IF2211 STRATEGI ALGORITMA

LAPORAN TUGAS KECIL 2: KOMPRESI GAMBAR DENGAN METODE QUADTREE

OLEH:

MUHAMMAD NAZIH NAJMUDIN - 13523144

MUHAMMAD RIZAIN FIRDAUS - 13523164

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

04/2025

A. Deskripsi Tugas

Quadtree adalah struktur data hirarkis yang digunakan untuk membagi ruang atau data menjadi bagian yang lebih kecil, yang sering digunakan dalam pengolahan gambar. Dalam konteks kompresi gambar, *Quadtree* membagi gambar menjadi blok-blok kecil berdasarkan keseragaman warna atau intensitas piksel. Prosesnya dimulai dengan membagi gambar menjadi empat bagian, lalu memeriksa apakah setiap bagian memiliki nilai yang seragam berdasarkan analisis sistem warna RGB, yaitu dengan membandingkan komposisi nilai merah (R), hijau (G), dan biru (B) pada piksel-piksel di dalamnya. Jika bagian tersebut tidak seragam, bagian tersebut akan terus dibagi hingga mencapai tingkat keseragaman tertentu atau ukuran minimum yang ditentukan.

Dalam implementasi teknis, sebuah *Quadtree* direpresentasikan sebagai simpul (*node*) dengan maksimal empat anak (*children*). Simpul daun (*leaf*) merepresentasikan area gambar yang seragam, sementara simpul internal menunjukkan area yang masih membutuhkan pembagian lebih lanjut. Setiap simpul menyimpan informasi seperti posisi (x, y), ukuran (*width*, *height*), dan nilai rata-rata warna atau intensitas piksel dalam area tersebut. Struktur ini memungkinkan pengkodean data gambar yang lebih efisien dengan menghilangkan redundansi pada area yang seragam. *Quadtree* sering digunakan dalam algoritma kompresi *lossy* karena mampu mengurangi ukuran file secara signifikan tanpa mengorbankan detail penting pada gambar.

Pada tugas kecil 2 ini, akan dibuat implementasi *Quadtree* dengan memanfaatkan algoritma *divide and conquer* untuk melakukan kompresi gambar.

B. Algoritma Divide and Conquer yang Digunakan

Para program ini, kompresi gambar dilakukan dengan algoritma *divide and conquer*, yaitu algoritma penyelesaian masalah dengan menyelesaikan masalah yang telah dipecah menjadi permasalahan-permasalahan yang lebih kecil, lalu disatukan kembali. Unsur *divide* pada kompresi *quadtree* terletak pada bagaimana quadtree itu dibuat, dan unsur *conquer* terletak pada bagaimana gambar dihasilkan dari penggabungan *node-node* pada *quadtree* menjadi gambar. Algoritma *divide and conquer* yang digunakan adalah sebagai berikut.

1. Program meminta input gambar serta inisialisasi persiapan datanya, mencakup metode penghitungan galat, ambang batas galat, ukuran blok minimum, serta alamat hasil kompresi.
2. Gambar ‘dipecah’ secara logik, dengan cara membangun objek *quadtree* yang merepresentasikan hasil gambar terkompresi.

3. Setiap *node* dalam *quadtree* merepresentasikan blok gambar dengan menyimpan posisi piksel sebagai patokan blok, rentang panjang dan lebar blok, serta warna yang dimiliki blok. Warna tersebut merupakan hasil normalisasi warna-warna piksel gambar awal pada patokan dan rentang panjang lebar blok.
4. Setiap *node* juga menyimpan pointer kepada 4 buah blok yang disimpan dalam array dan merupakan hasil pecahan kembali blok tersebut apabila blok masih bisa dipecah berdasarkan konfigurasi.
5. Dalam proses pembuatan *quadtree*, dilakukan proses rekursif pada setiap *node* untuk membuat *node*, serta mengecek apakah blok yang direpresentasikan *node* tersebut harus dipecah kembali sehingga akan dibuat 4 buah *node* sebagai cabangnya.
6. Pengecekan tersebut mencakup apakah blok belum memenuhi ukuran minimum, atau galat normalisasi warna masih diatas ambang batas berdasarkan metode galat yang telah dipilih, yaitu *variance*, *Mean Absolute Deviation* (*MAD*), *Max Pixel Difference* (*MPD*), atau *entropy*.
7. Apabila blok harus dipecah kembali, dibuat 4 buah *node* sebagai cabangnya. Apabila blok tidak dipecah kembali, array *node* diisi null dan *node* tersebut menjadi *leaf* dari *quadtree*. Proses dilakukan secara rekursif untuk setiap *node*.
8. Setelah *quadtree* rampung, gambar dibuat dengan merealisasikan blok-blok pada setiap *leaf node*, dan menggabungkannya menjadi satu gambar utuh dengan cara me-*render* gambar baru dengan konfigurasi-konfigurasi posisi piksel, rentang panjang dan lebar, serta warna dari setiap *leaf node*.
9. Gambar baru yang dibuat dari *quadtree* disimpan dalam alamat yang telah diinput sebelumnya.

C. Kode Sumber

a. Pranala Repositori

Repositori GitHub tugas ini dapat dikunjungi pada laman berikut:
https://github.com/inRiza/Tucil2_13523144_13523164/releases/latest

b. Struktur Repositori

```
Directory structure:
└── inriza-tucil2_13523144_13523164/
    ├── README.md
    ├── run.cmd
    ├── bin/
```

```
│   └── src/
│       └── app/
│           └── utils/
├── src/
│   └── app/
│       ├── ErrorMeasurement.java
│       ├── ImageProcessor.java
│       ├── Main.java
│       ├── QuadTreeNode.java
│       └── utils/
│           ├── GIFGenerator.java
│           ├── GifSequenceWriter.java
│           └── ImageUtils.java
└── test/
    ├── input/
    └── output/
```

c. Implementasi Program

i. ErrorMeasurement.java

```
package src.app;

import java.awt.image.BufferedImage;
import java.util.HashMap;

public class ErrorMeasurement {

    // Konstanta untuk perhitungan SSIM
    private static final double C1 = Math.pow(0.01 * 255, 2);
    private static final double C2 = Math.pow(0.03 * 255, 2);

    // Bobot untuk channel RGB
    private static final double WEIGHT_R = 0.299;
    private static final double WEIGHT_G = 0.587;
    private static final double WEIGHT_B = 0.114;

    // Variance
    public static double Variance(QuadTreeNode node, BufferedImage
image) {
        double sumR = 0, sumG = 0, sumB = 0;
        double sumSqR = 0, sumSqG = 0, sumSqB = 0;
        int pixelCount = node.width * node.height;

        for (int y = node.y; y < Math.min(node.y + node.height,
image.getHeight()); y++) {
            for (int x = node.x; x < Math.min(node.x + node.width,
image.getWidth()); x++) {
                int rgb = image.getRGB(x, y);
                int r = (rgb >> 16) & 0xFF;
                int g = (rgb >> 8) & 0xFF;
                int b = rgb & 0xFF;

                sumR += r;
                sumG += g;
                sumB += b;
                sumSqR += r * r;
                sumSqG += g * g;
                sumSqB += b * b;
            }
        }

        double varianceR = (sumSqR - (sumR * sumR) / pixelCount) /
pixelCount;
        double varianceG = (sumSqG - (sumG * sumG) / pixelCount) /
pixelCount;
        double varianceB = (sumSqB - (sumB * sumB) / pixelCount) /
```

```
pixelCount;

        return (varianceR + varianceG + varianceB) / 3;
    }

    // Mean Absolute Deviation
    public static double MeanAbsoluteDeviation(QuadTreeNode node,
BufferedImage image) {
        double sumR = 0, sumG = 0, sumB = 0;
        double N = node.width * node.height;

        double[] averages = { 0, 0, 0 };
        AverageColor(node, image, averages);
        double avgR = averages[0];
        double avgG = averages[1];
        double avgB = averages[2];

        for (int y = node.y; y < Math.min(node.y + node.height,
image.getHeight()); y++) {
            for (int x = node.x; x < Math.min(node.x + node.width,
image.getWidth()); x++) {
                int rgb = image.getRGB(x, y);
                sumR += Math.abs(((rgb >> 16) & 0xFF) - avgR);
                sumG += Math.abs(((rgb >> 8) & 0xFF) - avgG);
                sumB += Math.abs((rgb & 0xFF) - avgB);
            }
        }
        return (sumR + sumG + sumB) / (3 * N);
    }

    // Max Pixel Difference
    public static double MaxPixelDifference(QuadTreeNode node,
BufferedImage image) {
        int[][] maxmin = { { 0, 0 }, { 0, 0 }, { 0, 0 } };
        MaxMinColor(node, image, maxmin);
        int maxR = maxmin[0][0];
        int minR = maxmin[0][1];
        int maxG = maxmin[1][0];
        int minG = maxmin[1][1];
        int maxB = maxmin[2][0];
        int minB = maxmin[2][1];
        return ((maxR + maxG + maxB) - (minR + minG + minB)) / 3.0;
    }

    // Entropy
    public static double Entropy(QuadTreeNode node, BufferedImage
image) {
        @SuppressWarnings("unchecked")
        HashMap<Integer, Double>[] probs = (HashMap<Integer,
Double>[]) new HashMap[3];
        for (int i = 0; i < 3; i++) {
            probs[i] = new HashMap<>();
        }
        ProbabilityDistribution(node, image, probs);

        double entR = 0, entG = 0, entB = 0;
        for (int y = node.y; y < Math.min(node.y + node.height,
image.getHeight()); y++) {
            for (int x = node.x; x < Math.min(node.x + node.width,
image.getWidth()); x++) {
                int rgb = image.getRGB(x, y);
                int r = (rgb >> 16) & 0xFF;
                int g = (rgb >> 8) & 0xFF;
                int b = rgb & 0xFF;

                entR -= (probs[0].get(r) * Math.log(probs[0].get(r)) /
Math.log(2));
                entG -= (probs[1].get(g) * Math.log(probs[1].get(g)) /
Math.log(2));
                entB -= (probs[2].get(b) * Math.log(probs[2].get(b)) /
```

```
Math.log(2));
            }
        }
        return (entR + entG + entB) / 3.0;
    }

    // Helper Method
    private static void AverageColor(QuadTreeNode node, BufferedImage
image, double[] colorsResult) {
        long sumR = 0;
        long sumG = 0;
        long sumB = 0;
        int pixelCount = 0;

        for (int y = node.y; y < Math.min(node.y + node.height,
image.getHeight()); y++) {
            for (int x = node.x; x < Math.min(node.x + node.width,
image.getWidth()); x++) {
                int rgb = image.getRGB(x, y);
                sumR += (rgb >> 16) & 0xFF;
                sumG += (rgb >> 8) & 0xFF;
                sumB += rgb & 0xFF;
                pixelCount++;
            }
        }
        if (pixelCount > 0) {
            colorsResult[0] = (double) (sumR / pixelCount);
            colorsResult[1] = (double) (sumG / pixelCount);
            colorsResult[2] = (double) (sumB / pixelCount);
        }
    }

    private static void MaxMinColor(QuadTreeNode node, BufferedImage
image, int[][] colorsResult) {
        int maxR = 0;
        int maxG = 0;
        int maxB = 0;
        int minR = 255;
        int minG = 255;
        int minB = 255;
        for (int y = node.y; y < Math.min(node.y + node.height,
image.getHeight()); y++) {
            for (int x = node.x; x < Math.min(node.x + node.width,
image.getWidth()); x++) {
                int rgb = image.getRGB(x, y);
                int r = (rgb >> 16) & 0xFF;
                int g = (rgb >> 8) & 0xFF;
                int b = rgb & 0xFF;

                maxR = Math.max(r, maxR);
                maxG = Math.max(g, maxG);
                minB = Math.max(b, maxB);

                minR = Math.min(r, minR);
                minG = Math.min(g, minG);
                minB = Math.min(b, minB);
            }
        }
        colorsResult[0][0] = maxR;
        colorsResult[0][1] = minR;
        colorsResult[1][0] = maxG;
        colorsResult[1][1] = minG;
        colorsResult[2][0] = maxB;
        colorsResult[2][1] = minB;
    }

    private static void ProbabilityDistribution(QuadTreeNode node,
BufferedImage image,
            HashMap<Integer, Double>[] probabilities) {
```

```
        for (int y = node.y; y < Math.min(node.y + node.height,
image.getHeight()); y++) {
            for (int x = node.x; x < Math.min(node.x + node.width,
image.getWidth()); x++) {
                int rgb = image.getRGB(x, y);
                int r = (rgb >> 16) & 0xFF;
                int g = (rgb >> 8) & 0xFF;
                int b = rgb & 0xFF;

                probabilities[0].computeIfPresent(r, (_, val) ->
val++);
                probabilities[0].putIfAbsent(r, 1.0);
                probabilities[1].computeIfPresent(g, (_, val) ->
val++);
                probabilities[1].putIfAbsent(g, 1.0);
                probabilities[2].computeIfPresent(b, (_, val) ->
val++);
                probabilities[2].putIfAbsent(b, 1.0);
            }
        }

        for (int i = 0; i < 2; i++) {
            probabilities[i].replaceAll((_, val) -> val / (node.width
* node.height));
        }
    }

    // Structural Similarity Index (SSIM)
    public static double SSIM(QuadTreeNode node, BufferedImage image)
{
        double ssimR = 0, ssimG = 0, ssimB = 0;
        int validPixels = 0;

        // Hitung SSIM untuk setiap channel dalam blok
        for (int y = node.y; y < Math.min(node.y + node.height,
image.getHeight()); y++) {
            for (int x = node.x; x < Math.min(node.x + node.width,
image.getWidth()); x++) {
                int rgb = image.getRGB(x, y);
                int r = (rgb >> 16) & 0xFF;
                int g = (rgb >> 8) & 0xFF;
                int b = rgb & 0xFF;

                // Hitung SSIM untuk setiap channel
                ssimR += calculatePixelSSIM(r, r);
                ssimG += calculatePixelSSIM(g, g);
                ssimB += calculatePixelSSIM(b, b);

                validPixels++;
            }
        }

        if (validPixels == 0)
            return 0;

        // Rata-rata nilai SSIM untuk setiap channel
        ssimR /= validPixels;
        ssimG /= validPixels;
        ssimB /= validPixels;

        // Kombinasi tertimbang dari channel RGB
        return WEIGHT_R * ssimR + WEIGHT_G * ssimG + WEIGHT_B * ssimB;
    }

    private static double calculatePixelSSIM(int pixel1, int pixel2) {
        double meanX = pixel1;
        double meanY = pixel2;

        double varX = 0;
        double varY = 0;
```

```
        double covXY = 0;

        // Untuk pixel tunggal, varians dan kovarians adalah nol
        // Tetapi kita masih bisa menghitung komponen luminance dan
contrast

        double luminance = (2 * meanX * meanY + C1) / (meanX * meanX +
meanY * meanY + C1);
        double contrast = (2 * Math.sqrt(varX) * Math.sqrt(varY) + C2)
/ (varX + varY + C2);
        double structure = (covXY + C2 / 2) / (Math.sqrt(varX) *
Math.sqrt(varY) + C2 / 2);

        // Untuk pixel tunggal, struktur term adalah 1
        return luminance * contrast * 1.0;
    }
}
```

ii. ImageProcessor.java

```
package src.app;

import java.awt.image.BufferedImage;
import java.awt.Color;

public class ImageProcessor {
    public interface CompressionCallback {
        void onNodeProcessed(QuadTreeNode node, BufferedImage image,
int depth, boolean isLeaf);
    }

    public void compress(QuadTreeNode node, BufferedImage image,
            double threshold, int minBlockSize, int method) {
        compress(node, image, threshold, minBlockSize, method, null,
0, 0.0);
    }

    public void compress(QuadTreeNode node, BufferedImage image,
            double threshold, int minBlockSize, int method,
            CompressionCallback callback) {
        compress(node, image, threshold, minBlockSize, method,
callback, 0, 0.0);
    }

    public void compress(QuadTreeNode node, BufferedImage image,
            double threshold, int minBlockSize, int method,
            CompressionCallback callback, double targetCompression) {
        compress(node, image, threshold, minBlockSize, method,
callback, 0, targetCompression);
    }

    private void compress(QuadTreeNode node, BufferedImage image,
            double threshold, int minBlockSize, int method,
            CompressionCallback callback, int depth, double
targetCompression) {

        if (targetCompression > 0) {
            // Dynamic threshold adjustment
            double currentCompression =
calculateCompressionRatio(node);
            if (currentCompression < targetCompression) {
                threshold *= 0.9; // Decrease threshold to increase
compression
            } else if (currentCompression > targetCompression) {
                threshold *= 1.1; // Increase threshold to decrease
compression
            }
        }
```

```
        if (shouldSplit(node, image, threshold, minBlockSize, method))
{
            // Only capture frame for significant splits
            if (callback != null && (depth <= 3 || depth % 4 == 0)) {
                callback.onNodeProcessed(node, image, depth, false);
            }

            splitNode(node);

            // Process each child node
            for (QuadTreeNode child : node.children) {
                compress(child, image, threshold, minBlockSize,
method, callback, depth + 1, targetCompression);
            }

        } else {
            calculateAverageColor(node, image);

            // Only capture leaf nodes at very shallow depths
            if (callback != null && depth <= 3) {
                callback.onNodeProcessed(node, image, depth, true);
            }
        }
    }

    private void splitNode(QuadTreeNode node) {
        if (!node.isLeaf())
            return;

        int halfWidth = (int) Math.ceil(node.width / 2.0);
        int halfHeight = (int) Math.ceil(node.height / 2.0);

        node.children = new QuadTreeNode[4];
        node.children[0] = new QuadTreeNode(node.x, node.y, halfWidth,
halfHeight); // NW
        node.children[1] = new QuadTreeNode(node.x + halfWidth,
node.y, node.width - halfWidth, halfHeight); // NE
        node.children[2] = new QuadTreeNode(node.x, node.y +
halfHeight, halfWidth, node.height - halfHeight); // SW
        node.children[3] = new QuadTreeNode(node.x + halfWidth, node.y
+ halfHeight,
                node.width - halfWidth, node.height - halfHeight); //
SE
    }

    private boolean shouldSplit(QuadTreeNode node, BufferedImage
image,
            double threshold, int minBlockSize, int method) {
        // Tidak split jika sudah mencapai ukuran minimum
        if (node.width <= minBlockSize || node.height <= minBlockSize)
{
            return false;
        }

        double error = calculateError(node, image, method);
        return error > threshold;
    }

    private double calculateError(QuadTreeNode node, BufferedImage
image, int method) {
        switch (method) {
            case 1:
                return ErrorMeasurement.Variance(node, image);
            case 2:
                return ErrorMeasurement.MeanAbsoluteDeviation(node,
image);
            case 3:
                return ErrorMeasurement.MaxPixelDifference(node,
image);
```

```
            case 4:
                return ErrorMeasurement.Entropy(node, image);
            default:
                return ErrorMeasurement.Variance(node, image);
        }
    }

    private void calculateAverageColor(QuadTreeNode node,
BufferedImage image) {
        long sumR = 0, sumG = 0, sumB = 0;
        int pixelCount = 0;

        // Optimize by using direct pixel access
        int[] pixels = new int[node.width * node.height];
        image.getRGB(node.x, node.y, node.width, node.height, pixels,
0, node.width);

        for (int pixel : pixels) {
            sumR += (pixel >> 16) & 0xFF;
            sumG += (pixel >> 8) & 0xFF;
            sumB += pixel & 0xFF;
            pixelCount++;
        }

        if (pixelCount > 0) {
            node.r = (int) (sumR / pixelCount);
            node.g = (int) (sumG / pixelCount);
            node.b = (int) (sumB / pixelCount);
        }
    }

    private double calculateCompressionRatio(QuadTreeNode node) {
        int totalPixels = node.width * node.height;
        int leafNodes = countLeafNodes(node);
        return (double) leafNodes / totalPixels;
    }

    private int countLeafNodes(QuadTreeNode node) {
        if (node.isLeaf()) {
            return 1;
        }
        int count = 0;
        for (QuadTreeNode child : node.children) {
            count += countLeafNodes(child);
        }
        return count;
    }

    // Utility untuk mendapatkan warna berdasarkan kedalaman
    public static Color getDepthColor(int depth) {
        // Warna akan berubah berdasarkan kedalaman node
        int r = Math.min(255, 50 + depth * 40);
        int g = Math.min(255, 100 + depth * 30);
        int b = Math.min(255, 150 + depth * 20);
        return new Color(r, g, b);
    }
}
```

iii. Main.java

```
package src.app;

import src.app.utils.ImageUtils;
import src.app.utils.GIFGenerator;
import java.awt.image.BufferedImage;
import java.awt.Graphics2D;
import java.io.File;
import java.io.IOException;
```

```
import java.time.Duration;
import java.time.Instant;
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        // Bersihkan Terminal
        clearTerminal();

        Scanner scanner = new Scanner(System.in);

        try {
            System.out.println("[=================[ Program Kompresi
Gambar Quadtree ]=================]");

            // 1. Input nama file (tanpa path)
            String filename = getInputFilename(scanner);
            File inputFile = new File(filename);
            long inputSize = inputFile.length();

            // Validasi file exist dan path
            if (!validateInputFile(inputFile)) {
                return;
            }

            // 2. Input parameter kompresi
            int method = getCompressionMethod(scanner);
            double threshold = getThreshold(scanner);
            double targetCompression = getTargetCompression(scanner);
            int minBlockSize = getMinBlockSize(scanner);

            // 3. Proses gambar
            BufferedImage image = null;
            try {
                image =
ImageUtils.readImage(inputFile.getAbsolutePath());
            } catch (IOException e) {
                System.err.println("[Error] Image path tidak
terdeteksi oleh sistem");
                System.err.println("Detail: " + e.getMessage());
                return;
            }

            // Buat nama output otomatis
            String outputDirPath = getOutputDirectory(scanner);
            String outputFilename = generateOutputFilename(inputFile,
outputDirPath);
            File outputFile = new File(outputFilename);

            // Validasi dan buat direktori output
            if (!validateAndCreateOutputDirectory(outputFile,
scanner)) {
                return;
            }

            // Proses kompresi dan generate GIF
            processCompressionAndGenerateGIF(image, outputFile,
inputSize, method, threshold, minBlockSize, scanner,
                    targetCompression);

        } catch (Exception e) {
            System.err.println("[Error] " + e.getMessage());
            e.printStackTrace();
        } finally {
            scanner.close();
        }
    }

    private static boolean validateInputFile(File inputFile) {
        if (!inputFile.exists()) {
```

```
            System.err.println("[Error] File tidak ditemukan di " +
inputFile.getAbsolutePath());
            return false;
        }

        if (!inputFile.isFile()) {
            System.err.println("[Error] Path yang diberikan bukan
merupakan file: " + inputFile.getAbsolutePath());
            return false;
        }

        if (!inputFile.canRead()) {
            System.err.println("[Error] File tidak dapat dibaca: " +
inputFile.getAbsolutePath());
            return false;
        }

        // Validasi ekstensi file
        String fileName = inputFile.getName().toLowerCase();
        if (!fileName.endsWith(".jpg") && !fileName.endsWith(".jpeg")
&& !fileName.endsWith(".png")) {
            System.err.println("[Error] Format file tidak didukung.
Gunakan format JPG, JPEG, atau PNG");
            return false;
        }

        return true;
    }

    private static void clearTerminal() {
        try {
            if (System.getProperty("os.name").contains("Windows")) {
                new ProcessBuilder("cmd", "/c",
"cls").inheritIO().start().waitFor();
            } else {
                new
ProcessBuilder("clear").inheritIO().start().waitFor();
            }
        } catch (IOException | InterruptedException e) {
            e.printStackTrace();
        }
    }

    private static String getInputFilename(Scanner scanner) {
        System.out.print("\n[input] Masukkan nama absolute path gambar
\n>> ");
        return scanner.nextLine();
    }

    private static int getCompressionMethod(Scanner scanner) {
        System.out.println("\n[input] Pilih metode error: ");
        System.out.println("\t1. Variance\n\t2. MAD\n\t3. Max Pixel
Difference\n\t4. Entropy\n\t5. SSIM");
        System.out.print(">> Pilihan (1-5): ");
        return scanner.nextInt();
    }

    private static double getThreshold(Scanner scanner) {
        System.out.print("\n[input] Masukkan Threshold error (contoh:
10.0) \n>> ");
        return scanner.nextDouble();
    }

    private static double getTargetCompression(Scanner scanner) {
        System.out.print("\n[input] Masukkan target persentase
kompresi (0.0 - 1.0, 0 untuk nonaktif) \n>> ");
        double target = scanner.nextDouble();
        scanner.nextLine(); // Consume the newline
        if (target < 0 || target > 1) {
            System.out.println("[Warning] Target kompresi tidak valid,
```

```
menggunakan 0 (nonaktif)");
            return 0.0;
        }
        return target;
    }

    private static int getMinBlockSize(Scanner scanner) {
        System.out.print("\n[input] Masukkan Ukuran blok minimum
(contoh: 4) \n>> ");
        int size = scanner.nextInt();
        scanner.nextLine(); // Consume the newline
        return size;
    }

    private static String getOutputDirectory(Scanner scanner) {
        System.out.print("\n[input] Masukkan alamat absolut output
gambar \n>> ");
        String input = scanner.nextLine().trim();
        while (input.isEmpty()) {
            System.out.print(">> ");
            input = scanner.nextLine().trim();
        }
        return input;
    }

    private static String generateOutputFilename(File inputFile,
String outputDirPath) {
        String inputFileName = inputFile.getName();
        String extension =
inputFileName.substring(inputFileName.lastIndexOf("."));
        String baseName = inputFileName.substring(0,
inputFileName.lastIndexOf("."));

        // Cari index yang tersedia
        int index = 1;
        String outputFileName;
        File outputFile;
        do {
            outputFileName = baseName + "_compressed_" + index +
extension;
            outputFile = new File(outputDirPath + File.separator +
outputFileName);
            index++;
        } while (outputFile.exists());

        return outputDirPath + File.separator + outputFileName;
    }

    private static boolean validateAndCreateOutputDirectory(File
outputFile, Scanner scanner) {
        File outputDir = outputFile.getParentFile();
        if (outputDir != null) {
            if (!outputDir.exists()) {
                System.out.println(
                        "Sepertinya alamat absolut untuk penyimpanan
proses gambar yang diberikan belum ada, maukah saya buatkan alamat
absolutnya?(y/n)");
                String response =
scanner.nextLine().trim().toLowerCase();
                while (!response.equals("y") && !response.equals("n"))
{
                    System.out.print("Masukkan y atau n: ");
                    response =
scanner.nextLine().trim().toLowerCase();
                }
                if (response.equals("y")) {
                    if (!outputDir.mkdirs()) {
                        System.err.println("[Error] Gagal membuat
direktori output: " + outputDir.getAbsolutePath());
                        return false;
```

```
                    }
                    System.out.println("Berhasil membuat tempat
penyimpanan");
                } else {
                    System.out.println("Gagal menyimpan gambar");
                    return false;
                }
            }

            if (!outputDir.canWrite()) {
                System.err.println(
                        "[Error] Tidak memiliki izin untuk menulis ke
direktori: " + outputDir.getAbsolutePath());
                return false;
            }
        } else {
            System.err.println("[Error] Path output tidak valid");
            return false;
        }
        return true;
    }

    private static void processCompressionAndGenerateGIF(BufferedImage
image, File outputFile, long inputSize,
            int method, double threshold, int minBlockSize, Scanner
scanner, double targetCompression)
            throws IOException {
        // Inisialisasi GIF Generator dengan frame rate lebih cepat
        System.out.println("\n[Status] Memulai proses kompresi...");
        System.out.flush();
        GIFGenerator gifGen = new GIFGenerator(50); // Frame rate
lebih cepat

        // Proses kompresi
        System.out.println("[Status] Membuat frame awal...");
        System.out.flush();
        Instant startTime = Instant.now();
        QuadTreeNode root = new QuadTreeNode(0, 0, image.getWidth(),
image.getHeight());

        // Buat frame awal (single block)
        BufferedImage initialFrame = new BufferedImage(
                image.getWidth(),
                image.getHeight(),
                BufferedImage.TYPE_INT_RGB);
        Graphics2D g2d = initialFrame.createGraphics();
        g2d.drawImage(image, 0, 0, null);
        g2d.dispose();
        gifGen.addFrame(initialFrame, 200); // Kurangi delay frame
awal

        System.out.println("[Status] Memulai proses kompresi
gambar...");
        System.out.flush();
        // Proses kompresi dengan visualisasi
        new ImageProcessor().compress(root, image, threshold,
minBlockSize, method,
                (node, img, depth, isLeaf) -> {
                    // Hanya capture frame untuk split yang signifikan
dan kedalaman tertentu
                    if (depth <= 2 || (depth <= 4 && depth % 2 == 0))
{
                        // Buat frame baru yang menunjukkan state
kompresi saat ini
                        BufferedImage frame = new BufferedImage(
                                img.getWidth(),
                                img.getHeight(),
                                BufferedImage.TYPE_INT_RGB);
                        Graphics2D g = frame.createGraphics();
```

```
                        // Gambar state kompresi saat ini
                        BufferedImage currentState =
ImageUtils.reconstructImage(root, img.getWidth(),
                                img.getHeight());
                        g.drawImage(currentState, 0, 0, null);

                        // Gambar batas quadtree
                        ImageUtils.drawQuadTreeState(g, root, 0,
depth);

                        g.dispose();
                        gifGen.addFrame(frame, 50); // Kurangi delay
antar frame
                        frame.flush();
                    }
                }, targetCompression);

        System.out.println("[Status] Membuat frame akhir...");
        System.out.flush();
        // Tambah frame akhir
        BufferedImage finalFrame = new BufferedImage(
                image.getWidth(),
                image.getHeight(),
                BufferedImage.TYPE_INT_RGB);
        Graphics2D finalG2d = finalFrame.createGraphics();

        // Gambar hasil kompresi akhir
        BufferedImage finalCompressed =
ImageUtils.reconstructImage(root, image.getWidth(),
image.getHeight());
        finalG2d.drawImage(finalCompressed, 0, 0, null);

        // Gambar batas quadtree akhir
        ImageUtils.drawQuadTreeState(finalG2d, root, 0,
root.totalDepth());

        finalG2d.dispose();
        gifGen.addFrame(finalFrame, 200); // Kurangi delay frame akhir

        Instant endTime = Instant.now();
        Duration duration = Duration.between(startTime, endTime);

        System.out.println("[Status] Menyimpan hasil kompresi...");
        System.out.flush();
        // Simpan hasil hanya sekali
        saveResults(outputFile, finalCompressed, inputSize,
outputFile.length(), root, duration, scanner, gifGen);
    }

    private static void saveResults(File outputFile, BufferedImage
finalCompressed, long inputSize, long outputSize,
            QuadTreeNode root, Duration duration, Scanner scanner,
GIFGenerator gifGen) throws IOException {
        String absoluteOutputPath = outputFile.getAbsolutePath();
        System.out.println("\n[Status] Mencoba menyimpan ke: " +
absoluteOutputPath);
        System.out.flush();

        // Pastikan direktori output ada dan bisa ditulis
        File outputDir = outputFile.getParentFile();
        if (!outputDir.canWrite()) {
            System.err.println(
                    "[Error] Tidak memiliki izin untuk menulis ke
direktori: " + outputDir.getAbsolutePath());
            return;
        }

        // Simpan gambar kompresi
        try {
            System.out.println("[Status] Menyimpan gambar hasil
```

```
kompresi...");
            System.out.flush();

            // Pastikan direktori output ada
            if (!outputDir.exists()) {
                if (!outputDir.mkdirs()) {
                    System.err.println("[Error] Gagal membuat
direktori output: " + outputDir.getAbsolutePath());
                    return;
                }
            }

            // Simpan file
            ImageUtils.saveImage(finalCompressed, absoluteOutputPath);

            // Verifikasi file tersimpan
            if (!outputFile.exists()) {
                System.err.println("[Error] File tidak berhasil
disimpan di " + absoluteOutputPath);
                return;
            }

            // Verifikasi ukuran file
            long actualFileSize = outputFile.length();
            if (actualFileSize == 0) {
                System.err.println("[Error] File berhasil dibuat tapi
kosong: " + absoluteOutputPath);
                return;
            }

            System.out.println("[Status] File berhasil disimpan");
            System.out.println("[Status] Ukuran file: " +
actualFileSize + " bytes");
            System.out.flush();

            // Tampilkan hasil hanya sekali
            displayResults(duration, inputSize, actualFileSize, root,
absoluteOutputPath);

            // Tanya user apakah ingin menyimpan sebagai GIF
            handleGIFGeneration(outputFile, scanner, gifGen);

        } catch (IOException e) {
            System.err.println("[Error] Gagal menyimpan file");
            System.err.println("Detail: " + e.getMessage());
            e.printStackTrace();
        }
    }

    private static void displayResults(Duration duration, long
inputSize, long outputSize, QuadTreeNode root,
            String absoluteOutputPath) {
        System.out.println("\n[output] Kompresi berhasil!\n");
        System.out.println("[output] Waktu eksekusi         : " +
duration.toMillis() + " ms");
        System.out.println("[output] Ukuran gambar sebelum  : " +
inputSize + " bytes");
        System.out.println("[output] Ukuran gambar setelah  : " +
outputSize + " bytes");
        System.out.printf("[output] Persentase kompresi    : %.2f
%%\n",
                (1 - ((double) outputSize / inputSize)) * 100);
        System.out.println("[output] Kedalaman pohon        : " +
root.totalDepth());
        System.out.println("[output] Banyak simpul pohon    : " +
root.totalNode());
        System.out.println("\n[output] Gambar hasil kompresi:");
        System.out.println("\t " + absoluteOutputPath);
        System.out.println("\n[output] File berhasil disimpan dan
dapat diakses di alamat di atas");
```

```
        System.out.flush();
    }

    private static void handleGIFGeneration(File outputFile, Scanner
scanner, GIFGenerator gifGen) throws IOException {
        System.out.print("\n[input] Maukah anda menyimpan proses
kompresi sebagai GIF? (y/n)\n>> ");
        System.out.flush();
        String saveGifResponse =
scanner.nextLine().trim().toLowerCase();
        while (!saveGifResponse.equals("y") &&
!saveGifResponse.equals("n")) {
            System.out.print("Masukkan y atau n: ");
            saveGifResponse = scanner.nextLine().trim().toLowerCase();
        }

        if (saveGifResponse.equals("y")) {
            try {
                // Gunakan index yang sama dengan file kompresi
                String baseName = outputFile.getName();
                String baseNameWithoutExt = baseName.substring(0,
baseName.lastIndexOf("."));
                String gifFileName =
baseNameWithoutExt.replace("_compressed_", "_gif_") + ".gif";
                String gifFilePath = outputFile.getParent() +
File.separator + gifFileName;

                // Verifikasi path
                File gifFile = new File(gifFilePath);
                File parentDir = gifFile.getParentFile();

                if (parentDir != null && !parentDir.exists()) {
                    if (!parentDir.mkdirs()) {
                        System.err.println("[Error] Failed to create
directory: " + parentDir.getAbsolutePath());
                        return;
                    }
                }

                System.out.println("\n[Status] Memulai proses
pembuatan GIF...");
                System.out.flush();
                // Simpan GIF
                gifGen.saveGIF(gifFile.getAbsolutePath());
                System.out.println("\n[output] GIF successfully saved
to:");
                System.out.println("\t" + gifFile.getAbsolutePath());
                System.out.flush();

            } catch (IOException e) {
                System.err.println("[Error] Gagal menyimpan GIF");
                System.err.println("Detail: " + e.getMessage());
            }
        }


System.out.println("\n[=============================================
======================]\n");
        System.out.flush();
    }
}
```

iv. QuadTreeNode.java

```
package src.app;

import java.util.ArrayList;
```

```
public class QuadTreeNode {
    // Region properties
    public final int x, y; // Position in image (top-left corner)
    public final int width, height; // Dimensions of the region
    public int r, g, b; // Average color (if leaf node)
    public QuadTreeNode[] children; // Child nodes (NW, NE, SW, SE)
    public boolean processed; // Flag for visualization

    public QuadTreeNode(int x, int y, int width, int height) {
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
        this.children = null; // Initialize as null
        this.processed = false;
    }

    /**
     * Checks if this node is a leaf node (has no children)
     */
    public boolean isLeaf() {
        return children == null;
    }

    /**
     * Splits the current node into 4 child nodes
     */
    public void split() {
        if (!isLeaf())
            return;

        int halfWidth = (int) Math.ceil(width / 2.0);
        int halfHeight = (int) Math.ceil(height / 2.0);

        children = new QuadTreeNode[4];
        children[0] = new QuadTreeNode(x, y, halfWidth, halfHeight);
// NW
        children[1] = new QuadTreeNode(x + halfWidth, y, width -
halfWidth, halfHeight); // NE
        children[2] = new QuadTreeNode(x, y + halfHeight, halfWidth,
height - halfHeight); // SW
        children[3] = new QuadTreeNode(x + halfWidth, y + halfHeight,
                width - halfWidth, height - halfHeight); // SE
    }

    /**
     * Counts total number of nodes in the subtree
     */
    public int totalNode() {
        if (isLeaf()) {
            return 1;
        }

        int count = 1; // Count this node
        for (QuadTreeNode child : children) {
            if (child != null) {
                count += child.totalNode();
            }
        }
        return count;
    }

    /**
     * Calculates maximum depth of the tree
     */
    public int totalDepth() {
        if (isLeaf()) {
            return 1;
        }
```

```
        int maxDepth = 0;
        for (QuadTreeNode child : children) {
            if (child != null) {
                maxDepth = Math.max(maxDepth, child.totalDepth());
            }
        }
        return 1 + maxDepth;
    }

    /**
     * Gets all leaf nodes in the subtree
     */
    public ArrayList<QuadTreeNode> getLeafNodes() {
        ArrayList<QuadTreeNode> leaves = new ArrayList<>();
        collectLeaves(this, leaves);
        return leaves;
    }

    private void collectLeaves(QuadTreeNode node,
ArrayList<QuadTreeNode> leaves) {
        if (node.isLeaf()) {
            leaves.add(node);
        } else {
            for (QuadTreeNode child : node.children) {
                if (child != null) {
                    collectLeaves(child, leaves);
                }
            }
        }
    }

    /**
     * Gets the bounding box coordinates [x1, y1, x2, y2]
     */
    public int[] getBounds() {
        return new int[] { x, y, x + width, y + height };
    }

    /**
     * Checks if this node contains the given point
     */
    public boolean containsPoint(int px, int py) {
        return px >= x && px < x + width && py >= y && py < y +
height;
    }

    /**
     * Marks this node and all its children as processed
     * For visualization purposes
     */
    public void markAsProcessed() {
        this.processed = true;
        if (!isLeaf()) {
            for (QuadTreeNode child : children) {
                if (child != null) {
                    child.markAsProcessed();
                }
            }
        }
    }

    /**
     * Gets the center point of the node
     */
    public int[] getCenter() {
        return new int[] { x + width / 2, y + height / 2 };
    }
}
```

v. GIFGenerator.java

```
package src.app.utils;

import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import javax.imageio.ImageIO;
import javax.imageio.stream.FileImageOutputStream;
import javax.imageio.stream.ImageOutputStream;

public class GIFGenerator implements AutoCloseable {
    private File tempDir;
    private List<String> tempFiles;
    private List<Integer> delays;
    private int defaultDelayMs;
    private int frameCount;

    public GIFGenerator(int defaultDelayMs) {
        if (defaultDelayMs <= 0) {
            throw new IllegalArgumentException("Delay must be
positive");
        }
        this.defaultDelayMs = defaultDelayMs;
        this.tempFiles = new ArrayList<>();
        this.delays = new ArrayList<>();
        this.frameCount = 0;

        // Create temporary directory
        try {
            this.tempDir = new
File(System.getProperty("java.io.tmpdir"), "quadtree_gif_" +
System.currentTimeMillis());
            if (!this.tempDir.mkdirs()) {
                throw new IOException("Failed to create temporary
directory");
            }
        } catch (IOException e) {
            throw new RuntimeException("Failed to initialize GIF
generator", e);
        }
    }

    public void addFrame(BufferedImage image) {
        addFrame(image, defaultDelayMs);
    }

    public void addFrame(BufferedImage image, int customDelayMs) {
        if (image == null) {
            throw new IllegalArgumentException("Image cannot be
null");
        }

        try {
            // Save frame to temporary file
            String tempFileName = String.format("frame_%05d.png",
frameCount++);
            File tempFile = new File(tempDir, tempFileName);
            ImageIO.write(image, "png", tempFile);
            tempFiles.add(tempFile.getAbsolutePath());
            delays.add(Math.max(1, customDelayMs / 10)); // Minimum
1/100th second
        } catch (IOException e) {
            throw new RuntimeException("Failed to save frame", e);
        }
    }

    public void saveGIF(String outputPath) throws IOException {
```

```
        if (tempFiles.isEmpty()) {
            throw new IllegalStateException("No frames to save");
        }

        File outputFile = new File(outputPath);
        ImageOutputStream output = null;
        GifSequenceWriter writer = null;

        try {
            // Create parent directories if needed
            File parentDir = outputFile.getParentFile();
            if (parentDir != null && !parentDir.exists()) {
                if (!parentDir.mkdirs()) {
                    throw new IOException("Failed to create directory:
" + parentDir);
                }
            }

            // Read first frame to get image type
            BufferedImage firstFrame = ImageIO.read(new
File(tempFiles.get(0)));

            output = new FileImageOutputStream(outputFile);
            writer = new GifSequenceWriter(output,
firstFrame.getType(), delays.get(0), true);

            // Write first frame
            writer.writeToSequence(firstFrame);

            // Write remaining frames
            for (int i = 1; i < tempFiles.size(); i++) {
                BufferedImage frame = ImageIO.read(new
File(tempFiles.get(i)));
                writer.setDelay(delays.get(i));
                writer.writeToSequence(frame);
            }
        } finally {
            // Close resources
            if (writer != null)
                writer.close();
            if (output != null)
                output.close();
        }
    }

    @Override
    public void close() {
        cleanupTempFiles();
    }

    private void cleanupTempFiles() {
        for (String tempFile : tempFiles) {
            new File(tempFile).delete();
        }
        tempDir.delete();
    }
}
```

vi. GifSequenceWriter.java

```
package src.app.utils;

import javax.imageio.*;
import javax.imageio.metadata.*;
import javax.imageio.stream.*;
import java.awt.image.*;
import java.io.*;
import java.util.Iterator;
```

```
public class GifSequenceWriter {
    private ImageWriter gifWriter;
    private ImageWriteParam imageWriteParam;
    private IIOMetadata imageMetaData;
    private ImageOutputStream outputStream;
    private int delay;
    private boolean sequenceStarted = false;

    public GifSequenceWriter(
            ImageOutputStream outputStream,
            int imageType,
            int delay,
            boolean loop) throws IOException {

        if (outputStream == null) {
            throw new IllegalArgumentException("Output stream cannot
be null");
        }

        this.outputStream = outputStream;
        this.delay = delay;

        // Get GIF writer
        Iterator<ImageWriter> iter =
ImageIO.getImageWritersBySuffix("gif");
        if (!iter.hasNext()) {
            throw new IIOException("No GIF Image Writers Available");
        }

        this.gifWriter = iter.next();
        this.imageWriteParam = gifWriter.getDefaultWriteParam();

        // Configure image metadata
        ImageTypeSpecifier imageTypeSpecifier =
ImageTypeSpecifier.createFromBufferedImageType(imageType);
        this.imageMetaData = gifWriter.getDefaultImageMetadata(
                imageTypeSpecifier, imageWriteParam);

        configureRootMetadata(delay, loop);

        // Initialize sequence
        this.gifWriter.setOutput(outputStream);
        this.gifWriter.prepareWriteSequence(null);
        this.sequenceStarted = true;
    }

    public void setDelay(int delay) {
        this.delay = delay;
        configureRootMetadata(delay, true);
    }

    private void configureRootMetadata(int delay, boolean loop) {
        try {
            String metaFormatName =
imageMetaData.getNativeMetadataFormatName();
            IIOMetadataNode root = (IIOMetadataNode)
imageMetaData.getAsTree(metaFormatName);

            // Set delay time
            IIOMetadataNode gceNode = getNode(root,
"GraphicControlExtension");
            gceNode.setAttribute("delayTime",
Integer.toString(delay));
            gceNode.setAttribute("disposalMethod", "none");
            gceNode.setAttribute("userInputFlag", "FALSE");

            // Set looping
            if (loop) {
                IIOMetadataNode appExtensionsNode = getNode(root,
```

```
"ApplicationExtensions");
                IIOMetadataNode child = new
IIOMetadataNode("ApplicationExtension");
                child.setAttribute("applicationID", "NETSCAPE");
                child.setAttribute("authenticationCode", "2.0");
                child.setUserObject(new byte[] { 0x1, (byte) 0, (byte)
0 });
                appExtensionsNode.appendChild(child);
            }

            imageMetaData.setFromTree(metaFormatName, root);
        } catch (IIOInvalidTreeException e) {
            throw new RuntimeException("Failed to configure GIF
metadata", e);
        }
    }

    public void writeToSequence(BufferedImage img) throws IOException
{
        if (!sequenceStarted) {
            throw new IllegalStateException("Sequence not started");
        }
        if (gifWriter == null) {
            throw new IllegalStateException("Writer has been closed");
        }
        gifWriter.writeToSequence(new IIOImage(img, null,
imageMetaData), imageWriteParam);
    }

    public void close() throws IOException {
        if (sequenceStarted && gifWriter != null) {
            try {
                gifWriter.endWriteSequence();
            } finally {
                gifWriter.dispose();
                gifWriter = null;
                sequenceStarted = false;
            }
        }
    }

    private static IIOMetadataNode getNode(IIOMetadataNode rootNode,
String nodeName) {
        for (int i = 0; i < rootNode.getLength(); i++) {
            if
(rootNode.item(i).getNodeName().equalsIgnoreCase(nodeName)) {
                return (IIOMetadataNode) rootNode.item(i);
            }
        }
        IIOMetadataNode node = new IIOMetadataNode(nodeName);
        rootNode.appendChild(node);
        return node;
    }
}
```

vii. ImageUtils.java

```
package src.app.utils;

import java.awt.*;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import javax.imageio.ImageIO;
import src.app.QuadTreeNode;

public class ImageUtils {
    public static BufferedImage readImage(String filePath) throws
```

```
IOException {
        File file = new File(filePath);
        if (!file.exists()) {
            throw new IOException("File tidak ditemukan: " +
filePath);
        }
        BufferedImage image = ImageIO.read(file);
        if (image == null) {
            throw new IOException("Format gambar tidak didukung: " +
filePath);
        }
        return image;
    }

    public static void saveImage(BufferedImage image, String
outputPath) throws IOException {
        String format =
outputPath.substring(outputPath.lastIndexOf(".") + 1).toLowerCase();
        if (!ImageIO.write(image, format, new File(outputPath))) {
            throw new IOException("Tidak ditemukan writer untuk
format: " + format);
        }
    }

    public static BufferedImage reconstructImage(QuadTreeNode root,
int width, int height) {
        BufferedImage outputImage = new BufferedImage(width, height,
BufferedImage.TYPE_INT_RGB);
        if (root != null) {
            renderQuadTree(outputImage, root);
        }
        return outputImage;
    }

    private static void renderQuadTree(BufferedImage image,
QuadTreeNode node) {
        if (node == null)
            return;

        if (node.isLeaf()) {
            fillNode(image, node);
        } else {
            for (QuadTreeNode child : node.children) {
                renderQuadTree(image, child);
            }
        }
    }

    private static void fillNode(BufferedImage image, QuadTreeNode
node) {
        int rgb = (node.r << 16) | (node.g << 8) | node.b;
        int endY = Math.min(node.y + node.height, image.getHeight());
        int endX = Math.min(node.x + node.width, image.getWidth());

        for (int y = node.y; y < endY; y++) {
            for (int x = node.x; x < endX; x++) {
                image.setRGB(x, y, rgb);
            }
        }
    }

    // Method untuk visualisasi proses Quadtree
    public static BufferedImage drawQuadTree(BufferedImage original,
QuadTreeNode node) {
        BufferedImage frame = new BufferedImage(
                original.getWidth(),
                original.getHeight(),
                BufferedImage.TYPE_INT_RGB);

        Graphics2D g = frame.createGraphics();
```

```
        // Gambar gambar asli sebagai background
        g.drawImage(original, 0, 0, null);

        // Gambar garis pembatas quadtree
        drawTreeBoundaries(g, node, Color.RED, 1);

        g.dispose();
        return frame;
    }

    public static BufferedImage highlightNode(BufferedImage original,
QuadTreeNode node, Color highlightColor) {
        BufferedImage frame = drawQuadTree(original, node);
        Graphics2D g = frame.createGraphics();

        // Highlight node yang sedang diproses
        g.setColor(new Color(
                highlightColor.getRed(),
                highlightColor.getGreen(),
                highlightColor.getBlue(),
                100)); // Transparansi 100/255

        g.fillRect(node.x, node.y, node.width, node.height);

        // Gambar border tebal
        g.setColor(highlightColor.darker());
        g.setStroke(new BasicStroke(2));
        g.drawRect(node.x, node.y, node.width - 1, node.height - 1);

        g.dispose();
        return frame;
    }

    private static void drawTreeBoundaries(Graphics2D g, QuadTreeNode
node, Color color, int depth) {
        if (node == null)
            return;

        // Variasikan warna berdasarkan kedalaman
        Color depthColor = new Color(
                Math.min(255, color.getRed() + depth * 20),
                Math.min(255, color.getGreen() + depth * 10),
                Math.min(255, color.getBlue() + depth * 5));

        g.setColor(depthColor);
        g.setStroke(new BasicStroke(1));

        if (node.isLeaf()) {
            g.drawRect(node.x, node.y, node.width - 1, node.height -
1);
        } else {
            g.drawRect(node.x, node.y, node.width - 1, node.height -
1);
            for (QuadTreeNode child : node.children) {
                drawTreeBoundaries(g, child, color, depth + 1);
            }
        }
    }

    // Method untuk menggambar state quadtree
    public static void drawQuadTreeState(Graphics2D g, QuadTreeNode
node, int currentDepth, int maxDepth) {
        if (node == null || currentDepth > maxDepth)
            return;

        // Set warna berdasarkan kedalaman (memastikan nilai tetap
dalam range 0-255)
        int red = Math.max(0, Math.min(255, 255 - currentDepth * 30));
        int green = Math.max(0, Math.min(255, 100 + currentDepth *
```

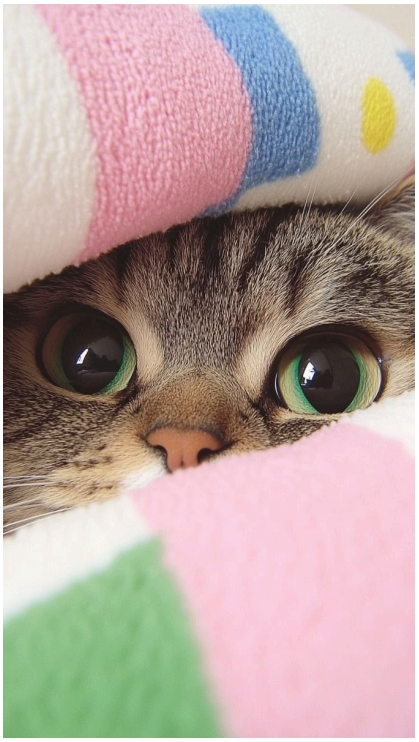
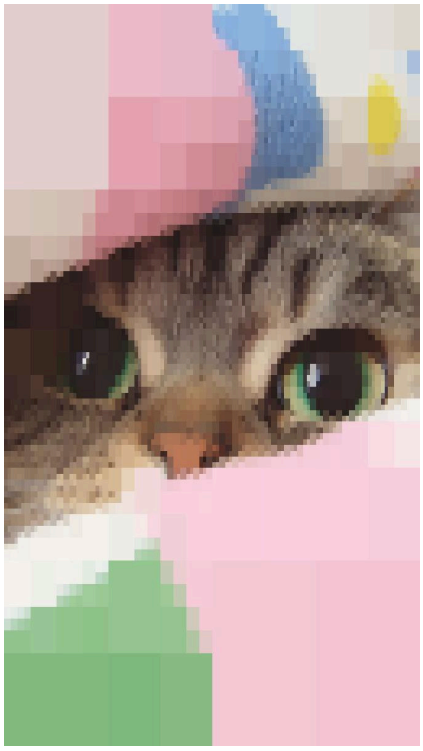
```
20));
        int blue = Math.max(0, Math.min(255, 100 + currentDepth *
20));

        Color color = new Color(red, green, blue);

        // Gambar batas node
        g.setColor(color);
        g.setStroke(new BasicStroke(1));
        g.drawRect(node.x, node.y, node.width - 1, node.height - 1);

        // Gambar anak secara rekursif
        if (!node.isLeaf()) {
            for (QuadTreeNode child : node.children) {
                drawQuadTreeState(g, child, currentDepth + 1,
maxDepth);
            }
        }
    }

    // Utility untuk menggambar teks di gambar
    public static BufferedImage addDebugText(BufferedImage image,
String text) {
        BufferedImage newImage = new BufferedImage(
                image.getWidth(),
                image.getHeight(),
                BufferedImage.TYPE_INT_RGB);

        Graphics2D g = newImage.createGraphics();
        g.drawImage(image, 0, 0, null);

        g.setColor(Color.WHITE);
        g.setFont(new Font("Arial", Font.BOLD, 14));
        g.drawString(text, 10, 20);

        g.dispose();
        return newImage;
    }
}
```

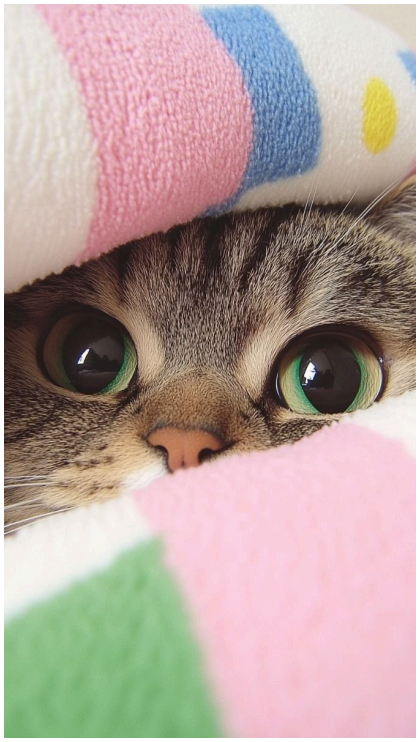
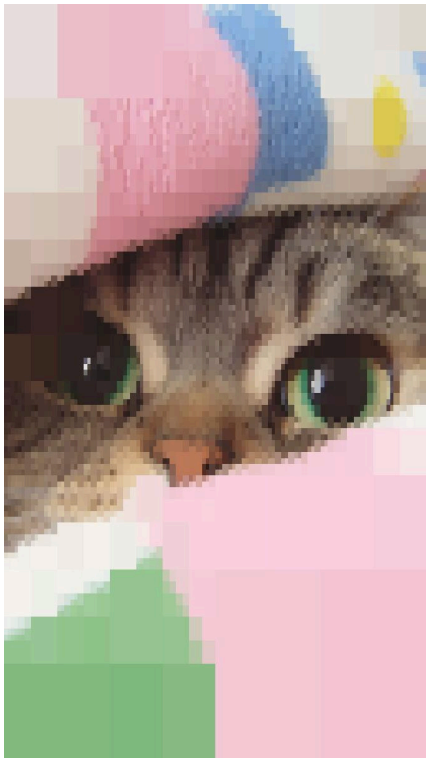
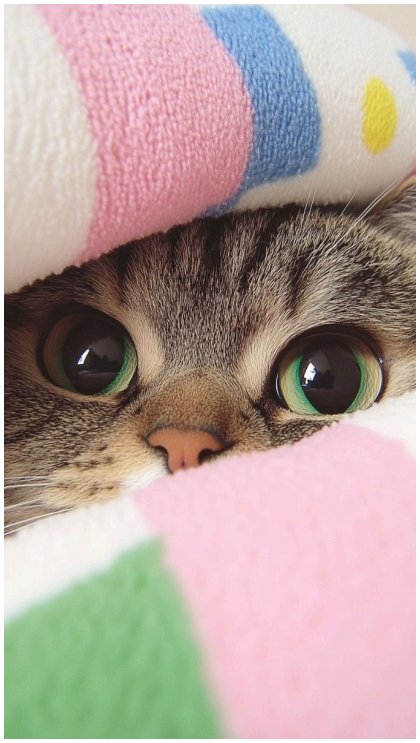
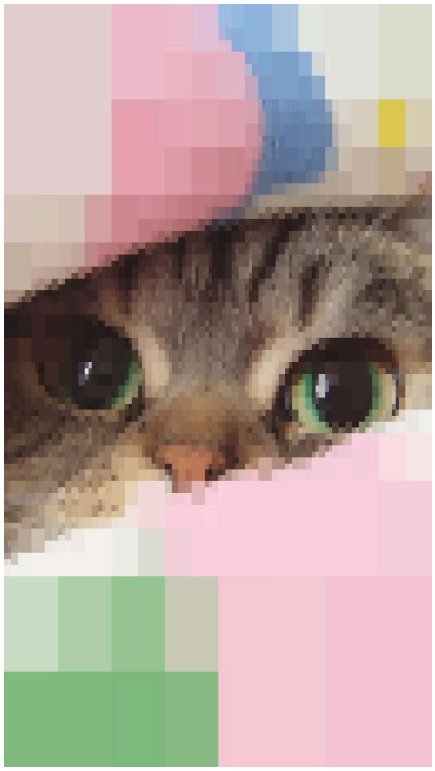
D. Pengujian

Terdapat 7 kasus uji untuk memastikan program dapat berjalan dengan baik. Kasus uji dan hasil pengujian tertera pada tabel berikut:

Tabel 1. Pengujian Program

No	Indikator	Input	Output
1	Galat *variance*	[input] Pilih metode error: 1. Variance 2. MAD 3. Max Pixel Difference 4. Entropy 5. SSIM >> Pilihan (1-5): 1 [input] Masukkan Threshold error (contoh: 10.0) >> 300 [input] Masukkan Ukuran blok minimum (contoh: 4) >> 10	[output] Kompresi berhasil! [output] Waktu eksekusi : 7142 ms [output] Ukuran gambar sebelum : 119541 [output] Ukuran gambar setelah : 50238 [output] Persentase kompresi : 57.97 % [output] Kedalaman pohon : 8 [output] Banyak simpul pohon : 4393

			
2	Galat *MAD*	[input] Pilih metode error: 1. Variance 2. MAD 3. Max Pixel Difference 4. Entropy 5. SSIM >> Pilihan (1-5): 2 [input] Masukkan Threshold error (contoh: 10.0) >> 10 [input] Masukkan Ukuran blok minimum (contoh: 4) >> 10	[output] Kompresi berhasil! [output] Waktu eksekusi : 8094 m [output] Ukuran gambar sebelum : 119541 [output] Ukuran gambar setelah : 53192 [output] Persentase kompresi : 55.50 % [output] Kedalaman pohon : 8 [output] Banyak simpul pohon : 5413

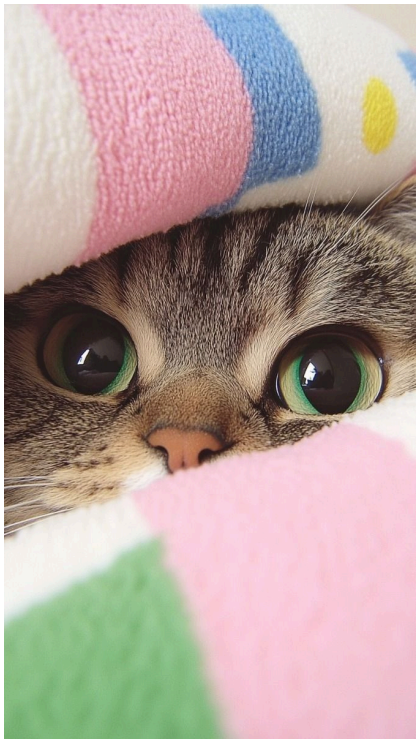
			
3	Galat *MPD*		

<table>
<tr>
<td></td>
<td></td>
<td>

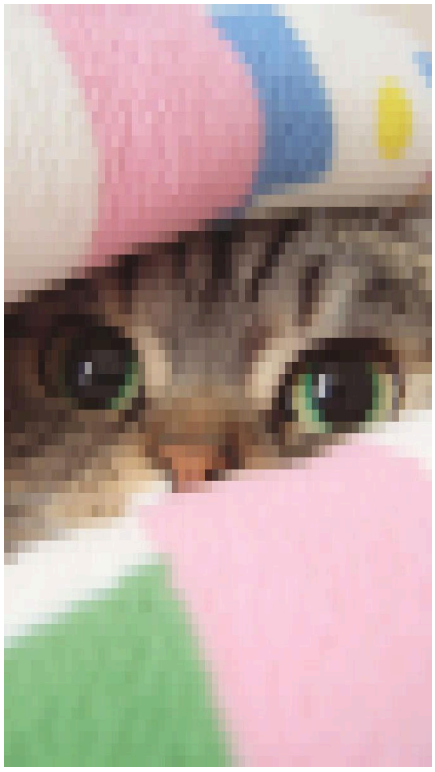
```
[input] Pilih metode error:
        1. Variance
        2. MAD
        3. Max Pixel Difference
        4. Entropy
        5. SSIM
>> Pilihan (1-5): 3

[input] Masukkan Threshold error (contoh: 10.0)
>> 20

[input] Masukkan Ukuran blok minimum (contoh: 4)
>> 10
```

</td>
<td>

```
[output] Kompresi berhasil!

[output] Waktu eksekusi          : 7056 ms
[output] Ukuran gambar sebelum   : 119541
[output] Ukuran gambar setelah   : 49275
[output] Persentase kompresi     : 58.78 %
[output] Kedalaman pohon         : 8
[output] Banyak simpul pohon     : 4273
```

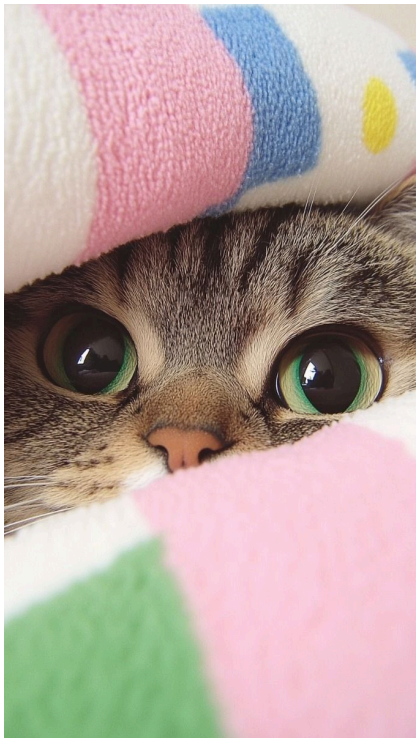
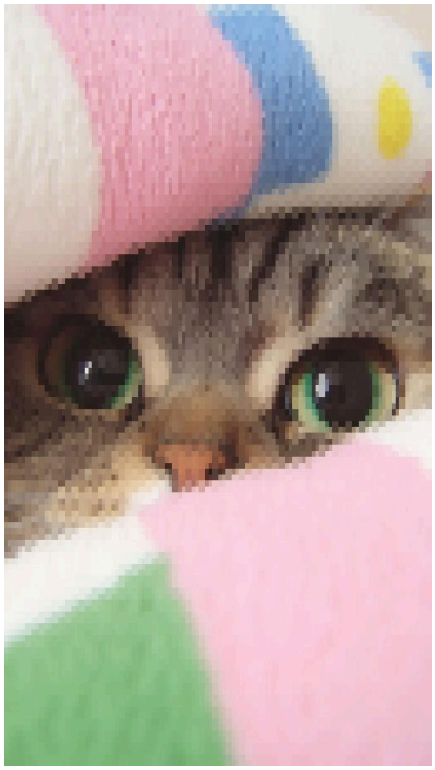
</td>
</tr>
<tr>
<td>4</td>
<td>Galat entropy</td>
<td>


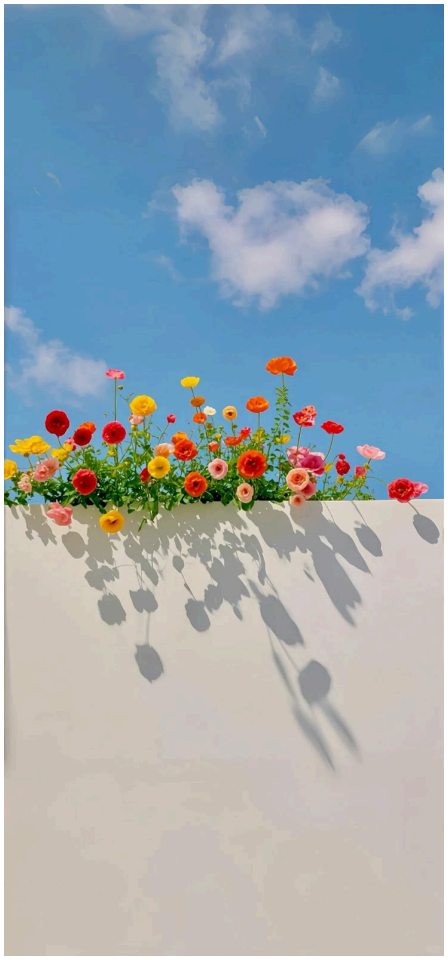
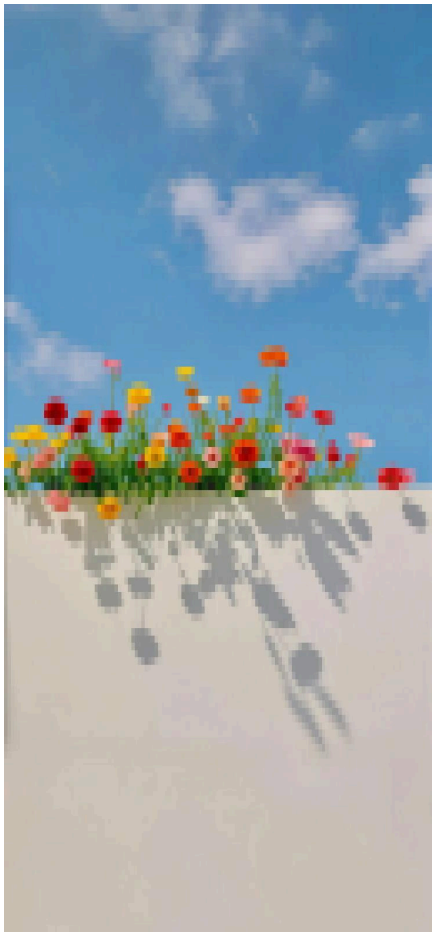
```
[input] Pilih metode error:
        1. Variance
        2. MAD
        3. Max Pixel Difference
        4. Entropy
        5. SSIM
>> Pilihan (1-5): 4

[input] Masukkan Threshold error (contoh: 10.0)
>> 6

[input] Masukkan Ukuran blok minimum (contoh: 4)
>> 10
```

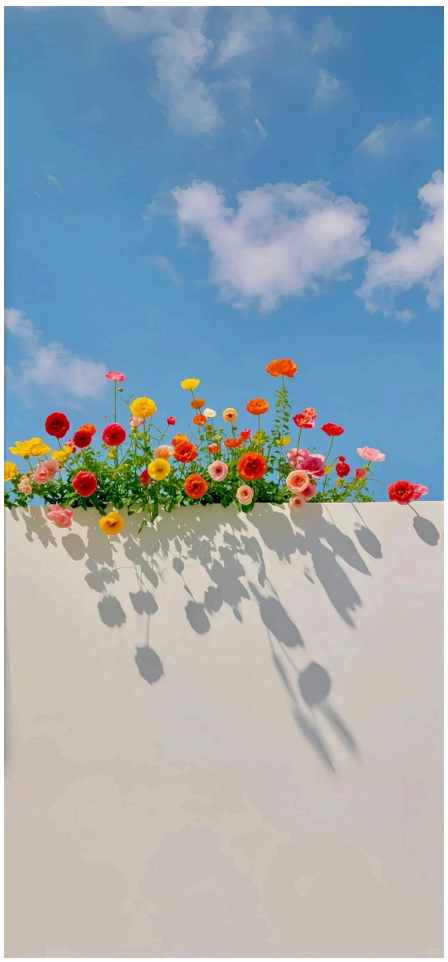
</td>
<td>


```
[output] Kompresi berhasil!

[output] Waktu eksekusi          : 14427 ms
[output] Ukuran gambar sebelum   : 119541
[output] Ukuran gambar setelah   : 50409
[output] Persentase kompresi     : 57.83 %
[output] Kedalaman pohon         : 7
[output] Banyak simpul pohon     : 5461
```

</td>
</tr>
</table>

5	Galat *SSIM*	 [input] Pilih metode error: 1. Variance 2. MAD 3. Max Pixel Difference 4. Entropy 5. SSIM >> Pilihan (1-5): 5 [input] Masukkan Threshold error (contoh: 10.0) >> 0.2 [input] Masukkan target persentase kompresi (0.0 - 1 >> 0 [input] Masukkan Ukuran blok minimum (contoh: 4) >> 10	 [output] Kompresi berhasil! [output] Waktu eksekusi : 9997 ms [output] Ukuran gambar sebelum : 119541 bytes [output] Ukuran gambar setelah : 60875 bytes [output] Persentase kompresi : 49.08 % [output] Kedalaman pohon : 8 [output] Banyak simpul pohon : 13909

6	Target kompresi	 [input] Pilih metode error: 1. Variance 2. MAD 3. Max Pixel Difference 4. Entropy 5. SSIM >> Pilihan (1-5): 5 [input] Masukkan Threshold error (contoh: 10.0) >> 0.2 [input] Masukkan Ukuran blok minimum (contoh: 4) >> 10 [input] Masukkan target persentase kompresi (0.0 - 1.0, 0 untuk nonaktif) >> 0.1	 [output] Kompresi berhasil! [output] Waktu eksekusi : 13733 ms [output] Ukuran gambar sebelum : 82504 bytes [output] Ukuran gambar setelah : 63314 bytes [output] Persentase kompresi : 23.26 % [output] Kedalaman pohon : 8 [output] Banyak simpul pohon : 21033

7	Pengujian GIF	 [input] Pilih metode error: 1. Variance 2. MAD 3. Max Pixel Difference 4. Entropy 5. SSIM >> Pilihan (1-5): 3 [input] Masukkan Threshold error (contoh: 10.0) >> 30 [input] Masukkan Ukuran blok minimum (contoh: 4) >> 10	test1_gif_1.gif [output] Kompresi berhasil! [output] Waktu eksekusi : 3323 ms [output] Ukuran gambar sebelum : 82504 [output] Ukuran gambar setelah : 43214 [output] Persentase kompresi : 47.62 % [output] Kedalaman pohon : 8 [output] Banyak simpul pohon : 2101

E. Analisis Hasil Pengujian

Pengujian memberikan hasil bahwa program dapat berjalan dengan lancar tanpa mengalami gangguan. Hasil menunjukkan bahwa perhitungan galat dengan metode *entropy* dan *SSIM* dapat memberikan hasil kompresi yang lebih baik, mencakup ukuran akhir yang lebih kecil, serta hasil gambar yang lebih baik dibandingkan metode lainnya. Meskipun memberikan hasil yang memuaskan, bahwa program berjalan dengan baik, implementasi GIF masih kurang sempurna, dan perlu ditingkatkan kembali.

F. Penjelasan Implementasi Bonus

a. Galat *SSIM*

Untuk perhitungan galat SSIM dimulai dengan melakukan pemecahan gambar menjadi blok-blok kecil dalam konteks Quadtree yakni dilakukan dengan membagi gambar menjadi sebuah kuadran yang lebih kecil hingga mencapai minimum atau threshold yang sudah ditentukan oleh pengguna. Lalu, dilakukan perhitungan statistik untuk setiap bloknya dengan menghitung rata-rata, variansi, dan kovarians antara dua blok gambar yang dibandingkan (proses ini dilakukan dalam fungsi yang menghitung SSIM untuk setiap bloknya yakni pada `calculatePixelSSIM`). Perhitungan statistik tersebut kemudian dipakai dalam formula SSIM pada tiap bloknya dan hasilnya digabungkan untuk mendapatkan nilai SSIM dari keseluruhan gambar. Dengan ini Algoritma Divide and Conquer menjadi pilihan algoritma untuk melakukan pemisahan gambar lalu menghitung SSIM tiap kuadran secara independen dan menggabungkan hasilnya untuk menghasilkan gambar secara keseluruhan.

b. Target Persentase Kompresi

Persentase kompresi dilakukan dengan melakukan pengambilan data hasil pengukuran (size) gambar asli sebelum terkompresi dan sesudah terkompresi. Lalu dari data yang diambil tersebut diambil hasil perhitungan persentase kompresi dari hasil perhitungan dari formula yang diberikan. Dengan adanya target persentase kompresi ini didapatkan hasil dari jumlah persentase hilangnya ukuran gambar yang asli dengan yang terkompresi. Sehingga, parameter ini dapat digunakan untuk memenuhi keinginan pengguna dalam memodifikasi hasil ukuran gambar yang diinginkan seusai kompresi.

c. Output berupa GIF

GIF dibuat dengan cara menyediakan frame untuk menampilkan proses pembentukan gambar kompresi dengan memperlihatkan proses Quadtree yang berjalan. Implementasi output berupa GIF ini dibuat dengan dasar untuk dapat memberikan visualisasi atas algoritma Divide and Conquer pada pembentukan proses kompresi gambar. Langkahnya yakni pada proses awal frame akan diinisialisasikan terlebih dahulu pada `GIFGenerator`, lalu ketika proses Quadtree berjalan `GIFGenerator` akan membuat frame baru dan menambahkannya lalu akan menyimpan proses tersebut dalam file .gif.

G. Lampiran

a. Tabel Pemenuhan Spesifikasi

No	Poin	Ya	Tidak
1	Program berhasil dikompilasi tanpa kesalahan.	✅	
2	Program berhasil dijalankan.	✅	
3	Program berhasil melakukan kompresi gambar sesuai parameter yang ditentukan.	✅	
4	Mengimplementasi seluruh metode perhitungan error wajib.	✅	
5	**[Bonus]** Implementasi persentase kompresi sebagai parameter tambahan.	✅	
6	**[Bonus]** Implementasi Structural Similarity Index (SSIM) sebagai metode pengukuran error.	✅	
7	**[Bonus]** Output berupa GIF Visualisasi Proses pembentukan Quadtree dalam Kompresi Gambar.	✅	
8	Program dan laporan dibuat (kelompok) sendiri.	✅	