Tugas Kecil Strategi Algoritma – IF2210

Penyelesaian Persoalan 15-Puzzle dengan Algoritma *Branch and Bound*

Disusun oleh:

Ignasius Ferry Priguna (13520126)

PROGRAM STUDI TEKNIK INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

BANDUNG

2022

BAB I

CARA KERJA PROGRAM *BRANCH AND BOUND*

Program ini menyelesaikan persoalan 15-Puzzle dengan algoritma *Branch and Bound*. Konfigurasi persoalan 15-Puzzle dapat diperoleh dengan 2 cara, dari pembacaan file teks dan dari pembangkitan konfigurasi secara acak. Sebelum solusi dicari dengan algoritma *branch and bound*, perlu dilakukan pengecekan terhadap konfigurasi awal dengan menghitung nilai berikut:

$$\sum_{i=1}^{16} KURANG(i) + X$$

Fungsi KURANG(i) mengembalikan banyaknya ubin bernomor lebih kecil dari i yang nilai posisinya (baris x 4 + kolom) lebih besar dari nilai posisi ubin bernomor i. Ubin kosong dianggap bernomor 16. Jika hasil perhitungan nilai dari rumus di atas ganjil, persoalan tidak bisa diselesaikan sehingga pencarian solusi tidak dilakukan. Jika genap, persoalan akan dicari solusinya dengan algoritma *branch and bound*.

Algoritma *branch and bounch* dalam program ini diimplementasikan dengan priority queue of Node. Sebuah Node menyimpan susunan angka board, gerakan yang harus dilakukan untuk mencapai susunan tersebut, dan cost dari Node tersebut. Cost Node dihitung berdasarkan jumlah angka yang tidak berada di tempat seharusnya pada board ditambah dengan banyaknya gerakan untuk mencapai susunan angka board yang tersimpan di Node tersebut. Priority queue yang dibuat memprioritaskan Node dengan cost terkecil.

Konfigurasi puzzle dicek apakah sudah berupa solusi. Jika sudah, pencarian tidak dilanjutkan dan solusi diperoleh. Jika belum berupa solusi, konfigurasi puzzle dimasukkan ke dalam queue sebagai Node. Kemudian dilakukan iterasi dengan langkah-langkah berikut sampai queue kosong.

1. Keluarkan Node terdepan pada antrian.
2. Cek apakah Node yang dikeluarkan merupakan solusi. Jika iya Node diset sebagai solusi dan pencarian dihentikan. Jika bukan, lanjutkan ke langkah berikutnya.

3. Cek arah gerak yang dapat dilakukan pada kondisi board saat ini dan tidak menyebabkan board kembali ke susunan 1 gerakan sebelumnya. Tambahkan Node sesuai arah gerak tersebut ke antrian.

Setelah proses iterasi tersebut, solusi dipastikan akan diperoleh jika tidak ada batasan memory. Namun karena keterbatasan memory, ada kemungkinan perangkat keras tidak mampu menampung banyaknya Node dalam queue sehingga akan menghasilkan *Out of Memory Error*.

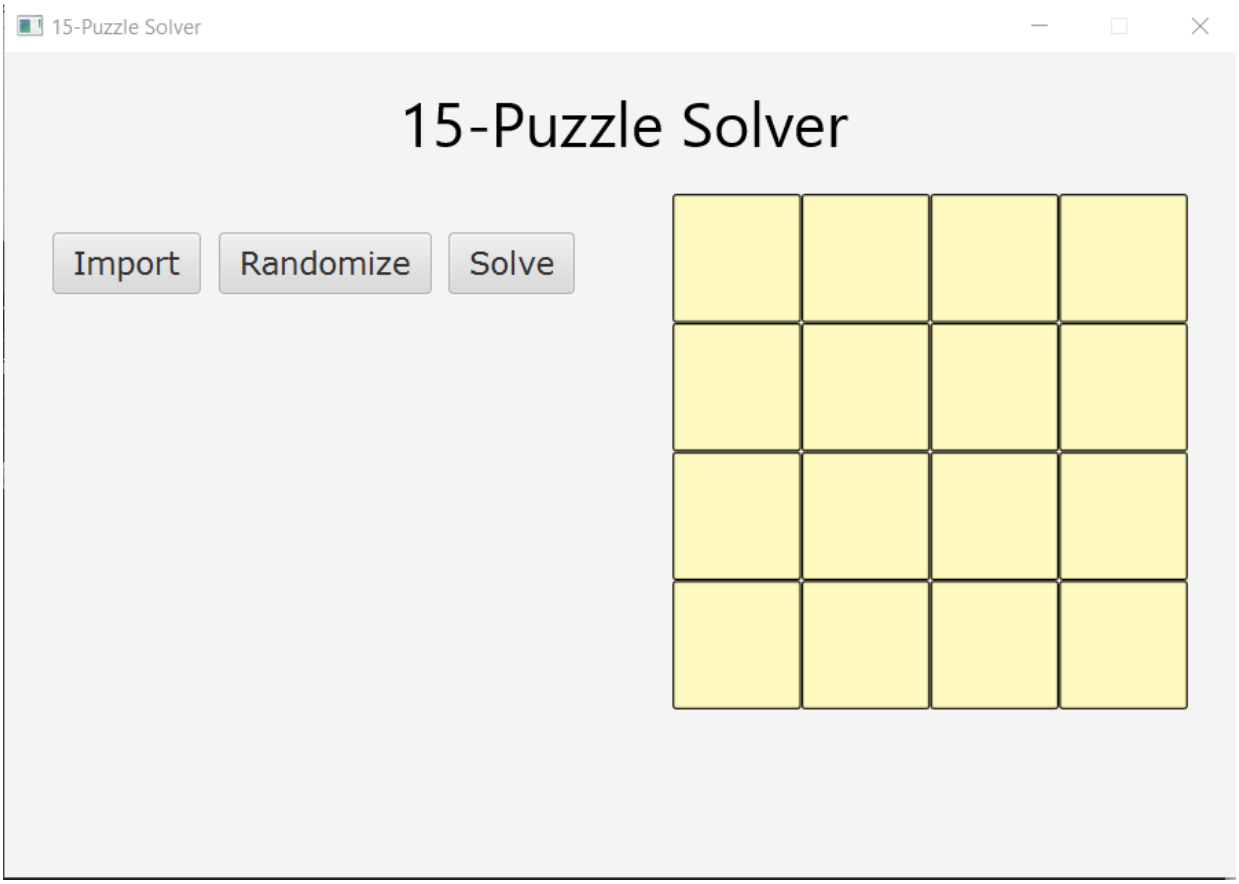
Kemudian, jika solusi berhasil didapatkan pergerakan ubin angka diperlihatkan di program dengan animasi. Animasi dibuat berdasarkan konfigurasi awal dan daftar gerakan yang harus dilakukan untuk mencapai solusi.

BAB II

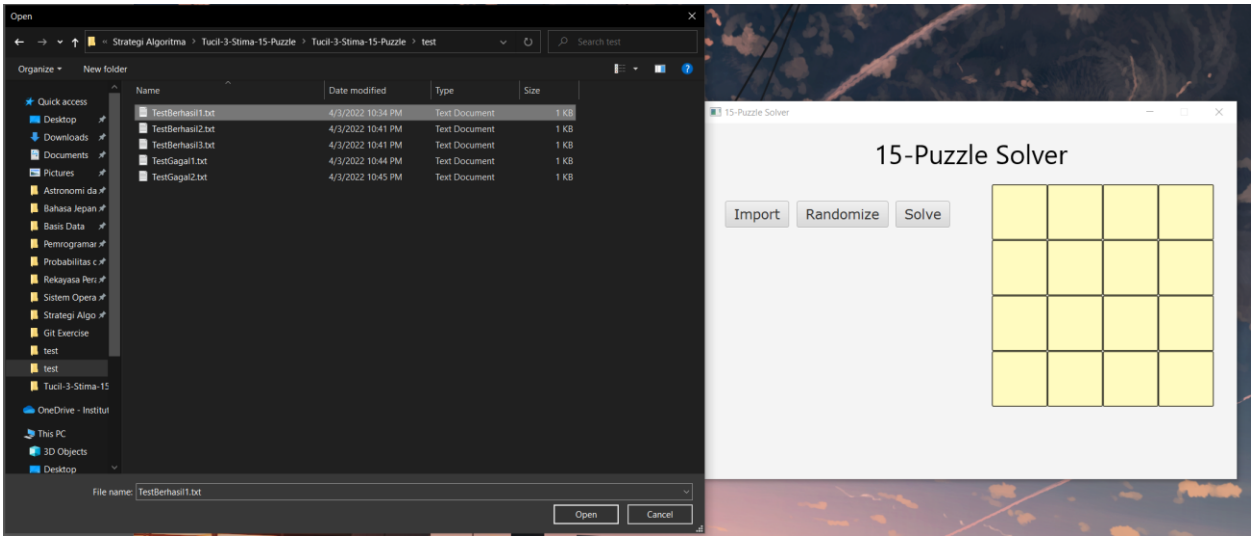
EKSPERIMEN

Setiap eksperimen dilakukan dengan file teks yang tersimpan di folder test. Urutan matriks dari posisi awal ke posisi akhir ditampilkan dalam bentuk animasi sehingga tidak di-*screenshot*. Sebagai gantinya, ditampilkan langkah-langkah yang diambil untuk mencapai posisi akhir.

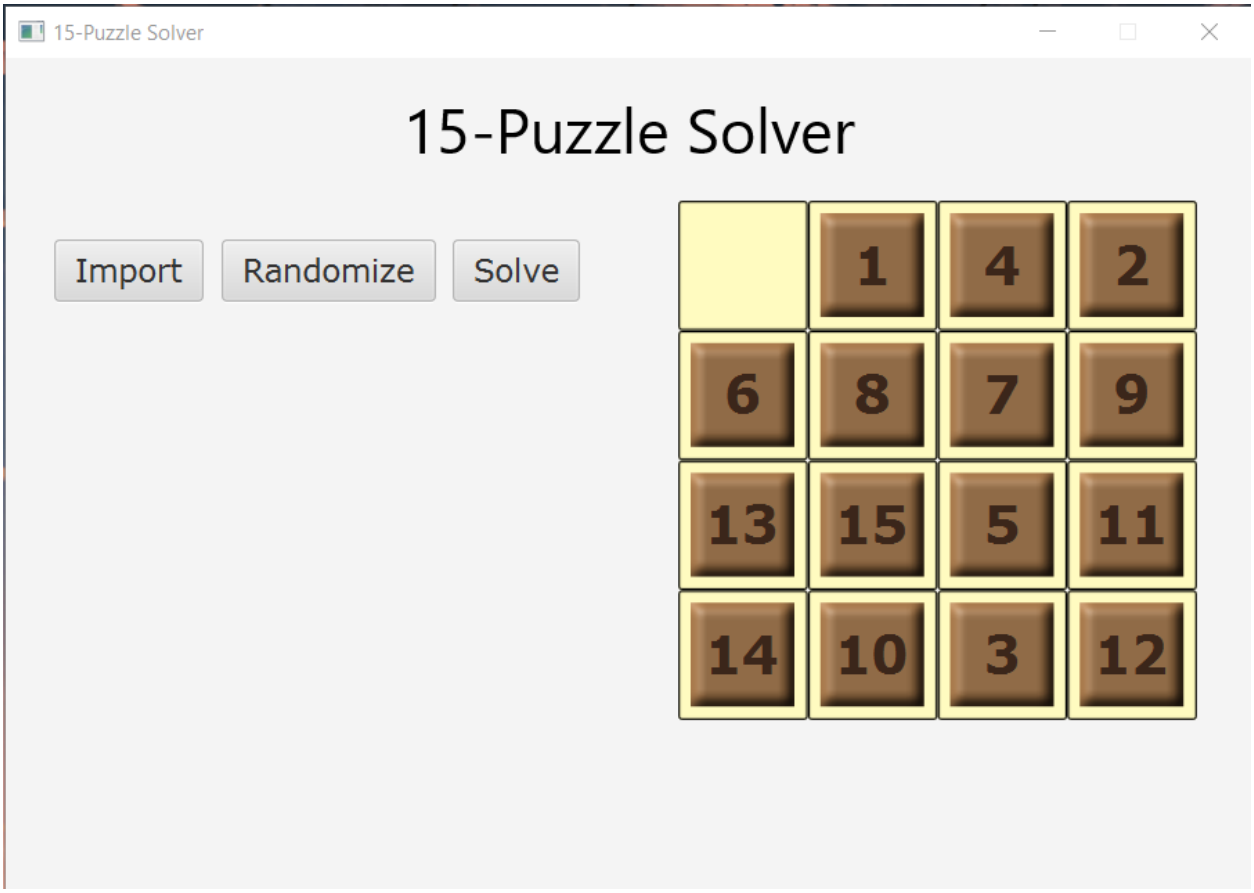
Tampilan Awal Program



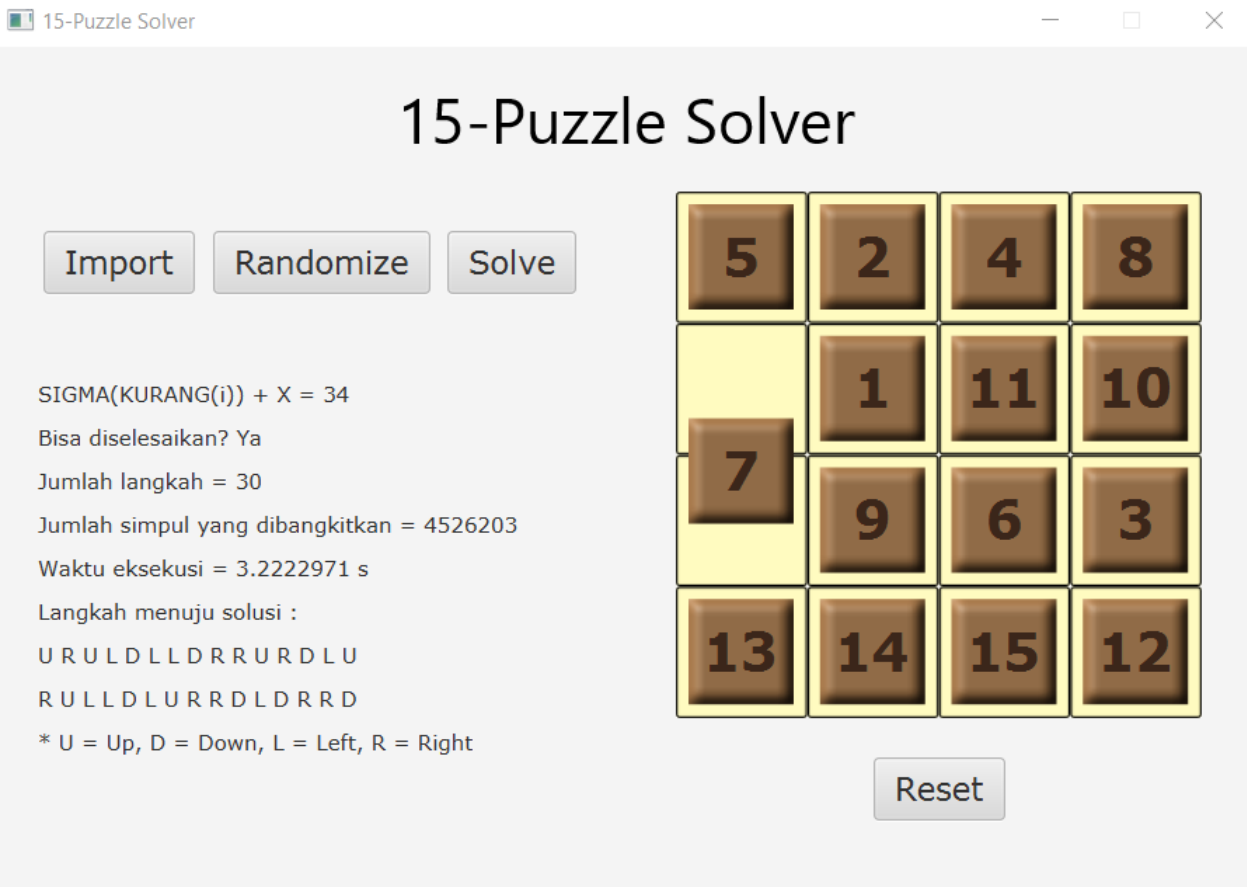
Input Konfigurasi dari File dengan Tombol Import



Penentuan Konfigurasi secara Acak dengan Tombil Randomize



Contoh Saat Animasi Berjalan

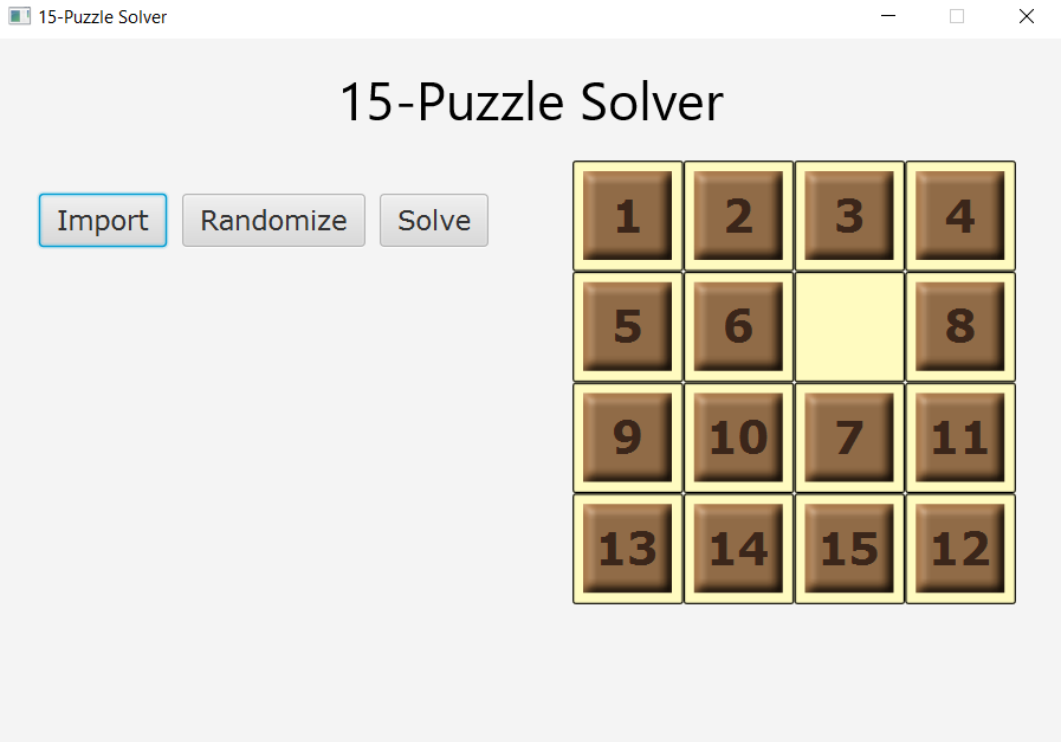


1. Eksperimen 1 : TestBerhasil1.txt

TestBerhasil1.txt

```
1 2 3 4
5 6 - 8
9 10 7 11
13 14 15 12
```

Kondisi Sebelum Pencarian

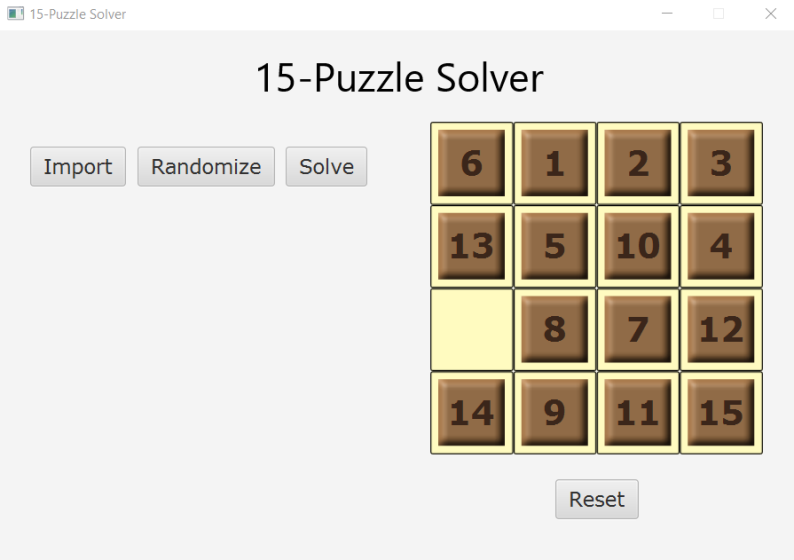


Kondisi Sesudah Pencarian

2. Eksperimen 2 : TestBerhasil2.txt

```
6 1 2 3
13 5 10 4
- 8 7 12
14 9 11 15
```

Kondisi Sebelum Pencarian

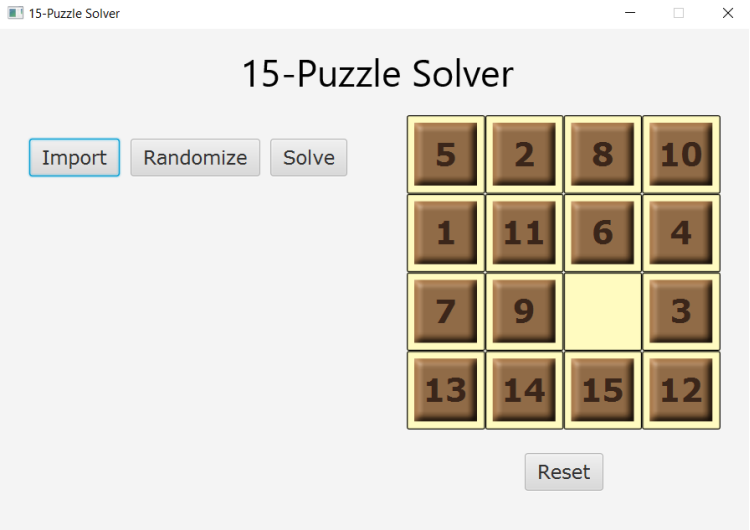


Kondisi Sesudah Pencarian

3. Eksperimen 3 : TestBerhasil3.txt

```
5 2 8 10
1 11 6 4
7 9 - 3
13 14 15 12
```

Kondisi Sebelum Pencarian



Kondisi Sesudah Pencarian

4. Eksperimen 4 : TestGagal1.txt

```
5  9  1  6
8  11  7  4
13  -  3  12
10  2  14  15
```

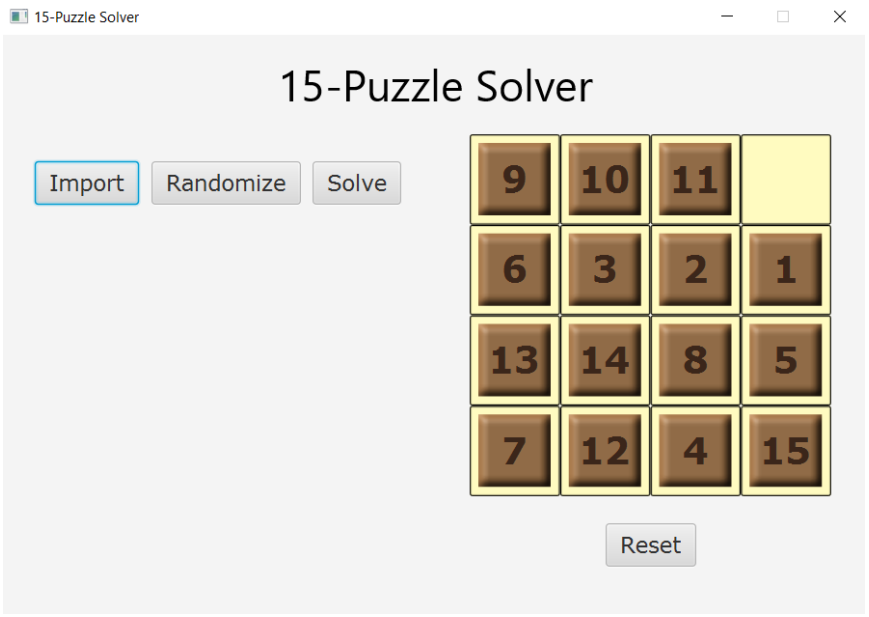
Kondisi Sebelum Pencarian

Kondisi Sesudah Pencarian

5. Eksperimen 5 : TestGagal2.txt

```
9 10 11 -
6 3 2 1
13 14 8 5
7 12 4 15
```

Kondisi Sebelum Pencarian



Kondisi Sesudah Pencarian

BAB III
SOURCE CODE PROGRAM

Program dibuat dengan bantuan IntelliJ IDEA dan menggunakan pustaka JavaFX. Elemen-elemen program seperti tombol dan label yang tersedia dari awal berjalannya program dibuat pada file fxml yang berlokasi di src/main/resources/com/app/Main.fxml. Sedangkan implementasi program dengan bahasa Java terdapat di src/main/java/com.app.

Implementasi dalam bahasa Java dibagi menjadi beberapa file dan package. File Main.java adalah file utama untuk menjalankan keseluruhan program. File Main1.java dibuat untuk keperluan pembuatan executable. File Controller.java adalah file yang mengatur interaksi pengguna dengan program. Package board berisi Board.java yang merepresentasikan susunan angka dalam puzzle. Package bnb berisi 2 file yaitu Solver.java dan Node.java. Solver.java berisi kelas yang mengatur penyelesaian persoalan puzzle dengan algoritma *branch and bound*. Sedangkan Node.java berisi kelas yang menjadi objek dalam antrian yang digunakan dalam proses pencarian. Package elements berisi 2 file yaitu InfoLabel.java dan NumberTile.java. Keduanya merupakan template objek yang muncul di layar saat program berjalan.

Main.java

```
package com.app;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.stage.Stage;

import java.io.IOException;

public class Main extends Application {
    @Override
```

```
    public void start(Stage stage) throws IOException {
        try{
            FXMLLoader fxmlLoader = new
FXMLLoader(Main.class.getResource("Main.fxml"));
            Scene scene = new Scene(fxmlLoader.load());
            stage.setTitle("15-Puzzle Solver");
            stage.setScene(scene);
            stage.setResizable(false);

            stage.show();
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }

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

Main1.java

```
package com.app;

public class Main1 {
    public static void main(String[] args) {
        Main.main(args);
    }
}
```

Controller.java

```
package com.app;

import com.app.bnb.Solver;
import com.app.board.Board;
import com.app.elements.InfoLabel;
import com.app.elements.NumberTile;
import javafx.animation.SequentialTransition;
import javafx.animation.TranslateTransition;
import javafx.fxml.FXML;
import javafx.scene.Group;
import javafx.scene.Node;
import javafx.scene.control.Button;
import javafx.scene.layout.VBox;
import javafx.stage.FileChooser;
import javafx.stage.Stage;
import javafx.util.Duration;

import java.io.File;


public class Controller {

    private Solver solver;
    private Board currentBoard;
    private boolean isConfigured = false;
    private boolean isSolved = false;
    private int step = 0;
    private SequentialTransition sq;
```

```
    @FXML private Group NumGrid;
    @FXML private VBox InfoBox;
    @FXML private Button ResetButton;

    @FXML protected void onImportButtonClick() {
        Stage stage = new Stage();
        FileChooser fileChooser = new FileChooser();
        File file = fileChooser.showOpenDialog(stage);

        try {
            if (file != null) {
                NumGrid.getChildren().clear();
                this.currentBoard = new Board();
                this.currentBoard.setContentsFromFile(file.getPath(),
false);
                this.solver = new Solver(this.currentBoard);
                configureInitialNumGrid();
                InfoBox.getChildren().clear();
                isConfigured = true;
                isSolved = false;
            }
        }
        catch (Exception e) {
            e.printStackTrace();
        }

    }

    @FXML protected void onRandomizeButtonClick() {
        NumGrid.getChildren().clear();
        this.currentBoard = new Board();
```

```
        this.currentBoard.setContentsRandomly();
        this.solver = new Solver(this.currentBoard);
        configureInitialNumGrid();
        InfoBox.getChildren().clear();
        isConfigured = true;
        isSolved = false;
    }

    @FXML protected void onSolveButtonClick() {
        ResetButton.setVisible(true);
        if (isConfigured && !isSolved){
            try {
                this.solver.solve();
                isSolved = true;
                isConfigured = false;
                this.step = 0;

                this.showSolveInfo();
                this.playSolutionAnimation();
            }
            catch (OutOfMemoryError e) {
                InfoBox.getChildren().clear();

                // Nilai fungsi pengecekan awal
                InfoLabel labelOutOfMem = new InfoLabel("Memory tidak
cukup untuk melanjutkan pencarian.");
                InfoBox.getChildren().add(labelOutOfMem);

                // Nilai fungsi pengecekan awal
                InfoLabel labelFungsiPengecekan = new
InfoLabel("SIGMA(KURANG(i)) + X = " + solver.initialCheckValue);
                InfoBox.getChildren().add(labelFungsiPengecekan);
```

```
                // Jumlah simpul yang dibangkitkan
                InfoLabel labelJumlahSimpul = new InfoLabel("Jumlah
simpul yang dibangkitkan = " + this.solver.totalNodesCreated);
                InfoBox.getChildren().add(labelJumlahSimpul);
            }


        }
        else if (isSolved) {
            this.playSolutionAnimation();
        }
    }

    @FXML protected void onResetButtonClick() {
        this.currentBoard = new Board(this.solver.problem);
        NumGrid.getChildren().clear();
        configureInitialNumGrid();
        this.step = 0;
    }

    private void configureInitialNumGrid() {
        for (int i = 0; i < 4; i++) {
            for (int j = 0; j < 4; j++) {
                int num = this.solver.problem.getContentsAt(i, j);
                if (num != 0) {
                    NumberTile numberTile = new NumberTile(num, i, j);
                    this.NumGrid.getChildren().add(numberTile);
                }
            }
        }
    }
```

```
    private void showSolveInfo() {
        // Hapus isi InfoBox
        InfoBox.getChildren().clear();

        // Nilai fungsi pengecekan awal
        InfoLabel labelFungsiPengecekan = new
InfoLabel("SIGMA(KURANG(i)) + X = " + solver.initialCheckValue);
        InfoBox.getChildren().add(labelFungsiPengecekan);

        // Bisa diselesaikan berdasarkan fungsi pengecekan awal atau
tidak
        String canBeSolved = this.solver.solvable ? "Ya" : "Tidak";
        InfoLabel labelBisaDiselesaikan = new InfoLabel("Bisa
diselesaikan? " + canBeSolved);
        InfoBox.getChildren().add(labelBisaDiselesaikan);

        if (this.solver.solvable) {
            // Jumlah langkah
            InfoLabel labelJumlahLangkah = new InfoLabel("Jumlah
langkah = " + this.solver.solution.getPathTaken().size());
            InfoBox.getChildren().add(labelJumlahLangkah);

            // Jumlah simpul yang dibangkitkan
            InfoLabel labelJumlahSimpul = new InfoLabel("Jumlah simpul
yang dibangkitkan = " + this.solver.totalNodesCreated);
            InfoBox.getChildren().add(labelJumlahSimpul);

            // Waktu eksekusi program
            InfoLabel labelWaktuEksekusi = new InfoLabel("Waktu
eksekusi = " + this.solver.executionTime + " s");
            InfoBox.getChildren().add(labelWaktuEksekusi);

            // Langkah solusi
```

```
            InfoLabel labelInfoLangkah = new InfoLabel("Langkah menuju
solusi : ");
            InfoBox.getChildren().add(labelInfoLangkah);


            int i = 0;
            while (i < this.solver.solution.getPathTaken().size()) {
                StringBuilder solution = new StringBuilder();
                int j = 0;
                while (i + j <
this.solver.solution.getPathTaken().size() && j < 15) {

solution.append(Character.toUpperCase(this.solver.solution.getPathTake
n().get(i + j)) + " ");
                    j++;
                }
                InfoLabel labeLangkahSolusi = new
InfoLabel(solution.toString());
                InfoBox.getChildren().add(labeLangkahSolusi);
                i += 15;
            }


            InfoLabel labelInfoHuruf = new InfoLabel("* U = Up, D =
Down, L = Left, R = Right");
            InfoBox.getChildren().add(labelInfoHuruf);
        }
    }


    private void playMove(int number, Character move) {
        for (Node n : NumGrid.getChildren()) {
            if (n instanceof NumberTile &&
n.getId().equals(Integer.toString(number))) {
                TranslateTransition translate = new
TranslateTransition();
                translate.setNode(n);
                translate.setDuration(Duration.millis(300));
```

```
                switch (move) {
                    case 'l' -> {
                        translate.setByX(75);
                    }
                    case 'r' -> {
                        translate.setByX(-75);
                    }
                    case 'u' -> {
                        translate.setByY(75);
                    }
                    case 'd' -> {
                        translate.setByY(-75);
                    }
                    default -> {}
                }

                this.sq.getChildren().add(translate);
            }
        }
        this.sq.playFromStart();
    }

    private void moveTile() {
        if (this.step < this.solver.solution.getPathTaken().size()) {
            char move =
this.solver.solution.getPathTaken().get(this.step);
            int moveRowLocation =
this.currentBoard.getEmptyLocationRow();
            int moveColLocation =
this.currentBoard.getEmptyLocationCol();

            switch (move) {
```

```
                case 'l' -> {
                    moveColLocation--;
                }
                case 'r' -> {
                    moveColLocation++;
                }
                case 'u' -> {
                    moveRowLocation--;
                }
                case 'd' -> {
                    moveRowLocation++;
                }
                default -> {}
            }

            int num = this.currentBoard.getContentsAt(moveRowLocation,
moveColLocation);
            this.playMove(num, move);
            this.currentBoard = new Board(this.currentBoard, move);
            this.step++;
        }
    }

    private void playSolutionAnimation() {
        this.sq = new SequentialTransition();
        while (this.step < this.solver.solution.getPathTaken().size())
{
            this.moveTile();
        }
    }
}
```

Solver.java

```
package com.app.bnb;

import com.app.board.Board;

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

public class Solver {
    public Board problem;
    public Node solution;
    public int initialCheckValue;
    public boolean solvable;
    public double executionTime;
    public int totalNodesCreated;

    public Solver() {
        this.problem = new Board();
        this.solution = new Node(this.problem);
        this.problem.setContentsRandomly();
        this.initialCheckValue = 0;
        this.solvable = false;
        this.executionTime = 0;
        this.totalNodesCreated = 0;
    }

    public Solver(Board board) {
        this.problem = new Board(board);
        this.solution = new Node(this.problem);
        this.initialCheckValue = 0;
```

```
        this.solvable = false;
        this.executionTime = 0;
        this.totalNodesCreated = 0;
    }

    public void solve() {
        // Mulai perhitungan waktu
        long startTime = System.nanoTime();

        // Cek apakah puzzle sudah berupa solusi
        if (problem.countMisplacedTiles() == 0) {
            solvable = true;
            totalNodesCreated++;
        }
        // Cek apakah puzzle bisa diselesaikan
        else if (isSolvable()) {
            solvable = true;
            // Buat priority queue dengan elemen awal kondisi awal
board
            PriorityQueue<Node> pq = new PriorityQueue<>();
            pq.add(new Node(this.problem));
            totalNodesCreated++;

            // Iterasi sampai pq kosong
            while (!pq.isEmpty()) {
                // Ambil node di antrian terdepan
                Node currentNode = pq.poll();

                if (currentNode.isSolution()) {
                    this.solution = currentNode;
                    break;
                }
```

```
                else {
                    // Tambahkan node baru ke antrian
                    List<Character> possibleMoves = new
ArrayList<>(currentNode.possibleMoves());
                    for (Character move : possibleMoves) {
                        Node newNode = new Node(currentNode, move);
                        pq.add(new Node(currentNode, move));
                        totalNodesCreated++;
                    }
                }
            }
        }

        // Hentikan perhitungan waktu
        long stopTime = System.nanoTime();
        this.executionTime = ((double) (stopTime - startTime) /
1000000000);
    }

    public boolean isSolvable() {
        // Mengembalikan true jika puzzle bisa diselesaikan
        // Ditentukan berdasarkan penjumlahan jumlah kurang(i) + X
        int value = 0;

        for (int i = 0; i < 16; i++) {
            value += kurang(i);
        }

        if ((this.problem.getEmptyLocationRow() +
this.problem.getEmptyLocationCol()) % 2 != 0) {
            value++;
        }
```

```
        // set initialCheckValue
        this.initialCheckValue = value;

        // Mengembalikan true jika hasil perhitungan genap
        return value % 2 == 0;

    }

    private int kurang(int I) {
        // Mengembalikan banyaknya ubin bernomor x sedemikian sehingga
x < I dan
        // POSISI(x) > POSISI(I).
        // POSISI(I) = posisi ubin bernomor I pada susunan yang
diperiksa

        int position = -1; // posisi i
        int res = 0;

        for (int i = 0; i < 4; i++) {
            for (int j = 0; j < 4; j++) {
                if (position == -1 && problem.getContentsAt(i, j) ==
I) {
                    // Set posisi i
                    position = i * 4 + j;
                } else if (position != -1 && problem.getContentsAt(i,
j) < I && problem.getContentsAt(i, j) != 0) {
                    // Tambah jumlah ubin dengan posisi(X) > posisi(i)
                    res++;
                } else if (position != -1 && I == 0) {
                    res++;
                }
            }
        }
```

```
        return res;
    }

    public void printSolution() {
        System.out.println("Posisi awal:");
        this.problem.printBoard();
        System.out.println("Nilai fungsi pengecekan awal: " +
this.initialCheckValue);
        System.out.println("Bisa diselesaikan? " + this.solvable);
        System.out.println("Solusi: ");
        this.solution.printNode();
        System.out.println("Jumlah simpul yang dibangkitkan: " +
this.totalNodesCreated);
        System.out.println("Waktu eksekusi: " + this.executionTime);
    }
}
```

Node.java

```
package com.app.bnb;

import com.app.board.Board;

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

public class Node implements Comparable<Node> {
    private Board board;
    private List<Character> pathTaken;
    private int cost;

    public Node(Board b) {
        this.board = new Board(b);
```

```
        this.pathTaken = new ArrayList<>();
        this.cost = 99;
    }

    public Node(Node previousNode, Character move) {
        this.board = new Board(previousNode.board, move);
        this.pathTaken = new ArrayList<>(previousNode.pathTaken);
        this.pathTaken.add(move);
        this.cost = this.board.countMisplacedTiles() +
this.pathTaken.size();
    }

    public int getCost() {
        return this.cost;
    }

    public int compareTo(Node n) {
        // Perbandingan cost untuk priority queue
        return Integer.compare(this.cost, n.cost);
    }

    public List<Character> getPathTaken() {
        return new ArrayList<>(pathTaken);
    }

    public List<Character> possibleMoves() {
        // Mengembalikan list gerakan yang dapat dilakukan selanjutnya
        // Gerakan dapat dilakukan jika tidak membuat tile keluar
batas dan tidak menyebabkan kembali ke posisi sebelumnya
        List<Character> l = new ArrayList<>();
        if (board.getEmptyLocationCol() != 0) {
            l.add('l');
```

```
    }
    if (board.getEmptyLocationRow() != 0) {
        l.add('u');
    }
    if (board.getEmptyLocationCol() != 3) {
        l.add('r');
    }
    if (board.getEmptyLocationRow() != 3) {
        l.add('d');
    }

    if (!pathTaken.isEmpty()) {
        Character c = pathTaken.get(pathTaken.size() - 1);
        if (c == 'l') {
            c = 'r';
        }
        else if (c == 'r') {
            c = 'l';
        }
        else if (c == 'u') {
            c = 'd';
        }
        else if (c == 'd') {
            c = 'u';
        }
        l.remove(c);
    }

    return l;
}

public boolean isSolution() {
```

```
        return board.countMisplacedTiles() == 0;
    }

    public void printNode() {
        System.out.println("Matriks: ");
        this.board.printBoard();
        this.pathTaken.forEach(n -> System.out.print(n + " "));
        System.out.println();
    }
}
```

Board.java

```
package com.app.board;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Random;
import java.util.List;

public class Board {
    private int[][] contents;
    private int emptyLocationRow;
    private int emptyLocationCol;

    public Board() {
        contents = new int[4][4];
        for (int i = 0; i < 4; i++) {
            for (int j = 0; j < 4; j++) {
                this.contents[i][j] = i + 1;
```

```
        }
    }
    this.contents[3][3] = 0;
    this.emptyLocationRow = 3;
    this.emptyLocationCol = 3;
}

public Board(Board board) {
    // Set isi board sesuai dengan board pada parameter
    contents = new int[4][4];
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            this.contents[i][j] = board.contents[i][j];
            if (this.contents[i][j] == 0) {
                this.emptyLocationRow = i;
                this.emptyLocationCol = j;
            }
        }
    }
}

public Board(Board board, Character move) {
    // Set isi board sesuai dengan board pada parameter
    contents = new int[4][4];
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            this.contents[i][j] = board.contents[i][j];
            if (this.contents[i][j] == 0) {
                this.emptyLocationRow = i;
                this.emptyLocationCol = j;
            }
        }
```

```
        }

        // Ubah isi board berdasarkan karakter move
        switch (move) {
            case 'l' -> {
this.contents[this.emptyLocationRow][this.emptyLocationCol] =
this.contents[this.emptyLocationRow][this.emptyLocationCol - 1];

this.contents[this.emptyLocationRow][this.emptyLocationCol - 1] = 0;
                this.emptyLocationCol--;
            }
            case 'r' -> {
this.contents[this.emptyLocationRow][this.emptyLocationCol] =
this.contents[this.emptyLocationRow][this.emptyLocationCol + 1];

this.contents[this.emptyLocationRow][this.emptyLocationCol + 1] = 0;
                this.emptyLocationCol++;
            }
            case 'u' -> {
this.contents[this.emptyLocationRow][this.emptyLocationCol] =
this.contents[this.emptyLocationRow - 1][this.emptyLocationCol];
                this.contents[this.emptyLocationRow -
1][this.emptyLocationCol] = 0;
                this.emptyLocationRow--;
            }
            case 'd' -> {
this.contents[this.emptyLocationRow][this.emptyLocationCol] =
this.contents[this.emptyLocationRow + 1][this.emptyLocationCol];
                this.contents[this.emptyLocationRow +
1][this.emptyLocationCol] = 0;
                this.emptyLocationRow++;
            }
```

```
            default -> {
            }
        }
    }

    public int getContentsAt(int i, int j) {
        return this.contents[i][j];
    }

    public int getEmptyLocationRow() {
        return this.emptyLocationRow;
    }

    public int getEmptyLocationCol() {
        return this.emptyLocationCol;
    }

    public void setContentsRandomly() {
        // Set isi board secara acak
        Random rand = new Random();
        List<Integer> numbers = new ArrayList<>();
        for (int i = 0; i < 16; i++) {
            numbers.add(i);
        }

        for (int i = 0; i < 4; i++) {
            for (int j = 0; j < 4; j++) {
                int randomIdx = rand.nextInt(numbers.size());
                int randomNum = numbers.get(randomIdx);
                numbers.remove(randomIdx);
                this.contents[i][j] = randomNum;
                if (randomNum == 0) {
```

```
                    this.emptyLocationRow = i;
                    this.emptyLocationCol = j;
                }
            }
        }
    }

    public void setContentsFromFile(String path, boolean inTestFolder)
throws IOException {
        // Set isi board berdasarkan file teks
        // Diasumsikan file selalu dalam format yang benar
        // Terdiri dari 16 angka, 4 angka per baris dipisahkan oleh
spasi
        // 1 angka digantikan - untuk merepresentasikan tile kosong
        BufferedReader br;
        if (inTestFolder) {
            br = new BufferedReader(new FileReader("Test/" + path));
        }
        else {
            br = new BufferedReader(new FileReader(path));
        }

        try {
            String line = br.readLine();

            for (int i = 0; i < 4; i++) {
                String[] s = line.split(" ");
                for (int j = 0; j < 4; j++) {
                    if (!s[j].equals("-")) {
                        this.contents[i][j] = Integer.parseInt(s[j]);
                    }
                    else {
```

```
                        this.contents[i][j] = 0;
                        this.emptyLocationRow = i;
                        this.emptyLocationCol = j;
                    }

                }
                line = br.readLine();
            }
        } finally {
            br.close();
        }
    }

    public int countMisplacedTiles() {
        // Menghitung jumlah tile yang tidak berada di posisi yang seharusnya
        int count = 0;
        for (int i = 0; i < 4; i++) {
            for (int j = 0; j < 4; j++) {
                if ((contents[i][j] != 0 && contents[i][j] != i * 4 + j + 1 && i * 4 + j + 1 != 16) ||
                        (i * 4 + j + 1 == 16 && contents[i][j] != 0)) {
                    count++;
                }
            }
        }
        return count;
    }

    public void printBoard() {
        for (int i = 0; i < 4; i++) {
            for (int j = 0; j < 4; j++) {
```

```
                System.out.print(this.contents[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

InfoLabel.java

```
package com.app.elements;

import javafx.geometry.Insets;
import javafx.scene.control.Label;
import javafx.scene.text.Font;

public class InfoLabel extends Label {
    public InfoLabel(String text) {
        this.setText(text);
        this.setPadding(new Insets(5));
        this.setFont(Font.font("verdana", 12));
        this.autosize();
    }
}
```

NumberTile.java

```
package com.app.elements;

import javafx.geometry.Pos;
import javafx.scene.control.Label;
```

```
import javafx.scene.effect.Light;
import javafx.scene.effect.Lighting;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Paint;
import javafx.scene.text.*;

public class NumberTile extends VBox {
    private static final int CELL_SIZE = 75;
    private static final int TILE_SIZE = 60;

    public NumberTile(int number, int row, int col) {
        Light.Distant light = new Light.Distant();
        light.setAzimuth(-135.0);

        Lighting lighting = new Lighting();
        lighting.setLight(light);
        lighting.setSurfaceScale(5.0);

        setMinSize(TILE_SIZE,TILE_SIZE);
        setPrefSize(TILE_SIZE, TILE_SIZE);
        setMaxSize(TILE_SIZE, TILE_SIZE);
        setStyle("-fx-background-color: #C18A54");
        setLayoutX((col + 0.5) * CELL_SIZE - TILE_SIZE / 2);
        setLayoutY((row + 0.5) * CELL_SIZE - TILE_SIZE / 2);
        setAlignment(Pos.CENTER);
        setId(Integer.toString(number));
        setEffect(lighting);

        Label label = new Label();
```

```
        label.setText(Integer.toString(number));

        label.setFont(Font.font("verdana", FontWeight.BOLD,
FontPosture.REGULAR, 30));

        label.setTextFill(Paint.valueOf("#422311"));

        this.getChildren().add(label);

    }

}
```

LINK DRIVE DAN GITHUB

1, Link Drive

https://drive.google.com/drive/folders/1C-YMo0ZqOwMkzofNF9qP3J-iRpv-NPlC?usp=sharing

2. Link Github

https://github.com/ignferry/Tucil-3-Stima-15-Puzzle

CHECKLIST

Poin	Ya	Tidak
1. Program berhasil dikompilasi	✓	
2. Program berhasil running	✓	
3. Program dapat menerima input dan menuliskan output.	✓	
4. Luaran sudah benar untuk semua data uji	✓	
5. Bonus dibuat	✓	