LAPORAN TUGAS KECIL STRATEGI ALGORITMA: PENYELESAIAN IQ PUZZLER PRO DENGAN ALGORITMA BRUTE FORCE

Laporan untuk memenuhi tugas besar mata kuliah IF2211 Strategi Algoritma

Disusun Oleh:

M Abizzar Gamadrian – 13523155

INSTITUT TEKNOLOGI BANDUNG

2025

Deskripsi Singkat Program

Program ini merupakan implementasi dari penyelesaian untuk permainan IQ Puzzler Pro menggunakan algoritma Brute Force. IQ Puzzler Pro adalah permainan puzzle yang terdiri dari sebuauh papan dan beberapa blok puzzle dengan bentuk-bentuk yang unik. Tujuan dari permainan ini sendiri adalah untuk mengisi seluruh area permukaan papan menggunakan blok-blok puzzle yang tersedia.

Program ini dibuat dengan menggunakan Bahasa Pemrogramman Java yang dilengkapi dengan fitur-fitur berikut:

- Membaca konfigurasi puzzle dari file teks
- Mencari Solusi menggunakan algoritma Brute Force (murni tanpa heuristic)
- Menampilkan Solusi dengan visualisasi berwarna
- Menyimpan Solusi dalam format teks dan gambar (bonus)
- Graphical User Interface untuk kemudahan menavigasi program (bonus)

Langkah-Langkah Algoritma

Algoritma Brute Force yang diimplementasikan mengikuti Langkah-langkah sebagai berikut:

1. **Inisialisasi**
    - Membaca dimensi papan (NxM) dan jumlah piece (P) sesuai dengan format yang ada di spesifikasi tugas kecil.
    - Membaca bentuk setiap piece dari file input.
    - Membuat pan kosong dengan ukuran NxM
2. **Proses Pencarian Solusi**
    - Mencari posisi kosong pertama pada papan
    - Untuk setiap piece yang tersisa:
        a) Generate semua kemungkinan orientasi piece (semua bentuk rotasi dan pencerminan)
        b) Untuk setiap orientasi:
            1) Cek apakah piece bisa ditempatkan pada posisi kosong
            2) Jika bisa, letakkan piece dan lanjut ke posisi kosong berikutnya
            3) Jika tidak bisa, coba orientasi yang lain dan jika tidak ada lanjut ke piece berikutnya
    - Jika semua pice sudah digunakan dan papan terisi penuh, Solusi ditemukan

- Jika tidak ada piece yang bisa ditempatkan, backtrack dan coba kombinasi lain

Visualisasi dan Output

- Menampilkan konfigurasi papan dengan warna berbeda untuk setiap piece
- Menampilkan statistic waktu eksekusi dan jumlah iterasi
- Menyimpan Solusi ke file jika diminta

Struktur Program

Package `model`

- **`Board.java`: Representasi papan permainan**

  Source code

```java
// Class untuk merepresentasikan papan puzzle, termasuk logika penempatan dan pengecekan piece

package model;

import java.util.Arrays;

public class Board {
    private int rows;
    private int cols;
    private char[][] grid;

    public Board(int rows, int cols) {
        this.rows = rows;
        this.cols = cols;
        this.grid = new char[rows][cols];
        // Inisialisasi grid kosong dengan '.'
        for (int i = 0; i < rows; i++) {
            Arrays.fill(grid[i], '.');
        }
    }

    // Mengcopy board
    public Board(Board other) {
        this.rows = other.rows;
        this.cols = other.cols;
        this.grid = new char[rows][cols];
        for (int i = 0; i < rows; i++) {
            System.arraycopy(other.grid[i], 0, this.grid[i], 0, cols);
        }
```

```
    }

    // Cek apakah piece bisa ditempatkan pada posisi tertentu
    public boolean canPlacePiece(Piece piece, Position pos) {
        boolean[][] shape = piece.getShape();
        for (int i = 0; i < piece.getHeight(); i++) {
            for (int j = 0; j < piece.getWidth(); j++) {
                if (shape[i][j]) {
                    int newRow = pos.getRow() + i;
                    int newCol = pos.getCol() + j;

                    // Cek batas batas board
                    if (newRow >= rows || newCol >= cols || newRow < 0 || newCol < 0) {
                        return false;
                    }
                    // Cek tumpang tindih antar piece
                    if (grid[newRow][newCol] != '.') {
                        return false;
                    }
                }
            }
        }
        return true;
    }

    // Menempatkan piece pada posisi yang sesuai
    public void placePiece(Piece piece, Position pos) {
        boolean[][] shape = piece.getShape();
        for (int i = 0; i < piece.getHeight(); i++) {
            for (int j = 0; j < piece.getWidth(); j++) {
                if (shape[i][j]) {
                    grid[pos.getRow() + i][pos.getCol() + j] = piece.getId();
                }
            }
        }
    }

    // Menghapus piece dari posisi tertentu
    public void removePiece(Piece piece, Position pos) {
        boolean[][] shape = piece.getShape();
        for (int i = 0; i < piece.getHeight(); i++) {
            for (int j = 0; j < piece.getWidth(); j++) {
                if (shape[i][j]) {
                    grid[pos.getRow() + i][pos.getCol() + j] = '.';
                }
            }
        }
    }
}
```

```
    // Mencari posisi kosong yang pertama
    public Position findFirstEmpty() {
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                if (grid[i][j] == '.') {
                    return new Position(i, j);
                }
            }
        }
        return null;
    }

    // Cek apakah board sudah terisi penuh
    public boolean isFull() {
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                if (grid[i][j] == '.') {
                    return false;
                }
            }
        }
        return true;
    }

    // Mengembalikan jumlah baris
    public int getRows() {
        return rows;
    }

    // Mengembalikan jumlah kolom
    public int getCols() {
        return cols;
    }

    // Mengembalikan array 2D representasi papan
    public char[][] getGrid() {
        return grid;
    }
}
```

- **`Piece.java`: Representasi piece puzzle**

  Source Code

```
// Class untuk merepresentasikan piece puzzle, termasuk rotasi dan pencerminan bentuk piece
```

```java
package model;

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

public class Piece {
    private char id;
    private boolean[][] shape;
    private int height;
    private int width;

    public Piece(char id, boolean[][] shape) {
        this.id = id;
        this.shape = shape;
        this.height = shape.length;
        this.width = shape[0].length;
    }

    // Mengcopy piece yang ada
    public Piece(Piece other) {
        this.id = other.id;
        this.height = other.height;
        this.width = other.width;
        this.shape = new boolean[height][width];
        for (int i = 0; i < height; i++) {
            System.arraycopy(other.shape[i], 0, this.shape[i], 0, width);
        }
    }

    // Rotasi 90 derajat searah jarum jam
    public Piece rotate() {
        boolean[][] newShape = new boolean[width][height];
        for (int i = 0; i < height; i++) {
            for (int j = 0; j < width; j++) {
                newShape[j][height - 1 - i] = shape[i][j];
            }
        }
        return new Piece(id, newShape);
    }

    // Pencerminan horizontal
    public Piece flipHorizontal() {
        boolean[][] newShape = new boolean[height][width];
        for (int i = 0; i < height; i++) {
            for (int j = 0; j < width; j++) {
```

```
                newShape[i][width - 1 - j] = shape[i][j];
            }
        }
        return new Piece(id, newShape);
    }

    // Mendapatkan semua kemungkinan orientasi piece
    public List<Piece> getAllOrientations() {
        Set<String> uniqueOrientations = new HashSet<>();
        List<Piece> orientations = new ArrayList<>();

        Piece current = this;
        // 4 rotasi
        for (int i = 0; i < 4; i++) {
            // Original dan flip horizontal untuk setiap rotasi
            Piece flipped = current.flipHorizontal();
            addUniqueOrientation(current, uniqueOrientations, orientations);
            addUniqueOrientation(flipped, uniqueOrientations, orientations);
            current = current.rotate();
        }

        return orientations;
    }

    // Menambahkan orientasi piece yang unik ke dalam list
    // Menggunakan Set untuk mengecek duplikat berdasarkan representasi string piece
    private void addUniqueOrientation(Piece piece, Set<String> uniqueOrientations, List<Piece>
orientations) {
        String key = piece.toString();
        if (uniqueOrientations.add(key)) {
            orientations.add(piece);
        }
    }

    // Getter untuk ID huruf piece (A-Z)
    public char getId() {
        return id;
    }

    // Getter untuk array 2D yang merepresentasikan bentuk piece
    // true = ada piece, false = kosong
    public boolean[][] getShape() {
        return shape;
    }

    // Getter untuk tinggi piece (jumlah baris)
    public int getHeight() {
        return height;
```

```
    }

    // Getter untuk lebar piece (jumlah kolom)
    public int getWidth() {
        return width;
    }
}
```

- **`Position.java`: Representasi posisi pada papan**

  Source code:

```
// Class untuk menyimpan koordinat posisi (row,col) pada papan puzzle

package model;

public class Position {
    private int row;
    private int col;

    public Position(int row, int col) {
        this.row = row;
        this.col = col;
    }

    // Mengembalikan nilai baris
    public int getRow() {
        return row;
    }

    // Mengembalikan nilai kolom
    public int getCol() {
        return col;
    }

    // Override metode toString untuk membuat format print yang sesuai
    @Override
    public String toString() {
        return "Position(" + row + ", " + col + ")";
    }
}
```

- **`PuzzleData.java`: Data konfigurasi puzzle**
  Source code:

```java
// Class untuk menyimpan data puzzle yang dibaca dari file input (ukuran board dan piece)

package model;

import java.util.List;

public class PuzzleData {
    private final int rows;
    private final int cols;
    private final int pieceCount;
    private final String caseType;
    private final List<Piece> pieces;

    // Constructor untuk membuat objek PuzzleData dengan parameter yang diberikan
    public PuzzleData(int rows, int cols, int pieceCount, String caseType, List<Piece> pieces) {
        this.rows = rows;
        this.cols = cols;
        this.pieceCount = pieceCount;
        this.caseType = caseType;
        this.pieces = pieces;
    }

    // Getter untuk jumlah baris board
    public int getRows() {
        return rows;
    }

    // Getter untuk jumlah kolom board
    public int getCols() {
        return cols;
    }

    // Getter untuk jumlah total piece puzzle
    public int getPieceCount() {
        return pieceCount;
    }

    // Getter untuk tipe kasus puzzle
    public String getCaseType() {
        return caseType;
    }

    // Getter untuk daftar piece puzzle
    public List<Piece> getPieces() {
        return pieces;
    }
```

```
}
```

- **`Solution.java`: Representasi solusi puzzle**

  Source code:

```java
// Class untuk menyimpan solusi puzzle yang ditemukan beserta statistik penyelesaiannya

package model;

import java.util.ArrayList;
import java.util.List;

public class Solution {
    private Board board;
    private List<Piece> usedPieces;
    private long executionTime;
    private long iterationCount;

    // Setter untuk mengupdate board dengan deep copy
    public void setBoard(Board board) {
        this.board = new Board(board);
    }

    // Constructor untuk membuat objek Solution dengan board awal
    public Solution(Board board) {
        this.board = new Board(board);
        this.usedPieces = new ArrayList<>();
        this.executionTime = 0;
        this.iterationCount = 0;
    }

    // Menambahkan piece yang digunakan ke dalam solusi
    public void addPiece(Piece piece) {
        usedPieces.add(new Piece(piece));
    }

    // Setter untuk waktu eksekusi
    public void setExecutionTime(long time) {
        this.executionTime = time;
    }

    // Setter untuk jumlah iterasi
    public void setIterationCount(long count) {
        this.iterationCount = count;
    }
```

```
    // Getter untuk board solusi
    public Board getBoard() {
        return board;
    }

    // Getter untuk daftar piece yang digunakan
    public List<Piece> getUsedPieces() {
        return usedPieces;
    }

    // Getter untuk waktu eksekusi
    public long getExecutionTime() {
        return executionTime;
    }

    // Getter untuk jumlah iterasi
    public long getIterationCount() {
        return iterationCount;
    }

}
```

Package `solver`

- **`BruteForceSolver.java`: Implementasi algoritma Brute Force**

Source code:

```
// Class implementasi algoritma brute force untuk mencari solusi puzzle

package solver;

import model.*;
import java.util.*;
import java.util.function.Supplier;

import javax.swing.SwingUtilities;

public class BruteForceSolver {
    private Board board;
    private List<Piece> pieces;
    private long startTime;
    private long iterationCount;
    private Solution solution;
    private SolverVisualizer visualizer;
```

```
    private static final int VISUALIZATION_DELAY = 100; // dalam ms
    private Supplier<Boolean> stopCheck;

    // Konstruktor untuk inisialisasi solver
    public BruteForceSolver(int rows, int cols, List<Piece> pieces,
            SolverVisualizer visualizer,
            Supplier<Boolean> stopCheck) {
        this.board = new Board(rows, cols);
        this.pieces = new ArrayList<>(pieces);
        this.iterationCount = 0;
        this.visualizer = visualizer;
        this.stopCheck = stopCheck;
    }

    // Fungsi utama untuk mencari solusi puzzle
    public Solution solve() {
        startTime = System.currentTimeMillis();
        solution = new Solution(board);

        // Menampilkan piece yang akan digunakan (untuk debugging)
        System.out.println("\nPiece yang akan digunakan:");
        for (Piece p : pieces) {
            System.out.println("Piece " + p.getId() + " Dimensi: " + p.getHeight() + "x" +
p.getWidth());
        }

        boolean found = solveRecursive(new ArrayList<>(pieces));

        if (found) {
            solution.setExecutionTime(System.currentTimeMillis() - startTime);
            solution.setIterationCount(iterationCount);
            return solution;
        }
        return null;
    }

    // Fungsi rekursif yang mencoba setiap kemungkinan penempatan piece
    private boolean solveRecursive(List<Piece> remainingPieces) {
        if (stopCheck.get()) {
            return false;
        }

        iterationCount++;

        // Update visualisasi setiap 10 iterasi
        if (iterationCount % 10 == 0 && visualizer != null) {
            SwingUtilities.invokeLater(() -> visualizer.updateVisualization(new Board(board)));
            try {
```

```
                Thread.sleep(VISUALIZATION_DELAY);
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }

        // Menampilkan status setiap 100 iterasi (untuk debugging)
        if (iterationCount % 100 == 0) {
            System.out.println("Pengukangan: " + iterationCount);
            System.out.println("Piece yang tersisa: " + remainingPieces.size());
            System.out.println("Status papan sekarang:");
            printBoard();
        }

        // Basis Rekursif: cek apakah semua piece sudah digunakan dan board terisi penuh
        if (remainingPieces.isEmpty()) {
            boolean isFull = board.isFull();
            System.out.println("Mengecek apakah papannya penuh: " + isFull);
            return isFull;
        }

        // Cari posisi kosong pertama
        Position emptyPos = board.findFirstEmpty();
        if (emptyPos == null) {
            System.out.println("Tidak ada posisi kosong");
            return false;
        }

        // Debug print
        System.out.println("Mencoba posisi: " + emptyPos.getRow() + "," + emptyPos.getCol());

        // Mencooba setiap piece yang tersisa
        for (int i = 0; i < remainingPieces.size(); i++) {
            Piece currentPiece = remainingPieces.get(i);

            // Mencoba setiap orientasi piece
            List<Piece> orientations = currentPiece.getAllOrientations();
            System.out
                    .println("Mencoba piece " + currentPiece.getId() + " dengan " +
orientations.size() + " orientasi");

            for (Piece orientation : orientations) {
                if (board.canPlacePiece(orientation, emptyPos)) {
                    // Letakkan piece di board
                    board.placePiece(orientation, emptyPos);
                    if (visualizer != null) {
                        SwingUtilities.invokeLater(() -> visualizer.updateVisualization(new
Board(board)));
```

```
                    }
                    // Hapus piece yang sudah digunakan dalam daftar
                    List<Piece> newRemaining = new ArrayList<>(remainingPieces);
                    newRemaining.remove(i);

                    // Rekursif untuk piece selanjutnya
                    if (solveRecursive(newRemaining)) {
                        solution.addPiece(orientation);
                        solution.setBoard(new Board(board)); // Simpan state board saat ini
                        return true;
                    }

                    // Backtracking ke posisi sebelumnya jika solusi belum ditemukan
                    board.removePiece(orientation, emptyPos);
                    if (visualizer != null) {
                        SwingUtilities.invokeLater(() -> visualizer.updateVisualization(new
Board(board)));
                    }
                }
            }
        }

        return false;
    }

    private void printBoard() {
        char[][] grid = board.getGrid();
        for (int i = 0; i < board.getRows(); i++) {
            for (int j = 0; j < board.getCols(); j++) {
                System.out.print(grid[i][j] + " ");
            }
            System.out.println();
        }
    }

}
```

- **`SolverVisualizer.java`: Interface untuk visualisasi proses solving**

  Source code:

```
// Interface untuk visualisasi proses solving puzzle secara real-time

package solver;
```

```
import model.Board;

public interface SolverVisualizer {
    void updateVisualization(Board board);
}
```

Package `util`

- **`ImageGenerator.java`: Generator gambar untuk output**

  Source code:

```
// Class untuk menghasilkan gambar PNG dari solusi puzzle

package util;

import model.*;
import java.awt.*;
import java.awt.image.*;
import javax.imageio.*;
import java.io.*;

public class ImageGenerator {
    // Konstanta untuk ukuran gambar
    private static final int CELL_SIZE = 50; // Ukuran setiap sel dalam pixel
    private static final int BORDER = 2; // Ukuran border antar sel
    private static final int PADDING = 20; // Padding di sekitar board

    // Warna untuk setiap piece (A-Z)
    private static final Color[] PIECE_COLORS = {
            new Color(231, 76, 60), // A - Merah terang
            new Color(46, 204, 113), // B - Hijau muda fresh
            new Color(52, 152, 219), // C - Biru laut
            new Color(155, 89, 182), // D - Ungu anggur
            new Color(241, 196, 15), // E - Kuning cerah
            new Color(230, 126, 34), // F - Orange segar
            new Color(26, 188, 156), // G - Turquoise
            new Color(52, 73, 94), // H - Abu gelap
            new Color(192, 57, 43), // I - Merah tua
            new Color(39, 174, 96), // J - Hijau daun
            new Color(41, 128, 185), // K - Biru medium
            new Color(142, 68, 173), // L - Ungu royal
            new Color(243, 156, 18), // M - Orange muda
            new Color(211, 84, 0), // N - Orange tua
            new Color(22, 160, 133), // O - Tosca gelap
            new Color(44, 62, 80), // P - Hitam keabu
```

```
            new Color(255, 99, 71), // Q - Salmon
            new Color(50, 205, 50), // R - Lime green
            new Color(30, 144, 255), // S - Dodger blue
            new Color(138, 43, 226), // T - Blue violet
            new Color(255, 215, 0), // U - Golden
            new Color(255, 140, 0), // V - Dark orange
            new Color(64, 224, 208), // W - Turquoise light
            new Color(119, 136, 153), // X - Slate gray
            new Color(220, 20, 60), // Y - Crimson
            new Color(34, 139, 34) // Z - Forest green
    };

    // Menyimpan solusi puzzle sebagai gambar
    public static void saveToImage(Solution solution, String filename) throws IOException {
        Board board = solution.getBoard();
        int width = (board.getCols() * (CELL_SIZE + BORDER)) + (2 * PADDING);
        int height = (board.getRows() * (CELL_SIZE + BORDER)) + (2 * PADDING);

        // Buat gambar baru dengan kualitas render yang bagus
        BufferedImage image = new BufferedImage(width, height, BufferedImage.TYPE_INT_ARGB);
        Graphics2D g2d = image.createGraphics();

        try {
            // Set background putih
            g2d.setColor(Color.WHITE);
            g2d.fillRect(0, 0, width, height);

            // Set pengaturan rendering untuk hasil yang lebih baik
            g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);
            g2d.setRenderingHint(RenderingHints.KEY_RENDERING, RenderingHints.VALUE_RENDER_QUALITY);

            // Gambar board
            char[][] grid = board.getGrid();
            for (int row = 0; row < board.getRows(); row++) {
                for (int col = 0; col < board.getCols(); col++) {
                    int x = PADDING + (col * (CELL_SIZE + BORDER));
                    int y = PADDING + (row * (CELL_SIZE + BORDER));

                    char piece = grid[row][col];
                    if (piece != '.') {
                        // Gambar piece
                        g2d.setColor(getPieceColor(piece));
                        g2d.fillRect(x, y, CELL_SIZE, CELL_SIZE);

                        // Tambah label
                        g2d.setColor(Color.WHITE);
                        g2d.setFont(new Font("Arial", Font.BOLD, CELL_SIZE / 2));
                        FontMetrics metrics = g2d.getFontMetrics();
```

```
                        String text = String.valueOf(piece);
                        int textX = x + (CELL_SIZE - metrics.stringWidth(text)) / 2;
                        int textY = y + ((CELL_SIZE + metrics.getAscent()) / 2);
                        g2d.drawString(text, textX, textY);
                    } else {
                        // Gambar cell kosong
                        g2d.setColor(Color.LIGHT_GRAY);
                        g2d.fillRect(x, y, CELL_SIZE, CELL_SIZE);
                    }

                    // Gambar border
                    g2d.setColor(Color.BLACK);
                    g2d.drawRect(x, y, CELL_SIZE, CELL_SIZE);
                }
            }

            // Tambah informasi statistik
            g2d.setColor(Color.BLACK);
            g2d.setFont(new Font("Arial", Font.PLAIN, 12));
            String stats = String.format("Waktu: %d ms | Pengulangan: %d",
                    solution.getExecutionTime(), solution.getIterationCount());
            g2d.drawString(stats, PADDING, height - PADDING / 2);

            // Simpan gambar
            File outputFile = new File(filename);
            boolean success = ImageIO.write(image, "PNG", outputFile);

            if (!success) {
                throw new IOException("Tidak bisa menyimpan gambar ke " + filename);
            }

            System.out.println("Gambar berhasil disimpan ke: " + outputFile.getAbsolutePath());

        } finally {
            g2d.dispose();
        }
    }

    private static Color getPieceColor(char piece) {
        int index = piece - 'A';
        if (index >= 0 && index < PIECE_COLORS.length) {
            return PIECE_COLORS[index];
        }
        return Color.GRAY;
    }
}
```

- **`PuzzleReader.java`: Pembaca file input puzzle**

  Source Code:

```java
// Class untuk membaca dan memparse file input puzzle

package util;

import model.*;
import java.io.*;
import java.util.*;

public class PuzzleReader {
    // Membaca data puzzle dari file dan mengubahnya menjadi objek PuzzleData
    public static PuzzleData readFromFile(String filename) throws IOException {
        try (BufferedReader reader = new BufferedReader(new FileReader(filename))) {
            System.out.println("Membaca file: " + filename);

            // Baca dimensi board dan jumlah piece dari baris pertama
            String firstLine = reader.readLine();
            System.out.println("Line pertama: " + firstLine);
            String[] dimensions = firstLine.split(" ");
            int N = Integer.parseInt(dimensions[0]);
            int M = Integer.parseInt(dimensions[1]);
            int P = Integer.parseInt(dimensions[2]);

            // Baca tipe kasus puzzle (DEFAULT/CUSTOM/PYRAMID) *untuk program ini hanya
            // menggunakan Default
            String caseType = reader.readLine();
            System.out.println("Tipe Puzzle: " + caseType);

            // Baca data piece satu per satu
            List<Piece> pieces = new ArrayList<>();
            List<String> currentPieceLines = new ArrayList<>();
            char currentId = 'A';

            String line;
            while ((line = reader.readLine()) != null && pieces.size() < P) {
                if (!line.trim().isEmpty()) {
                    // Jika baris berisi karakter berbeda dengan ID saat ini,
                    // berarti ini piece baru
                    if (!line.contains(String.valueOf(currentId)) && !currentPieceLines.isEmpty()) {
                        System.out.println("Membuat piece " + currentId + ":");
                        for (String l : currentPieceLines) {
                            System.out.println("\t" + l);
                        }
                        pieces.add(createPiece(currentId, currentPieceLines));
                        currentPieceLines = new ArrayList<>();
```

```
                        currentId++;
                    }
                    currentPieceLines.add(line);
                }
            }

            // Proses piece terakhir jika masih ada
            if (!currentPieceLines.isEmpty()) {
                System.out.println("Membuat piece " + currentId + ":");
                for (String l : currentPieceLines) {
                    System.out.println("\t" + l);
                }
                pieces.add(createPiece(currentId, currentPieceLines));
            }

            return new PuzzleData(N, M, P, caseType, pieces);
        }
    }

    // Membuat objek Piece dari data teks yang dibaca
    private static Piece createPiece(char id, List<String> lines) {
        if (lines.isEmpty()) {
            throw new IllegalArgumentException("Data piece kosong untuk piece " + id);
        }

        // Menentukan dimensi piece
        int height = lines.size();
        int width = lines.stream().mapToInt(String::length).max().orElse(0);
        boolean[][] shape = new boolean[height][width];

        System.out.println("Membuat piece " + id + " dengan dimensi " + height + "x" + width);
        for (int i = 0; i < height; i++) {
            String line = lines.get(i);
            System.out.println("Memproses line: " + line);
            for (int j = 0; j < line.length(); j++) {
                shape[i][j] = line.charAt(j) == id;
            }
        }

        return new Piece(id, shape);
    }
}
```

GUI Components

- **`App.java`: Entry point aplikasi**

Source Code:

```java
// File utama yang berisi main method untuk menjalankan aplikasi GUI IQ Puzzler Pro Solver

import javax.swing.*;

public class App {
    public static void main(String[] args) {
        // Mengatur tampilan GUI agar sesuai dengan sistem operasi yang digunakan
        try {
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        } catch (Exception e) {
            e.printStackTrace();
        }

        // Membuat dan menampilkan GUI
        SwingUtilities.invokeLater(() -> {
            MainWindow mainWindow = new MainWindow();
            mainWindow.setVisible(true);
        });
    }
}
```

- **`MainWindow.java`: Jendela utama aplikasi**

  Source code:

```java
// Class untuk implementasi GUI utama yang menampilkan puzzle board dan mengontrol interaksi dengan
user

import javax.swing.*;
import javax.swing.filechooser.FileNameExtensionFilter;
import java.awt.*;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;

import model.*;
import solver.*;
import util.*;

public class MainWindow extends JFrame implements SolverVisualizer {
    // Komponen GUI
    private JPanel boardPanel; // Panel untuk papan puzzle
    private JTextField filePathField; // Field untuk path file input
```

```
    private JLabel timeLabel; // Label waktu eksekusi
    private JLabel iterationsLabel; // Label jumlah iterasi
    private JButton saveButton; // Tombol simpan solusi
    private JButton solveButton; // Tombol mulai solving
    private JButton stopButton; // Tombol stop solving
    private JButton resetButton; // Tombol reset papan
    private Solution solution; // Solusi yang ditemukan
    private volatile boolean stopRequested = false;
    private static final int CELL_SIZE = 40;
    private static final Color[] PIECE_COLORS = {
            new Color(231, 76, 60), // A - Merah terang
            new Color(46, 204, 113), // B - Hijau muda fresh
            new Color(52, 152, 219), // C - Biru laut
            new Color(155, 89, 182), // D - Ungu anggur
            new Color(241, 196, 15), // E - Kuning cerah
            new Color(230, 126, 34), // F - Orange segar
            new Color(26, 188, 156), // G - Turquoise
            new Color(52, 73, 94), // H - Abu gelap
            new Color(192, 57, 43), // I - Merah tua
            new Color(39, 174, 96), // J - Hijau daun
            new Color(41, 128, 185), // K - Biru medium
            new Color(142, 68, 173), // L - Ungu royal
            new Color(243, 156, 18), // M - Orange muda
            new Color(211, 84, 0), // N - Orange tua
            new Color(22, 160, 133), // O - Tosca gelap
            new Color(44, 62, 80), // P - Hitam keabu
            new Color(255, 99, 71), // Q - Salmon
            new Color(50, 205, 50), // R - Lime green
            new Color(30, 144, 255), // S - Dodger blue
            new Color(138, 43, 226), // T - Blue violet
            new Color(255, 215, 0), // U - Golden
            new Color(255, 140, 0), // V - Dark orange
            new Color(64, 224, 208), // W - Turquoise light
            new Color(119, 136, 153), // X - Slate gray
            new Color(220, 20, 60), // Y - Crimson
            new Color(34, 139, 34) // Z - Forest green
    };

    // Konstruktor utama
    public MainWindow() {
        setupWindow();
        createComponents();
        layoutComponents();
    }

    // Setup properti window utama
    private void setupWindow() {
        setTitle("IQ Puzzler Pro Solver");
```

```
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(600, 500);
        setLocationRelativeTo(null);
    }

    // Membuat semua komponen GUI
    private void createComponents() {
        // Input Panel
        JPanel inputPanel = new JPanel(new BorderLayout(5, 0));
        filePathField = new JTextField();
        filePathField.setEditable(false);
        JButton browseButton = new JButton("Browse");
        browseButton.addActionListener(_ -> handleBrowseButton());
        inputPanel.add(filePathField, BorderLayout.CENTER);
        inputPanel.add(browseButton, BorderLayout.EAST);

        // Control Panel
        JPanel controlPanel = new JPanel();
        solveButton = new JButton("Solve");
        resetButton = new JButton("Reset");
        stopButton = new JButton("Stop");
        saveButton = new JButton("Simpan Solusi");

        solveButton.addActionListener(_ -> handleSolveButton());
        resetButton.addActionListener(_ -> handleResetButton());
        stopButton.addActionListener(_ -> handleStopButton());
        saveButton.addActionListener(_ -> handleSaveButton());

        saveButton.setEnabled(false);
        stopButton.setEnabled(false);

        controlPanel.add(solveButton);
        controlPanel.add(resetButton);
        controlPanel.add(stopButton);
        controlPanel.add(saveButton);

        // Board Panel
        boardPanel = new JPanel();
        boardPanel.setPreferredSize(new Dimension(400, 300));
        boardPanel.setBorder(BorderFactory.createLineBorder(Color.BLACK));

        // Stats Panel
        JPanel statsPanel = new JPanel(new GridLayout(2, 1, 5, 5));
        timeLabel = new JLabel("Waktu: -");
        iterationsLabel = new JLabel("Pengulangan: -");
        statsPanel.add(timeLabel);
        statsPanel.add(iterationsLabel);
        statsPanel.setBorder(BorderFactory.createTitledBorder("Statistics"));
```

```
        // Main Layout
        setLayout(new BorderLayout(10, 10));
        JPanel topPanel = new JPanel(new BorderLayout(5, 5));
        topPanel.add(inputPanel, BorderLayout.NORTH);
        topPanel.add(controlPanel, BorderLayout.SOUTH);

        add(topPanel, BorderLayout.NORTH);
        add(boardPanel, BorderLayout.CENTER);
        add(statsPanel, BorderLayout.SOUTH);

        // Add padding
        ((JPanel) getContentPane()).setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));
    }

    // Mengatur layout komponen
    private void layoutComponents() {
        pack();
    }

    // Handler untuk tombol Browse
    private void handleBrowseButton() {
        JFileChooser fileChooser = new JFileChooser();
        fileChooser.setFileFilter(new FileNameExtensionFilter("Text Files", "txt"));

        if (fileChooser.showOpenDialog(this) == JFileChooser.APPROVE_OPTION) {
            filePathField.setText(fileChooser.getSelectedFile().getAbsolutePath());
            resetBoard();
        }
    }

    // Update tampilan board saat solving
    @Override
    public void updateVisualization(Board board) {
        updateBoard(board);
    }

    // Handler untuk tombol Stop
    private void handleStopButton() {
        stopRequested = true;
        stopButton.setEnabled(false);
    }

    // Handler untuk tombol Solve
    private void handleSolveButton() {
        if (filePathField.getText().isEmpty()) {
            JOptionPane.showMessageDialog(this,
                    "Masukkan file puzzlenya terlebih dahulu.",
```

```
                "Error",
                JOptionPane.ERROR_MESSAGE);
        return;
    }

    // Reset flag dan update state tombol
    stopRequested = false;
    solveButton.setEnabled(false);
    stopButton.setEnabled(true);
    resetButton.setEnabled(false);
    saveButton.setEnabled(false);

    // Mulai proses solving di thread terpisah
    new Thread(() -> {
        try {
            PuzzleData puzzleData = PuzzleReader.readFromFile(filePathField.getText());
            BruteForceSolver solver = new BruteForceSolver(
                    puzzleData.getRows(),
                    puzzleData.getCols(),
                    puzzleData.getPieces(),
                    this,
                    () -> stopRequested);

            solution = solver.solve();

            SwingUtilities.invokeLater(() -> {
                stopButton.setEnabled(false);
                solveButton.setEnabled(true);
                resetButton.setEnabled(true);

                if (solution != null && !stopRequested) {
                    updateBoard(solution.getBoard());
                    timeLabel.setText("Waktu: " + solution.getExecutionTime() + " ms");
                    iterationsLabel.setText("Pengulangan: " + solution.getIterationCount());
                    saveButton.setEnabled(true);
                } else if (stopRequested) {
                    JOptionPane.showMessageDialog(this,
                            "Proses pencarian dihentikan",
                            "Stopped",
                            JOptionPane.INFORMATION_MESSAGE);
                } else {
                    JOptionPane.showMessageDialog(this,
                            "Tidak ada solusi mas!",
                            "Result",
                            JOptionPane.INFORMATION_MESSAGE);
                }
            });
        } catch (Exception e) {
```

```
                SwingUtilities.invokeLater(() -> {
                    stopButton.setEnabled(false);
                    solveButton.setEnabled(true);
                    resetButton.setEnabled(true);
                    JOptionPane.showMessageDialog(this,
                            "Error saat menyelesaikan puzzle: " + e.getMessage(),
                            "Error",
                            JOptionPane.ERROR_MESSAGE);
                });
            }
        }).start();
    }

    // Handler untuk tombol Reset
    private void handleResetButton() {
        resetBoard();
    }

    // Handler untuk tombol Save
    private void handleSaveButton() {
        if (solution == null)
            return;

        JFileChooser fileChooser = new JFileChooser();
        fileChooser.addChoosableFileFilter(new FileNameExtensionFilter("Text Files", "txt"));
        fileChooser.addChoosableFileFilter(new FileNameExtensionFilter("PNG Images", "png"));
        fileChooser.setAcceptAllFileFilterUsed(false);

        // Set default nama file
        fileChooser.setSelectedFile(new File("solusi.txt"));

        if (fileChooser.showSaveDialog(this) == JFileChooser.APPROVE_OPTION) {
            try {
                File file = fileChooser.getSelectedFile();
                String path = file.getPath();

                // Pastikan ekstensi file benar
                if (!path.endsWith(".txt") && !path.endsWith(".png")) {
                    FileNameExtensionFilter filter = (FileNameExtensionFilter)
fileChooser.getFileFilter();
                    String ext = filter.getExtensions()[0];
                    path = path + "." + ext;
                    file = new File(path);
                }

                System.out.println("Menyimpan ke: " + path); // Debug print

                // Save sesuai format
```

```
            if (path.endsWith(".txt")) {
                saveToTxt(file);
            } else if (path.endsWith(".png")) {
                saveToImage(file);
            }

            JOptionPane.showMessageDialog(this,
                    "Solusi berhasil disimpan ke: " + path,
                    "Success",
                    JOptionPane.INFORMATION_MESSAGE);
        } catch (Exception e) {
            e.printStackTrace(); // Print stack trace untuk debug
            JOptionPane.showMessageDialog(this,
                    "Error saat menyimpan solusi: " + e.getMessage(),
                    "Error",
                    JOptionPane.ERROR_MESSAGE);
        }
    }
}

// Menyimpan solusi ke file txt
private void saveToTxt(File file) throws IOException {
    try (BufferedWriter writer = new BufferedWriter(new FileWriter(file))) {
        // Write board configuration
        char[][] grid = solution.getBoard().getGrid();
        for (int i = 0; i < solution.getBoard().getRows(); i++) {
            for (int j = 0; j < solution.getBoard().getCols(); j++) {
                writer.write(grid[i][j] + " ");
            }
            writer.newLine();
        }

        // Write statistics
        writer.newLine();
        writer.write("Execution time: " + solution.getExecutionTime() + " ms");
        writer.newLine();
        writer.write("Iterations: " + solution.getIterationCount());
    }
}

// Menyimpan solusi ke file gambar
private void saveToImage(File file) throws IOException {
    ImageGenerator.saveToImage(solution, file.getPath());
}

// Reset tampilan board ke kondisi awal
private void resetBoard() {
    boardPanel.removeAll();
```

```
        timeLabel.setText("Waktu: -");
        iterationsLabel.setText("Pengulangan: -");
        saveButton.setEnabled(false);
        solution = null;
        boardPanel.revalidate();
        boardPanel.repaint();
    }

    // Update tampilan board dengan konfigurasi baru
    private void updateBoard(Board board) {
        boardPanel.removeAll();
        boardPanel.setLayout(new GridLayout(board.getRows(), board.getCols(), 1, 1));

        // Tambahkan animasi fade in/out
        for (int i = 0; i < board.getRows(); i++) {
            for (int j = 0; j < board.getCols(); j++) {
                JPanel cell = new JPanel();
                cell.setPreferredSize(new Dimension(CELL_SIZE, CELL_SIZE));
                char piece = board.getGrid()[i][j];

                if (piece != '.') {
                    cell.setBackground(getPieceColor(piece));
                    JLabel label = new JLabel(String.valueOf(piece));
                    label.setForeground(Color.WHITE);
                    label.setFont(label.getFont().deriveFont(Font.BOLD));
                    cell.add(label);
                } else {
                    cell.setBackground(new Color(240, 240, 240));
                }

                cell.setBorder(BorderFactory.createLineBorder(Color.GRAY));
                boardPanel.add(cell);
            }
        }

        boardPanel.revalidate();
        boardPanel.repaint();
    }

    // Mendapatkan warna untuk piece berdasarkan huruf (A-Z)
    private Color getPieceColor(char piece) {
        int index = piece - 'A';
        if (index >= 0 && index < PIECE_COLORS.length) {
            return PIECE_COLORS[index];
        }
        return Color.GRAY;
    }
```

```
}
```

Analisis Algoritma

Kompleksitas Waktu

Kompleksitas waktu algoritma Brute Force dalam kasus terburuk adalah: O(P! × R × N × M) dimana:

- P = jumlah piece
- R = jumlah kemungkinan rotasi dan pencerminan setiap piece
- N × M = ukuran papan

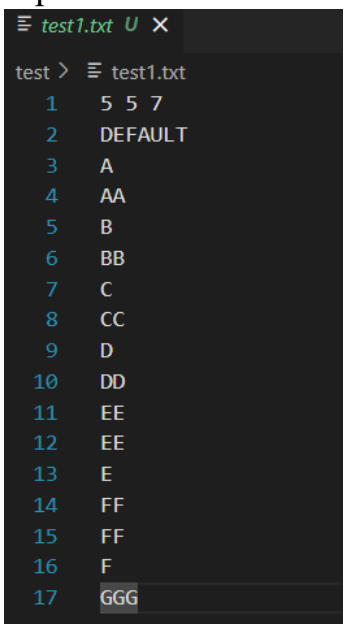
Hal ini karena:

1. Untuk setiap posisi kosong, kita harus mencoba setiap piece yang tersisa (P!)
2. Setiap piece memiliki beberapa kemungkinan orientasi (R)
3. Untuk setiap penempatan piece, kita harus mengecek validitas (N×M)

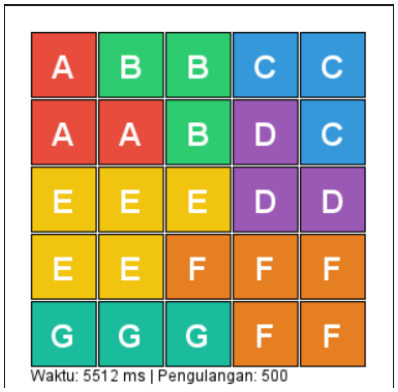
Pengujian

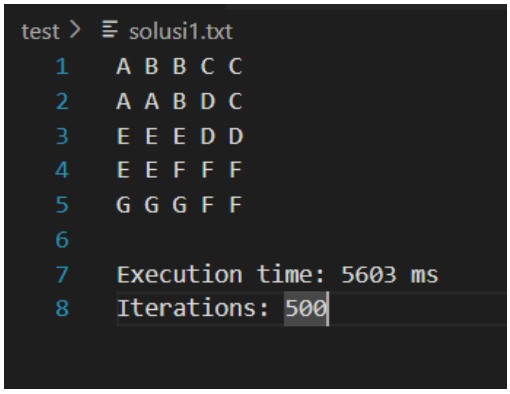
Test case 1:

Input:



Output:

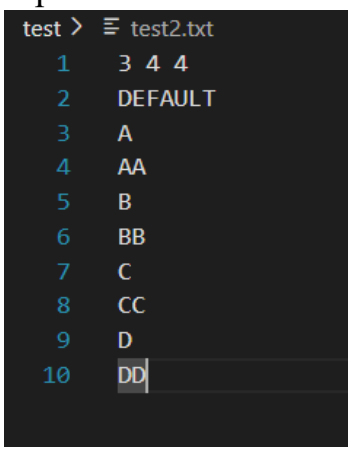




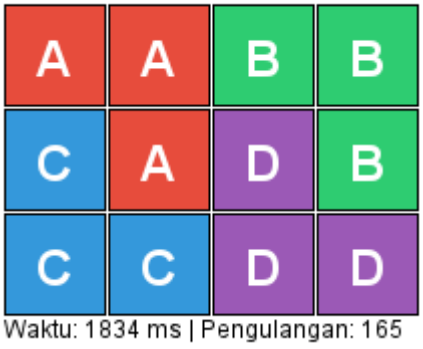
Waktu Runtime: 5512ms
Jumlah Pengulangan: 500 kali

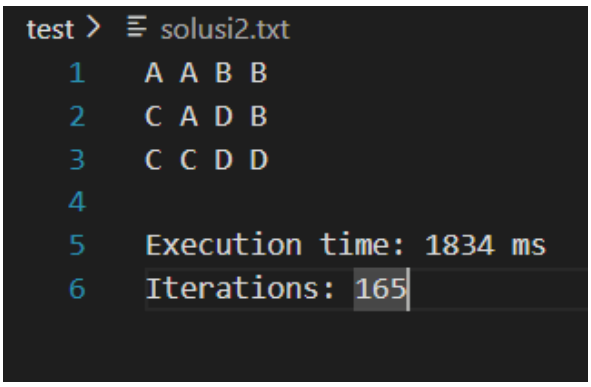
Test case 2:

Input:



Output:





Waktu Runtime: 1834ms
Jumlah Pengulangan: 165 kali

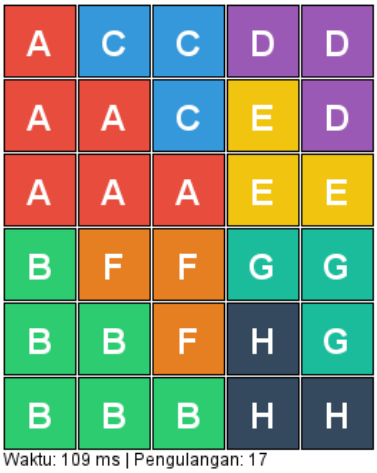
Test case 3:

Input:

test > test3.txt

```
6 5 8
DEFAULT
A
AA
AAA
B
BB
BBB
C
CC
D
DD
E
EE
F
FF
G
GG
HH
H
```

Output:



test > solusi3.txt

```
A C C D D
A A C E D
A A A E E
B F F G G
B B F H G
B B B H H

Execution time: 109 ms
Iterations: 17
```

Waktu Runtime: 109ms
Jumlah Pengulangan: 17 kali

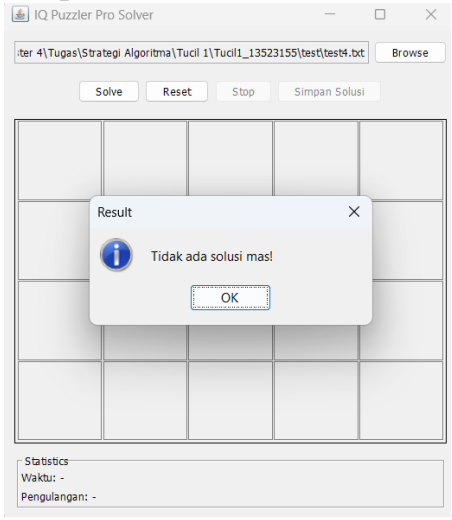
Test case 4:

Input:

test > test4.txt

```
4 5 5
DEFAULT
AAAAA
BBBBB
CCCCC
DDDDDD
```

Output:



Waktu Runtime: -
Jumlah Pengulangan: -

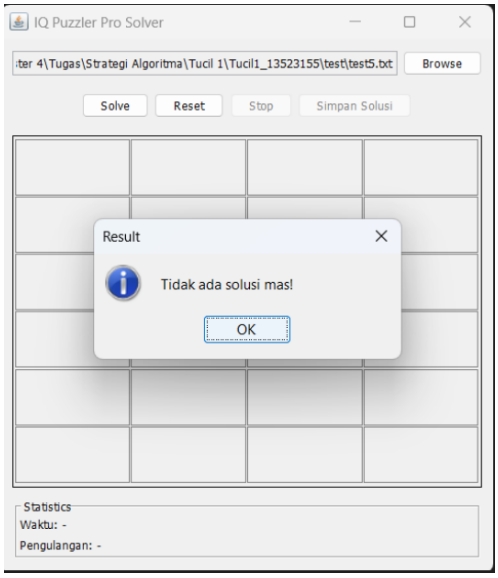
Test case 5:

Input:

test > test5.txt

```
6 4 4
DEFAULT
AAAA
B
BBB
CC
 C
DDD
 D
```

Output:

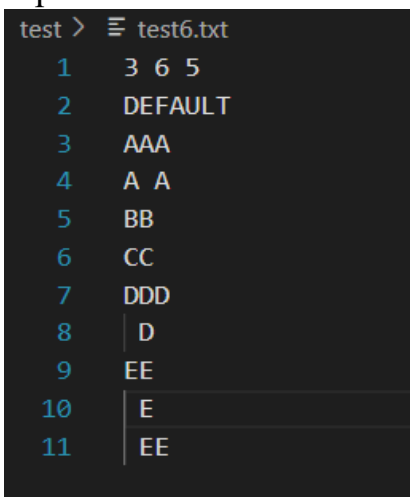


Waktu Runtime: -

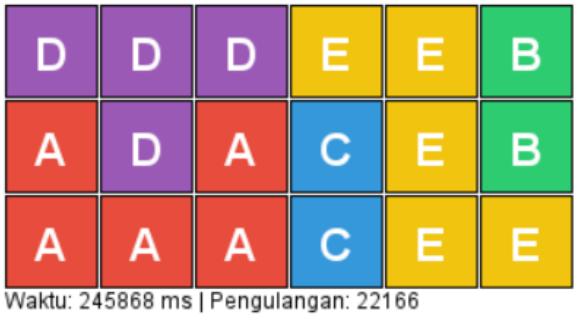
Jumlah Pengulangan: -

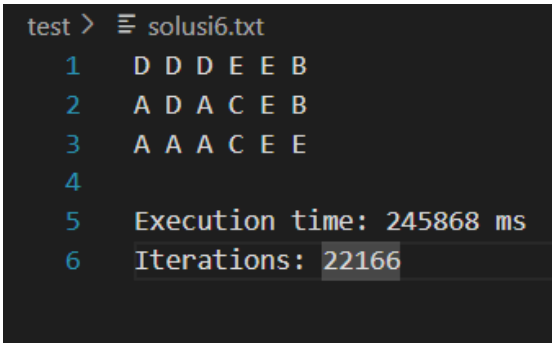
Test case 6:

Input:



Output:





Waktu Runtime: 245868ms (≈4,097menit)

Jumlah Pengulangan: 22.166 kali

Test case 7:

Input:

test > test7.txt

```
5 5 8
DEFAULT
A A
 A
A A
 B
B
 B
C
 C
D
 D
EEE
E
FF
F
 F
GG
G
 G
H
```

Output:

test > solusi7.txt

```
A B A D C
B A H C D
A B A E E
F G F G E
F F G G E

Execution time: 8911815 ms
Iterations: 811566
```

Waktu Runtime: 8911815ms (≈2,475jam)
Jumlah Pengulangan: 811.566 kali

Bonus yang Dikerjakan

1. Output Berupa Gambar

Program dapat menyimpan solusi dalam format gambar PNG dengan fitur:

- Warna berbeda untuk setiap piece
- Grid yang jelas
- Label piece
- Informasi statistik

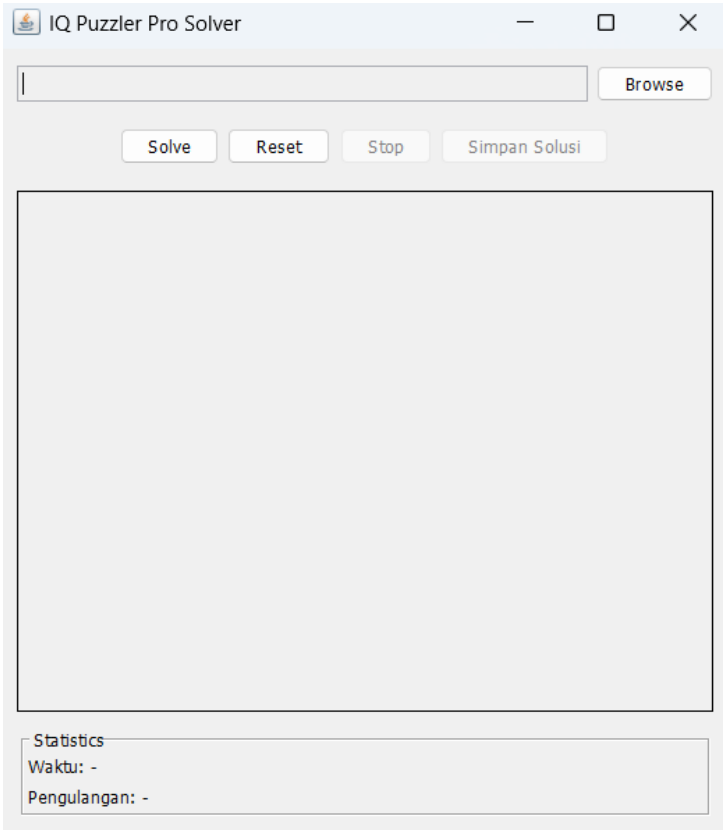
2. Graphical User Interface

Program dilengkapi dengan GUI yang memiliki fitur:

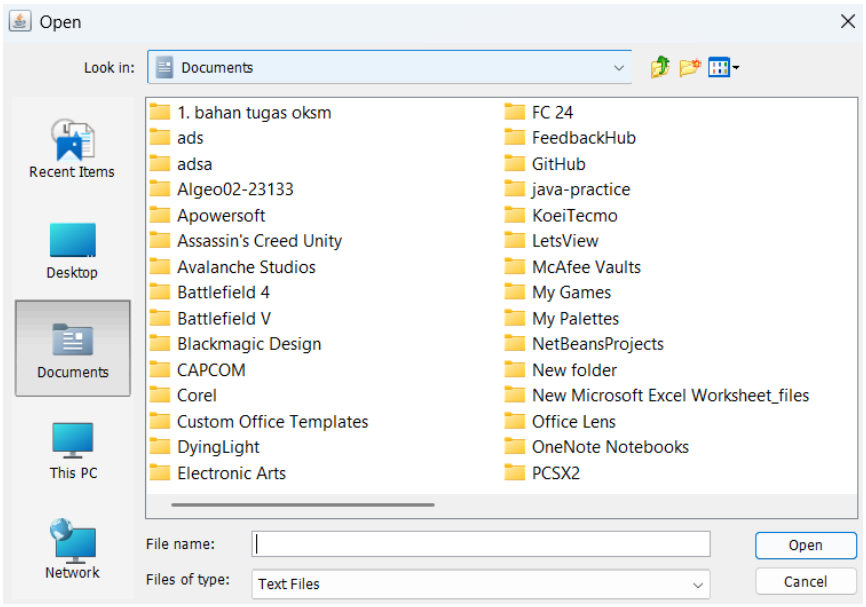
- File picker untuk memilih file input
- Visualisasi proses solving secara real-time
- Tombol kontrol (Solve, Stop, Reset)
- Opsi penyimpanan solusi dalam format teks/gambar
- Tampilan statistik (waktu eksekusi dan jumlah iterasi)

Dokumentasi GUI

- Tampilan Awal

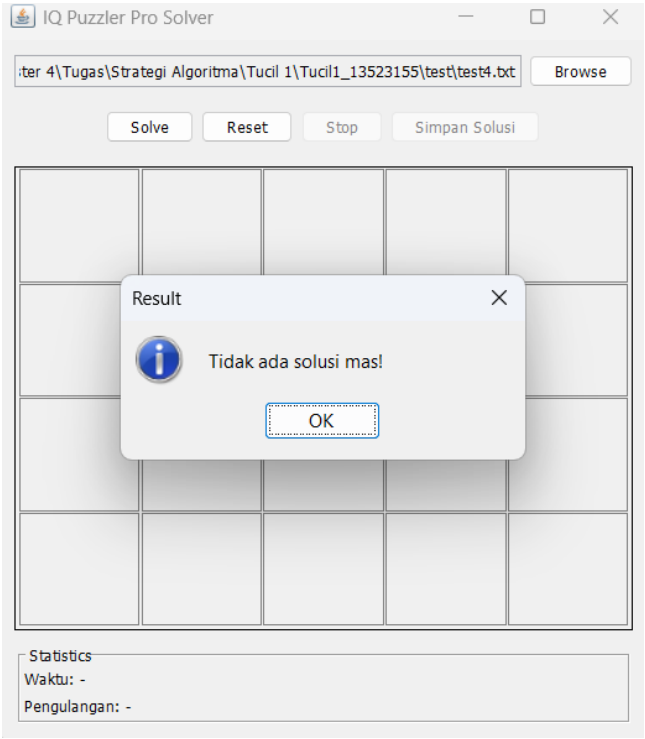


- Tampilan Mencari Folder



- Tampilan Ketika Sedang Melakukan Pencarian

- Tampilan Ketika Tidak Menemukan Solusi



- Tampilan Ketika Menemukan Solusi

Kesimpulan

Program berhasil mengimplementasikan algoritma Brute Force untuk menyelesaikan puzzle IQ Puzzler Pro dengan berbagai fitur tambahan yang

memudahkan penggunaan. Meskipun kompleksitas waktu cukup tinggi, program tetap dapat menyelesaikan puzzle dengan ukuran moderat dalam waktu yang reasonable.

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	√	

Pranala Ke Repository:

https://github.com/AbizzarG/Tucil1_13523155