LAPORAN TUGAS KECIL 1

IF2211 Strategi Algoritma

IQ Puzzler Pro dengan Algoritma Brute Force

Oleh :
Reza Ahmad Syarif (13523119)

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG JL. GANESA 10, BANDUNG 40132 2024

DAFTAR ISI

1 DESKRIPSI MASALAH

1.1 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

2 ALGORITMA BRUTE FORCE

Penyelesaian IQ Puzzler Pro dengan algoritma Brute Force ini melibatkan proses backtracking. Seluruh kemungkinan penempatan keping akan dicoba semua hingga menemukan salah satu solusi yang valid atau menyimpulkan bahwa tidak terdapat solusi yang valid. Beberapa komponen atau kelas yang memiliki peran utama pada algoritma Brute Force dengan backtracking ini yaitu Board, Piece, Solver, dan Main.

Kelas Board merepresentasikan papan permainan N x M yang akan menyimpan status dari papan tersebut antara sel kosong atau sudah diisi oleh suatu keping. Kelas Board akan menentukan suatu blok dapat dapat ditempatkan pada posisi tertentu atau tidak dengan menggunakan method *canPlacePiece*. Apabila tidak bisa, proses akan mundur dan keping akan dihapus dari papan menggunakan method *removePiece*.

Kelas Piece merepresentasikan keping puzzle sesuai dengan input file. Setiap keping puzzle dalam input akan dibentuk berbagai variasi dengan melakukan rotasi dan refleksi setiap kepingnya. Rotasi dilakukan dengan memutar keping dengan kelipatan 90 derajat (90, 180,270) serta refleksi terhadap sumbu horizontal dan vertikal.

Kelas Solver akan menjalankan algoritma Brute Force dengan proses backtracking. Solver akan mencoba menempatkan setiap keping pada papan dengan mencoba semua kemungkinan variasi yang telah dibentuk. Apabila satu variasi gagal ditempatkan, proses akan mundur ke variasi lainnya.

Kelas Main berfungsi sebagai kelas yang akan mengurusi bagian I/O atau interaksi dengan pengguna. Main akan melakukan *parsing* file untuk mendapatkan informasi mengenai dimensi papan, jumlah keping, bentuk papan, dan bentuk masing-masing keping. Apabila input telah di-*parsing,* Main akan menginisiasi kelas Board serta kelas Solver untuk memulai melakukan algoritma Brute Force dan juga menghitung waktu pencarian.

Secara sistematis, berikut merupakan langkah-langkah penyelesaian IQ Puzzler Pro menggunakan algoritma Brute Force dengan backtracking.

1. Kelas Main akan melakukan *parsing* file sesuai dengan input user. Setelah proses tersebut selasai, main akan menginisiasi kelas Board untuk membentuk papan dengan dimensi N x M serta kelas Solver untuk dengan papan dan keping yang sudah dibentuk. Kelas main kemudian memanggil method *runSolver* dari kelas Solver untuk mencari solusi.

```
Board board = new Board(N, M);
Solver solver = new Solver(board, pieces);
long startTime = System.currentTimeMillis();
boolean solve = solver.runSolver();
long endTIme = System.currentTimeMillis();
long executionTime = endTIme - startTime;
```

2. Pada kelas Solver akan dilakukan proses Brute Force dengan backtracking. Method runSolver akan memanggil method solve yang dimulai dari index 0 yaitu keping pertama.

```
public boolean runSolver() {
    return solve(index:0);
}
```

3. Method solve akan terlebih dahulu akan mengecek apakah semua blok sudah ditempatkan pada papan secara valid. Jika benar maka solusi ditemukan. Jika tidak maka lanjut ke langkah selanjutnya.

```
private boolean solve(int index) {
    if (index == pieces.size()) {
        return board.isFull();
    }
```

4. Ambil salah satu keping dari pieces sesuai dengan index saat ini. Hasilkan semua kemungkinan bentuk keping mulai dengan rotasi dan refleksi dengan menggunakan method *generateVariations* dari kelas Piece.

```
Piece piece = pieces.get(index);
List<Piece> variations = piece.generateVariations();
```

5. Setelah dihasilkan variasi, akan dilakukan looping setiap dari variasi tersebut. Looping bagian dalam akan mencoba semua posisi baris dan kolom pada papan.
6. Setiap variasi dan posisi tersebut akan dilakukan pengecekan apakah keping dapat ditempatkan pada papan menggunakan method canPlacePiece. Jika bisa maka akan dilakukan rekursif dengan memanggil method solve dengan index yang bertambah satu atau dengan kata lain lanjut ke keping selanjutnya.
7. Apabila method canPlacePiece menghasilkan nilai false atau keping tidak bisa ditempatkan, keping akan dihapus dari papan dan akan dilakukan proses backtracking dengan mencoba posisi dan variasi lain.
8. Apabila tidak ada posisi maupun variasi yang valid, method solve akan menghasilkan nilai false yang berarti tidak ditemukan solusi yang valid untuk papan dan keping-keping pada input.

```
private boolean solve(int index) {
    if (index == pieces.size()) {
        return board.isFull();
    }

    Piece piece = pieces.get(index);
    List<Piece> variations = piece.generateVariations();

    for (Piece variant : variations) {
        for (int r = 0; r < board.getRows(); r++) {
            for (int c = 0; c < board.getCols(); c++) {
                attemptCount++;
                if (board.canPlacePiece(variant, r, c)) {
                    if (solve(index+1)) {
                        return true;
                    }
                    board.removePiece(variant, r, c);
                }
            }
        }
    }
    return false;
}
```

3 SOURCE CODE PROGRAM DALAM JAVA

3.1 Source Code

3.1.1 Board.java

```
import java.util.*;

public class Board {
    private final int rows, cols;
    private final char[][] board;
    private final Map<Character, String> colorMap = new HashMap<>();

    // ANSI Colors untuk memberikan warna pada piece
    private static final String[] colors = {
        "\u001B[31m", "\u001B[32m",
        "\u001B[33m", "\u001B[34m",
        "\u001B[35m", "\u001B[36m",
        "\u001B[91m", "\u001B[92m",
        "\u001B[93m", "\u001B[94m",
        "\u001B[95m", "\u001B[96m",
        "\u001B[97m", "\u001B[90m",
        "\u001B[37m", "\u001B[30m",
        "\u001B[41m", "\u001B[42m",
        "\u001B[43m", "\u001B[44m",
        "\u001B[45m", "\u001B[46m",
        "\u001B[47m", "\u001B[100m",
        "\u001B[101m", "\u001B[102m",
        "\u001B[103m"
    };

    private static final String reset = "\u001B[0m";

    public Board(int rows, int cols) {
        this.rows = rows;
        this.cols = cols;
        this.board = new char[rows][cols];
        for (char[] row : board) {
            for (int i = 0; i < row.length; i++) {
                row[i] = '.';
            }
        }
    }

    public int getRows() {
        return rows;
    }

    public int getCols() {
        return cols;
    }

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

    public boolean canPlacePiece(Piece piece, int row, int col) {
        char[][] shape = piece.getPiece();
        int pieceRows = shape.length;
        int pieceCols = shape[0].length;
```

```
        if (row + pieceRows > rows || col + pieceCols > cols) {
            return false;
        }

        for (int r = 0; r < pieceRows; r++) {
            for (int c = 0; c < pieceCols; c++) {
                if (shape[r][c] != ' ' && board[row + r][col + c] != '.') {
                    return false;
                }
            }
        }

        for (int r = 0; r < pieceRows; r++) {
            for (int c = 0; c < pieceCols; c++) {
                if (shape[r][c] != ' ') {
                    board[row + r][col + c] = shape[r][c];
                    if (!colorMap.containsKey(shape[r][c])) {
                        colorMap.put(shape[r][c], colors[colorMap.size()]);
                    }
                }
            }
        }
        return true;
    }

    public void removePiece(Piece piece, int row, int col) {
        char[][] shape = piece.getPiece();
        int pieceRows = shape.length;
        int pieceCols = shape[0].length;

        for (int r = 0; r < pieceRows; r++) {
            for (int c = 0; c < pieceCols; c++) {
                if (shape[r][c] != ' ') {
                    board[row + r][col + c] = '.';
                }
            }
        }
    }

    public boolean isFull() {
        for (char[] row : board) {
            for (char cell : row) {
                if (cell == '.') {
                    return false;
                }
            }
        }
        return true;
    }

    public String getBoardString(boolean withColor) {
        StringBuilder sb = new StringBuilder();
        for (char[] row : board) {
            for (char cell : row) {
                if (withColor && cell != '.' && colorMap.containsKey(cell)) {
                    sb.append(colorMap.get(cell)).append(cell).append(' ').append(reset);
                } else {
                    sb.append(cell).append(' ');
                }
            }
```

```
            sb.append("\n");
        }
        return sb.toString();
    }
}
```

3.1.2 Piece.java

```
import java.util.*;

public class Piece {
    private final char[][] piece;
    private final char label;

    public Piece(List<String> lines) {
        this.label = lines.get(0).trim().charAt(0);

        int maxCols = 0;
        for (String line : lines) {
            int lastIdx = line.lastIndexOf(line.replaceAll(" ","").charAt(line.replaceAll(" ",
"").length() - 1));
            if (lastIdx + 1 > maxCols) {
                maxCols = lastIdx + 1;
            }
        }

        int rows = lines.size();
        piece  = new char[rows][maxCols];
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < maxCols; j++) {
                piece[i][j] = ' ';
            }
        }

        for (int i = 0; i < rows; i++) {
            String line = lines.get(i);
            for (int j = 0; j < line.length(); j++) {
                piece[i][j] = line.charAt(j);
            }
        }
    }

    public Piece(char[][] piece, char label) {
        this.piece = piece;
        this.label = label;
    }

    public char[][] getPiece() {
        return piece;
    }

    public char getLabel() {
        return label;
    }

    private char[][] trimShape(char[][] piece) {
        int rows = piece.length;
        int cols = piece[0].length;
        int top = rows, bottom = 0, left = cols, right = 0;
        for (int i = 0; i < rows; i++) {
```

```
            for (int j = 0; j < cols; j++) {
                if (piece[i][j] != ' ') {
                    top = Math.min(top, i);
                    bottom = Math.max(bottom, i);
                    left = Math.min(left, j);
                    right = Math.max(right, j);
                }
            }
        }
        char[][] trimmed = new char[bottom - top + 1][right - left + 1];
        for (int i = top; i <= bottom; i++) {
            for (int j = left; j <= right; j++) {
                trimmed[i - top][j - left] = piece[i][j];
            }
        }
        return trimmed;
    }

    private char[][] rotate(char[][] piece) {
        int rows = piece.length;
        int cols = piece[0].length;
        char[][] rotated = new char[cols][rows];
        for (int i = 0; i < cols; i++) {
            for (int j = 0; j < rows; j++) {
                rotated[i][j] = piece[rows - j - 1][i];
            }
        }
        return trimShape(rotated);
    }

    private char[][] flipHorizontal(char[][] piece) {
        int rows = piece.length;
        int cols = piece[0].length;
        char[][] flipped = new char[rows][cols];
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                flipped[i][j] = piece[i][cols - j - 1];
            }
        }
        return trimShape(flipped);
    }

    private char[][] flipVertical(char[][] piece) {
        int rows = piece.length;
        int cols = piece[0].length;
        char[][] flipped = new char[rows][cols];
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                flipped[i][j] = piece[rows - i - 1][j];
            }
        }
        return trimShape(flipped);
    }

    public List<Piece> generateVariations() {
        Set<String> uniqueVariations = new HashSet<>();
        List<Piece> variations = new ArrayList<>();

        char[][] currentPiece = piece;

        for (int i = 0; i < 4; i++) {
```

```
            for (char[][] variation : new char[][][]{currentPiece, flipHorizontal(currentPiece),
flipVertical(currentPiece)}) {
                String key = Arrays.deepToString(variation);
                if (uniqueVariations.add(key)) {
                    variations.add(new Piece(variation, label));
                }
            }
            currentPiece = rotate(currentPiece);
        }

        return variations;
    }
}
```

3.1.3 Solver.java

```
import java.util.*;
public class Solver {
    private final Board board;
    private final List<Piece> pieces;
    private long attemptCount = 0;

    public Solver(Board board, List<Piece> pieces) {
        this.board = board;
        this.pieces = pieces;
    }

    public long getAttemptCount() {
        return attemptCount;
    }

    public boolean runSolver() {
        return solve(0);
    }

    private boolean solve(int index) {
        if (index == pieces.size()) {
            return board.isFull();
        }

        Piece piece = pieces.get(index);
        List<Piece> variations = piece.generateVariations();

        for (Piece variant : variations) {
            for (int r = 0; r < board.getRows(); r++) {
                for (int c = 0; c < board.getCols(); c++) {
                    attemptCount++;
                    if (board.canPlacePiece(variant, r, c)) {
                        if (solve(index+1)) {
                            return true;
                        }
                        board.removePiece(variant, r, c);
                    }
                }
            }
        }
        return false;
    }
}
```

3.1.4 Main.java

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

public class Main {
    public static void main(String[] args) {
        Scanner userInput = new Scanner(System.in);
        System.out.print("\nMasukkan file input: ");
        String inputName = userInput.nextLine();
        String inputFolder = "../test/input";
        File inputFile = new File(inputFolder, inputName);
        try {
            Scanner input = new Scanner(inputFile);
            int N = input.nextInt();
            int M = input.nextInt();
            int P = input.nextInt();
            input.nextLine();

            String caseType = input.nextLine();
            if (!caseType.equals("DEFAULT")) {
                System.out.println("Jenis kasus yang tersedia hanya DEFAULT");
                input.close();
                return;
            }

            List<Piece> pieces = new ArrayList<>();
            List<String> block = new ArrayList<>();
            char lastLetter = ' ';
            while (input.hasNextLine()) {
                String line = input.nextLine();
                int firstIndex = 0;
                while (firstIndex < line.length() && line.charAt(firstIndex) == ' ') {
                    firstIndex++;
                }
                if (firstIndex == line.length()) {
                    continue;
                }
                char firstChar = line.charAt(firstIndex);
                if (lastLetter != ' ' && firstChar != lastLetter) {
                    if (!block.isEmpty()) {
                        pieces.add(new Piece(block));
                        block.clear();
                    }
                }
                lastLetter = firstChar;
                block.add(line);
            }
            if (!block.isEmpty()) {
                pieces.add(new Piece(block));
            }
            input.close();

            if (pieces.size() != P) {
                System.out.println("Jumlah blok terdata tidak sesuai dengan input");
                return;
            }

            Board board = new Board(N, M);
            Solver solver = new Solver(board, pieces);
            long startTime = System.currentTimeMillis();
```

```
            boolean solve = solver.runSolver();
            long endTIme = System.currentTimeMillis();
            long executionTime = endTIme - startTime;
            StringBuilder result = new StringBuilder();

            if (solve) {
                result.append("Solusi ditemukan!\n");
                result.append(board.getBoardString(false)).append("\n");

                System.out.println("\nSolusi ditemukan!");
                System.out.println(board.getBoardString(true));
            } else {
                result.append("Solusi tidak ditemukan.\n");
                System.out.println("\nSolusi tidak ditemukan.");
            }

            result.append("Waktu pencarian: ").append(executionTime).append(" ms\n\n");
            result.append("Banyak kasus yang ditinjau: ").append(solver.getAttemptCount());
            System.out.println("Waktu pencarian: " + executionTime + " ms\n");
            System.out.println("Banyak kasus yang ditinjau: " + solver.getAttemptCount());

            System.out.println("\nApakah anda ingin menyimpan solusi? (y/n)");
            String save = userInput.nextLine().trim().toLowerCase();

            if (save.equals("y")) {
                System.out.println("\nMasukkan nama file output: ");
                String outputName = userInput.nextLine();
                String outputFolder = "../test/output";
                File folder = new File(outputFolder);
                if (!folder.exists()) {
                    folder.mkdirs();
                }
                File outputFile = new File(folder, outputName);
                try (PrintWriter writer = new PrintWriter(outputFile)) {
                    writer.println(result);
                    System.out.println("\nHasil telah disimpan di " + outputFile);
                } catch (IOException e) {
                    System.out.println("Gagal menyimpan file output: " + e.getMessage());
                }
            }
        } catch (FileNotFoundException e) {
            System.out.println("\nFile tidak ditemukan\n");
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }

        System.out.println();
        userInput.close();
    }
}
```

3.1.5 PuzzleSolverGUI.java

```
import javax.swing.*;
import java.awt.*;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.PrintWriter;
```

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

public class PuzzleSolverGUI {
    private JFrame frame;
    private JButton loadButton, solveButton, saveTxtButton, savePngButton;
    private JPanel mainPanel, boardContainer, boardPanel;
    private JLabel executionTimeLabel, attemptCountLabel;
    private Board board;
    private List<Piece> pieces;
    private Solver solver;
    private int N, M;
    private Map<Character, Color> colorMap;

    public PuzzleSolverGUI() {
        frame = new JFrame("IQ Puzzler Pro Solver");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(1000, 1000);
        frame.setLayout(new BorderLayout());

        JPanel topPanel = new JPanel();
        loadButton = new JButton("Load File");
        solveButton = new JButton("Solve");
        saveTxtButton = new JButton("Save To TXT");
        savePngButton = new JButton("Save To PNG");

        topPanel.add(loadButton);
        topPanel.add(solveButton);
        topPanel.add(saveTxtButton);
        topPanel.add(savePngButton);

        frame.add(topPanel, BorderLayout.NORTH);

        mainPanel = new JPanel();
        mainPanel.setLayout(new BorderLayout());
        frame.add(mainPanel, BorderLayout.CENTER);

        boardContainer = new JPanel();
        boardContainer.setPreferredSize(new Dimension(300, 300));
        boardContainer.setLayout(new BorderLayout());

        boardPanel = new JPanel();
        boardPanel.setPreferredSize(new Dimension(300, 300));
        boardContainer.add(boardPanel, BorderLayout.CENTER);

        JPanel infoPanel = new JPanel();
        executionTimeLabel = new JLabel("Waktu Pencarian: -");
        attemptCountLabel = new JLabel("Banyak kasus yang ditinjau: -");
        infoPanel.add(executionTimeLabel);
        infoPanel.add(attemptCountLabel);

        mainPanel.add(boardContainer, BorderLayout.CENTER);
        mainPanel.add(infoPanel, BorderLayout.SOUTH);

        loadButton.addActionListener(_ -> loadFile());
        solveButton.addActionListener(_ -> solvePuzzle());
        saveTxtButton.addActionListener(_ -> saveToTxt());
        savePngButton.addActionListener(_ -> saveToPng());
```

```
        frame.setVisible(true);
    }

    private void loadFile() {
        JFileChooser fileChooser = new JFileChooser("../test/input");
        int returnValue = fileChooser.showOpenDialog(frame);

        if (returnValue == JFileChooser.APPROVE_OPTION) {
            File selectedFile = fileChooser.getSelectedFile();
            try {
                Scanner scanner = new Scanner(selectedFile);
                N = scanner.nextInt();
                M = scanner.nextInt();
                int P = scanner.nextInt();
                scanner.nextLine();

                if (!scanner.nextLine().equals("DEFAULT")) {
                    JOptionPane.showMessageDialog(frame, "Jenis kasus yang tersedia hanya
DEFAULT", "Error", JOptionPane.ERROR_MESSAGE);
                    scanner.close();
                    return;
                }

                pieces = new ArrayList<>();
                List<String> currentBlock = new ArrayList<>();
                char lastLetter = ' ';

                while (scanner.hasNextLine()) {
                    String line = scanner.nextLine();
                    if (line.isEmpty()) continue;

                    int firstNonSpaceIndex = 0;
                    while (firstNonSpaceIndex < line.length() && line.charAt(firstNonSpaceIndex)
== ' ') {
                        firstNonSpaceIndex++;
                    }
                    if (firstNonSpaceIndex == line.length()) continue;

                    char firstChar = line.charAt(firstNonSpaceIndex);
                    if (lastLetter != ' ' && firstChar != lastLetter) {
                        if (!currentBlock.isEmpty()) {
                            pieces.add(new Piece(currentBlock));
                            currentBlock.clear();
                        }
                    }

                    lastLetter = firstChar;
                    currentBlock.add(line);
                }

                if (!currentBlock.isEmpty()) {
                    pieces.add(new Piece(currentBlock));
                }

                scanner.close();

                if (pieces.size() != P) {
                    JOptionPane.showMessageDialog(frame, "Jumlah blok terdata tidak sesuai
dengan input", "Error", JOptionPane.ERROR_MESSAGE);
                    return;
                }
```

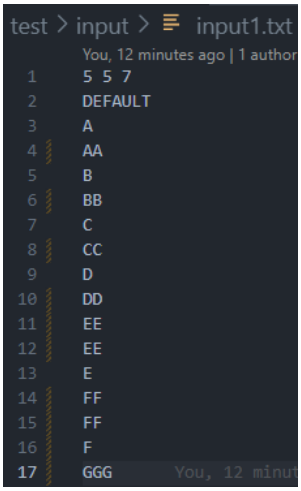
```
                board = new Board(N, M);
                solver = new Solver(board, pieces);
                JOptionPane.showMessageDialog(frame, "File berhasil dimuat!", "Info",
JOptionPane.INFORMATION_MESSAGE);
            } catch (FileNotFoundException e) {
                JOptionPane.showMessageDialog(frame, "File tidak ditemukan!", "Error",
JOptionPane.ERROR_MESSAGE);
            }
        }
    }

    private void solvePuzzle() {
        if (board == null || solver == null) {
            JOptionPane.showMessageDialog(frame, "Load File terlebih dahulu!", "Error",
JOptionPane.ERROR_MESSAGE);
            return;
        }

        long startTime = System.currentTimeMillis();
        boolean solve = solver.runSolver();
        long endTime = System.currentTimeMillis();

        executionTimeLabel.setText("Waktu Pencarian: " + (endTime - startTime) + " ms");
        attemptCountLabel.setText("Banyak kasus yang ditinjau: " + solver.getAttemptCount());

        if (solve) {
            drawBoard();
        } else {
            JOptionPane.showMessageDialog(frame, "Tidak ada solusi!", "Info",
JOptionPane.INFORMATION_MESSAGE);
        }
    }

    private void drawBoard() {
        boardPanel.removeAll();
        boardPanel.setLayout(new GridLayout(N, M));
        boardPanel.setPreferredSize(new Dimension(300, 300));
        char[][] grid = board.getBoard();

        colorMap = new HashMap<>();
        Random rand = new Random();

        for (int r = 0; r < N; r++) {
            for (int c = 0; c < M; c++) {
                char block = grid[r][c];
                JPanel cell = new JPanel(new BorderLayout());
                JLabel label = new JLabel(String.valueOf(block), SwingConstants.CENTER);

                if (block != ' ') {
                    colorMap.putIfAbsent(block, new Color(rand.nextInt(256), rand.nextInt(256),
rand.nextInt(256)));
                    cell.setBackground(colorMap.get(block));
                } else {
                    cell.setBackground(Color.WHITE);
                }

                cell.setBorder(BorderFactory.createLineBorder(Color.BLACK));
                cell.add(label, BorderLayout.CENTER);
                boardPanel.add(cell);
            }
```

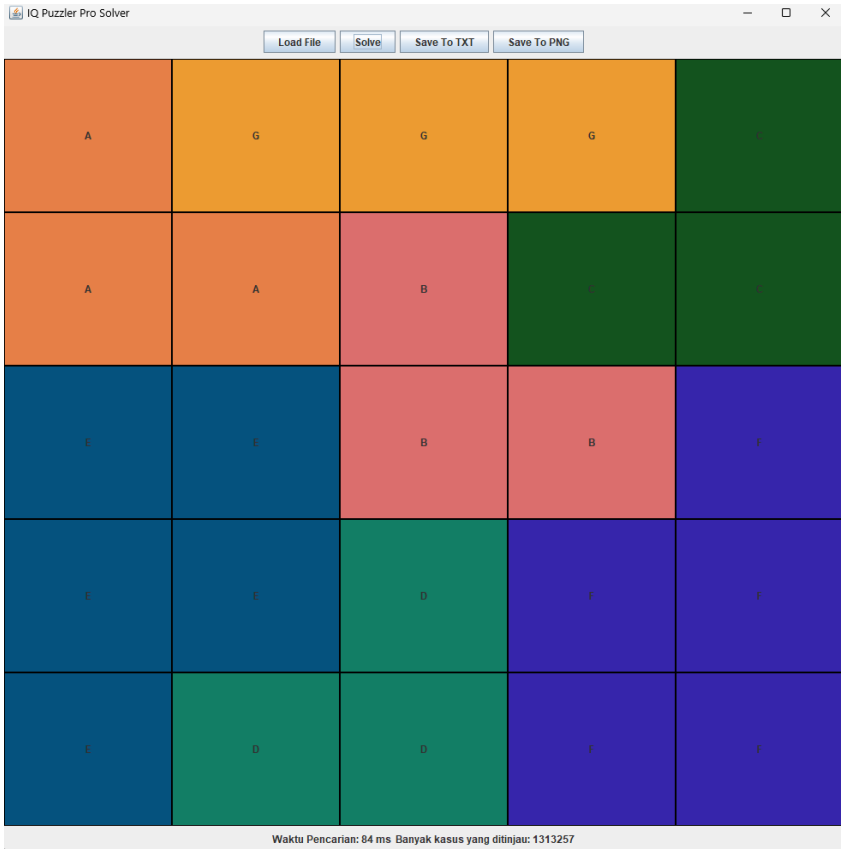
```
        }

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

    private void saveToTxt() {
        JFileChooser fileChooser = new JFileChooser("../test/output");
        fileChooser.setDialogTitle("Simpan sebagai");
        fileChooser.setFileSelectionMode(JFileChooser.FILES_ONLY);

        int returnValue = fileChooser.showSaveDialog(frame);
        if (returnValue == JFileChooser.APPROVE_OPTION) {
            File selectedFile = fileChooser.getSelectedFile();
            String fileName = selectedFile.getAbsolutePath();

            if (!fileName.endsWith(".txt")) {
                fileName += ".txt";
            }

            try (PrintWriter writer = new PrintWriter(fileName)) {
                writer.print(board.getBoardString(false));
                writer.println();
                writer.println(executionTimeLabel.getText());
                writer.println();
                writer.println(attemptCountLabel.getText());
                JOptionPane.showMessageDialog(frame, "Hasil disimpan ke " + fileName, "Info",
JOptionPane.INFORMATION_MESSAGE);
            } catch (FileNotFoundException e) {
                JOptionPane.showMessageDialog(frame, "Gagal menyimpan file!", "Error",
JOptionPane.ERROR_MESSAGE);
            }
        }
    }

    private void saveToPng() {
        BufferedImage image = new BufferedImage(boardContainer.getWidth(),
boardContainer.getHeight(), BufferedImage.TYPE_INT_ARGB);
        Graphics2D g2d = image.createGraphics();
        boardContainer.printAll(g2d);
        g2d.dispose();

        JFileChooser fileChooser = new JFileChooser("../test/output");
        fileChooser.setDialogTitle("Simpan sebagai");
        fileChooser.setFileSelectionMode(JFileChooser.FILES_ONLY);

        int returnValue = fileChooser.showSaveDialog(frame);
        if (returnValue == JFileChooser.APPROVE_OPTION) {
            File selectedFile = fileChooser.getSelectedFile();
            String fileName = selectedFile.getAbsolutePath();

            if (!fileName.endsWith(".png")) {
                fileName += ".png";
            }

            try {
                ImageIO.write(image, "png", new File(fileName));
                JOptionPane.showMessageDialog(frame, "Gambar disimpan ke " + fileName, "Info",
JOptionPane.INFORMATION_MESSAGE);
            } catch (IOException e) {
```

```
                JOptionPane.showMessageDialog(frame, "Gagal menyimpan gambar!", "Error",
JOptionPane.ERROR_MESSAGE);
            }
        }
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(PuzzleSolverGUI::new);
    }
}
```

4 EKSPERIMEN

4.1 Test Case 1 (Bonus GUI)





test > output > output1.txt

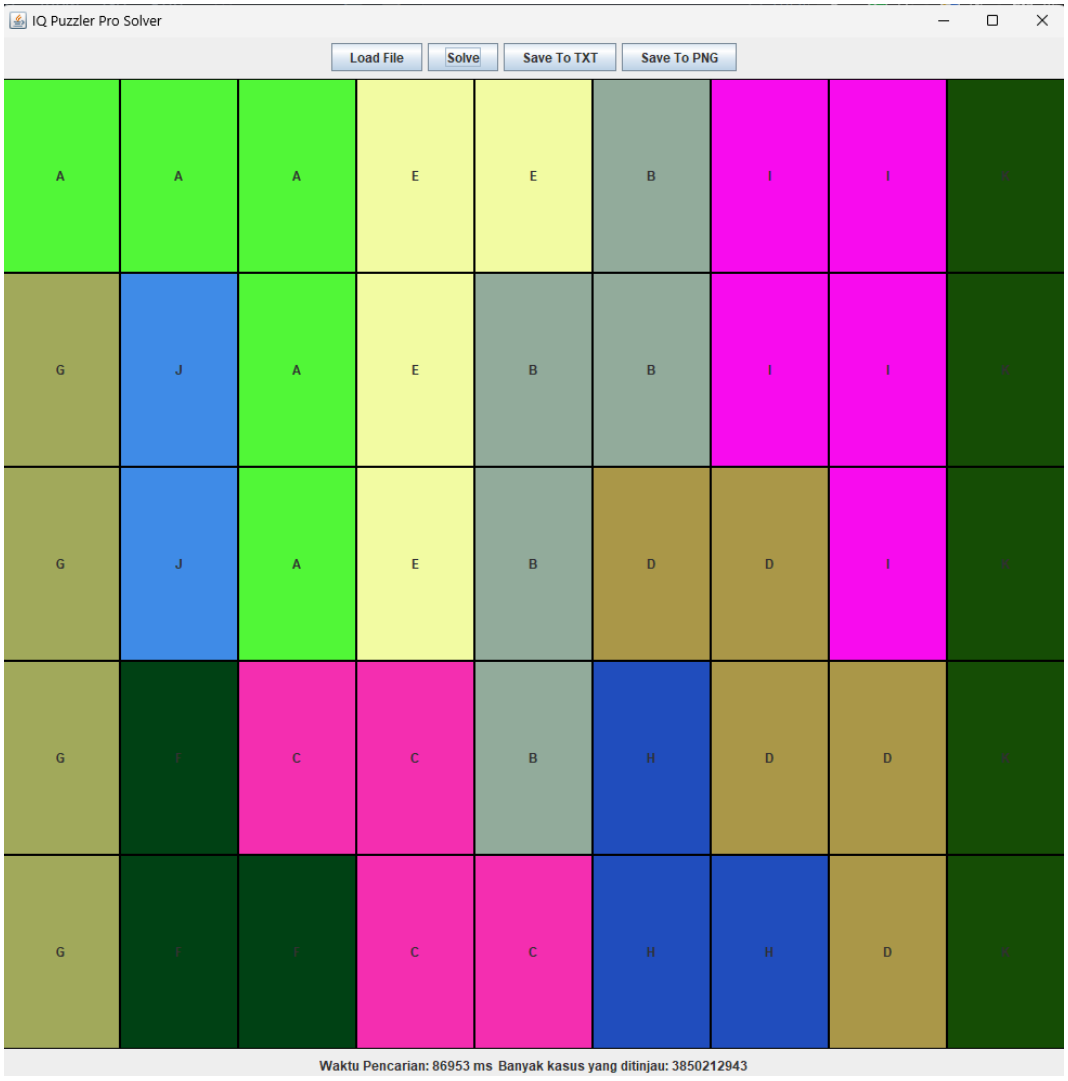
You, 1 second ago | 1 author (You)

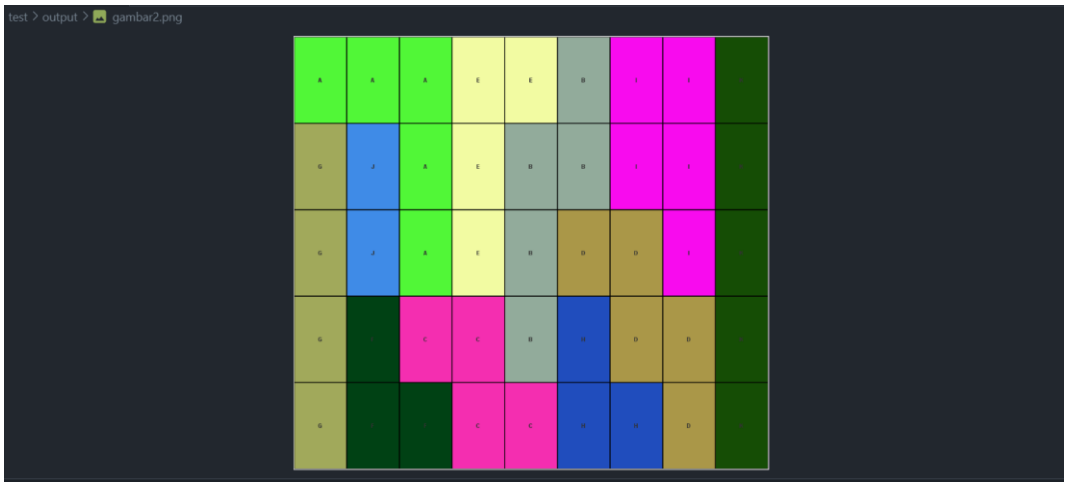
```
A G G G C
A A B C C
E E B B F
E E D F F
E D D F F

Waktu Pencarian: 84 ms

Banyak kasus yang ditinjau: 1313257

```

4.2 Test Case 2 (Bonus Gambar)

test > input > input2.txt

```
5 9 11
DEFAULT
AAA
  A
  A
 B
BB
B
B
CC
 CC
DD
 DD
  D
EE
E
E
FF
F
G
G
G
G
HH
H
I
II
II
JJ
KKKKK
```

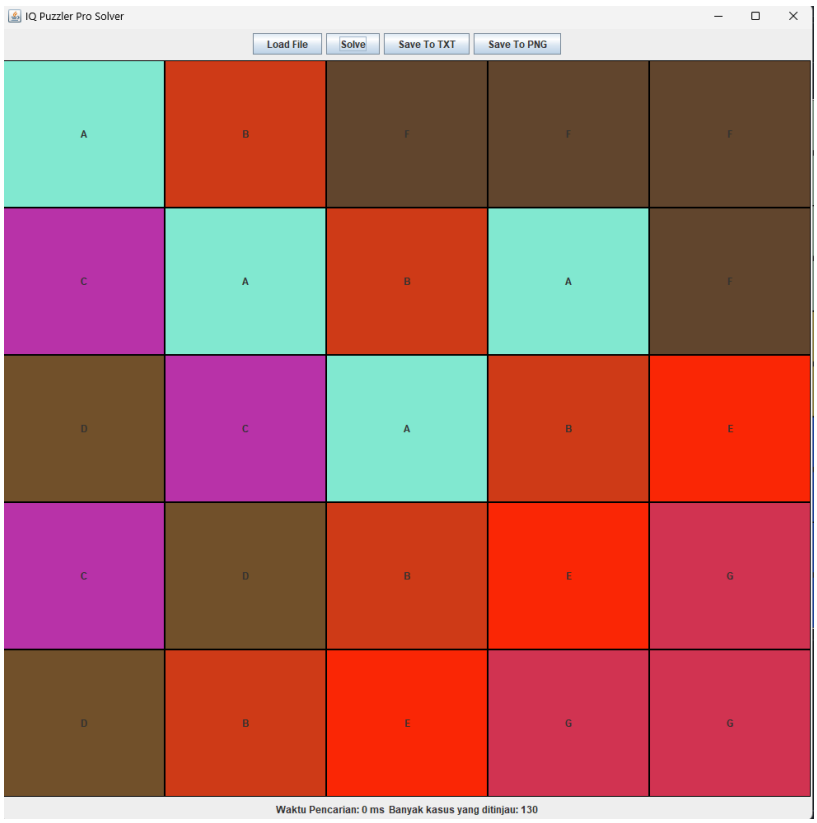

IQ Puzzler Pro Solver
Load File
Solve
Save To TXT
Save To PNG
A A A E E B I I K
G J A E B B I I K
G J A E B D D I K
G F C C B H D D K
G F F C C H H D K
Waktu Pencarian: 86953 ms Banyak kasus yang ditinjau: 3850212943


test > output > gambar2.png

4.3 Test Case 3

test > input > input3.txt

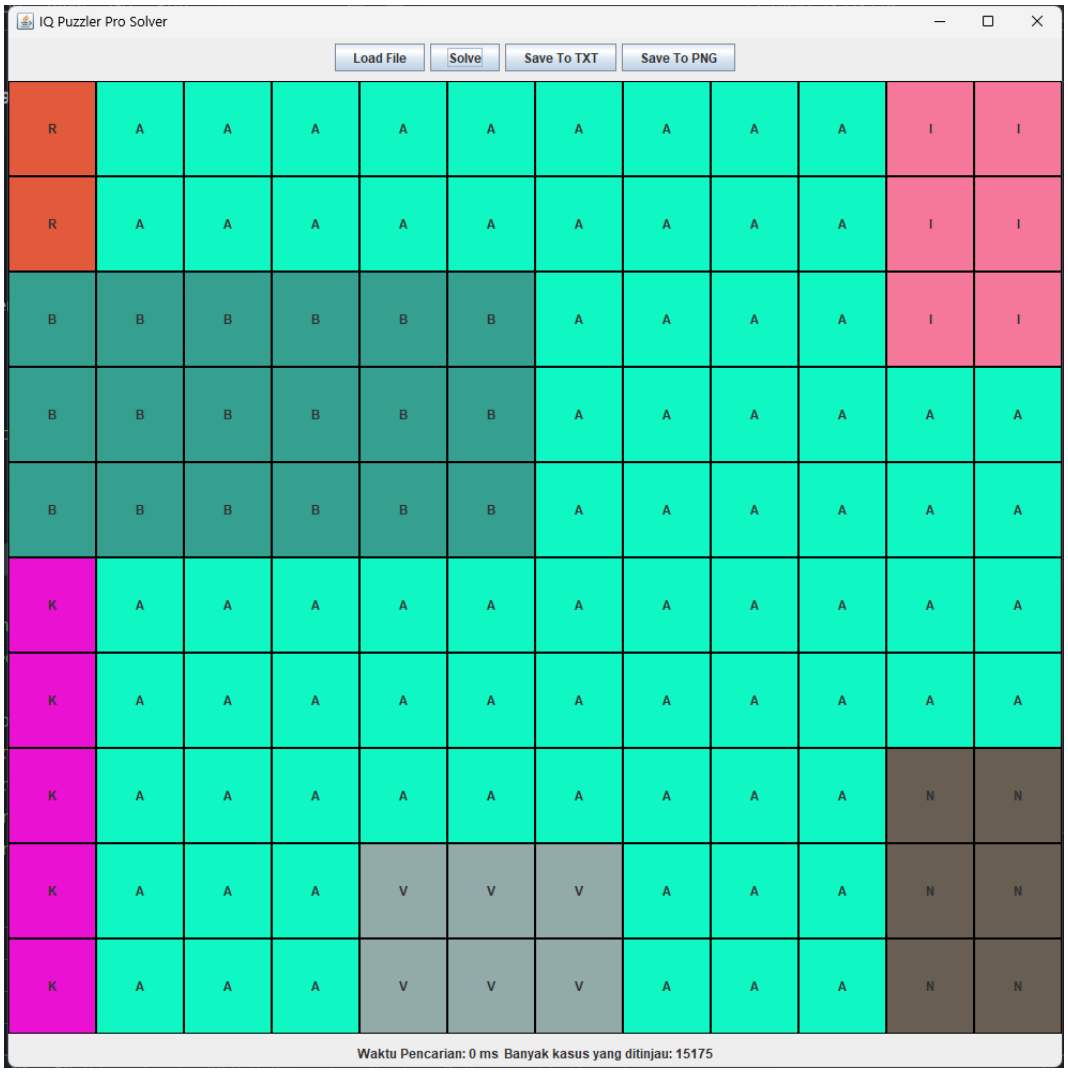
```
5 5 7
DEFAULT
A
 A A
  A
B
 B
  B
 B
B
C
 C
C
D
 D
D
E
 E
  E
FFF
  F
GG
 G
```



4.4 Test Case 4 (Among Us)

test > input > input4.txt

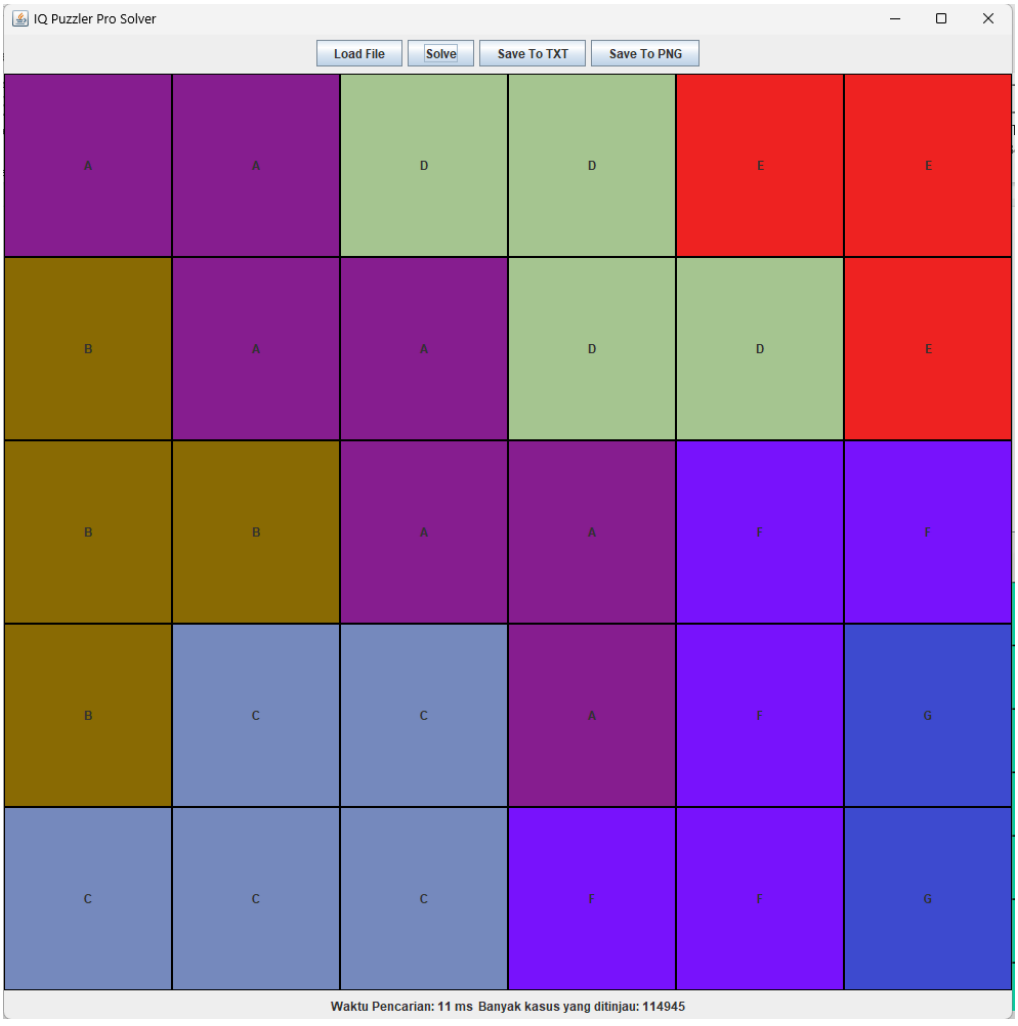
```
10 12 7
DEFAULT
 AAAAAAAAA
 AAAAAAAAA
      AAAA
      AAAAAA
      AAAAAA
 AAAAAAAAAAA
 AAAAAAAAAAA
 AAAAAAAAA
 AAA   AAA
 AAA   AAA
BBBBBB
BBBBBB
BBBBBB
RR
KKKKK
VVV
VVV
III
III
NNN
NNN
```



4.5 Test Case 5

test > input > input5.txt

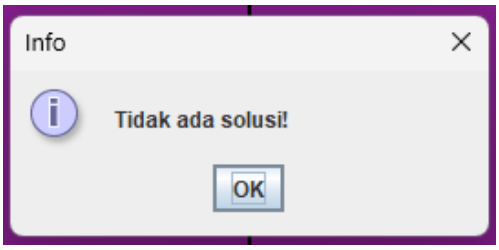
```
5 6 7
DEFAULT
AA
 AA
  AA
   A
B
BB
B
C
CC
CC
D
DD
 D
EE
E
FF
 F
 FF
GG
```



4.6 Test Case 6 (Keping lebih kecil)

test > input > smallerpieces.txt

```
4 4 7
DEFAULT
A
B
C
D
E
F
G
```



test > output > outputsp.txt

```
. . . .
. . . .
. . . .
. . . .

Waktu Pencarian: 14035 ms

Banyak kasus yang ditinjau: 101395472

```

4.7 Test Case 7 (Keping lebih besar)

test > input > biggerpieces.txt

```
4 4 3
DEFAULT
AAA
BB
BBBB
CC
CCCC
```

test > output > outputbp.txt

```
. . . .
. . . .
. . . .
. . . .

Waktu Pencarian: 1 ms

Banyak kasus yang ditinjau: 16416

```

5 LAMPIRAN

5.1 Tautan Repository Github

https://github.com/Rejaah/Tucil1_13523119