
    }
}
```

-On s'intéresse au cas où la cellule regardée est morte. Si le nombre de cellules vivantes autour est exactement 3, alors elle va devenir vivante à la génération suivante, la fonction retournera true. Sinon, elle retournera false (nombre de cellules vivantes autour !=3).

```
    else //cellule [i,j] morte
    {
        if (nombrecellulesvivantesautour == 3)
        {
            celluledevientvivante = true; //la cellule devient vivante
        }
    }
    return celluledevientvivante;

}
```

- **int TaillePopulation(int[,] grille) :**

Cette fonction qui prend en paramètre la grille et qui retourne un entier, permet de retourner la taille de la population (population 1 pour l'étape 2).

Toute la grille est parcourue, le compteur int taillepopulation sera incrémenté dès que la valeur rencontrée sera "1".

```
static int TaillePopulation(int[,] grille) //Fonction qui compte le nombre de cellules vivantes
{
    int taillepopulation = 0;
    for (int i = 0; i < grille.GetLength(0); i++)
    {
        for (int j = 0; j < grille.GetLength(1); j++)
        {
            if (grille[i, j] == 1)
            {
                taillepopulation = taillepopulation + 1;
            }
        }
    }
    return taillepopulation;
}
```

ETAPE 2 : DEUX POPULATIONS (FONCTIONS JEUDELAVIE3 ET JEUDELAVIE4)

Description des 2 fonctions principales :

- **void JeuDeLaVie3(int nombreLignes, int nombreColonnes, int nombreCellulesVivantesAuDemarrage, int taillecellule) :**

Cette fonction est utilisée dans le cas où l'utilisateur souhaite un jeu DLV variante (2 populations) sans visualisation des états futurs (3). Elle fait appel à plusieurs sous-fonctions.

-La grille est créée et elle est affichée avec le nom de la génération (à savoir 0) et le nombre de cellules vivantes.

-Une boucle for permet de passer aux générations suivantes. Pour chaque génération sont affichés la grille, le numéro de la génération ainsi que la taille de la population (nombre de cellules vivantes).

Une grille "intermédiaire" qui ne sera pas affichée permet de savoir si une cellule de la population 1 va mourir (3) ou devenir vivante (2), ou si une cellule de la population 2 va mourir (6) ou devenir vivante (5) sans interférer dans le comptage des cellules vivantes de la population 1 ou 2 à la génération actuelle. Cette grille sera par ailleurs affichée si l'utilisateur demande la visualisation intermédiaire.

Ensuite, cette grille intermédiaire composée de 0,1,2,3,4,5 et 6 est transformée en grille de 0,1 et 4 correspondant à la génération suivante. Le 2 (cellule de la population 1 à naître) devient un 1, le 5 (cellule de la population 2 à naître) devient un 4, et le 3 (cellule de la population 1 à mourir) et 6 (cellule de la population 2 à mourir) deviennent un 0. La nouvelle génération est alors atteinte.

```
static void JeuDeLaVie3(int nombreLignes, int nombreColonnes, int nombreCellulesVivantesAuDemarrage, int taillecellule)
{
    int[,] grille = CreationGrille2(nombreLignes, nombreColonnes, nombreCellulesVivantesAuDemarrage);
    Fenetre gui = new Fenetre(grille, taillecellule, 0, 0, "Génération : 0");
    gui.RafraichirTout();
    gui.changerMessage("Génération : 0 " + "  Population 1 : " + TaillePopulation(grille) + "  Population 2 : "+ TaillePopulation2(grille));
    Console.WriteLine("Génération : 0");
    AfficherGrille(grille);
    Console.WriteLine("Population 1 : " + TaillePopulation(grille));
    Console.WriteLine("Population 2 : " + TaillePopulation2(grille));
    Console.ReadLine();
    Console.WriteLine();
    for (int k = 1; k < 20; k++)
    {
        Console.WriteLine("Génération : " + k);
        for (int i = 0; i < nombreLignes; i++)
        {
            for (int j = 0; j < nombreColonnes; j++)
            {
                if (CelluleDevientVivantePop1(grille,i,j)&&(grille[i,j]==0)) //cellule morte devient vivante de population 1
                {
                    grille[i, j] = 2;
                }
                if (CelluleDevientVivantePop2(grille,i,j)&&(grille[i,j]==0)) //cellule morte devient vivante de population 2
                {
                    grille[i, j] = 5;
                }
                if (!CelluleDevientVivantePop1(grille,i,j)&&(grille[i,j]==1)) //cellule vivante de la population 1 devient morte
                {
                    grille[i, j] = 3;
                }
                if (!CelluleDevientVivantePop2(grille, i, j) && (grille[i, j] == 4))  //cellule vivante de la population 2 devient morte
                {
                    grille[i, j] = 6;
                }
            }
        }
        for (int i = 0; i < nombreLignes; i++)
        {
            for (int j = 0; j < nombreColonnes; j++)
            {
                if (grille[i, j] == 2)
                {
                    // une cellule à naître => cellule vivante de population 1
                    grille[i, j] = 1;
                }
                if (grille[i, j] == 5)
                {
                    //une cellule à naître => cellule vivante de population 2
                    grille[i, j] = 4;
                }
                if ((grille[i, j] == 3)||(grille[i,j]==6))
                {
                    // une cellule à mourir => cellule morte
                    grille[i, j] = 0;
                }
            }
        }
        AfficherGrille(grille);
        gui.RafraichirTout();
        gui.changerMessage("Génération : " + k + "  Population 1 : " + TaillePopulation(grille) + "  Population 2 : " + TaillePopulation2(grille));
        Console.WriteLine("Population 1 : " + TaillePopulation(grille));
        Console.WriteLine("Population 2 : " + TaillePopulation2(grille));
        Console.ReadLine();
        Console.WriteLine();
    }
}
```

- **void JeuDeLaVie4(int nombreLignes, int nombreColonnes, int nombreCellulesVivantesAuDemarrage, int taillecellule) :**

Cette fonction est utilisée dans le cas où l'utilisateur souhaite un jeu DLV variante (2 populations) avec visualisation des états futurs (4). Elle a un comportement quasiment similaire à la fonction void JeuDeLaVie3.

La boucle commence dès la génération 0. A chaque tour, la grille à la génération k est affichée ainsi que le numéro de cette génération et la taille de la population. Ensuite, la grille intermédiaire dont le rôle est expliqué ci-dessus est affiché avec l'inscription "génération k+". Enfin la grille de la génération suivante est affichée et ainsi de suite...

```
static void JeuDeLaVie4(int nombreLignes, int nombreColonnes, int nombreCellulesVivantesAuDemarrage, int taillecellule)
{
    int[,] grille = CreationGrille2(nombreLignes, nombreColonnes, nombreCellulesVivantesAuDemarrage);
    Fenetre gui = new Fenetre(grille, taillecellule, 0, 0, "Génération : 0");

    for (int k = 0; k < 20; k++)
    {
        gui.RafraichirTout();
        gui.changerMessage("Génération : " + k + "  Population 1 : " + TaillePopulation(grille) +"  Population 2 : "+TaillePopulation2(grille));
        Console.WriteLine("Génération : " + k);
        AfficherGrille(grille);
        Console.WriteLine("Population 1 : " + TaillePopulation(grille));
        Console.WriteLine("Population 2 : " + TaillePopulation2(grille));
        Console.ReadLine();
        for (int i = 0; i < nombreLignes; i++)
        {
            for (int j = 0; j < nombreColonnes; j++)
            {
                if (CelluleDevientVivantePop1(grille, i, j) && (grille[i, j] == 0)) //cellule morte devient vivante de population 1
                {
                    grille[i, j] = 2;
                }
                if (CelluleDevientVivantePop2(grille, i, j) && (grille[i, j] == 0)) //cellule morte devient vivante de population 2
                {
                    grille[i, j] = 5;
                }
                if (!CelluleDevientVivantePop1(grille, i, j) && (grille[i, j] == 1)) //cellule vivante de la population 1 devient morte
                {
                    grille[i, j] = 3;
                }
                if (!CelluleDevientVivantePop2(grille,i,j)&&(grille[i,j]==4))  //cellule vivante de la population 2 devient morte
                {
                    grille[i, j] = 6;
                }
            }
        }
        Console.WriteLine("Génération : " + k + "+");
        AfficherGrille(grille);
        gui.RafraichirTout();
        gui.changerMessage("Génération : " + k + " + ");

        for (int i = 0; i < nombreLignes; i++)
        {
            for (int j = 0; j < nombreColonnes; j++)
            {
                if (grille[i, j] == 2)
                {
                    // une cellule à naître => cellule vivante de population 1
                    grille[i, j] = 1;
                }
                if (grille[i, j] == 5)
                {
                    //une cellule à naître => cellule vivante de population 2
                    grille[i, j] = 4;
                }
                if ((grille[i, j] == 3) || (grille[i, j] == 6))
                {
                    // une cellule à mourir => cellule morte
                    grille[i, j] = 0;
                }
            }
        }
        Console.ReadLine();
        Console.WriteLine();
    }
}
```

Description des différentes sous-fonctions utilisées :

- **int[,] CreationGrille2 (int nombreLignes, int nombreColonnes, int nombreCellulesVivantesAuDemarrage) :**

Cette fonction, qui prend en paramètres le nombre de lignes, le nombre de colonnes et le nombre de cellules vivantes au démarrage, retourne la grille correspondant à la génération 0. Elle permet de générer une grille d'entiers (contenant des 0, 1 et 4), où "0" représente une cellule morte, "1" représente une cellule vivante de la population 1 et "2" une cellule vivante de la population 2. Les cellules vivantes, dont le nombre est déterminé grâce au taux de remplissage de cellules vivantes au démarrage demandé à l'utilisateur, qui sont moitié population 1/ moitié population 2, sont placées aléatoirement dans la grille.

-On crée un tableau (int [] positionCellulesVivantesAuDemarrage) qui contiendra les positions des cellules vivantes à la génération 0. La première moitié du tableau correspond aux positions des cellules vivantes de la population 1 et la deuxième moitié les positions des cellules vivantes de la population 2. Le tableau est initialisé avec des –1.

```
static int[,] CreationGrille2(int nombreLignes, int nombreColonnes, int nombreCellulesVivantesAuDemarrage)
{
    int[,] grille = new int[nombreLignes, nombreColonnes];
    int nombredecellules1 = Convert.ToInt32(nombreCellulesVivantesAuDemarrage / 2); //nombre de cellules de la population 1
    int nombredecellules2 = (nombreLignes*nombreColonnes) - nombredecellules1;  //nombre de cellules de la population 2

    Random generateur = new Random(); //générateur aléatoire
    int[] positionCellulesVivantes = new int[nombreCellulesVivantesAuDemarrage]; //tableau qui va contenir les positions des cellules vivantes de la population 1 et 2
    int position;
    int positionCourante = 0;
    bool celluleVivante1;
    bool celluleVivante2;
    bool celluleOccupee;

    // Initialisation du tableau avec des -1
    for (int i = 0; i < nombreCellulesVivantesAuDemarrage; i++)
    {
        positionCellulesVivantes[i] = -1;
    }
```

-On génère aléatoirement les positions des cellules vivantes. A chaque génération aléatoire d'une position, on regarde si cette position n'a pas déjà été générée grâce à la variable bool celluleOccupee. Si la cellule va déjà être occupée par une cellule vivante de la population 1 ou 2 car la position a déjà été générée, alors on fait i-1 pour refaire un tour de boucle pour avoir une autre position. (Même codage que pour l'étape 1 →on aurait pu faire une sous-fonction)

```
// Génération aléatoire des positions des cellules vivantes de la population 1 et 2
for (int i = 0; i < nombreCellulesVivantesAuDemarrage; i++)
{
    position = generateur.Next(1, (nombreLignes * nombreColonnes) + 1); //position appartient à [1,*]
    position--; //on aura donc une position qui appartient à l'intervalle [0,*-1]
    // Verifier que cette position n'est pas déjà occupée par une cellule vivante
    celluleOccupee = false;
    for (int j = 0; j < nombreCellulesVivantesAuDemarrage; j++)
    {
        if (positionCellulesVivantes[j] == position)
        {
            celluleOccupee = true;
            break;
        }
    }

    if (!celluleOccupee)
    {
        positionCellulesVivantes[i] = position;
    }
    else
    {
        // refaire un tour supplémentaire car position déjà occupée par une cellule vivante
        i--;
    }
}
```

-On fait le tour de la grille en attribuant la valeur 1 à une case dont la position correspond à une position de cellule vivante de la population 1 (bool celluleVivante1=true), la valeur 4 à une case dont la position correspond à une position de cellule vivante de la population 2 (bool celluleVivante2=true) et 0 sinon. Pour que les positions générées correspondent à des positions d'une grille, on passe par l'intermédiaire de la variable int positionCourante.

```
for (int i = 0; i < grille.GetLength(0); i++)
{
    for (int j = 0; j < grille.GetLength(1); j++)
    {
        // Est-ce que ma position courante est une cellule vivante de la population 1?
        celluleVivante1 = false;
        for (int k = 0; k < nombredecellules1; k++)
        {
            if (positionCellulesVivantes[k] == positionCourante)
            {
                celluleVivante1 = true;
                break;
            }
        }
        // Est-ce que ma position courante est une cellule vivante de la population 2?
        celluleVivante2 = false;
        for (int k = nombredecellules1; k < nombreCellulesVivantesAuDemarrage; k++)
        {
            if (positionCellulesVivantes[k] == positionCourante)
            {
                celluleVivante2 = true;
                break;
            }
        }
        if (celluleVivante1)
        {
            grille[i, j] = 1;
        }  else if (celluleVivante2)
        {
             grille[i, j] = 4;
        }
        else
        {
            grille[i, j] = 0;
        }
        positionCourante = positionCourante + 1;
    }
}
return grille;
```

- **int CelluleVivanteInt2(int [,] grille, int i, int j, int population) :**

Cette fonction, qui prend en paramètres la grille, un numéro de ligne et un numéro de colonne, ainsi que la population de la cellule regardée (1 ou 2) retourne 1 si la cellule regardée est vivante à la génération actuelle et 0 si la cellule est morte. Cette fonction va être utilisée pour compter le nombre de cellules vivantes de la même population autour d'une cellule (grille[i,j]).

-Tout d'abord, on rend la grille circulaire car c'est dans cette fonction que pourront être donnés en paramètres pour i et j des valeurs telles que –1, -2 ou grille.GetLength(0).

```
static int CelluleVivanteInt2(int [,] grille, int i, int j, int population)
{
    int cellulevivanteInt2 = 0;
    int nbLignes = grille.GetLength(0);
    int nbColonnes = grille.GetLength(1);

    // Tester si on doit prendre la case circulaire
    if (i == -1)
    {
        i = nbLignes - 1;
    }
    if (i==-2)
    {
        i = nbLignes - 2;
    }
    if (j == -1)
    {
        j = nbColonnes - 1;
    }
    if (j==-2)
    {
        j = nbColonnes - 2;
    }
    if (i == nbLignes)
    {
        i = 0;
    }
    if (i==nbLignes+1)
    {
        i = 1;
    }
    if (j == nbColonnes)
    {
        j = 0;
    }
    if (j==nbColonnes+1)
    {
        j = 1;
    }
```

-Si la cellule appartient à la population 1 et est vivante (1) ou va mourir à la génération suivante (3), c'est qu'elle est vivante à la génération actuelle, donc on lui attribue la valeur 1. Si elle est morte (0) ou qu'elle va naître à la génération suivante (2), c'est qu'elle est morte à la génération actuelle, donc on lui attribue la valeur 0.

Si la cellule appartient à la population 2 et est vivante (4) ou va mourir à la génération suivante (6), c'est qu'elle est vivante à la génération actuelle, donc on lui attribue la valeur 1. Si elle est morte (0) ou qu'elle va naître à la génération suivante (5), c'est qu'elle est morte à la génération actuelle, donc on lui attribue la valeur 0.

```
    // cellule vivante ou à mourir
    if (population==1)
    {

        if ((grille[i, j] == 1) || (grille[i, j] == 3))
        {
            cellulevivanteInt2 = 1;
        }
    }
    if (population ==2)
    {
        if ((grille[i, j] == 4) || (grille[i, j] == 6))
        {
            cellulevivanteInt2 = 1;
        }
    }
    return cellulevivanteInt2;
}
```

- **int NombreCellulesVivantesAutourRang1(int [,] grille, int i, int j,int population) :**

Cette fonction, qui prend en paramètres la grille, un numéro de ligne et de colonne, et la population de la cellule regardée (1 ou 2), retourne le nombre de cellules vivantes appartenant à cette même population situées autour de la cellule regardée (grille[i,j]) au rang 1. Elle fait appel à la fonction int CelluleVivanteInt2(int [,] grille, int i, int j, int population).

```
static int NombreCellulesVivantesAutourRang1(int [,] grille, int i, int j,int population)  //Fo
{
    int nombrecellulesvivantesautourrang1 = 0;
    nombrecellulesvivantesautourrang1 = CelluleVivanteInt2(grille, i - 1, j - 1,population) +
                        CelluleVivanteInt2(grille, i - 1, j,population) +
                        CelluleVivanteInt2(grille, i - 1, j + 1,population) +
                        CelluleVivanteInt2(grille, i, j - 1,population) +
                        CelluleVivanteInt2(grille, i, j + 1,population) +
                        CelluleVivanteInt2(grille, i + 1, j - 1,population) +
                        CelluleVivanteInt2(grille, i + 1, j,population) +
                        CelluleVivanteInt2(grille, i + 1, j + 1,population);
    return nombrecellulesvivantesautourrang1;
}
```

- **int NombreCellulesVivantesAutourRang2(int [,] grille, int i, int j, int population) :**

Cette fonction, qui prend en paramètres la grille, un numéro de ligne et de colonne, et la population de la cellule regardée (1 ou 2), retourne le nombre de cellules vivantes appartenant à cette même population situées autour de la cellule regardée (grille[i,j]) au rang 2. Elle fait appel aux fonctions int CelluleVivanteInt2(int [,] grille, int i, int j, int population) et int NombreCellulesVivantesAutourRang1(int [,] grille, int i, int j,int population).

```
static int NombreCellulesVivantesAutourRang2(int [,] grille, int i, int j, int population)  //Fonction qui calcule le nombre de cellules vivantes d'une c
{
    int nombrecellulesvivantesautourrang2 = 0;
    nombrecellulesvivantesautourrang2 = NombreCellulesVivantesAutourRang1(grille,i,j,population) + CelluleVivanteInt2(grille, i - 2, j - 2,population) +
                        CelluleVivanteInt2(grille, i - 2, j - 1,population) + CelluleVivanteInt2(grille, i - 2, j,population) +
                        CelluleVivanteInt2(grille, i - 2, j + 1,population) + CelluleVivanteInt2(grille, i - 2, j + 2,population) +
                        CelluleVivanteInt2(grille, i - 1, j - 2,population) + CelluleVivanteInt2(grille, i - 1, j + 2,population) +
                        CelluleVivanteInt2(grille, i, j - 2,population) + CelluleVivanteInt2(grille, i, j + 2,population) +
                        CelluleVivanteInt2(grille, i + 1, j - 2,population) + CelluleVivanteInt2(grille, i + 1, j + 2,population) +
                        CelluleVivanteInt2(grille, i + 2, j - 2,population) + CelluleVivanteInt2(grille, i + 2, j - 1,population) +
                        CelluleVivanteInt2(grille, i + 2, j,population) + CelluleVivanteInt2(grille, i + 2, j + 1,population) +
                        CelluleVivanteInt2(grille, i + 2, j + 2,population);
    return nombrecellulesvivantesautourrang2;
}
```

- **bool CelluleDevientVivantePop1(int [,] grille, int i, int j) :**

Cette fonction, qui prend en paramètres la grille, un numéro de ligne et un numéro de colonne, retourne true si la cellule regardée va rester vivante de population 1 ou alors naître et faire partie de la population 1, et retourne false sinon.

-On s'intéresse aux cellules vivantes de la population 1.

```
static bool CelluleDevientVivantePop1(int [,] grille, int i, int j)
{
    int nombrecellulesvivantesautourPop1rang1 = NombreCellulesVivantesAutourRang1(grille,i,j,1);
    int nombrecellulesvivantesautourPop1rang2 = NombreCellulesVivantesAutourRang2(grille,i,j,1);
    int nombrecellulesvivantesautourPop2rang1 = NombreCellulesVivantesAutourRang1(grille, i, j,2);
    int nombrecellulesvivantesautourPop2rang2 = NombreCellulesVivantesAutourRang2(grille, i, j,2);
    int taillepopulation1 = TaillePopulation1(grille);
    int taillepopulation2 = TaillePopulation2(grille);
    bool celluledevientvivantepop1 = false;

    if (CelluleVivanteInt2(grille, i, j,1) == 1) //cellule [i,j] vivante de population 1
    {
        if ((nombrecellulesvivantesautourPop1rang1 == 2) || (nombrecellulesvivantesautourPop1rang1 == 3))
        {
            celluledevientvivantepop1 = true; //la cellule reste vivante
        }
    }
```

-On s'intéresse aux cellules mortes.

```
    if(CelluleVivanteInt2(grille,i,j,1)==0) //cellule [i,j] morte
    {
        if ((nombrecellulesvivantesautourPop1rang1 == 3)&&(nombrecellulesvivantesautourPop2rang1!=3))
        {
            celluledevientvivantepop1 = true; //la cellule devient vivante de population 1
        }
        if ((nombrecellulesvivantesautourPop1rang1==3)&&(nombrecellulesvivantesautourPop2rang1==3))
        {
            if (nombrecellulesvivantesautourPop1rang2 > nombrecellulesvivantesautourPop2rang2)
            {
                celluledevientvivantepop1=true;
            }
            if (nombrecellulesvivantesautourPop1rang2==nombrecellulesvivantesautourPop2rang2)
            {
                if(taillepopulation1>taillepopulation2)
                {
                    celluledevientvivantepop1 = true;
                }
                if(taillepopulation1==taillepopulation2)
                {
                    celluledevientvivantepop1 = false;
                }
            }
        }
    }
    return celluledevientvivantepop1;
```

- **bool CelluleDevientVivantePop2(int [,] grille, int i, int j) :**

Le fonctionnement de cette fonction est similaire à la fonction bool CelluleDevientVivantePop1(int [,] grille, int i, int j) mais elle concerne les cellules qui vont rester ou devenir de population 2.

- **int TaillePopulation2(int [,] grille) :**

Cette fonction qui prend en paramètre la grille et qui retourne un entier, permet de retourner la taille de la population 2.

Toute la grille est parcourue, le compteur int taillepopulation2 sera incrémenté dès que la valeur rencontrée sera "4".

```
static int TaillePopulation2(int[,] grille)  //Fonction qui compte le nombre de cellules vivantes de la population 1
{
    int taillePopulation2 = 0;
    for (int i = 0; i < grille.GetLength(0); i++)
    {
        for (int j = 0; j < grille.GetLength(1); j++)
        {
            if (grille[i, j] == 4)
            {
                taillePopulation2 = taillePopulation2 + 1;
            }
        }
    }
    return taillePopulation2;
}
```