Tableaux et Algorithmes

ESILV – Année 1 – 2020

TD8 – TD9

Préambule

Vous déposerez sur DVO un fichier .zip contenant uniquement :

1. le fichier Program.cs contenant votre code
    a. Chaque fonction de votre code doit être commentée. Expliquez la signature de chacune d'entre elle (type de sortie, nom de la fonction, type et sémantique des paramètres en entrée)
2. le mini-rapport au format PDF.

La deadline de votre problème est au début de votre TD10, TD pendant lequel vous aurez une revue de code.

Attention :

Pensez à faire des sauvegardes... si pas de rendu, alors 00/20.

Un rendu sans le rapport sera noté 00/20.

Un programme ne compilant pas ou ne s'exécutant pas sera noté 00/20.

Une triche entraîne une note de 00/20 au module.

ENONCE

Ce problème, se déroulant sur deux séances de TD, est de programmer un jeu de la vie (même s'il ne s'agit pas d'un jeu à proprement parler puisqu'il n'y a pas de joueur). Il s'agit d'un automate cellulaire, comme cela est très bien expliqué sur Wikipédia : https://fr.wikipedia.org/wiki/Jeu_de_la_vie.

On se propose a minima de développer un jeu de la vie tel que imaginé par J.H. Conway (étape 1), mais l'objectif principal est de développer une variante de ce jeu où ici deux populations de cellules vont évoluer sur une même grille selon des règles spécifiques (étape 2).

Attention ne disposant pas d'une grille de dimensions infinies, on adoptera les deux conventions

suivantes d'une grille torique (en plus des règles d'évolution des cellules données ci-après) :

1. les bords gauche et droit de la grille sont connectés

    ⇒ la case à gauche de celle la plus à gauche est la case la plus à droite.

    ⇒ la case à droite de celle la plus à droite est la case la plus à gauche.

2. les bords haut et bas de la grille sont connectés

    ⇒ la case au-dessus de celle la plus en haut est la case la plus en bas.

    ⇒ la case en-dessous de celle la plus en bas est la case la plus en haut.

Mise en place

- Créer un nouveau projet "Console Application" dans Visual Studio.
- La compilation se fera en mode Debug.
- Afin que le jeu soit plus convivial, une interface graphique est fournie. (voir les sections Mise en place de la GUI (Graphical User Interface) et Utilisation de la GUI en Annexe, à n'utiliser qu'après la fin de la 1ère séance de TD)

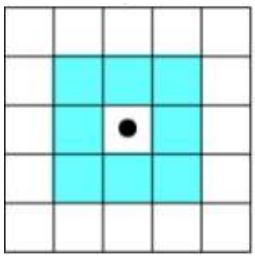
Étape 1 (permettant avec le rapport une note maximale de 10/20)

Programmer le jeu de base proposé par J.H. Conway, où les cellules naissent, meurent ou survivent à chaque génération, sur une grille en 2D, selon les trois règles simples suivantes :

1. R1 si une cellule vivante est entourée de moins de 2 cellules vivantes alors elle meurt à la génération suivante (cas de sous-population).
2. R2 si une cellule vivante est entourée de plus de 3 cellules vivantes alors elle meurt à la génération suivante (cas de sur-population).
3. R3 si une cellule morte est entourée exactement de 3 cellules vivantes alors elle naît à la génération suivante.
4. Dans tous les autres cas, les cellules restent dans le même état (mort ou vivant) à la génération suivante.

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'application des règles ci-dessus).

Le voisinage d'une cellule à prendre en compte pour les règles tient compte des 8 cellules qui l'entourent distantes de 1 case (ici en bleu clair le voisinage de la cellule marquée d'un point).


Voisinage au rang 1

Les éléments suivants sont à prendre en compte :

1. La grille manipulée est une matrice d'entiers
    a. où par exemple 0 signifie cellule morte et 1 cellule vivante (les autres valeurs restant disponibles pour les besoins du problème).
    b. D'un point de vue mémoire, on utilisera toujours la même matrice durant le programme
2. Au lancement du programme, les valeurs suivantes sont codées en dur dans le Main ou demandées à l'utilisateur :
    - nombre de lignes et nombre de colonnes
    - taux de remplissage de cellules vivantes au départ ⇔ valeur réelle ∈ [0.1, 0.9]
    - [optionnel] la taille d'une cellule (si GUI utilisée)

    Puis un menu s'affiche en proposant :
    - Jeu DLV classique sans visualisation intermédiaire des états futurs
    - Jeu DLV classique avec visualisation des états futurs (à naître et à mourir)

3. Pour l'affichage de la grille en Console,
    - une cellule morte est affichée par un point ('.')
    - une cellule vivante est affichée par un dièse ('#')
    - une cellule à naître est affichée par un tiret ('-') ← avec états futurs visibles
    - une cellule à mourir est affichée par une étoile ('*') ← avec états futurs visibles

4. À chaque génération sera affiché (en plus de la matrice) :
    - le numéro de la génération, et
    - la taille totale de la population (i.e. le nombre de cellules vivantes).

5. Le passage à la génération suivante se fera en appuyant sur une touche ou en marquant une pause de 1seconde avec l'instruction System.Threading.Thread.Sleep(1000);

Exemple

Grille de départ 5 x 5 cellules, taux de remplissage 12% :

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

Affichage sans visualisation des états futurs :

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

 t0         t1         t2
```

Affichage avec visualisation des états futurs :

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

 t0         t0+        t1         t1+        t2
```

Étape 2 (permettant avec le rapport une note maximale de 16/20)

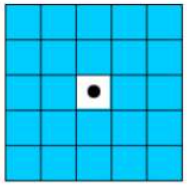
Programmer la variante décrite ci-dessous du jeu de la vie, où deux populations distinctes de cellules cohabitent sur la grille.

Dans cette variante, le principe général est la même : des cellules qui évoluent en même temps de génération en génération sur une même grille en 2D selon certaines règles d'évolution.

Les règles d'évolution sont maintenant les suivantes :

1. 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).

2. 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).
3. 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.
4. 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 de nouveau 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 toujours en cas d'égalité des tailles totales des populations la cellule reste morte.


voisinage rang 2.

Le voisinage au rang 2 d'une cellule tient compte des 24 cellules qui l'entourent, distantes de 1 et 2 cases.

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

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'application des règles ci-dessus).

Vous ferez en sorte de détecter d'une (pseudo-)stabilisation de la grille, et de mettre fin au jeu

AU LANCEMENT DU PROGRAMME,

- le taux de remplissage est global, et vaut pour moitié pour les cellules de la première population et pour moitié les cellules de la seconde population.
- le menu affiché doit maintenant proposer les éléments suivants :

  (a) Jeu DLV classique sans visualisation intermédiaire des états futurs

  (b) Jeu DLV classique avec visualisation intermédiaire des états futurs

  (c) Jeu DLV variante (2 populations) sans visualisation des états futurs ← new !

(d) Jeu DLV variante (2 populations) avec visualisation des états futurs ← new !

- À chaque génération sera affiché (en plus de la matrice) :
  - le numéro de la génération, et
  - la taille totale de chaque population.

Etape 3 : Modélisation de la propagation d'un virus : le cas du Covid (sur 4 points)

Cette étude est un prétexte et aucune conclusion ne pourra évidemment être tirée de vos résultats par rapport au Covid.

Nous laissons libre cours à votre créativité sur cette partie du projet. Vous pouvez ajouter d'autres critères si nécessaire.

Etape 3-1

Dans le cadre d'une épidémie, les personnes peuvent être saines et non immunisées, saines et immunisées (dans le cas du Covid, seulement si la personne a déjà eu au cours d'une génération antérieure le virus) ou malades.

Une personne malade du Covid par exemple peut contaminer en moyenne 2,5 personnes saines et non-immunisées. (ceci peut être évidemment un paramètre). Nous supposons que la contamination ne se fait qu'au rang 1 et de manière individuelle.

Au vu des données actuelles (et peu fiables du moment), le malade peut être atteint sur une échelle de 4

- de façon légère (0/4) sans symptômes,
- avec de légers symptômes (1/4 toux et fièvre),
- avec en plus des précédents symptômes, des courbatures et de grandes fatigues (2/4)
- avec des problèmes respiratoires (3/4) graves
- Et enfin au stade 4/4, la personne meurt.

L'évolution de la maladie n'est pas encore connue mais vous pouvez faire varier l'évolution des stades avec des poids statistiques paramétrables

- Passer du stade 0-1 : 60% à la génération suivante
- Passer du stade 1-2 : 20% à la génération suivante
- Passer du stade 2-3 : 10% à la génération suivante
- Passer du stade 3-4 : 2% à la génération suivante

Il faudra attendre 5 générations pour guérir au stade 0. Plus 5 générations seront nécessaires à chaque stade de la maladie. Exemple : Le stade 2 nécessitera donc 15 générations pour revenir au stade sain.

Pour faire la simulation, vous pouvez partir d'un ou plusieurs cas aléatoires de personnes infectées au stade que vous paramétrez.

Etape 3-2

Introduisez la notion de confinement,

- Une personne saine peut être intouchable car confinée.
- Une personne infectée peut ne plus contaminer car confinée

Dans un premier temps, les personnes confinées peuvent l'être sans limite de générations.

Ensuite, connaissant le nombre de générations utiles pour vaincre la maladie aux différents stade, les personnes peuvent être libérées de leurs contraintes.

Vous pouvez jouer sur la proportion de personnes confinées.

Faîtes une étude de la propagation dans les 2 cas (Etape 3-1 et 3-2)

Bibliographie :

http://www.tangentex.com/ACEpidemie.htm

Mini-rapport (format PDF)

Un mini-rapport de 5 pages maximum au format PDF est à rendre obligatoirement avec le code.

Doivent y figurer :

- vos nom et prénom ;
- une explication en français de comment est gérée globalement chaque fonctionnalité proposée dans le menu.
    - Notamment, comment est calculé l'état futur de la grille et comment la génération suivante est appliquée sur la grille (dans chaque cas de 1 ou 2 populations).
    - Un petit schéma peut éventuellement venir compléter l'explication
- Une explication en français de l'utilité et du fonctionnement des fonctions principales dans le code
- Explication de vos travaux liées aux « épidémies » (Etape 3)

(e.g. captures d'écran, ...), en respectant le nombre max de pages.

Avertissement !

Une attention particulière devra être portée sur la qualité du code (en plus des informations fournies dans le rapport) :

- noms explicites des variables et des fonctions
- commentaires à bon escient
- découpage pertinent en (sous-)fonctions
- chaque fonction s'occupe d'une action (simple) et le fait bien (toutes les valeurs possibles en paramètre sont traitées)

Annexe 1 – Générer des valeurs aléatoires

Pour générer des valeurs entières aléatoires, voici comment procéder :

Commencer par la création du générateur de nombres aléatoires,

```
Random generateur = new Random(); // utiliser un seul et unique générateur !!!
```

puis créer autant de valeurs aléatoires que nécessaire avec la fonction `Next(min,maxExclu)`

```
int valeur1 = generateur.Next(1,10); // valeur1 dans l'intervalle [1,9]
int valeur2 = generateur.Next(1,10); // valeur2 dans l'intervalle dans [1,9]
//...
int valeurN = generateur.Next(1,10); // valeurN dans l'intervalle dans [1,9]
```

Autre exemple possible (avec la même variable `generateur` créé précédemment) :

```
for(int nb = 0 ; nb < N ; nb++)
{
  int valeur = generateur.Next(1,10);
  //...
}
```

Ne surtout pas utiliser plusieurs générateurs

```
Random generateur1 = new Random();
int valeur1 = generateur1.Next(1,10);
Random generateur2 = new Random();     // NON ! (il existe déjà un générateur)
int valeur2 = generateur2.Next(1,10); // ne pas utiliser ce second générateur
//...
Random generateurN = new Random();     // NON !
int valeurN = generateurN.Next(1,10); // ne pas utiliser ce N-ième générateur
```

```
for(int nb = 0 ; nb < N ; nb++)
{
  Random aleatoire = new Random(); // NON ! nouveau générateur à chaque tour
  int valeur = aleatoire.Next(1,10);
  //...
}
```

Annexe 2 Mise en place de la GUI sous Windows

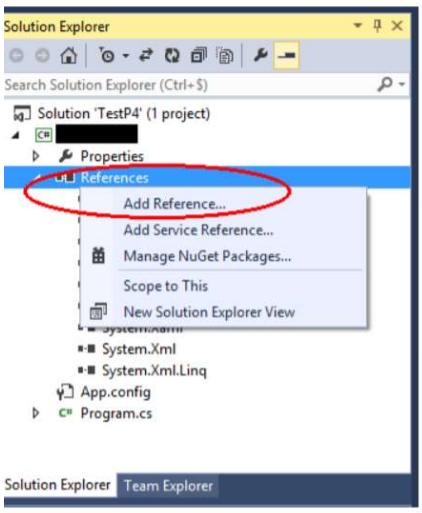
1. Une fois le projet Visual Studio de type "Console Application" de type Framework créé, compiler au moins une fois.

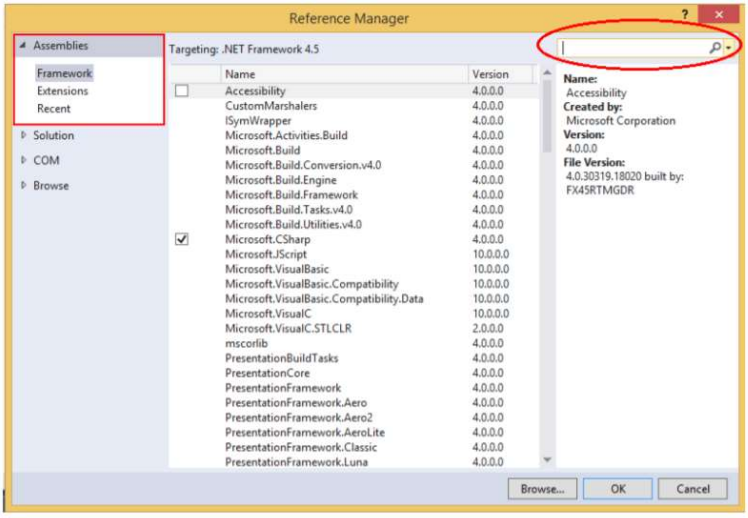
2. Récupérer sur DVO le fichier EsilvGui.dll

Copier ce fichier à la racine du projet Visual Studio (i.e. au même emplacement que le fichier .sln du projet).

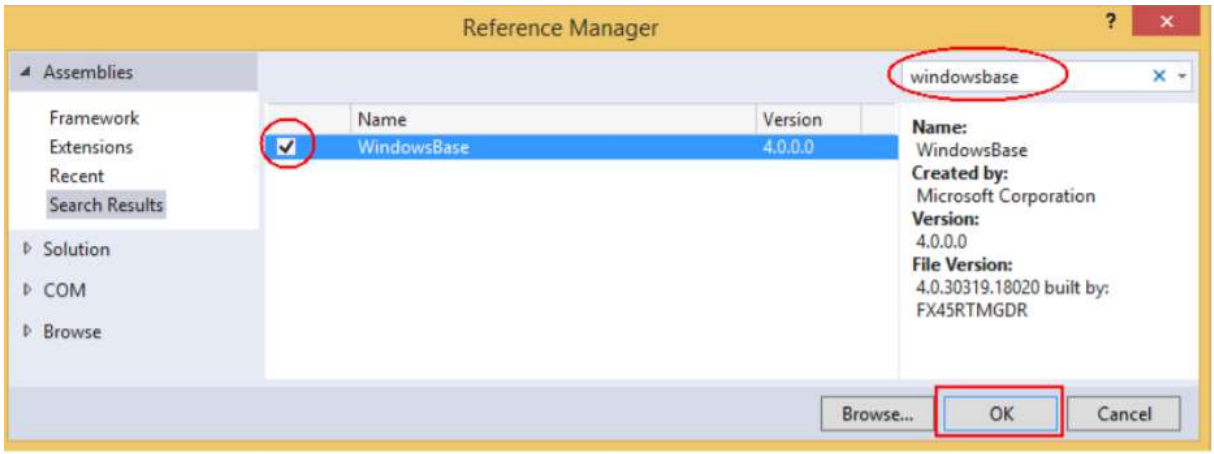
3. Dans Visual Studio, en colonne de droite "Explorateur de solution" ("Solution Explorer", en anglais). Si la colonne est non visible, aller dans le menu "Affichage" ("View", en anglais).

- Clic droit sur "Référence", puis "Ajouter une référence"
- Dans la zone de recherche (à droite de la nouvelle fenêtre), saisir "WindowsBase"





Sélectionner l'élément "WindowsBase"



Cliquer sur OK en bas à droite de cette nouvelle fenêtre.

4. Recommencer l'étape précédente de façon à ajouter les références :

- "PresentationCore"
- "PresentationFramework"
- "System.Xaml"

5. Il faut maintenant ajouter au projet la DLL précédemment téléchargée, pour cela :

- Clic droit sur "Référence", puis "Ajouter une référence"
- Cliquer sur "Parcourir" ("Browse", en anglais) en bas à droite de la nouvelle fenêtre
- Sélectionner dans votre disque dur, le fichier EsilvGui.dll qui a été placé à la racine du dossier de votre projet Visual Studio.
  Attention, le nom du fichier doit être exactement EsilvGui.dll (et pas du genre EsilvGui (1).dll).

6. Enfinn, aller dans le menu Outils > Options > Débogage > Général et décocher

"Activer les outils de débogage d'interface utilisateur pour XAML"

Annexe 3 Utilisation de la GUI

Pour utiliser l'interface graphique, voici un exemple ci-dessous :

- Rappel, c'est une matrice d'entiers qui est manipulée. → Ici, par exemple, la matrice grille définie en ligne 16. **La GUI est juste là pour faire... moderne.**
- Concernant l'interface graphique (GUI)
  - Pour la créer, merci de faire comme cela est codé en ligne 20.
  - Dès qu'un ensemble de cases de la matrice a été modifié (génération suivante), il faut en informer l'interface graphique tel que cela est fait en ligne 29.
  - Changement possible du message en bas de la GUI comme présenté en ligne 32.
- Les couleurs dans la GUI associées aux entiers contenus dans la matrice sont :
  - valeur 0 → couleur par défaut
  - 1 → noir, 2 → vert, 3 → rouge, 4 → bleu, 5 → jaune, 6 → orange.
- Ajouter :
  - Using EsilvGui; (ici en ligne 6) en début de fichier, ainsi que
  - Using System.Threading.Tasks; (si cela n'est déjà fait comme ici en ligne 5).
  - Au-dessus du Main, bien ajouter le texte [System.STAThreadAttribute()] (ici, ligne 12)

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6  using EsilvGui;
7
8  namespace NOM_DE_VOTRE_PROJET
9  {
10   class Program
11   {
12     [System.STAThreadAttribute()]
13     static void Main(string[] args)
14     {
15       // *** Mise en place du jeu ***
16       int[,] grille = new int[20, 25];   // 20 lignes, 25 colonnes
17       InitialisationGrille(grille);      // fonction à coder (ou pas !?)
18
19       // *** Mise en place de la GUI ***
20       Fenetre gui = new Fenetre(grille, 15, 0, 0, "Jeu de la vie");
21                                 // taille cellule de 15 pixels
22                                 // fenêtre en haut à gauche, décalages (0,0)
23                                 // message initial : "Jeu de la vie"
24
25       // ** Exemple totalement fictif de modification de cellules **
26       grille[4, 3] = 1;
27       grille[4, 4] = 1;
28       grille[4, 5] = 1;
29       gui.RafraichirTout(); // mise à jour graphique
30
31       // ** Exemple de modification du message en bas de la fenêtre **
32       gui.ChangerMessage("Génération 1. Actuellement 3 cellules vivantes.");
33
34       // ...
35       Console.ReadKey();
36     }
37   }
38 }
```

Exemple d'utilisation de la GUI, avec deux populations sur une matrice 50x50 :