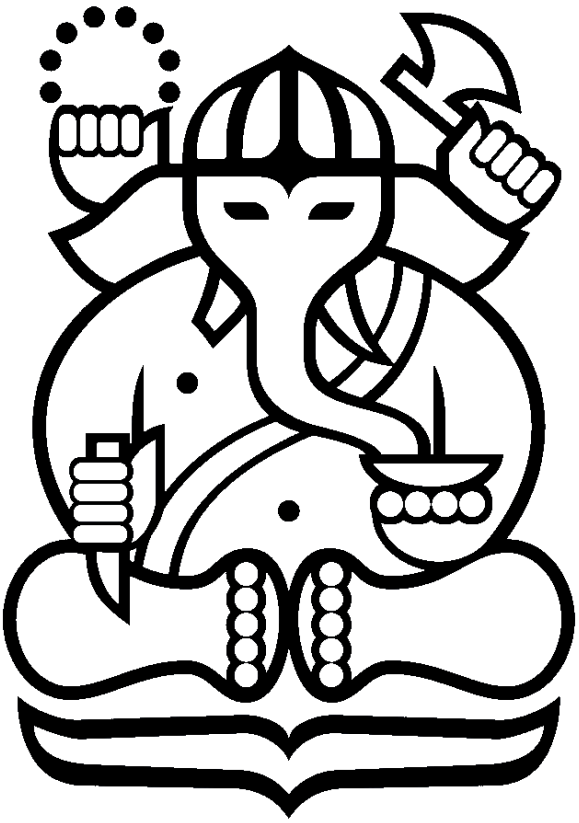
Laporan Tugas Kecil 1
IF2211 Strategi Algoritma



Disusun oleh

Hasri Fayadh Muqaffa - 13523156

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
SEMESTER 2 TAHUN 2024/2025

DAFTAR ISI

BAB I
PENDAHULUAN

1. Deskripsi Masalah

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:

a. Board (Papan) – Board merupakan komponen utama yang menjadi tujuan permainan dimana pemain harus mampu mengisi seluruh area papan menggunakan blok-blok yang telah disediakan.
b. 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.

Spesifikasi Program adalah sebagai berikut:

- Buatlah program sederhana dalam bahasa Java yang mengimplementasikan *algoritma Brute Force* untuk mencari solusi dalam permainan IQ Puzzler Pro.
- Algoritma brute force yang diimplementasikan harus bersifat "murni", tidak boleh memanfaatkan heuristik.
- Papan yang perlu diisi mulanya akan selalu kosong.
- Sebuah blok puzzle bisa saja dirotasi maupun dicerminkan sebelum diletakan pada papan.
- Input: program akan memberikan pengguna sebuah prompt untuk memilih file *test case* berekstensi .txt, kemudian program membaca file *test case* tersebut yang berisi
    1) Dimensi Papan terdiri atas dua buah variabel N dan M yang membentuk papan berdimensi NxM.
    2) Banyak blok puzzle direpresentasikan oleh variabel integer P.
    3) Jenis kasus sebuah variabel string S yang digunakan untuk mengidentifikasi kasus konfigurasi, hanya mungkin bernilai salah satu diantara DEFAULT/CUSTOM/PYRAMID.
    4) Bentuk blok puzzle yang dilambangkan oleh konfigurasi *Character* berupa huruf. Akan ada P buah blok puzzle berbeda yang dibentuk oleh P buah huruf berbeda. *Character* yang digunakan adalah huruf A-Z dalam kapital.
- Output:
    1) Tampilkan konfigurasi blok puzzle yang berhasil mengisi papan. Gunakan print berwarna untuk menunjukkan blok puzzle dengan jelas. Pastikan setiap blok puzzle berbeda memiliki warna berbeda. Beri tahu pengguna apabila puzzle tidak memiliki solusi.

2) Waktu eksekusi program dalam *milisecond* (tidak termasuk waktu membaca masukan dan menyimpan solusi, cukup waktu pencarian oleh algoritma).
3) Banyak kasus atau jumlah iterasi yang ditinjau oleh algoritma brute force.
4) Prompt untuk menyimpan solusi dalam sebuah berkas berekstensi .txt (Struktur untuk file output dibebaskan).

2. Algoritma Brute Force

Algoritma Brute Force adalah pendekatan yang berupaya menemukan solusi untuk semua kemungkinan konfigurasi secara komprehensif. Dalam permainan IQ Puzzler Pro, awalnya papan kosong dan harus diisi dengan sejumlah blok puzzle yang unik, sehingga algoritma ini akan mencari seluruh kombinasi penempatan, rotasi, dan pencerminan dari blok pada papan agar papan terisi penuh, atau jika tidak ada solusi, akan ditampilkan bahwa solusi tidak ditemukan

BAB II

PENERAPAN ALGORITMA BRUTE FORCE

1. Inisialisasi Papan dan Blok

- Kelas SolvePuzzle akan mendeklarasikan variabel-variabel yang dibutuhkan selama penyelesaian puzzle. N dan M sebagai ukuran dari board, board sebagai matriks dari board, blocks sebagai list dari sejumlah blok, solved sebagai boolean untuk mengecek apakah puzzle dapat diselesaikan atau tidak, dan count untuk menghitung kasus yang terjadi

- Papan akan kosong dengan diisikan ‘.’ di setiap *cell*

  Potongan kode dari kelas SolvePuzzle

```
public class SolvePuzzle {
    private int N, M;
    private char[][] board;
    private List<char[][]> blocks;
    private boolean solved = false;
    private long count = 0;

    public SolvePuzzle(int N, int M, List<char[][]> blocks) {
        this.N = N;
        this.M = M;
        this.blocks = blocks;
        this.board = new char[N][M];

        // Inisialisasi board dengan '.'
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < M; j++) {
                board[i][j] = '.';
            }
        }
    }
…
```

2. **Fungsi solve(int blockIdx)**
   - Di awal, program akan mengecek apakah board bisa dipenuhi dengan blok yang ada. Pengecekan ini dilakukan dengan menjumlahkan semua karakter dari blok-blok yang bukan ‘.’, lalu membandingkannya dengan ukuran dari board. Jika jumlah karakter yang ada di semua blok lebih kecil daripada ukuran papan (`countAlphabetsInBlocks(blocks) < N * M`), otomatis papan tidak akan penuh dan variabel solved akan tetap bernilai false. Kemudian program akan return.
   - Basis rekursi dari program ini adalah ketika (`blockIdx >= blocks.size()`) yang berarti semua blok telah digunakan dan dipasang. Jika board telah terisi semua (`isBoardFull()`), variabel solved akan bernilai true. Kemudian, program akan return.
   - Setelah itu, program akan melakukan *generating* untuk mendapatkan berbagai variasi blok. variasi blok didapatkan dari rotasi dan juga pencerminan, sehingga suatu blok memiliki maksimal 8 variasi.
   - Selanjutnya akan dilakukan berbagai kemungkinan. Perulangan pertama akan mencoba semua baris sebagai posisi awal penempatan blok. Perulangan kedua akan mencoba semua kolom sebagai posisi awal penempatan blok. Perulangan ketiga akan mencoba setiap variasi blok di posisi yang telah ditentutukan oleh baris (perulangan satu) dan kolom (perulangan kedua).

- Sebelum menempatkan blok, (`isValidPlaceBlock(var, r, c)` akan mengecek terlebih dahulu, apakah blok bisa ditempatkan di posisi baris (r) dan kolom (k) tersebut.
- Jika blok bisa ditempatkan pada posisi baris (r) dan kolom (k), maka program akan menjalankan fungsi (`placeValidBlock(var, r, c, s))`, dan count akan bertambah satu. Untuk kasus tertentu, yaitu ketika blok memiliki karakter awal bukan alfabet melainkan spasi, maka akan dicari terlebih dahulu karakter dari blok tersebut agar memudahkan dalam proses menempatkan blok. Kemudian, akan dipanggil fungsi solve dengan blockIdx bertambah satu
- Jika solusi telah ditemukan, yaitu solved bernilai true, program akan menghentikan percobaan lain.
- Jika tidak berhasil, program akan menjalankan backtrack dengan menghapus blok terakhir yang baru saja ditempatkan.

Potongan program dari fungsi solve(int blockIdx)

```
public void solve(int blockIdx) {
        // Kalau jumlah alfabet di blok kurang dari N * M,
maka board tidak mungkin penuh
        if (countAlphabetsInBlocks(blocks) < N * M) {
            solved = false;
            return;
        }

        // Jika semua blok sudah diletakkan, maka board uda
selesai (tinggal di cek aja penuh atau engga)
        if (blockIdx >= blocks.size()) {
            if (isBoardFull()) {
                solved = true;
            }
            return;
        }

        char[][] block = blocks.get(blockIdx);
        List<char[][]> variations =
generateVariations(block);
        // printVariations(variations); // Debugging

        for (int r = 0; r <= N; r++) {
            for (int c = 0; c <= M; c++) {
                for (char[][] var : variations) {
                     if (isValidPlaceBlock(var, r, c)) {
                        // Untuk kasus input yang awalnya
bukan alfabet, cari alfabetnya dulu
                        char s = findAlfabet(var);
                        placeValidBlock(var, r, c, s);
                        count++;
                        solve(blockIdx + 1);
                        if (solved) {
```

```
                            return;
                        }
                        removeLastBlock(var, r, c);
                    }
                }
            }
        }
    }
```

3. **Fungsi isValidPlaceBlock(char[][] block, int r, int c)**
    - Di awal, program akan melakukan pengecekan untuk memastikan posisi awal blok tidak keluar dari board
    - Setelah itu, program akan melakukan iterasi baris dan kolom untuk melakukan pengecekan di tiap sel. Program akan melakukan pengecekan bahwa *cell* pada blok berisi karakter alfabet atau bukan dan melakukan pengecekan pada posisi pada board (apakah sudah ada karakter alfabet lain atau belum)

    Potongan program dari isValidPlaceBlock(char[][] block, int r, int c)

```
// Mengecek apakah block dapat diletakkan pada board
    private boolean isValidPlaceBlock(char[][] block, int r,
int c) {
        int rowBlock = block.length;
        int colBlock = block[0].length;

        if (r < 0 || r + rowBlock > N || c < 0 || c +
colBlock > M) {
            return false;
        }

        for (int i = 0; i < rowBlock; i++) {
            for (int j = 0; j < colBlock; j++) {
                if (block[i][j] != '.' && board[r + i][c + j]
!= '.') {
                    return false;
                }
            }
        }
        return true;
    }
```

4. **Fungsi placeValidBlock(char[][] block, int r, int c, char s)**
    - Program akan melakukan iterasi pada baris dan kolom, lalu melakukan pengecekan pada blok. Jika *cell* blok berisi karakter alfabet (bukan spasi yang

ditandai dengan ‘.’), *cell* karakter alfabet dari blok akan mengisi posisi dari papan tersebut.

Potongan program placeValidBlock(char[][] block, int r, int c, char s)

```
// Menempatkan block yang valid pada board
    private void placeValidBlock(char[][] block, int r, int
c, char s) {
        int rowBlock = block.length;
        int colBlock = block[0].length;
        for (int i = 0; i < rowBlock; i++) {
            for (int j = 0; j < colBlock; j++) {
                if (block[i][j] != '.') {
                    board[r + i][c + j] = s;
                }
            }
        }
    }
```

5. **Fungsi removeLastBlock(char[][] block, int r, int c)**
    - Program akan melakukan iterasi pada baris dan kolom, lalu melakukan pengecekan pada blok. Jika *cell* blok berisi karakter alfabet (bukan spasi yang ditandai dengan ‘.’), *cell* dari board akan diisi dengan ‘.’ yang menandakan bahwa *cell* pada board tersebut kembali kosong

    Potongan program removeLastBlock(char[][] block, int r, int c)

```
// Menghapus block terakhir dari board (backtracking)
    private void removeLastBlock(char[][] block, int r, int
c) {
        int rowBlock = block.length;
        int colBlock = block[0].length;
        for (int i = 0; i < rowBlock; i++) {
            for (int j = 0; j < colBlock; j++) {
                if (block[i][j] != '.') {
                    board[r + i][c + j] = '.';
                }
            }
        }
    }
```

6. **Fungsi generateVariations(char[][] block)**
    - Pertama, program akan melakukan iterasi sebanyak 4 kali untuk mendapatkan variasi dari rotasi (untuk 0, 90, 180, dan 270 derajat). Setiap variasi yang didapatkan akan disalin ke variations.

- Setelah itu, blok akan dicerminkan dan dilakukan iterasi sebanyak 4 kali juga untuk mendapatkan variasi dari rotasinya.
- Dengan demikian, suatu blok memiliki maksimal 8 variasi yang dapat terjadi

```
// Menghasilkan semua variasi unik dari blok (rotasi 90
derajat dan pencerminan horizontal)
    private List<char[][]> generateVariations(char[][] block)
{
        List<char[][]> variations = new ArrayList<>();

        // Variasi dari blok asli sebanyak 4 kali
        char[][] currentBlock = block;
        for (int i = 0; i < 4; i++) {
            variations.add(copyMatrix(currentBlock));
            currentBlock = rotate(currentBlock);
        }

        // Variasi dari blok yang dicerminkan dan rotasinya
sebanya 4 kali juga
        char[][] mirrored = mirror(block);
        for (int i = 0; i < 4; i++) {
            variations.add(copyMatrix(mirrored));
            mirrored = rotate(mirrored);
        }
        return variations;
    }
```

7. **Fungsi rotate(char[][] block)**
    - Program akan melakukan iterasi untuk baris dan kolom blok, lalu memutarkan elemen sejauh 90 derajat

```
// Rotasi matrix 90 derajat
    private char[][] rotate(char[][] block) {
        int row = block.length;
        int col = block[0].length;
        char[][] rotated = new char[col][row];
        for (int i = 0; i < row; i++) {
            for (int j = 0; j < col; j++) {
                rotated[j][row - 1 - i] = block[i][j];
            }
        }
        return rotated;
    }
```

8. **Fungsi mirror(char[][] block)**
    - Program akan melakukan iterasi pada baris dan kolom blok, lalu akan dicerminkan

```
// Pencerminan horizontal matrix
    private char[][] mirror(char[][] block) {
        int row = block.length;
        int col = block[0].length;
        char[][] mirrored = new char[row][col];
        for (int i = 0; i < row; i++) {
            for (int j = 0; j < col; j++) {
                mirrored[i][col - 1 - j] = block[i][j];
            }
        }
        return mirrored;
    }
```

BAB III

SOURCE CODE

1. **Struktur Proyek**

Proyek ini memiliki struktur direktori dan file sebagai berikut:

```
Tucil1_13523156/
├── bin/
│   └── ColorBoard.class
│   ├── GUIPuzzle.class
│   ├── GUIPuzzle$BoardPanel.class
│   ├── Main.class
```

```
│   ├── ReadInput.class
│   ├── SolvePuzzle.class
├── doc/
├── src/
│   └── ColorBoard.java
│   ├── GUIPuzzle.java
│   ├── Main.java
│   ├── ReadInput.java
│   ├── SolvePuzzle.java
├── test/
│   └── result
│   ├── testcase
└── README.md
```

Keterangan

- bin/: Berisi file class dari setiap source code yang ada di di src. File ini lah yang dapat dieksekusi untuk menjalankan programnya
- doc/: Berisi laporan tugas kecil 1 dalam bentuk .pdf
- src/: Berisi kode sumber program
    a. ColorBoard.java : Implementasi untuk membuat board berwarna dan impelementasi untuk menyimpan solusi dalam bentuk .txt atau .jpg
    b. GUIPuzzle : Implementasi GUI dari program IQ Puzzler Pro
    c. Main.java : Entry point program jika ingin dieksekusi melalui CLI. Main.java akan menjalankan ReadInput, SolvePuzzle, dan ColorBoard
    d. ReadInput.java : Berisi implementasi untuk membaca dan melakukan *parsing* pada data input sekaligus melakukan validasi input
    e. SolvePuzzle.java : Implementasi algoritma brute force untuk menyelesaikan masalah puzzle
- test/: Berisi direktori testcase dan result
    a. testcase: Berisi file test untuk program
    b. result: Berisi file penyimpanan solusi dari testcase
- README.md : Berisi file readme yang mencakup deskripsi singkat, persyaratan, instalasi, kompilasi, cara menjalankan program dan nama author

2. ColorBoard.java

- ColorBoard merupakan program untuk menampilkan dan menyimpan board warna dalam dua format, yaitu .txt dan .jpg
- Terdapat dua array warna, yaitu array colors dan image_colors. colors berisi kode ANSI untuk mewarnai karakter pada terminal, sedangkan image_colors digunakan untuk mewarnai karakter dan *cell* pada format .jpg
- Program menampilkan board melalui fungsi printBoard

Potongan Program dari ColorBoard.java

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

public class ColorBoard {
    private static final String reset = "\u001B[0m";
    private static final String[] colors = {
        "\u001B[38;5;9m",  // A - Red
        "\u001B[38;5;10m", // B - Green
        "\u001B[38;5;11m", // C - Yellow
        "\u001B[38;5;12m", // D - Blue
        "\u001B[38;5;13m", // E - Purple
        "\u001B[38;5;14m", // F - Cyan
        "\u001B[38;5;1m",  // G - Bright Red
        "\u001B[38;5;2m",  // H - Bright Green
        "\u001B[38;5;3m",  // I - Bright Yellow
        "\u001B[38;5;4m",  // J - Bright Blue
        "\u001B[38;5;5m",  // K - Bright Purple
        "\u001B[38;5;6m",  // L - Bright Cyan
        "\u001B[38;5;0m",  // M - Black
        "\u001B[38;5;1m",  // N - Red Background
        "\u001B[38;5;2m",  // O - Green Background
        "\u001B[38;5;3m",  // P - Yellow Background
        "\u001B[38;5;4m",  // Q - Blue Background
        "\u001B[38;5;5m",  // R - Purple Background
        "\u001B[38;5;6m",  // S - Cyan Background
        "\u001B[38;5;7m",  // T - Gray Background
        "\u001B[38;5;8m",  // U - Bright Red Background
        "\u001B[38;5;9m",  // V - Bright Green Background
        "\u001B[38;5;10m", // W - Bright Yellow Background
        "\u001B[38;5;11m", // X - Bright Blue Background
        "\u001B[38;5;12m", // Y - Bright Purple Background
        "\u001B[38;5;13m"  // Z - Bright Cyan Background
    };

    public static final Color[] image_colors = {
        new Color(194, 54, 33),    // Red
        new Color(37, 188, 36),    // Green
        new Color(173, 173, 39),   // Yellow
        new Color(73, 46, 225),    // Blue
        new Color(211, 56, 211),   // Purple
        new Color(51, 187, 200),   // Cyan
        new Color(255, 0, 0),      // Bright Red
        new Color(0, 255, 0),      // Bright Green
        new Color(255, 255, 0),    // Bright Yellow
        new Color(0, 0, 255),      // Bright Blue
        new Color(255, 0, 255),    // Bright Purple
        new Color(0, 255, 255),    // Bright Cyan
        Color.BLACK,               // Black
```

```
        new Color(194, 54, 33),    // Red Background
        new Color(37, 188, 36),    // Green Background
        new Color(173, 173, 39),   // Yellow Background
        new Color(73, 46, 225),    // Blue Background
        new Color(211, 56, 211),   // Purple Background
        new Color(51, 187, 200),   // Cyan Background
        Color.GRAY,                // Gray Background
        new Color(255, 0, 0),      // Bright Red Background
        new Color(0, 255, 0),      // Bright Green Background
        new Color(255, 255, 0),    // Bright Yellow Background
        new Color(0, 0, 255),      // Bright Blue Background
        new Color(255, 0, 255),    // Bright Purple Background
        new Color(0, 255, 255)     // Bright Cyan Background
    };

    // Mencetak board dengan block berwarna ke layar
    public static void printBoard(char[][] board) {
        Map<Character, String> CM = new HashMap<>();

        for (int i = 0; i < board.length; i++) {
            for (int j = 0; j < board[i].length; j++) {
                char c = board[i][j];
                if (c != '.' && !CM.containsKey(c)) {
                    CM.put(c, colors[c - 'A']);
                }
                if (c != '.') {
                    System.out.print(CM.get(c) + c + reset);
                } else {
                    System.out.print(c);
                }
            }
            System.out.println();
        }
    }

    // Menyimpan solusi dalam format txt
    public static void saveTextSolution(char[][] board, String
filePath) throws IOException {
        try (FileWriter writer = new FileWriter(filePath)) {
            for (int i = 0; i < board.length; i++) {
                for (int j = 0; j < board[i].length; j++) {
                    writer.write(board[i][j]);
                }
                writer.write(System.lineSeparator());
            }
        }
    }

    // Menyimpan solusi dalam format jpg
    public static void saveImageSolution(char[][] board, String
filePath) throws IOException {
```

```
        int cellSize = 100;
        int width = board[0].length * cellSize;
        int height = board.length * cellSize;

        BufferedImage image = new BufferedImage(width, height,
BufferedImage.TYPE_INT_RGB);
        Graphics2D g = image.createGraphics();
        g.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);

        Map<Character, Color> CM = new HashMap<>();

        for (int i = 0; i < board.length; i++) {
            for (int j = 0; j < board[i].length; j++) {
                char c = board[i][j];
                if (c != '.' && !CM.containsKey(c)) {
                    CM.put(c, image_colors[c - 'A']);
                }
                g.setColor(c == '.' ? Color.WHITE : CM.get(c));
                g.fillRect(j * cellSize, i * cellSize, cellSize,
cellSize);
                g.setColor(Color.BLACK);
                g.drawRect(j * cellSize, i * cellSize, cellSize,
cellSize);
                if (c != '.') {
                    g.setColor(Color.BLACK);
                    g.setFont(new Font("Lexend", Font.BOLD,
cellSize / 2));
                    FontMetrics fm = g.getFontMetrics();
                    int x = j * cellSize + (cellSize -
fm.charWidth(c)) / 2;
                    int y = i * cellSize + ((cellSize -
fm.getHeight()) / 2) + fm.getAscent();
                    g.drawString(String.valueOf(c), x, y);
                }
            }
        }
        g.dispose();
        ImageIO.write(image, "jpg", new File(filePath));
    }
}
```

3. **GUIPuzzle.java**

- Program ini merupakan implementasi GUI untuk IQ Puzzler Pro. Program ini juga sebagai *entry point* untuk melaksanakan program melalui GUI

Potongan Program dari GUIPuzzle.java

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

public class GUIPuzzle {
    private JFrame frame;
    private JPanel mainPanel, topPanel, bottomPanel;
    private JButton loadButton, solveButton, saveButton;
    private BoardPanel boardPanel;
    private JLabel infoLabel;
    private JFileChooser fileChooser;
    private SolvePuzzle currentSolver; // Stores current solver
after loading
    private File currentPuzzleFile; // Stores current puzzle file
after loading

    public GUIPuzzle() {
        initializeGUI();
    }

    private void initializeGUI() {
        frame = new JFrame("Puzzle Solver");
        mainPanel = new JPanel(new BorderLayout());

        // untuk tombol di atas
        topPanel = new JPanel();
        loadButton = new JButton("Load Puzzle");
        solveButton = new JButton("Solve Puzzle");
        saveButton = new JButton("Save Solution");

        loadButton.addActionListener(_ -> loadPuzzle());
        solveButton.addActionListener(_ -> solvePuzzle());
        saveButton.addActionListener(_ -> saveSolution());

        topPanel.add(loadButton);
        topPanel.add(solveButton);
        topPanel.add(saveButton);

        // untuk board
        boardPanel = new BoardPanel();
        boardPanel.setPreferredSize(new Dimension(400, 400));

        JPanel boardContainer = new JPanel(new GridBagLayout());
        boardContainer.add(boardPanel);

        // untuk info di bawah
        bottomPanel = new JPanel();
        infoLabel = new JLabel("Selamat Datang di IQ Puzzle
Solver!");
        bottomPanel.add(infoLabel);
```

```
        mainPanel.add(topPanel, BorderLayout.NORTH);
        mainPanel.add(boardContainer, BorderLayout.CENTER);
        mainPanel.add(bottomPanel, BorderLayout.SOUTH);

        frame.add(mainPanel);
        frame.pack();
        frame.setSize(800, 800);
        frame.setLocationRelativeTo(null);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);

        fileChooser = new JFileChooser();
    }

    private void loadPuzzle() {
        // me-reset solver dan board
        currentSolver = null;
        boardPanel.setBoard(null);
        infoLabel.setText("Selamat Datang di IQ Puzzle Solver!");

        // Me-reset input data di ReadInput
        ReadInput.N = 0;
        ReadInput.M = 0;
        ReadInput.P = 0;
        ReadInput.S = "";
        ReadInput.blocks.clear();

        String[] options = {"Browse File", "Enter Path"};
        int choice = JOptionPane.showOptionDialog(frame,
                "Pilih metode untuk load file:",
                "Load Puzzle",
                JOptionPane.DEFAULT_OPTION,
                JOptionPane.QUESTION_MESSAGE,
                null, options, options[0]);

        String filePath = null;
        if (choice == 0) { // Untuk metode browser file
            int returnValue = fileChooser.showOpenDialog(frame);
            if (returnValue == JFileChooser.APPROVE_OPTION) {
                File file = fileChooser.getSelectedFile();
                filePath = file.getAbsolutePath();
                currentPuzzleFile = file;
            }
        } else if (choice == 1) { // Untuk metode input path
            filePath = JOptionPane.showInputDialog(frame, "Masukkan
path file:");
            if (filePath != null && !filePath.trim().isEmpty()) {
                currentPuzzleFile = new File(filePath);
            }
        }
```

```
        if (filePath == null || filePath.trim().isEmpty()) {
            JOptionPane.showMessageDialog(frame, "File tidak valid
atau tidak dipilih.", "Error", JOptionPane.ERROR_MESSAGE);
            return;
        }

        try {
            // Baca dan parse file puzzle
            boolean isValidInput = ReadInput.readFile(filePath);
            if (!isValidInput) {
                JOptionPane.showMessageDialog(frame, "Input ada
yang tidak valid. Program dihentikan", "Error",
JOptionPane.ERROR_MESSAGE);
                return;
            }
            // Update info label dengan detail puzzle
            String info = "<html>Ukuran Papan: " + ReadInput.N + "
x " + ReadInput.M +
                          "<br>Jumlah Blok: " +
ReadInput.blocks.size() +
                          "<br>Jenis Kasus: " + ReadInput.S +
"</html>";
            infoLabel.setText(info);

            // inisialiasi board
            char[][] initialBoard = new
char[ReadInput.N][ReadInput.M];
            for (int i = 0; i < ReadInput.N; i++) {
                for (int j = 0; j < ReadInput.M; j++) {
                    initialBoard[i][j] = '.';
                }
            }
            boardPanel.setBoard(initialBoard);
            boardPanel.setPreferredSize(new Dimension(ReadInput.M *
boardPanel.getCellSize(),
                                                      ReadInput.N *
boardPanel.getCellSize()));
            boardPanel.revalidate();

            // inisialisasi solvepuzzle
            currentSolver = new SolvePuzzle(ReadInput.N,
ReadInput.M, ReadInput.blocks);
        } catch (IOException ex) {
            JOptionPane.showMessageDialog(frame, "Terjadi
kesalahan: " + ex.getMessage(), "Error",
JOptionPane.ERROR_MESSAGE);
        }
    }

    private void solvePuzzle() {
        if (currentSolver == null) {
```

```
            JOptionPane.showMessageDialog(frame, "Puzzle belum
di-load.", "Selamat Datang di IQ Puzzle Solver!",
JOptionPane.INFORMATION_MESSAGE);
            return;
        }
        long startTime = System.currentTimeMillis();
        currentSolver.solve(0);
        long endTime = System.currentTimeMillis();

        if (currentSolver.isSolved()) {
            boardPanel.setBoard(currentSolver.getBoard());
            String info = "<html>Solusi ditemukan!<br>" +
                          "Waktu Pencarian: " + (endTime -
startTime) + " ms<br>" +
                          "Banyak Kasus yang Ditinjau: " +
currentSolver.getCount() + "</html>";
            infoLabel.setText(info);
        } else {
            String info = "<html>Solusi tidak ditemukan.<br>" +
                          "Banyak Kasus yang Ditinjau: " +
currentSolver.getCount() + "</html>";
            infoLabel.setText(info);
        }
    }

    private void saveSolution() {
        // Mengecek uda solve atau belom
        if (currentSolver == null || !currentSolver.isSolved()) {
            JOptionPane.showMessageDialog(frame, "Tidak ada solusi
untuk disimpan.", "Info", JOptionPane.INFORMATION_MESSAGE);
            return;
        }

        String[] formatOptions = {"TXT", "JPG"};
        int formatChoice = JOptionPane.showOptionDialog(frame,
                "Ingin Menyimpan Solusi dalam Format Apa?",
                "Pilih Format",
                JOptionPane.DEFAULT_OPTION,
                JOptionPane.QUESTION_MESSAGE,
                null,
                formatOptions,
                formatOptions[0]);
        if (formatChoice == JOptionPane.CLOSED_OPTION) {
            return;
        }
       // browse folder untuk menyimpan file
        int returnValue = fileChooser.showSaveDialog(frame);
        if (returnValue == JFileChooser.APPROVE_OPTION) {
            File file = fileChooser.getSelectedFile();
            String filePath = file.getAbsolutePath();
            try {
```

```
                if (formatChoice == 0) { // Untuk format text
                    if (!filePath.toLowerCase().endsWith(".txt")) {
                        filePath += ".txt";
                    }

ColorBoard.saveTextSolution(currentSolver.getBoard(), filePath);
                    JOptionPane.showMessageDialog(frame, "Solusi
berhasil disimpan di " + filePath,
                            "Berhasil",
JOptionPane.INFORMATION_MESSAGE);
                } else if (formatChoice == 1) { // Untuk format
image
                    if (!(filePath.toLowerCase().endsWith(".jpg")
|| filePath.toLowerCase().endsWith(".jpeg"))) {
                        filePath += ".jpg";
                    }

ColorBoard.saveImageSolution(currentSolver.getBoard(), filePath);
                    JOptionPane.showMessageDialog(frame, "Solusi
berhasil disimpan di " + filePath,
                            "Berhasil",
JOptionPane.INFORMATION_MESSAGE);
                }
            } catch (IOException ex) {
                JOptionPane.showMessageDialog(frame, "Terjadi
kesalahan saat menyimpan: " + ex.getMessage(),
                        "Error", JOptionPane.ERROR_MESSAGE);
            }
        }
    }

    // Panel untuk menampilkan board
    class BoardPanel extends JPanel {
        private char[][] board;
        private int cellSize = 40;

        public int getCellSize() {
            return cellSize;
        }

        public void setBoard(char[][] board) {
            this.board = board;
            repaint();
        }

        @Override
        protected void paintComponent(Graphics g) {
            super.paintComponent(g);
            if (board == null) {
                return;
            }
```

```
            int rows = board.length;
            int cols = board[0].length;
            for (int i = 0; i < rows; i++) {
                for (int j = 0; j < cols; j++) {
                    char c = board[i][j];
                    if (c == '.') {
                        g.setColor(Color.WHITE);
                    } else {
                        g.setColor(getColorForChar(c));
                    }
                    g.fillRect(j * cellSize, i * cellSize,
cellSize, cellSize);
                    g.setColor(Color.BLACK);
                    g.drawRect(j * cellSize, i * cellSize,
cellSize, cellSize);

                    if (c != '.') {
                        g.setColor(Color.BLACK);
                        Font lexendFont = new Font("Lexend",
Font.BOLD, cellSize - 10);
                        g.setFont(lexendFont);
                        FontMetrics fm =
g.getFontMetrics(lexendFont);
                        String letter = String.valueOf(c);
                        int textWidth = fm.stringWidth(letter);
                        int textHeight = fm.getAscent();
                        int x = j * cellSize + (cellSize -
textWidth) / 2;
                        int y = i * cellSize + ((cellSize -
fm.getHeight()) / 2) + fm.getAscent();
                        g.drawString(letter, x, y);
                    }
                }
            }
        }

        private Color getColorForChar(char c) {
            int index = c - 'A';
            Color[] colors = {
                new Color(194, 54, 33),    // Red
                new Color(37, 188, 36),    // Green
                new Color(173, 173, 39),   // Yellow
                new Color(73, 46, 225),    // Blue
                new Color(211, 56, 211),   // Purple
                new Color(51, 187, 200),   // Cyan
                new Color(255, 0, 0),      // Bright Red
                new Color(0, 255, 0),      // Bright Green
                new Color(255, 255, 0),    // Bright Yellow
                new Color(0, 0, 255),      // Bright Blue
                new Color(255, 0, 255),    // Bright Purple
                new Color(0, 255, 255),    // Bright Cyan
```

```
                Color.BLACK,                // Black
                new Color(194, 54, 33),     // Red Background
                new Color(37, 188, 36),     // Green Background
                new Color(173, 173, 39),    // Yellow Background
                new Color(73, 46, 225),     // Blue Background
                new Color(211, 56, 211),    // Purple Background
                new Color(51, 187, 200),    // Cyan Background
                Color.GRAY,                 // Gray Background
                new Color(255, 0, 0),       // Bright Red Background
                new Color(0, 255, 0),       // Bright Green
Background
                new Color(255, 255, 0),     // Bright Yellow
Background
                new Color(0, 0, 255),       // Bright Blue
Background
                new Color(255, 0, 255),     // Bright Purple
Background
                new Color(0, 255, 255)      // Bright Cyan
Background
            };
            if (index >= 0 && index < colors.length)
                return colors[index];
            return Color.DARK_GRAY;
        }
    }

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

4. Main.java

- Main.java merupakan program sebagai *entry point* untuk versi CLI. Program dimulai dengan membaca input dan diproses oleh ReadInput untuk di-*parsing* dan divalidasi. Setelah itu, baru akan di-solve dimulai dengan index blok ke-0. Setelah diproses akan ditampilkan output dan menyimpan solusi jika pengguna ingin menyimpan

  Potongan Program dari Main.java

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

public class Main {
    public static void main(String[] args) {
        // Membaca path file dari input user
        Scanner scanner = new Scanner(System.in);
        System.out.print("Masukkan path file: ");
        String filePath = scanner.nextLine();

        try {
            boolean isValidInput = ReadInput.readFile(filePath);
            if (!isValidInput) {
                System.out.println("Input ada yang tidak valid.
Program dihentikan,");
                return;
            }
            // ReadInput.printParsedData(); // Debugging

            SolvePuzzle solver = new SolvePuzzle(ReadInput.N,
ReadInput.M, ReadInput.blocks);
            long startTime = System.currentTimeMillis();
            solver.solve(0);
            long endTime = System.currentTimeMillis();

            if (solver.isSolved()) {
                System.out.println("Solusi ditemukan!!!");
                System.out.println("===== Board =====");
                ColorBoard.printBoard(solver.getBoard());
                System.out.println("=================\n");

                System.out.println("Waktu Pencarian " + (endTime -
startTime) + " ms");
                System.out.println("Banyak Kasus yang Ditinjau: " +
solver.getCount());

                System.out.println("Apakah Anda ingin Menyimpan
Solusi (y/n)? ");
                String save =
scanner.nextLine().trim().toLowerCase();
                if (save.equals("y")) {
                    System.out.println("Ingin Menyimpan Solusi
dalam Format Apa (txt/jpg)? ");
                    String format =
scanner.nextLine().trim().toLowerCase();

                    // Menyesuaikan nama file sesuai dengan tc ke
berapa
                    String fileName = new File(filePath).getName();
                    String baseName = fileName.substring(0,
fileName.lastIndexOf('.'));
```

```
                    String suffix = baseName.replace("tc", "");
                    String resultFileName = "result" + suffix + "."
+ format;
                    String savePath = "../test/result/" +
resultFileName;

                    if (format.equals("txt")) { // Menyimpan solusi
dalam format text

ColorBoard.saveTextSolution(solver.getBoard(), savePath);
                        System.out.println("Solusi berhasil
disimpan di " + savePath);
                    } else if (format.equals("jpg")) { // Menyimpan
solusi dalam format jpg

ColorBoard.saveImageSolution(solver.getBoard(), savePath);
                        System.out.println("Solusi berhasil
disimpan di " + savePath);
                    } else {
                        System.out.println("Format tidak valid");
                    }
                } else if (save.equals("n")) {
                    System.out.println("Solusi tidak disimpan");
                } else {
                    System.out.println("Input tidak valid");
                }
            } else {
                System.out.println("Solusi tidak ditemukan.");
                System.out.println("Banyak Kasus yang Ditinjau: " +
solver.getCount());
            }
        } catch (IOException e) {
            System.out.println("Ada kesalahan");
        } finally {
            scanner.close();
        }
    }
}
```

5. **ReadInput.java**

- Program ini akan membaca nilai dan melakukan validasi. Nilai yang akan dibaca berupa ukuran board, yaitu M dan N, jumlah blok, yaitu P, jenis kasus, yaitu S, dan blok-blok dari puzzle

  Potongan Program dari ReadInput.java

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

public class ReadInput {
    static int N, M, P;
    static String S;
    static List<char[][]> blocks = new ArrayList<>();

    // Membaca file dan memparsing input
    static boolean readFile(String filePath) throws IOException {
        try (BufferedReader br = new BufferedReader(new
FileReader(filePath))) {
            // Membaca N, M, P
            String[] firstLine = br.readLine().split(" ");
            // Validasi inputan N, M dan P (Harus berbentuk
integer)
            try {
                N = Integer.parseInt(firstLine[0]);
                M = Integer.parseInt(firstLine[1]);
                P = Integer.parseInt(firstLine[2]);
            } catch (NumberFormatException e) {
                System.out.println("Input N, M, atau P tidak
valid.");
                return false;
            }

            // Membaca Jenis Kasus
            S = br.readLine().trim();

            // Membaca block
            List<String> blockLines = new ArrayList<>();
            char prev = '0';

            String L;
            while ((L = br.readLine()) != null) {
                // System.out.println(L); // Debugging
                if (L.trim().isEmpty()) {
                    continue;
                }

                // Melakukan validasi inputan blok apakah semuanya
alfabet atau spasi
                if (!L.matches("[A-Z ]+")) {
                    System.out.println("Inputan block ada yang
tidak valid");
                    return false;
                }

                // Menentukan alfabet pertama
                String charWithoutSpace = L.trim();
                char firstChar = charWithoutSpace.charAt(0);
```

```
                // Jika blockLines tidak kosong dan alfabet
berubah, blok sebelumnya uda kelar
                if (!blockLines.isEmpty() && firstChar != prev) {
                    char[][] block = convertToBlock(blockLines);
                    if (block == null) {
                        System.out.println("Blok tidak valid");
                        return false;
                    }
                    blocks.add(block);
                    blockLines.clear();
                }

                blockLines.add(L);
                prev = firstChar;
            }

            // Jika sudah sampai di blok terakhir
            if (!blockLines.isEmpty()) {
                char[][] block = convertToBlock(blockLines);
                if (block == null) {
                    System.out.println("Blok tidak valid");
                    return false;
                }
                blocks.add(block);
            }

            // Validasi Jumlah Block
            if (blocks.size() != P) {
                System.out.println(blocks.size());
                System.out.println("Jumlah blok tidak sesuai dengan
P");
                return false;
            }

            br.close();
        }
        // printParsedData(); // Debugging
        return true;
    }

    // Mengubah list string menjadi array 2D
    static char[][] convertToBlock(List<String> blockLines) {
        int row = blockLines.size();
        int col = 0;

        // Mencari lebar maksimum dari suatu blok
        for (String line : blockLines) {
            col = Math.max(col, line.length());
        }
```

```
        char[][] block = new char[row][col];
        char firstBlockChar = ' ';
        boolean validBlock = false;

        for (int i = 0; i < row; i++) {
            String L = blockLines.get(i);
            for (int j = 0; j < col; j++) {
                if (j < L.length()) {}
                char c = j < L.length() ? L.charAt(j) : '.';
                block[i][j] = (c == ' ' ? '.' : c);

                // Validasi apakah block memiliki alfabet yang sama
                if (c != '.' && c != ' ') {
                    if (!validBlock) {
                        firstBlockChar = c;
                        validBlock = true;
                    } else if (c != firstBlockChar) {
                        return null;
                    }
                }
            }
        }
        return block;
    }

    // Mencetak hasil data input yang uda di-parsing
    static void printParsedData() {
        System.out.println("Ukuran Papan: " + N + " x " + M);
        System.out.println("Jumlah Blok: " + P);
        System.out.println("Jenis Kasus: " + S);
        System.out.println("Blok Puzzle:");
        for (char[][] block : blocks) {
            printBlock(block);
        }
    }

    // Mencetak blok dalam bentuk matriks
    static void printBlock(char[][] block) {
        for (char[] row : block) {
            for (char cell : row) {
                System.out.print(cell + "");
            }
            System.out.println();
        }
    }
}
```

6. **SolvePuzzle.java**

- Program ini merupakan implementasi brute force untuk menyelesaikan puzzle seperti yang telah dijelaskan pada BAB II
  Potongan Program dari SolvePuzzle.java

```
import java.util.*;

public class SolvePuzzle {
    private int N, M;
    private char[][] board;
    private List<char[][]> blocks;
    private boolean solved = false;
    private long count = 0;

    public SolvePuzzle(int N, int M, List<char[][]> blocks) {
        this.N = N;
        this.M = M;
        this.blocks = blocks;
        this.board = new char[N][M];

        // Inisialisasi board dengan '.'
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < M; j++) {
                board[i][j] = '.';
            }
        }
    }

    public boolean isSolved() {
        return solved;
    }

    public long getCount() {
        return count;
    }

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

    public void solve(int blockIdx) {
        // Kalau jumlah alfabet di blok kurang dari N * M, maka
board tidak mungkin penuh
        if (countAlphabetsInBlocks(blocks) < N * M) {
            solved = false;
            return;
        }

        // Jika semua blok sudah diletakkan, maka board uda selesai
(tinggal di cek aja penuh atau engga)
        if (blockIdx >= blocks.size()) {
```

```
            if (isBoardFull()) {
                solved = true;
            }
            return;
        }

        char[][] block = blocks.get(blockIdx);
        List<char[][]> variations = generateVariations(block);
        // printVariations(variations); // Debugging

        for (int r = 0; r <= N; r++) {
            for (int c = 0; c <= M; c++) {
                for (char[][] var : variations) {
                     if (isValidPlaceBlock(var, r, c)) {
                        // Untuk kasus input yang awalnya bukan
alfabet, cari alfabetnya dulu
                        char s = findAlfabet(var);
                        placeValidBlock(var, r, c, s);
                        count++;
                        solve(blockIdx + 1);
                        if (solved) {
                            return;
                        }
                        removeLastBlock(var, r, c);
                    }
                }
            }
        }
    }

    // Mengecek apakah board sudah terisi penuh
    private boolean isBoardFull() {
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < M; j++) {
                if (board[i][j] == '.') {
                    return false;
                }
            }
        }
        return true;
    }

    // Mencari alfabet pada blok (khusus input yang awalnya bukan
alfabet)
    public char findAlfabet(char[][] block) {
        for (int i = 0; i < block.length; i++) {
            for (int j = 0; j < block[i].length; j++) {
                if (block[i][j] != '.') {
                    return block[i][j];
                }
            }
```

```
        }
        return '\0';
    }

    // Mengecek apakah block dapat diletakkan pada board
    private boolean isValidPlaceBlock(char[][] block, int r, int c)
{
        int rowBlock = block.length;
        int colBlock = block[0].length;

        if (r < 0 || r + rowBlock > N || c < 0 || c + colBlock > M)
{
            return false;
        }

        for (int i = 0; i < rowBlock; i++) {
            for (int j = 0; j < colBlock; j++) {
                if (block[i][j] != '.' && board[r + i][c + j] !=
'.') {
                    return false;
                }
            }
        }
        return true;
    }

    // Menempatkan block yang valid pada board
    private void placeValidBlock(char[][] block, int r, int c, char
s) {
        int rowBlock = block.length;
        int colBlock = block[0].length;
        for (int i = 0; i < rowBlock; i++) {
            for (int j = 0; j < colBlock; j++) {
                if (block[i][j] != '.') {
                    board[r + i][c + j] = s;
                }
            }
        }
    }

    // Menghapus block terakhir dari board (backtracking)
    private void removeLastBlock(char[][] block, int r, int c) {
        int rowBlock = block.length;
        int colBlock = block[0].length;
        for (int i = 0; i < rowBlock; i++) {
            for (int j = 0; j < colBlock; j++) {
                if (block[i][j] != '.') {
                    board[r + i][c + j] = '.';
                }
            }
        }
```

```
    }

    // Menghasilkan semua variasi unik dari blok (rotasi 90 derajat
dan pencerminan horizontal)
    private List<char[][]> generateVariations(char[][] block) {
        List<char[][]> variations = new ArrayList<>();

        // Variasi dari blok asli sebanyak 4 kali
        char[][] currentBlock = block;
        for (int i = 0; i < 4; i++) {
            variations.add(copyMatrix(currentBlock));
            currentBlock = rotate(currentBlock);
        }

        // Variasi dari blok yang dicerminkan dan rotasinya sebanya
4 kali juga
        char[][] mirrored = mirror(block);
        for (int i = 0; i < 4; i++) {
            variations.add(copyMatrix(mirrored));
            mirrored = rotate(mirrored);
        }
        return variations;
    }

    // Membuat salinan matrix
    private char[][] copyMatrix(char[][] matrix) {
        int row = matrix.length;
        int col = matrix[0].length;
        char[][] copy = new char[row][col];
        for (int i = 0; i < row; i++) {
            System.arraycopy(matrix[i], 0, copy[i], 0, col);
        }
        return copy;
    }

    // Rotasi matrix 90 derajat
    private char[][] rotate(char[][] block) {
        int row = block.length;
        int col = block[0].length;
        char[][] rotated = new char[col][row];
        for (int i = 0; i < row; i++) {
            for (int j = 0; j < col; j++) {
                rotated[j][row - 1 - i] = block[i][j];
            }
        }
        return rotated;
    }

    // Pencerminan horizontal matrix
    private char[][] mirror(char[][] block) {
        int row = block.length;
```

```
        int col = block[0].length;
        char[][] mirrored = new char[row][col];
        for (int i = 0; i < row; i++) {
            for (int j = 0; j < col; j++) {
                mirrored[i][col - 1 - j] = block[i][j];
            }
        }
        return mirrored;
    }

    // Menghitung jumlah alfabet pada semua blok
    public int countAlphabetsInBlocks(List<char[][]> blocks) {
        int total = 0;

        for (char[][] block : blocks) {
            total += countAlphabetsInBlock(block);
        }

        return total;
    }

    // Menghitung jumlah alfabet pada satu blok
    private int countAlphabetsInBlock(char[][] block) {
        int cnt = 0;

        for (int i = 0; i < block.length; i++) {
            for (int j = 0; j < block[i].length; j++) {
                if (block[i][j] != '.' && block[i][j] != ' ') {
                    cnt++;
                }
            }
        }

        return cnt;
    }

    // Menampilkan board saat ini untuk debugging
    public void printWhiteBoard() {
        System.out.println("===== Board =====");
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < M; j++) {
                System.out.print(board[i][j]);
            }
            System.out.println();
        }
        System.out.println("=================\n");
    }

    // Menampilkan semua variasi blok untuk debugging
    public void printVariations(List<char[][]> variations) {
        int index = 1;
```

```
        for (char[][] var : variations) {
            System.out.println("----- Variation #" + index++ + "
-----");
            for (char[] row : var) {
                for (char ch : row) {
                    System.out.print(ch);
                }
                System.out.println();
            }
            System.out.println("----------------------\n");
        }
    }
}
```

BAB IV

EKSPERIMEN

Uji Coba akan dilakukan menggunakan file .txt yang memiliki struktur berikut

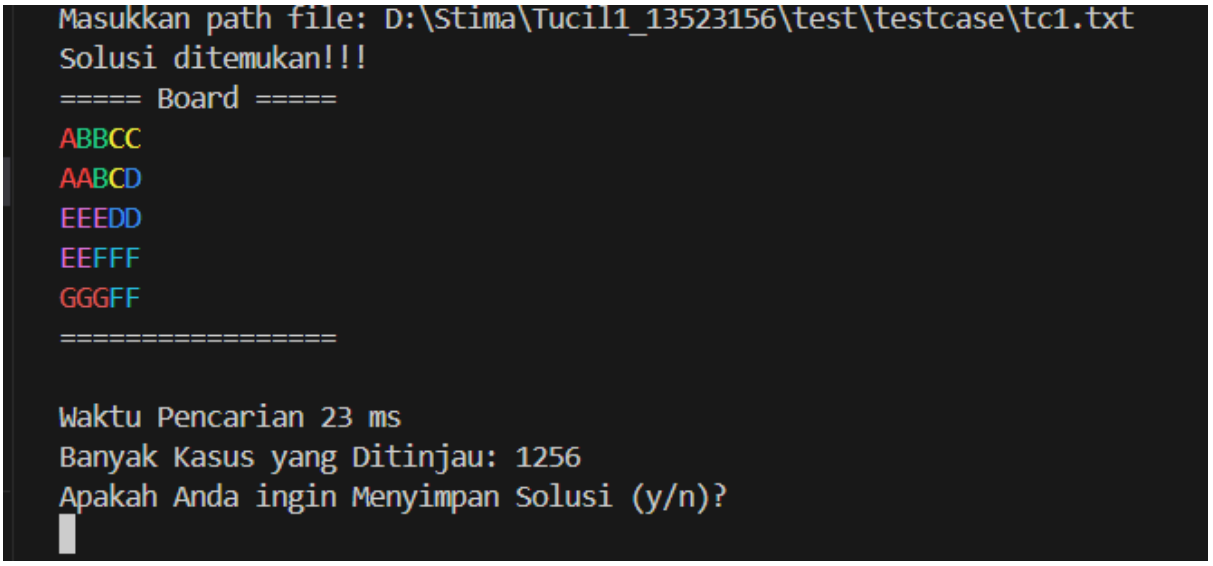
```
N M P
S
puzzle_1_shape
puzzle_2_shape
…
puzzle_P_shape
```

Dengan N dan M sebagai ukuran dari board, P sebagai jumlah blok puzzle, S sebagai jenis kasus yang hanya bernilai DEFAULT, dan jumlah blok yang direpresentasikan oleh P huruf dengan masukan kapital dibawahnya. Output akan ditampilkan dari CLI dan GUI serta Image dengan ekstensi .jpg

- **Test Case 1**

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

Output yang dihasilkan:

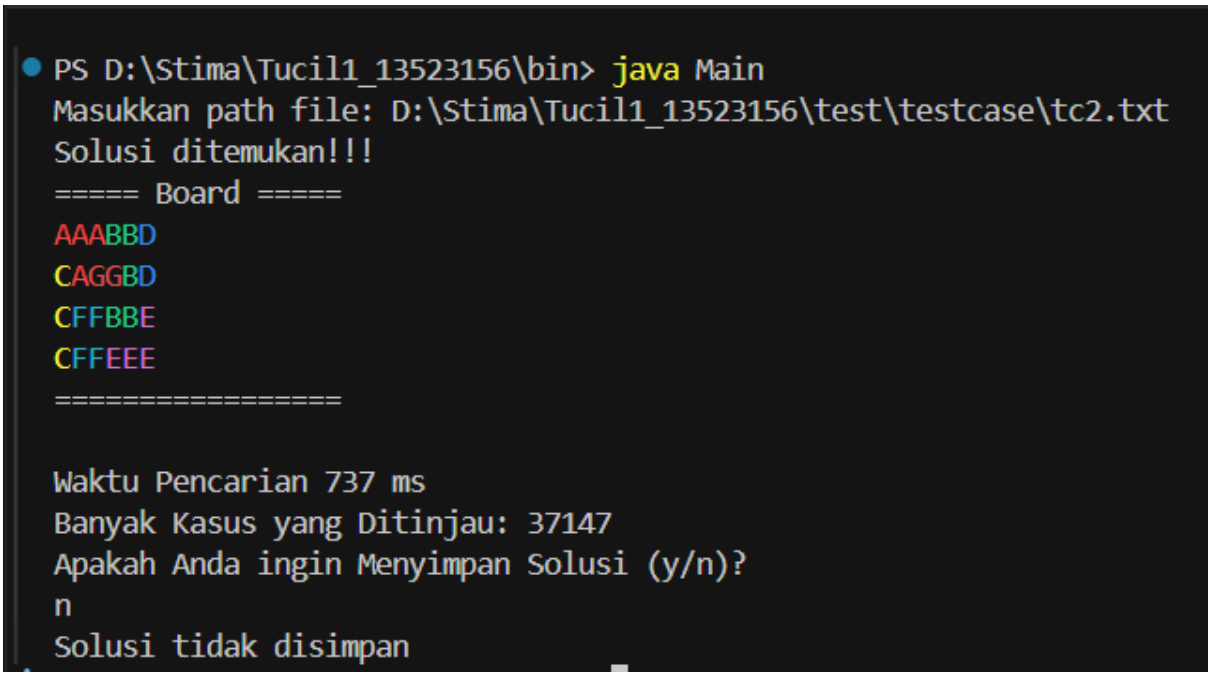


Gambar 4.1 Output Test Case 1 Menggunakan CLI

- **Test Case 2**

```
4 6 7
DEFAULT
AAA
 A
B B
BBB
CCC
DD
EEE
  E
FF
FF
GG
```

Output yang dihasilkan:



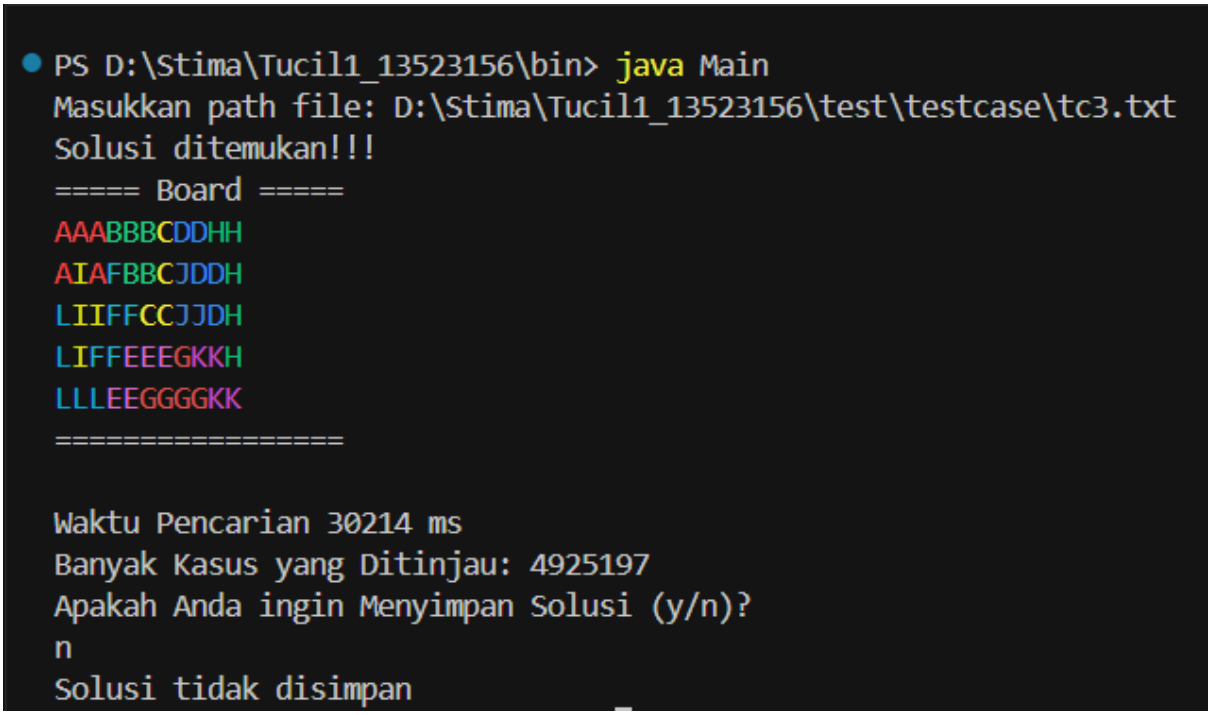
Gambar 4.2 Output Test Case 2 Menggunakan CLI

- **Test Case 3**

```
5 11 12
DEFAULT
AAA
A A
BBB
 BB
CC
C
C
 DD
DD
D
 E
EE
E
E
 F
FF
 FF
```

```
 G
GGGG
H
HHHH
III
 I
 J
JJ
 K
KK
K
  L
  L
LLL
```

Output yang dihasilkan:



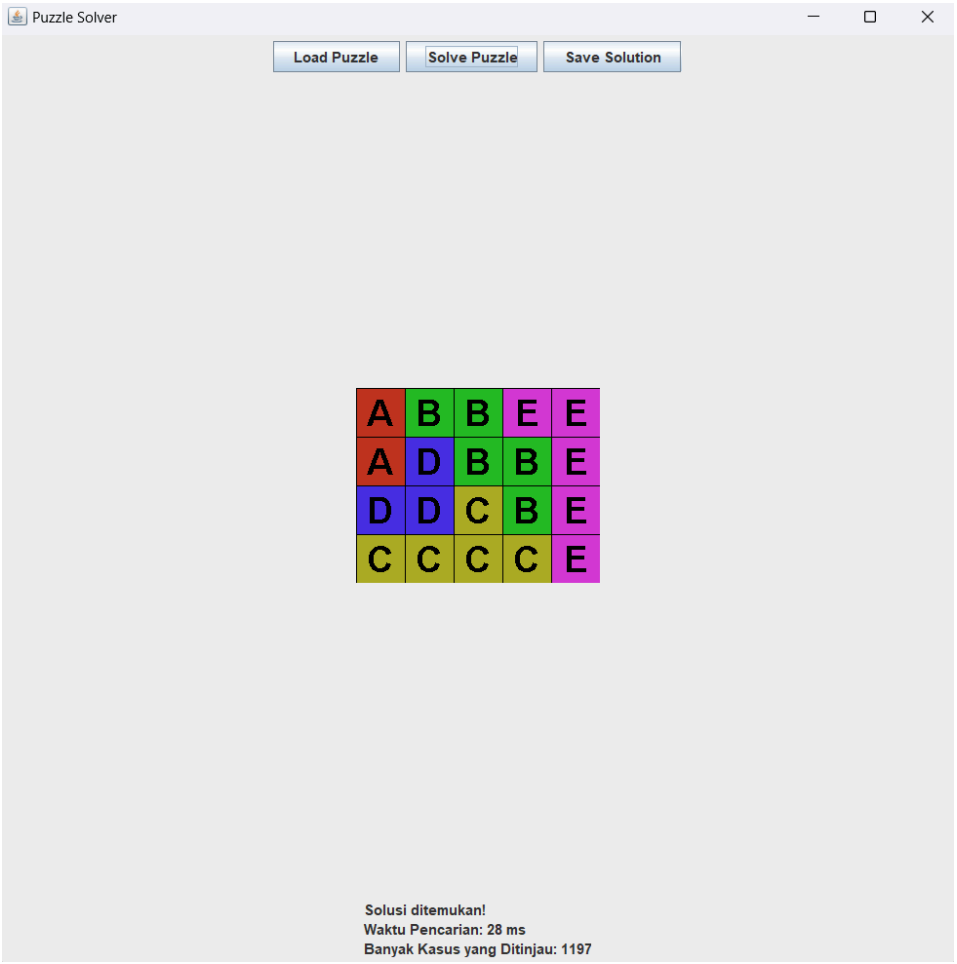
Gambar 4.3 Output Test Case 3 Menggunakan CLI

- **Test Case 4**

```
4 5 5
```

```
DEFAULT
AA
 BB
BB
B
C
CC
C
C
DD
D
EEEE
   E
```

Output yang dihasilkan:

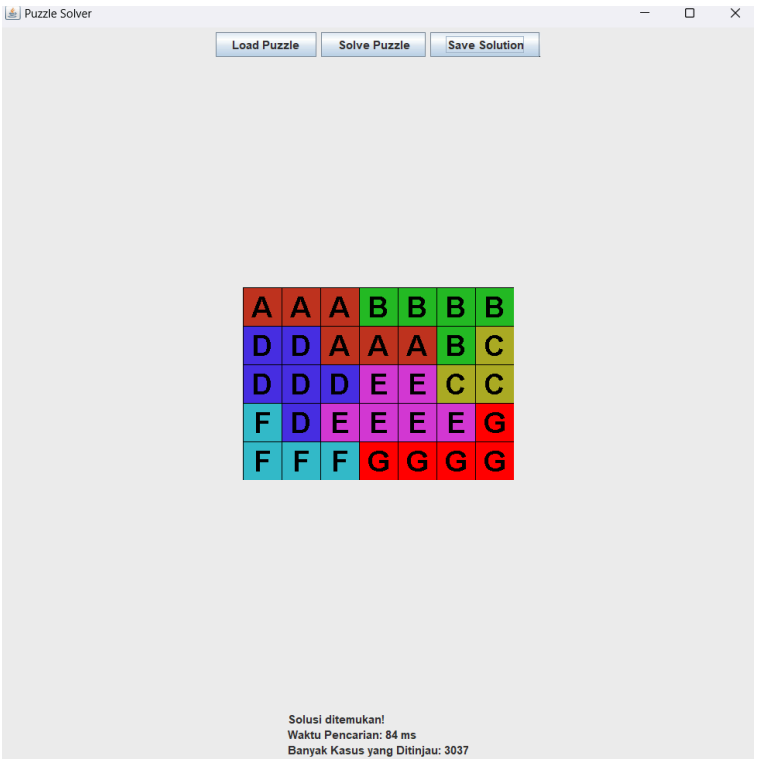


Gambar 4.4 Output Test Case 4 Menggunakan GUI

- **Test Case 5**

```
5 7 7
DEFAULT
AAA
  AAA
BBBB
  B
CC
 C
DD
DDD
 D
 EE
EEEE
FFF
F
G
GGGG
```

Output yang dihasilkan:

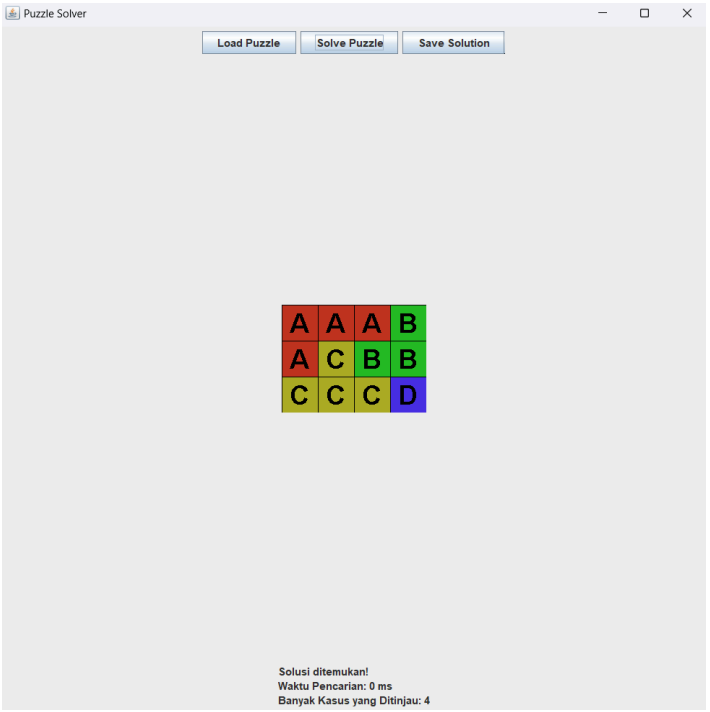


Gambar 4.5 Output Test Case 5 Menggunakan GUI

- **Test Case 6**

```
3 4 4
DEFAULT
AAA
A
BB
 B
C
CC
C
D
```

Output yang dihasilkan:

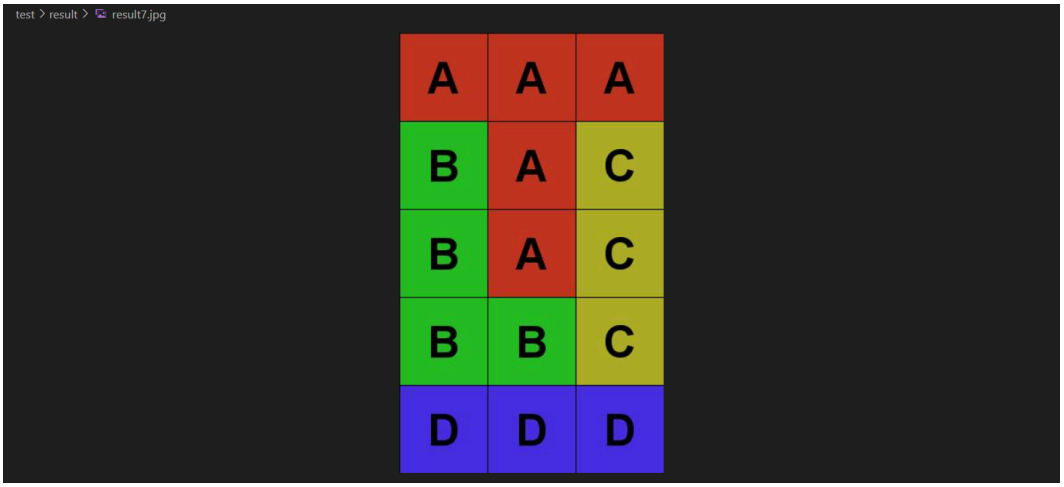


Gambar 4.6 Output Test Case 6 Menggunakan GUI

- **Test Case 7**

```
5 3 4
DEFAULT
AAA
 A
 A
B
B
BB
CCC
DDD
```

Output yang dihasilkan:

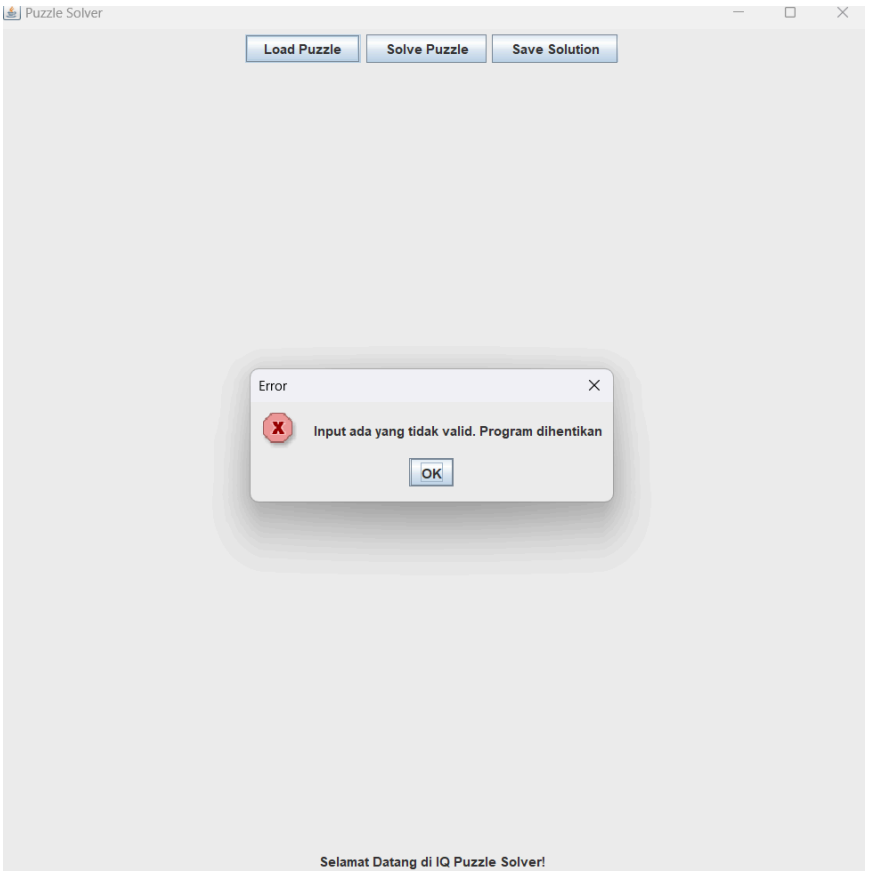


Gambar 4.7 Output Test Case 7 Menggunakan Format JPG

- **Test Case 8**

```
4 9 6
DEFAULT
AA
BB
CC
DDD
EEE
```

Output yang dihasilkan:

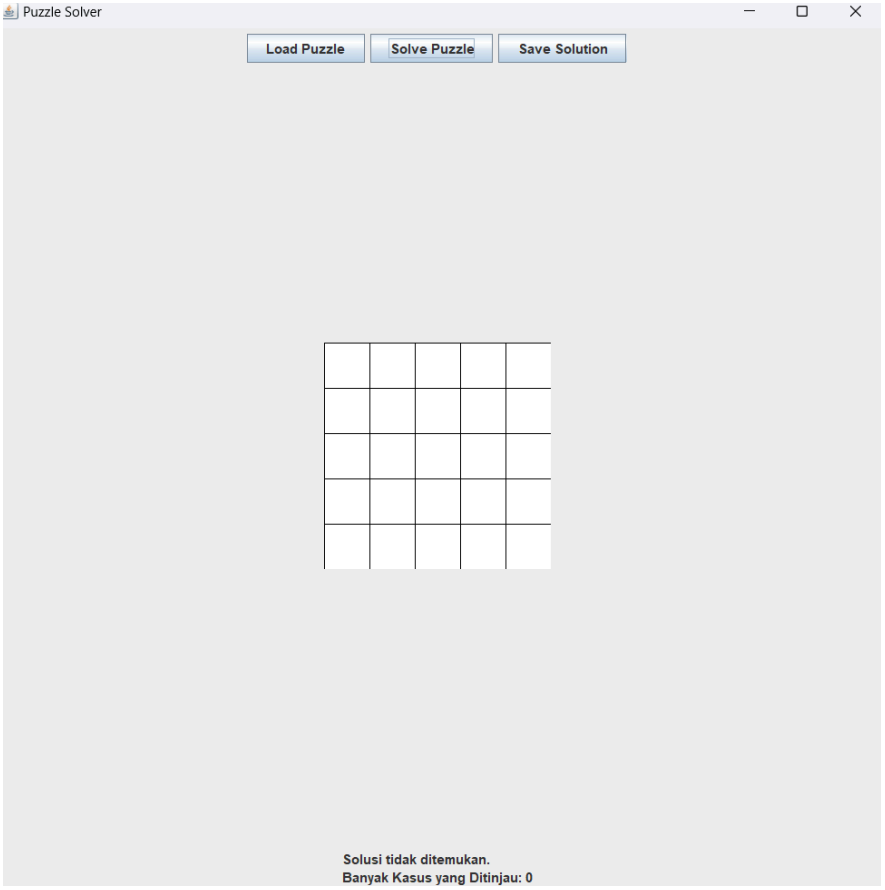


Gambar 4.8 Output Test Case 8 Menggunakan GUI. Kasus Di Mana Jumlah Blok Tidak Sama Dengan P

- **Test Case 9**

```
5 5 1
DEFAULT
AAAAA
AAAAAA
AAAA
AAAAA
AAAAA
```

Output yang dihasilkan:



Gambar 4.9 Output Test Case 9 Menggunakan GUI. Kasus Di Mana Puzzle Tidak Terselesaikan

LAMPIRAN

- Pranala Repository:
  https://github.com/Inforable/Tucil1_13523156

- Tabel Pengecekan Fitur

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