INFORMA2 S.A.S

Parcial 2: Implementación

Juan Pablo Cruz Gómez

Erika Dayana León Quiroga

Despartamento de Ingeniería Electrónica y Telecomunicaciones
Universidad de Antioquia
Medellín
Septiembre de 2021

Índice

1. Sección introductoria

En este informe se mostrará y explicará la implementación del problema presentado en el Parcial 2 de la materia Informática II, también se mostrarán las clases implementadas, los módulos del código implementado, la estructura del circuito montado en Tinkercad para la demostración del programa y los problemas que se presentaron durante el desarrollo de la implementación del problema.

2. Clases implementadas

En esta sección se presentarán las clases que se utilizarán para la solución del problema planteado.

2.1. Clase QImage.

Esta clase es proporcionada por Qt, sirve para el manejo de datos de imágenes, especialmente para el acceso y manipulación de pixeles. En este caso se utilizan los métodos que nos permiten conocer el número de pixeles a lo ancho y a lo largo de la imagen (.width, .height), para de esta forma obtener su tamaño. También se utiliza el método .pixelColor el cual retorna el color del pixel en unas coordenadas determinadas, en el caso de este problema se utiliza el .pixelColor dentro de un doble ciclo for que nos proporciona cada una de las coordenadas de la imagen, y de esta forma obtenemos la intensidad de color en cada uno de los pixeles de la imagen.

3. Módulos del código implementado

1. Se incluye el código para que el usuario ingrese la ruta del archivo por consola y de esta forma cargar la imagen que se encuentra en la misma con ayuda de la clase QImage. También se crea el archivo de escritura que se usará más adelante para sacar la información del procesamiento de la imagen. Se declara el valor del alto y ancho de la imagen cargada anteriormente y el alto y el ancho de la matriz auxiliar que se usará para convertir la matriz original en una de 16x16.

```
string rutaImagen;
cout<<"Ejemplo: ../Parcial2/banderas/NombreDeLaImagen.png"<<endl;
cout<<"Ingrese la ruta de la imagen: ";
cin>> rutaImagen;

//Cargo la imagen en la ruta definida por filename
QImage imagenOriginal(rutaImagen.c_str());
ofstream archivoBandera;
archivoBandera.open("RGBbandera.txt",ios::out);

int anchoOriginal=imagenOriginal.width();
int altoOriginal=imagenOriginal.height();

//El ancho y alto de la imagen se dividen entre 16, pues es el numero
int AltoMatrizAux = altoOriginal/16;
int AnchoMatrizAux = anchoOriginal/16;

cout<<AnchoMatrizAux<<" x "<<AltoMatrizAux<<endl;
//Matriz tridimensional en donde se guardaran las tres matrices RGB
int MatrizRGB[3][altoOriginal][anchoOriginal+1];
int MatrizAux[AnchoMatrizAux][AltoMatrizAux];
```

Figura 1: Declaración de algunas variables

2. En esta parte del código se leen cada uno de los pixeles de la imagen para guardar su intensidad de color rojo, verde y azul, se hace uso de una matriz tridimensional llamada MatrizRGB para este propósito.

```
for(int i=0;i<3;i++){
    //Se recorre el largo y ancho de la imagen
    for(int indY = 0; indY < altoOriginal; indY++){
        for(int indX = 0; indX < anchoOriginal; indX++){

            if(i==0){
                MatrizRGB[i][indY][indX] = imagenOriginal.pixelColor(indX,indY).red();
                }
            else if(i==1){
                MatrizRGB[i][indY][indX] = imagenOriginal.pixelColor(indX,indY).green();
                }
            else{
                MatrizRGB[i][indY][indX] = imagenOriginal.pixelColor(indX,indY).blue();
                }
                }
            }
        }
```

Figura 2: Matriz RGB

3. Después de tener las matrices de intensidad rojo, verde y azul guardadas en la metriz tridimensional MatrizRGB, procedemos a hacer el submuestreo de cada una de ellas, esto se logrará utilizando un while y un contador que se ocuparán de pasar por las tres matrices. Luego, a cada una de ellas se le irán sacando unas matrices auxiliares, las cuales tendrán un ancho y un largo de el largo y el ancho de las matrices originales dividido entre 16, que es el número de LEDs disponibles para la representación en Tinkercad.

```
while(cont<3){
for(int indY = 0; indY < altoOriginal; indY+= AltoMatrizAux){
     for(int indX = 0; indX < anchoOriginal; indX+= AnchoMatrizAux){

         for(int i=0; i<AltoMatrizAux; i++){
         for(int j=0; j<AnchoMatrizAux; j++){
             MatrizAux[i][j]=MatrizRGB[cont][indY+i][indX+j];
                sumatoria=sumatoria+MatrizAux[i][j];
                }
            }
         promedio=round(sumatoria/(AltoMatrizAux*AnchoMatrizAux));
         promedios[posPromedio]=promedio;
         posPromedio++;
         sumatoria=0;
        }
    }
```

Figura 3: Matriz auxiliar.

4. Cada que se añade un elemento a la matriz auxiliar, este mismo elemento va sumándose con los demás elementos de la misma matriz para luego sacarle el promedio dividiendo entre la multiplicación del ancho por el alto de la matriz auxiliar. Este promedio calculado se va guardando en un arreglo llamado 'promedios' de tamaño 256.

```
        MatrizAux[i][j]=MatrizRGB[cont][indY+i][indX+j];
            sumatoria=sumatoria+MatrizAux[i][j];
            }
        }
    promedio=round(sumatoria/(AltoMatrizAux*AnchoMatrizAux));
    promedios[posPromedio]=promedio;
    posPromedio++;
    sumatoria=0;
```

Figura 4: Cálculo del promedio.

5. Se recorre newMATRIZ que es la matriz obtenida por medio de promedios de la matriz original para convertirla a tamaño de 16x16, y se va rellenando haciendo uso del arreglo 'promedios', que iniciará en cero e irá aumentando de uno en uno hasta acabar el arreglo de tamaño 256.

   En esta misma parte del código se escribe el archivo de texto llamado 'RGBbandera.txt', el cual contendrá la información de las tres matrices de intensidad necesarias para la simulación de Tinkercad.

```
            posPromedio=0;
            //Ciclo para rellenar la nueva matriz de intensidad con tamaño 16x16
            archivoBandera<<"{";
            archivoBandera<<endl;
            for(int newfila=0; newfila<16; newfila++){
                archivoBandera<<"{";
                for(int newcolumna=0; newcolumna<16; newcolumna++){
                    newMATRIZ[cont][newfila][newcolumna]=promedios[posPromedio];
                    posPromedio++;
                    archivoBandera<<newMATRIZ[cont][newfila][newcolumna]<<",";
                    cout<<newMATRIZ[cont][newfila][newcolumna]<<" ";
                }
                archivoBandera<<"},";
                archivoBandera<<endl;

                cout<<endl;
            }
            cont++;
            posPromedio=0;
            archivoBandera<<"},";
            archivoBandera<<endl;
            cout<<endl;
    }
            archivoBandera.close();
```

Figura 5: Nueva matriz 16x16

4. Estructura del circuito

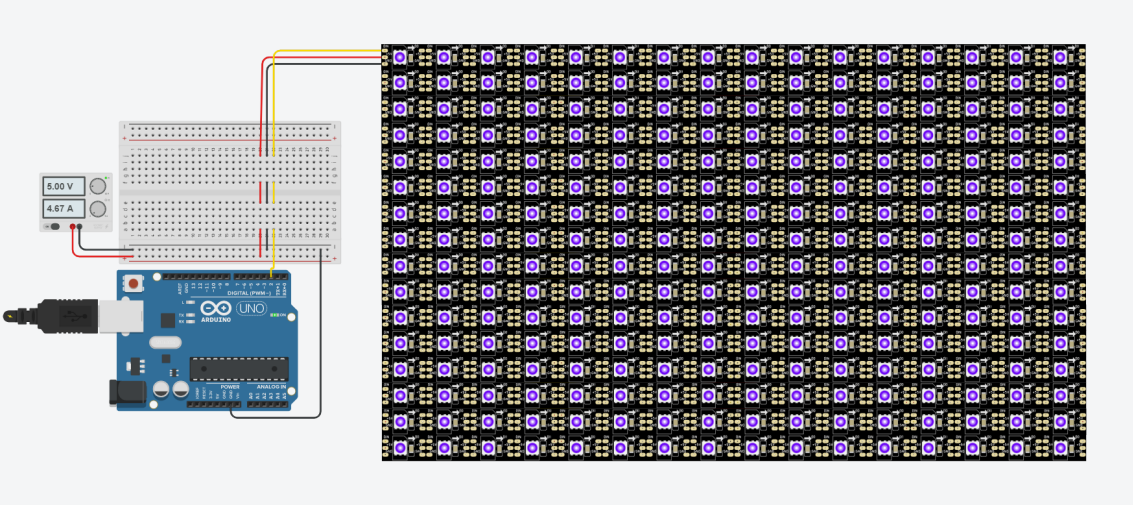

Figura 6: Montaje de la matriz de LEDs de 16x16

5. Problemas presentados

El principal problema presentado fue el encontrar una forma de reducir o amplificar el tamaño de las imágenes a 16x16, que es el tamaño decidido para representar las imágenes mediante las tiras de Neo Pixeles.

No se hacía un submuestreo correcto ya que estábamos tomando en el código los componentes RGB de la imagen columna por columna y los estábamos guardando fila por fila, por lo cual no quedaban matrices con los componentes RGB correctas para se simulados en la plataforma de Tinkercad.

Teníamos problemas en la lectura de la matriz 16 x16 en tinkercad y se solucionó poniendo la matriz 16 x 17.

6. Manual de usuario

Siga los siguientes pasos para un correcto funcionamiento del programa.

1. Ubique la carpeta en su computador en donde se guarda el programa de qt.

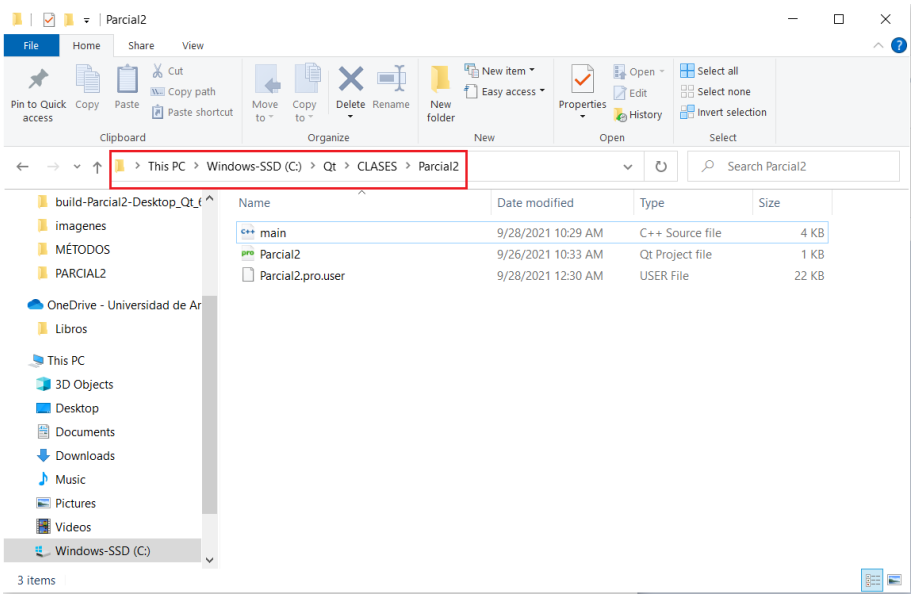

Figura 7: Ubicar la carpeta en donde se guarda el programa

2. Cree una carpeta para guardar en ella las imágenes de las banderas que posteriormente se procesarán.

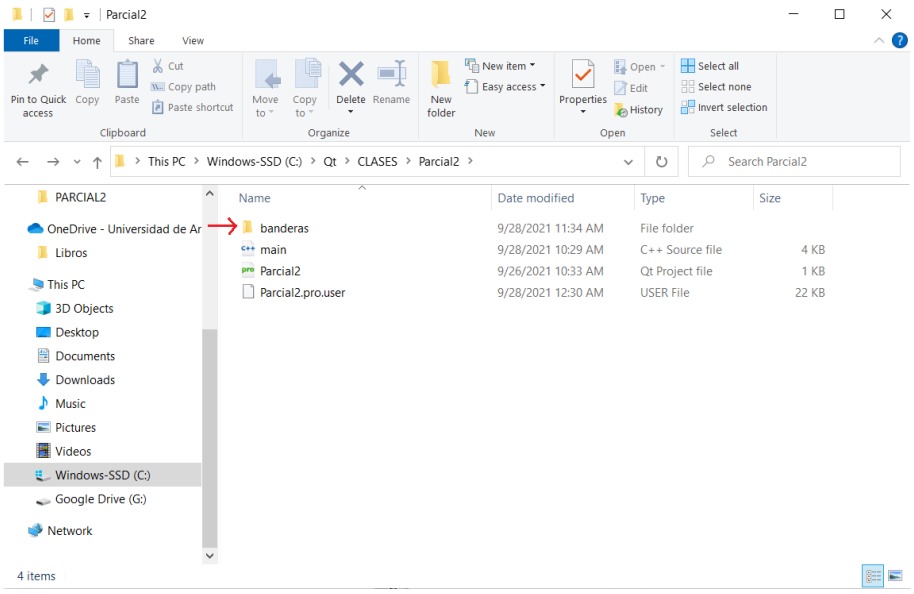

Figura 8: Crear la carpeta en donde se guardarán las imágenes

3. Abra el programa y presione el botón ubicado en la parte inferior izquierda de su pantalla encerrado y señalado en rojo que se muestra en la figura 9 para iniciar el programa, o la combinación de teclas Ctrl+R.

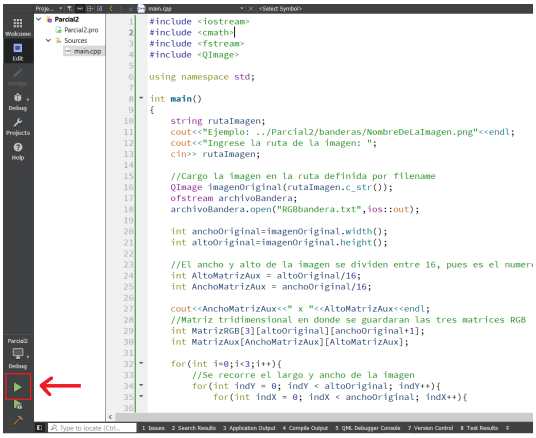

Figura 9: Manual de usuario: Primer paso.

4. Después de realizar el paso anterior, una ventana emergente como la que se muestra en la figura 10 se mostrará en la pantalla y será el lugar en donde se ingrese el único dato necesario para el programa. El dato a ingresar es la ruta de la imagen y se muestra un ejemplo de cómo debe ser escrito.

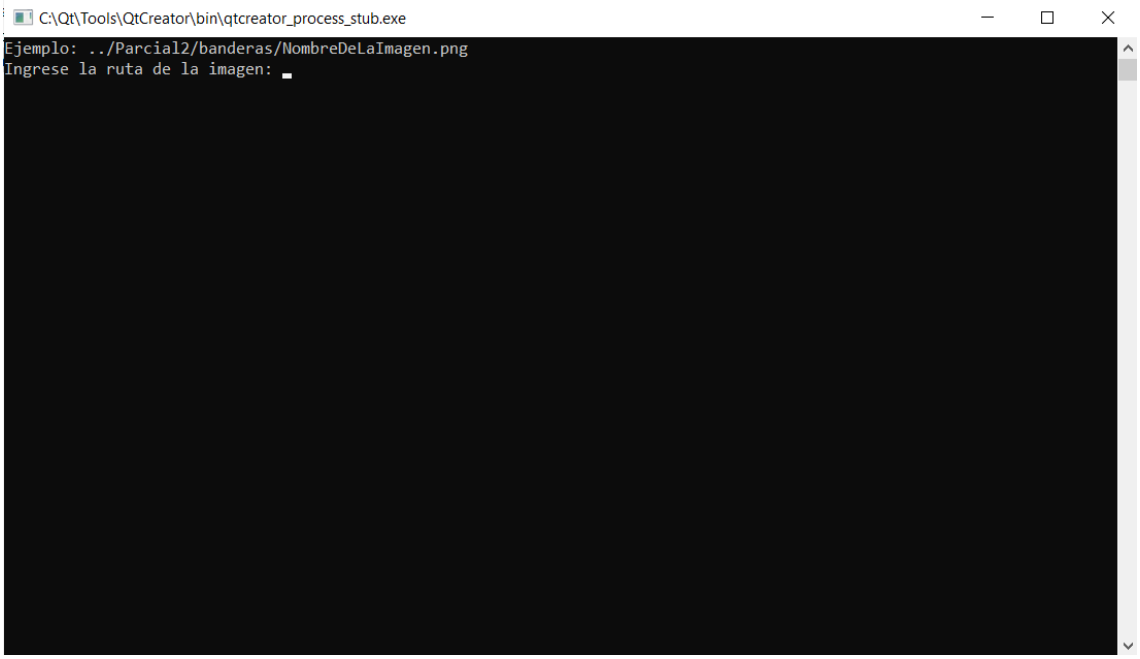

Figura 10: Ingreso de datos.

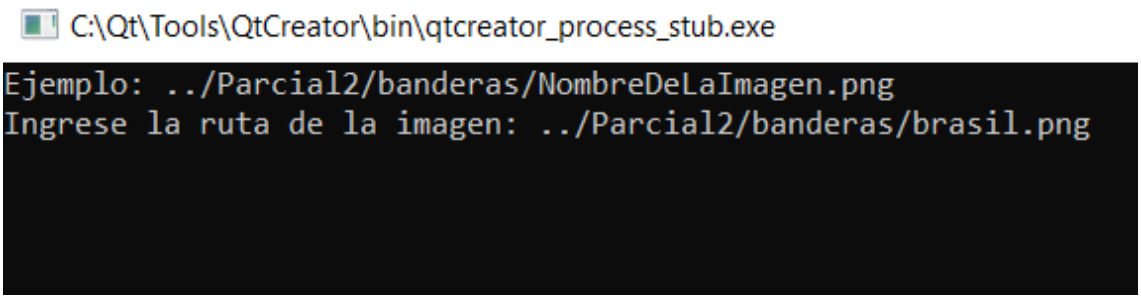

Figura 11: Ejemplo de ingreso de datos.

5. Después de ingresar la ruta de la imagen el programa procesará la imagen y creará un archivo de texto en donde se guardará la información que se tiene que incluir en el código de Tinkercad.

   El archivo de texto se guardará en una carpeta en la que su nombre comienza con 'build-' y sigue con el nombre de la carpeta en donde se guarda el programa, en este ejemplo 'Parcial2'.

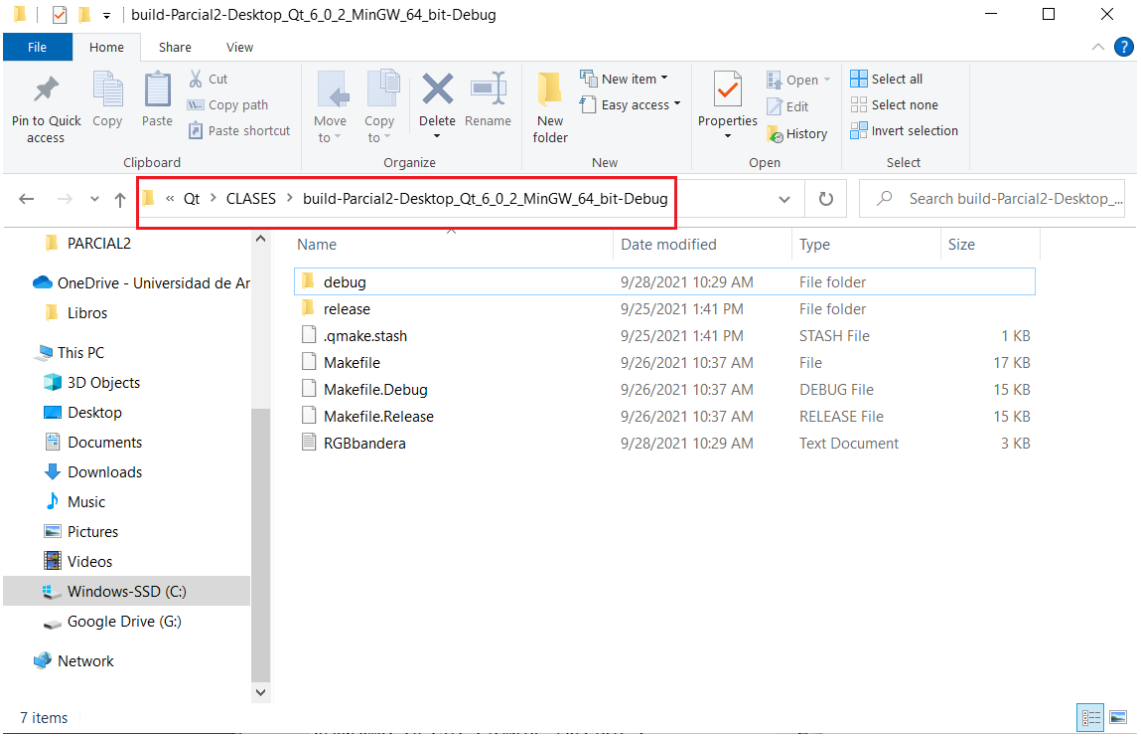

Figura 12: Ejemplo de ingreso de datos.

6. Se abre el archivo RGBbandera.txt y se copia todo su contenido.

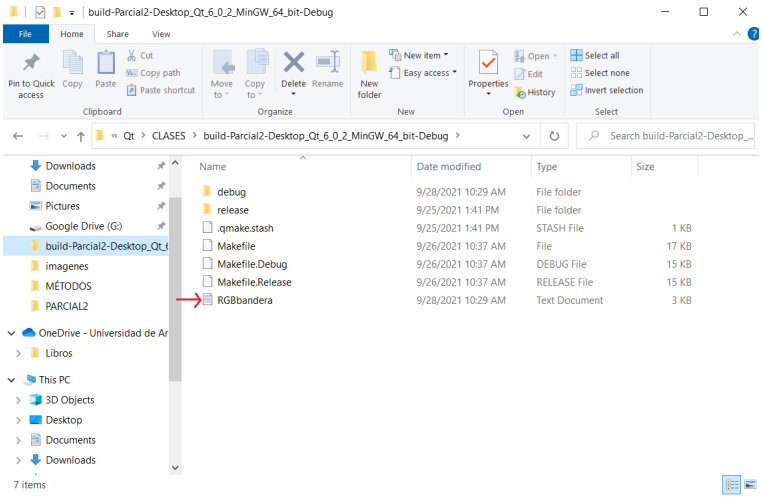

Figura 13: Archivo .txt para abrir.

7. Se abre en el navegador el siguiente link que nos llevará al circuito montado en Tinkercad: https://www.tinkercad.com/things/bFvlFZ9hjM0

   Posterior a esto se abre la ventana de código y en la sección demarcada con '//COPIAR TEXTO DESDE ESTE PUNTO' se copia el contenido del archivo RGBbanderas.txt.

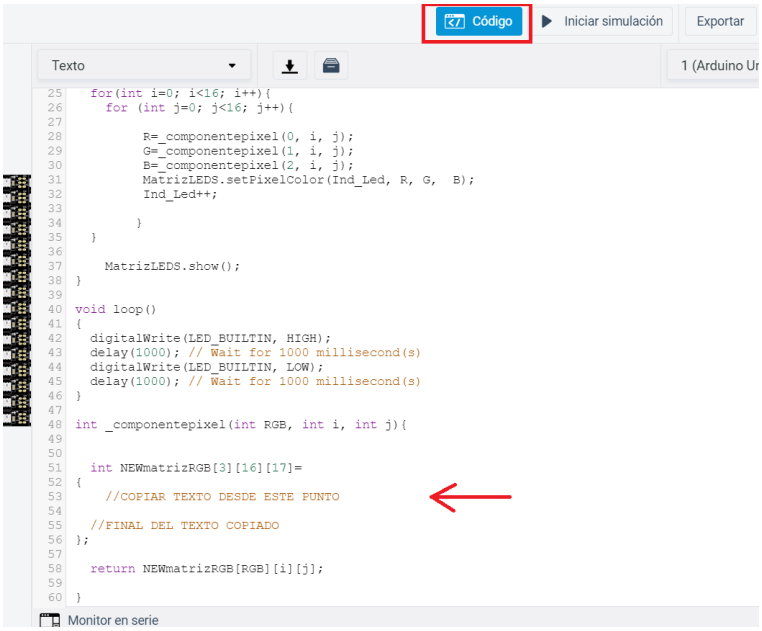

Figura 14: Tinkercad.

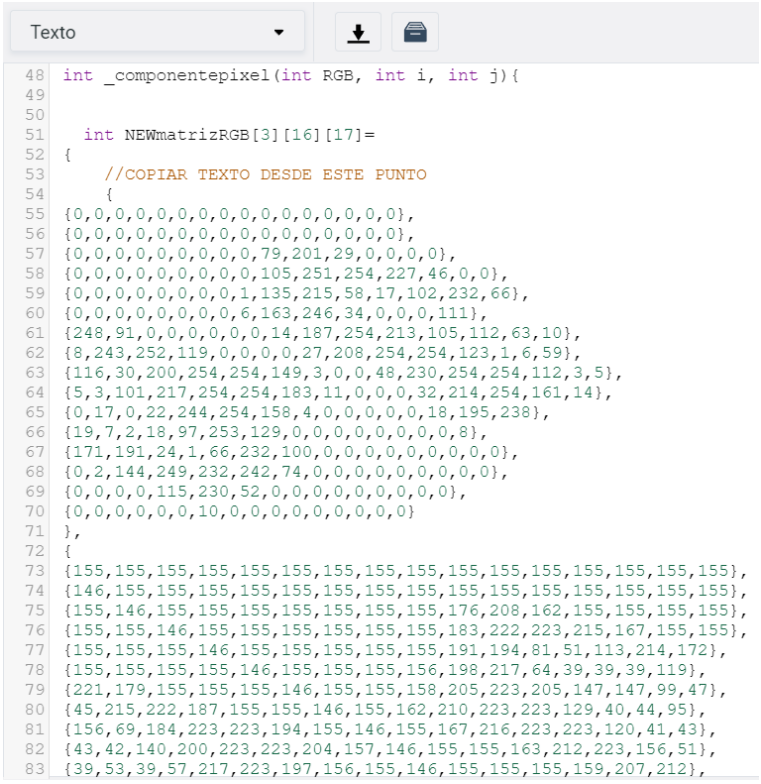

Figura 15: Ejemplo de la información copiada del archivo.

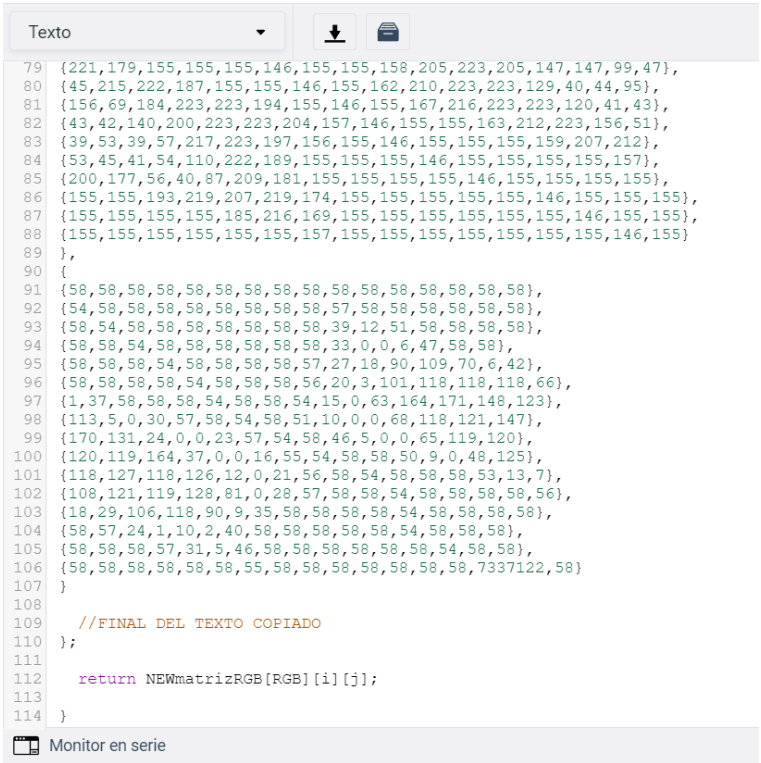

Figura 16: Ejemplo de la información copiada del archivo.