IFT1025 Été 2010 — Devoir 1

Miklós Csűrös

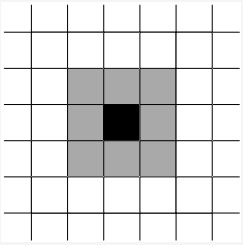
24 mai 2010

À remettre avant 10 :30 vendredi le 4 juin par courriel à csuros@iro.umontreal. . . . Ce travail est destiné à des équipes de 2 ou 3 étudiant/es.

1 Jeu de la vie

Dans ce TP, vous avez à implanter un automate cellulaire nommé «Jeu de la vie» avec une interface graphique.

Le jeu se déroule sur une grille à deux dimensions, théoriquement infinie (mais de longueur et de largeur finies et plus ou moins grandes dans la pratique), dont les cases — qu'on appelle des «cellules», par analogie avec les cellules vivantes — peuvent prendre deux états distincts : «vivantes» ou «mortes».



Le jeu simule l'évolution d'une colonie de cellules par générations synchronisées : à chaque étape, l'état de toutes les cellules change en même temps. L'évolution d'une cellule dans une étape est entièrement déterminée par l'état de ses huit voisines.

Règles pour une étape

- ⋆ Une cellule morte possédant exactement trois voisines vivantes devient vivante (elle naît).
- ⋆ Une cellule vivante possédant deux ou trois voisines vivantes le reste, sinon elle meurt. Donc une cellule vivante avec moins que deux ou plus que trois voisines vivantes meurt.

La Figure 1 montre un exemple de générations successives.

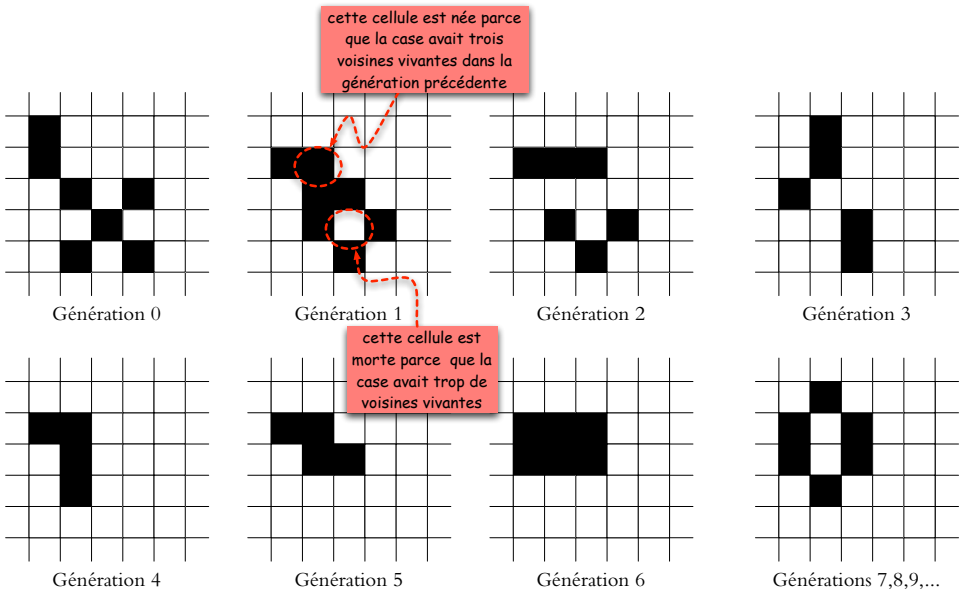
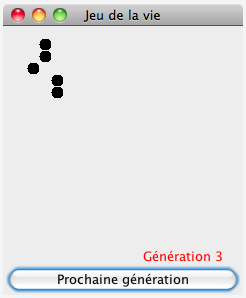


FIG. 1 – Étapes en Jeu de la vie

2 Implémentation

Vous devez implanter le Jeu de vie sur une grille finie. L'état de la grille sera montré étape par étape en utilisant une interface graphique en Java Swing. Vous devez écrire le code pour représenter une grille finie, ainsi que pour l'afficher. Je vous fournis le code pour initialiser le `JFrame` de l'application. La grille peut être initialisée par un fichier qui spécifie la taille de la grille et énumère les cellules vivantes, ou bien l'initialisation peut être au hasard.



L'application comprend un bouton «Prochaine génération» et un `JPanel` montrant les cellules vivantes, tous contenus dans un `JFrame`. Quand le bouton est pressé, l'état de la nouvelle génération est calculé et l'affichage est mise à jour.

2.1 Structure de données pour l'état des cellules

La classe `jeudelavie.Grille` utilise un tableau de taille $n \times n$ pour stocker l'état de cellules sur une grille de taille $n \times n$, et une variable `int generation`. Méthodes publiques à implémenter :

- Instanciation : `public Grille(int n)`. L'argument `n` donne la taille de la grille. Toutes les cellules sont initialisées comme «mortes». Un compteur de générations/étapes est initialisée à 0.
- Nouvelle génération : `public void nextGeneration()`. Calcule l'état de toutes les cellules dans la prochaine génération et incrémente le compteur de générations.
- `public int getGeneration()`. Retourne la valeur du compteur de générations.
- `public boolean isAlive(int row, int col)`. Retourne l'état (`true` pour vivant et `false` pour mort) de la cellule en rangée `row` et colonne `col`.
- `public void setState(int row, int col, boolean state)`. L'état de la cellule en rangée `row` et colonne `col` est affecté à `state`. Les indices des rangées et des colonnes sont de 0 à $(n-1)$.
- `public int getSize()`. Retourne la taille n de la grille.

2.2 Visualisation

La classe `jeudelavie.GUI` contient la méthode `main` pour executer l'application. Cette classe vous est fournie. Vous ne devez que changer l'initialisation du composant graphique pour l'affichage de la grille (dans le code : `JPanel grille_affichee`). Vous devez écrire la classe[1] pour ce dernier (p.e., `jeudelavie.GrilleAffichee`). Les cellules vivantes doivent être affichées par un rectangle ou un cercle (votre choix : `fillRectangle` ou `fillOval`) rempli de taille $t \times t$, où t est spécifié lors d'instanciation. Donc la cellule en rangée r et colonne c doit être affichée aux coordonnées $c \times t, r \times t$. Il faut aussi afficher le compteur de générations (`drawString`). Il est fortement récommandé d'implanter `GrilleAffichee` comme une sous-classe de `JPanel` où vous devez fournir le code pour l'instanciation (le constructeur prend une `Grille` et la taille t), et le dessin (`paintComponent`) :

[1] En principe, on pourrait même inclure le code de graphisme dans la classe `Grille`. Mais, en général, il est une bonne idée de séparer son code de structures de données et celui du graphisme.

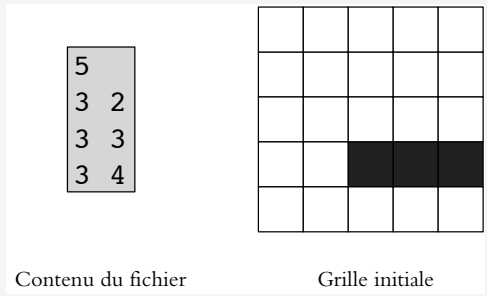
```
package jeudelavie;
// ... commentaires et imports
public class GrilleAffichee extends JPanel
{
    public GrilleAffichee(Grille G, int t)
    {
        // votre code
    }

    public paintComponent(Graphics G)
    {
        super.paintComponent(G); // code original pour JPanel
        // ... votre code
    }
}
```

2.3 Execution

Il y a deux façons de lancer l'application :

- ⋆ `jeudelavie.GUI -rnd` n p initialise une grille $n \times n$ au hasard (cellules vivantes avec probabilité p).
- ⋆ `jeudelavie.GUI -in` f initialise la grille à partir d'un fichier f.

Initialisation par un fichier. Le fichier d'entrée est un fichier texte où les champs sont séparés par des espaces blancs (`TAB` ou espace).



La première ligne spécifie la taille n Chaque ligne suivante doit contenir deux entiers, séparés par des espaces blancs, quie correspondent à la rangée et à la colonne d'une cellule vivante. La cellule à gauche en haut est en rangée 0 et colonne 0.

Les fichiers `blinker.grille`, `lwss.grille`, `gun.grille` et `etapes.grille` (c'est l'exemple de la Figure 1) vous sont fournis pour tester l'application. La lecture du fichier est implémenté dans la classe `GUI`.

Pour l'initialisation des cellules vivantes en génération 0, vous devez implanter la méthode

► `public void initCells(ArrayList<Point> cellules)`

de l'objet `Grille`. L'argument est la liste de cellules vivantes en génération 0. Toute autre cellule commence par l'état «mort». La méthode `initCells` est appellée par les méthodes statiques `random` et `readFile` de la classe `GUI`. Une cel-

lule vivante dans la liste `cellules` correspond à un objet de type `java.awt.Point` dont les variables publiques `x` et `y` spécifient la rangée et la colonne (dans ce même ordre).

3 Remise de travail

Compilez un archive jar nommé `Jeudelavie.jar` qui contient les classes `Grille`, `GUI`, et votre classe d'affichage de la grille. L'archive doit inclure les sources `.java` et les classes compilées `.class`. Envoyez le fichier comme attachement dans un courriel à `csuros@iro.umontreal...`. Votre programme doit être fonctionnel, clair, commenté et original (plagiat=0).