UNIVERSIDAD DE MARGARITA

ALMA MATER DEL CARIBE

VICERRECTORADO ACADÉMICO

DECANATO DE INGENIERIA

UNIDAD CURRICULAR: ESTRUCTURA DE DATOS

PROYECTO I

FACILITADOR:

PROF. CESAR REQUENA

ELABORADO POR:

ROLANDO RIVAS C.I: 30.910.806

ANDRES OSORIO C.I: 30.919.870

EL VALLE DEL ESPÍRITU SANTO, FEBRERO DEL 2024

Al iniciar el proyecto se muestra un menú donde se encuentran 2 botones, Jugar y Salir, al presionar el botón de jugar se redirecciona al programa en sí, con el cursor seleccionas en qué lugar caerá la fruta, las cuales salen en orden aleatorio las 5 primeras, teniendo cada una algunas características las cuales se utilizan para compararlas a la hora de la fusión o colisión.

A la hora de chocar frutas se verifica su posición usando la librería de vectores 2D y el tipo de frutas, para verificar si estás se chocan se tratan como a un círculo los cuales la distancia mínima en caso de que se choquen es la suma de sus radios, deduciendo así si chocan o no, en caso de que las frutas choquen y sea iguales se llama la función "Fusión" para así crear la nueva fruta, sumando sus características.

En caso de que no sean la misma fruta, ocurre una colisión en la que se ve implicado el tamaño para hacer la reacción de rebote entre ellas. Continuamente por cada fusión de frutas se irá añadiendo puntaje al marcador, contando también el número de clics que se dan en la partida.

El juego acaba cuando las frutas sobrepasan el límite del frasco que se encuentra, mostrando luego un menú de Game over con un botón de salir para terminar el proceso de ejecución del programa.

Cuando hablamos de las colisiones, estas funcionan actualizando la posición de las bolas utilizando el vector unitario, que permite tener un vector con la misma dirección, pero con distinta magnitud, calculamos la distancia entre los dos puntos utilizando la formula de distancia euclidiana para compararla con la distancia mínima.

En si el programa se ejecuta completamente en la clase “Animación”, ahí se inicia declarando las variables donde se almacenará cada posición donde toque el cursor, para así dejar caer las frutas en la posición deseada, en esta clase también se van utilizando métodos como el limitador de fps, la configuración de JFrame donde se muestra el juego, la creación de cada fruta mostrada y algunas otras necesarias para el óptimo funcionamiento del programa.

El agrupamiento de las frutas en el programa se da a partir de una cola, donde se van añadiendo y así poder manejar datos de las mismas en el programa. Parte importante del

proyecto se da con el uso de librerías correctamente, de las mas destacadas está la librería de “Vector2D” utilizada para las colisiones anteriormente explicada, también la de graphics como su nombre lo dice, para mostrar en pantalla lo necesario para el usuario poder interactuar con el programa, la librería de elipse para dibujar en pantalla las frutas y una gran variedad para el funcionamiento interno del programa.

Para finalizar tenemos el package de imágenes en donde se guardan toda la variedad de imágenes utilizadas en el proyecto, para así luego cargarlas con la clase “CargadorDeImágenes” como un asset a cada una de las frutas generadas incluyendo en esos assets el fondo de juego, para así finalizar la interfaz donde el usuario interactúa con el programa.

PSEUOCODIGO:

```
// Clase principal que extiende JPanel e implementa ActionListener

class Animacion extends JPanel implements ActionListener {

    // Variables de coordenadas del cursor

    public static int cursorX;

    private static int cursorY;


    // Clase para controlar el siguiente elemento

    private Siguiente siguiente = null;


    // Fuente para el contador de clics y puntaje

    Font Arial24 = new Font("Arial", Font.BOLD, 24);


    // Contador de clics

    private int clickCounter;
```

```
    // Constructor de la clase
    public Animacion(int pixelWidth, int pixelHeight, int fps) {
        super(true);
        // Configuración del temporizador para la animación
        this.timer = new Timer(1000 / fps, this);
        this.deltaT = 1.0 / fps;

        // Creación de la lista de bolas
        this.lista = new Colisiones(pixelWidth / pixelsPerMeter, pixelHeight / pixelsPerMeter,
this);

        // Configuración del JPanel
        this.setOpaque(false);
        this.setPreferredSize(new Dimension(pixelWidth, pixelHeight));

        // Inicialización de la clase Siguiente
        siguiente = new Siguiente();

        // Inicialización del GeneradorFrutas
        ballFactory = new GeneradorFrutas(lista, siguiente);
    }
}
```

```
// Cargar los assets
declararAssets.init();

// Factor de conversión de píxeles a metros
private static final double pixelsPerMeter = 200;
private Colisiones lista;
GeneradorFrutas ballFactory = null;
private Timer timer;
private double deltaT;

// Método para iniciar la animación
public void start() {
    timer.start();
}

// Método para detener la animación
public void stop() {
    timer.stop();
}

// Obtener el contador de clics
public int getClickCounter() {
    return clickCounter;
```

```
}

// Establecer el contador de clics
public void setClickCounter(int clickCounter) {
    this.clickCounter = clickCounter;
}

// Método para dibujar las bolas en el panel
@Override
protected void paintComponent(Graphics g) {
    // Dibujar fondo
    super.paintComponents(g);
    Graphics2D g2 = (Graphics2D) g;
    g2.drawImage(declararAssets.fondo, 0, 0, this);

    // Dibujar cada bola en la lista
    for (Bola b : lista.Bolas) {
        double x = b.posicionActual.getX() - b.radio;
        double y = b.posicionActual.getY() + b.radio;

        // Coordenadas Y invertidas
        Ellipse2D.Double e = new Ellipse2D.Double(
                x * pixelsPerMeter,
```

```
            this.getHeight() - (y * pixelsPerMeter),
            b.radio * 2 * pixelsPerMeter,
            b.radio * 2 * pixelsPerMeter);

        // Asignación de colores según el tipo de bola
        if (b instanceof Datil) {
            int xPos = (int) (b.posicionActual.getX() * pixelsPerMeter) - declararAssets.datil.getWidth() / 2;
            int yPos = this.getHeight() - (int) (b.posicionActual.getY() * pixelsPerMeter) - declararAssets.datil.getHeight() / 2;
            g.drawImage(declararAssets.datil, xPos, yPos, this);
        }
        // Dibujar las bolas según su tipo
if (b instanceof Mamon) {
    int xPos = (int) (b.posicionActual.getX() * pixelsPerMeter) - declararAssets.mamon.getWidth() / 2;
    int yPos = this.getHeight() - (int) (b.posicionActual.getY() * pixelsPerMeter) - declararAssets.mamon.getHeight() / 2;
    g.drawImage(declararAssets.mamon, xPos, yPos, this);
}
if (b instanceof Mamey) {
    int xPos = (int) (b.posicionActual.getX() * pixelsPerMeter) - declararAssets.mamey.getWidth() / 2;
```

```
    int yPos = this.getHeight() - (int) (b.posicionActual.getY() * pixelsPerMeter) - declararAssets.mamey.getHeight() / 2;

    g.drawImage(declararAssets.mamey, xPos, yPos, this);

}

if (b instanceof Cereza) {

    int xPos = (int) (b.posicionActual.getX() * pixelsPerMeter) - declararAssets.cereza.getWidth() / 2;

    int yPos = this.getHeight() - (int) (b.posicionActual.getY() * pixelsPerMeter) - declararAssets.cereza.getHeight() / 2;

    g.drawImage(declararAssets.cereza, xPos, yPos, this);

}

if (b instanceof Pumalaca) {

    int xPos = (int) (b.posicionActual.getX() * pixelsPerMeter) - declararAssets.pumalaca.getWidth() / 2;

    int yPos = this.getHeight() - (int) (b.posicionActual.getY() * pixelsPerMeter) - declararAssets.pumalaca.getHeight() / 2;

    g.drawImage(declararAssets.pumalaca, xPos, yPos, this);

}

if (b instanceof Kiwi) {

    int xPos = (int) (b.posicionActual.getX() * pixelsPerMeter) - declararAssets.kiwi.getWidth() / 2;

    int yPos = this.getHeight() - (int) (b.posicionActual.getY() * pixelsPerMeter) - declararAssets.kiwi.getHeight() / 2;

    g.drawImage(declararAssets.kiwi, xPos, yPos, this);

}
```

```
// Dibujar las bolas según su tipo
if (b instanceof Parchita) {
    int xPos = (int) (b.posicionActual.getX() * pixelsPerMeter) - declararAssets.parchita.getWidth() / 2;
    int yPos = this.getHeight() - (int) (b.posicionActual.getY() * pixelsPerMeter) - declararAssets.parchita.getHeight() / 2;
    g.drawImage(declararAssets.parchita, xPos, yPos, this);
}
if (b instanceof Mango) {
    int xPos = (int) (b.posicionActual.getX() * pixelsPerMeter) - declararAssets.mango.getWidth() / 2;
    int yPos = this.getHeight() - (int) (b.posicionActual.getY() * pixelsPerMeter) - declararAssets.mango.getHeight() / 2;
    g.drawImage(declararAssets.mango, xPos, yPos, this);
}
if (b instanceof Coco) {
    int xPos = (int) (b.posicionActual.getX() * pixelsPerMeter) - declararAssets.coco.getWidth() / 2;
    int yPos = this.getHeight() - (int) (b.posicionActual.getY() * pixelsPerMeter) - declararAssets.coco.getHeight() / 2;
    g.drawImage(declararAssets.coco, xPos, yPos, this);
}
if (b instanceof Patilla) {
    int xPos = (int) (b.posicionActual.getX() * pixelsPerMeter) - declararAssets.patilla.getWidth() / 2;
```

```
    int yPos = this.getHeight() - (int) (b.posicionActual.getY() * pixelsPerMeter) - declararAssets.patilla.getHeight() / 2;
    g.drawImage(declararAssets.patilla, xPos, yPos, this);
}

// Rellenar y dibujar la elipse correspondiente

// Dibujar la cajita donde caen las frutas
g2.setColor(Color.black);
BasicStroke grosorLinea = new BasicStroke(5);
g2.setStroke(grosorLinea);
Path2D jar = new Path2D.Double();

// Parte izquierda
jar.moveTo(600, 10);
jar.lineTo(600, 870);

// Dibujar la parte derecha de la cajita donde caen las frutas
jar.lineTo(1320, 870);
jar.lineTo(1320, 10);

g2.draw(jar);
Toolkit.getDefaultToolkit().sync();
```

```
// Mostrar el contador de turnos y el puntaje del jugador
g.setFont(Arial24);
g.setColor(Color.BLACK);
g.drawString("Turnos: " + clickCounter, 60, 40);
g.drawString("Puntaje: " + lista.obtenerPuntaje(), 60, 120);
}

// Método para dibujar la vista previa de la próxima bola
/*if (Objects.equals(ballFactory.siguiente.getValue(), "Datil")) {
    // Lógica para dibujar la vista previa del datil
    // ...
    return null;
}*/
// Método que se llama en cada paso de la animación
@Override
public void actionPerformed(ActionEvent e) {
    try {
        lista.step(deltaT);
    } catch (IOException ex) {
    }
    this.repaint();
}

// Método para crear la ventana
public static void main(String[] args) {
    javax.swing.SwingUtilities.invokeLater(new Runnable() {
        @Override
        public void run() {
```

```
            // Creación de la animación
            Animacion anim = new Animacion(1440, 900, 120);
            // Configuración de los listeners del mouse
            anim.addMouseListener(new entradaPorMouse(anim, pixelsPerMeter));
            anim.addMouseMotionListener(new MouseMotionAdapter() {
                @Override
                public void mouseMoved(MouseEvent e) {
                    cursorX = e.getX();
                    cursorY = e.getY();
                }
            });

            // Configuración del JFrame y visualización de la animación
            JFrame frame = new JFrame("Suika game");
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.add(anim);
            frame.pack();
            frame.setVisible(true);
           // Configurar la ventana
           frame.setResizable(false);
           frame.setLocationRelativeTo(null);
           frame.setVisible(true);

           // Iniciar la animación
           anim.start();
       }
    });
}
```