

Dokumentacja techniczna programu Photo Editor

Poniższa dokumentacja opisuje funkcjonalność oraz wady stworzonego oprogramowania. Podczas implementacji nie zostały użyte biblioteki zewnętrzne. Obrazy, które widoczne są w tym dokumencie nie należą do mnie, a także nie mam do nich praw autorskich.

Oprogramowanie (wersja z kodem) jest również dostępne pod wskazanym poniżej linkiem:

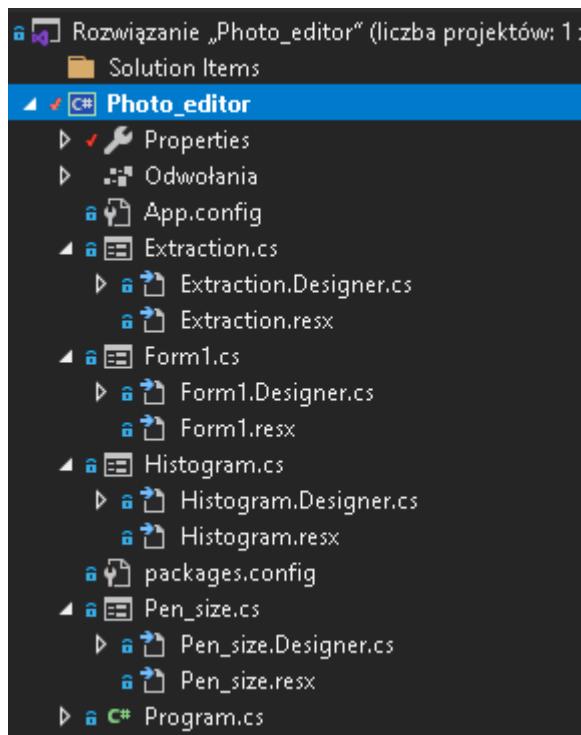
https://github.com/Kinoruu/Photo_editor

1. Wykorzystane biblioteki:

```
1  using System;
2  using System.Collections.Generic;
3  using System.ComponentModel;
4  using System.Data;
5  using System.Drawing;
6  using System.Linq;
7  using System.Text;
8  using System.Threading;
9  using System.Threading.Tasks;
10 using System.Windows.Forms;
11 using System.Collections;
12 using System.Drawing.Imaging;
13 using System.Runtime.InteropServices;
14 using System.IO;
15 using System.Drawing.Printing;
16 using System.Runtime.Serialization.Formatters.Binary;
17 using System.Runtime.Serialization;
```

Rysunek 1: Zrzut ekranu implementacji bibliotek

2. Przeważająca część funkcjonalności programu znajduje się w pliku Form1.cs, mimo to program jest w ciągły rozwijany i mimo zawartości kodu lub obecności przycisków w layoucie programu nie koniecznie oznacza, że dana funkcja jest obecnie dostępna.



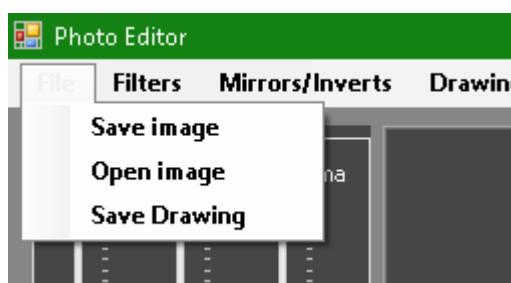
Rysunek 2: Zrzut ekranu podglądu rozwiązania programu (Solution)

3. Wykorzystane obrazy podczas tworzenia pokazowej funkcjonalności :

<https://i.pinimg.com/originals/42/f2/0a/42f20a9ded5ab1457674f19e07c3bba6.jpg>

https://static.wikia.nocookie.net/blackpink/images/7/70/Sour_Candy.jpg/revision/latest/scale-to-width-down/1000?cb=20200607184444&path-prefix=pl

4. Otwieranie i zapisywanie obrazów:



Rysunek 3: Zrzut ekranu pokazujący rozwijane menu po kliknięciu przycisku "File"

```

void openImage() //funkcja otwierająca obraz
{
    ....DialogResult dr = openFileDialog1.ShowDialog();
    ....if (dr == DialogResult.OK)
    {
        ....file = Image.FromFile(openFileDialog1.FileName);
        ....//original = Image.FromFile(openFileDialog1.FileName);
        ....newBitmap = new Bitmap(openFileDialog1.FileName);
        ....newBitmapTemp = new Bitmap(openFileDialog1.FileName);
        ....newBitmapTemp2 = new Bitmap(openFileDialog1.FileName);
        ....newBitmapSize2 = new Bitmap(openFileDialog1.FileName);
        ....pictureBox.Image = file;
        ....pictureBox1.Image = file;
        ....opened = true;
        ....Contrast_Bar.Value = 25;
        ....Brightness_Bar.Value = 1;
        ....Gamma_Bar.Value = 25;
        ....Gamma_Value_label.Text = Gamma_Bar.Value.ToString();
        ....Contrast_Value_label.Text = Contrast_Bar.Value.ToString();
    }
}

```

Rysunek 4: Zrzut ekranu pokazujący kod otwierania okna dialogowego pobierającego zdjęcie z dysku

```

void saveImage() //funkcja zapisująca obraz
{
    ....if (opened)
    {
        ....SaveFileDialog sfd = new SaveFileDialog();
        ....sfd.Filter = "Images|*.png;*.bmp;*.jpg";
        ....ImageFormat format = ImageFormat.Png; .....//z góry ustalony format to png, gdyż nie tracimy na jakości
        ....if (sfd.ShowDialog() == System.Windows.Forms.DialogResult.OK)
        {
            ....string ext = Path.GetExtension(sfd.FileName);
            ....switch (ext)
            {
                ....case ".jpg":
                    ....format = ImageFormat.Jpeg;
                    ....break;
                ....case ".bmp":
                    ....format = ImageFormat.Bmp;
                    ....break;
                ....}
                ....pictureBox.Image.Save(sfd.FileName, format);
            ....}
        ....else { MessageBox.Show("No image loaded, first upload image"); }
    }
}

```

Rysunek 5: Zrzut ekranu pokazujący kod otwierania okna dialogowego zapisującego zdjęcie z dysku

5. HUE – zmiana wartości RGB (dolny panel)

```
void hue() //funkcja zmieniająca wartości RGB obrazu
{
    float changered = Red_Bar.Value * 0.1f;
    float changegreen = Green_Bar.Value * 0.1f;
    float changeblue = Blue_Bar.Value * 0.1f;

    Red_Value_label.Text = (changered * 10).ToString();
    Blue_Value_label.Text = (changeblue * 10).ToString();
    Green_Value_label.Text = (changegreen * 10).ToString();

    reload();
    if (!opened)
    {
    }
    else
    {
        Image img = pictureBox.Image;
        Bitmap bmpInverted = new Bitmap(img.Width, img.Height);

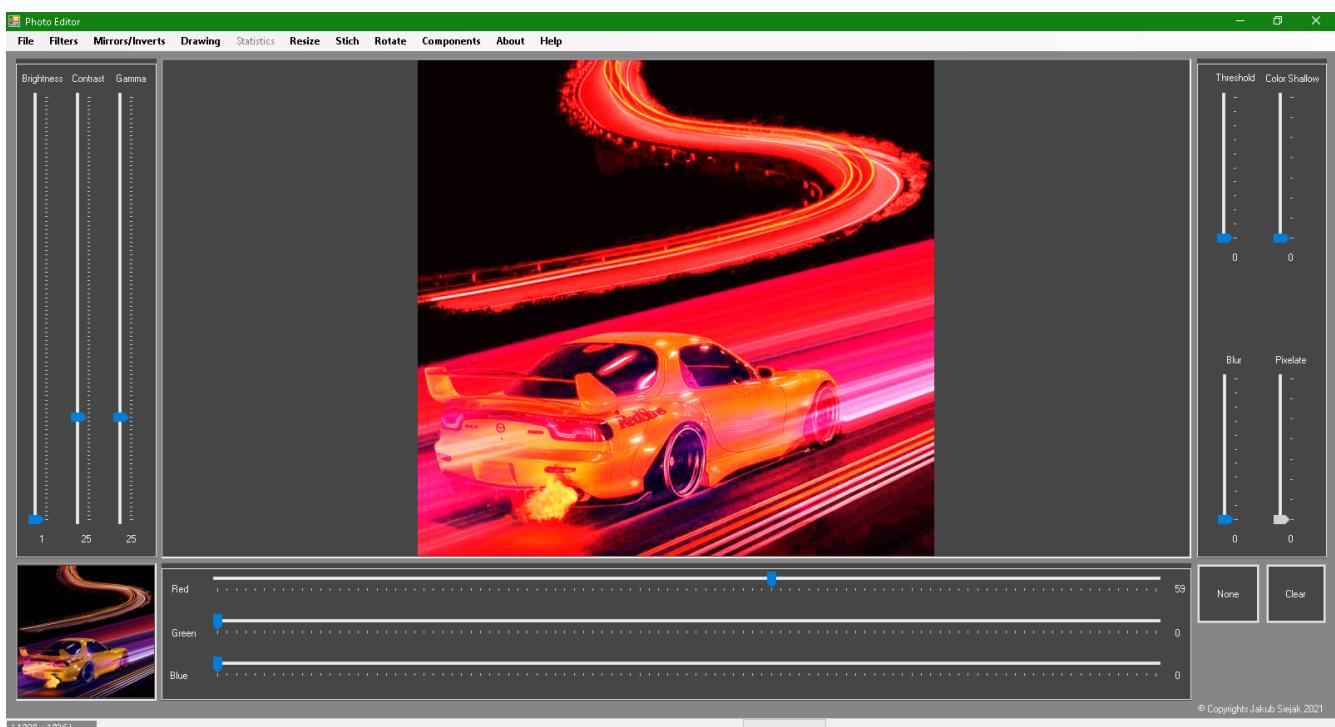
        ImageAttributes ia = new ImageAttributes();
        ColorMatrix cmPicture = new ColorMatrix(new float[][][]
        {
            new float[] {1 + changered, 0, 0, 0, 0},
            new float[] {0, 1 + changegreen, 0, 0, 0},
            new float[] {0, 0, 1 + changeblue, 0, 0},
            new float[] {0, 0, 0, 1, 0},
            new float[] {0, 0, 0, 0, 1}
        });
        iaSetColorMatrix(cmPicture);
        Graphics g = Graphics.FromImage(bmpInverted);

        g.DrawImage(img, new Rectangle(0, 0, img.Width, img.Height), 0, 0, img.Width, img.Height, GraphicsUnit.Pixel, ia);
        g.Dispose();
        pictureBox.Image = bmpInverted;
    }
}
```

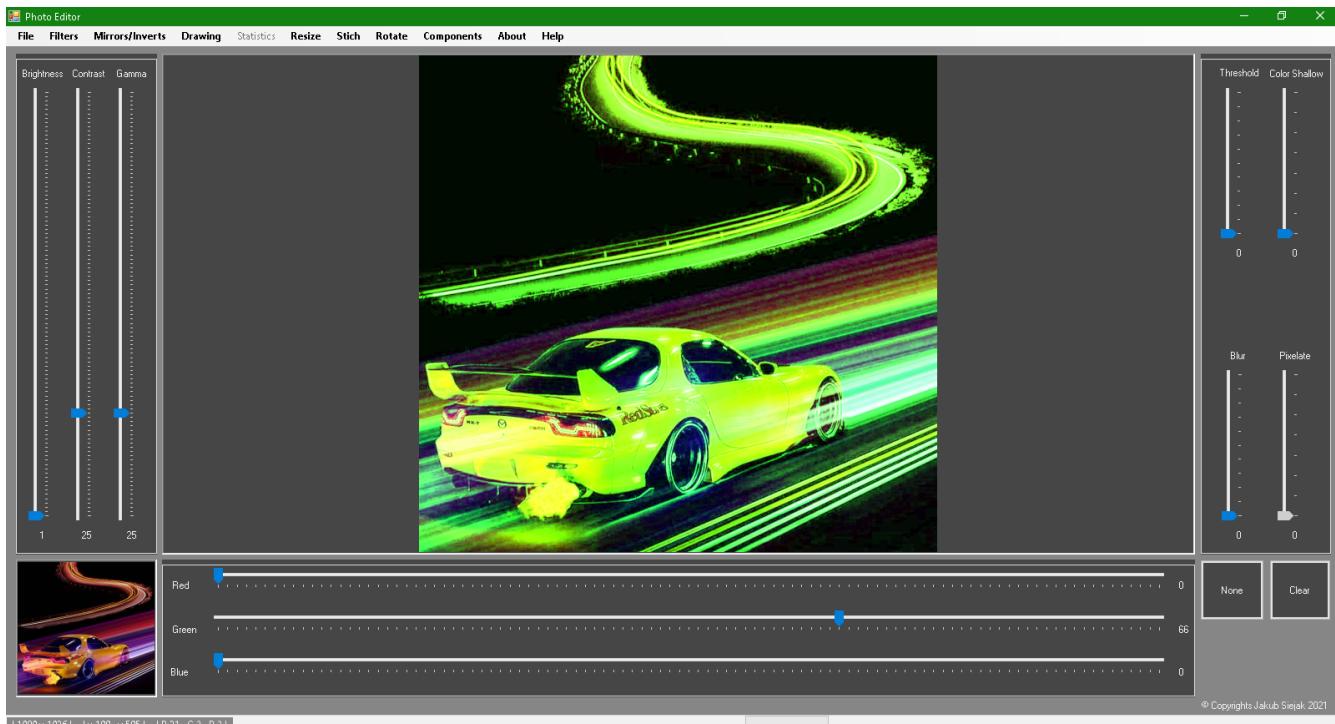
Rysunek 6: Zrzut ekranu pokazujący kod zmiany wartości rgb



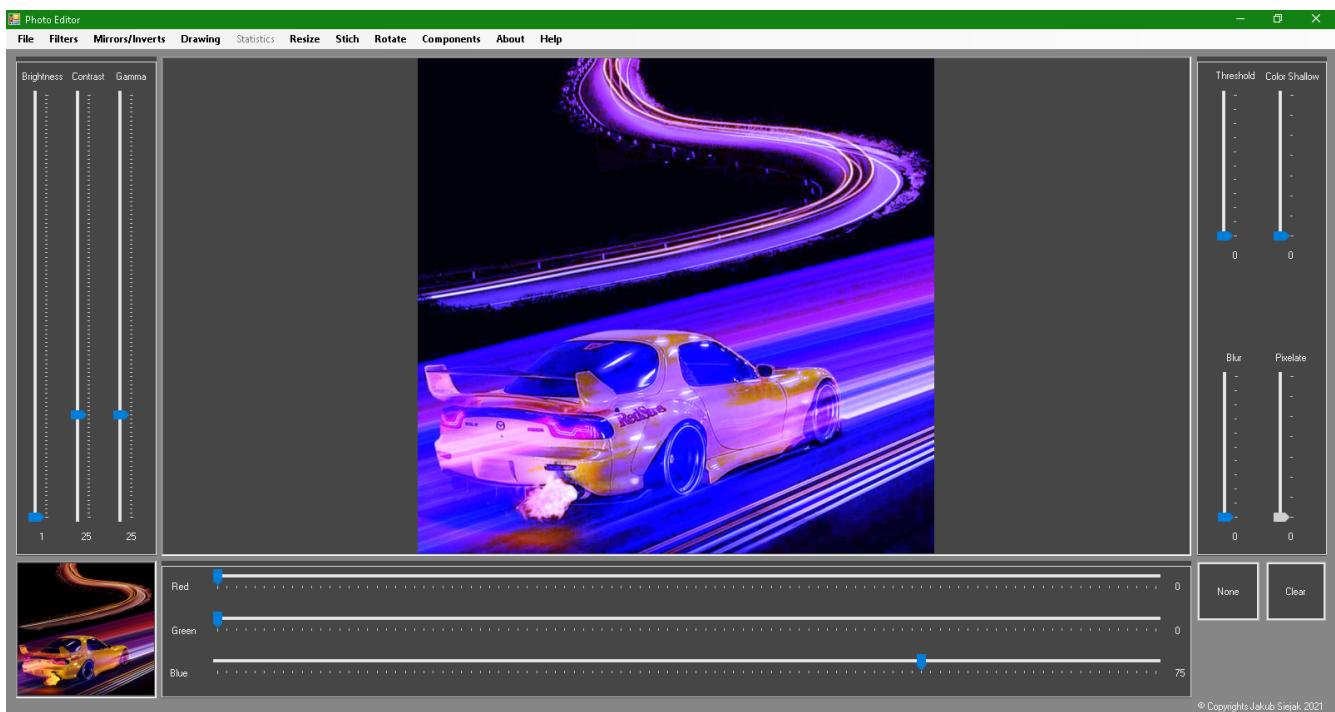
Rysunek 7: Zrzut ekranu pokazujący obraz oglądowy



Rysunek 8: Zrzut ekranu pokazujący efekt zmiany wartości suwaka R



Rysunek 9: Zrzut ekranu pokazujący efekt zmiany wartości suwaka G



Rysunek 10: Zrzut ekranu pokazujący efekt zmiany wartości suwaka B

6. Brightness – jasność

```
void brightness() //funkcja zmieniająca jasność obrazu
{
    float changebrightness = Brightness_Bar.Value / 255.0f;
    int val = (int)(changebrightness * 255);

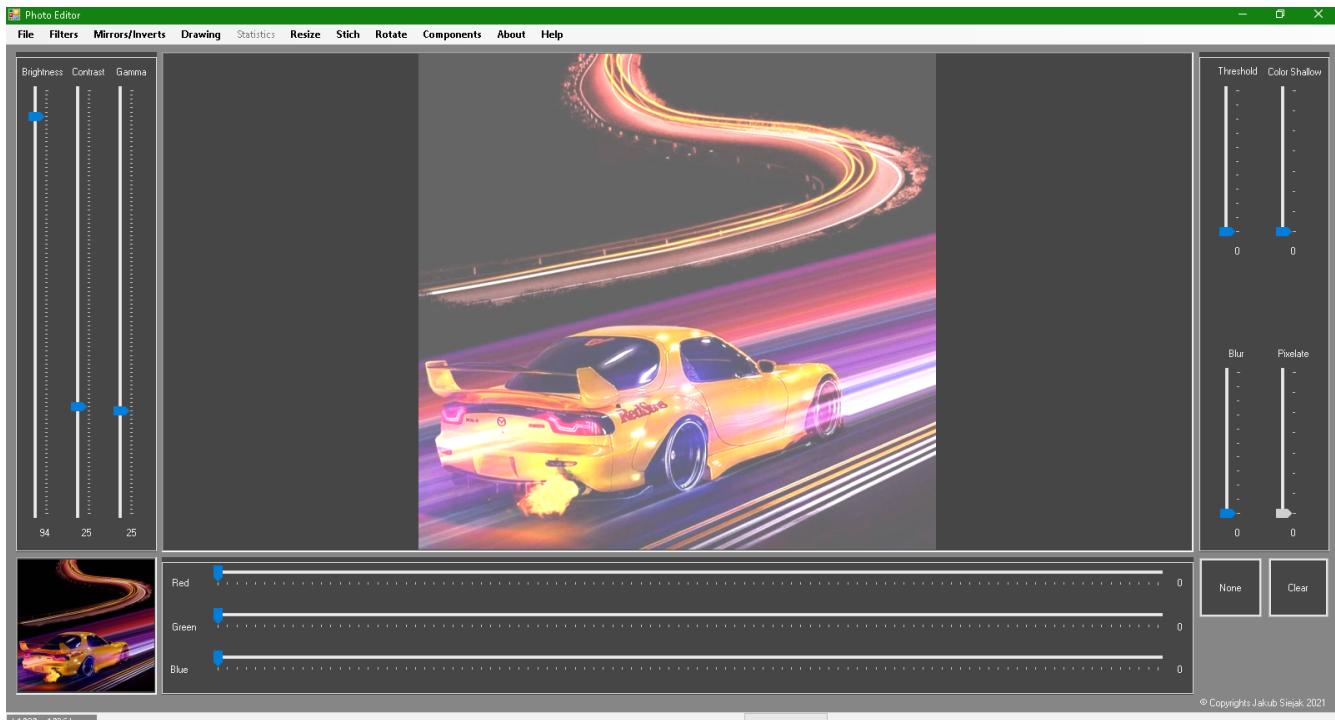
    Brightness_Value_label.Text = val.ToString();

    reload();
    if (!opened)
    {
    }
    else
    {
        Image img = pictureBox.Image;
        Bitmap bmpInverted = new Bitmap(img.Width, img.Height);

        ImageAttributes ia = new ImageAttributes();
        ColorMatrix cmPicture = new ColorMatrix(new float[][][]
        {
            new float[]{1, 0, 0, 0, 0},
            new float[]{0, 1, 0, 0, 0},
            new float[]{0, 0, 1, 0, 0},
            new float[]{0, 0, 0, 1, 0},
            new float[]{changebrightness, changebrightness, changebrightness, 1, 0}
        });
        iaSetColorMatrix(cmPicture);
        Graphics g = Graphics.FromImage(bmpInverted);

        g.DrawImage(img, new Rectangle(0, 0, img.Width, img.Height), 0, 0, img.Width, img.Height, GraphicsUnit.Pixel, ia);
        g.Dispose();
        ia.Dispose();
        pictureBox.Image = bmpInverted;
    }
}
```

Rysunek 11: Zrzut ekranu pokazujący kod odpowiadający za zmianę jasności obrazu



Rysunek 12: Zrzut ekranu pokazujący efekt zmiany wartości suwaka odpowiadającego za jasność

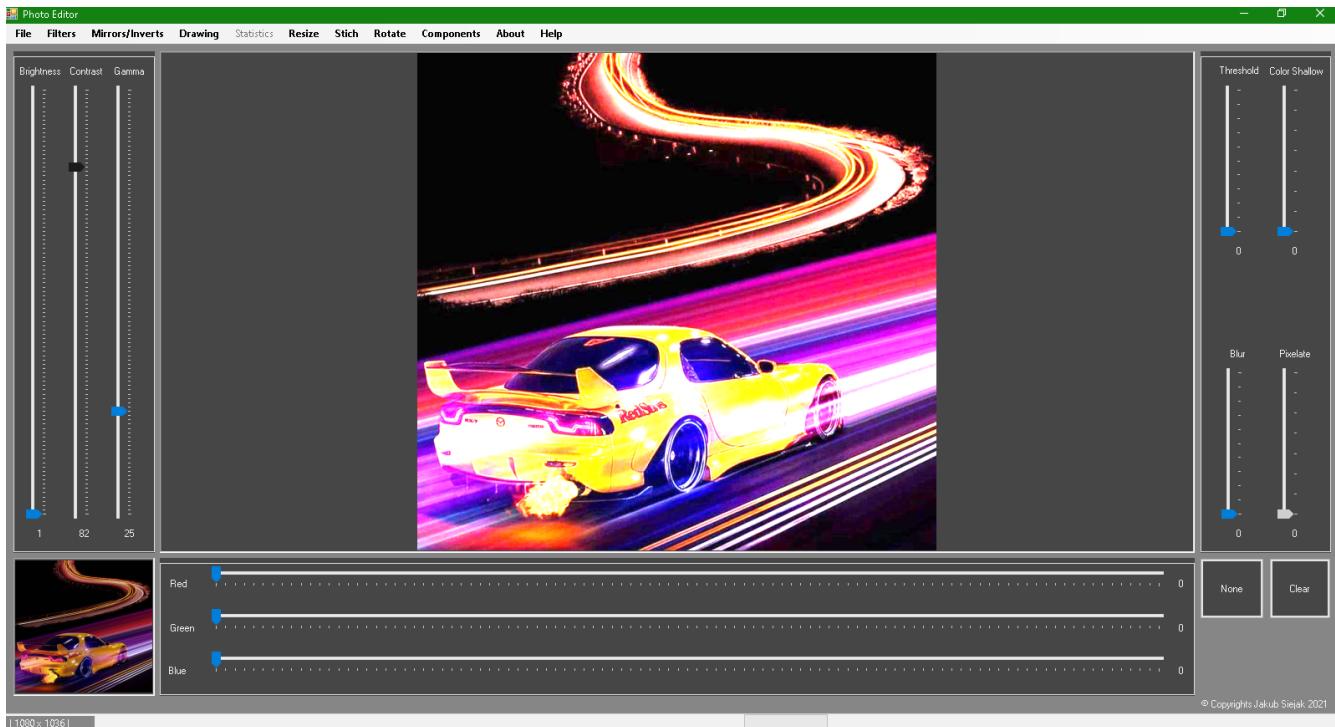
7. Contrast – kontrast:

```
void contrast() //funkcja zmieniająca kontrast obrazu
{
    float changecontrast = Contrast_Bar.Value * 0.041f;
    int val = (int)(changecontrast * 24.4);
    Contrast_Value_label.Text = val.ToString();
    reload();
    if (!opened)
    {
    }
    else
    {
        Image img = pictureBox.Image;
        Bitmap bmpInverted = new Bitmap(img.Width, img.Height);

        ImageAttributes ia = new ImageAttributes();
        ColorMatrix cmPicture = new ColorMatrix(new float[][][]
        {
            new float[] { changecontrast, 0, 0, 0, 0 },
            new float[] { 0, changecontrast, 0, 0, 0 },
            new float[] { 0, 0, changecontrast, 0, 0 },
            new float[] { 0, 0, 0, 1f, 0 },
            new float[] { 0.001f, 0.001f, 0.001f, 1, 1f }
        });
        iaSetColorMatrix(cmPicture);
        Graphics g = Graphics.FromImage(bmpInverted);

        g.DrawImage(img, new Rectangle(0, 0, img.Width, img.Height), 0, 0, img.Width, img.Height, GraphicsUnit.Pixel, ia);
        g.Dispose();
        ia.Dispose();
        pictureBox.Image = bmpInverted;
    }
}
```

Rysunek 13: Zrzut ekranu pokazujący kod odpowiadający za zmianę kontrastu obrazu



Rysunek 14: Zrzut ekranu pokazujący efekt zmiany wartości suwaka odpowiadającego za kontrast

8. Gamma:

```
void gamma() //funkcja zmieniająca współczynnik gamma obrazu
{
    .. reload();
    .. if (!opened)
    {
    .. }
    .. else
    {
        .. Gamma_Value_label.Text = Gamma_Bar.Value.ToString();

        .. float changegamma = Gamma_Bar.Value * 0.04f;

        .. //Gamma_Value_label.Text = Contrast_Bar.Value.ToString();

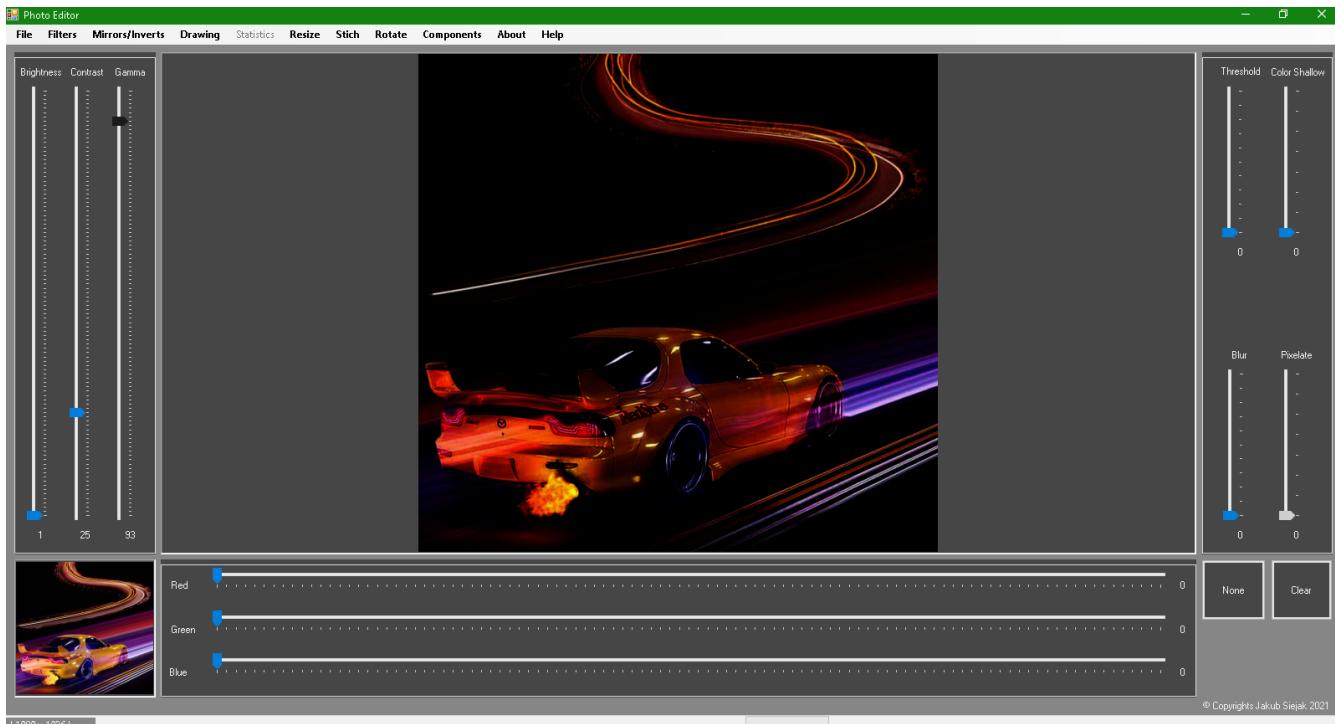
        .. Image img = pictureBox.Image;
        .. Bitmap bmpInverted = new Bitmap(newBitmap.Width, newBitmap.Height);

        .. Graphics g = Graphics.FromImage(bmpInverted);
        .. ImageAttributes ia = new ImageAttributes();

        .. ia.SetGamma(changegamma);

        .. g.DrawImage(newBitmap, new Rectangle(0, 0, newBitmap.Width, newBitmap.Height), 0, 0, newBitmap.Width, newBitmap.Height, GraphicsUnit.Pixel, ia);
        .. g.Dispose();
        .. ia.Dispose();
        .. pictureBox.Image = bmpInverted;
    }
}
```

Rysunek 15: Zrzut ekranu pokazujący kod odpowiadający za zmianę wartości gamma obrazu



Rysunek 16: Zrzut ekranu pokazujący efekt zmiany wartości suwaka odpowiadającego za współczynnik gamma

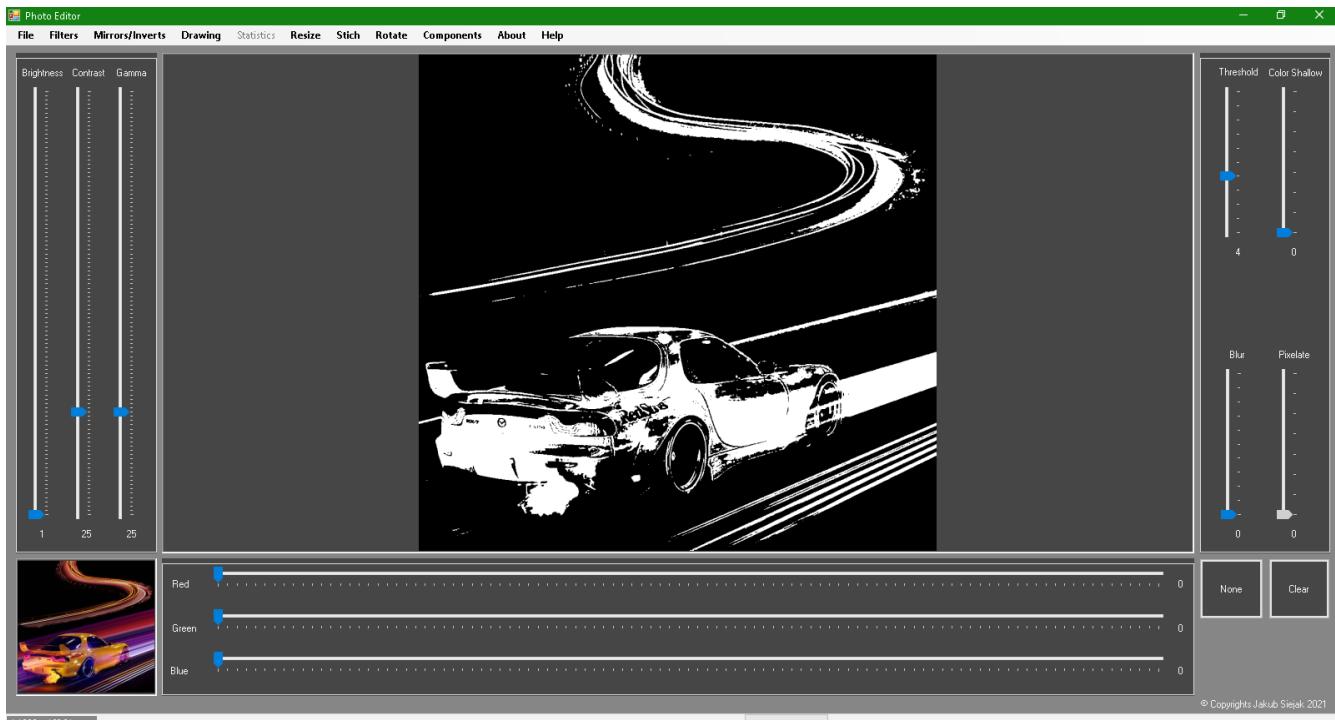
9. Threshold – binaryzacja obrazu:

```
void threshold()
{
    if (!opened)
    {
    }
    else
    {
        Threshold_Value_label.Text = Threshold_Bar.Value.ToString();

        float threshold = Threshold_Bar.Value * 25.5f;

        for (int y = 0; y < newBitmap.Height; y++)
        {
            for (int x = 0; x < newBitmap.Width; x++)
            {
                try
                {
                    if ((newBitmap.GetPixel(x, y).GetBrightness() * 255) > threshold)
                    {
                        newBitmapTemp.SetPixel(x, y, Color.White);
                    }
                    else
                    {
                        newBitmapTemp.SetPixel(x, y, Color.Black);
                    }
                }
                catch (Exception) {}
            }
            Form1_progressBar.Value = (100 * y) / newBitmap.Height;
        }
        Form1_progressBar.Value = 0;
        pictureBox.Image = newBitmapTemp;
    }
}
```

Rysunek 17: Zrzut ekranu pokazujący kod odpowiadający za binaryzację obrazu



Rysunek 18: Zrzut ekranu pokazujący efekt użycia filtru binaryzującego

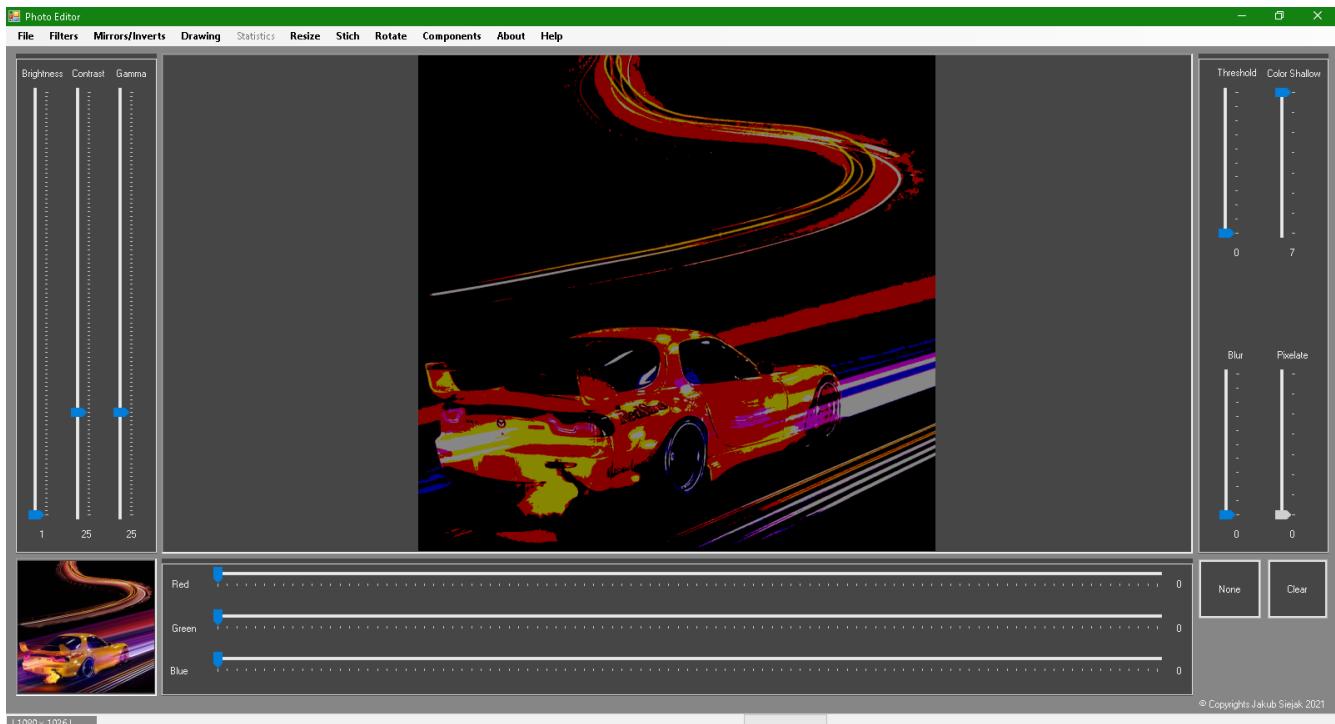
10. Color shallowing – zmniejszanie rozpiętości palety barw:

```
void color_shallowing()
{
    if (!opened)
    {
    }
    else
    {
        Color_Shallowing_Value_label.Text = Color_Shallowing_Bar.Value.ToString();

        double xx = Math.Pow(2, Color_Shallowing_Bar.Value);
        int shallow = Convert.ToInt32(xx);

        for (int y = 0; y < newBitmap.Height; y++)
        {
            for (int x = 0; x < newBitmap.Width; x++)
            {
                try
                {
                    Color pixel = newBitmap.GetPixel(x, y);
                    int R = (((int)pixel.R) / shallow) * shallow;
                    int G = (((int)pixel.G) / shallow) * shallow;
                    int B = (((int)pixel.B) / shallow) * shallow;
                    newBitmapTemp.SetPixel(x, y, Color.FromArgb(R, G, B));
                }
                catch (Exception) {}
            }
            Form1_progressBar.Value = (100 * y) / newBitmap.Height;
        }
        Form1_progressBar.Value = 0;
        pictureBox.Image = newBitmapTemp;
    }
}
```

Rysunek 19: Zrzut ekranu pokazujący kod odpowiadający za zmniejszenie rozpiętości palety kolorystycznej obrazu



Rysunek 20: Zrzut ekranu pokazujący efekt użycia filtru spłycającego kolory

11. Blur – rozmywanie obrazu:

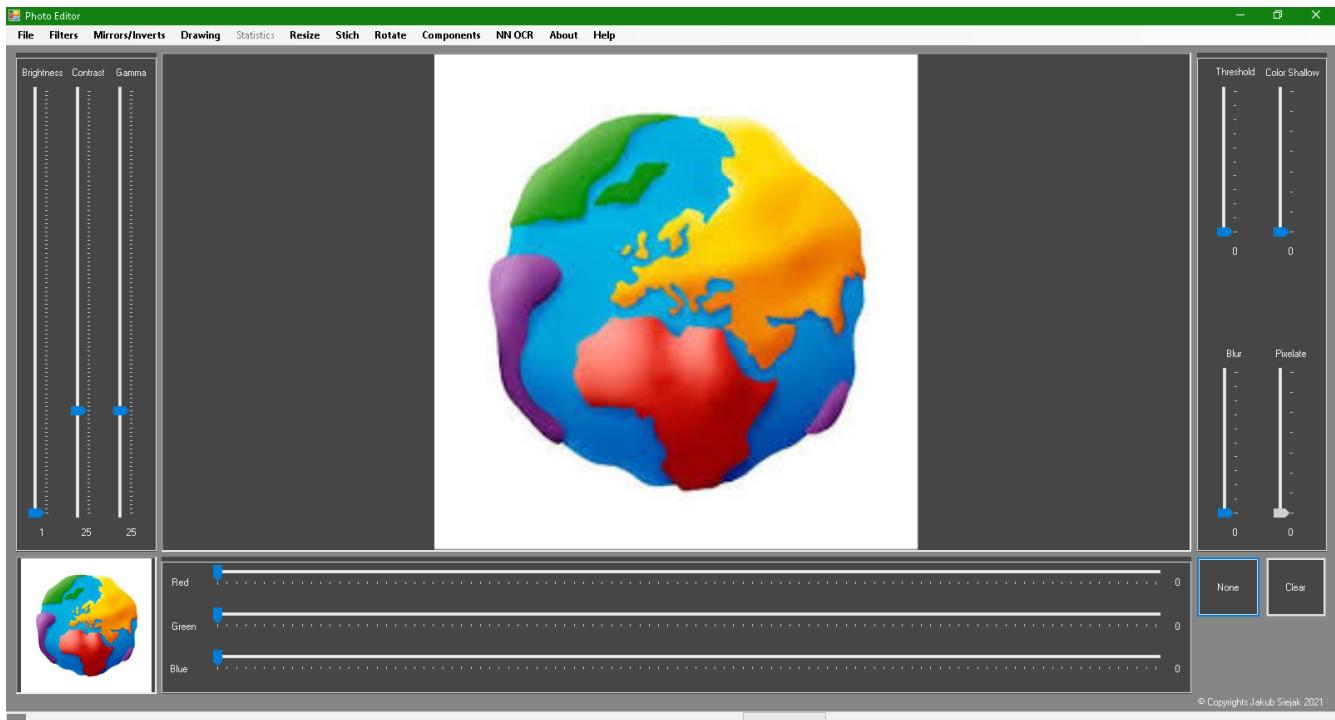
```
void blur() //funkcja rozmywająca obraz poprzez uśrednianie wartości pikseli
{
    if (!opened)
    {
        MessageBox.Show("Open an Image then apply changes");
    }
    else
    {
        /*var options = new ParallelOptions()
        {
            MaxDegreeOfParallelism = 32000
        };*/
        Blur_Value_label.Text = Blur_Bar.Value.ToString();
        newBitmapTemp = newBitmap;
        int blur = Convert.ToInt32(Blur_Bar.Value * 10);

        for (int i = 0; i < blur; i++)
        {
            for (int x = 0; x < newBitmap.Width; x++)
            //Parallel.For(0, newBitmap.Width, x =>
            {
                //Parallel.For(0, newBitmap.Height, y =>
                //for (int y = 0; y < newBitmap.Height; y++)
                {
                    try
                    {
                        Color prevX = newBitmapTemp.GetPixel(x - 1, y);
                        Color nextX = newBitmapTemp.GetPixel(x + 1, y);
                        Color prevY = newBitmapTemp.GetPixel(x, y - 1);
                        Color nextY = newBitmapTemp.GetPixel(x, y + 1);

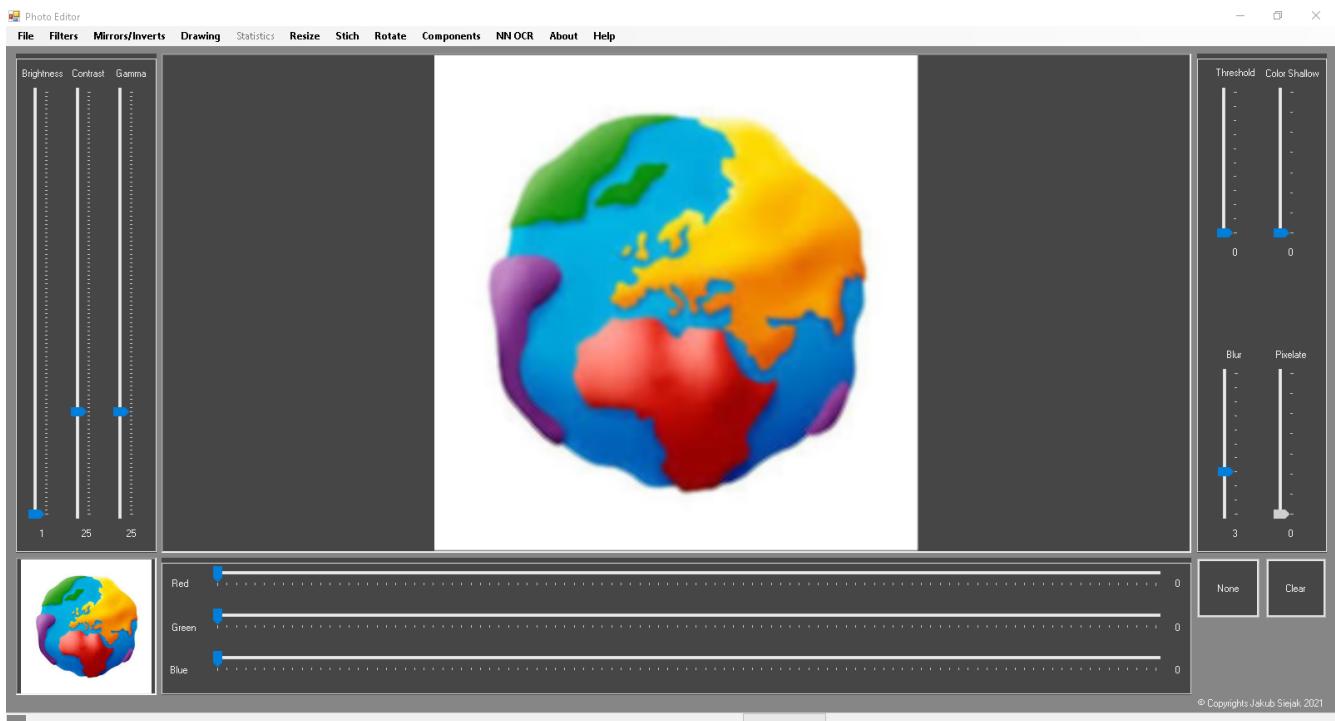
                        int avgR = (int)((prevX.R + nextX.R + prevY.R + nextY.R) / 4);
                        int avgG = (int)((prevX.G + nextX.G + prevY.G + nextY.G) / 4);
                        int avgB = (int)((prevX.B + nextX.B + prevY.B + nextY.B) / 4);

                        newBitmapTemp2.SetPixel(x, y, Color.FromArgb(avgR, avgG, avgB));
                    }
                    catch (Exception){ }
                });
            } //);
            Form1_progressBar.Value = (100 * i) / blur;
        }
        pictureBox.Image = newBitmapTemp2;
        Form1_progressBar.Value = 0;
    }
}
```

Rysunek 21: Zrzut ekranu pokazujący kod odpowiadający za rozmywanie obrazu



Rysunek 22: Zrzut ekranu przedstawiający obraz poglądowy



Rysunek 23: Zrzut ekranu pokazujący efekt użycia filtru rozmazującego obraz

12. Grayscale

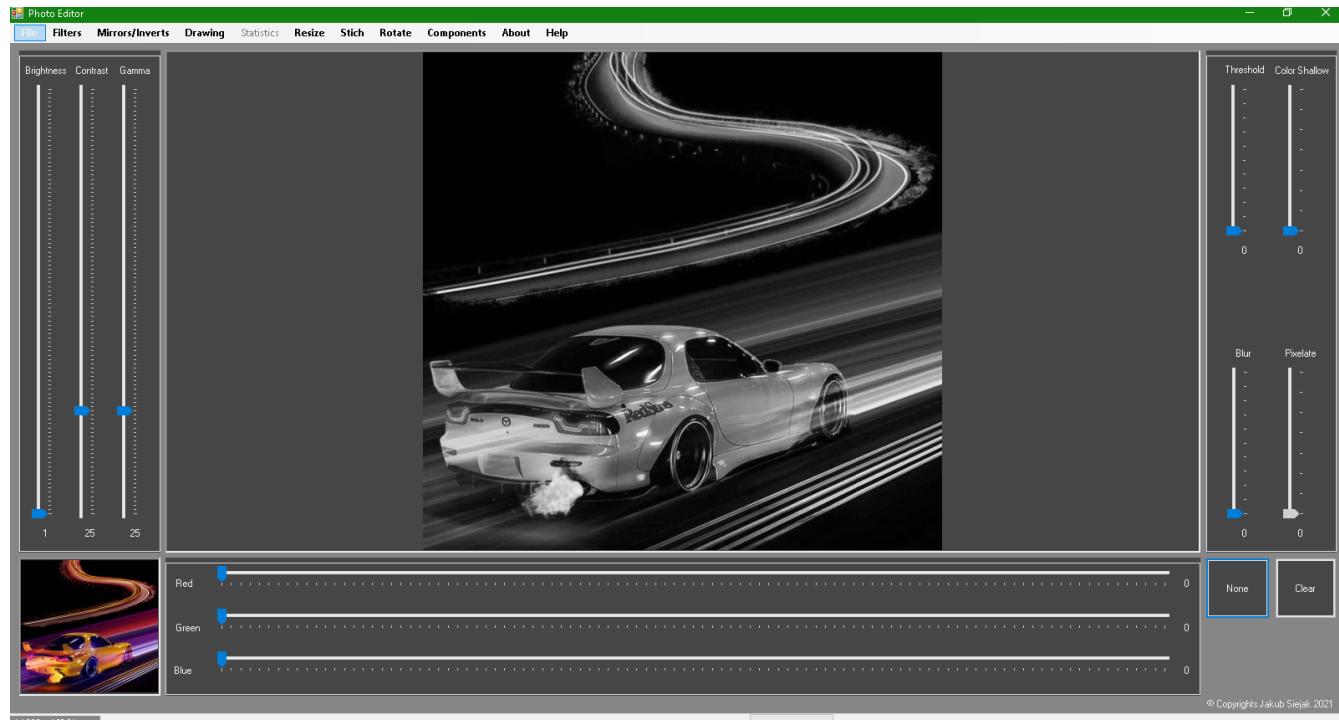
```
void grayscale() //funkcja zmieniająca kolory w odcienie szarości || brak progressbara i wielowątkowości
{
    if (!opened)
    {
        MessageBox.Show("Open an Image then apply changes");
    }
    else
    {
        Red_Bar.Value = 0;
        Green_Bar.Value = 0;
        Blue_Bar.Value = 0;
        Image img = pictureBox.Image;
        Bitmap bmpInverted = new Bitmap(img.Width, img.Height);

        ImageAttributes ia = new ImageAttributes();
        ColorMatrix cmPicture = new ColorMatrix(new float[][][]
        {
            new float[]{0.299f, 0.299f, 0.299f, 0, 0},
            new float[]{0.587f, 0.587f, 0.587f, 0, 0},
            new float[]{0.114f, 0.114f, 0.114f, 0, 0},
            new float[]{0, 0, 0, 1, 0},
            new float[]{0, 0, 0, 0, 0}
        });
        iaSetColorMatrix(cmPicture);
        Graphics g = Graphics.FromImage(bmpInverted);

        g.DrawImage(img, new Rectangle(0, 0, img.Width, img.Height), 0, 0, img.Width, img.Height, GraphicsUnit.Pixel, ia);

        g.Dispose();
        pictureBox.Image = bmpInverted;
    }
}
```

Rysunek 24: Zrzut ekranu pokazujący kod odpowiadający za zmianę obrazu na obraz w odcieniach szarości



Rysunek 25: Zrzut ekranu pokazujący efekt użycia filtru zamieniającego obraz na kolory szarości

13. Sepia

```
void sephia() //funkcja zmieniająca oryginalny obraz w jego odpowiednik w kolorach sepii || brak progres i wielo
{
    if (!opened)
    {
        MessageBox.Show("Open an Image then apply changes");
    }
    else
    {
        Red_Bar.Value = 0;
        Green_Bar.Value = 0;
        Blue_Bar.Value = 0;
        Image img = pictureBox.Image;
        Bitmap bmpInverted = new Bitmap(img.Width, img.Height);

        ImageAttributes ia = new ImageAttributes();
        ColorMatrix cmPicture = new ColorMatrix(new float[][][]
        {
            new float[]{0.393f, 0.349f, 0.272f, 0, 0},
            new float[]{0.769f, 0.686f, 0.534f, 0, 0},
            new float[]{0.189f, 0.168f, 0.131f, 0, 0},
            new float[]{0, 0, 0, 1, 0},
            new float[]{0, 0, 0, 0, 1}
        });
        iaSetColorMatrix(cmPicture);
        Graphics g = Graphics.FromImage(bmpInverted);

        g.DrawImage(img, new Rectangle(0, 0, img.Width, img.Height), 0, 0, img.Width, img.Height, GraphicsUnit.Pixel, ia);

        g.Dispose();
        pictureBox.Image = bmpInverted;
    }
}
```

Rysunek 26: Zrzut ekranu pokazujący kod odpowiadający za zmianę obrazu na obraz w odcieniach brązu



Rysunek 27: Zrzut ekranu pokazujący efekt użycia filtru zmieniającego obraz na odcienie brązu

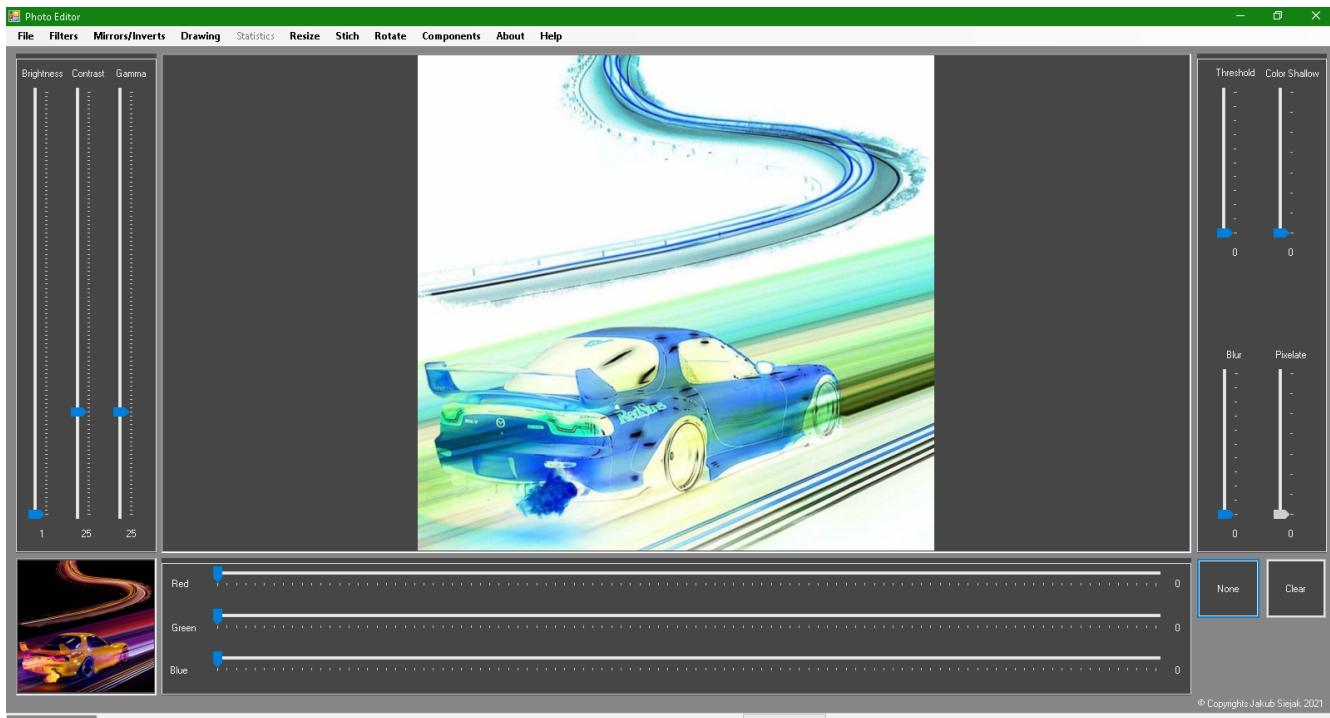
14. Negative

```
void invert() // funkcja odwracająca kolory(negatyw)
{
    if (!opened)
    {
        MessageBox.Show("Open an Image then apply changes");
    }
    else
    {
        for (int x = 0; x < newBitmap.Width; x++)
        {
            for (int y = 0; y < newBitmap.Height; y++)
            {
                try
                {
                    Color pixel = newBitmap.GetPixel(x, y);

                    int red = pixel.R;
                    int green = pixel.G;
                    int blue = pixel.B;

                    newBitmap.SetPixel(x, y, Color.FromArgb(255 - red, 255 - green, 255 - blue));
                }
                catch (Exception) {}
            }
            Form1_progressBar.Value = (x * 100) / newBitmap.Width;
        }
        pictureBox.Image = newBitmap;
        Form1_progressBar.Value = 0;
    }
}
```

Rysunek 28: Zrzut ekranu pokazujący kod odpowiadający za odwracanie kolorystyki obrazu



Rysunek 29: Zrzut ekranu pokazujacy efekt użycia filtru odwracajqcego kolory

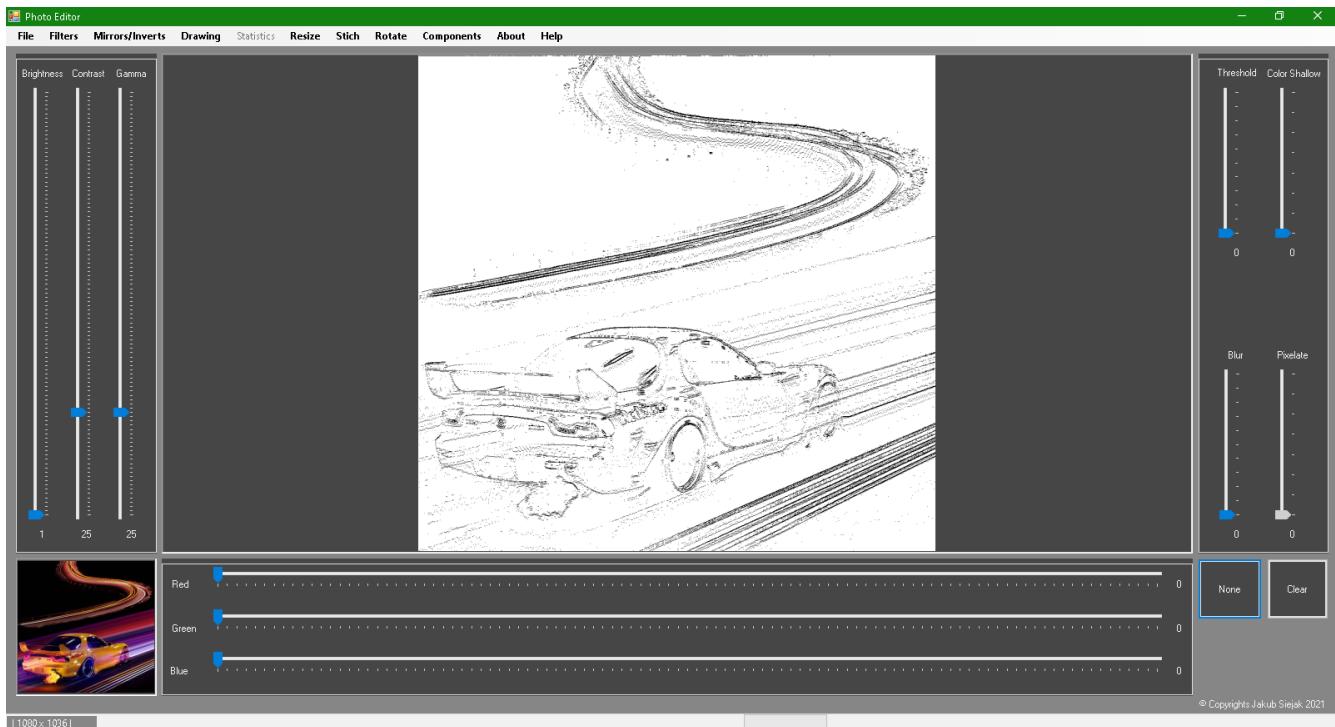
15. Edge (Black on White)

```
691     void edge() //funkcja znajdująca krawędzie na podstawie różnicy wartości pikseli(imitacja emboss)
692     {
693         if (!opened)
694         {
695             MessageBox.Show("Open an Image then apply changes");
696         }
697         else
698         {
699              Bitmap nB = new Bitmap(newBitmap.Width, newBitmap.Height);
700             for (int x = 0; x <= newBitmap.Width - 1; x++)
701             {
702                 for (int y = 0; y <= newBitmap.Height - 1; y++)
703                 {
704                     nB.SetPixel(x, y, Color.White);
705                 }
706             }
707             for (int x = 1; x < newBitmap.Width; x++)
708             {
709                 for (int y = 1; y < newBitmap.Height; y++)
710                 {
711                     try
712                     {
713                         Color pixel = newBitmap.GetPixel(x, y);
714                         int colVal = (pixel.R + pixel.G + pixel.B);
715                         if (lastCol == 0) lastCol = (pixel.R + pixel.G + pixel.B);
716                         int diff;
717                         if (colVal > lastCol)
718                         {
719                             diff = colVal - lastCol;
720                         }
721                         else
722                         {
723                             diff = lastCol - colVal;
724                         }
725                         if (diff > 100)
726                         {
727                             nB.SetPixel(x, y, Color.Black);
728                             lastCol = colVal;
729                         }
730                     }
731                     catch (Exception) { }
732                 }
733             }
734         }
735     }
```

Rysunek 30: Zrzut ekranu pokazujący kod odpowiadający za pozostawianie krawędzi obrazu w kolorze czarnym na białym tle cz.1

```
733     for (int y = 1; y < newBitmap.Height; y++)
734     {
735         try
736         {
737             Color pixel = newBitmap.GetPixel(x, y);
738             int colVal = (pixel.R + pixel.G + pixel.B);
739             if (lastCol == 0) lastCol = (pixel.R + pixel.G + pixel.B);
740             int diff;
741             if (colVal > lastCol)
742             {
743                 diff = colVal - lastCol;
744             }
745             else
746             {
747                 diff = lastCol - colVal;
748             }
749             if (diff > 100)
750             {
751                 nB.SetPixel(x, y, Color.Black);
752                 lastCol = colVal;
753             }
754         }
755         catch (Exception) { }
756     }
757     Form1_progressBar.Value = (100 * x) / newBitmap.Width;
758 }
759 pictureBox.Image = nB;
760 Form1_progressBar.Value = 0;
761 }
762 }
```

Rysunek 31: Zrzut ekranu pokazujący kod odpowiadający za pozostawianie krawędzi obrazu w kolorze czarnym na białym tle cz.2



Rysunek 32: Zrzut ekranu pokazujący efekt użycia filtru zaznaczającego czarne krawędzie na białym tle

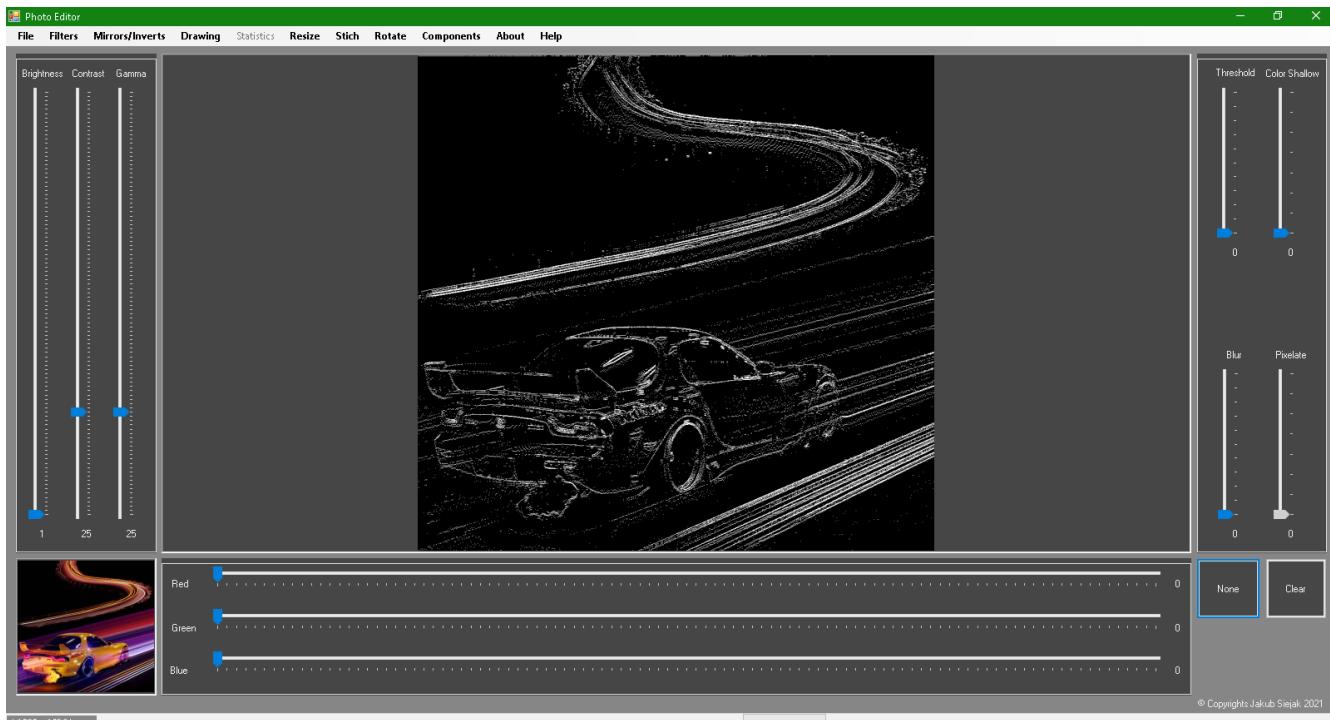
16. Edge (White on Black)

```
764     void edge_WoB() //funkcja znajdująca krawędzie na podstawie różnicy wartości pikseli( imitacja emboss)
765     {
766         if (!opened)
767         {
768             MessageBox.Show("Open an Image then apply changes");
769         }
770         else
771         {
772             Bitmap nB = new Bitmap(newBitmap.Width, newBitmap.Height);
773             for (int x = 0; x <= newBitmap.Width - 1; x++)
774             {
775                 for (int y = 0; y <= newBitmap.Height - 1; y++)
776                 {
777                     nB.SetPixel(x, y, Color.Black);
778                 }
779             }
780             for (int x = 1; x < newBitmap.Width; x++)
781             {
782                 for (int y = 1; y <= newBitmap.Height - 1; y++)
783                 {
784                     try
785                     {
786                         Color pixel = newBitmap.GetPixel(x, y);
787                         int colVal = (pixel.R + pixel.G + pixel.B);
788                         if (lastCol == 0) lastCol = (pixel.R + pixel.G + pixel.B);
789                         int diff;
790                         if (colVal > lastCol)
791                         {
792                             diff = colVal - lastCol;
793                         }
794                         else
795                         {
796                             diff = lastCol - colVal;
797                         }
798                         if (diff > 100)
799                         {
800                             nB.SetPixel(x, y, Color.White);
801                             lastCol = colVal;
802                         }
803                     }
804                 }
805             }
806         }
807     }
```

Rysunek 33: Zrzut ekranu pokazujący kod odpowiadający za pozostawianie krawędzi obrazu w kolorze białym na czarnym tle cz.1

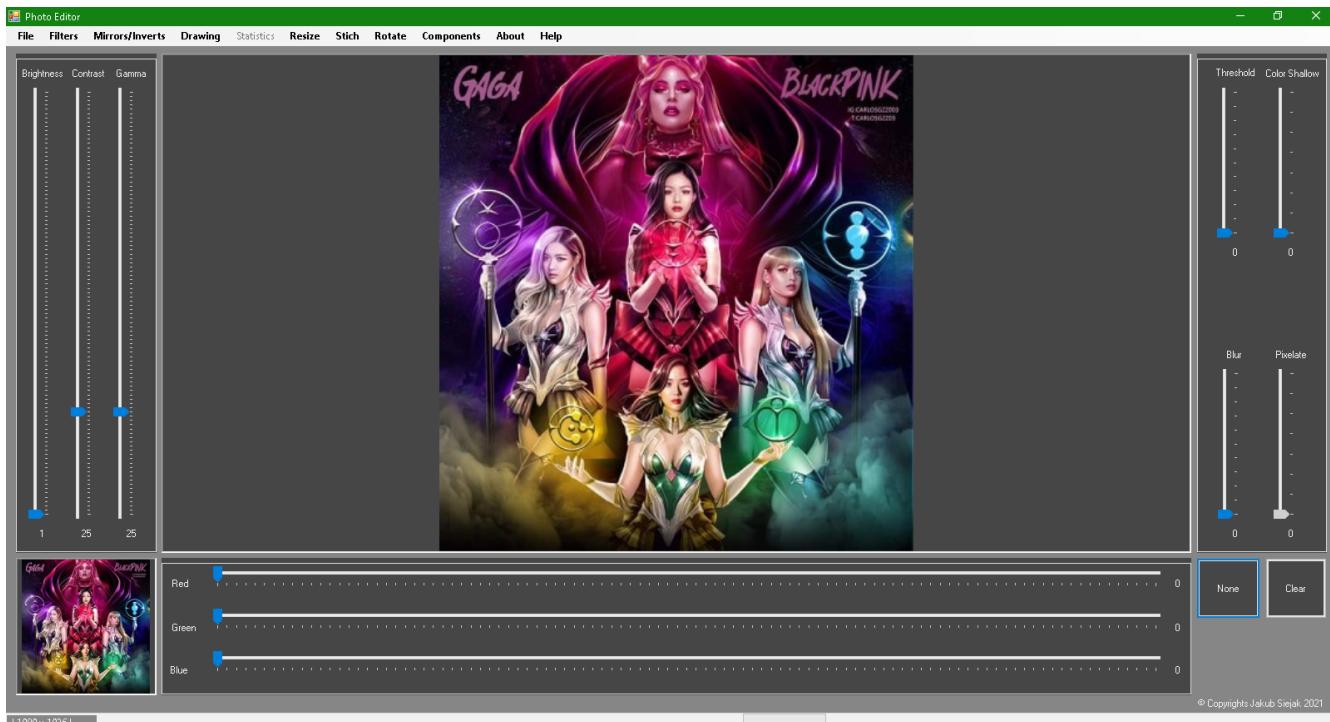
```
805     }  
806     catch (Exception){}  
807     }  
808     for (int y = 1;y <=newBitmap.Height - 1;y++)  
809     {  
810         try  
811         {  
812             Color pixel = newBitmap.GetPixel(x, y);  
813             int colVal = (pixel.R + pixel.G + pixel.B);  
814             if (lastCol == 0) lastCol = (pixel.R + pixel.G + pixel.B);  
815             int diff;  
816             if (colVal > lastCol)  
817             {  
818                 diff = colVal - lastCol;  
819             }  
820             else  
821             {  
822                 diff = lastCol - colVal;  
823             }  
824             if (diff > 100)  
825             {  
826                 nB.SetPixel(x, y, Color.White);  
827             }  
828         }  
829         catch (Exception){}  
830     }  
831     Form1_progressBar.Value = (100 * x) / newBitmap.Width;  
832 }  
833 pictureBox.Image = nB;  
834 Form1_progressBar.Value = 0;  
835 }  
836 }
```

Rysunek 34: Zrzut ekranu pokazujący kod odpowiadający za pozostawianie krawędzi obrazu w kolorze białym na czarnym tle cz.2



Rysunek 35: Zrzut ekranu pokazujący efekt użycia filtru zaznaczającego białe krawędzie na czarnym tle

17. Basic Extraction



Rysunek 36: Zrzut ekranu przedstawiający obraz poglądowy

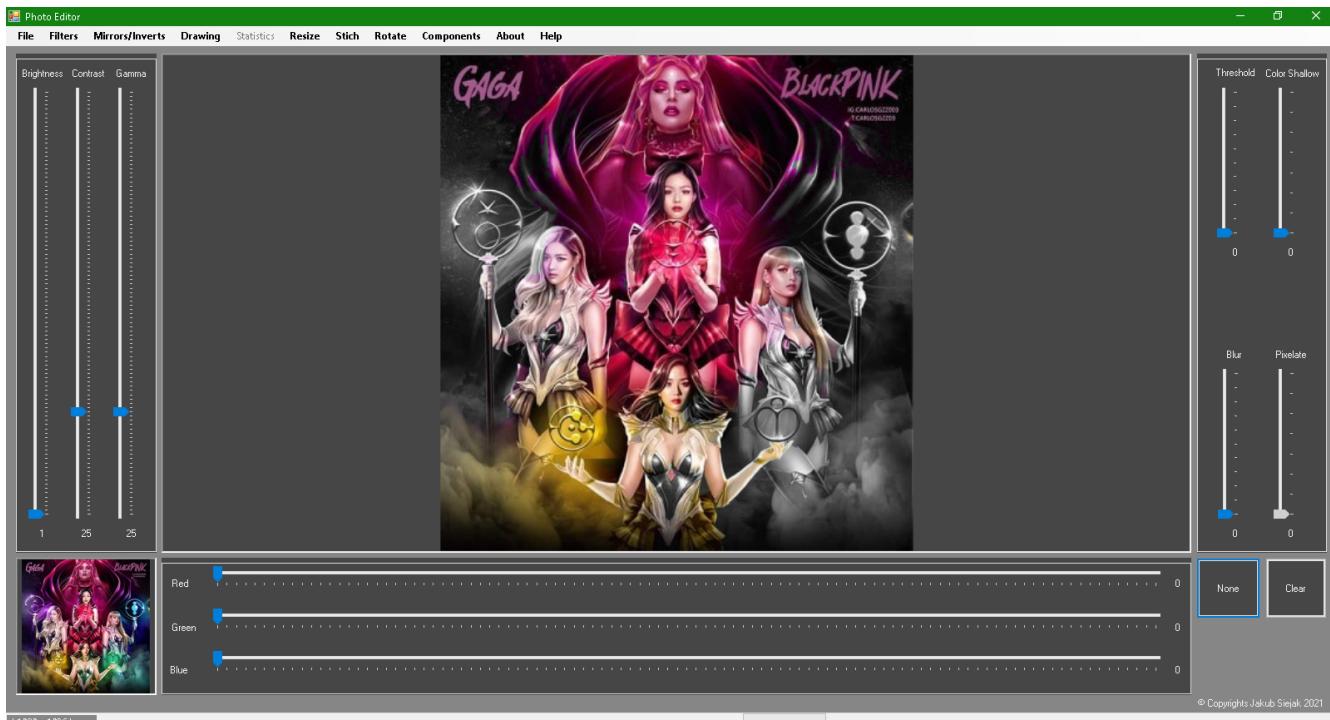
bR-Extraction

```
void comp_bR()
{
    if (!opened)
    {
        MessageBox.Show("Open an Image then apply changes");
    }
    else
    {
        int width = pictureBox.Image.Width;
        int height = pictureBox.Image.Height;

        Bitmap newBitmapTemp = new Bitmap(width, height);

        for (int y = 0; y < newBitmap.Height; y++)
        {
            for (int x = 0; x < newBitmap.Width; x++)
            {
                try
                {
                    if ((x > 0) && (x <= width))
                    {
                        Color pixel = newBitmap.GetPixel(x, y);
                        int R = (int)pixel.R;
                        int G = (int)pixel.G;
                        int B = (int)pixel.B;
                        if (((R > G) && (R > B)))
                        {
                            newBitmapTemp.SetPixel(x, y, Color.FromArgb(R, G, B));
                        }
                        else
                        {
                            int GRAY = (((R) + (G) + (B)) / 3);
                            newBitmapTemp.SetPixel(x, y, Color.FromArgb(GRAY, GRAY, GRAY));
                        }
                    }
                }
                catch (Exception){ }
            }
            Form1.progressBar.Value = (100 * y) / newBitmap.Height;
        }
        newBitmap = newBitmapTemp;
        pictureBox.Image = newBitmap;
        Form1.progressBar.Value = 0;
    }
}
```

Rysunek 37: Zrzut ekranu pokazujący kod odpowiadający za ekstrakcję uogólnionego koloru czerwonego



Rysunek 38: Zrzut ekranu pokazujący efekt użycia filtru wyodrębniającego uogólniony kolor czerwony

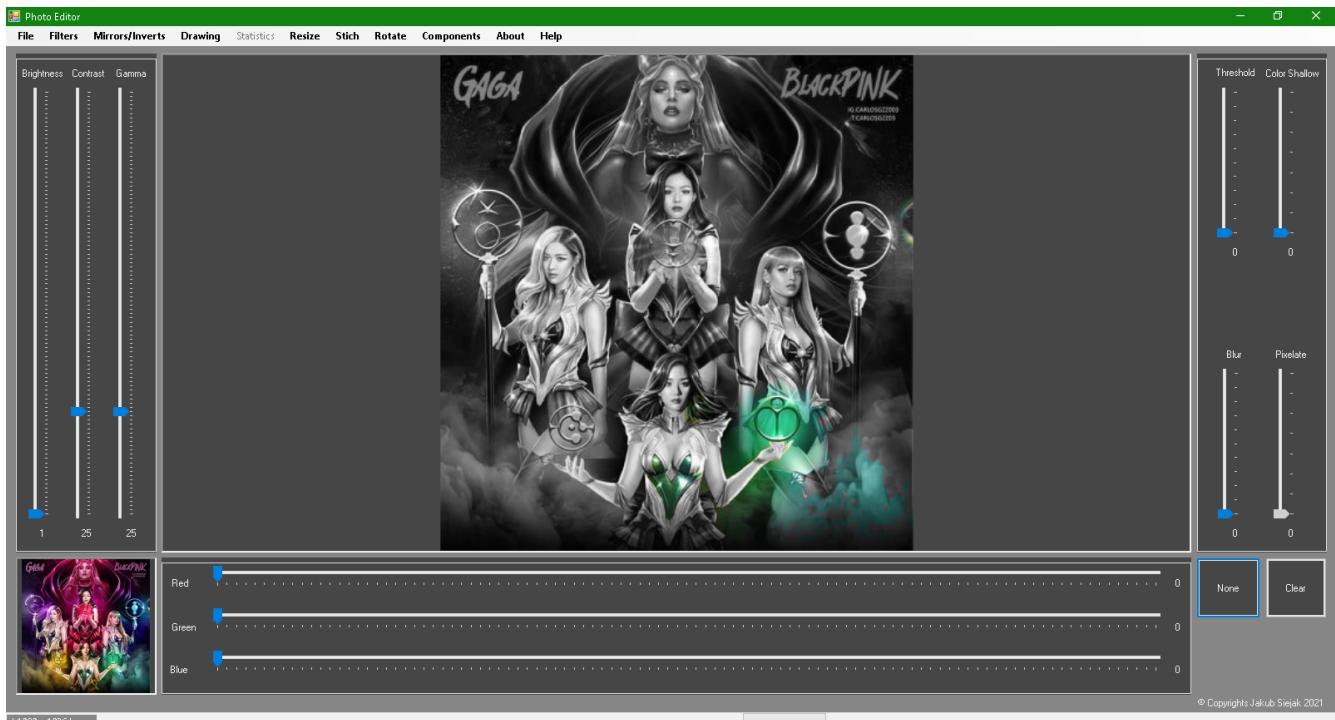
bG-Extraction

```
void comp_bG()
{
    if (!opened)
    {
        MessageBox.Show("Open an Image then apply changes");
    }
    else
    {
        int width = pictureBox.Image.Width;
        int height = pictureBox.Image.Height;

        Bitmap newBitmapTemp = new Bitmap(width, height);

        for (int y = 0; y < newBitmap.Height; y++)
        {
            for (int x = 0; x < newBitmap.Width; x++)
            {
                try
                {
                    if ((x > 0) && (x <= width))
                    {
                        Color pixel = newBitmap.GetPixel(x, y);
                        int R = (int)pixel.R;
                        int G = (int)pixel.G;
                        int B = (int)pixel.B;
                        if (((G > R) && (G > B)))
                        {
                            newBitmapTemp.SetPixel(x, y, Color.FromArgb(R, G, B));
                        }
                        else
                        {
                            int GRAY = (((R) + (G) + (B)) / 3);
                            newBitmapTemp.SetPixel(x, y, Color.FromArgb(GRAY, GRAY, GRAY));
                        }
                    }
                }
                catch (Exception) { }
            }
            Form1_progressBar.Value = (100 * y) / newBitmap.Height;
        }
        newBitmap = newBitmapTemp;
        pictureBox.Image = newBitmap;
        Form1_progressBar.Value = 0;
    }
}
```

Rysunek 39: Zrzut ekranu pokazujący kod odpowiadający za ekstrakcję uogólnionego koloru zielonego



Rysunek 40: Zrzut ekranu pokazujący efekt użycia filtru wyodrębniającego uogólniony kolor zielony

bB-Extraction

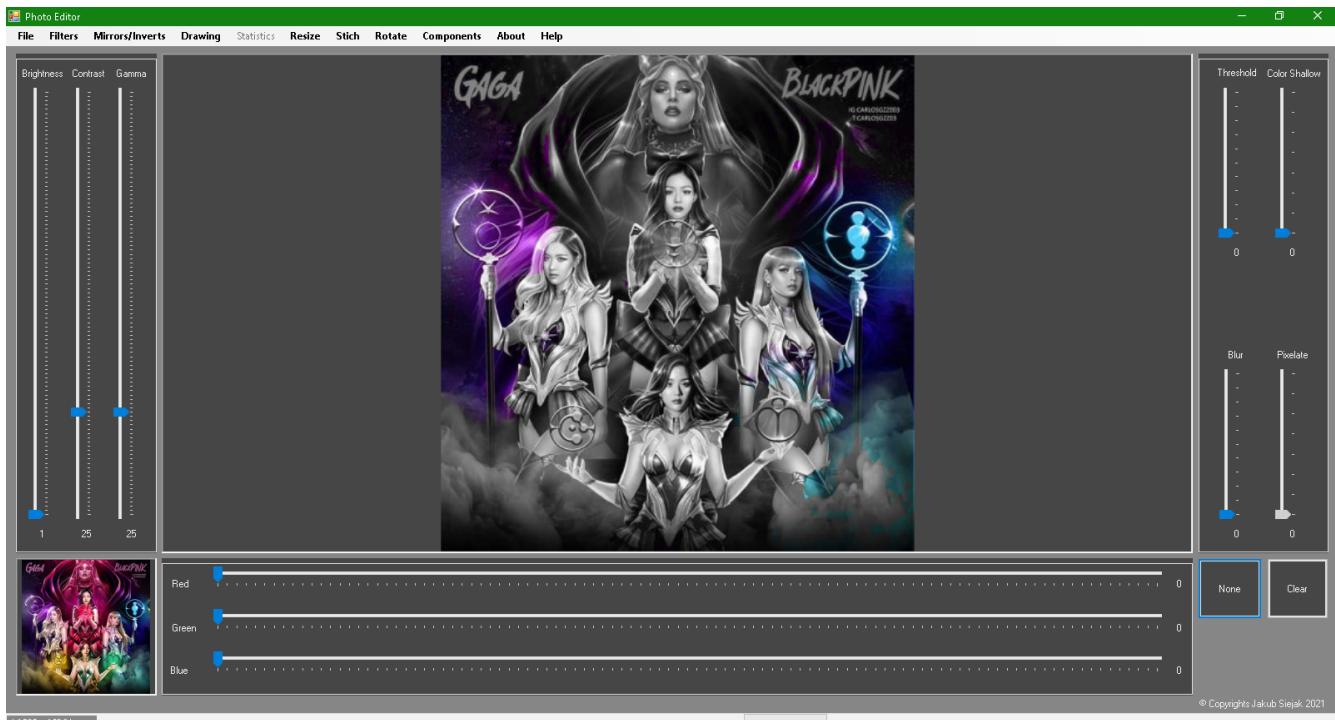
```
void comp_bB()
{
    if (!opened)
    {
        MessageBox.Show("Open an Image then apply changes");
    }
    else
    {
        int width = pictureBox.Image.Width;
        int height = pictureBox.Image.Height;

        Bitmap newBitmapTemp = new Bitmap(width, height);

        for (int y = 0; y < newBitmap.Height; y++)
        {
            for (int x = 0; x < newBitmap.Width; x++)
            {
                try
                {
                    if ((x > 0) && (x <= width))
                    {
                        Color pixel = newBitmap.GetPixel(x, y);
                        int R = (int)pixel.R;
                        int G = (int)pixel.G;
                        int B = (int)pixel.B;
                        if (((B > G) && (B > R)))
                        {
                            newBitmapTemp.SetPixel(x, y, Color.FromArgb(R, G, B));
                        }
                        else
                        {
                            int GRAY = (((R) + (G) + (B)) / 3);
                            newBitmapTemp.SetPixel(x, y, Color.FromArgb(GRAY, GRAY, GRAY));
                        }
                    }
                }
                catch (Exception){}

                Form1.progressBar.Value = (100 * y) / newBitmap.Height;
            }
            newBitmap = newBitmapTemp;
            pictureBox.Image = newBitmap;
            Form1.progressBar.Value = 0;
        }
    }
}
```

Rysunek 41: Zrzut ekranu pokazujący kod odpowiadający za ekstrakcję uogólnionego koloru niebieskiego



Rysunek 42: Zrzut ekranu pokazujący efekt użycia filtru wyodrębniającego uogólniony kolor niebieski

18. Real Extraction

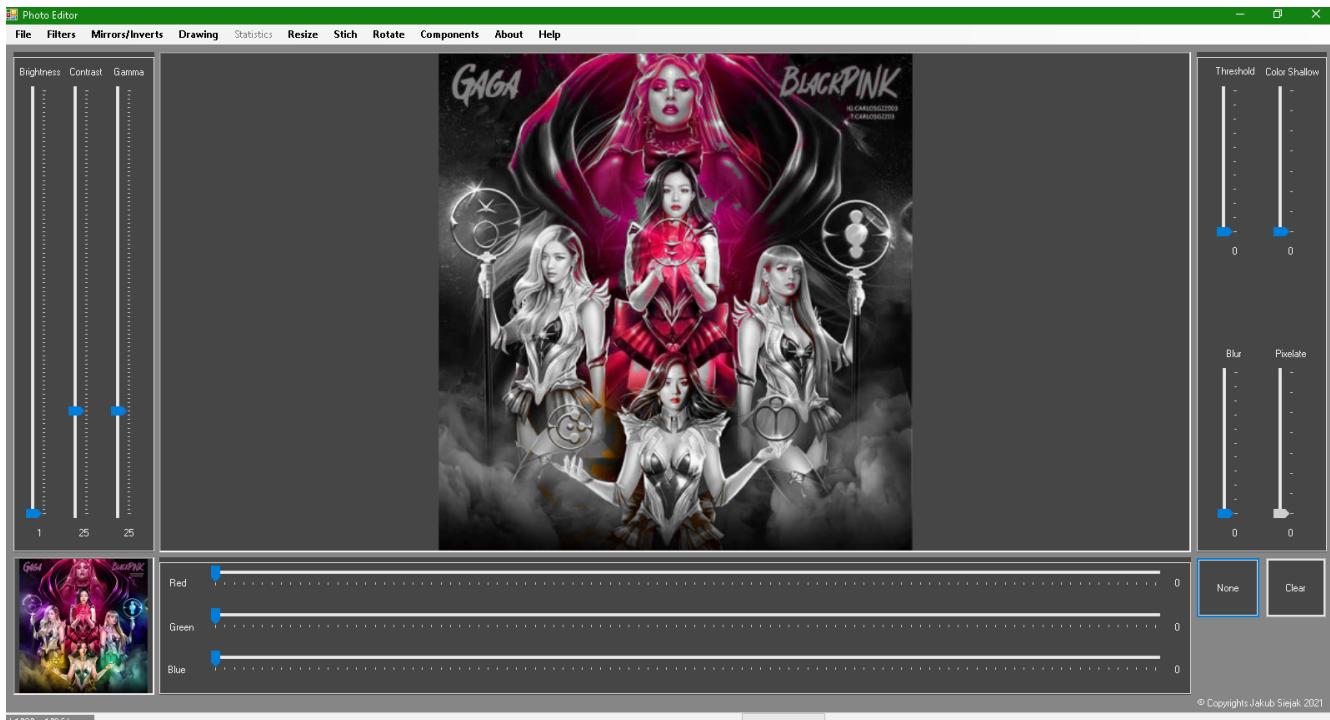
rR-Extraction

```
void comp_rR()
{
    if (!opened)
    {
        MessageBox.Show("Open an Image then apply changes");
    }
    else
    {
        int width = pictureBox.Image.Width;
        int height = pictureBox.Image.Height;

        Bitmap newBitmapTemp = new Bitmap(width, height);

        for (int y = 0; y < newBitmap.Height; y++)
        {
            for (int x = 0; x < newBitmap.Width; x++)
            {
                try
                {
                    if ((x > 0) && (x <= width))
                    {
                        Color pixel = newBitmap.GetPixel(x, y);
                        int R = (int)pixel.R;
                        int G = (int)pixel.G;
                        int B = (int)pixel.B;
                        if (((R > (G + B)) && (B <= (R * 0.65))) && (G <= (R * 0.65)))
                        {
                            newBitmapTemp.SetPixel(x, y, Color.FromArgb(R, G, B));
                        }
                        else
                        {
                            int GRAY = (((R) + (G) + (B)) / 3);
                            newBitmapTemp.SetPixel(x, y, Color.FromArgb(GRAY, GRAY, GRAY));
                        }
                    }
                }
                catch (Exception) { }
            }
            Form1_progressBar.Value = (100 * y) / newBitmap.Height;
        }
        newBitmap = newBitmapTemp;
        pictureBox.Image = newBitmap;
        Form1_progressBar.Value = 0;
    }
}
```

Rysunek 43: Zrzut ekranu pokazujący kod odpowiadający za ekstrakcję koloru czerwonego



Rysunek 44: Zrzut ekranu pokazujący efekt użycia filtru wyodrębniającego kolor czerwony

rG-Extraction

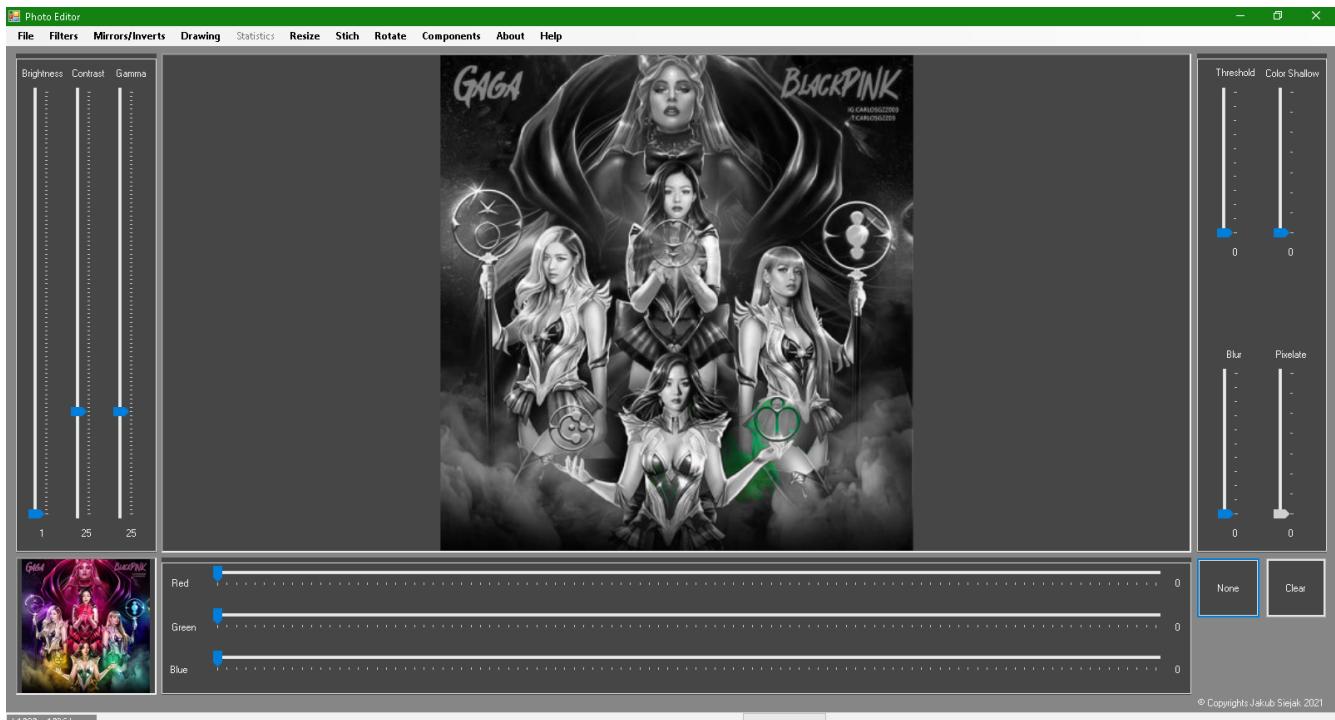
```
void comp_rG()
{
    if (!opened)
    {
        MessageBox.Show("Open an Image then apply changes");
    }
    else
    {
        int width = pictureBox.Image.Width;
        int height = pictureBox.Image.Height;

        Bitmap newBitmapTemp = new Bitmap(width, height);

        for (int y = 0; y < newBitmap.Height; y++)
        {
            for (int x = 0; x < newBitmap.Width; x++)
            {
                try
                {
                    if ((x > 0) && (x <= width))
                    {
                        Color pixel = newBitmap.GetPixel(x, y);
                        int R = (int)pixel.R;
                        int G = (int)pixel.G;
                        int B = (int)pixel.B;
                        if (((G > (R + B)) && (B <= (G * 0.65))) && (R <= (G * 0.65)))
                        {
                            newBitmapTemp.SetPixel(x, y, Color.FromArgb(R, G, B));
                        }
                        else
                        {
                            int GRAY = (((R) + (G) + (B)) / 3);
                            newBitmapTemp.SetPixel(x, y, Color.FromArgb(GRAY, GRAY, GRAY));
                        }
                    }
                }
                catch (Exception){}

                Form1_progressBar.Value = (100 * y) / newBitmap.Height;
            }
            newBitmap = newBitmapTemp;
            pictureBox.Image = newBitmap;
            Form1_progressBar.Value = 0;
        }
    }
}
```

Rysunek 45: Zrzut ekranu pokazujący kod odpowiadający za ekstrakcję koloru zielonego



Rysunek 46: Zrzut ekranu pokazujący efekt użycia filtru wyodrębniającego kolor zielony

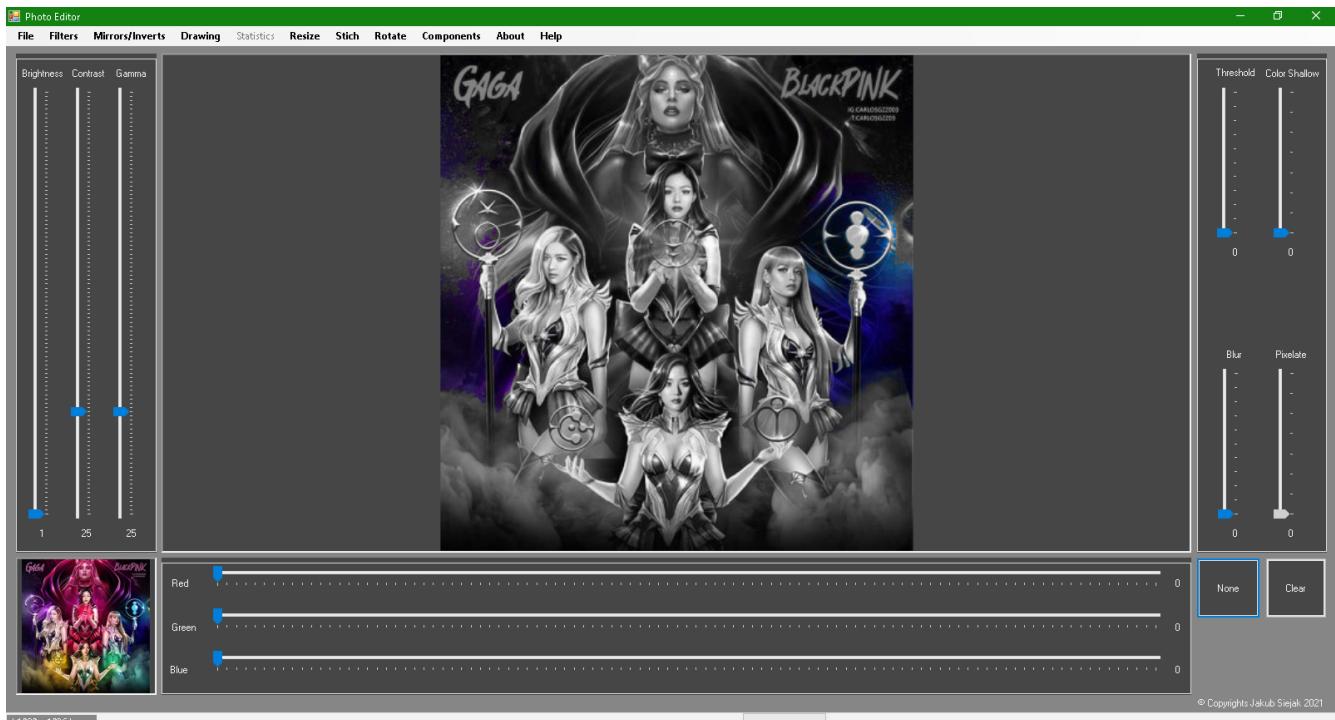
rB-Extraction

```
void comp_rB()
{
    if (!opened)
    {
        MessageBox.Show("Open an Image then apply changes");
    }
    else
    {
        int width = pictureBox.Image.Width;
        int height = pictureBox.Image.Height;

        Bitmap newBitmapTemp = new Bitmap(width, height);

        for (int y = 0; y < newBitmap.Height; y++)
        {
            for (int x = 0; x < newBitmap.Width; x++)
            {
                try
                {
                    if ((x > 0) && (x <= width))
                    {
                        Color pixel = newBitmap.GetPixel(x, y);
                        int R = (int)pixel.R;
                        int G = (int)pixel.G;
                        int B = (int)pixel.B;
                        if (((B > (G + R)) && (R <= (B * 0.65)) && (G <= (B * 0.65))))|
                        {
                            newBitmapTemp.SetPixel(x, y, Color.FromArgb(R, G, B));
                        }
                        else
                        {
                            int GRAY = (((R) + (G) + (B)) / 3);
                            newBitmapTemp.SetPixel(x, y, Color.FromArgb(GRAY, GRAY, GRAY));
                        }
                    }
                }
                catch (Exception){}
            }
            Form1.progressBar.Value = (100 * y) / newBitmap.Height;
        }
        newBitmap = newBitmapTemp;
        pictureBox.Image = newBitmap;
        Form1.progressBar.Value = 0;
    }
}
```

Rysunek 47: Zrzut ekranu pokazujący kod odpowiadający za ekstrakcję koloru niebieskiego



Rysunek 48: Zrzut ekranu pokazujący efekt użycia filtru wyodrębniającego kolor niebieski

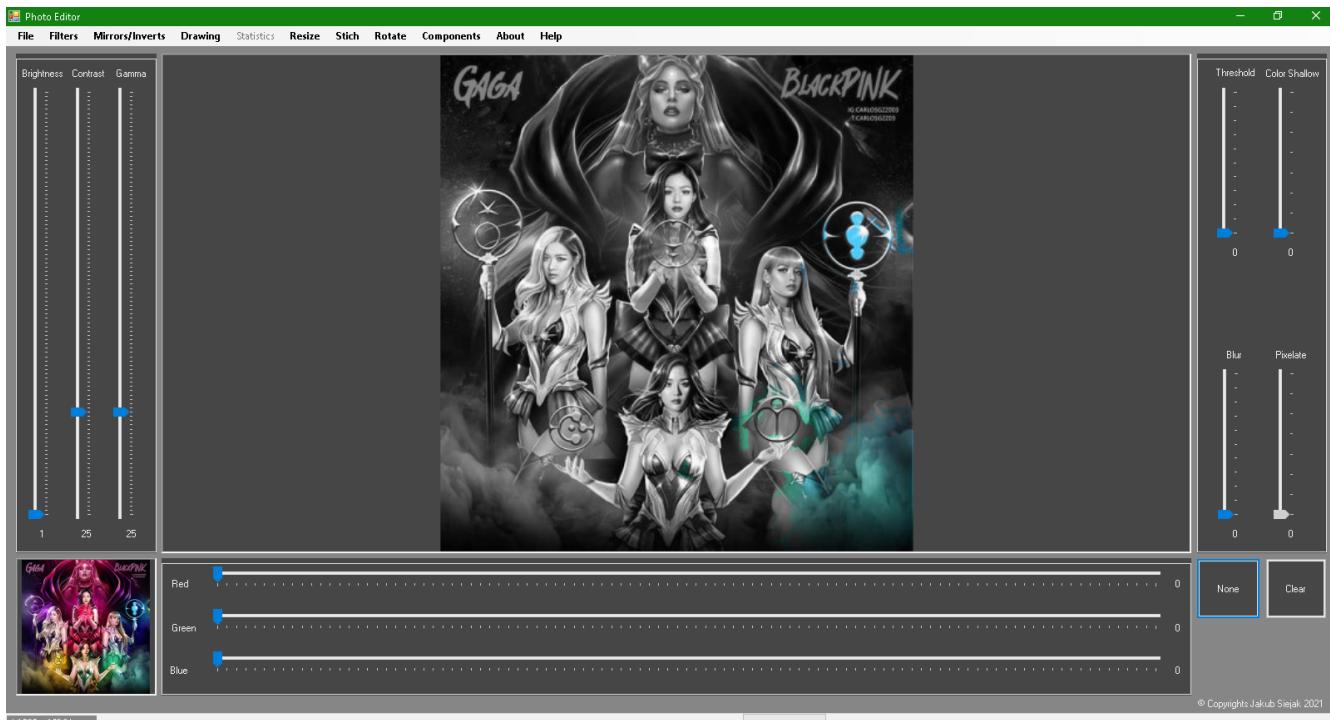
rC-Extraction

```
void comp_rC()
{
    if (!opened)
    {
        MessageBox.Show("Open an Image then apply changes");
    }
    else
    {
        int width = pictureBox.Image.Width;
        int height = pictureBox.Image.Height;

        Bitmap newBitmapTemp = new Bitmap(width, height);

        for (int y = 0; y < newBitmap.Height; y++)
        {
            for (int x = 0; x < newBitmap.Width; x++)
            {
                try
                {
                    if ((x > 0) && (x <= width))
                    {
                        Color pixel = newBitmap.GetPixel(x, y);
                        int R = (int)pixel.R;
                        int G = (int)pixel.G;
                        int B = (int)pixel.B;
                        if (((B * 0.65) >= R) && ((G * 0.65) >= R) && (G >= (0.75 * B)) && (B >= (0.75 * G)))
                        {
                            newBitmapTemp.SetPixel(x, y, Color.FromArgb(R, G, B));
                        }
                        else
                        {
                            int GRAY = (((R) + (G) + (B)) / 3);
                            newBitmapTemp.SetPixel(x, y, Color.FromArgb(GRAY, GRAY, GRAY));
                        }
                    }
                }
                catch (Exception) {}
            }
            Form1_progressBar.Value = (100 * y) / newBitmap.Height;
        }
        newBitmap = newBitmapTemp;
        pictureBox.Image = newBitmap;
        Form1_progressBar.Value = 0;
    }
}
```

Rysunek 49: Zrzut ekranu pokazujący kod odpowiadający za ekstrakcję koloru cyjanowego



Rysunek 50: Zrzut ekranu pokazujący efekt użycia filtra wyodrębniającego kolor cyjanowy

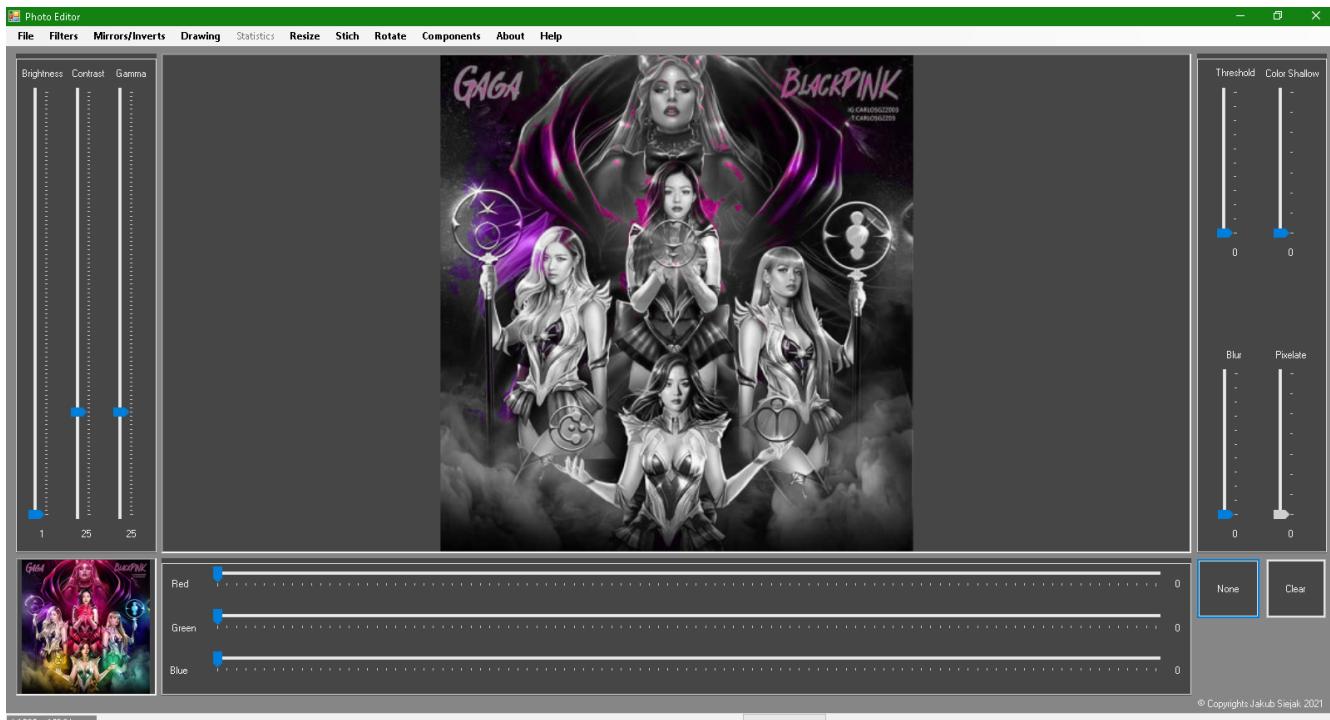
rM-Extraction

```
void comp_rM()
{
    if (!opened)
    {
        MessageBox.Show("Open an Image then apply changes");
    }
    else
    {
        int width = pictureBox.Image.Width;
        int height = pictureBox.Image.Height;

        Bitmap newBitmapTemp = new Bitmap(width, height);

        for (int y = 0; y < newBitmap.Height; y++)
        {
            for (int x = 0; x < newBitmap.Width; x++)
            {
                try
                {
                    if ((x > 0) && (x <= width))
                    {
                        Color pixel = newBitmap.GetPixel(x, y);
                        int R = (int)pixel.R;
                        int G = (int)pixel.G;
                        int B = (int)pixel.B;
                        if (((B * 0.65) >= G) && ((R * 0.65) >= G) && (R >= (0.65 * B)) && (B >= (0.65 * R)))
                        {
                            newBitmapTemp.SetPixel(x, y, Color.FromArgb(R, G, B));
                        }
                        else
                        {
                            int GRAY = (((R) + (G) + (B)) / 3);
                            newBitmapTemp.SetPixel(x, y, Color.FromArgb(GRAY, GRAY, GRAY));
                        }
                    }
                    catch (Exception) {}
                }
                Form1_progressBar.Value = (100 * y) / newBitmap.Height;
            }
            newBitmap = newBitmapTemp;
            pictureBox.Image = newBitmap;
            Form1_progressBar.Value = 0;
        }
    }
}
```

Rysunek 51: Zrzut ekranu pokazujący kod odpowiadający za ekstrakcję koloru megentowego



Rysunek 52: Zrzut ekranu pokazujący efekt użycia filtru wyodrębniającego kolor megentowy

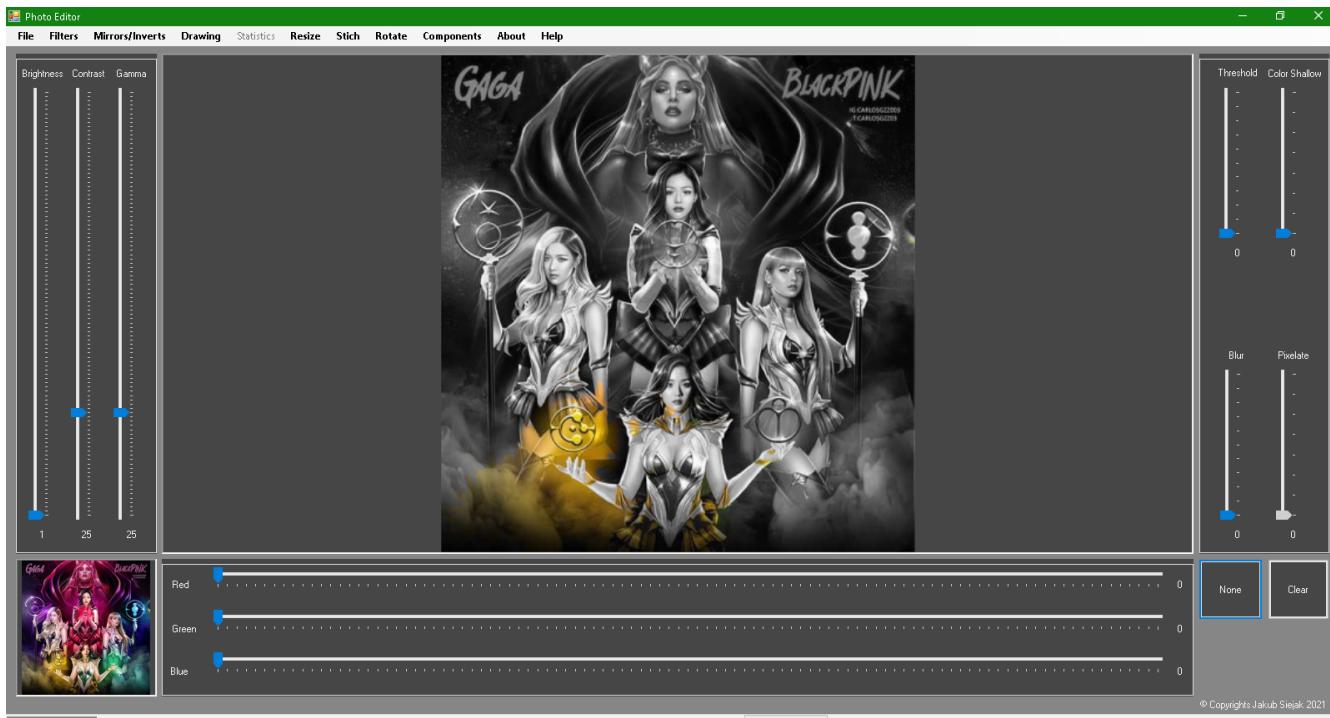
rY-Extraction

```
void comp_rY()
{
    if (!opened)
    {
        MessageBox.Show("Open an Image then apply changes");
    }
    else
    {
        int width = pictureBox.Image.Width;
        int height = pictureBox.Image.Height;

        Bitmap newBitmapTemp = new Bitmap(width, height);

        for (int y = 0; y < newBitmap.Height; y++)
        {
            for (int x = 0; x < newBitmap.Width; x++)
            {
                try
                {
                    if ((x > 0) && (x <= width))
                    {
                        Color pixel = newBitmap.GetPixel(x, y);
                        int R = (int)pixel.R;
                        int G = (int)pixel.G;
                        int B = (int)pixel.B;
                        if (((G * 0.65) >= B) && ((G * 0.65) >= R) && (G > (0.6 * R)) && (R > (0.65 * G)))
                        {
                            newBitmapTemp.SetPixel(x, y, Color.FromArgb(R, G, B));
                        }
                        else
                        {
                            int GRAY = (((R) + (G) + (B)) / 3);
                            newBitmapTemp.SetPixel(x, y, Color.FromArgb(GRAY, GRAY, GRAY));
                        }
                    }
                }
                catch (Exception) { }
            }
            Form1_progressBar.Value = (100 * y) / newBitmap.Height;
        }
        newBitmap = newBitmapTemp;
        pictureBox.Image = newBitmap;
        Form1_progressBar.Value = 0;
    }
}
```

Rysunek 53: Zrzut ekranu pokazujący kod odpowiadający za ekstrakcję koloru żółtego



Rysunek 54: Zrzut ekranu pokazujący efekt użycia filtru wyodrębniającego kolor żółty

19. Krótki opis kodu:

Form.cs

- implementacja bibliotek:

“System”, “System.Collections.Generic”, “System.ComponentModel”, “System.Data”,
“System.Drawing”, “System.Linq”, “System.Text”, “System.Threading”, “System.Threading.Tasks”,
“System.Windows.Forms”, “System.Collections”, “System.Drawing.Imaging”,
“System.Runtime.InteropServices”, “System.IO”, “System.Drawing.Printing”,
“System.Runtime.Serialization.Formatters.Binary”, “System.Runtime.Serialization”,
“System.Diagnostics”,

- implementacja funkcji:

“hue”, “contrast”, “brightness”, “gamma”, “saveImage”, “openImage”, “grayscale”, “sepia”,
“blur”, “invert”, “edge”, “mirror_l”, “mirror_r”, “mirror_t”, “mirror_b”, “mirror_horizontal”,
“mirror_vertical”, “bilinear_x2”, “bilinear_x3”, “nearest_neighbour_x3”, “add_RGB_ver”,
“add_RGB_ver_2”, “mirror_stich_right”, “mirror_stich_left”, “mirror_stich_top”,
“mirror_stich_bottom”, “rotate_left”, “rotate_right”, “comp_rgb”, “comp_rgb2”, “comp_ycbcr”,
“threshold”, “color_shallowing”, “stich_horizontal”, “stich_vertical”, “blur”,
“edge_WoB”, “stich_sec_left”, “comp_rR”, “comp_rG”, “comp_rB”, “comp_rc”, “comp_rM”, “comp_rY”,
“stich_left”, “stich_right”, “stich_top”, “stich_bottom”, “comp_bR”, “comp_bG”, “comp_bB”,
“add_RGB_ver_3”, “add_RGB_ver_4”, “cyan_to_white”,

- implementacja reakcji na przyciski oraz suwaki.

Pen_size.cs:

- implementacja bibliotek:

System, System.Collections.Generic, System.ComponentModel, System.Data, System.Drawing,
System.Linq, System.Text, System.Threading.Tasks, System.Windows.Forms,

- implementacja funkcji:

“Pen_size_OK_button_Click”

20. Błędy

- Błąd polegający na nienakładaniu się wszystkich filtrów. To znaczy gdy używamy filtru A i nie zapiszemy jego efektu, ale zastosujemy filtr B to efektem końcowym będzie nałożenie filtru B na obraz oryginalny.