



Creación de una Interfaz Gráfica para Cargar y Visualizar Imágenes JPEG

Grupo: 4BV1

Profesor: De la O Torres Saúl

Alumno: Miranda San Martin Angel

Boleta: 2023630959

Instituto Politécnico Nacional / Escuela Superior de Cómputo

Fecha de entrega: 07/10/2024

Contents

1	Introducción	2
2	Desarrollo	3
2.1	Estructura del Proyecto	3
2.1.1	Bootstrap.java	3
2.1.2	ImageProcessor.java	3
2.1.3	PixelProcessor.java	4
2.1.4	BrightnessPixelProcessor.java	4
2.1.5	GrayScalePixelProcessor.java	4
2.1.6	BufferedImageContainer.java	4
2.1.7	MainFrame.java	4
2.2	Interfaz Gráfica	4
2.2.1	ControlPanel.java	5
2.2.2	MenuBar.java	5
2.3	Procesamiento de Imágenes	5
3	Funcionamiento	6
3.1	Pantalla Principal	6
3.2	Visualización de Imagen	7
3.3	Ajuste de Brillo y Contraste	8
3.4	Selección de Canal de Color	9
3.4.1	Canal Rojo	10
3.4.2	Canal Verde	11
3.4.3	Canal Azul	12
3.5	Aplicación de Filtro Monocromático	12
3.6	Guardado de la Imagen Modificada	13
3.7	Visualización de la Imagen Guardada	14
4	Conclusión	16

Chapter 1

Introducción

El procesamiento digital de imágenes es un campo interdisciplinario que involucra conceptos de matemáticas, informática, y electrónica, y su aplicación se extiende a una gran variedad de áreas como la medicina, la astronomía, el reconocimiento de patrones, la visión por computadora, y más. Su objetivo principal es la manipulación de imágenes digitales a través de algoritmos, permitiendo desde la corrección de color y brillo hasta la detección de características avanzadas.

En esta práctica, el enfoque fue desarrollar una aplicación en Java que permita cargar, visualizar y aplicar transformaciones básicas a imágenes en formato JPEG. Para ello, se emplearon las APIs de **Swing** y **AWT** (Abstract Window Toolkit), que proporcionan un conjunto de componentes gráficos y herramientas que facilitan la creación de interfaces gráficas de usuario (GUI).

Swing es una extensión de AWT que ofrece un conjunto más amplio de componentes GUI (botones, tablas, listas, etc.) y permite mayor personalización. AWT, por otro lado, proporciona los componentes gráficos básicos y permite manipular gráficos directamente a través de objetos como **BufferedImage**, fundamental para el procesamiento de imágenes en Java.

Para comprender mejor esta práctica, es crucial tener conocimientos previos sobre:

- **Modelos de color:** Las imágenes digitales suelen representarse mediante el modelo RGB (Rojo, Verde, Azul), donde cada píxel tiene valores que representan la intensidad de cada color.
- **Manipulación de píxeles:** Procesar una imagen implica modificar sus píxeles uno por uno, lo que puede hacerse para ajustar brillo, contraste, aplicar filtros, entre otros.
- **Interfaces gráficas en Java:** Comprender cómo funciona la arquitectura de Swing y AWT facilita la creación de aplicaciones interactivas y con componentes visuales.

Chapter 2

Desarrollo

2.1 Estructura del Proyecto

El proyecto está dividido en varias clases, que se encargan de tareas específicas para modularizar el código y facilitar su mantenimiento. La estructura sigue un diseño orientado a objetos, donde cada clase tiene una responsabilidad claramente definida. Las principales clases son:

2.1.1 Bootstrap.java

La clase `Bootstrap` actúa como el punto de entrada de la aplicación. En su método principal, utiliza el patrón de diseño **Singleton** para crear instancias de los procesadores de píxeles y de la interfaz gráfica principal (`MainFrame`). Este patrón asegura que solo exista una instancia de cada componente, lo que optimiza el uso de recursos y simplifica la gestión de dependencias.

La clase crea un objeto de `BufferedImageContainer`, que es el contenedor de la imagen cargada, y un `ImageProcessor`, que coordina el procesamiento de la imagen a través de los distintos filtros y ajustes.

2.1.2 ImageProcessor.java

El `ImageProcessor` es el encargado de gestionar el ciclo de procesamiento de la imagen. Recibe una lista de procesadores de píxeles (`PixelProcessor`) que se aplican secuencialmente a cada píxel de la imagen. Este patrón modular permite añadir fácilmente nuevos filtros en el futuro sin afectar la arquitectura existente.

Por ejemplo, el procesamiento se realiza recorriendo cada píxel de la imagen, extrayendo sus valores RGB y pasando estos valores por cada uno de los procesadores, como el de brillo o el de contraste.

2.1.3 PixelProcessor.java

`PixelProcessor` es una interfaz que define el comportamiento de los procesadores de píxeles. Todos los procesadores (brillo, contraste, escala de grises, etc.) implementan esta interfaz, lo que permite intercambiarlos o agregarlos de manera flexible sin modificar el código base del `ImageProcessor`.

Cada procesador recibe un `PixelProcessorContext`, que contiene el valor del píxel actual, su posición (x, y), y la configuración actual (brillo, contraste, etc.). Luego, realiza la transformación correspondiente y devuelve el valor modificado del píxel.

2.1.4 BrightnessPixelProcessor.java

Este procesador ajusta el brillo de cada píxel sumando o restando un valor a los canales RGB. Se asegura de que el valor final de cada canal esté en el rango válido de 0 a 255, utilizando las funciones `Math.min()` y `Math.max()` para evitar valores fuera del rango.

2.1.5 GrayScalePixelProcessor.java

El procesador de escala de grises convierte una imagen en color en una imagen en blanco y negro utilizando una fórmula estándar para combinar los valores RGB en un solo valor de luminosidad. Esta fórmula toma un porcentaje de cada canal (30% de rojo, 59% de verde y 11% de azul) para calcular el valor de gris resultante.

2.1.6 BufferedImageContainer.java

El `BufferedImageContainer` se encarga de gestionar la imagen cargada y su versión modificada. Permite al usuario restaurar la imagen original si es necesario. Además, la clase utiliza `BufferedImageHelper` para crear copias de la imagen original y preservar los datos originales.

2.1.7 MainFrame.java

La clase `MainFrame` contiene la ventana principal de la aplicación, que incluye componentes como el `ControlPanel` y el `ImagePanel`. El `ControlPanel` permite al usuario ajustar el brillo y el contraste de la imagen, mientras que el `ImagePanel` muestra la imagen cargada y actualizada en tiempo real.

2.2 Interfaz Gráfica

La interfaz gráfica se basa en los componentes de Swing y AWT. A continuación, se describen algunos de los elementos más importantes:

2.2.1 ControlPanel.java

El `ControlPanel` contiene los controles deslizantes (sliders) que permiten al usuario modificar el brillo y el contraste de la imagen cargada. Cada cambio en el valor del slider dispara un evento que llama al `ImageProcessor` para actualizar la imagen en tiempo real.

2.2.2MenuBar.java

El menú de la aplicación proporciona opciones para cargar y guardar imágenes. La opción de cargar permite al usuario seleccionar una imagen JPEG desde su sistema de archivos, mientras que la opción de guardar almacena la imagen modificada en el disco.

2.3 Procesamiento de Imágenes

El procesamiento de imágenes digitales implica la manipulación de los valores de píxeles que componen la imagen. En este proyecto, el procesamiento se realiza a nivel de píxel, utilizando los valores RGB para aplicar ajustes de brillo, contraste y convertir a escala de grises.

Cada píxel es un conjunto de valores que representan la intensidad de los colores primarios (rojo, verde y azul). Al modificar estos valores, podemos cambiar la apariencia de la imagen de manera significativa. El reto en esta práctica fue manipular estos valores de forma precisa para obtener los resultados deseados.

Chapter 3

Funcionamiento

3.1 Pantalla Principal

La pantalla principal de la aplicación muestra un área donde el usuario puede cargar una imagen arrastrando un archivo o haciendo clic para seleccionar una imagen desde el explorador de archivos.

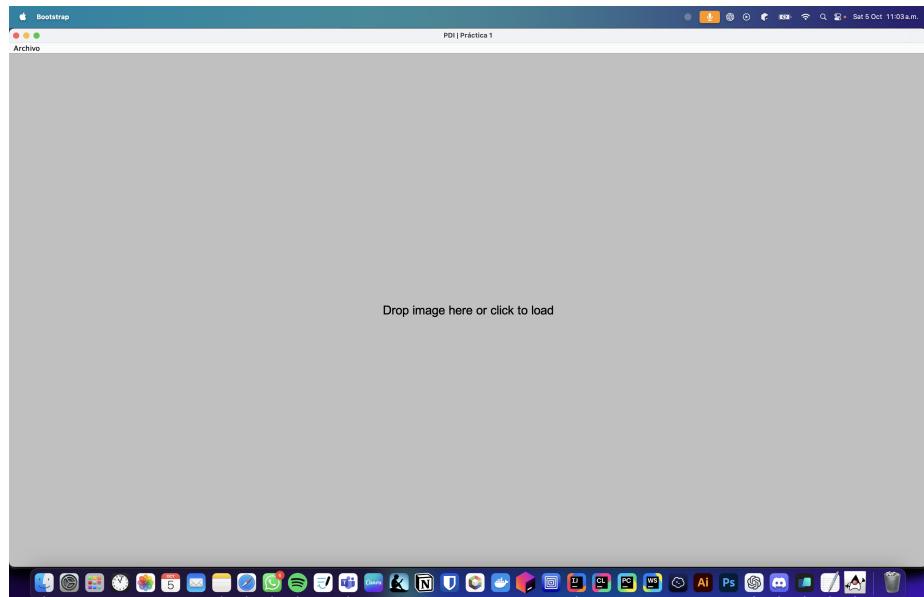


Figure 3.1: Pantalla principal de la aplicación

3.2 Visualización de Imagen

Una vez que la imagen es cargada, se puede visualizar dentro del área principal de la aplicación. En este caso, la imagen seleccionada es “La Noche Estrellada“ de Van Gogh.

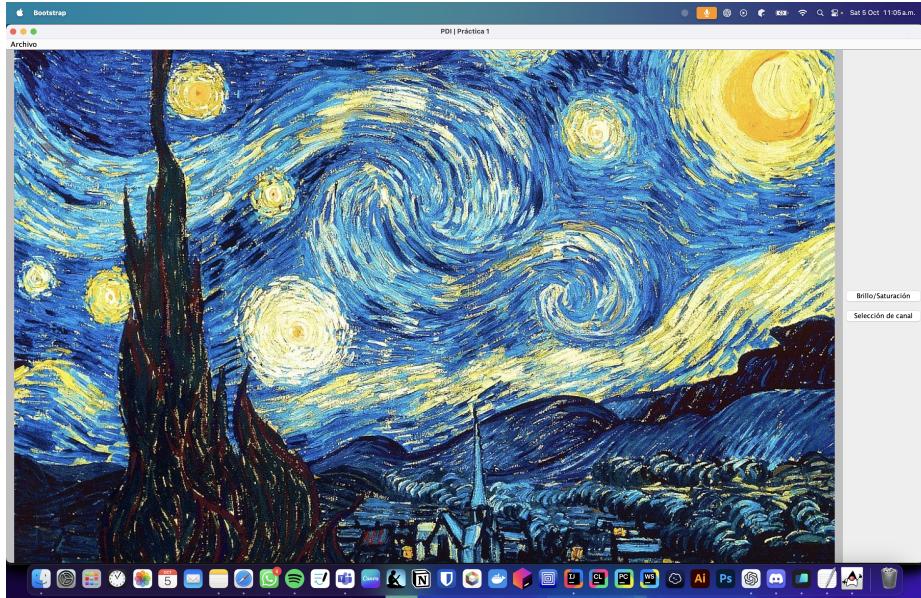


Figure 3.2: Imagen cargada y visualizada

3.3 Ajuste de Brillo y Contraste

El panel de Brillo y Contraste permite ajustar estos parámetros en tiempo real, mostrando los cambios directamente en la imagen visualizada.

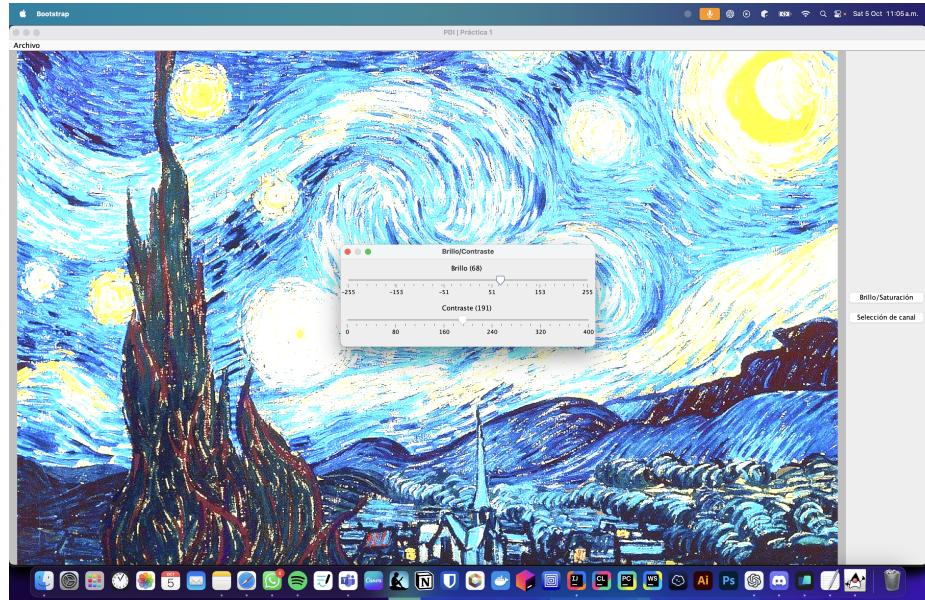


Figure 3.3: Ajuste de brillo y contraste

3.4 Selección de Canal de Color

El usuario puede seleccionar uno de los canales de color (Rojo, Verde o Azul) para visualizar sólo ese canal de la imagen.

3.4.1 Canal Rojo

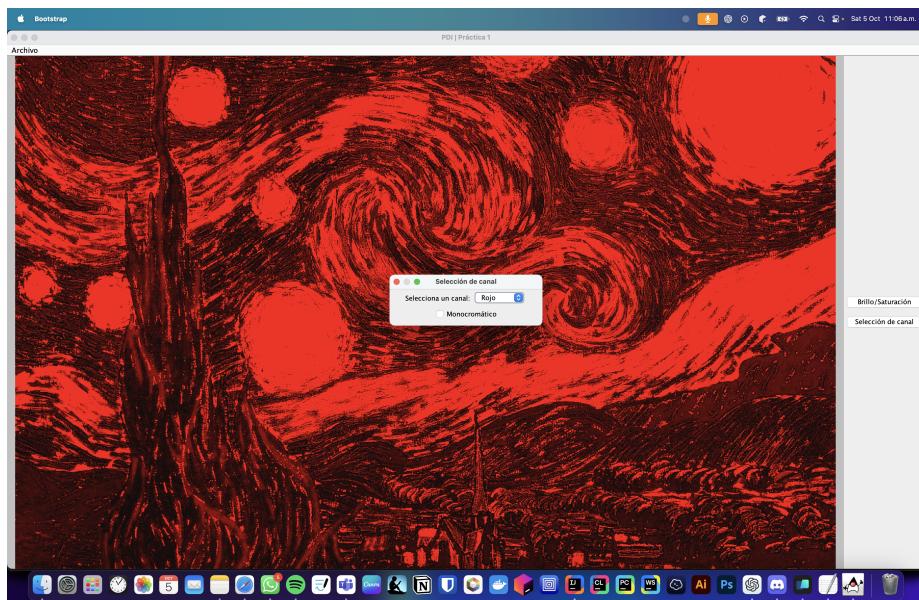


Figure 3.4: Visualización del canal rojo

3.4.2 Canal Verde

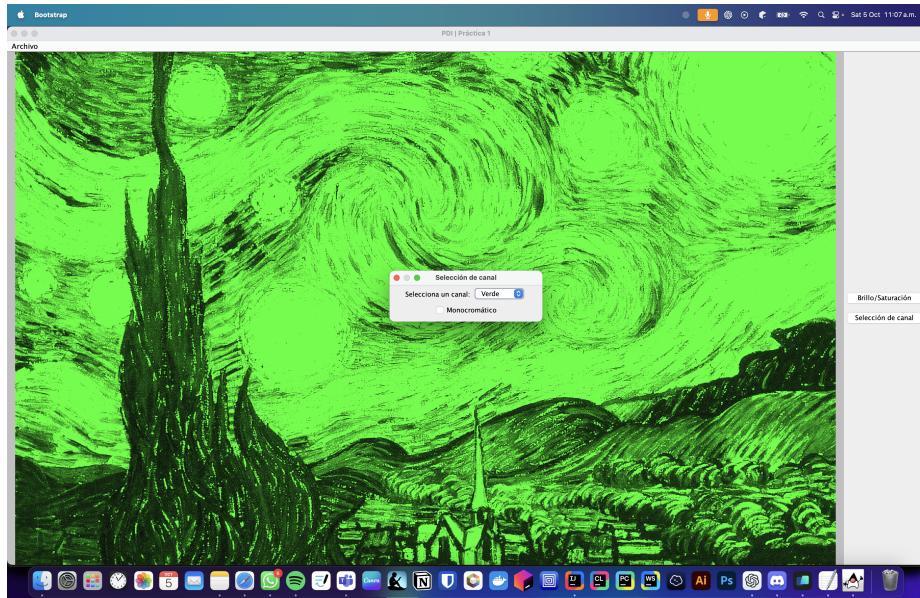


Figure 3.5: Visualización del canal verde

3.4.3 Canal Azul

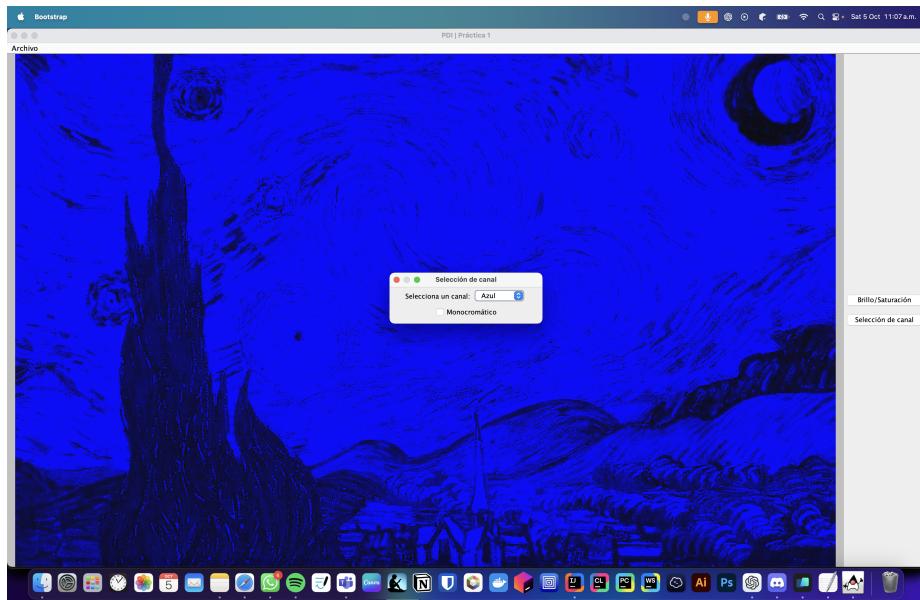


Figure 3.6: Visualización del canal azul

3.5 Aplicación de Filtro Monocromático

El usuario también puede aplicar un filtro monocromático para convertir la imagen a escala de grises.

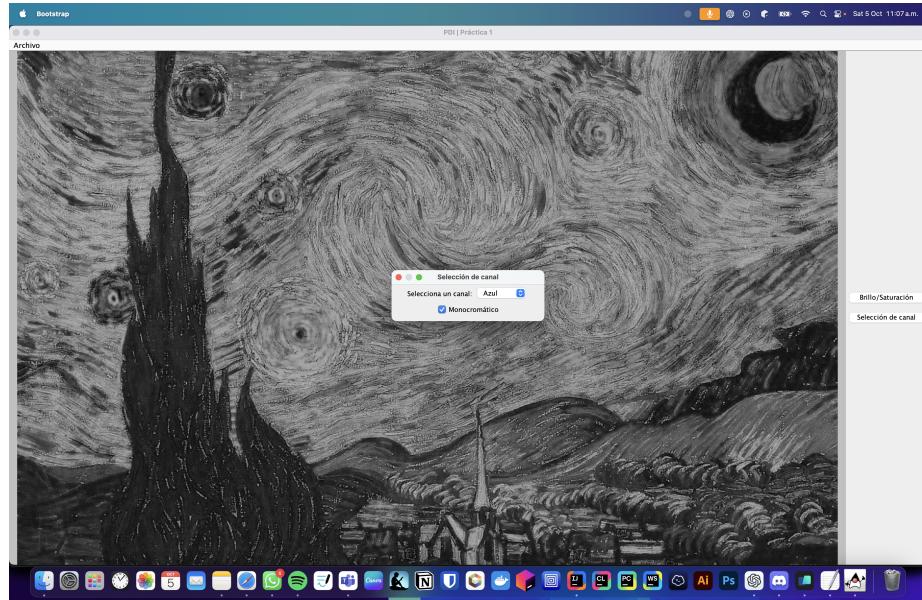


Figure 3.7: Aplicación del filtro monocromático

3.6 Guardado de la Imagen Modificada

Una vez que la imagen ha sido modificada, el usuario puede guardarla, seleccionando el nombre y la ubicación en su sistema de archivos.

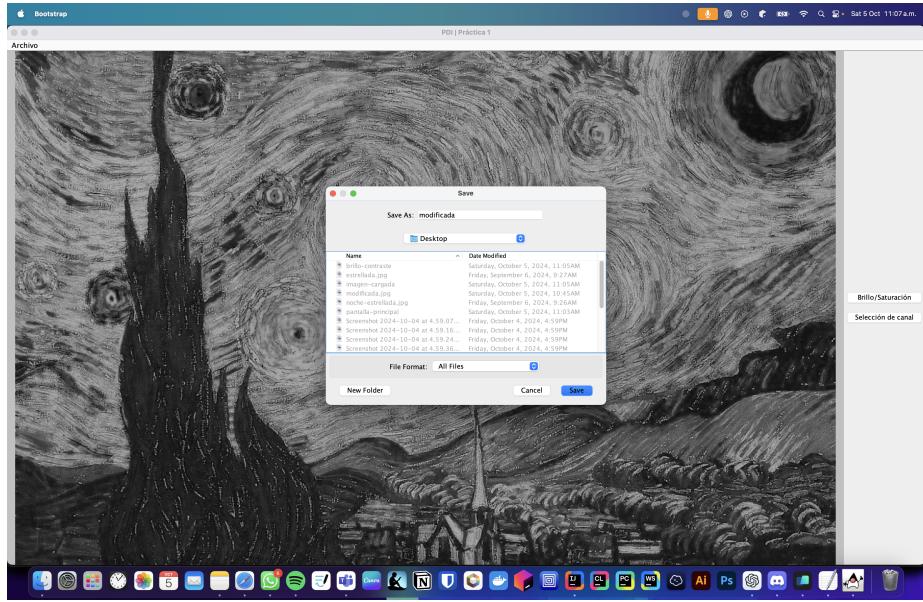


Figure 3.8: Guardado de la imagen modificada

3.7 Visualización de la Imagen Guardada

Finalmente, la imagen guardada puede ser abierta y visualizada con el visor de imágenes del sistema operativo.

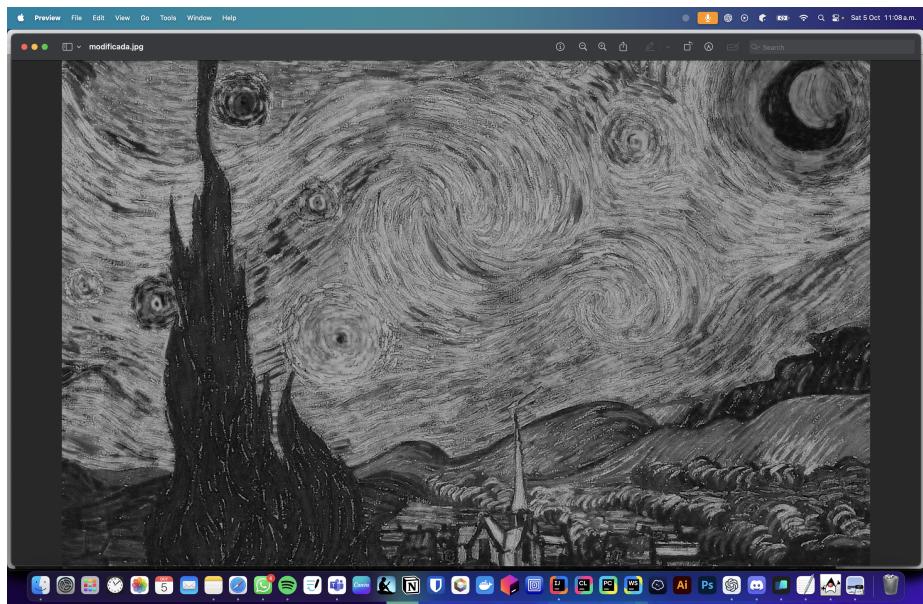


Figure 3.9: Imagen modificada guardada visualizada

Chapter 4

Conclusión

Esta práctica fue fundamental para entender cómo las imágenes pueden ser manipuladas a nivel de píxel utilizando Java. Al trabajar con las APIs de Swing y AWT, se aprendió cómo construir interfaces gráficas interactivas, lo cual es clave en muchas aplicaciones modernas.

Una de las principales conclusiones es la importancia de la modularidad en el diseño de software. Al dividir el código en clases específicas, fue posible mantener una estructura clara y fácil de mantener. Además, el uso de interfaces permitió la creación de nuevos procesadores de píxeles sin alterar el código existente, lo que demuestra la flexibilidad del diseño orientado a objetos.

Como mejora, se podría extender la funcionalidad de la aplicación para soportar más formatos de imagen, o incluir algoritmos de procesamiento más avanzados, como la detección de bordes o la segmentación de imágenes. También se podría optimizar el rendimiento para manejar imágenes de mayor resolución sin comprometer la velocidad de procesamiento.

Finalmente, esta práctica proporcionó una experiencia valiosa en la creación de aplicaciones gráficas y en el procesamiento de imágenes, temas que son esenciales en el campo de la visión por computadora y la inteligencia artificial.