**Project Java**

The project involves performing a simulation of an ant colony. The project must include at least 3 different types of ants: the queens (who do not move and live for several years), the workers (who live a few weeks) and the warriors (who live a few months). The queen produces workers and warriors at a certain pace and in a certain proportion. The workers spend their time looking for food or materials to build the nest. They move randomly to search but then return directly to the nest (using the pheromones deposited on the ground) to report what they found. Warriors roam randomly in search of threat (large animal or other ant hill) and attack intruders. Threatening animals see their strength drop with each attack of a warrior and die if they are attacked enough. The behavior of threatening animals (how to move, speed, etc.) depends on the type of animal.

Ants consume the food stored in the nest. Each type of ant has its own consumption (for example the queen consumes much more food than workers or warriors). The quantity of food present in the anthill must allow to feed all the ants, otherwise their life expectancy decreases strongly. When there is not enough food, the queen is fed first, then the workers, and the warriors are the least priority. When an ant is not fed for a certain time (depending on its type), it dies.

The land is broken down into small squares that can contain the nest, food (in finite quantity), building materials, a threat or an ant. The contents of the boxes may change randomly over time (food or a threat may appear on a box). The content can also change depending on the actions of the ants (if a warrior kills a threatening animal, the corpse is food).

**Timing :**

The simulation is done in turn, all the entities evolve once per turn. To delay the game, you can use the try {Thread.sleep (n);} catch (InterruptedException e) {} statement that puts the program on hold n milliseconds. One can also use the class javax.swing.Timer which triggers the call of a function at regular intervals.

**Graphic**

For viewing and controlling the program, a GUI\_for\_Displayable class is provided. It displays any object that implements the Displayable interface provided as well. This interface imposes that each displayable object gives the form which represents it (instance of java.awt.Shape), the color used to display it (instance of java.awt.Color) and the text associated with its position. To have multi-line text, you must add \ n in the string.

The GUI\_for\_Displayable class provides constructors for creating an interface with a colored background or with a background image, methods for adding and removing a Displayable interface, and a getClic () method that returns an instance of java.awt.MouseEvent corresponding to the user's last untreated click (clicks are stored in a queue). A MouseEvent describes a click with a position, the button pressed, the keys pressed at the moment of the click (Shift, Alt, ...).

The class also offers a setBottomFieldText (String s) method that specifies the text displayed in the text component at the bottom of the window, and a displayMessage (String s) method that opens a dialog window to display the string. It is possible to display the ratio production / consumption, but this ratio can also be displayed using curves by a Displayable type object.

**Bonus :**

It is possible to simulate the creation of a new nest from a queen: regularly (usually in the spring), a queen gives birth to sexed ants (and winged to be able to go to found new nests far from where they are born). Females that are fertilized can start a new nest. Unfertilized females and males die in a few days.

It is possible to simulate cooperative behaviors. If you consider that a pile of food, or materials, is too important to be brought back to the anthill on a trip by an ant, she will ask her colleagues. She must be able to inform other workers of her discovery and they will follow the trail of pheromones to go get food or materials.

It is possible to simulate malnutrition: an ant that has not eaten when it should move less quickly.

It is possible to represent several nests, and to simulate wars between ant hills.

It is possible to simulate catastrophic events (passing an anteater that eats a lot of the occupants of the nest, flooding the nest after a heavy rain, etc.) to see how and in how long the nest will rebuild (or no).

**Class you need to use for the display**

**Displayable.java :**

package display;

public interface Displayable{

public java.awt.Shape getShape();

public java.awt.Color getColor();

public String getString();

public java.awt.Point getStringPosition();

}

**GUI\_for\_Displayable.java :**

package display;

import javax.swing.\*;

import java.awt.\*;

import java.awt.geom.Rectangle2D;

import java.awt.event.\*;

import java.io.IOException;

import java.lang.IllegalArgumentException;

import java.util.ArrayList;

import java.awt.image.BufferedImage;

import javax.imageio.ImageIO;

import java.io.File;

public class GUI\_for\_Displayable extends JFrame implements MouseListener{

private int offset = 12;

private float fontSize = 10.5f;

private Graphic p;

private Color bg = null;

private BufferedImage img = null;

private ArrayList<Displayable> elements;

private ArrayList<MouseEvent> clics;

private JTextField jtf;

public GUI\_for\_Displayable(String title, int width, int height, Color bg){

super(title);

this.elements = new ArrayList<Displayable>();

this.clics = new ArrayList<MouseEvent>();

this.bg = bg;

this.setLayout(new BorderLayout());

this.p = new Graphic();

this.p.addMouseListener(this);

this.getContentPane().add(this.p,BorderLayout.CENTER);

jtf = new JTextField(20);

this.getContentPane().add(jtf,BorderLayout.SOUTH);

this.setSize(width,height);

this.setResizable(false);

this.setLocation((java.awt.Toolkit.getDefaultToolkit().getScreenSize().width-this.getWidth())/2,(java.awt.Toolkit.getDefaultToolkit().getScreenSize().height-this.getHeight())/2);

this.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

this.setVisible(true);

}

public GUI\_for\_Displayable(String title, String imageFile) throws IOException{

super(title);

this.elements = new ArrayList<Displayable>();

this.clics = new ArrayList<MouseEvent>();

this.bg = Color.BLACK;

this.img = ImageIO.read(new File(imageFile));

this.setLayout(new BorderLayout());

this.p = new Graphic();

this.p.addMouseListener(this);

this.getContentPane().add(this.p,BorderLayout.CENTER);

jtf = new JTextField(20);

this.getContentPane().add(jtf,BorderLayout.SOUTH);

this.setSize(img.getWidth(),img.getHeight()+48);

this.setResizable(false);

this.setLocation((java.awt.Toolkit.getDefaultToolkit().getScreenSize().width-this.getWidth())/2,(java.awt.Toolkit.getDefaultToolkit().getScreenSize().height-this.getHeight())/2);

this.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

this.setVisible(true);

}

public void addDisplayable(Displayable d){

this.elements.add(d);

this.p.repaint();

}

public boolean removeDisplayable(Displayable d){

if(this.elements.remove(d)){

this.p.repaint();

return true;

}

else return false;

}

public void mouseClicked(MouseEvent e){

this.clics.add(e);

}

public void mouseEntered(MouseEvent e){}

public void mouseExited(MouseEvent e){}

public void mousePressed(MouseEvent e){}

public void mouseReleased(MouseEvent e){}

public MouseEvent getClic(){

if(this.clics.isEmpty()) return null;

else return this.clics.remove(0);

}

private class Graphic extends JPanel{

public void paint(Graphics gr){

if(GUI\_for\_Displayable.this.bg != null){

gr.setColor(GUI\_for\_Displayable.this.bg);

gr.fillRect(0,0,this.getWidth(),this.getHeight());

}

if(GUI\_for\_Displayable.this.img != null){

gr.drawImage(GUI\_for\_Displayable.this.img,0,0,this);

}

for(Displayable d:GUI\_for\_Displayable.this.elements){

gr.setColor(d.getColor());

Shape sh = d.getShape();

if(sh instanceof java.awt.geom.Line2D || sh instanceof java.awt.geom.Path2D){

((Graphics2D) gr).setStroke(new BasicStroke(5.0f));

((Graphics2D) gr).draw(sh);

}

else ((Graphics2D) gr).fill(sh);

gr.setColor(Color.WHITE);

Font f = gr.getFont().deriveFont(Font.BOLD);

f = f.deriveFont(fontSize+2);

gr.setFont(f);

Point p = d.getStringPosition();

String[] s = d.getString().split("\n");

/\*int max = 0;

for(int i=1;i<s.length;i++){if(s[i].length()>s[max].length()) max = i;}

Rectangle2D r2d = f.getStringBounds(s[max],((Graphics2D) gr).getFontRenderContext());

gr.setColor(Color.WHITE);

gr.fillRect(p.x,p.y,(int) r2d.getWidth(),(int) (r2d.getHeight()\*s.length+((s.length-1)\*GUI\_for\_Displayable.this.offset)));

gr.setColor(Color.BLACK);\*/

for(int i=0;i<s.length;i++) gr.drawString(s[i],p.x,p.y+(i+1)\*GUI\_for\_Displayable.this.offset);

}

}

}

public void displayMessage(String m){

JOptionPane.showMessageDialog(this,m);

}

public void setBottomFieldText(String s){

this.jtf.setText(s);

}

}