LAPORAN TUGAS KECIL I

IF2211 STRATEGI ALGORITMA

Penyelesaian IQ Puzzler Pro dengan Algoritma Brute Force

Disusun oleh:

Muhammad Zahran Ramadhan Ardiana 13523104

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

2025

BAGIAN I
ALGORITMA BRUTE FORCE

Algoritma Brute Force adalah metode penyelesaian masalah yang menguji setiap kemungkinan hingga menemukan solusi yang tepat, tanpa menggunakan pendekatan optimasi atau aturan-aturan tertentu. Pengujian setiap kemungkinan menyebabkan algoritma tersebut mempunya kompleksitas yang tinggi sehingga membutuhkan kekuatan komputasi yang memadai untuk memacahkan masalah tersebut, sehingga algoritma ini kurang cocok untuk dimensi percobaan yang besar.

Pada permasalahan IQ Puzzler Pro, algoritma brute Brute Force dilakukan dengan cara menguji setiap susunan blok yang ditempatkan pada board yang tersedia. Detail algoritma yang diterapkan adalah sebagai berikut:

1. Misalkan setiap blok direpresentasikan sebagai array yang terdiri dari koordinat {baris, kolom} dengan titik (0,0) di pojok kiri atas. Sebagai contoh, blok berbentuk "+" direpresentasikan oleh {0,0}, {1,-1}, {1,0}, {1,1}, dan {2,0}.
2. Membentuk setiap kombinasi transformasi dari blok dengan melakukan rotasi dan flip atau keduanya.
3. Pemasangan blok dimulai dengan mencari sel kosong pada board, dimulai dari baris paling atas dan iterasi per kolom.
4. Untuk setiap sel kosong yang ditemukan, dilakukan pengecekan apakah salah satu transformasi blok dapat ditempatkan tanpa bertabrakan dengan blok lain.
5. Pemasangan blok dilakukan secara rekursif dengan basis bahwa jika tidak ada sel kosong yang tersisa pada board, maka solusi telah ditemukan. Jika suatu blok tidak dapat dipasang, algoritma mengembalikan *false* dan melakukan backtracking dengan menghapus blok sebelumnya, lalu mencoba transformasi lain atau blok yang berbeda.
6. Blok yang telah dipasang dihapus dari mapping untuk mencegah pemasangan blok yang sama dengan transformasi berbeda.

Iterasi atau jumlah langkah dihitung setiap kali dilakukan percobaan pemasangan blok; misalnya, pada board 3×3 dengan 9 blok, setiap langkah pemasangan dihitung sebagai satu iterasi (9 langkah). Kompleksitas waktu terburuk adalah O(n!) di mana n merupakan jumlah blok

yang tersedia, karena algoritma harus menguji semua urutan pemasangan blok sebelum menemukan solusi yang valid.

BAGIAN II
SOURCE PROGRAM

Program penyelesaian Puzzle IQ Pro ditulis dalam bahasa java. Struktur dari program ini dibagi menjadi 6 file sebagai berikut:

1. PuzzleSolverGUI.java
2. Puzzle.java
3. Solver.java
4. Input.java
5. InputValidator.java
6. Debug.java

2.1. PuzzleSolverGUI.java

```java
public class PuzzleSolverGUI extends JFrame {
    private Puzzle puzzle;
    private static File selectedFile;
    private JLabel infoLabel;
    private JPanel boardPanel;
    private JLabel statusLabel;

    public PuzzleSolverGUI() {
        setTitle(title:"Puzzle Solver");
        setSize(width:800, height:600);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new BorderLayout());

        // Panel atas untuk input file
        JPanel topPanel = new JPanel();
        topPanel.setLayout(new FlowLayout());

        JButton loadButton = new JButton(text:"Pilih File");
        loadButton.addActionListener(e -> chooseFile());
        topPanel.add(loadButton);

        JButton saveButton = new JButton(text:"Simpan Solusi");
        saveButton.addActionListener(e -> saveSolution());
        topPanel.add(saveButton);

        // Status Label
        statusLabel = new JLabel(text:"Silakan pilih file puzzle.");
        topPanel.add(statusLabel);
```

```
        // Panel board
        boardPanel = new JPanel();
        boardPanel.setLayout(new GridLayout(rows:1, cols:1));
        boardPanel.add(new JLabel(text:"Belum ada puzzle yang dimuat", SwingConstants.CENTER));

        add(topPanel, BorderLayout.NORTH);
        add(boardPanel, BorderLayout.CENTER);

        // Drag and drop file
        setTransferHandler(new TransferHandler() {
            @Override
            public boolean canImport(TransferSupport support) {
                return support.isDataFlavorSupported(DataFlavor.javaFileListFlavor);
            }

            @Override
            public boolean importData(TransferSupport support) {
                try {
                    List<File> files = (List<File>) support.getTransferable().getTransferData(DataFlavor.ja
                    if (!files.isEmpty()) {
                        selectedFile = files.get(index:0);
                        loadPuzzle();
                    }
                } catch (Exception e) {
                    e.printStackTrace();
                    return false;
                }
                return true;
            }
        });

        setVisible(b:true);
    }

    private void chooseFile() {
        JFileChooser fileChooser = new JFileChooser();
        fileChooser.setDialogTitle(dialogTitle:"Pilih File Puzzle");
        int userSelection = fileChooser.showOpenDialog(this);

        if (userSelection == JFileChooser.APPROVE_OPTION) {
            selectedFile = fileChooser.getSelectedFile();
            loadPuzzle();
        }
    }

    private void loadPuzzle() {
        if (selectedFile == null) {
            return;
        }
        long startTime = System.nanoTime();

        puzzle = new Puzzle();
        Input.inputPuzzleData(puzzle, selectedFile);

        if (puzzle.board == null) {
            statusLabel.setText(text:"Input tidak valdi!");
            return;
        }

        boolean solved = Solver.solvePuzzle(puzzle);
```

```
        boolean solved = solver.solvePuzzle(puzzle);
        long endTime = System.nanoTime();
        puzzle.time = (endTime - startTime) / 1_000_000;

        if (!solved) {
            statusLabel.setText("Tidak ada solusi! (" + puzzle.time + " ms, " + puzzle.countStep +  " langkah)");
        } else {
            statusLabel.setText("Solusi ditemukan dalam " + puzzle.time + " ms dan " + puzzle.countStep + " langkah.");
        }

        updateBoard();
    }

    private void updateBoard() {
        boardPanel.removeAll();
        boardPanel.setLayout(new GridLayout(puzzle.rows, puzzle.cols));

        for (int i = 0; i < puzzle.rows; i++) {
            for (int j = 0; j < puzzle.cols; j++) {
                JLabel cell = new JLabel(" " + puzzle.board[i][j], SwingConstants.CENTER);
                cell.setOpaque(isOpaque:true);
                cell.setBorder(BorderFactory.createLineBorder(Color.BLACK));
                cell.setBackground(getColor(puzzle.board[i][j]));
                boardPanel.add(cell);
            }
        }
        boardPanel.revalidate();
        boardPanel.repaint();
    }

    private Color getColor(char block) {
        switch (block) {
            case 'A': return Color.RED;
            case 'B': return Color.BLUE;
            case 'C': return Color.GREEN;
            case 'D': return Color.YELLOW;
            case 'E': return Color.ORANGE;
            case 'F': return Color.PINK;
            case 'G': return Color.MAGENTA;
            case 'H': return Color.CYAN;
            case 'I': return Color.LIGHT_GRAY;
            case 'J': return new Color(r:255, g:140, b:0);
            case 'K': return new Color(r:75, g:0, b:130);
            case 'L': return new Color(r:139, g:69, b:19);
            case 'M': return new Color(r:210, g:105, b:30);
            case 'N': return new Color(r:112, g:128, b:144);
            case 'O': return new Color(r:255, g:215, b:0);
            case 'P': return new Color(r:0, g:128, b:128);
            case 'Q': return new Color(r:255, g:99, b:71);
            case 'R': return new Color(r:128, g:0, b:0);
            case 'S': return new Color(r:0, g:255, b:127);
            case 'T': return new Color(r:70, g:130, b:180);
            case 'U': return new Color(r:218, g:112, b:214);
            case 'V': return new Color(r:46, g:139, b:87);
            case 'W': return new Color(r:255, g:20, b:147);
            case 'X': return new Color(r:199, g:21, b:133);
            case 'Y': return new Color(r:189, g:183, b:107);
            case 'Z': return new Color(r:255, g:228, b:181);
            default: return Color.WHITE;
        }
    }
```

```java
    private void saveSolution() {
        if (selectedFile == null || puzzle == null || puzzle.board == null) {
            statusLabel.setText(text:"Belum ada solusi mas, pilih file dulu!");
            return;
        }

        File directory = new File(pathname:"solutions/");
        if (!directory.exists()) {
            directory.mkdirs();
        }

        String fileName = selectedFile.getName().replace(target:".txt", replacement:"_solution.txt");

        try (FileWriter writer = new FileWriter("test/solutions/" + fileName)) {
            for (char[] row : puzzle.board) {
                writer.write(new String(row) + "\n");
            }
            statusLabel.setText("Solusi disimpan: " + fileName);
        } catch (IOException e) {
            statusLabel.setText(text:"Gagal menyimpan solusi!");
        }
    }

    Run | Debug | Run main | Debug main
    public static void main(String[] args) {
        new PuzzleSolverGUI();
    }
}
```

2.2. Puzzle.java

```java
public class Puzzle {
    public int rows, cols;
    public int countStep = 0;
    public long time = 0;
    public char[][] board;
    public Map<Character, List<List<int[]>>> blockMap = new LinkedHashMap<>();

    public static List<int[]> rotateBlock(List<int[]> block, int angle) {
        int maxRow = 0, maxCol = 0;
        for (int[] pos : block) {
            maxRow = Math.max(maxRow, pos[0]);
            maxCol = Math.max(maxCol, pos[1]);
        }

        List<int[]> rotatedBlock = new ArrayList<>();
        for (int[] pos : block) {
            int newRow = 0, newCol = 0;
            switch (angle) {
                case 90 -> {
                    newRow = pos[1];
                    newCol = maxRow - pos[0];
                }
```

```
            case 180 -> {
                newRow = maxRow - pos[0];
                newCol = maxCol - pos[1];
            }
            case 270 -> {
                newRow = maxCol - pos[1];
                newCol = pos[0];
            }
            default -> throw new IllegalArgumentException(s:"Sudut rotasi ngaco nih");
        }
        rotatedBlock.add(new int[]{newRow, newCol});
    }
    rotatedBlock = normalize(rotatedBlock);
    return rotatedBlock;
}

public static List<int[]> flipBlock(List<int[]> block) {
    int maxRow = 0, maxCol = 0;
    for (int[] pos : block) {
        maxRow = Math.max(maxRow, pos[0]);
        maxCol = Math.max(maxCol, pos[1]);
    }

    List<int[]> flippedBlock = new ArrayList<>();
    for (int[] pos : block) {   //flip horizontal
        int newRow = pos[0];
        int newCol = maxCol - pos[1];
        flippedBlock.add(new int[]{newRow, newCol});
    }
    flippedBlock = normalize(flippedBlock);
    return flippedBlock;
}

public static List<int[]> normalize(List<int[]> positions) {
    int minRow = Integer.MAX_VALUE, minCol = Integer.MAX_VALUE;
    for (int[] pos : positions) {
        minRow = Math.min(minRow, pos[0]);
    }
    for (int[] pos : positions) {
        if (pos[0] == minRow)
            minCol = Math.min(minCol, pos[1]);
    }

    List<int[]> normalized = new ArrayList<>();
    for (int[] pos : positions) {
        normalized.add(new int[]{pos[0] - minRow, pos[1] - minCol});
    }

    return normalized;
}
```

```java
    public static List<List<int[]>> generateTransformations(List<int[]> baseShape) {
        List<List<int[]>> transformations = new ArrayList<>();
        Set<Set<String>> uniqueShapes = new HashSet<>();
        transformations.add(normalize(baseShape)); // Bentuk awal
        Set<String> baseSet = toSet(normalize(baseShape));
        uniqueShapes.add(baseSet);

        List<List<int[]>> candidates = Arrays.asList(
            rotateBlock(normalize(baseShape), angle:90),
            rotateBlock(normalize(baseShape), angle:180),
            rotateBlock(normalize(baseShape), angle:270),
            flipBlock(normalize(baseShape)),
            rotateBlock(flipBlock(normalize(baseShape)), angle:90),
            rotateBlock(flipBlock(normalize(baseShape)), angle:180),
            rotateBlock(flipBlock(normalize(baseShape)), angle:270)
        );

        for (List<int[]> candidate : candidates) {
            Set<String> candidateSet = toSet(candidate);
            if (!uniqueShapes.contains(candidateSet)) {
                uniqueShapes.add(candidateSet);
                transformations.add(candidate);
            }
        }

        return transformations;
    }

    private static Set<String> toSet(List<int[]> shape) {
        Set<String> set = new HashSet<>();
        for (int[] point : shape) {
            set.add(point[0] + "," + point[1]);
        }
        return set;
    }
}
```

2.3. Solver.java

```java
public class Solver {
    public static int[] findNextEmpty(char[][] board) {
        int[] pos = new int[2];
        for (int i = 0; i < board.length; i++) {
            for (int j = 0; j < board[0].length; j++) {
                if (board[i][j] == ' ') {
                    pos[0] = i;
                    pos[1] = j;
                    return pos;
                }
            }
        }
        return null;
    }

    public static boolean isFit(char[][] board, char blockChar, List<int[]> block, int row, int col) {
        for (int[] pos : block) {
            int newRow = row + pos[0];
            int newCol = col + pos[1];
            if (newRow < 0 || newRow >= board.length || newCol < 0 || newCol >= board[0].length) {
                return false;
            }
            if (board[newRow][newCol] != ' ') {
                return false;
            }
        }
        return true;
    }

    public static boolean solvePuzzle(Puzzle puzzle) {
        List<Character> blockChars = new ArrayList<>(puzzle.blockMap.keySet());
        //printBoard(puzzle.board);

        int[] pos = findNextEmpty(puzzle.board);
        if (pos == null) {
            return true;
        }
        int row = pos[0];
        int col = pos[1];

        //ambil tiap karakter block
        for (int i = 0; i < blockChars.size(); i++) {
            char blockChar = blockChars.get(i);
            List<List<int[]>> blockVariants = puzzle.blockMap.remove(blockChar); // Hapus blok sebelum rekursi
            if (blockVariants == null) continue;

            //ambil tiap rotasi nya
            for (int j = 0; j < blockVariants.size(); j++) {
                List<int[]> block = blockVariants.get(j);

                puzzle.countStep++;
                if (isFit(puzzle.board, blockChar, block, row, col)) {
                    //System.out.println(MainTest.countStep);
                    for (int[] posBlock : block) {
                        int newRow = row + posBlock[0];
                        int newCol = col + posBlock[1];
                        puzzle.board[newRow][newCol] = blockChar;
                    }
                    if (solvePuzzle(puzzle)) {
                        return true;
                    }
```

```
                // Backtrack (gagal pasang blok jadi dikosongin lagi)
                for (int[] posBlock : block) {
                    int newRow = row + posBlock[0];
                    int newCol = col + posBlock[1];
                    puzzle.board[newRow][newCol] = ' ';
                }
            }
        }
        // Kembalikan blok ke list jika gagal dipasang
        puzzle.blockMap.put(blockChar, blockVariants);

    }
    return  false;
}

public static void printBoard(char[][] board) {
    for (char[] row : board) {
        for (char c : row) {
            System.out.print(c);
        }
        System.out.println();
    }
}
}
```

2.4. Input.java

```
public class Input {
    public static void inputPuzzleData(Puzzle puzzle, File file){
        boolean valid = inputValidator.isValidInput(file);
        if(!valid){
            return;
        }

        try {
            Scanner fileScanner = new Scanner(file);
            int m = fileScanner.nextInt();
            int n = fileScanner.nextInt();
            int pieces = fileScanner.nextInt();
            int objCount = 0;
            int row = 0;

            fileScanner.nextLine();
            String mode = fileScanner.nextLine();
            char[][] board = new char[m][n];

            //buat boardnya ygy
            if ("DEFAULT".equals(mode)) {
                for (int i = 0; i < m; i++) {
                    for (int j = 0; j < n; j++) {
                        board[i][j] = ' ';
                    }
                }
```

```
            }
        }else if ("CUSTOM".equals(mode)) {
            for (int i = 0; i < m; i++) {
                String line = fileScanner.nextLine();
                for (int j = 0; j < n; j++) {
                    if (line.charAt(j) == 'X') {
                        board[i][j] = ' ';
                    } else {
                        board[i][j] = '.';
                    }
                }
            }
        }

        //buat block2nya dalam bentuk koordinat
        Map<Character, List<int[]>> blockList = new LinkedHashMap<>();
        while (fileScanner.hasNextLine() && objCount < pieces) {
            String line = fileScanner.nextLine();
            if (line.isEmpty()) break;

            for(int col=0; col<line.length(); col++){
                char c = line.charAt(col);
                if(c != ' '){
                    if(!blockList.containsKey(c)){
                        blockList.put(c, new ArrayList<>());   //masukkan key baru
                        row = 0;                               //reset row
                        //objCount++;
                    }
                    blockList.get(c).add(new int[]{row, col});
                }
            }
            row++;
        }

        //nah kalau ini tiap block dibuat variasi transformasinya (rotasi,flip atau keduanya)
        Map<Character, List<List<int[]>>> blockMap = new LinkedHashMap<>();
        for (Map.Entry<Character, List<int[]>> entry : blockList.entrySet()) {
            List<List<int[]>> transformations = Puzzle.generateTransformations(entry.getValue());
            blockMap.put(entry.getKey(), transformations);
        }

        puzzle.rows = m;
        puzzle.cols = n;
        puzzle.board = board;
        puzzle.blockMap = blockMap;
    }

    catch (FileNotFoundException e) {
        JOptionPane.showMessageDialog(parentComponent:null, message:"File tidak ditemukan.", title
        System.out.println(x:"File tidak ditemukan. Pastikan path sudah benar.");
    }
  }
}
```

2.5. InputValidator.java

```java
public class InputValidator {
    public static boolean isValidInput(File file){
        try {
            Scanner fileScanner = new Scanner(file);

            //cek input 3 angka pertama
            if (!fileScanner.hasNextInt()) {
                System.out.println(x:"Error: Input pertama bukan angka!");
                JOptionPane.showMessageDialog(parentComponent:null, message:"Error: Input pertama bukan angka!

                return false;
            }
            int m = fileScanner.nextInt();

            if (!fileScanner.hasNextInt()) {
                System.out.println(x:"Error: Input kedua bukan angka!");
                JOptionPane.showMessageDialog(parentComponent:null, message:"Error: Input kedua bukan angka!",
                return false;
            }
            int n = fileScanner.nextInt();

            if (!fileScanner.hasNextInt()) {
                System.out.println(x:"Error: Input ketiga bukan angka!");
                JOptionPane.showMessageDialog(parentComponent:null, message:"Error: Input ketiga bukan angka!"
                return false;
            }
            int blocks = fileScanner.nextInt();

            fileScanner.nextLine();
            String mode = fileScanner.nextLine();
            //System.out.println(mode);


            if (!"DEFAULT".equals(mode) && !"CUSTOM".equals(mode)) {
                System.out.println(x:"Error: Mode tidak sesuai!");
                JOptionPane.showMessageDialog(parentComponent:null, message:"Error: Mode tidak sesuai!", title
                return false;
            }

            int slotCount = m*n;
            if ("CUSTOM".equals(mode)) {
                int boardRow = 0;
                slotCount = 0;
                while (fileScanner.hasNextLine() && boardRow < m) {
                    String line = fileScanner.nextLine();
                    if (line.length() != n) {
                        System.out.println(x:"Error: Panjang board tidak sesuai!");
                        JOptionPane.showMessageDialog(parentComponent:null, message:"Error: Panjang board tida
                        return false;
                    }
                    for (int i = 0; i < line.length(); i++) {
                        if (line.charAt(i) == 'X') {
                            slotCount++;
                        }
                    }
                    boardRow++;
                }
            }
```

```java
//cek input block
int row = 0;
int pieces = 0;
int blockCount = 0;
Map<Character, List<int[]>> blockList = new LinkedHashMap<>();
while (fileScanner.hasNextLine()) {
    String line = fileScanner.nextLine();

    if (line.isEmpty()) break;

    Set<Character> charSet = new HashSet<>();
    for (int col = 0; col < line.length(); col++) {
        char c = line.charAt(col);

        if (c != ' ') {
            //cek jumlah pieces/karakter
            pieces++;

            //1 baris ada huruf berbeda
            charSet.add(c);
            if (charSet.size() > 1) {
                System.out.println(x:"1 baris ada huruf berbeda");
                JOptionPane.showMessageDialog(parentComponent:null, message:"1 baris ada huruf berbe
                return false;
            }

            //cek posisi block apakah ada yang tidak menyatu atau terpisah
            if(!blockList.containsKey(c)){
                blockList.put(c, new ArrayList<>());
                blockCount++;
                //System.out.println("Block baru: " + c + " " + blockCount);
            }
            blockList.get(c).add(new int[]{row, col});
        }
    }
    row++;
}


if (!isConnectedBlock(blockList)) {
    System.out.println(x:"Block tidak terhubung");
    JOptionPane.showMessageDialog(parentComponent:null, message:"Block tidak terhubung", title:"Hasi
    return false;
}

if ((pieces != slotCount)) {
    System.out.println(x:"Jumlah pieces tidak sesuai ya");
    JOptionPane.showMessageDialog(parentComponent:null, message:"Jumlah pieces tidak sesuai", title:
    return false;
}

if (isLargerThanBoard(blockList, m, n)) {
    System.out.println(x:"Block lebih besar dari board");
    JOptionPane.showMessageDialog(parentComponent:null, message:"Block lebih besar dari board", titl
    return false;
}

if (blockCount != blocks) {
    System.out.println(x:"Jumlah block tidak sesuai");
    return false;
```

```
            return true;
        } catch (FileNotFoundException e) {
            return false;
        }
    }

    public static boolean isConnectedBlock(Map<Character, List<int[]>> blockList){
        for (Map.Entry<Character, List<int[]>> entry : blockList.entrySet()) {
            List<int[]> block = entry.getValue();
            int[][] directions = {{0, 1}, {0, -1}, {1, 0}, {-1, 0}};
            Queue<int[]> queue = new LinkedList<>();

            Set<String> visited = new HashSet<>();
            queue.add(block.get(index:0));
            visited.add(block.get(index:0)[0] + "," + block.get(index:0)[1]);

            while (!queue.isEmpty()) {
                int[] current = queue.poll();
                for (int[] dir : directions) {
                    int newRow = current[0] + dir[0];
                    int newCol = current[1] + dir[1];

                    String newPosStr = newRow + "," + newCol;
                    if (block.stream().anyMatch(b -> b[0] == newRow && b[1] == newCol) &&
                        !visited.contains(newPosStr)) {
                        queue.add(new int[]{newRow, newCol});
                        visited.add(newPosStr);
                    }
                }
            }

            if (visited.size() != block.size()) {
                return false;
            }
        }
        return true;
    }

    public static boolean isLargerThanBoard(Map<Character, List<int[]>> blockList, int m, int n){
        for (Map.Entry<Character, List<int[]>> entry : blockList.entrySet()) {
            List<int[]> block = entry.getValue();
            int maxRow = 0, maxCol = 0, minRow = 100, minCol = 100;
            for (int[] pos : block) {
                maxRow = Math.max(maxRow, pos[0]);
                minRow = Math.min(minRow, pos[0]);
                maxCol = Math.max(maxCol, pos[1]);
                minCol = Math.min(minCol, pos[1]);
            }

            int length1 = maxRow - minRow + 1;
            int length2 = maxCol - minCol + 1;
            if (length1 > m || length2 > n) {
                return true;
            }
        }
        return false;
    }
}
```

2.6. Debug.java

```java
public class Debug {
    public static void printBlockVariant(Map<Character, List<List<int[]>>> shapeMap){
        for (Map.Entry<Character, List<List<int[]>>> entry : shapeMap.entrySet()) {
            System.out.println("Huruf '" + entry.getKey() + "' memiliki bentuk:");
            int index = 1;
            for (List<int[]> shape : entry.getValue()) {
                System.out.println("Bentuk " + index + ": " + coordinatesToString(shape));
                index++;
            }
        }
    }

    private static String coordinatesToString(List<int[]> coordinates) {
        StringBuilder sb = new StringBuilder();
        sb.append(str:"[");
        for (int[] coord : coordinates) {
            sb.append(str:"(").append(coord[0]).append(str:",").append(coord[1]).append(str:"), ");
        }
        if (!coordinates.isEmpty()) sb.setLength(sb.length() - 2);
        sb.append(str:"]");
        return sb.toString();
    }

    //Test piece
    public static void printBlockList(Map<Character, List<int[]>> puzzle){
        for (Map.Entry<Character, List<int[]>> entry : puzzle.entrySet()) {
            List<int[]> positions = entry.getValue();
            System.out.print(entry.getKey() + ": ");
            for (int[] pos : positions) {
                System.out.print(Arrays.toString(pos) + " ");
            }
            System.out.println();
        }
    }
    public static void printBlock(List<int[]> block){
        for (int[] pos : block) {
            System.out.println(Arrays.toString(pos));
        }
    }
}
```

BAGIAN III

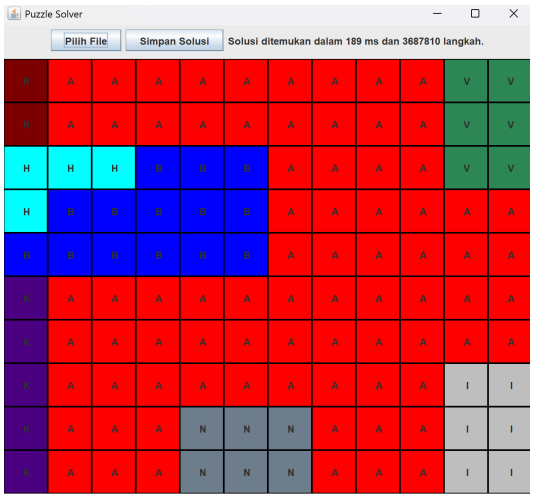
SCREENSHOT HASIL TEST

File Input test2.txt:

```
6 6 7
DEFAULT
AAA
 A
BB
BB
 B
CC
CC
CC
 C
DDD
EEEE
 E
GG
GGG
 G
F  F
FFFF
```

File input amogus.txt:

```
10 12 8
DEFAULT
AAAAAAAAA
AAAAAAAAA
AAAA
AAAAAA
AAAAAA
AAAAAAAAAAA
AAAAAAAAAAA
AAAAAAAAA
AAA   AAA
AAA   AAA
HHH
H
BBB
BBBBB
BBBBBB
RR
KKKKK
VVV
VVV
III
III
NNN
NNN
```



File input test6.txt:

```
5 7 5
CUSTOM
...X...
.XXXXX.
XXXXXXX
.XXXXX.
...X...
A
AAA
B
BB
CCCC
C
D
EE
EEE
E
```

LINK REPOSITORY

CHECK LIST

Poin	Ya	Tidak
Program berhasil dikompilasi tanpa kesalahan	✓	
Program berhasil dijalankan	✓	
Solusi yang diberikan program benar dan mematuhi aturan permainan	✓	
Program dapat membaca masukan berkas .txt serta menyimpan solusi dalam berkas .txt	✓	
Program memiliki Graphical User Interface (GUI)	✓	
Program dapat menyimpan solusi dalam bentuk file gambar		✓
Program dapat menyelesaikan kasus konfigurasi custom	✓	
Program dapat menyelesaikan kasus konfigurasi Piramida (3D)		✓
Program dibuat oleh saya sendiri	✓	