

LAPORAN TUGAS KECIL 1
IF2211 STRATEGI
ALGORITMA

Penyelesaian IQ
Puzzler Pro dengan
Algoritma Brute Force



Disusun oleh :
Dzaky Aurelia Fawwaz - 13523065

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN
INFORMATIKA INSTITUT TEKNOLOGI
BANDUNG
JL. GANESHA 10, BANDUNG 401322025

DAFTAR ISI

BAB I.....	4
DESKRIPSI MASALAH.....	4
1.1 Pendahuluan	4
BAB II.....	5
ALGORITMA BRUTE FORCE.....	5
2.1 Pengertian Algoritma Brute Force	5
2.2 Algoritma Brute Force dalam Solver IQ Puzzler Pro	5
2.3 Pseudocode Algoritma Brute Force dalam Solver IQ Puzzler Pro	6
BAB III	8
IMPLEMENTASI	8
3.1 Board.....	8
3.2Point	8
3.3 Piece.....	9
3.4 PuzzleSolver	9
3.5 PuzzleInput	10
3.6 FileHandler	10
3.7 PirimaryController	11
BAB IV	12
SOURCE CODE.....	12
4.1 Repository Github.....	12
4.2 Struktur Program.....	12
4.3 Antarmuka Program.....	13
4.4 Source Code	14
4.4.1 Board.java	14
4.4.2 Point.java	15
4.4.4 PuzzleSolver.java.....	16
4.4.5 PuzzleInput.java.....	17
4.4.6 App.java.....	17
4.4.7 FileHandler.java.....	18
4.4.8 PrimaryController.java.....	19
4.4.9 primary.fxml	21

4.4.10 style.css	22
BAB V	23
EKSPERIMEN	23
5.1 Test Uji 1.....	23
5.2 Test Uji 2.....	23
5.3 Test Uji 3.....	24
5.4 Test Uji 4.....	25
5.5 Test Uji 5.....	26
5.6 Test Uji 6.....	27
5.7 Test Uji 7.....	28
5.8 Test Uji 8.....	28
5.9 Test Uji 9.....	29
Lampiran	31
Tabel Kelengkapan Spesifikasi.....	31

BAB I

DESKRIPSI MASALAH

1.1 Pendahuluan



Gambar 1 Permainan IQ Puzzler Pro

(Sumber: <https://www.smartgamesusa.com>)

IQ Puzzler Pro adalah permainan papan yang diproduksi oleh perusahaan Smart Games. Tujuan dari permainan ini adalah pemain harus dapat mengisi seluruh papan dengan piece (blok puzzle) yang telah tersedia. Komponen penting dari permainan IQ Puzzler Pro terdiri dari:

1. Board (Papan)

Board merupakan komponen utama yang menjadi tujuan permainan dimana pemain harus mampu mengisi seluruh area papan menggunakan blok-blok yang telah disediakan.

2. Blok/Piece

Blok adalah komponen yang digunakan pemain untuk mengisi papan kosong hingga terisi penuh. Setiap blok memiliki bentuk yang unik dan semua blok harus digunakan untuk menyelesaikan puzzle.

Permainan dimulai dengan papan yang kosong. Pemain dapat meletakkan blok puzzle sedemikian sehingga tidak ada blok yang bertumpang tindih (kecuali dalam kasus 3D). Setiap blok puzzle dapat dirotasikan maupun dicerminkan. Puzzle dinyatakan selesai jika dan hanya jika papan terisi penuh dan seluruh blok puzzle berhasil diletakkan.

BAB II

ALGORITMA BRUTE FORCE

2.1 Pengertian Algoritma Brute Force

Algoritma brute force adalah pendekatan yang lempang (*straightforward*) untuk memecahkan suatu persoalan. Metode ini dibangun berdasarkan pernyataan yang jelas pada persoalan dan definisi konsep yang terlibat. Karakteristik utamanya adalah memecahkan masalah dengan cara yang sangat sederhana, langsung, dan jelas (*obvious way*). Prinsip dasarnya sangatlah sederhana - dengan mencoba semua kemungkinan solusi secara sistematis tanpa menggunakan strategi optimasi kompleks. Pendekatan ini pada dasarnya mengandalkan kekuatan komputasi untuk menguji setiap kemungkinan hingga menemukan solusi yang benar, tanpa mempertimbangkan efisiensi waktu atau sumber daya komputasi yang digunakan.

2.2 Algoritma Brute Force dalam Solver IQ Puzzler Pro

Algoritma penyelesaian permainan IQ Puzzler Pro dengan metode Brute Force adalah sebagai berikut:

1. Inisialisasi board puzzle dengan sel-sel kosong.
2. Mulai proses pencarian solusi dari piece pertama dalam daftar.
3. Untuk setiap piece, hasilkan semua kemungkinan rotasi dan pencerminan:
 - Rotasi 0° , 90° , 180° , 270°
 - Pencerminan horizontal dan vertikal
 - Total 8 variasi untuk setiap piece
4. Gunakan daftar posisi kosong pada papan untuk mencoba menempatkan piece.
5. Setiap kali mencoba menempatkan piece, periksa validitas penempatan:
 - Piece tidak boleh keluar batas papan
 - Piece tidak boleh tumpang tindih dengan piece lain
 - Piece hanya dapat ditempatkan pada sel kosong
6. Jika penempatan valid, letakkan piece pada posisi tersebut dan lanjutkan pencarian dengan piece berikutnya.
7. Jika tidak ada posisi valid untuk suatu piece setelah mencoba semua variasi, lakukan backtrack:
 - Hapus piece terakhir yang ditempatkan
 - Kembali ke piece sebelumnya

- Coba variasi atau posisi lain
8. Proses pencarian berlanjut hingga:
- Papan terisi penuh dan semua piece berhasil ditempatkan (solusi ditemukan)
 - Seluruh kemungkinan telah dicoba tanpa menemukan solusi
9. Selama proses pencarian, catat:
- Jumlah iterasi/kasus yang diperiksa
 - Waktu eksekusi algoritma
10. Hentikan pencarian dan tampilkan:
- Konfigurasi solusi jika berhasil ditemukan
 - Pesan tidak ada solusi jika seluruh kemungkinan telah dicoba

2.3 Pseudocode Algoritma Brute Force dalam Solver IQ Puzzler Pro

```

Fungsi solve():
    // Catat waktu mulai eksekusi algoritma
    start_time = get_current_time()
    // Mulai proses pemecahan teka-teki dari keping pertama (indeks 0)
    return solvePuzzle(0)

Fungsi solvePuzzle(piece_index):
    // Kondisi dasar: jika semua keping telah dicoba
    // Periksa apakah sudah mencapai akhir daftar keping
    If piece_index >= total_pieces:
        // Catat waktu selesai
        end_time = get_current_time()
        // Periksa apakah papan sudah terisi penuh
        return is_board_full()

    // Ambil keping saat ini dari daftar keping
    piece = piece_list[piece_index]
    current_piece = piece

    // Dapatkan daftar posisi kosong di papan
    empty_positions = get_empty_board_positions()

    // Iterasi untuk mencoba pencerminan dan rotasi keping
    // Lakukan 2 kali (0 dan 1) untuk mencakup orientasi normal dan tercermin
    For i = 0 to 1:
        // Putar keping sebanyak 4 kali (0, 90, 180, 270 derajat)
        For r = 0 to 3:
            // Coba setiap posisi kosong
            For each position in empty_positions:
                // Hitung jumlah iterasi yang dilakukan
                increment_iterations

            // Periksa apakah keping bisa diletakkan di posisi ini

```

```
If can_place_piece(current_piece, position):
    // Letakkan keping di posisi yang dipilih
    place_piece(current_piece, position)

    // Coba pecahkan untuk keping berikutnya secara rekursif
    If solvePuzzle(piece_index + 1):
        // Jika solusi ditemukan, kembalikan true
        return true

    // Jika tidak berhasil, hapus keping dan coba posisi lain
    remove_piece(current_piece, position)

    // Putar keping untuk mencoba orientasi berbeda
    current_piece = rotate_piece(current_piece)

    // Cerminkan keping untuk mencoba orientasi terbalik
    current_piece = mirror_piece(piece)

// Jika tidak ada solusi ditemukan setelah semua percobaan
return false
```

BAB III

IMPLEMENTASI

3.1 Board

- Attribute

Nama	Tipe	Deskripsi
Grid	Char[]	Matrix mempresentasikan board
Rows	Int	Jumlah baris
Cols	Int	Jumlah kolom
emptyPoints	List<Point>	Titik yang kosong pada grid

- Method

Nama	Tipe	Deskripsi
Board(int rows,int cols, char[][] grid)	Konstruktor	Membuat objek papan
isFull()	boolean	Memeriksa apakah grid sudah terisi semua
canPlacePiece (Piece piece,int x, int y)	boolean	Memeriksa apakah bisa menempatkan suatu piece pada koordinat tertentu
placePiece (Piece piece, int startX, int startY)	void	Memasang piece di suatu koordinat pada board
removePiece (Piece piece, int startX, int startY)	void	Me-remove piece pada suatu koordinat pada board
boardToString()	String	Mempresentasikan board dalam string
getRows()	Int	Mengembalikan jumlah baris papan
getCols()	Int	Mengembalikan jumlah kolom papan
getEmptyPoints	List<Point>	Mengembalikan titik kosong pada papan

3.2 Point

- Attribute

Nama	Tipe	Deskripsi
x	Int	Koordinat x
y	Int	Koordinat y

- Method

Nama	Tipe	Deskripsi
Equals	boolean	Memeriksa kesamaan antara dua objek Point
translate(int dx, int dy)	Point	Membuat titik baru dengan perpindahan dx dan dy dari titik saat ini
Point (int x, int y)	konsttuktur	Membuat objek Point dengan koordinat x dan y
hashCode()		Mengembalikan nilai hash code untuk objek Point

3.3 Piece

- Attribute

Nama	Tipe	Deskripsi
points	List<point>	Titik-titik yg mempreentasikan piece
id	char	Alfabet (A-Z)

- Method

Nama	Tipe	Deskripsi
rotate()	Piece	Merotasi piece
Mirror()	Piece	Mencerminkan piece
createfromLines()	Piece	Membuat piece dari List<String>

3.4 PuzzleSolver

- Attribute

Nama	Tipe	Deskripsi
board	Board	Papan permainan yg akan diisi piece-piece
pieces	List<Pieces>	Daftar piece yg dipakai
iterations	Long	Jumlah kasus ditinjau
startTime	Long	Waktu awal permainan
endTime	Long	Waktu akhir permainan

- Method

Nama	Tipe	Deskripsi
PuzzleSolver	Konstruktor	Menginisialisasi objek puzzlesolver
Solve	boolean	Memulai pencarian

solvePuzzle(int pieceindex)	Boolean	Algoritma rekursif yang mencoba menyusun blok pada papan.
getExecutionTime()	Long	Mengembalikan waktu eksekusi
getIterations()	Long	Mengembalikan jumlah iterasi
getBoardState()	String	Mengembalikan start board
getPuzzleRows()	Int	Mengembalikan rows papan
getPuzzleCols()	Int	Mengembalikan cols papan

3.5 PuzzleInput

- Attribute

Nama	Tipe	Deskripsi
N	Int	Jumlah baris papan
M	Int	Jumlah kolom papan
P	Int	Jumlah poece
Mode	String	Mode konfigurasi
Pieces	List<Piece>	Daftar piece
Grid	Char[]	Matriks awal papan

- Method

Nama	Tipe	Deskripsi
PuzzleInput	Konstruktor	Konstruktor

3.6 FileHandler

- Attribute

Nama	Tipe	Deskripsi
COLORS	Color[]	Array of color

- Method

Nama	Tipe	Deskripsi
readInputFile(File inputFile)	PuzzleInput	Membaca file input puzzle dan menghasilkan objek PuzzleInput
createSolutionImage(String boardState, int N, int M, long executionTime, long iterations)	WritableImage	Membuat gambar visual solusi puzzle

saveText(File outputFile, String boardState, long executionTime, long iterations)	Void	Menyimpan solusi puzzle dalam format teks
saveImage(File outputFile, String boardState, int N, int M, long executionTime, long iterations)	void	Menyimpan solusi puzzle dalam format gambar PNG

3.7 PrimaryController

- Attribute

Nama	Tipe	Deskripsi
chooseFileButton	Button	Tombol untuk memilih file
solveButton	Button	Tombol untuk menyelesaikan puzzle
saveImageButton	Button	Tombol untuk Menyimpan solusi sebagai gambar
saveTextButton	Button	Tombol untuk menyimpan output sebagai text
fileLabel	Label	Menampilkan file yg dipilih
Boardgrid	GridPane	Grid untuk menampilkan puzzle
outputArea	TextArea	Area teks untuk menampilkan informasi output
loadingIndicator	ProgressIndicator	Loading saat proses pencarian jawaban
selectedFile	File	File puzzle yg dipilih
Solver	PuzzleSolver	Objek untuk menyelesaikan puzzle
N	Int	Jumlah kolom
M	int	Jumlah baris

- Method

Nama Method	Tipe	Deskripsi
chooseFile()	void	Membuka dialog pemilihan file puzzle
clearBoard()	Void	Membersihkan papan puzzle dan area output
solvePuzzle()	Void	Memulai proses penyelesaian puzzle

updateBoard(String BoardState)	Void	Memperbarui tampilan grid dengan state papan puzzle
saveImage()	Void	Menyimpan solusi puzzle dalam format gambar
Savetext()	Void	Menyimpan solusi puzzle dalam format teks

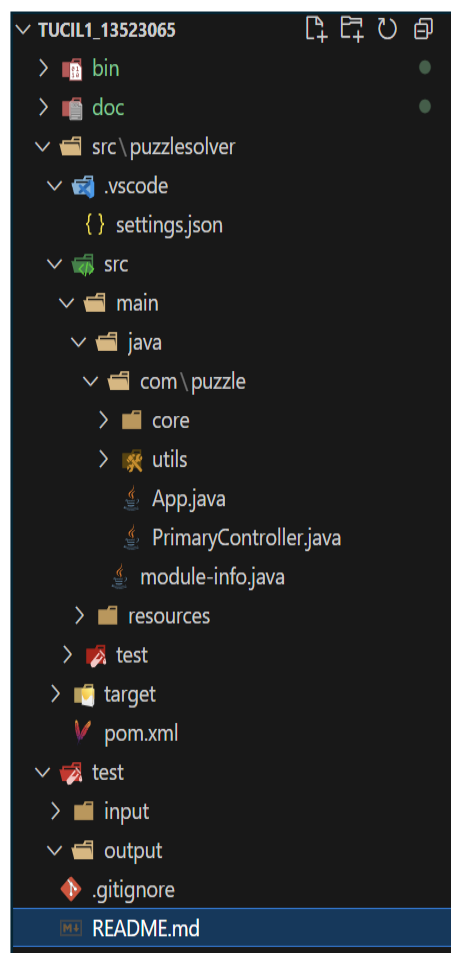
BAB IV

SOURCE CODE

4.1 Repository Github

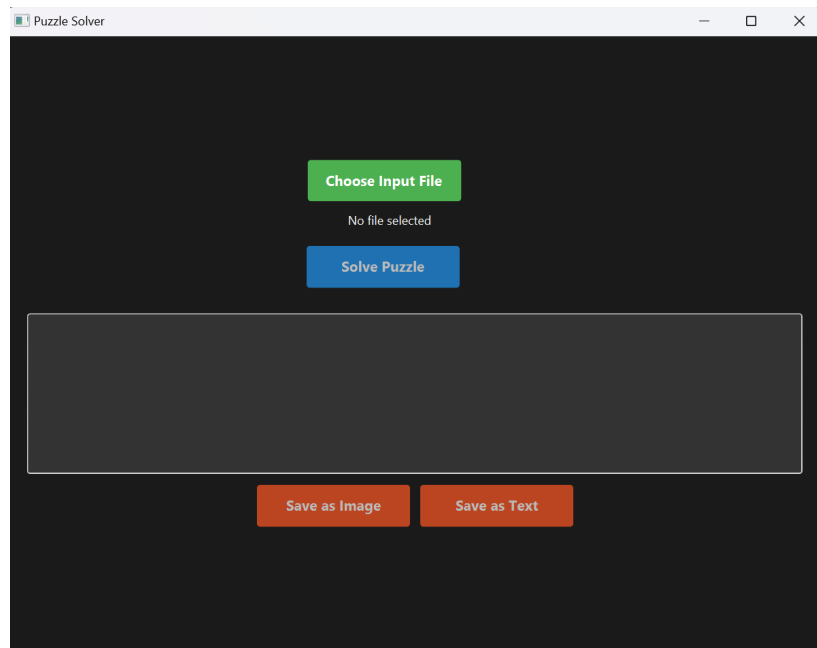
https://github.com/WwzFwz/Tucil1_13523065

4.2 Struktur Program

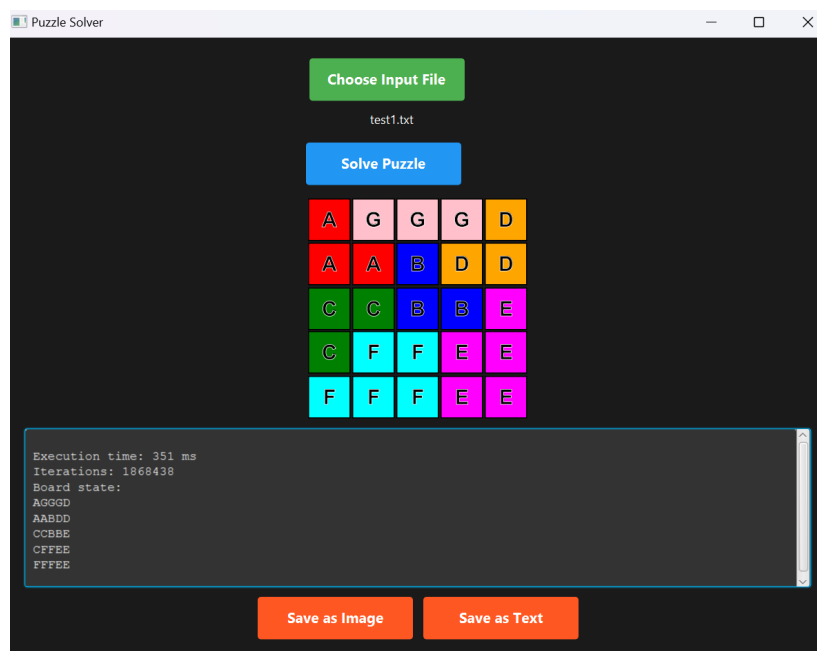


Gambar 4.2.1 Struktur Program

4.3 Antarmuka Program



Gambar 4.3.1 Antarmuka Program sebelum menerima input



Gambar 4.3.2 Antarmuka Program setelah menerima input

4.4 Source Code

4.4.1 Board.java

```
package com.puzzle.core;
import java.util.ArrayList;
import java.util.List;

public class Board{
    private char[][] grid;
    private final int rows;
    private final int cols;
    private List<Point> emptyPoints;

    public Board(int rows,int cols, char[][] grid){
        this.rows = rows;
        this.cols = cols;
        this.grid = grid ;
        this.emptyPoints = new ArrayList<>();
        for(int i = 0 ; i < rows ; i ++){
            for(int j = 0 ; j < cols ; j ++){
                if (grid[i][j] == '.') {
                    emptyPoints.add(new Point(j, i));
                }
            }
        }
    }

    public boolean isFull() {
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                if (grid[i][j] == '.') return false;
            }
        }
        return true;
    }

    public boolean canPlacePiece(Piece piece,int x, int y){
        List<Point> piecePoints = piece.getPointsAt(x,y);
        for(Point p : piecePoints){
            if(p.x >= cols || p.y >= rows || p.x < 0 || p.y < 0 || grid[p.y][p.x] != '.') {
                return false;
            }
        }
        return true;
    }

    public void placePiece(Piece piece, int startX, int startY) {
        List<Point> piecePoints = piece.getPointsAt(startX, startY);
        for (Point p : piecePoints) {
            grid[p.y][p.x] = piece.getId();
            emptyPoints.remove(p);
        }
    }

    public void removePiece(Piece piece, int startX, int startY) {
        List<Point> piecePoints = piece.getPointsAt(startX, startY);
        for (Point p : piecePoints) {
            grid[p.y][p.x] = '.';
            emptyPoints.add(new Point(p.x, p.y));
        }
    }

    public String boardToString() {
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                sb.append(grid[i][j]);
            }
            sb.append(c: '\n');
        }
        return sb.toString();
    }

    public int getRows() { return rows; }
    public int getCols() { return cols; }
    public List<Point> getEmptyPoints() {
        return emptyPoints;
    }
}
```

4.4.2 Point.java

```
package com.puzzle.core;

public class Point{
    public int x;
    public int y;

    public Point(int x,int y){
        this.x = x ;
        this.y = y;
    }
    public Point translate(int dx, int dy) {
        return new Point(x + dx, y + dy);
    }
    @Override
    public boolean equals(Object obj) {
        if (this == obj) return true;
        if (!(obj instanceof Point)) return false;
        Point other = (Point) obj;
        return this.x == other.x && this.y == other.y;
    }
    @Override
    public int hashCode() {
        return 31 * x + y;
    }
}
```

4.4.3 Piece.java

```
package com.puzzle.core;
import java.util.ArrayList;
import java.util.List;
public class Piece{
    private List<Point> points;
    private char id;
    public Piece(List<Point> points, char id) {
        this.points = points;
        this.id = id;
    }
    public static Piece createFromLines(List<String> lines, char id) {
        List<Point> points = new ArrayList<>();
        for (int y = 0; y < lines.size(); y++) {
            String line = lines.get(y);
            for (int x = 0; x < line.length(); x++) {
                if (line.charAt(x) == id) {
                    points.add(new Point(x, y));
                }
            }
        }
        return new Piece(points, id);
    }
    public Piece rotate(){
        List<Point> rotatedPoint = new ArrayList<>();
        for(Point p : points){
            rotatedPoint.add(new Point(-p.y,p.x));
        }
        return new Piece(rotatedPoint,id);
    }
    public Piece mirror(){
        List<Point> mirroredPoint = new ArrayList<>();
        for(Point p : points){
            mirroredPoint.add(new Point(-p.x,p.y));
        }
        return new Piece (mirroredPoint,id);
    }
    public List<Point> getPointsAt(int a, int b) {
        List<Point> translatedPoints = new ArrayList<>();
        for (Point p : points) {
            translatedPoints.add(p.translate(a, b));
        }
        return translatedPoints;
    }
    public char getId() { return id; }
}
```

4.4.4 PuzzleSolver.java

```
package com.puzzle.core;
import java.util.ArrayList;
import java.util.List;

public class PuzzleSolver {
    private Board board;
    private List<Piece> pieces;
    private long iterations;
    private long startTime;
    private long endTime;

    public PuzzleSolver(int rows, int cols, List<Piece> pieces, char [][] initialGrid) {
        this.board = new Board(rows, cols, initialGrid);
        this.pieces = pieces;
        this.iterations = 0;
    }

    public boolean solve() {
        startTime = System.currentTimeMillis();
        return solvePuzzle(pieceIndex:0);
    }

    private boolean solvePuzzle(int pieceIndex) {
        if (pieceIndex >= pieces.size()) {
            endTime = System.currentTimeMillis();
            return board.isFull();
        }

        Piece piece = pieces.get(pieceIndex);
        Piece currentPiece = piece;
        List<Point> positions = new ArrayList<>(board.getEmptyPoints());
        for (int i = 0; i < 2; i++) {
            for (int r = 0; r < 4; r++) {
                for (Point p : positions) {
                    iterations++;
                    if (board.canPlacePiece(currentPiece, p.x, p.y)) {
                        board.placePiece(currentPiece, p.x, p.y);
                        if (solvePuzzle(pieceIndex + 1)) {
                            return true;
                        }
                        board.removePiece(currentPiece, p.x, p.y);
                    }
                }
                currentPiece = currentPiece.rotate();
            }
            currentPiece = piece.mirror();
        }

        return false;
    }

    public long getExecutionTime() {
        return endTime - startTime;
    }

    public long getIterations() {
        return iterations;
    }

    public String getBoardState() {
        return board.boardToString();
    }

    public int getPuzzleRows() {
        return board.getRows();
    }

    public int getPuzzleCols() {
        return board.getCols();
    }
}
```


4.4.5 PuzzleInput.java

```
package com.puzzle.core;
import java.util.List;

public class PuzzleInput {
    public final int N, M, P;
    public final String mode;
    public final List<Piece> pieces;
    public final char[][] grid;

    public PuzzleInput(int N, int M, int P, String mode, List<Piece> pieces, char[][] grid) {
        this.N = N;
        this.M = M;
        this.P = P;
        this.mode = mode;
        this.pieces = pieces;
        this.grid = grid;
    }
}
```

4.4.6 App.java

```
package com.puzzle;

import java.io.IOException;

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

public class App extends Application {

    private static Scene scene;

    @Override
    public void start(Stage stage) throws IOException {
        javafx.scene.text.Font.getFontFamilies();
        stage.setTitle(value:"Puzzle Solver");
        scene = new Scene(loadFXML(fxml:"primary"), width:800, height:600);
        scene.getStylesheets().clear();
        scene.getStylesheets().add(App.class.getResource(name:"styles.css").toExternalForm());
        stage.setScene(scene);
        stage.show();
    }

    private static Parent loadFXML(String fxml) throws IOException {
        FXMLLoader fxmlLoader = new FXMLLoader(App.class.getResource(fxml + ".fxml"));
        return fxmlLoader.load();
    }

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

4.4.7 FileHandler.java

```

public class PuzSolver {
    public static final Color[] colors = {
        Color.WHITE, Color.LIGHTGRAY, Color.DARKGRAY, Color.BLACK,
        Color.BLUE, Color.LIGHTBLUE, Color.LIGHTGREEN, Color.DARKGREEN,
        Color.CYAN, Color.PINK, Color.BROWN
    };

    public static PuzzleInput readInputFile(File inputFile) throws IOException {
        try {
            Scanner reader = new Scanner(new FileReader(inputFile));
            String direction = reader.readLine();
            if (direction == null || direction.trim().isEmpty()) {
                throw new IOException("message: format error: missing M x N line");
            }
            String[] dims = direction.trim().split("\\s+");
            if (dims.length != 4) {
                throw new IOException("message: format error: M x N line must contain exactly 4 numbers");
            }
            int M = Integer.parseInt(dims[0]);
            int N = Integer.parseInt(dims[1]);
            int W = Integer.parseInt(dims[2]);
            int H = Integer.parseInt(dims[3]);
            if (M <= 0 || M <= 0 || W <= 0 || H <= 0) {
                throw new IOException("message: invalid input: M, N, W, H must be positive");
            }
            String mode = reader.readLine().trim();
            char[][] grid = new char[M][N];
            if (mode.matches("\\d{4}")) {
                for (int i = 0; i < M; i++) {
                    for (int j = 0; j < N; j++) {
                        grid[i][j] = '-';
                    }
                }
            } else {
                throw new IOException("message: invalid mode");
            }
            ArrayList<Place> places = new ArrayList<>();
            ArrayList<CurrentLine> currentLines = new ArrayList<>();
            String line;
            char currentId = 'A';
            int curWidth = 0;
            while ((line = reader.readLine()) != null) {
                line = line.trim();
                if (line.isEmpty()) {
                    char pieceId = line.charAt(0);
                    int curCurrentLineWidth = 0;
                    for (char c : line.toCharArray()) {
                        if (c == pieceId) {
                            curCurrentLineWidth++;
                        }
                    }
                    curWidth = Math.max(curWidth, curCurrentLineWidth);
                    if (currentId == 'A') {
                        currentId = pieceId;
                    } else if (isUsedBy(currentId)) {
                        if (curCurrentLineWidth > M || curWidth > N) {
                            throw new IOException("piece " + pieceId + " size exceeds board dimensions");
                        }
                        places.add(new Place(currentLineId, currentLine, currentId));
                        currentLine = new ArrayList<>();
                        currentId = pieceId;
                        curWidth = curCurrentLineWidth;
                    }
                }
                for (char c : line.toCharArray()) {
                    if (c != '-' && c != currentId) {
                        throw new IOException("message: invalid piece format: alone letters in same piece");
                    }
                }
                currentLines.add(line);
            }
            if (currentLineId.isEmpty()) {
                int height = currentLines.size();
                int maxLineLen = Math.max(height, curWidth);
                if (maxLineLen > Math.min(M, N)) {
                    throw new IOException("piece " + currentId + " size exceeds board dimensions");
                }
                places.add(new Place(currentLineId, currentLine, currentId));
            }
            if (places.isEmpty()) {
                throw new IOException("message: number of pieces (" + places.size() + ") does not match P (" + P + ")");
            }
            HashSet<Character> uniqueIds = new HashSet<>();
            for (Place place : places) {
                if (isUsedBy(place.getId())) {
                    throw new IOException("duplicate piece id found: " + place.getId());
                }
            }
            return new PuzzleInput(M, N, P, mode, places, grid);
        } catch (NumberFormatException e) {
            throw new IOException("message: invalid number format for M x N line");
        }
    }

    public static WriteableImage createSolutionImage(String boardData, int W, int H, long executionTime, long iterations) {
        double CHL_SIZE = 50;
        double extraWeight = 50;
        Canvas canvas = new Canvas(W + CHL_SIZE, H + CHL_SIZE + extraWeight);
        GraphicsContext gc = canvas.getGraphicsContext2D();
        gc.setFill(Color.WHITE);
        gc.fillRect(0, 0, canvas.getWidth(), canvas.getHeight());
        String[] lines = boardData.split("\\n");
        for (int i = 0; i < H; i++) {
            for (int j = 0; j < W; j++) {
                char c = lines[i].charAt(j);
                if (c != '-' && c != ' ') {
                    gc.setFill(colors[c]);
                    gc.fillRect(j * CHL_SIZE, i * CHL_SIZE, CHL_SIZE, CHL_SIZE);
                }
            }
            gc.setStroke(Color.BLACK);
            gc.strokeRect(j * CHL_SIZE, i * CHL_SIZE, CHL_SIZE, CHL_SIZE);
            if (c != '-' && c != ' ') {
                gc.setFill(Color.BLACK);
                gc.strokeRect(j * CHL_SIZE, i * CHL_SIZE, CHL_SIZE, CHL_SIZE);
                gc.fillText(String.valueOf(c),
                    j * CHL_SIZE + CHL_SIZE / 2,
                    i * CHL_SIZE + 2 * CHL_SIZE / 3);
            }
        }
        gc.setFill(Color.BLACK);
        gc.fillRect(0, 0, canvas.getWidth(), canvas.getHeight());
        gc.setStroke(Color.GRAY);
        gc.strokeRect(0, 0, canvas.getWidth(), canvas.getHeight());
        gc.fillText("Execution Time: " + executionTime + " ms",
            0, W + CHL_SIZE + 20);
        gc.fillText("Iterations: " + iterations,
            0, W + CHL_SIZE + 40);
        WriteableImage image = new WriteableImage((int) canvas.getWidth(),
            canvas.getWidth(), canvas.getHeight());
        return image;
    }

    public static void saveInput(File inputFile, String boardData, long executionTime, long iterations) throws IOException {
        try {
            FileWriter writer = new FileWriter(new FileWriter(inputFile));
            writer.println("solution");
            writer.println(boardData);
            writer.println("Execution Time: " + executionTime + " ms");
            writer.println("Iterations: " + iterations);
        }
    }

    public static void saveImage(File outputFile, String boardData, int W, int H, long executionTime, long iterations) {
        try {
            WriteableImage image = createSolutionImage(boardData, W, H, executionTime, iterations);
            WriteableImage writeableImage = image;
            double extraWeight = WriteableImage.fromXpixmap(writeableImage, null);
            ImageIO.write(writeableImage, "png", outputFile);
        } catch (IOException ex) {
            System.out.println("Error saving image: " + ex.getMessage());
        }
    }
}

```

4.4.8 PrimaryController.java

```
public class PrimaryController {
    @FXML private Button chooseFileButton;
    @FXML private Button solveButton;
    @FXML private Button saveImageButton;
    @FXML private Button saveTextButton;
    @FXML private Label fileLabel;
    @FXML private GridPane boardGrid;
    @FXML private TextArea outputArea;
    @FXML private ProgressIndicator loadingIndicator;
    private File selectedFile;
    private PuzzleSolver solver;
    private int N, M;
    private static final int CELL_SIZE = 40;

    @FXML
    private void chooseFile() {
        FileChooser fileChooser = new FileChooser();
        fileChooser.setTitle(value:"Open Puzzle File");
        fileChooser.getExtensionFilters().add(
            new FileChooser.ExtensionFilter(description:"Text Files", ...extensions:".*.txt")
        );

        selectedFile = fileChooser.showOpenDialog(chooseFileButton.getScene().getWindow());
        if (selectedFile != null) {
            fileLabel.setText(selectedFile.getName());
            solveButton.setDisable(value:false);
            clearBoard();
        }
    }

    private void clearBoard() {
        boardGrid.getChildren().clear();
        outputArea.clear();
        saveImageButton.setDisable(value:true);
        saveTextButton.setDisable(value:true);
    }

    @FXML
    private void solvePuzzle() {
        Platform.runLater(() -> {
            loadingIndicator.setVisible(value:true);
            solveButton.setDisable(value:true);
        });
        new Thread(() -> {
            try {
                PuzzleInput input = FileHandler.readInputFile(selectedFile);
                solver = new PuzzleSolver(input.N, input.M, input.pieces, input.grid);
                N = input.N;
                M = input.M;

                Platform.runLater(() -> {
                    outputArea.clear();
                });

                boolean solved = solver.solve();
                long executionTime = solver.getExecutionTime();
                long iterations = solver.getIterations();

                Platform.runLater(() -> {
                    outputArea.appendText("\nExecution time: " + executionTime + " ms\n");
                    outputArea.appendText("Iterations: " + iterations + "");

                    if (solved) {
                        outputArea.appendText("\nBoard state:\n");
                        outputArea.appendText(solver.getBoardState());
                        updateBoard(solver.getBoardState());
                        saveImageButton.setDisable(value:false);
                        saveTextButton.setDisable(value:false);
                    } else {
                        outputArea.appendText("\nNo solution exists!\n");
                    }

                    loadingIndicator.setVisible(value:false);
                    solveButton.setDisable(value:false);
                });
            } catch (IOException e) {
                Platform.runLater(() -> {
                    outputArea.appendText("Error: " + e.getMessage() + "\n");
                    loadingIndicator.setVisible(value:false);
                    solveButton.setDisable(value:false);
                });
            }
        }).start();
    }
}
```

```

    }

    private void updateBoard(String boardState) {
        boardGrid.getChildren().clear();
        String[] rows = boardState.split(regex: "\\n");

        for (int i = 0; i < N; i++) {
            for (int j = 0; j < M; j++) {
                StackPane cellStack = new StackPane();
                Rectangle cell = new Rectangle(CELL_SIZE, CELL_SIZE);
                char c = rows[i].charAt(j);

                if (c != '.' && c != ' ') {
                    cell.setFill(FileHandler.COLORS[c - 'A']);
                    Text text = new Text(String.valueOf(c));
                    text.setFont(Font.font(Family: "Arial", FontWeight: BOLD, size: 20));
                    text.setFill(Color.BLACK);
                    text.setStroke(Color.WHITE);
                    text.setStrokeWidth(value: 0.5);
                    cellStack.getChildren().addAll(cell, text);
                } else {
                    cell.setFill(Color.WHITE);
                    cellStack.getChildren().add(cell);
                }

                cell.setStroke(Color.BLACK);
                cell.setStrokeWidth(value: 1);

                boardGrid.add(cellStack, j, i);
            }
        }
    }

    @FXML
    private void saveImage() {
        try {
            FileChooser fileChooser = new FileChooser();
            fileChooser.setTitle(value: "Save Image");
            fileChooser.getExtensionFilters().add(
                new FileChooser.ExtensionFilter(description: "PNG Files", ..extensions: "*.png")
            );
            fileChooser.setInitialFileName(value: "solution_image.png");

            File file = fileChooser.showSaveDialog(saveImageButton.getScene().getWindow());
            if (file != null) {
                FileHandler.saveImage(file,
                    solver.getBoardState(),
                    N, M,
                    solver.getExecutionTime(),
                    solver.getIterations());
                outputArea.appendText("Solution saved as image: " + file.getName() + "\\n");
            }
        } catch (Exception e) {
            outputArea.appendText("Error saving image: " + e.getMessage() + "\\n");
        }
    }

    @FXML
    private void saveText() {
        try {
            FileChooser fileChooser = new FileChooser();
            fileChooser.setTitle(value: "Save Text");
            fileChooser.getExtensionFilters().add(
                new FileChooser.ExtensionFilter(description: "Text Files", ..extensions: "*.txt")
            );
            fileChooser.setInitialFileName(value: "solution_text.txt");

            File file = fileChooser.showSaveDialog(saveTextButton.getScene().getWindow());
            if (file != null) {
                FileHandler.saveText(file,
                    solver.getBoardState(),
                    solver.getExecutionTime(),
                    solver.getIterations());
                outputArea.appendText("Solution saved as text: " + file.getName() + "\\n");
            }
        } catch (IOException e) {
            outputArea.appendText("Error saving text: " + e.getMessage() + "\\n");
        }
    }
}

```

4.4.9 primary.fxml

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.layout.*?>
<?import javafx.scene.control.*?>
<?import javafx.geometry.Insets?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.TextArea?>

<VBox alignment="CENTER" spacing="10" xmlns:fx="http://javafx.com/fxml"
    fx:controller="com.puzzle.PrimaryController">
    <padding>
        <Insets top="20" right="20" bottom="20" left="20"/>
    </padding>

    <VBox alignment="CENTER" spacing="10">
        <Button fx:id="chooseFileButton" text="Choose Input File" onAction="#chooseFile"/>
        <Label fx:id="fileLabel" text="No file selected"/>
    </VBox>

    <HBox alignment="CENTER" spacing="10">
        <Button fx:id="solveButton" text="Solve Puzzle" onAction="#solvePuzzle" disable="true"/>
        <ProgressIndicator fx:id="loadingIndicator" visible="false"/>
    </HBox>

    <GridPane fx:id="boardGrid" alignment="CENTER" hgap="1" vgap="1"/>

    <TextArea fx:id="outputArea" editable="false" prefRowCount="10" prefColumnCount="40"/>

    <HBox alignment="CENTER" spacing="10">
        <Button fx:id="saveImageButton" text="Save as Image" onAction="#saveImage" disable="true"/>
        <Button fx:id="saveTextButton" text="Save as Text" onAction="#saveText" disable="true"/>
    </HBox>
</VBox>
```

4.4.10 style.css

```
.root {
  -fx-background-color: #1a1a1a;
}

.button {
  -fx-background-color: #2196F3;
  -fx-text-fill: white;
  -fx-font-weight: bold;
  -fx-font-size: 14px;
  -fx-min-width: 150px;
  -fx-min-height: 40px;
}

.button:hover {
  -fx-background-color: #1976D2;
}

.button:disabled {
  -fx-opacity: 0.7;
}

#chooseFileButton {
  -fx-background-color: #4CAF50;
  -fx-translate-x: -30px;
}

#chooseFileButton:hover {
  -fx-background-color: #388E3C;
}

#saveImageButton, #saveTextButton {
  -fx-background-color: #FF5722;
}

#saveImageButton:hover, #saveTextButton:hover {
  -fx-background-color: #E64A19;
}

.label {
  -fx-text-fill: white;
  -fx-translate-x: -25px;
}

.text-area {
  -fx-control-inner-background: #333333;
  -fx-text-fill: white;
  -fx-font-family: 'Courier New';
}

.text-area .content {
  -fx-background-color: #333333;
}

.grid-pane {
  -fx-background-color: #333333;
  -fx-grid-lines-visible: true;
}

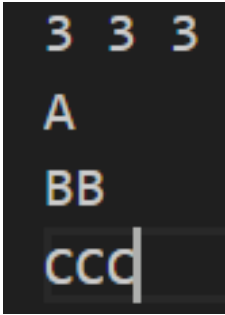
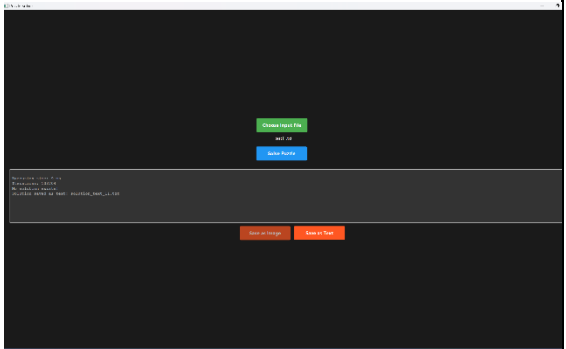
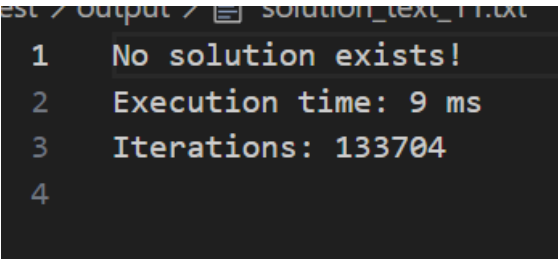

.progress-indicator {
  -fx-progress-color: #2196F3;
}

.progress-indicator .percentage {
  -fx-fill: white;
}
```

BAB V

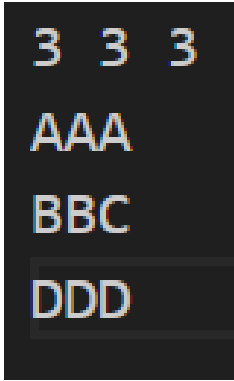
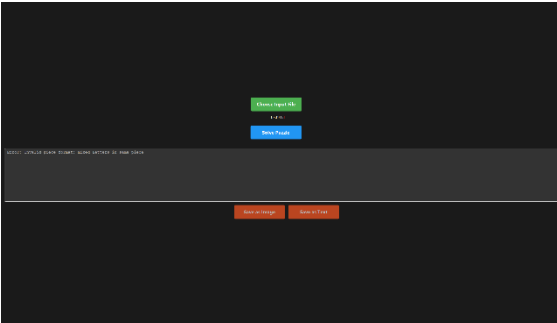
EKSPERIMEN

5.1 Test Uji 1

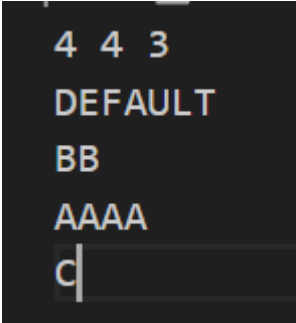
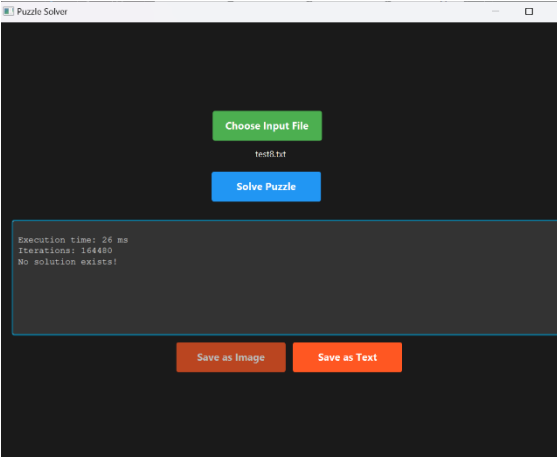
Input File	Output GUI
	
Output Text (.txt)	Output Image (.png)
	
Keterangan: Tidak ada Penyelesaian	

5.2 Test Uji 2

Input File	Output GUI
------------	------------

	
Output Text (.txt)	Output Image (.png)
-	-
Keterangan: Invalid piece format , ada lebih dari 1 karakter dalam 1 line ,yakni BBC	

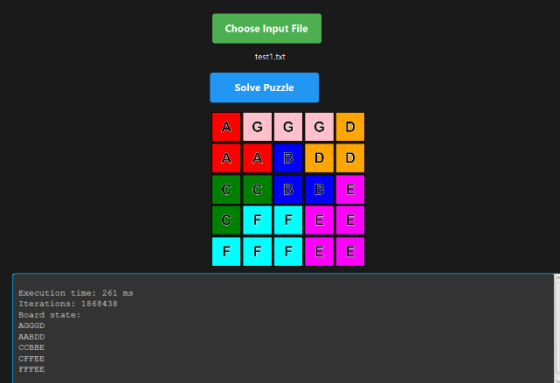
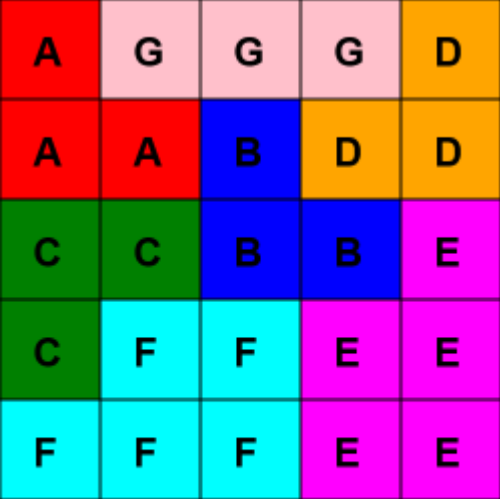
5.3 Test Uji 3

Input File	Output GUI
	
Output Text (.txt)	Output Image (.png)
-	-
Keterangan: Input tidak valid, ada ukuran piece yg melebihi ukura board	

--

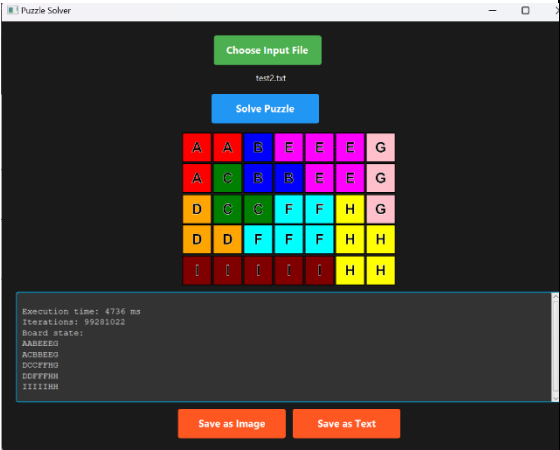
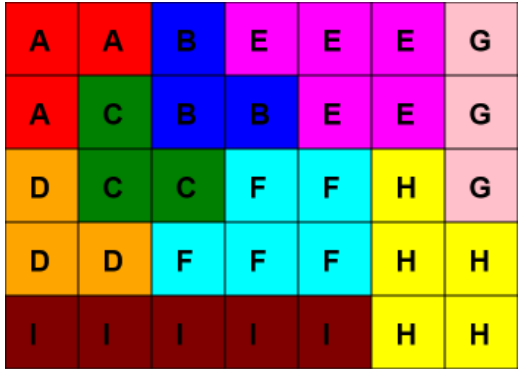
5.4

Test Uji 4

Input File	Output GUI
<pre> 5 5 7 DEFAULT A AA B BB C CC D DD EE EE E FF FF F GGG </pre>	
Output Text (.txt)	Output Image (.png)
<pre> 1 Solution: 2 AGGGD 3 AABDD 4 CCBBE 5 CFFEE 6 FFFEE 7 8 Execution time: 204 ms 9 Iterations: 1868438 10 </pre>	 <p>Execution time: 204 ms Iterations: 1868438</p>
<p>Keterangan:</p> <p>Ada solusi</p>	

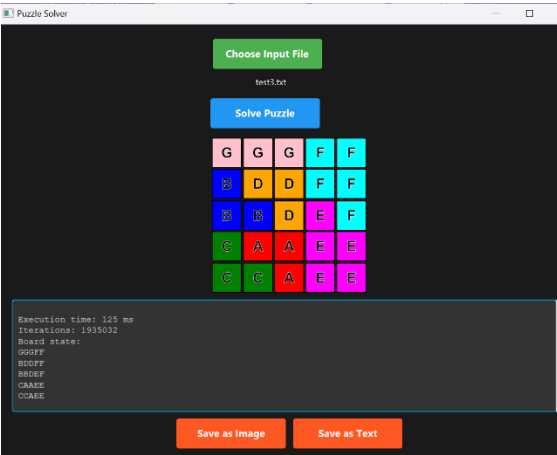

5.5

Test Uji 5

Input File	Output GUI
<pre> 5 7 9 DEFAULT AA A B BB C CC D DD EE EE E FF FF F GGG HHH HH IIIII </pre>	
Output Text (.txt)	Output Image (.png)
<pre> 1 Solution: 2 AABEEEG 3 ACBBEEG 4 DCCFFHG 5 DDFFFHH 6 IIIIIHH 7 8 Execution time: 5343 ms 9 Iterations: 99281022 0 </pre>	 <p>Execution time: 5343 ms Iterations: 99281022</p>
<p>Keterangan:</p> <p>Ada solusi</p>	

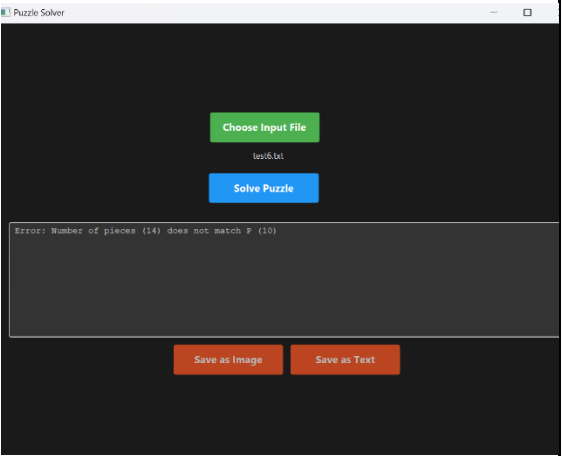
5.6

Test Uji 6

Input File	Output GUI
<pre> 5 5 7 DEFAULT GGG B BB C CC E EE EE D DD FF FF F A AA </pre>	 <p>Execution time: 125 ms Iterations: 1935032 Board state: GGGFF BDDFF BBDEF CAAEE CCAEE</p>
Output Text (.txt)	Output Image (.png)
<pre> Solution: GGGFF BDDFF BBDEF CAAEE CCAEE Execution time: 97 ms Iterations: 1935032 </pre>	 <p>Execution time: 97 ms Iterations: 1935032</p>
<p>Keterangan:</p> <p>Ada solusi</p>	

5.7

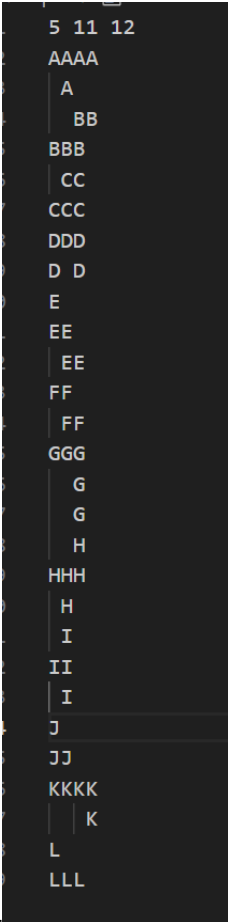
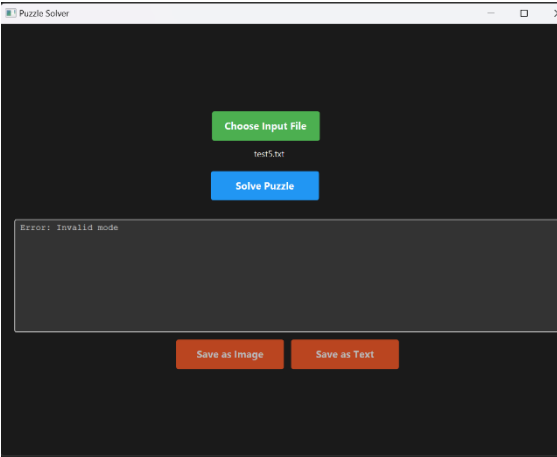
Test Uji 7

Input File	Output GUI
<pre> 5 5 10 DEFAULT GGG EE BB C FF CC E A EE D DD FF F B AA 1 </pre>	
Output Text (.txt)	Output Image (.png)
-	-
Keterangan: Jumlah piece berbeda sehingga tidak valid	

5.8

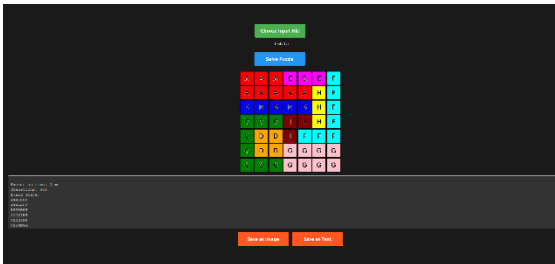
Test Uji 8

Input File	Output GUI
------------	------------

	
Output Text (.txt)	Output Image (.png)
-	-
Keterangan: Tidak ad mode hingga input tidak valid.	

5.9 Test Uji 9

Input File	Output GUI
------------	------------

<div><div>7 7 9</div><div>DEFAULT</div><div>AAA</div><div>AAAAA</div><div>BBBBB</div><div>CCC</div><div>C</div><div>C</div><div>CCC</div><div>EEE</div><div>FFF</div><div>F</div><div>F</div><div>F</div><div>F</div><div>GGGG</div><div>GGGG</div><div>HHH</div><div>I</div><div>II</div><div>DD</div><div>DD</div></div>																																																		
Output Text (.txt)	Output Image (.png)																																																	
<div><div>Solution:</div><div>AAAEFEF</div><div>AAAAAHF</div><div>BBBBBHF</div><div>CCCIHF</div><div>CDDIFFF</div><div>CDDGGGG</div><div>CCCGGGG</div><div>Execution time: 0 ms</div><div>Iterations: 931</div></div>	<div><table><tr><td>A</td><td>A</td><td>A</td><td>E</td><td>E</td><td>E</td><td>F</td></tr><tr><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td><td>H</td><td>F</td></tr><tr><td>B</td><td>B</td><td>B</td><td>B</td><td>B</td><td>H</td><td>F</td></tr><tr><td>C</td><td>C</td><td>C</td><td>I</td><td>I</td><td>H</td><td>F</td></tr><tr><td>C</td><td>D</td><td>D</td><td>I</td><td>F</td><td>F</td><td>F</td></tr><tr><td>C</td><td>D</td><td>D</td><td>G</td><td>G</td><td>G</td><td>G</td></tr><tr><td>C</td><td>C</td><td>C</td><td>G</td><td>G</td><td>G</td><td>G</td></tr></table><div>Execution time: 0 ms Iterations: 931</div></div>	A	A	A	E	E	E	F	A	A	A	A	A	H	F	B	B	B	B	B	H	F	C	C	C	I	I	H	F	C	D	D	I	F	F	F	C	D	D	G	G	G	G	C	C	C	G	G	G	G
A	A	A	E	E	E	F																																												
A	A	A	A	A	H	F																																												
B	B	B	B	B	H	F																																												
C	C	C	I	I	H	F																																												
C	D	D	I	F	F	F																																												
C	D	D	G	G	G	G																																												
C	C	C	G	G	G	G																																												
<div>Keterangan:</div> <div>Ada solusi</div>																																																		

Lampiran

Tabel Kelengkapan Spesifikasi

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

Link github : https://github.com/WwzFwz/Tucil1_13523065