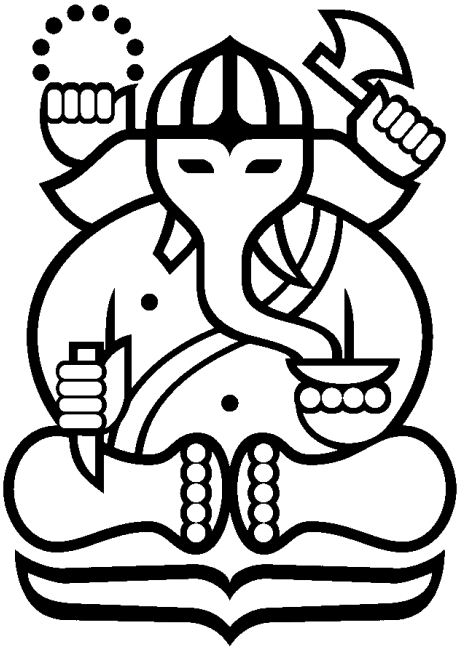
Laporan Tugas Kecil 1 IF2211 Strategi Algoritma

Penyelesaian IQ Puzzler Pro dengan Algoritma Brute Force



Disusun oleh:

Ranashahira Reztaputri (13523007)

K-01

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

2025

DAFTAR ISI

BAB I
Deskripsi Masalah dan Algoritma

1.1. Algoritma Brute Force

Algoritma brute force merupakan pendekatan yang sederhana dan lugas untuk memecahkan masalah dengan cara mencoba setiap solusi yang mungkin hingga menemukan solusi terbaik. Algoritma ini tidak menggunakan trik atau jalan pintas yang cerdas untuk mengurangi ruang pencarian atau meningkatkan efisiensi. Algoritma brute force menjamin solusi pasti benar karena menelusuri seluruh kemungkinan. Akibatnya, secara umum, brute force bekerja dengan lambat, terutama ketika terdapat banyak kemungkinan solusi yang perlu dicoba. Algoritma ini tentunya memiliki kelebihan dan kekurangan. Kelebihan dari brute force adalah pendekatannya yang sederhana sehingga membuat ide penyelesaiannya lebih mudah dipikirkan dan diimplementasikan. Di sisi lain, kekurangan dari brute force adalah waktu penyelesaiannya yang cenderung lambat dan tidak efisien.

1.2. Deskripsi Tugas

Gambar 1 Permainan IQ Puzzler Pro
(Sumber: https://www.smartgamesusa.com)

IQ Puzzler Pro adalah permainan papan yang diproduksi oleh perusahaan Smart Games. Tujuan dari permainan ini adalah pemain harus dapat mengisi seluruh papan dengan piece (blok puzzle) yang telah tersedia.

Komponen penting dari permainan IQ Puzzler Pro terdiri dari:

1. **Board (Papan)** – Board merupakan komponen utama yang menjadi tujuan permainan dimana pemain harus mampu mengisi seluruh area papan menggunakan blok-blok yang telah disediakan.

2. **Blok/Piece** – Blok adalah komponen yang digunakan pemain untuk mengisi papan kosong hingga terisi penuh. Setiap blok memiliki bentuk yang unik dan semua blok harus digunakan untuk menyelesaikan puzzle.

Permainan dimulai dengan **papan yang kosong**. Pemain dapat meletakkan blok puzzle sedemikian sehingga **tidak ada blok yang bertumpang tindih** (kecuali dalam kasus 3D). Setiap blok puzzle dapat **dirotasikan** maupun **dicerminkan**. Puzzle dinyatakan **selesai** jika dan hanya jika papan **terisi penuh** dan **seluruh blok puzzle berhasil diletakkan**.

Tugas anda adalah menemukan **cukup satu solusi** dari permainan **IQ Puzzler Pro** dengan menggunakan ***algoritma Brute Force***, atau menampilkan bahwa solusi tidak ditemukan jika tidak ada solusi yang mungkin dari puzzle.

Ilustrasi kasus

Diberikan sebuah wadah berukuran 11 x 5 serta 12 buah blok puzzle dan beberapa blok telah ditempatkan dengan bentuk sebagai berikut.

Gambar 2 Awal Permainan Game IQ Puzzler Pro
(Sumber: https://www.smartgamesusa.com)

Pemain berusaha untuk mengisi bagian papan yang kosong dengan menggunakan blok yang tersedia

PERHATIAN: Untuk tucil ini permainan diawali dengan papan kosong

Gambar 3 Pemain Mencoba Semua Kemungkinan
(Sumber: https://www.smartgamesusa.com)

Puzzle berikut dinyatakan telah selesai karena papan sudah terisi penuh dan seluruh blok telah digunakan.

Gambar 4 Pemain Menyelesaikan Permainan
(Sumber: https://www.smartgamesusa.com)

Agar lebih jelas, amati video cara bermain berikut:
https://youtube.com/shorts/MWiPAS3wfGM?feature=shared

1.3. Penyelesaian IQ Puzzler Pro dengan Algoritma Brute Force

Pada tugas kecil ini, penyelesaian IQ Puzzler Pro dilakukan dengan menggunakan algoritma brute force (exhaustive search), dengan langkah-langkah sebagai berikut:

1. Blok puzzle direpresentasikan dalam bentuk koordinat dan setiap bentuk (dalam hal ini dilambangkan dengan huruf alfabet A-Z) dimasukkan ke dalam list blok. Dalam implementasi ini, penulis menggunakan ArrayList untuk penyimpanan lokasi blok dan HashMap untuk menyimpan jenis blok dan *state*-nya (apakah sudah diletakkan atau belum).
2. Peletakkan blok dilakukan secara rekursif dengan dua basis, yaitu papan penuh, yang berarti solusi ditemukan, dan blok habis, yang berarti solusi tidak ditemukan. Kondisi di mana papan penuh tetapi blok belum habis sudah ditangani dengan adanya validasi setelah memasukkan file input yang mengecek apakah luas papan sama dengan total luas blok.
3. Peletakkan dilakukan dimulai dari blok pertama sampai blok terakhir. Semua kemungkinan peletakan blok dicoba satu persatu, yaitu kondisi blok dicerminkan maupun diputar 90 derajat, terdapat 8 kemungkinan peletakan blok yaitu 4 kemungkinan rotasi dan 4 kemungkinan pencerminan secara horizontal, kemungkinan transformasi ini sudah mencakup pencerminan vertikal. Semua kemungkinan transformasi blok disimpan dalam sebuah list yang akan diiterasikan.
4. Jika peletakan papan tidak memberikan solusi, maka kondisi papan akan dikembalikan ke state semula sebelum blok tersebut diletakkan. Kemudian, pencarian akan dilanjutkan dengan meletakkan blok selanjutnya. Proses tersebut akan terus berlanjut sampai mencapai basis.
5. Ketika sudah mencapai basis, program selesai. Program akan menampilkan salah satu solusi jika papan bisa mencapai kondisi penuh dengan menggunakan semua blok dan akan menampilkan pesan jika tidak ada solusi.

BAB II

Implementasi

2.1. Default Solver dan Custom Solver

Methods	Deskripsi
isFilled()	Mengembalikan boolean dengan nilai true jika board sudah penuh dan false jika board belum penuh.
normalize()	Membuat nilai koordinat blok menjadi positif setelah dirotasi maupun dicerminkan.
rotate(List<Pair> shape)	Merotasikan koordinat blok puzzle sebesar 90 derajat searah jarum jam.
reflectX(List<Pair> shape)	Mencerminkan koordinat blok puzzle terhadap sumbu X.
canPlace(int x, int y, List<Pair> shape)	Mengembalikan boolean yang bernilai true jika blok shape dapat diletakkan pada board mulai dari titik awal [x][y].
placeBlock(int x, int y, List<Pair> shape, char c, boolean place)	Menempatkan blok puzzle pada posisi [x][y] jika place bernilai true dan menghapus blok puzzle pada posisi [x][y] jika place bernilai false.
transform(List<Pair> shape)	Mengembalikan list dari semua kemungkinan transformasi blok, yang meliputi rotasi maupun pencerminan.
solve(int idx)	Fungsi utama perhitungan solusi penempatan blok yang berupa fungsi rekursif. Fungsi ini meninjau semua kemungkinan peletakan blok.
readBlocks(List<String> blockLines)	Membaca dan memvalidasi input blok puzzle dan menerjemahkannya ke dalam bentuk koordinat.
printBoard()	Mencetak board dengan warna berbeda untuk tiap huruf.

2.2. Class Tambahan untuk Representasi Blok

2.2.1. Shape

Variable	Deskripsi
symbol	Berisi character berupa huruf alfabet A-Z yang melambangkan jenis blok puzzle.
placed	Berisi boolean yang bernilai true jika blok puzzle tersebut sudah diletakkan pada board.

2.2.2. Pair

Variable	Deskripsi
x	Posisi baris suatu potongan blok.
y	Posisi kolom suatu potongan blok.

BAB III

Source Code

3.1. Repository Github

https://github.com/rararana/Tucil1_13523007

3.2. Source Code Program

3.2.1. Main.java

```
import javax.swing.SwingUtilities;

public class Main {
    public static void main(String[] args) {
        SwingUtilities.invokeLater(GUI::new);
    }
}
```

3.2.2. Header.java

```
import java.io.*;
import java.awt.*;
import java.awt.image.BufferedImage;
import javax.imageio.ImageIO;

public class Header{
    public static int N, M, P;
    public static String S;
    public static Color getColor(char c){
        return switch (c){
            case 'A' -> new Color(255, 0, 0);
            case 'B' -> new Color(0, 0, 255);
            case 'C' -> new Color(0, 255, 0);
            case 'D' -> new Color(255, 255, 0);
            case 'E' -> new Color(255, 165, 0);
            case 'F' -> new Color(128, 0, 128);
            case 'G' -> new Color(0, 255, 255);
            case 'H' -> new Color(255, 20, 147);
            case 'I' -> new Color(139, 69, 19);
```

```
            case 'J' -> new Color(0, 128, 128);
            case 'K' -> new Color(173, 255, 47);
            case 'L' -> new Color(75, 0, 130);
            case 'M' -> new Color(255, 192, 203);
            case 'N' -> new Color(0, 0, 139);
            case 'O' -> new Color(255, 140, 0);
            case 'P' -> new Color(153, 50, 204);
            case 'Q' -> new Color(34, 139, 34);
            case 'R' -> new Color(255, 99, 71);
            case 'S' -> new Color(47, 79, 79);
            case 'T' -> new Color(72, 61, 139);
            case 'U' -> new Color(220, 20, 60);
            case 'V' -> new Color(0, 191, 255);
            case 'W' -> new Color(255, 215, 0);
            case 'X' -> new Color(0, 250, 154);
            case 'Y' -> new Color(186, 85, 211);
            case 'Z' -> new Color(70, 130, 180);
            default -> Color.WHITE;
        };
    }
    //KONSTRUKTOR
    public static void setHeader(int n, int m, int p, String s){
        N = n;
        M = m;
        P = p;
        S = s;
    }
    public static void saveAsFile(char[][] board, String filename){
        try(BufferedWriter writer = new BufferedWriter(new FileWriter(filename))){
            for(char[] b : board){
                writer.write(b);
                writer.newLine();
            }
            System.out.println("Solusi berhasil disimpan dengan nama " + filename);
        } catch (IOException e) {
            System.out.println("Terjadi kesalahan saat menulis ke file: " + e.getMessage());
        }
    }
    public static void saveAsImage(char[][] board, String filename){
        int width = M*50;
        int height = N*50;
```

```
                BufferedImage image = new BufferedImage(width, height,
BufferedImage.TYPE_INT_RGB);
                Graphics2D blockImage = image.createGraphics();
                blockImage.setColor(Color.WHITE);
                blockImage.fillRect(0, 0, width, height);

                for(int i=0; i<N; i++){
                    for(int j=0; j<M; j++){
                        if(board[i][j] == ' ') continue;
                        blockImage.setColor(getColor(board[i][j]));
                        blockImage.fillRect(j*50, i*50, 50, 50);
                    }
                }
                blockImage.dispose();
                try {
                    ImageIO.write(image, "png", new File(filename));
                    System.out.println("Solusi berhasil disimpan sebagai " + filename);
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }

        class Shape{
            private final char symbol;
            private boolean placed;

            Shape(char symbol){
                this.symbol = symbol;
                this.placed = false;
            }

            char getSymbol(){
                return symbol;
            }

            boolean isPlaced(){
                return placed;
            }

            void setPlaced(boolean placed){
                this.placed = placed;
```

```
    }
}
class Pair{
    int x, y;
    Pair(int x, int y){
        this.x = x;
        this.y = y;
    }
}
```

3.2.3. DefaultSolver.java

```
import java.util.*;

public class DefaultSolver{
    public static final String RESET = "\033[0m";  // Text Reset
    static final String[] ANSI = new String[26];
    static long start, end;
    static{
        ANSI[0] = "\033[0;30m";   // BLACK
        ANSI[1] = "\033[0;31m";     // RED
        ANSI[2] = "\033[0;32m";   // GREEN
        ANSI[3] = "\033[0;33m";  // YELLOW
        ANSI[4] = "\033[0;34m";    // BLUE
        ANSI[5] = "\033[0;35m";  // PURPLE
        ANSI[6] = "\033[0;36m";    // CYAN
        ANSI[7] = "\033[0;37m";   // WHITE
        ANSI[8] = "\033[1;30m";  // BLACK BOLD
        ANSI[9] = "\033[1;31m";    // RED BOLD
        ANSI[10] = "\033[1;32m";  // GREEN BOLD
        ANSI[11] = "\033[1;33m"; // YELLOW BOLD
        ANSI[12] = "\033[1;34m";   // BLUE BOLD
        ANSI[13] = "\033[1;35m"; // PURPLE BOLD
        ANSI[14] = "\033[1;36m";   // CYAN BOLD
        ANSI[15] = "\033[1;37m";  // WHITE BOLD
        ANSI[16] = "\033[4;30m";  // BLACK UNDERLINE
        ANSI[17] = "\033[4;31m";    // RED UNDERLINE
        ANSI[18] = "\033[4;32m";  // GREEN UNDERLINE
        ANSI[19] = "\033[4;33m"; // YELLOW UNDERLINE
        ANSI[20] = "\033[4;34m";   // BLUE UNDERLINE
        ANSI[21] = "\033[4;35m"; // PURPLE UNDERLINE
```

```
        ANSI[22] = "\033[4;36m";   // CYAN UNDERLINE
        ANSI[23] = "\033[4;37m";  // WHITE UNDERLINE
        ANSI[24] = "\033[40m";  // BLACK BG
        ANSI[25] = "\033[41m";    // RED BG
    }

    static char[][] board;
    List<Shape> shapes;
    Map<Character, List<Pair>> blockMap;
    static long count = 0;
    static boolean foundSolution;

    //KONSTRUKTOR
    public DefaultSolver(){
        board = new char[Header.N][Header.M];
        this.shapes = new ArrayList<>();
        this.blockMap = new HashMap<>();

        for(int i=0; i<Header.N; i++){
            for(int j=0; j<Header.M; j++){
                board[i][j] = '.';
            }
        }
    }
    private boolean isFilled() {
        for(int i=0; i<Header.N; i++){
            for(int j=0; j<Header.M; j++){
                if(board[i][j] == '.'){
                    return false;
                }
            }
        }
        return true;
    }
    private List<Pair> normalize(List<Pair> shape){
        int minRow = Integer.MAX_VALUE, minCol = Integer.MAX_VALUE;
        for(Pair p : shape){
            minRow = Math.min(minRow, p.x);
            minCol = Math.min(minCol, p.y);
        }
        List<Pair> normalized = new ArrayList<>();
        for(Pair p : shape){
```

```
            normalized.add(new Pair(p.x-minRow, p.y-minCol));
        }
        return normalized;
    }
    private List<Pair> rotate(List<Pair> shape){
        List<Pair> rotated = new ArrayList<>();
        for(Pair p : shape){
            rotated.add(new Pair(p.y, -p.x));
        }
        return normalize(rotated);
    }
    private List<Pair> reflectX(List<Pair> shape){
        List<Pair> reflected = new ArrayList<>();
        for(Pair p : shape){
            reflected.add(new Pair(p.x, -p.y));
        }
        return normalize(reflected);
    }
    private boolean canPlace(int x, int y, List<Pair> shape){
        for(Pair p : shape){
            int nx = x + p.x, ny = y + p.y;
            if(nx >= Header.N || ny >= Header.M || board[nx][ny] != '.'){
                return false;
            }
        }
        return true;
    }
    private void placeBlock(int x, int y, List<Pair> shape, char c, boolean place){
        for(Pair p : shape){
            int nx = x + p.x, ny = y + p.y;
            board[nx][ny] = place ? c : '.';
        }
    }
    private List<List<Pair>> transform(List<Pair> shape){
        List<List<Pair>> transformations = new ArrayList<>();
        transformations.add(shape);
        transformations.add(reflectX(shape));

        shape = rotate(shape); //90 deg clockwise
        transformations.add(shape);
        transformations.add(reflectX(shape));
```

```
        shape = rotate(shape); //180 deg clockwise
        transformations.add(shape);
        transformations.add(reflectX(shape));

        shape = rotate(shape); //270 deg clockwise
        transformations.add(shape);
        transformations.add(reflectX(shape));

        return transformations;
    }
    private boolean solve(int idx){
        if(isFilled()) return true;
        if(idx >= Header.P) return false;

        Shape currentShape = shapes.get(idx);
        if(currentShape.isPlaced()) return solve(idx + 1);

        List<Pair> shapeBlock = blockMap.get(currentShape.getSymbol());
        List<List<Pair>> allTransforms = transform(shapeBlock);

        for(List<Pair> transformed : allTransforms){
            for(int i=0; i<Header.N; i++) {
                for (int j=0; j<Header.M; j++) {
                    if(canPlace(i, j, transformed)){
                        placeBlock(i, j, transformed, currentShape.getSymbol(), true);
                        currentShape.setPlaced(true);
                        count++;
                        if(solve(idx + 1)) return true;
                        currentShape.setPlaced(false);
                        placeBlock(i, j, transformed, currentShape.getSymbol(), false);
                    }
                }
            }
        }
        return false;
    }
    public void readBlocks(List<String> blockLines){
        int i = 0, blockSum = 0, cnt = 1;
        char prev = ' ';
        boolean flag = true;
        for(String line : blockLines){
            for(int j=0; j<line.length(); j++){
```

```
                char a = line.charAt(j);
                if(a == ' ') continue;
                if(flag || a == prev){
                    flag = false;
                    blockSum++;
                    blockMap.computeIfAbsent(a, key -> new ArrayList<>()).add(new Pair(i, j));
                }else{
                    cnt++;
                    shapes.add(new Shape(prev));
                    blockSum++;
                    i = 0;
                    blockMap.computeIfAbsent(a, key -> new ArrayList<>()).add(new Pair(i, j));
                }
                prev = a;
            }
            i++;
        }
        shapes.add(new Shape(prev));
        //start = System.currentTimeMillis();
        if(cnt != Header.P){
            //System.out.println("Jumlah blok puzzle tidak sesuai.");
            foundSolution = false;
        } else if(blockSum != Header.N*Header.M) {
            //System.out.println("Tidak ada solusi.");
            foundSolution = false;
        } else {
            if(solve(0)){
                //System.out.println("Solusi ditemukan!");
                //printBoard();
                foundSolution = true;
            }else{
                //System.out.println("Tidak ada solusi.");
                foundSolution = false;
            }
        }
        //end = System.currentTimeMillis();
    }
    public static void printBoard(){
        for(int i=0; i<Header.N; i++){
            for(int j=0; j<Header.M; j++){
                System.out.print(ANSI[board[i][j]-'A']+board[i][j]+RESET);
            }
```

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

3.2.6. CustomSolver.java

```
    import java.util.*;

    public class CustomSolver{
        public static final String RESET = "\033[0m";  // Text Reset
        static final String[] ANSI = new String[26];
        static long start, end;
        static{
            ANSI[0] = "\033[0;30m";   // BLACK
            ANSI[1] = "\033[0;31m";     // RED
            ANSI[2] = "\033[0;32m";   // GREEN
            ANSI[3] = "\033[0;33m";  // YELLOW
            ANSI[4] = "\033[0;34m";    // BLUE
            ANSI[5] = "\033[0;35m";  // PURPLE
            ANSI[6] = "\033[0;36m";    // CYAN
            ANSI[7] = "\033[0;37m";   // WHITE
            ANSI[8] = "\033[1;30m";  // BLACK BOLD
            ANSI[9] = "\033[1;31m";    // RED BOLD
            ANSI[10] = "\033[1;32m";  // GREEN BOLD
            ANSI[11] = "\033[1;33m"; // YELLOW BOLD
            ANSI[12] = "\033[1;34m";   // BLUE BOLD
            ANSI[13] = "\033[1;35m"; // PURPLE BOLD
            ANSI[14] = "\033[1;36m";   // CYAN BOLD
            ANSI[15] = "\033[1;37m";  // WHITE BOLD
            ANSI[16] = "\033[4;30m";  // BLACK UNDERLINE
            ANSI[17] = "\033[4;31m";    // RED UNDERLINE
            ANSI[18] = "\033[4;32m";  // GREEN UNDERLINE
            ANSI[19] = "\033[4;33m"; // YELLOW UNDERLINE
            ANSI[20] = "\033[4;34m";   // BLUE UNDERLINE
            ANSI[21] = "\033[4;35m"; // PURPLE UNDERLINE
            ANSI[22] = "\033[4;36m";   // CYAN UNDERLINE
            ANSI[23] = "\033[4;37m";  // WHITE UNDERLINE
            ANSI[24] = "\033[40m";  // BLACK BG
            ANSI[25] = "\033[41m";    // RED BG
        }
```

```
static char[][] board;
List<Shape> shapes;
Map<Character, List<Pair>> blockMap;
static long count = 0;
static boolean foundSolution;

//KONSTRUKTOR
public CustomSolver(){
    board = new char[Header.N][Header.M];
    this.shapes = new ArrayList<>();
    this.blockMap = new HashMap<>();
}
private boolean isFilled(){
    for(int i=0; i<Header.N; i++){
        for(int j=0; j<Header.M; j++){
            if(board[i][j] == 'X'){
                return false;
            }
        }
    }
    return true;
}
private List<Pair> normalize(List<Pair> shape){
    int minRow = Integer.MAX_VALUE, minCol = Integer.MAX_VALUE;
    for(Pair p : shape){
        minRow = Math.min(minRow, p.x);
        minCol = Math.min(minCol, p.y);
    }
    List<Pair> normalized = new ArrayList<>();
    for(Pair p : shape){
        normalized.add(new Pair(p.x-minRow, p.y-minCol));
    }
    return normalized;
}
private List<Pair> rotate(List<Pair> shape){
    List<Pair> rotated = new ArrayList<>();
    for(Pair p : shape){
        rotated.add(new Pair(p.y, -p.x));
    }
    return normalize(rotated);
}
```

```
    private List<Pair> reflectX(List<Pair> shape){
        List<Pair> reflected = new ArrayList<>();
        for(Pair p : shape){
            reflected.add(new Pair(p.x, -p.y));
        }
        return normalize(reflected);
    }
    private boolean canPlace(int x, int y, List<Pair> shape){
        for(Pair p : shape){
            int nx = x + p.x, ny = y + p.y;
            if(nx >= Header.N || ny >= Header.M || board[nx][ny] != 'X'){
                return false;
            }
        }
        return true;
    }
    private void placeBlock(int x, int y, List<Pair> shape, char c, boolean place){
        for(Pair p : shape){
            int nx = x + p.x, ny = y + p.y;
            board[nx][ny] = place ? c : 'X';
        }
    }
    private List<List<Pair>> transform(List<Pair> shape){
        List<List<Pair>> transformations = new ArrayList<>();
        transformations.add(shape);
        transformations.add(reflectX(shape));

        shape = rotate(shape); //90 deg clockwise
        transformations.add(shape);
        transformations.add(reflectX(shape));

        shape = rotate(shape); //180 deg clockwise
        transformations.add(shape);
        transformations.add(reflectX(shape));

        shape = rotate(shape); //270 deg clockwise
        transformations.add(shape);
        transformations.add(reflectX(shape));

        return transformations;
    }
    private boolean solve(int idx){
```

```
        if(isFilled()) return true;
        if(idx >= Header.P) return false;

        Shape currentShape = shapes.get(idx);
        if(currentShape.isPlaced()) return solve(idx + 1);

        List<Pair> shapeBlock = blockMap.get(currentShape.getSymbol());
        List<List<Pair>> allTransforms = transform(shapeBlock);

        for(List<Pair> transformed : allTransforms){
            for(int i=0; i<Header.N; i++) {
                for(int j=0; j<Header.M; j++){
                    if(canPlace(i, j, transformed)){
                        placeBlock(i, j, transformed, currentShape.getSymbol(), true);
                        currentShape.setPlaced(true);
                        count++;
                        if(solve(idx + 1)) return true;
                        currentShape.setPlaced(false);
                        placeBlock(i, j, transformed, currentShape.getSymbol(), false);
                    }
                }
            }
        }
        return false;
    }
    public void readBlocks(List<String> blockLines){
        int i = 0, blockSum = 0, cnt = 1, row = 0, cntX = 0;
        char prev = ' ';
        boolean flag = true;
        for(String line : blockLines){
            for(int j=0; j<line.length(); j++){
                if(row < Header.N){
                    board[row][j] = line.charAt(j);
                    if(board[row][j] == 'X') cntX++;
                }else{
                    char a = line.charAt(j);
                    if(a == ' ') continue;
                    if(flag || a == prev){
                        flag = false;
                        blockSum++;
                        blockMap.computeIfAbsent(a, key -> new ArrayList<>()).add(new Pair(i, j));
                    }else{
```

```
                    cnt++;
                    shapes.add(new Shape(prev));
                    blockSum++;
                    i = 0;
                    blockMap.computeIfAbsent(a, key -> new ArrayList<>()).add(new Pair(i, j));
                }
                prev = a;
            }

        }
        i++;
        row++;
    }
    shapes.add(new Shape(prev));
    //start = System.currentTimeMillis();
    if(cnt != Header.P){
        // System.out.println("Jumlah blok puzzle tidak sesuai.");
        foundSolution = false;
    } else if(blockSum != cntX) {
        //System.out.println("Tidak ada solusi.");
        foundSolution = false;
    } else {
        if(solve(0)){
            //System.out.println("Solusi ditemukan!");
            //printBoard();
            foundSolution = true;
        }else{
            //System.out.println("Tidak ada solusi.");
            foundSolution = false;
        }
    }
    //end = System.currentTimeMillis();
}
public static void printBoard(){
    for(int i=0; i<Header.N; i++){
        for(int j=0; j<Header.M; j++){
            if(board[i][j] == '.'){
                System.out.print(' ');
                board[i][j] = ' ';
            } else System.out.print(ANSI[board[i][j]-'A']+board[i][j]+RESET);
        }
        System.out.println();
```

```
            }
        }
    }
```

3.3. Cara Menjalankan Program

1. Buka terminal atau Command Prompt
2. Ganti direktori ke folder src
3. Compile dengan mengetikkan javac Main.java
4. Jalankan program dengan mengetikkan java Main
5. GUI akan terbuka dan input dapat dimasukkan
6. Solusi ditampilkan (jika ada) dan terdapat opsi untuk menyimpan solusi

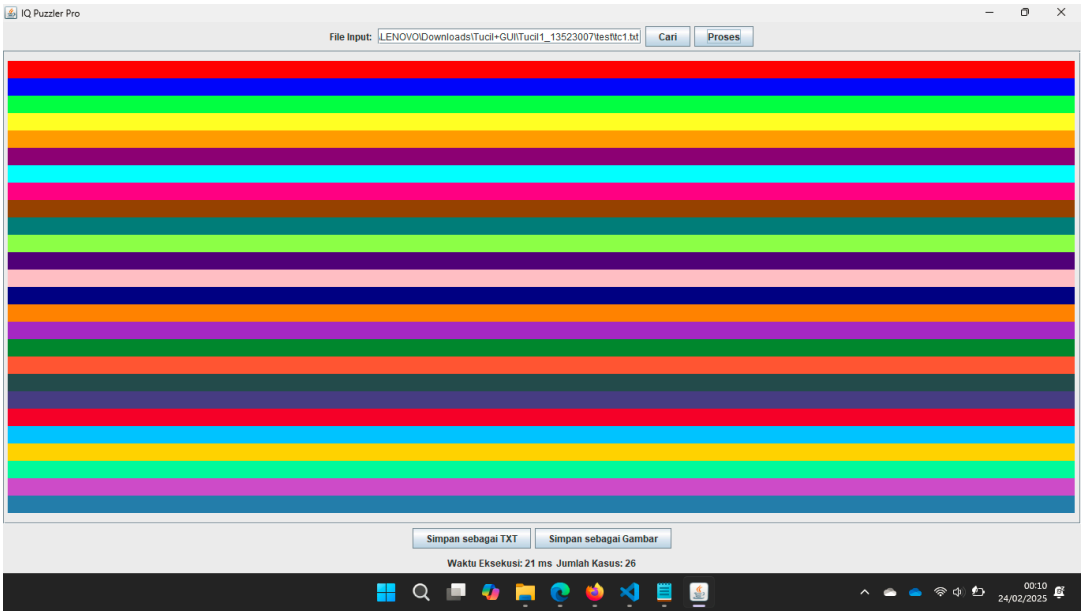
BAB IV

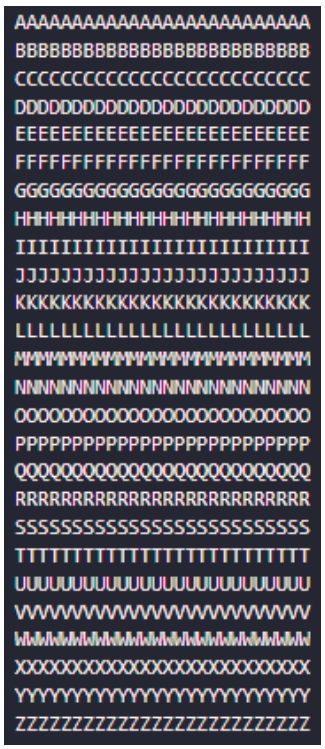
Eksperimen

4.1. Test Case 1

4.1.1. Input

4.1.2. Output





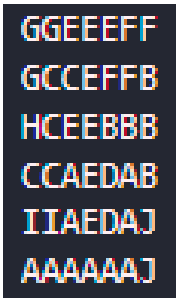
Di atas merupakan hasil menyimpan solusi dalam bentuk txt dan png (solusitc1.txt dan solusitc1.png)

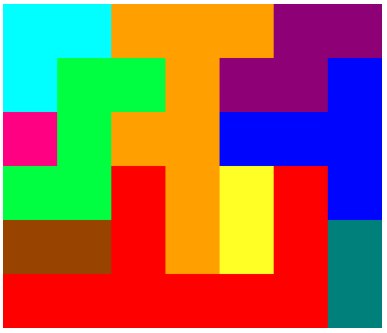
4.2. Test Case 2

4.2.1. Input

```
6 7 10
DEFAULT
  A  A
  A  A
AAAAAA
BBB
 B
 B
 CC
 C
CC
D
D
E E
EEEEE
E
F
FF
 F
GG
G
H
II
JJ
```

4.2.2. Output

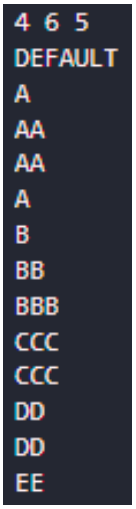




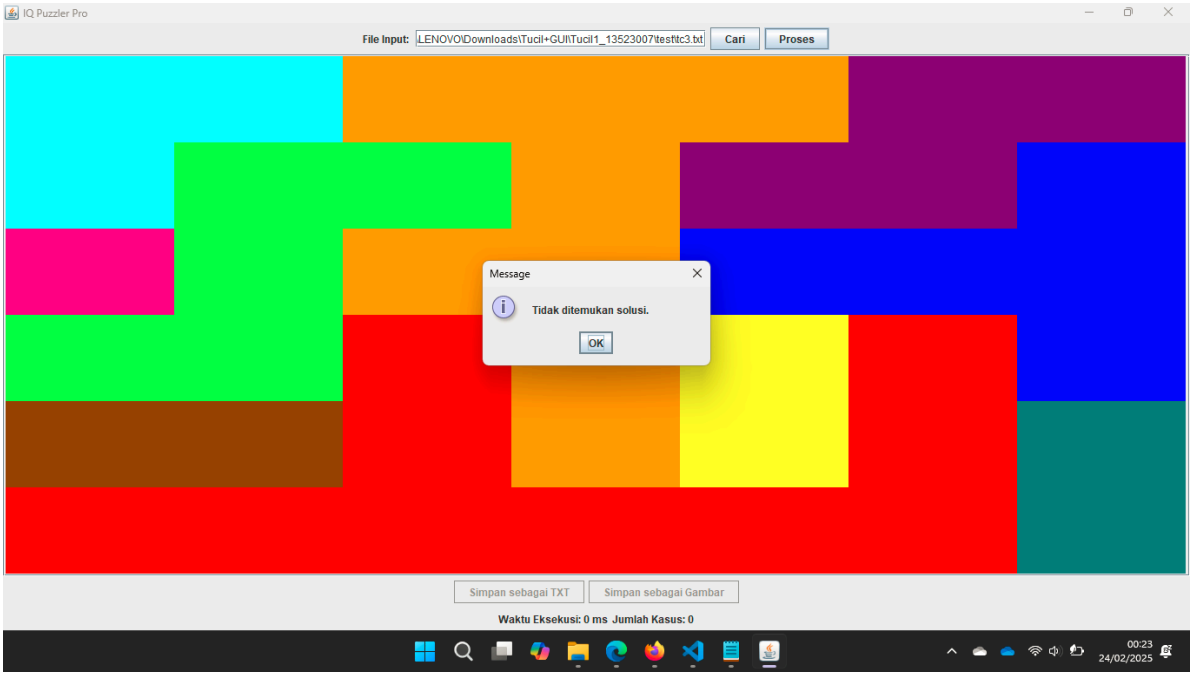
Di atas merupakan hasil menyimpan solusi dalam bentuk txt dan png (solusitc2.txt dan solusitc2.png)

4.3. Test Case 3

4.3.1. Input



4.3.2. Output

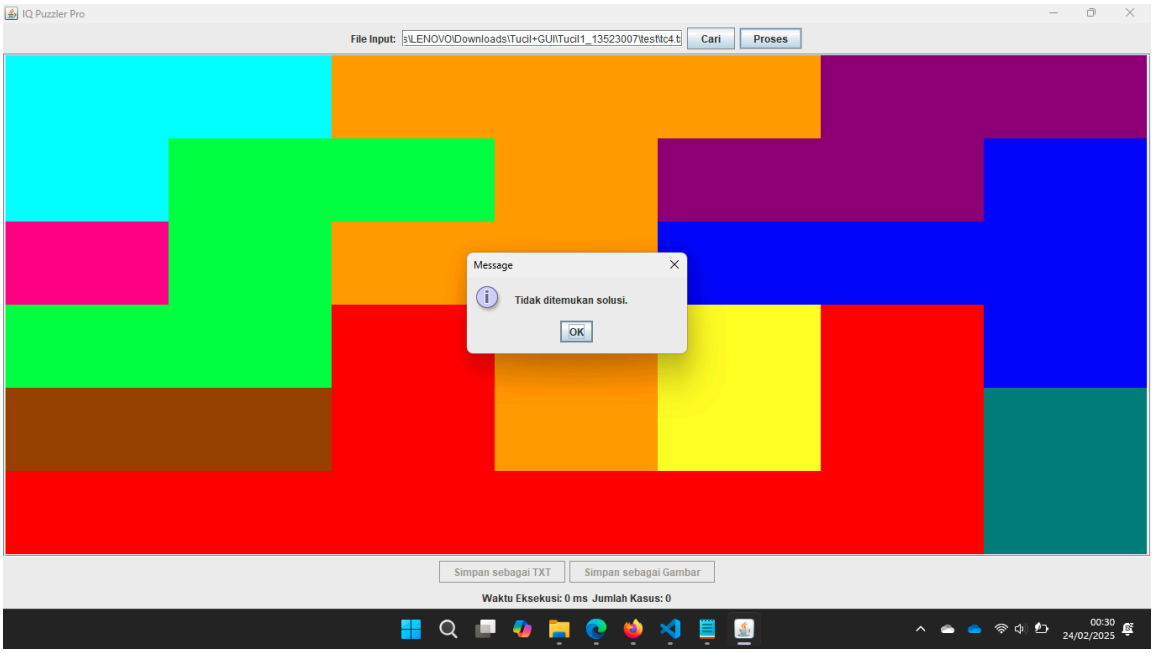


4.4. Test Case 4

4.4.1. Input

```
4 6 6
DEFAULT
A
AA
AA
A
B
BB
BBB
CCC
CCC
DD
DD
EE
F
```

4.4.2. Output

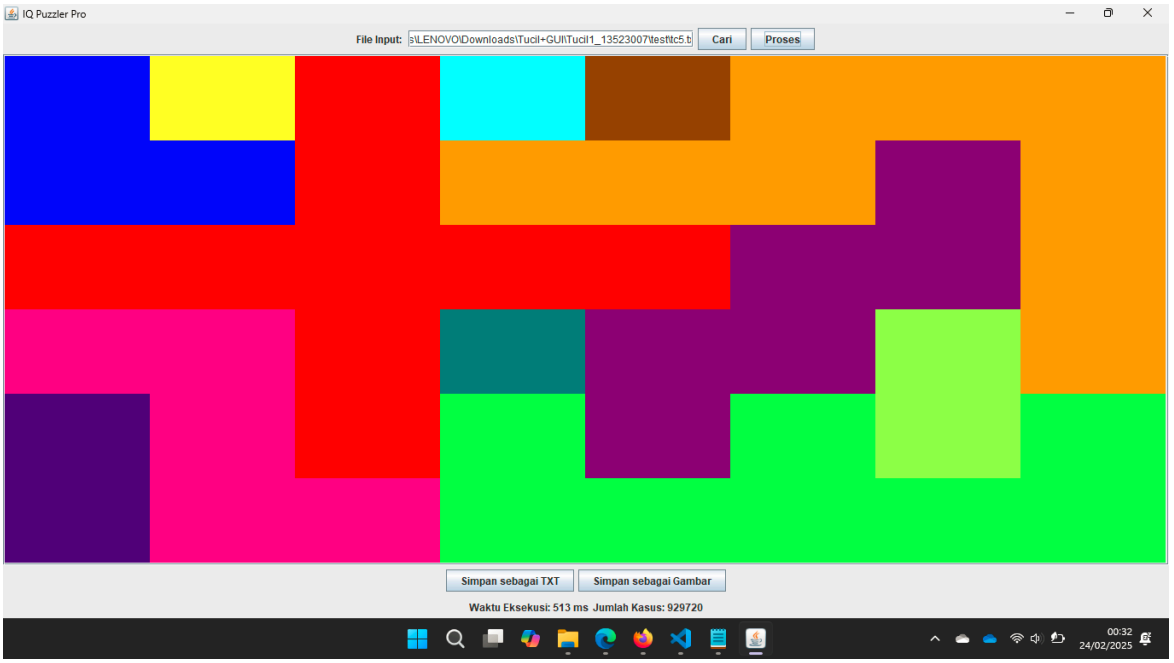


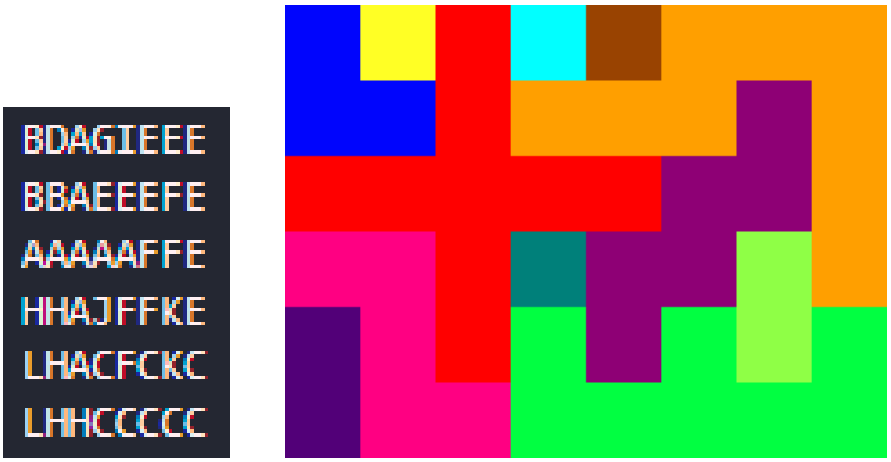
4.5. Test Case 5

4.5.1. Input

```
6 8 12
DEFAULT
  A
  A
AAAAA
  A
  A
B
BB
CC
 C
CC
 C
CC
D
  E
  E
  EE
   E
EEEE
  F
 FF
FF
F
G
 HH
 H
HH
I
J
KK
L
L
```

4.5.2. Output





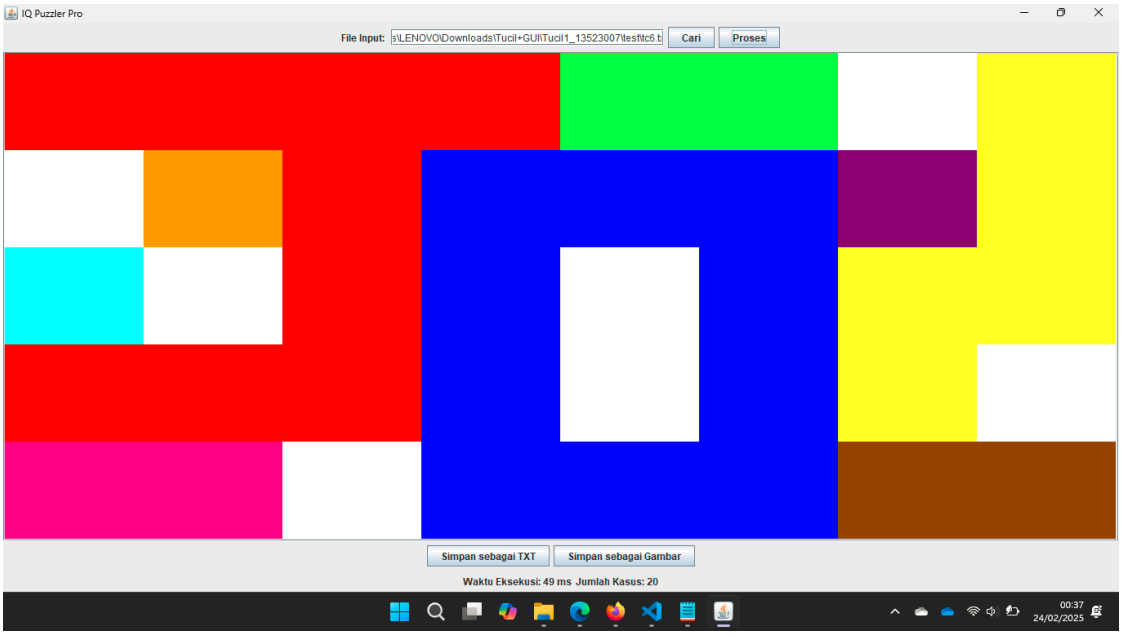
Di atas merupakan hasil menyimpan solusi dalam bentuk txt dan png (solusitc5.txt dan solusitc5.png)

4.6 Test Case 6

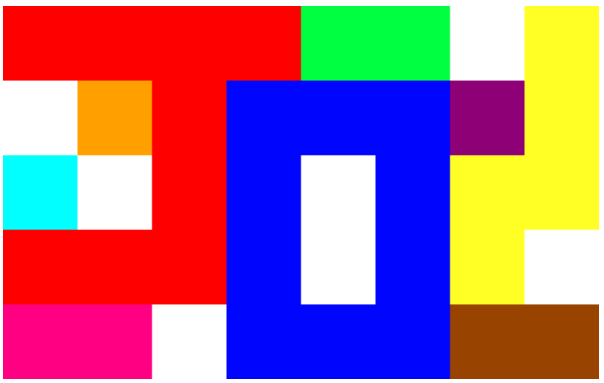
4.6.1. Input

```
5 8 9
CUSTOM
XXXXXX.X
.XXXXXXX
X.XX.XXX
XXXX.XX.
XX.XXXXX
A
AAAA
A  A
A  A
BBBB
B  B
BBBB
C
C
DDD
  DD
E
F
G
H
H
I
I
```

4.6.2. Output



```
AAAACC.D
.EABBBFD
G.AB.BDD
AAAB.BD.
HH.BBBII
```



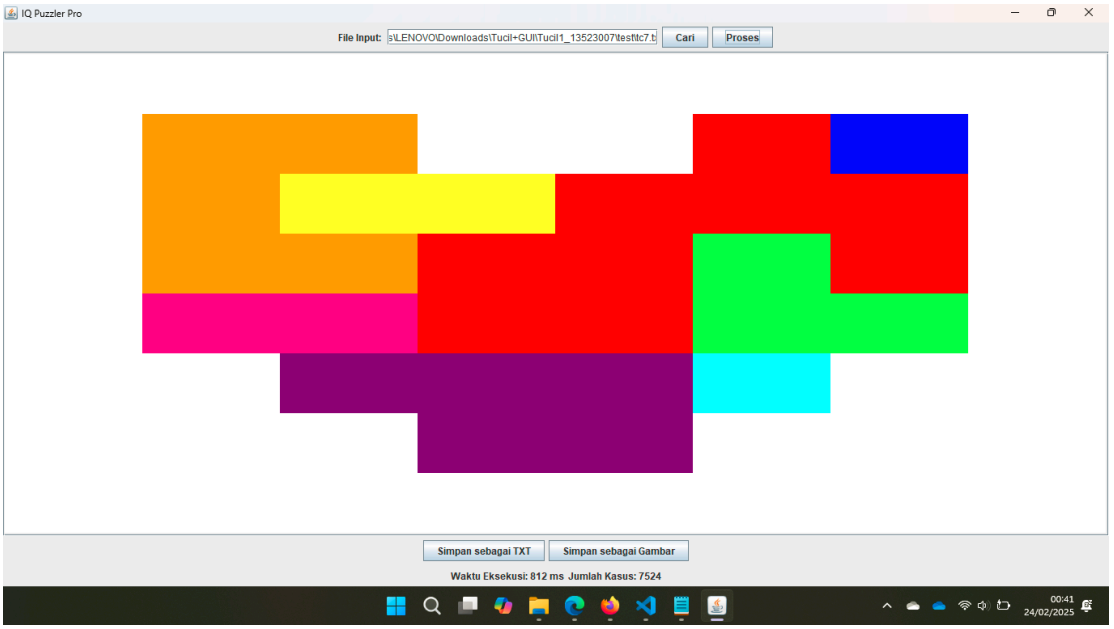
Di atas merupakan hasil menyimpan solusi dalam bentuk txt dan png (solusitc6.txt dan solusitc6.png)

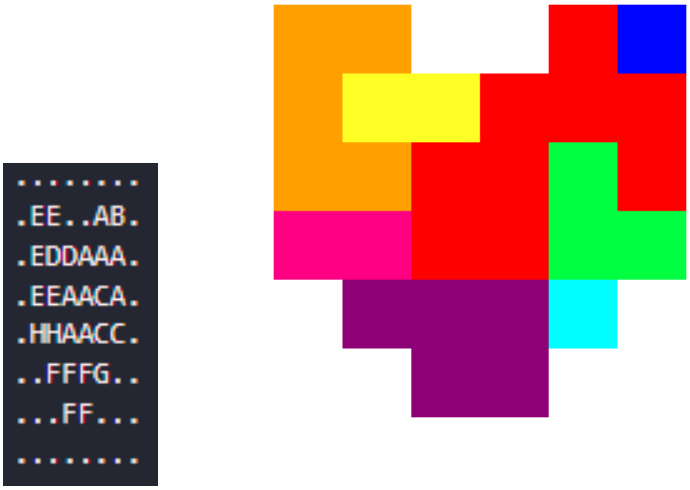
4.7 Test Case 7

4.7.1. Input

```
8 8 8
CUSTOM
........
.XX..XX.
.XXXXXX.
.XXXXXX.
.XXXXXX.
..XXXX..
...XX...
........
 A
AAA
A AA
  AA
B
C
CC
DD
EE
 E
EE
FF
FFF
G
HH
```

4.7.2. Output





Di atas merupakan hasil menyimpan solusi dalam bentuk txt dan png (solusitc7.txt dan solusitc7.png)

BAB V
Lampiran

No	Poin	Ya	Tidak
1	Program berhasil dikompilasi tanpa kesalahan	V	
2	Program berhasil dijalankan	V	
3	Solusi yang diberikan program benar dan mematuhi aturan permainan	V	
4	Program dapat membaca masukan berkas .txt serta menyimpan solusi dalam berkas .txt	V	
5	Program memiliki *Graphical User Interface* (GUI)	V	
6	Program dapat menyimpan solusi dalam bentuk file gambar	V	
7	Program dapat menyelesaikan kasus konfigurasi *custom*	V	
8	Program dapat menyelesaikan kasus konfigurasi Piramida (3D)		V
9	Program dibuat oleh saya sendiri	V	