TD 12 d'informatique UVSQ 2001-2002

Cours : Mathieu Le Coz – Marianne Durand
TD : Matthieu Finiasz

fevrier 2002

Un petit jeu de la Vie...

Introduction

Le principe du "jeu de la vie" est d'essayer de reproduire le comportement d'une population à travers des regles d'évolution très basiques. On considère un grand territoire rectangulaire que l'on divise en petites cellules carrées. Chaque cellule peut être pleine ou vide et on fait évoluer l'ensemble de la population tour par tour. Les règles sont simples :

- si la cellule a 0 ou 1 voisin, elle devient vide ou le reste si elle l'était déjà
- si la cellule a 2 voisins, elle reste dans le même état (vide si elle était vide, pleine si elle était pleine)
- si la cellule a 3 voisins, elle se remplit ou reste pleine si elle l'était déjà
- si la cellule a entre 4 et 8 voisins, elle se vide par surpopulation...

Le but est de laisser évoluer l'ensemble des cellules sur un grand nombre de tour et de regarder leur comportement : cycles, état stable...

Pour l'instant nous nous contenterons de programmer une version unidimensionnelle du problème et de regarder ses cycles.

1 Une classe pour les diriger tous...

Le programme ne sera constitué que d'une seule classe publique, `Life`, dans le fichier `Life.java`. Cette classe contient la méthode `public static void main(String args[])` qui représente le corps du programme. Toutes les autres méthodes de la classe devront être déclarées `static`.

On considère un tableau d'entier (`int[]`) de longueur `n` qui contiendra des 0 et des 1 : 1 si la cellule est pleine et 0 si elle est vide. Ecrivez une méthode `int[] randomize(int n)` qui prend en entrée un entier correspondant à la taille du tableau que l'on veut obtenir et qui retourne une tableau de cette longueur, rempli aléatoirement de 0 ou de 1. On rappelle que pour tirer un entier au hasard entre 0 et 1 on peut utiliser la ligne de commandes : `(int) Math.floor(Math.random()*2)`. Il faut ensuite utiliser une boucle `for` pour remplir toutes les cases du tableau.

On veut aussi pouvoir visualiser facilement l'état de notre population. Pour cela écrivez une méthode `static void afficher(int[] tab)` qui prend en argument une liste d'entiers et qui affiche sont contenu à l'écran, soit directement avec des 0 et des 1, soit pour faire plus joli avec par exemple des + pour les 1 et des − pour les 0.

Dans le `main` créez une variable `int n` et donnez y une valeur de votre choix (pas tellement plus de 30 ou 40 pour tenir sur une seul ligne du terminal). Créez un tableau aléatoire de longueur `n` et faites le afficher.

— **Compilez** et corrigez d'éventuelles erreurs —

N'hésitez pas à executer plusieurs fois le programme pour regarder les différentes sorties aléatoires.

2 Evolution :

On va maintenant essayer de faire évoluer ce tableau. On considère la règle suivante :

- si les voisins d'une cases sont tous les deux vides ou tous les deux pleins, alors la case change d'état (elle se vide si elle était pleine et se remplit si elle était vide)
- si l'un des voisins est vide et l'autre est plein la case reste telle qu'elle

On considère que le tableau est cyclique, c'est-à-dire que la première et la dernière case sont voisines. Ainsi les voisins de la première case sont la deuxième et la dernière case.

Écrivez une méthode `static int[] evolve(int [] tab)` qui prend en argument un tableau d'entier correspondant à une population et qui renvoie le tableau correspondant à cette population après une étape de l'évolution. ⚠ il faut certainement traiter les cas des deux cases des bouts à part. On peut aussi utiliser la commande `%` qui retourne le reste d'une division entière : par exemple `7 % 4 = 3` ou `132 % 25 = 7`.

Dans le `main` écrivez une boucle qui affiche une vingtaine d'évolutions successives d'un tableau aléatoire.

— **Compilez** et corrigez d'éventuelles erreurs —

3 Comparaisons :

Ecrivez une méthode `static boolean compare(int[] tab1, int[] tab2)` qui prend en arguments deux tableau et les compare. Elle renvoie `true` s'il sont identiques et `false` autrement.

À l'aide de la méthode compare on va essayer de trouver des *points fixes*, c'est-à-dire des tableaux de population qui restent identiques quand ils évoluent. Dans le `main` écrivez une boucle `for` qui va executer 5000 fois les instructions suivantes :

- tirer un tableau aléatoire `int[] t` de la longueur de votre choix (entre 8 et 40)
- le comparer avec sont évolution
- s'ils sont identiques afficher le tableau : c'est un point fixe.

— **Compilez** et corrigez d'éventuelles erreurs —

On va maintenant chercher des *cycles de longueur deux*, c'est-à-dire des populations qui redeviennent identiques après 2 évolutions. Modifiez la boucle précédente pour que la comparaison ne se fasse pas entre une population et son évolution, mais entre un population et l'évolution de son évolution. On peut ainsi voir, en plus de points fixes, des cycles d'ordre 2.

— **Compilez** et corrigez d'éventuelles erreurs —

4 Lecture d'arguments :

Jusqu'à présent si vous vouliez changer la longueur des tableaux il fallait modifier votre `main` et recompiler le programme. On va maintenant lire les arguments du programme pour pouvoir donner la longueur directement comme argument. L'argument `String args[]` de la fonction `main` est la liste des chaines de caractères données en argument au programme. Par exemple quand vous executez `java Life coucou les enfants` la variable `args[]` vaut {`"coucou"`,`"les"`,`"enfants"`}, donc `args[0]` vaut `"coucou"`. Avec la méthode `Integer.parseInt()` on peut convertir une chaine de caractères en le nombre qu'elle contient. par exemple `Integer.parseInt("10")` renvoie l'entier 10.

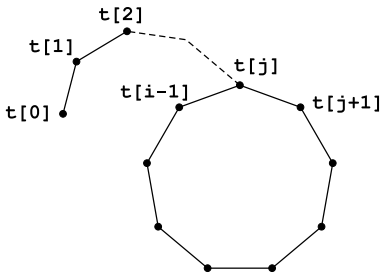
Modifiez le `main` pour qu'il stoque dans une variable `int n` la valeur qu'on lui passe en premier argument. ⚠ il faut penser à d'abord vérifier que l'on a bien donné un argument au programme. Dans le cas contraire faites lui afficher un message d'erreur (par exemple : `"Donnez un argument au programme !"`) et quitter le programme (avec `System.exit(1)`).

Utilisez cette variable `n` comme longueur pour vos tableaux de population.

— **Compilez** et corrigez d'éventuelles erreurs —

5 Étude des cycles :

On veut étudier les cycles des populations. Pour cela on va tirer une population au hasard et regarder toutes ces évolutions. Cependant pour savoir si il y a un cycle il faut regarder si à un moment elle retombe sur un état déjà rencontré : il faut donc garder un historique des populations précédentes.



La plus simple (mais pas le plus efficace) est de créer un tableau de populations (un `int[][]`) qui contiendra toutes les populations depuis le début, et à chaque étape de comparer la nouvelle population à toutes les anciennes. Ainsi si à l'étape `i` on se rend compte que la population est la même qu'à l'étape `j`, c'est que l'on a un cycle de longueur `i-1-j` qui commence à l'étape `j`. Pour simplifier on va considerer que l'on laisse tomber les cycles trop long et on se limite à une longueur fixée, par exemple à 5000.

Ecrivez ce qu'il faut dans le `main` pour pouvoir repérer les cycles. Une fois que vous en avez trouvé un, faites afficher l'ensemble des états (en séparant bien les états qui mennent au cycle de ceux du cycle à proprement parlé) et faites afficher la longueur du cycle.

— **Compilez** et corrigez d'éventuelles erreurs —