

 <p>ÉCOLE SUPÉRIEURE D'INFORMATIQUE</p>	Projet UML / Java	UE : programmation & modélisation objet
		[A2]

**EXIA ANNEE 2**

# J-Sim Forest



## Le contexte

L'ONF (l'Office National des Forêts) a fait appel à vous pour le développement d'un outil qui permettrait de simuler le développement des arbres et végétaux dans une forêt. Ce type de simulation est très utile pour permettre aux décideurs de cet organisme de prendre des décisions en fonction des projections dans le temps que le logiciel fournira. Il peut permettre par exemple d'analyser l'occupation des sols, la diffusion d'un incendie, ou d'une invasion d'insectes. C'est à partir de ces besoins que le projet **J-Sim Forest** a vu le jour. Le langage Java a été choisi par l'ONF pour sa portabilité sur les différents OS.

Dans ce document, il vous est présenté le contexte « scientifique » qui est simplement ici pour formaliser le discours. **Il ne faut pas bloquer sur cette partie**, puisque la partie suivante « modèles de la simulation » est très explicite, et n'a pas de dépendance directe avec la partie formelle.

**Note importante :** N'oubliez pas que vous pouvez utiliser le forum du module « Projet UML/JAVA » présent sur Moodle pour poser vos questions. Les réponses seront ainsi visibles par tous.

## Définitions

Ce type de projet nécessite un minimum de formalisme. Un modèle de calcul tiré de l'Informatique théorique et des Mathématiques vous permettra de mener à bien votre projet : **les automates cellulaires**.

### Définition d'un automate cellulaire

Un automate cellulaire  $A = (d, S, V, \delta)$  est la donnée de :

1. Sa *dimension*  $d \in \mathbb{N}^*$ , qui est la dimension de son *réseau*  $\mathbb{Z}^d$ .
2. Un ensemble fini d'états  $S$ , appelé *alphabet*.
3. Son voisinage  $V = (v_1, v_2, \dots, v_n) \in (\mathbb{Z}^d)^n$
4. Sa règle locale de transition  $\delta: S^n \rightarrow S$

















On appelle alors *configuration* l'attribution d'un état à chaque cellule du réseau (étape 0).

### Exemple de l'automate élémentaire à 1 dimension

Soit l'automate élémentaire  $A = (d, S, V, \delta)$  défini comme :

1.  $d=1$  : les cellules de l'automate correspondent à des entiers relatifs.
2.  $S = \{\blacksquare, \square\}$  : les cellules n'ont que deux états possibles.
3.  $V = \{-1, 0, 1\}$  : l'état au temps  $t + 1$  d'une cellule  $\gamma$  dépend de l'état au temps  $t$  de  $\gamma$  et de ses deux voisins immédiats  $\gamma + 1$  et  $\gamma - 1$ .

La règle locale de transition de l'automate élémentaire est une fonction de  $S^3$  dans  $S$ . On peut donner cette règle  $\delta$  sous forme d'un tableau :

Pour la deuxième colonne, si la cellule  $\gamma$  du milieu est *noire*, que sa voisine à droite  $\gamma + 1$  est *blanche*, et que celle de gauche  $\gamma - 1$  est *noire*, alors la cellule  $\gamma$  sera *noire* à l'étape d'après ( $t+1$ ).

Un exemple d'exécution de cet automate est donné sur la page suivante.



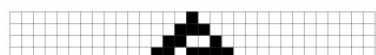
Etape 0



Etape 1

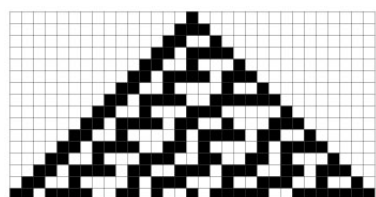


Etape 2



Etape 4

...



Etape 15

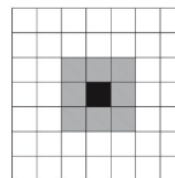
*Exemple de déroulement de l'automate cellulaire élémentaire (Wolfram 1983; 2002, p. 57).*

## Automate cellulaire à 2 dimensions

Pour ce projet, il faudra utiliser un automate à 2 dimensions, avec un certain nombre d'états. Un automate cellulaire 2D est une grille à maillage carré, soit un tableau à deux dimensions. Chaque case du tableau représente une cellule. Chaque cellule peut être dans un certain état. En fonction de ses voisins, elle change d'état à  $t+1$ . L'illustration la plus connue est celle du Jeu de la Vie (Conway, 1970).

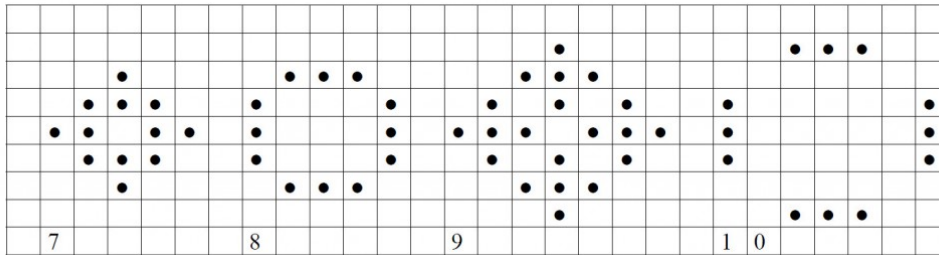
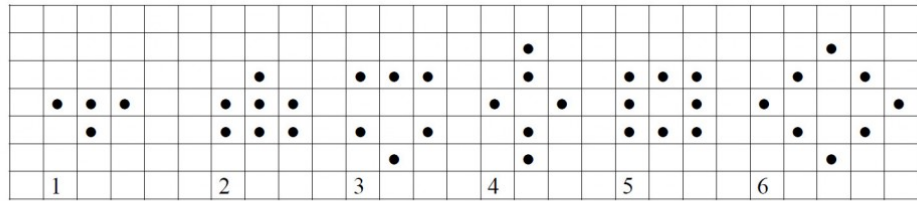
(pour plus d'infos, [http://fr.wikipedia.org/wiki/Jeu\\_de\\_la\\_vie](http://fr.wikipedia.org/wiki/Jeu_de_la_vie))

On se base sur le *voisinage de Moore* : une cellule possède 8 voisins.



Dans cet automate, les cellules peuvent prendre 2 états : *vivante* ou *morte*. Dans nos illustrations, **noir=vivant** et **blanc=mort**. A chaque étape, l'état suivant de chaque cellule est décidé en fonction de son état actuel et du nombre de cellules qui l'entourent :

- une cellule vivante ne survivra que si elle a 2 ou 3 voisins précisément (elle ne doit être ni isolée ni étouffée), sinon elle meurt.
- une cellule apparaît dans une case vide à l'étape suivante, si et seulement si cette case est entourée de précisément 3 cellules (car il faut de la vie pour engendrer la vie).



Exemple de déroulement du jeu de la vie sur 10 étapes.

## Modèles de la simulation

Votre travail consiste donc en la *simulation d'un automate cellulaire à 2 dimensions* dont voici les spécificités.

### Modèle de la forêt

Ce modèle représente la manière dont les arbres se développent dans leur espace naturel :

- la grille 2D représente un espace naturel (de taille 100x100 par exemple).
- chaque cellule peut prendre 4 états :
  - « vide » : ☐
  - « jeune pousse » : ☐
  - « arbuste » : ☐
  - « arbre » : ☐
- le *voisinage de Moore* est utilisé: une cellule possède 8 voisins (voir illustration plus haut).
- les règles de transition de l'automate sont les suivantes :
  - Naissance des arbres :
    - une cellule « vide » au temps  $t$  passera à l'état « jeune pousse » au temps  $t+1$  :
      - si elle possède au moins 2 cellules voisines à l'état « arbre »
      - OU
      - si elle possède au moins 3 cellules voisines à l'état « arbuste »
      - OU
      - si elle possède 1 cellule voisine à l'état « arbre » ET 2 cellules voisines à l'état « arbuste ».
    - sinon, la cellule reste « vide » au temps  $t+1$ .
  - Croissance des jeunes pousses :
    - une cellule « jeune pousse » au temps  $t$  passera à l'état « arbuste » au temps  $t+1$  si elle possède un nombre de voisins à l'état « arbre » ou « arbuste » inférieur ou égal à 3.
    - sinon la cellule reste à l'état « jeune pousse » au temps  $t+1$ .
  - Croissance des arbustes :
    - une cellule « arbuste » au temps  $t$  passera à l'état « arbre » au temps  $t+2$ .

Remarque : on ne simulera pas la mort des arbres. Les arbustes mettent plus de temps que les jeunes pousses à changer d'état.

## Modèles des risques

Une fois que la génération de forêt aura été effectuée, il est nécessaire de modéliser certains risques. Nous nous intéresserons à deux d'entre eux ici : les *feux de forêts* et les *invasions d'insectes*.

### Feux de forêts

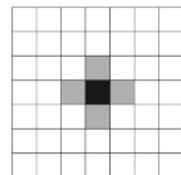
On simule ici la propagation des feux de forêts. Plus l'arbre est mature, plus il a de chances de brûler. Ce modèle est un modèle stochastique ; il fait appel à des règles de transitions *aléatoires* :

- en plus des 4 états de base du modèle de la forêt, 2 états viennent s'ajouter au modèle :
  - « en feu » : ■
  - « en cendre » : ■
- le *voisinage de Moore* est utilisé: une cellule possède 8 voisins (voir illustration plus haut).
- de nouvelles règles de transition s'ajoutent à l'automate:
  - Propagation du feu :
    - une cellule « jeune pousse », « arbuste », ou « arbre » au temps  $t$  qui possède au moins une cellule « en feu » dans son voisinage peut passer à l'état « en feu » au temps  $t+1$  selon les probabilités suivantes :
      - une « jeune pousse » a 25% de chances de passer à l'état « en feu ».
      - un « arbuste » a 50% de chances de passer à l'état « en feu ».
      - un « arbre » a 75% de chances de passer à l'état « en feu ».
  - Extinction du feu :
    - une cellule « en feu » au temps  $t$  passe à l'état « en cendre » à temps  $t+1$ .
  - Disparition des cendres :
    - une cellule « en cendre » au temps  $t$  passe à l'état « vide » à temps  $t+1$ .

### Invasion d'insectes

On simule ici l'attaque d'insectes sur les végétaux. Les jeunes arbres sont les plus vulnérables à ces attaques. Ce modèle est également un modèle stochastique ; il fait appel à des règles de transitions *aléatoires* :

- en plus des 4 états de base et des 2 états pour les feux de forêts, 1 état vient s'ajouter au modèle :
  - « infecté » : ■
- le *voisinage de Von Neumann* est utilisé : une cellule possède 4 voisins.
- de nouvelles règles de transition s'ajoutent à l'automate:
  - Propagation de l'infection :
    - une cellule « jeune pousse » ou « arbuste » au temps  $t$  qui possède au moins une cellule « infecté » dans son voisinage (de Von Neumann) peut passer à l'état « infecté » au temps  $t+1$  selon les probabilités suivantes :
      - une « jeune pousse » a 75% de chances de passer à l'état « infecté ».
      - un « arbuste » a 50% de chances de passer à l'état « infecté ».
      - un « arbre » a 25% de chances de passer à l'état « infecté ».
  - Disparition de l'infection :
    - Une cellule « infecté » au temps  $t$  passera à l'état « vide » au temps  $t+1$ .



# Simulation

Cette partie décrit le travail à réaliser : vous allez devoir implémenter les modèles qui vous ont été donnés ci-avant.

## Interfaces

- Initialisation de la simulation
  - Choix de la taille de la grille (100x100 par défaut).
  - Affichage d'une grille vide.
  - Nombre de pas de temps à simuler (la simulation s'arrête une fois ce nombre de pas atteint).
  - Choix de la rapidité d'exécution de la simulation.
  - Placement sur la grille d'éléments forestiers avec choix de l'élément (« jeune pousse », « arbuste », « arbre »).
- Exécution de la simulation (mode « croissance des arbres »)
  - Permettre l'exécution de la simulation en fonction du nombre de pas spécifié (mis à jour de la grille automatiquement).
  - Permettre l'exécution de la simulation en mode « pas à pas ».
  - Afficher le nombre de pas en cours.
  - Afficher la densité de cellules à l'état « jeune pousse », « arbuste » et « arbre ».
- Exportation/Chargement (voir Données)
  - Permettre de sauvegarder/charger la configuration actuelle de la simulation dans une BD.
  - Permettre d'exporter les différentes densités de cellules de la simulation en .csv par pas de temps (pour permettre un affichage graphique sous tableur par exemple).
- Exécution de la simulation (mode « feux de forêt »)
  - Placement sur la grille d'éléments de type « en feu ».
  - Permettre l'exécution de la simulation en fonction du nombre de pas spécifié (mis à jour de la grille automatiquement).
  - Permettre l'exécution de la simulation en mode « pas à pas ».
  - Afficher le nombre de pas en cours.
  - Afficher la densité de cellules à l'état « jeune pousse », « arbuste », « arbre » et « en feu ».
- Exécution de la simulation (mode « invasion d'insectes »)
  - Placement sur la grille d'éléments de type « infecté ».
  - Permettre l'exécution de la simulation en fonction du nombre de pas spécifié (mis à jour de la grille automatiquement).
  - Permettre l'exécution de la simulation en mode « pas à pas ».
  - Afficher le nombre de pas en cours.
  - Afficher la densité de cellules à l'état « jeune pousse », « arbuste », « arbre » et « infecté ».

## Données

- Hébergement des données dans une base (JDBC ou ODBC).
  - Configuration de la simulation en cours
    - Enregistrer la configuration en cours de la simulation (pour chaque cellule, on veut enregistrer sa position et son état).
    - Charger une configuration stockée en base (pour chaque cellule, on veut retrouver l'état dans lequel elle se trouvait au moment d'enregistrer).
    - Enregistrer/Charger les paramètres de la simulation (nombre de pas, vitesse, taille de la grille).
- Exportation de résultats dans un fichier (indépendant de la BD) (bonus)
  - Exporter les résultats de la simulation par pas selon la densité (entre 0 et 1) de chaque cellule possédant le même état (au format texte .csv, dans un ou plusieurs fichiers).

*Exemple :*

```
jeune pousse;arbuste;arbre;vide
0.1 ;0.05 ;0.08 ;0.77
0.11 ;0.06 ;0.08 ;0.75
...
```

## Déroulement du projet

Le projet débute le **Jeudi 31 Janvier 2013** et se termine le **Lundi 11 Février 2013** par une soutenance.

Il est conseillé de réaliser le projet par groupe de 3.

### Gestion de planning

Comme pour tout projet, une bonne gestion du temps sera nécessaire :

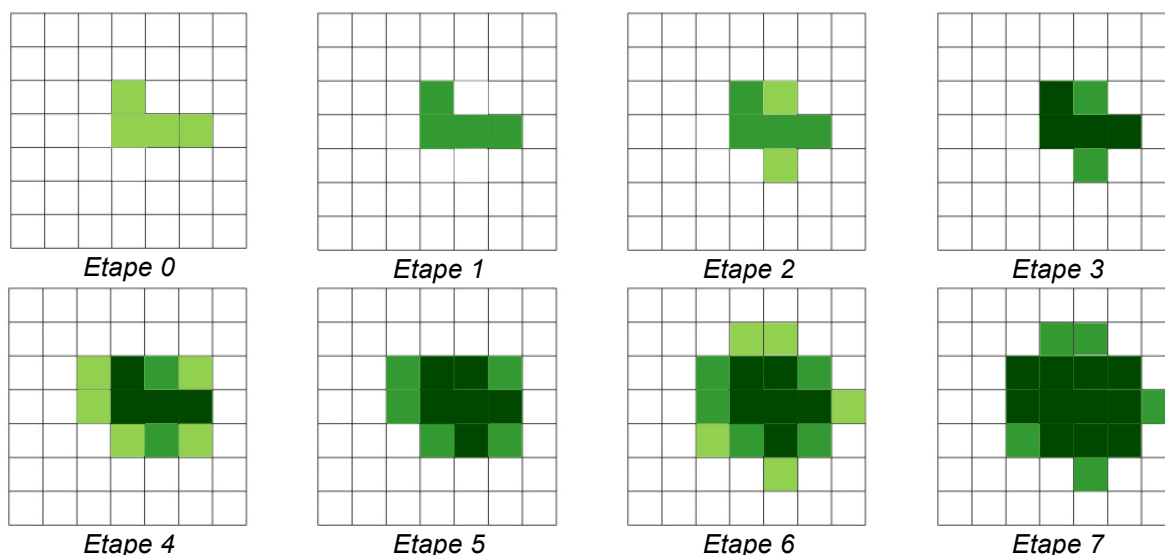
- Fixez-vous d'abord un plan d'attaque :
  - définissez les objectifs que vous vous fixez.
  - définissez les dates clefs, e.g. celles qui vous permettront de choisir telle ou telle option de développement suivant le temps disponible.
- Lancez le projet.
- Faites tous les jours des points d'avancement : ce sont ces points d'avancement qui vous permettent de piloter votre projet.
- Ne quittez pas des yeux les minimums attendus.
- Débutez le rapport et le diaporama de soutenance dès le début du projet puis complétez-les tout au long du projet : ces documents sont toujours à peu près identiques et ne doivent pas consommer un ou deux jours.
- N'oubliez pas que le nombre de membres de votre équipe doit faire votre force : aucun co-équipier ne doit rester sans tâche, et si c'est le cas c'est que votre préparation du projet a mal été faite.

## Dernières précisions

Dans cette partie, des précisions sont données sur le modèle et la simulation.

### Le modèle

Voici un exemple de déroulement du modèle forestier pour une certaine configuration initiale :



## La simulation

Pour la partie simulation, pensez à fournir une architecture logicielle simple, mais intelligente. Tous les concepts que vous avez vus en Java vous seront utiles : classes concrètes, classes abstraites, interfaces, exceptions, collections, événements, objets Swing, et liaison à une BD. Prévoyez un certain temps de discussion sur la partie analyse du problème afin de partir sur une « bonne » architecture logicielle dès le départ. Utilisez à bon escient les diagrammes UML pour discuter avec votre équipe (surtout les diagrammes de classes).

Pour la partie affichage de la grille, partez sur la librairie standard **Java2D**. Ne perdez pas de temps à vouloir optimiser cet affichage : si votre programme redessine toute la grille et les cellules à chaque pas de temps, cela ne pose pas de problème (voir annexe pour avoir une idée).

L'utilisation de threads est possible (car pratique) mais pas obligatoire. Vous pouvez faire le projet sans. Vous devrez être en mesure d'expliquer ce que vous avez fait au jury s'ils vous posent la question.

## Restitution du travail

### Fonctionnement

Pour situer ce qui est demandé sur l'aspect fonctionnel :

- La solution est considérée comme « *répondant parfaitement au cahier des charges fourni* » si **le programme simule le modèle de forestier ET au moins un des deux modèles de risques (feux de forêt OU invasion d'insectes)**.
- La solution comporte « *Quelques écarts* » si **seul le modèle forestier est implémenté**.

### Réalisation technique

Ci-dessous les points qu'est en droit d'attendre le client (jury) pour évaluer que votre programme est correctement réalisé :

- Vous avez, durant votre analyse, utilisé au moins **4 différents types de diagrammes UML** (dont certains qui explicitement donnés, voir plus bas).
- Votre solution utilise **les mécanismes classiques de la POO** (classes, attributs, méthodes, héritage, etc.), mais également les concepts de **classes abstraites** et/ou **interfaces**, le tout de **manière pertinente** (pas la peine d'insérer des interfaces ou classes abstraites qui ne servent à rien, cela se verra et vous sera reproché).
- Votre solution permet la **connexion à la base de données**, ainsi que les **fonctions de sauvegarde/chargement** de configuration de la simulation. De plus, l'**exportation en .csv** fonctionne.
- L'utilisation d'éléments Swing est avérée (*voir partie Interfaces*), et tous les événements qui les concernent sont gérés.



## Rapport

Le rapport doit être remis, sous forme électronique ou papier (au bon vouloir de votre encadrant) pour le **Vendredi 8 Février 2013 à 17h**.

Ce rapport contiendra au minimum :

### Partie 1 : Présentation

- Introduction
- Rôles des membres du groupe
- Rappel de la demande

### Partie 2 : Gestion de projet

- Planning prévisionnel
- Planning (réalisé)
- Synthèse sur l'organisation et le déroulement du projet (dont la répartition de la charge de travail)

### Partie 3 : Développement

- Analyse fonctionnelle (les diagrammes de cas d'utilisation seront présentés dans cette partie) – Ne pas oublier l'ordonnancement des priorités
- Maquette de l'IHM
- Conception UML commentée. Vos choix conceptuels doivent être justifiés.  
En cas d'écart entre l'analyse et la réalisation, des justifications doivent être fournies dans l'analyse des écarts. Tout diagramme non commenté ne sera pas pris en compte. La conception est une composante majeure du projet au même titre que la réalisation. Au minimum, parmi les 4 diagrammes qui sont à fournir, doivent être présentés:
  - Au moins un diagramme de cas d'utilisation détaillé
  - Au moins un diagramme de séquence et/ou d'activité
  - Au moins un diagramme de classesAinsi que un ou plusieurs diagrammes de type :
  - Paquetage, Etats-transition

Plus votre analyse sera complète, mieux ce sera. Les diagrammes UML ne servent pas à faire « joli ». Ils doivent avoir leur utilité. L'évolution de votre conception au cours du projet (modification d'un diagramme, ajout d'un diagramme) doit être clairement exposée, commentée, et justifiée.

Un point concernant votre conception UML pourra être planifiée avec votre encadrant. Le jour et l'heure de ce point vous sera précisée par celui-ci. Cette conception s'affinera bien évidemment tout au long du projet.

### Partie 4 : Conclusion & perspective

- Analyse des problèmes rencontrés et solutions apportées.  
Dans le cas où des solutions n'ont pu être trouvées (écarts entre la conception UML et le livrable technique, fonctionnalités non implémentées...), il faut proposer des pistes de solutions ou fournir des explications.  
Vous pouvez distinguer 3 types de solutions : curatives, préventives, correctives. Il est conseillé de résumer les problèmes et les solutions dans un tableau à la fin de cette partie.
- Evolution proposée pour le système
- Bilan personnel & groupe  
Le bilan personnel est une capitalisation d'expérience.  
Dans le bilan personnel apparaîtront les compétences initiales au démarrage du projet, et les compétences acquises durant le projet. Sur l'ensemble des compétences, il faudra distinguer les compétences initiales et celles acquises qui ont été nécessaires pour le bon déroulement du projet.

## Soutenance

Elle a lieu le **Lundi 11 Février 2012**.

La présentation est destinée au client (ONF).

Le travail, réalisé en groupe de trois, sera soutenu de préférence sous la forme suivante :

- présentation de 20 minutes,
- des questions/réponses individuelles de 10 minutes maximum,
- une délibération du jury,
- une restitution de 10 minutes.

## Petit guide de solutions techniques

### Architecture

Ce projet est simple s'il est bien construit, et pour bien le construire il faut passer le temps nécessaire à faire son étude, et qui dit étude dit UML. Par contre ne forcez pas pour produire du diagramme, vous n'y parviendriez pas. Il y a deux réflexions à mener de front :

- les besoins utilisateurs
- l'architecture

Le premier point est impératif car c'est celui-ci qui fixe le « quoi faire », le second l'est tout autant car il fixe presque le « comment faire ».

### Données

- Hébergement dans une base de données  
Pour cette partie là il n'y a a priori aucune difficulté particulière. Il ne faut pas trouver de code SQL dans l'interface graphique, ou plus généralement en dehors de la couche d'accès aux données.
- Modélisation de la base de données  
Faites quelque chose de cohérent et de simple. Le projet porte sur Java et UML, pas sur Merise.

### Interfaces

- Affichage de la simulation
  - L'utilisation de Java2D n'est pas trop compliquée. Vous pouvez à peu près tout dessiner. Dans votre cas, inutile de compliquer le travail : vous pouvez afficher ce qu'on vous demande en utilisant uniquement les méthodes dessinant des lignes et des rectangles. Tutos officiels : <http://docs.oracle.com/javase/tutorial/2d/index.html>
- Exécution de la simulation
  - Comme expliqué plus haut, votre simulation ne doit pas nécessairement utiliser des threads de manière explicite. L'utilisation de la classe `Timer` par exemple peut suffire. Un petit exemple est disponible en annexe. Le timer peut être arrêté avec la méthode `stop()`.
  - La simulation peut se faire en 2 temps : génération de la forêt, puis une fois celle-ci terminée, on place des éléments « en feu » ou « infecté » à la souris, et on fait continuer la simulation avec seulement les règles du modèle de risque sélectionné. Si le modèle forestier peut fonctionner en même temps que le modèle de risque, c'est encore mieux, mais ce n'est pas une contrainte du projet

## Annexe – Exemple d’affichage de disques avec Java2D

```
import java.awt.Color;
import java.awt.Graphics;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.Timer;

public class RandomDiscs extends JFrame
{
    private MyPanel panel;

    public RandomDiscs(int nbcount,int time)
    {
        super("My Simulation");
        panel=new MyPanel(nbcount,time);
        this.add(panel);

        setLocation(50, 50);
        setSize(400, 300);
        setVisible(true);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    public static void main(String[] args)
    {
        RandomDiscs rdiscs = new RandomDiscs(10,1000); //dessine 10 ellipses, 1 par seconde, puis arrête de
dessiner
    }

    //un JPanel "maison" implémentant l'interface ActionListener
    private class MyPanel extends JPanel implements ActionListener
    {
        private Timer clock;
        private int counter=0, stopcount; //permet d'arrêter l'affichage

        public MyPanel(int nbcount, int time)
        {
            stopcount=nbcount;
            clock = new Timer(time, this); //on prépare le timer
            clock.start(); //on démarre le timer
            this.setBackground(Color.white);
        }

        @Override
        public void actionPerformed(ActionEvent e) //sur chaque "top" d'horloge du Timer, une action est
faite.
        {
            //System.out.println("Action "+counter);
            if(counter>=stopcount-1) //si on arrive au nombre de pas de temps désiré, on stoppe le Timer.
                clock.stop();
            else
                repaint(); //sinon on redessine le JPanel.

            counter++;
        }

        @Override
        public void paint(Graphics g) //méthode où l'on dessine ses objets
        {
            // Trois nombres aléatoires entre 0 et 255 pour la couleur.
            int r = (int) (Math.random() * 256);
            int v = (int) (Math.random() * 256);
            int b = (int) (Math.random() * 256);

            // Coordonnées et taille de l'ovale.
            int x = (int) (Math.random() * getWidth());
            int y = (int) (Math.random() * getHeight());
            int taille = 10 + (int) (Math.random() * 65);

            // Dessine le disque.
            g.setColor(new Color(r, v, b));
            g.fillOval(x, y, taille, taille);
        }
    }
}
```