
TP

Implémentation en C# d'un jeu de la vie

Octobre 2020

Présentation :

Le jeu de la vie est un jeu de simulation qui permet d'observer l'évolution de cellules sur une grille. A chaque pas de temps, certaines cellules apparaissent et d'autres disparaissent en fonction de règles simples. Il est alors possible de voir émerger des comportements typiques plus ou moins chaotiques ou encore des convergences vers des états stables.



FIGURE 1 – Exemple de visuel pour une simulation d'un jeu de la vie à un temps donné où les cellules sont représentées en blanc sur un fond noir.

Règles de simulation :

Cette simulation ne repose que sur deux règles :

- ★ Si au temps T , à un emplacement donné E il n'y a aucune cellule mais que sur les emplacements voisins on compte exactement 3 cellules alors au temps $T+1$ une cellule apparaît à cet emplacement E .
- ★ Si au temps T à un emplacement donné E , il y a une cellule et qu'autour d'elle il y a exactement 2 ou 3 cellules vivantes alors la cellule de l'emplacement E reste en place

au temps $T+1$, sinon elle disparaît.

Tout se joue alors sur les conditions initiales, c'est-à-dire le **nombre de cellules** et **leurs positions** au lancement de la simulation. Il est en effet possible de voir apparaître des comportements très intéressants en changeant simplement les conditions de départs. C'est un problème qui n'est pas compliqué en soit mais pour autant qui peut être qualifié de **complexe**.

Ce travail va vous permettre d'approfondir vos connaissances en programmation orientée objet ainsi que vous permettre de manipuler l'environnement graphique proposé par le langage C au travers des Windows Form.

Implémentation :

Dans une premier temps, il est important d'avoir le réflexe de développer vos applications dans des environnements textuels, en console. Une fois que votre code sera propre et fonctionnel, il sera alors possible d'ajouter la couche graphique. Les étapes de développement sont importantes pour assurer le maximum d'indépendance entre le code source qui s'occupe de la mécanique du jeu et le code permettant de gérer l'affichage graphique. Prenez l'habitude de bien séparer chaque composante de votre projet.

En savoir plus sur le modèle MVC

Prenez le réflexe de créer un nouveau dépôt git pour chaque nouveau projet. Vous pouvez travailler à plusieurs si vous le souhaitez et vous entraîner ainsi à utiliser git sur un projet de groupe.

Partie 1 - Environnement textuel

Si vous ressentez le besoin d'être guidé dans votre travail, vous trouverez dans cette partie des indications sur les classes à implémenter, les attributs et les méthodes ainsi que les interactions entre les différentes classes. Le but de cette UE est de vous rendre capable de réaliser ce type de travail sans indication. Si vous ne ressentez pas le besoin d'être quigué, vous pouvez simplement consulter le diagramme de classes et commencer votre implémentation. Dans tous les cas, prenez bien le temps d'observer et de comprendre le diagramme de classes (Fig. 5) pour avoir en tête l'organisation des classes les unes par rapport aux autres et le rôle de chacune d'elle.

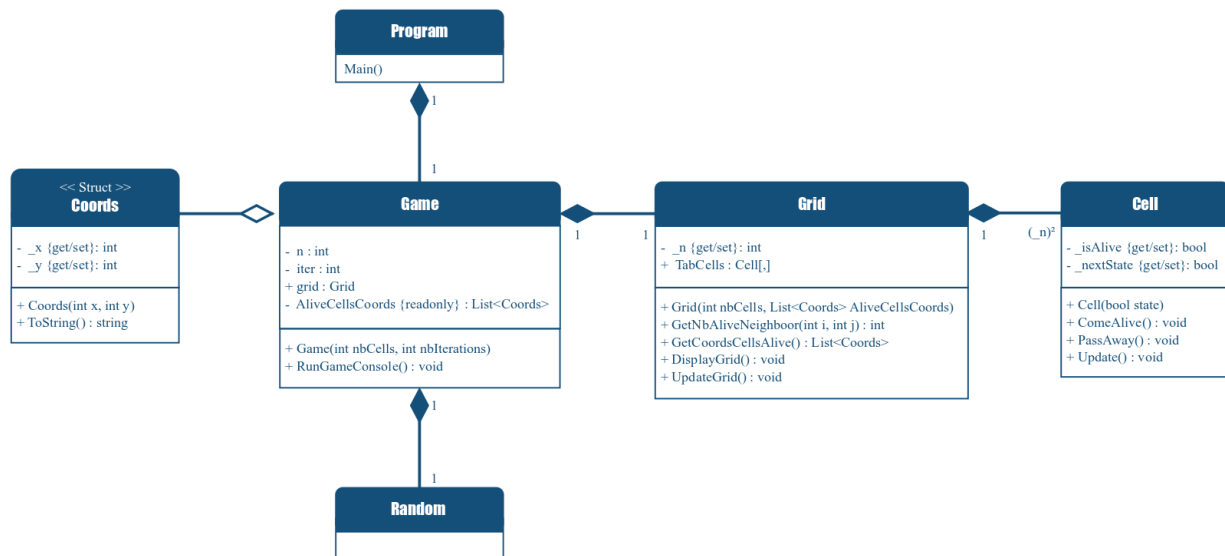


FIGURE 2 – Représentation UML du projet textuel du jeu de la vie

Un système d'étoile devant chaque méthode permet de vous informer sur la difficulté de leur implémentation. Une étoile correspond à une méthode très simple ou entièrement guidée. Trois étoiles correspondent à une méthode peu guidée ou demandant de mettre au point un algorithme et de réfléchir aux différents cas d'application (autrement dit il vous faudra plus de temps pour l'implémenter et c'est normal). Entraînez-vous à utiliser git en effectuant un commit commenté pour chaque nouvelle méthode fonctionnelle.

1. Créer un nouveau projet et vérifier qu'il est pour le moment configuré pour une sortie de type console en modifiant le fichier de configuration :
Projet > Propriétés de NomDeVotreProjet... > Application > Type de sortie > Application Console
Sauvegarder vos changements.

2. La classe Cell

Les attributs :

- `private bool _isAlive; // État de la cellule`
- `public bool isAlive { get { return _isAlive; } set { _isAlive = value; } } //`
Accesseurs en lecture et en écriture pour _isAlive
- `private bool _nextState; // Stockage temporaire de l'état de la cellule pour le pro-`
chain pas de la simulation
- `public bool nextState { get { return _nextState; } set { _nextState = value; } } //`
Accesseurs en lecture et en écriture pour _nextState

Remarque : L'attribut _isAlive est un attribut privé, pour y accéder on définit ici deux moyens, un accès en lecture grâce au mot clé get et un accès en écriture grâce au mot clé set.

Rappel sur les accesseurs

[Point de vigilance] Assurez-vous de bien comprendre la différence entre `_isAlive` (attribut de classe) et `isAlive` (accesseur qui permet de lire ou modifier l'attribut qui lui est associé)

Les méthodes à implémenter :

- ★ `public Cell(bool state) // Constructeur de la classe Cell qui modifie l'attribut`
_isAlive via son accesseur en écriture pour lui attribuer la valeur state.
- ★ `public void ComeAlive() // Méthode qui modifie à true l'attribut _nextState via`
son accesseur en écriture.
- ★ `public void PassAway() // Méthode qui modifie à false l'attribut _nextState via`
son accesseur en écriture.
- ★ `public void Update() // Méthode qui met à jour l'attribut _isAlive via son ac-`
cesseur en écriture en lui associant la valeur contenue dans la variable _nextState
via son accesseur en lecture

3. La classe **Grid**

Les attributs :

- `private int _n; // taille de la grille`
- `public int n { get { return _n; } set { _n = value; } } // accesseurs en lecture et en écriture`
- `Cell[,] TabCells; // Tableau à deux dimensions contenant des objets de type Cell`

Rappel sur les tableaux

Les méthodes :

***** public Grid(int nbCells, List<Coords> AliveCellsCoords)** - *Constructeur de la class Grid*

→ `this.n = nbCells; // Initialisation de l'attribut _n au travers de l'accesseur en écriture`

→ `TabCells = new Cell[n,n]; // Création d'un tableau à deux dimensions de taille n,n`

→ */* Remplissage du tableau avec à chaque emplacement une instance d'une cellule Cell créée vivante (true) si les coordonnées sont dans la liste AliveCellsCoords ou absente (false) sinon. */*
`for (int i = 0; i < n; i++)`
`{`
`for (int j = 0; j < n; j++)`
`{`
`*** Instructions ***`
`}`
`}`

**** public int getNbAliveNeighboor(int i, int j)** *// Méthode qui permet de déterminer le nombre de cellules vivantes autour d'un emplacement de coordonnées (i,j)*

**** public List<Coords> getCoordsNeighbors(int i, int j)** *// Méthode qui permet de déterminer toutes les coordonnées valides autour d'un emplacement de coordonnées (i,j) (attention à la gestion des cas particulier en bordure de grille)*

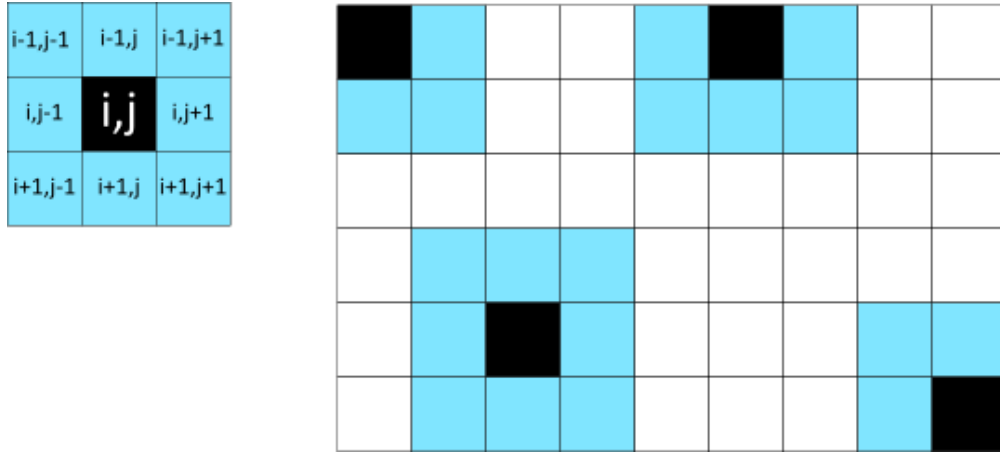


FIGURE 3 – Rappel des coordonnées des cellules voisines pour une cellule de coordonnées (i,j) et représentation des cellules voisines (bleues) pour les cellules identifiées en noir.

- ★★ **public List<Coords> getCoordsCellsAlive()** // Méthode qui permet de déterminer la liste des coordonnées de toutes les cellules vivantes de la grille.
- ★★ **public void DisplayGrid()** // Méthode qui permet d'afficher une représentation de grille en console avec un X à chaque emplacement où une cellule est vivante

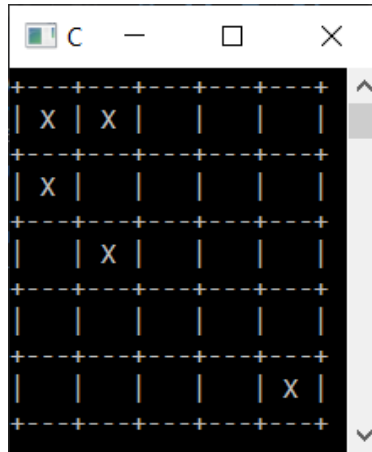


FIGURE 4 – Exemple de visuel attendu

- ★★★ **public void UpdateGrid()** // Méthode qui parcourt chaque cellule et qui met à jour leur attribut `_nextStep`, via son accesseur en écriture, en fonction des règles de la simulation. L'attribut est mis à `true` si la cellule reste en vie ou apparaît et à `false` si la cellule à cet emplacement disparaît ou reste absente. Une fois toute la grille parcourue, une deuxième passe est effectué pour associer la valeur de `nextStep` à l'attribut `isAlive` de chaque cellule.

4. La classe **Game**

Les attributs :

- private int n // *taille de la grille*
- private int iter // *nombre d'itérations de la simulation*
- public Grid grid // *grille des emplacements possibles*
- List<Coords> AliveCellsCoords // *liste des coordonnées des cellules vivantes en début de simulation.*

Pensez à ajouter l'espace de nom nécessaire à l'utilisation des structures de données Liste en début de programme :

```
using System.Collections.Generic;
```

Rappel sur les espaces de noms

Les méthodes :

- ★ **public Game(int nbCells, int nbIterations)** // *Constructeur de la class Game*
→ Initialise les attributs n, iter, AliveCellsCoords (quelques exemples fournis) et une nouvelle grille de taille n x n à partir de la liste AliveCellsCoords construite.
- ★ **public void RunGameConsole()** // *Méthode de supervision qui implémente le mécanisme au coeur de la simulation d'après les étapes suivantes.*
→ Appel à la fonction *DisplayGrid* sur l'objet grid // *Affiche en console une représentation graphique de la grille et des cellules vivantes.*
→ Boucle sur le nombre d'itérations souhaité qui répète les étapes suivantes :
 - (a) Appel à la fonction **UpdateGrid** sur l'objet grid // *Applique les règles du jeu et détermine quelles seront les cellules vivantes au prochain pas de la simulation.*
 - (b) Appel à la fonction **DisplayGrid** sur l'objet grid.
 - (c) Thread.Sleep(1000); // *Met en pause le programme 1s avant d'entamer le prochain passage dans la boucle.*

Pensez à ajouter l'espace de nom nécessaire à l'utilisation de thread en début de programme :

```
using System.Threading;
```

5. La classe **Program**

Le code source est fourni. Il n'attend aucune modification.

Partie 2 - Environnement graphique

Il est maintenant temps de mettre en place un environnement graphique pour pouvoir observer l'évolution de votre simulation dans une fenêtre (voir Fig. 1). Les étapes suivantes sont très détaillées. Vous pourrez vous y référer de nouveau pour d'autres programmes en environnement graphique. N'hésitez pas à rechercher des ressources complémentaires pour aller plus loin.

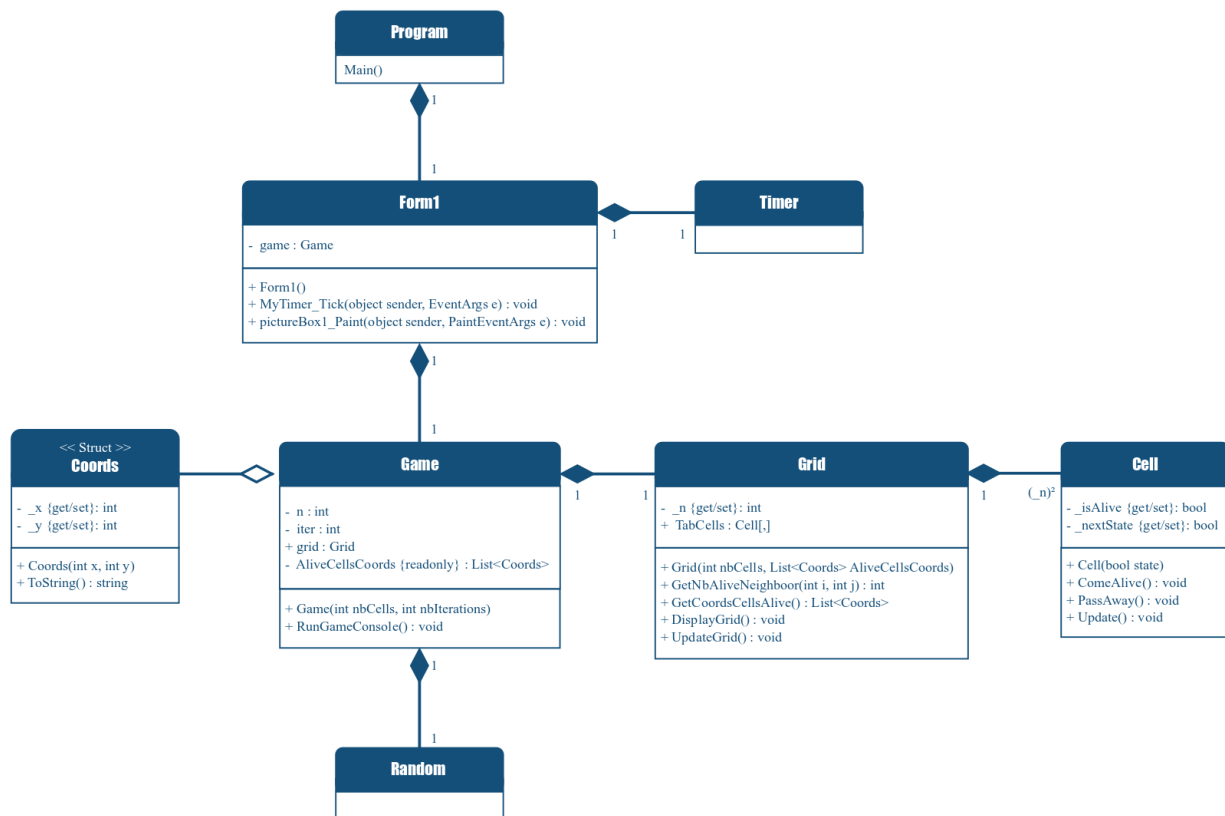


FIGURE 5 – Représentation UML du projet graphique du jeu de la vie

1. Commencer par ouvrir un nouveau projet en choisissant cette fois le modèle "Application Windows Forms" qui vous permettra de commencer à manipuler la fenêtre directement.
2. Placez-vous dans le fichier Forms1.cs. Vous devriez voir un modèle de fenêtre graphique vide.
3. Si la boîte à outils ne s'affiche pas directement sur la gauche, aller la chercher dans Affichage > Boîte à outils.
4. Pour les besoins de base de cet exercice vous allez manipuler uniquement une PictureBox que vous trouverez dans les Contrôles Communs de la boîte à outils. Cliquer-Déposer la PictureBox sur la fenêtre.

5. Dans l'onglet "Propriétés" en bas à droite, vous pouvez voir que vous avez maintenant un nouvel objet nommé pictureBox1 à l'intérieur de votre objet Form1. Vous pouvez manipuler les différentes propriétés proposées pour en comprendre le fonctionnement.
6. Pour voir le code source de la classe Form1, regarder en haut à droite dans la section "Explorateur de solutions". Vous devriez retrouver le nom de fichier Form1.cs. Faites un clic droit dessus et choisissez "Afficher le code" ou utilisez directement le raccourci clavier F7 depuis l'onglet Form1.cs [Design]

En savoir plus sur les classes partielles

7. Ouvrez également le fichier Form1.Designer.cs et parcourez le, notamment la région "Code généré par le Concepteur Windows Form" qui vous permettra de faire le lien entre les outils de manipulation graphique en cliquer-déposer et le code correspondant. Vous devriez retrouver les deux composants pictureBox1 et Form1 que l'on va configurer.
8. Modifier la signature de la méthode InitializeComponent en lui ajoutant un paramètre entier n.

```
private void InitializeComponent(int n)
{
    Instructions
}
```

9. Et pour rester cohérent, il nous faut également modifier l'appel à cette méthode dans Form1.cs

```
public Form1()
{
    int n = 5;
    InitializeComponent(n);
}
```

Maintenant que la méthode InitializeComponent connaît la valeur de n (la taille de la grille), on va pouvoir l'utiliser pour redimensionner la fenêtre ainsi que la pictureBox.

10. Modifier le code Form1.Designer.cs à partir des informations suivantes :
 - ★ Pour une grille de taille n x n en affichage console, on souhaite une grille de taille 5n x 5n en affichage graphique pour pouvoir allouer un emplacement de 5 par 5 à chaque cellule sans en modifier le nombre.
 - ★ La grille doit avoir un arrière plan foncé
 - ★★ La grille doit être centrée dans la fenêtre
 - ★ Le titre de la fenêtre doit être "Le jeu de la vie"
11. Importer vos classes Cell, Grid et Game.
12. Récupérer le code source fourni de la classe Program.

13. Ajouter la méthode de rendu suivante à votre classe Game

```
public void Paint(Graphics g)
{
    SolidBrush whiteBrush = new SolidBrush(Color.White);
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            if (grid.lCells[i,j].isAlive)
            {
                g.FillRectangle(whiteBrush, j * 5, i * 5, 5, 5);
            }
        }
    }
}
```

14. Quelques modifications du fichier Game.cs sont nécessaires.

- ★ L'attribut *iter* de la classe Game n'est plus utile en mode graphique. Vous allez donc définir un deuxième constructeur qui n'aura comme paramètre qu'un entier *n*. Reprenez les instructions de votre premier constructeur excepté l'instruction concernant l'attribut *iter*.
- ★ Modifier le constructeur initial pour qu'il hérite du nouveau constructeur. Il ne contiendra ainsi qu'une seule instruction supplémentaire permettant d'initialiser l'attribut *iter*.

Ressource sur le chaînage de constructeurs

15. Terminer avec le fichier Form1.cs

- ★ Déclarer un attribut privé *game* de type Game
- ★ Déclarer un attribut public *n* de type entier
- ★ En première instruction du constructeur de Form1, initialiser *n* à 40.
- ★ A la suite des dernières instructions du constructeur de Form1, initialiser *game* avec une nouvelle instance de Game.
- ★ A la suite des dernières instructions du constructeur de Form1, déclarer grâce aux instructions suivante un timer. Il permettra à la fenêtre de se rafraîchir toutes les 600ms en calculant à chaque fois le prochain état de la simulation :

```
Timer MyTimer = new Timer();
MyTimer.Interval = (600);
MyTimer.Tick += new EventHandler(MyTimer_Tick);
MyTimer.Start();
```
- ★★ Déclarer une nouvelle méthode de signature **private void MyTimer_Tick(object sender, EventArgs e)** qui fait appel à la méthode de mise à jour de la grille de

la classe Grille puis qui fait appel à la méthode Refresh()

Ressource Refresh

Ressource Timer

- ★ Déclarer la méthode de signature **private void pictureBox1_Paint(object sender, PaintEventArgs e)** qui fait appel à la fonction Paint de la classe Game avec comme argument e.Graphics.

Améliorations libres

Une fois que vous avez pu admirer vos simulations tourner dans une fenêtre graphique, entraînez-vous en implémentant les fonctionnalités suivantes. Aucune consigne ne vous est donnée pour vous guider, c'est un travail en toute autonomie.

1. Ajouter un affichage de la génération en cours
2. Ajouter une fin de simulation quand la grille n'évolue plus.
3. Ajouter une génération aléatoire du nombre de cellules initiales et de leurs coordonnées.
4. Améliorer l'interface en ajoutant la possibilité de faire une pause ou de recommencer une simulation au début