

Annexe 1 : Guide de l'utilisation

En lance l'applet de Game Of Life, il y a deux parties d'interface :

- Une partie Gille et Cellules
- Une autre partie panneau de contrôle.

Sur la partie panneau de contrôle, certain bouton pour paramètre les génération du cellules et certains autre menu déroulant sont pour choisis l'algorithme et choisi la forme qui a été enregistrer.

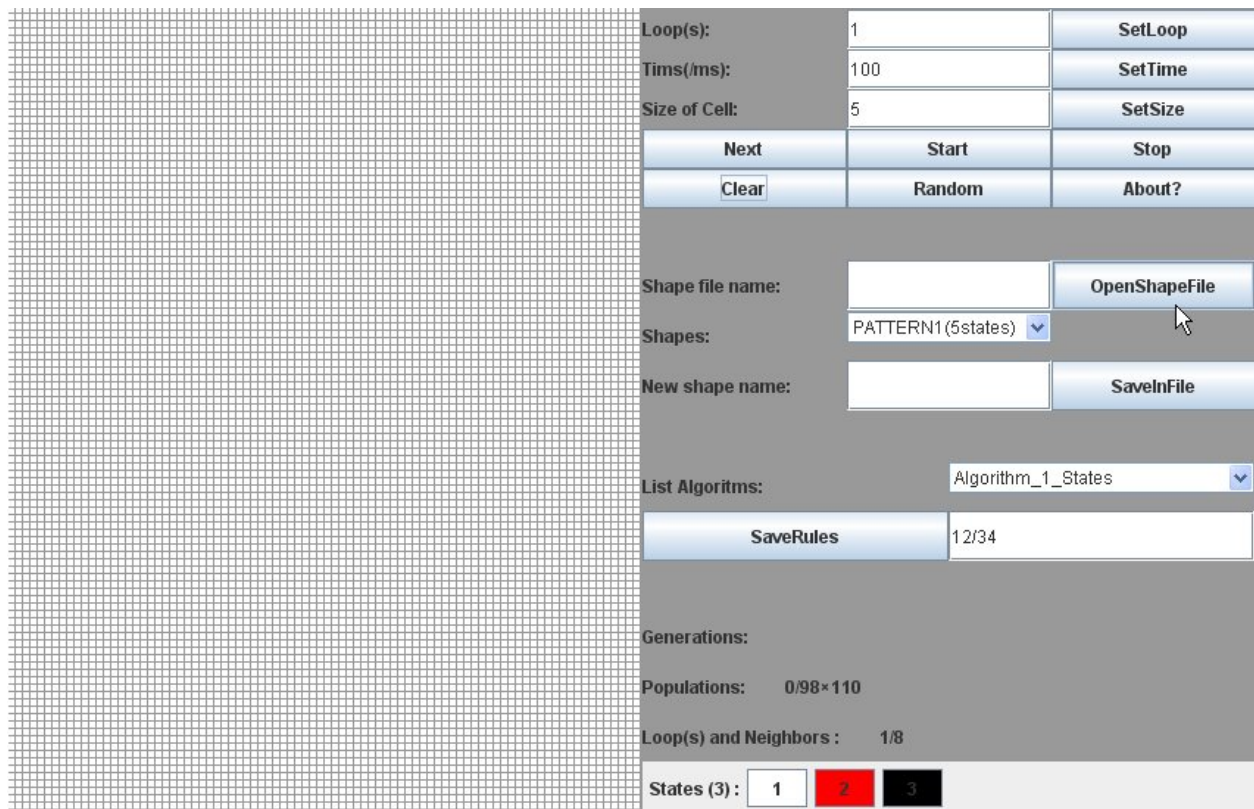
Au moment de charge l'applet, certain forme fichier est charge automatiquement par la prédéfini dans un fichier '\source\shapes\shapes.txt'. Tous les formes dans ces fichier est charger et mettre dans la menu déroulant.

Une entre de teste « Shape file name » est pour charger un autre fichier de forme (il est sous forme XML) dans ton disque local. La chemin de fichier est commence par '**file:/**'. (Exemple : 'file:/c:/exemple.xml'). Et pour valider le chargement appuyez sur le bouton « OpenShapeFile », si tous se passe bien, le forme est ajouté dans la menu déroulant « Shapes », si non il va afficher un erreur sur la panneau.

Pour créer une fichier de forme, il faut **créer le fichier (XML) par la main**. La fichier XML est contient :

```
<?xml version="1.0" encoding="UTF-8"?>
<shapes>
</shapes>
```

Pour Save une forme dans un fichier, il faut avoir une fichier Open en cours. Et plus donnée la nom de cette forma dans la entre de texte « New shape name » et appuyer la bouton « SaveInFile ».



Annexe 2 : Développer une classe de l'algorithme

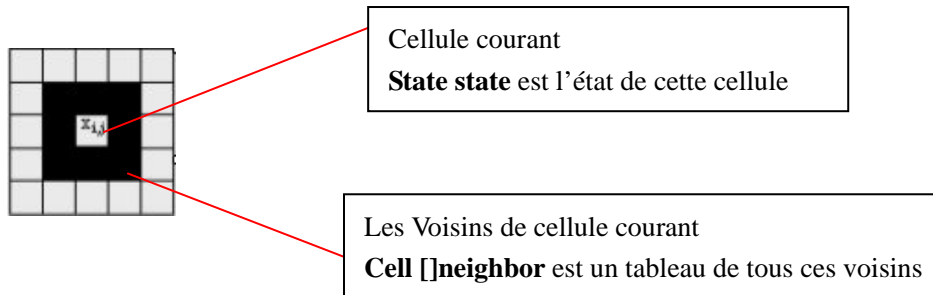
Pour créer un algorithme, il suffit de créer une classe qui étend la classe « Algorithm ».

Dans la nouvelle classe algorithme, il faut :

- Compléter la méthode :

public abstract State getCellStateByNeighbor(State state, Cell []neighbor);

Deux paramètres sont la valeur d'entrée de cellule courante.



- Donner la valeur des trois variabilités :

/*

* The objet States.

* In your class, you must redefine this value.

* They are made up of the color.

* Ex: 'Color colors[]=new Color[]{ Color.white,Color.red,Color.black};'

* 'states=new States(3,colors);'

*/

public States states;

/*

* The number loop, its value is possible change with applet interface.

* In your class, for the method Util.getNombreNeighbor, you need this value.

*/

public int neighbor_loop;

/*

* The String rule, its value is possible change with applet interface.

* The form of this value is what you want (Ex: '14/45/12' or '1,2,3/4,5'...)

* In your class, for the method getCellStateByNeighbor(), you need this value.

*/

public String rules;

Attention : Après la création de la classe, pour que la classe soit valable dans le programme, il faut ajouter le nom de classe dans le fichier «\source\algorithm\listeAlgorithms.txt».

Annexe 3 : Un exemple de la classe d'algorithme

```
package org.univ.paris5.GameOfLife.algorithms;

import org.univ.paris5.GameOfLife.Algorithm;
import org.univ.paris5.GameOfLife.States;
import org.univ.paris5.GameOfLife.State;
import org.univ.paris5.GameOfLife.Cell;
import org.univ.paris5.GameOfLife.Util;

import java.awt.Color;

/**
 * One algorithm with 3 state.
 *
 * @author PengFei DONG
 * @author Ke LIANG
 */
public class Algorithm_8_States extends Algorithm {
    public Algorithm_8_States(){
        Color colors[]=new Color[]{ Color.white,Color.red,Color.black,Color.green };
        states=new States(4,colors);
        neighbor_loop=1;
        rules="22/22";
    }
    public State getCellStateByNeighbor(State state, Cell []neighbor ){
        int n1=0,n2=0,n3=0,n4=0,i;
        String nb[]=rules.split("/");
        String nb1=nb[0];
        String nb2=nb[1];
        int nombreNeighbor=Util.getNombreNeighbor(neighbor_loop);
        //Algorithme....
        for(i=0;i<nombreNeighbor;i++)
        {
            if(neighbor[i].state.getIndex().equals("1")){ n1++;continue;}
            if(neighbor[i].state.getIndex().equals("2")){ n2++;continue;}
            if(neighbor[i].state.getIndex().equals("3")){ n3++;continue;}
            if(neighbor[i].state.getIndex().equals("4")){ n4++;continue;}
        }

        if (state.getIndex().equals("2"))
        {
            if(nb1.contains(""+n2)) {return states.getStateByIndex("2");}
        }
    }
}
```

```

else
{
    if(nb1.contains(""+n3)) {return states.getStateByIndex("3");}
    else
    {
        return states.getStateByIndex("1");}
    }
}
else
{
    if (state.getIndex().equals("3"))
    {
        if(nb1.contains(""+n2)) {return states.getStateByIndex("2");}
        else
        {if(nb1.contains(""+n4)) {return states.getStateByIndex("4");}
        else
            return states.getStateByIndex("1");
        }
    }
    else
    {
        if(nb2.contains(""+n1)) {return states.getStateByIndex("3");}
        else
        {
            if(nb2.contains(""+n2)) {return states.getStateByIndex("2");}
            else
            {   if(n1>n4) return states.getStateByIndex("1");
                //System.out.println("*****3");
                else return states.getStateByIndex("4");}
            }
        }
    }
}
}
}

```

Annexe 4 : Les classes diagramme en UML

■ Les classes d'élément



■ Les classes d'algorithme

