LAPORAN TUGAS KECIL 1
IF2211 STRATEGI ALGORITMA
Penyelesaian **IQ Puzzler Pro** dengan Algoritma Brute Force

Disusun oleh:

Muhammad Raihaan Perdana

13523124

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
JL. GANESA 10, BANDUNG 40132

SEMESTER GENAP 2024/2025

DAFTAR ISI

BAB 1

DESKRIPSI PROGRAM & ALGORITMA BRUTE FORCE

1.1. Deskripsi Program

Program ini merupakan implementasi penyelesaian permainan **IQ Puzzler Pro** menggunakan algoritma brute force. **IQ Puzzler Pro** adalah permainan puzzle dimana pemain harus mengisi seluruh papan dengan blok-blok puzzle yang tersedia. Program ini dibuat menggunakan bahasa pemrograman Java dan memiliki dua mode interface: Command Line Interface (CLI) dan Graphical User Interface (GUI).

Fitur utama program:

- Membaca konfigurasi puzzle dari file input (.txt)
- Mencari solusi menggunakan algoritma brute force murni
- Menampilkan solusi dengan warna berbeda untuk setiap piece
- Menyimpan solusi dalam format teks (.txt)
- Visualisasi proses pencarian solusi melalui GUI
- Menyimpan solusi dalam format gambar (.png/.jpg)

1.2. Algoritma Program

Algoritma yang diimplementasikan dalam Tugas Kecil 1 ini adalah algoritma *brute force* yang menerapkan konsep dasar yaitu pendekatan yang *straightforward*, sederhana, dan langsung dalam memecahkan masalah. Seperti yang diketahui bahwa algoritma *brute force* memecahkan persoalan dengan cara yang sangat jelas (*obvious way*) - dalam hal ini dengan mencoba semua kemungkinan penempatan *piece* pada *board puzzle*.

Pada implementasinya dalam kode program, metode *brute force* ini diwujudkan melalui kelas PuzzleSolver yang menggunakan pendekatan rekursif. Program akan mencoba menempatkan setiap *piece puzzle* ke semua posisi yang mungkin pada *board*, dengan mempertimbangkan semua orientasi *piece* yang memungkinkan (baik rotasi maupun pencerminan). Pendekatan ini sangat sesuai dengan karakteristik *brute force* yang disebutkan dalam slide perkuliahan yaitu - "Just do it!" atau "Just Solve it!" - dimana kita benar-benar mencoba semua kemungkinan yang ada tanpa menggunakan strategi optimisasi khusus.

Berikut adalah langkah-langkah algoritma yang diimplementasikan:

1.2.1. Inisialisasi

- Baca ukuran papan (N×M) dan daftar *piece* dari file input
- Siapkan papan kosong

1.2.2. Untuk setiap piece secara berurutan

- Bangkitkan semua orientasi *piece* (rotasi dan pencerminan)
- Untuk setiap orientasi:
    - Untuk setiap posisi pada papan (i,j):
        - Jika piece dapat ditempatkan pada posisi (i,j):
            - Tempatkan *piece*
            - Rekursi untuk *piece* berikutnya
            - Jika rekursi berhasil, selesai
            - Jika tidak, hapus *piece (backtrack)*

1.2.3. Kondisi basis

- Jika semua *piece* telah ditempatkan dan papan terisi penuh, solusi ditemukan
- Jika tidak ada posisi valid untuk piece saat ini, *backtrack*

Pendekatan yang dilakukan di atas memenuhi karakteristik *brute force* sebagai berikut:

- Program mencoba SEMUA kemungkinan penempatan piece (*exhaustive search*)
- Untuk setiap *piece*, dicoba semua orientasi yang mungkin (rotasi dan pencerminan)
- Untuk setiap orientasi, dicoba semua posisi yang mungkin pada *board*
- Kompleksitas waktu yang tinggi sesuai karakteristik *brute force*
- Tidak menggunakan heuristik atau optimisasi khusus

BAB 2

STRUKTUR & SOURCE CODE PROGRAM

2.1. Struktur Program

Program terdiri dari beberapa *package* dan *class* utama:

1. Package model:
    a. Board.java: Representasi papan permainan
    b. Piece.java: Representasi piece puzzle
    c. Position.java: Representasi posisi pada papan
2. Package solver:
    a. PuzzleSolver.java: Implementasi algoritma brute force
3. Package utils:
    a. FileHandler.java: Penanganan input/output file
4. Package config:
    a. Constants.java: Konstanta untuk warna output
5. Package main:
    a. Main.java: Entry point program
    b. PuzzleGUI.java: Implementasi grafis antarmuka

2.2. Source Code Program

2.2.1 Package Model

Package model berfungsi sebagai representasi dasar dari komponen-komponen permainan, dimana Board.java menangani seluruh operasi terkait papan permainan seperti penempatan dan penghapusan piece, validasi keadaan papan, serta visualisasi papan dalam bentuk string. Piece.java mengimplementasikan logika untuk merepresentasikan setiap *piece puzzle* termasuk operasi rotasi dan pencerminan, sementara Position.java menyediakan abstraksi untuk menangani koordinat pada papan.

Board.java

```
package model;
import java.util.ArrayList;
import java.util.List;
import config.Constants;
public class Board {
    private final int rows;
    private final int cols;
    private char[][] grid;
    private List<Piece> placedPieces;
    private String gameType;
    public Board(int rows, int cols, String gameType) {
        this.rows = rows;
        this.cols = cols;
        this.grid = new char[rows][cols];
```

```
            this.placedPieces = new ArrayList<>();
            this.gameType = gameType;
            for (int i = 0; i < rows; i++) {
                for (int j = 0; j < cols; j++) {
                    grid[i][j] = '.';
                }
            }
        }
        public Board(Board other) {
            this.rows = other.rows;
            this.cols = other.cols;
            this.gameType = other.gameType;
            this.grid = new char[rows][cols];
            this.placedPieces = new ArrayList<>(other.placedPieces);
            // salin grid
            for (int i = 0; i < rows; i++) {
                   System.arraycopy(other.grid[i], 0, this.grid[i], 0,
cols);
            }
        }

        public boolean canPlacePiece(Piece piece, Position pos) {
                                    List<Position>    piecePositions    =
piece.getOccupiedPositions();

            // cek untuk setiap posisi yang akan ditempati piece
            for (Position piecePos : piecePositions) {
                int newRow = pos.getRow() + piecePos.getRow();
                int newCol = pos.getCol() + piecePos.getCol();

                // cek apakah posisi valid dalam board
                 if (!new Position(newRow, newCol).isValid(rows, cols))
{
                    return false;
                }

                // cek apakah posisi sudah ditempati
                if (grid[newRow][newCol] != '.') {
                    return false;
                }
            }

            return true;
        }

        public void placePiece(Piece piece, Position pos) {
                                    List<Position>    piecePositions    =
piece.getOccupiedPositions();

            // menempatkan piece pada grid
            for (Position piecePos : piecePositions) {
                int newRow = pos.getRow() + piecePos.getRow();
                int newCol = pos.getCol() + piecePos.getCol();
                grid[newRow][newCol] = piece.getSymbol();
            }

            placedPieces.add(piece);
        }
```

```
        public void removePiece(Piece piece, Position pos) {
                                    List<Position>    piecePositions    =
piece.getOccupiedPositions();

            // Hapus piece dari grid
            for (Position piecePos : piecePositions) {
                int newRow = pos.getRow() + piecePos.getRow();
                int newCol = pos.getCol() + piecePos.getCol();
                grid[newRow][newCol] = '.';
            }

            placedPieces.remove(piece);
        }

        public boolean isSolved() {
            for (int i = 0; i < rows; i++) {
                for (int j = 0; j < cols; j++) {
                    if (grid[i][j] == '.') {
                        return false;
                    }
                }
            }
            return true;
        }

        public void printBoard() {
            for (int i = 0; i < rows; i++) {
                for (int j = 0; j < cols; j++) {
                    char symbol = grid[i][j];
                    if (symbol != '.') {
                        int colorIndex = symbol - 'A';
                                if (colorIndex >= 0 && colorIndex <
Constants.COLORS.length) {

System.out.print(Constants.COLORS[colorIndex]      +      symbol      +
Constants.ANSI_RESET);
                        } else {
                            System.out.print(symbol);
                        }
                    } else {
                        System.out.print('.');
                    }
                }
                System.out.println();
            }
        }

        public int getRows() {
            return rows;
        }

        public int getCols() {
            return cols;
        }

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

```
        }

        public List<Piece> getPlacedPieces() {
            return placedPieces;
        }

        @Override
        public String toString() {
            StringBuilder sb = new StringBuilder();
            for (int i = 0; i < rows; i++) {
                for (int j = 0; j < cols; j++) {
                    sb.append(grid[i][j]);
                }
                sb.append('\n');
            }
            return sb.toString();
        }
    }
```

Piece.java

```
package model;
import java.util.ArrayList;
import java.util.List;
import java.util.Stack;
public class Piece {
    private char symbol;
    private boolean[][] shape;
    private int height;
    private int width;
    private int blockCount;

    public Piece(char symbol, List<String> pieceLines) {
        this.symbol = symbol;
        this.blockCount = 0;

        // menentukan dimensi piece
        this.height = pieceLines.size();
        this.width = pieceLines.stream()
                    .mapToInt(String::length)
                    .max()
                    .orElse(0);

        // inisialisasi shape
        this.shape = new boolean[height][width];
        for (int i = 0; i < height; i++) {
            String line = pieceLines.get(i);
            for (int j = 0; j < line.length(); j++) {
                shape[i][j] = (line.charAt(j) == symbol);
                if (shape[i][j]) {
                    blockCount++;
                }
            }
```

```
        }

        if (!isConnected()) {
                    throw new IllegalArgumentException("Piece harus
terhubung!");
        }
    }

    // salin constructor
    public Piece(Piece other) {
        this.symbol = other.symbol;
        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);
        }
    }

    public char getSymbol() {
        return symbol;
    }

    public boolean[][] getShape() {
        return shape;
    }

    public int getHeight() {
        return height;
    }

    public int getWidth() {
        return width;
    }

    public int getBlockCount() {
        return blockCount;
    }

    // rotasi piece 90 derajat searah jarum jam
    public void 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];
            }
        }
        shape = newShape;
        int tempDim = height;
        height = width;
        width = tempDim;
    }
```

```
    // pencerminan piece secara horizontal
    public void flip() {
        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];
            }
        }
        shape = newShape;
    }

    // generate semua kemungkinan orientasi piece
    public List<Piece> getAllOrientations() {
        List<Piece> orientations = new ArrayList<>();
        Piece current = new Piece(this);

        // 4 rotasi tanpa flip
        for (int i = 0; i < 4; i++) {
            orientations.add(new Piece(current));
            current.rotate();
        }

        // flip piece dan 4 rotasi lagi
        current = new Piece(this);
        current.flip();
        for (int i = 0; i < 4; i++) {
            orientations.add(new Piece(current));
            current.rotate();
        }

        return orientations;
    }

      // mendapatkan list posisi yang ditempati piece relatif terhadap
(0,0)
    public List<Position> getOccupiedPositions() {
        List<Position> positions = new ArrayList<>();
        for (int i = 0; i < height; i++) {
            for (int j = 0; j < width; j++) {
                if (shape[i][j]) {
                    positions.add(new Position(i, j));
                }
            }
        }
        return positions;
    }

    // method untuk mengecek apakah piece terhubung
    private boolean isConnected() {
        if (blockCount == 0) return true;

        boolean[][] visited = new boolean[height][width];
        int connectedBlocks = 0;
```

```
        // cari posisi blok pertama
        int startI = -1, startJ = -1;
        outer:
        for (int i = 0; i < height; i++) {
            for (int j = 0; j < width; j++) {
                if (shape[i][j]) {
                    startI = i;
                    startJ = j;
                    break outer;
                }
            }
        }

        // hitung blok yang terhubung
        Stack<int[]> stack = new Stack<>();
        stack.push(new int[]{startI, startJ});

          int[][] directions = {{-1,0}, {1,0}, {0,-1}, {0,1}}; // atas,
bawah, kiri, kanan

        while (!stack.isEmpty()) {
            int[] current = stack.pop();
            int i = current[0], j = current[1];

            if (i < 0 || i >= height || j < 0 || j >= width ||
                !shape[i][j] || visited[i][j]) continue;

            visited[i][j] = true;
            connectedBlocks++;

            for (int[] dir : directions) {
                stack.push(new int[]{i + dir[0], j + dir[1]});
            }
        }

        return connectedBlocks == blockCount;
    }

    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < height; i++) {
            for (int j = 0; j < width; j++) {
                sb.append(shape[i][j] ? symbol : '.');
            }
            sb.append('\n');
        }
        return sb.toString();
    }
}
```

Position.java

```
package model;
```

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

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

    public int getRow() {
        return row;
    }

    public int getCol() {
        return col;
    }

    public void setRow(int row) {
        this.row = row;
    }

    public void setCol(int col) {
        this.col = col;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj) return true;
        if (obj == null || getClass() != obj.getClass()) return false;
        Position position = (Position) obj;
        return row == position.row && col == position.col;
    }

    @Override
    public String toString() {
        return "(" + row + ", " + col + ")";
    }

    // method untuk mendapatkan posisi baru hasil pergeseran
    public Position offset(int rowOffset, int colOffset) {
                return new Position(this.row + rowOffset, this.col +
colOffset);
    }
```

```
    // method untuk memeriksa apakah posisi valid dalam board berukuran
rows x cols
    public boolean isValid(int rows, int cols) {
        return row >= 0 && row < rows && col >= 0 && col < cols;
    }
}
```

2.2.2. Package Solver

Package solver dengan class utama PuzzleSolver.java merupakan inti dari program yang mengimplementasikan algoritma brute force untuk mencari solusi puzzle. Class ini menerapkan pendekatan rekursif dengan backtracking untuk mencoba semua kemungkinan penempatan piece pada board.

Solver.java

```
package solver;

import model.Board;
import model.Piece;
import model.Position;
import java.util.*;
import java.util.function.Consumer;

public class PuzzleSolver {
    private Board board;
    private List<Piece> pieces;
    private long iterations;
    private long startTime;
    private long executionTime;
    private Board solution;
    private Consumer<Board> progressCallback;

    public PuzzleSolver(Board board, List<Piece> pieces) {
        this.board = board;
        this.pieces = pieces;
        this.iterations = 0;
        this.solution = null;
    }

    public void setProgressCallback(Consumer<Board> callback) {
```

```
        this.progressCallback = callback;
    }

    private void updateProgress() {
        if (progressCallback != null) {
            progressCallback.accept(new Board(board));
        }
    }

    public boolean solve() {
        startTime = System.currentTimeMillis();
        boolean result = solveRecursive(0);
        executionTime = System.currentTimeMillis() - startTime;
        return result;
    }

    // fungsi rekursif utk mencoba semua kemungkinan penempatan piece
    // mencoba setiap piece secara berurutan dari indeks 0 sampai n-1
    private boolean solveRecursive(int pieceIndex) {
        // basis: semua piece sudah dicoba
        if (pieceIndex >= pieces.size()) {
            if (board.isSolved()) {
                solution = new Board(board);
                return true;
            }
            return false;
        }

           // mendapatkan piece yg akan digunakan utk iterasi ini dan
mendapatkan semua orientasi yang mungkin
        Piece currentPiece = pieces.get(pieceIndex);
        List<Piece> orientations = currentPiece.getAllOrientations();

        // coba pada setiap orientasi piece
        for (Piece orientation : orientations) {
            // coba setiap posisi pada board
            for (int row = 0; row < board.getRows(); row++) {
                for (int col = 0; col < board.getCols(); col++) {
                    Position pos = new Position(row, col);

                    // jika berhasil menempatkan piece
                    if (board.canPlacePiece(orientation, pos)) {
```

```
                            iterations++;          // menginkremen jumlah
iterasi
                        board.placePiece(orientation, pos);
                        updateProgress();

                        // rekursi (coba piece selanjutnya)
                        if (solveRecursive(pieceIndex + 1)) {
                            return true;
                        }

                           // backtrack: hapus piece jika tidak mendapat
solusi
                        board.removePiece(orientation, pos);
                        updateProgress();
                    }
                }
            }
        }

        return false;
    }

    public long getIterations() {
        return iterations;
    }

    public long getExecutionTime() {
        return executionTime;
    }

    public Board getSolution() {
        return solution;
    }
}
```

2.2.3. Package Utils

Package utils berisi class pembantu dimana FileHandler.java menangani operasi I/O untuk membaca konfigurasi puzzle dari file dan menyimpan solusi, sedangkan Constants.java menyimpan konstanta-konstanta yang digunakan untuk memberikan output berwarna.

```
    package utils;
```

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

    public class FileHandler {
        public static class PuzzleInput {
            public int rows;
            public int cols;
            public int numPieces;
            public String gameType;
            public List<Piece> pieces;
            public int totalBlocks;

                public PuzzleInput(int rows, int cols, int numPieces,
String gameType, List<Piece> pieces) {
                this.rows = rows;
                this.cols = cols;
                this.numPieces = numPieces;
                this.gameType = gameType;
                this.pieces = pieces;

                this.totalBlocks = pieces.stream()
                                         .mapToInt(Piece::getBlockCount)
                                         .sum();

                if (gameType.equals("DEFAULT") &&
                    totalBlocks != rows * cols) {
                    throw new IllegalArgumentException(
                        "Total blok (" + totalBlocks +
                         ") tidak sesuai dengan ukuran papan (" + (rows
* cols) + ")!"
                    );
                }
            }
        }

          public static PuzzleInput readInput(String filename) throws
IOException {
                 try (BufferedReader reader = new BufferedReader(new
FileReader(filename))) {
                // baca dimensi board dan jumlah piece
                String dimensionLine = reader.readLine();
```

```
                if (dimensionLine == null) {
                    throw new IllegalArgumentException("File kosong");
                }

                                                String[]    dimensions    =
dimensionLine.trim().split("\\s+");
                if (dimensions.length != 3) {
                       throw new IllegalArgumentException("Format input
tidak valid!");
                }

                int rows = Integer.parseInt(dimensions[0]);
                int cols = Integer.parseInt(dimensions[1]);
                int numPieces = Integer.parseInt(dimensions[2]);

                    if (rows <= 0 || cols <= 0 || numPieces <= 0 ||
numPieces > 26) {
                    throw new IllegalArgumentException(
                        "Jumlah pieces atau dimensi tidak valid!"
                    );
                }

                // baca tipe game
                String gameType = reader.readLine().trim();
                                    if   (!gameType.equals("DEFAULT")   &&
!gameType.equals("CUSTOM") && !gameType.equals("PYRAMID")) {
                    throw new IllegalArgumentException("Game type tidak
valid!");
                }

                // baca piece-piece
                List<Piece> pieces = new ArrayList<>();
                String line;
                List<String> currentPieceLines = new ArrayList<>();
                char currentSymbol = '\0';
                int piecesRead = 0;

                while ((line = reader.readLine()) != null) {
                    if (line.trim().isEmpty()) continue;

                    // cari karakter pertama yang bukan spasi
                    char firstNonSpace = '\0';
                    for (char c : line.toCharArray()) {
```

```
                        if (c != ' ') {
                            firstNonSpace = c;
                            break;
                        }
                    }

                    if (firstNonSpace < 'A' || firstNonSpace > 'Z') {
                        throw new IllegalArgumentException(
                                        "Piece symbol tidak valid: " +
firstNonSpace
                        );
                    }

                     if (currentSymbol == '\0' || firstNonSpace !=
currentSymbol) {
                        // simpan piece sebelumnya jika ada
                        if (!currentPieceLines.isEmpty()) {

pieces.add(createPieceWithLeadingSpaces(currentSymbol,
currentPieceLines));
                            piecesRead++;
                            currentPieceLines = new ArrayList<>();
                        }
                        currentSymbol = firstNonSpace;
                    }
                     currentPieceLines.add(line); // simpan line dengan
spasi awalnya
                }

                // tambahkan piece terakhir
                if (!currentPieceLines.isEmpty()) {

pieces.add(createPieceWithLeadingSpaces(currentSymbol,
currentPieceLines));
                    piecesRead++;
                }

                if (piecesRead != numPieces) {
                    throw new IllegalArgumentException(
                          "Jumlah pieces tidak sesuai! (Seharusnya: " +
numPieces +
                                  ", tetapi jumlah pieces saat ini: " +
piecesRead + ")"
```

```
                    );
                }

                return new PuzzleInput(rows, cols, numPieces, gameType,
pieces);

            } catch (FileNotFoundException e) {
                        throw new FileNotFoundException("File tidak
ditemukan!");
            } catch (NumberFormatException e) {
                throw new IllegalArgumentException("Format input tidak
valid!");
            }
        }

        private static Piece createPieceWithLeadingSpaces(char symbol,
List<String> lines) {
            try {
                // normalisasi line
                List<String> normalizedLines = new ArrayList<>();

                // minimum leading spaces
                int minLeadingSpaces = Integer.MAX_VALUE;
                for (String line : lines) {
                    int leadingSpaces = 0;
                            while (leadingSpaces < line.length() &&
line.charAt(leadingSpaces) == ' ') {
                        leadingSpaces++;
                    }
                        if (leadingSpaces < line.length()) { // hanya
dihitung jika line tidak spasi semua
                            minLeadingSpaces = Math.min(minLeadingSpaces,
leadingSpaces);
                    }
                }

                // hapus minimum leading spaces
                for (String line : lines) {
                    if (line.trim().isEmpty()) continue;
                                        String normalizedLine =
line.substring(minLeadingSpaces);
                    normalizedLines.add(normalizedLine);
                }
```

```
                return new Piece(symbol, normalizedLines);
            } catch (IllegalArgumentException e) {
                throw new IllegalArgumentException("Format piece tidak
valid: " + e.getMessage());
            }
        }

        public static void writeSolution(String filename, String
solution, long executionTime, long iterations) {
                try (PrintWriter writer = new PrintWriter(new
FileWriter(filename))) {
                writer.println("Solusi:");
                writer.println(solution);
                writer.println();
                        writer.printf("Waktu eksekusi: %d ms%n",
executionTime);
                writer.printf("Jumlah iterasi: %d%n", iterations);
            } catch (IOException e) {
                    System.err.println("Gagal menulis ke file: " +
e.getMessage());
            }
        }
    }
```

2.2.4. Package Main

Untuk antarmuka program, package main menyediakan dua opsi penggunaan melalui Main.java sebagai entry point. Program dapat dijalankan baik dalam mode Command Line Interface (CLI) untuk penggunaan yang lebih sederhana, maupun mode Graphical User Interface (GUI) melalui PuzzleGUI.java yang menyediakan visualisasi interaktif dari proses pencarian solusi.

Main.java

```
package main;

import model.Board;
import solver.PuzzleSolver;
import utils.FileHandler;
import utils.FileHandler.PuzzleInput;

import javax.swing.*;
import java.io.IOException;
```

```java
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        System.out.print("Pilih mode (1: GUI, 2: CLI): ");
        Scanner scanner = new Scanner(System.in);
        int mode = scanner.nextInt();

        if (mode == 1) {
            SwingUtilities.invokeLater(() -> {
                PuzzleGUI gui = new PuzzleGUI();
                gui.setVisible(true);
            });
        } else {
            runCLI(scanner);
        }
    }

    private static void runCLI(Scanner scanner) {
        try {
            // Baca input file
            System.out.print("Masukkan nama file input (.txt): ");
            scanner.nextLine();
            String inputFilename = scanner.nextLine();
            PuzzleInput input = FileHandler.readInput(inputFilename);

            // Buat board dan solver
                    Board board = new Board(input.rows, input.cols,
input.gameType);
                      PuzzleSolver solver = new PuzzleSolver(board,
input.pieces);

            boolean hasSolution = solver.solve();

            if (hasSolution) {
                System.out.println("\nSolusi ditemukan:");
                solver.getSolution().printBoard();
                        System.out.printf("\nWaktu eksekusi: %d ms%n",
solver.getExecutionTime());
                             System.out.printf("Jumlah iterasi: %d%n",
solver.getIterations());
```

```
                System.out.print("\nApakah anda ingin menyimpan solusi?
(ya/tidak): ");
                String saveResponse = scanner.nextLine().toLowerCase();

                if (saveResponse.equals("ya")) {
                        System.out.print("\nMasukkan nama file output
(.txt): ");
                    String outputFilename = scanner.nextLine();
                    FileHandler.writeSolution(
                        outputFilename,
                        solver.getSolution().toString(),
                        solver.getExecutionTime(),
                        solver.getIterations()
                    );
                     System.out.println("Solusi berhasil disimpan ke " +
outputFilename);
                }
            } else {
                System.out.println("Tidak ada solusi yang ditemukan.");
            }

        } catch (IOException e) {
            System.err.println("Error: " + e.getMessage());
        } catch (IllegalArgumentException e) {
            System.err.println("Error: " + e.getMessage());
        } finally {
            scanner.close();
        }
    }
}
```

PuzzleGUI.java

```
package main;

import model.Board;
import solver.PuzzleSolver;
import utils.FileHandler;
import utils.FileHandler.PuzzleInput;

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

```java
import java.io.File;
import java.io.IOException;
import java.util.List;

public class PuzzleGUI extends JFrame {
    private JButton chooseFileBtn;
    private JButton solveBtn;
    private JButton saveBtn;
    private JButton saveImageBtn;
    private JLabel statusLabel;
    private JPanel boardPanel;
    private JLabel timeLabel;
    private JLabel iterationLabel;
    private PuzzleInput input;
    private Board solution;
    private Board currentBoard;
    private long executionTime;
    private long iterations;
    private Color[] pieceColors;
    private JSlider speedSlider;
    private int animationDelay = 100;

    public PuzzleGUI() {
        initializeColors();
        setupWindow();
        setupComponents();
        setupLayout();
    }

    private void initializeColors() {
        pieceColors = new Color[26];
        pieceColors[0] = new Color(255, 87, 87);
        pieceColors[1] = new Color(98, 182, 87);
        pieceColors[2] = new Color(255, 189, 74);
        pieceColors[3] = new Color(74, 137, 255);
        pieceColors[4] = new Color(187, 74, 255);
        pieceColors[5] = new Color(74, 255, 231);
        for (int i = 6; i < 26; i++) {
            pieceColors[i] = new Color(
                (int)(Math.random() * 200) + 55,
                (int)(Math.random() * 200) + 55,
                (int)(Math.random() * 200) + 55
            );
```

```
        }
    }

    private void setupWindow() {
        setTitle("IQ Puzzler Pro Solver");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(800, 700);
        setLocationRelativeTo(null);
    }

    private void setupComponents() {
        chooseFileBtn = new JButton("Pilih file input");
        solveBtn = new JButton("Selesaikan Puzzle");
        saveBtn = new JButton("Simpan Solusi");
        saveImageBtn = new JButton("Simpan sebagai gambar");
        statusLabel = new JLabel("Pilih sebuah file input");
        timeLabel = new JLabel("Time: -");
        iterationLabel = new JLabel("Iterations: -");

        speedSlider = new JSlider(JSlider.HORIZONTAL, 0, 1000, 100);
        speedSlider.setMajorTickSpacing(200);
        speedSlider.setMinorTickSpacing(50);
        speedSlider.setPaintTicks(true);
        speedSlider.setPaintLabels(true);
        speedSlider.addChangeListener(_ -> {
            animationDelay = 1000 - speedSlider.getValue();
        });

        boardPanel = new JPanel() {
            @Override
            protected void paintComponent(Graphics g) {
                super.paintComponent(g);
                drawBoard(g);
            }
        };
        boardPanel.setPreferredSize(new Dimension(400, 400));
        boardPanel.setBackground(Color.WHITE);

        chooseFileBtn.addActionListener(_ -> chooseFile());
        solveBtn.addActionListener(_ -> solvePuzzle());
        saveBtn.addActionListener(_ -> saveSolution());

        solveBtn.setEnabled(false);
```

```
        saveBtn.setEnabled(false);

        saveImageBtn.setEnabled(false);
        saveImageBtn.addActionListener(_ -> saveAsImage());
    }

    private void setupLayout() {
        setLayout(new BorderLayout());

        JPanel controlPanel = new JPanel();
                    controlPanel.setLayout(new   BoxLayout(controlPanel,
BoxLayout.Y_AXIS));

        JPanel buttonPanel = new JPanel();
        buttonPanel.add(chooseFileBtn);
        buttonPanel.add(solveBtn);
        buttonPanel.add(saveBtn);
        buttonPanel.add(saveImageBtn);

        JPanel sliderPanel = new JPanel(new BorderLayout());

sliderPanel.setBorder(BorderFactory.createTitledBorder("Kecepatan
animasi"));
        sliderPanel.add(speedSlider);

        controlPanel.add(buttonPanel);
        controlPanel.add(sliderPanel);

        JPanel infoPanel = new JPanel();
                        infoPanel.setLayout(new    BoxLayout(infoPanel,
BoxLayout.Y_AXIS));
        infoPanel.add(statusLabel);
        infoPanel.add(timeLabel);
        infoPanel.add(iterationLabel);

        add(controlPanel, BorderLayout.NORTH);
        add(boardPanel, BorderLayout.CENTER);
        add(infoPanel, BorderLayout.SOUTH);
    }

    private void chooseFile() {
        JFileChooser fileChooser = new JFileChooser();
        fileChooser.setCurrentDirectory(new File("."));
```

```
        int result = fileChooser.showOpenDialog(this);

        if (result == JFileChooser.APPROVE_OPTION) {
            File selectedFile = fileChooser.getSelectedFile();
            try {
                input = FileHandler.readInput(selectedFile.getPath());
                statusLabel.setText("File berhasil dimuat");
                solveBtn.setEnabled(true);
                solution = null;
                currentBoard = null;
                saveBtn.setEnabled(false);
                timeLabel.setText("Waktu Eksekusi: -");
                iterationLabel.setText("Jumlah Iterasi: -");
                boardPanel.repaint();
            } catch (Exception e) {
                JOptionPane.showMessageDialog(this,
                    "Gagal membaca file: " + e.getMessage(),
                    "Gagal",
                    JOptionPane.ERROR_MESSAGE);
            }
        }
    }

    private void solvePuzzle() {
        if (input == null) return;

        solveBtn.setEnabled(false);
        chooseFileBtn.setEnabled(false);
        statusLabel.setText("Menyelesaikan puzzle...");

        new SwingWorker<Void, Board>() {
            @Override
            protected Void doInBackground() throws Exception {
                    Board board = new Board(input.rows, input.cols,
input.gameType);
                     PuzzleSolver solver = new PuzzleSolver(board,
input.pieces);

                solver.setProgressCallback(currentState -> {
                    try {
                        Thread.sleep(animationDelay);
                        publish(currentState);
                    } catch (InterruptedException e) {
```

```
                        Thread.currentThread().interrupt();
                    }
                });

                boolean hasSolution = solver.solve();

                if (hasSolution) {
                    solution = solver.getSolution();
                    executionTime = solver.getExecutionTime();
                    iterations = solver.getIterations();
                    SwingUtilities.invokeLater(() -> {
                        statusLabel.setText("Solusi didapatkan!");
                                timeLabel.setText(String.format("Waktu
Eksekusi: %d ms", executionTime));
                            iterationLabel.setText(String.format("Jumlah
Iterasi: %d", iterations));
                        saveBtn.setEnabled(true);
                        saveImageBtn.setEnabled(true);
                        boardPanel.repaint();
                    });
                } else {
                    SwingUtilities.invokeLater(() -> {
                        statusLabel.setText("Tidak ada solusi!");
                    });
                }

                SwingUtilities.invokeLater(() -> {
                    solveBtn.setEnabled(true);
                    chooseFileBtn.setEnabled(true);
                });

                return null;
            }

            @Override
            protected void process(List<Board> chunks) {
                // Update board display dengan state terbaru
                currentBoard = chunks.get(chunks.size() - 1);
                boardPanel.repaint();
            }
        }.execute();
    }
```

```
    private void saveSolution() {
        if (solution == null) return;

        JFileChooser fileChooser = new JFileChooser();
        fileChooser.setCurrentDirectory(new File("."));
                            if    (fileChooser.showSaveDialog(this)    ==
JFileChooser.APPROVE_OPTION) {
            String filename = fileChooser.getSelectedFile().getPath();
            if (!filename.endsWith(".txt")) {
                filename += ".txt";
            }
               FileHandler.writeSolution(filename, solution.toString(),
executionTime, iterations);
            JOptionPane.showMessageDialog(this,
                "Solusi berhasil disimpan!",
                "Berhasil",
                JOptionPane.INFORMATION_MESSAGE);
        }
    }

    private void drawBoard(Graphics g) {
        Board boardToDraw = currentBoard != null ? currentBoard :
                            (solution != null ? solution : null);

        if (boardToDraw == null && input == null) return;

        int rows = input != null ? input.rows : boardToDraw.getRows();
        int cols = input != null ? input.cols : boardToDraw.getCols();

        int cellSize = Math.min(boardPanel.getWidth() / cols,
                                boardPanel.getHeight() / rows);

        // Draw grid
        g.setColor(Color.LIGHT_GRAY);
        for (int i = 0; i <= rows; i++) {
            g.drawLine(0, i * cellSize, cols * cellSize, i * cellSize);
        }
        for (int j = 0; j <= cols; j++) {
            g.drawLine(j * cellSize, 0, j * cellSize, rows * cellSize);
        }

        // Draw pieces
        if (boardToDraw != null) {
```

```
            char[][] grid = boardToDraw.getGrid();
            for (int i = 0; i < rows; i++) {
                for (int j = 0; j < cols; j++) {
                    char piece = grid[i][j];
                    if (piece != '.') {
                        int colorIndex = piece - 'A';
                        g.setColor(pieceColors[colorIndex]);
                        g.fillRect(j * cellSize + 1, i * cellSize + 1,
                                cellSize - 1, cellSize - 1);

                        g.setColor(Color.BLACK);
                         g.setFont(new Font("Arial", Font.BOLD, cellSize
/ 2));
                        FontMetrics fm = g.getFontMetrics();
                        String text = String.valueOf(piece);
                               int textX = j * cellSize + (cellSize -
fm.stringWidth(text)) / 2;
                               int textY = i * cellSize + (cellSize +
fm.getAscent()) / 2;
                        g.drawString(text, textX, textY);
                    }
                }
            }
        }
    }

    private void saveAsImage() {
        if (solution == null) return;

        JFileChooser fileChooser = new JFileChooser();
        fileChooser.setCurrentDirectory(new File("."));

                                        fileChooser.setFileFilter(new
javax.swing.filechooser.FileFilter() {
            public boolean accept(File f) {
                                         return   f.isDirectory()   ||
f.getName().toLowerCase().endsWith(".png")
                    || f.getName().toLowerCase().endsWith(".jpg");
            }
            public String getDescription() {
                return "File gambar (*.png, *.jpg)";
            }
        });
```

```
                        if    (fileChooser.showSaveDialog(this)    ==
JFileChooser.APPROVE_OPTION) {
            String filename = fileChooser.getSelectedFile().getPath();
                    if  (!filename.toLowerCase().endsWith(".png")  &&
!filename.toLowerCase().endsWith(".jpg")) {
                filename += ".png";
            }

            try {
                int scale = 50;
                int rows = solution.getRows();
                int cols = solution.getCols();
                  BufferedImage image = new BufferedImage(cols * scale,
rows * scale,

BufferedImage.TYPE_INT_RGB);
                Graphics2D g2d = image.createGraphics();


                g2d.setColor(Color.WHITE);
                g2d.fillRect(0, 0, cols * scale, rows * scale);


                g2d.setColor(Color.LIGHT_GRAY);
                for (int i = 0; i <= rows; i++) {
                          g2d.drawLine(0, i * scale, cols * scale, i *
scale);
                }
                for (int j = 0; j <= cols; j++) {
                          g2d.drawLine(j * scale, 0, j * scale, rows *
scale);
                }

                // Draw pieces
                char[][] grid = solution.getGrid();
                for (int i = 0; i < rows; i++) {
                    for (int j = 0; j < cols; j++) {
                        char piece = grid[i][j];
                        if (piece != '.') {
                            int colorIndex = piece - 'A';
                            g2d.setColor(pieceColors[colorIndex]);
```

```
                            g2d.fillRect(j * scale + 1, i * scale + 1,
scale - 1, scale - 1);

                        g2d.setColor(Color.BLACK);
                            g2d.setFont(new Font("Arial", Font.BOLD,
scale / 2));
                        FontMetrics fm = g2d.getFontMetrics();
                        String text = String.valueOf(piece);
                                int textX = j * scale + (scale -
fm.stringWidth(text)) / 2;
                                int textY = i * scale + (scale +
fm.getAscent()) / 2;
                        g2d.drawString(text, textX, textY);
                    }
                }
            }

            g2d.dispose();

            // Save image
                                                String extension =
filename.substring(filename.lastIndexOf('.') + 1).toLowerCase();
            ImageIO.write(image, extension, new File(filename));

            JOptionPane.showMessageDialog(this,
                "Gambar berhasil disimpan",
                "Berhasil",
                JOptionPane.INFORMATION_MESSAGE);

        } catch (IOException ex) {
            JOptionPane.showMessageDialog(this,
                "Gagal menyimpan gambar: " + ex.getMessage(),
                "Gagal",
                JOptionPane.ERROR_MESSAGE);
        }
    }
}
```

2.2.5. Package Config

Package config berisi class Constant.java yang di dalamnya terdapat konstanta untuk warna blok-blok yang akan ditampilkan.

Constant.java

```java
package config;

public class Constants {
    // ANSI Color codes untuk output berwarna
    public static final String ANSI_RESET = "\u001B[0m";
    public static final String[] COLORS = {
        "\u001B[31m", // RED
        "\u001B[32m", // GREEN
        "\u001B[33m", // YELLOW
        "\u001B[34m", // BLUE
        "\u001B[35m", // PURPLE
        "\u001B[36m", // CYAN
        "\u001B[91m", // BRIGHT RED
        "\u001B[92m", // BRIGHT GREEN
        "\u001B[93m", // BRIGHT YELLOW
        "\u001B[94m", // BRIGHT BLUE
        "\u001B[95m", // BRIGHT PURPLE
        "\u001B[96m", // BRIGHT CYAN
        "\u001B[41m", // RED BACKGROUND
        "\u001B[42m", // GREEN BACKGROUND
        "\u001B[43m", // YELLOW BACKGROUND
        "\u001B[44m", // BLUE BACKGROUND
        "\u001B[45m", // PURPLE BACKGROUND
        "\u001B[46m", // CYAN BACKGROUND
        "\u001B[101m", // BRIGHT RED BACKGROUND
        "\u001B[102m", // BRIGHT GREEN BACKGROUND
        "\u001B[103m", // BRIGHT YELLOW BACKGROUND
        "\u001B[104m", // BRIGHT BLUE BACKGROUND
        "\u001B[105m", // BRIGHT PURPLE BACKGROUND
        "\u001B[106m", // BRIGHT CYAN BACKGROUND
        "\u001B[97m", // WHITE
        "\u001B[37m"  // GRAY
    };
}
```

BAB 3

HASIL & ANALISIS PENGUJIAN

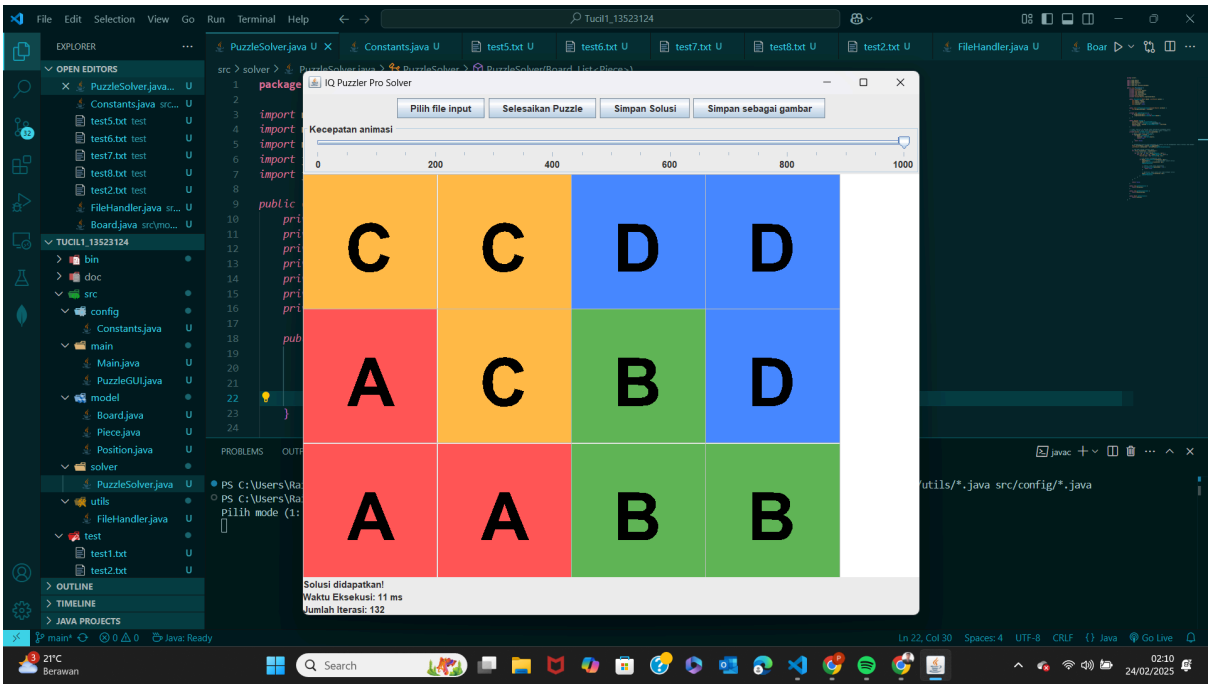
3.1. Hasil Pengujian

Berikut merupakan hasil pengujian dengan 8 *test case* yang tersedia:

3.1.1. Test Case 1

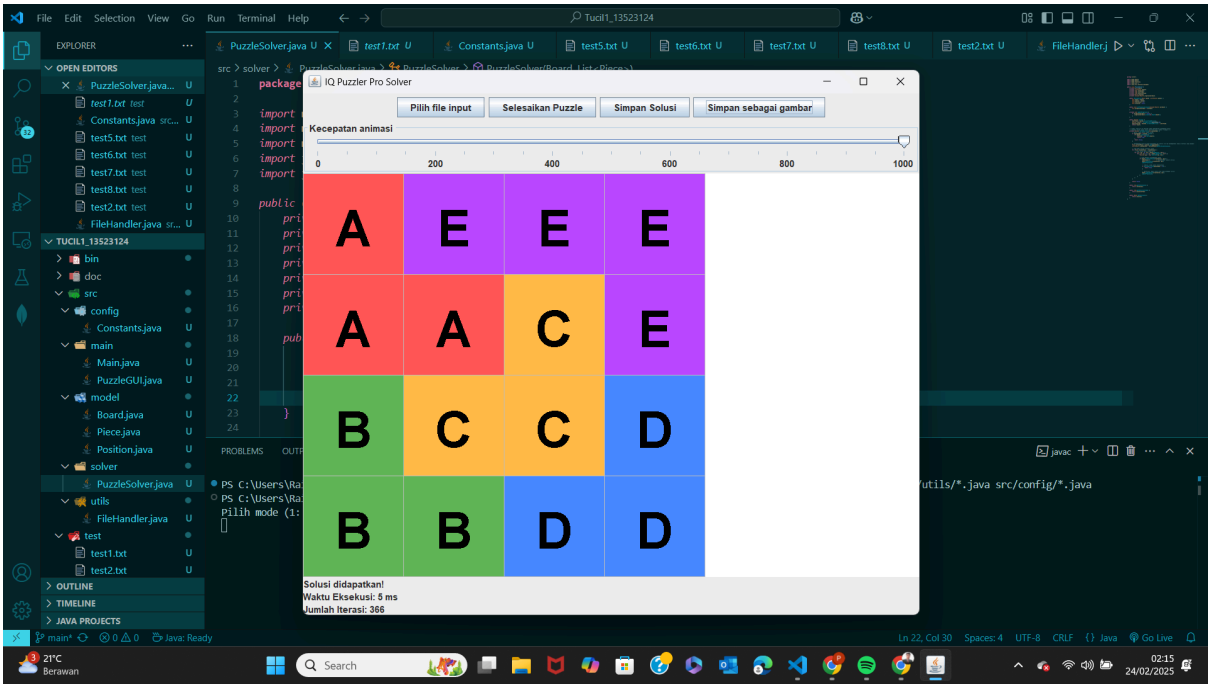
test > test1.txt

```
3 4 4
DEFAULT
A
AA
B
BB
C
CC
D
DD
```



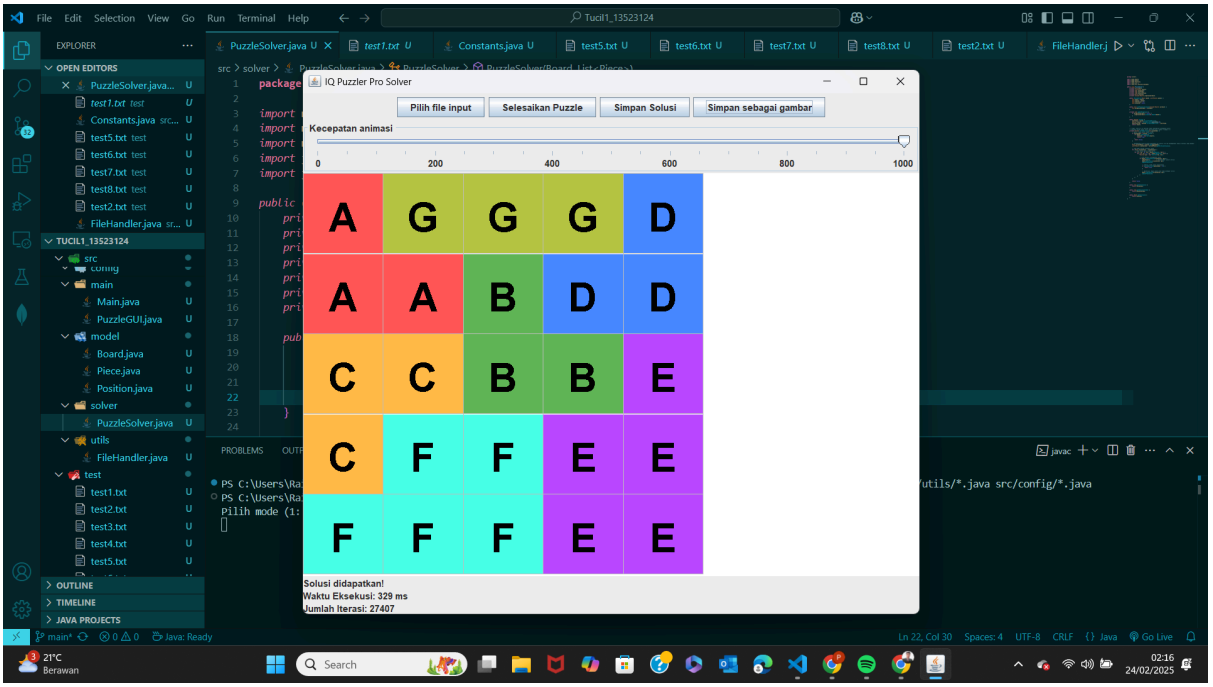
3.1.2. Test Case 2

```
test > test2.txt
1   4 4 5
2   DEFAULT
3   A
4   AA
5   B
6   BB
7   C
8   CC
9   D
10  DD
11  E
12  EEE
```

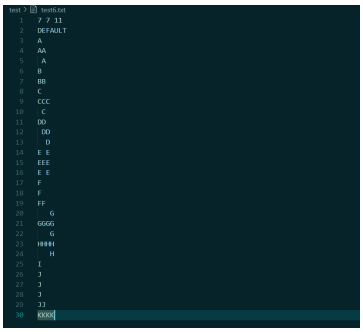


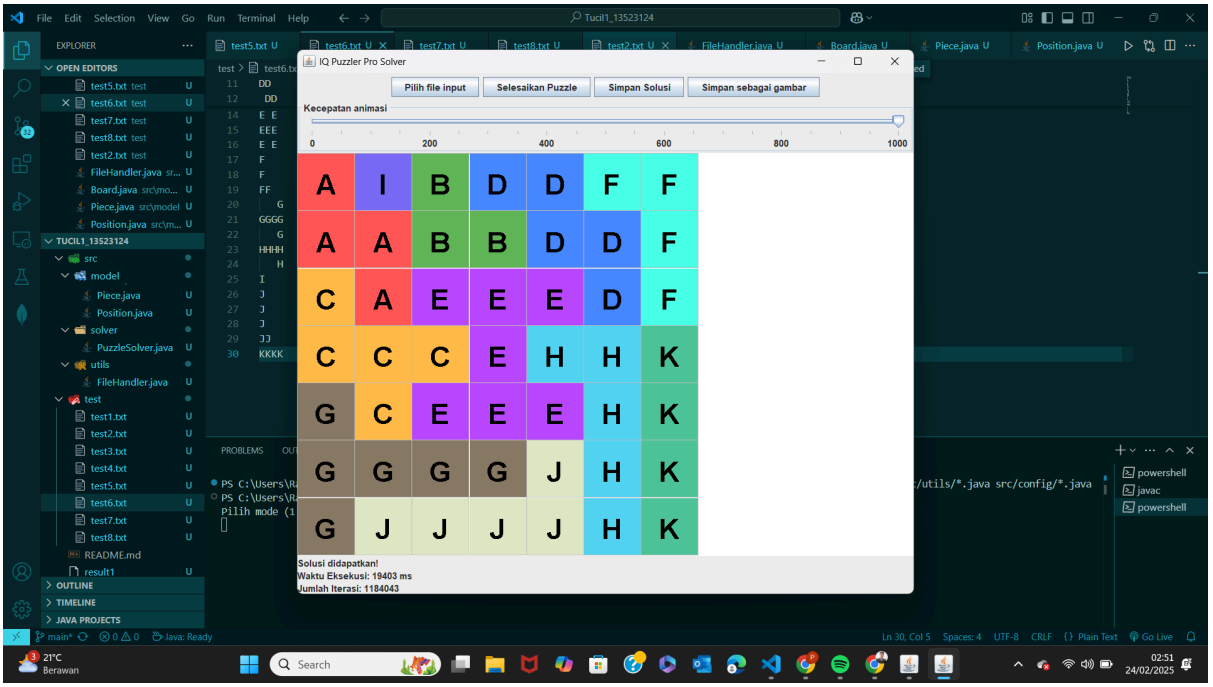
3.1.3. Test Case 3

```
test > test3.txt
1   5 5 7
2   DEFAULT
3   A
4   AA
5   B
6   BB
7   C
8   CC
9   D
10  DD
11  EE
12  EE
13  E
14  FF
15  FF
16  F
17  GGG
```



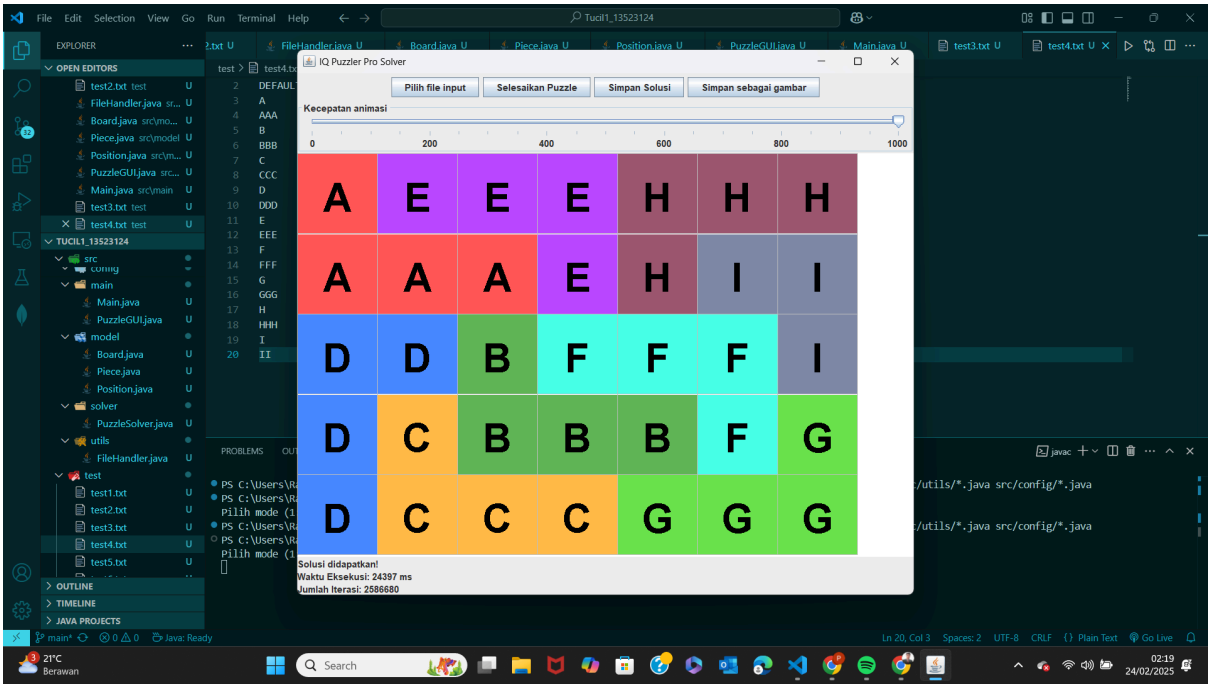
3.1.4. Test Case 4





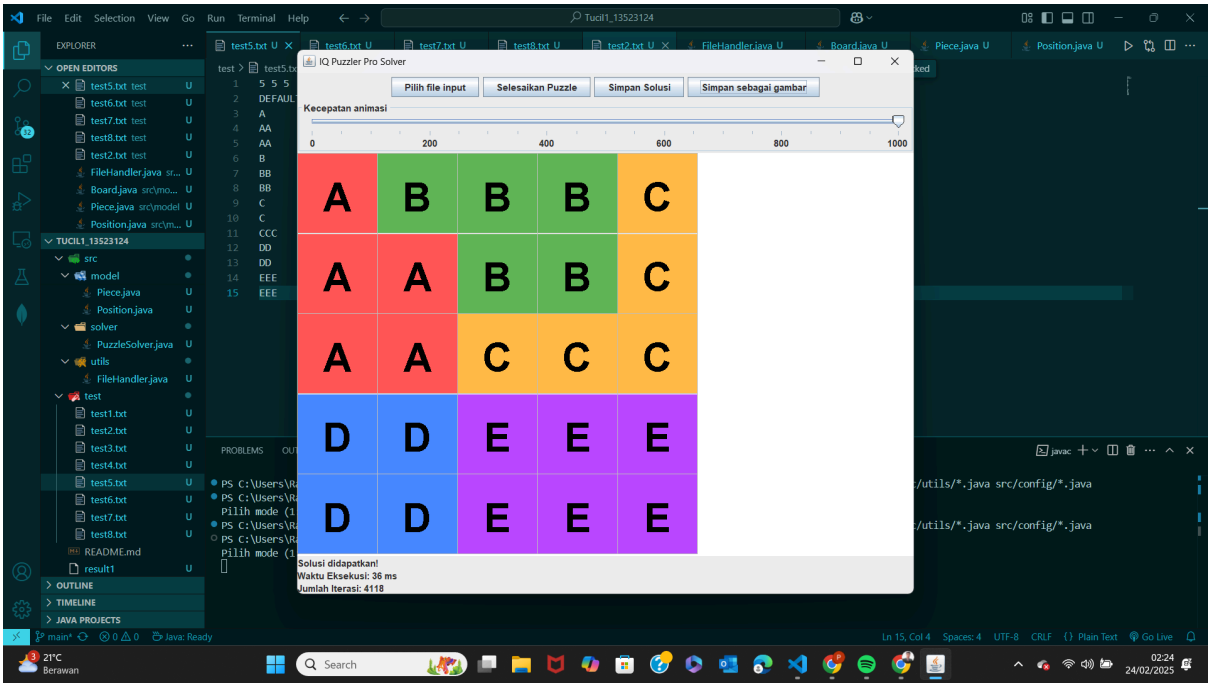
3.1.5. Test Case 5

```
5 7 9
DEFAULT
A
AAA
B
BBB
C
CCC
D
DDD
E
EEE
F
FFF
G
GGG
H
HHH
I
II
```

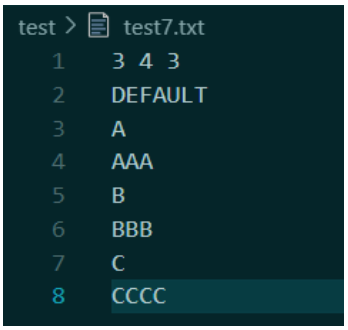


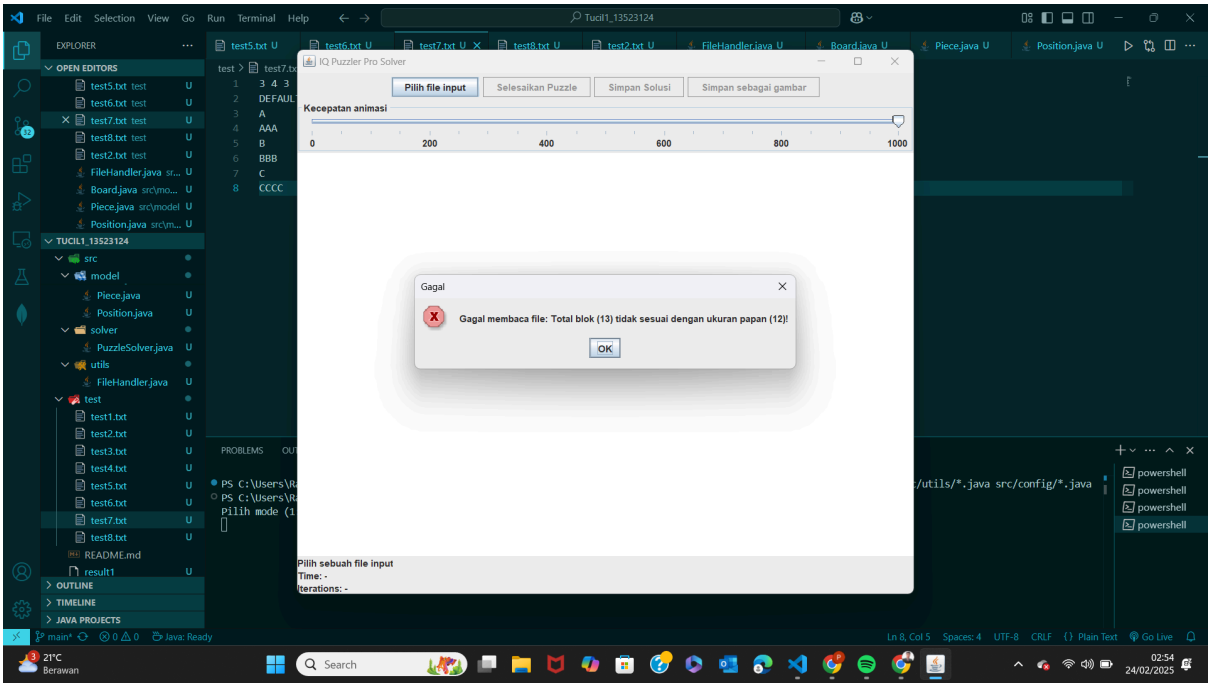
3.1.6. Test Case 6

```
5 5 5
DEFAULT
A
AA
AA
B
BB
BB
C
C
CCC
DD
DD
EEE
EEE
```

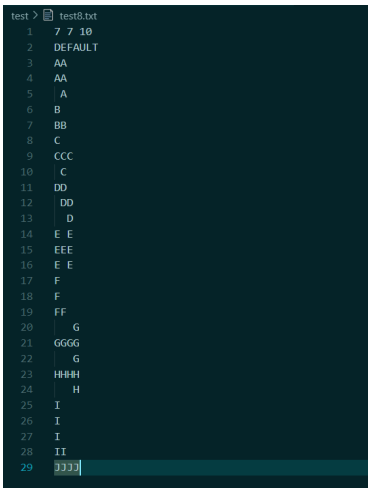


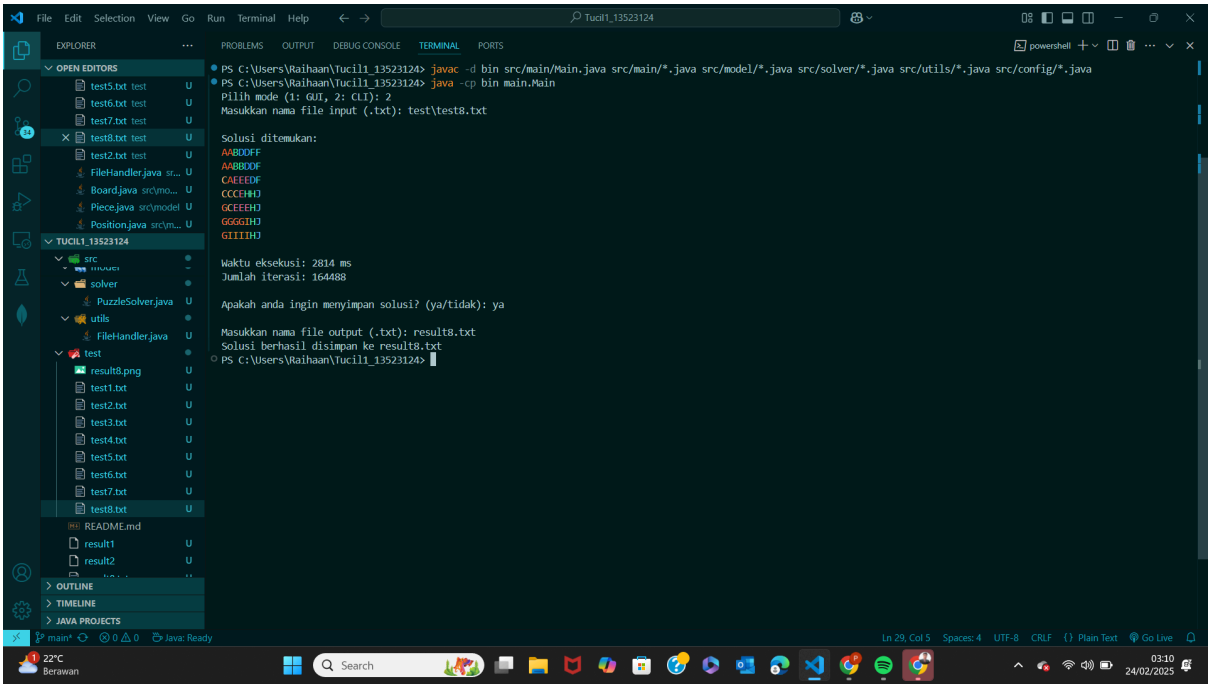
3.1.7. Test Case 7





3.1.8. Test Case 8





3.2. Analisis Pengujian

Algoritma brute force yang digunakan dalam program IQ Puzzler Pro menunjukkan pertumbuhan kompleksitas eksponensial seiring bertambahnya ukuran dan kompleksitas *puzzle*. Dari hasil pengujian terlihat bahwa jumlah iterasi dan waktu eksekusi meningkat drastis, seperti pada test2.txt yang hanya memerlukan 366 iterasi dalam 23 ms, sedangkan test6.txt mencapai lebih dari 1 juta iterasi dengan waktu eksekusi 17,6 detik. Hal ini terjadi karena *brute force* mencoba semua kemungkinan kombinasi penempatan blok tanpa strategi optimasi, menyebabkan lonjakan eksplorasi pada ruang pencarian yang besar. Akibatnya, efisiensi algoritma menurun secara signifikan pada puzzle yang lebih besar, membuatnya tidak praktis untuk digunakan tanpa teknik optimasi tambahan.

BAB 4

KESIMPULAN

4.1 Kesimpulan

Implementasi algoritma *brute force* dalam program ini menunjukkan bahwa meskipun algoritma *brute force* memiliki kompleksitas waktu yang tinggi, pendekatan ini masih merupakan solusi yang valid dan efektif untuk permasalahan IQ Puzzler Pro dengan ukuran input yang *reasonable*. Penggunaan rekursi dan backtracking dalam implementasi memungkinkan program untuk menjelajahi seluruh ruang solusi secara sistematis, sehingga menjamin ditemukannya solusi jika solusi tersebut ada. Hal ini sesuai dengan filosofi dasar algoritma *brute force* yang mengutamakan kebenaran dan kelengkapan solusi di atas efisiensi. Namun, seperti yang terlihat dari hasil eksekusi, pertumbuhan jumlah iterasi yang eksponensial menyebabkan peningkatan waktu eksekusi yang drastis pada *puzzle* yang lebih besar, membuatnya kurang praktis tanpa optimasi tambahan. Meskipun mungkin bukan merupakan solusi yang paling efisien, implementasi ini memberikan pemahaman yang mendalam tentang bagaimana sebuah algoritma *brute force* bekerja dan bagaimana konsep-konsep dasar seperti rekursi dan *backtracking* dapat digunakan untuk menyelesaikan permasalahan optimasi kombinatorial.

BAB 5

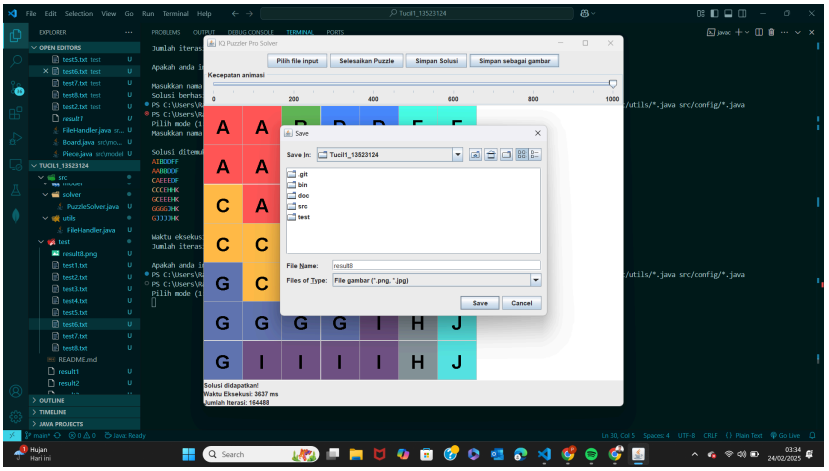
BONUS, REPOSITORI, TABEL EVALUASI

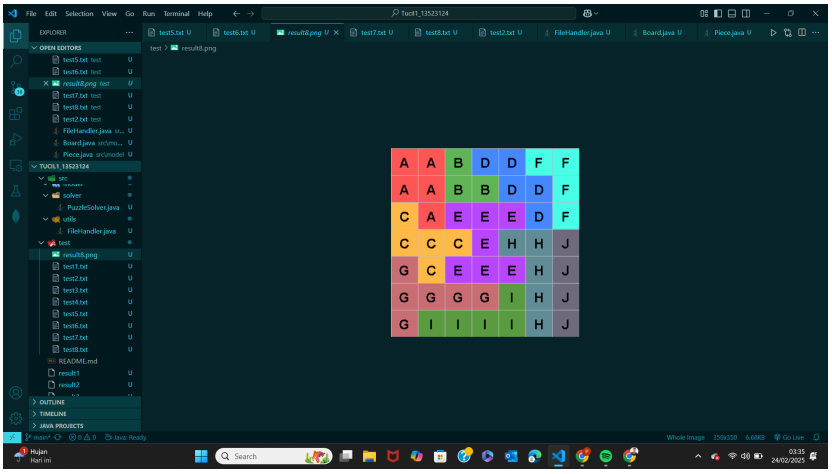
5.1. Bonus yang Diimplementasikan

Program ini mencakup juga beberapa bonus yang diimplementasikan

5.1.1. Output berupa Gambar (2 poin)

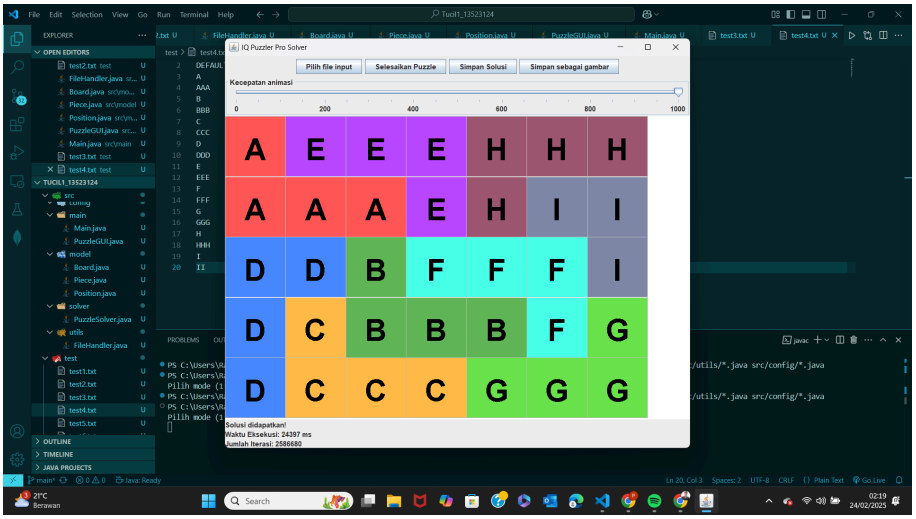
- Solusi dapat disimpan dalam format PNG/JPG
- Setiap piece memiliki warna berbeda
- Ukuran gambar disesuaikan dengan dimensi papan





5.1.2. Graphical User Interface (8 poin)

- Visualisasi proses pencarian solusi
- Slider untuk mengatur kecepatan animasi
- Tombol untuk menyimpan solusi (teks/gambar)
- Tampilan waktu eksekusi dan jumlah iterasi



5.2. Pranala ke Repositori

Link Repositori GitHub: https://github.com/fliegenhaan/Tucil1_13523124.git

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	✓	