Tugas Kecil 1 IF2211 Strategi Algoritma

Semester II tahun 2024/2025

Penyelesaian IQ Puzzler Pro dengan Algoritma Brute Force

Muhammad Ra'if Alkautsar 13523011

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2025

DAFTAR ISI

BAB I
DESKRIPSI PERMASALAHAN

A. Deskripsi

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.

B. Ilustrasi Kasus

Diberikan sebuah wadah berukuran 11 x 5 serta 12 buah blok puzzle dan beberapa blok telah ditempatkan dengan bentuk sebagai berikut. Pemain berusaha untuk mengisi bagian papan yang kosong dengan menggunakan blok yang tersedia. Untuk tugas kecil ini, permainan diawali dengan papan kosong.

Gambar 2 Awal Permainan Game IQ Puzzler Pro

(Sumber: https://www.smartgamesusa.com)

Gambar 3 Pemain Mencoba Semua Kemungkinan

(Sumber: https://www.smartgamesusa.com)

Permainan dinyatakan selesai jika pemain mampu mengisi seluruh papan dengan blok (dalam tugas kecil ini ada kemungkinan pemain tidak dapat mengisi seluruh papan). 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)

BAB II
ALGORITMA BRUTE FORCE

Pemecahan papan IQ Puzzler Pro dilakukan dengan menggunakan algoritma *brute force* yang melibatkan rekursi dan backtracking.

Dalam program ini, objek Block adalah sebuah objek yang terdiri atas Ball. Ball adalah sebuah objek yang memiliki atribut x (integer), y (integer), dan id (character). Sementara itu, blok adalah objek yang memiliki properti Set of Ball (sebuah set yang terdiri atas Ball unik, menyatakan bentuk dari blok) dan juga List of Set of Ball (sebuah list yang berisi Set of Ball, menyatakan semua rotasi unik yang mungkin dari blok). Terdapat sebuah objek Board yang memiliki atribut integer N dan M, sebuah matriks yang elemen-elemennya bertipe data char, dan sebuah variabel counter bernama iteration yang menghitung jumlah iterasi ketika melakukan pemecahan nantinya.

Program ini memanfaatkan kelas HashSet dan List yang disediakan oleh library java.util. Fitur dari HashSet antara lain: tidak ada duplikat, operasi add, remove, dan contain yang berkompleksitas waktu O(1), dan tak terurut. Fitur dari List antara lain: ada duplikat, akses terindeks, dan terurut.

Pertama, papan diinisialisasi terlebih dahulu. Program akan meminta masukan ukuran papan dari pengguna (N untuk jumlah baris dan M untuk jumlah kolom) beserta jumlah blok. Masukan blok-blok yang berbentuk String akan di-parse menjadi List of Blocks. Perhitungan tiap rotasi dari suatu blok dilakukan pada saat constructor dari Block dipanggil. Normalisasi juga dilakukan terhadap tiap blok supaya sel paling atas kirinya berada pada titik (0,0).

Berikut adalah kode untuk melakukan pre-komputasi semua rotasi. HashSet digunakan untuk menyetor hasil tiap rotasi.

```
1.      private List<Set<Ball>> precomputeRotations() {
2.          Set<Set<Ball>> uniqueRotations = new HashSet<>();
3.          Set<Ball> current = balls;
4.          for (int i = 0; i < 4; i++) {
5.              current = rotate90(current);
6.              uniqueRotations.add(current);
7.              uniqueRotations.add(mirror(current));
8.          }
9.          return new ArrayList<>(uniqueRotations);
10.     }
11.
12.     private Set<Ball> rotate90(Set<Ball> balls) {
13.         Set<Ball> rotated = new HashSet<>();
14.         for (Ball ball : balls) {
15.             rotated.add(new Ball(-ball.y, ball.x, ball.id));
16.         }
17.         return normalize(rotated);
18.     }
19.
20.     private Set<Ball> mirror(Set<Ball> balls) {
21.         Set<Ball> mirrored = new HashSet<>();
22.         for (Ball ball : balls) {
23.             mirrored.add(new Ball(ball.x, -ball.y, ball.id));
24.         }
25.         return normalize(mirrored);
26.     }
27.
```

Kemudian, untuk menyiapkan Board, tiap sel atau elemen pada board diubah menjadi karakter '0' yang mengindikasikan bahwa sel tersebut kosong. Pada mode custom, sel-sel yang diblokir akan di-assign dengan nilai '.' yang mengindikasikan bahwa sel tersebut tidak bisa ditempati oleh blok. Kode lebih detil terdapat pada bab kode sumber. Setelah itu, akan dilakukan pemecahan papan.

```
    public boolean solveDefault(List<Block> blocks, int index) {
        if (index == blocks.size()) return isBoardFilled();
        Block block = blocks.get(index);
        for (Set<Ball> rotation : block.getRotations()) {
            for (int x = 0; x < N; x++) {
                for (int y = 0; y < M; y++) {
                    iteration++;
                    if (placeBlock(rotation, x, y, index + 1)) {
                        if (solveDefault(blocks, index + 1)) return true;
                        removeBlock(rotation, x, y);
                    }
                }
            }
        }
        return false;
    }
```

Fungsi solve mencoba untuk memasang blok satu per satu di papan melalui cara rekursif.

A. Base Case

Kasus basis dari solve adalah apabila semua blok yang ada pada list telah digunakan, maka akan dilakukan pemanggilan ke suatu fungsi isBoardFilled() untuk memeriksa apakah papan sudah terisi penuh (dengan cara memeriksa apakah masih ada sel yang bernilai '0'). Jika benar, maka fungsi rekursi akan mengembalikan nilai true. Namun, jika tidak, maka akan dilakukan iterasi semua kemungkinan.

B. Iterasi

Ada tiga for loop pada algoritma: paling luar adalah mengiterasi tiap rotasi, lalu di dalamnya adalah mengiterasi tiap posisi pada papan, lalu di dalamnya lagi adalah pemeriksaan apakah blok dapat dipasang. Ada pula sebuah counter berupa variabel global untuk menghitung berapa banyak iterasi atau percobaan pemasangan yang telah dilakukan. Variabel ini digunakan untuk menghitung iterasi brute force yang telah dilakukan.

C. Pemasangan Blok

Metode placeBlock memeriksa apakah blok bisa dipasang pada posisi (x,y). Cara kerja dari metode tersebut adalah dengan mengiterasi tiap Ball dengan posisi relatif dari titik pemasangan dan memeriksa apakah ada ruang kosong. Apabila ada ruang tersedia, maka blok akan dipasang dengan menaruh character id di sel-sel yang ditempati.

```
    public boolean placeBlock(Set<Ball> block, int x, int y, int index) {
        for (Ball ball : block) {
            int newX = x + ball.x;
            int newY = y + ball.y;
            if (newX < 0 || newX >= N || newY < 0 || newY >= M || grid[newX][newY] !=
'0') {
                return false;
            }
```

```
        }
        for (Ball ball : block) {
            grid[x + ball.x][y + ball.y] = ball.id;
        }
        return true;
    }

    public void removeBlock(Set<Ball> block, int x, int y) {
        for (Ball ball : block) {
            grid[x + ball.x][y + ball.y] = '0';
        }
    }

```

D. Rekursi dan Backtracking

Setelah sebuah blok dipasang dengan sukses, akan dilakukan pemanggilan rekursif terhadap fungsi solve sendiri dengan argumen blok selanjutnya atau index + 1. Jika pemanggilan rekursif mengembalikan true, berarti solusi telah ditemukan dan pengembalian true tersebut akan berlanjut sampai fungsi pemanggil paling awal. Namun, jika pemanggilan rekursif mengembalikan false, maka algoritma akan melakukan backtracking dengan menghapus blok lalu mencoba posisi dan orientasi lain.

Jika tak ada penempatan pada blok saat ini (dalam rotasi maupun posisi) yang memberikan solusi, fungsi akan mengembalikan false. Jika masih ada tingkat rekursi sebelumnya, maka algoritma akan mencoba konfigurasi lain untuk blok sebelumnya. Jika tidak, program akan menyatakan bahwa tidak ada solusi yang ditemukan.

BAB III
KODE SUMBER

Kode sumber terbagi menjadi tujuh kelas: Ball, Block, Board, Parser, ParsedData, MainCLI, dan MainGUI. Kode sumber ini terletak pada folder src. Struktur folder dari program adalah sebagai berikut:

```
.
├── bin
├── src
├── test
└── docs
```

A. Ball

```java
import java.util.Objects;

public class Ball {
    int x, y;
    char id;

    public Ball(int x, int y, char id) {
        this.x = x;
        this.y = y;
        this.id = id;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (!(o instanceof Ball)) return false;
        Ball other = (Ball) o;
        return this.x == other.x && this.y == other.y;
    }

    @Override
    public int hashCode() {
        return Objects.hash(x, y);
    }

    @Override
    public String toString() {
        return "(" + x + ", " + y + ")";
    }
}

```

B. Block

```java
import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

public class Block {
    private final Set<Ball> balls;
    private final List<Set<Ball>> rotations;

    public Block(Set<Ball> balls) {
        // Menormalisasi bola supaya x dan y minimal-nya menjadi 0
```

```
        this.balls = normalize(balls);
        // Menghitung semua rotasi untuk menghemat performa.
        this.rotations = precomputeRotations();
    }

    private Set<Ball> normalize(Set<Ball> balls) {
        int minX = Integer.MAX_VALUE, minY = Integer.MAX_VALUE;
        for (Ball ball : balls) {
            minX = Math.min(minX, ball.x);
            minY = Math.min(minY, ball.y);
        }
        Set<Ball> normalized = new HashSet<>();
        for (Ball ball : balls) {
            normalized.add(new Ball(ball.x - minX, ball.y - minY, ball.id));
        }
        return normalized;
    }

    private List<Set<Ball>> precomputeRotations() {
        Set<Set<Ball>> uniqueRotations = new HashSet<>();
        Set<Ball> current = balls;
        for (int i = 0; i < 4; i++) {
            current = rotate90(current);
            uniqueRotations.add(current);
            uniqueRotations.add(mirror(current));
        }
        return new ArrayList<>(uniqueRotations);
    }

    private Set<Ball> rotate90(Set<Ball> balls) {
        Set<Ball> rotated = new HashSet<>();
        for (Ball ball : balls) {
            rotated.add(new Ball(-ball.y, ball.x, ball.id));
        }
        return normalize(rotated);
    }

    private Set<Ball> mirror(Set<Ball> balls) {
        Set<Ball> mirrored = new HashSet<>();
        for (Ball ball : balls) {
            mirrored.add(new Ball(ball.x, -ball.y, ball.id));
        }
        return normalize(mirrored);
    }

    public List<Set<Ball>> getRotations() {
        return rotations;
    }
}

```

C. Board

```
import javax.imageio.ImageIO;
import java.awt.*;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Set;

```

```java
public class Board {
    private final int N, M;
    private final char[][] grid;
    private int iteration;

    public Board(int N, int M) {
        this.N = N;
        this.M = M;
        this.grid = new char[N][M];
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < M; j++ ) {
                grid[i][j] = '0';
            }
        }
    }

    public int getIteration() {
        return iteration;
    }

    public char[][] getGrid() {
        return grid;
    }

    public boolean placeBlock(Set<Ball> block, int x, int y, int index) {
        for (Ball ball : block) {
            int newX = x + ball.x;
            int newY = y + ball.y;
            if (newX < 0 || newX >= N || newY < 0 || newY >= M || grid[newX][newY] != '0') {
                return false;
            }
        }
        for (Ball ball : block) {
            grid[x + ball.x][y + ball.y] = ball.id;
        }
        return true;
    }

    public void removeBlock(Set<Ball> block, int x, int y) {
        for (Ball ball : block) {
            grid[x + ball.x][y + ball.y] = '0';
        }
    }

    public boolean isBoardFilled() {
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < M; j++) {
                if (grid[i][j] == '0') return false;
            }
        }
        return true;
    }

    public boolean solveDefault(List<Block> blocks, int index) {
        if (index == blocks.size()) return isBoardFilled();
        Block block = blocks.get(index);
        for (Set<Ball> rotation : block.getRotations()) {
            for (int x = 0; x < N; x++) {
                for (int y = 0; y < M; y++) {
                    iteration++;
                    if (placeBlock(rotation, x, y, index + 1)) {
                        if (solveDefault(blocks, index + 1)) return true;
                        removeBlock(rotation, x, y);
                    }
                }
```

```
            }
        }
        return false;
    }

    public void initializeCustom(List<String> config) {
        for (int i = 0; i < N; i++) {
            String row = config.get(i);
            for (int j = 0; j < M; j++) {
                char ch = row.charAt(j);
                if (ch == '.') {
                    grid[i][j] = '.';
                } else {
                    grid[i][j] = '0';
                }
            }
        }
    }

    public boolean solveCustom(List<Block> blocks, List<String> config, int index) {
        if (index == 0) {
            initializeCustom(config);
        }
        if (index == blocks.size()) return isBoardFilled();
        Block block = blocks.get(index);
        for (Set<Ball> rotation : block.getRotations()) {
            for (int x = 0; x < N; x++) {
                for (int y = 0; y < M; y++) {
                    iteration++;
                    if (placeBlock(rotation, x, y, index + 1)) {
                        if (solveCustom(blocks, config, index + 1)) return true;
                        removeBlock(rotation, x, y);
                    }
                }
            }
        }
        return false;
    }

    public void printBoard() {
        Map<Character, String> colorMap = new HashMap<>();
        String[] colors = {
                "\u001B[31m",
                "\u001B[32m",
                "\u001B[33m",
                "\u001B[34m",
                "\u001B[35m",
                "\u001B[36m",
                "\u001B[91m",
                "\u001B[92m",
                "\u001B[93m",
                "\u001B[94m",
                "\u001B[95m",
                "\u001B[96m",
        };

        int colorIndex = 0;
        for (char c = 'A'; c <= 'Z'; c++) {
            colorMap.put(c, colors[colorIndex % colors.length]);
            colorIndex++;
        }

        final String RESET = "\u001B[0m";

        for (int i = 0; i < N; i++) {
            for (int j = 0; j < M; j++) {
```

```
                char cell = grid[i][j];
                if (colorMap.containsKey(cell)) {
                    System.out.print(colorMap.get(cell) + cell + RESET);
                } else {
                    System.out.print(cell);
                }
            }
            System.out.println();
        }
    }

    public void saveBoardToFile(String fileName) {
        File outputFile = new File(fileName);
        outputFile.getParentFile().mkdirs();

        try (FileWriter writer = new FileWriter(outputFile)) {
            for (int i = 0; i < N; i++) {
                for (int j = 0; j < M; j++) {
                    writer.write(grid[i][j]);
                }
                writer.write("\n");
            }
            System.out.println("Solusi disimpan ke: " + fileName);
        } catch (IOException e) {
            System.err.println("Error menyimpan ke file: " + e.getMessage());
        }
    }

    public void saveBoardtoImage(String fileName) {
        int cellSize = 30;
        int width = M * cellSize;
        int height = N * cellSize;
        BufferedImage image = new BufferedImage(width, height,
BufferedImage.TYPE_INT_RGB);
        Graphics2D g2d = image.createGraphics();

        g2d.setColor(Color.WHITE);
        g2d.fillRect(0, 0, width, height);

        Map<Object, Color> colorMap = new HashMap<>();
        Color[] fixedColors = {
                Color.RED,
                Color.GREEN,
                Color.YELLOW,
                Color.BLUE,
                Color.MAGENTA,
                Color.CYAN,
                Color.ORANGE,
                new Color(144, 238, 144),
                new Color(255, 255, 224),
                new Color(173, 216, 230),
                Color.MAGENTA,
                new Color(224, 255, 255)
        };

        for (int i = 0; i < 12; i++) {
            char letter = (char) ('A' + i);
            colorMap.put(letter, fixedColors[i]);
        }

        for (int i = 0; i < 14; i++) {
            char letter = (char) ('A' + 12 + i);
            float hue = (float)(i / 14.0);
            Color generated = Color.getHSBColor(hue, 0.8f, 0.8f);
            colorMap.put(letter, generated);
        }
```

```
208.
209.         for (int i = 0; i < N; i++) {
210.             for (int j = 0; j < M; j++) {
211.                 char cell = grid[i][j];
212.                 Color bg;
213.                 if (cell == '0') {
214.                     bg = Color.WHITE;
215.                 } else if (cell == '.') {
216.                     bg = Color.GRAY;
217.                 } else {
218.                     bg = colorMap.getOrDefault(cell, Color.LIGHT_GRAY);
219.                 }
220.
221.                 g2d.setColor(bg);
222.                 g2d.fillRect(j * cellSize, i * cellSize, cellSize, cellSize);
223.
224.                 g2d.setColor(Color.BLACK);
225.                 g2d.drawRect(j * cellSize, i * cellSize, cellSize, cellSize);
226.
227.                 if (cell != '0' && cell != '.') {
228.                     g2d.setColor(Color.WHITE);
229.                     g2d.setFont(new Font("SansSerif", Font.BOLD, 16));
230.                     FontMetrics fm = g2d.getFontMetrics();
231.                     int textWidth = fm.stringWidth(String.valueOf(cell));
232.                     int textHeight = fm.getAscent();
233.                     int x = j * cellSize + (cellSize - textWidth) / 2;
234.                     int y = i * cellSize + (cellSize + textHeight) / 2 -
fm.getDescent();
235.                     g2d.drawString(String.valueOf(cell), x, y);
236.                 }
237.             }
238.         }
239.         g2d.dispose();
240.
241.         try {
242.             fileName = fileName.substring(0, fileName.length()-4) + ".jpg";
243.             ImageIO.write(image, "png", new File(fileName));
244.             System.out.println("Gambar papan disimpan ke " + fileName);
245.         } catch (IOException e) {
246.             System.err.println("Error menyimpan gambar papan: " + e.getMessage());
247.         }
248.     }
249. }
250.
```

D. Parser

```
 1. import java.io.File;
 2. import java.io.FileNotFoundException;
 3. import java.util.*;
 4.
 5. // Parser - sebuah class untuk mengolah input.
 6. public class Parser {
 7.
 8.     private static char getFirstNonSpace(String line) {
 9.         for (int i = 0; i < line.length(); i++) {
10.             char ch = line.charAt(i);
11.             if (!Character.isWhitespace(ch)) {
12.                 return ch;
13.             }
14.         }
15.         return ' ';
16.     }
17.
18.     // Konversi dari teks ke blok.
```

```
    public static List<Block> textToBlocks(int P, List<String> allLines) {
        List<List<String>> blockGroups = new ArrayList<>();
        if (allLines.isEmpty()) {
            System.err.println("Tidak ada baris blok.");
            System.exit(1);
        }
        // Mulai dari baris pertama.
        List<String> currentGroup = new ArrayList<>();
        currentGroup.add(allLines.get(0));
        char currentKey = getFirstNonSpace(allLines.get(0));

        // Memroses baris-baris yang tersisa.
        for (int i = 1; i < allLines.size(); i++) {
            String line = allLines.get(i);
            if (line.isEmpty()) continue; // Melewati baris kosong.
            char lineKey = getFirstNonSpace(line);
            if (lineKey != currentKey) {
                blockGroups.add(currentGroup);
                currentGroup = new ArrayList<>();
                currentKey = lineKey;
            }
            currentGroup.add(line);
        }
        if (!currentGroup.isEmpty()) {
            blockGroups.add(currentGroup);
        }
        if (blockGroups.size() != P) {
            System.err.println("Jumlah blok yang seharusnya: " + P + ". Jumlah blok
yang ditemukan: " + blockGroups.size() + ".");
            System.exit(1);
        }
        List<Block> blocks = new ArrayList<>();
        for (List<String> group : blockGroups) {
            char id = getFirstNonSpace(group.get(0));
            Set<Ball> balls = new HashSet<>();
            for (int r = 0; r < group.size(); r++) {
                String row = group.get(r);
                for (int c = 0; c < row.length(); c++) {
                    if (row.charAt(c) == id) {
                        balls.add(new Ball(r, c, id));
                    }
                }
            }
            blocks.add(new Block(balls));
        }
        return blocks;
    }

    // Mengolah seluruh input.
    public static ParsedData parse(File inputFile) {
        ParsedData data = new ParsedData();
        try (Scanner scanner = new Scanner(inputFile)) {
            // Kasus membaca input kosong.
            if (!scanner.hasNextLine()) {
                System.err.println("File input kosong!");
                System.exit(1);
            }

            // Membaca ukuran papan dan jumlah blok.
            String headerLine = scanner.nextLine().trim();
            String[] headerParts = headerLine.split(" ");

            // Kasus baris pertama salah.
            if (headerParts.length != 3) {
                System.err.println("Baris N, M, dan P salah!");
                System.exit(1);
```

```
            }
            data.N = Integer.parseInt(headerParts[0]);
            data.M = Integer.parseInt(headerParts[1]);
            data.P = Integer.parseInt(headerParts[2]);

            // Membaca baris mode.
            data.mode = scanner.nextLine().trim();
            if (data.mode.equalsIgnoreCase("DEFAULT")) {

            }
            else if (data.mode.equalsIgnoreCase("CUSTOM")) {
                data.boardConfig = new ArrayList<>();
                // Membaca sebanyak N baris untuk konfigurasi papan.
                for (int i = 0; i < data.N; i++) {
                    if (scanner.hasNextLine()) {
                        data.boardConfig.add(scanner.nextLine());
                    }
                }
            }
            else {
                System.err.println("Mode tidak ada atau tidak valid!");
                System.exit(1);
            }

            List<String> blockLines = new ArrayList<>();
            while (scanner.hasNextLine()) {
                String line = scanner.nextLine();
                if (!line.isEmpty()) {
                    blockLines.add(line);
                }
            }
            data.blocks = textToBlocks(data.P, blockLines);
        } catch (FileNotFoundException e) {
            System.err.println("File tidak ditemukan!");
            System.exit(1);
        }
        return data;
    }
}

```

E. ParsedData

```
import java.util.List;

public class ParsedData {
    int N, M, P;
    String mode;
    List<String> boardConfig;
    List<Block> blocks;
}

```

F. MainCLI

```
import java.io.File;
import java.util.*;

public class MainCLI {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String fileName;

        File inputFile;
        while (true) {
            System.out.print("Masukkan nama file input: ");
```

```
            fileName = scanner.nextLine();
            inputFile = new File("../bin/input", fileName);

            if (inputFile.exists() && inputFile.isFile()) {
                break;
            }
            System.err.println("File '" + fileName + "' tidak ditemukan atau tidak
valid.");
        }

        ParsedData data = Parser.parse(inputFile);
        Board board = new Board(data.N, data.M);
        boolean solved;
        long startTime = System.nanoTime();

        if (data.mode.equalsIgnoreCase("CUSTOM")) {
            solved = board.solveCustom(data.blocks, data.boardConfig, 0);
        } else {
            solved = board.solveDefault(data.blocks, 0);
        }
        long endTime = System.nanoTime();
        double elapsedTime = (endTime - startTime) / 1_000_000.0;

        if (solved) {
            board.printBoard();
        } else {
            System.out.println("Tidak ada solusi.");
        }
        System.out.println("Waktu pencarian: " + elapsedTime + " ms");
        System.out.println("Jumlah iterasi: " + board.getIteration());

        System.out.print("Simpan hasil ke teks? (y/n)");
        String choice1 = scanner.nextLine().trim().toLowerCase();
        if (choice1.equals("y")) {
            board.saveBoardToFile("../test/result_" + fileName);
        }

        System.out.print("Simpan hasil sebagai gambar? (y/n)");
        String choice2 = scanner.nextLine().trim().toLowerCase();
        if (choice2.equals("y")) {
            board.saveBoardtoImage("../test/image_" + fileName);
        }


        scanner.close();
    }
}

```

G. MainGUI

```
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.stage.Stage;
import javafx.stage.FileChooser;
import javafx.scene.Scene;
import javafx.scene.layout.*;
import javafx.scene.control.*;
import javafx.geometry.*;
import javafx.scene.paint.Color;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import java.io.File;
```

```
14. import java.util.*;
15. import javafx.scene.text.Font;
16. 
17. public class MainGUI extends Application {
18.     @Override
19.     public void start(Stage primaryStage) {
20.         primaryStage.setTitle("");
21. 
22.         Button btnSelectFile = new Button("Select Puzzle File");
23.         GridPane boardPane = new GridPane();
24.         boardPane.setHgap(2);
25.         boardPane.setVgap(2);
26.         boardPane.setAlignment(Pos.CENTER);
27. 
28.         Label lblIterations = new Label("Iterations: ");
29.         Label lblTime = new Label("Time Elapsed: ");
30. 
31.         VBox root = new VBox(10);
32.         root.setPadding(new Insets(10));
33.         root.setAlignment(Pos.CENTER);
34. 
35.         Image logo = new Image("file:logo.png");
36.         ImageView logoView = new ImageView(logo);
37.         logoView.setFitWidth(320);
38.         logoView.setPreserveRatio(true);
39.         VBox.setMargin(logoView, new Insets(0, 0, 20, 0));
40.         root.getChildren().add(0, logoView);
41.         final Board[] board = {null};
42.         final String[] fileName = {null};
43. 
44.         Button btnSaveResultToText = new Button("Save Result to Text");
45.         btnSaveResultToText.setDisable(true);
46.         Button btnSaveResultToImage = new Button("Save Result to Image");
47.         btnSaveResultToImage.setDisable(true);
48. 
49.         btnSelectFile.setOnAction(new EventHandler<ActionEvent>() {
50.             @Override
51.             public void handle(ActionEvent e) {
52.                 FileChooser fileChooser = new FileChooser();
53.                 fileChooser.setTitle("Select Puzzle File");
54.                 fileChooser.getExtensionFilters().add(new
FileChooser.ExtensionFilter("Text Files", "*.txt"));
55.                 File selectedFile = fileChooser.showOpenDialog(primaryStage);
56.                 if (selectedFile != null) {
57.                     fileName[0] = selectedFile.getName();
58.                     ParsedData data = Parser.parse(selectedFile);
59.                     board[0] = new Board(data.N, data.M);
60.                     long startTime = System.nanoTime();
61.                     boolean solved;
62.                     if (data.mode.equalsIgnoreCase("CUSTOM")) {
63.                         solved = board[0].solveCustom(data.blocks, data.boardConfig,
0);
64.                     } else {
65.                         solved = board[0].solveDefault(data.blocks, 0);
66.                     }
67.                     long endTime = System.nanoTime();
68.                     double elapsedTime = (endTime - startTime) / 1_000_000.0;
69.                     lblTime.setText("Time Elapsed: " + elapsedTime + " ms");
70.                     lblIterations.setText("Iterations: " + board[0].getIteration());
71. 
72.                     if (solved) {
73.                         boardPane.getChildren().clear();
74.                         char[][] grid = board[0].getGrid();
75. 
76.                         Map<Character, Color> colorMap = new HashMap<>();
77. 
```

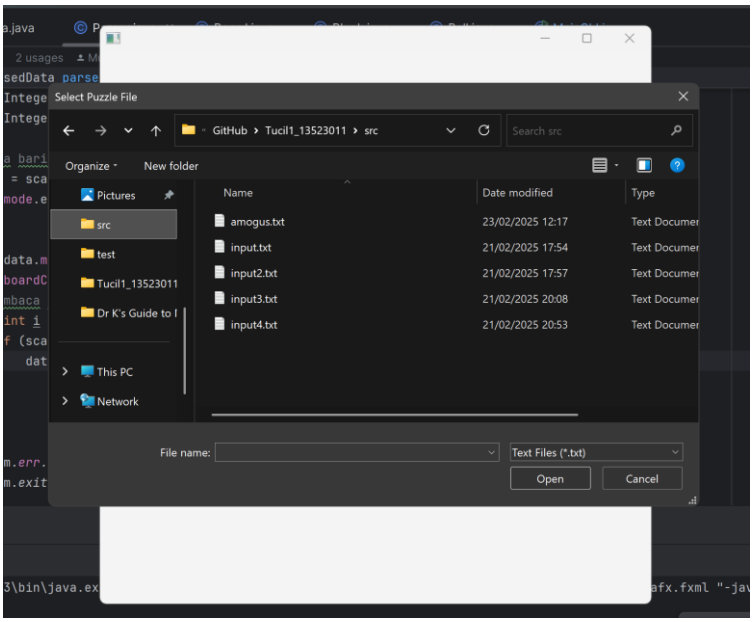
```
                        Color[] fixedColors = {
                                Color.RED,
                                Color.GREEN,
                                Color.YELLOW,
                                Color.BLUE,
                                Color.MAGENTA,
                                Color.CYAN,
                                Color.ORANGERED,
                                Color.LIGHTGREEN,
                                Color.LIGHTYELLOW,
                                Color.LIGHTBLUE,
                                Color.MAGENTA,
                                Color.LIGHTCYAN
                        };

                        for (int i = 0; i < 12; i++) {
                            char letter = (char) ('A' + i);
                            colorMap.put(letter, fixedColors[i]);
                        }

                        int remaining = 14;
                        for (int i = 0; i < remaining; i++) {
                            char letter = (char) ('A' + 12 + i);
                            double hue = (i * 360.0) / remaining;
                            Color generated = Color.hsb(hue, 0.8, 0.8);
                            colorMap.put(letter, generated);
                        }

                        for (int i = 0; i < data.N; i++) {
                            for (int j = 0; j < data.M; j++) {
                                char cell = grid[i][j];
                                Label lbl = new Label(String.valueOf(cell));
                                lbl.setMinSize(30, 30);
                                lbl.setAlignment(Pos.CENTER);
                                lbl.setFont(Font.font(14));
                                if (cell == '0') {
                                    lbl.setStyle("-fx-background-color: white; -fx-
border-color: black;");
                                } else if (cell == '.') {
                                    lbl.setStyle("-fx-background-color: gray; -fx-
border-color: black;");
                                } else {
                                    Color col = colorMap.getOrDefault(cell,
Color.LIGHTGRAY);
                                    String hex = String.format("#%02X%02X%02X",
                                            (int)(col.getRed()*255),
                                            (int)(col.getGreen()*255),
                                            (int)(col.getBlue()*255));
                                    lbl.setStyle("-fx-background-color: " + hex + "; -
fx-border-color: black;");
                                }
                                boardPane.add(lbl, j, i);
                            }
                            btnSaveResultToText.setDisable(false);
                            btnSaveResultToImage.setDisable(false);
                        }
                    } else {
                        Alert alert = new Alert(Alert.AlertType.ERROR);
                        alert.setTitle("No Solution");
                        alert.setHeaderText(null);
                        alert.setContentText("Tidak ada solusi yang ditemukan!");
                        alert.showAndWait();
                    }
                }
            }
        });
```

```
140.
141.         btnSaveResultToText.setOnAction(new EventHandler<ActionEvent>() {
142.             @Override
143.             public void handle(ActionEvent e) {
144.                 board[0].saveBoardToFile("../test/result_" + fileName[0]);
145.                 Alert alert = new Alert(Alert.AlertType.INFORMATION);
146.                 alert.setTitle("Saved");
147.                 alert.setContentText("Solution text saved in test.");
148.                 alert.showAndWait();
149.             }
150.         });
151.
152.         btnSaveResultToImage.setOnAction(new EventHandler<ActionEvent>() {
153.             @Override
154.             public void handle(ActionEvent e) {
155.                 board[0].saveBoardtoImage("../test/image_" +
fileName[0].replace(".txt", ".jpg"));
156.                 Alert alert = new Alert(Alert.AlertType.INFORMATION);
157.                 alert.setTitle("Saved");
158.                 alert.setContentText("Image saved in test.");
159.                 alert.showAndWait();
160.             }
161.         });
162.
163.         root.getChildren().addAll(btnSelectFile, boardPane, lblIterations, lblTime,
btnSaveResultToText, btnSaveResultToImage);
164.         Scene scene = new Scene(root, 600, 600);
165.         primaryStage.setScene(scene);
166.         primaryStage.show();
167.     }
168.
169.     public static void main(String[] args) {
170.         launch(args);
171.     }
172. }
173.
```

BAB IV
PENGUJIAN

A. Pengujian Input/Output File

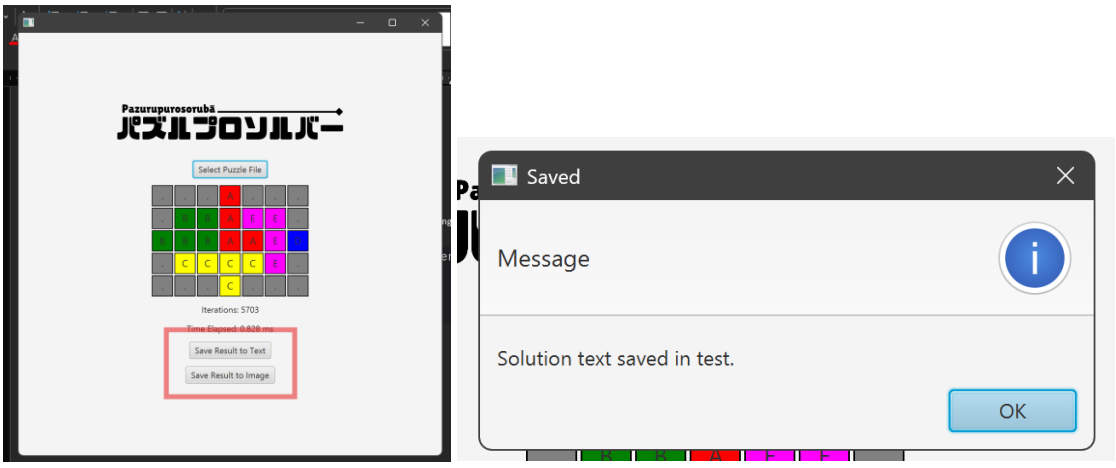
Untuk memasukkan input melalui GUI, pengguna harus memilih file melalui browser.



Sementara untuk memasukkan input melalui CLI, pengguna harus memasukkan nama file yang terletak di /bin/input.

```
"C:\Program Files\Java\jdk-23\bin\java.exe" "-javaagen
Masukkan nama file input:
```

Pengguna dapat menyimpan gambar atau teks dengan menekan tombol Save Result to Text atau Save Result to Image. Hasil akan tersimpan di folder test.

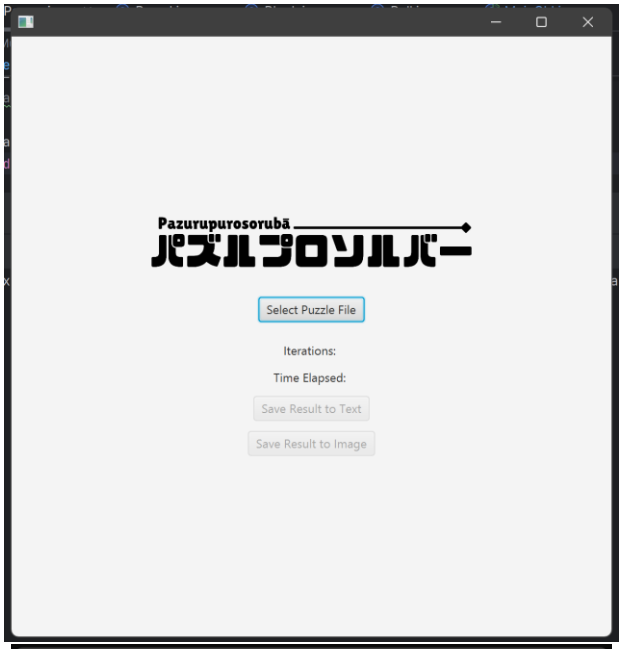


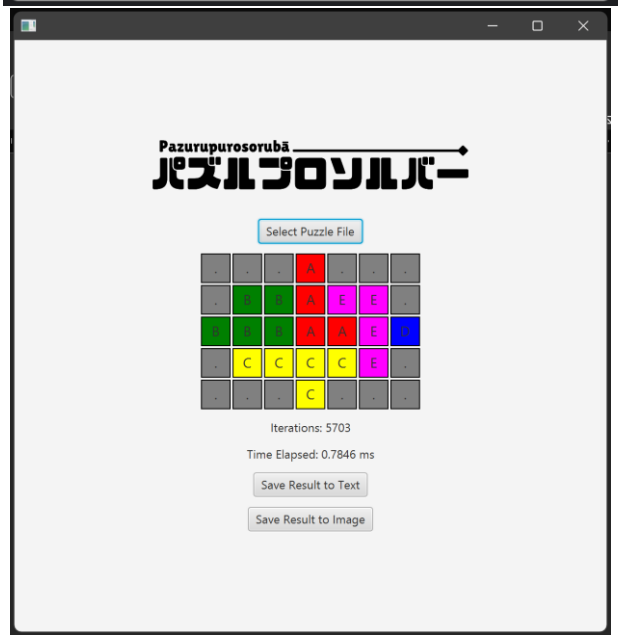
Sementara di CLI, pengguna dapat memasukkan perintah ‘y’ jika prompt untuk menyimpan teks atau gambar muncul.

```
Simpan hasil ke teks? (y/n)y
Solusi disimpan ke: ../test/result_input.txt
Simpan hasil sebagai gambar? (y/n)y
Gambar papan disimpan ke ../test/image_input.jpg
```

B. Pengujian Graphical User Interface (GUI)

Ada tiga tombol dalam GUI, yaitu Select File yang tersedia sejak program dijalankan, dan Save Result to Text dan Save Result to Image yang baru tersedia setelah program melakukan pemecahan papan. Setelah pengguna memilih file .txt masukan, GUI akan menampilkan solusi



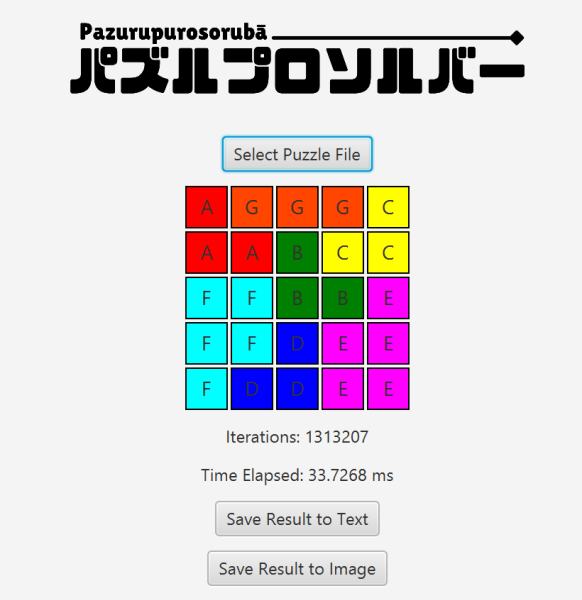


C. Pengujian Algoritma

Berikut adalah pengujian dari tujuh test case.

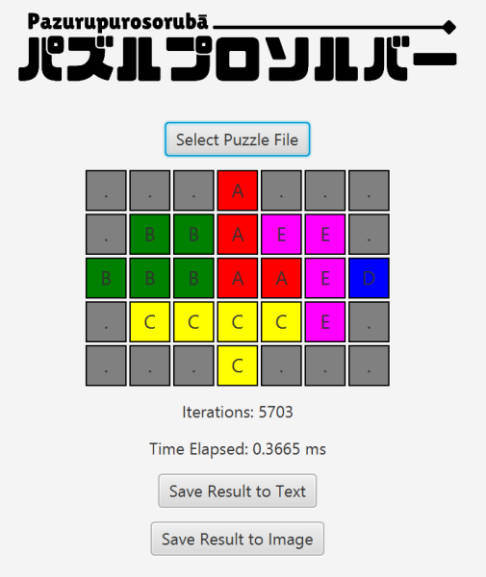
1. 5 5 7 DEFAULT

```
5 5 7
DEFAULT
A
AA
B
BB
C
CC
D
DD
EE
EE
E
FF
FF
F
GGG
```



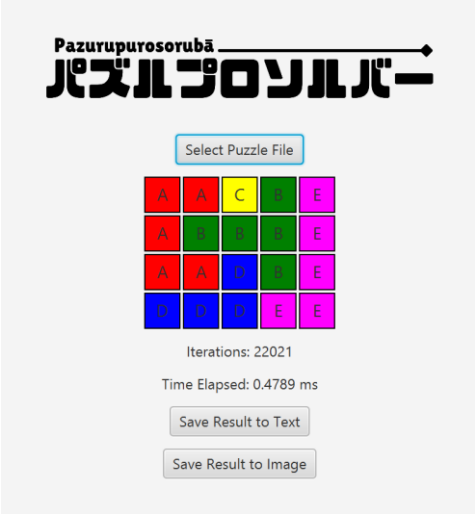
2. 5 7 5 CUSTOM

```
5 7 5
CUSTOM
...X...
.XXXXX.
XXXXXXX
.XXXXX.
...X...
A
AAA
BB
BBB
CCCC
 C
D
EEE
E

```



3. 4 5 5 DEFAULT

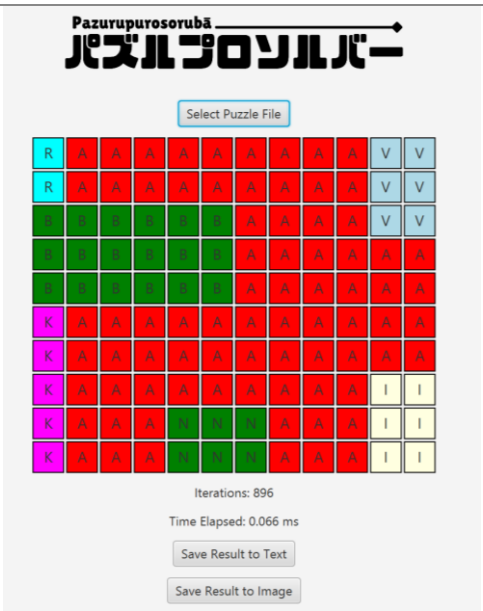
```
4 5 5
DEFAULT
AA
A
AA
B
BBB
B
C
D
D
DD
E
EEEE
```



4. 10 12 8 DEFAULT

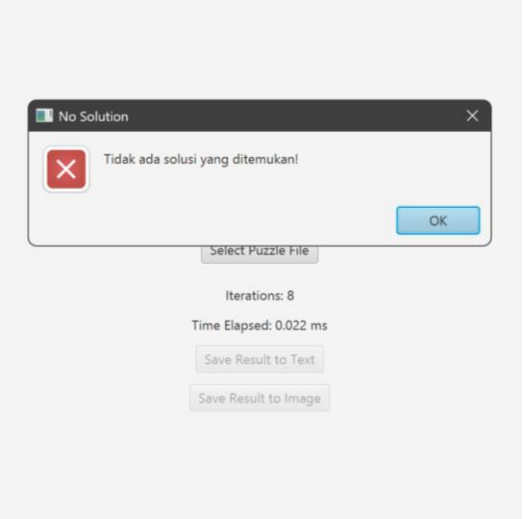
```
10 12 7
DEFAULT
 AAAAAAAAA
 AAAAAAAAA
```

```
 5.        AAAA
 6.        AAAAAA
 7.        AAAAAA
 8.  AAAAAAAAAAA
 9.  AAAAAAAAAAA
10.  AAAAAAAAA
11.  AAA    AAA
12.  AAA    AAA
13. BBBBBB
14. BBBBBB
15. BBBBBB
16. RR
17. KKKKK
18. VVV
19. VVV
20. III
21. III
22. NNN
23. NNN
24.
```



5. 2 2 1 DEFAULT

```
1. 2 2 1
2. DEFAULT
3. A
4. A
5. A
6. A
```

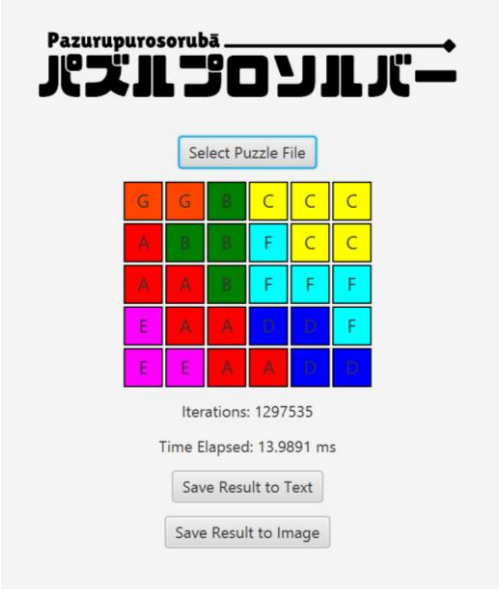


6. 5 9 11 DEFAULT

```
 1. 5 9 11
 2. DEFAULT
 3. AAA
 4.   A
 5.   A
 6.  B
 7. BB
 8. B
 9. B
10. CC
11.  CC
12. DD
13.  DD
14.   D
15. EE
16. E
17. E
18. FF
19. F
20. G
21. G
22. G
23. G
24. HH
25. H
26. I
27. II
28. II
29. JJ
30. KKKKK
```

7. 5 6 7 DEFAULT

```
 1. 5 6 7
 2. DEFAULT
 3. AA
 4.  AA
 5.   AA
 6.    A
 7. B
 8. BB
 9. B
10. C
11. CC
12. CC
13. D
14. DD
15.  D
16. EE
17. E
18. FF
19.  F
20.  FF
21. GG
```



LAMPIRAN

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

Link Repository: https://github.com/mraifalkautsar/Tucil1_13523011