

Sistemas multimedia (2018-2019)
GRADO EN INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE GRANADA

Domentación Paint. Práctica de Evaluación.

Montserrat Rodríguez Zamorano

5 de julio de 2019

Índice

1. Descripción del sistema	1
2. Requisitos	1
2.1. Requisitos funcionales	1
2.1.1. Carácter general	1
2.1.2. Dibujo	1
2.1.3. Procesamiento de imágenes	2
2.1.4. Sonido	3
2.1.5. Vídeo	3
2.2. Requisitos no funcionales	4
3. Análisis	5
4. Diseño	7
4.1. VentanaPrincipal	7
4.2. Lienzo	7
4.3. Ventanas internas	8
4.4. Figura	9
4.5. Gestión de eventos	9
4.6. Procesamiento de imágenes (I)	12
4.7. Procesamiento de imágenes (II)	15
4.7.1. Operador <i>LookUpOp</i>	15
4.7.2. Componente a componente	16
4.7.3. Primera operación píxel a píxel	18
4.7.4. Segunda operación píxel a píxel	20
4.8. Sonido	21
4.9. Vídeo	21
5. Implementación	22
5.1. Ventana principal	22
5.2. VentanaMultimediaImagen	22
5.3. VentanaMultimediaCamara	23
5.4. VentanaMultimediaVLCPlayer	23
6. Validación	24
6.1. Generales	24
6.1.1. Crear una nueva imagen	24
6.1.2. Abrir un nuevo archivo	24
6.1.3. Guardar un archivo	25
6.1.4. Ocultar/mostrar barras de herramientas	25
6.1.5. Acerca de	26
6.2. Gráficos	26
6.3. Barra de atributos	26
6.3.1. Selección de una figura	27
6.3.2. Editar los atributos de una figura	28
6.4. Procesamiento de imágenes	29
6.4.1. Duplicar	29
6.4.2. Modificar brillo	30
6.4.3. Filtro de emborronamiento	30
6.4.4. Filtro de enfoque	30
6.4.5. Filtro de relieve	31

6.4.6.	Contraste normal	31
6.4.7.	Iluminado	31
6.4.8.	Oscurecido	32
6.4.9.	Invertir colores	32
6.4.10.	Conversión a espacios RGB, YCC, GRAY	32
6.4.11.	Giro libre	33
6.4.12.	Escalado	33
6.4.13.	Tintado	33
6.4.14.	Ecualización	34
6.4.15.	Filtro sepia	34
6.4.16.	Umbralización	34
6.4.17.	Operador <i>LookupOp</i> basado en una función propia	35
6.4.18.	Operación de diseño propio: filtro violeta	35
6.5.	Sonido	35
6.5.1.	Reproducción de sonido	35
6.5.2.	Grabación de sonido	35
6.6.	Vídeo	36
6.6.1.	WebCam	36
7.	Mejoras para posteriores versiones	37

1. Descripción del sistema

Para la evaluación de la asignatura de *Sistemas multimedia* se quiere realizar una aplicación multimedia que permita gestionar gráficos, imágenes, sonido y vídeo. Se podrá crear, editar, procesar y visualizar contenido multimedia de distintos tipos.

Esta aplicación tendrá una ventana principal con diferentes barras de herramientas y botones que permitirán realizar las distintas operaciones sobre el contenido multimedia. Se intentará que la interfaz sea lo más intuitiva posible y facilitar así el uso de la aplicación al usuario final.

Con este fin, se realiza una aplicación que se llamará *Paint*.

2. Requisitos

2.1. Requisitos funcionales

2.1.1. Carácter general

- [RFCG-1] Creación de una nueva imagen en una nueva ventana.
- [RFCG-2] Abrir un fichero de imagen.
- [RFCG-3] Abrir un fichero de sonido.
- [RFCG-4] Abrir un fichero de vídeo.
- [RFCG-5] Guardar una imagen y sus figuras dibujadas.
- [RFCG-6] Ocultar las barras de herramientas.
- [RFCG-7] Visualizar las barras de herramientas.
- [RFCG-8] Consultar el nombre del programa, versión y autor.

2.1.2. Dibujo

- [RFD-1] Dibujar las siguientes formas geométricas con sus propios atributos independientes.
 - Línea.
 - Rectángulo.
 - Elipse.
 - Rectángulo con esquinas redondeadas.
- [RFD-2] El lienzo mantendrá todas las figuras que se vayan dibujando.
- [RFD-3] Elegir el color de trazo de dibujo.
 - Rojo.
 - Azul.
 - Negro.
 - Blanco.
 - Verde.
- [RFD-4] No llenar la figura.
- [RFD-5] Rellenar con color la figura.

- Rojo.
 - Azul.
 - Negro.
 - Blanco.
 - Verde.
- [RFD-6] Colorear con diferentes tipos de relleno la figura.
 - Degradado horizontal.
 - Degradado vertical.
 - Degradado diagonal.
 - [RFD-7] Escoger el tipo de línea con la que se pintará la figura.
 - Continua.
 - Discontinua.
 - Punteada.
 - [RFD-8] Seleccionar una figura dibujada.
 - [RFD-9] Editar una figura dibujada.
 - [RFD-10] Mover una figura dibujada.
 - [RFD-11] Consultar la lista de figuras dibujadas.
 - [RFD-12] Consultar los atributos de una figura dibujada.
 - [RFD-13] Asociar un grado de transparencia a una figura.
 - [RFD-14] Activar alisado de bordes de una figura.
 - [RFD-15] Desactivar alisado de bordes de una figura.
 - [RFD-16] Consultar los atributos de dibujo activo en un lienzo.

2.1.3. Procesamiento de imágenes

- [RFPI-1] Duplicar una imagen.
- [RFPI-2] Modificar el brillo de una imagen.
- [RFPI-3] Aplicar filtro para emborronar una imagen.
- [RFPI-4] Aplicar filtro para enfocar una imagen.
- [RFPI-5] Aplicar filtro de relieve a una imagen.
- [RFPI-6] Aplicar contraste a una imagen.
- [RFPI-7] Iluminar una imagen.
- [RFPI-8] Oscurecer una imagen.
- [RFPI-9] Extraer las bandas de una imagen.
- [RFPI-10] Invertir los colores de una imagen.
- [RFPI-11] Convertir una imagen a los siguientes espacios:

- RGB
 - YCC
 - GRAY
- [RFPI-12] Girar una imagen a cualquier ángulo.
 - [RFPI-13] Tintar una imagen.
 - [RFPI-14] Escoger nivel y color de tintado de una imagen.
 - [RFPI-15] Ecualizar una imagen.
 - [RFPI-16] Umbralizar una imagen en niveles de gris.
 - [RFPI-17] Escoger nivel de umbralización.
 - [RFPI-18] Aplicar filtro sepia a una imagen.
 - [RFPI-19] Aplicar filtro violeta a una imagen.
 - [RFPI-20] Aplicar filtro potenciador del color rojo (diseño propio).
 - [RFD-21] Aplicar filtro media de bandas azul y verde (diseño propio).
 - [RFPI-22] Aplicar filtro cosinosoidal.
 - [RFPI-23] El procesamiento de imágenes no se aplicarán sobre las figuras dibujadas.
 - [RFPI-24] Escalar una imagen.

2.1.4. Sonido

- [RFS-1] Reproducir audios.
- [RFS-2] Grabar sonidos.
- [RFS-3] Parar la grabación.
- [RFS-4] Parar la reproducción.

2.1.5. Vídeo

- [RFV-1] Mostrar la secuencia que capte la Webcam.
- [RFV-2] Capturar una imagen desde la Webcam.
- [RFV-3] Comenzar la reproducción de un vídeo.
- [RFV-4] Parar la reproducción de un vídeo.
- [RFV-5] Pausar la reproducción de un vídeo.

2.2. Requisitos no funcionales

- [RNF-1] Se mostrará en la barra de estado el pixel en el que está situado el ratón.
- [RNF-2] Se habilitarán en cada momento sólo los botones que pueden utilizarse.
- [RNF-3] Los botones tendrán asociados un *ToolTipText* para facilitar el uso del programa.
- [RNF-4] Al seleccionar una figura se activarán sus propiedades en la barra de atributos.
- [RNF-5] Si hay una figura seleccionada, al pulsar el ratón sobre en otro punto, deberá deseleccionarse la figura.
- [RNF-6] El título de una nueva ventana abierta será el nombre del fichero si se trata de una imagen abierta o guardada.
- [RNF-7] El título de una nueva ventana abierta será *Nueva* si se trata de una imagen creada por el usuario.
- [RNF-8] El título de una nueva ventana abierta será *Captura* si se trata de una captura captada de un vídeo o de la WebCam.
- [RNF-9] La *BoundingBox* será un rectángulo de color azul con cierto grado de transparencia, con línea discontinua.
- [RNF-10] Cuando se cree una imagen se lanzará un diálogo que permita elegir las dimensiones de la imagen.
- [RNF-11] Se indicará entre corchetes en el título de la ventana el espacio de color en el que está la imagen.
- [RNF-12] Cuando se extraigan las bandas de una imagen se indicará en el título de la ventana la banda en la que se encuentra.

3. Análisis

Se hará un análisis del problema y de las posibles soluciones, intentando que sea lo más independiente posible del lenguaje de programación. En el apartado de *Diseño* se especificarán detalles relacionados con la programación.

En primer lugar, se plantea cómo debería ser el área de dibujo, que almacenará los atributos con los que se pintará la siguiente figura. Con este fin se desarrollará una clase *Lienzo*. Esta clase será la encargada de pintar las figuras. Cada figura dibujada en el área de dibujo tendrá los atributos activos en ese momento en el lienzo, si bien sus atributos podrán modificarse a posteriori cuando se seleccione esta figura ([RFD-8:10]), así como consultar los atributos de cada una de las figuras dibujadas en ese lienzo ([RFD-11,12]).

La solución planteada durante el desarrollo de las prácticas no es muy flexible: la biblioteca `java.awt.Graphics` ofrece importantes limitaciones, ya que no permite que los objetos tengan propiedades de color, forma o grosor, entre otras. De esta forma, todas las figuras de un mismo lienzo se pintarán con los mismos atributos ([RFD-1:7,13:15]).

Para que cada figura tenga sus propios atributos se desarrollará una clase propia *Figura*, que almacenará cada uno de los atributos de la figura y permitirá que cada figura tenga los suyos propios: color, grosor, etc. Sin embargo, la línea, por ejemplo, no puede tener relleno, por lo que no tiene sentido guardar este atributo. Se desarrollará una clase propia para cada una de las figuras que pueden dibujarse: Línea, rectángulo, elipse, rectángulo redondeado.

Para intentar mejorar el diseño de esta clase, en lugar de que las formas hereden directamente de *Figura*, se dividirán en dos grupos: figuras rellenable y figuras lineales. De esta forma, las operaciones comunes relativas al relleno podrán realizarse en el ámbito de la figura rellenable, y no tener que implementarlas en cada una de las clases derivadas. Con esta jerarquía de clases, evitamos que figuras lineales tengan atributos que no deben tener, como por ejemplo, tipo de relleno, así como evitamos que en figuras con propiedades similares (elipse y rectángulo, por ejemplo) se repitan funciones o atributos.

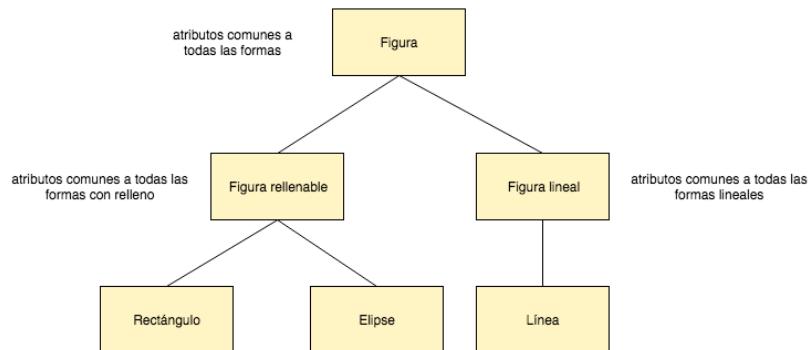


Figura 3.1: Esquema del diseño de la clase *Figura*

Para el procesamiento de imágenes se hará uso de las bibliotecas proporcionadas por el profesor, así como de las disponibles en *Java*. Esto permitirá cubrir los requisitos ([RFPI-1:18,22,23]). Los requisitos ([RFPI-19:21]) necesitará el desarrollo de clases propias. Se darán más detalles en el apartado de diseño.

En cuanto al manejo de eventos, se usará en muchas ocasiones las funciones que ofrece Java para la gestión de eventos (por ejemplo, para el uso de botones en la ventana principal). Sin embargo, para la comunicación entre el lienzo y la ventana principal será necesario crear una clase manejadora para la gestión de eventos relacionados con el lienzo. La existencia de esta clase permitirá por ejemplo informar de la falta de atributos necesarios para el dibujo de las figuras y poder lanzar mensajes de error desde la ventana principal (por ejemplo: no se ha seleccionado una forma de dibujo) o informar a la ventana principal de la creación de una nueva figura, para que aparezca en una lista desplegable de figuras seleccionables.

Para terminar, los requisitos de sonido y vídeo están cubiertos por las bibliotecas proporcionadas por el profesor, por lo que no se necesitará desarrollar ninguna clase adicional (**[RFV-1:5,RFS-1:4]**).

4. Diseño

Siguiendo la propuesta que se ha planteado en el análisis, se plantean las siguientes jerarquías de clases. Mientras la propuesta anterior intentaba ser independiente del lenguaje de programación, en este apartado se especifican aspectos concretos de la implementación en *Java*.

Se intentará dar una descripción general, pero para más detalles es preferible consultar los diagramas de clase. Muchas de las explicaciones se han obtenido de los guiones de prácticas y apuntes de la asignatura.

4.1. VentanaPrincipal

La clase `VentanaPrincipal` implementará la visión principal del programa. Heredará de `JFrame` y contendrá múltiples elementos `Swing`:

- **Barras de herramientas:** serán del tipo `JToolBar`, y se tratan de las barras superior, izquierda y derecha **[RFCG-6:7]**. Serán las que contendrán la mayoría de los demás elementos `Swing`.
- **Barra de estado:** se trata de un elemento `JPanel` que contendrá elementos del tipo `JLabel` **[RNF-1]**. El texto de estas etiquetas indicarán, por ejemplo, que una figura ha sido añadida.
- **Listas desplegables:** se utilizarán `JComboBox` para la selección de colores, tipo de relleno, tipo de línea y figuras.
- **Botones JToggleButton:** pueden quedarse seleccionados, por lo que se utilizarán para las formas agrupándose en un `ButtonGroup`.
- **Botones JButton:** En el caso de por ejemplo, los filtros, no tiene demasiado interés que un botón quede seleccionado, por lo que para la mayoría de las operaciones de procesamiento de imágenes se utilizará este tipo de botones **[RFCG-1:5]**.
- **Escritorio:** se trata de un elemento del tipo `JDesktopPane`. Este escritorio será aquel que contendrá las ventanas multimedia de distintos tipos.

4.2. Lienzo

La clase `Lienzo` heredará de `JPanel`. Esta clase tendrá asociados los atributos con los que se pintará la siguiente figura **[RFD-3:7,13:15]**, así como un vector de las figuras dibujadas **[RFD-2]**. El objeto de tipo `Lienzo` será también el que se encargue de pintar las figuras a través del método `paint`, así como de guardar la figura seleccionada y de pintar la bounding box.

Es necesario distinguir entre lienzo e imagen, aunque muchas veces se utilice la palabra lienzo con el mismo significado. Se creará la clase `LienzoImagen`, que heredará de `Lienzo`. Esta clase permitirá definir un área de dibujo, la imagen, que podrá ser creada (un rectángulo blanco de unas dimensiones determinadas, simulando un lienzo real) o abierta. En el método `paint` se llamará a `clip`, de forma que no se mostrará aquello que se pinte fuera de las dimensiones que tenga la imagen.

`Lienzo` hará uso de las funciones de gestión de eventos que ofrece *Java*.

- `formMousePressed`: cuando se presione el ratón en el lienzo se creará (en el caso de que estén seleccionados los atributos necesarios) una figura. Creará y añadirá una figura a la lista de `Figura` **[RFD-1]**.
- `formMouseDragged`: llamará a `updateShape` para actualizar la figura si seguimos arrastrando el ratón para aumentar el tamaño de la figura que se ha creado.

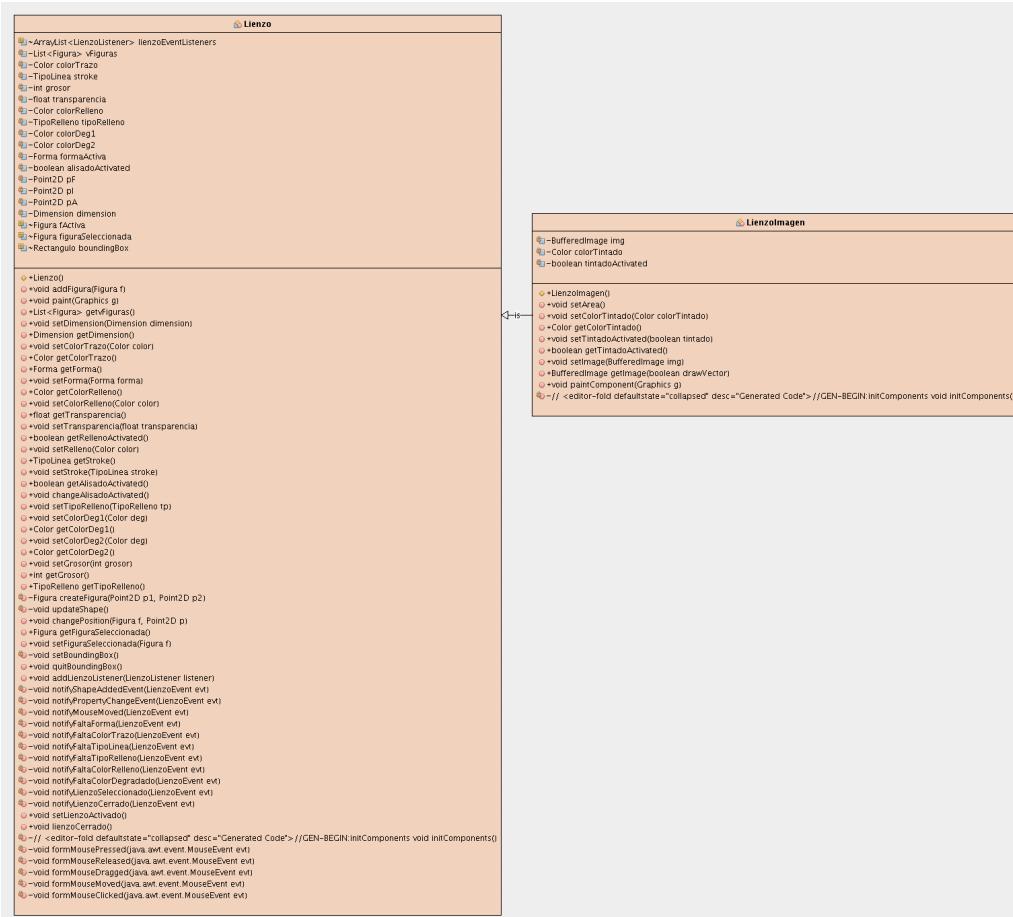


Figura 4.1: Diagrama de clases Lienzo

- **formMouseReleased:** finalizará la creación de una figura.
- **formMouseMoved:** servirá para mostrar en la barra de estado el pixel del lienzo en el que se encuentra el ratón. Para mostrarlo en la barra de estado será necesario lanzar un evento `LienzoEvent`.

4.3. Ventanas internas

Se definirá una clase `VentanaMultimedia` que heredará `JInternalFrame`. De esta clase derivarán diferentes ventanas en función del archivo multimedia que contengan.

- **VentanaMultimediaImagen:** se encargará de almacenar el contenido multimedia de tipo imagen. Tendrá un objeto de clase `LienzoImagen`, que será un rectángulo en blanco en el caso de que sea creada por el usuario, y una imagen en el caso de que sea una imagen abierta. Contendrá también un elemento de tipo `JScrollPane` para el caso en que la ventana no pueda contener la imagen completa.

Se hará uso de las funciones `formInternalFrameActivated` y `formInternalFrameDeactivated` para actualizar los atributos de la barra de herramientas cuando se seleccione una ventana imagen, así como para activar/desactivar los atributos de procesamiento de imágenes.

- **VentanaMultimediaCamara:** se encargará de almacenar las imágenes que pueden verse desde la WebCam así como las funciones encargadas de gestionar este contenido. Se hará uso de la función `formInternalFrameClosing` para cerrar la cámara cuando se cierre la ventana que muestra la imagen de la Webcam.
- **VentanaMultimediaVLCPlayer:** se encargará de almacenar vídeos así como las funciones encargadas de gestionar este contenido.

4.4. Figura

Se definirá la clase abstracta **Figura**. Esta clase almacenará los atributos de la figura a crear. Las clases **FiguraRellenable** y **FiguraLineal** heredarán de **Figura**. Aunque en este caso no hay más de una figura lineal, este esquema permite que en posteriores versiones se añadan curvas u otro tipo de figuras lineales.

De **FiguraRellenable** heredarán **Rectangulo**, **Elipse** y **RectanguloRedondeado**. De **FiguraLineal** heredará la clase **Línea**. Cada una de estas clases tendrá asociado un objeto **Shape**, además de los atributos de las clases de las que hereda:

- Línea: `Line2D`.
- Rectángulo: `Rectangle2D`.
- Rectángulo redondeado: `RoundRectangle2D`.
- Elipse: `Ellipse2D`.

Es esta jerarquía de clases la que permite que cada figura tenga sus propios atributos, así como la posibilidad de modificarlos **[RFD-8:10]**, llamando a los *setters* del objeto.

Esto permitirá consultar también los atributos de una figura, llamando a los distintos *getters* para actualizar los atributos de la barra de figuras. Como **Figura** es una clase abstracta, debe tener como mínimo un método abstracto. Los métodos abstractos de la clase serán los siguientes:

- **setLocation:** se encarga de cambiar la posición de una figura. En principio se pretendía aprovechar el método que proporcionaba **Graphics**, pero se ha optado por implementarlo en cada clase de forma manual, ya que para `Rectangle2D` y `Ellipse2D` no está disponible el uso de esta función.
- **updateShape:** actualiza una figura a partir de los puntos de inicio y final.

Más detalles acerca de los atributos asociados a la clase **Figura**, así como métodos para utilizarlos pueden consultarse en el diagrama.

4.5. Gestión de eventos

Se hará uso de las funciones de gestión de eventos que ofrece *Java* para los botones, **ActionEvent**, **FocusGained**, **FocusLost**,... Sin embargo, es necesario implementar las funciones para la gestión de eventos en el lienzo, de forma que podamos establecer una comunicación entre lienzo y ventana principal.

- **LienzoEvent:** representa un evento lanzado por un objeto **Lienzo**. Heredará de la clase **EventObject** y tendrá asociados gran parte de los atributos de la clase **Lienzo**.

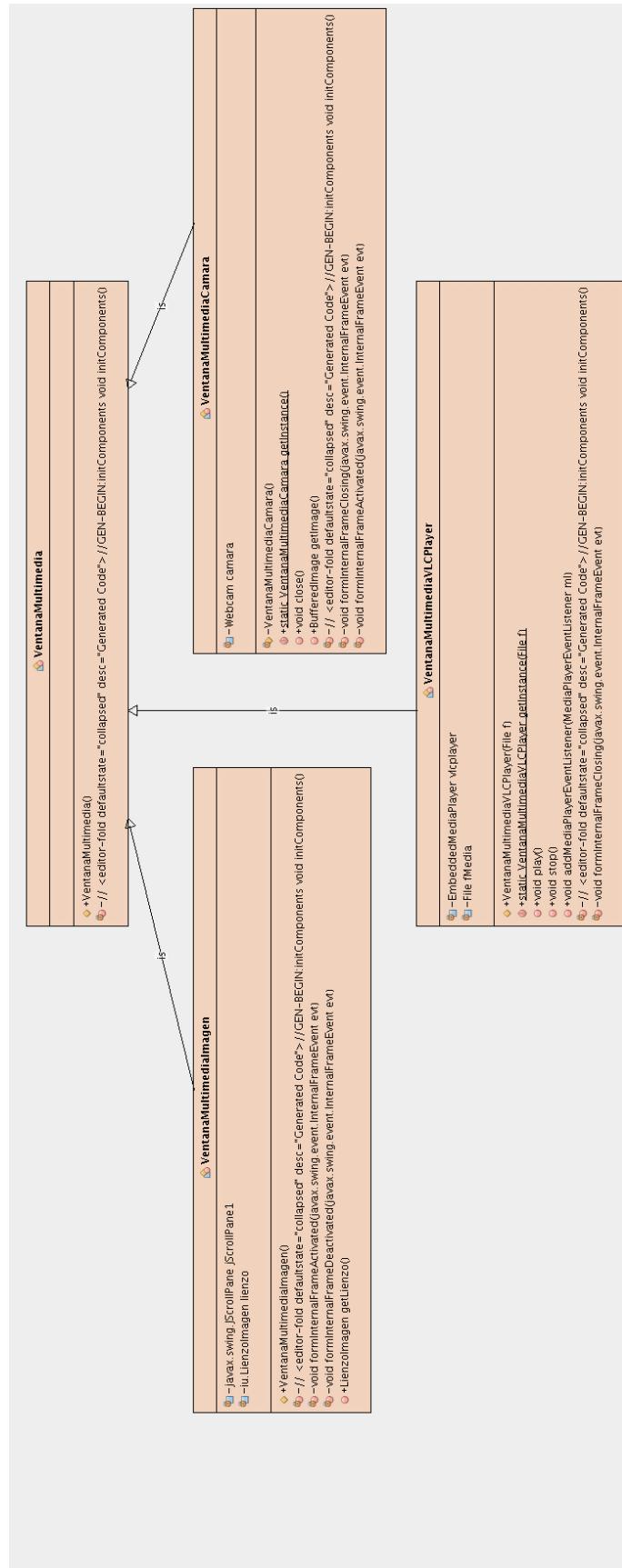


Figura 4.2: Diagrama de clases **VentanMultimedia**

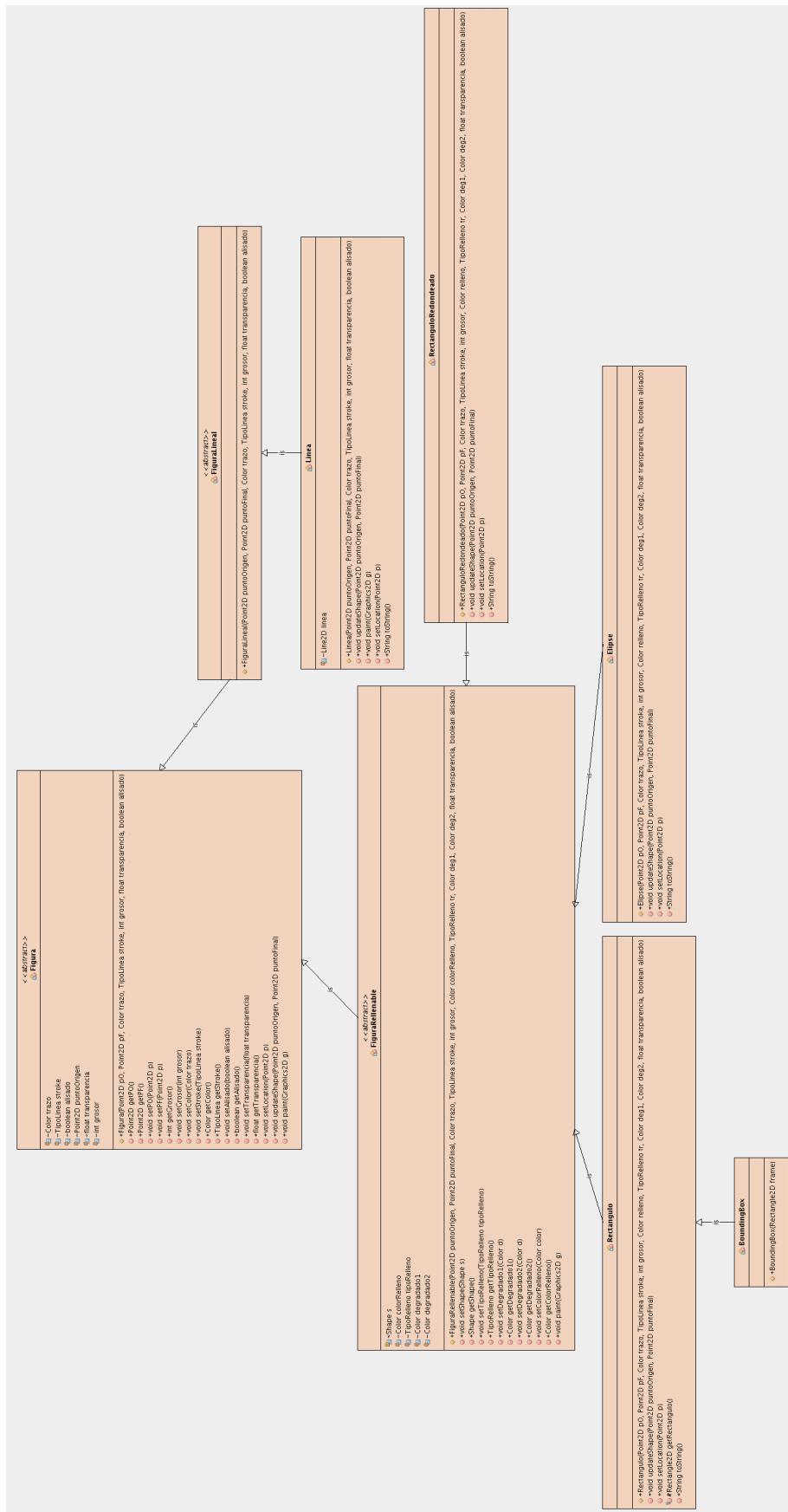


Figura 4.3: Diagrama de clases Figura

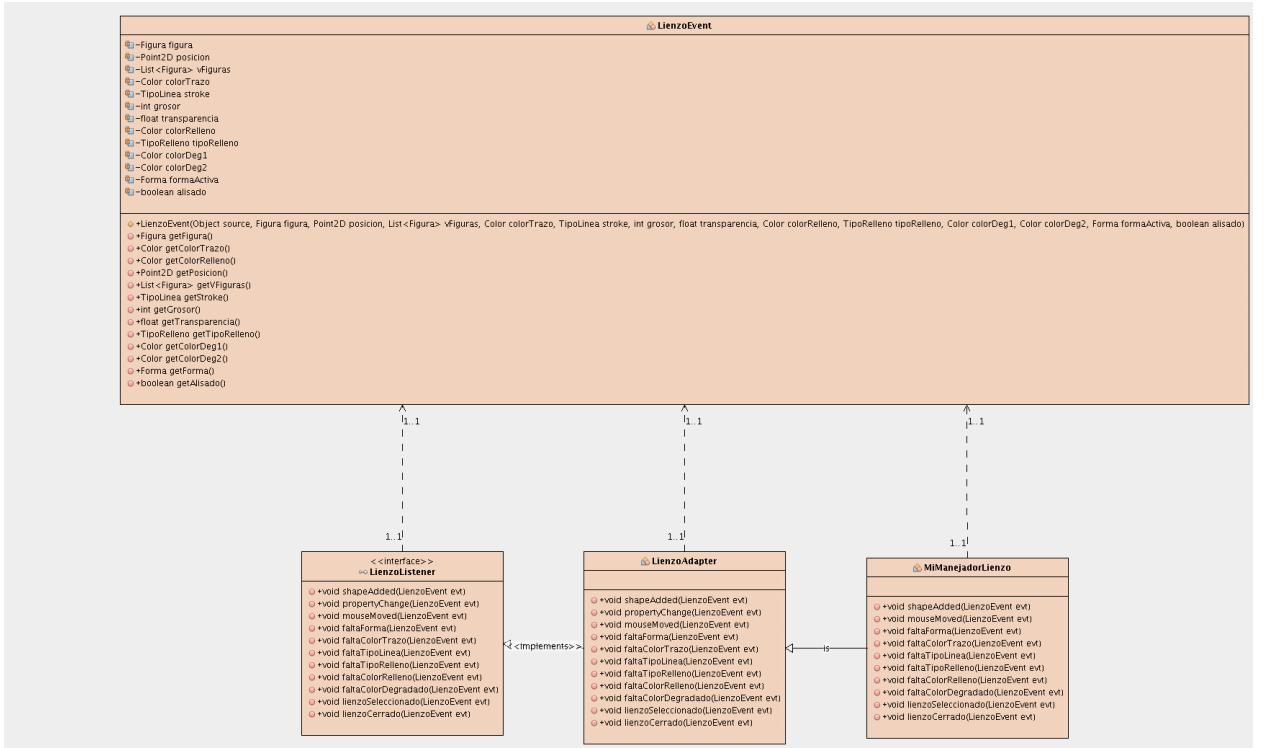


Figura 4.4: Diagrama de clases LienzoEvent, LienzoListener, LienzoAdapter y MiManejadorLienzo

- **LienzoListener:** interfaz que hereda de `EventListener`. Definirá las distintas causas que generan el evento. Por ejemplo, el movimiento del ratón.
- **LienzoAdapter:** clase que implementa la interfaz `LienzoListener` y de la que heredará `MiManejadorLienzo`, de ahí el nombre de *Adapter*.
- **MiManejadorLienzo:** hereda de `LienzoListener`. Será necesario asociar un manejador a cada uno de los lienzos. Esta clase estará implementada dentro de `VentanaPrincipal` y será la que modificará los elementos `Swing` cuando el lienzo lance un evento. Por ejemplo: si se añade una figura en el lienzo, la barra de estado de la ventana principal indicará que se ha añadido dicha figura [RFD-11,12,16].

4.6. Procesamiento de imágenes (I)

Este apartado expondrá algunos aspectos generales del procesamiento de imágenes.

Las imágenes serán del tipo `BufferedImage`, ofrecida por Java y se contará con la biblioteca `sm.image`, así como con las funciones ofrecidas por Java. En el lienzo existirá un método `getImage(boolean)`, que nos devolverá la imagen con las figuras que se han pintado sobre ella (si el parámetro es `true`) o solo la imagen en caso contrario. Como no es necesario que el procesamiento se aplique sobre las figuras dibujadas, en las funciones relacionadas con el procesamiento de imágenes el valor del parámetro será `false` [RFPI-23].

Para duplicar una imagen [RFPI-1] será necesario crear una nueva `BufferedImage`, de forma que no se trate de una referencia a la imagen original. Si no fuese una copia, los cambios de una se

aplicarían en la otra, de forma que no se estaría duplicando la imagen, sino abriendo la misma en una ventana diferente.

De forma general, para crear una copia de una imagen se utilizarán los métodos `getColorModel`, `copyData` y `isAlphaPremultiplied` de la clase `BufferedImage`, y por tanto ofrecidas por *Java*.

Para modificar el brillo de la imagen **[RFPI-2]** se utilizará la operación `RescaleOp`, disponible en la biblioteca de *Java*. Se hará una copia de la imagen en el lienzo activo, ya que el cambio no será definitivo hasta que se desactive el deslizador (`FocusLost`).

Para aplicar los filtros de emborronamiento, enfoque y relieve **[RFPI-3,4,5]** en primer lugar será necesario crear una máscara en función del filtro, haciendo uso de la clase `KernelProducer` disponible en la biblioteca `sm.image` y, una vez creada la máscara, haciendo uso de `ConvolveOp` (*Java*). Se utilizarán máscaras predefinidas:

- `TYPE_RELIEVE_3x3` para el filtro de relieve.
- `TYPE_BINOMIAL_3x3` para el filtro de emborronamiento.
- `TYPE_ENFOQUE_3x3` para el filtro de enfoque.

Para las operaciones de contraste, iluminado, oscurecido y negativo **[RFPI-6,7,8,10]**, al tratarse de una transformación sobre los niveles de color, se hará uso de `LookUpOp` (*Java*). Para ello, se creará un objeto `LookUpTable` haciendo uso de `LookUpTableProducer` (`sm.image`) con un tipo determinado en función de la operación de la que se trate:

- Contraste: `TYPE_SFUNCION`.
- Iluminado: `TYPE_LOGARITHM`.
- Oscurecido: `TYPE_POWER`.
- Negativo: `TYPE_NEGATIVE`.

Se utilizará este objeto `LookUpTable` para crear el objeto `LookUpOp`, y llamar al método `filter`. La imagen destino será la misma que la imagen fuente.

Para la extracción de bandas **[RFPI-9]**, crearemos una imagen por cada una de las bandas de colores (roja, verde y azul), y las mostraremos en `VentanaMultimediaImagen` separadas en niveles de gris.

- Se creará el modelo de color de la nueva imagen basada en niveles de gris haciendo uso de `CS_GRAY`.
- Se creará el nuevo raster a partir del de la imagen original.
- Crear una nueva imagen haciendo uso del modelo de color y el raster de la banda correspondiente.

La conversión a otros espacios de color **[RFPI-11]** se hará haciendo uso de `ColorConvertOp` y usando instancias de `ColorSpace`: `CS_LINEAR_RGB` para el espacio de color RGB, `CS_PYCC` para YCC y `CS_GRAY` para GREY.

Para el caso de el giro libre (en el que se incluye los casos particulares de 90° , 180° , 270°) y el escalado se utilizará la clase `AffineTransformOp` disponible en *Java* **[RFPI-12,24]**. Para el giro se utilizará la función `getRotateInstance` y para el escalado `getScaleInstance`, usando los parámetros deseados (giro/escala). El giro tendrá asociado un elemento `Slider` para ajustar el ángulo al que se desea girar la imagen.

Para el filtro sepia y para la umbralización [RFPI-18,16] ha sido necesario crear clases propias, **SepiaOp** y **UmbralizacionOp**. Como en el siguiente apartado se hace un análisis más detallado de las clases de creación propia, aquí tan sólo se detalla la formulación matemática asociada a estos operadores.

Se debe tener en cuenta que en la umbralización el valor **umbral** es la intensidad a partir de la cual el filtro realiza la transformación a color blanco. Por debajo de este umbral, la transformación será a color negro.

El nivel de umbralización se podrá determinar gracias a un **Slider** [RFPI-17].

```

for(int x=0; x<src.getWidth(); x++){
    for(int y=0; y<src.getHeight(); y++){
        int sepiaR, sepiaG, sepiaB;
        int srcR, srcG, srcB;
        Color colorSrc = new Color(src.getRGB(x, y));
        Color colorDest;
        //color del pixel
        srcR = colorSrc.getRed();
        srcG = colorSrc.getGreen();
        srcB = colorSrc.getBlue();
        //convertirlo a sepia
        sepiaR = (int) Math.min(255, 0.393*srcR+0.769*srcG
            +0.189*srcB);
        sepiaG = (int) Math.min(255, 0.349*srcR+0.686*srcG
            +0.168*srcB);
        sepiaB = (int) Math.min(255, 0.272*srcR+0.534*srcG
            +0.131*srcB);

        colorDest = new Color(sepiaR, sepiaG, sepiaB);
        dest.setRGB(x, y, colorDest.getRGB());
    }
}

```

Listing 1: Filtro sepia

```

for(int x=0; x<src.getWidth(); x++){
    for(int y=0; y<src.getHeight(); y++){
        int srcR, srcG, srcB;
        Color colorSrc = new Color(src.getRGB(x, y));
        //colores del pixel
        srcR = colorSrc.getRed();
        srcG = colorSrc.getGreen();
        srcB = colorSrc.getBlue();
        //intensidad como media de sus componentes
        int intensidad = (srcR + srcG + srcB)/3;
        if(intensidad>=umbral){
            dest.setRGB(x, y, Color.WHITE.getRGB());
        }
        else{//intensidad < umbral
            dest.setRGB(x, y, Color.BLACK.getRGB());
        }
    }
}

```

}

Listing 2: Filtro umbralización

Para el tintado **[RFPI-13]**, así como para el ecualizado **[RFPI-15]**, se utilizarán los operadores `TintOp` y `EqualizationOp`, proporcionados por `sm.image`. Simplemente será necesario crear los objetos y llamar al método `filter`. En el caso del tintado hay tres elementos a tener en cuenta:

- `JToggleButton` para activar/desactivar el tintado. Cuando se active el tintado, se habilitarán los siguientes dos elementos.
- `JComboBox` con objetos de tipo `Color`. Permitirá escoger el tipo de color con el que se quiere tintar la imagen.
- `Slider` para determinar el nivel de tintado**[RFPI-14]**.

4.7. Procesamiento de imágenes (II)

En este apartado nos centraremos en las operaciones de diseño propio. Las operaciones que heredan de `BufferedImageOp` implementan operaciones punto a punto sobre la imagen, distinguiendo entre:

- Operaciones píxel a píxel: para asignar el valor de una de las componentes del píxel se necesitará conocer el valor de otras componentes.
- Operaciones componente a componente: no se necesitará conocer el valor de las otras componentes para asignar el valor de las componentes de un píxel.

Para estas operaciones será necesario crear clases propias, en las que se sobrecargará el método `filter`. Este método tiene como parámetros `src`, `dest`, donde `src` es la imagen sobre la que se aplica el filtro y `dest` la imagen en la que se guardará el resultado de aplicar el filtro.

Como queremos que el filtro se aplique sobre la misma imagen de origen, de forma que las operaciones se apliquen de forma concatenada, `src` y `dest` serán la misma imagen.

En el caso de que se defina una transformación sobre los niveles de color haciendo uso de `LookUpOp`, no será necesaria la creación de una clase propia, pero sí de una función que devuelva el objeto `LookUpTable` que defina la transformación deseada.

De la misma forma que en el caso anterior, una vez creado el objeto `LookUpOp`, se utilizará el método `filter`, donde `src` y `dest` serán la misma imagen, para que el filtro se aplique sobre la misma imagen.

4.7.1. Operador *LookUpOp*

Se definirá la función $f(x) = 255|\cos(0,006x)|$, donde 0,06 será la velocidad angular **[RFPI-22]**. De esta forma, será posible predecir el comportamiento de la función, ya que el coseno es una función periódica. Como se puede ver en la gráfica a continuación, predecir el comportamiento de la función sin reducir la velocidad angular sería imposible a simple vista.

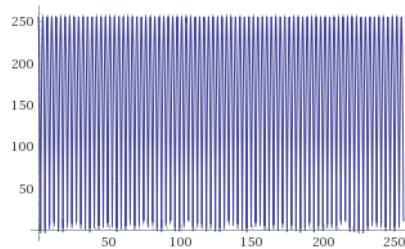


Figura 4.5: Gráfica $f(x) = 255|\cos(x)|$

Se escoge esa velocidad angular porque es lo suficientemente baja como para que no se complete un ciclo. A continuación se pinta la gráfica de $f(x)$. Se puede observar que se parece mucho a la función utilizada para invertir colores pero con $f(x)$ más altos. Se pintan las gráficas juntas para poder compararlas.

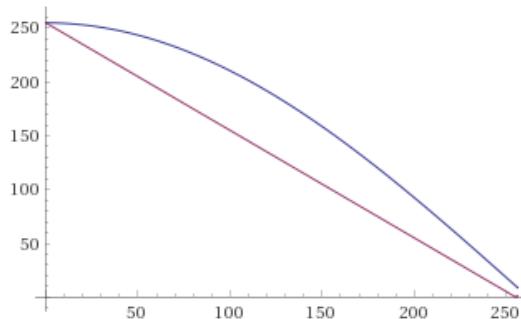


Figura 4.6: Comparativa gráfica operación definida (azul) y gráfica operación invertir colores (morada)

Por tanto el resultado esperado será una imagen similar a la que se obtendría invirtiendo los colores, con una mayor iluminación. A continuación puede comprobarse que el resultado obtenido es el esperado.



Figura 4.7: Comparativa aplicación filtro cosinusoidal y filtro negativo

4.7.2. Componente a componente

La función componente a componente modificará cada una de las componentes (R, G, B) de forma separada.

En este caso se ha implementado un filtro violeta. Para aplicar un tono morado a la imagen, se tendrá en cuenta que la mezcla de rojo y azul es la que da lugar al morado. Por tanto, para obtener el efecto deseado se reducirá el valor de la componente verde y se aumentará el valor de la componente roja y azul [RFPI-19].

Además, como se ha aumentado el valor de las componentes de forma significativa, habrá un aumento de la luminosidad en estos colores y de la intensidad de la imagen en general.

```
for(int x=0; x<src.getWidth(); x++) {
    for(int y=0; y<src.getHeight(); y++) {
        int destR, destG, destB;
        int srcR, srcG, srcB;
        Color colorSrc = new Color(src.getRGB(x, y));
        Color colorDest;
        //color del pixel
        srcR = colorSrc.getRed();
        srcG = colorSrc.getGreen();
        srcB = colorSrc.getBlue();
        //operacion componente a componente
        destR = (int) Math.min(255, 1.3*srcR);
        destG = (int) Math.min(255, 0.2*srcG);
        destB = (int) Math.min(255, 1.5*srcB);

        colorDest = new Color(destR, destG, destB);
        dest.setRGB(x, y, colorDest.getRGB());
    }
}
```

Listing 3: Operación componente a componente

El resultado de aplicar el filtro puede verse a continuación. Como se puede ver, se obtiene el resultado deseado.



Figura 4.8: Aplicación filtro violeta sobre Fry.jpg



Figura 4.9: Aplicación filtro violeta

4.7.3. Primera operación píxel a píxel

La función píxel a píxel modificará cada componente teniendo en cuenta los demás componentes, de forma que el valor de cada componente podrá ser una combinación de las otras.

La operación implementada en este caso es una especie de "filtro rojizo", aunque no llega a serlo como tal. El valor del color rojo se conservará tal y como está, mientras que los valores verde y azul serán una media de los otros tres, por lo que aquellos píxeles en los que estos valores sean predominantes tendrán un color grisáceo [RFPI-21].

Se espera que en la primera aplicación del filtro, la imagen tomará este color rojizo-anaranjado, pues habrá sido la única componente que habrá mantenido su valor. Sin embargo, al estar esta componente presente en la media que asigna los valores verde y azul, a través de las sucesivas aplicaciones del filtro estos dos valores convergerán al valor de la componente roja, convirtiéndose en un filtro blanco y negro.

```

for(int x=0; x<src.getWidth(); x++) {
    for(int y=0; y<src.getHeight(); y++) {
        int destR, destG, destB;
        int srcR, srcG, srcB;
        Color colorSrc = new Color(src.getRGB(x, y));
        Color colorDest;
        //color del pixel
        srcR = colorSrc.getRed();
        srcG = colorSrc.getGreen();
        srcB = colorSrc.getBlue();
        //operacion pixel a pixel
        destR = srcR;
        destG = Math.min(255, (srcR+srcG+srcB)/3);
        destB = Math.min(255, (srcR+srcG+srcB)/3);

        colorDest = new Color(destR, destG, destB);
        dest.setRGB(x, y, colorDest.getRGB());
    }
}

```

Listing 4: Operación pixel a pixel



Figura 4.10: Aplicaciones sucesivas operación píxel a píxel (I)

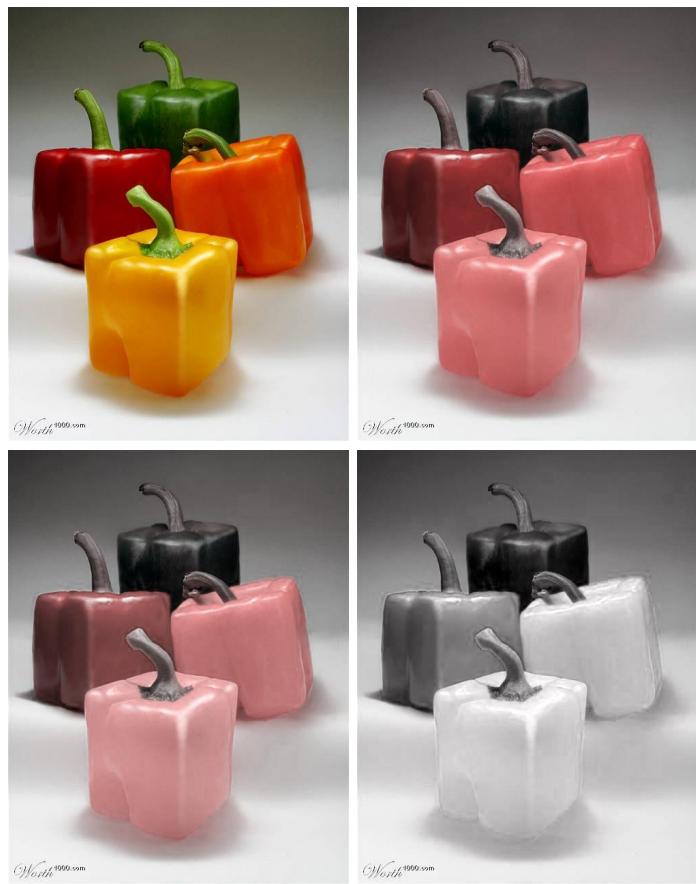


Figura 4.11: Aplicaciones sucesivas operación píxel a píxel (II)

4.7.4. Segunda operación píxel a píxel

Esta función es una variante de la primera, en la que se pretende que el filtro anterior no sea a través de las aplicaciones un filtro blanco y negro, sino un filtro que potencie el color rojo. Para ello, simplemente se elimina la componente roja de la media, de forma que los valores azules y verde no convergerán al valor de la componente roja [RFPI-20].

```
for(int x=0; x<src.getWidth(); x++) {
    for(int y=0; y<src.getHeight(); y++) {
        int destR, destG, destB;
        int srcR, srcG, srcB;
        Color colorSrc = new Color(src.getRGB(x, y));
        Color colorDest;
        //color del pixel
        srcR = colorSrc.getRed();
        srcG = colorSrc.getGreen();
        srcB = colorSrc.getBlue();
        //operacion pixel a pixel
        destR = srcR;
        destG = (srcG+srcB)/2;
        destB = (srcG+srcB)/2;

        colorDest = new Color(destR, destG, destB);
        dest.setRGB(x, y, colorDest.getRGB());
    }
}
```

Listing 5: Operación pixel a pixel

A continuación pueden verse ejemplos de la aplicación de este filtro, que ofrece el resultado esperado.

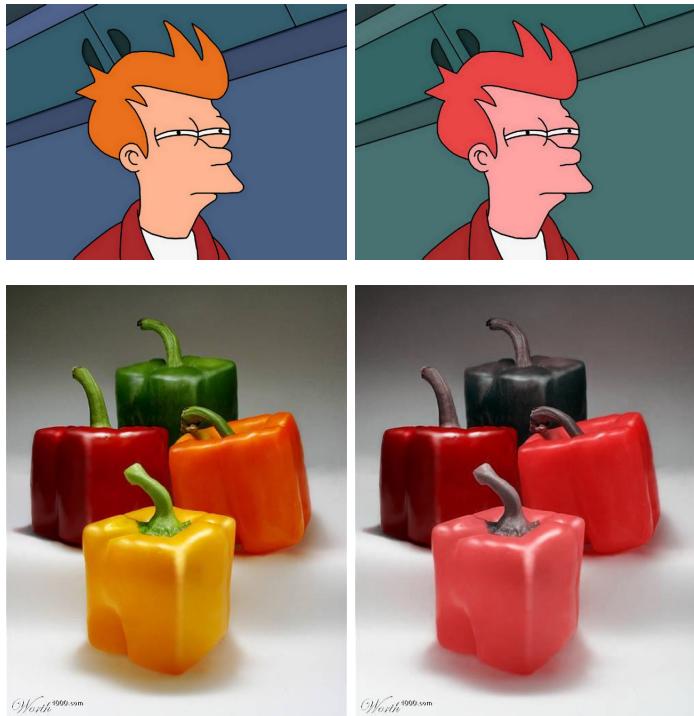


Figura 4.12: Aplicación filtro rojizo

4.8. Sonido

Se hará uso del paquete `sm.sound`, que incluirá dos módulos:

- **SMPlayer**: se harán uso de las funciones `play()`, `stop()` [RFS-1,4].
- **SMRecorder**: se harán uso de las funciones `record()`, `stop()` [RFS-2,3].

Cuando se inicialice un `player` o un `recorder` será necesario asociar un manejador que implementa `LineListener` haciendo uso de la función `addLineListener` que nos permita habilitar/deshabilitar los botones relacionados con el sonido en función de las acciones que realicemos. Por ejemplo, si se inicia la grabación, se debe deshabilitar el botón de inicio de grabación y habilitar el de detención [RNF-2].

4.9. Vídeo

Para usar la Webcam se hará uso de la biblioteca `github.sarxos.webcam.Webcam`. Para abrir la Webcam se usará el método `getInstance()`, que creará una instancia de `Webcam` e inicializará la `VentanaMultimediaCamara` que contendrá las imágenes que se obtendrán desde la cámara [RFV-1]. Para hacer una captura de pantalla se utilizará el método `getImagen()`, que devolverá una objeto del tipo `BufferedImage` [RFV-2].

Se hará uso de los métodos y las ventajas que ofrece la biblioteca `VLCj`, que permite acceder a formatos y códecs actualizados [RFV-3,4]. Se asociará un manejador para poder gestionar los eventos relacionados con el vídeo. La clase manejadora de eventos implementará `MediaPlayerEventAdapter` y, al igual que el manejador de sonido, habilitará/deshabilitará los botones relacionados con el vídeo [RFV-2][RNF-2].

5. Implementación

A continuación se muestra el resultado de la implementación de las clases diseñadas en el apartado anterior.

5.1. Ventana principal

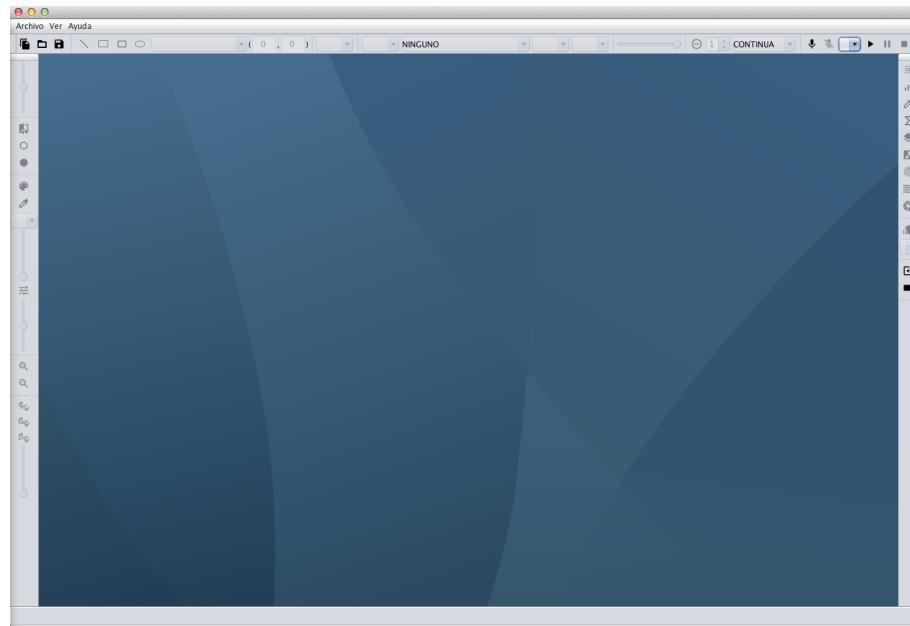


Figura 5.1: Vista VentanaPrincipal

5.2. VentanaMultimediaImagen



Figura 5.2: Vista VentanaMultimediaImagen

5.3. VentanaMultimediaCamara

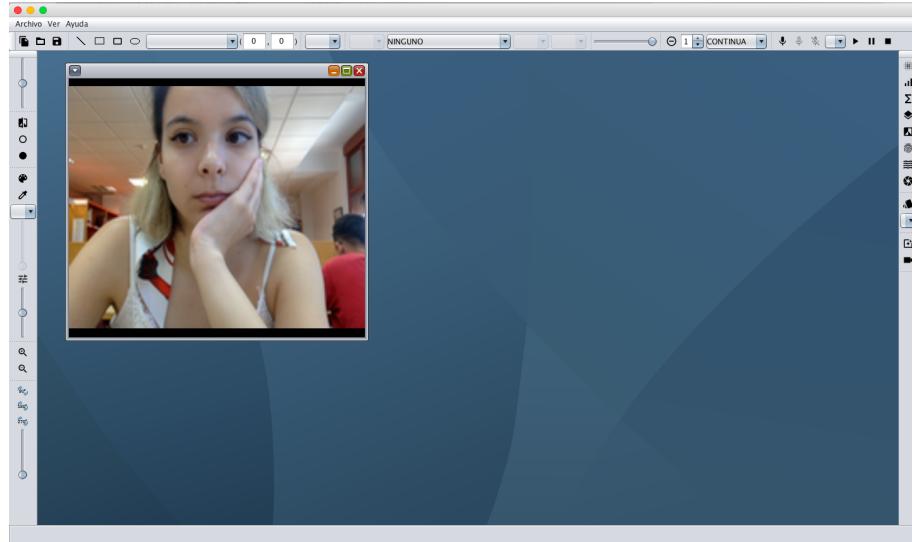


Figura 5.3: Vista VentanaMultimediaCamara

5.4. VentanaMultimediaVLCPlayer

No se ha podido obtener la vista. He tenido problemas para cargar las librerías en mi ordenador con sistema operativo *MacOS*, por lo que intenté probar el programa en un ordenador con un sistema operativo distinto. Se hizo la prueba en un ordenador *Windows* y la reproducción funcionaba, aunque sólo el audio: la terminal mostraba un error relacionado con la tarjeta gráfica.

6. Validación

En este apartado se comprobará que el programa implementado en base a la solución propuesta cumple con lo esperado.

6.1. Generales

6.1.1. Crear una nueva imagen

A la hora de crear una nueva imagen, el usuario podrá escoger las dimensiones del área de dibujo a través de ventanas emergentes. Para crear un archivo se puede tanto pulsar el botón habilitado para ello como acceder desde el menú **Archivo**.

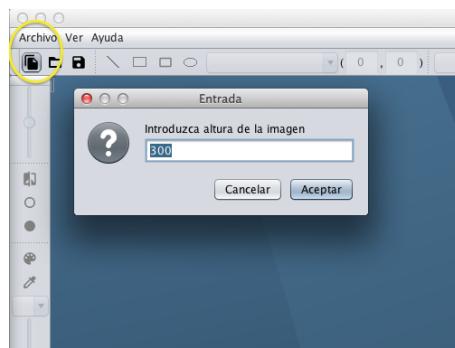


Figura 6.1: Diálogo para escoger las dimensiones de la nueva imagen

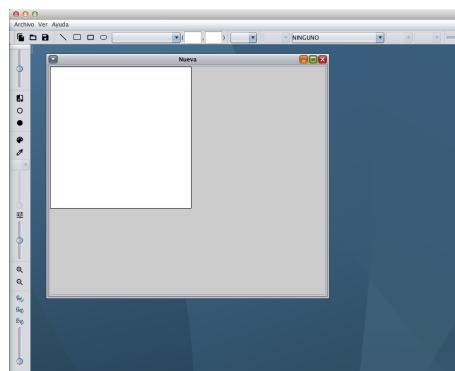


Figura 6.2: Crear nuevo archivo

6.1.2. Abrir un nuevo archivo

Para abrir un archivo se puede tanto pulsar el botón habilitado para ello como acceder desde el menú **Archivo**. Si se abre un archivo de sonido, se añadirá a la lista desplegable de audios, mientras que si se abre una imagen o un vídeo, se abrirá una ventana multimedia que contendrá el archivo deseado.

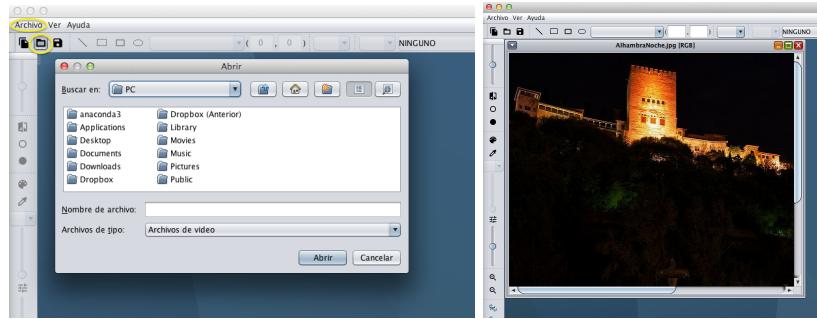


Figura 6.3: Abrir una imagen

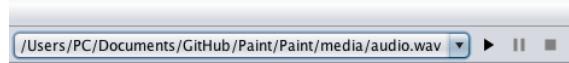


Figura 6.4: Abrir archivo de sonido

6.1.3. Guardar un archivo

Para guardar un archivo se puede tanto pulsar el botón habilitado para ello como acceder desde el menú **Archivo**.

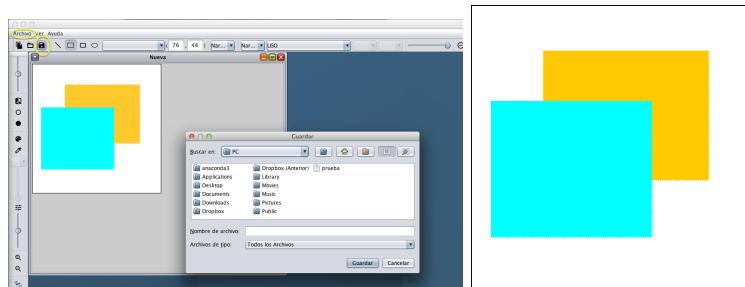


Figura 6.5: Guardar una imagen

6.1.4. Ocultar/mostrar barras de herramientas

Será posible ocultar/mostrar las barras de herramientas y de estado haciendo uso de las checkboxs disponibles en el menú **Ver**.



Figura 6.6: Ocultar barras de herramientas

6.1.5. Acerca de

El menú **Ayuda** contendrá la opción **Acerca de** que lanzará el diálogo con la información del programa.



Figura 6.7: Diálogo con el nombre del programa, versión y autor

6.2. Gráficos

6.3. Barra de atributos

La barra de atributos servirá para seleccionar las características para pintar la próxima figura y para modificarlas. Sin embargo, facilitará su uso al usuario inhabilitando aquellos elementos que no pueden utilizarse con las opciones seleccionadas.

Por ejemplo, no tiene sentido que las opciones de relleno estén activadas cuando estamos dibujando una línea.

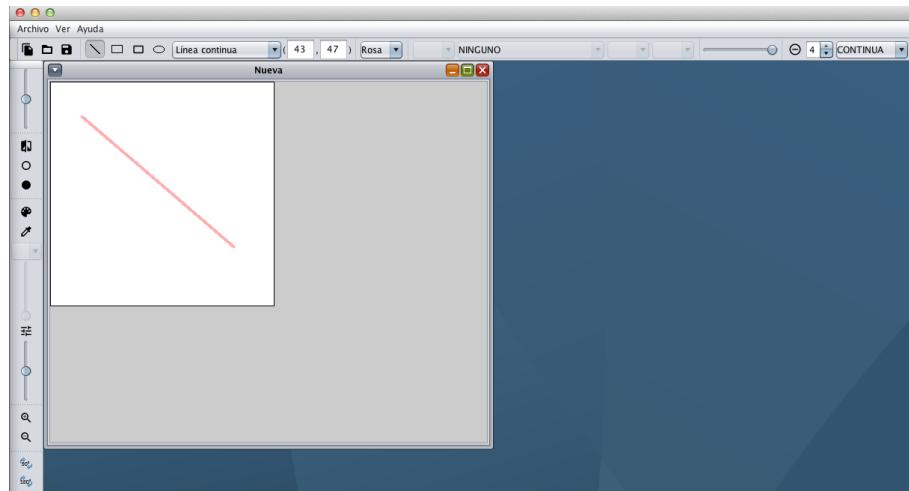


Figura 6.8: Los elementos se activan/desactivan en función de las características de la figura que seleccionamos

También se lanzarán mensajes de error si intentamos dibujar una figura sin haber seleccionado todos sus argumentos: por ejemplo, si indicamos que la figura está rellena pero no seleccionamos un color de relleno.

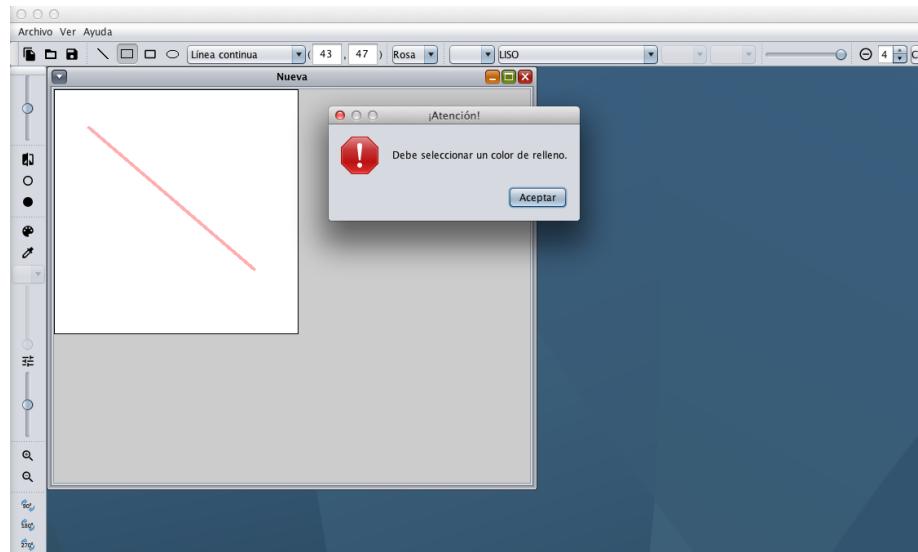


Figura 6.9: Mensajes de advertencia en el dibujado de las figuras

En este aspecto hay un problema: se ha tenido ciertos problemas con las listas desplegables. En particular, no hay una opción de no línea o línea nula, por lo que aunque al crear o abrir una imagen aparezca seleccionada la opción de línea continua, es necesario seleccionarla para que realmente se inicialice el objeto.

Algo parecido pasa con los colores: si se abre el ComboBox, aunque no se seleccione ningún color, si se vuelve a cerrar se seleccionará el color negro. Es necesario hacer clic sobre uno de los colores para que se pueda crear la figura.

6.3.1. Selección de una figura

Comprobamos que cuando se selecciona una figura, sus atributos se seleccionan automáticamente y correctamente en la barra de herramientas. Para seleccionar una figura, hay que seleccionar la ventana que contiene la imagen en la que está la figura deseada y buscarla en la lista desplegable de figuras.

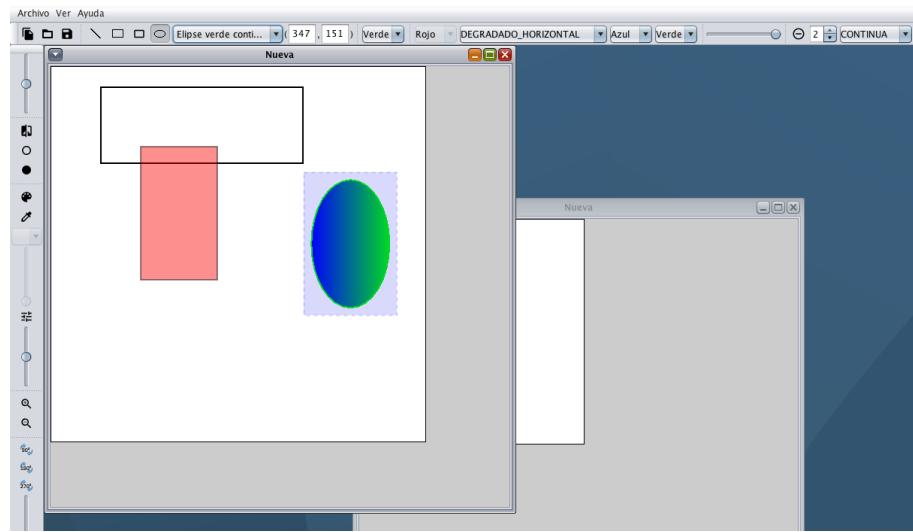


Figura 6.10: Selección de una figura

6.3.2. Editar los atributos de una figura

Se comprueba que pueden modificarse las propiedades de una figura. Para hacerlo, solo hay que seleccionar la figura a modificar y seleccionar los atributos deseados.

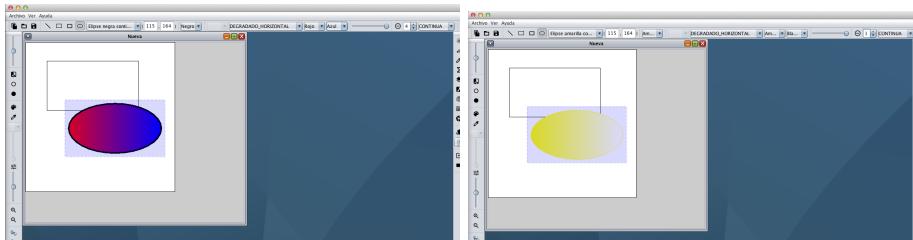


Figura 6.11: Cambio de atributos de una imagen

La posición es un atributo más de la figura. Como se puede ver, se puede seleccionar la figura y moverla. Para ello hay que introducir la posición en los campos de texto habilitados para ello (señalados en la imagen con el círculo amarillo).

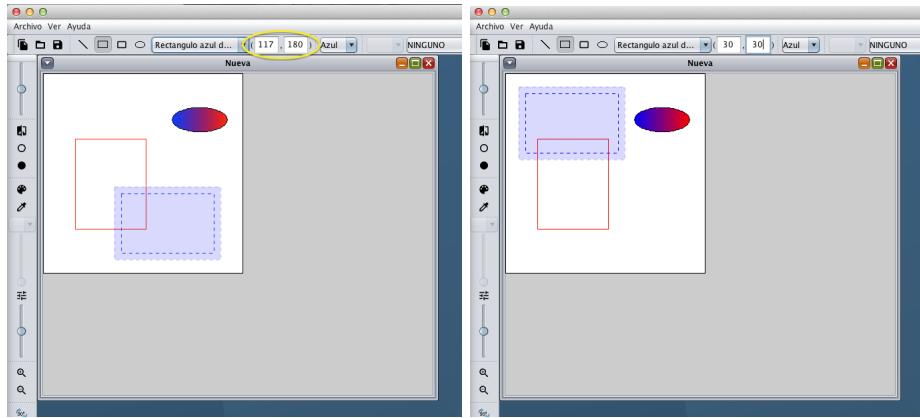


Figura 6.12: Mover figura

6.4. Procesamiento de imágenes

A continuación se probarán los diferentes filtros disponibles en la aplicación.

6.4.1. Duplicar

Se comprobará si la duplicación de la imagen se ha realizado correctamente, de forma que si se modifica una, los cambios no se aplican en la otra.



Figura 6.13: Duplicar imagen

6.4.2. Modificar brillo



Figura 6.14: Modificación del brillo

6.4.3. Filtro de emborronamiento

Se ha aplicado varias veces el filtro para que sea visible.



Figura 6.15: Aplicación filtro emborronamiento

6.4.4. Filtro de enfoque

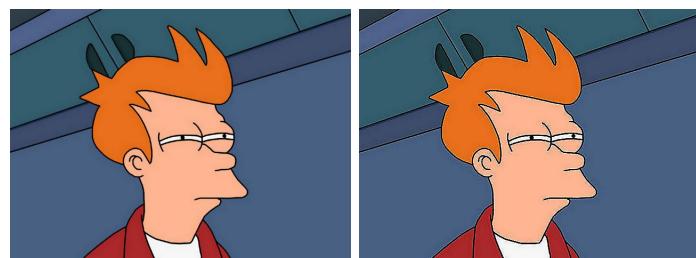


Figura 6.16: Aplicación filtro enfoque

6.4.5. Filtro de relieve



Figura 6.17: Aplicación filtro relieve

6.4.6. Contraste normal



Figura 6.18: Aplicación contraste

6.4.7. Iluminado

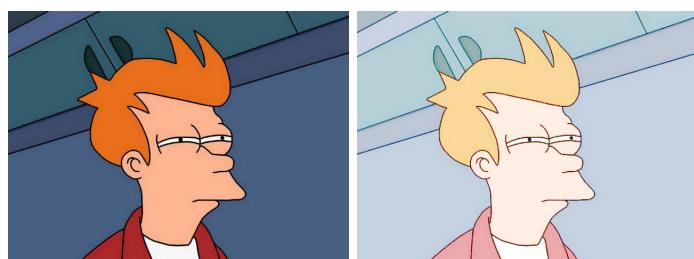


Figura 6.19: Aplicación iluminación

6.4.8. Oscurecido



Figura 6.20: Aplicación oscurecido

6.4.9. Invertir colores



Figura 6.21: Aplicación filtro negativo

6.4.10. Conversión a espacios RGB, YCC, GRAY

Para comprobar que la conversión se ha realizado correctamente, se extraen las bandas de colores y se comparan. En la imagen se pueden ver la tercera banda de una imagen en espacio de color RGB y YCC, respectivamente.



Figura 6.22:

6.4.11. Giro libre

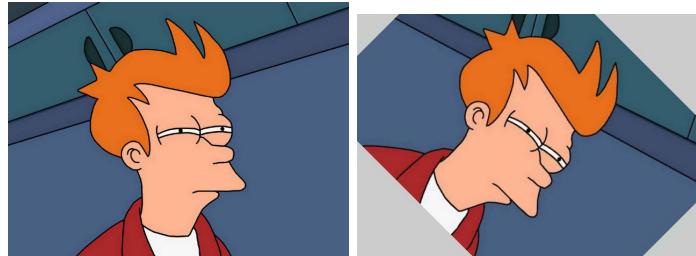


Figura 6.23: Giro libre

6.4.12. Escalado

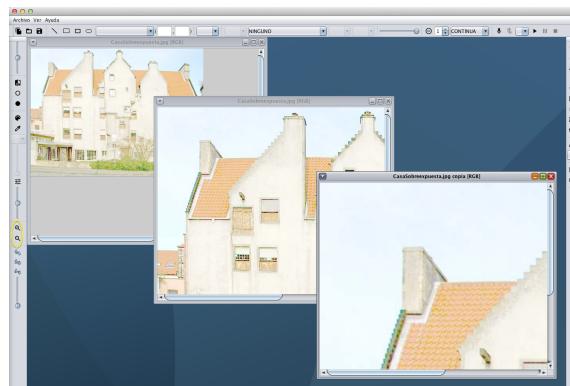


Figura 6.24: Escalado de la imagen

6.4.13. Tintado



Figura 6.25: Tintado de la imagen

6.4.14. Ecualización



Figura 6.26: Aplicación ecualización

6.4.15. Filtro sepia

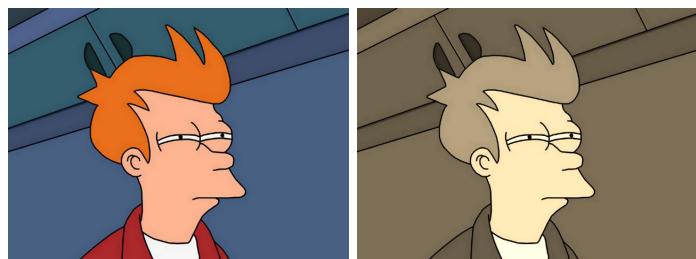


Figura 6.27: Aplicación filtro sepia

6.4.16. Umbralización

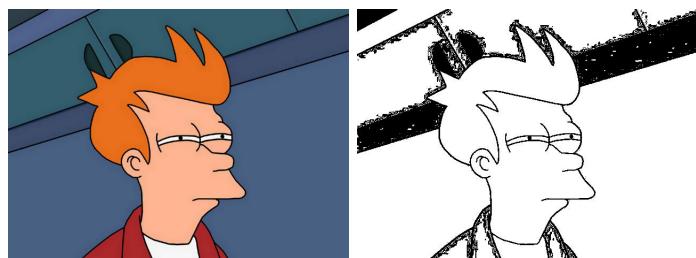


Figura 6.28: Umbralizar imagen

6.4.17. Operador *LookupOp* basado en una función propia



Figura 6.29: Operación cosinusoide

6.4.18. Operación de diseño propio: filtro violeta



Figura 6.30: Aplicación filtro violeta

6.5. Sonido

6.5.1. Reproducción de sonido

A continuación se muestran las funcionalidades del sonido. Como se puede comprobar, cuando se activa el sonido, no estarán activos los botones para el procesamiento de imágenes.



Figura 6.31: Abrir archivo de sonido

Como puede verse, si el archivo de audio no está reproduciéndose no podrán pulsarse el botón de parada. El botón de pausa no estará disponible para el sonido.

6.5.2. Grabación de sonido

El programa nos permitirá también grabar sonido. Una vez que se comience la grabación se deshabilitará el botón de grabar y se activará el de detener a grabación.

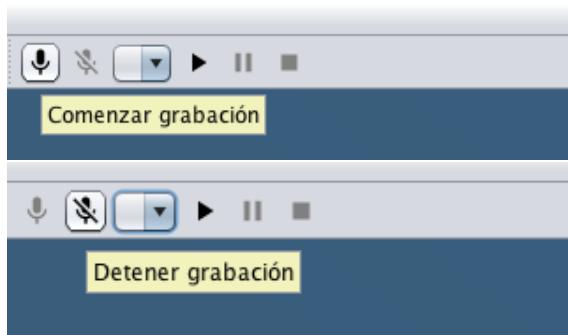


Figura 6.32: Grabar/Detener grabación

Una vez que se haya detenido la grabación, se podrá guardar. Será necesario escribir la extensión cuando se guarde el archivo (au o wav). Un vez guardada la grabación, se añadirá a la lista de audios y se podrá reproducir.

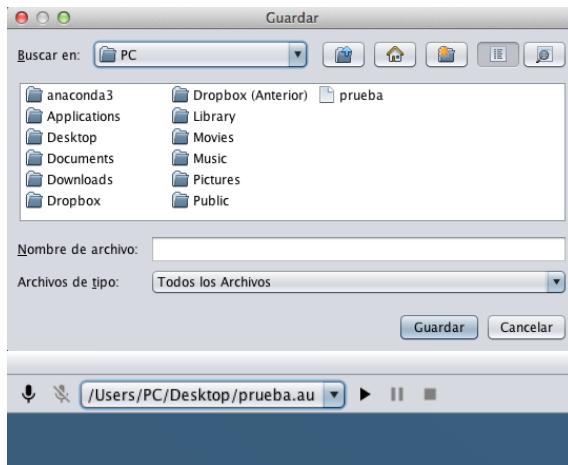


Figura 6.33: Guardar grabación

6.6. Vídeo

A continuación se muestran las funcionalidades del vídeo y la cámara. Como se puede comprobar, cuando se activa la WebCam, no estarán activos los botones para el procesamiento de imágenes.

6.6.1. WebCam

En primer lugar, comprobamos que puede abrirse correctamente la cámara. Una vez hecho esto, se comprobará si se realizan las capturas de pantalla correctamente. Si no hay ventana de cámara abierta, no se podrá sacar ninguna captura de pantalla.

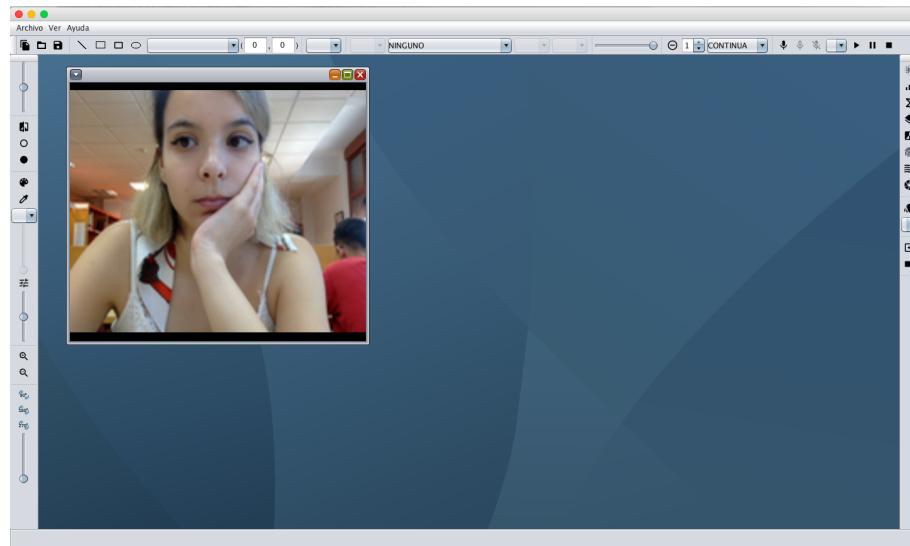


Figura 6.34: Webcam abierta

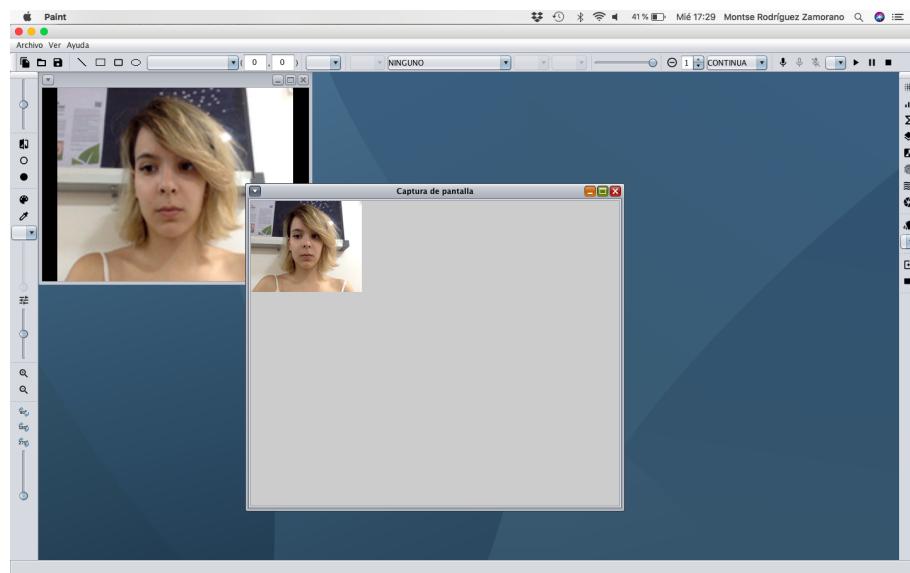


Figura 6.35: Captura de pantalla

7. Mejoras para posteriores versiones

- El botón de reproducir no se desactiva aunque esté programado en los manejadores de audio y de vídeo.
- Mejorar las listas desplegables de relleno y tipo de línea añadiendo dibujos o iconos.
- Buscar una alternativa para el vídeo disponible para todos los sistemas operativos.

Referencias

- [1] Apuntes de la asignatura y guiones de prácticas.
<https://decsai.ugr.es>
- [2] Documentación Java
<https://docs.oracle.com/javase/7/docs/api/java.awt/>
- [3] Tutoriales para el uso de vlcj
<http://capricasoftware.co.uk/projects/vlcj-4/tutorials>
- [4] Código para ComboBoxs personalizados [https://docs.oracle.com/javase/8/docs/api/javax/swing>ListCellRenderer.html](https://docs.oracle.com/javase/8/docs/api/javax/swing/ListCellRenderer.html)