# Informe Parcial 2

Informática II

**Daniel Perez Gallego CC. 1193088770**
**Jorge Montaña Cisneros CC. 1007327968**

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

# Índice



# 1. Clases implementadas

## 1.1. Menu

Clase iteractiva con el usuario, delegada de pedir el nombre de la imagen con su formato, almacenada en la carpeta 'Imagenes'. Retorna la variable 'im', correspondiente a la imagen cargada con el tipo QImage.

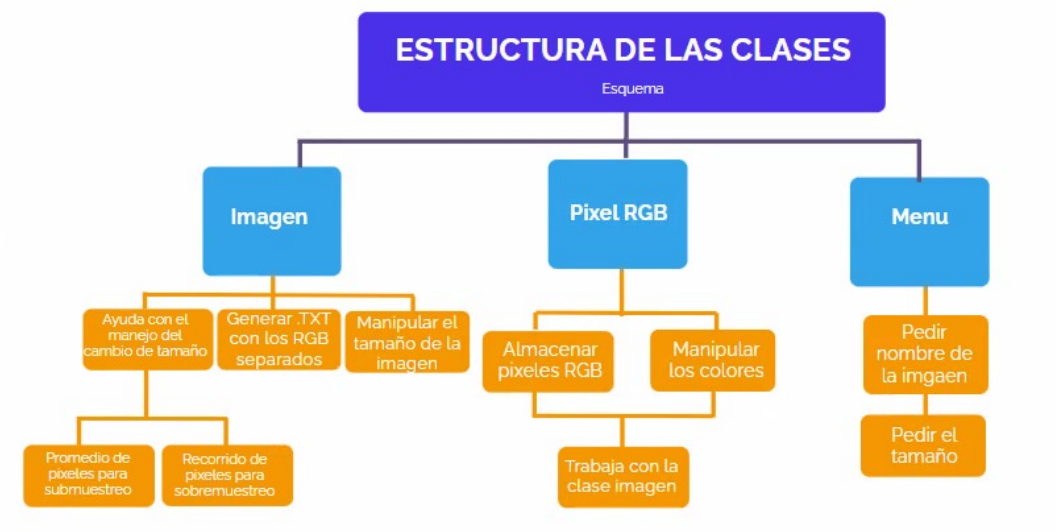
## 1.2. Imagen

Encargada de manejar la imagen con los parámetros de su alto y ancho, promediar los colores por bloques y crear el .txt generado con el formato adecuado.

## 1.3. Pixel RGB

Es la encargada de almacenar los pixeles RGB de la imagen, tiene como parámetro los colores del RGB

## 2. Esquema de las clases




## 3. Módulos de código de interacción

```
class Imagen
{
private:
    int fila, columna;
    vector<vector<Pixel_RGB>> Pixel_color;
public:
    Imagen();
    Imagen(int M, int N);
    void set_color(int x, int y, Pixel_RGB color);
    int getFila() const;
    void setFila(int value);
    int getColumna() const;
    void setColumna(int value);
    void imprimir_pruebas();
    void txt_generado();
    Pixel_RGB Promedio_Color(int fo, int cantidadF, int co, int
    cantidadC);
    Pixel_RGB recorrer(int fo, int co);
};
```

Listing 1: Parámetros de la clase imagen

```
Pixel_RGB Imagen::recorrer (int fo, int co)
{
    int limF = fo;
    int limC = co;
    int Red = 0, Green= 0, Blue = 0;
    for (int f=fo; f<=limF; f++ ) {
        for (int c=co; c<=limC; c++ ) {
            Red = Pixel_color[f][c].getRed();
            Green = Pixel_color[f][c].getGreen();
            Blue = Pixel_color[f][c].getBlue();
        }
    }

    return Pixel_RGB(Red, Green, Blue);
}
```

Listing 2: Clase recorrer

```
Pixel_RGB Imagen::Promedio_Color(int fo, int cantidadF, int co, int
     cantidadC)
{
    int limF = fo+cantidadF;
    int limC = co+cantidadC;
    int pixeles = cantidadF*cantidadC;
    int sumaRed = 0, sumaGreen= 0, sumaBlue = 0;
    for (int f=fo; f<limF; f++ ) {
        for (int c=co; c<limC; c++ ) {
            sumaRed += Pixel_color[f][c].getRed();
            sumaGreen += Pixel_color[f][c].getGreen();
            sumaBlue += Pixel_color[f][c].getBlue();
        }
    }

    return Pixel_RGB(sumaRed/pixeles, sumaGreen/pixeles, sumaBlue/
    pixeles);
}
```

Listing 3: Clase promedio color

Se evidencia la utilización de la clase pixel RGB, donde se almacenan los pixeles en una función de la clase imagen.

```
int main()
{
    QImage im=menu();
    Imagen imagen_original(im.width(),im.height());
    Imagen reducida;
    Imagen agrandada;
    int x, y;
```

```
    cout<<"Ingrese el tamano de la imagen deseado: "<<endl;
    cout<<"Ancho: ";
    cin>>x;
    cout<<"Alto: ";
    cin>>y;
    cout<<endl;


    for(int filas=0;filas<im.width();++filas){
        for(int columnas=0;columnas<im.height();++columnas){
              Pixel_RGB color(im.pixelColor(filas,columnas).red(),
    im.pixelColor(filas,columnas).green(),im.pixelColor(filas,
    columnas).blue());
              imagen_original.set_color(filas,columnas,color);
        }
    }
    if(im.width() > x && im.width() > y){

        reducida=reducir(imagen_original,x,y);
        reducida.txt_generado();
       // reducida.imprimir_pruebas();
    }
    else if(im.width() < x && im.width() < y){
        agrandada=agrandar(imagen_original,x,y);
        agrandada.txt_generado();
    }
    else if(im.width() == x && im.width()== y){
        imagen_original.txt_generado();
    }


}
}
```

```
Imagen reducir(Imagen imagen,int m, int n){
    Imagen reducida(m,n);
    int bloqueF= imagen.getFila()/m;//obtenemos bloque de filas x
    int bloqueC= imagen.getColumna()/n;//obtenemos bloque de
    columnas y
    int c=0, f=0;

    int fila0=imagen.getFila();
    int Columna0=imagen.getColumna();


    if(bloqueF*m<fila0){
        int dif=fila0-bloqueF*m;
        fila0-=dif;
    }
    if(bloqueC*n<Columna0){
        int dif=Columna0-bloqueC*n;
        Columna0-=dif;
    }

    for (int i=0; i<fila0; i+=bloqueF) {
        c=0;
        for (int j=0; j<Columna0; j+=bloqueC ) {
```

```
            Pixel_RGB nuevoPixel = imagen.Promedio_Color(i,bloqueF,
    j,bloqueC);
            reducida.set_color(c,f,nuevoPixel);
            c++;
        }
        f++;
    }
    return reducida;
}
```

Listing 4: funcion para reducir

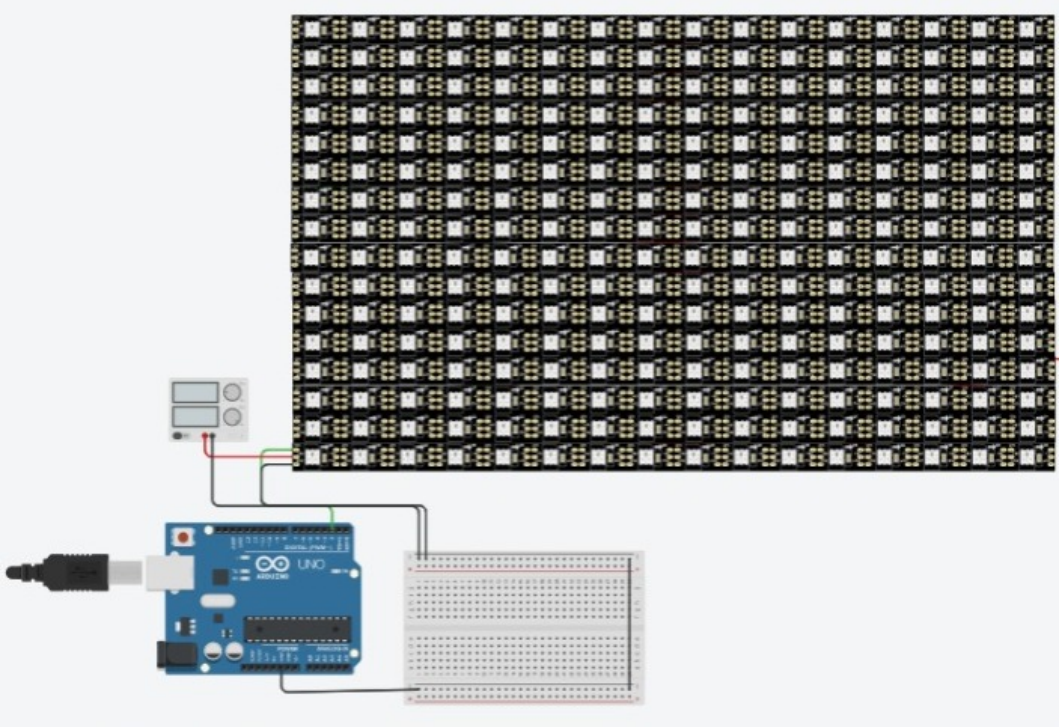
```
Imagen agrandar(Imagen imagen,int x, int y){
    int bloqueF = x/imagen.getFila();
    int bloqueC = y/imagen.getColumna();
    Imagen agrandada(x,y);
    int c=0, f=0;int difF=0, difC=0;


   for(int k=0;k<bloqueF;k++){
    difF=x-bloqueF*imagen.getFila();
    for(int i=0;i<imagen.getFila();i++){
        if(bloqueF==1){
            if(difF>=0){
                difF--;
                 i= 0;
            }
            else{
            bloqueF = x/imagen.getFila();
            }
        }
        else if(bloqueF>1){
            if(difF>=0 and k>0){
                difF--;
                i= 0;
            }
            else{
            bloqueF = x/imagen.getFila();
            }
        }
        c=0;
        difC=y-bloqueC*imagen.getColumna();
        for(int j=0;j<imagen.getColumna();j++){
            if(difC>0){
                bloqueC = y/imagen.getColumna()+1;
                difC--;
            }
            else{
            bloqueC= y/imagen.getColumna();
            }
            for(int l=0;l<bloqueC;l++){
            Pixel_RGB nuevoPixel= imagen.recorrer(i,j);
            agrandada.set_color(c,f,nuevoPixel);
            c++;
           }
          }
          f++;
        }
```

```
47      }
48
49      return agrandada;
50  }
```

Listing 5: Funcion para agrandar

## 4. Estructura del circuito montado

Para la matriz de LEDs en Tinkerdad, diseñamos un circuito de 16x16 LEDs, hecha con tiras de neopixel. Cada una con su salida conectada a la entrada de la fila/tira superior, la potencia conectada a un sumnistro de energía y todas las coneciones para que el circuito funcione con normalidad



## 5. Problemas presentados

Justo como lo analizamos, el método para reducir y amplificar la imgagen fué la parte más complicada en la implementación, a pesar de que buscamos varias métodos, a la hora de codificarlo se complicaba y comenzamos a buscar un método para simplificarlo, hasta el punto donde consideramos aplicar un nuevo

método y empezar casi desde 0.

Problemas para la función de sobremuestreo, con los bloques impares.

Desconocíamos el formato que debían ser escritos los RGB en el .txt generado para tinkercad y si teníamos que insertar algún método para que el usuario no tenga que copiar y pegar el RGB en el tinkercad

La conexión del circuito fué un problema menor gracias a la búsqueda de documentación y videos sobre el código y la conexión en tinkercad; sin embargo, pensábamos que se encenderían los LEDs rápido, pero como no lo hacian debido a toda la información que se procesaba, abortabamos el proceso pensando que el circuito estaba malo, pero no lo estaba, solo éramos muy impacientes.