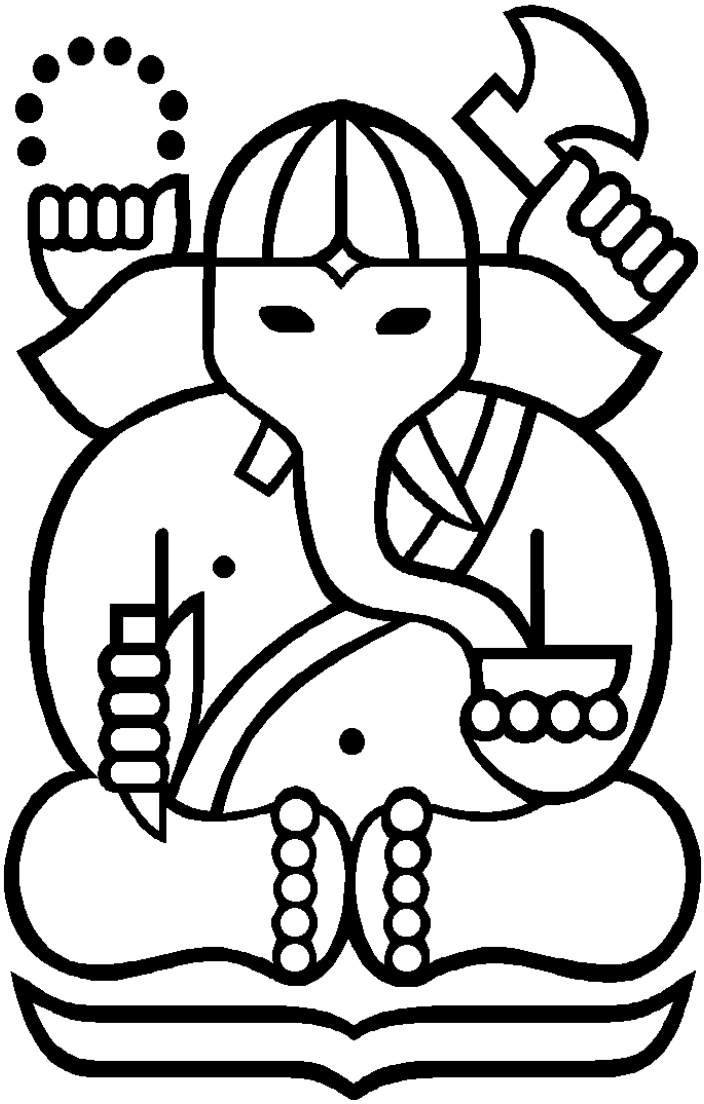
Tugas Kecil 1 IF2211 Strategi Algoritma

Penyelesain IQ Puzzler Pro dengan Algoritma Brute Force



Disusun oleh:

Orvin Andika Ikhsan Abhista – 13523017

INSTITUT TEKNOLOGI BANDUNG
2025

DAFTAR ISI

BAB 1

Deskripsi Masalah dan Algoritma

1.1. Algoritma Brute Force

Algoritma Brute Force adalah algoritma penyelesaian masalah secara *straightforward* dengan mencoba semua kemungkinan penyelesaian satu per satu hingga menemukan solusi yang benar. Proses ini dilakukan tanpa mempertimbangkan cara yang lebih efisien atau pola tertentu.

Algoritma Brute Force memecahkan persoalan secara sangat sederhana, langsung, jelas caranya, dan dipahami. Algoritma ini akan mengeksplorasi seluruh kemungkinan yang ada sehingga jawaban yang benar pasti akan ditemukan walaupun memakan waktu yang sangat lama dan memakan daya komputasi yang besar.

1.2. IQ Puzzler Pro

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.

1.3. Algoritma Penyelesaian IQ Puzzler Pro dengan Algoritma Brute Force

Berikut adalah langkah-langkah penyelesaian permainan IQ Puzzler Pro menggunakan Algoritma Brute Force:

1. Pengguna memasukkan input berupa file teks (berekstensi .txt).
2. Program akan membaca dimensi papan, jumlah blok, bentuk papan, dan blok-blok yang digunakan dalam permainan.
3. Program akan menghasilkan blok baru hasil dari rotasi dan pencerminan semua blok yang ada.

4. Program akan menghasilkan semua kombinasi dari urutan penempatan blok.
5. Pada semua urutan, program akan mencoba menempatkan semua blok dalam papan. Jika semua blok berhasil ditempatkan dan papan penuh maka solusi ditemukan. Jika tidak, program akan mencoba urutan selanjutnya. Jika sampai urutan terakhir syarat tersebut belum terpenuhi maka permainan dianggap tidak memiliki solusi.
6. Jika solusi ada dan pengguna memilih opsi simpan sebagai txt maka program akan menyimpan solusi dalam file txt (berekstensi .txt).
7. Jika solusi ada dan pengguna memilih opsi simpan sebagai gambar maka program akan menyimpan solusi sebagai gambar dengan ekstensi .png.

BAB 2

Implementasi Algoritma

Algoritma ini diimplementasikan dalam bahasa pemrograman Java. Struktur dari program ini dibagi menjadi 2 folder dan 5 file yaitu folder application, folder source, file Main.java, file Input.java, file Solver.java, Output.java,dan file OutputImage.java

2.1. Main.java

File Main.java adalah file utama yang berisi GUI dan pemanggilan terhadap fungsi-fungsi lain.

Fungsi	Deskripsi
`start`	Memuat GUI
`chooseFile`	Mengambil input file dari pengguna
`loadInput`	Memuat input dari file yang dipilih
`runSolver`	Memanggil fungsi untuk mencari solusi berdasarkan input
`saveSolutionToTxt`	Memanggil fungsi untuk menyimpan solusi sebagai file teks
`saveSolutionToImage`	Memanggil fungsi untuk menyimpan solusi sebagai file gambar

2.2. Input.java

File Input.java berisi kode yang digunakan untuk menerima dan memproses input pengguna.

Fungsi	Deskripsi
`Input`	Memuat input pengguna
`readInput`	Menerima dan memproses input
`parseBlock`	Memproses input blok
`blockAlternatives`	Menambahkan semua alternatif bentuk blok
`rotate`	Merotasi blok
`mirror`	Mencerminkan blok
`toString`	Mengubah block menjadi string
`copy`	Menyalin blok

2.3. Solver.java

File Solver.java berisi kode yang digunakan untuk mencari solusi permainan IQ Puzzler Pro.

Fungsi	Deskripsi
`Solver`	Memuat solusi
`getBoard`	Mengembalikan papan permainan
`solve`	Mencari solusi permainan
`placeBlock`	Menempatkan semua blok dalam satu urutan
`canPlace`	Merotasi blok
`setBlock`	Menempatkan satu blok
`generatePermutations`	Menghasilkan permutasi urutan penempatan blok
`permute`	Melakukan permutasi
`isBoardFull`	Mengecek apakah papan sudah penuh
`isSolutionFound`	Mengecek apakah solusi sudah ditemukan
`getSolution`	Mengembalikan solusi
`getAttempt`	Mengembalikan jumlah kasus yang ditinjau
`setAttempt`	Mengatur nilai attempt

2.4. Output.java

File Output.java berisi kode untuk memproses dan menampilkan hasil algoritma.

Fungsi	Deskripsi
`saveToFile`	Menyimpan solusi menjadi file teks
`getColoredSolution`	Mengembalikan solusi dengan warna yang berbeda tiap blok

2.5. OutputImage.java

File OutputImage.java berisi kode untuk memproses dan menyimpan solusi menjadi gambar.

Fungsi	Deskripsi
`saveSolutionAsImage`	Menyimpan solusi menjadi file gambar

BAB 3

Source Code Program

3.1. Repository

Repository dapat diakses pada tautan: https://github.com/orvin14/Tucil1_13523017

3.2. Source Code

3.2.1. Main.java

```
package application;

import javafx.application.Application;
import javafx.fxml.FXML;
import javafx.geometry.Pos;
import javafx.stage.FileChooser;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.image.Image;
import javafx.scene.layout.Background;
import javafx.scene.layout.BackgroundFill;
import javafx.scene.layout.CornerRadii;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.text.Text;
import javafx.scene.text.TextFlow;
import javafx.geometry.Insets;
import java.io.File;
import java.io.IOException;
import Source.Input;
import Source.Solver;
import Source.Output;
import Source.OutputImage;

public class Main extends Application {
    private TextArea outputArea;
    private Button btnSaveTxt, btnSaveImg, btnSolve;
    private Solver solver;
    private VBox outputBox;
    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("IQ Puzzler Solver");
        Image icon = new Image("idea.png");
        primaryStage.getIcons().add(icon);
        outputBox = new VBox();
        outputBox.setSpacing(5);

        outputBox.setBackground(new Background(new BackgroundFill(
```

```
                Color.WHITE, CornerRadii.EMPTY, Insets.EMPTY
            )));

        Button btnOpenFile = new Button("File...");
        btnSolve = new Button("Solve");
        btnSaveTxt = new Button("Simpan ke TXT");
        btnSaveImg = new Button("Simpan ke Gambar");
        outputArea = new TextArea();
        outputArea.setEditable(false);
        outputBox.getChildren().add(outputArea);

        btnSolve.setDisable(true);
        btnSaveTxt.setDisable(true);
        btnSaveImg.setDisable(true);

        final String[] filePath = {null};

        btnOpenFile.setOnAction(e -> {
            File file = chooseFile(primaryStage);
            if (file != null) {
                filePath[0] = file.getAbsolutePath();
                try {
                    loadInput(filePath[0]);
                    btnSolve.setDisable(false);
                } catch (Exception ex) {
                    outputArea.setText("Error: " +
ex.getMessage());
                    ex.printStackTrace();
                }
            }
        });

        btnSolve.setOnAction(e -> {
           outputBox.getChildren().remove(outputArea);
           solver.setAttempt(0);
            if (filePath[0] != null) {
                try {
                    runSolver(filePath[0]);
                } catch (Exception ex) {
                    outputArea.setText("Error: " +
ex.getMessage());
                    ex.printStackTrace();
                }
            }
        });

        btnSaveTxt.setOnAction(e ->
saveSolutionToTxt(primaryStage));
        btnSaveImg.setOnAction(e ->
saveSolutionToImage(primaryStage));
        HBox saveButtons = new HBox(10, btnSaveTxt, btnSaveImg);
        saveButtons.setAlignment(Pos.CENTER);
        VBox root = new VBox(10, btnOpenFile, btnSolve,
outputBox,saveButtons);
        Scene scene = new Scene(root, 600, 400);
```

```
        primaryStage.setScene(scene);
        primaryStage.show();
    }

    private File chooseFile(Stage stage) {
        FileChooser fileChooser = new FileChooser();
        fileChooser.setTitle("Pilih File Input");
        fileChooser.getExtensionFilters().add(new
FileChooser.ExtensionFilter("Text Files", "*.txt"));
        return fileChooser.showOpenDialog(stage);
    }

    private void loadInput(String filePath) throws IOException {
        Input input = new Input(filePath);
    }

    private void runSolver(String filePath) throws IOException {
        Input input = new Input(filePath);
        solver = new Solver(input);

        long start = System.currentTimeMillis();
        solver.solve();
        long end = System.currentTimeMillis();
        long waktu = end - start;
        if (solver.isSolutionFound()) {
           outputBox.getChildren().clear();
            TextFlow formattedText =
Output.getColoredSolution(solver.getBoard());
            outputBox.getChildren().clear();
            outputBox.getChildren().add(formattedText);

            btnSaveTxt.setDisable(false);
            btnSaveImg.setDisable(false);
        } else {
           Label noSolution = new Label("\nTidak ada solusi yang
ditemukan.");
            outputBox.getChildren().clear();
            outputBox.getChildren().add(noSolution);
        }
        Label time = new Label("\nWaktu Eksekusi: " + waktu + "
ms");
        Label kasus = new Label("\nPercobaan: " +
Solver.getAttempt());
        outputBox.getChildren().add(time);
        outputBox.getChildren().add(kasus);
    }

    private void saveSolutionToTxt(Stage stage) {
        if (solver == null || !solver.isSolutionFound()) {
            outputArea.appendText("\nTidak ada solusi untuk
disimpan.");
            return;
        }

        FileChooser fileChooser = new FileChooser();
```

```
        fileChooser.setTitle("Simpan Solusi ke TXT");
        fileChooser.getExtensionFilters().add(new
FileChooser.ExtensionFilter("Text Files", "*.txt"));
        File file = fileChooser.showSaveDialog(stage);

        if (file != null) {
            Output.saveToFile(file.getAbsolutePath(),
solver.getSolution());
            outputArea.appendText("\nSolusi disimpan ke " +
file.getAbsolutePath());
        }
    }

    private void saveSolutionToImage(Stage stage) {
        if (solver == null || !solver.isSolutionFound()) {
            outputArea.appendText("\nTidak ada solusi untuk
disimpan.");
            return;
        }

        FileChooser fileChooser = new FileChooser();
        fileChooser.setTitle("Simpan Solusi ke Gambar");
        fileChooser.getExtensionFilters().add(new
FileChooser.ExtensionFilter("PNG Images", "*.png"));
        File file = fileChooser.showSaveDialog(stage);

        if (file != null) {
            OutputImage.saveSolutionAsImage(Solver.getBoard(),
file.getAbsolutePath());
            outputArea.appendText("\nSolusi disimpan sebagai
gambar di " + file.getAbsolutePath());
        }
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

3.2.2. Input.java

```
package Source;

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

public class Input {
    public int rows, cols, blockCount;
    private char[][] board;
    public List<List<char[][]>> blocks;
    public List<Character> blockType;

    public Input(String filename) throws IOException {
        readInput(filename);
```

```
        board = new char[rows][cols];
        for (char[] row : board) Arrays.fill(row, '.');
    }

    private void readInput(String filename) throws IOException {
    BufferedReader br = new BufferedReader(new
FileReader(filename));
    String line = br.readLine();

    if (line == null) {
        br.close();
        throw new IOException("Empty input file");
    }
    StringTokenizer st = new StringTokenizer(line);
    rows = Integer.parseInt(st.nextToken());
    cols = Integer.parseInt(st.nextToken());
    blockCount = Integer.parseInt(st.nextToken());
    blocks = new ArrayList<>();
    blockType = new ArrayList<>();

    Map<Character, List<String>> blockShapes = new
LinkedHashMap<>();

    line = br.readLine();
    while ((line = br.readLine()) != null) {
        line = line.trim();
        if (line.isEmpty()) continue;
        char blockChar = line.charAt(0);
        blockShapes.putIfAbsent(blockChar, new ArrayList<>());
        blockShapes.get(blockChar).add(line);
    }
    br.close();

    for (Map.Entry<Character, List<String>> entry :
blockShapes.entrySet()) {
        char blockChar = entry.getKey();
        List<String> shapeLines = entry.getValue();
        blockType.add(blockChar);
        blocks.add(blockAlternatives(parseBlock(shapeLines,
blockChar)));
    }
    if (blockType.size() != blockCount) {
        System.out.println("Error: Expected " + blockCount + "
blocks, but found " + blockType.size());
        System.exit(1);
    }
}

    private char[][] parseBlock(List<String> shapeLines, char
blockChar) {
    int minRow = Integer.MAX_VALUE, maxRow = Integer.MIN_VALUE;
    int minCol = Integer.MAX_VALUE, maxCol = Integer.MIN_VALUE;

    for (int r = 0; r < shapeLines.size(); r++) {
        String row = shapeLines.get(r);
```

```
        for (int c = 0; c < row.length(); c++) {
            if (row.charAt(c) == blockChar) {
                minRow = Math.min(minRow, r);
                maxRow = Math.max(maxRow, r);
                minCol = Math.min(minCol, c);
                maxCol = Math.max(maxCol, c);
            }
        }
    }

    if (minRow == Integer.MAX_VALUE) {
        System.out.println("Error: Block " + blockChar + " not
found.");
        return new char[0][0];
    }
    int blockRows = maxRow - minRow + 1;
    int blockCols = maxCol - minCol + 1;
    char[][] block = new char[blockRows][blockCols];

    for (char[] row : block) Arrays.fill(row, '.');
    for (int r = minRow; r <= maxRow; r++) {
        String row = shapeLines.get(r);
        for (int c = minCol; c <= Math.min(maxCol, row.length() -
1); c++) {
            if (row.charAt(c) == blockChar) {
                block[r - minRow][c - minCol] = blockChar;
            }
        }
    }

    return block;
}

    private List<char[][]> blockAlternatives(char[][] block) {
        Set<String> seen = new HashSet<>();
        List<char[][]> alternatives = new ArrayList<>();

        String original = toString(block);
        seen.add(original);
        alternatives.add(copy(block));

        for (int i = 0; i < 3; i++) {
            block = rotate(block);
            String rotated = toString(block);
            if (!seen.contains(rotated)) {
                seen.add(rotated);
                alternatives.add(copy(block));
            }
            char[][] mirrored = mirror(block);
            String mirroredBlock = toString(mirrored);
            if (!seen.contains(mirroredBlock)) {
                seen.add(mirroredBlock);
                alternatives.add(copy(mirrored));
            }
        }
```

```
        return alternatives;
    }

    private char[][] rotate(char[][] block) {
        int r = block.length, c = block[0].length;
        char[][] rotated = new char[c][r];
        for (int i = 0; i < r; i++) {
            for (int j = 0; j < c; j++) {
                rotated[j][r - 1 - i] = block[i][j];
            }
        }
        return rotated;
    }

    private char[][] mirror(char[][] block) {
        int r = block.length, c = block[0].length;
        char[][] mirrored = new char[r][c];
        for (int i = 0; i < r; i++) {
            for (int j = 0; j < c; j++) {
                mirrored[i][c - 1 - j] = block[i][j];
            }
        }
        return mirrored;
    }

    private String toString(char[][] block) {
        StringBuilder sb = new StringBuilder();
        for (char[] row : block) sb.append(new
String(row)).append("|");
        return sb.toString();
    }

    private char[][] copy(char[][] block) {
        char[][] newBlock = new
char[block.length][block[0].length];
        for (int i = 0; i < block.length; i++) {
            System.arraycopy(block[i], 0, newBlock[i], 0,
block[i].length);
        }
        return newBlock;
    }
}
```

3.2.3. Solver.java

```
package Source;

import java.util.*;

public class Solver {
    private int rows, cols, blockCount;
    private static char[][] board;
    private List<List<char[][]>> blocks;
    private List<Character> blockType;
    private static int attempt = 0;
```

```
    private static boolean found = false;

    public Solver(Input input) {
        this.rows = input.rows;
        this.cols = input.cols;
        this.blockCount = input.blockCount;
        this.blocks = input.blocks;
        this.blockType = input.blockType;
        this.board = new char[rows][cols];
        for (char[] row : board) Arrays.fill(row, '.');
    }

    public static char[][] getBoard() {
        return board;
    }

    public void solve() {
        List<Integer> order = new ArrayList<>();
        for (int i = 0; i < blockCount; i++) order.add(i);
        List<List<Integer>> permutations =
generatePermutations(order);
        for (List<Integer> perm : permutations) {
            if (placeBlock(perm)&&isBoardFull()) {
                found = true;
                return;
            }
        }
        System.out.println("Tidak ada solusi yang ditemukan.");
    }

    private boolean placeBlock(List<Integer> order) {
        char[][] tempBoard = new char[rows][cols];
        for (char[] row : tempBoard) Arrays.fill(row, '.');
        for (int index = 0; index < blockCount; index++) {
            boolean placed = false;
            char label = blockType.get(order.get(index));
            for (char[][] b : blocks.get(order.get(index))) {
                for (int row = 0; row < rows; row++) {
                    for (int col = 0; col < cols; col++) {
                        attempt++;
                        if (canPlace(tempBoard, b, row, col)) {
                            setBlock(tempBoard, b, row, col,
label);
                            placed = true;
                            break;
                        }
                    }
                    if (placed) break;
                }
                if (placed) break;
            }
            if (!placed) return false;
        }
        board = tempBoard;
        return true;
```

```
    }

    private boolean canPlace(char[][] tempBoard, char[][] block,
int row, int col) {
        int br = block.length;
        int bc = block[0].length;
        if (row + br > rows || col + bc > cols) return false;
        for (int i = 0; i < br; i++) {
            for (int j = 0; j < bc; j++) {
                if (block[i][j] != '.' && tempBoard[row + i][col +
j] != '.') return false;
            }
        }
        return true;
    }

    private void setBlock(char[][] tempBoard, char[][] block, int
row, int col, char character) {
        for (int i = 0; i < block.length; i++) {
            for (int j = 0; j < block[i].length; j++) {
                if (block[i][j] != '.') {
                    tempBoard[row + i][col + j] = character;
                }
            }
        }
    }

    private List<List<Integer>> generatePermutations(List<Integer>
list) {
        List<List<Integer>> result = new ArrayList<>();
        permute(list, 0, result);
        return result;
    }

    private void permute(List<Integer> arr, int l,
List<List<Integer>> result) {
        if (l == arr.size()) {
            result.add(new ArrayList<>(arr));
            return;
        }
        for (int i = l; i < arr.size(); i++) {
            Collections.swap(arr, l, i);
            permute(arr, l + 1, result);
            Collections.swap(arr, l, i);
        }
    }
    private boolean isBoardFull() {
        for (char[] row : board) {
            for (char cell : row) {
                if (cell == '.') {
                    return false;
                }
            }
        }
        return true;
```

```
    }

    public static boolean isSolutionFound() {
        return found;
    }

    public static String getSolution() {
        StringBuilder sb = new StringBuilder();
        for (char[] row : board) {
            sb.append(new String(row)).append("\n");
        }
        return sb.toString();
    }

    public static int getAttempt() {
        return attempt;
    }
    public static void setAttempt(int value) {
        attempt = value;
    }


}
```

3.2.4. Output.java

```
package Source;

import java.awt.*;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;
import javax.imageio.ImageIO;
import javafx.scene.paint.Color;
import javafx.scene.text.Text;
import javafx.scene.text.TextFlow;


public class Output {
    private static final Map<Character, Color> blockColors = new
HashMap<>();
    static {
        Color[] colors = {
            Color.RED, Color.GREEN, Color.BLUE, Color.YELLOW,
Color.MAGENTA, Color.CYAN,
            Color.ORANGE, Color.PURPLE, Color.TEAL,
Color.DEEPPINK, Color.LIMEGREEN,
            Color.DARKTURQUOISE, Color.DARKMAGENTA,
Color.DARKGOLDENROD, Color.AQUA,
            Color.CRIMSON, Color.GOLD, Color.ORCHID
        };
```

```
        for (int i = 0; i < 26; i++) {
            blockColors.put((char) ('A' + i), colors[i %
colors.length]);
        }
        blockColors.put('.', Color.BLACK);
    }
    public static void saveToFile(String filename, String
solution) {
        try (FileWriter writer = new FileWriter(filename)) {
            writer.write(solution);
            System.out.println("Hasil disimpan ke: " + filename);
        } catch (IOException e) {
            System.err.println("Gagal menyimpan file: " +
e.getMessage());
        }
    }
    public static TextFlow getColoredSolution(char[][] board) {
        TextFlow textFlow = new TextFlow();

        for (char[] row : board) {
            for (char cell : row) {
                Color color = blockColors.getOrDefault(cell,
Color.BLACK);

                Text text = new Text(cell + " ");
                text.setFill(color);
                textFlow.getChildren().add(text);
            }
            textFlow.getChildren().add(new Text("\n"));
        }
        return textFlow;
    }
}
```

3.2.5. OutputImage.java

```
package Source;
import java.awt.Color;
import java.awt.Font;
import java.awt.FontMetrics;
import java.awt.Graphics2D;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.util.*;

import javax.imageio.ImageIO;


public class OutputImage{
    private static final int CELL_SIZE = 50;
    private static final int PADDING = 10;
```

```
    private static final Map<Character, Color> blockColors = new
HashMap<>();
    static {
        Color[] colors = {
            Color.RED, Color.GREEN, Color.BLUE, Color.YELLOW,
Color.MAGENTA, Color.CYAN,
            new Color(255, 165, 0),  // ORANGE
            new Color(128, 0, 128),  // PURPLE
            new Color(0, 128, 128),  // TEAL
            new Color(255, 20, 147), // DEEPPINK
            new Color(50, 205, 50),  // LIMEGREEN
            new Color(0, 206, 209),  // DARKTURQUOISE
            new Color(139, 0, 139),  // DARKMAGENTA
            new Color(184, 134, 11), // DARKGOLDENROD
            new Color(0, 255, 255),  // AQUA
            new Color(220, 20, 60),  // CRIMSON
            new Color(255, 215, 0),  // GOLD
            new Color(218, 112, 214) // ORCHID
        };

        for (int i = 0; i < 26; i++) {
            blockColors.put((char) ('A' + i), colors[i %
colors.length]);
        }
        blockColors.put('.', Color.BLACK);
    }
     public static void saveSolutionAsImage(char[][] board, String
filename) {
        int rows = board.length;
        int cols = board[0].length;

        int width = cols * CELL_SIZE + PADDING * 2;
        int height = rows * CELL_SIZE + PADDING * 2;

        BufferedImage image = new BufferedImage(width, height,
BufferedImage.TYPE_INT_RGB);
        Graphics2D g2d = image.createGraphics();

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

        for (int r = 0; r < rows; r++) {
            for (int c = 0; c < cols; c++) {
                char block = board[r][c];
                g2d.setColor(blockColors.getOrDefault(block,
Color.BLACK));
                g2d.fillRect(PADDING + c * CELL_SIZE, PADDING + r
* CELL_SIZE, CELL_SIZE, CELL_SIZE);

                g2d.setColor(Color.BLACK);
                g2d.setFont(new Font("Arial", Font.BOLD, 30));
                FontMetrics fm = g2d.getFontMetrics();
                int textX = PADDING + c * CELL_SIZE + (CELL_SIZE -
fm.charWidth(block)) / 2;
```

```
                int textY = PADDING + r * CELL_SIZE + (CELL_SIZE +
fm.getAscent()) / 2 - 5;
                g2d.drawString(String.valueOf(block), textX,
textY);
            }
        }

        g2d.dispose();

        try {
            ImageIO.write(image, "png", new File(filename));
            System.out.println("Solusi disimpan sebagai " +
filename);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

BAB 4

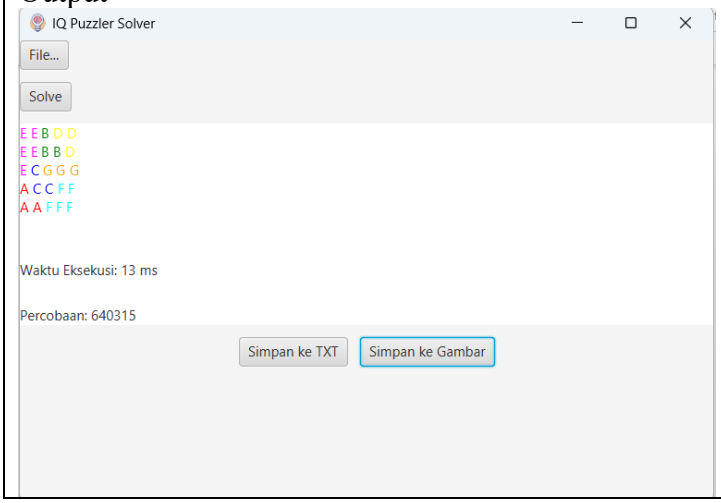
Input dan Output Program

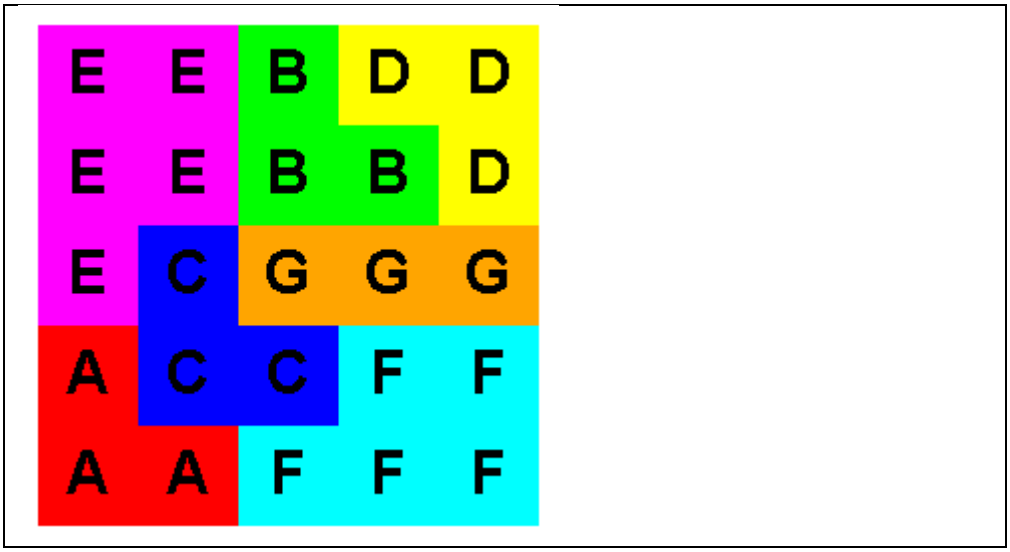
1.

Input

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

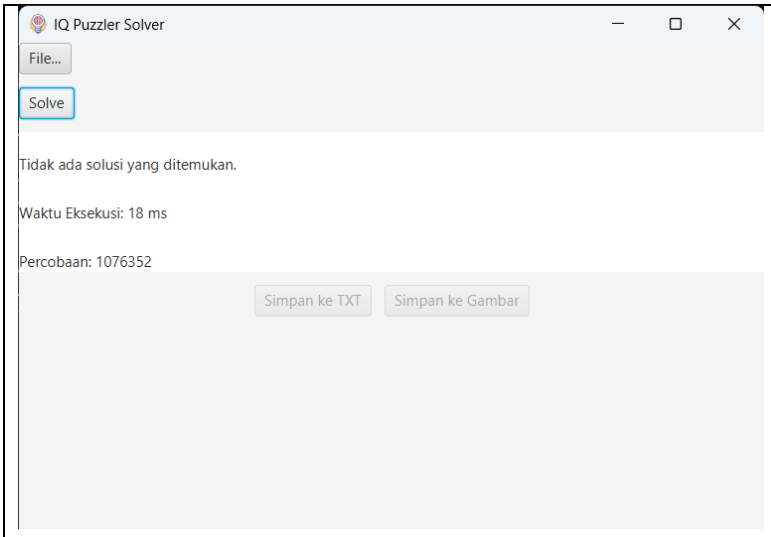
Output





2.

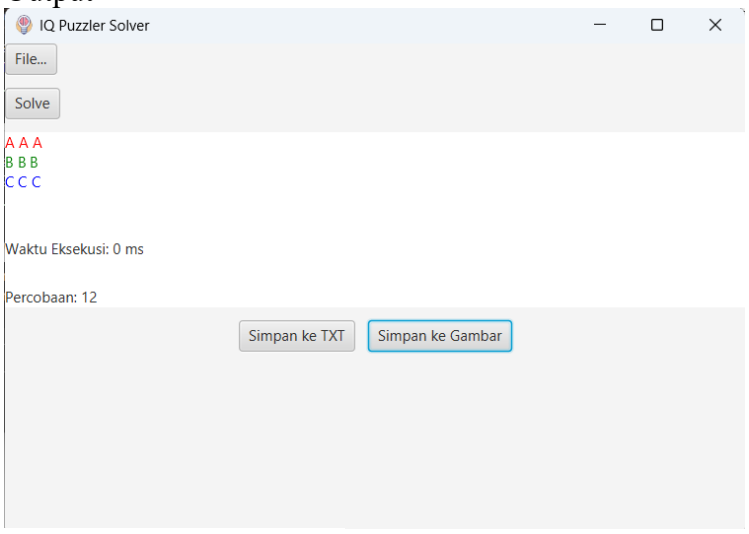
Input
5 5 7 DEFAULT A AA B BB C CC D DD EE EE E FF FF F GG
Output

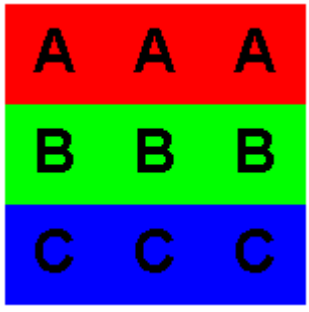


3.

Input
3 3 3
DEFAULT
AAA
BBB
CCC

Output

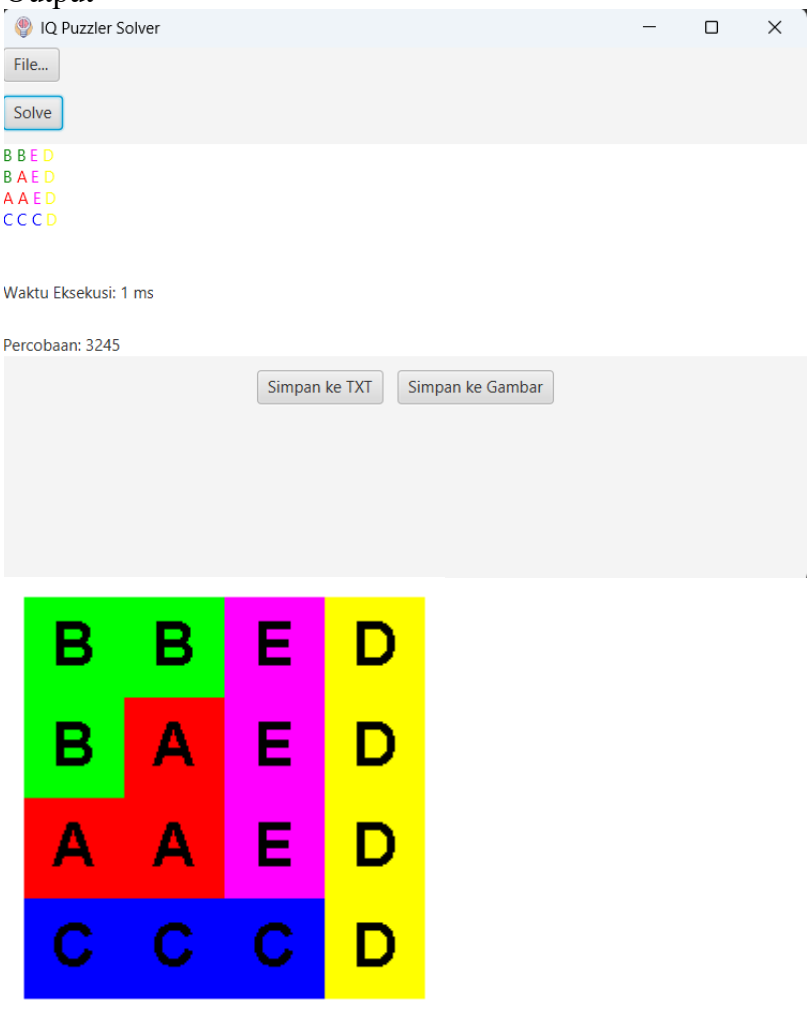




4.

Input
4 4 5
DEFAULT
A
AA
BB
B
CCC
DDDD
E
E
E

Output

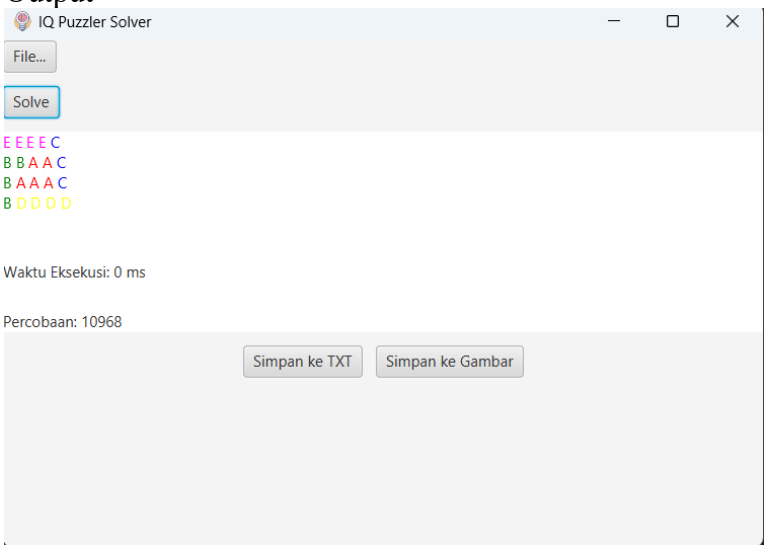


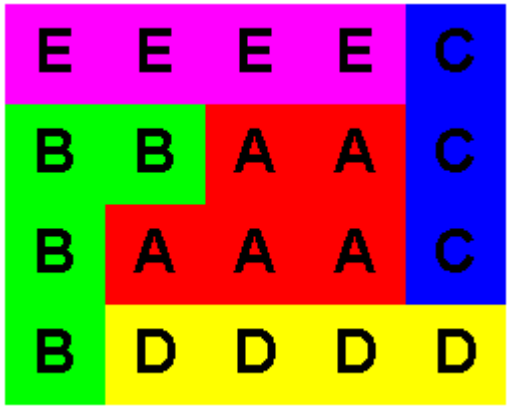
5.

Input
4 5 5

DEFAULT
A
AA
AA
BB
B
B
C
C
C
D
D
D
D
EEEE

Output



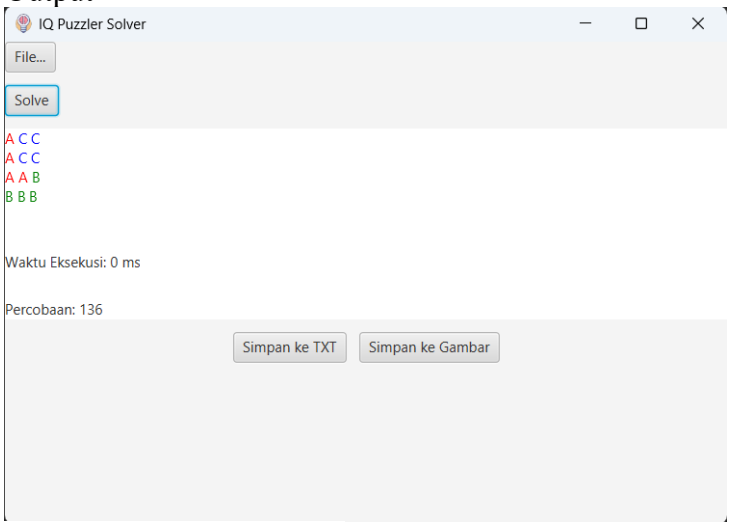


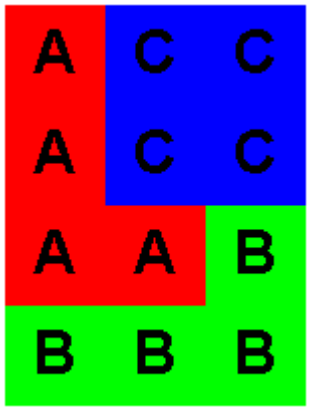
6.

Input
4 3 3

DEFAULT
A
A
AA
BB
B
B
CC
CC

Output



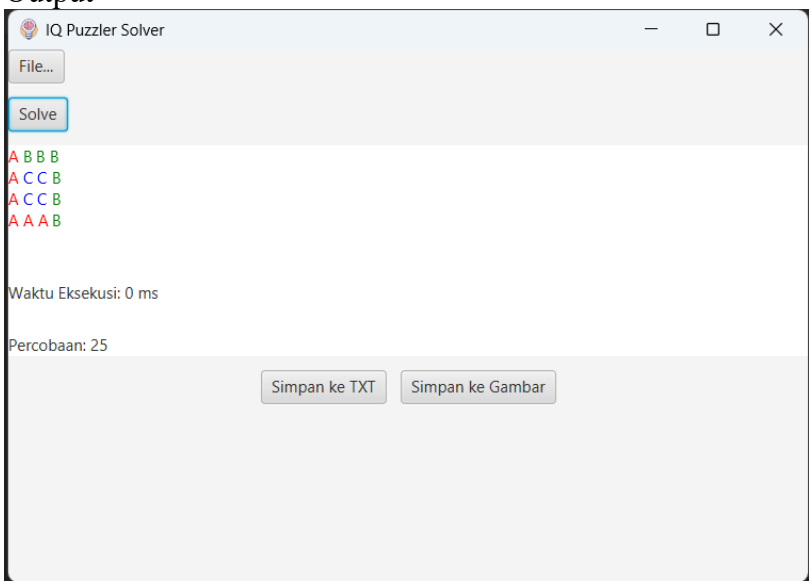


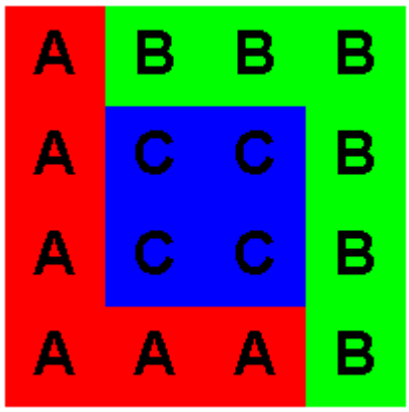
7.

Input
4 4 3
DEFAULT
A
A
A
AAA
BBBB
B
B

CC
CC

Output


IQ Puzzler Solver
File...
Solve
A B B B
A C C B
A C C B
A A A B
Waktu Eksekusi: 0 ms
Percobaan: 25
Simpan ke TXT
Simpan ke Gambar


A B B B
A C C B
A C C B
A A A B

BAB 5

Lampiran

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

Referensi

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/02-Algoritma-Brute-Force-(2025)-Bag1.pdf