

**Laporan Tugas Kecil 1 IF2211 Strategi Algoritma
Semester II tahun 2024/2025
Penyelesaian IQ Puzzler Pro dengan Algoritma Brute Force**



Oleh:
Andi Farhan Hidayat (13523128)

**PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2024**

Daftar Isi

Daftar Isi.....	2
Bab I: Deskripsi Masalah.....	3
2.1. Algoritma Brute Force.....	5
2.2. Bahasa Pemrograman Java.....	5
Bab III: Metode.....	6
3.1. Pembuatan Model.....	6
3.1.1. Kelas Board.....	6
3.1.2. Kelas Block.....	6
3.2. Pembuatan Utils.....	6
3.2.1. Kelas InputHandler.....	6
3.2.2. Kelas OutputHandler.....	7
3.3. Pembuatan Algoritma/Solver.....	7
BAB IV: Impelementasi.....	8
4.1. Teknologi Implementasi.....	8
4.2. Struktur Implementasi Program.....	8
4.2.1. Board.java.....	8
4.2.2. Block.java.....	9
4.2.3. InputHandler.java.....	10
4.2.4. OutputHandler.java.....	10
4.2.5. BruteForce.java.....	10
4.2.6. Main.java.....	11
4.2.7. IQPuzzlerGUI.java.....	11
4.2.8. NewCellRendererer.java.....	11
4.3. Kode Implementasi Program.....	11
4.3.1. Board.java.....	11
4.3.2. Block.java.....	12
4.3.3. InputHandler.java.....	14
4.3.4. OutputHandler.java.....	16
4.3.5. BruteForce.java.....	17
4.3.6. Main.java.....	20
4.3.7. IQPuzzlerGUI.java.....	21
4.3.8. NewCellRendererer.java.....	25
BAB V: Pengujian.....	27
5.1. Tampilan Aplikasi.....	27
5.2. Pengujian.....	27
Lampiran.....	32

Bab I: Deskripsi Masalah



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

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

Komponen penting dari permainan IQ Puzzler Pro terdiri dari:

1. **Board (Papan)** – Board merupakan komponen utama yang menjadi tujuan permainan dimana pemain harus mampu mengisi seluruh area papan menggunakan blok-blok yang telah disediakan.
2. **Blok/Piece** – Blok adalah komponen yang digunakan pemain untuk mengisi papan kosong hingga terisi penuh. Setiap blok memiliki bentuk yang unik dan semua blok harus digunakan untuk menyelesaikan puzzle.

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

Ilustrasi kasus

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



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

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

Bab II: Teori Singkat

2.1. Algoritma Brute Force

Algoritma *Brute Force* adalah pendekatan penyelesaian masalah dengan mencoba semua kemungkinan solusi yang ada sampai menemukan solusi yang tepat. Contoh penggunaan algoritma ini adalah mencoba semua kombinasi kunci untuk membuka sebuah kunci tanpa tahu mana yang benar. Algoritma ini bekerja dengan menguji setiap kemungkinan solusi satu per satu tanpa menggunakan strategi atau optimasi khusus. Begitu solusi yang benar ditemukan, proses berhenti. Contoh sederhananya adalah ketika lupa kata sandi 3 digit, Brute Force akan mencoba semua kombinasi dari 000 sampai 999 sampai menemukan kata sandi yang benar.

Fungsi utama dari *Brute Force* adalah untuk menyelesaikan masalah yang memiliki solusi terbatas dan dapat dihitung. Algoritma ini cocok digunakan ketika semua kemungkinan solusi dapat diuji secara sistematis. Keunggulan *Brute Force* terletak pada kesederhanaannya sehingga algoritma ini mudah dipahami dan tidak memerlukan pengetahuan khusus tentang struktur masalah. Selain itu, *Brute Force* selalu menemukan solusi yang tepat jika solusi tersebut memang ada karena algoritma ini mencoba semua kemungkinan tanpa melewatkan satu pun.

Namun, *Brute Force* memiliki kelemahan utama, yaitu tidak efisien untuk masalah besar. Ketika jumlah kemungkinan solusi sangat besar, seperti mencari kata sandi 10 digit, Brute Force akan memakan waktu dan sumber daya yang sangat banyak. Oleh sebab itu, algoritma ini lebih cocok digunakan untuk masalah kecil atau ketika tidak ada algoritma lain yang lebih efisien. Contoh penggunaan *Brute Force* termasuk mencari kombinasi kata sandi, menyelesaikan permainan puzzle, atau mencari elemen tertentu dalam daftar dengan memeriksa setiap elemen satu per satu.

Kesimpulannya, *Brute Force* adalah algoritma yang sederhana dan efektif untuk masalah kecil atau ketika solusi eksak diperlukan. Meskipun mudah dipahami dan selalu menemukan solusi yang tepat, Brute Force tidak efisien untuk masalah besar karena memerlukan waktu dan sumber daya yang besar. Oleh karena itu, penting untuk mempertimbangkan penggunaan Brute Force hanya ketika masalah yang dihadapi memiliki kemungkinan solusi yang terbatas atau ketika tidak ada alternatif algoritma yang lebih efisien.

2.2. Bahasa Pemrograman Java

Bab III: Metode

3.1. Pembuatan Model

3.1.1. Kelas Board

Kelas Board adalah kelas untuk menyimpan data board dari puzzle. Kelas ini akan membuat objek yang dapat menyimpan berbagai informasi, yaitu rows, cols, jumlah block (pieces), dan tipe masukan.

Kelas Board memetakan data dengan menggunakan matrix dari *character*. Dengan begitu, objek dari kelas Board dapat menyimpan dan menampilkan data dari tiap-tiap block dengan lebih mudah. Terdapat pula fungsi-fungsi getter-setter untuk tiap-tiap informasi private yang dimiliki oleh kelas Board. Selain itu, fungsi override toString() juga diimplementasikan ulang untuk dapat memudahkan dalam menampilkan informasi dari Board untuk GUI.

3.1.2. Kelas Block

Kelas Block merepresentasikan sebuah blok dalam permainan IQ Puzzler Pro. Setiap block memiliki simbol (karakter) yang unik dan bentuk (shape) yang didefinisikan sebagai matriks boolean (boolean[][]) dengan nilai true menunjukkan bagian yang terisi dan false menunjukkan bagian yang kosong. Bentuk blok diinisialisasi melalui metode parseShape yang mengubah data string (misalnya, ["## ", " # "]) menjadi matriks boolean. Kelas ini juga menyediakan fungsionalitas untuk menghasilkan semua orientasi yang mungkin dari blok, termasuk rotasi 90 derajat dan pencerminan dengan memastikan tidak ada duplikasi orientasi menggunakan Set untuk menyimpan bentuk yang sudah ada. Dengan demikian, kelas Block memungkinkan pengelolaan dan manipulasi bentuk blok secara efisien yang berguna dalam algoritma penyelesaian puzzle. Pencegahan duplikasi dengan menggunakan Set membantu dalam meningkatkan waktu eksekusi kode program karena tidak perlu mengulang hal yang sama. Pencegahan duplikasi tersebut juga masih termasuk dalam upaya untuk melakukan algoritma *Brute Force* sebab semua kemungkinan tetap dicoba. Pencegahan duplikasi tersebut juga tidak menggunakan metode heuristik.

3.2. Pembuatan Utils

3.2.1. Kelas InputHandler

Kelas InputHandler berperan untuk menangani input dari pengguna, baik melalui antarmuka baris perintah (CLI) maupun antarmuka grafis (GUI). Kelas ini menyediakan metode untuk meminta *path file* (*promptFile* dan *promptFileGUI*), membaca isi file (*readFile*), serta memproses data dari file tersebut untuk membuat objek Board dan daftar Block (*parseBoard* dan *parseBlock*). Metode *parseBoard* mengurai dimensi papan (baris, kolom, dan jumlah blok) serta jenis papan dari baris pertama dan kedua file. Sementara itu, *parseBlock* membaca bentuk blok dari file dan memastikan tidak ada simbol blok yang duplikat, serta membuat objek Block untuk setiap bentuk yang valid. Dengan demikian, kelas ini memungkinkan pengambilan dan pemrosesan data input secara terstruktur yang kemudian digunakan untuk inisialisasi papan dan blok dalam permainan puzzle. Pada kelas ini juga telah banyak dilakukan peningkatan untuk mencegah berbagai input error.

3.2.2. Kelas *OutputHandler*

Kelas *OutputHandler* bertanggung jawab untuk menyimpan hasil solusi dari permainan puzzle ke dalam file txt. Kelas ini menerima informasi tentang apakah solusi ditemukan (*solved*), waktu yang dibutuhkan untuk mencari solusi (*time*), dan jumlah iterasi yang dilakukan (*iterations*). Metode *saveSolution* menulis hasil tersebut ke file yang ditentukan, termasuk board solusi jika ditemukan, atau pesan bahwa solusi tidak ditemukan. Jika direktori output (*test/output*) belum ada, direktori tersebut akan dibuat secara otomatis. Dengan demikian, kelas ini memastikan bahwa hasil pemrosesan puzzle dapat disimpan dan diakses untuk referensi lebih lanjut. Implementasi *OutputHandler* juga telah diperhatikan untuk dapat diintegrasikan dengan fitur GUI.

3.3. Pembuatan Algoritma/*Solver*

Kelas *BruteForce* adalah implementasi algoritma *brute force* untuk menyelesaikan permainan IQ Puzzler Pro dengan mencoba semua kemungkinan penempatan blok pada papan. Algoritma ini bekerja secara rekursif dengan setiap blok dicoba untuk ditempatkan di semua posisi dan orientasi yang mungkin pada papan (rotasi dan pencerminan). Metode *solveRecursive* adalah inti dari algoritma ini yang mencoba menempatkan blok satu per satu dan memeriksa apakah penempatan tersebut valid menggunakan metode *canPlace*. Jika penempatan valid, blok ditempatkan menggunakan metode *place*, dan algoritma melanjutkan ke blok berikutnya. Jika tidak ada solusi yang ditemukan, blok dihapus menggunakan metode *remove*, dan algoritma mencoba kemungkinan lain.

Metode *solve* mengawali proses pencarian solusi dengan mencatat waktu mulai dan mengakhiri pencarian setelah semua kemungkinan dicoba. Jika solusi ditemukan, papan dicetak dan informasi waktu serta jumlah iterasi ditampilkan. Kelas ini juga menyediakan metode *getIterations* untuk mendapatkan jumlah iterasi yang dilakukan selama pencarian solusi. Dengan pendekatan *brute force*, algoritma ini menjamin menemukan solusi jika ada, meskipun mungkin memerlukan waktu yang lama untuk masalah yang kompleks.

Secara keseluruhan, kelas *BruteForce* adalah implementasi yang kuat dan sederhana untuk menyelesaikan puzzle dengan mencoba semua kemungkinan kombinasi penempatan blok. Meskipun tidak efisien untuk masalah besar, algoritma ini efektif untuk masalah kecil dan memberikan solusi yang pasti jika ada.

BAB IV: Impelementasi

4.1. Teknologi Implementasi

Program ini dibuat menggunakan bahasa pemrograman Java menggunakan bantuan *framework* Java Swing pada IDE IntelliJ.

4.2. Struktur Implementasi Program

4.2.1. Board.java

Kategori	Nama	Tipe Data	Fungsi/Keterangan
Atribut	rows	int	Jumlah baris pada papan.
	cols	int	Jumlah kolom pada papan.
	pieces	int	Jumlah blok/pieces yang akan ditempatkan pada papan.
	type	String	Jenis program.
	board	char[][]	Matriks 2D yang merepresentasikan papan dengan setiap sel berisi karakter atau 0.
Metode	Board(int, int, int, String)	Konstruktor	Menginisialisasi objek Board dengan baris, kolom, jumlah blok, dan tipe program.
	printBoard()	void	Mencetak isi board dengan sel kosong (0) ditampilkan sebagai _.
	getRows()	int	Mengembalikan jumlah baris pada board.
	getCols()	int	Mengembalikan jumlah kolom pada board.
	getCell(int, int)	char	Mengembalikan nilai sel pada baris dan kolom tertentu.
	setCell(int, int, char)	void	Mengatur nilai sel pada baris dan kolom tertentu dengan nilai yang diberikan.

	toString()	String	Mengembalikan representasi string dari board dengan sel kosong ditampilkan sebagai <code>_</code> . Ada untuk membantu menampilkan isi board di GUI.
--	------------	--------	--

4.2.2. Block.java

Kategori	Nama	Tipe Data	Fungsi/Keterangan
Atribut	symbol	char	Karakter yang mewakili <i>block</i> .
	shape	boolean[][]	Matriks 2D yang merepresentasikan bentuk blok (<i>true</i> untuk terisi, <i>false</i> untuk kosong).
	orientationsStore	List<boolean[][]>	Daftar orientasi unik dari <i>block</i> yang sudah dihitung.
Metode	Block(char, List<String>)	Konstruktor	Menginisialisasi objek <i>Block</i> dengan simbol dan bentuk yang diberikan.
	parseShape(List<String>)	boolean[][]	Mengubah daftar string menjadi matriks boolean yang merepresentasikan bentuk <i>block</i> .
	getSymbol()	char	Mengembalikan simbol <i>block</i> .
	getShape()	boolean[][]	Mengembalikan bentuk <i>block</i> .
	getAllOrientations()	List<boolean[][]>	Mengembalikan semua orientasi unik dari <i>block</i> (rotasi dan pencerminan).
	addUniqueOrientation(boolean[][], List<boolean[][]>, Set<String>)	void	Menambahkan orientasi unik ke daftar orientasi.
	shapeToString(boolean[][])	String	Mengubah matriks bentuk <i>block</i> menjadi string untuk perbandingan unik.

	rotate90(boolean[][])	boolean[][]	Memutar bentuk <i>block</i> sebesar 90 derajat.
	mirror(boolean[][])	boolean[][]	Mencerminkan bentuk <i>block</i> secara horizontal.

4.2.3. InputHandler.java

4.2.4. OutputHandler.java

4.2.5. BruteForce.java

Kategori	Nama	Tipe Data	Fungsi/Keterangan
Atribut	board	Board	Objek board tempat <i>block</i> akan ditempatkan.
	blocks	List<Block>	Daftar <i>block</i> yang akan ditempatkan pada papan.
	iterations	int	Jumlah iterasi yang dilakukan selama pencarian solusi.
Metode	BruteForce(Board, List<Block>)	Konstruktor	Menginisialisasi objek <i>BruteForce</i> dengan board dan daftar <i>block</i> .
	solve()	OutputHandler	Memulai proses pencarian solusi, mengembalikan hasil berupa <i>OutputHandler</i> .
	solveRecursive(int)	boolean	Metode rekursif untuk mencoba menempatkan <i>block</i> pada board.
	canPlace(boolean[][], int, int)	boolean	Memeriksa apakah block dapat ditempatkan pada posisi tertentu di board.
	place(boolean[][], char, int, int)	void	Menempatkan <i>block</i> pada <i>board</i> dengan simbol tertentu.
	remove(boolean[][], int, int)	void	Menghapus <i>block</i> dari <i>board</i> pada posisi tertentu.

	getIterations()	int	Mengembalikan jumlah iterasi yang dilakukan selama pencarian solusi.
--	-----------------	-----	--

4.2.6. Main.java

4.2.7. IQPuzzlerGUI.java

4.2.8. NewCellRenderer.java

4.3. Kode Implementasi Program

4.3.1. Board.java

```
package models;

public class Board {
    private final int rows;
    private final int cols;
    private final int pieces;
    private final String type;
    private char[][] board;

    public Board(int rows, int cols, int pieces, String type){
        this.rows = rows;
        this.cols = cols;
        this.pieces = pieces;
        this.type = type;
        this.board = new char[rows][cols];
    }

    public void printBoard(){
        for (char[] row: board){
            for (char cell: row) {
                System.out.print(cell == 0? '_' : cell + " ");
            }
            System.out.println();
        }
    }

    public int getRows() {
        return rows;
    }

    public int getCols() {
        return cols;
    }

    public char getCell(int row, int col) {
        return board[row][col];
    }
}
```

```

    public void setCell(int row, int col, char value) {
        board[row][col] = value;
    }

    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                char cell = board[i][j];
                sb.append(cell == 0 ? '_' : cell).append(' ');
            }
            sb.append('\n');
        }
        return sb.toString();
    }
}

```

4.3.2. Block.java

```

package models;

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

public class Block {
    private final char symbol;
    private final boolean[][] shape;
    private List<boolean[][]> orientationsStore;

    public Block(char symbol, List<String> shape) {
        this.symbol = symbol;
        this.shape = parseShape(shape);
    }

    private boolean[][] parseShape(List<String> shape) {
        int rows = shape.size();
        int cols = shape.stream().mapToInt(String::length).max().orElse(0);
        boolean[][] shapeMatrix = new boolean[rows][cols];

        for (int i = 0; i < rows; i++) {
            String line = shape.get(i);
            for (int j = 0; j < line.length(); j++) {
                shapeMatrix[i][j] = (line.charAt(j) != ' ');
            }
        }
    }
}

```

```

        return shapeMatrix;
    }

    public char getSymbol() {
        return symbol;
    }

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

    public List<boolean[][]> getAllOrientations() {
        if (orientationsStore != null) {
            return orientationsStore;
        }

        Set<String> seenOrientations = new HashSet<>();
        List<boolean[][]> orientations = new ArrayList<>();
        boolean[][] currentShape = this.shape;

        for (int i = 0; i < 4; i++) {
            addUniqueOrientation(currentShape, orientations, seenOrientations);
            addUniqueOrientation(mirror(currentShape), orientations,
seenOrientations);
            currentShape = rotate90(currentShape);
        }

        orientationsStore = orientations;
        return orientations;
    }

    private void addUniqueOrientation(boolean[][] shape, List<boolean[][]>
orientations, Set<String> seenOrientations) {
        String shapeKey = shapeToString(shape);
        if (!seenOrientations.contains(shapeKey)) {
            orientations.add(shape);
            seenOrientations.add(shapeKey);
        }
    }

    private String shapeToString(boolean[][] shape) {
        StringBuilder sb = new StringBuilder();
        for (boolean[] row : shape) {
            for (boolean cell : row) {
                sb.append(cell ? '1' : '0');
            }
            sb.append('|');
        }
        return sb.toString();
    }

```

```

    }

    private boolean[][] rotate90(boolean[][] shape) {
        int rows = shape.length;
        int cols = shape[0].length;
        boolean[][] rotated = new boolean[cols][rows];

        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                rotated[j][rows - 1 - i] = shape[i][j];
            }
        }

        return rotated;
    }

    private boolean[][] mirror(boolean[][] shape) {
        int rows = shape.length;
        int cols = shape[0].length;
        boolean[][] mirrored = new boolean[rows][cols];

        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                mirrored[i][cols - 1 - j] = shape[i][j];
            }
        }

        return mirrored;
    }
}

```

4.3.3. InputHandler.java

```

package utils;

import models.Block;
import models.Board;

import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.util.*;

public class InputHandler {
    public static File promptFile() {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Masukkan path dari file test case: ");
        String filePath = scanner.nextLine();
        filePath = "test/input/" + filePath;
    }
}

```

```

        File file = new File(filePath);
        return file.exists() ? file : null;
    }

    public static File promptFileGUI(String filePath) {
        filePath = "test/input/" + filePath;
        File file = new File(filePath);
        return file.exists() ? file : null;
    }

    public static List<String> readFile(File file) {
        try {
            return Files.readAllLines(file.toPath());
        } catch (IOException e) {
            e.printStackTrace();
            return List.of();
        }
    }

    public static Board parseBoard(List<String> lines) {
        String[] dimensions = lines.getFirst().split(" ");
        int rows = Integer.parseInt(dimensions[0]);
        int cols = Integer.parseInt(dimensions[1]);
        int pieces = Integer.parseInt(dimensions[2]);

        if (pieces > 26) {
            System.err.println("Jumlah blok/pieces hanya bisa kurang dari 26.");
            return null;
        }

        String type = lines.get(1);
        return new Board(rows, cols, pieces, type);
    }

    public static List<Block> parseBlock(List<String> lines) {
        List<Block> blocks = new ArrayList<>();
        Set<Character> seenSymbols = new HashSet<>();
        int i = 2;

        while (i < lines.size()) {
            String line = lines.get(i).trim();
            if (line.isEmpty()) {
                i++;
                continue;
            }

            char symbol = line.charAt(0);
            if (seenSymbols.contains(symbol)) {

```

```

        System.err.println("Error: Blok '" + symbol + "' muncul
berulang. Program dibatalkan.");
        return List.of();
    }

    List<String> shape = new ArrayList<>();
    while (i < lines.size()) {
        line = lines.get(i);
        if (line.trim().isEmpty()) {
            break;
        }

        boolean isBlockLine = true;
        for (int j = 0; j < line.length(); j++) {
            if (line.charAt(j) != ' ' && line.charAt(j) != symbol) {
                isBlockLine = false;
                break;
            }
        }

        if (!isBlockLine) {
            if (line.trim().charAt(0) != symbol) {
                break;
            }
        }

        shape.add(line);
        i++;
    }

    blocks.add(new Block(symbol, shape));
    seenSymbols.add(symbol);
}

return blocks;
}
}

```

4.3.4. OutputHandler.java

```

package utils;

import models.Board;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;

```



```

public class OutputHandler {
    private final boolean solved;
    private final long time;
    private final int iterations;

    public OutputHandler(boolean solved, long time, int iterations) {
        this.solved = solved;
        this.time = time;
        this.iterations = iterations;
    }

    public boolean saveSolution(String filePath, Board board) {
        File outputDir = new File("test/output");
        if (!outputDir.exists()) {
            outputDir.mkdirs();
        }

        try (FileWriter writer = new FileWriter(filePath)) {
            if (solved) {
                writer.write("Solusi Ditemukan!\n");
                for (int i = 0; i < board.getRows(); i++) {
                    for (int j = 0; j < board.getCols(); j++) {
                        char cell = board.getCell(i, j);
                        writer.write(cell == 0 ? '_' : cell);
                        writer.write(' ');
                    }
                    writer.write("\n");
                }
            } else {
                writer.write("Solusi tidak ditemukan!\n");
            }

            writer.write("Waktu pencarian: " + time + "ms\n");
            writer.write("Banyak kasus: " + iterations);
            return true;
        } catch (IOException e) {
            e.printStackTrace();
            return false;
        }
    }
}

```

4.3.5. BruteForce.java

```

package algorithm;

import models.Block;

```

```

import models.Board;
import utils.OutputHandler;

import java.util.List;

public class BruteForce {
    private final Board board;
    private final List<Block> blocks;
    private int iterations = 0;

    public BruteForce(Board board, List<Block> blocks) {
        this.board = board;
        this.blocks = blocks;
    }

    public OutputHandler solve() {
        long start = System.currentTimeMillis();
        boolean solved = solveRecursive(0);
        long end = System.currentTimeMillis();

        if (solved) {
            System.out.println("Solusi Ditemukan!");
            board.printBoard();
        } else {
            System.out.println("Solusi tidak ditemukan!");
        }
        long time = end - start;
        System.out.println("Waktu pencarian: " + time + "ms");
        System.out.println("Banyak kasus: " + iterations);

        return new OutputHandler(solved, time, iterations);
    }

    public boolean solveRecursive(int blockIndex) {
        if (blockIndex >= blocks.size()) {
            return true;
        }

        Block block = blocks.get(blockIndex);
        List<boolean[][]> orientations = block.getAllOrientations();

        int maxRow = board.getRows();
        int maxCol = board.getCols();

        for (boolean[][] orientation : orientations) {
            int blockRows = orientation.length;
            int blockCols = orientation[0].length;

            for (int row = 0; row <= maxRow - blockRows; row++) {

```

```

        for (int col = 0; col <= maxCol - blockCols; col++) {
            if (canPlace(orientation, row, col)) {
                place(orientation, block.getSymbol(), row, col);
                iterations++;

                if (solveRecursive(blockIndex + 1)) {
                    return true;
                }
                remove(orientation, row, col);
            }
        }
    }
    return false;
}

private boolean canPlace(boolean[][] shape, int startRow, int startCol) {
    for (int i = 0; i < shape.length; i++) {
        for (int j = 0; j < shape[i].length; j++) {
            if (shape[i][j] && board.getCell(startRow + i, startCol + j) !=
0) {
                return false;
            }
        }
    }
    return true;
}

private void place(boolean[][] shape, char symbol, int startRow, int
startCol) {
    for (int i = 0; i < shape.length; i++) {
        for (int j = 0; j < shape[i].length; j++) {
            if (shape[i][j]) {
                board.setCell(startRow + i, startCol + j, symbol);
            }
        }
    }
}

private void remove(boolean[][] shape, int startRow, int startCol) {
    for (int i = 0; i < shape.length; i++) {
        for (int j = 0; j < shape[i].length; j++) {
            if (shape[i][j]) {
                board.setCell(startRow + i, startCol + j, (char) 0);
            }
        }
    }
}

```

```

    public int getIterations() {
        return this.iterations;
    }
}

```

4.3.6. Main.java

```

import algorithm.BruteForce;
import models.Block;
import models.Board;
import utils.InputHandler;
import utils.OutputHandler;

import java.io.File;
import java.util.List;
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        File file = InputHandler.promptFile();
        if (file == null) {
            System.out.println("File tidak ditemukan!");
            return;
        }

        List<String> lines = InputHandler.readFile(file);
        if (lines.isEmpty()) {
            System.out.println("File kosong!");
            return;
        }

        Board board = InputHandler.parseBoard(lines);
        List<Block> blocks = InputHandler.parseBlock(lines);

        BruteForce bf = new BruteForce(board, blocks);
        OutputHandler output = bf.solve();

        Scanner scanner = new Scanner(System.in);
        System.out.print("\nApakah ingin menyimpan solusi? (ya/tidak): ");
        String saveOption = scanner.nextLine().trim().toLowerCase();

        if (saveOption.equals("ya")) {
            System.out.print("Masukkan cukup nama file untuk menyimpan solusi (tanpa .txt): ");
            String fileName = scanner.nextLine().trim();

            String outputPath = "test/output/" + fileName + ".txt";
            boolean saved = output.saveSolution(outputPath, board);
        }
    }
}

```

```

        if (saved) {
            System.out.println("Solusi berhasil disimpan di: " +
outputPath);
        } else {
            System.out.println("Gagal menyimpan solusi.");
        }
    } else {
        System.out.println("Solusi tidak disimpan.");
    }
}
}

```

4.3.7. IQPuzzlerGUI.java

```

package gui;

import algorithm.BruteForce;
import models.Block;
import models.Board;
import utils.InputHandler;
import utils.OutputHandler;

import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.JTableHeader;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import java.util.List;

public class IQPuzzlerGUI extends JFrame {
    private JPanel panelMain;
    private JTable outputArea;
    private JTextField txtInputPath;
    private JButton solveButton;
    private JTextField txtOutputPath;
    private JButton saveButton;
    private JLabel labelStatistik;
    private JLabel labelSolved;
    private JLabel labelTitle;
    private JPanel panelInput;
    private JPanel panelOutput;
    private JScrollPane scrollTable;

    private Board board;
    private List<Block> blocks;
    private boolean solved;
    private int time;
}

```

```

private int iterations;

public IQPuzzlerGUI() {
    setContentPane(panelMain);
    Font titleFont = new Font("SansSerif", Font.BOLD, 20);
    Font descFont = new Font("SansSerif", Font.PLAIN, 12);
    labelTitle.setFont(titleFont);
    setTitle("Andi Frame's IQ Puzzler Pro GUI");
    setSize(1200, 800);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setVisible(true);
    outputArea.setDefaultRenderer(Object.class, new NewCellRenderer());
    panelMain.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

    solveButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            labelSolved.setText("Loading...");
            solveButton.setEnabled(false);
            saveButton.setEnabled(false);

            SwingWorker<Void, Void> worker = new SwingWorker<Void, Void>() {
                @Override
                protected Void doInBackground() throws Exception {
                    File inputFile =
InputHandler.promptFileGUI(txtInputPath.getText());
                    if (inputFile == null) {
                        JOptionPane.showMessageDialog(null, "File tidak
ditemukan!", "Error", JOptionPane.ERROR_MESSAGE);
                        labelSolved.setText("Status Idle");
                        solved = false;
                        return null;
                    }

                    java.util.List<String> lines =
InputHandler.readFile(inputFile);
                    if (lines.isEmpty()) {
                        JOptionPane.showMessageDialog(null, "File kosong!",
"Error", JOptionPane.ERROR_MESSAGE);
                        labelSolved.setText("Status Idle");
                        solved = false;
                        return null;
                    }

                    board = InputHandler.parseBoard(lines);
                    blocks = InputHandler.parseBlock(lines);

                    if (board == null){

```

```

        JOptionPane.showMessageDialog(null, "Terjadi masalah
pada file input! \nJumlah blok/pieces hanya bisa kurang dari 26.", "Error",
JOptionPane.ERROR_MESSAGE);
        labelSolved.setText("Status Idle");
        solved = false;
        return null;
    }

    if (blocks.isEmpty()) {
        JOptionPane.showMessageDialog(null, "Terjadi masalah
pada pembacaan blok! \nSimbol blok tidak boleh berulang.", "Error",
JOptionPane.ERROR_MESSAGE);
        labelSolved.setText("Status Idle");
        solved = false;
        return null;
    }

    BruteForce bf = new BruteForce(board, blocks);
    long startTime = System.currentTimeMillis();
    solved = bf.solveRecursive(0);
    long endTime = System.currentTimeMillis();
    time = (int) (endTime - startTime);
    iterations = bf.getIterations();

    return null;
}

@Override
protected void done() {
    solveButton.setEnabled(true);
    saveButton.setEnabled(true);
    if (solved) {
        labelSolved.setText("Solusi Ditemukan!\n");
        updateTable(board);
    } else {
        labelSolved.setText("Solusi tidak ditemukan!\n");
    }
    labelStatistik.setText(time + "ms dengan " + iterations
+ " percobaan.");
    labelStatistik.setFont(descFont);
}

};

worker.execute();
}

});

saveButton.addActionListener(new ActionListener() {

```

```

        @Override
        public void actionPerformed(ActionEvent e) {
            if (board == null) {
                JOptionPane.showMessageDialog(null, "Tidak ada solusi untuk
disimpan!", "Error", JOptionPane.ERROR_MESSAGE);
                return;
            }

            String fileName = txtOutputPath.getText();
            String filePath = "test/output/" + fileName;
            OutputHandler outputHandler = new OutputHandler(solved, time,
iterations);
            boolean saved = outputHandler.saveSolution(filePath, board);

            if (saved) {
                JOptionPane.showMessageDialog(null, "Solusi berhasil
disimpan di: " + filePath, "Info", JOptionPane.INFORMATION_MESSAGE);
            } else {
                JOptionPane.showMessageDialog(null, "Gagal menyimpan
solusi.", "Error", JOptionPane.ERROR_MESSAGE);
            }
        }
    });
}

private void updateTable(Board board) {
    DefaultTableModel model = new DefaultTableModel(board.getRows(),
board.getCols());

    for (int i = 0; i < board.getRows(); i++) {
        for (int j = 0; j < board.getCols(); j++) {
            char cell = board.getCell(i, j);
            model.setValueAt(cell == 0 ? "" : cell, i, j);
        }
    }

    outputArea.setModel(model);

    JTableHeader header = outputArea.getTableHeader();
    header.setVisible(false);

    outputArea.setRowHeight(30);
    for (int i = 0; i < outputArea.getColumnCount(); i++) {
        outputArea.getColumnModel().getColumn(i).setPreferredWidth(30);
    }
}

public static void main(String[] args) {
    SwingUtilities.invokeLater(() -> new IQPuzzlerGUI().setVisible(true));
}

```



```
}  
}
```

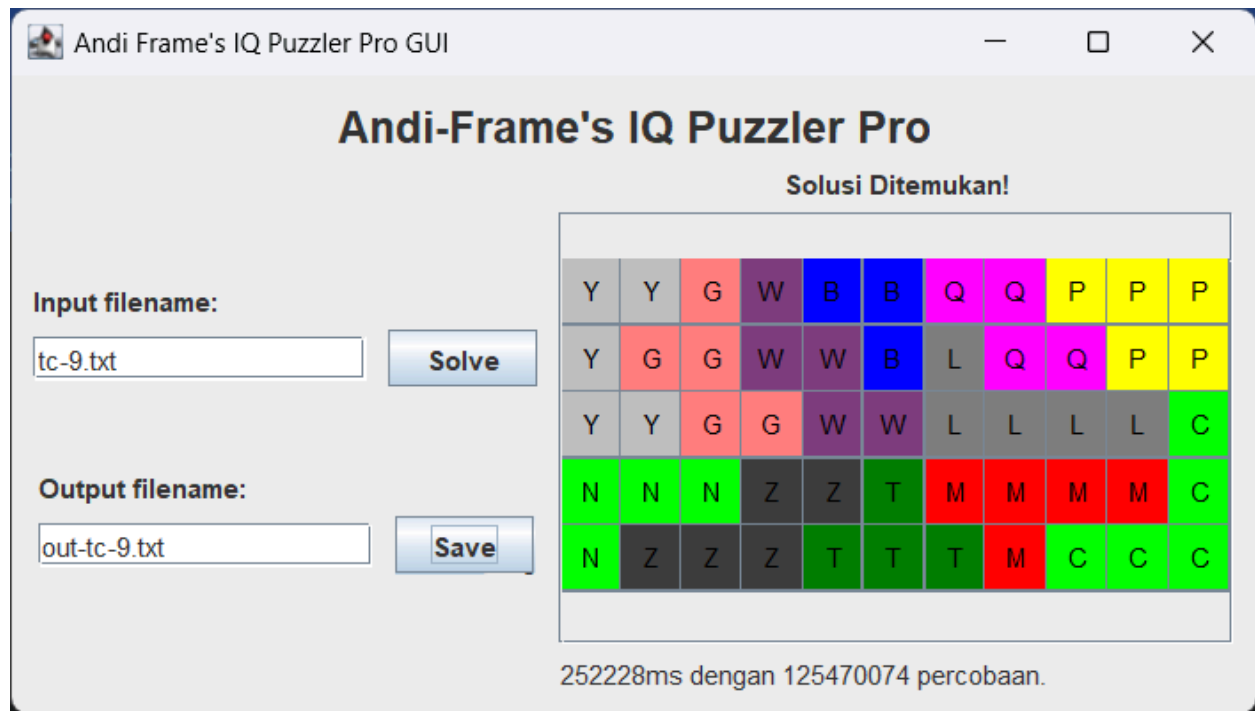
4.3.8. NewCellRenderer.java

```
package gui;  
  
import javax.swing.*;  
import javax.swing.table.DefaultTableCellRenderer;  
import java.awt.*;  
  
public class NewCellRenderer extends DefaultTableCellRenderer {  
    public NewCellRenderer() {  
        setHorizontalAlignment(JLabel.CENTER);  
    }  
  
    @Override  
    public Component getTableCellRendererComponent(JTable table, Object value,  
boolean isSelected, boolean hasFocus, int row, int column) {  
        Component cell = super.getTableCellRendererComponent(table, value,  
isSelected, hasFocus, row, column);  
  
        if (value != null) {  
            char blockSymbol = value.toString().charAt(0);  
            Color color = getColorForBlock(blockSymbol);  
            cell.setBackground(color);  
            cell.setForeground(Color.BLACK);  
        } else {  
            cell.setBackground(Color.WHITE);  
            cell.setForeground(Color.BLACK);  
        }  
  
        return cell;  
    }  
  
    private Color getColorForBlock(char blockSymbol) {  
        return switch (blockSymbol) {  
            case 'A' -> Color.RED;  
            case 'B' -> Color.BLUE;  
            case 'C' -> Color.GREEN;  
            case 'D' -> Color.YELLOW;  
            case 'E' -> Color.ORANGE;  
            case 'F' -> new Color(128, 128, 255);  
            case 'G' -> new Color(255, 128, 128);  
            case 'H' -> new Color(128, 255, 128);  
            case 'I' -> new Color(128, 0, 128);  
            case 'J' -> new Color(0, 128, 128);  
            case 'K' -> new Color(128, 128, 0);  
            case 'L' -> new Color(128, 128, 128);  
        };  
    }  
}
```

```
    case 'M' -> new Color(255, 0, 0);
    case 'N' -> new Color(0, 255, 0);
    case 'O' -> new Color(0, 0, 255);
    case 'P' -> new Color(255, 255, 0);
    case 'Q' -> new Color(255, 0, 255);
    case 'R' -> new Color(0, 255, 255);
    case 'S' -> new Color(128, 0, 0);
    case 'T' -> new Color(0, 128, 0);
    case 'U' -> new Color(0, 0, 128);
    case 'V' -> new Color(128, 128, 64);
    case 'W' -> new Color(128, 64, 128);
    case 'X' -> new Color(64, 128, 128);
    case 'Y' -> new Color(192, 192, 192);
    case 'Z' -> new Color(64, 64, 64);
    default -> Color.WHITE;
};
}
}
```

BAB V: Pengujian


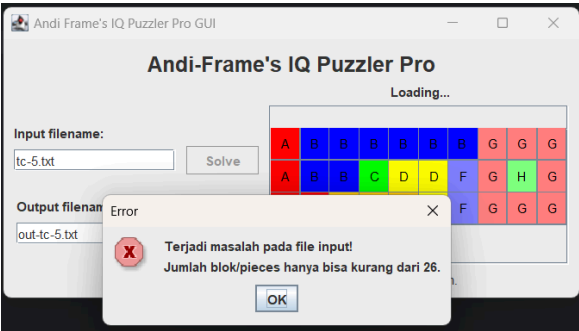
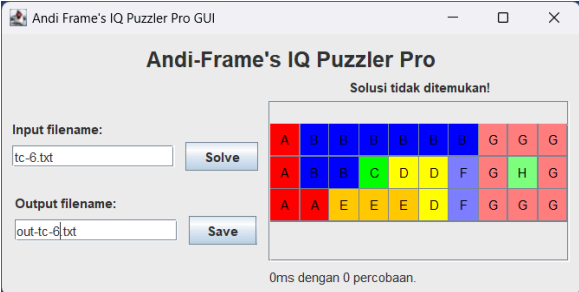
5.1. Tampilan Aplikasi

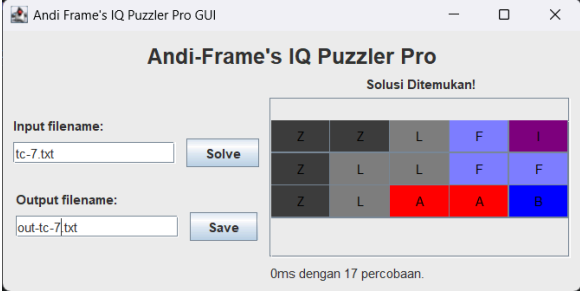
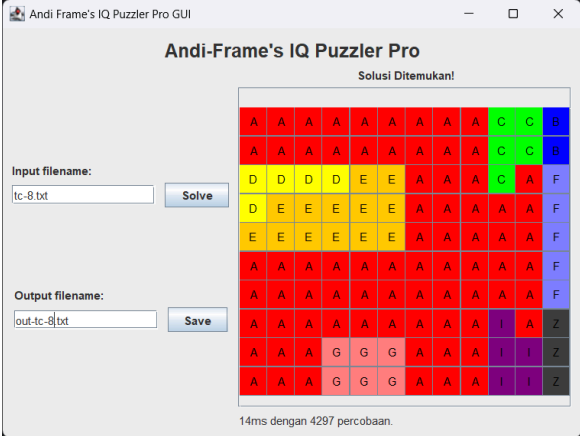


5.2. Pengujian

No.	Masukan	Keluaran .txt	Tampilan
1.	<pre> 5 11 12 DEFAULT A A A AA A B B BBB CCCC C D DD DD E EEE E F FFFF GGG G HHH HH I </pre>	<pre> Solusi Ditemukan! A K K C C C C D G G G A K B C B E E D D G F A A B B B L E E D D F J A H H H L E I I F F J J J H H L L L I I F Waktu pencarian: 2001381ms Banyak kasus: 940079150 </pre>	<p>The screenshot shows the 'Andi-Frame's IQ Puzzler Pro GUI' window. The title bar reads 'Andi Frame's IQ Puzzler Pro GUI'. The main window has a title 'Andi-Frame's IQ Puzzler Pro' and a subtitle 'Solusi Ditemukan!'. On the left, there are two input fields: 'Input filename:' with 'tc-9.txt' and 'Output filename:' with 'out-tc-9.txt'. Below these are 'Solve' and 'Save' buttons. On the right, a 5x11 grid of colored squares contains the solution. Below the grid, it says '2001381ms dengan 940079150 percobaan.'</p>

	<pre>II I JJ J KK K L L LLL</pre>		
2.	<pre>5 5 7 DEFAULT A AA B BB C CC D DD EE EE E FF FF F GGG</pre>	<pre>Solusi Ditemukan! A G G G C A A B C C E E B B F E E D F F E D D F F Waktu pencarian: 3ms Banyak kasus: 7166</pre>	
3.	<pre>5 5 7 DEFAULT A AA B BB A AA D DD EE EE E FF FF F GGG</pre>	<pre>Solusi tidak ditemukan! Waktu pencarian: 0ms Banyak kasus: 0</pre>	

4.	<pre> 3 10 8 DEFAULT A A AA BB BB B B B B C DD D E E E FF GGG G G GGG H </pre>	<pre> Solusi Ditemukan! A B B B B B G G A B B C D D F G H G A A E E E D F G G G Waktu pencarian: 46ms Banyak kasus: 249542 </pre>	
5.	<pre> 3 10 27 DEFAULT A A AA BB BB B B B B C DD D E E E FF GGG G G GGG H </pre>		
6.	<pre> 2 2 2 DEFAULT A A AA BB </pre>	<pre> Solusi tidak ditemukan! Waktu pencarian: 0ms Banyak kasus: 0 </pre>	

7.	<pre>3 5 6 DEFAULT ZZ Z Z F FF I L LL L A A B</pre>	<pre>Solusi Ditemukan! Z Z L F I Z L L F F Z L A A B Waktu pencarian: 0ms Banyak kasus: 17</pre>	
8.	<pre>10 12 9 DEFAULT AAAAAAA AA AAAAAAA AA AAA A AAAAA AAAAA AAAAA AAAAA AAAAA AAAAA AA A AAA AAA AAA AAA AAA B B CC CC C D D D DD EE EEEE EEEEEE F F F F F II II</pre>	<pre>Solusi Ditemukan! A A A A A A A A A C C B A A A A A A A A A C C B D D D D E E A A A C A F D E E E E E A A A A A F E E E E E E A A A A A F A A A A A A A A A A A F A A A A A A A A A I A Z A A A G G G A A A I I Z A A A G G G A A A I I Z Waktu pencarian: 14ms Banyak kasus: 4297</pre>	

	<div>I GG GG GG ZZZ</div>																																																									
9.	<div>5 11 12 DEFAULT YY Y YY W WW WW L LLLL Z ZZ Z Z C C CCC TTT T M MMM G GG GG QQ QQ BB B PP PPP N NNN</div>	<div>Solusi Ditemukan! Y Y G W B B Q Q P P P Y G G W W B L Q Q P P Y Y G G W W L L L L C N N N Z Z T M M M M C N Z Z Z T T T M C C C Waktu pencarian: 252228ms Banyak kasus: 125470074</div>	<div><div>Andi Frame's IQ Puzzler Pro GUI</div><div>Andi-Frame's IQ Puzzler Pro</div><div>Solusi Ditemukan!</div><div><div>Input filename: tc-9.txt</div><div>Solve</div><div>Output filename: out-tc-9.txt</div><div>Save</div></div><div><table><tr><td>Y</td><td>Y</td><td>G</td><td>W</td><td>B</td><td>B</td><td>Q</td><td>Q</td><td>P</td><td>P</td><td>P</td></tr><tr><td>Y</td><td>G</td><td>G</td><td>W</td><td>W</td><td>B</td><td>L</td><td>Q</td><td>Q</td><td>P</td><td>P</td></tr><tr><td>Y</td><td>Y</td><td>G</td><td>G</td><td>W</td><td>W</td><td>L</td><td>L</td><td>L</td><td>L</td><td>C</td></tr><tr><td>N</td><td>N</td><td>N</td><td>Z</td><td>Z</td><td>T</td><td>M</td><td>M</td><td>M</td><td>M</td><td>C</td></tr><tr><td>N</td><td>Z</td><td>Z</td><td>Z</td><td>T</td><td>T</td><td>T</td><td>M</td><td>C</td><td>C</td><td>C</td></tr></table></div><div>252228ms dengan 125470074 percobaan.</div></div>	Y	Y	G	W	B	B	Q	Q	P	P	P	Y	G	G	W	W	B	L	Q	Q	P	P	Y	Y	G	G	W	W	L	L	L	L	C	N	N	N	Z	Z	T	M	M	M	M	C	N	Z	Z	Z	T	T	T	M	C	C	C
Y	Y	G	W	B	B	Q	Q	P	P	P																																																
Y	G	G	W	W	B	L	Q	Q	P	P																																																
Y	Y	G	G	W	W	L	L	L	L	C																																																
N	N	N	Z	Z	T	M	M	M	M	C																																																
N	Z	Z	Z	T	T	T	M	C	C	C																																																

Lampiran

A. Checklist Spesifikasi Program

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

B. Pranala Repository Kode Program

Program dalam laporan ini dapat diakses melalui pranala berikut:

https://github.com/andi-frame/Tucil1_13523128