CUT THE ROPE

Materia:

Programación lll

Equipo:

José Ramón Robledo Aguilar

Jessica Guerrero Carrera

Adriana Torres De León

Ingeniería en Sistemas Computacionales

4°A

Profesor:

Eduardo Serna Pérez

27 junio 2019

CUT THE ROPE

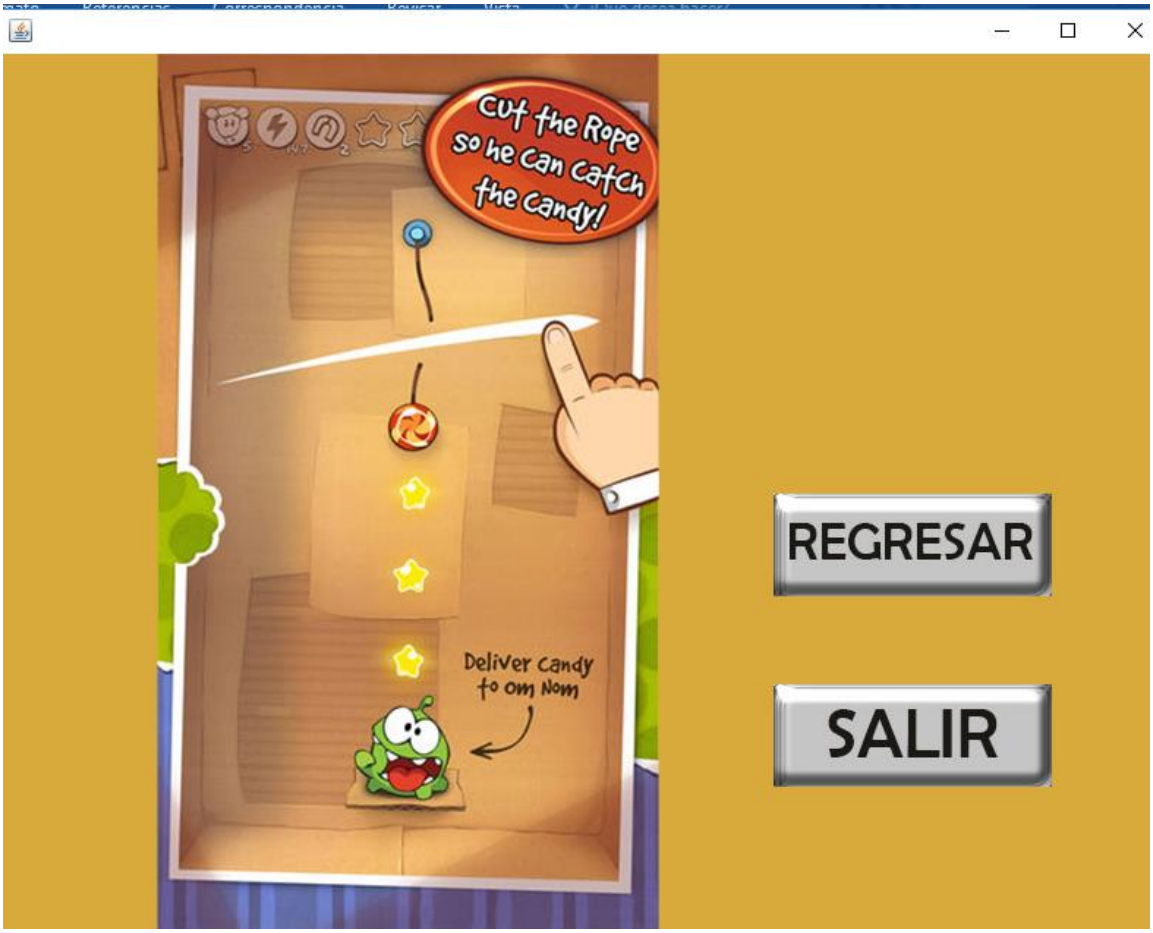
Nuestro proyecto consiste en un juego llamado Cut the Rope, su objetivo es dar un caramelo a la mascota que está atado a unas cuerdas que tenemos que cortar para lograr alimentar a la mascota mientras colectas estrellas con el movimiento del caramelo, los cuales te dan una mejor puntuación.

El juego se desarrolla de la siguiente manera:

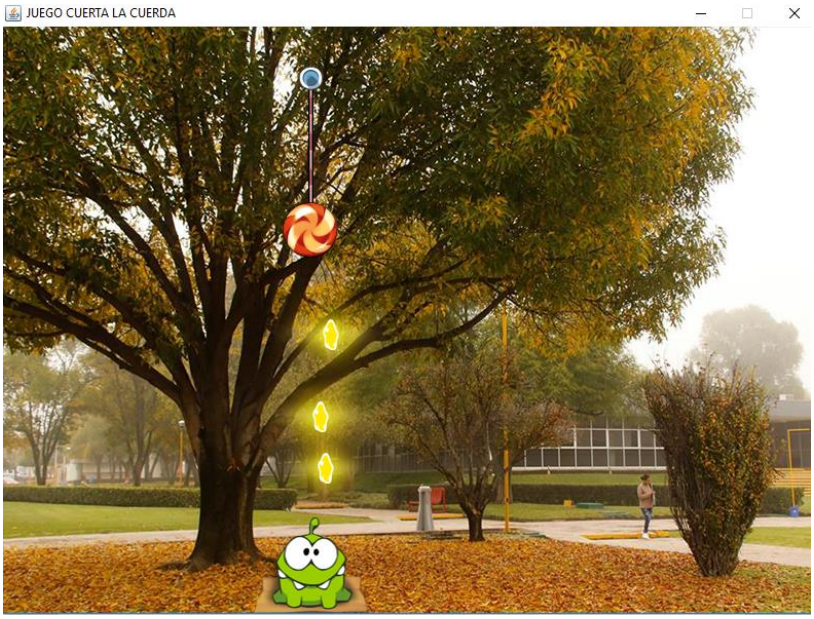
Tiene un menú de inicio donde presenta el juego

El botón instrucciones muestra cómo se debe jugar, salir pues como su nombre lo advierte termina la aplicación y Jugar va directamente al funcionamiento.

Daremos clic en las instrucciones y se mostrara esto:



Después vemos el juego, mostrara el primer nivel, el juego cuenta con 6 niveles:



El juego se va desarrollando con los diferentes niveles.

Fortalezas:

Es un juego fluido, cuenta con sonido y es bastante entretenido con los 6 niveles que van aumentando de dificultad.

Para el código usamos las librerías Graphics2d.

Dabilidades:

Un error que tenemos es que en determinado punto el sonido se detiene y en algunos niveles, al momento en que la mascota atrapa el caramelo, termina mostrando la imagen de la mascota triste, el cual únicamente cuando el caramelo sale de pantalla, o cuando se destruye.

BITACORA

Estos fueron las actividades que realizamos a grandes rasgos:

Integrante:	Actividades realizadas:	Tiempo que tomó en realizar dicha actividad:
Ramón	Realizo la construcción de los niveles.	5 días

Jessica	Implementó los menús y el diseño del flujo del juego	3 días
Adriana	Realizo los gráficos necesarios para el proyecto e investigo la forma de usar la gravedad y animaciones de java en netbeans.	3 días

CONCLUSIÓN.

Como conclusión java es un lenguaje bastante bueno para el programador, en este proyecto aprendimos el uso de las animaciones, hilos, de las gravedades para el diseño del juego.

Es interesante lo que nos ofrece el lenguaje, botones ya realizado, todo lo necesario para realizar una de las mejores interfaces.

Código:

------VInstrucciones.java-----

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */


/**
 *
 * @author
 */
public class VInstrucciones extends javax.swing.JFrame {

    /**
     * Creates new form VInstrucciones
     */
    public VInstrucciones() {
        initComponents();
        this.setLocationRelativeTo(null);
    }


    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
```

```
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        jPanel1 = new javax.swing.JPanel();
        Salir = new javax.swing.JButton();
        Regresar = new javax.swing.JButton();
        jLabel1 = new javax.swing.JLabel();

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        setMaximumSize(new java.awt.Dimension(800, 600));
        setMinimumSize(new java.awt.Dimension(800, 600));
        setSize(new java.awt.Dimension(800, 600));
        getContentPane().setLayout(new org.netbeans.lib.awtextra.AbsoluteLayout());

        jPanel1.setMaximumSize(new java.awt.Dimension(800, 600));
        jPanel1.setName(""); // NOI18N
        jPanel1.setPreferredSize(new java.awt.Dimension(800, 600));
        jPanel1.setLayout(new org.netbeans.lib.awtextra.AbsoluteLayout());

        Salir.setIcon(new javax.swing.ImageIcon(getClass().getResource("/salir.jpg"))); // NOI18N
        Salir.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                SalirActionPerformed(evt);
            }
        });
        jPanel1.add(Salir, new org.netbeans.lib.awtextra.AbsoluteConstraints(530, 430, 190, 70));

        Regresar.setIcon(new javax.swing.ImageIcon(getClass().getResource("/REGRE.jpg"))); // NOI18N
        Regresar.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                RegresarActionPerformed(evt);
            }
        });
        jPanel1.add(Regresar, new org.netbeans.lib.awtextra.AbsoluteConstraints(530, 300, 190, 70));

        jLabel1.setIcon(new javax.swing.ImageIcon(getClass().getResource("/NuevaIn.jpg"))); // NOI18N
        jPanel1.add(jLabel1, new org.netbeans.lib.awtextra.AbsoluteConstraints(0, 0, 800, 600));

        getContentPane().add(jPanel1, new org.netbeans.lib.awtextra.AbsoluteConstraints(0, 0, 800, 600));

        pack();
```

```
    }// </editor-fold>

    private void RegresarActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        new Ventana1().setVisible(true);
        this.setVisible(false);
    }

    private void SalirActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        System.exit(0);
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        /* Set the Nimbus look and feel */
        //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
        /* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
         * For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
         */
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
        } catch (ClassNotFoundException ex) {
            java.util.logging.Logger.getLogger(VInstrucciones.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        } catch (InstantiationException ex) {
            java.util.logging.Logger.getLogger(VInstrucciones.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        } catch (IllegalAccessException ex) {
            java.util.logging.Logger.getLogger(VInstrucciones.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {
            java.util.logging.Logger.getLogger(VInstrucciones.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        }
        //</editor-fold>

        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new VInstrucciones().setVisible(true);
```

```
        }
    });
  }

  // Variables declaration - do not modify
  private javax.swing.JButton Regresar;
  private javax.swing.JButton Salir;
  private javax.swing.JLabel jLabel1;
  private javax.swing.JPanel jPanel1;
  // End of variables declaration
}
```

-----Ventana1.java-----

```
import Animacion.Ventana2;
import java.applet.Applet;
import java.applet.AudioClip;
import javax.swing.JFrame;
import javax.swing.SwingUtilities;

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

/**
 *
 * @author
 */
public class Ventana1 extends javax.swing.JFrame {


  /**
   * Creates new form Ventana1
   */
  public Ventana1() {
    initComponents();
    this.setLocationRelativeTo(null);
    Sound.BALL.play();
  }


  /**
   * This method is called from within the constructor to initialize the form.
   * WARNING: Do NOT modify this code. The content of this method is always
   * regenerated by the Form Editor.
   */
  @SuppressWarnings("unchecked")
  // <editor-fold defaultstate="collapsed" desc="Generated Code">
  private void initComponents() {


    jPanel1 = new javax.swing.JPanel();
    Jugar = new javax.swing.JButton();
    Ins = new javax.swing.JButton();
    Salir = new javax.swing.JButton();
    jLabel2 = new javax.swing.JLabel();


setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
```

```
        setMaximumSize(new
java.awt.Dimension(800, 600));

        getContentPane().setLayout(new
org.netbeans.lib.awtextra.AbsoluteLayout());


        jPanel1.setLayout(new
org.netbeans.lib.awtextra.AbsoluteLayout());


        Jugar.setIcon(new
javax.swing.ImageIcon(getClass().getResource("/j
ug.jpg"))); // NOI18N

        Jugar.setBorder(null);

        Jugar.setBorderPainted(false);

        Jugar.addActionListener(new
java.awt.event.ActionListener() {

            public void
actionPerformed(java.awt.event.ActionEvent evt)
{

                JugarActionPerformed(evt);

            }

        });

        jPanel1.add(Jugar, new
org.netbeans.lib.awtextra.AbsoluteConstraints(30
, 460, 190, 70));


        Ins.setIcon(new
javax.swing.ImageIcon(getClass().getResource("/i
nstrucciones.jpg"))); // NOI18N

        Ins.setBorder(null);

        Ins.addActionListener(new
java.awt.event.ActionListener() {

            public void
actionPerformed(java.awt.event.ActionEvent evt)
{

                InsActionPerformed(evt);

            }

        });

        jPanel1.add(Ins, new
org.netbeans.lib.awtextra.AbsoluteConstraints(31
0, 460, 190, 70));


        Salir.setIcon(new
javax.swing.ImageIcon(getClass().getResource("/s
alir.jpg"))); // NOI18N

        Salir.setBorder(null);

        Salir.addActionListener(new
java.awt.event.ActionListener() {

            public void
actionPerformed(java.awt.event.ActionEvent evt)
{

                SalirActionPerformed(evt);

            }

        });

        jPanel1.add(Salir, new
org.netbeans.lib.awtextra.AbsoluteConstraints(59
0, 460, 190, 70));


        jLabel2.setFont(new
java.awt.Font("Tahoma", 2, 11)); // NOI18N

        jLabel2.setIcon(new
javax.swing.ImageIcon(getClass().getResource("/
PortadaNueva.jpg"))); // NOI18N

        jLabel2.setBorder(new
javax.swing.border.SoftBevelBorder(javax.swing.
border.BevelBorder.RAISED));

        jPanel1.add(jLabel2, new
org.netbeans.lib.awtextra.AbsoluteConstraints(0,
0, -1, 600));


        getContentPane().add(jPanel1, new
org.netbeans.lib.awtextra.AbsoluteConstraints(0,
0, -1, 600));
```

```
        pack();
    }// </editor-fold>

    private void SalirActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        System.exit(0);
    }

    private void InsActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        new VInstrucciones().setVisible(true);
        this.setVisible(false);
    }

    private void JugarActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        this.setVisible(false);
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                Ventana2 v = new Ventana2();
                JFrame frame = new JFrame();
                frame.setTitle("JUEGO CUERTA LA CUERDA");
                frame.getContentPane().add(v);
                frame.setSize(800, 600);
                frame.setLocationRelativeTo(null);
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                frame.setResizable(false);
                frame.setVisible(true);
                v.requestFocus();
                v.Comenzar();
            }

        });
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        /* Set the Nimbus look and feel */
        //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
        /* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
         * For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
         */
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
```

```
            }
        }
    } catch (ClassNotFoundException ex) {
java.util.logging.Logger.getLogger(Ventana1.class
.getName()).log(java.util.logging.Level.SEVERE,
null, ex);
    } catch (InstantiationException ex) {
java.util.logging.Logger.getLogger(Ventana1.class
.getName()).log(java.util.logging.Level.SEVERE,
null, ex);
    } catch (IllegalAccessException ex) {
java.util.logging.Logger.getLogger(Ventana1.class
.getName()).log(java.util.logging.Level.SEVERE,
null, ex);
    } catch
(javax.swing.UnsupportedLookAndFeelException
ex) {
java.util.logging.Logger.getLogger(Ventana1.class
.getName()).log(java.util.logging.Level.SEVERE,
null, ex);
    }
    //</editor-fold>

    /* Create and display the form */
    java.awt.EventQueue.invokeLater(new
Runnable() {
        public void run() {
            new Ventana1().setVisible(true);

        }
    });
  }

  // Variables declaration - do not modify
  private javax.swing.JButton Ins;
  private javax.swing.JButton Jugar;
  private javax.swing.JButton Salir;
  private javax.swing.JLabel jLabel2;
  private javax.swing.JPanel jPanel1;
  // End of variables declaration
}
class Sound {
        public static final AudioClip BALL =
Applet.newAudioClip(Sound.class.getResource("r
ope.wav"));
}
```

----- Ventana2.java -----

```
/*
 * To change this license header, choose License
Headers in Project Properties.
 * To change this template file, choose Tools |
Templates
 * and open the template in the editor.
 */
package Animacion;

import static
Animacion.TodaAnimacion.JuegoIni.*;
import java.awt.Rectangle;
import java.io.IOException;
import javax.imageio.ImageIO;
import java.awt.Font;
import java.awt.Shape;
import java.awt.font.TextLayout;
```

```
import java.io.InputStream;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.Point;
import java.util.ArrayList;
import java.util.List;
import java.awt.Canvas;
import java.awt.Color;
import java.awt.image.BufferStrategy;
import java.awt.Stroke;
import java.awt.BasicStroke;
import java.awt.Graphics2D;
import java.awt.RenderingHints;
import java.awt.image.BufferedImage;
import java.awt.Polygon;
import java.awt.geom.Point2D;
import java.awt.geom.AffineTransform;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.util.HashMap;
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;



/**
 *
 * @author
 */
public class Ventana2 extends Canvas {

    public static final int AnchoPantalla = 800;
    public static final int AltoPantalla = 600;

    private BufferStrategy bs;
    private TodaAnimacion ani;

    public Ventana2() {
        setBackground(Color.BLACK);
        Mouse mouse = new Mouse();
        addMouseListener(mouse);
        addMouseMotionListener(mouse);
        ani = new TodaAnimacion();
    }

    public void Comenzar() {
        ani.Comenzar();
        createBufferStrategy(2);
        bs = getBufferStrategy();
        new Thread(new Ciclo()).start();
    }

    public void Actualizar() {
        ani.Actualizar();
    }

    public void ActualizarCosas() {
        ani.ActualizarCosas();
    }
```

```
    public void Dibujar(Graphics2D g) {
        ani.Dibujar(g);
    }

    private class Ciclo implements Runnable {
        @Override
        public void run() {
            boolean running = true;
            while (running) {
                Time.Actualizar();
                Actualizar();
                while (Time.needsUpdate()) {
                    ActualizarCosas();
                }
                Graphics2D g = (Graphics2D)
bs.getDrawGraphics();
                g.setBackground(Color.BLACK);
                g.clearRect(0, 0, getWidth(),
getHeight());

g.setRenderingHint(RenderingHints.KEY_ANTIALI
ASING, RenderingHints.VALUE_ANTIALIAS_ON);

g.setRenderingHint(RenderingHints.KEY_INTERPO
LATION,
RenderingHints.VALUE_INTERPOLATION_BILINEA
R);
                Dibujar(g);
                g.dispose();
                bs.show();

                try {
                    Thread.sleep(5);
                } catch (InterruptedException ex) {
                }
            }
        }
    }

}

class TodaAnimacion {

    private final Estructura EstructuraPe = new
Estructura(800, 600, 10);
    protected List<Todo> todito = new
ArrayList<Todo>();
    protected List<Todo> TodoNivel = new
ArrayList<Todo>();

    public static enum JuegoIni { IniciarJ, es, titulo,
NIV, AMONOS, JUGAR, NivelCOMPLE, PERDIO2 }

    private JuegoIni JuegoCuerda =
JuegoIni.IniciarJ;

    private int currentLevel = 1;

    public TodaAnimacion() {
    }

    public Estructura getEstructura() {
        return EstructuraPe;
    }

    public JuegoIni getJuegoIni() {
        return JuegoCuerda;
```

```
    }

    public void setState(JuegoIni JuegoCuerda) {
        if (this.JuegoCuerda != JuegoCuerda) {
            this.JuegoCuerda = JuegoCuerda;
            for (Todo tod : TodoNivel) {
                tod.JuegoCuerdaChanged(JuegoCuerda);
            }
            for (Todo tod : todito) {
                tod.JuegoCuerdaChanged(JuegoCuerda);
            }
        }
    }

    public void Comenzar() {
        createEntidadesTodas2();
        ComenzarEntidadesTodas2();
    }

    private Efecto1Todo fadeEffect;
    private CortinaTodo curtain;
    private void createEntidadesTodas2() {
        fadeEffect = new Efecto1Todo(this);
        curtain = new CortinaTodo(this);

        todito.add(new IniciaTodo(this, fadeEffect));
        todito.add(new preTodo(this, fadeEffect));
        todito.add(new TitleTodo(this, fadeEffect,
curtain));

        todito.add(curtain);
        todito.add(new PerderTodo(this, fadeEffect,
curtain));
        todito.add(new LevelClearedTodo(this,
fadeEffect, curtain));
        todito.add(fadeEffect);
    }

    private void EntidadesTodas() {
        TodoNivel.add(new BackgroundTodo(this));

        TodoNivel.add(new PetTodo(this, curtain));

        for (AirCushion airCushion :
EstructuraPe.getAirCushions()) {
            TodoNivel.add(new AirCushionTodo(this,
airCushion));
        }

        for (Rope rope : EstructuraPe.getRopes()) {
            TodoNivel.add(new RopeTodo(this, rope));
        }
        // pin ropes
        for (PinRope pinRope :
EstructuraPe.getPinRopes()) {
            TodoNivel.add(new PinRopeTodo(this,
pinRope));
        }
        // espinas
        for (Spikes spikes :
EstructuraPe.getSpikesList()) {
            TodoNivel.add(new SpikesTodo(this,
spikes));
        }
```

```
        // dulce
        TodoNivel.add(new DulceClasTodo(this));
        // stars
        for (Star star : EstructuraPe.getStars()) {
            TodoNivel.add(new StarTodo(this, star));
        }
        // burbuja
        for (Bubble bubble :
EstructuraPe.getBubbles()) {
            TodoNivel.add(new BubbleTodo(this,
bubble));
        }
    }

    private void ComenzarEntidadesTodas2() {
        for (Todo tod : todito) {
            tod.Comenzar();
        }
    }

    private void ComenzarEntidadesTodas3() {
        for (Todo tod : TodoNivel) {
            tod.Comenzar();
        }
    }

    public void Actualizar() {
        for (Todo tod : TodoNivel) {
            tod.Actualizar();
        }
        for (Todo tod : todito) {
            tod.Actualizar();
        }
    }

    public void ActualizarCosas() {
        if (Mouse.pressed) {
            List<Point> trail =
EstructuraPe.getSlashTrail().getTrail();
            if (trail.size() > 0) {
                Point p = trail.get(trail.size() - 1);
                if (p != null && p.x >= 0 && p.y >= 0) {
                    EstructuraPe.addSlashTrail((int) (p.x +
0.5 * (Mouse.x - p.x)), (int) (p.y + 0.5 * (Mouse.y -
p.y)));
                }
                EstructuraPe.addSlashTrail((int)
Mouse.x, (int) Mouse.y);
            }
        }
        else {
            EstructuraPe.addSlashTrail(-1, -1);
        }
        for (Todo tod : TodoNivel) {
            tod.ActualizarCosas();
        }
        for (Todo tod : todito) {
            tod.ActualizarCosas();
        }
        EstructuraPe.Actualizar();
    }

    public void Dibujar(Graphics2D g) {
```

```
        for (Todo tod : TodoNivel) {
            if (tod.isVisible()) {
                tod.Dibujar(g);
            }
        }
        for (Todo tod : todito) {
            if (tod.isVisible()) {
                tod.Dibujar(g);
            }
        }

        EstructuraPe.getSlashTrail().drawDebug(g);
    }


    public void ComenzarNivel(int level) {
        currentLevel = level;

EstructuraPe.ComenzarNivel("/Imagenes/level_"
+ level + ".txt");
        TodoNivel.clear();
        EntidadesTodas();
        ComenzarEntidadesTodas3();
        setState(JUGAR);
    }
    public void ReiniciarNivel() {
        ComenzarNivel(currentLevel);
    }

    public void backToTitle() {
        setState(titulo);
    }

    public void nextLevel() {
        ComenzarNivel(currentLevel + 1);
    }
}

 class Animacion {

    private Map<String, List<BufferedImage>>
frames = new HashMap<String,
List<BufferedImage>>();
    private Map<String, Point> positions = new
HashMap<String, Point>();
    private Map<String, Boolean> loops = new
HashMap<String, Boolean>();

    private double frameRate;

    private String currentAnimacionName;
    private List<BufferedImage> currentAnimacion;
    private double currentFrame;
    private Point currentPosition;
    private boolean currentLoop;

    public void addFrames(String animationName,
String fileBaseName, int ComenzarFrame, int
endFrame, int x, int y, boolean loop) {
        addFrames(animationName, fileBaseName,
ComenzarFrame, endFrame, x, y, loop, 50);
    }

    public void addFrames(String animationName,
String fileBaseName, int ComenzarFrame, int
```

```
endFrame, int x, int y, boolean loop, double
frameRate) {
      List<BufferedImage> f = new
ArrayList<BufferedImage>();
      for (int i = ComenzarFrame; i <= endFrame;
i++) {
         try {
            BufferedImage sprite =
ImageIO.read(getClass().getResourceAsStream("/I
magenes/" + fileBaseName + i + ".png"));
            f.add(sprite);
         } catch (IOException ex) {

Logger.getLogger(Animacion.class.getName()).log
(Level.SEVERE, null, ex);
            System.exit(-1);
         }
      }
      frames.put(animationName, f);
      positions.put(animationName, new Point(x,
y));
      loops.put(animationName, loop);
      this.frameRate = frameRate;
   }

   public String getCurrentAnimacionName() {
      return currentAnimacionName;
   }

   public List<BufferedImage>
getCurrentAnimacion() {
      return currentAnimacion;
   }

   public double getCurrentFrame() {
      return currentFrame;
   }

   public void setCurrentFrame(double
currentFrame) {
      this.currentFrame = currentFrame;
   }

   public Point getCurrentPosition() {
      return currentPosition;
   }

   public boolean isCurrentLoop() {
      return currentLoop;
   }

   public boolean isFinished() {
      if (currentLoop) {
         return false;
      }
      else if ((int) currentFrame ==
currentAnimacion.size() - 1) {
         return true;
      }
      return false;
   }

   public void selectAnimacion(String
animationName) {
      currentAnimacionName = animationName;
```

```
      currentAnimacion =
frames.get(animationName);
      currentFrame = 0;
      currentPosition =
positions.get(animationName);
      currentLoop = loops.get(animationName);
   }

   public void Actualizar() {
      if (currentAnimacion == null) {
         return;
      }
      currentFrame = (currentFrame + frameRate *
Time.getDelta());
      if (currentLoop) {
         currentFrame = currentFrame %
currentAnimacion.size();
      }
      else {
         if ((int) currentFrame >
currentAnimacion.size() - 1) {
            currentFrame = currentAnimacion.size()
- 1;
         }
      }
   }

   public void Dibujar(Graphics2D g) {
      if (currentAnimacion == null) {
         return;
      }
      BufferedImage sprite =
currentAnimacion.get((int) currentFrame);
      g.drawImage(sprite, currentPosition.x,
currentPosition.y, null);
   }

}

 class Todo extends TodaAnimacion{

   protected boolean visible = false;
   protected TodaAnimacion ani;
   protected BufferedImage image;

   protected int InsPunt;
   protected long waitTime;

   public Todo(TodaAnimacion ani) {
      this.ani = ani;
   }

   public boolean isVisible() {
      return visible;
   }

   public void setVisible(boolean visible) {
      this.visible = visible;
   }

   protected BufferedImage
loadImageFromResource(String resource) {
      try {
```

```
            image =
ImageIO.read(getClass().getResourceAsStream(re
source));
        } catch (IOException ex) {

Logger.getLogger(Todo.class.getName()).log(Leve
l.SEVERE, null, ex);
            System.exit(-1);
        }
        return image;
    }

    public void Comenzar() {
    }

    public void Actualizar() {
        switch (ani.getJuegoIni()) {
            case IniciarJ: ActualizarInitializing(); break;
            case es: Actualizarpre(); break;
            case titulo: ActualizarTitle(); break;
            case NIV: ActualizarLevelSelect(); break;
            case AMONOS: ActualizarReady(); break;
            case JUGAR: ActualizarPlaying(); break;
            case NivelCOMPLE:
ActualizarLevelCleared(); break;
            case PERDIO2: ActualizarPerder(); break;
        }
    }

    public void ActualizarCosas() {
        switch (ani.getJuegoIni()) {
            case IniciarJ: ActualizarCosasInitializing();
break;
            case es: ActualizarCosaspre(); break;
            case titulo: ActualizarCosasTitle(); break;
            case NIV: ActualizarCosasLevelSelect();
break;
            case AMONOS: ActualizarCosasReady();
break;
            case JUGAR: ActualizarCosasPlaying();
break;
            case NivelCOMPLE:
ActualizarCosasLevelCleared(); break;
            case PERDIO2: ActualizarCosasPerder();
break;
        }
    }

    public void Dibujar(Graphics2D g) {
        g.drawImage(image, 0, 0, null);
    }

    protected void ActualizarInitializing() {
    }

    protected void Actualizarpre() {
    }

    protected void ActualizarTitle() {
    }

    protected void ActualizarLevelSelect() {
    }

    protected void ActualizarReady() {
    }
```

```
protected void ActualizarPlaying() {
}

protected void ActualizarLevelCleared() {
}

protected void ActualizarPerder() {
}

protected void ActualizarCosasInitializing() {
}

protected void ActualizarCosaspre() {
}

protected void ActualizarCosasTitle() {
}

protected void ActualizarCosasLevelSelect() {
}

protected void ActualizarCosasReady() {
}

protected void ActualizarCosasPlaying() {
}

protected void ActualizarCosasLevelCleared() {
}

   protected void ActualizarCosasPerder() {
   }

   public void JuegoCuerdaChanged(JuegoIni newJuegoIni) {
   }

   protected void setCurrentWaitTime() {
      waitTime = System.currentTimeMillis();
   }

   protected boolean checkPassedTime(double time) {
      return (System.currentTimeMillis() - waitTime) * 0.001 >= time;
   }
}


 class FontRenderer {

   private static Font font;

   static {
      try {
         InputStream is = FontRenderer.class.getResourceAsStream("/Imagenes/GOODDP.TTF");
         font = Font.createFont(Font.TRUETYPE_FONT, is);
         font = font.deriveFont(40f);
      } catch (Exception ex) {
```

```
Logger.getLogger(FontRenderer.class.getName()).
log(Level.SEVERE, null, ex);
            System.exit(-1);
        }
    }

    public static void Dibujar(Graphics2D g, String
text, int x, int y) {
        g.setFont(font);

        AffineTransform tranform = new
AffineTransform();
        tranform.translate(x, y);
        TextLayout layout = new TextLayout(text,
font, g.getFontRenderContext());
        Shape outline = layout.getOutline(tranform);
        g.setColor(Color.BLACK);
        g.setStroke(new BasicStroke(4f));
        g.draw(outline);

        g.setColor(Color.WHITE);
        g.drawString(text, x, y);
    }

}

class Mouse extends MouseAdapter {

    public static double x;
    public static double y;
    public static boolean pressed;
    public static boolean pressedConsumed;

    @Override
    public void mouseMoved(MouseEvent e) {
        x = e.getX();
        y = e.getY();
    }

    @Override
    public void mouseDragged(MouseEvent e) {
        x = e.getX();
        y = e.getY();
    }

    @Override
    public void mousePressed(MouseEvent e) {
        pressed = true;
        pressedConsumed = false;
    }

    @Override
    public void mouseReleased(MouseEvent e) {
        pressed = false;
    }

}

class Time {

    private static double frameRate = 1 / 30.0;
```

```
private static int ActualizarCount;

private static double delta;
private static double previous = -1;
private static double current;
private static double unprocessed;

public static double getDelta() {
   return delta;
}

public static double getCurrent() {
   return current;
}

public static boolean needsUpdate() {
   if (ActualizarCount > 0) {
      ActualizarCount--;
      return true;
   }
   return false;
}

public static void Actualizar() {
   current = System.nanoTime() * 0.000000001;
   if (previous < 0) {
      previous = current;
   }
   delta = current - previous;
   previous = current;
   unprocessed += delta;
      while (unprocessed > frameRate) {
         unprocessed -= frameRate;
         ActualizarCount++;
      }
   }

}
```

```
class AirCushionTodo extends Todo implements
aire {

   private AirCushion airCushion;
   private Animacion animation;

   public AirCushionTodo(TodaAnimacion ani,
AirCushion airCushion) {
      super(ani);
      this.airCushion = airCushion;
      airCushion.addListener(this);
      animation = new Animacion();
      loadAllAnimacions();
   }

   private void loadAllAnimacions() {
      animation.addFrames("pump", "pump", 0, 5,
0, 0, false, 8);
      animation.selectAnimacion("pump");
      animation.setCurrentFrame(5);
   }

   @Override
```

```
    public void Actualizar() {
        animation.Actualizar();
    }

    @Override
    public void Dibujar(Graphics2D g) {
        AffineTransform originalTransform =
g.getTransform();
        g.translate(airCushion.getPosition().x,
airCushion.getPosition().y);
        g.rotate((1 + airCushion.getDirection()) *
Math.toRadians(90));
        g.translate(-48, -64);
        animation.Dibujar(g);
        g.setTransform(originalTransform);
    }

    public void
JuegoCuerdaChanged(TodaAnimacion.JuegoIni
newJuegoIni) {
        visible = newJuegoIni ==
TodaAnimacion.JuegoIni.JUGAR;
    }

    @Override
    public void AreRosaGuada() {
        animation.selectAnimacion("pump");
    }

}

class BackgroundTodo extends Todo {

    public BackgroundTodo(TodaAnimacion ani) {
        super(ani);
    }

    @Override
    public void Comenzar() {

loadImageFromResource("/Imagenes/atras.png")
;
    }

    @Override
    public void JuegoCuerdaChanged(JuegoIni
newJuegoIni) {
        visible = (newJuegoIni != JuegoIni.IniciarJ)
                && (newJuegoIni != JuegoIni.es)
                && (newJuegoIni != JuegoIni.titulo);
    }

}

class BubbleTodo extends Todo implements
Burbuja {

    private Bubble bubble;
    private AffineTransform transform = new
AffineTransform();
    private Animacion animation;
    private boolean burst;
    private Point burstPosition = new Point();

    public BubbleTodo(TodaAnimacion ani, Bubble
bubble) {
```

```
        super(ani);
        this.bubble = bubble;
        bubble.addListener(this);

loadImageFromResource("/Imagenes/bubble2.png");
        animation = new Animacion();
        loadAllAnimacions();
    }

    private void loadAllAnimacions() {
        animation.addFrames("flight", "bubble_flight", 0, 13, 0, 0, true, 20);
        animation.addFrames("burst", "bubble_pop", 0, 11, 0, 0, false, 40);
    }

    @Override
    public void ActualizarPlaying() {
        if (bubble.getDulceClas() != null || burst) {
            animation.Actualizar();
        }
        if (burst && animation.isFinished()) {
            burst = false;
        }
    }

    @Override
    public void Dibujar(Graphics2D g) {
        if (burst) {
            AffineTransform originalTransform = g.getTransform();
            g.translate(burstPosition.x, burstPosition.y);
            g.scale(1.2, 1.2);
            g.translate(-63, -58);
            animation.Dibujar(g);
            g.setTransform(originalTransform);
            return;
        }
        if (!bubble.isVisible()) {
            return;
        }
        if (bubble.getDulceClas() != null) {
            AffineTransform originalTransform = g.getTransform();
            g.translate(bubble.getPosition().x, bubble.getPosition().y);
            g.scale(1.3, 1.3);
            g.translate(-37, -36);
            animation.Dibujar(g);
            g.setTransform(originalTransform);
        }
        else {
            transform.setToIdentity();
            transform.translate(bubble.getPosition().x, bubble.getPosition().y);
            transform.translate(-image.getWidth() / 2, -image.getHeight() / 2);
            g.drawImage(image, transform, null);
        }
    }
```

```
    public void
JuegoCuerdaChanged(TodaAnimacion.JuegoIni
newJuegoIni) {
        visible = newJuegoIni ==
TodaAnimacion.JuegoIni.JUGAR;
    }

    @Override
    public void bur() {
        burst = true;

burstPosition.setLocation(bubble.getPosition().x,
bubble.getPosition().y);
        animation.selectAnimacion("burst");
    }

    @Override
    public void DulceRoto() {
        animation.selectAnimacion("flight");
    }

}

class DulceClasTodo extends Todo implements
Dulce {

    private DulceClas candy;
    private AffineTransform transform = new
AffineTransform();

    private BufferedImage candyLeft;
    private BufferedImage candyRight;
    private Particle candyLeftParticle;
    private Particle candyRightParticle;
    private boolean destroyed;
    private double angle;

    public DulceClasTodo(TodaAnimacion ani) {
        super(ani);
        this.candy =
ani.getEstructura().getDulceClas();
        candy.addListener(this);
        candyLeft =
loadImageFromResource("/Imagenes/candy_left.
png");
        candyRight =
loadImageFromResource("/Imagenes/candy_righ
t.png");

loadImageFromResource("/Imagenes/candy.png"
);
    }

    @Override
    public void ActualizarCosasPlaying() {
        if (destroyed) {
            angle += 0.2;
        }
    }

    @Override
    public void Dibujar(Graphics2D g) {
        if (candy.isVisible()) {
            transform.setToIdentity();
            transform.translate(candy.getPivot().x,
candy.getPivot().y);
            transform.rotate(candy.getAngle());
```

```
            transform.translate(-27, -27);
            g.drawImage(image, transform, null);
        }
        else if (destroyed) {
            transform.setToIdentity();

transform.translate(candyLeftParticle.position.x,
candyLeftParticle.position.y);
            transform.rotate(angle);
            transform.translate(-27, -27);
            g.drawImage(candyLeft, transform, null);


            transform.setToIdentity();

transform.translate(candyRightParticle.position.x,
candyRightParticle.position.y);
            transform.rotate(-angle);
            transform.translate(-27, -27);
            g.drawImage(candyRight, transform, null);
        }
    }


    public void
JuegoCuerdaChanged(TodaAnimacion.JuegoIni
newJuegoIni) {
        visible = newJuegoIni ==
TodaAnimacion.JuegoIni.JUGAR;
    }


    @Override
    public void DulceRotoD() {
        destroyed = true;
        World world =
ani.getEstructura().getWorld();
        double x = candy.getPivot().x;
        double y = candy.getPivot().y;
        candyLeftParticle = new Particle(world, x, y);
        candyRightParticle = new Particle(world, x,
y);
        world.addParticle(candyLeftParticle);
        world.addParticle(candyRightParticle);
        candyLeftParticle.addForce(new Vector(-10,
0));
        candyRightParticle.addForce(new Vector(10,
0));
    }


}



class CortinaTodo extends Todo {


    private BufferedImage top;
    private BufferedImage bottom;


    private double currentP = 1;
    private double targetP = 1;


    private double topY;
    private double bottomY;


    public CortinaTodo(TodaAnimacion ani) {
        super(ani);
    }


    @Override
```

```
    public void Comenzar() {
        top = loadImageFromResource("/Imagenes/cortinaArriba.png");
        bottom = loadImageFromResource("/Imagenes/cortinaAbajo.png");
    }

    @Override
    public void ActualizarCosas() {
        double dif = targetP - currentP;
        int s = dif > 0 ? 1 : -1;
        if (dif > 0.01 || dif < -0.01) {
            currentP = currentP + s * 0.04;
        }
        else {
            currentP = targetP;
        }
        topY = -Ventana2.AltoPantalla / 2 * currentP;
        bottomY = Ventana2.AltoPantalla / 2 + Ventana2.AltoPantalla / 2 * currentP;
        visible = currentP < 1;
    }

    @Override
    public void Dibujar(Graphics2D g) {
        g.drawImage(top, 0, (int) topY, null);
        g.drawImage(bottom, 0, (int) bottomY, null);
    }

    public void open() {
        targetP = 1;
    }

    public void close() {
        targetP = 0;
    }

    public void openInstantly() {
        targetP = 1;
        currentP = 1;
    }

    public void closeInstantly() {
        targetP = 0;
        currentP = 0;
    }

    public boolean isFinished() {
        return currentP == targetP;
    }

}

class Efecto1Todo extends Todo {

    private Color[] colorsBlack = new Color[256];
    private Color[] colorsWhite = new Color[256];
    private Color color;
    private Color targetColor = Color.WHITE;

    private double alpha = 0;
```

```
    private double targetAlpha = 0;

    public Efecto1Todo(TodaAnimacion ani) {
        super(ani);
    }

    public Color getTargetColor() {
        return targetColor;
    }

    public void setTargetColor(Color targetColor) {
        this.targetColor = targetColor;
    }

    public double getAlpha() {
        return alpha;
    }

    public void setAlpha(double alpha) {
        this.alpha = alpha;
    }

    public double getTargetAlpha() {
        return targetAlpha;
    }

    public void setTargetAlpha(double targetAlpha)
{
        this.targetAlpha = targetAlpha;
    }

    @Override
    public void Comenzar() {
        for (int c = 0; c < colorsWhite.length; c++) {
            colorsBlack[c] = new Color(0, 0, 0, c);
            colorsWhite[c] = new Color(255, 255, 255,
c);
        }
        color = colorsWhite[255];
    }

    @Override
    public void Actualizar() {
        double dif = targetAlpha - alpha;
        double s = dif > 0 ? 1 : -1;
        if (dif > 0.01 || dif < -0.01) {
            double delta = Time.getDelta();
            alpha = alpha + s * delta * 0.5;
            alpha = alpha > 1 ? 1 : alpha;
            alpha = alpha < 0 ? 0 : alpha;
        }
        else {
            alpha = targetAlpha;
        }

        if (targetColor == Color.WHITE) {
            color = colorsWhite[(int) (255 * alpha)];
        }
        else {
            color = colorsBlack[(int) (255 * alpha)];
        }
        visible = alpha > 0;
```

```
    }

    @Override
    public void Dibujar(Graphics2D g) {
        g.setColor(color);
        g.fillRect(0, 0, Ventana2.AnchoPantalla,
Ventana2.AltoPantalla);
    }

    public void fadeIn() {
        targetAlpha = 0;
    }

    public void fadeOut() {
        targetAlpha = 1;
    }

    public boolean fadeEffectFinished() {
        return alpha == targetAlpha;
    }

}


class PerderTodo extends Todo {

    private Efecto1Todo fadeEffect;
    private CortinaTodo curtain;

    private BufferedImage Fallado;
    private boolean FalladoNivel;
    private Botoncito BotonOtraV;
    private Botoncito BotonRein;
    private Botones BotonOtraVListener;
    private Botones BotonReinListener;
    private boolean BotonOtraVPressed;
    private boolean BotonReinPressed;

    public PerderTodo(TodaAnimacion ani,
Efecto1Todo fadeEffect, CortinaTodo curtain) {
        super(ani);

        Fallado = new BufferedImage(200, 63,
BufferedImage.TYPE_INT_ARGB);
        Graphics2D g = (Graphics2D)
Fallado.getGraphics();

g.setRenderingHint(RenderingHints.KEY_ANTIALI
ASING, RenderingHints.VALUE_ANTIALIAS_ON);
        FontRenderer.Dibujar(g, "UPS PERDISTE", 5,
42);

        this.fadeEffect = fadeEffect;
        this.curtain = curtain;
        BotonOtraV = new Botoncito(ani, "Otra vez",
50, 42, 235, 330);
        BotonRein = new Botoncito(ani, "Reiniciar",
50, 42, 415, 330);
        BotonOtraVListener = new Botones() {
            @Override
            public void PresionarCl() {
                BotonOtraVPressed = true;
            }
        };
```

```
BotonOtraV.setListener(BotonOtraVListener);
        BotonReinListener = new Botones() {
            @Override
            public void PresionarCl() {
                BotonReinPressed = true;
            }
        };
        BotonRein.setListener(BotonReinListener);
    }

    @Override
    protected void ActualizarPerder() {
    }

    @Override
    protected void ActualizarCosasPerder() {
        switch (InsPunt) {
            case 0:
                setCurrentWaitTime();
                InsPunt = 1;
            case 1:
                if (!checkPassedTime(1)) {
                    return;
                }
                BotonOtraV.setVisible(true);
                BotonRein.setVisible(true);
                FalladoNivel = true;

                InsPunt = 2;
            case 2:
                BotonOtraV.Actualizar();
                BotonRein.Actualizar();
                if (BotonOtraVPressed) {
                    ani.ReiniciarNivel();
                }
                else if (BotonReinPressed) {
                    InsPunt = 3;
                }
                return;
            case 3:
                fadeEffect.setTargetColor(Color.BLACK);
                fadeEffect.fadeOut();
                InsPunt = 4;
            case 4:
                if (!fadeEffect.fadeEffectFinished()) {
                    return;
                }
                curtain.openInstantly();
                ani.backToTitle();
        }
    }

    @Override
    public void Dibujar(Graphics2D g) {
        if (!FalladoNivel) {
            return;
        }

        g.drawImage(Fallado, 320, 180, null);

        if (BotonOtraV.isVisible()) {
```

```
            BotonOtraV.Dibujar(g);
        }
        if (BotonRein.isVisible()) {
            BotonRein.Dibujar(g);
        }
    }

    @Override
    public void JuegoCuerdaChanged(JuegoIni newJuegoIni) {
        visible = false;
        if (newJuegoIni == PERDIO2) {
            visible = true;
            InsPunt = 0;
            FalladoNivel = false;
            BotonOtraVPressed = false;
            BotonReinPressed = false;
            BotonOtraV.reset();
            BotonRein.reset();
            BotonOtraV.setVisible(false);
            BotonRein.setVisible(false);
        }
    }

}

class IniciaTodo extends Todo {
    private Efecto1Todo fadeEffect;
    public IniciaTodo(TodaAnimacion ani, Efecto1Todo fadeEffect) {
        super(ani);
        this.fadeEffect = fadeEffect;

    }

    @Override
    protected void ActualizarCosasInitializing() {
        switch (InsPunt) {
            case 0:

                InsPunt = 1;
            case 1:
                fadeEffect.setTargetColor(Color.WHITE);
                InsPunt = 2;
            case 2:
                ani.setState(es);
        }
    }

}

class LevelClearedTodo extends Todo {

    private Efecto1Todo fadeEffect;
    private CortinaTodo curtain;

    private BufferedImage NivelCompleto;

    private BufferedImage EstrellaOn;
    private BufferedImage EstrellitaOff;
    private boolean showStars;
```

```
    private Botoncito BotonOtraV;
    private Botoncito BotonRein;
    private Botoncito BotonSig;
    private Botones BotonOtraVListener;
    private Botones BotonReinListener;
    private Botones BotonSigListener;
    private boolean BotonOtraVPressed;
    private boolean BotonReinPressed;
    private boolean BotonSigPressed;

    public LevelClearedTodo(TodaAnimacion ani, Efecto1Todo fadeEffect, CortinaTodo curtain) {
        super(ani);
        NivelCompleto = new BufferedImage(200, 63, BufferedImage.TYPE_INT_ARGB);
        Graphics2D g = (Graphics2D) NivelCompleto.getGraphics();

g.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);
        FontRenderer.Dibujar(g, "  YAAS GANASTE ", 5, 42);

        EstrellaOn = loadImageFromResource("/Imagenes/star_result_0g.png");
        EstrellitaOff = loadImageFromResource("/Imagenes/star_result_1g.png");
        this.fadeEffect = fadeEffect;
        this.curtain = curtain;
        BotonOtraV = new Botoncito(ani, "Otra vez", 50, 42, 135, 330);
        BotonRein = new Botoncito(ani, "Reiniciar", 50, 42, 315, 330);
        BotonSig = new Botoncito(ani, "Siguiente", 55, 42, 495, 330);
        BotonOtraVListener = new Botones() {
            @Override
            public void PresionarCl() {
                BotonOtraVPressed = true;
            }
        };

BotonOtraV.setListener(BotonOtraVListener);
        BotonReinListener = new Botones() {
            @Override
            public void PresionarCl() {
                BotonReinPressed = true;
            }
        };
        BotonRein.setListener(BotonReinListener);
        BotonSigListener = new Botones() {
            @Override
            public void PresionarCl() {
                BotonSigPressed = true;
            }
        };
        BotonSig.setListener(BotonSigListener);
    }

    @Override
    protected void ActualizarLevelCleared() {
    }

    @Override
```

```
protected void ActualizarCosasLevelCleared() {
    switch (InsPunt) {
        case 0:
            setCurrentWaitTime();
            InsPunt = 1;
        case 1:
            if (!checkPassedTime(1)) {
                return;
            }
            BotonOtraV.setVisible(true);
            BotonRein.setVisible(true);
            BotonSig.setVisible(true);
            showStars = true;

            InsPunt = 2;
        case 2:
            BotonOtraV.Actualizar();
            BotonRein.Actualizar();
            BotonSig.Actualizar();
            if (BotonOtraVPressed) {
                ani.ReiniciarNivel();
            }
            else if (BotonReinPressed) {
                InsPunt = 3;
            }
            else if (BotonSigPressed) {
                ani.nextLevel();
            }
            return;
        case 3:
            fadeEffect.setTargetColor(Color.BLACK);
            InsPunt = 4;
        case 4:
            if (!fadeEffect.fadeEffectFinished()) {
                return;
            }
            curtain.openInstantly();
            ani.backToTitle();
    }
}

private int countObtainedStars() {
    int count = 0;
    for (Star star : ani.getEstructura().getStars()) {
        if (!star.isVisible()) {
            count++;
        }
    }
    return count;
}

@Override
public void Dibujar(Graphics2D g) {
    if (!showStars) {
        return;
    }

    g.drawImage(NivelCompleto, 320, 100, null);

    int starsCount = countObtainedStars();
    g.drawImage(starsCount > 0 ? EstrellaOn :
EstrellitaOff, 210, 170, null);
```

```
        g.drawImage(starsCount > 1 ? EstrellaOn : EstrellitaOff, 335, 170, null);
        g.drawImage(starsCount > 2 ? EstrellaOn : EstrellitaOff, 460, 170, null);

        if (BotonOtraV.isVisible()) {
            BotonOtraV.Dibujar(g);
        }
        if (BotonRein.isVisible()) {
            BotonRein.Dibujar(g);
        }
        if (BotonSig.isVisible()) {
            BotonSig.Dibujar(g);
        }
    }

    @Override
    public void JuegoCuerdaChanged(JuegoIni newJuegoIni) {
        visible = false;
        if (newJuegoIni == NivelCOMPLE) {
            visible = true;
            InsPunt = 0;
            showStars = false;
            BotonOtraVPressed = false;
            BotonReinPressed = false;
            BotonSigPressed = false;
            BotonOtraV.reset();
            BotonRein.reset();
            BotonSig.reset();
            BotonOtraV.setVisible(false);
            BotonRein.setVisible(false);
            BotonSig.setVisible(false);
        }
    }

}

class preTodo extends Todo {

    private Efecto1Todo fadeEffect;

    private SlashTrail trail = new SlashTrail(25, 0.4);
    private List<Point> points = new ArrayList<Point>();
    private int handWriteEffectIndex;
    private boolean handWriteEffectFinished;

    private BufferedImage cover;
    private double coverX = -50;

    public preTodo(TodaAnimacion ani, Efecto1Todo fadeEffect) {
        super(ani);
        this.fadeEffect = fadeEffect;
        trail.setColor(new Color(230, 230, 230));
    }

    @Override
    protected void ActualizarCosaspre() {
        switch (InsPunt) {
            case 0:
```

```
            InsPunt = 1;
        case 1:
            InsPunt = 2;
        case 2:
            handWriteEffectIndex = 0;
            handWriteEffectFinished = false;
            InsPunt = 3;
        case 3:
            ActualizarHandWriteTrail();
            if (handWriteEffectFinished) {
                InsPunt = 4;
            }
            return;
        case 4:
            fadeEffect.setTargetColor(Color.BLACK);
            InsPunt = 5;
        case 5:
            ani.setState(titulo);
    }
  }

  private void ActualizarHandWriteTrail() {
    int r = 3;
    if (handWriteEffectIndex < points.size() - (r -
1))  {
        for (int k = 0; k < r; k++) {
            Point p =
points.get(handWriteEffectIndex++);
            trail.addTrail(p.x, p.y);
        }
    }
    else {
        trail.addTrail(-1, -1);
        handWriteEffectFinished = true;
    }

    coverX += 2.65;
  }

  @Override
  public void Dibujar(Graphics2D g) {
    g.drawImage(image, 0, 0, null);
    g.drawImage(cover, (int) coverX, 286, null);
    trail.drawDebug(g);
  }

  @Override
  public void JuegoCuerdaChanged(JuegoIni
newJuegoIni) {
    visible = false;
    if (newJuegoIni == es) {
        visible = true;
        InsPunt = 0;
    }
  }

}

class PetTodo extends Todo implements
Animalito, escuchar {

  private CortinaTodo curtain;
```

```
    private Pet pet;
    private Animacion animation;
    private boolean NivelCompleto;
    private boolean JuegoPer;

    public PetTodo(TodaAnimacion ani,
CortinaTodo curtain) {
        super(ani);
        this.pet = ani.getEstructura().getPet();
        this.pet.addListener(this);
        this.animation = new Animacion();
        this.curtain = curtain;

loadImageFromResource("/Imagenes/support.pn
g");
        ani.getEstructura().addListener(this);
        loadAllAnimacions();
    }

    private void loadAllAnimacions() {
        int x = (int) this.pet.getPosition().x - 40;
        int y = (int) this.pet.getPosition().y - 50;
        animation.addFrames("normal",
"pet_normal", 0, 18, x, y, true);
        animation.addFrames("openMouth",
"pet_openMouth", 0, 3, x, y, false);
        animation.addFrames("closeMouth",
"pet_closeMouth", 0, 3, x, y, false);
        animation.addFrames("chew", "pet_chew",
0, 33, x - 10, y, false);
        animation.addFrames("sad", "pet_sad", 0,
13, x - 10, y, false);
        animation.selectAnimacion("normal");
    }

    @Override
    public void ActualizarCosasPlaying() {
        ActualizarAnimacion();
        switch (InsPunt) {
            case 0:
                setCurrentWaitTime();
                InsPunt = 1;
            case 1:
                if (!checkPassedTime(0.5)) {
                    return;
                }
                curtain.open();
                InsPunt = 2;
            case 2:
                if (!NivelCompleto && !JuegoPer) {
                    return;
                }
                setCurrentWaitTime();
                InsPunt = 3;
            case 3:
                if (!checkPassedTime(JuegoPer ? 2.0 :
1.0)) {
                    return;
                }
                curtain.close();
                InsPunt = 4;
            case 4:
                if (!curtain.isFinished()) {
                    return;
```

```
            }
            if (NivelCompleto) {
               ani.setState(NivelCOMPLE);
            }
            else if (JuegoPer) {
               ani.setState(PERDIO2);
            }
      }
   }

   private void ActualizarAnimacion() {
      animation.Actualizar();
      if
(animation.getCurrentAnimacionName().equals("
closeMouth") && animation.isFinished()) {
         animation.selectAnimacion("normal");
      }
   }

   @Override
   public void Dibujar(Graphics2D g) {
      int x = (int) (pet.getPosition().x -
pet.getRadius());
      int y = (int) (pet.getPosition().y -
pet.getRadius());
      g.drawImage(image, x - 37, y + 2, null);
      animation.Dibujar(g);
   }

   @Override
   public void DulceComido() {
      animation.selectAnimacion("chew");
   }

   @Override
   public void DulceNo() {
      animation.selectAnimacion("openMouth");
   }

   @Override
   public void DulceFuera() {
      animation.selectAnimacion("closeMouth");
   }

   @Override
   public void Fallado() {
      animation.selectAnimacion("sad");
      JuegoPer = true;
   }

   @Override
   public void NivelCopleLis() {
      NivelCompleto = true;
   }

   public void
JuegoCuerdaChanged(TodaAnimacion.JuegoIni
newJuegoIni) {
      visible = false;
      if (newJuegoIni ==
TodaAnimacion.JuegoIni.JUGAR) {
         visible = true;
         InsPunt = 0;
         NivelCompleto = false;
```

```
        JuegoPer = false;
      }
   }

}

class PinRopeTodo extends Todo implements romper {

   private static final Stroke ROPE_STROKE = new BasicStroke(2);
   private static final Stroke OUTLINE_STROKE = new BasicStroke(5);
   private static final Stroke DASHED_STROKE = new BasicStroke(3.0f, BasicStroke.CAP_BUTT, BasicStroke.JOIN_MITER, 10.0f, new float[] { 10.0f }, 0.0f);

   private PinRope pinRope;
   private Rope rope;

   public PinRopeTodo(TodaAnimacion ani, PinRope pinRope) {
      super(ani);
      this.pinRope = pinRope;
      this.pinRope.addListener(this);
      loadImageFromResource("/Imagenes/pin2.png");
   }

   @Override
   public void Actualizar() {
   }

   private static Color[] OUTLINE_COLOR = { Color.BLACK } ;
   private static Color[] ROPE_COLOR = { new Color(250, 180, 180), new Color(200, 110, 110) };

   @Override
   public void Dibujar(Graphics2D g) {
      Stroke originalStroke = g.getStroke();
      int x = (int) (pinRope.getPosition().x - image.getWidth() / 2);
      int y = (int) (pinRope.getPosition().y - image.getHeight() / 2);

      g.setColor(Color.BLACK);
      g.setStroke(DASHED_STROKE);
      double radius = pinRope.getRadius();
      g.drawOval((int) (pinRope.getPosition().x - radius)
            , (int) (pinRope.getPosition().y - radius)
            , (int) (2 * radius), (int) (2 * radius));

      if (rope != null) {
         g.setStroke(OUTLINE_STROKE);
         drawRope(g, OUTLINE_COLOR);

         g.setStroke(ROPE_STROKE);
         drawRope(g, ROPE_COLOR);
      }

      g.drawImage(image, x, y, null);
      g.setStroke(originalStroke);
   }
```

```
    private void drawRope(Graphics2D g, Color[] colors) {
        int colorIndex = 0;
        for (Stick stick : rope.getSticks()) {
            if (stick != null && stick.isVisible()) {
                g.setColor(colors[colorIndex]);
                colorIndex = (colorIndex + 1) % colors.length;
                stick.getLine().drawDebug(g);
            }
        }
    }

    @Override
    public void CuerdaRotaL(Rope rope) {
        //System.out.println("cuerda creada !");
        this.rope = rope;
    }

    public void JuegoCuerdaChanged(TodaAnimacion.JuegoIni newJuegoIni) {
        visible = newJuegoIni == TodaAnimacion.JuegoIni.JUGAR;
    }

}

class RopeTodo extends Todo {

    private static final Stroke ROPE_STROKE = new BasicStroke(2);
    private static final Stroke OUTLINE_STROKE = new BasicStroke(5);

    private Rope rope;

    public RopeTodo(TodaAnimacion ani, Rope rope) {
        super(ani);
        this.rope = rope;
        loadImageFromResource("/Imagenes/pin2.png");
    }

    @Override
    public void Actualizar() {
    }

    private static Color[] OUTLINE_COLOR = { Color.BLACK } ;
    private static Color[] ROPE_COLOR = { new Color(250, 180, 180), new Color(200, 110, 110) };

    @Override
    public void Dibujar(Graphics2D g) {
        Stroke originalStroke = g.getStroke();

        g.setStroke(OUTLINE_STROKE);
        drawRope(g, OUTLINE_COLOR);

        g.setStroke(ROPE_STROKE);
        drawRope(g, ROPE_COLOR);

        g.setStroke(originalStroke);
```

```
        int x = (int) (rope.getFirstParticle().position.x
- image.getWidth() / 2);
        int y = (int) (rope.getFirstParticle().position.y
- image.getHeight() / 2);
        g.drawImage(image, x, y, null);
    }

    private void drawRope(Graphics2D g, Color[]
colors) {
        int colorIndex = 0;
        for (Stick stick : rope.getSticks()) {
            if (stick != null && stick.isVisible()) {
                g.setColor(colors[colorIndex]);
                colorIndex = (colorIndex + 1) %
colors.length;
                stick.getLine().drawDebug(g);
            }
        }
    }

    public void
JuegoCuerdaChanged(TodaAnimacion.JuegoIni
newJuegoIni) {
        visible = newJuegoIni ==
TodaAnimacion.JuegoIni.JUGAR;
    }

}

class SpikesTodo extends Todo {

    private Spikes spikes;

    public SpikesTodo(TodaAnimacion ani, Spikes
spikes) {
        super(ani);
        this.spikes = spikes;

        int sw = spikes.getRectangle().width;
        int n = 0;
        switch (sw) {
            case 95: n = 1; break;
            case 170: n = 2; break;
            case 255: n = 3; break;
            case 325: n = 4; break;
            default : throw new
RuntimeException("spikes width " + sw + " not
valid !");
        }

loadImageFromResource("/Imagenes/spikes_" +
n + ".png");
    }

    @Override
    public void Actualizar() {
    }

    @Override
    public void Dibujar(Graphics2D g) {
        int x = (int) (spikes.getRectangle().x - 20);
        int y = (int) (spikes.getRectangle().y - 10);
        g.drawImage(image, x, y, null);
    }
```

```
    public void
JuegoCuerdaChanged(TodaAnimacion.JuegoIni
newJuegoIni) {
        visible = newJuegoIni ==
TodaAnimacion.JuegoIni.JUGAR;
    }

}

class StarTodo extends Todo implements
Estrellita {

    private Star star;
    private AffineTransform transform = new
AffineTransform();

    private BufferedImage bloom;
    private double bloomStartValue;
    private double bloomScale;

    private Animacion animation;

    public StarTodo(TodaAnimacion ani, Star star) {
        super(ani);
        this.star = star;
        star.addListener(this);
        bloom =
loadImageFromResource("/Imagenes/star_bloom
.png");
        bloomStartValue = 999 * Math.random();

loadImageFromResource("/Imagenes/star2.png")
;
        animation = new Animacion();
        loadAllAnimacions();
    }

    private void loadAllAnimacions() {
        int x = (int) star.getPosition().x - 15;
        int y = (int) star.getPosition().y - 15;
        animation.addFrames("idle", "star_idle", 0,
17, x, y, true, 0.01);

        x = (int) star.getPosition().x - 71;
        y = (int) star.getPosition().y - 75;
        animation.addFrames("disappear",
"star_disappear", 0, 10, x, y, false, 20);

        animation.selectAnimacion("idle");
    }

    @Override
    public void Actualizar() {
        animation.Actualizar();
        bloomScale = 1.4 + 0.2 *
Math.sin(bloomStartValue
                + System.nanoTime() * 0.000000005);
    }

    @Override
    public void Dibujar(Graphics2D g) {
        if (!star.isVisible()) {
            animation.Dibujar(g);
            return;
        }
```

```
        transform.setToIdentity();
        transform.translate(star.getPosition().x,
star.getPosition().y);
        transform.scale(bloomScale, bloomScale);
        transform.translate(-bloom.getWidth() / 2, -
bloom.getHeight() / 2);
        g.drawImage(bloom, transform, null);
        animation.Dibujar(g);
    }

    public void
JuegoCuerdaChanged(TodaAnimacion.JuegoIni
newJuegoIni) {
        visible = newJuegoIni ==
TodaAnimacion.JuegoIni.JUGAR;
    }

    @Override
    public void MuereEstrella() {
        animation.selectAnimacion("disappear");
    }

}

class TitleTodo extends Todo {

    private Efecto1Todo fadeEffect;
    private CortinaTodo curtain;

    private BufferedImage title;
    private BufferedImage titleShadow;
    private AffineTransform titleShadowTransform
= new AffineTransform();
    // private double titleShadowAngle =
Math.toRadians(45);

    private Botoncito button;
    private Botones buttonListener;
    private boolean gameStarted;

    public TitleTodo(TodaAnimacion ani,
Efecto1Todo fadeEffect, CortinaTodo curtain) {
        super(ani);
        titleShadow =
loadImageFromResource("/Imagenes/Universida
d.png");

loadImageFromResource("/Imagenes/Universida
d.png");
        Graphics2D g = (Graphics2D)
image.getGraphics();

g.setRenderingHint(RenderingHints.KEY_ANTIALI
ASING, RenderingHints.VALUE_ANTIALIAS_ON);

        this.fadeEffect = fadeEffect;
        this.curtain = curtain;

        button = new Botoncito(ani, "JUGAR", 60, 42,
315, 370);
        buttonListener = new Botones() {
            @Override
            public void PresionarCl() {
                gameStarted = true;
```

```
        }
    };
}

@Override
protected void ActualizarTitle() {
    button.Actualizar();
}

@Override
protected void ActualizarCosasTitle() {

    switch (InsPunt) {
        case 0:
            setCurrentWaitTime();
            InsPunt = 1;
        case 1:
            if (!checkPassedTime(1)) {
                return;
            }
            fadeEffect.fadeIn();
            InsPunt = 2;
        case 2:
            button.setListener(buttonListener);
            InsPunt = 3;
        case 3:
            if (gameStarted) {
                curtain.close();
                InsPunt = 4;
            }
            return;
        case 4:
            if (!curtain.isFinished()) {
                return;
            }
            setCurrentWaitTime();
            InsPunt = 5;
        case 5:
            if (!checkPassedTime(0.1)) {
                return;
            }
            ani.ComenzarNivel(1);
    }
}

    @Override
    public void Dibujar(Graphics2D g) {
        g.drawImage(image, 0, 0, null);

        titleShadowTransform.setToIdentity();

titleShadowTransform.translate(Ventana2.Ancho
Pantalla / 2, Ventana2.AltoPantalla / 2);
        titleShadowTransform.translate(-
titleShadow.getWidth() / 2, -
titleShadow.getHeight() / 2);
        g.drawImage(titleShadow,
titleShadowTransform, null);

        g.drawImage(title, 180, 150, null);

        if (button.isVisible()) {
            button.Dibujar(g);
        }
```

```
    }

    @Override
    public void JuegoCuerdaChanged(JuegoIni newJuegoIni) {
        visible = false;
        if (newJuegoIni == titulo) {
            visible = true;
            InsPunt = 0;
            button.setListener(null);
            button.reset();
            gameStarted = false;
        }
    }

}

class Line {

    private Vector a;
    private Vector b;

    private Vector normal = new Vector();
    private Vector v = new Vector();

    public Line(Vector a, Vector b) {
        this.a = a;
        this.b = b;
    }

    public Line(double x1, double y1, double x2, double y2) {
        a = new Vector(x1, y1);
        b = new Vector(x2, y2);
    }

    public Vector getA() {
        return a;
    }

    public void setA(Vector a) {
        this.a = a;
    }

    public Vector getB() {
        return b;
    }

    public void setB(Vector b) {
        this.b = b;
    }

    public Vector getNormal() {
        getV();
        normal.set(-v.y, v.x);
        normal.normalize();
        return normal;
    }

    public Vector getV() {
        v.set(b);
```

```
        v.sub(a);
        return v;
    }

    public void drawDebug(Graphics2D g) {
        g.drawLine((int) a.x, (int) a.y, (int) b.x, (int)
b.y);
    }

    private final Vector vTmp = new Vector();

    public Vector getSegIntersectionPoint(Line l2) {
        vTmp.set(l2.a);
        vTmp.sub(a);
        double s1 = getNormal().dot(vTmp);
        vTmp.set(l2.b);
        vTmp.sub(a);
        double s2 = getNormal().dot(vTmp);

        vTmp.set(a);
        vTmp.sub(l2.a);
        double s3 = l2.getNormal().dot(vTmp);
        vTmp.set(b);
        vTmp.sub(l2.a);
        double s4 = l2.getNormal().dot(vTmp);

        if (s1 * s2 > 0 || s3 * s4 > 0) {
            return null;
        }
        return getIntersectionPoint(l2);
    }

    public Vector getIntersectionPoint(Line l2) {
        vTmp.set(l2.b);
        vTmp.sub(a);
        double d1 = l2.getNormal().dot(vTmp);
        vTmp.set(getV());
        double d2 = l2.getNormal().dot(vTmp);
        if (d1 == 0) {
            return null;
        }
        vTmp.scale(d1 / d2);
        vTmp.add(a);
        return vTmp;
    }

    private final Vector p1cl = new Vector();
    private final Vector p2cl = new Vector();
    private final Vector perp = new Vector();

    public boolean intersectsWithCircle(Vector
circlePivot, double circleRadius) {
        p1cl.set(a);
        p1cl.sub(circlePivot);
        p2cl.set(b);
        p2cl.sub(circlePivot);
        getV().setPerp(perp);
        if (perp.getSign(p1cl) * perp.getSign(p2cl) <
0) {
            v.normalize();
            return (Math.abs(p1cl.perpDot(v)) <=
circleRadius);
        } else {
```

```
        return (p1cl.getSize() <= circleRadius) ||
(p2cl.getSize() <= circleRadius);
      }
   }

}

 class Particle {

   public final World world;
   public final Vector position = new Vector();
   public final Vector previousPosition = new
Vector();
   public final Vector velocity = new Vector();
   public final Vector force = new Vector();

   public double restitution = 0.9;
   public double friction = 1.0;

   public boolean pinned = false;

   public Particle(World world, double x, double
y) {
      this(world, x, y, x, y);
   }

   public Particle(World view, double x, double y,
double prevX, double prevY) {
      this.world = view;
      position.set(x, y);
      previousPosition.set(prevX, prevY);
   }

public World getWorld() {
   return world;
}

public Vector getVelocity() {
   return velocity;
}

public double getRestitution() {
   return restitution;
}

public double getFriction() {
   return friction;
}

public boolean isPinned() {
   return pinned;
}

public void setPinned(boolean pinned) {
   this.pinned = pinned;
}

public void addForce(Vector a) {
   force.add(a);
}

public void Actualizar() {
   if (pinned) {
```

```
            return;
        }
        velocity.set(position);
        velocity.sub(previousPosition);
        velocity.scale(friction);
        previousPosition.set(position);
        position.add(velocity);
        position.add(force);
        force.set(world.getGravity());
    }

    public void ActualizarConstrain() {
        if (pinned) {
            return;
        }
        velocity.set(position);
        velocity.sub(previousPosition);
        velocity.scale(friction);

        if (position.x > world.getWidth()) {
            position.x = world.getWidth();
            previousPosition.x = position.x + velocity.x
* restitution;
        }
        else if (position.x < 0) {
            position.x = 0;
            previousPosition.x = position.x + velocity.x
* restitution;
        }
    }

    public void drawDebug(Graphics2D g) {
        g.setColor(Color.RED);
        g.fillOval((int) (position.x - 3), (int) (position.y
- 3), 6, 6);
    }


}

class Stick {


    private Particle a;
    private Particle b;
    private double size;
    private double elasticity;
    private boolean visible;
    private final Line line;
    private final Vector vTmp = new Vector();

    public Stick(Particle a, Particle b, double
elasticity, boolean visible) {
        this.a = a;
        this.b = b;
        this.elasticity = 1 - elasticity;
        this.visible = visible;

        vTmp.set(b.position);
        vTmp.sub(a.position);
        this.size = vTmp.getSize();

        this.line = new Line(a.position, b.position);
    }
```

```
public Particle getA() {
    return a;
}

public void setA(Particle a) {
    this.a = a;
    line.setA(a.position);
}

public Particle getB() {
    return b;
}

public void setB(Particle b) {
    this.b = b;
    line.setB(b.position);
}

public double getSize() {
    return size;
}

public void setSize(double size) {
    this.size = size;
}

public boolean isVisible() {
    return visible;
}

public void setVisible(boolean visible) {
    this.visible = visible;
}

public Line getLine() {
    return line;
}

public void Actualizar() {
    vTmp.set(b.position);
    vTmp.sub(a.position);
    double currentSize = vTmp.getSize();
    double dif = (currentSize - size) * 0.5;
    vTmp.normalize();
    vTmp.scale(dif * elasticity);
    if (!a.isPinned()) {
        a.position.add(vTmp);
    }
    if (!b.isPinned()) {
        b.position.sub(vTmp);
    }
}

public void drawDebug(Graphics2D g) {
    g.setColor(Color.BLUE);
    line.drawDebug(g);
}

}

class Vector {
```

```
public double x;
public double y;

public Vector() {
}

public Vector(double x, double y) {
   this.x = x;
   this.y = y;
}

public Vector(Vector v) {
   this.x = v.x;
   this.y = v.y;
}

public void set(double x, double y) {
   this.x = x;
   this.y = y;
}

public void set(Vector v) {
   this.x = v.x;
   this.y = v.y;
}

public void add(Vector v) {
   this.x += v.x;
   this.y += v.y;
}

public void sub(Vector v) {
   this.x -= v.x;
   this.y -= v.y;
}

public void scale(double s) {
   scale(s, s);
}

public void scale(double sx, double sy) {
   this.x *= sx;
   this.y *= sy;
}

public double getSize() {
   return Math.sqrt(x * x + y * y);
}

public void normalize() {
   double sizeInv = 1 / getSize();
   scale(sizeInv, sizeInv);
}

public int getSign(Vector v) {
   return (y * v.x > x * v.y) ? -1 : 1;
}

public double dot(Vector v) {
   return x * v.x + y * v.y;
}
```

```
    public double cross(Vector v) {
        return x * v.y - y * v.x;
    }

    public void setPerp(Vector v) {
        v.set(-y, x);
    }

    private static final Vector perpTmp = new
Vector();
    public double perpDot(Vector v) {
        setPerp(perpTmp);
        return perpTmp.dot(v);
    }

    @Override
    public String toString() {
        return "Vector{" + "x=" + x + ", y=" + y + '}';
    }

}

class World {

    private final int width;
    private final int height;
    private final Vector gravity = new Vector(0,
0.5);
    private final List<Particle> particles = new
ArrayList<Particle>();
    private final List<Stick> sticks = new
ArrayList<Stick>();

    public World(int width, int height) {
        this.width = width;
        this.height = height;
    }

    public int getWidth() {
        return width;
    }

    public int getHeight() {
        return height;
    }

    public Vector getGravity() {
        return gravity;
    }

    public List<Particle> getParticles() {
        return particles;
    }

    public List<Stick> getSticks() {
        return sticks;
    }

    public void Actualizar() {
        for (Particle particle : particles) {
            particle.Actualizar();
```

```
    }
    for (int i = 0; i < 5; i++) {
        for (Stick stick : sticks) {
            stick.Actualizar();
        }
        for (Particle particle : particles) {
            particle.ActualizarConstrain();
        }
    }
}

public void clear() {
    particles.clear();
    sticks.clear();
}

public void addParticle(Particle point) {
    particles.add(point);
}

public void addStick(Stick stick) {
    sticks.add(stick);
}

public void drawDebug(Graphics2D g) {
    for (Particle particle : particles) {
        particle.drawDebug(g);
    }
    for (Stick stick : sticks) {
        if (stick.isVisible()) {
            stick.drawDebug(g);
        }
    }
  }
}

class Botoncito extends Todo {

    private BufferedImage imageOn;
    private BufferedImage imageOff;
    private BufferedImage imageOver;
    private boolean pressed;
    private boolean over;

    private final Vector position = new Vector();
    private final Rectangle rectangle = new
Rectangle();
    private Botones listener;

    public Botoncito(TodaAnimacion ani, String
text, int textX, int textY, int buttonX, int buttonY)
{
        super(ani);
        imageOn =
loadImageFromResource("/Imagenes/button_on.
png");
        imageOff =
loadImageFromResource("/Imagenes/button_off.
png");
        imageOver =
loadImageFromResource("/Imagenes/button_ove
r.png");
```

```
        renderText((Graphics2D)
imageOn.getGraphics(), text, textX, textY);
        renderText((Graphics2D)
imageOff.getGraphics(), text, textX, textY);
        renderText((Graphics2D)
imageOver.getGraphics(), text, textX, textY);

        position.set(buttonX, buttonY);
        rectangle.setBounds(buttonX, buttonY,
image.getWidth(), image.getHeight());

        visible = true;
    }

    private void renderText(Graphics2D g, String
text, int textX, int textY) {

g.setRenderingHint(RenderingHints.KEY_ANTIALI
ASING, RenderingHints.VALUE_ANTIALIAS_ON);
        FontRenderer.Dibujar(g, text, textX, textY);
    }

    public Botones getListener() {
        return listener;
    }

    public void setListener(Botones listener) {
        this.listener = listener;
    }

    public void reset() {
        pressed = false;
    }

    @Override
    public void Actualizar() {
        if (!visible) {
            over = false;
            pressed = false;
            return;
        }

        over = rectangle.contains(Mouse.x,
Mouse.y);
        if (over && !Mouse.pressedConsumed &&
Mouse.pressed) {
            Mouse.pressedConsumed = true;
            pressed = true;
            if (listener != null) {
                listener.PresionarCl();
            }
        }
    }

    @Override
    public void Dibujar(Graphics2D g) {
        if (pressed) {
            g.drawImage(imageOn, (int) position.x,
(int) position.y, null);
        }
        else if (over) {
            g.drawImage(imageOver, (int) position.x,
(int) position.y, null);
        }
        else {
```

```
        g.drawImage(imageOff, (int) position.x,
(int) position.y, null);
      }
   }

}



//PARA CARGAR NIVEL

class AirCushion {

   private final Estructura EstructuraPe;
   private final Vector position = new Vector();
   private final double radius = 30;
   private final int direction; // 0=right, 1=down,
2=left, 3=up
   private final Polygon influenceArea = new
Polygon();
   private final Line line = new Line(0, 0, 0, 0);
   private final Vector wind = new Vector();
   private long firedStartTime;
   private final Vector vTmp = new Vector();
   private final List<aire> listeners = new
ArrayList<aire>();

   public AirCushion(Estructura EstructuraPe, int
x, int y, int direction) {
      this.EstructuraPe = EstructuraPe;
      this.position.set(x, y);
      this.direction = direction;
      createWind();
      createInfluenceArea();
   }

   private void createWind() {
      double wx = 0.75 *
Math.cos(Math.toRadians(90 * direction));
      double wy = 0.75 *
Math.sin(Math.toRadians(90 * direction));
      this.wind.set(wx, wy);
   }

   private void createInfluenceArea() {
      influenceArea.addPoint((int) radius, (int)
radius);
      influenceArea.addPoint((int) (5 * radius),
(int) (2 * radius));
      influenceArea.addPoint((int) (5 * radius),
(int) (-2 * radius));
      influenceArea.addPoint((int) radius, (int) -
radius);
      // move and rotate to world space according
to direction
      Point2D ps = new Point2D.Double();
      Point2D pd = new Point2D.Double();
      AffineTransform transform = new
AffineTransform();
      transform.translate(position.x, position.y);
      transform.rotate(direction *
Math.toRadians(90));
      for (int p = 0; p < influenceArea.npoints; p++)
{
         ps.setLocation(influenceArea.xpoints[p],
influenceArea.ypoints[p]);
```

```
        ps.setLocation(influenceArea.xpoints[p],
influenceArea.ypoints[p]);
        transform.transform(ps, pd);
        influenceArea.xpoints[p] = (int) pd.getX();
        influenceArea.ypoints[p] = (int) pd.getY();
      }
    }

    public Estructura getEstructura() {
      return EstructuraPe;
    }

    public Vector getPosition() {
      return position;
    }

    public double getRadius() {
      return radius;
    }

    public int getDirection() {
      return direction;
    }

    public Polygon getInfluenceArea() {
      return influenceArea;
    }

    public boolean isFired() {
      return System.currentTimeMillis() -
firedStartTime < 200;
    }

    public void addListener(aire listener) {
      listeners.add(listener);
    }

    public void Actualizar() {
      for (int i = 0; i < influenceArea.npoints; i++) {
        line.getA().set(influenceArea.xpoints[i],
influenceArea.ypoints[i]);
        int nextIndex = (i + 1) %
influenceArea.npoints;

line.getB().set(influenceArea.xpoints[nextIndex],
influenceArea.ypoints[nextIndex]);
        Vector candyPivot =
EstructuraPe.getDulceClas().getPivot();

        if ((influenceArea.contains(candyPivot.x,
candyPivot.y)
            ||
line.intersectsWithCircle(candyPivot,
EstructuraPe.getDulceClas().getRadius()))
            && isFired()) {

EstructuraPe.getDulceClas().addForce(wind);
          return;
        }
      }
    }

    public void drawDebug(Graphics2D g) {
      if (isFired()) {
```

```
            g.setColor(Color.RED);
        }
        else {
            g.setColor(Color.GREEN);
        }
        g.drawOval((int) (position.x - radius), (int)
(position.y - radius), (int) (2 * radius), (int) (2 *
radius));
        g.draw(influenceArea);
    }

    public void tryToFire(double x, double y) {
        vTmp.set(x, y);
        vTmp.sub(position);
        if (EstructuraPe.isPlaying() && vTmp.getSize()
<= radius) {
            fire();
            fireOnAirCushionFire();
        }
    }

    private void fire() {
        firedStartTime = System.currentTimeMillis();
    }

    private void fireOnAirCushionFire() {
        for (aire listener : listeners) {
            listener.AreRosaGuada();
        }
    }
}
```

```
class Bubble {

    private final Estructura EstructuraPe;
    private final Vector position = new Vector();
    private double radius;
    private boolean visible = true;
    private DulceClas candy;

    private final Vector upForce;

    private final Vector vTmp = new Vector();
    private final List<Burbuja> listeners = new
ArrayList<Burbuja>();

    public Bubble(Estructura EstructuraPe, double
x, double y, double radius) {
        this.EstructuraPe = EstructuraPe;
        this.radius = radius;
        this.position.set(x, y);
        this.upForce = new Vector(0, -1);
    }

    public Estructura getEstructura() {
        return EstructuraPe;
    }

    public Vector getPosition() {
        return position;
    }

    public double getRadius() {
```

```
    return radius;
  }

  public boolean isVisible() {
    return visible;
  }

  public DulceClas getDulceClas() {
    return candy;
  }

  public void addListener(Burbuja listener) {
    listeners.add(listener);
  }

  public void Actualizar() {
    if (!visible) {
      return;
    }
    else if (candy != null) {
      if (!candy.isVisible()) {
        burst();
        return;
      }

      if (candy.getAttachedRopes().isEmpty()) {
        upForce.y = -0.6;
      }
      else if (candy.getPoints()[0].getVelocity().y
> 0.1) {
        upForce.y -= 0.02;
      }

      candy.addForce(upForce);

      Vector candyPivot = candy.getPivot();
      position.x += (candyPivot.x - position.x) *
0.95;
      position.y += (candyPivot.y - position.y) *
0.95;
    }
    else {
      Vector candyPivot =
EstructuraPe.getDulceClas().getPivot();
      vTmp.set(candyPivot);
      vTmp.sub(position);
      if (candy == null && vTmp.getSize() <=
radius && EstructuraPe.getDulceClas().isVisible())
{
        candy = EstructuraPe.getDulceClas();
        fireOnDulceClasCaught();
      }
    }
  }

  public void tryToBurst(double x, double y) {
    if (!visible) {
      return;
    }
    vTmp.set(x, y);
    vTmp.sub(position);
    if (EstructuraPe.isPlaying() && candy != null
&& vTmp.getSize() <= radius) {
      burst();
```

```
        }
    }

    private void burst() {
        visible = false;
        fireOnBurst();
    }

    public void drawDebug(Graphics2D g) {
        AffineTransform at = g.getTransform();
        g.translate(position.x, position.y);
        g.setColor(Color.CYAN);
        g.drawOval((int) (-radius), (int) (-radius), (int)
(2 * radius), (int) (2 * radius));
        g.setTransform(at);
    }

    private void fireOnBurst() {
        for (Burbuja listener : listeners) {
            listener.bur();
        }
    }

    private void fireOnDulceClasCaught() {
        for (Burbuja listener : listeners) {
            listener.DulceRoto();
        }
    }

}
```

```
class DulceClas {

    private final Estructura EstructuraPe;
    private final Vector position = new Vector();
    private double radius;
    private final Vector pivot = new Vector();
    private boolean visible = true;
    private Particle[] particles = new Particle[4];
    private Stick[] sticks = new Stick[6];
    private final List<Rope> attachedRopes = new
ArrayList<Rope>();
    private final Vector vTmp = new Vector();
    private final List<Dulce> listeners = new
ArrayList<Dulce>();
    private boolean destroyed = false;

    public DulceClas(Estructura EstructuraPe,
double x, double y, double radius) {
        this.EstructuraPe = EstructuraPe;
        this.position.set(x, y);
        this.radius = radius;
        create();
    }

    public void addListener(Dulce listener) {
        listeners.add(listener);
    }

    public Estructura getEstructura() {
        return EstructuraPe;
    }
```

```
public Vector getPosition() {
   return position;
}

public double getRadius() {
   return radius;
}

public boolean isDestroyed() {
   return destroyed;
}

public Vector getPivot() {
   pivot.set(particles[1].position);
   pivot.sub(particles[3].position);
   pivot.scale(0.5);
   pivot.add(particles[3].position);
   return pivot;
}

public void setPivot(double x, double y) {
   getPivot();
   pivot.x = x - pivot.x;
   pivot.y = y - pivot.y;
   for (Particle p : particles) {
      p.position.x += pivot.x;
      p.position.y += pivot.y;
   }
}

public double getAngle() {
   vTmp.set(particles[1].position);
   vTmp.sub(particles[3].position);
   return Math.atan2(vTmp.y, vTmp.x);
}

public boolean isVisible() {
   return visible;
}

void setVisible(boolean visible) {
   this.visible = visible;
}

public Particle[] getPoints() {
   return particles;
}

public Stick[] getSticks() {
   return sticks;
}

List<Rope> getAttachedRopes() {
   return attachedRopes;
}

private final List<Rope> attachedRopesTmp =
new ArrayList<Rope>();

public void detachAllRopes() {
   attachedRopesTmp.clear();
   attachedRopesTmp.addAll(attachedRopes);
```

```
        for (Rope rope : attachedRopesTmp) {
            rope.dettachDulceClas();
        }
    }

    private void create() {
        World world = EstructuraPe.getWorld();
        world.addParticle(particles[0] = new
Particle(world, position.x, position.y));
        world.addParticle(particles[1] = new
Particle(world, position.x + radius, position.y +
radius));
        world.addParticle(particles[2] = new
Particle(world, position.x, position.y + 2 *
radius));
        world.addParticle(particles[3] = new
Particle(world, position.x - radius, position.y +
radius));

        world.addStick(sticks[0] = new
Stick(particles[0], particles[1], 0, true));
        world.addStick(sticks[1] = new
Stick(particles[1], particles[2], 0, true));
        world.addStick(sticks[2] = new
Stick(particles[2], particles[3], 0, true));
        world.addStick(sticks[3] = new
Stick(particles[3], particles[0], 0, true));
        world.addStick(sticks[4] = new
Stick(particles[0], particles[2], 0, true));
        world.addStick(sticks[5] = new
Stick(particles[1], particles[3], 0, true));
    }

    public void Actualizar() {
        Vector candyPivot = getPivot();
        if (!EstructuraPe.isLevelFailured()
                && ((candyPivot.y < (3 * -radius)
                || candyPivot.y >
(EstructuraPe.getWorld().getHeight() + 3 *
radius)))) {
            EstructuraPe.NivelFallado();
        }
    }

    public void drawDebug(Graphics2D g) {
        AffineTransform at = g.getTransform();
        g.translate(getPivot().x, getPivot().y);
        g.rotate(getAngle());
        g.setColor(Color.MAGENTA);
        g.drawOval((int) (-radius), (int) (-radius), (int)
(2 * radius), (int) (2 * radius));
        g.drawLine(0, 0, (int) radius, 0);
        g.setTransform(at);
    }

    public void addForce(Vector force) {
        for (Particle p : particles) {
            p.addForce(force);
        }
    }

    public void destroy() {
        if (destroyed) {
            return;
        }
        destroyed = true;
        visible = false;
        detachAllRopes();
```

```
        fireOnDulceClasDestroyed();
        EstructuraPe.NivelFallado();
    }

    private void fireOnDulceClasDestroyed() {
        for (Dulce listener : listeners) {
            listener.DulceRotoD();
        }
    }

}

class LevelLoader {

    private Estructura EstructuraPe;
    private Map<String, Object> objects = new
HashMap<String, Object>();

    public LevelLoader(Estructura EstructuraPe) {
        this.EstructuraPe = EstructuraPe;
    }

    public void loadFromResource(String
levelName) {
        objects.clear();
        try {
            load(levelName);
        } catch (Exception ex) {

Logger.getLogger(LevelLoader.class.getName()).lo
g(Level.SEVERE, null, ex);
            System.exit(-1);
        }
    }

    private void load(String levelName) throws
Exception {
        BufferedReader br = new
BufferedReader(new
InputStreamReader(getClass().getResourceAsStre
am(levelName)));
        String line = null;
        while ((line = br.readLine()) != null) {
            line = line.trim();
            if (line.isEmpty() || line.startsWith("#")) {
                continue;
            }
            String[] args = line.split(",");
            String cmd = args[0].trim();
            String name = args[1].trim();

            if (cmd.equals("ac")) {
                int x = Integer.parseInt(args[2].trim());
                int y = Integer.parseInt(args[3].trim());
                int direction =
Integer.parseInt(args[4].trim());
                AirCushion airCushion =
EstructuraPe.createAirCushion(x, y, direction);
                objects.put(name, airCushion);
            }

            else if (cmd.equals("bu")) {
                int x = Integer.parseInt(args[2].trim());
                int y = Integer.parseInt(args[3].trim());
```

```
            int radius =
Integer.parseInt(args[4].trim());

            Bubble bubble =
EstructuraPe.createBubble(x, y, radius);

            objects.put(name, bubble);

        }


        else if (cmd.equals("ca")) {

            int x = Integer.parseInt(args[2].trim());

            int y = Integer.parseInt(args[3].trim());

            int radius =
Integer.parseInt(args[4].trim());

            DulceClas candy =
EstructuraPe.createDulceClas(x, y, radius);

            objects.put(name, candy);

        }


        else if (cmd.equals("cr")) {

            String ropeName = args[1].trim();

            int candyPositionIndex =
Integer.parseInt(args[2].trim());

            Rope rope = (Rope)
objects.get(ropeName);

            rope.attach(EstructuraPe.getDulceClas(),
candyPositionIndex);

        }


        else if (cmd.equals("pe")) {

            int x = Integer.parseInt(args[2].trim());

            int y = Integer.parseInt(args[3].trim());

            int radius =
Integer.parseInt(args[4].trim());

            int closeDistance =
Integer.parseInt(args[5].trim());

            Pet pet = EstructuraPe.createPet(x, y,
radius, closeDistance);

            objects.put(name, pet);

        }

        else if (cmd.equals("pi")) {

            String ropeName = args[2].trim();

            Rope rope = (Rope)
objects.get(ropeName);

            Pin pin =
EstructuraPe.createPin(rope.getFirstParticle());

            objects.put(name, pin);

        }


        else if (cmd.equals("pr")) {

            int x = Integer.parseInt(args[2].trim());

            int y = Integer.parseInt(args[3].trim());

            int radius =
Integer.parseInt(args[4].trim());

            int ropeLength =
Integer.parseInt(args[5].trim());

            PinRope pinRope =
EstructuraPe.createPinRope(x, y, radius,
ropeLength);

            objects.put(name, pinRope);

        }


        else if (cmd.equals("ro")) {

            int x1 = Integer.parseInt(args[2].trim());

            int y1 = Integer.parseInt(args[3].trim());

            int x2 = Integer.parseInt(args[4].trim());

            int y2 = Integer.parseInt(args[5].trim());

            Rope rope =
EstructuraPe.createRope(x1, y1, x2, y2);

            objects.put(name, rope);
```

```
        }


        else if (cmd.equals("sp")) {
            int x = Integer.parseInt(args[2].trim());
            int y = Integer.parseInt(args[3].trim());
            int w = Integer.parseInt(args[4].trim());
            int h = Integer.parseInt(args[5].trim());
            Spikes spikes =
EstructuraPe.createSpikes(x, y, w, h);
            objects.put(name, spikes);
        }


        else if (cmd.equals("st")) {
            int x = Integer.parseInt(args[2].trim());
            int y = Integer.parseInt(args[3].trim());
            int radius =
Integer.parseInt(args[4].trim());
            Star star = EstructuraPe.createStar(x, y,
radius);
            objects.put(name, star);
        }
      }
   }


}



class Estructura {


   private final World world;
   private final SlashTrail slashTrail;
   private final Line cutLine = new Line(0, 0, 0, 0);
   private Pet pet;
   private DulceClas candy;
   private final List<Rope> ropes = new
ArrayList<Rope>();
   private final List<Pin> pins = new
ArrayList<Pin>();
   private final List<Bubble> bubbles = new
ArrayList<Bubble>();
   private final List<Spikes> spikesList = new
ArrayList<Spikes>();
   private final List<AirCushion> airCushions =
new ArrayList<AirCushion>();
   private final List<PinRope> pinRopes = new
ArrayList<PinRope>();
   private final List<Star> stars = new
ArrayList<Star>();


   private List<escuchar> listeners = new
ArrayList<escuchar>();


   private String NombreActualNivel;
   private final LevelLoader levelLoader;
   private boolean NivelFallado;
   private boolean NivelCompleto;


   private final Vector vTmp = new Vector();


   public Estructura(int width, int height, int
slashTrailSize) {
      world = new World(width, height);
      slashTrail = new SlashTrail(slashTrailSize);
      levelLoader = new LevelLoader(this);
   }
```

```
    public World getWorld() {
        return world;
    }

    public SlashTrail getSlashTrail() {
        return slashTrail;
    }

    public void addSlashTrail(int x, int y) {
        slashTrail.addTrail(x, y);
        tryToCutRope();
        tryToBurstBubbles(x, y);
        tryToFireAirCushions(x, y);
    }

    private void tryToCutRope() {
        for (int i = 0; i < slashTrail.getTrail().size() - 1;
i++) {
            Point p1 = slashTrail.getTrail().get(i);
            Point p2 = slashTrail.getTrail().get(i + 1);
            if (p1 != null && p2 != null) {
                cutLine.getA().set(p1.x, p1.y);
                cutLine.getB().set(p2.x, p2.y);
                for (Rope rope : ropes) {
                    rope.cut(cutLine);
                }
            }
        }
    }

    private void tryToBurstBubbles(double x,
double y) {
        for (Bubble bubble : bubbles) {
            bubble.tryToBurst(x, y);
        }
    }

    private void tryToFireAirCushions(double x,
double y) {
        for (AirCushion airCushion : airCushions) {
            airCushion.tryToFire(x, y);
        }
    }

    public Pet getPet() {
        return pet;
    }

    public DulceClas getDulceClas() {
        return candy;
    }

    public List<Rope> getRopes() {
        return ropes;
    }

    public List<Pin> getPins() {
        return pins;
    }

    public List<PinRope> getPinRopes() {
```

```
    return pinRopes;
}

public List<Star> getStars() {
    return stars;
}

public List<Bubble> getBubbles() {
    return bubbles;
}

public List<Spikes> getSpikesList() {
    return spikesList;
}

public List<AirCushion> getAirCushions() {
    return airCushions;
}

public void addListener(escuchar listener) {
    listeners.add(listener);
}

public List<escuchar> getListeners() {
    return listeners;
}

public String getCurrentLevelName() {
    return NombreActualNivel;
}

    public boolean isLevelFailured() {
        return NivelFallado;
    }

    public boolean isLevelCleared() {
        return NivelCompleto;
    }

    public boolean isPlaying() {
        return !NivelCompleto && !NivelFallado;
    }

    public Pet createPet(double x, double y, double
radius, double closeDistance) {
        return pet = new Pet(this, x, y, radius,
closeDistance);
    }

    public DulceClas createDulceClas(double x,
double y, double radius) {
        return candy = new DulceClas(this, x, y,
radius);
    }

    public Rope createRope(double x1, double y1,
double x2, double y2) {
        vTmp.set(x2 - x1, y2 - y1);
        double segmentSize = 15;
        int segmentsNumber = (int) (vTmp.getSize() /
segmentSize);
        Rope rope = new Rope(this, x1, y1, x2, y2,
segmentsNumber);
```

```
        ropes.add(rope);
        return rope;
    }

    public Pin createPin(Particle p) {
        Pin pin = new Pin(p);
        pins.add(pin);
        return pin;
    }

    public Bubble createBubble(double x, double y, double radius) {
        Bubble bubble = new Bubble(this, x, y, radius);
        bubbles.add(bubble);
        return bubble;
    }

    public Spikes createSpikes(int x, int y, int w, int h) {
        Spikes spikes = new Spikes(this, x, y, w, h);
        spikesList.add(spikes);
        return spikes;
    }

    public AirCushion createAirCushion(int x, int y, int direction) {
        AirCushion airCushion = new AirCushion(this, x, y, direction);
        airCushions.add(airCushion);
        return airCushion;
    }

    public PinRope createPinRope(double x, double y, double radius, double ropeLength) {
        PinRope pinRope = new PinRope(this, x, y, radius, ropeLength);
        pinRopes.add(pinRope);
        return pinRope;
    }

    public Star createStar(double x, double y, double radius) {
        Star star = new Star(this, x, y, radius);
        stars.add(star);
        return star;
    }

    // ---

    private void clear() {
        NivelCompleto = false;
        NivelFallado = false;
        world.clear();
        slashTrail.clear();
        pet = null;
        candy = null;
        ropes.clear();
        pins.clear();
        bubbles.clear();
        spikesList.clear();
        airCushions.clear();
        pinRopes.clear();
        stars.clear();
    }
```

```
public void Actualizar() {
    if (candy != null) {
        candy.Actualizar();
    }
    if (pet != null) {
        pet.Actualizar();
    }
    ActualizarBubble();
    ActualizarSpikes();
    ActualizarAirCushions();
    ActualizarPinRopes();
    ActualizarStars();
    world.Actualizar();
}

private void ActualizarBubble() {
    for (Bubble bubble : bubbles) {
        bubble.Actualizar();
    }
}

private void ActualizarSpikes() {
    for (Spikes spike : spikesList) {
        spike.Actualizar();
    }
}

private void ActualizarAirCushions() {
    for (AirCushion airCushion : airCushions) {
        airCushion.Actualizar();
    }
}

private void ActualizarPinRopes() {
    for (PinRope pinRope : pinRopes) {
        pinRope.Actualizar();
    }
}

private void ActualizarStars() {
    for (Star star : stars) {
        star.Actualizar();
    }
}

public void drawDebug(Graphics2D g) {
    if (pet != null) {
        pet.drawDebug(g);
    }
    if (candy != null && candy.isVisible()) {
        candy.drawDebug(g);
    }
    for (Rope rope : ropes) {
        rope.drawDebug(g);
    }
    for (Pin pin : pins) {
        pin.drawDebug(g);
    }
    for (Spikes spike : spikesList) {
        spike.drawDebug(g);
    }
```

```
    for (AirCushion airCushion : airCushions) {
        airCushion.drawDebug(g);
    }
    for (PinRope pinRope : pinRopes) {
        pinRope.drawDebug(g);
    }
    for (Star star : stars) {
        if (star.isVisible()) {
            star.drawDebug(g);
        }
    }
    for (Bubble bubble : bubbles) {
        if (bubble.isVisible()) {
            bubble.drawDebug(g);
        }
    }
    slashTrail.drawDebug(g);
}

public void ComenzarNivel(String levelName) {
    clear();
    levelLoader.loadFromResource(levelName);
    NombreActualNivel = levelName;
    System.gc();
}

public void retryCurrentLevel() {
    if (NombreActualNivel != null) {
        ComenzarNivel(NombreActualNivel);
    }
}
```

```
    void NivelFallado() {
        if (NivelFallado) {
            return;
        }
        NivelFallado = true;
        fireOnFailured();
    }

    void NivelCompleto() {
        NivelCompleto = true;
        candy.setVisible(false);
        fireOnLevelCleared();
    }

    private void fireOnFailured() {
        for (escuchar listener : listeners) {
            listener.Fallado();
        }
    }

    private void fireOnLevelCleared() {
        for (escuchar listener : listeners) {
            listener.NivelCopleLis();
        }
    }

}

class Pet {
```

```
    private final Estructura EstructuraPe;
    private final Vector position = new Vector();
    private double radius;
    private final List<Animalito> listeners = new
ArrayList<Animalito>();
    private final Vector vTmp = new Vector();
    private final double closeDistance;
    private boolean candyClose;


    public Pet(Estructura EstructuraPe, double x,
double y, double radius, double closeDistance) {
        this.EstructuraPe = EstructuraPe;
        this.radius = radius;
        this.position.set(x, y);
        this.closeDistance = closeDistance;
    }


    public Estructura getEstructura() {
        return EstructuraPe;
    }


    public Vector getPosition() {
        return position;
    }


    public double getRadius() {
        return radius;
    }


    public void addListener(Animalito listener) {
        listeners.add(listener);
    }


    public void Actualizar() {
        if (!EstructuraPe.isLevelCleared()) {
            Vector candyPivot =
EstructuraPe.getDulceClas().getPivot();
            vTmp.set(candyPivot);
            vTmp.sub(position);
            if (vTmp.getSize() <= radius &&
!EstructuraPe.getDulceClas().isDestroyed()) {
                fireOnDulceClasEaten();
                EstructuraPe.NivelCompleto();
            }
            else if (!candyClose && vTmp.getSize() <=
(radius + closeDistance)) {
                candyClose = true;
                fireOnDulceClasClose();
            }
            else if (candyClose && vTmp.getSize() >
(radius + closeDistance)) {
                candyClose = false;
                fireOnDulceClasEscaped();
            }


        }
    }


    public boolean isDulceClasAbove() {
        Vector candyPivot =
EstructuraPe.getDulceClas().getPivot();
        return Math.abs(candyPivot.x - position.x) <
(1.5 * radius)
                && candyPivot.y < position.y;
```

```
    }

    public void drawDebug(Graphics2D g) {
        g.setColor(Color.GRAY);
        g.drawOval((int) (position.x - radius), (int) (position.y - radius), (int) (2 * radius), (int) (2 * radius));
        g.drawOval((int) (position.x - (radius + closeDistance)), (int) (position.y - (radius + closeDistance)), (int) (2 * (radius + closeDistance)), (int) (2 * (radius + closeDistance)));
    }

    private void fireOnDulceClasClose() {
        for (Animalito listener : listeners) {
            listener.DulceNo();
        }
        //System.out.println("NEEL");
    }

    private void fireOnDulceClasEscaped() {
        for (Animalito listener : listeners) {
            listener.DulceFuera();
        }
        //System.out.println("SE FUEE");
    }

    private void fireOnDulceClasEaten() {
        for (Animalito listener : listeners) {
            listener.DulceComido();
        }
        //System.out.println("Si comio :)");
    }

}

class Pin {

    private final Particle p;

    public Pin(Particle p) {
        this.p = p;
        p.setPinned(true);
    }

    public Pin(Particle p, double x, double y) {
        this.p = p;
        p.position.set(x, y);
        p.setPinned(true);
    }

    public void drawDebug(Graphics2D g) {
        g.setColor(Color.RED);
        g.fillOval((int) (p.position.x - 3), (int) (p.position.y), 6, 6);
    }

}

class PinRope {
    private final Estructura EstructuraPe;
    private final Vector position = new Vector();
    private final double radius; // influence area;
    private final double ropeLength;
```

```
    private final Vector vTmp = new Vector();

    private final List<romper> listeners = new
ArrayList<romper>();

    private Particle p;

    private Rope rope;

    public PinRope(Estructura EstructuraPe, double
x, double y, double radius, double ropeLength) {

        this.EstructuraPe = EstructuraPe;

        this.position.set(x, y);

        this.radius = radius;

        this.ropeLength = ropeLength;

    }

    public Estructura getEstructura() {

        return EstructuraPe;

    }

    public Vector getPosition() {

        return position;

    }

    public double getRadius() {

        return radius;

    }

    public Particle getP() {

        return p;

    }


    public Rope getRope() {

        return rope;

    }


    public void addListener(romper listener) {

        listeners.add(listener);

    }

    public void Actualizar() {

        if (rope == null) {

            Vector candyPivot =
EstructuraPe.getDulceClas().getPivot();

            vTmp.set(candyPivot);

            vTmp.sub(position);

            if (vTmp.getSize() <= (radius +
EstructuraPe.getDulceClas().getRadius()) &&
EstructuraPe.getDulceClas().isVisible()) {

                rope =
EstructuraPe.createRope(position.x, position.y,
position.x, position.y + ropeLength);

                p = rope.getFirstParticle();

                p.setPinned(true);

                rope.attach(EstructuraPe.getDulceClas(),
0);

                fireOnRopeCreated();

            }

        }

    }

    public void drawDebug(Graphics2D g) {

        g.setColor(Color.PINK);

        g.fillOval((int) (position.x - 3), (int) (position.y
- 3), 6, 6);

        g.drawOval((int) (position.x - radius), (int)
(position.y - radius), (int) (2 * radius), (int) (2 *
radius));

    }

    private void fireOnRopeCreated() {

        for (romper listener : listeners) {

            listener.CuerdaRotaL(rope);

        }

    }
```

```
}
class Rope {
   public static final double ELASTICITY = 0.1;
   private final Estructura EstructuraPe;
   private final Vector a = new Vector();
   private final Vector b = new Vector();
   private final int segmentsNumber;
   private Particle[] particles;
   private Stick[] sticks;
   private static Line cutLineTmp = new Line(0, 0,
0, 0);
   private boolean cut = false;
   private long cutTime;

   public Rope(Estructura EstructuraPe, double
x1, double y1, double x2, double y2, int
segmentsNumber) {
      this.EstructuraPe = EstructuraPe;
      a.set(x1, y1);
      b.set(x2, y2);
      this.segmentsNumber = segmentsNumber;
      create();
   }
   public Estructura getEstructura() {
      return EstructuraPe;
   }
   public Particle getFirstParticle() {
      return particles[0];
   }
   public Particle getLastParticle() {
      return particles[particles.length - 1];
   }
   public int getSegmentsNumber() {
      return segmentsNumber;
   }
   public Particle[] getParticles() {
      return particles;
   }
   public Stick[] getSticks() {
      return sticks;
   }
   public boolean isCut() {
      return cut;
   }
   public long getCutTime() {
      return cutTime;
   }
   private void create() {
      World world = EstructuraPe.getWorld();
      Vector vTmp = new Vector();
      Vector vTmp2 = new Vector();
      vTmp.set(b);
      vTmp.sub(a);
      double ropeSize = vTmp.getSize();
      double ropeSegmentSize = ropeSize /
segmentsNumber;
      vTmp.normalize();
      vTmp.scale(ropeSegmentSize);
      particles = new Particle[segmentsNumber +
1];
      vTmp2.set(a);
      for (int p = 0; p < particles.length; p++) {
         world.addParticle(particles[p] = new
Particle(world, vTmp2.x, vTmp2.y));
```

```
            vTmp2.add(vTmp);
        }
        sticks = new Stick[segmentsNumber + 1];
        for (int s = 0; s < particles.length - 1; s++) {
            world.addStick(sticks[s] = new
Stick(particles[s], particles[s+1], ELASTICITY,
true));
        }
    }
    public void attach(DulceClas candy, int
positionIndex)
{EstructuraPe.getWorld().addStick(sticks[sticks.le
ngth - 1] = new Stick(getLastParticle(),
candy.getPoints()[positionIndex], ELASTICITY,
true));
        sticks[sticks.length -
1].setSize(sticks[0].getSize());
        candy.getAttachedRopes().add(this);
    }
    public void dettachDulceClas() {
        World world = EstructuraPe.getWorld();
        Particle cb = sticks[sticks.length - 1].getB();
        Particle dp = new Particle(world,
cb.position.x, cb.position.y);
        sticks[sticks.length - 1].setB(dp);
        world.addParticle(dp);

EstructuraPe.getDulceClas().getAttachedRopes().r
emove(this);
    }
    public void cut(Line line) {
        cut(line.getA().x, line.getA().y, line.getB().x,
line.getB().y);
    }
public void cut(double x1, double y1, double x2,
double y2) {
        if (!EstructuraPe.isPlaying() || cut) {
            return;
        }
        World world = EstructuraPe.getWorld();
        cutLineTmp.getA().set(x1, y1);
        cutLineTmp.getB().set(x2, y2);
        for (Stick stick : sticks) {
            Vector ip =
cutLineTmp.getSegIntersectionPoint(stick.getLine
());
            if (ip != null) {
                Particle previousB = stick.getB();
                Particle np1 = new Particle(world, ip.x,
ip.y);
                world.addParticle(np1);
                stick.setB(np1);
                Particle np2 = new Particle(world, ip.x,
ip.y);
                world.addParticle(np2);
                world.addStick(new Stick(np2,
previousB, ELASTICITY, true));
                dettachDulceClas();
                cut = true;
                cutTime = System.currentTimeMillis();
                break;
            }
        }
    }
    public void drawDebug(Graphics2D g) {
        g.setColor(Color.ORANGE);
        for (Stick stick : sticks) {
            if (stick != null && stick.isVisible()) {
                stick.drawDebug(g);
```

```
        }
      }
    }
}
class SlashTrail {
    private Color color = Color.WHITE;
    private final List<Point> trail = new
ArrayList<Point>();
    private final int size;
    private final Stroke[] strokes;
    private boolean visible;
    public SlashTrail(int size) {
        this(size, 1);
    }
    public SlashTrail(int size, double scale) {
        this.size = size;
        strokes = new Stroke[size];
        createStrokes(scale);
    }
    public Color getColor() {
        return color;
    }
    public void setColor(Color color) {
        this.color = color;
    }
    public List<Point> getTrail() {
        return trail;
    }
    public Stroke[] getStrokes() {
        return strokes;
    }
    public boolean isVisible() {
        return visible;
    }
    private void createStrokes(double scale) {
        for (int i = 0; i < strokes.length; i++) {
            strokes[i] = new BasicStroke((float) (1 + i *
scale), BasicStroke.CAP_ROUND,
BasicStroke.JOIN_ROUND);
        }
    }
    public int getSize() {
        return size;
    }
    public void addTrail(int x, int y) {
        if (x < 0 || y < 0) {
            trail.add(null);
        }
        else {
            trail.add(new Point(x, y));
        }

        while (trail.size() > size) {
            trail.remove(0);
        }
        visible = true;
    }
    public void clear() {
        trail.clear();
    }
    public void drawDebug(Graphics2D g) {
        if (!visible) {
```

```
            return;
        }
        Stroke originalStroke = g.getStroke();
        for (int i = 0; i < trail.size() - 1; i++) {
            g.setStroke(strokes[i]);
            Point p1 = trail.get(i);
            Point p2 = trail.get(i + 1);
            if (p1 != null && p2 != null) {
                g.setColor(color);
                g.drawLine(p1.x, p1.y, p2.x + 1, p2.y + 1);
            }
        }
        g.setStroke(originalStroke);
    }
}
class Spikes {
    private final Estructura EstructuraPe;
    private final Polygon polygon = new Polygon();
    private final Line line = new Line(0, 0, 0, 0);
    private final Rectangle rectangle = new
Rectangle();
        public Spikes(Estructura EstructuraPe, int x,
int y, int w, int h) {
        this.EstructuraPe = EstructuraPe;
        polygon.addPoint(x, y);
        polygon.addPoint(x + w, y);
        polygon.addPoint(x + w, y + h);
        polygon.addPoint(x, y + h);
        rectangle.setBounds(x, y, w, h);
    }
    public Polygon getPolygon() {
        return polygon;
    }
    public Rectangle getRectangle() {
        return rectangle;
    }
    public void Actualizar() {
        for (int i = 0; i < polygon.npoints; i++) {
            line.getA().set(polygon.xpoints[i],
polygon.ypoints[i]);
            int nextIndex = (i + 1) % polygon.npoints;
            line.getB().set(polygon.xpoints[nextIndex],
polygon.ypoints[nextIndex]);
            Vector candyPivot =
EstructuraPe.getDulceClas().getPivot();
            if (polygon.contains(candyPivot.x,
candyPivot.y) ||
line.intersectsWithCircle(candyPivot,
EstructuraPe.getDulceClas().getRadius())) {
                EstructuraPe.getDulceClas().destroy();
                return;
            }
        }
    }
    public void drawDebug(Graphics2D g) {
        g.setColor(Color.PINK);
        g.draw(polygon);
    }
}
class Star {
    private final Estructura EstructuraPe;
    private final Vector position = new Vector();
    private double radius;
    private boolean visible = true;
```

```
    private final List<Estrellita> listeners = new
ArrayList<Estrellita>();

    private final Vector vTmp = new Vector();

    public Star(Estructura EstructuraPe, double x,
double y, double radius) {

        this.EstructuraPe = EstructuraPe;

        this.radius = radius;

        this.position.set(x, y);

    }

    public Estructura getEstructura() {

        return EstructuraPe;

    }

    public Vector getPosition() {

        return position;

    }

    public double getRadius() {

        return radius;

    }

    public boolean isVisible() {

        return visible;

    }

    public void addListener(Estrellita listener) {

        listeners.add(listener);

    }

    public void Actualizar() {

        if (!EstructuraPe.isLevelCleared()) {

            Vector candyPivot =
EstructuraPe.getDulceClas().getPivot();

            vTmp.set(candyPivot);

            vTmp.sub(position);

            if (visible && vTmp.getSize() <= (radius +
EstructuraPe.getDulceClas().getRadius())) {

                visible = false;

                fireOnStarCaught();

            }

        }

    }

    public void drawDebug(Graphics2D g) {

        g.setColor(Color.YELLOW);

        g.fillOval((int) (position.x - radius), (int)
(position.y - radius), (int) (2 * radius), (int) (2 *
radius));

    }

    private void fireOnStarCaught() {

        for (Estrellita listener : listeners) {

            listener.MuereEstrella();

        }

    }

}
```

Bibliografía:

Oracle (1993, 2018). Uses of Class java.awt.Graphics2D Recuperado de: https://docs.oracle.com/javase/7/docs/api/java/awt/class-use/Graphics2D.html

Oracle (1993, 2018). Class Graphics2D Recuperado de: https://docs.oracle.com/javase/7/docs/api/java/awt/Graphics2D.html