## Manejo de archivos

A la hora de cargar imágenes desde archivo en la aplicación, realizo un ajuste de tamaño para que la imagen quepa dentro del canvas sin distorsionar la figura. Esto implica cierta pérdida en la calidad de la imagen, que es compensada con la funcionalidad de descargar tanto la imagen editada con lápiz, goma o incorporación de figuras geométricas, como la imagen resultante de aplicar cualquier filtro de los descritos en el presente informe a la imagen editada.

## Paint

Para desarrollar la funcionalidad de paint, habilité las funcionalidad de lápiz y goma (lápiz con color blanco) y brindé la posibilidad de escoger tanto el grosor como el color a utilizar. Luego implementé las funciones empezarDibujar, seguirDibujando y terminaDibujar para manejar los eventos del mouse (https://javascript.info/mouse-events-basics) y crear las líneas que unen cada par de puntos registrados del movimiento del mouse.
Adicionalmente creé un botón para dibujar rectángulos utilizando las funciones antes mencionadas.

## Conversión de colores

Para realizar la conversión de escala de colores utilicé información de los links https://stackoverflow.com/questions/17242144/javascript-convert-hsb-hsv-color-to-rgb-accurately, y https://gist.github.com/mjackson/5311256 y luego la comparé con el conversor https://www.rapidtables.com/convert/color/rgb-to-hsv.html y el visto en clase http://colorizer.org/.
Para los valores H (de HSV y HSL) los resultados son normalizados a una escala [0 - 1], para completar la conversión implemente un corrector que pasa ese valor normalizado a su equivalente en grados.

## Filtros simples

### Black & White (Binarización)

Consiste en convertir cada pixel a blanco (255, 255, 255) o negro (0, 0, 0).

### Negativo (inversión de color)

Consiste en invertir el color de cada pixel, esto sería pasar los componentes R, G y B a su valor opuesto (255 - R, 255 - G, 255 - B).
https://stackoverflow.com/questions/8662349/convert-negative-image-to-positive/8662445

### Escala de grises

Consiste en transformar cada pixel a una tonalidad de gris, para realizarlo los componentes R, G, y B deben tomar el mismo valor. Por preferencia práctica dicho valor se conforma del promedio de los 3 componentes. (R, G y B = (R + G + B) / 3 ).

### Filtro brillo

Consiste simplemente, en multiplicar el valor de cada componente por un factor para manipular el brillo de cada pixel. (R = R * factor, G = G * factor, B = B * factor).
El valor del factor lo debe ingresar el usuario seleccionando el % que desea, para esto implementé una barra de progreso basada en la obtenida en el siguiente link:
https://www.w3schools.com/howto/howto_js_progressbar.asp
La lógica adicional fue la obtención de la posición seleccionada por el usuario (evento click en el div de la barra de progreso) y su equivalente a porcentaje, para este cálculo necesité conocer el tamaño del div (utilizando el método **getBoundingClientRect()**).
Para este filtro, se utiliza un tope de 3.

### Sepia

El filtro Sepia es una variante del “escala de grises” pero en lugar de tomar el promedio de los componentes RGB, toma un promedio ponderado.
Para implementarlo obtuve los valores de ponderación del link:
https://stackoverflow.com/questions/1061093/how-is-a-sepia-tone-created

## Filtros complejos

### Filtro luminosidad

Consiste en convertir los componentes RGB de cada pixel a su equivalente HSL, realizar desplazamientos en el componente L (Lightness) y finalmente re-convertir a RGB.
El factor se utiliza el mismo implementado en el filtro de brillo, con el mismo tope.

### Filtro saturación

Consiste en convertir los componentes RGB de cada pixel a su equivalente HSL, realizar desplazamientos en el componente S (Saturation) y finalmente re-convertir a RGB.
El factor se utiliza el mismo implementado en el filtro de brillo, con el mismo tope.

### Blur

Para realizar este filtro, se debe tomar los valores de todos los pixeles vecinos y el propio y los promedia para finalmente multiplicarlos por el factor. Se utiliza un tope de 3.

Para la implementación necesité trabajar con una copia de la matriz.
La base del filtro la obtuve del link: https://processing.org/examples/blur.html y lo comparé con lo generado en https://pinetools.com/blur-image.

## Detección de bordes

Para la implementación de este filtro encontré varias alternativas, pero me incliné por la descripta en el siguiente link: https://processing.org/examples/edgedetection.html
La forma de resolverlo fue recorrer cada pixel, buscar los vecinos (utilizando la función getVecinos(originalImageData, i, j)), se acumula el tono de gris (promedio de componentes RGB) de todos los vecinos y finalmente se calcula la diferencia entre el pixel actual y sus vecinos (como muestra la matriz del link adjunto):

```
{{ -1, -1, -1},
 { -1,  9, -1},
 { -1, -1, -1}};
```