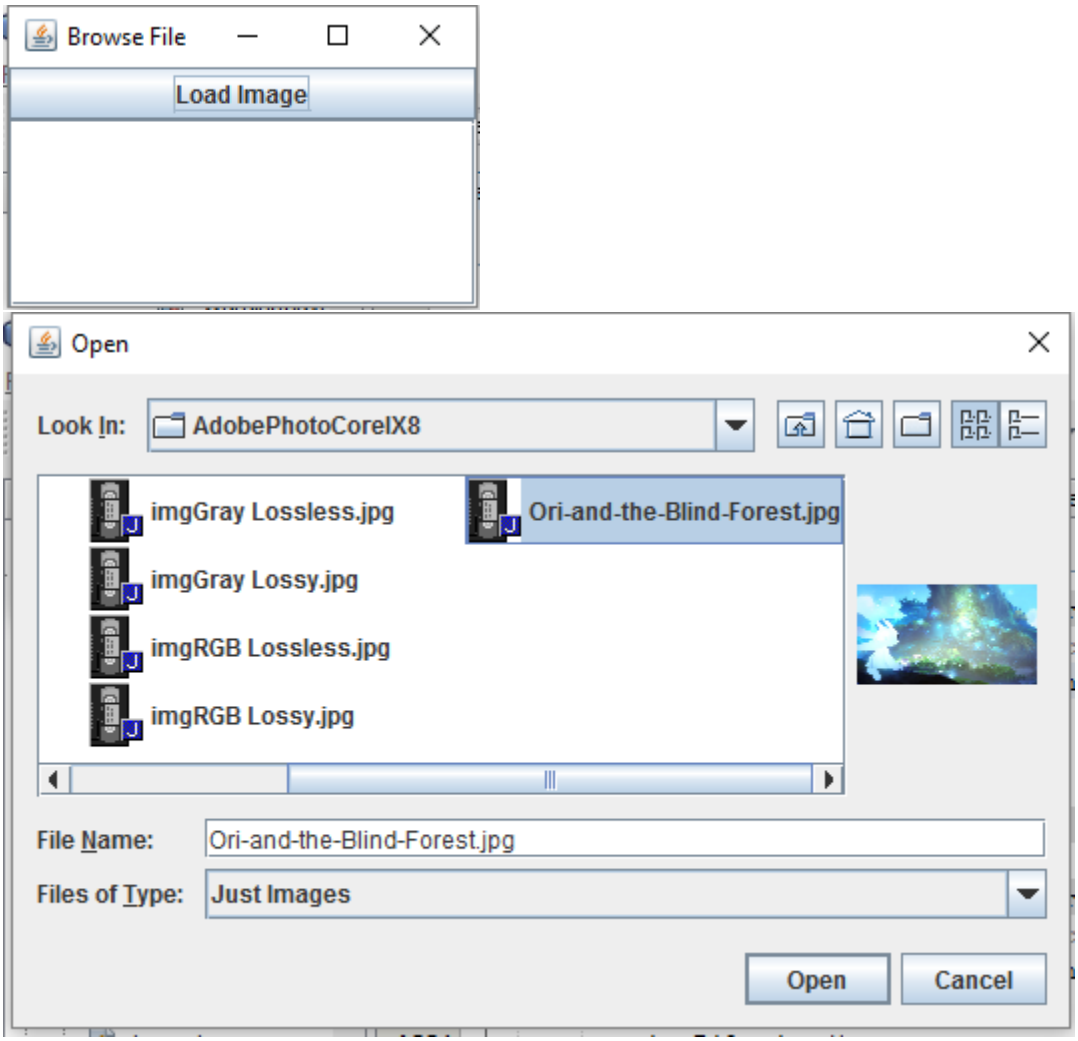
Ketika run:

1. Pilih lokasi gambar, dan open, akan otomatis ke frame berikutnya setelah dibuka.



2. Ketika open, membuka class main.EditField, di class ini melakukan operasi mengubah image menjadi array r, g, dan b, dengan memanggil constructor class LoadImg.

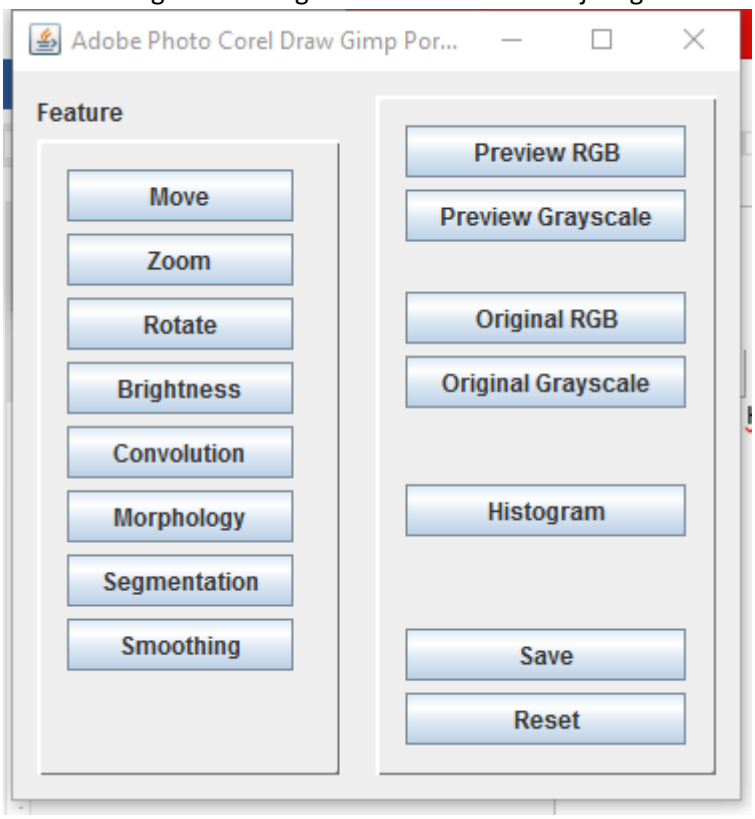
```
for(int i=0; i<w; i++){
    for(int j=0; j<h; j++){
        c = new Color(img.getRGB(i, j));
        this.R[i][j] = c.getRed();
        this.G[i][j] = c.getGreen();
        this.B[i][j] = c.getBlue();
    }
}
```

Kemudian class main.EditField memanggil constructor RGBtoGrayscale untuk mengubah dari array r, g, dan b menjadi array grayscale.

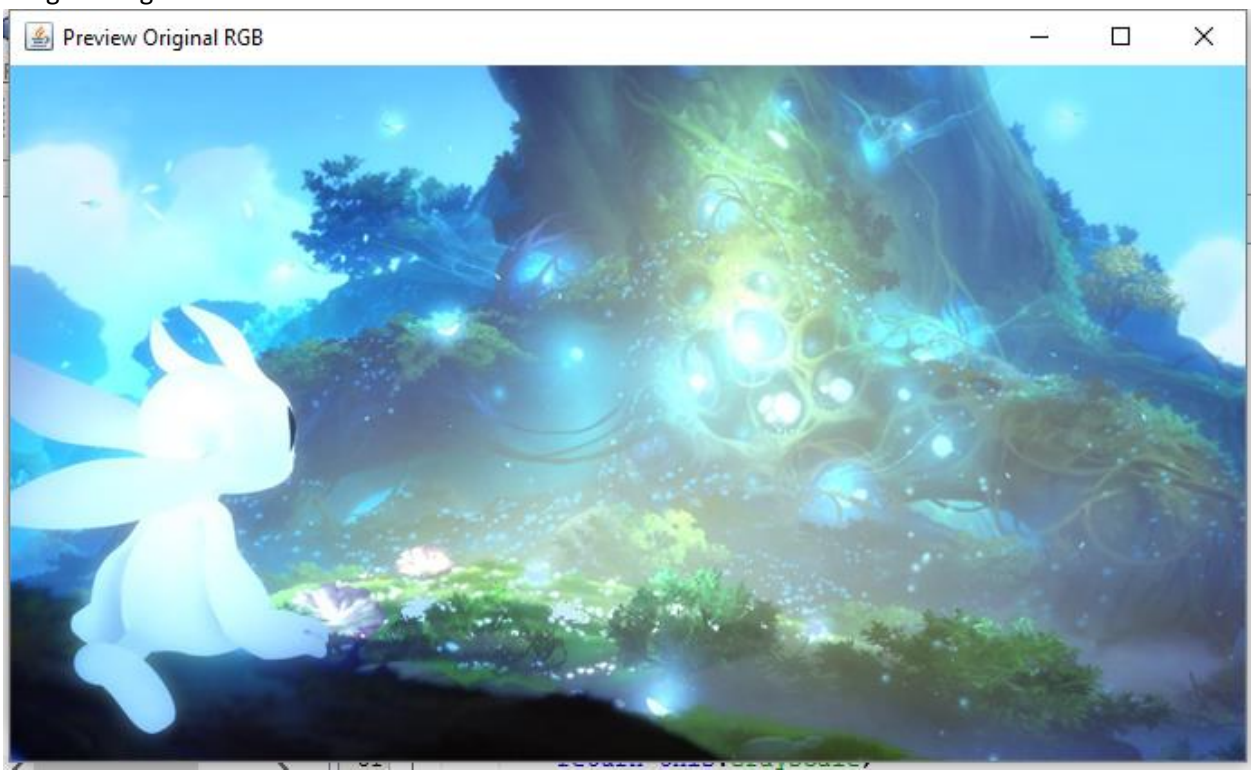
```
for(int i=0; i<w; i++){
    for(int j=0; j<h; j++){
        this.Grayscale[i][j] = calcGray(R[i][j], G[i][j], B[i][j]);
    }
}
public int calcGray(int r, int g, int b){
    return ((r+g+b)/3);
}
```

Setelah itu, class main.EditField memanggil main frame EditFieldView untuk melakukan operasi fitur fitur yang ada.

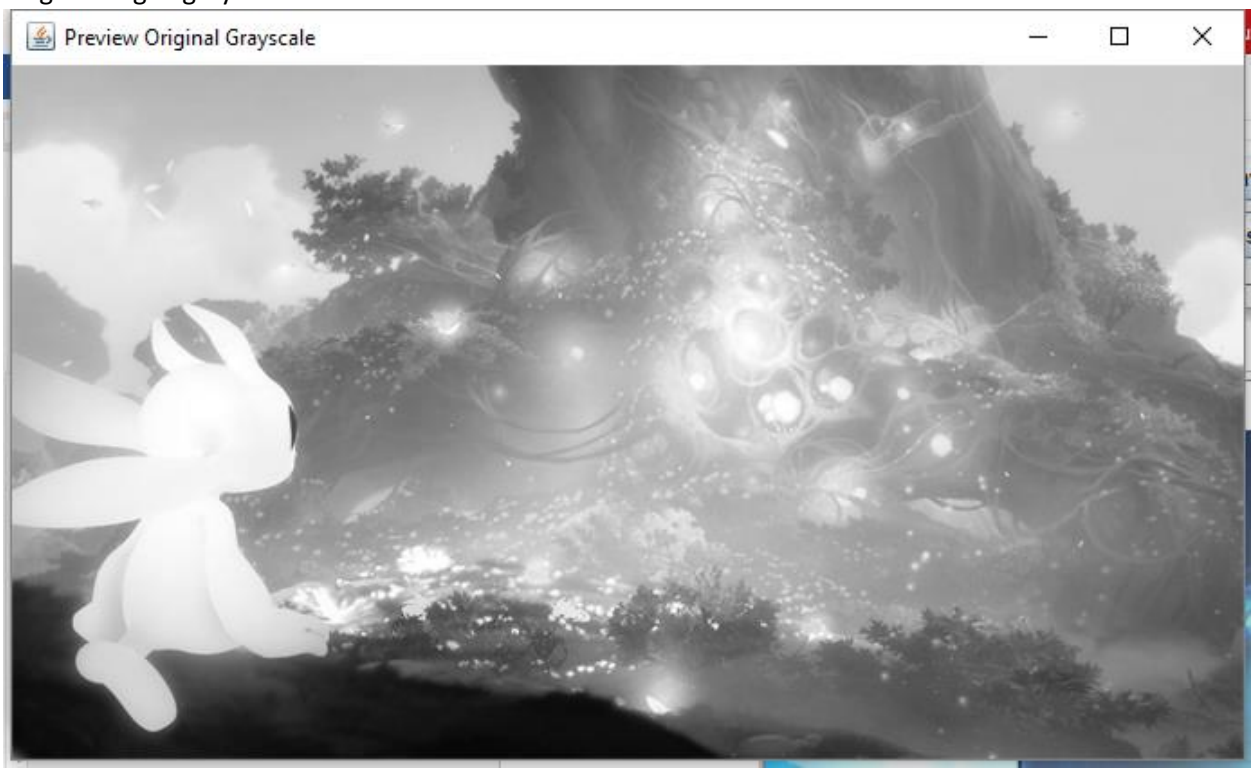
3. Preview RGB = melihat gambar semetara hasil modifikasi
   Preview Grayscale = melihat gambar sementara dalam grayslcale
   Original rgb = melihat gambar asli , bisa untuk perbandingan sebelum dan sesudah edit
   Original grayscale = melihat gambar asli dalam grayscale
   Reset = mengembalikan gambar sementara menjadi gambar awal



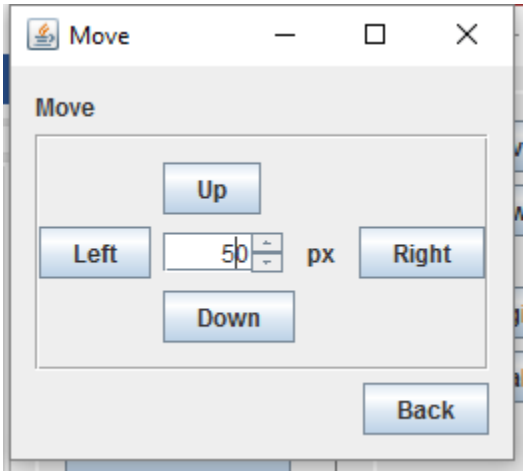
4. Original img in RGB:



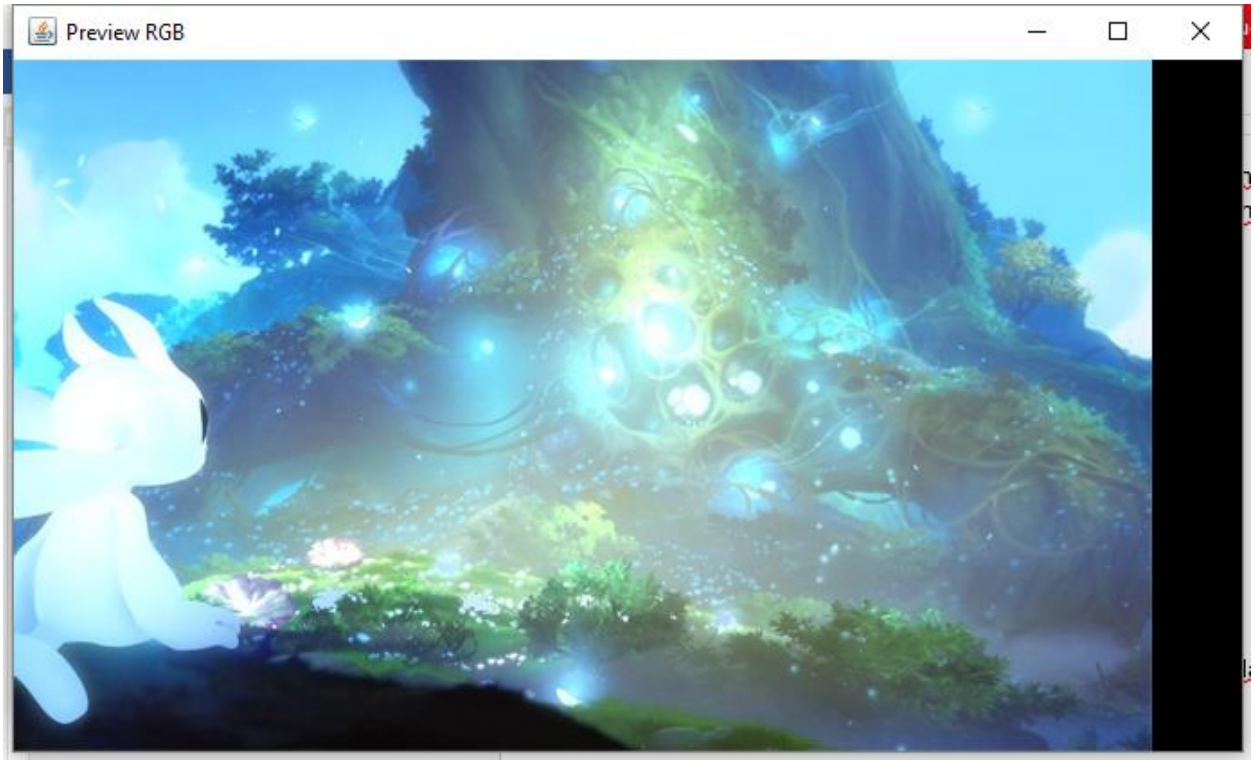
Original img in grayscale :



5. Fitur Move untuk pergeseran gambar. Isi ingin menggeser berapa jauh jarak pergeseran, up, left, down, right, untuk melakukan pergeseran dengan arah tersebut.



Hasil geser ke kiri dengan nilai 50 (hasil tidak langsung dilihatkan, untuk melihat hasilnya pada menu, preview rgb atau priview grayscale):

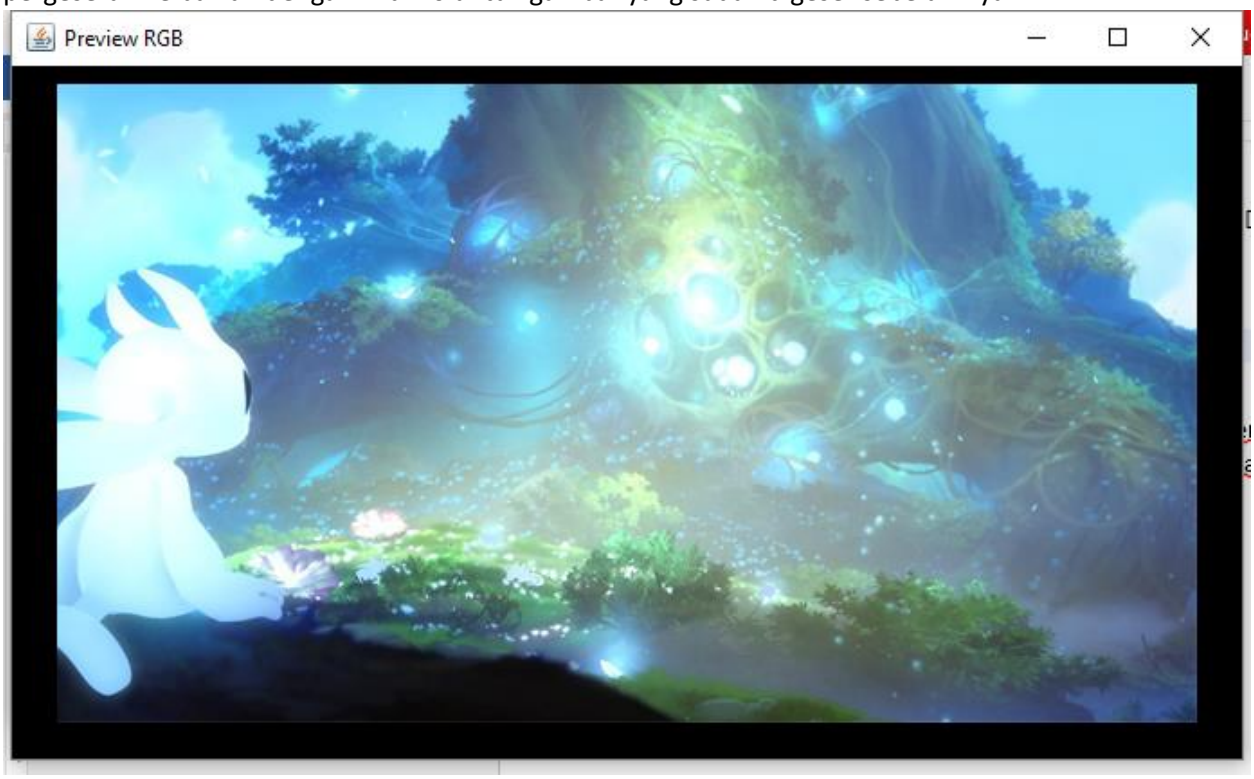


Penjelasan : fungsi ini akan menggeser ke arah yang dipilih dengan jarak yang diisikan. bagian di arah berlawanan pergeseran gambar akan menjadi hitam. Gambar baru hasil pergeseran itu sebenarnya pada titik gambar baru mengambil titik dari +n jarak pergeseran dari gambar awalnya.

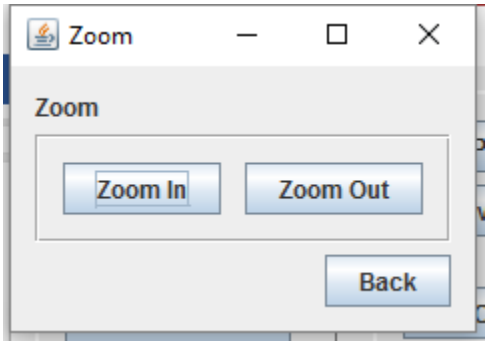
```
//geser ke kiri
public int[][] GeserL(int[][] img, int n){
    n = Math.abs(n);
    int w = img.length;
    int h = img[0].length;
    int[][] temp = new int[w][h];
    for(int i=0; i<w-n; i++){
        for(int j=0; j<h; j++){
            temp[i][j] = img[n+i][j];
        }
    }
    return temp;
}
```

Code untuk aalgoritma pergeseran ke kiri sejauh n.

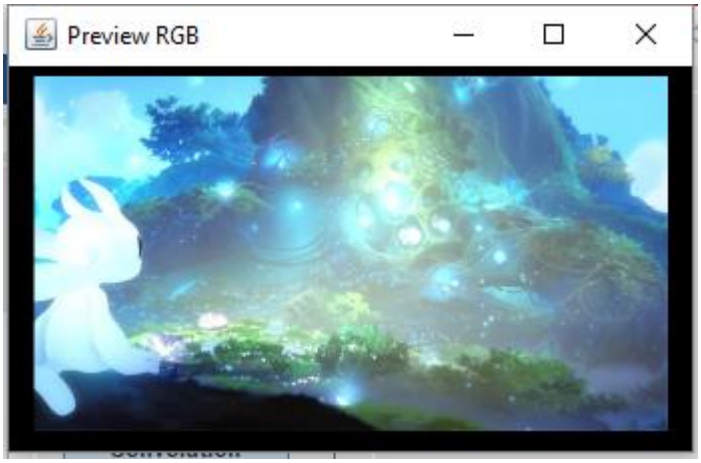
Hasil pergeseran ke kanan dengan nilai 25, pergeseran ke atas dengan nilai 30, dan kemudian pergeseran ke bawah dengan nilai 10 untuk gambar yang sudah digeser sebelumnya :



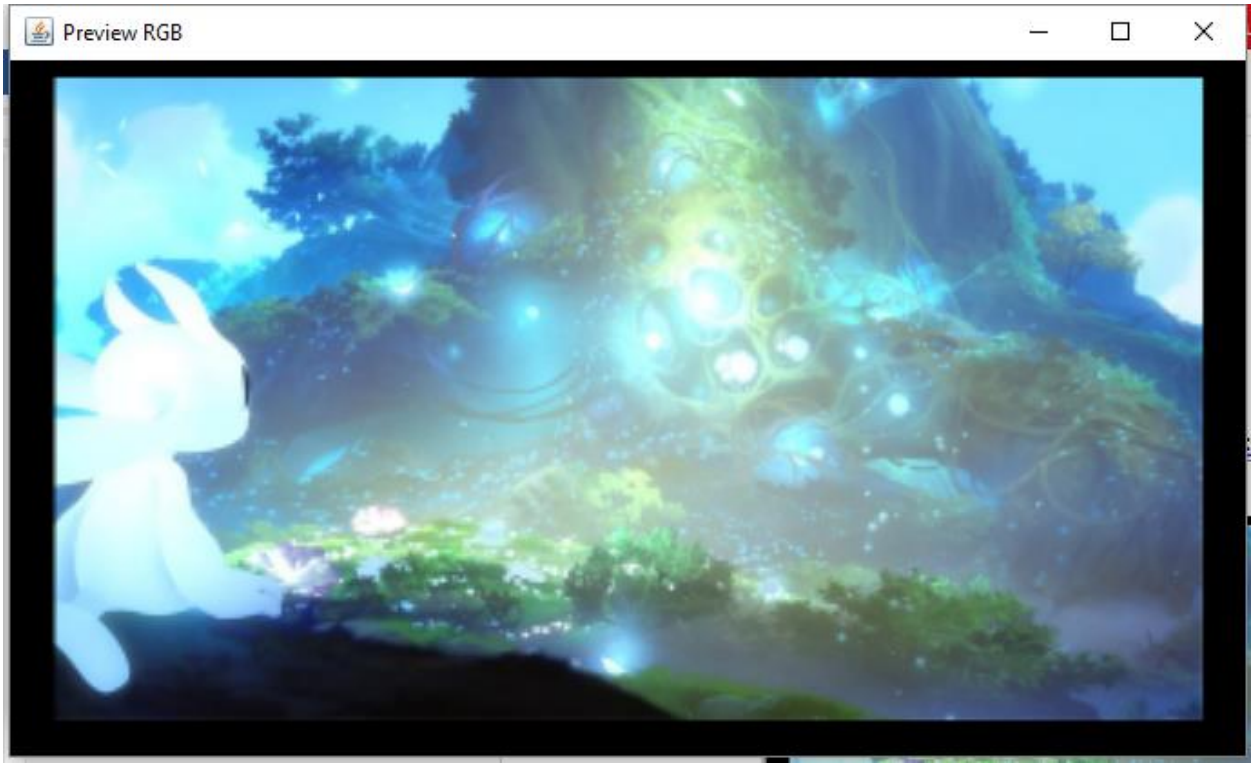
6. Fitur zoom untuk memperbesar atau mengecilkan ukuran gambar. Di sini perbesaran hanya 2x dari ukuran gambar, dan pengecilan hanya 2x dari ukuran gambar.



Berikut hasil zoom out pertama dari gambar :



Hasil zoom in untuk gambar yang sudah di zoom out sebelumnya :



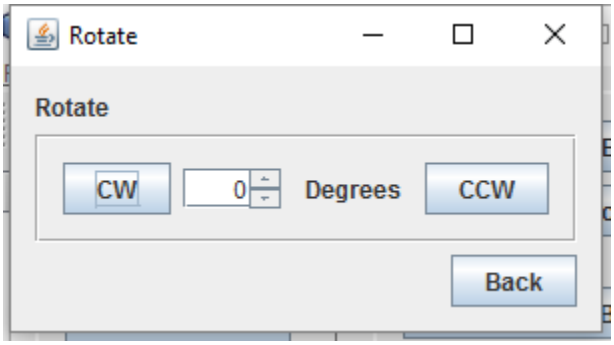
Algoritma zoom in di sini memperbesar ke samping kiri dan kana dahulu kemudian ke atas dank e bawah. nilai titik antara titik gambar adalah nilai rata-rata dari kiri dan kanan titik gambar awal untuk row wise zooming. Kemudian nili antar titik yang masih kosong adalah nilai dari rata-rata titik di atas dan di bawahnya.

```
for(int i=0; i<w0; i++){
    for(int j=0; j<h0; j++){
        temp[i+(i*scale)][j+(j*scale)] = img[i][j];
        //Row wise zooming
        if(i!=w0-1)temp[i+(i*scale)+1][j+(j*scale)] = (img[i+1][j]+img[i][j])/2;;
    }
}
//Column wise zooming
for(int i=0; i<w1; i++){
    for(int j=0; j<h0-1; j++){
        int j1 = j+1;
        temp[i][j+(j*scale)+1] = (temp[i][j1+(j1*scale)] + temp[i][j+(j*scale)])/2;
    }
}
```

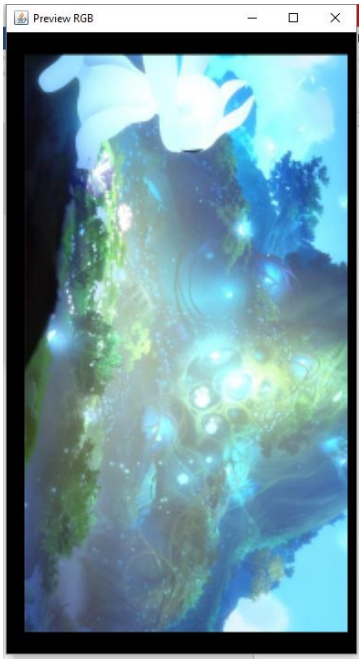
Untuk zoom out, jika ukuran tinggi atau lebar gambar ganjil, maka baris paling bawah atau kolom paling kanan akan diabaikan untuk menghindari array out of bound. Nilai titik baru itu diambil dari nilai rata-rata 4.

```
if(w0%2==1)w0--;
if(h0%2==1)h0--;

temp = new int[w1][h1];

for(int i=0; i<w0; i+=2){
    for(int j=0; j<h0; j+=2){
        temp[i/2][j/2] = (img[i][j]+img[i+1][j]+img[i][j+1]+img[i+1][j+1])/4;
    }
}
```

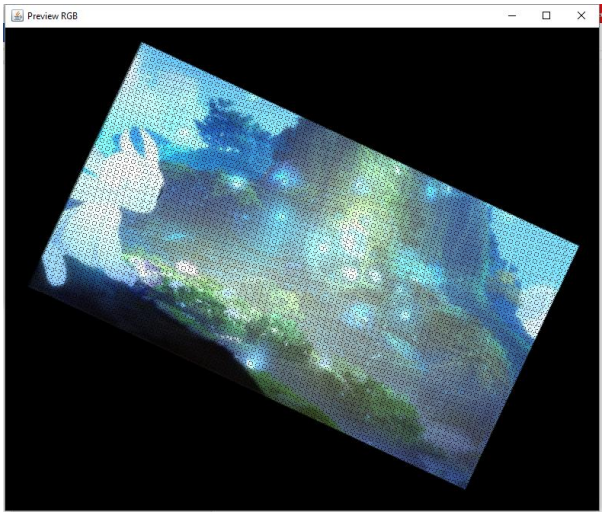
7. Fitur rotate akan membuat gambar berotasi sebanyak nilai derajat yang dituliiskan searah jarum jam (CW) atau berlawanan arah jarum jam(CCW).



Hasil rotasi 90 derajat CW:



Hasil rotasi 65 derajat CCW dari gambar sebelumnya:



Untuk rotasi 90 derajat, gambar hasil rotasi adalah dengan tinggi baru=lebar gambar awal dan lebar baru=tinggi gambar awal. Untuk rotasi yang lebih dari 90 derajat akan diputar 90 derajat dahulu sampai sisa sudut yang harus diputar lebih kecil dari 90.

```
int p = n/90;

for(int i=0; i<p; i++){
    imgRot = rotate90CW(imgRot);
    n-=90;
}
```

Untuk rotasi dibawah 90 derajat, ukuran gambar barunya tergantung dari sudut sisa rotasi. Dengan lebar baru w1= w0*cos(n) + h0*sin(n) dan tinggi baru h1=w0*sin(n) + h0*cos(n).

```
int w1 = (int)Math.floor((Math.cos(Math.toRadians(n))*w0)+(Math.sin(Math.toRadians(n))*h0));
int h1 = (int)Math.floor((Math.sin(Math.toRadians(n))*w0)+(Math.cos(Math.toRadians(n))*h0));
```

Untuk menentukan titik tujuan dari titik awal adalah berikut untuk rotasi searah jarum jam (CW):

```
a = (int)((i*cosA)-(j*sinA));
b = (int)((i*sinA)+(j*cosA));
```

Sedangkan untuk rotasi yang berlawanan arah jarum jam (CCW):

```
a = (int)((i*cosA)+(j*sinA));
b = (int)(-(i*sinA)+(j*cosA));
```

Data dari titik tujuan disimpan pada array berikut:

```
dest[i][j][0]=a;
dest[i][j][1]=b;
```

Namun, nilai tujuan bisa negative, jadi harus dikurangkan dengan nilai paling kecil dari semua titik tujuan untuk masing-masing a(x tujuan) dan b(y tujuan) supaya semua nilai >=0.

```
x = dest[i][j][0] - xmin;
y = dest[i][j][1] - ymin;
temp[x][y] = imgRot[i][j];
```

Untuk rotasi 90 derajat hanya melakukan tukar nilai.

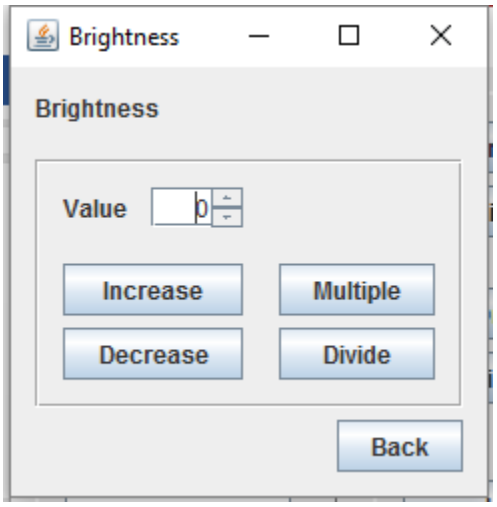
CW:

```
for(int i=0; i<h; i++){
    for(int j=0; j<w; j++){
        temp[i][j] = img[j][h-i-1];
    }
}
```

CCW:

```
for(int i=0; i<h; i++){
    for(int j=0; j<w; j++){
        temp[i][j] = img[w-j-1][i];
    }
}
```

8. Fitur brightness terdapat penambahan, pengurangan, perkalian, dan pembagian brightness dengan nilai yang kita berikan.

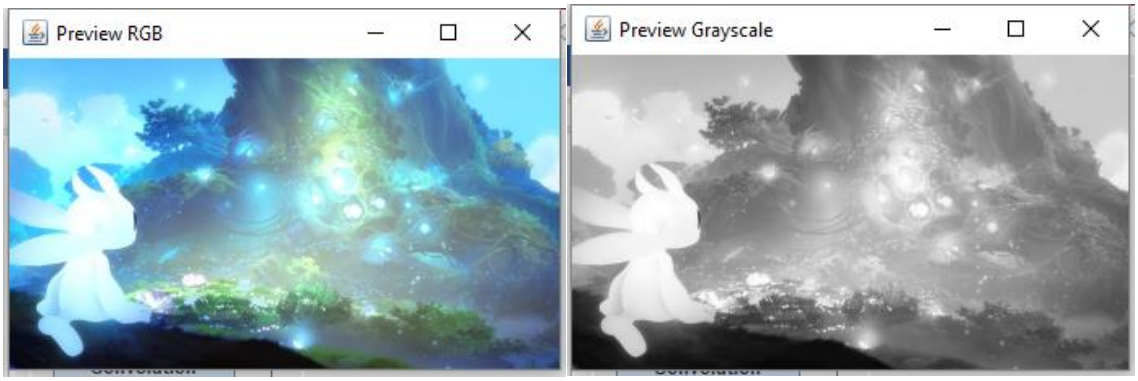


Untuk fungsi penjumlahan dan pengurangan brightness terdapat pada Brigthness1.java. penumlahan dan pengurangan brightness ini akan menambahakan atau mengurangkan dengan value untuk setiap nilai titik. Untuk perkalian dan pembagian brigtness terdapat pada Brightness2.java. nilai dari setiap titik pada gambar akan diklikan atau dibagi dengn value yang kita berikan.

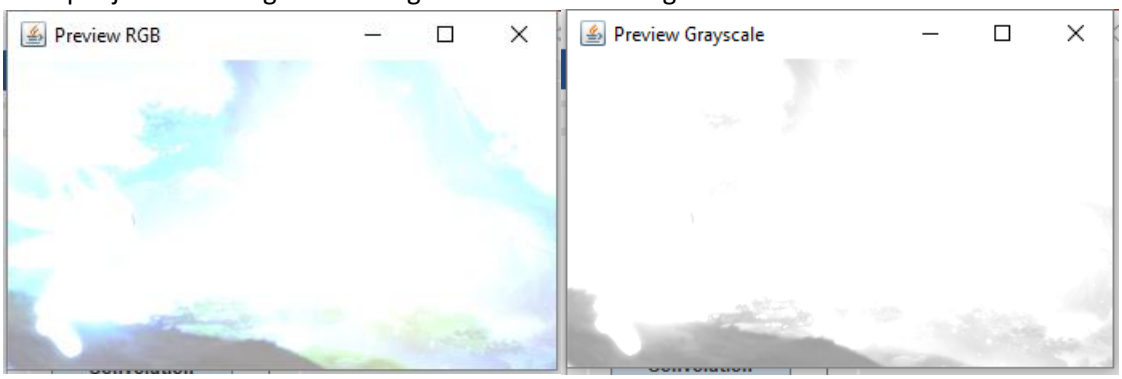
Terdapat fungsi setVal untuk menjaga nilai yang dihasilkan tidak kurang dari 0 atau lebih dari 255.

```
public int setVal(int value){
    if(value < 0) return 0;
    if(value > 255) return 255;
    return value;
}
```

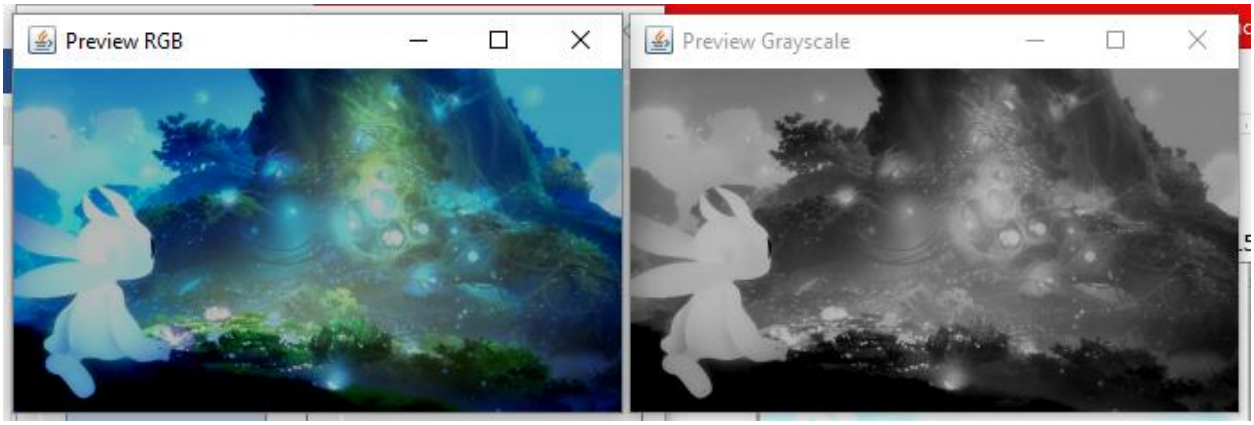
Img awal:



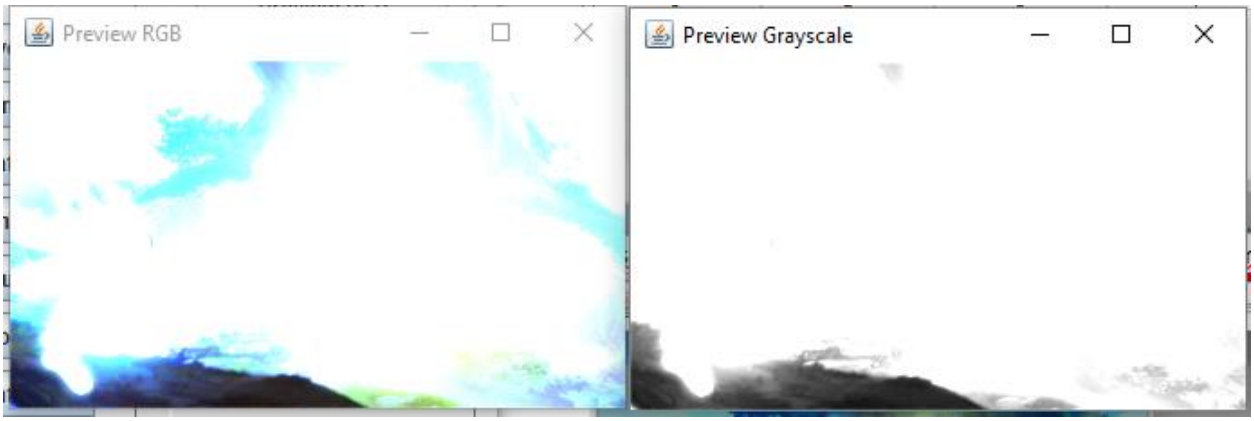
Hasil penjumlahan brightness dengan value=150 untuk gambar awal:



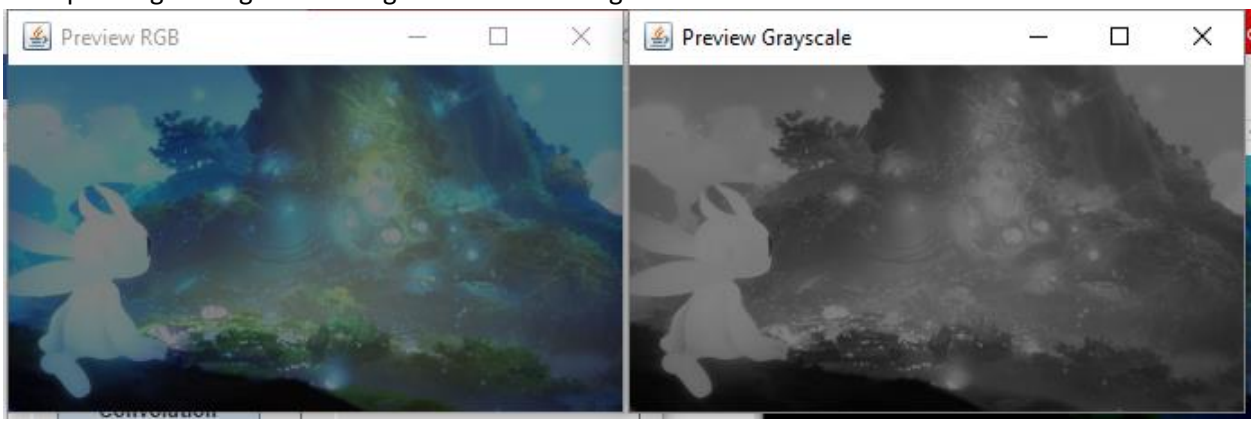
Hasil pengurangan brightness dengan value=70 untuk gambar awal:



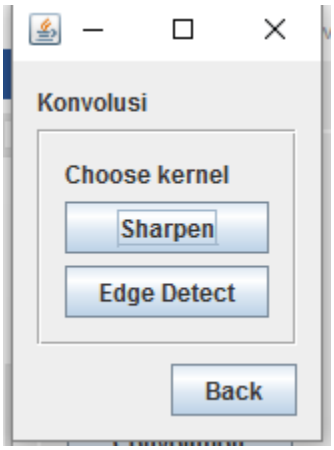
Hasil perkalian brightness dengan value=3 untuk gambar awal:



Hasil pembagian brightness dengan value=2 untuk gambar awal:



9. Fitur convolution untuk melakukan konvolusi gambar awal dengan kernel yand dipilih. Terdapat kernel yang disediakan yaitu berikut:

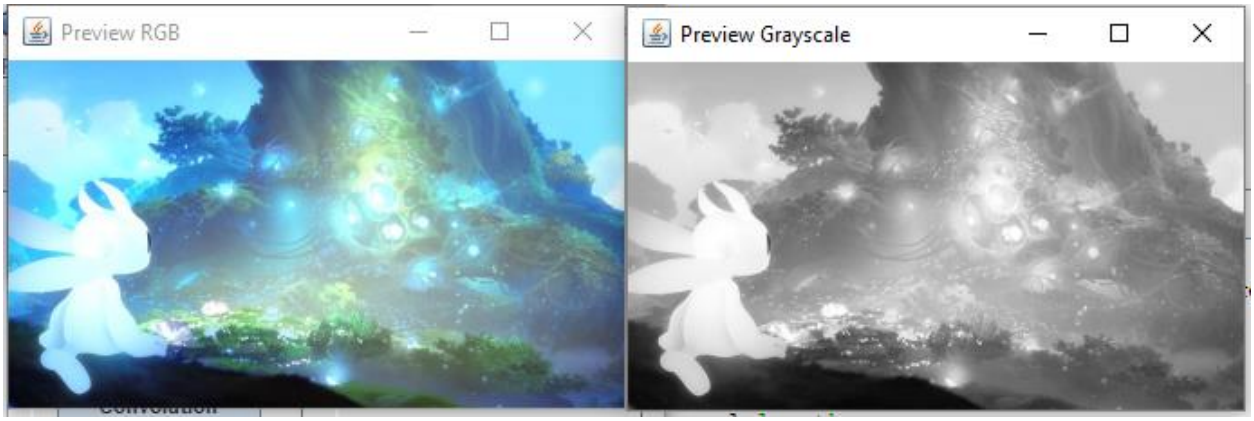


Untuk setiap titik pada gambar dijalankan kernel untuk menentukan nilai baru untuk titik tersebut yang merupakan hasil dari konvolusi terhadap matrix kernel. W=lebar gambar awal, h=tinggi gambar awal, wk=lebar kernel, hk=tinggi kernel, serta x dan y merupakan poros kernel.

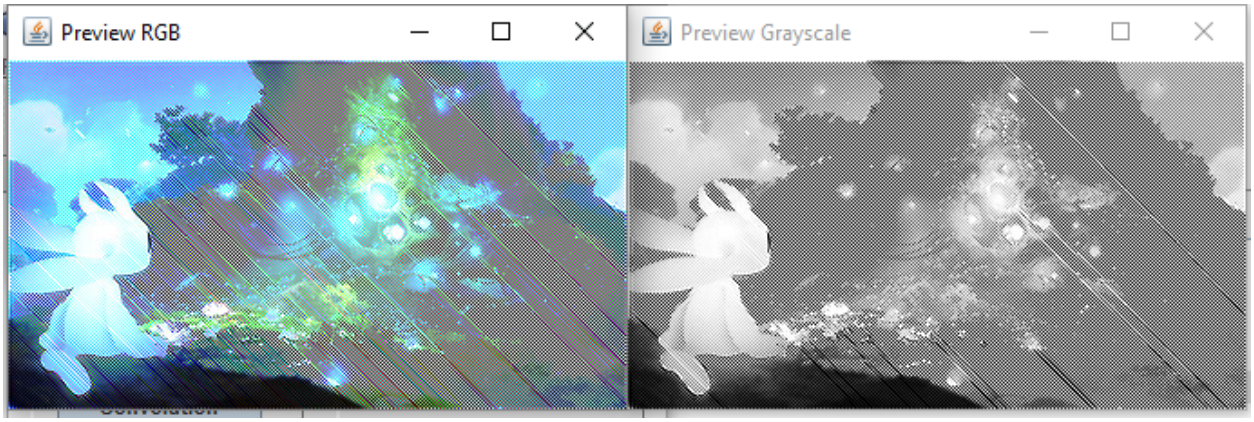
```
for(int i=0; i<w; i++){
    for(int j=0; j<h; j++){
        int sum=0;
        for(int a=0; a<wk; a++){
            for(int b=0; b<hk; b++){
                try{
                    sum += (img[i+a-x][j+b-y] * kernel[a][b]);
                }catch(Exception e){

                }
            }
        }
        temp[i][j]= setVal(sum);
    }
}
```

Class EdgeDetection.java dan Sharpening.java merupakan kelas yang berisi kernel untuk dilakukannya konvolusi edged detection dan image sharpening.

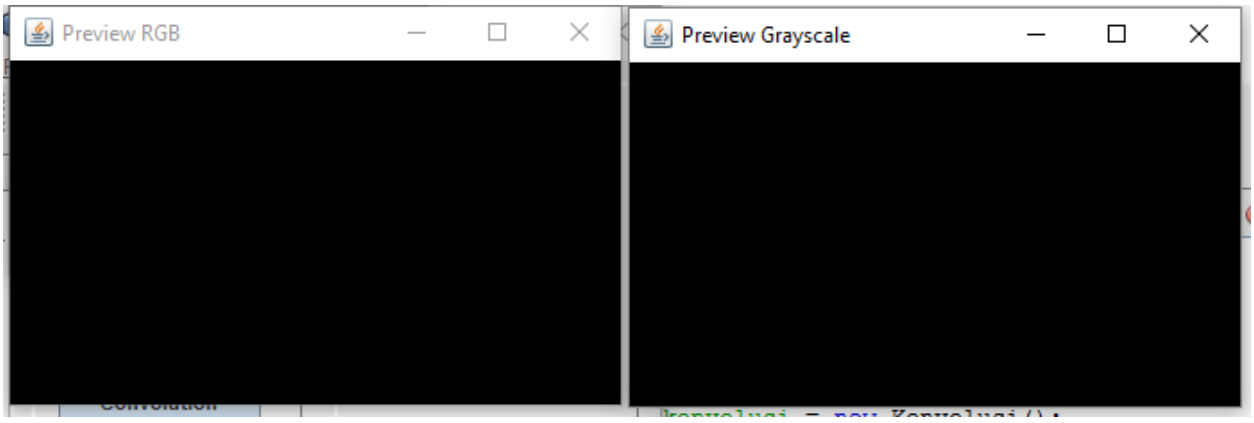
Berikut merupakan gambar asli sebelum dilakukannya konvolusi:



Barikut hasil sharpening untuk gambar awal:



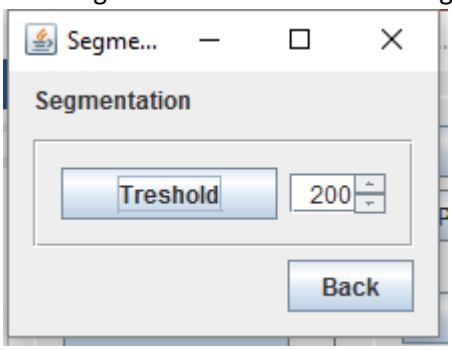
Berikut hasil edge detection untuk gambar awal:



$\begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$ Matrix kernel edge Detection

$\begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix}$ Matrix kernel image sharpen

10. Fitur Segmentation disini adalah dengan segmentasi menggunakan segmentation threshold saja.

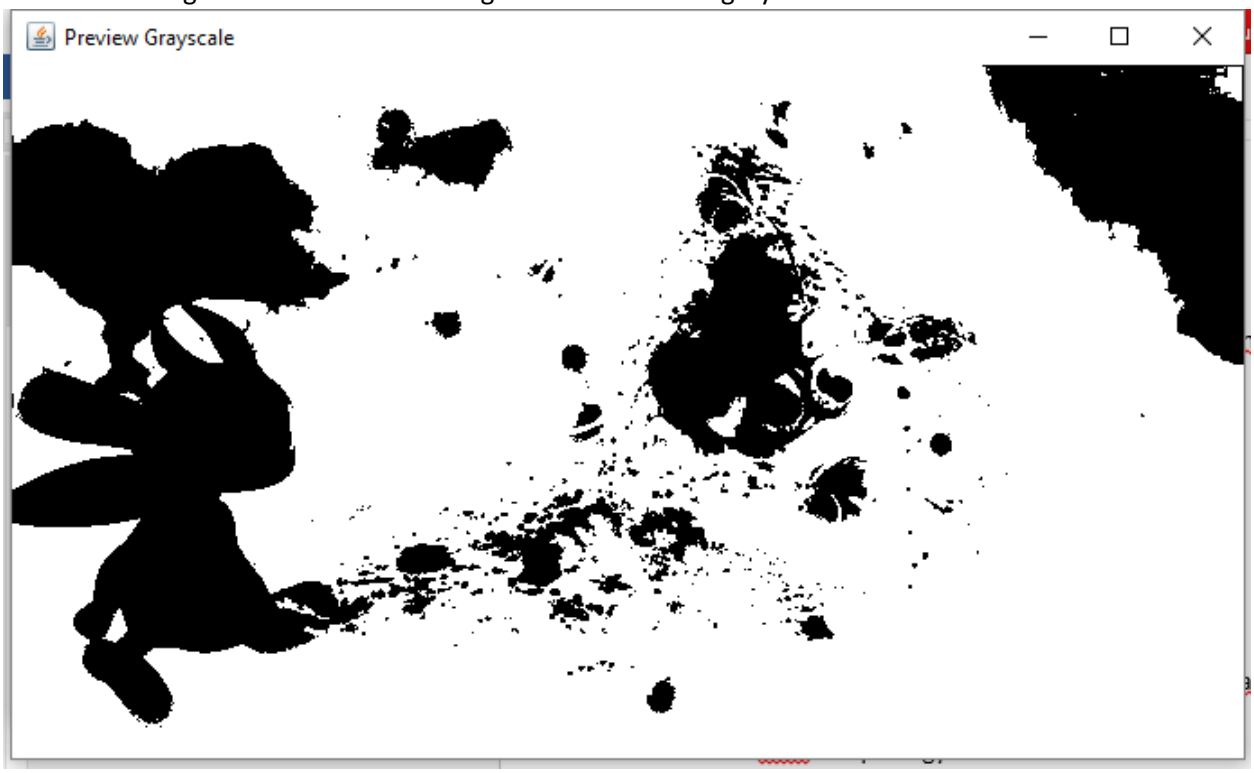


Setiap titik pada gambar akan menjadi bernilai 255 jika lebih kecil dari nilai treshold dan 0 untuk yang sama dengna threshold atau diatasnya

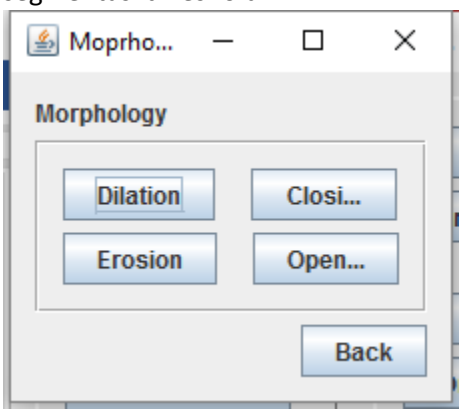
```
int[][] temp = new int[x][y];

for(int i=0; i<x; i++){
    for(int j=0; j<y; j++){
        if(img[i][j]<tresh){
            temp[i][j]=255;
        }
    }
}
return temp;
```

Gambar awal dalam grayscale:

Berikut hasil segmentasi treshold untuk gambar awal dalam graysclae:



11. Fitur Morphology berisi dilasi, erosi, closing, dan opening untuk image dalam grayscale yang sudah disegmentasi. Jika gambar belum disegmentasi, ketika memilih salah satu jenis morphology yang diinginkan, makan akan secara otomatis dilakukannya segmentasi dengan segmentasi threshold.



Untuk dilasi, fungsi akan memperluas area yang berwarna putih. Untuk erosi, fungsi akan mengecilkan area yang berwarna putih. Fungsi closing merupakan fungsi yang menjalankan dilasi terlebih dahulu kemudian erosi. Fungsi opening merupakan fungsi yang menjalankan erosi terlebih dahulu kemudian dilasi. Closing dan opening di sini dilakukan untuk gambar yang sudah dilakukan segmentasi.

Pada erosi, dia akan memberikan nilai titik tersebut 255 jika titik tersebut dan disekeliling kiri, kanan, atas, dan bawah dari titik tersebut berwarna putih. Dengan begitu, area yang berwarna putih akan menjadi lebih sedikit.

```
int[][] temp2 = new int[w+2][h+2];

for(int i=1; i<=w; i++){
    for(int j=1; j<=h; j++){
        if(temp[i][j]==255){
            boolean keliling = true;
            if(temp[i-1][j]!=255)keliling=false;
            if(temp[i][j-1]!=255)keliling=false;
            if(temp[i][j+1]!=255)keliling=false;
            if(temp[i+1][j]!=255)keliling=false;

            if(keliling){
                temp2[i][j]=255;
            }
        }
    }
}

return temp2;
```

Pada dilasi, akan mengecek nilai pada suatu titik, jika titik tersebut putih, makan akan langsung semua titik di sekelilingnya akan menjadi warna putih.

```
int[][] temp2 = new int[w+2][h+2];

for(int i=1; i<=w; i++){
    for(int j=1; j<=h; j++){
        if(temp[i][j]==255){
            boolean keliling = true;
            if(temp[i-1][j]!=255)keliling=false;
            if(temp[i][j-1]!=255)keliling=false;
            if(temp[i][j+1]!=255)keliling=false;
            if(temp[i+1][j]!=255)keliling=false;

            if(keliling){
                temp2[i][j]=255;
            }
        }
    }
}

return temp2;
```

Array temp merupakan array yang berisi hasil threshold gambar, posisinya bergeser ke x+1 dan y+1. Array temp2 merupakan array yang berisi hasil dari dilasi atau erosi.

Fungsi closingRes pada class Closing,java akana mengembalikan array hasil dari closing dimana hasil dari fungsi closing ini adalah hasil dari fungsi dilasi yang diikuti hasil dari fungsi erosi.

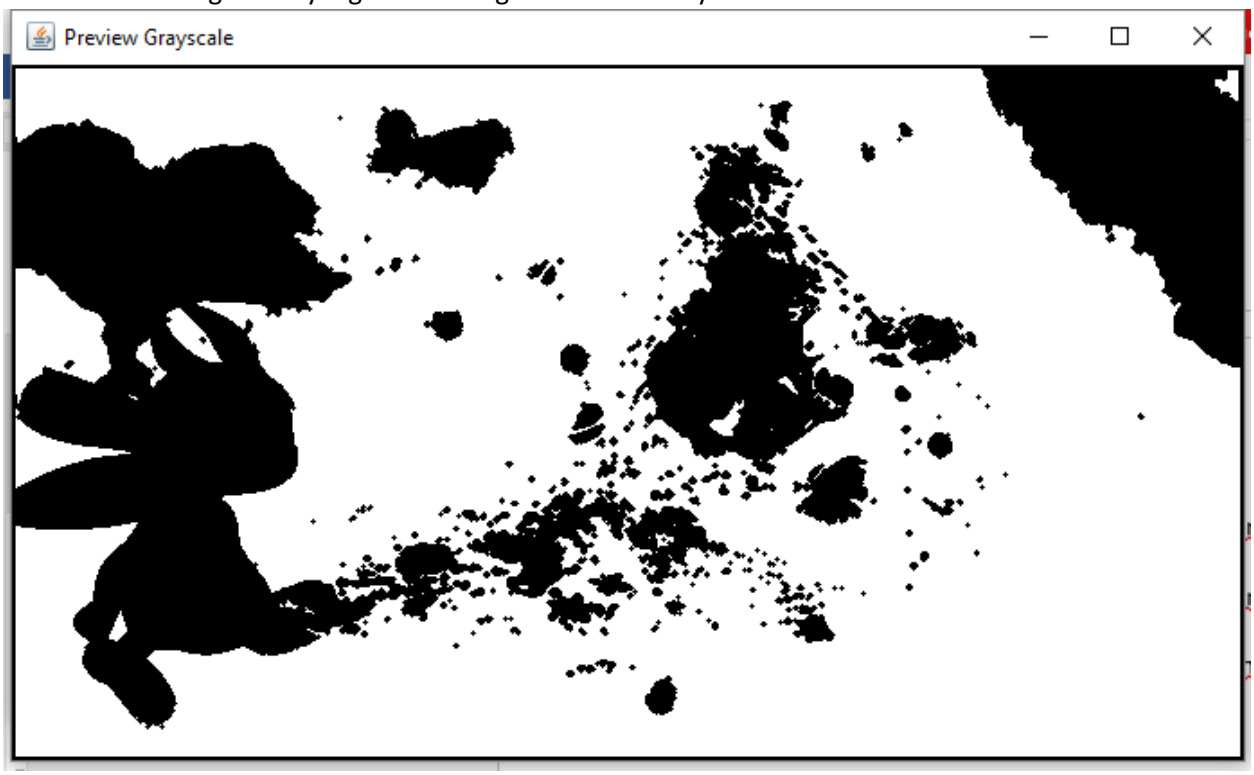
```
public int[][] ClosingRes(int[][] img){
    Dilasi dilasi = new Dilasi();
    Erosi erosi = new Erosi();

    return erosi.ErosiRes(dilasi.DilasiRes(img));
}
```

Fungsi OpeningRes pada class Opening.java akan mengembalikan array hasil dari opening dimana hasil dari fungsi ini adalah hasil dari fungsi erosi yang diikuti hasil dari fungsi dilasi.

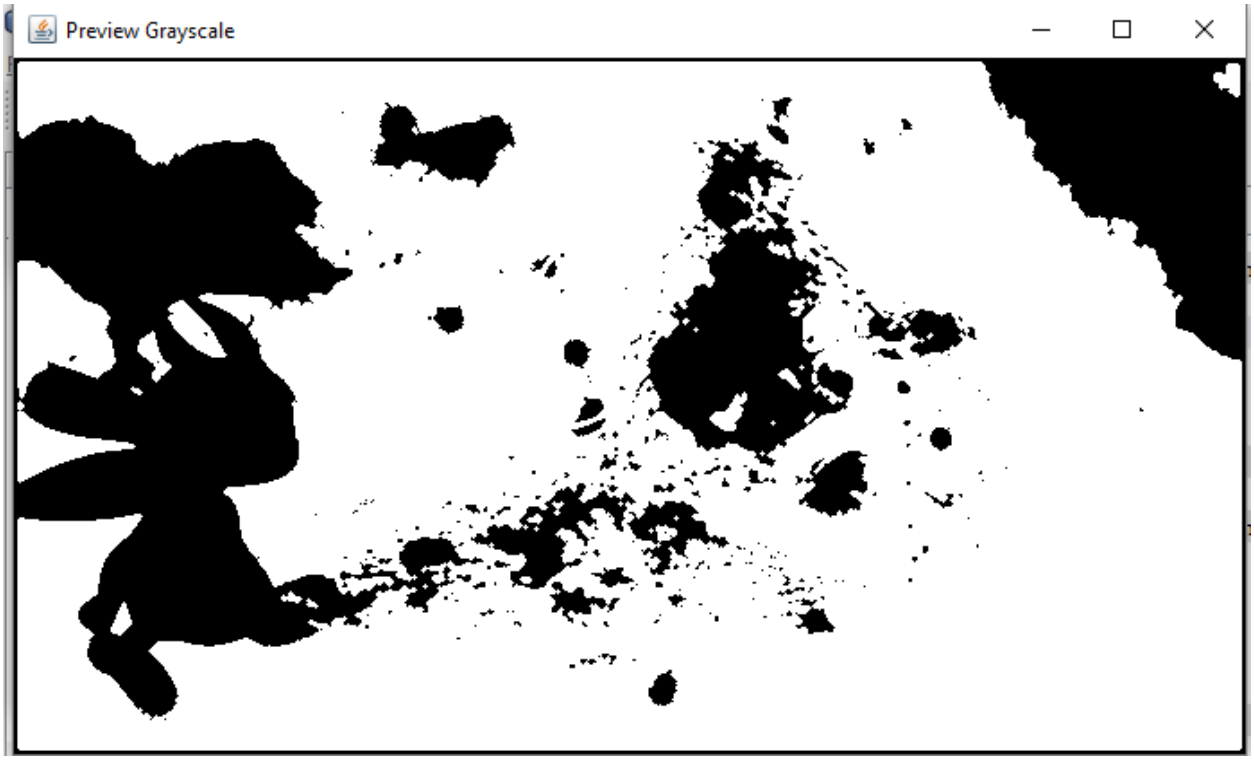
```
public int[][] OpeningRes(int[][] img){
    Dilasi dilasi = new Dilasi();
    Erosi erosi = new Erosi();

    return dilasi.DilasiRes(erosi.ErosiRes(img));
}
```

Hasil dilasi untuk gambar yang sudah disegment sebelumnya:

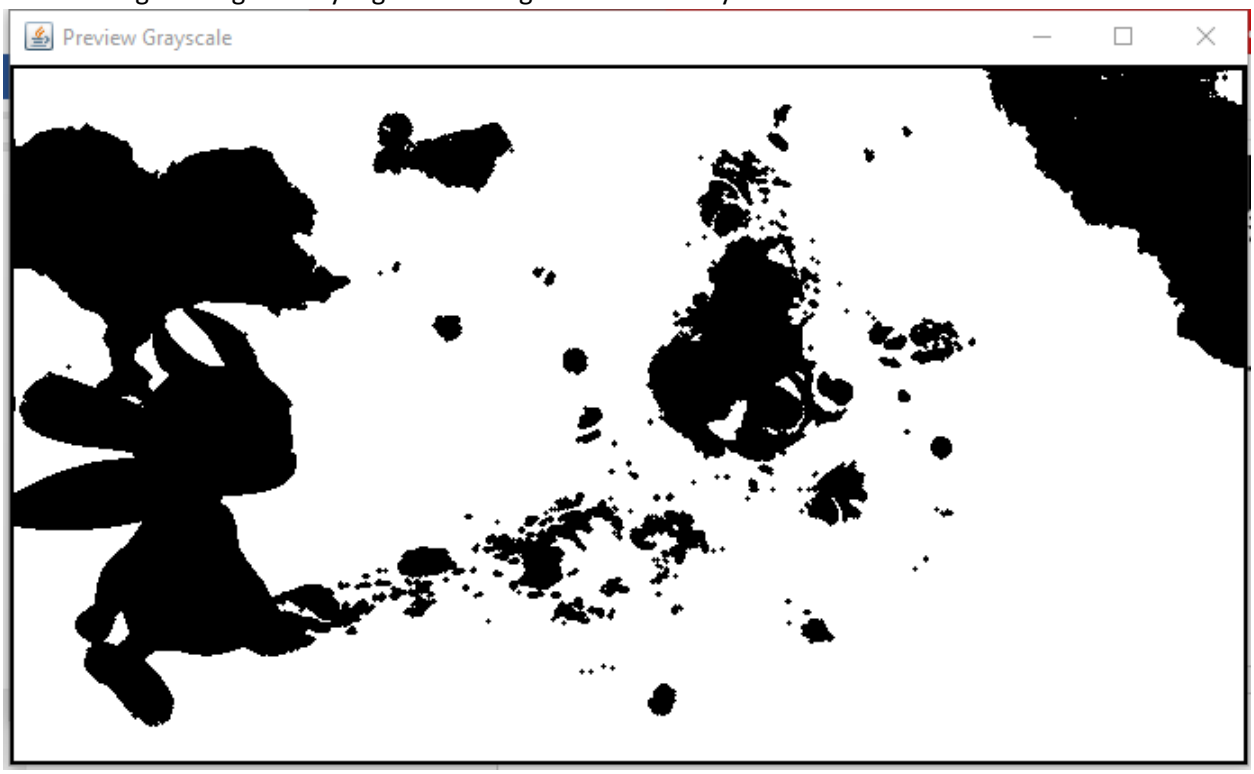
Hasil erosi untuk gambar yang sudah disegment sebelumnya:



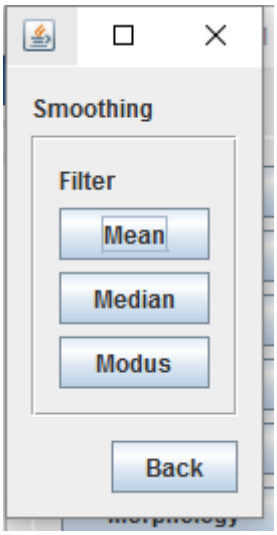
Hasil opening untuk gambar yang sudah disegment sebelumnya:



Hasil closing untuk gambar yang sudah disegment sebelumnya:



12. Fitur smoothing akan melakukan filter dengan pilihan filter yang kita pilih. Mean filter akan memberikan nilai rata rata, median, nilai tengah, dan modus, nilai yang paling banyak keluar.



Pada mean filtering, nilai baru untuk titik tersebut merupakan rata rata dari dari titik dia dan 8 titik di sekelililingnya. Nilai rata-rata tersebut akan menjadi nilai baru untuk titik tersebut

```
for(int i=0; i<w; i++){
    for(int j=0; j<h; j++){
        int sum = 0;

        for(int a=0; a<3; a++){
            for(int b=0; b<3; b++){
                if((i-1)>=0 && (j-1)>=0 && (i+1)<w && (j+1)<h)
                    sum += img[i+a-1][j+b-1];
            }
        }

        temp[i][j] = sum/9;
    }
}
```

Pada median filtering, nilai baru untuk titik tersebut merupakan nilai tengah dari barisan nilai antara dirinya dan sekelilingnya. barisan itu diurutkan terlebih dahulu dan kemudian nilai tengah pada baris itu lah yang menjadi nilai baru untuk titik tersebut.

```
int x = 0;

for(int a=0; a<3; a++){
    for(int b=0; b<3; b++){
        if((i-1)>=0 && (j-1)>=0 && (i+1)<w && (j+1)<h){
            bil[x] += img[i+a-1][j+b-1];
        }else{
            bil[x] = 0;
        }
        x++;
    }
}
Arrays.sort(bil);
temp[i][j] = bil[4];
```

Pada modus filtering, nilai baru titik tersebut adalah nilai yang paling sering keluar dari baris nilai titik tersebut dan titik di sekelilingnya. Jika kemmunculan setiap nilai itu 1, maka nilai titik tersebut tidak berubah.

Untuk menentukan baris:

```
for(int a=0; a<3; a++){
    for(int b=0; b<3; b++){
        if((i-1)>=0 && (j-1)>=0 && (i+1)<w && (j+1)<h){
            bil[x] += img[i+a-1][j+b-1];
        }else{
            bil[x] = 0;
        }
        x++;
    }
}
```

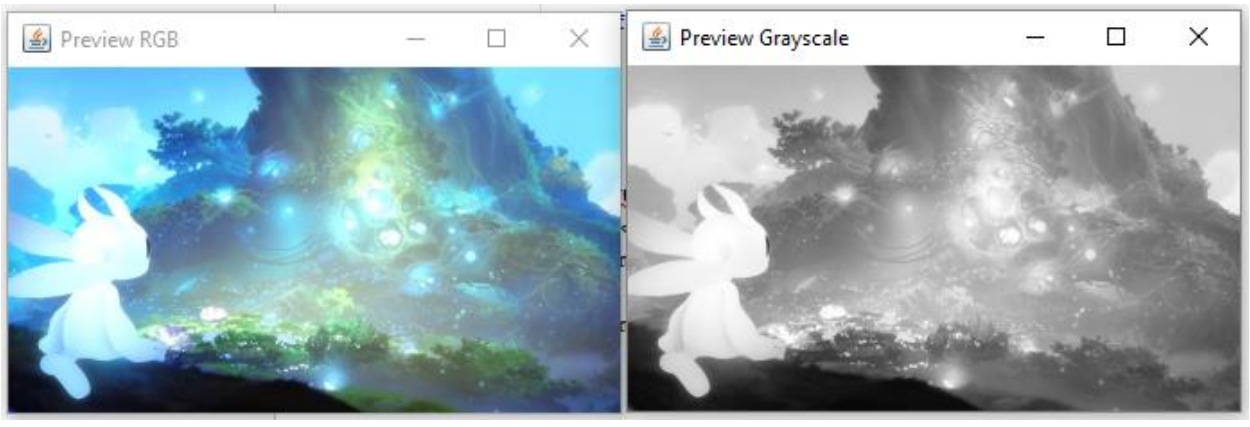
Untuk mencari modus:

```
x = -1; int max=-1;
for(int a=0; a<9; a++){
    int n = 0;
    for(int b=0; b<9; b++){
        if(bil[a]==bil[b])n++;
    }
    if(n>x && n>1){
        max = n;
        x=a;
    }
}
```

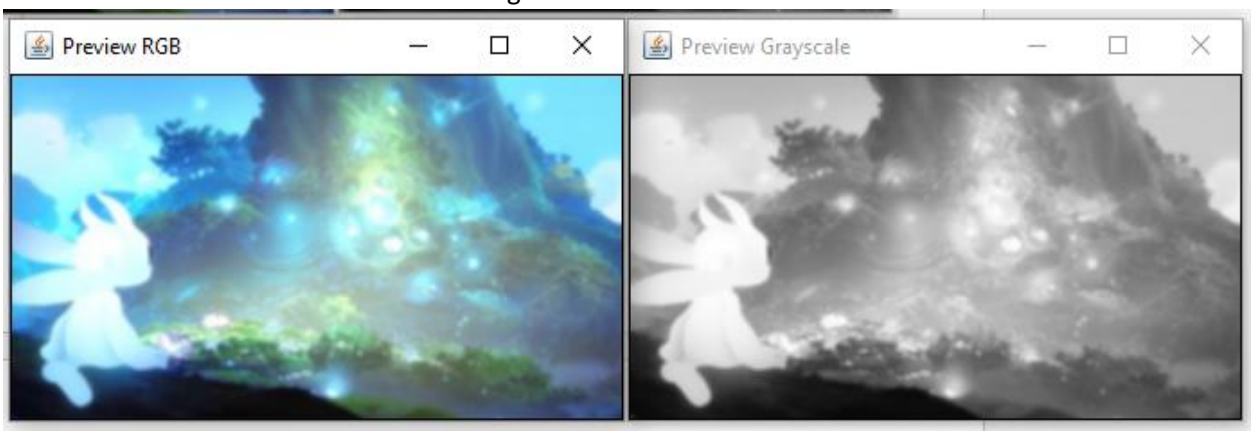
Nilai baru:

```
if(max<0){
    temp[i][j] = bil[4];
}else{
    temp[i][j] = bil[x];
}
```

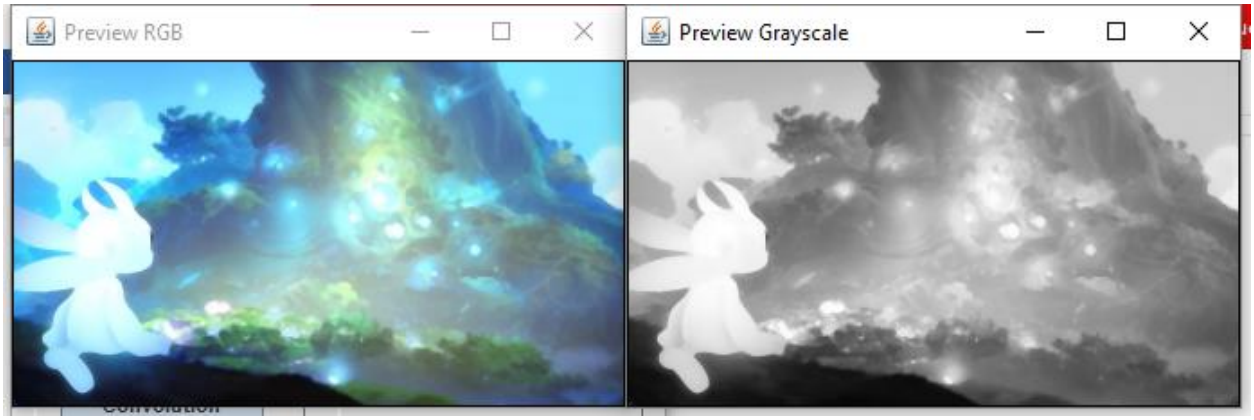
Gambar awal sebelum dilakukan smoothing:



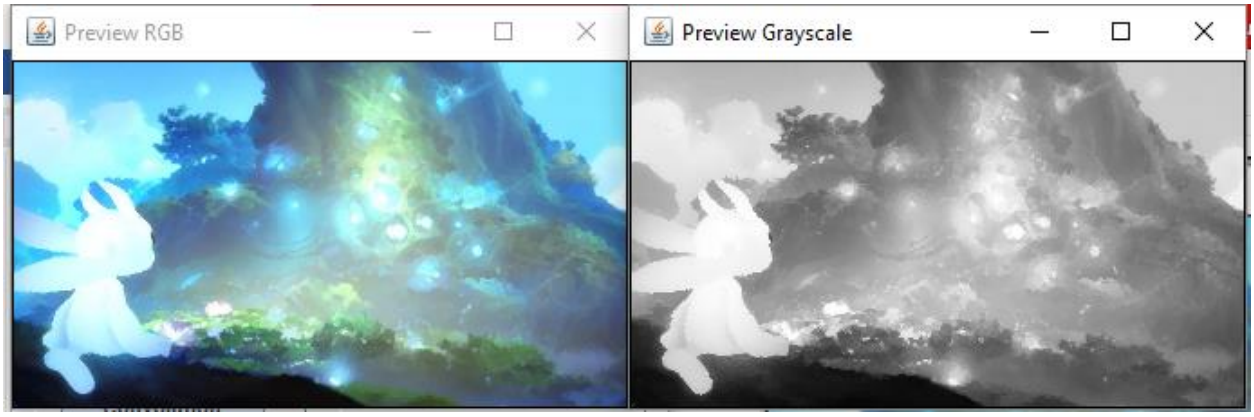
Gambar setelah dilakukan mean smoothing:



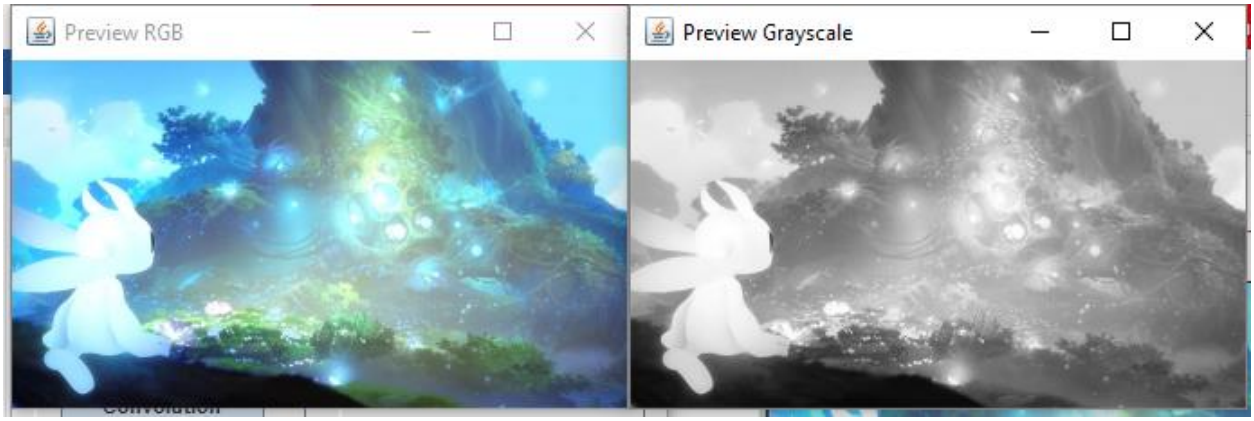
Gambar setelah dilakukan median smoothing:



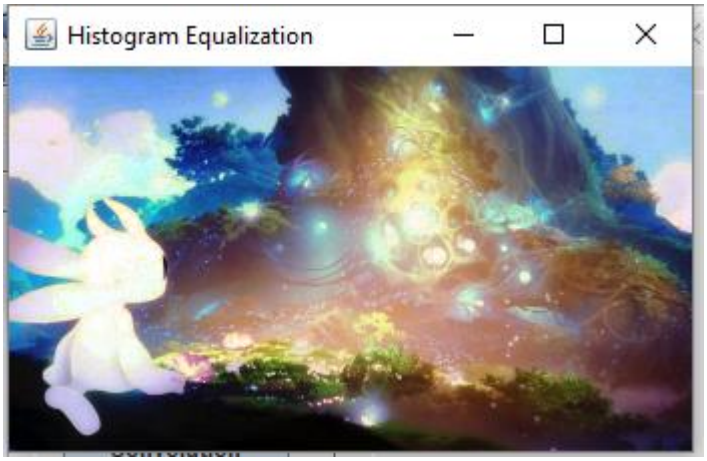
Gambar setelah dilakukan modus smoothing:



13. Fitur histogram akan memberikan persebaran nilai untuk setiap warna. Menghitung jumlah setiap warna yang dikelompokkan berddasarkan nilai kemunculannya mulai dari nilai 0 sampai nilai 255.



Untuk dari gambar tersebut, didapatkan persebarannya pada file histogram.txt. fungsi ini akan menampilkan juga Histogram Equalizationnya namun hanya dalam rgbnya saja. Berikut adalah Equalizationnya:



Pertama, diambil semua nilai r, g, dan b, dari gambar, kemudian dihitung jumlah untuk setiap warna pada setiap level. Kemudian dicari persebarannya. Dari hasil persebaran tersebut, bisa dilihat equalizationnya.

```
arrRGB = getRGB();
sumRGB = sumRGB(arrRGB);
cdfRGB = cdfRGB(sumRGB);
this.img = histEq(cdfRGB);
```

Dari array gambar, langsung dimasukkan ke dalam array baru untuk menampung semua nilai dari warna.

```
public int[][] getRGB(){
    int[][] arrRGB = new int[4][w*h];

    int k = 0;
    for(int i=0; i<w; i++){
        for(int j=0; j<h; j++){
            arrRGB[0][k]=this.R[i][j];
            arrRGB[1][k]=this.G[i][j];
            arrRGB[2][k]=this.B[i][j];
            arrRGB[3][k]=this.Grays[i][j];
            k++;
        }
    }

    return arrRGB;
}
```

Pada fungsi sumRGB ini, menghitung jumlah setiap warna untuk setiap level dari level 0 sampai level 255. Nilai dari fungsi ini yang akan di print untuk histogram.

```
public int[][] sumRGB(int[][] arrRGB){
    int[][] rgbSum = new int[4][256]; int size = this.w*this.h;
    for(int k=0; k<256; k++){
        int sumR = 0; int sumG = 0; int sumB = 0; int sumGray = 0;
        for(int i=0; i<size; i++){if(arrRGB[0][i]==k)sumR++;}
        rgbSum[0][k]=sumR;
        for(int i=0; i<size; i++){if(arrRGB[1][i]==k)sumG++;}
        rgbSum[1][k]=sumG;
        for(int i=0; i<size; i++){if(arrRGB[2][i]==k)sumB++;}
        rgbSum[2][k]=sumB;
        for(int i=0; i<size; i++){if(arrRGB[3][i]==k)sumGray++;}
        rgbSum[3][k]=sumGray;
    }
    return rgbSum;
}
```

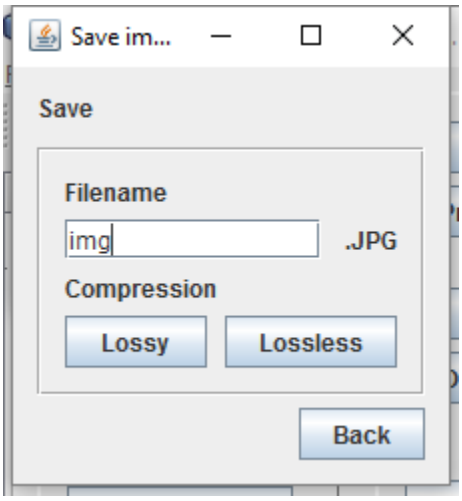
Fingsi cdfRGB merupakan fungsi untuk melihat cdf. Awalnya dicari probabilitas setiap warna pada level tersebut. Kemudian cdfnya merupakan kumulatif hasil dari penjumlahan probabilitasnya.

```
public float[][] cdfRGB(int[][] rgbSum){
    float size = (w*h);
    float[][] rgbProb=new float [4][256];
    float[][] rgbCDF=new float [4][256];
    for(int i=0; i<256; i++){
        rgbProb[0][i]=(float)rgbSum[0][i]/size;
        rgbProb[1][i]=(float)rgbSum[1][i]/size;
        rgbProb[2][i]=(float)rgbSum[2][i]/size;
        rgbProb[3][i]=(float)rgbSum[2][i]/size;
    }
    float sumR=0,sumG=0,sumB=0,sumGray=0;
    for(int i=0; i<256; i++){
        sumR=sumR+rgbProb[0][i]; rgbCDF[0][i]=sumR;
        sumG=sumG+rgbProb[1][i]; rgbCDF[1][i]=sumG;
        sumB=sumB+rgbProb[2][i]; rgbCDF[2][i]=sumB;
        sumGray=sumGray+rgbProb[3][i]; rgbCDF[2][i]=sumGray;
    }
    return rgbCDF;
}
```

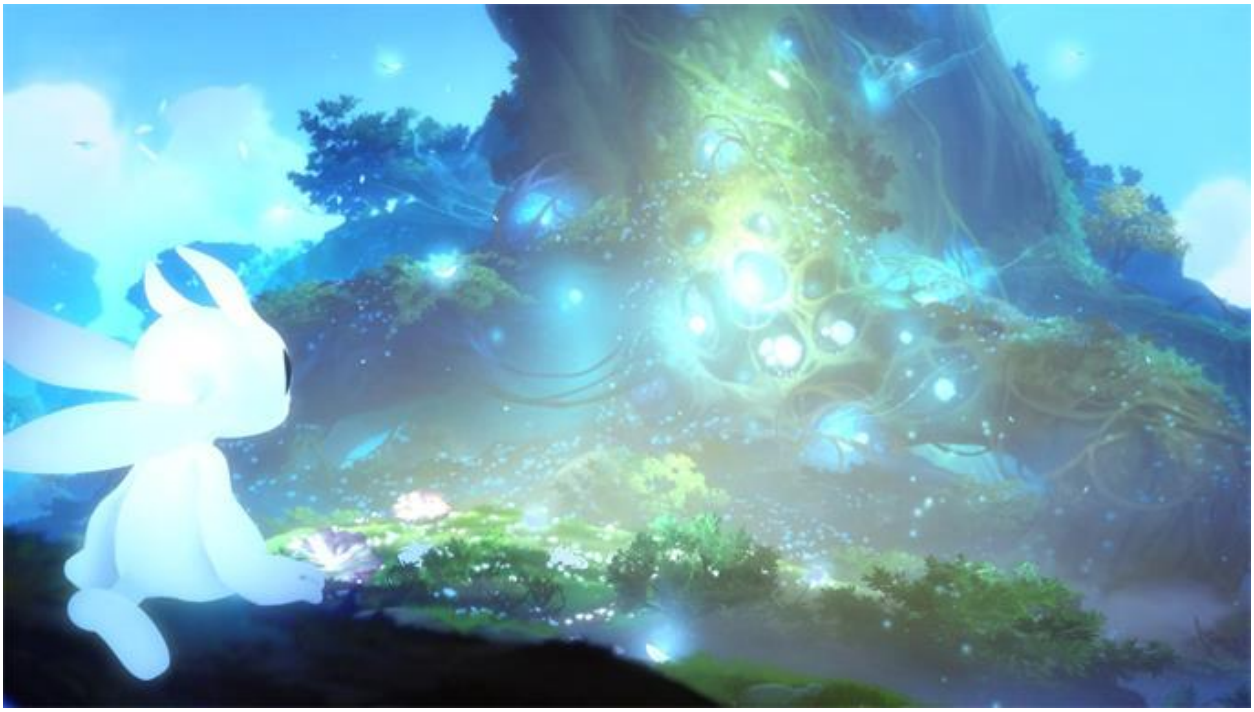
Nilai equalizationnya nilai hasil pembulatan dari 255*nilai cdf untuk setiap level pada setiap warna. Warna barunya dilihat dari array equalizationnya.

```
public BufferedImage histEq(float[][] rgbCDF){
    BufferedImage imgEq=new BufferedImage(w, h, BufferedImage.TYPE_INT_RGB);
    for(int i=0; i<256; i++){
        rgbEq[0][i]=Math.round(255*rgbCDF[0][i]);
        rgbEq[1][i]=Math.round(255*rgbCDF[1][i]);
        rgbEq[2][i]=Math.round(255*rgbCDF[2][i]);
        grayEq[i]=Math.round(255*rgbCDF[3][i]);
    }
    int r=0; int g=0; int b=0; int gray=0;
    for(int i=0; i<w; i++){
        for(int j=0; j<h; j++){
            r=this.R[i][j]; g=this.G[i][j]; b=this.B[i][j];

            r=rgbEq[0][r]; g=rgbEq[1][g]; b=rgbEq[2][b];

            Color c1= new Color(r,g,b);
            imgEq.setRGB(i, j, c1.getRGB());
        }
    }
    return imgEq;
}
```

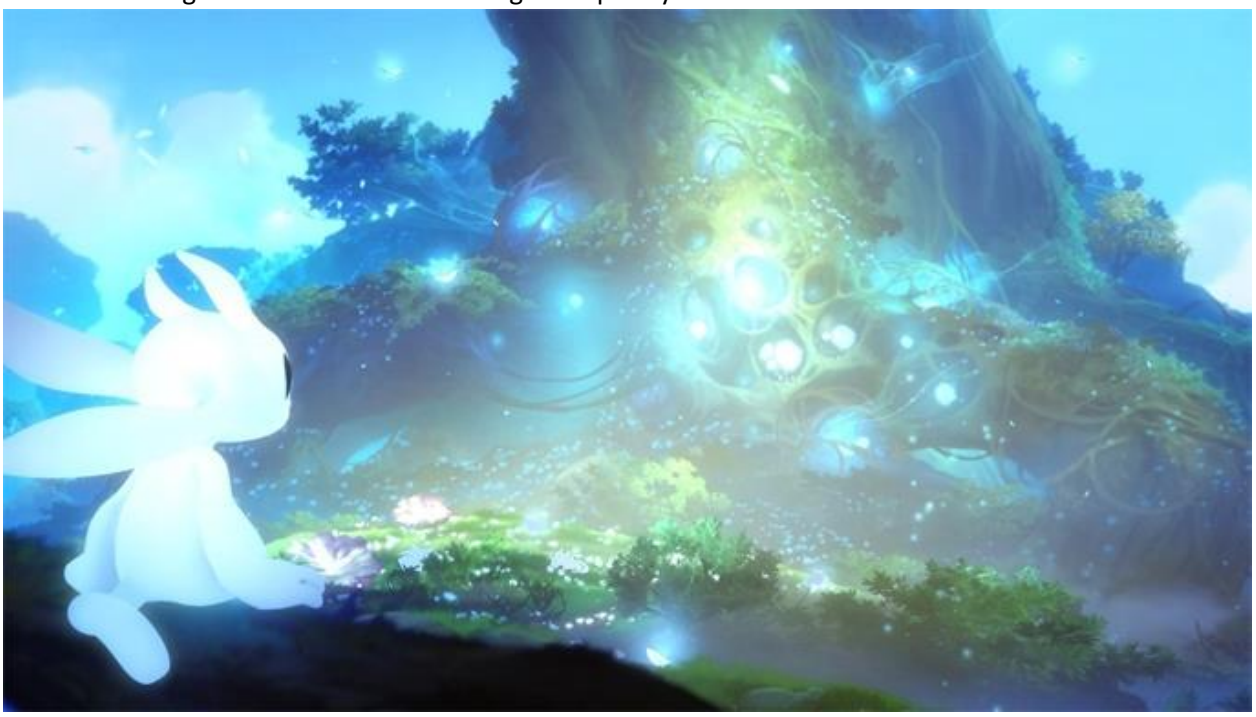
14. Fitur save akan menyimpan gambar. Di sana diberikan pilihan untuk disimpan dengan kompressi lossless atau lossy.



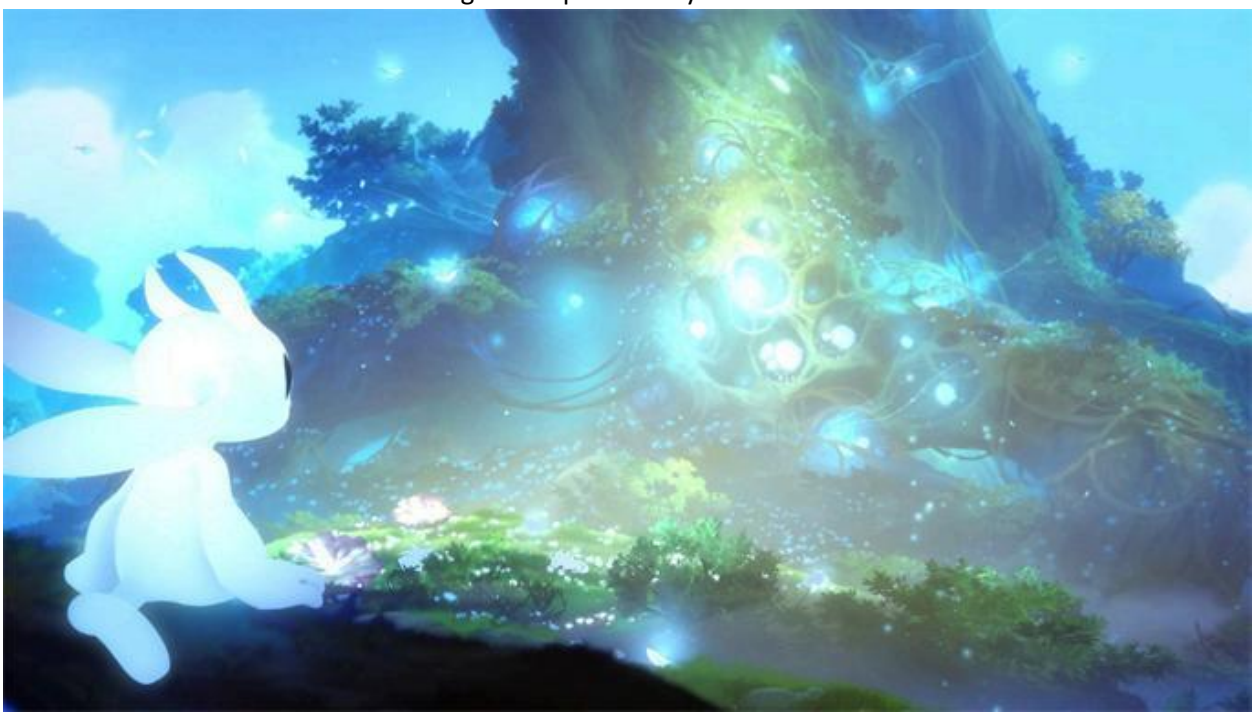
Untuk gambar awal berikut, akan disave dengan dua opsi tersebut. Berikut gambar asli:



Berikut adalah gambar setelah di-save denga kompressy lossless:



Dan di bawah ini adalah hasil save dengan kompressi lossy:



Berikut merupakan perbandingan ukuran untuk ketiga file hasil save:

imgRGB Lossless	5/18/2017 6:37 PM	JPG File	177 KB
imgRGB Lossy	5/18/2017 6:37 PM	JPG File	33 KB
Ori-and-the-Blind-Forest	5/18/2017 3:01 PM	JPG File	187 KB

“Ori-and-the-Blind-Forest.jpg” berukuran 187 KB yang merupakan gambar asli. “imgRGB Lossless.jpg” merupakan hasil save dengan kompressy lossless dan “imgRGB Lossy.jpg” merupakan hasil dari kompressy lossless.

Pada lossless, fungsi di bawah ini untuk menghitung jumlah warna yang dimasukkan ke dalam array list. Dalam arraylist of integer tersebut akan menyimpan nilai dengan urutan, nilai untuk warnanya diikuti berapa kalai kemunculannya.

```
public ArrayList<Integer> compressThis(int[][] img){
    ArrayList<Integer> temp = new ArrayList<>();
    int v = -1; int n = 0;
    for(int y=0; y<h; y++){
        for(int x=0; x<w; x++){
            if(n==0){
                v = img[x][y]; n++;
            }else{
                if(v==img[x][y]){
                    n++;
                }else{
                    temp.add(v);
                    temp.add(n);
                    v = img[x][y];
                    n = 1;
                }
            }
        }
    }
    return temp;
}
```

Pada lossy, variasi warnanya menjadi tidak banyak, setiap range 5 level dijadikan 1. Nilai baru untuk warna tersebut diambil dari nilai bawah range tersebut. findLoss untuk menentukan nilai tersebut pada range mana, kemudian nilai baru dari warna tersebut tinggal dikali dengan 5 saja.

```
Color c;
for(int i=0; i<w; i++){
    for(int j=0; j<h; j++){
        int rVal = findLoss(R[i][j])*5;
        int gVal = findLoss(G[i][j])*5;
        int bVal = findLoss(B[i][j])*5;
        c = new Color(rVal, gVal, bVal);
        this.imgRGB.setRGB(i,j,c.getRGB());
        int grayColor = findLoss(gray[i][j])*5;
        c = new Color(grayColor, grayColor, grayColor);
        this.imgGray.setRGB(i,j,c.getRGB());
    }
}
public int findLoss(int val){
    return val/5;
}
```