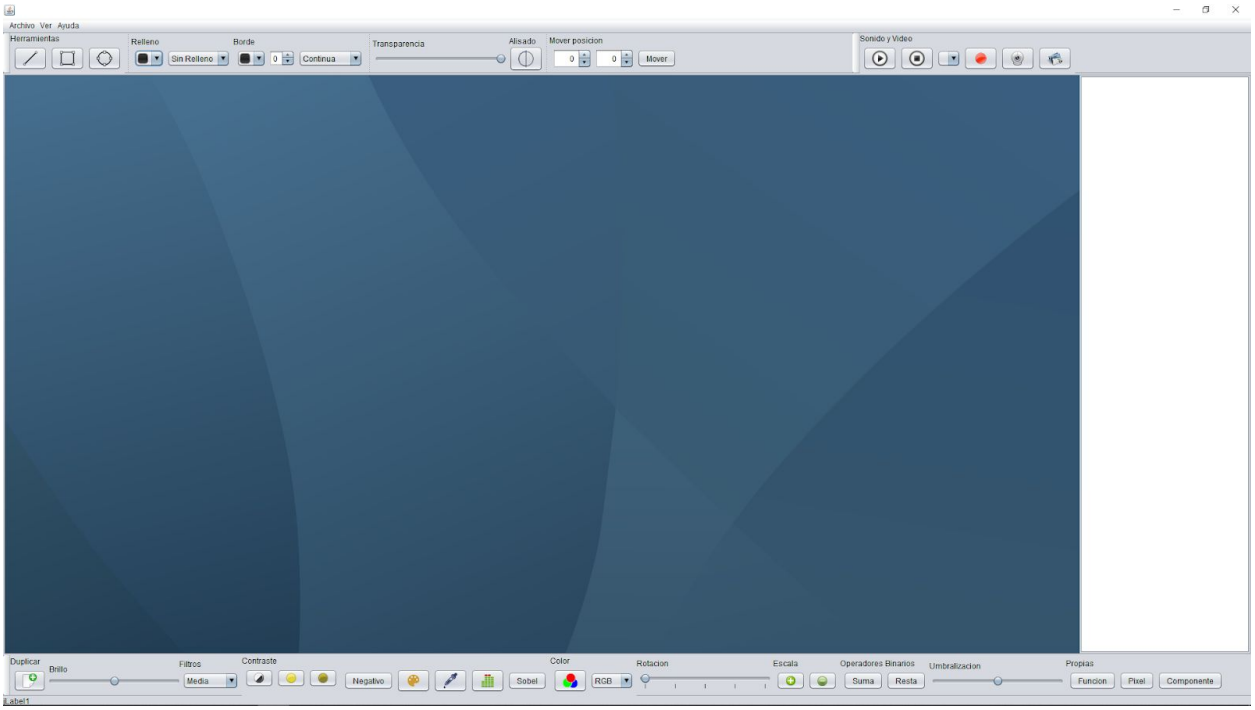
Paula Ruiz García

Documentación - Sistemas Multimedia



(*)Aclarar que a la hora de la estética no me gustaban que los botones de guardar, nuevo y abrir estuvieran en las barras de herramientas, por lo que para acceder a ellos se tendrá que hacer mediante los botones dentro del menú Archivo situado en la barra de navegación.

Buscamos realizar una aplicación multimedia que permita gestionar de forma integral diferentes tipos de medios como gráficos, imágenes, sonido y vídeo. Sobre cada tipo de medio se podrán llevar a cabo diferentes tareas que, en función del medio, abarcaran desde la creación y/o captura hasta la edición, reproducción y procesamiento. Para ello, la aplicación contará con un conjunto de menús y barras de herramientas que permitirán llevar a cabo dichas tareas.

Para un mejor análisis de los problemas y sus respectivas soluciones, dividiremos el contenido en los cinco grandes bloques principales de nuestro código. Dentro de cada uno tendremos sus respectivos requisitos, análisis, diseño y codificación, ya que los bloques son parcialmente independientes unos de otros.

Carácter General - Interfaz de Usuario

Nuestra aplicación permite trabajar de forma integrada, con los todos los medios estudiados a lo largo del curso. Para ello, nuestra aplicación dispondrá de un escritorio central en el que se podrán alojar ventanas internas de diferentes tipos en función del medio.

Requisitos Funcionales

RF 1.1 - La aplicación contará con un menú de carácter general, en el cual se incluirán los botones que comentaremos a continuación, y a cada herramienta se le asociará su respectivo icono y “ToolTipText”.

RF 1.2 - En la barra de herramientas se dispondrá del menú “Archivo” que contendrá los botones de “Nuevo”, “Abrir”, y “Guardar”.

RF 1.3 - La aplicación contará con un botón “Nuevo” que permitirá crear una imagen, la cual aparecerá en su propia ventana interna. Además el usuario podrá indicar el tamaño del lienzo.

RF 1.4 - La aplicación contará con un botón “Abrir”, que permitirá seleccionar un fichero de imagen, sonido o video. Dependiendo del tipo de archivo este se mostrará en su tipo de ventana predeterminado (imagen y video) o en la lista de reproducción de la interfaz (sonido). Si el tipo de fichero no está dentro de los permitidos se lanzará un diálogo que informe del problema.

RF 1.5 - La aplicación contará con un botón “Guardar”, el cual lanzará un diálogo que permitirá guardar la imagen de la ventana que esté seleccionada. Se podrán almacenar, tanto imágenes como figuras, en cualquiera de los formatos reconocidos por Java, obteniendo dicho formato a partir de la extensión indicada por el usuario. Asociado al diálogo guardar definiremos filtros para que solo muestre extensiones correspondientes a los formatos admitidos.

RF 1.6 - En la barra de herramientas se dispondrá del menú “Ver”, que permitirá ocultar o visualizar las distintas barras de herramientas que usamos.

RF 1.7 - En la barra de herramientas se dispondrá del menú “Ayuda”, que contendrá el botón “Acerca de” el cual lanzará un diálogo con el nombre del programa, versión y autor.

RF 1.8 - Existirá un escritorio central en el cual poder trabajar en tiempo real todas las funcionalidades al mismo tiempo.

RF 1.9 - Dentro del escritorio, se podrán alojar las distintas ventanas internas asociadas a nuestro diferentes medios, entre ella una para Imágenes y Figuras, otra para el Video y otra para la Cámara. En mi caso, cada ventana interna será independiente de las demás, aunque todas heredarán de JInternalFrame.

Diseño

Para el momento de alojar distintas ventanas dentro del escritorio, mi solución ha sido la creación de distintas clases que heredarán de *JInternalFrame*, y a las que se le asignará cada una su función.

Además, hemos creado un nuevo paquete en nuestra biblioteca, el cual contiene varias clases auxiliares, que nos ayudan a desarrollar nuestra interfaz.

En este paquete podemos ver dos clases *Lienzo*, una para dibujo y otra para imágenes. En la de dibujo se alojan todas las variables necesarias para el pintado de las formas y sus propiedades adecuadas. En la de imágenes, estan los métodos necesarios para insertar imágenes en el lienzo y poder dibujar formas en ellas, además de poder guardarlas después.

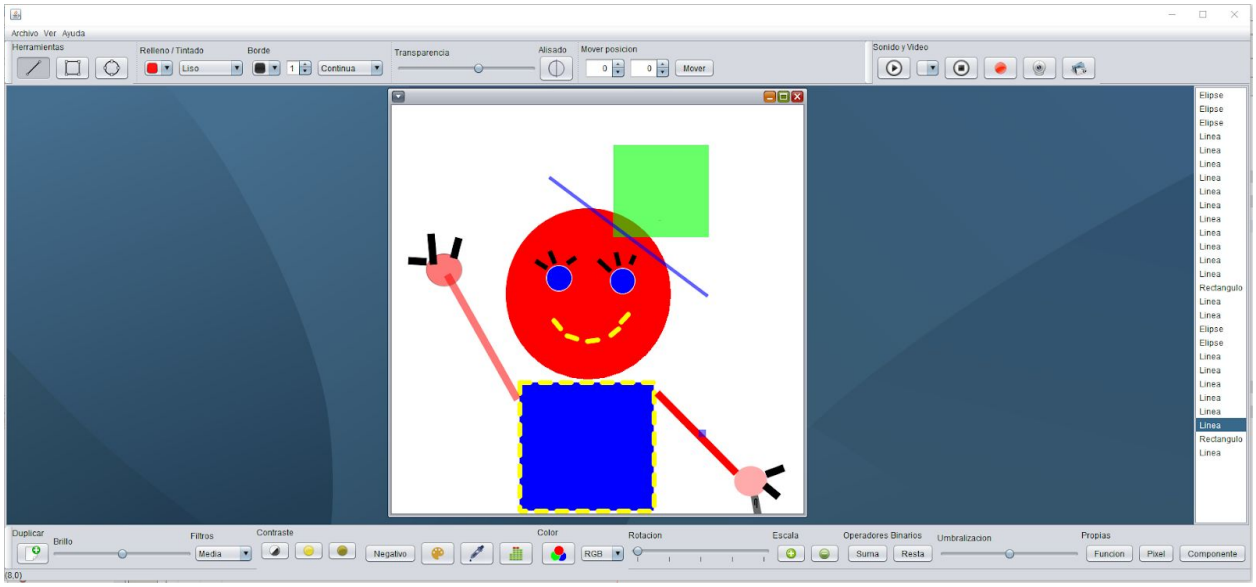
A continuación, podemos ver una clase *Lista*, la cual hereda de *AbstractListModel*, y nos permite adaptar la anchura de la lista según los elementos que contengan. Además, es la que nos permite añadir en una lista propia las figuras que tiene cada lienzo en su vector de figuras y así las podrá mostrar en la interfaz. Con esta solución el usuario podrá seleccionar la figura que busca editar, mediante la lista, y podrá editarla según sus valores, que están siendo mostrados en la barra de herramientas de dibujo. (*La parte de la clase propia de la Lista la implemente primariamente junto a mi compañero Manuel Ortiz, aunque es posible que ambos hayamos hechos varios cambios desde ese momento*)

También podemos encontrar una clase de tipo *Enum* que llamaremos *Herramientas,* que almacena el nombre de los distintos tipos de figuras que podemos dibujar.

Y la última clase que podemos ver es el *ColorRender*, que hereda de *JPanel* e implementa *ListCellRenderer*, que es el que nos ayuda a crear el desplegable de los colores, cogiendo los asignados y colocando el color como fondo del botón para mostrarlo. (*Para este apartado estuve buscando en internet en algunos proyectos de github y me ayudaron algunos compañeros porque no termine de entender como iba el Render, al final conseguí entenderlo*)

Dibujo

La aplicación permitirá dibujar diferentes formas geométricas con diferentes atributos sobre una imagen. Para ello, se incorporará una barra de herramientas que dé acceso a todos los elementos necesarios para poder dibujar, incluyendo formas y atributos.



Requisitos

RF 2.1 - El lienzo mantendrá todas las figuras que se vayan dibujando.

RF 2.2 - Cada figura tendrá sus propios atributos, independientes del resto de formas.

RF 2.3 - Cuando se dibuje la forma por primera vez, ésta usará los atributos que estén activos en ese momento.

RF 2.4 - Deberá de existir un panel en el que se muestre la lista de figuras que hay en el lienzo activo. Dicha lista tendrá que actualizarse cada vez que cambiemos de ventana y, dada una ventana de dibujo, cada vez que añadamos una nueva figura.

RF 2.5 - El usuario podrá seleccionar cualquier figura a través del panel de figuras. La figura seleccionada deberá indicarse mediante un rectángulo discontinuo a su alrededor.

RF 2.6 - Al seleccionar una figura, deberán de activarse sus propiedades en la barra de dibujo, así como deseleccionarse la forma de dibujo que estuviese activa. Por otro lado, si se pulsa el botón de una forma de dibujo, automáticamente deberá de deseleccionarse la forma seleccionada.

RF 2.7 - Para la figura seleccionada, se podrán editar sus atributos, esto es, se podrá modificar cualquiera de sus propiedades sin más que cambiarla en la barra de herramientas.

RF 2.8 - Se podrá cambiar la localización de la figura seleccionada. Para ello, se incluirá como información en la barra de dibujo la coordenada (x,y) en la que se encuentra localizada la forma. El usuario podrá editar esa posición y, al hacerlo, la figura deberá moverse a la localización indicada.

RF 2.9 - La aplicación deberá permitir dibujar, al menos, las siguientes formas geométricas: Línea recta, Rectángulo y Elipse.

Análisis

A la hora de realizar el dibujo, nos damos cuenta que con las bibliotecas que Java nos proporciona no va a ser posible el cumplir nuestros requisitos, ya que por mucho que usemos *Graphics2D* y las distintas clases de las propiedades, si lo montamos todo en una única clase Lienzo no va a ser posible que cada figura tenga sus propiedades individuales del resto de figuras del lienzo, porque todas siempre se pintarán con los mismos atributos, y se estarían pintando de nuevo cada vez que cambiasemos de clase.

Diseño

Soluciones

Para solucionar el problema presentado en el análisis sobre las figuras y sus atributos hemos encontrado una solución la cual trata sobre realizar clases propias para cada una de las figuras, en las cuales añadimos los atributos a cada una de ellas.

Como buscamos que todo sea más fácil, lo primero que hemos hecho ha sido crear una clase abstracta a la cual llamaremos *Figura*, la cual contendrá todas propiedades que debe presentar una figura. Para ello definiremos todos los métodos necesarios sobre los que operan las propiedades de los dibujos de manera que podamos modificarlas a nuestro gusto.

Además esta clase también contendrá una serie de métodos abstractos que definiremos según la figura que queramos dibujar en cada momento y sus propiedades, eston son:

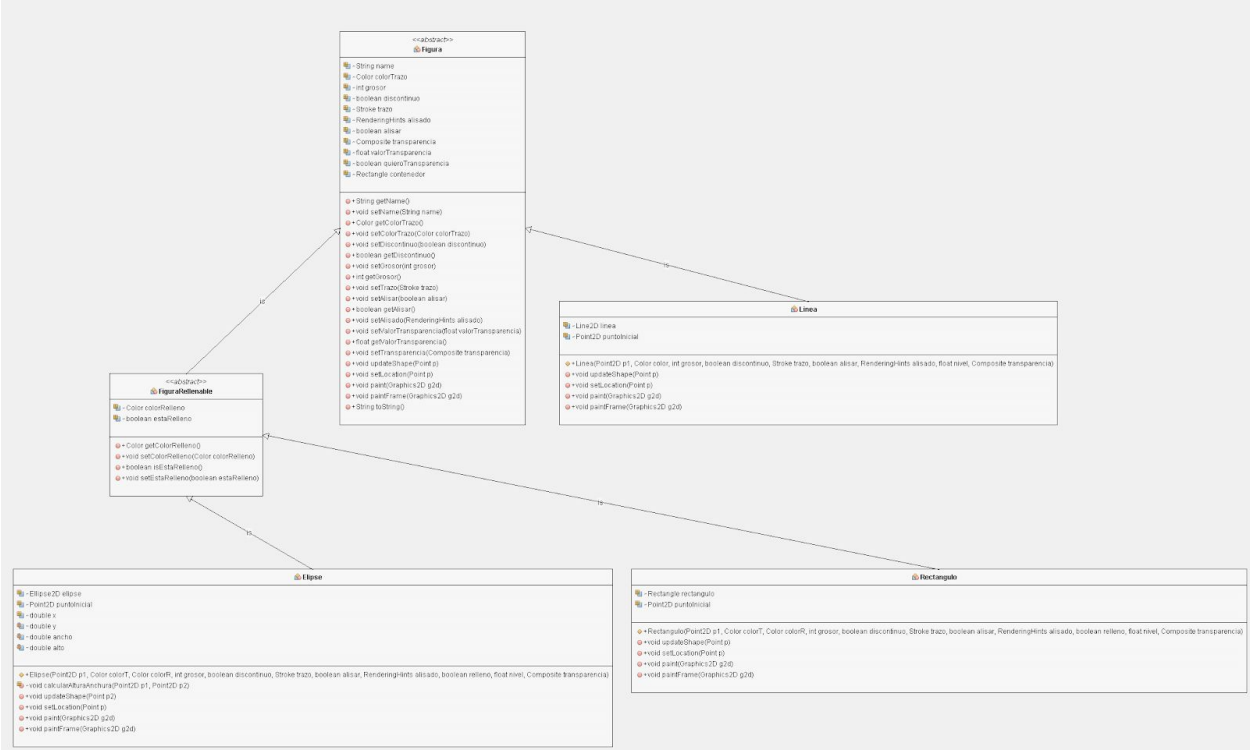
- *updateShape* para asignar los puntos que ocupa la figura en el lienzo.
- *setLocation* para modificar su posición si el usuario lo pide.
- *paint* para pintar la figura con las propiedades elegidas.
- *paintFrame* para pintar su rectángulo selector.

Existirá también una bifurcación para aquellas figuras que sean rellenables que será una nueva clase abstracta que heredara de *Figura*, y la llamaremos *FiguraRellenable*. Esta clase la asociaremos a todas aquellas figuras que son cerradas y por tanto pueden ser rellenadas. Y con ello conseguimos un mayor agrupamiento de clases, una mejor herencia y evitar ese “copia y pega” que siempre queda demasiado feo en códigos.

A continuación, es hora de ponernos a crear nuestras propias clases para las distintas figuras que vamos a usar. Por ello crearemos dos clases que heredarán de *FiguraRellenable,* que son *Rectangulo* y *Elipse*, ya que son figuras cerradas. Y luego, crearemos la clase *Línea*, que hereda directamente de *Figura* ya que no es posible rellenarla.

Para la creación de la formas hemos seguido usando las clases *Shape* que nos proporciona Java, ya que nos solucionan el problema de ponernos a implementar nosotros las distintas creaciones de las figuras. Por lo que cada figura tendrá su equivalente *Shape* como una variable más dentro las clases propias.

Herencia de clases



Procesamiento de Imágenes

El procesamiento de imágenes, requiere la incorporación de un tipo específico de ventana interna (el mismo que el de dibujo). Este tipo de ventana interna deberá poder mostrar imágenes, tanto leídas por un fichero como las generadas al capturar una imagen instantánea a partir de un video o una webcam.

La aplicación permitirá aplicar distintas operaciones que se podrán llevar a cabo sobre cualquier imagen.

Requisitos

RF 3.1 - La aplicación contará con una barra de herramientas para imágenes, en la cual se incluirán los botones que comentaremos a continuación, y a cada herramienta se le asociará su respectivo “ToolTipText” y podrá tener o no un icono asociado.

RF 3.2 - La aplicación contará con un botón “Duplicar”, el cual creará una nueva ventana interna con una copia de la imagen.

RF 3.3 - La aplicación contará con un deslizador mediante el cual podremos modificar el brillo de nuestra imagen.

RF 3.4 - La aplicación contará con un desplegable con varios tipos de filtros de tipo “Emborronamiento”, “Enfoque”, “Relieve” y “Fronteras”.

RF 3.5 - La aplicación contará con una serie de botones para marcar los distintos tipos de contraste, que son contraste normal, iluminado y oscurecido.

RF 3.6 - La aplicación contará con un botón “Negativo”, el cual invertirá los colores de la imagen.

RF 3.7 - La aplicación contará con un botón para la extracción de bandas.

RF 3.8 - La aplicación contará con un desplegable para la conversión a los espacios RGB, YCC y GRAY.

RF 3.9 - La aplicación contará con un deslizador mediante el cual será posible el giro libre de la imagen. (* Este apartado fue implementado para la práctica 11, y como no terminó de funcionar, le pregunté a mi compañera Laura Gómez)

RF 3.10 - La aplicación contará con dos botones mediante los cuales se podrá escalar la imagen (tanto para aumentarla como para disminuirla).

RF 3.11 - La aplicación contará con un botón “Tintado” el cual procederá a “tintar” la imagen según el color seleccionado en la parte de arriba.

RF 3.12 - La aplicación contará con un botón “Ecualización”.

RF 3.13 - La aplicación contará con un botón “Sepia”.

RF 3.14 - La aplicación contará con un deslizador para la “Umbralización”, basado en niveles de gris cogerá el valor del deslizador para modificar el umbral.

RF 3.15 - La aplicación contará con dos botones mediante los cuales se podrán realizar operaciones binarias sobre las imagenes (suma y resta).

RF 3.16 - La aplicación contará con un botón “Sobel”, el cual detectará las fronteras de la imagen.

RF 3.17 - La aplicación contará con un botón que aplicará sobre la imagen una función “LookupOp” definida por el estudiante.

RF 3.18 - La aplicación contará con un botón que aplicará sobre la imagen una función “Pixel a Pixel” definida por el estudiante.

RF 3.19 - La aplicación contará con un botón que aplicará sobre la imagen una función “Componente a Componente” definida por el estudiante.

Análisis

Para el procesamiento de imágenes, podemos ver como Java si tiene una serie de bibliotecas que nos permiten trabajar con ellas.

Además, nuestro profesor también nos ha proporcionado una serie de bibliotecas para poder aplicar varios tipos de filtros sin necesidad de implementarlos nosotros. Muchos de ellos también han sido implementados en prácticas, por lo que solo hemos tenido problemas para implementar los filtros nuevos que se piden en la evaluación, para los cuales hemos creado funciones propias y las hemos llamado desde nuestra Ventana Principal, para aplicarlas sobre la imagen que carguemos.

En el diseño procederemos a comentarlas.

Diseño

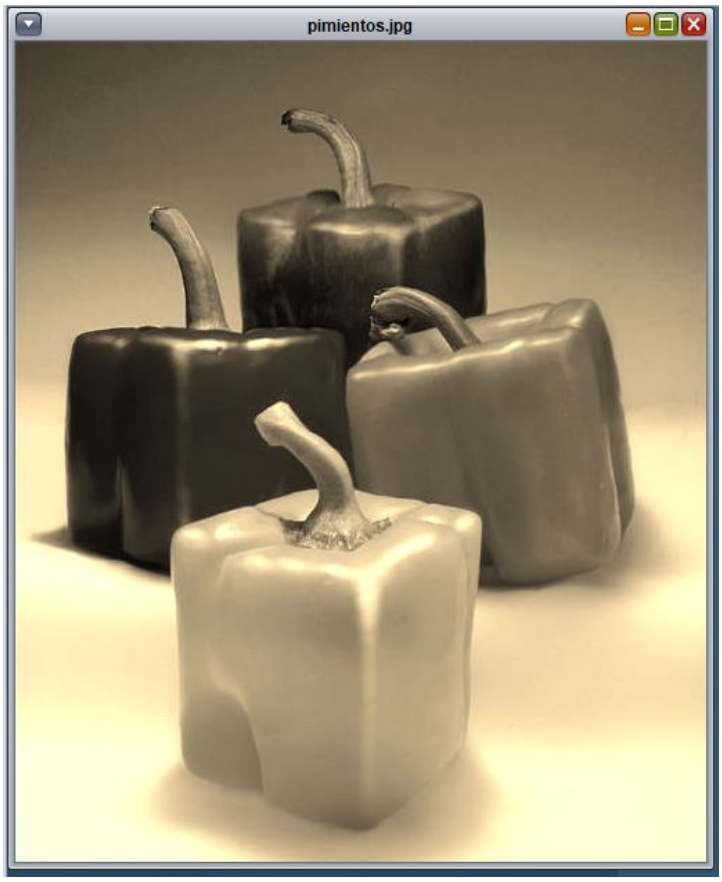
Sepia

El operador “Sepia” se trata de unos de los efectos más clásicos a la hora de procesar una imagen, ya que en él se modifica el tono y la saturación para darle un aspecto como de “fotografía antigua”.

El constructor utilizado será por defecto. Para conseguir este efecto recorreremos la imagen pixel a pixel y a cada componente le asignaremos los valores de la siguiente ecuación (la cual nos proporcionan en el guión de prácticas 12).

*sepiaR = min(255, 0.393*R + 0.769*G + 0.189*B);*
*sepiaG = min(255, 0.349*R + 0.686*G + 0.168*B);*
*sepiaB = min(255, 0.272*R + 0.534*G + 0.131*B);*

Demostración



Umbralización

Nos piden realizar una umbralización en escala de grises. Para ello lo primero que definiremos será una variable umbral, asociada a un int, a la cual le daremos valores mediante el constructor.

Entonces, una vez obtenido el umbral, recorreremos la imagen pixel a pixel, gracias a esta forma de recorrer la imagen, podremos calcular la intensidad mediante la media de sus componentes

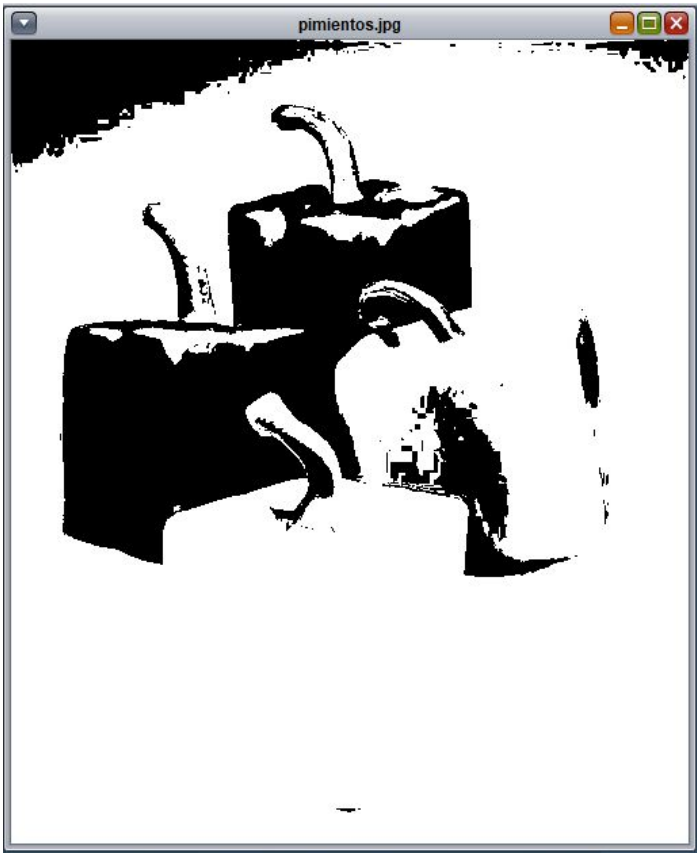
intensidad(x,y) = (r(x,y) + g(x,y) + b(x,y))

y según la intensidad dada y el umbral aplicaremos la siguiente función a cada componente:

componente(x,y) = { 255 si intensidad(x,y) >= umbral o 0 si intensidad(x,y) < umbral }

Ambas funciones vienen dadas en el guión la práctica 12.

Demostración



Cambio de Color (Pixel a Pixel)

Esta función va en línea con las anteriores. En este caso, también recorreremos la imagen píxel a píxel, pero aquí nuestra función lo que calculara es el "color complementario" de la imagen. Para este cálculo necesitaremos el paso de los valores RGB de la imagen a HSV, por lo que en esta clase es necesario implementar dos funciones más aparte de la específica filter, una primera para pasar los valores de HSV a RGB y otra después para volver a pasar los valores a RGB.

Este cambio se realiza mediante una serie de funciones matemáticas, en el primer caso, para pasar de RGB a HSV, serían las siguientes(*):

$$H = \begin{cases} \text{no definido}, & \text{si } MAX = MIN \\ 60° \times \frac{G-B}{MAX-MIN} + 0°, & \text{si } MAX = R \text{ y } G \geq B \\ 60° \times \frac{G-B}{MAX-MIN} + 360°, & \text{si } MAX = R \text{ y } G < B \\ 60° \times \frac{B-R}{MAX-MIN} + 120°, & \text{si } MAX = G \\ 60° \times \frac{R-G}{MAX-MIN} + 240°, & \text{si } MAX = B \end{cases}$$

$$S = \begin{cases} 0, & \text{si } MAX = 0 \\ 1 - \frac{MIN}{MAX}, & \text{en otro caso} \end{cases}$$

$$V = MAX$$

(*) en este caso, intente implementar yo la función del paso a HSL, pero después de diferentes fallos de que no me cogiese bien los valores o me dividiese por cero, decidi buscar e implementar una funcion ya hecha, la cual he cogido del siguiente enlace: https://stackoverflow.com/questions/2399150/convert-rgb-value-to-hsv

Por ello, se implementan ambas funciones en la clase, de tal manera que en la función filter se llama primero al cambio de RGB a HSV, una vez que tenemos los valores HSV, al valor H(cuyo significado es “Hue” o Matiz) se le añade una suma de 180º, y a esa suma le hacemos un módulo 360, para que no se sobrepase de los valores propios (ya que H va de 0 a 360).

Y después toca pasar de HSV a RGB mediante las siguientes funciones:

$$H_i = \left\lfloor \frac{H}{60} \right\rfloor \bmod 6; H \leq 360$$

$$f = \left(\frac{H}{60} \bmod 6 \right) - H_i$$

$$p = V(1 - S)$$

$$q = V(1 - fS),$$

$$t = V(1 - (1 - f)S)$$

$$\text{si } H_i = \begin{cases} 0, & R = V \\ & G = t \\ & B = p \\ 1, & R = q \\ & G = V \\ & B = p \\ 2, & R = p \\ & G = V \\ & B = t \\ 3, & R = p \\ & G = q \\ & B = V \\ 4, & R = t \\ & G = p \\ & B = V \\ 5, & R = V \\ & G = p \\ & B = q \end{cases}$$

Demostracion

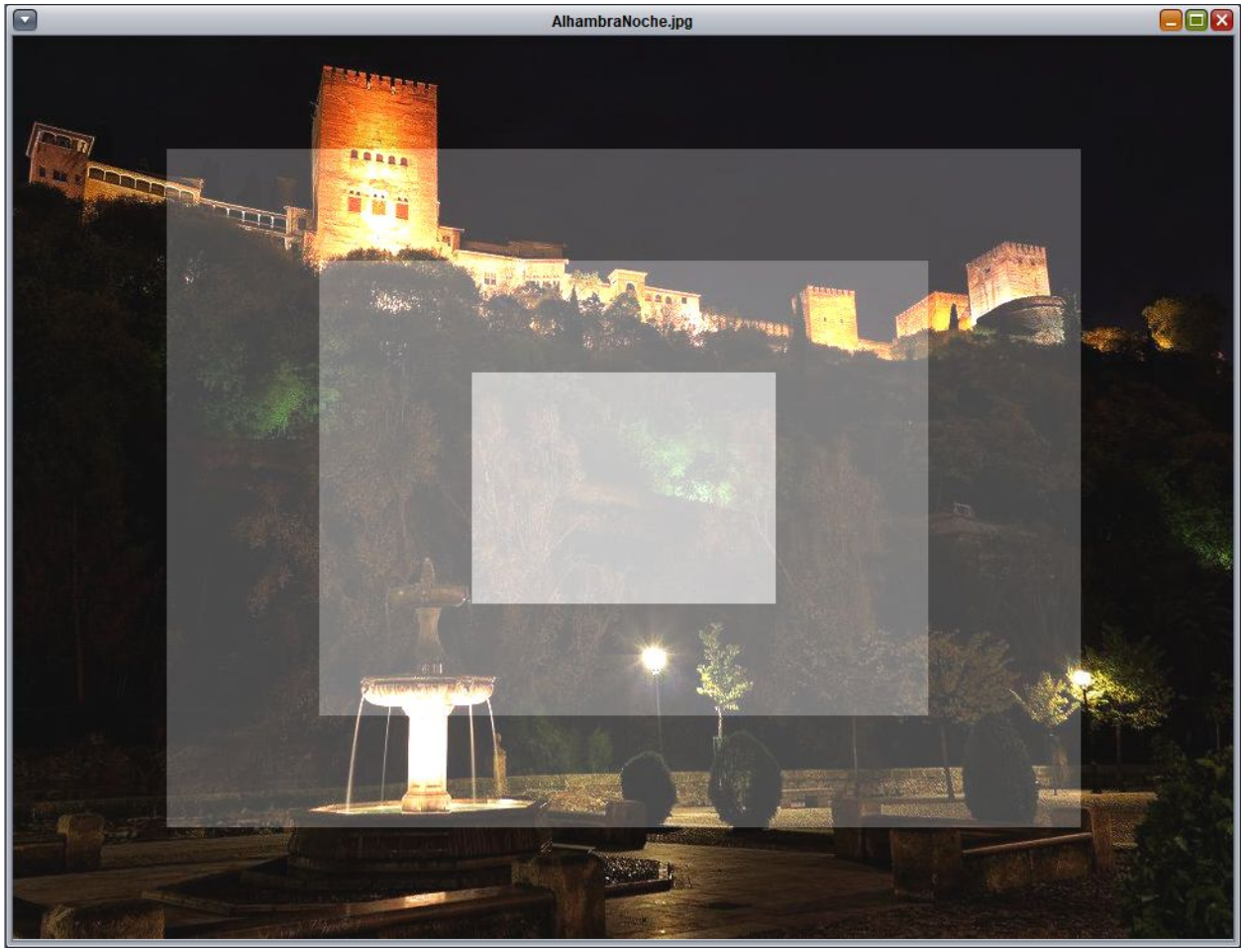
Iluminador por zonas (Componente a Componente)

En este caso, recorreremos la función componente a componente de tal forma que según la posición en la que esté situado el componente se aumente el brillo del mismo o no. Las zonas se asignan desde el centro, y se van desde un punto central a su alrededor.

En mi caso cojo el ancho y el alto de la figura y los divido en 8 partes, y a continuación, voy cogiendo cambiando los cuadrados de dentro para que se creen 4 cuadrados alrededor del

centro, los que más dentro de la imagen se encuentren más iluminados estarán que los de fuera.

Demostración



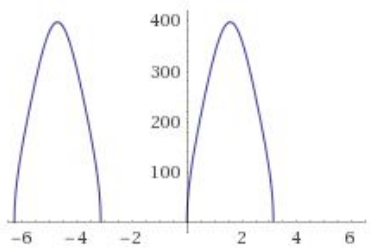
Función LookupOp

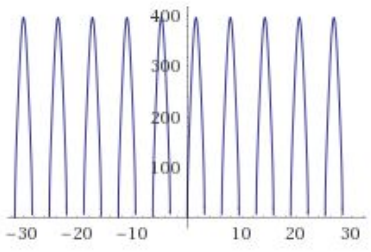
Aplicamos la siguiente función mediante el filtro LookupOp.

$$\tan\left(\sqrt{\sin(a\ i)}\right)$$

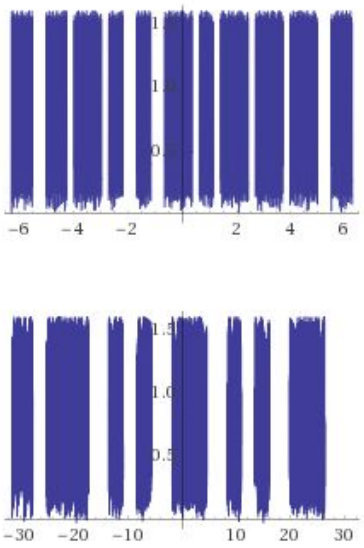
Esta función lo que va haciendo es recorrer la imagen byte a byte, y según los valores, multiplica por 128 (valor pasado por parámetro como a) el byte, y luego calcula su seno, a continuación calcula la raíz de ese valor, y finalmente calcula la tangente del mismo. Todo siempre multiplicado por una constante para captar siempre valores menores a 255.

Esto, sin dara valor a la variable a produciría la siguiente función:

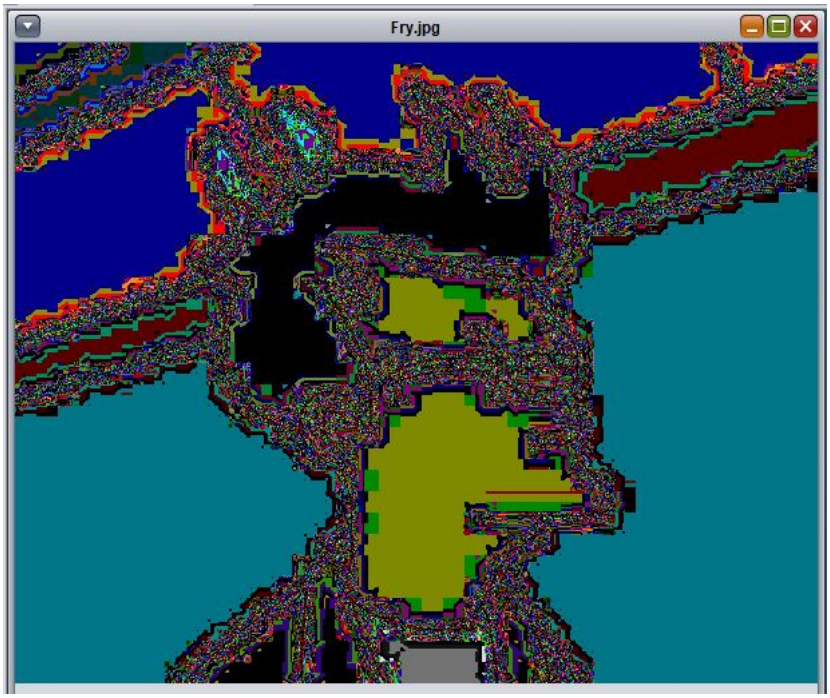




Pero como le estamos asignando al parámetro a el número 128, las curvas se van haciendo cada vez más pequeñas, lo que hace que se produzca el ruido dentro de la imagen.



Demostración



Sonido

Requisitos

RF 4.1 - La aplicacion debera permitir reproducir los sonidos añadidos en la lista de reproducción. Para ello

RF 4.2 - La aplicación deberá permitir grabar sonidos y guardarlos.

RF 4.3 - La reproducción se aplicará sobre los formatos y codecs reconocidos por Java.

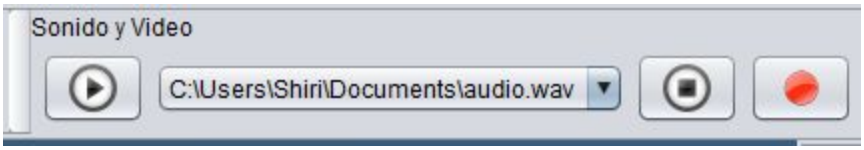
RF 4.4 - Se podrán fijar los parámetros de digitalización para la grabación.

Análisis

Este apartado lo hemos implementado principalmente con el guión de la práctica 13, por lo que no ha habido problemas a la hora de la implementación.

Diseño

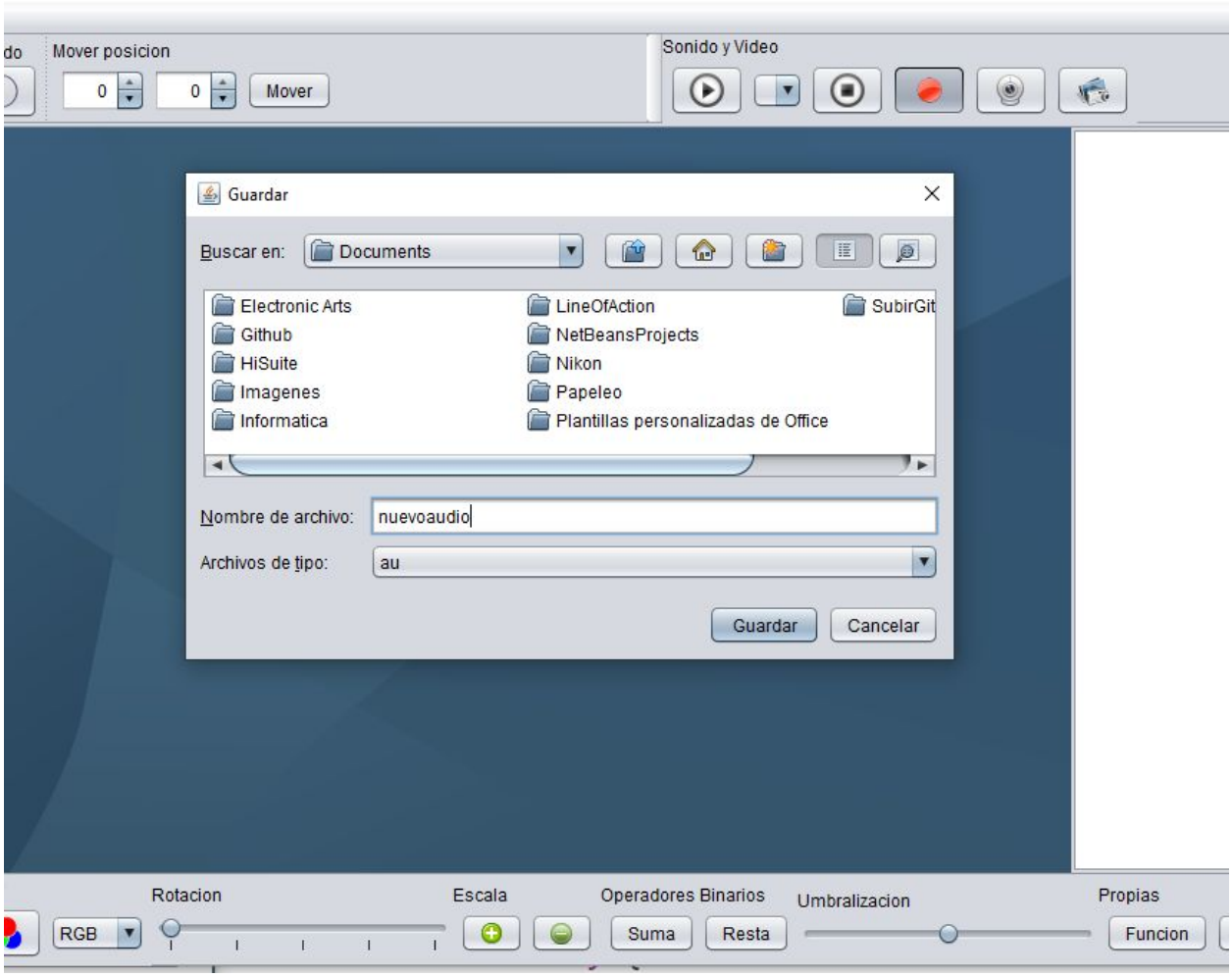
Vamos a comentar un poco el diseño de la interfaz.



Empezaremos comentando los botones de reproducción y pausa, los cuales hacen que podamos reproducir un sonido que esté ubicado dentro de nuestra lista de reproducción. Como lo hemos implementado según explica el guión de prácticas no habría más que comentar.

Es importante siempre seleccionar bien la opción que queremos escuchar para que el reproductor nos reproduzca el archivo adecuado.

También comentar que para abrir un archivo de audio se usará la opción de abrir archivos situado en el menú Archivo de la interfaz.



Además tenemos la opción de grabación de audio, hemos optado por la opción fácil, la cual es indicar primero sobre que archivo grabaremos y guardarlo, para guardar el archivo utilizaremos el formato AU que está puesto como predeterminado en las opciones de tipo de archivos. Una vez situado el archivo empezaremos a grabar. Para dejar de grabar tenemos que pulsar el

mismo botón de detención del sonido, y una vez pulsado, la grabación se añadirá a nuestra lista de reproducción y podremos proceder a reproducir nuestra grabación.

Contamos con un manejador de audio que bloquea nuestros botones según se puedan usar o no, gracias a eso no podemos reproducir mientras estemos grabando y viceversa. Además si estamos reproduciendo o grabando el único botón disponible será el de detención.

Video

Requisitos

RF 5.1 - La aplicación permitirá reproducir vídeo.

RF 5.2 - La aplicación permitirá mostrar la secuencia que esté captando la webcam.

RF 5.3 - La reproducción se aplicará sobre los formatos y códecs reconocidos por Java.

RF 5.4 - Se permitirá al usuario capturar imágenes de las ventanas de video y cámara que se estén reproduciendo en ese momento, concretamente de la ventana activa en el momento en el que se pulse el botón de captura. La imagen capturada se mostrará en una nueva ventana interna.

Análisis

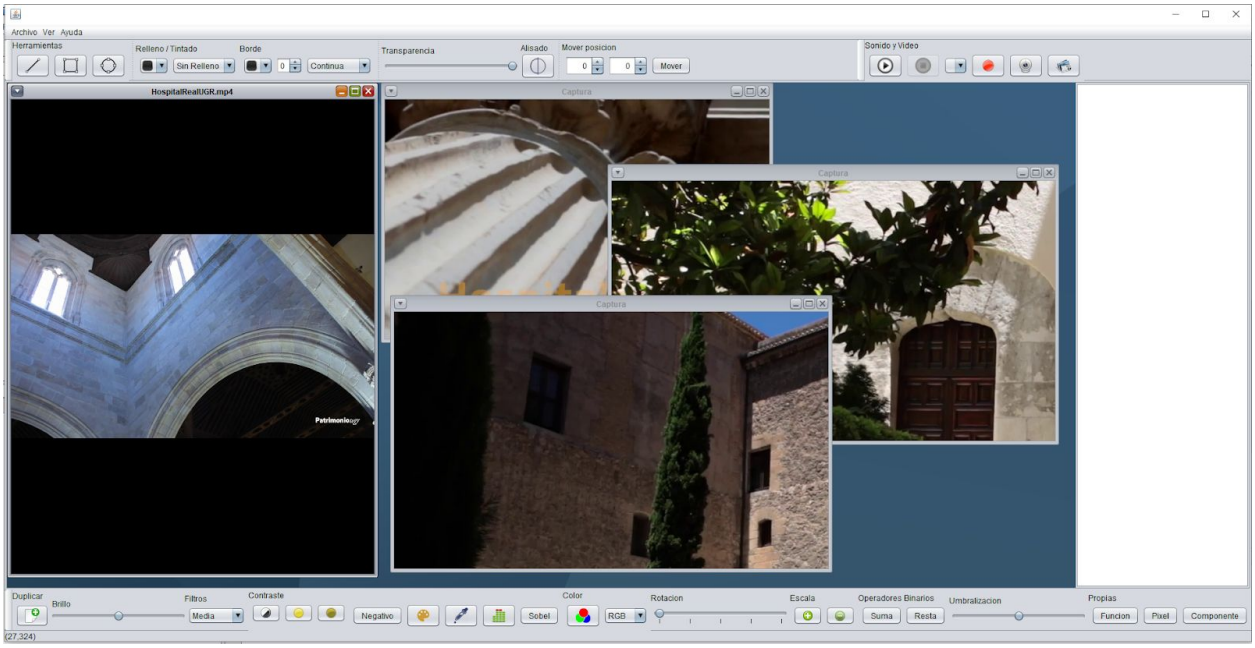
Igual que en el sonido, este código está principalmente obtenido del guión de prácticas 14.

Diseño

De la misma forma que en la parte de sonido, los archivos de audio se pueden abrir desde la opción de general del menú Archivo de la interfaz.

Además también hemos optado por usar los mismos botones para reproducción de audio y de video. Por ello una vez que le abramos un fichero de video y lo podremos reproducir con los mismos botones con los que reproducimos audio.

Además, hemos implementado una opción para capturar partes de un video, y que estas se muestren en nuevas ventanas internas de carácter general, con las cuales podemos jugar con el tratamiento de imágenes y guardarlas.



También hemos implementado la opción de mostrar lo captado mediante la webCam de nuestro ordenador en una ventana interna en nuestro escritorio, junto con la opción además de hacer captura de la misma y que estas también se muestren en una ventana interna aparte, pudiendo jugar también con el tratamiento de imágenes sobre ellas.