

LAPORAN TUGAS BESAR

IF2211 Strategi Algoritma


IQ Puzzle Zolver

Dipersiapkan oleh:

- Brian Albar Hadian 13523048

Sekolah Teknik Elektro dan Informatika - Institut Teknologi Bandung

Jl. Ganesha 10, Bandung 40132

	Sekolah Teknik Elektro dan Informatika ITB	Nomor Dokumen		Halaman
		IF2211-TK-13523048-01		39
		Revisi	0	24 Februari 2025

1 Ringkasan

Pembuatan program berbasis CLI (command-line interface) dalam bahasa Java ini berfungsi untuk menyelesaikan IQ Puzzle Solver yang dapat dijalankan dalam environment seperti UNIX dan Windows dengan menggunakan Java Runtime Environment.

Dalam laporan ini, dapat ditemui beberapa hal yang kami implementasikan dalam program berbasis bahasa C. Diantaranya:

- Spesifikasi fitur utama,
- Struktur Objek,
- Pengetesan, dan
- Pengerjaan

2 Penjelasan Tambahan Spesifikasi Tugas

2.1 *Penyelesaian IQ Solver*

Fitur Penyelesaian IQ Solver diterapkan dengan memanfaatkan algoritma brute force berdasarkan masukan pengguna berupa file.txt. File tersebut memiliki struktur dasar yang mencakup baris pertama berupa jumlah baris, kolom, dan banyak blok puzzle yang digunakan, baris kedua berupa mode dari penyelesaian IQ Solver, dan baris-baris selanjutnya yang berisi huruf alfabet dan merepresentasikan bentuk dari blok puzzle yang akan diuji. Keluaran dari fitur ini adalah memberikan gambaran akhir penyelesaian terhadap IQ Puzzle dengan menggunakan seluruh blok puzzle yang telah diberikan.

2.2 *Keluaran berupa Gambar*

Fitur keluaran berupa gambar merupakan fitur tambahan yang dapat mengembalikan hasil operasi pencarian solusi IQ Solver dalam format gambar. Gambar tersebut akan berisi warna dengan konfigurasi yang telah ditentukan sebelumnya. Keluaran ini hanya dapat dikembalikan jika puzzle memiliki solusi yang benar sehingga dapat diperoleh matriks akhir. Keluaran kemudian akan disimpan pada directory yang sama dengan format nama sesuai dengan kehendak pengguna.

3 Strategi Penyelesaian

Program yang dibuat menerima berbagai bentuk data dan menerapkannya dalam bentuk objek dengan berfokus pada pengembangan berorientasi objek. Objek yang penulis gunakan diantaranya:

3.1 *Puzzle*

Objek Puzzle dibangun atas representasi setiap objek yang diberikan oleh pengguna melalui masukan file. Objek ini dibentuk dengan memanfaatkan struktur list of list dan menerima masukan puzzle pada file extension .txt sedemikian sehingga struktur puzzle dapat disimpan dalam bentuk matriks. Objek ini digunakan untuk memanipulasi data yang berkaitan dengan puzzle, diantaranya adalah mengembalikan dan memanipulasi data jumlah baris, jumlah kolom, struktur puzzle, karakter yang melambangkan puzzle tersebut, status pencerminan horizontal dan vertikal, serta status transpose dari puzzle. Berbagai fitur ini kemudian diterapkan lebih lanjut untuk membentuk method berupa modifyPuzzle() yang berfungsi untuk melakukan pencerminan sekaligus rotasi menurut perhitungan tertentu, flipVerticalPuzzle() yang berfungsi untuk melakukan pencerminan menurut sumbu horizontal dari puzzle, flipHorizontalPuzzle() yang berfungsi untuk melakukan pencerminan menurut sumbu vertikal dari puzzle, dan transposePuzzle() yang berfungsi untuk menerapkan transpose dari puzzle yang ada.

Lokasi : src/Puzzle (.java & .class)

3.2 *PuzzleMap*

Objek PuzzleMap dibangun untuk merepresentasikan suatu matriks yang dapat diisi oleh objek puzzle. Objek ini dibentuk dengan menerima masukan jumlah baris dan kolom yang diberikan melalui *file processing* sedemikian sehingga dibentuk suatu list of list dengan kondisi kosong berisi '?' dan kondisi terisi berisi '+'. Objek ini digunakan untuk memanipulasi data yang berkaitan dengan penempatan objek Puzzle pada PuzzleMap dan status dari PuzzleMap itu sendiri, diantaranya adalah mengembaikan jumlah baris, jumlah kolom, blok puzzle yang sudah terpasang, elemen pada puzzleMap, menampilkan kondisi PuzzleMap, mengembalikan kondisi PuzzleMap seperti semula, mengembalikan jumlah blok kosong pada PuzzleMap, pencarian lokasi untuk melakukan penyimpanan terhadap objek Puzzle, pengujian objek Puzzle terhadap PuzzleMap pada posisi tertentu, pengembalian Puzzle yang belum digunakan, dan pencetakan bentuk objek Puzzle pada PuzzleMap pada koordinat tertentu.

Lokasi: src/PuzzleMap (.class & .java)

3.3 *Main*

Spesifikasi dari program utama adalah membaca file dengan format tertentu, melakukan standarisasi terhadap bentuk objek list of list sebelum dilakukan konversi menjadi objek Puzzle, dan menghitung jumlah perulangan serta waktu yang telah dilewati dalam menguji program. Beberapa fungsi tersebut diimplementasikan sebagai method objek, diantaranya adalah getCharFromCharMatrix() berfungsi memperoleh huruf yang melambangkan blok puzzle dalam

list of list, organizeList berfungsi melakukan restrukturisasi terhadap spasi dan bentuk puzzle yang belum seragam, standardizePuzzle berfungsi melakukan konversi list of list menjadi Puzzle, dan factorial berfungsi membentuk fungsi matematika faktorial untuk memberikan batas atas terhadap jumlah percobaan.

Lokasi: Main (.class & .java)

4 Strategi Penyelesaian

4.1 Konsep Dasar

Strategi penyelesaian yang digunakan adalah strategi algoritma *brute force*. Strategi ini berfokus pada repetisi dan memanfaatkan permutasi dari setiap urutan yang ada untuk menyelesaikan persoalan. Strategi ini cocok digunakan ketika penyelesaian suatu persoalan dapat didekati dengan percobaan berulang kali, baik dengan mengurut langkah penyelesaiannya ataupun tidak. Karena sifatnya yang bergantung pada repetisi, algoritma ini biasa digunakan sebagai solusi terakhir dalam menyelesaikan persoalan apapun karena setiap persoalan pasti dapat diselesaikan menggunakan urutan langkah-langkah yang diketahui, dengan syarat bahwa setiap langkah pasti didefinisikan jelas batasan-batasannya. Karena sifatnya yang mudah dipahami, algoritma ini biasa digunakan sebagai pembandingan terhadap algoritma penyelesaian lain terkait pengukuran waktu dan memori.

Strategi algoritma *brute force* biasa digunakan dalam berbagai persoalan, diantaranya *traveling salesman problem*, *integer knapsack problem*, *n-queens problem*, *closest pair*, dan masih banyak lagi. Namun, strategi ini juga memiliki beberapa kelemahan, diantaranya adalah penggunaan sumber daya memori yang cukup besar, pencarian solusi yang waktu lama, serta ketergantungan terhadap spesifikasi perangkat keras.

4.2 Penerapan

<algoritma penyelesaian, penjelasan terhadap tiap langkah, status puzzle dan puzzleMap tiap langkah>

Penyelesaian persoalan IQ Puzzle Solver dilakukan dengan pendekatan algoritma *brute force*. Penyelesaian ini dapat dijabarkan sesuai dengan tahapan sebagai berikut.

1. Transformasi masukan pengguna menjadi objek Puzzle.

Hal ini dilakukan dengan melakukan pembacaan file terhadap masukan teks file dengan extension .txt. Selanjutnya, dilakukan pengelompokkan sesuai dengan huruf alfabetis yang membentuk objek Puzzle tersebut. Hal ini dilakukan berulang hingga seluruh blok puzzle dapat diperoleh. Kemudian, dilakukan standarisasi terhadap setiap blok puzzle sedemikian sehingga dapat membentuk matriks dengan menambahkan spasi hingga terbentuk matriks dengan ukuran baris m dan ukuran kolom n dengan m dan n adalah jumlah baris terbanyak dan jumlah kolom terbanyak pada puzzle tersebut. Selanjutnya,

puzzle-puzzle yang telah diperoleh akan disimpan pada suatu List of Puzzle dengan nama puzzleList.

2. Lakukan penempatan Puzzle terhadap PuzzleMap secara iteratif

Algoritma penyelesaian utama memanfaatkan algoritma *brute force* dinyatakan sebagai berikut.

- a. Ambil Puzzle dari PuzzleList secara acak
- b. Lakukan validasi bahwa banyak kotak kosong pada PuzzleMap masih cukup untuk dipenuhi oleh objek Puzzle tersebut
 - i. Jika benar, dilakukan algoritma *searching* untuk menemukan letak koordinat sedemikian sehingga objek Puzzle dapat memenuhi blok tersebut tanpa melanggar batasan bahwa Puzzle tidak boleh saling menindih
 1. Jika dapat dilakukan, maka objek Puzzle dimodifikasi sesuai dengan koordinat tersebut dan mengikuti bentuk objek Puzzle dengan pengisian koordinat yang bersangkutan dengan karakter yang dilambangkan oleh Puzzle tersebut.
 2. Jika tidak dapat dilakukan, dilakukan proses pencerminan terhadap sumbu horizontal terhadap objek Puzzle tersebut dan diuji kembali apakah dapat memenuhi koordinat tersebut.
 - a. Jika tidak dapat dilakukan lagi, dilakukan proses pencerminan terhadap sumbu vertikal dan dilakukan proses yang sama.
 - i. Jika tidak dapat dilakukan lagi, dilakukan proses pencerminan terhadap sumbu horizontal dan dilakukan proses yang sama.
 1. Jika tidak dapat dilakukan lagi, dilakukan proses rotasi dan pengujian dimulai kembali dari tahap (2).
 - a. Jika tidak dapat dilakukan hingga 8 kali modifikasi, maka dilakukan penggantian objek Puzzle.
 - ii. Jika salah, dipilih kembali objek Puzzle lain yang belum digunakan dan dilakukan pengujian (b)
 - c. Jika salah satu proses sudah mengakibatkan PuzzleMap penuh dan seluruh karakter yang dilambangkan blok puzzle sudah berada pada atribut CharInMap pada PuzzleMap, maka diperoleh bahwa hasil telah ditemukan.
 3. Percobaan dihentikan ketika diperoleh bahwa PuzzleMap sudah penuh ataupun percobaan sudah melewati batas waktu yang ditentukan.

Hasil PuzzleMap ditampilkan dengan urutan yang telah diperoleh dan pengguna dapat melakukan penyimpanan terhadap hasil tersebut. Kemudian, pengguna akan diminta untuk

memilih mode penyimpanan hasil tersebut, baik dalam format .txt ataupun .png. Terakhir, pengguna akan diminta untuk memasukkan nama bagi file simpanan tersebut. Jika tidak diperoleh hasil sama sekali, maka program akan mengembalikan jumlah perulangan dan waktu yang terlewat dalam menyelesaikan percobaan tersebut.

4.3 *Sumber Kode*

Main.java

```
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import java.awt.image.BufferedImage;
import java.util.Scanner;
// import javax.imageio.ImageIO;

import javax.imageio.ImageIO;

import src.Puzzle;
import src.PuzzleMap;

public class Main {

    // instantiation variable
    private static Integer rows;
    private static Integer columns;
    private static Integer blocks;
    private static String gameMode;
    private static File targetFile;
    private static List<List<Character>> letterBlock = new ArrayList<>();
    private static List<List<List<Character>>> puzzle;

    // main program
    public static void main(String[] args) {
        try {
            // reading the content of the directory
            String directoryPath = "./test/";
            Scanner inputScanner = new Scanner(System.in);
```

```

String fileName;

File directory = new File(directoryPath);

if (directory.exists() && directory.isDirectory()) {
    String[] files = directory.list();

    if (files != null) {
        for (String file : files) {
            System.out.println(file);
        }
    }

    // inputting the name of the directory
    System.out.println("Enter filename : ");
    fileName = inputScanner.nextLine();
    targetFile = new File(directory, fileName);
    while (!targetFile.exists() || !targetFile.isFile()) {
        System.out.println("Enter filename again : ");
        fileName = inputScanner.nextLine();
        targetFile = new File(directory, fileName);
    }
}

Puzzle[] puzzleList;
Scanner scanner = new Scanner(targetFile);
String line = scanner.nextLine();

// reading the config
for (int i = 0; i < 2 && scanner.hasNextLine(); i++) {

    // new

    // old; deprecated
    List<Character> configListPerLine = new ArrayList<>();
    for (int j = 0; j < line.length(); j++) {
        configListPerLine.add(line.charAt(j));
    }
}

```



```

        // read the m x n and how many puzzle block
        if (i == 0) {
            String[] parts = line.split(" ");
            rows = Integer.parseInt(parts[0]);
            columns = Integer.parseInt(parts[1]);
            blocks = Integer.parseInt(parts[2]);
            line = scanner.nextLine();
        } else { // i must be 1
            StringBuilder modeBuilder = new StringBuilder();
            for (char ch : configListPerLine) {
                modeBuilder.append(ch);
            }
            gameMode = modeBuilder.toString();
        }
    }

    // reading puzzle
    while (scanner.hasNextLine()) {

        // variable instantiation
        List<Character> letterBlockPerLine = new ArrayList<>();

        // reading puzzle block per line
        line = scanner.nextLine();
        for (int i = 0; i < line.length(); i++) {
            letterBlockPerLine.add(line.charAt(i));
        }

        // check if the next line has the same character as the
list of character above it
        letterBlock.add(letterBlockPerLine);
    }
    scanner.close();

    // validation for config data
    if (blocks instanceof Integer && blocks != 0) {
        if (gameMode.equals("DEFAULT")) {
            // organize letterBlock into shapes
            puzzle = organizeList(letterBlock);
        }
    }
}

```

```

        // standardize all the spaces and make it all in
puzzle format and stored in a list[]
        puzzleList = standardizePuzzle(puzzle);

        // main algorithm
        PuzzleMap MainMap = new PuzzleMap(rows, columns);

        // fill the main map

        // variables
        boolean full = false;
        int operationIter = 0;
        // int maxIter = 8 * puzzleList.length *
factorial(puzzleList.length) * 100;
        long timeStart = System.currentTimeMillis();
        long duration = 300000;

        while (MainMap.getCharInMap().size() <=
puzzleList.length && !full && (System.currentTimeMillis() - timeStart) <
duration) {

            // variables
            int nBlockPuzzle = 0;
            Puzzle puzzle = MainMap.puzzleNotUsed(MainMap,
puzzleList);

            // validation
            if (puzzle != null) {
                // get how many block is filled by puzzle
                for (int i = 0; i < puzzle.getRows(); i++) {
                    for (int j = 0; j < puzzle.getColumns();
j++) {

                        if (puzzle.getElement(i, j) == '1') {
                            nBlockPuzzle++;
                        }
                    }
                }

                // check are there enough empty blocks for
puzzle

```

```

        if (MainMap.emptyBoxInMap() >= nBlockPuzzle &&
MainMap.isThereAValidPosition(MainMap, puzzle)) {
            // trace each position for availability to
be filled

            int k = 0;
            boolean found = false;
            while (!found && k < MainMap.getRows()) {
                int l = 0;
                while (!found && l <
MainMap.getColumns()) {
                    if (MainMap.canBlockFit(k, l,
MainMap, puzzle)) { // block dapat masuk

                        found = true;
                        MainMap.setMapAfterPuzzle(k,
l, MainMap, puzzle);

                    } else { // block cannot fit
                        int modifyPuzzleIter = 0;

                        // check every possibility
                        while (modifyPuzzleIter < 8 &&
!found) {

                            puzzle.modifyPuzzle();
                            if (MainMap.canBlockFit(k,
l, MainMap, puzzle)) { // block dapat masuk

                                found = true;

                                MainMap.setMapAfterPuzzle(k, l, MainMap, puzzle);

                            } else {
                                modifyPuzzleIter++;
                            }
                        }

                        puzzle.modifyPuzzle(); //
reset to first position

                        // resetting variables
                        if (!found) {
                            // check other coordinate
                            l++;
                        }
                    }
                }
            }
        }
    }
}

```

```

        }
        if (!found) {
            k++;
        }
    }
    } else { // there are no enough space in main
map
        MainMap.resetPuzzleMap();
    }

    // check if it's full
    if ((MainMap.getCharInMap().size() ==
puzzleList.length) && (MainMap.emptyBoxInMap() == 0)) {
        full = true;
    }
    operationIter++;

    // main debug
    // MainMap.getPuzzleMap();
    // System.out.println(MainMap.getCharInMap());

    }
}

// keeping the result as a .txt file
if (full) {
    MainMap.getPuzzleMap();
    long timeStop = System.currentTimeMillis();
    System.out.println("Elapsed time : " + (timeStop -
timeStart) + " ms.");

    System.out.println("Iteration tried : " +
operationIter);

    System.out.println("Want to save file : (y/n) ");
    String choice = inputScanner.nextLine();
    if (choice.equals("y")) {
        // bonus : converting into image
        System.out.println("Convert to image : (y/n)
");

        String convert = inputScanner.nextLine();

```

```

        System.out.println("Enter name to be saved:");
    );

        String fileSavedName =
inputScanner.nextLine();

        if (convert.equals("Y")) {
            // converting into image
            int width = MainMap.getColumns();
            int height = MainMap.getRows();
            List<Integer> colorHex = new
ArrayList<>();

            Random random = new Random();
            for (int i = 0; i < puzzleList.length;
i++) {

                int color = random.nextInt(0xFFFFFFFF +
1);

                colorHex.add(color);
            }

            // Create a BufferedImage
            BufferedImage image = new
BufferedImage(width, height, BufferedImage.TYPE_INT_RGB);

            // Convert matrix to image pixels
            for (int y = 0; y < height; y++) {
                for (int x = 0; x < width; x++) {
                    // searching index of character in
charMap

                    boolean found = false;
                    int i = 0;

                    while (!found && i <
MainMap.getCharInMap().size()) {

                        if
(MainMap.getCharInMap().get(i) == MainMap.getElement(y, x)) {
                            image.setRGB(x, y,
colorHex.get(i)); // Set pixel to black

                            found = true;
                        }
                        i++;
                    }
                }
            }
        }
    }
}

```

```

    }

    // Save the image as a PNG file
    try {

        int numberOfOutput =
random.nextInt(10);

        File output = new File("output" +
numberOfOutput + ".png");

        ImageIO.write(image, "png", output);
        System.out.println("Image saved as " +
output + ".png");

    } catch (Exception e) {
        e.printStackTrace();
    }
    } else {
        writeMatrixToFile(MainMap, fileSavedName);
    }
    } else {
        System.out.println("Thank you for using this
product.");
    }
    inputScanner.close();
    } else {
        System.out.println("Brute force cannot be
done.\nPlease try another file.");
    }

    } else {
        System.out.println("No service as such. Please try
again.");
    }
    } else {
        System.out.println("Not valid datatype for number of
blocks");
    }

    } catch (Exception e) {

```

```

        System.out.println("Error " + e + " has occurred : " +
e.getMessage());
    }
}

// func : get index element that is not ' ' in letterBlock
private static char getCharFromCharMatrix (List<List<Character>>>
letterBlockShape) {
    // search for existing alphabet
    int i = 0;

    if (letterBlockShape.isEmpty()) {
        return '~';
    } else {
        while (letterBlock.get(0).get(i) == ' ') {
            i++;
        }
        return letterBlockShape.get(0).get(i);
    }
}

private static List<List<List<Character>>> organizeList
(List<List<Character>>> letterBlock) {
    List<List<List<Character>>> puzzle = new ArrayList<>();
    List<List<Character>>> letterBlockShape = new ArrayList<>();
    for (int i = 0; i < (letterBlock.size()); i++) {
        if (letterBlockShape.isEmpty()) {
            letterBlockShape.add(letterBlock.get(i));
        } else { // letterBlockShape not empty

            int k = 0;
            while (letterBlock.get(i).get(k) == ' ') {
                k++;
            }

            // final check
            if (letterBlock.get(i).get(k) ==
getCharFromCharMatrix(letterBlockShape)) {
                letterBlockShape.add(letterBlock.get(i));
            } else {

```

```

        puzzle.add(letterBlockShape);
        letterBlockShape = new ArrayList<>();
        letterBlockShape.add(letterBlock.get(i));
    }

}

// case : handling end in line shapes
if (i == (letterBlock.size() - 1)) {
    puzzle.add(letterBlockShape);
}

}

return puzzle;
}

// func : standardize puzzle
private static Puzzle[] standardizePuzzle (List<List<List<Character>>>
rawPuzzle) {
    Puzzle[] puzzleListNew = new Puzzle[rawPuzzle.size()];

    // for adding remaining spaces if needed
    for (int i = 0; i < rawPuzzle.size(); i++) {
        int maxCol = rawPuzzle.get(i).get(0).size();
        for (int j = 0; j < rawPuzzle.get(i).size(); j++) {
            if (rawPuzzle.get(i).get(j).size() > maxCol) {
                maxCol = rawPuzzle.get(i).get(j).size();
            }
        }

        // add all remaining spaces if needed
        for (int k = 0; k < rawPuzzle.get(i).size(); k++) {
            if (rawPuzzle.get(i).get(k).size() < maxCol) {
                rawPuzzle.get(i).get(k).add(' ');
            }
        }
    }

    // status : rawPuzzle has been added with spaces if needed

```



```

        // converting each matrix into puzzle
        for (int i = 0; i < rawPuzzle.size(); i++) {
            Puzzle puzzlePieces = new Puzzle(rawPuzzle.get(i));
            puzzleListNew[i] = puzzlePieces;
        }

        return puzzleListNew;
    }

    public static int factorial(int x){
        if (x == 0) {
            return 1;
        } else {
            return x * factorial(x - 1);
        }
    }

    // writing result into file
    public static void writeMatrixToFile(PuzzleMap map, String filename) {
        try (BufferedWriter writer = new BufferedWriter(new
FileWriter(filename))) {
            writer.newLine();
            for (int i = 0; i < map.getRows(); i++) {
                for (int j = 0; j < map.getColumns(); j++) {
                    writer.write(map.getElement(i, j));
                }
                writer.newLine();
            }
            System.out.println("Matrix saved to " + filename);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

```

Puzzle.java

```

package src;

import java.util.ArrayList;
import java.util.List;

```

```

/* Puzzle class attributes
 * 1. row and column
 * 2. list of list of '+'
 */

public class Puzzle {

    // attributes
    private int rows;
    private int columns;
    private List<List<Character>> matrix;
    private char character;
    private boolean isFlipVertical;
    private boolean isFlipHorizontal;
    private boolean isTransposed;

    // constructor
    public Puzzle(List<List<Character>> rawPuzzle) {
        // reading row
        // System.out.println(rawPuzzle.size()); // for debug
        this.rows = rawPuzzle.size();

        // reading column
        this.columns = rawPuzzle.get(0).size(); // for debug
        // System.out.println(rawPuzzle.get(0).size());

        // make the puzzle in matrix format
        this.matrix = new ArrayList<>();
        for (int i = 0; i < rawPuzzle.size(); i++) {
            List<Character> rows = new ArrayList<>();
            for (int j = 0; j < rawPuzzle.get(i).size(); j++) {
                if (rawPuzzle.get(i).get(j) != ' ') {
                    rows.add('1');
                } else {
                    rows.add('0');
                }
            }
        }
    }
}

```

```

        }
        matrix.add(rows);
    }

    // get the character of the current matrix
    int i = 0;
    while (rawPuzzle.get(0).get(i) == ' ') {
        i++;
    }
    this.character = rawPuzzle.get(0).get(i);

    // status of puzzle
    this.isFlipHorizontal = false;
    this.isFlipVertical = false;
    this.isTransposed = false;
}

// Getter : rows
public int getRows() {
    return this.rows;
}

// Getter : columns
public int getColumns() {
    return this.columns;
}

// Getter : character
public char getCharacter() {
    return this.character;
}

// getter : matrix
public List<List<Character>> getMatrix () {
    return this.matrix;
}

// getter : element in matrix
public char getElement(int rows, int columns) {
    return this.getMatrix().get(rows).get(columns);
}

```

```
}

// getter : flipped vertical status
public boolean getStatusFlippedVertical() {
    return this.isFlipVertical;
}

// getter : flipped vertical status
public boolean getStatusFlippedHorizontal() {
    return this.isFlipHorizontal;
}

// getter : transposed status
public boolean getStatusTransposed() {
    return this.isTransposed;
}

// setter : columns
public void setColumns(int newColumns) {
    this.columns = newColumns;
}

// setter : rows
public void setRows(int newRows) {
    this.rows = newRows;
}

// setter : characters
public void setCharacter(char newCharacter) {
    this.character = newCharacter;
}

// setter : matrix
public void setMatrix (List<List<Character>> newMatrix) {
    this.matrix = newMatrix;
}

// setter : matrix per rows
public void setRowsInMatrix (Integer i, List<Character> newRows) {
    this.matrix.set(i, newRows);
}
```

```

}

// setter : set flipped vertical status
public void setStatusFlippedVertical(boolean bool) {
    this.isFlipVertical = bool;
}

// setter : set flipped vertical status
public void setStatusFlippedHorizontal(boolean bool) {
    this.isFlipHorizontal = bool;
}

// setter : set transposed status
public void setTransposed(boolean bool){
    this.isTransposed = bool;
}

// setter : transpose puzzle
public void transposePuzzle () {
    // variable
    List<List<Character>> transposed = new ArrayList<>();

    for (int i = 0; i < this.getColumns(); i++) {
        List<Character> transposedRows = new ArrayList<>();
        for (int j = 0; j < this.getRows(); j++) {
            transposedRows.add(this.getElement(j, i));
        }
        transposed.add(transposedRows);
    }
    this.setMatrix(transposed);

    // changing state of the puzzle
    int temp = this.getColumns();
    this.setColumns(this.getRows());
    this.setRows(temp);
    if (this.getStatusTransposed() == true) {
        this.setTransposed(false);
    } else {
        this.setTransposed(true);
    }
}

```

```

    }

    // func : flip horizontal puzzle
    public void flipHorizontalPuzzle () {
        // fill all the final matrix
        for (int i = 0; i < this.getRows(); i++) {
            List<Character> finalRows =
this.getMatrix().get(i).reversed();
            this.setRowsInMatrix(i, finalRows);
        }
        // changing status of isflippedhorizontal
        if (this.getStatusFlippedHorizontal() == true) {
            this.setStatusFlippedHorizontal(false);
        } else {
            this.setStatusFlippedHorizontal(true);
        }
    }

    // func : flip vertical puzzle
    public void flipVerticalPuzzle() {
        // transpose, reverse, transpose

        // transposing
        // variable
        this.transposePuzzle();
        this.flipHorizontalPuzzle();
        this.transposePuzzle();

        // Changing status of flipping
        if (this.getStatusFlippedHorizontal() == true) {
            this.setStatusFlippedHorizontal(false);
        } else {
            this.setStatusFlippedHorizontal(true);
        }
        if (this.getStatusFlippedVertical() == true) {
            this.setStatusFlippedVertical(false);
        } else {
            this.setStatusFlippedVertical(true);
        }
    }

```

```

    }

    public void modifyPuzzle() {
        /*
        * urutan
        * cek awal
        * cek flip vertikal
        * cek flip vertikal + horizontal
        * cek flip horizontal
        */
        if (this.getStatusFlippedVertical() == false &&
this.getStatusFlippedHorizontal() == false) {
            this.flipVerticalPuzzle();
        } else if (this.getStatusFlippedVertical() == true &&
this.getStatusFlippedHorizontal() == false){
            this.flipHorizontalPuzzle();
        } else if (this.getStatusFlippedVertical() == true &&
this.getStatusFlippedHorizontal() == true){
            this.flipVerticalPuzzle();
        } else if (this.getStatusTransposed() == false){ // default case
            this.flipHorizontalPuzzle();
            this.transposePuzzle();
        } else {
            this.transposePuzzle();
            this.flipVerticalPuzzle();
        }
        // System.out.println(this.getMatrix());
    }
}

```

PuzzleMap.java

```
package src;

import java.util.ArrayList;

import java.util.List;

import java.util.Random;

public class PuzzleMap {

    // attributes

    private int rows;

    private int columns;

    private List<List<Character>> map;

    private List<Character> charInMap;

    // constructor

    public PuzzleMap(int rows, int columns){

        this.rows = rows;

        this.columns = columns;

        this.map = new ArrayList<>();

        for (int i = 0; i < rows; i++) {

            List<Character> row = new ArrayList<>();

            for (int j = 0; j < columns; j++) {

                row.add('?');

            }

        }

    }

}
```



```
    }

    map.add(row);

}

this.charInMap = new ArrayList<>();

}

// getter : rows

public int getRows() {

    return this.rows;

}

// getter : columns

public int getColumns() {

    return this.columns;

}

// getter : element

public char getElement(int rows, int columns) {

    return this.map.get(rows).get(columns);

}

// getter : charInMap

public List<Character> getCharInMap() {
```

```
        return this.charInMap;

    }

    // setter : rows

    public void setRows(int newRows) {

        this.rows = newRows;

    }

    // setter : columns

    public void setColumns (int newColumns) {

        this.columns = newColumns;

    }

    // setter : element

    public void setElement (int rows, int columns, Character c) {

        map.get(rows).set(columns, c);

    }

    // func : show map

    public void getPuzzleMap(){

        for (int i = 0; i < this.map.size(); i++) {

            List<Character> row = this.map.get(i);

            for (int j = 0; j < row.size(); j++) {
```

```

        System.out.print(row.get(j));

    }

    System.err.println();

}

System.out.println();

}

// func : set map empty after finding blockage
public void resetPuzzleMap() {

    for (int i = 0; i < this.map.size(); i++) {

        for (int j = 0; j < this.map.get(i).size(); j++) {

            this.setElement(i, j, '?');

        }

    }

    this.charInMap = new ArrayList<>();

}

// func : check if the puzzle map has still slot left
public int emptyBoxInMap(){

    int n = 0;

    for (int i = 0; i < this.getRows(); i++) {

        for (int j = 0; j < this.getColumns(); j++) {

```

```

        if (this.getElement(i, j) == '?') {

            n++;

        }

    }

}

return n;

}

// func : check if the puzzle blocks can fit with current coordinate

public boolean canBlockFit(int rows, int cols, PuzzleMap map, Puzzle
puzzle) {

    // variables

    boolean fit = true;

    // count how many blocks puzzle will take

    int nBlocksPuzzle = 0;

    for (int i = 0; i < puzzle.getRows(); i++) {

        for (int j = 0; j < puzzle.getColumns(); j++) {

            if (puzzle.getElement(i, j) != '0') {

                nBlocksPuzzle++;

            }

        }

    }

```

```

    }

    // check if the block can fit

    if (map.emptyBoxInMap() >= nBlocksPuzzle) {

        int i = 0;

        while (i < puzzle.getRows() && fit) {

            int j = 0;

            while (j < puzzle.getColumns() && fit) {

                if ((i + rows) < map.getRows() && ((j + cols) <
map.getColumns())) {

                    // System.out.println("map at " + i + rows + " and
" + j + cols + " position : " + map.getElement(i + rows, j + cols));

                    if (puzzle.getElement(i, j) == '1' &&
map.getCharInMap().contains(map.getElement(i + rows, j + cols))) {

                        fit = false;

                    }

                    j++;

                } else {

                    fit = false;

                }

            }

            if (fit) {

                i++;
            }
        }
    }
}

```

```

        }

    }

    } else {

        fit = false;

    }

    return fit;

}

// func : set the map after puzzle blocks being placed

    public void setMapAfterPuzzle(int rows, int cols, PuzzleMap map,
Puzzle puzzle) {

        for (int i = 0; i < puzzle.getRows(); i++) {

            for (int j = 0; j < puzzle.getColumns(); j++) {

                if (puzzle.getElement(i, j) == '1') {

                    map.setElement(i + rows, j + cols,
puzzle.getCharacter());

                }

            }

        }

        map.charInMap.add(puzzle.getCharacter());

    }

// func : check for the puzzle block has been placed before or not

    public Puzzle puzzleNotUsed (PuzzleMap map, Puzzle[] puzzleList) {

```

```

        // variables

        Random random = new Random();

        int i = random.nextInt(puzzleList.length);

        boolean found = false;

        while (!found && (map.getCharInMap().size() < puzzleList.length))
    {

        if (!map.isPuzzleUsed(map, puzzleList[i])) {

            found = true;

        } else {

            i = random.nextInt(puzzleList.length);

        }

    }

    if (found) {

        return puzzleList[i];

    } else {

        return null;

    }

}

public boolean isPuzzleUsed (PuzzleMap map, Puzzle puzzle) {

    // / variables

    boolean used = false;

```

```

int i = 0;

while (!used && i < map.getCharInMap().size()) {

    if (map.getCharInMap().get(i) == puzzle.getCharacter()) {

        used = true;

    } else {

        i++;

    }

}

return used;

}

```

// func : to check is there a position that can be filled by a particular puzzle block

```

public boolean isThereAValidPosition(PuzzleMap map, Puzzle puzzle) {

    // variables

    for (int i = 0; i < map.getRows(); i++) {

        for (int j = 0; j < map.getColumns(); j++) {

            if (map.canBlockFit(i, j, map, puzzle)) {

                return true; // Return immediately when found

            }

        }

    }

}

```



```
        return false; // No valid position found
    }
}
```

5 Data Test

5.1 Test 1

- File : test1.txt

5 5 7

DEFAULT

A

AA

B

BB

C

CC

D

DD

EE

EE

E

FF

FF

F

GGG

- Hasil

EEGCC

EEGCD

FEGDD

FFBBA

FFBAA

Elapsed time : 67 ms.

Iteration tried : 935

- Penjelasan

Puzzle berhasil ditemukan penyelesaiannya dalam waktu 67 ms dan perulangan sebanyak 935 kali.

5.2 Test 2

- File : test2.txt

3 3 3

DEFAULT

A

AA

- B
- BB
- CCC
- Hasil
 - CAA
 - CAB
 - CBB
 - Elapsed time : 10 ms.
 - Iteration tried : 11
- Penjelasan
 - Pencarian berhasil ditemukan dengan perulangan 11 kali dan percobaan selama 10 milisekon.

5.3 *Test 3*

- File : test3.txt
 - 4 4 5
 - DEFAULT
 - A
 - AA
 - B
 - BB
 - BB
 - CCC
 - D
 - DD
 - EE
- Hasil
 - test3.txt
 - AEEB
 - AABB
 - DDBB
 - DCCC
 - Elapsed time : 27 ms.
 - Iteration tried : 375
- Penjelasan
 - Puzzle berhasil ditemukan penyelesaiannya dalam waktu 27 ms dan perulangan sebanyak 375 kali.

5.4 Test 4

- File : test4.txt
2 2 2
DEFAULT
A
B
BB
- Hasil
BA
BB

Elapsed time : 4 ms.

Iteration tried : 4

- Penjelasan
Puzzle berhasil ditemukan penyelesaiannya dalam waktu 4 ms dan perulangan sebanyak 4 kali.

5.5 Test 5

- File : test5.txt
1 1 1
DEFAULT
A
- Hasil
A

Elapsed time : 2 ms.

Iteration tried : 1

- Penjelasan
Puzzle berhasil ditemukan penyelesaiannya dalam waktu 2 ms dan perulangan sebanyak 1 kali.

5.6 Test 6

- File : test6.txt
5 5 7
DEFAULT
A
AA
B
BB

C
CC
D
DD
E
EE
FFF
GG
GG
GG
G
- Hasil
DDAAF
BDACF
BBCCF
EGGGG
EEGGG

Elapsed time : 43 ms.

Iteration tried : 597

- Penjelasan
Puzzle berhasil ditemukan penyelesaiannya dalam waktu 43 ms dan perulangan sebanyak 597 kali.

5.7 Test 7

- File : test7.txt
3 3 3
DEFAULT
AA
BB
CC
CC
C
- Hasil
CCA
CCA
CBB

Elapsed time : 18 ms.

Iteration tried : 20

- Penjelasan
Puzzle berhasil ditemukan penyelesaiannya dalam waktu 18 ms dan perulangan sebanyak 20 kali.

6 Lampiran

6.1 Repository Sumber Kode

Github : [BrianHadianSTEI23/tucil1-13523048](https://github.com/BrianHadianSTEI23/tucil1-13523048)

6.2 Tabel Verifikasi

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