```java
package Serveur.fenetre;

import java.awt.Color;
import java.awt.FontMetrics;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Rectangle;
import java.awt.RenderingHints;

import Serveur.maths.vectors.Vector2d;

public class Drawer {
    private Graphics g;
    private FontMetrics metrics = null;

    public Drawer(Graphics g) {
        setGraphics(g);
    }

    public void setAntiAliasing(boolean antiAliasing) {
        if (antiAliasing)
            ((Graphics2D) g).setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                    RenderingHints.VALUE_ANTIALIAS_ON);
        else
            ((Graphics2D) g).setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                    RenderingHints.VALUE_ANTIALIAS_OFF);
    }

    /**
     * @param xa
     *             : l'abscisse point d'origine de la ligne
     * @param ya
     *             : l'ordonnée point d'origine de la ligne
     * @param xb
     *             : l'abscisse le point d'arrivée de la ligne
     * @param yb
     *             : l'ordonnée le point d'arrivée de la ligne
     * @param largeur
     *             : la largeur de la ligne
     */
    public void drawBigLine(double xa, double ya, double xb, double yb,
            double largeur) {
        Vector2d normale = new Vector2d(yb - ya, xa - xb).normalize()
                .mult(largeur / 2);
        Vector2d A = new Vector2d(xa, ya).add(normale);
        Vector2d B = new Vector2d(xa, ya).sub(normale);
        Vector2d C = new Vector2d(xb, yb).sub(normale);
        Vector2d D = new Vector2d(xb, yb).add(normale);

        g.fillPolygon(
                new int[]{(int) Math.round(A.x), (int) Math.round(B.x),
                        (int) Math.round(C.x), (int) Math.round(D.x)},
                new int[]{(int) Math.round(A.y), (int) Math.round(B.y),
                        (int) Math.round(C.y), (int) Math.round(D.y)},
                4);
    }

    /**
     * @param xa
     *             : l'abscisse point d'origine de la ligne
     * @param ya
     *             : l'ordonnée point d'origine de la ligne
```

```
63      * @param xb
64      *            : l'abscisse le point d'arrivée de la ligne
65      * @param yb
66      *            : l'ordonnée le point d'arrivée de la ligne
67      * @param largeur
68      *            : la largeur de la ligne
69      */
70     public void drawBigRoundedLine(double xa, double ya, double xb, double yb,
71             double largeur) {
72         drawBigLine(xa, ya, xb, yb, largeur);
73         g.fillOval((int) (xa - Math.round(largeur / 2)),
74                 (int) (ya - Math.round(largeur / 2)),
75                 (int) (Math.round(largeur)), (int) (Math.round(largeur)));
76         g.fillOval((int) (xb - Math.round(largeur / 2)),
77                 (int) (yb - Math.round(largeur / 2)),
78                 (int) (Math.round(largeur)), (int) (Math.round(largeur)));
79     }
80
81     public Graphics getGraphics() {
82         return g;
83     }
84
85     public void setGraphics(Graphics g) {
86         this.g = g;
87     }
88
89     /**
90      * Permet d'obtenir une couleur, en fonction d'une valeur de x, parmis un
91      * <br>
92      * dégradé correspondant a une liste de couleur et une liste d'extremums
93      * <br>
94      * correspondants.
95      *
96      * @param colors
97      *            : La liste des couleurs. Dois avoir le même nombre d'éléments
98      *            que extremums.
99      * @param extremums
100     *            : La liste des extremums. Dois avoir le même nombre d'éléments
101     *            que colors.
102     * @param x
103     *            : La valeur à convertir en couleur.
104     * @return La couleur correspondante à la description.
105     */
106    public void setScaleColor(Color[] colors, float[] extremums, float x) {
107        g.setColor(getScaleColor(colors, extremums, x));
108    }
109
110    /**
111     * Permet d'obtenir une couleur, en fonction d'une valeur de x, parmis un
112     * <br>
113     * dégradé correspondant a une liste de couleur et une liste d'extremums
114     * <br>
115     * correspondants.
116     *
117     * @param colors
118     *            : La liste des couleurs. Dois avoir le même nombre d'éléments
119     *            que extremums.
120     * @param extremums
121     *            : La liste des extremums. Dois avoir le même nombre d'éléments
122     *            que colors.
123     * @param x
124     *            : La valeur à convertir en couleur.
```

```java
     * @return La couleur correspondante à la description.
     */
    public static Color getScaleColor(Color[] colors, float[] extremums,
            float x) {
        int nbElements = colors.length;
        float min = extremums[0], max = extremums[nbElements - 1];

        if (nbElements != extremums.length)
            return null;

        if (x < min)
            x = max - (max - x) % (max - min);
        else if (x > max)
            x = (x - min) % (max - min) + min;

        Color retour = null;

        for (int i = 0; i < nbElements - 1 && retour == null; i++) {
            if (x >= extremums[i] && x <= extremums[i + 1]) {
                float ratio = (x - extremums[i])
                        / (extremums[i + 1] - extremums[i]);
                retour = new Color(
                        (int) (ratio * colors[i + 1].getRed())
                                + (int) ((1 - ratio) * colors[i].getRed()),
                        (int) (ratio * colors[i + 1].getGreen())
                                + (int) ((1 - ratio) * colors[i].getGreen()),
                        (int) (ratio * colors[i + 1].getBlue())
                                + (int) ((1 - ratio) * colors[i].getBlue()));
            }
        }

        return retour;
    }

    /**
     * Ecrit un texte centré dans la zone désirée.
     *
     * @param g
     *            la destination
     * @param text
     *            le texte à écrire
     * @param rect
     *            le rectangle dans lequel il dois être centré
     * @param font
     *            la police à utiliser
     */
    public void drawCenteredString(String text, Rectangle rect) {
        if (metrics == null)
            metrics = g.getFontMetrics();

        int x = (rect.width - metrics.stringWidth(text)) / 2;
        int y = ((rect.height - metrics.getHeight()) / 2) + metrics.getAscent();
        g.drawString(text, x + rect.x, y + rect.y);
    }

    /**
     * Ecrit un texte et réalise la translation désirée.
     *
     * @param g
     *            la destination
     * @param text
     *            le texte à écrire
```

```
187      * @param x
188      *            les coordonnées initiales du texte
189      * @param y
190      *            les coordonnées initiales du texte
191      * @param translate_x
192      *            le taux de décalage du texte en abscisse
193      * @param translate_y
194      *            le taux de décalage du texte en ordonnée
195      * @param font
196      *            la police à utiliser
197      */
198     public void drawTranslatedString(String text, int x, int y,
199             double translate_x, double translate_y) {
200         if (metrics == null)
201             metrics = g.getFontMetrics();
202 
203         int pos_x = x + (int) (metrics.stringWidth(text) * translate_x);
204         int pos_y = y + (int) (metrics.getHeight() * translate_y);
205         g.drawString(text, pos_x, pos_y);
206     }
207 }
```