Laporan Tugas Kecil 3

IF2211 Strategi Algoritma

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

oleh

Raden Rifqi Rahman (13520166)

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

BANDUNG

2022

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

Cara kerja aplikasi dalam menyelesaikan 15-puzzle menggunakan algoritma *branch and bound*

Untuk menyelesaikan 15-puzzle, aplikasi melakukan langkah-langkah sebagai berikut.

Dari masukan yang diberikan, bangun sebuah matriks bilangan bulat 4 × 4 yang berisi nilai-nilai masukan. Cek apakah nilai-nilai masukan yang diberikan dapat membangun matriks 15-puzzle yang valid, yaitu matriks yang berisi nilai 1 hingga 15 dan nilai kosong yang masing-masing muncul **tepat 1 kali**. Jika masukan tidak valid, tampilkan pesan kesalahan.

Untuk setiap elemen matriks *i*, hitung nilai l_i (*) dari elemen tersebut dengan nilai kosong dianggap 16. Nilai *l* dikalkulasi sebagai berikut.

l_i = banyak elemen *e* sehingga *e* < *i* dan posisi(*e*) > posisi(*i*)
dengan
posisi(*e*) = indeks baris *e* * 4 + indeks kolom *e*

Selanjutnya, hitung nilai *x* berdasarkan posisi elemen kosong yang dikalkulasi sebagai berikut.

x = (indeks baris elemen kosong + indeks kolom elemen kosong) mod 2

Jika nilai dari $\sum_{i=1}^{16} l_i + x$ adalah ganjil, maka puzzle tidak memiliki solusi dan tampilkan pesan kesalahan.

Selanjutnya, misalkan sebuah *priority queue Q* menyimpan *state-state* puzzle dengan prioritas nilai *cost* terendah dari *state*. Nilai *cost* dari sebuah *state* puzzle dihitung sebagai berikut.

cost(*S*) = kedalaman *state parent S* + banyak sel tak kosong pada S yang tidak sesuai *state goal*

Masukan matriks puzzle *M* ke dalam *Q*. Selama *M* bukan merupakan *goal* dari 15-puzzle, *expand M* dengan cara menggerakan sel kosong pada *M* ke bawah, kanan, kiri, dan atas—jika memungkinkan—sehingga didapatkan *state-state* baru. Simpan *state* yang dibangkitkan oleh *M* dan belum pernah dibangkitkan sebelumnya ke dalam *Q*. Kemudian ambil matriks dari *state* yang berada pada elemen terdepan *Q* dan simpan sebagai *M*.

Jika *M* adalah *goal*, telusuri *parent* dari *M* hingga mencapai *state* awal puzzle yang diberikan. Dengan demikian, akan diperoleh jalur '*path*' *P* dari *state goal* menuju *state* awal puzzle. Selanjutnya, balik *path P* sehingga diperoleh *path P'* dari *state* awal puzzle menuju *state goal*. *Path P'* inilah yang merupakan solusi menuju *state goal* dari 15-puzzle.

* Nilai l_i adalah nilai dari fungsi *KURANG*(*i*).

Source code aplikasi

Berikut adalah source code aplikasi dalam bahasa Java yang telah diminifikasi untuk menyelesaikan 15-puzzle dengan algoritma *branch and bound*.

Application.java

```java
package id.ac.itb.stei.informatika.fifteenp;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.scene.image.Image;
import javafx.stage.Stage;
import java.io.IOException;
public class Application extends javafx.application.Application {
    @Override
    public void start(Stage stage) throws IOException {
        FXMLLoader fxmlLoader = new
FXMLLoader(Application.class.getResource("main.fxml"));
        Scene scene = new Scene(fxmlLoader.load(), 1080, 540);
        stage.setTitle("15-Puzzle Solver");
        stage.setScene(scene);
        stage.getIcons().add(new
Image(getClass().getResourceAsStream("fifteenp.png")));
        stage.show();
    }
    public static void main(String[] args) {
        launch();
    }
}
```

MainController.java

```java
package id.ac.itb.stei.informatika.fifteenp;
import id.ac.itb.stei.informatika.fifteenp.io.FifteenMatrixParser;
import id.ac.itb.stei.informatika.fifteenp.io.FileReader;
import id.ac.itb.stei.informatika.fifteenp.util.Direction;
import id.ac.itb.stei.informatika.fifteenp.util.FifteenMatrix;
import javafx.fxml.FXML;
import javafx.geometry.Pos;
import javafx.scene.control.Alert;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.layout.*;
import javafx.scene.paint.Color;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;
import javafx.scene.text.TextAlignment;
import javafx.stage.FileChooser;
import java.io.File;
import java.util.ArrayList;
import java.util.Random;
public class MainController {
    @FXML
    private FifteenMatrixController matrixController;
    @FXML
    private GridPane mainContainer;
    @FXML
    private Label labelCell0;
    @FXML
    private Label labelCell1;
    @FXML
    private Label labelCell2;
    @FXML
    private Label labelCell3;
```

```
    @FXML
    private Label labelCell4;
    @FXML
    private Label labelCell5;
    @FXML
    private Label labelCell6;
    @FXML
    private Label labelCell7;
    @FXML
    private Label labelCell8;
    @FXML
    private Label labelCell9;
    @FXML
    private Label labelCell10;
    @FXML
    private Label labelCell11;
    @FXML
    private Label labelCell12;
    @FXML
    private Label labelCell13;
    @FXML
    private Label labelCell14;
    @FXML
    private Label labelCell15;
    @FXML
    private Button prevButton;
    @FXML
    private Button nextButton;
    @FXML
    private Button chooseFileButton;
    @FXML
    private Button solveButton;
    @FXML
    private Button randomButton;
    @FXML
    private Label depthLabel;
    @FXML
    private Label stateLabel;
    @FXML
    private Label sumLowerLabel;
    @FXML
    private Label lowerLabel;
    @FXML
    private Label execTimeLabel;
    private Label[] labelCells;
    private ArrayList<FifteenMatrix> solutionPath;
    private ArrayList<Direction> solutionDir;
    private int currentDepth;
    private final Font defaultFont = Font.font("Arial", FontWeight.BOLD, 16);
    private final Font defaultFontSmall = Font.font("Arial", FontWeight.BOLD, 12);
    private final Font defaultFontBig = Font.font("Arial", FontWeight.BOLD, 20);
    private final String[] postfix = {"ns", "us", "ms", "s"};
    @FXML
    private void initialize() {
        this.mainContainer.setBackground(new Background(
                new BackgroundFill(Color.valueOf("#37474f"), null, null)
        ));
        Border border = new Border(
```

```
                new BorderStroke(
                        Color.WHITESMOKE,
                        BorderStrokeStyle.SOLID,
                        CornerRadii.EMPTY,
                        BorderStroke.DEFAULT_WIDTHS
                )
        );
        this.labelCells = new Label[]{
                labelCell0, labelCell1, labelCell2, labelCell3,
                labelCell4, labelCell5, labelCell6, labelCell7,
                labelCell8, labelCell9, labelCell10, labelCell11,
                labelCell12, labelCell13, labelCell14, labelCell15,
        };
        this.depthLabel.setFont(this.defaultFontBig);
        this.nextButton.setFont(this.defaultFont);
        this.prevButton.setFont(this.defaultFont);
        for (Label cell: this.labelCells) {
            cell.setFont(this.defaultFont);
            cell.getStyleClass().add("white");
            cell.setBorder(border);
            cell.getParent().getParent().prefHeight(Double.POSITIVE_INFINITY);
            cell.getParent().getParent().prefWidth(Double.POSITIVE_INFINITY);
            HBox parent = (HBox) cell.getParent();
            VBox grandParent = (VBox) parent.getParent();
            parent.prefHeightProperty().bind(grandParent.heightProperty());
            parent.prefWidthProperty().bind(grandParent.widthProperty());
            cell.prefHeightProperty().bind(parent.heightProperty());
            cell.prefWidthProperty().bind(parent.widthProperty());
            cell.setTextAlignment(TextAlignment.CENTER);
            cell.setAlignment(Pos.CENTER);
        }
        this.execTimeLabel.setFont(this.defaultFontSmall);
        this.stateLabel.setFont(this.defaultFontSmall);
        this.sumLowerLabel.setFont(this.defaultFontBig);
        this.solveButton.setFont(this.defaultFont);
        this.randomButton.setFont(this.defaultFont);
        this.chooseFileButton.setFont(this.defaultFont);
        this.lowerLabel.setFont(this.defaultFontSmall);
        this.randomButton.setStyle("-fx-background-color: transparent;" +
                "-fx-text-fill: white; -fx-cursor: hand;");
        this.randomButton.setBorder(border);
        this.chooseFileButton.setStyle("-fx-background-color: transparent;" +
                "-fx-text-fill: white; -fx-cursor: hand;");
        this.chooseFileButton.setBorder(border);
        this.solveButton.setStyle("-fx-background-color: transparent;" +
                "-fx-text-fill: white; -fx-cursor: hand;");
        this.solveButton.setBorder(border);
        this.nextButton.setStyle("-fx-background-color: transparent;" +
                "-fx-text-fill: white; -fx-cursor: hand;");
        this.nextButton.setBorder(border);
        this.prevButton.setStyle("-fx-background-color: transparent;" +
                "-fx-text-fill: white; -fx-cursor: hand;");
        this.prevButton.setBorder(border);
    }
    @FXML
    protected void onSolve() {
        FifteenMatrix matrix;
        try {
```

```
            matrix = matrixController.parse();
        } catch (IllegalArgumentException ignored) {
            this.alert("The given matrix is invalid.");
            return;
        }
        try {
            this.currentDepth = 0;
            this.displayLowerValues(matrix);
            this.displayLowerValuesSum(matrix.lowerSum());
            FifteenPuzzle solver = new FifteenPuzzle(matrix);
            long start = System.nanoTime();
            solver.solve();
            long end = System.nanoTime();
            this.displayExecutionTime(end - start);
            this.solutionPath = solver.getSolutionPathMatrix();
            this.solutionDir = solver.getSolutionPathDir();
            this.matrixController.setMatrix(this.solutionPath.get(0));
            this.disablePrevButton();
            this.setGeneratedStatesText(solver.generatedStates());
            if (this.solutionDir.size() != 0) {
                this.matrixController.setDirection(this.solutionDir.get(0));
                this.enableNextButton();
            }
            this.setLabelText(1, this.solutionPath.size());
        } catch (IllegalArgumentException ignored) {
            this.alert("This puzzle is unsolvable.");
        }
    }
    @FXML
    protected void onNext() {
        this.enablePrevButton();
        this.currentDepth++;
        this.matrixController.setMatrix(this.solutionPath.get(
                        this.currentDepth));
        if (this.currentDepth == this.solutionDir.size()) {
            this.matrixController.setDirection(null);
        } else {
            this.matrixController.setDirection(this.solutionDir.get(
                    this.currentDepth));
        }
        this.setLabelText(this.currentDepth + 1, this.solutionPath.size());
        if (this.currentDepth + 1 == this.solutionPath.size()) {
            this.disableNextButton();
        } else {
            this.enableNextButton();
        }
    }
    @FXML
    protected void onPrev() {
        this.enableNextButton();
        this.currentDepth--;
        this.matrixController.setMatrix(this.solutionPath.get(
                this.currentDepth));
        this.matrixController.setDirection(this.solutionDir.get(
                this.currentDepth));
        this.setLabelText(this.currentDepth + 1, this.solutionPath.size());
        if (this.currentDepth == 0) {
            this.disablePrevButton();
```

```
        } else {
            this.enablePrevButton();
        }
    }
    @FXML
    protected void onRandom() {
        Random random = new Random();
        int steps = (random.nextInt() % 28 + 28) % 28 + 12;
        FifteenMatrix matrix = FifteenPuzzle.SOLUTION.copy();
        for (int i = 0; i < steps; i++) {
            while (true) {
                int dirIndex = (random.nextInt() % 4 + 4) % 4;
                try {
                    matrix = matrix.moveBlankTile(
                            Direction.DIRECTIONS[dirIndex]
                    );
                    break;
                } catch (IndexOutOfBoundsException ignored) {
                }
            }
        }
        this.matrixController.setMatrix(matrix);
    }
    @FXML
    private void onChooseFile() {
        FileChooser dialog = new FileChooser();
        File input = dialog.showOpenDialog(null);
        if (input != null) {
            String filename = input.toString();

            FileReader reader = new FileReader();
            try {
                reader.readFile(filename);
            } catch (Throwable ignored) {
                this.alert("There was an error reading the file.");
            }

            try {
                String fileContent = reader.result();
                FifteenMatrixParser parser = new FifteenMatrixParser();
                parser.parse(fileContent);
                this.matrixController.setMatrix(parser.result());
            } catch (Throwable ignored) {
                this.alert("The file has an invalid format.");
            }
        }
    }
    private void setLabelText(int depth, int maxDepth) {
        this.depthLabel.setText(depth + "/" + maxDepth);
    }
    private void setGeneratedStatesText(Integer states) {
        this.stateLabel.setText("Generated "
                + states.toString() + " states");
    }
    private void disablePrevButton() {
        prevButton.setDisable(true);
    }
    private void enablePrevButton() {
```

```
        prevButton.setDisable(false);
    }
    private void disableNextButton() {
        nextButton.setDisable(true);
    }
    private void enableNextButton() {
        nextButton.setDisable(false);
    }
    private void displayLowerValues(FifteenMatrix matrix) {
        for (int i = 1; i < 16; i++) {
            Integer l = matrix.lower(i);
            this.labelCells[i - 1].setText(l.toString());
        }
        Integer l = matrix.lowerNull();
        this.labelCells[15].setText(l.toString());
    }
    private void displayExecutionTime(double duration) {
        int i = 0;
        double factor = 1;
        while (i < 3 && duration * factor > 100) {
            factor *= 1e-3;
            i++;
        }
        String durationString = String.format("%.3f", duration * factor);
        String execTime = "Solved in " + durationString
                + " " + this.postfix[i];
        this.execTimeLabel.setText(execTime);
    }
    private void alert(String message) {
        Alert alert = new Alert(Alert.AlertType.ERROR);
        alert.setContentText(message);
        alert.show();
    }
    private void displayLowerValuesSum(Integer sum) {
        this.sumLowerLabel.setText(sum.toString());
    }
}
```

FifteenMatrixController.java

```
package id.ac.itb.stei.informatika.fifteenp;
import id.ac.itb.stei.informatika.fifteenp.util.Direction;
import id.ac.itb.stei.informatika.fifteenp.util.FifteenMatrix;
import id.ac.itb.stei.informatika.fifteenp.util.FifteenMatrixBuilder;
import javafx.fxml.FXML;
import javafx.scene.control.TextArea;
import javafx.scene.layout.*;
import javafx.scene.paint.Color;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;
public class FifteenMatrixController {
    @FXML
    private TextArea cell0;
    @FXML
    private TextArea cell1;
    @FXML
    private TextArea cell2;
    @FXML
```

```
    private TextArea cell3;
    @FXML
    private TextArea cell4;
    @FXML
    private TextArea cell5;
    @FXML
    private TextArea cell6;
    @FXML
    private TextArea cell7;
    @FXML
    private TextArea cell8;
    @FXML
    private TextArea cell9;
    @FXML
    private TextArea cell10;
    @FXML
    private TextArea cell11;
    @FXML
    private TextArea cell12;
    @FXML
    private TextArea cell13;
    @FXML
    private TextArea cell14;
    @FXML
    private TextArea cell15;
    private TextArea[] cells;
    private FifteenMatrix currentMatrix;
    private Border swapBorderStyle;
    private Border defaultBorderStyle;
    @FXML
    public void initialize() {
        this.swapBorderStyle = new Border(new BorderStroke(
                Color.valueOf("#ffb300"),
                BorderStrokeStyle.SOLID,
                CornerRadii.EMPTY,
                BorderStroke.THICK
        ));
        this.cells = new TextArea[]{
                cell0, cell1, cell2, cell3,
                cell4, cell5, cell6, cell7,
                cell8, cell9, cell10, cell11,
                cell12, cell13, cell14, cell15,
        };
        this.defaultBorderStyle = new Border(
                new BorderStroke(
                        Color.WHITESMOKE,
                        BorderStrokeStyle.SOLID,
                        CornerRadii.EMPTY,
                        BorderStroke.DEFAULT_WIDTHS
                )
        );
        for (TextArea cell: this.cells) {
            cell.setFont(
                    Font.font("Arial", FontWeight.BOLD, 24)
            );
            cell.setStyle("-fx-background-color: transparent;" +
                    "-fx-text-fill: whitesmoke;" +
                    "-fx-line-spacing: 0;");
```

```
            cell.setBorder(defaultBorderStyle);
            cell.textProperty().addListener((observable, oldValue, newValue) -> {
                if (!newValue.matches("\\d*")) {
                    cell.setText(oldValue);
                } else if (!newValue.equals("")) {
                    Integer value = Integer.valueOf(newValue);
                    if (value > 15) {
                        cell.setText(oldValue);
                    }
                }
            });
        }
    }
    public void setMatrix(FifteenMatrix matrix) {
        this.currentMatrix = matrix;
        for (int i = 0; i < 4; i++) {
            for (int j = 0; j < 4; j++) {
                TextArea cell = this.cells[i * 4 + j];
                Integer value = matrix.get(i, j);
                if (value == null) {
                    cell.setText("");
                } else {
                    cell.setText(value.toString());
                }
            }
        }
    }
    public void setDirection(Direction dir) {
        for (TextArea cell: this.cells) {
            cell.setBorder(defaultBorderStyle);
        }
        if (dir != null) {
            int blankTileIndex = this.currentMatrix.blankTileIndex();
            int swapIndex = switch (dir) {
                case UP -> blankTileIndex - 4;
                case DOWN -> blankTileIndex + 4;
                case RIGHT -> blankTileIndex + 1;
                case LEFT -> blankTileIndex - 1;
            };
            TextArea blankTileCell = this.cells[blankTileIndex];
            TextArea swapCell = this.cells[swapIndex];
            blankTileCell.setBorder(this.swapBorderStyle);
            swapCell.setBorder(this.swapBorderStyle);
        }
    }
    public FifteenMatrix parse() {
        FifteenMatrixBuilder builder = new FifteenMatrixBuilder();
        for (TextArea cell: cells) {
            Integer value;
            if (cell.getText().equals("")) {
                value = null;
            } else {
                value = Integer.valueOf(cell.getText());
            }
            builder.append(value);
        }
        return builder.build();
```

```
    }
}
```

FifteenPuzzle.java

```
package id.ac.itb.stei.informatika.fifteenp;
import id.ac.itb.stei.informatika.fifteenp.util.Direction;
import id.ac.itb.stei.informatika.fifteenp.util.FifteenMatrix;
import id.ac.itb.stei.informatika.fifteenp.util.FifteenMatrixBuilder;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.PriorityQueue;
public class FifteenPuzzle {
    private FifteenMatrix puzzle;
    private PriorityQueue<FifteenMatrixNode> queue;
    private ArrayList<FifteenMatrixNode> evaluatedStates;
    private ArrayList<FifteenMatrix> solutionPathMatrix;
    private ArrayList<Direction> solutionPathDir;
    public static final FifteenMatrix SOLUTION
        = new FifteenMatrixBuilder()
            .append(1).append(2).append(3).append(4)
            .append(5).append(6).append(7).append(8)
            .append(9).append(10).append(11).append(12)
            .append(13).append(14).append(15).append(null)
            .build();
    public static final long SOLUTION_ID
            = FifteenPuzzle.SOLUTION.identity();
    public FifteenPuzzle(FifteenMatrix puzzle) {
        this.puzzle = puzzle;
        this.queue = new
PriorityQueue<>(Comparator.comparingInt(FifteenMatrixNode::cost));
        this.evaluatedStates = new ArrayList<>();
        this.solutionPathMatrix = new ArrayList<>();
        this.solutionPathDir = new ArrayList<>();
    }
    public void solve() {
        this.checkSolvability();
        this.setup();
        FifteenMatrixNode state = this.queue.poll();
        while (state.matrixId() != FifteenPuzzle.SOLUTION_ID) {
            this.expand(state);
            state = this.queue.poll();
        }
        this.evaluatedStates.add(state);
        this.buildSolution();
    }
    private void checkSolvability() throws IllegalArgumentException {
        boolean solvable = this.puzzle.lowerSum() % 2 == 0;
        if (!solvable) {
            throw new IllegalArgumentException();
        }
    }
    private void setup() {
        this.queue.add(new FifteenMatrixNode(
                this.puzzle.identity(),
                null,
                null,
```

```
                1,
                1
        ));
    }
    private boolean has(long stateMatrixId) {
        for (FifteenMatrixNode node: this.evaluatedStates) {
            if (node.matrixId() == stateMatrixId) {
                return true;
            }
        }
        for (FifteenMatrixNode node: this.queue) {
            if (node.matrixId() == stateMatrixId) {
                return true;
            }
        }
        return false;
    }
    private void expand(FifteenMatrixNode state) {
        for (Direction dir: Direction.DIRECTIONS) {
            if (dir.flip() == state.dir()) {
                continue;
            }
            try {
                FifteenMatrix newMatrix = FifteenMatrix.from(state.matrixId())
                        .moveBlankTile(dir);
                long newMatrixId = newMatrix.identity();
                if (!this.has(newMatrixId)) {
                    FifteenMatrixNode node = new FifteenMatrixNode(
                            newMatrixId,
                            state,
                            dir,
                            state.depth() + 1,
                            state.depth() + newMatrix.mismatchedTiles()
                    );
                    this.queue.offer(
                        node
                    );
                }
            } catch (IndexOutOfBoundsException ignored) {
            }
        }
        this.evaluatedStates.add(state);
    }
    private void buildSolution() {
        FifteenMatrixNode node = this.evaluatedStates.get(
                this.evaluatedStates.size() - 1
        );
        while (node.parent() != null) {
            this.solutionPathMatrix.add(FifteenMatrix.from(
                    node.matrixId()
            ));
            this.solutionPathDir.add(node.dir());
            node = node.parent();
        }
        this.solutionPathMatrix.add(FifteenMatrix.from(
                node.matrixId()
        ));
        Collections.reverse(this.solutionPathMatrix);
```

```
        Collections.reverse(this.solutionPathDir);
    }
    public ArrayList<FifteenMatrix> getSolutionPathMatrix() {
        return this.solutionPathMatrix;
    }
    public ArrayList<Direction> getSolutionPathDir() {
        return this.solutionPathDir;
    }
    public int generatedStates() {
        return this.queue.size() + this.evaluatedStates.size();
    }
}
record FifteenMatrixNode(long matrixId,
                         FifteenMatrixNode parent,
                         Direction dir,
                         int depth,
                         int cost) {
}
```

util/FifteenMatrix.java

```
package id.ac.itb.stei.informatika.fifteenp.util;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Objects;
public class FifteenMatrix extends Matrix<Integer> {
    public FifteenMatrix() {
        super(4, 4);
    }
    public FifteenMatrix copy() {
        FifteenMatrixBuilder builder = new FifteenMatrixBuilder();
        for (int i = 0; i < 4; i++) {
            for (int j = 0; j < 4; j++) {
                Integer elem = this.values.get(i).get(j);
                builder.append(elem);
            }
        }
        return builder.build();
    }
    public FifteenMatrix moveBlankTile(Direction dir) {
        switch (dir) {
            case UP:
                return this.up();
            case DOWN:
                return this.down();
            case RIGHT:
                return this.right();
            case LEFT:
            default:
                return this.left();
        }
    }
    public FifteenMatrix up() throws IndexOutOfBoundsException {
        int blankTileIndex = this.blankTileIndex();
        if (blankTileIndex < 4) {
            throw new IndexOutOfBoundsException();
        }
        FifteenMatrix newMatrix = this.copy();
```

```
        newMatrix.swap(blankTileIndex, blankTileIndex - 4);
        return newMatrix;
    }
    public FifteenMatrix down() throws IndexOutOfBoundsException {
        int blankTileIndex = this.blankTileIndex();
        if (blankTileIndex > 12) {
            throw new IndexOutOfBoundsException();
        }
        FifteenMatrix newMatrix = this.copy();
        newMatrix.swap(blankTileIndex, blankTileIndex + 4);
        return newMatrix;
    }
    public FifteenMatrix right() throws IndexOutOfBoundsException {
        int blankTileIndex = this.blankTileIndex();
        if (blankTileIndex % 4 == 3) {
            throw new IndexOutOfBoundsException();
        }
        FifteenMatrix newMatrix = this.copy();
        newMatrix.swap(blankTileIndex, blankTileIndex + 1);
        return newMatrix;
    }
    public FifteenMatrix left() throws IndexOutOfBoundsException {
        int blankTileIndex = this.blankTileIndex();
        if (blankTileIndex % 4 == 0) {
            throw new IndexOutOfBoundsException();
        }
        FifteenMatrix newMatrix = this.copy();
        newMatrix.swap(blankTileIndex, blankTileIndex - 1);
        return newMatrix;
    }
    @Deprecated
    public boolean equals(FifteenMatrix other) {
        for (int i = 0; i < 4; i++) {
            for (int j = 0; j < 4; j++) {
                if (!Objects.equals(this.values.get(i).get(j),
                    other.values.get(i).get(j))) {
                    return false;
                }
            }
        }
        return true;
    }
    public int lowerSum() {
        int blankTileIndex = this.blankTileIndex();
        int factor = (blankTileIndex / 4 + blankTileIndex % 4) % 2;
        int sum = 0;
        for (int i = 1; i < 16; i++) {
            sum += this.lower(i);
        }
        return sum + this.lowerNull() + factor;
    }
    public int lower(Integer value) throws IllegalArgumentException {
        if (value == null) {
            return this.lowerNull();
        }
        if (value < 1 || value > 15) {
            throw new IllegalArgumentException();
        }
```

```
        int opposite = 0;
        iteration:
        for (int i = 0; i < 4; i++) {
            for (int j = 0; j < 4; j++) {
                Integer elem = this.values.get(i).get(j);
                if (elem != null && elem < value) {
                    opposite++;
                }
                if (elem != null && elem == value) {
                    break iteration;
                }
            }
        }
        return value - 1 - opposite;
    }
    public int lowerNull() {
        int opposite = 0;
        iteration:
        for (int i = 0; i < 4; i++) {
            for (int j = 0; j < 4; j++) {
                Integer elem = this.values.get(i).get(j);
                if (elem != null) {
                    opposite++;
                } else {
                    break iteration;
                }
            }
        }
        return 15 - opposite;
    }
    public int mismatchedTiles() {
        int mismatches = 0;
        for (int i = 0; i < 4; i++) {
            for (int j = 0; j < 4; j++) {
                Integer elem = this.values.get(i).get(j);
                if (elem != null && elem != 4 * i + j + 1) {
                    mismatches++;
                }
            }
        }
        return mismatches;
    }
    public int blankTileIndex() {
        for (int i = 0; i < 4; i++) {
            for (int j = 0; j < 4; j++) {
                if (this.values.get(i).get(j) == null) {
                    return 4 * i + j;
                }
            }
        }
        return -1;
    }
    private void swap(int firstIndex, int secondIndex)
            throws IndexOutOfBoundsException {
        if (firstIndex < 0 || firstIndex > 15
        || secondIndex < 0 || secondIndex > 15) {
            throw new IndexOutOfBoundsException();
        }
```

```
        Integer a = this.values.get(firstIndex / 4)
                .get(firstIndex % 4);
        Integer b = this.values.get(secondIndex / 4)
                .get(secondIndex % 4);
        this.values.get(firstIndex / 4)
                .set(firstIndex % 4, b);
        this.values.get(secondIndex / 4)
                .set(secondIndex % 4, a);
    }
    public static FifteenMatrix from(long id) {
        ArrayList<Integer> values = new ArrayList<>();
        for (int i = 0; i < 4; i++) {
            for (int j = 0; j < 4; j++) {
                int value = (int) (((id % 16) + 16) % 16);
                if (value == 0) {
                    values.add(null);
                } else {
                    values.add(value);
                }
                id >>= 4;
            }
        }
        Collections.reverse(values);
        FifteenMatrixBuilder builder = new FifteenMatrixBuilder();
        for (Integer value: values) {
            builder.append(value);
        }
        return builder.build();
    }
    public long identity() {
        long id = 0;
        for (int i = 0; i < 4; i++) {
            for (int j = 0; j < 4; j++) {
                id <<= 4;
                Integer elem = this.values.get(i).get(j);
                if (elem != null) {
                    id += elem;
                }
            }
        }
        return id;
    }
}
```

util/Matrix.java

```
package id.ac.itb.stei.informatika.fifteenp.util;
import java.util.ArrayList;
public class Matrix<T> {
    protected final int rows;
    protected final int cols;
    ArrayList<ArrayList<T>> values;
    public Matrix(int rows, int cols) {
        this.rows = rows;
        this.cols = cols;
        this.values = new ArrayList<>(this.rows);
```

```
        for (int i = 0; i < this.rows; i++) {
            ArrayList<T> row = new ArrayList<>(this.cols);
            for (int j = 0; j < this.cols; j++) {
                row.add(null);
            }
            this.values.add(row);
        }
    }
    @Override
    public String toString() {
        String res = "";
        for (int i = 0; i < this.rows; i++) {
            for (int j = 0; j < this.cols; j++) {
                if (j != 0) {
                    res += " ";
                }
                res += this.get(i, j).toString();
            }
            res += "\n";
        }
        return res;
    }
    public int rows() {
        return this.rows;
    }
    public int cols() {
        return this.cols;
    }
    public boolean hasIndices(int rowIndex, int colIndex) {
        return rowIndex >= 0 && rowIndex < this.rows &&
                colIndex >= 0 && colIndex < this.cols;
    }
    public void set(int rowIndex, int colIndex, T element) {
        this.values.get(rowIndex).set(colIndex, element);
    }
    public T get(int rowIndex, int colIndex) {
        return this.values.get(rowIndex).get(colIndex);
    }
}
```

util/FifteenMatrixBuilder.java

```
package id.ac.itb.stei.informatika.fifteenp.util;
import java.util.ArrayList;
public class FifteenMatrixBuilder {
    private Integer[] values;
    private int cursor;
    public FifteenMatrixBuilder() {
        this.values = new Integer[16];
        for (int i = 0; i < 16; i++) {
            this.values[i] = null;
        }
        this.cursor = 0;
    }
    public FifteenMatrixBuilder append(Integer value)
        throws IllegalArgumentException, IndexOutOfBoundsException {
        if (value != null && (value < 0 || value > 15)) {
            throw new IllegalArgumentException();
```

```
        }
        if (this.cursor > 16) {
            throw new IndexOutOfBoundsException();
        }
        this.values[this.cursor] = value;
        this.cursor++;
        return this;
    }
    public FifteenMatrix build() throws IllegalArgumentException {
        FifteenMatrix puzzle = new FifteenMatrix();
        ArrayList<Boolean> flags = new ArrayList<>();
        int count = 16;
        while (count-- != 0) {
            flags.add(false);
        }
        for (int i = 0; i < 16; i++) {
            int flagIndex;
            if (this.values[i] == null) {
                flagIndex = 15;
            } else {
                flagIndex = this.values[i] - 1;
            }
            int row = i / 4;
            int col = i % 4;
            puzzle.set(row, col, this.values[i]);
            if (flags.get(flagIndex)) {
                throw new IllegalArgumentException();
            } else {
                flags.set(flagIndex, true);
            }
        }
        return puzzle;
    }
}
```

util/Direction.java

```
package id.ac.itb.stei.informatika.fifteenp.util;
public enum Direction {
    UP,
    DOWN,
    RIGHT,
    LEFT;
    public static final Direction[] DIRECTIONS = {
            Direction.DOWN,
            Direction.RIGHT,
            Direction.LEFT,
            Direction.UP,
    };
    public Direction flip() {
        return switch (this) {
            case UP -> Direction.DOWN;
            case DOWN -> Direction.UP;
            case RIGHT -> Direction.LEFT;
            case LEFT -> Direction.RIGHT;
        };
    }
}
```

io/FileReader.java

```java
package id.ac.itb.stei.informatika.fifteenp.io;
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
public class FileReader {
    private String result = "";
    public FileReader() {
    }
    public boolean readFile(String path) {
        this.result = "";
        try {
            File file = new File(path);
            Scanner reader = new Scanner(file);
            if (reader.hasNextLine()) {
                this.result += reader.nextLine();
            }
            while (reader.hasNextLine()) {
                this.result += "\n" + reader.nextLine();
            }
        } catch (FileNotFoundException e) {
            return false;
        }
        return true;
    }
    public String result() {
        return this.result;
    }
}
```

io/FifteenMatrixParser.java

```java
package id.ac.itb.stei.informatika.fifteenp.io;
import id.ac.itb.stei.informatika.fifteenp.util.FifteenMatrix;
import id.ac.itb.stei.informatika.fifteenp.util.FifteenMatrixBuilder;
public class FifteenMatrixParser {
    private FifteenMatrix result;
    private FifteenMatrixBuilder builder;
    public FifteenMatrixParser() {
        this.builder = new FifteenMatrixBuilder();
    }
    public void parse(String input) {
        String[] lines = input.split("\n");
        if (lines.length != 4) {
            throw new IllegalArgumentException();
        }
        for (String line: lines) {
            this.parseLine(line);
        }
        this.result = builder.build();
    }
    public FifteenMatrix result() {
        return this.result;
    }
    private void parseLine(String line) throws IllegalArgumentException {
        String[] values = line.split(" ");
```

```
        if (values.length != 4) {
            throw new IllegalArgumentException();
        }
        for (String value: values) {
            this.parseValue(value);
        }
    }
    private void parseValue(String valueString) throws IllegalArgumentException {
        if (valueString.equals("")) {
            throw new IllegalArgumentException();
        }

        if (valueString.equals("-")) {
            builder.append(null);
        } else {
            Integer value = Integer.valueOf(valueString);
            builder.append(value);
        }
    }
}
```

Instansi-instansi persoalan 15-puzzle untuk pengujian

Berikut adalah instansi-instansi 15-puzzle yang digunakan dalam melakukan pengujian dalam bentuk file teks dan gambar.

(*Catatan : Seluruh file teks yang digunakan dalam melakukan pengujian dapat dilihat pada laman test.*)

13.txt

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

Gambar 1. Instansi 15-puzzle yang direpresentasikan file 13.txt.

16.txt

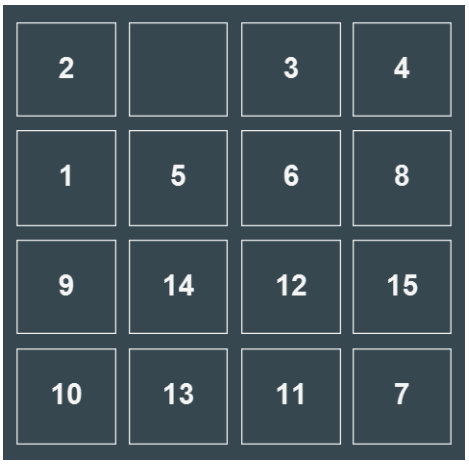
```
2 - 3 4
1 5 6 8
```

```
14 13 7 12
9 11 10 15
```

Gambar 2. Instansi 15-puzzle yang direpresentasikan file 16.txt.

20.txt

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



Gambar 3. Instansi 15-puzzle yang direpresentasikan file 20.txt.

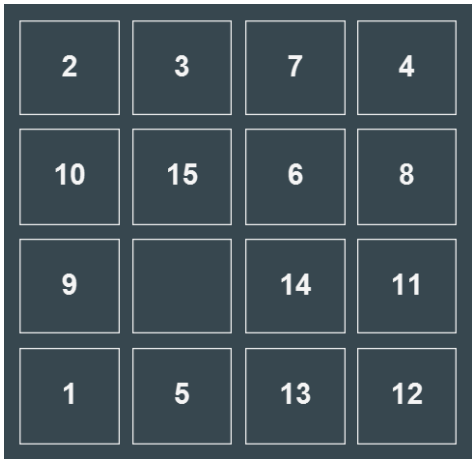
25.txt

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

Gambar 4. Instansi 15-puzzle yang direpresentasikan file 25.txt.

26.txt

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



Gambar 5. Instansi 15-puzzle yang direpresentasikan file 26.txt.

unsolvable1.txt

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

Gambar 6. Instansi 15-puzzle yang direpresentasikan file unsolvable1.txt.

unsolvable2.txt

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

Gambar 7. Instansi 15-puzzle yang direpresentasikan file unsolvable2.txt.

Contoh penggunaan aplikasi

Aplikasi tersebut telah diuji dengan 9 kasus uji, yaitu:

1) 5 kasus uji dari file masukan dengan masing-masing memiliki solusi pada kedalaman 13, 16, 20, 25, dan 26;

2) 1 kasus uji acak '*randomized*' dengan puzzle yang dibangkitkan '*generated*' oleh aplikasi;

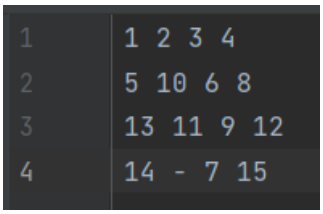
3) 2 kasus uji *puzzle* yang tidak memiliki solusi; dan

4) 1 kasus uji matriks *puzzle* yang tidak valid.

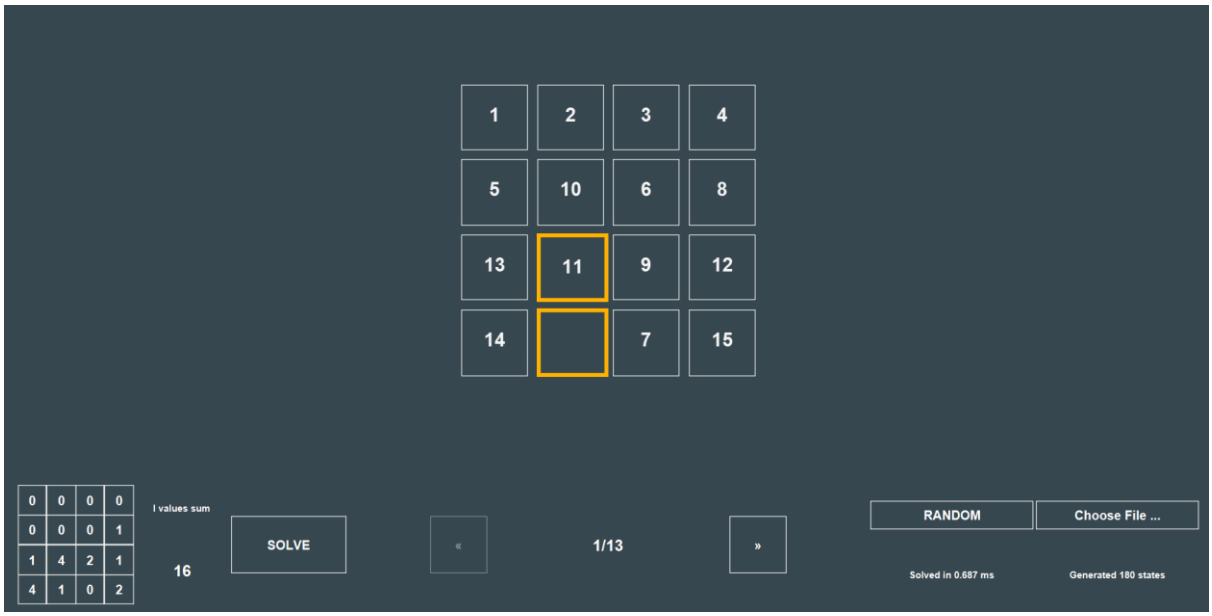
Berikut adalah *input* dan *output* dalam menjalankan aplikasi.

(*Catatan : Seluruh tampilan matriks keluaran aplikasi dalam melakukan pengujian dapat dilihat pada laman demo dalam bentuk animasi GIF.*)

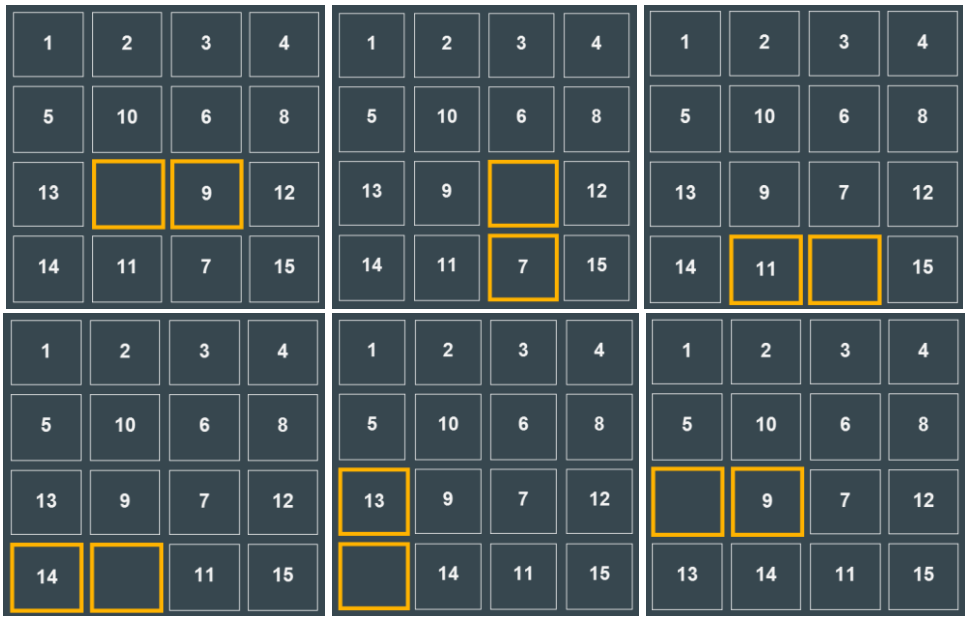
1. *Puzzle* yang *solvable* dengan kedalaman solusi 13



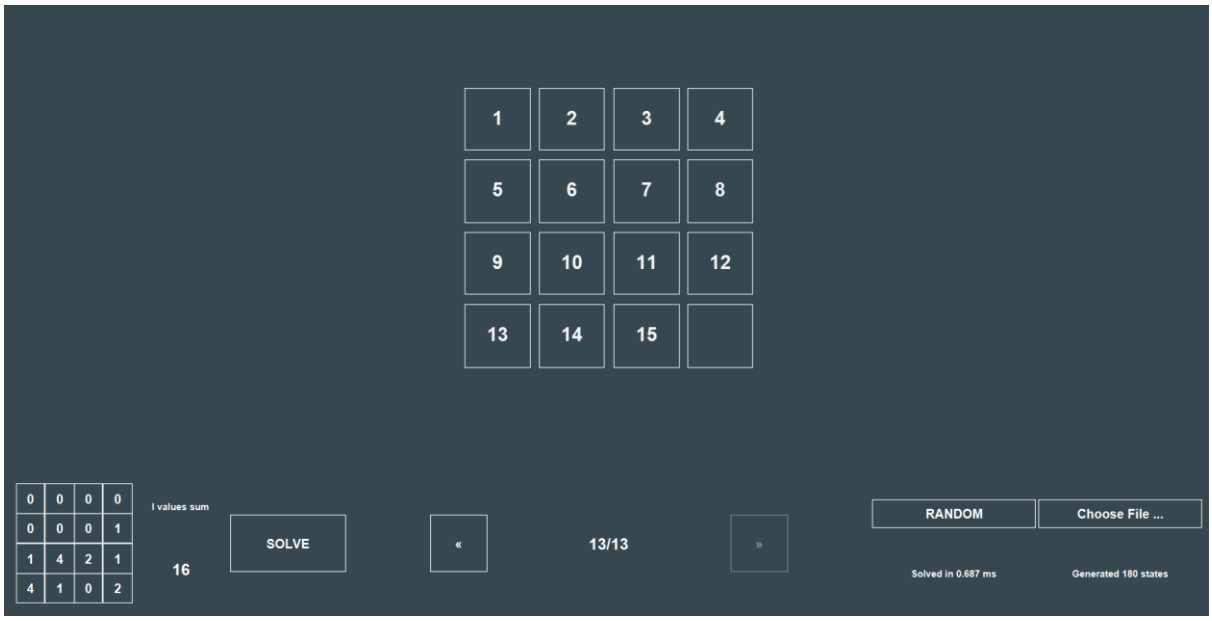
Gambar 8. *Input puzzle* yang *solvable* dengan kedalaman solusi 13.



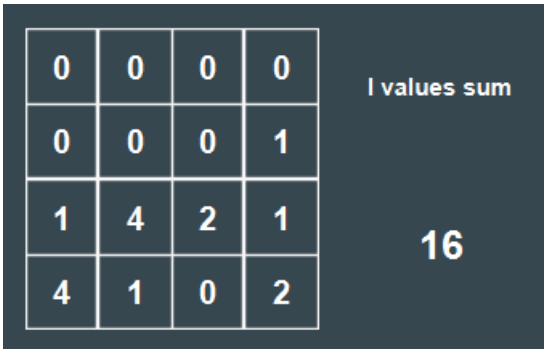
Gambar 9. *State* awal *puzzle* yang *solvable* dengan kedalaman solusi 13.



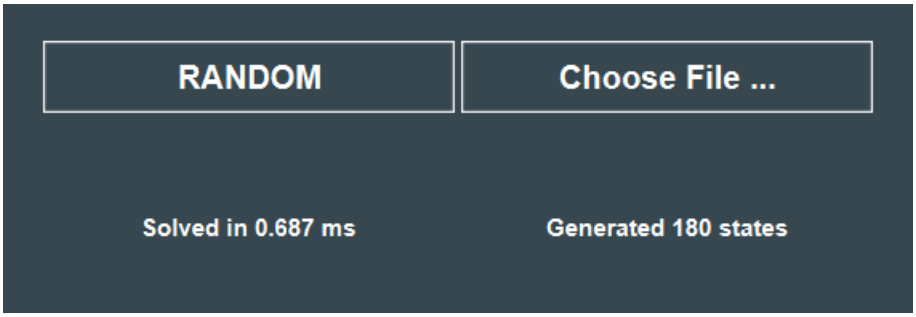
Gambar 10. *State* antara *puzzle* yang *solvable* dengan kedalaman solusi 13.



Gambar 11. *State* akhir *puzzle* yang *solvable* dengan kedalaman solusi 13.

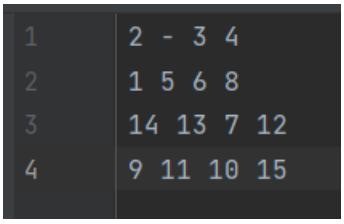


Gambar 12. *Output* nilai *l* untuk setiap elemen matriks dan jumlah seluruh nilai *l* dengan nilai *x*.

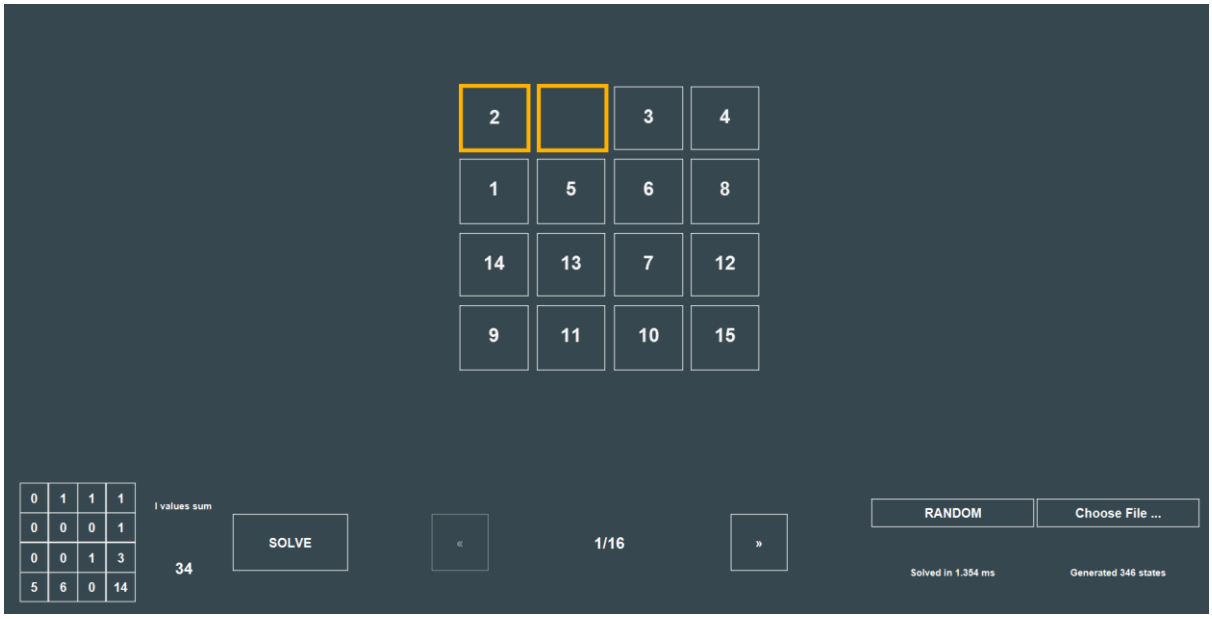


Gambar 13. *Output* lama eksekusi dan banyak *state* yang dibangkitkan pada *puzzle*.

2. *Puzzle* yang *solvable* dengan kedalaman solusi 16

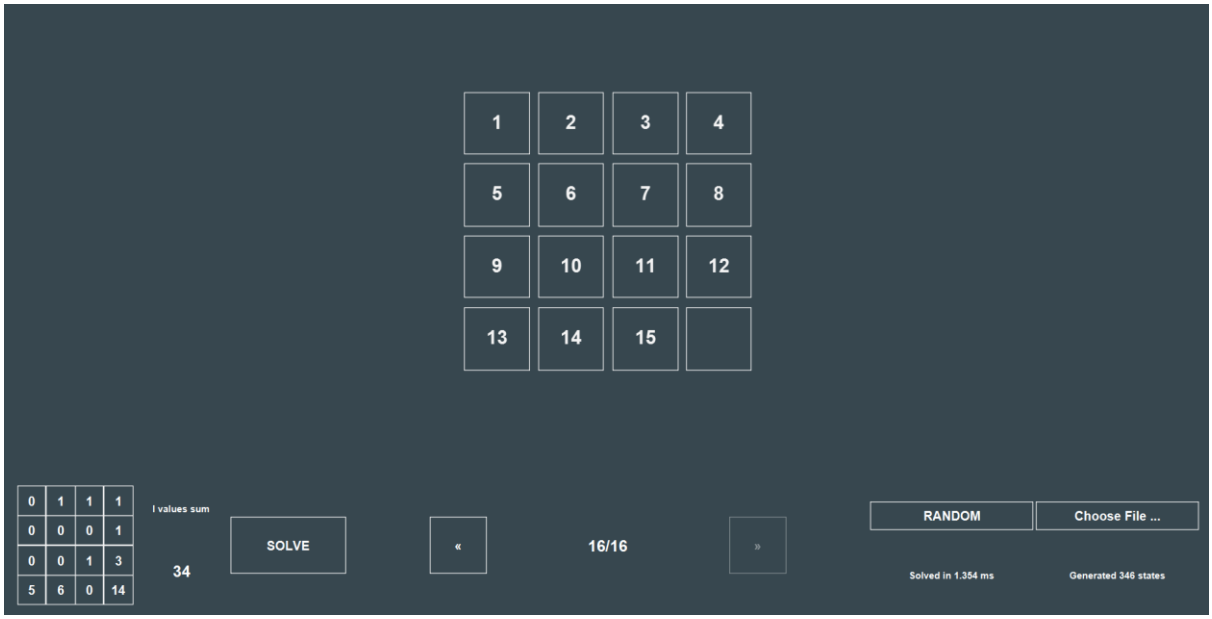


Gambar 14. *Input puzzle* yang *solvable* dengan kedalaman solusi 16.

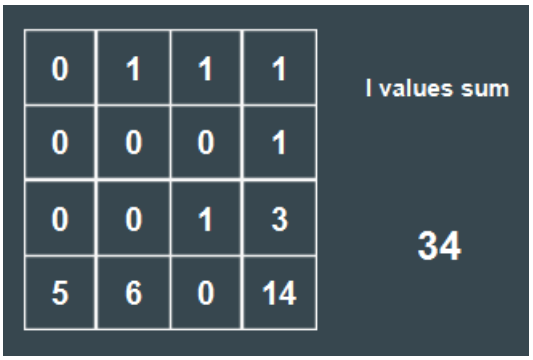


Gambar 15. *State* awal *puzzle* yang *solvable* dengan kedalaman solusi 16.

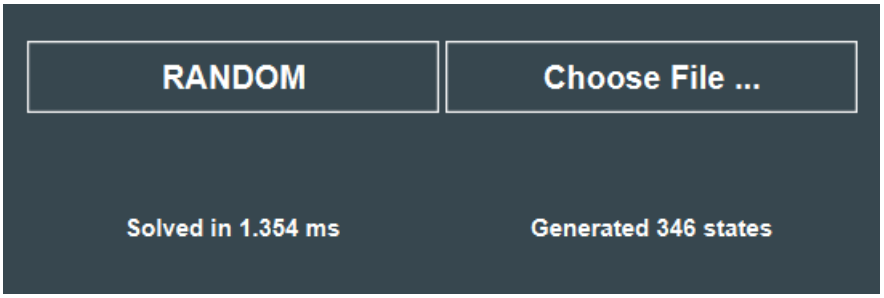
Gambar 16. *State* antara *puzzle* yang *solvable* dengan kedalaman solusi 16.



Gambar 17. *State* akhir *puzzle* yang *solvable* dengan kedalaman solusi 16.

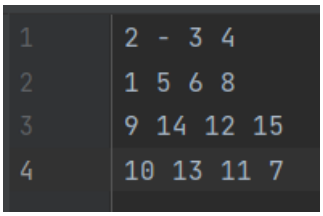


Gambar 18. *Output* nilai *l* untuk setiap elemen matriks dan jumlah seluruh nilai *l* dengan nilai *x*.

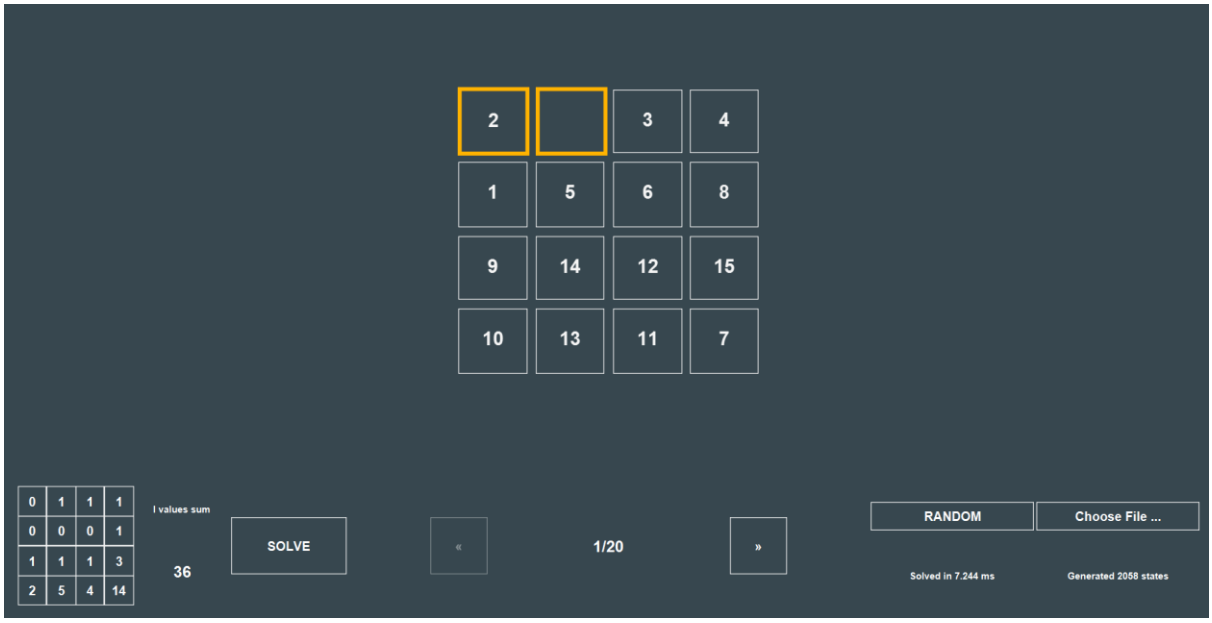


Gambar 19. *Output* lama eksekusi dan banyak *state* yang dibangkitkan pada *puzzle*.

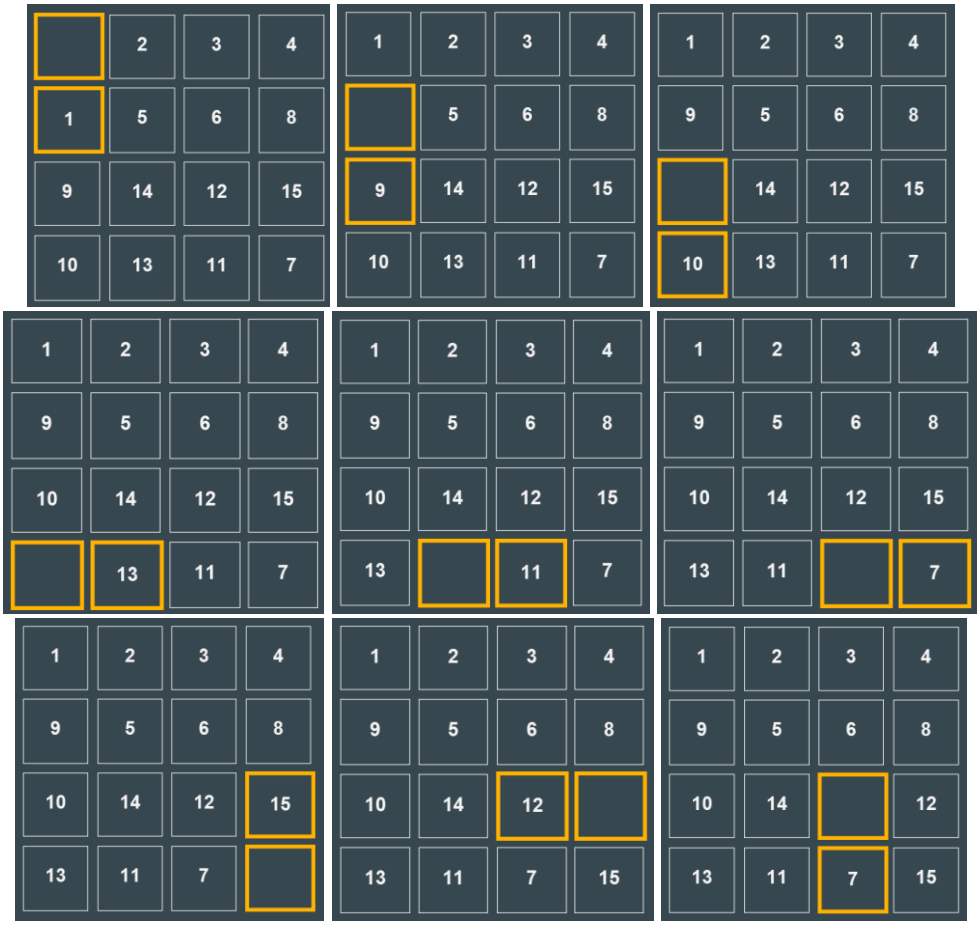
3. *Puzzle* yang *solvable* dengan kedalaman solusi 20



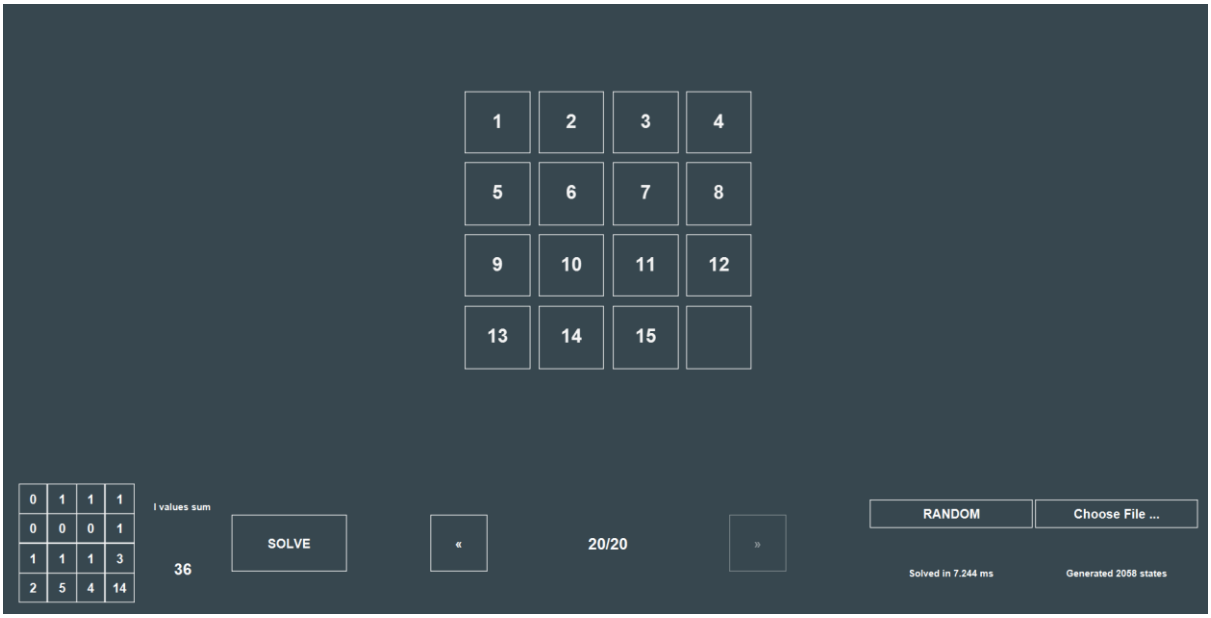
Gambar 20. *Input puzzle* yang *solvable* dengan kedalaman solusi 20.



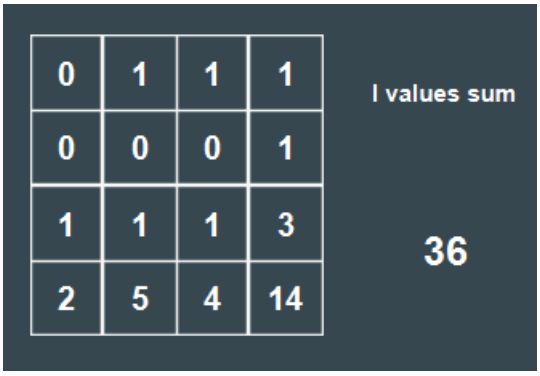
Gambar 21. *State* awal *puzzle* yang *solvable* dengan kedalaman solusi 20.



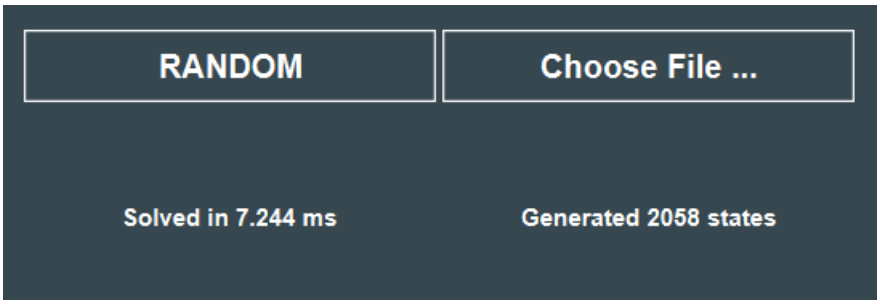
Gambar 22. *State* antara *puzzle* yang *solvable* dengan kedalaman solusi 20.



Gambar 23. *State* akhir *puzzle* yang *solvable* dengan kedalaman solusi 20.

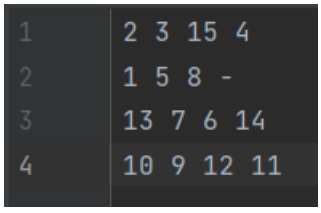


Gambar 24. *Output* nilai *l* untuk setiap elemen matriks dan jumlah seluruh nilai *l* dengan nilai *x*.

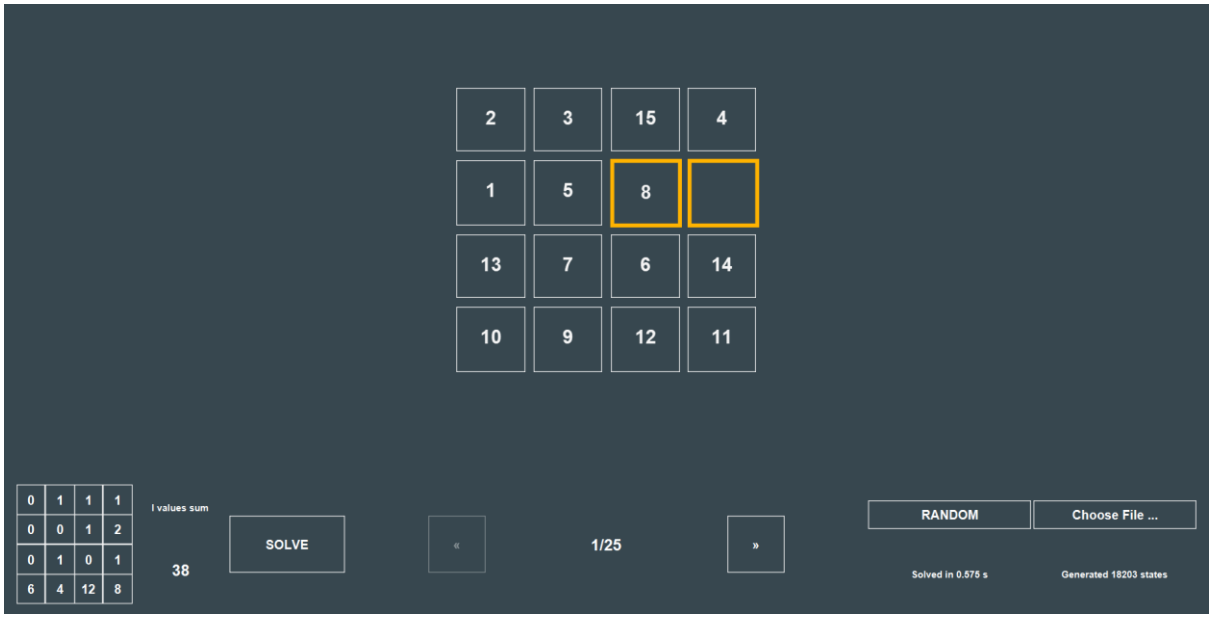


Gambar 25. *Output* lama eksekusi dan banyak *state* yang dibangkitkan pada *puzzle*.

4. *Puzzle* yang *solvable* dengan kedalaman solusi 25



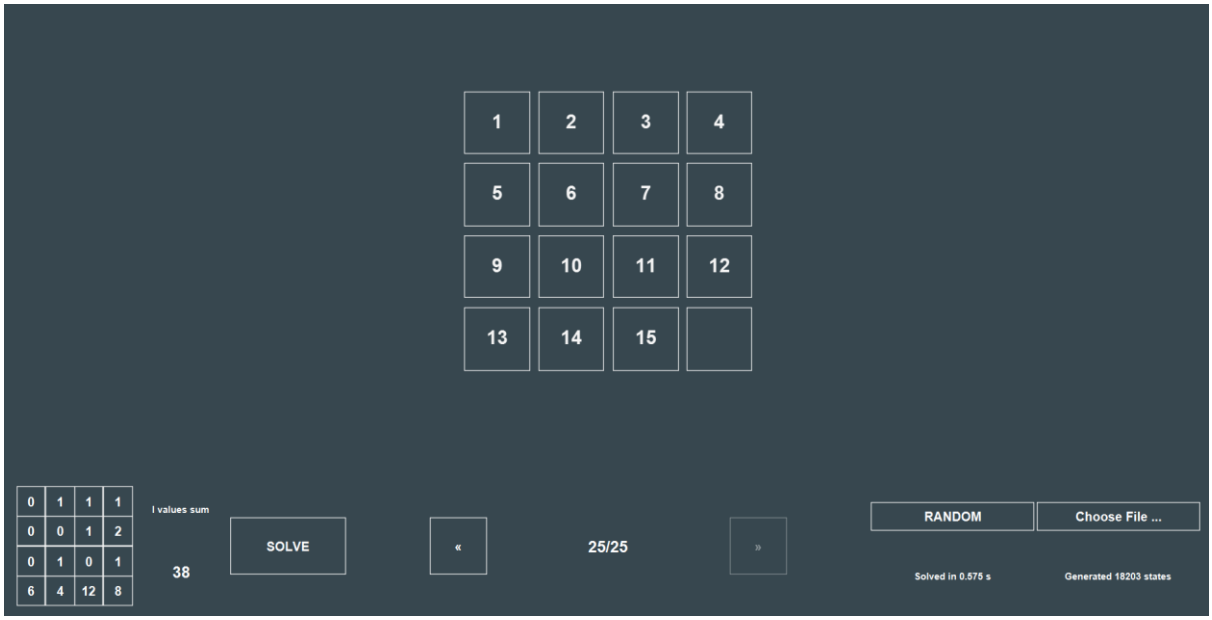
Gambar 26. *Input puzzle* yang *solvable* dengan kedalaman solusi 25.



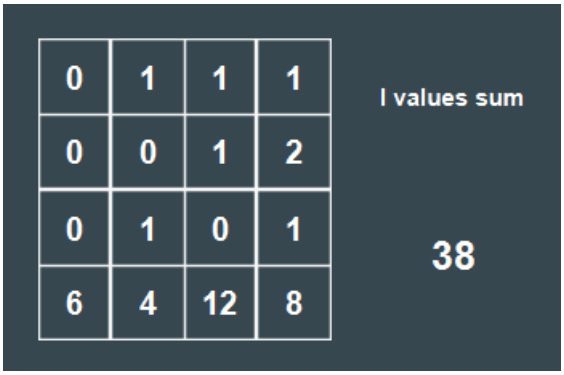
Gambar 27. *State* awal *puzzle* yang *solvable* dengan kedalaman solusi 25.


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

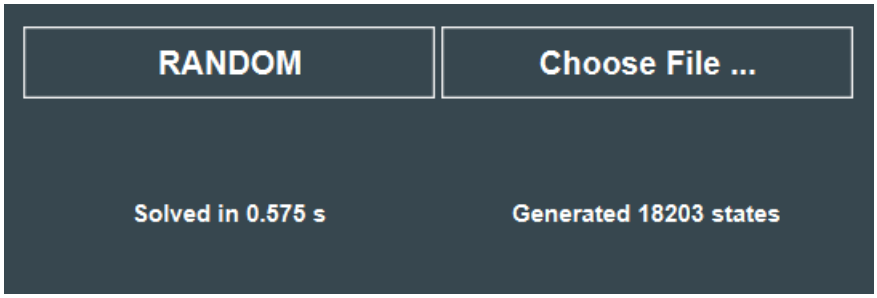
Gambar 28. *State* antara *puzzle* yang *solvable* dengan kedalaman solusi 25.



Gambar 29. *State* akhir *puzzle* yang *solvable* dengan kedalaman solusi 25.

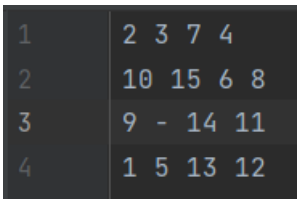


Gambar 30. *Output* nilai *l* untuk setiap elemen matriks dan jumlah seluruh nilai *l* dengan nilai *x*.

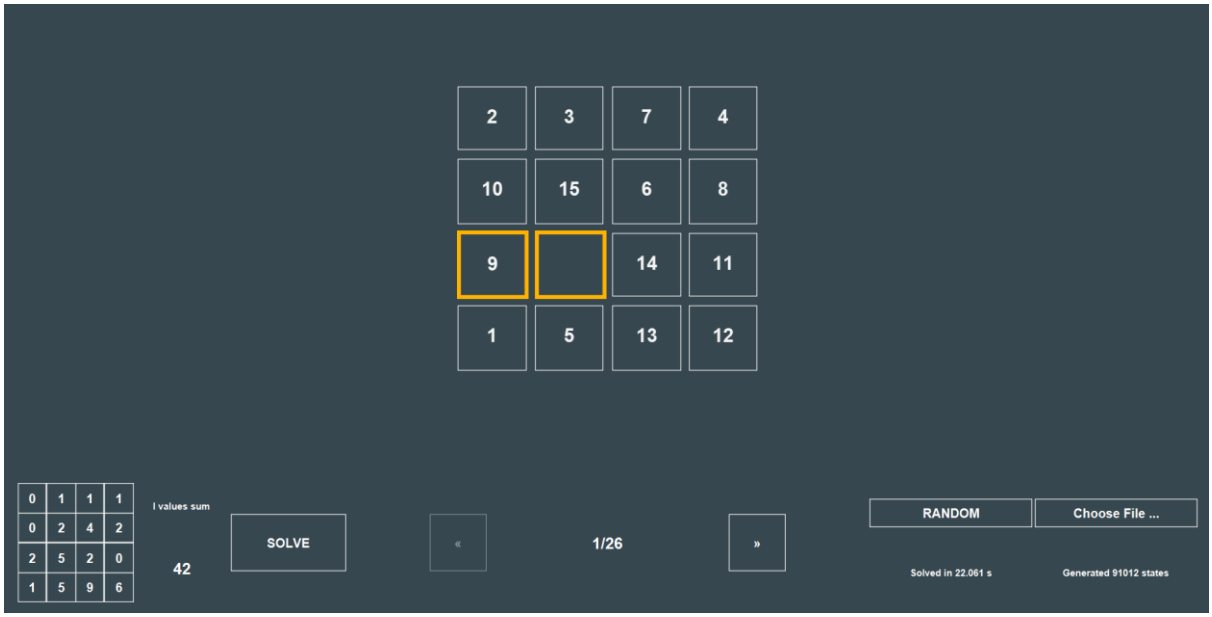


Gambar 31. *Output* lama eksekusi dan banyak *state* yang dibangkitkan pada *puzzle*.

5. *Puzzle* yang *solvable* dengan kedalaman solusi 26



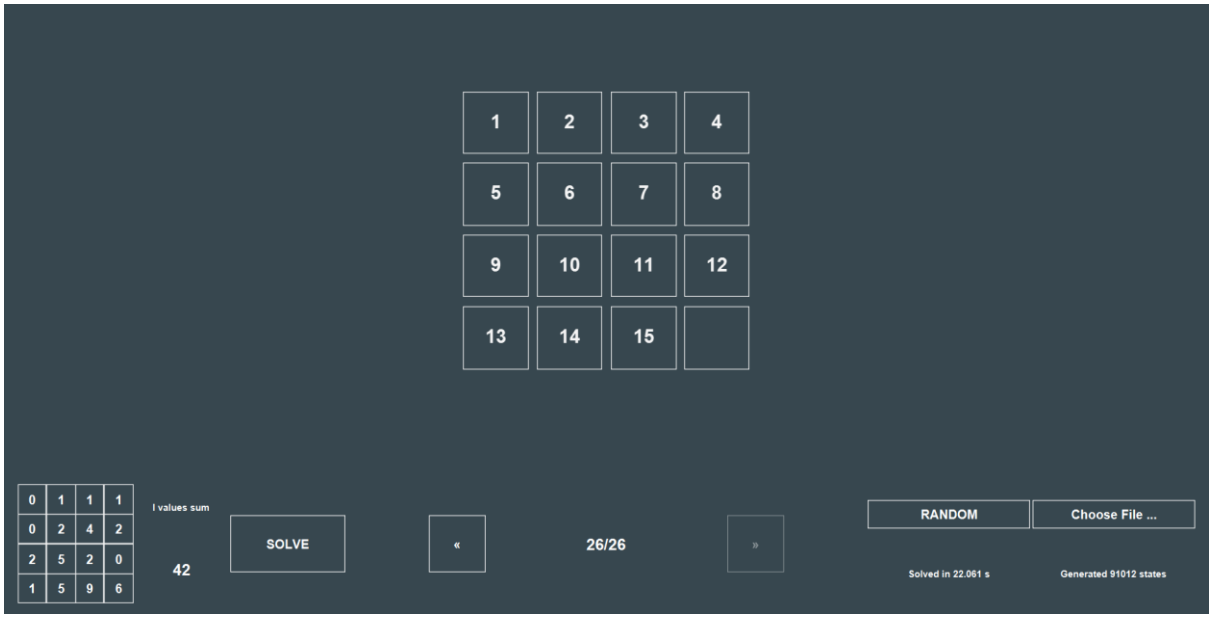
Gambar 32. *Input puzzle* yang *solvable* dengan kedalaman solusi 26.



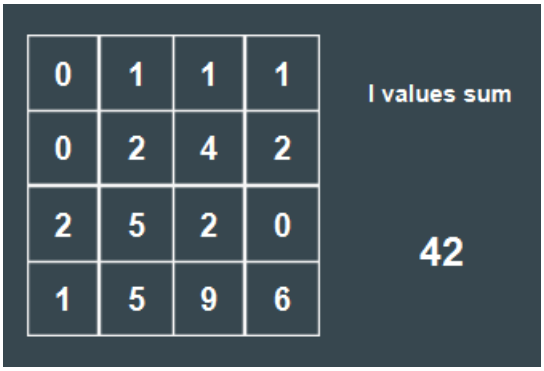
Gambar 33. *State* awal *puzzle* yang *solvable* dengan kedalaman solusi 26.


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

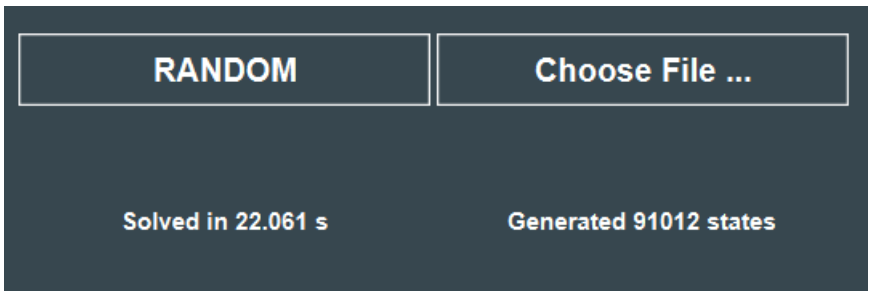
Gambar 34. *State* antara *puzzle* yang *solvable* dengan kedalaman solusi 26.



Gambar 35. *State* akhir *puzzle* yang *solvable* dengan kedalaman solusi 26.

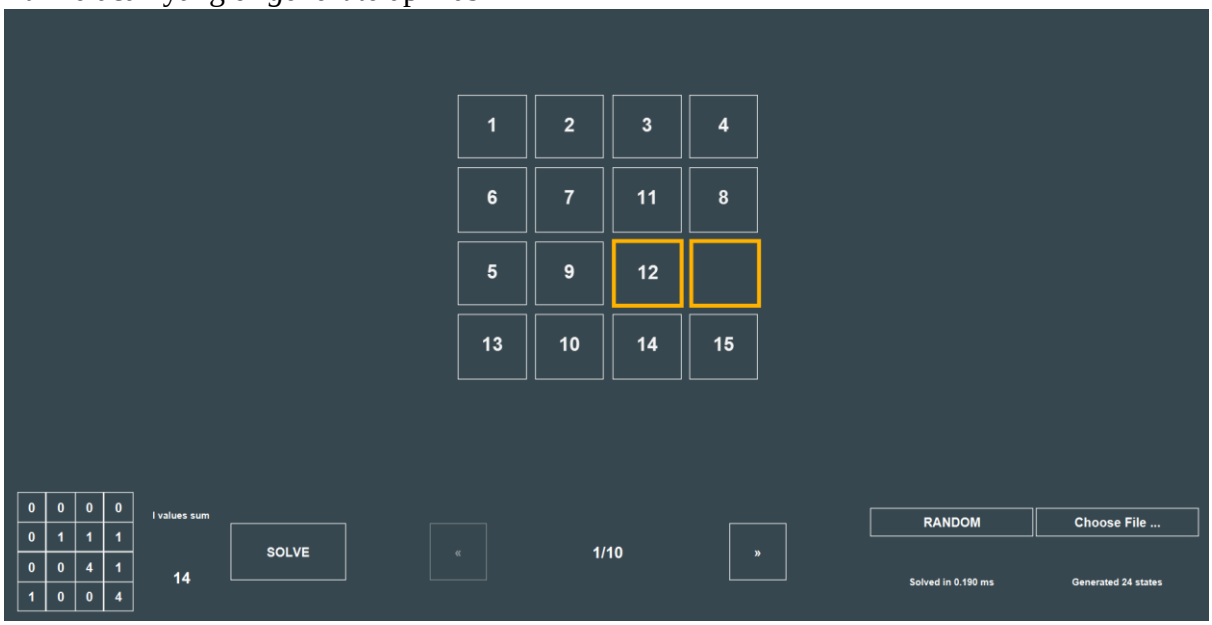


Gambar 36. *Output* nilai *l* untuk setiap elemen matriks dan jumlah seluruh nilai *l* dengan nilai *x*.



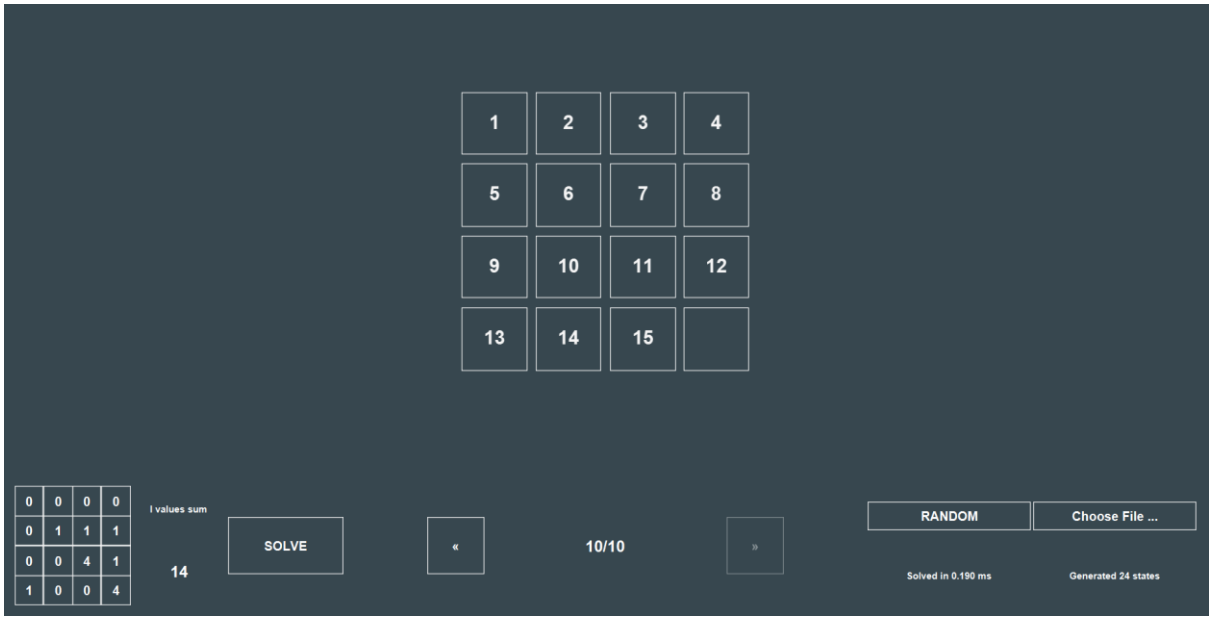
Gambar 37. *Output* lama eksekusi dan banyak *state* yang dibangkitkan pada *puzzle*.

6. *Puzzle* acak yang di-*generate* aplikasi

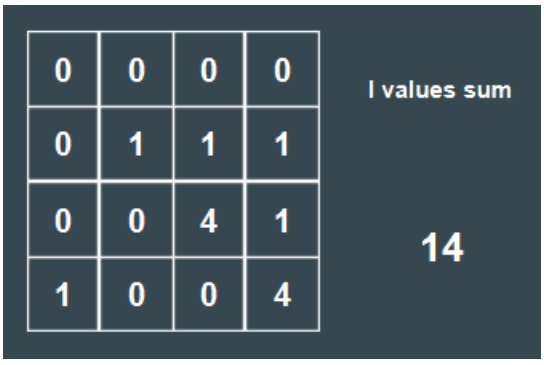


Gambar 38. *State* awal *puzzle* acak.

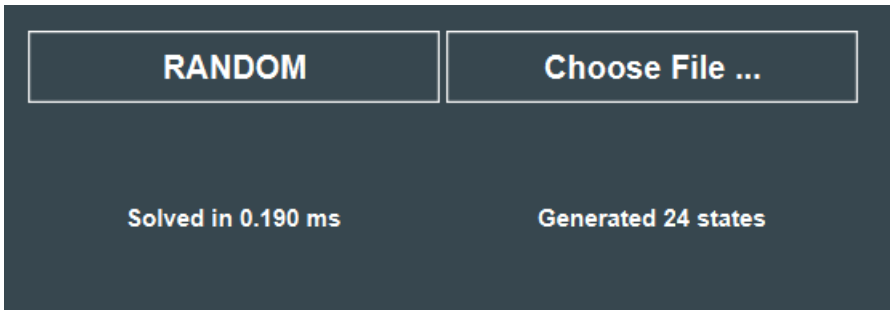
Gambar 39. *State* antara *puzzle* acak.



Gambar 40. *State* akhir *puzzle* acak.

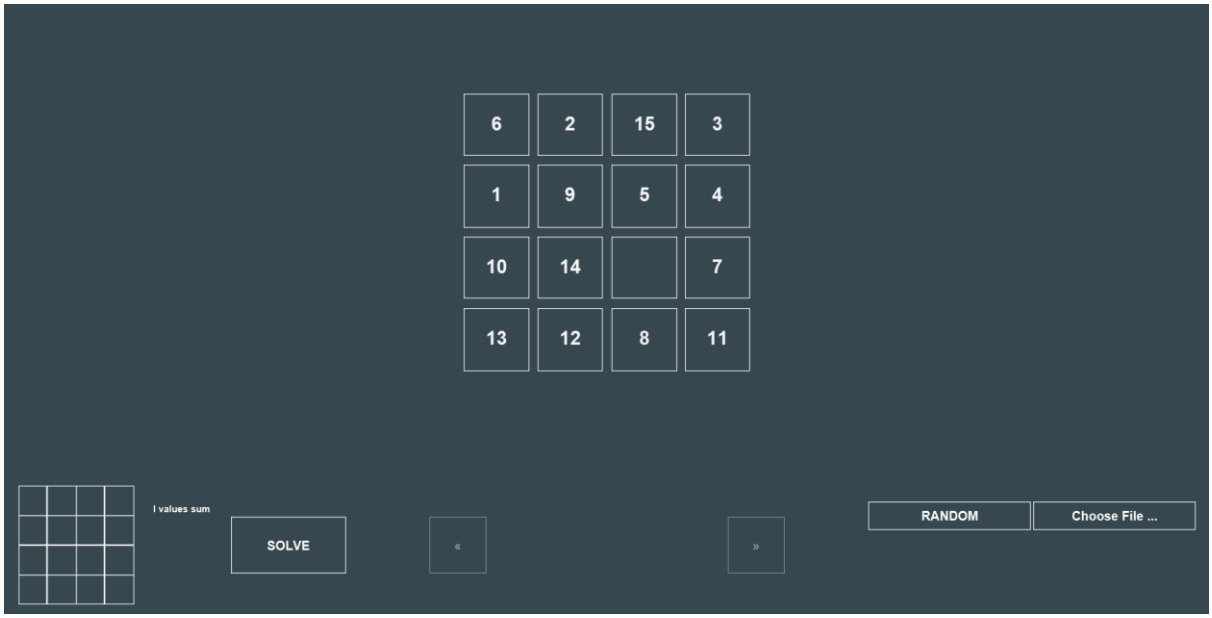


Gambar 41. *Output* nilai *l* untuk setiap elemen matriks dan jumlah seluruh nilai *l* dengan nilai *x*.

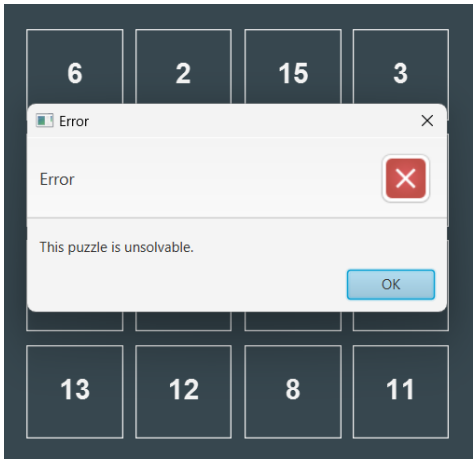


Gambar 42. *Output* lama eksekusi dan banyak *state* yang dibangkitkan pada *puzzle*.

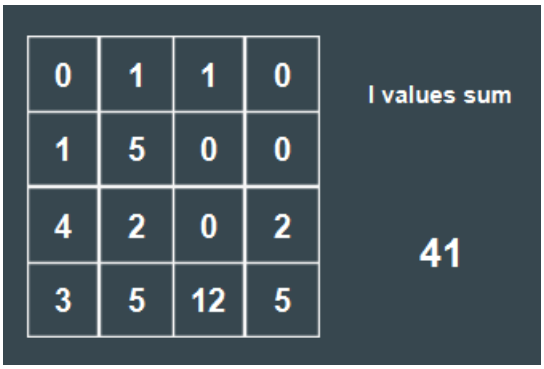
7. *Puzzle* yang tidak memiliki solusi 1



Gambar 43. *Input puzzle* yang *unsolvable* 1.

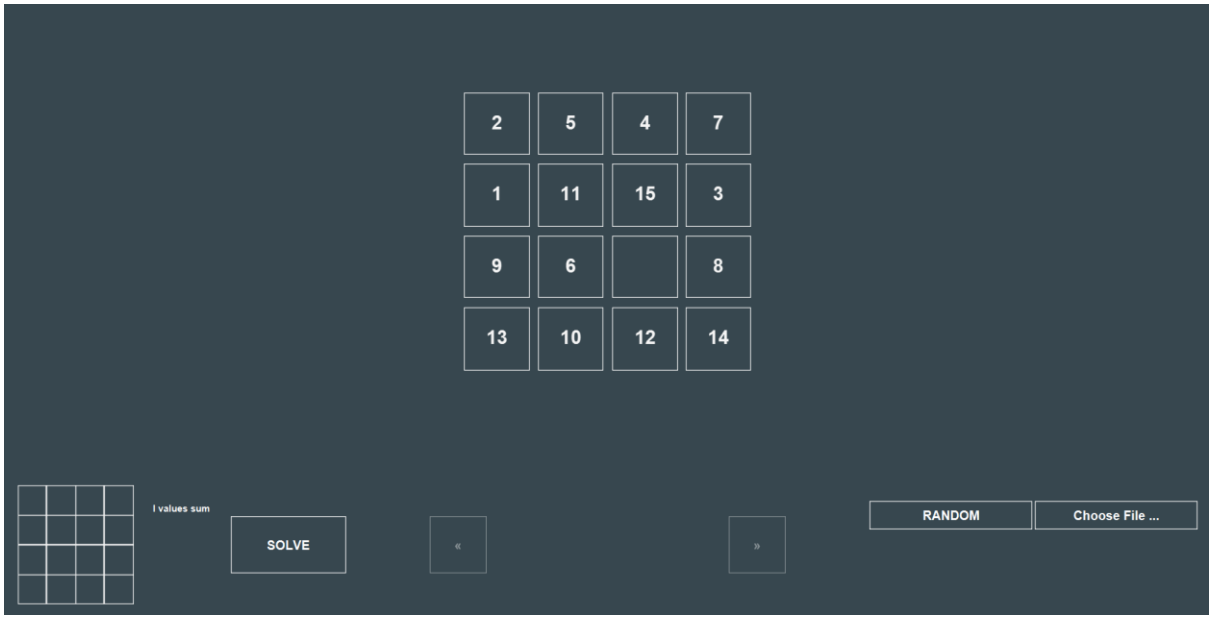


Gambar 44. *Output puzzle* yang *unsolvable* 1.

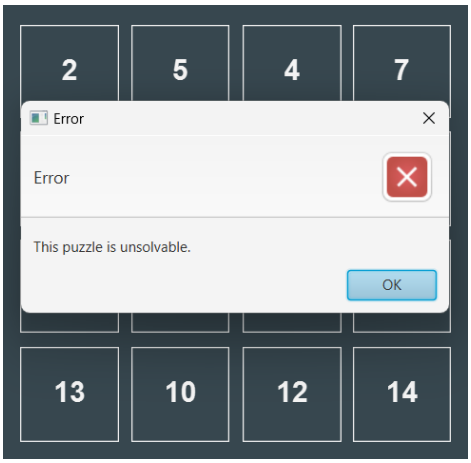


Gambar 45. *Output* nilai *l* untuk setiap elemen matriks dan jumlah seluruh nilai *l* dengan nilai *x*.

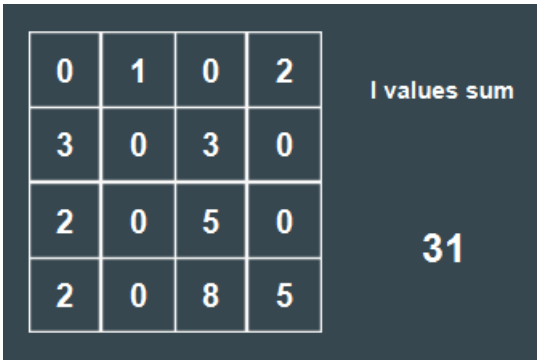
8. *Puzzle* yang tidak memiliki solusi 2



Gambar 46. *Input puzzle* yang *unsolvable* 2.

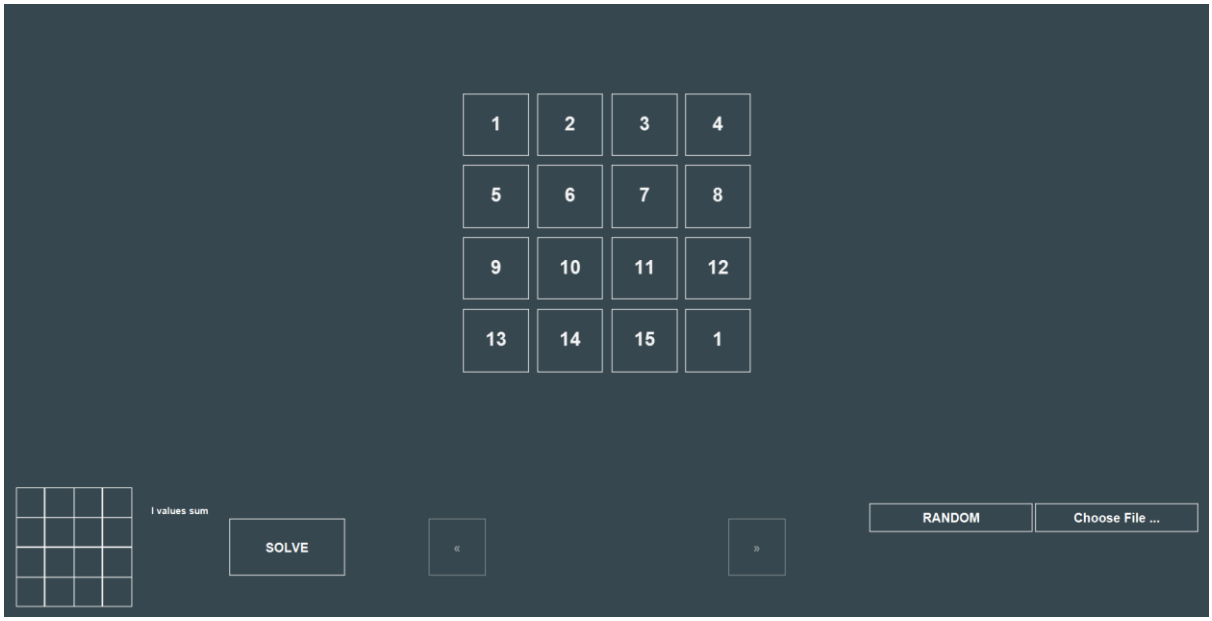


Gambar 47. *Output puzzle* yang *unsolvable* 2.

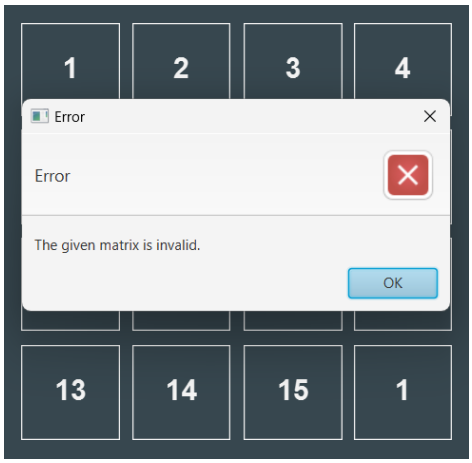


Gambar 48. *Output* nilai *l* untuk setiap elemen matriks dan jumlah seluruh nilai *l* dengan nilai *x*.

9. Matriks *puzzle* yang tidak valid



Gambar 49. *Input puzzle* yang tidak valid.



Gambar 50. *Output puzzle* yang tidak valid.

Alamat *repository source code* aplikasi

Source code aplikasi yang telah dibuat dapat diakses melalui tautan ini.

Check list

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	✓	