IFT1025 Automne 2009 — Devoir 1

Miklós Csűrös

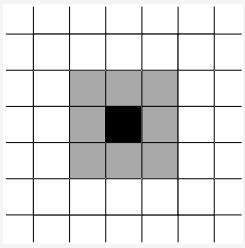
8 septembre 2009

À remettre avant 13 :30 jeudi le 17 septembre par courriel à dift1025@iro.umontreal. . . .

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.

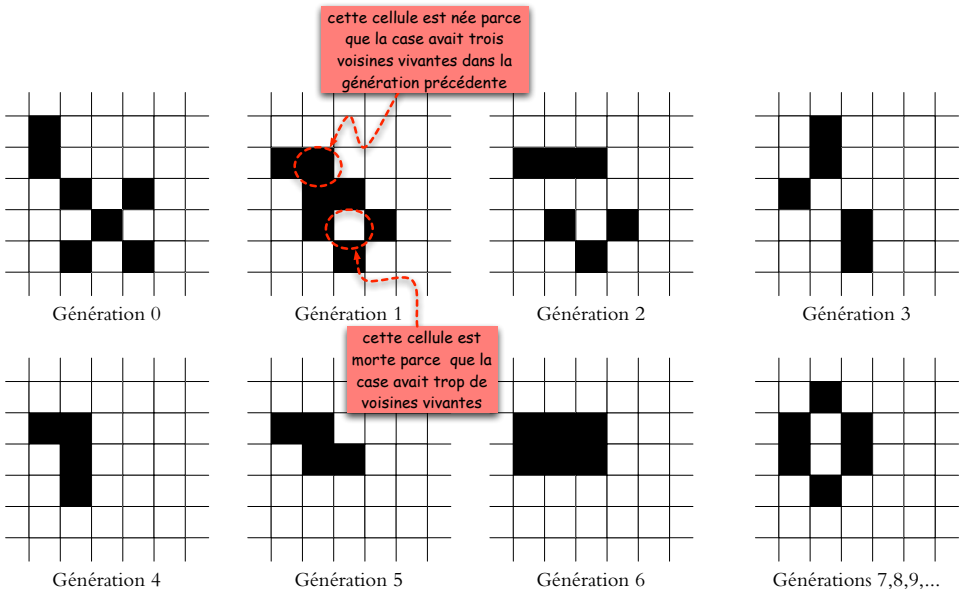
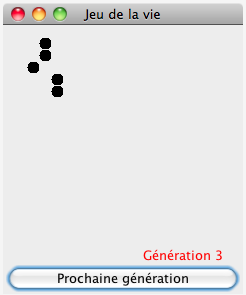


FIG. 1 – Étapes en Jeu de la vie

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. 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 de la classe à implémenter :

- ▶ Instantiation : `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.
- ▶ `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`.
- ▶ `public int getSize()`. Retourne la taille n de la grille.

2.2 Visualisation

La classe `jeudelavie.Visualisation` sert à afficher l'état des cellules d'une `Grille`. Les cellules vivantes doivent être afichées par un rectangle ou un cercle (votre choix : `fillRectangle` ou `fillOval`) rempli de taille $t \times t$, où t est spécifié lors d'instantiation. 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`). La classe `Visualisation` est une extension de la classe `javax.swing.JPanel`. Méthodes publiques de la classe à implémenter :

- ▶ Instantiation : `public Visualisation(Grille grille, int t)`.
- ▶ `public void paintComponent(Graphics g)`. Il faut fournir votre propre code pour afficher l'état de la grille.

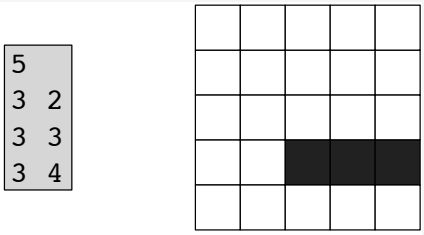
2.3 Execution

La classe `jeudelavie.Lancer` contient la méthode `main` pour executer l'application. Cette classe vous est fournie. Il y a deux façons de lancer l'application :

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

Initialisation aléatoire. La méthode `Grille.random(int n, double p)` est implémentée déjà dans le fichier `Grille.java`. La méthode retourne une grille de taille `n` × `n` où chaque cellule est vivante avec la probabilité donnée.

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



Contenu du fichier Grille initiale

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 `Lancer` par la méthode

```
private static ArrayList<ArrayList<String>> readFields(...).
```

Vous devez implémenter la méthode

► `public static Grille read(ArrayList<ArrayList<String>> L)`

dans la classe `Grille`. L'argument est une liste de listes de `String`s lus dans le fichier d'entrée. Chaque élément de la liste correspond à une ligne dans le fichier et donne les mots qui étaient séparés par des espaces blancs dans l'entrée. Donc, le premier élément devrait contenir l'encodage (en `String`) d'un nombre entier (la taille) et chaque élément après devrait contenir l'encodage d'une rangée et d'une colonne pour une cellule vivante.

3 Soumission

Compilez un archive jar nommé `Jeudelavie.jar` qui contient les classes `Grille`, `Visualisation` et `Lancer`. L'archive doit inclure les sources `.java` et les classes compilées `.class`. Envoyez le fichier comme attachement dans un courriel à `dift1025@iro.umontreal...`. Votre programme doit être fonctionnel, clair, commenté et original (plagiat=0).

Travail estimé : 120–160 lignes de code (incluant commentaires et lignes vides pour lisibilité).