```java
package Serveur.triangulation;

import java.awt.Graphics;
import java.awt.Rectangle;
import java.io.Closeable;
import java.util.ArrayList;

import Serveur.maths.Utils;
import Serveur.maths.vectors.Vector2d;

public class Capteur implements Closeable {
    public static final int MAX_LENGTH_HISTORIQUE = 400;
    public static final int ALPHA_SMOOTH = 10;

    private ArrayList<Vector2d> historiquePos = new ArrayList<Vector2d>();
    private ArrayList<Double> historiqueDistance = new ArrayList<Double>();
    private ArrayList<Double> historiqueRSSI = new ArrayList<Double>();
    private ArrayList<Long> historiqueTempsRSSI = new ArrayList<Long>();
    private ArrayList<Double> historiqueSmoothRSSI = new ArrayList<Double>();
    private ArrayList<Double> historiqueFiltredRSSI = new ArrayList<Double>();
    private Vector2d pos;
    private double distance = 1;
    private double RSSI = 0;

    public Capteur(double x, double y) {
        setPosition(x, y);
    }

    public Vector2d getPosition() {
        return pos.clone();
    }

    public Vector2d[] getHistoriquePos() {
        return historiquePos.toArray(new Vector2d[historiquePos.size()]);
    }

    public void setPosition(double x, double y) {
        pos = new Vector2d(x, y);
        historiquePos.add(pos);

        while (historiquePos.size() > MAX_LENGTH_HISTORIQUE)
            historiquePos.remove(0);
    }

    public double getDistance() {
        return distance;
    }

    public Double[] getHistoriqueDistances() {
        return historiqueDistance
                .toArray(new Double[historiqueDistance.size()]);
    }

    public void setDistance(double distance) {
        this.distance = distance;
        historiqueDistance.add(distance);

        while (historiqueDistance.size() > MAX_LENGTH_HISTORIQUE)
            historiqueDistance.remove(0);
    }

    public double getRSSI() {
```

```
63          return RSSI;
64      }
65
66      public Double[] getHistoriqueRSSI() {
67          return historiqueRSSI.toArray(new Double[historiqueRSSI.size()]);
68      }
69
70      public Double[] getHistoriqueSmoothRSSI() {
71          return historiqueSmoothRSSI
72                  .toArray(new Double[historiqueSmoothRSSI.size()]);
73      }
74
75      public Double[] getHistoriqueFiltredRSSI() {
76          return historiqueFiltredRSSI
77                  .toArray(new Double[historiqueFiltredRSSI.size()]);
78      }
79
80      public void setRSSI(double RSSI) {
81          this.RSSI = RSSI;
82          historiqueRSSI.add(RSSI);
83          historiqueTempsRSSI.add(System.currentTimeMillis());
84
85          while (historiqueRSSI.size() > MAX_LENGTH_HISTORIQUE)
86              historiqueRSSI.remove(0);
87          while (historiqueTempsRSSI.size() > MAX_LENGTH_HISTORIQUE)
88              historiqueTempsRSSI.remove(0);
89
90          int size = historiqueRSSI.size();
91          double smooth = RSSI;
92
93          if (size >= ALPHA_SMOOTH) {
94              smooth = 0;
95
96              for (int i = 1; i <= ALPHA_SMOOTH; i++)
97                  smooth += historiqueRSSI.get(size - i);
98
99              smooth /= ALPHA_SMOOTH;
100             historiqueSmoothRSSI.add(smooth);
101
102             while (historiqueSmoothRSSI.size() > MAX_LENGTH_HISTORIQUE)
103                 historiqueSmoothRSSI.remove(0);
104         }
105
106         double filtred = smooth;
107
108         if (size == 1)
109         {
110             historiqueFiltredRSSI.add(RSSI);
111         }
112         else if (size >= 2)
113         {
114             double dt = (double) (historiqueTempsRSSI.get(size - 1) -
    historiqueTempsRSSI.get(size - 2)) / 1000;
115             double tau = 5 * dt;
116             double lastRSSI = historiqueRSSI.get(size - 2);
117             double lastFiltred = historiqueFiltredRSSI.get(size - 2);
118             double coeff = 1 / (tau + dt / 2);
119             int method = 1;
120             filtred = RSSI;
121
122             if (method == 1)
123                 filtred = dt / tau * (RSSI - lastFiltred) + lastFiltred;
```

```
124             else if (method == 2)
125                     filtred = coeff * ((tau - dt / 2) * lastFiltred + dt * (RSSI -
    lastRSSI));
126
127             historiqueFiltredRSSI.add(filtred);
128         }
129
130         while (historiqueFiltredRSSI.size() > MAX_LENGTH_HISTORIQUE)
131             historiqueFiltredRSSI.remove(0);
132
133         double distance = 0.02 * Math.pow(10, -filtred / 20); // en metres
134         setDistance(distance);
135     }
136
137     public void paintComponent(Graphics g, double xmin, double xmax,
138             double ymin, double ymax) {
139         Rectangle rect = g.getClipBounds();
140
141         int x = (int) Utils.map(pos.x, xmin, xmax, rect.x, rect.x + rect.width);
142         int y = (int) Utils.map(pos.y, ymin, ymax, rect.y + rect.height,
143                 rect.y);
144         double distance = getDistance();
145
146         int radius_x = (int) (distance * rect.width / (xmax - xmin));
147         int radius_y = (int) (distance * rect.height / (ymax - ymin));
148
149         int epaisseur = 3;
150
151         for (int i = 0; i < epaisseur; i++)
152             g.drawOval(x - (radius_x + i), y - (radius_y + i),
153                     2 * (radius_x + i), 2 * (radius_y + i));
154
155         g.drawLine(x - 3, y - 3, x + 3, y + 3);
156         g.drawLine(x - 3, y + 3, x + 3, y - 3);
157     }
158
159     public void close() {
160         historiquePos.clear();
161         historiqueRSSI.clear();
162         historiqueTempsRSSI.clear();
163         historiqueDistance.clear();
164         historiqueSmoothRSSI.clear();
165         historiqueFiltredRSSI.clear();
166     }
167 }
```