

LAPORAN TUGAS KECIL 1
PENYELESAIAN IQ PUZZLER PRO DENGAN ALGORITMA
BRUTE FORCE

Disusun untuk Memenuhi Tugas Kecil 1 Mata Kuliah Strategi Algoritma IF2211

Dosen Pengampu:

Dr. Ir. Rinaldi Munir, M.T.



Karol Yangqian Poetracahya

13523093

Kelas K2

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2025

BAB I

DESKRIPSI MASALAH

1.1. Deskripsi Masalah

IQ Puzzler Pro adalah permainan papan produksi Smart Games dengan tujuan pemain harus dapat mengisi seluruh papan dengan *piece* atau blok *puzzle* yang diberikan. Terdapat dua komponen penting dari permainan IQ Puzzler Pro:

1. Board (Papan)

Board merupakan komponen permainan berupa bidang dengan lubang-lubang yang tersusun membentuk kisi-kisi (grid) dengan baris dan kolom. Sebuah board dengan konfigurasi 2D berukuran M baris dan N kolom memiliki lubang sebanyak $M \times N$. Setiap lubang memiliki bentuk identik dan simetris sehingga sebuah *piece* dapat diletakkan di atas board dengan berbagai orientasi.

2. Blok/Piece

Piece adalah sebuah komponen permainan yang akan disusun di atas board. Sebuah *piece* tersusun atas segmen-segmen dengan bentuk yang sesuai dengan lubang pada board dan setiap segmen pada satu *piece* terhubung satu sama lain membentuk sebuah struktur geometris yang dapat diletakkan di atas board dengan berbagai orientasi. Beberapa *piece* dengan berbagai bentuk dan warna dapat disusun untuk mengisi board sebagian maupun hingga penuh.

Pada program ini, permainan dimulai dengan board yang kosong dan diberikan sejumlah *piece* dengan berbagai bentuk. Program melakukan percobaan secara iteratif untuk meletakkan *piece* di atas board sedemikian sehingga tidak ada *piece* yang tumpang tindih. Setiap *piece* dapat dirotasikan maupun dicerminkan untuk setiap percobaan. Program mencoba semua konfigurasi peletakkan *piece* satu per satu menggunakan algoritma *brute force* hingga ditemukan konfigurasi yang mengisi penuh board dan menggunakan seluruh *piece* yang diberikan. Apabila konfigurasi tersebut tidak ditemukan, maka program menampilkan bahwa board dan *piece* yang diberikan tidak memiliki solusi. Board yang terisi penuh terlihat seperti pada gambar 1.



Gambar 1. Board yang terisi penuh

BAB II

IMPLEMENTASI DAN ALGORITMA

2.1. Implementasi Program

Program ini ditulis dengan paradigma berorientasi objek. Board adalah sebuah objek yang memiliki *list of pieces*, *matrix of booleans* yang merepresentasikan konfigurasi awal board yang kosong, dan *matrix of booleans* yang merepresentasikan lokasi apa saja pada board yang belum diisi dengan segmen sebuah piece. Di sisi lain, piece adalah sebuah objek dengan bentuk dan posisi pada board yang direpresentasikan sebagai *list of integer pairs* (koordinat). Piece juga memiliki atribut id, simbol, dan keadaan orientasinya yang bernilai 0 sampai 7. Sebuah objek piece dapat diletakkan pada board jika dan hanya jika posisi dan orientasinya valid, yakni semua segmen berada di dalam board dan tidak tumpang tindih dengan piece lain yang sudah ada di dalam board. Algoritma *brute force* untuk penyelesaian masalah diimplementasikan dalam fungsi solve dalam kelas statik Solver. Dibuat juga kelas-kelas lain untuk mendukung penulisan kode seperti Pair, Vec2I, ProgramInput, Result, Ansi256, Mode, dan FileHandle. Namun, laporan ini hanya akan memaparkan kelas-kelas yang relevan pada domain masalah, yakni Board2D, Piece2D, Solver, dan Main. Berikut adalah rancangan dasar kelas-kelas tersebut:

1. Board2D

Atribut	Deskripsi
<code>private final ArrayList<Piece2D> pieces;</code>	Menyimpan piece yang berhasil diletakkan di board
<code>private final boolean[][] board;</code>	Konfigurasi awal board kosong (true: lubang; false: tembok)
<code>private final int rows;</code>	Jumlah baris board
<code>private final int cols;</code>	Jumlah kolom board
<code>private boolean[][] availablePlaces;</code>	Lokasi yang sudah terisi oleh piece pada board (true: lubang kosong; false: terisi piece atau tembok)

Metode	Deskripsi
<code>public boolean isSolved()</code>	Metode ini memeriksa apakah board sudah diselesaikan atau belum dengan mengecek apakah setiap elemen <code>availablePlaces</code> bernilai false.
<code>public boolean placePiece(Piece2D piece)</code>	Metode ini memeriksa apakah piece dapat diletakkan di board atau tidak. Jika bisa, piece ditambahkan ke atribut <code>pieces</code> . Metode mengembalikan true jika piece berhasil diletakkan dan false jika tidak.
<code>public void popLastPiece()</code>	Menghapus piece terakhir dalam list dan mengubah elemen pada <code>availablePlaces</code> yang bersesuaian menjadi true
<code>public void printBoardStandard(Map<Integer, Integer> pieceColors)</code>	Menampilkan board dan konfigurasi piece di dalamnya pada terminal.

2. Piece2D

Atribut	Deskripsi
<code>private final int id;</code>	Nomor unik tiap piece
<code>private final String symbol;</code>	Simbol piece yang akan di- <i>print</i>
<code>private final Vec2I[] shape;</code>	Koordinat setiap segmen piece pada board
<code>private int state;</code>	Keadaan orientasi (0-3: keadaan setia dirotasikan 90 derajat; 4-7: pencerminan terhadap keadaan 0-3)

Metode	Deskripsi
<code>public String getSymbol()</code>	Metode ini mengembalikan simbol piece.
<code>public void setState(int r, int c, int state)</code>	Metode ini meletakkan piece pada koordinat (r, c) dengan keadaan rotasi dan pencerminan state.
<code>public void rotateOnce()</code>	Metode ini merotasikan piece 90 derajat berlawanan arah jarum jam dengan koordinat segmen pertama sebagai sumbu putar.
<code>public void flipHorizontal()</code>	Metode ini mencerminkan piece terhadap garis horizontal yang melalui koordinat segmen pertama.
<code>public void setPosition(int x, int y)</code>	Metode ini menentukan posisi baru piece pada board. Posisi piece dinyatakan oleh koordinat segmen pertama piece.

3. Solver

Atribut	Deskripsi
<code>private static long iterations = 0;</code>	Jumlah iterasi
<code>private static long startTime = 0;</code>	Waktu awal algoritma dijalankan
<code>private static Map<Integer, Integer> colorMap;</code>	Mapping warna simbol piece
<code>private static Board2D board;</code>	Board yang dilibatkan pada program
<code>private static ArrayList<Piece2D> givenPieces;</code>	Piece yang diberikan
<code>private static boolean colorMapSet = false;</code>	Status apakah mapping warna sudah diberikan
<code>private static Result result;</code>	Hasil program yang berisi data board, waktu yang berlalu, jumlah iterasi, dan apakah board berhasil

	diselesaikan
--	--------------

Metode	Deskripsi
<pre>public static boolean solve(ProgramInput programInput, Board2D boardToSolve, boolean printResult, boolean printProgress)</pre>	Metode ini adalah fungsi utama yang dipanggil pada program main untuk melakukan penyelesaian permainan. Metode solve memanggil metode attempt untuk menjalankan algoritma utama.
<pre>private static boolean attempt(int pieceIndex, int r, int c, int rot, boolean printProgress)</pre>	Metode ini mengimplementasikan algoritma <i>brute force</i> dengan pendekatan rekursif untuk menyelesaikan permainan.
<pre>public static void setColorMap(String filename)</pre>	Metode ini membaca file mapping warna dan menyimpannya sebagai objek Map pada atribut statik kelas Solver.

4. Main

Metode	Deskripsi
<pre>public static void main(String[] args)</pre>	Metode ini memanggil <code>startProgram()</code> .
<pre>public static void startProgram()</pre>	Metode ini berisi alur utama keseluruhan program dan menampilkan antarmuka pengguna.

2.2. Algoritma

Brute force adalah pendekatan yang lurus dan lempang (*straightforward*) untuk pemecahan suatu masalah. Algoritma ini bersifat sederhana, langsung, jelas caranya, dan mudah dipahami. Biasanya algoritma *brute force* ditulis berdasarkan pernyataan dalam persoalan (*problem statement*) dan definisi atau konsep yang dilibatkan. Berikut adalah algoritma penyelesaian permainan dengan pendekatan *brute force*:

1. Program menerima input ukuran dan konfigurasi board serta jumlah dan bentuk setiap piece yang diberikan dengan cara membacanya dari file berekstensi .txt.
2. Koordinat ditetapkan menjadi (0, 0) dan keadaan orientasi menjadi 0.
3. Jika seluruh piece sudah diletakkan, kembalikan true jika board sudah penuh atau false jika tidak (pengembalian false mengakibatkan pemanggilan fungsi rekursi lagi untuk memeriksa konfigurasi lain).
4. Piece dicoba untuk diletakkan pada koordinat koordinat dan keadaan orientasi saat ini.
5. Jika piece berhasil diletakkan sebagai pada langkah 4, masuk ke langkah 6. Jika tidak, langsung lanjut ke langkah 7.
6. Kembali ke langkah 3 untuk piece berikutnya secara rekursif. Jika semua sisa piece berhasil diletakkan, kembalikan true jika board penuh atau false jika tidak. Jika tidak semua sisa piece dapat diletakkan, hapus/angkat kembali piece terakhir.
7. Ubah piece ke keadaan berikutnya lalu kembali ke langkah 3 secara rekursif. Dalam hal ini, keadaan berikutnya adalah keadaan posisi (koordinat) dan orientasi yang direpresentasikan sebagai bilangan 0 sampai 7 seperti pada rancangan kelas.

Algoritma di atas akan mencoba semua kemungkinan posisi dan orientasi setiap piece hingga mencapai konfigurasi yang membuat board penuh dan semua piece terpakai.

BAB III

SOURCE CODE

3.1. Repository Program

Tautan menuju repository program adalah sebagai berikut:

https://github.com/karolyangqian/Tucil1_13523093

3.2. *Source Code* Program

3.2.1. Board2D.java

```
package iqpuzzlerpro;
import java.util.*;

public class Board2D {
    private final ArrayList<Piece2D> pieces;
    private final boolean[][] board;
    private final int rows;
    private final int cols;
    private boolean[][] availablePlaces;

    public Board2D(int rows, int cols) {
        this.rows = rows;
        this.cols = cols;
        this.board = new boolean[rows][cols];
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                board[i][j] = true;
            }
        }
        this.pieces = new ArrayList<>();
        this.availablePlaces = board.clone();
    }

    public Board2D(int rows, int cols, boolean[][] board) {
        this.rows = rows;
        this.cols = cols;
        this.board = board;
        this.pieces = new ArrayList<>();
        this.availablePlaces = board.clone();
    }
}
```

```

    }

    public Board2D copy() {
        boolean[][] newBoardArray = new boolean[board.length][];
        for (int i = 0; i < board.length; i++) {
            newBoardArray[i] = board[i].clone();
        }

        boolean[][] newAvailablePlacesArray = new
boolean[availablePlaces.length][];
        for (int i = 0; i < availablePlaces.length; i++) {
            newAvailablePlacesArray[i] = board[i].clone();
        }

        Board2D newBoard = new Board2D(rows, cols, newBoardArray);
        newBoard.availablePlaces = newAvailablePlacesArray;

        for (Piece2D piece : pieces) {
            if (!newBoard.placePiece(piece)) {
                System.err.println("Error copying piece");
            }
        }
        return newBoard;
    }

    public int getRows() {
        return rows;
    }

    public int getCols() {
        return cols;
    }

    public void clearPieces() {
        pieces.clear();
        for (int i = 0; i < rows; i++) {
            System.arraycopy(board[i], 0, availablePlaces[i], 0, cols);
        }
    }
}

```

```

public boolean[][] getEmptyBoard() {
    return board.clone();
}

private boolean segmentIsPlacable(Vec2I segment) {
    return segment.x >= 0 && segment.x < rows && segment.y >= 0 &&
segment.y < cols && availablePlaces[segment.x][segment.y];
}

public void printPieces() {
    for (Piece2D p : pieces) {
        p.printShape();
    }
}

public boolean piecePlacable(Piece2D piece) {
    for (Vec2I v : piece.getShape()) {
        if (!segmentIsPlacable(v)) {
            return false;
        }
    }
    return true;
}

public boolean placePiece(Piece2D piece) {
    boolean placable = piecePlacable(piece);
    if (placable) {
        for (Vec2I v : piece.getShape()) {
            availablePlaces[v.x][v.y] = false;
        }
        pieces.add(piece);
    }
    return placable;
}

public void popLastPiece() {

```

```

        Piece2D lastPiece = pieces.get(pieces.size()-1);
        for (Vec2I v : lastPiece.getShape()) {
            availablePlaces[v.x][v.y] = true;
        }
        this.pieces.removeLast();
    }

    public void printEmptyBoard(String empty, String wall) {
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                if (board[i][j]) {
                    System.out.print(empty);
                } else {
                    System.out.print(wall);
                }
            }
            System.out.println();
        }
    }

    public int[][] getBoardMatrix() {
        int[][] boardMatrix = new int[rows][cols]; // -1 for wall, 0 for
empty, id for piece
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                boardMatrix[i][j] = board[i][j] ? 0 : -1;
            }
        }
        for (int i = pieces.size()-1; i >= 0; i--) {
            for (Vec2I v : pieces.get(i).getShape()) {
                boardMatrix[v.x][v.y] = pieces.get(i).getId();
            }
        }
        return boardMatrix;
    }

    public void printBoard(boolean toFile, String path, Map<Integer,
Integer> pieceColors, String empty, String wall, int background, boolean
printId) {

```

```

        int[][] boardMatrix = getBoardMatrix();
        Map<Integer, Piece2D> pieceMap = new HashMap<>();
        for (Piece2D piece : pieces) {
            pieceMap.put(piece.getId(), piece);
        }

        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                switch (boardMatrix[i][j]) {
                    case -1 -> System.out.print(wall);
                    case 0 -> System.out.print(empty);
                    default ->
                        System.out.print(Ansi256.coloredText(pieceMap.get(boardMatrix[i][j]).get
                            Symbol(), pieceColors.get(boardMatrix[i][j]), background));
                }
            }
            System.out.println();
        }

        if (printId) {
            for (int i = 0; i < rows; i++) {
                for (int j = 0; j < cols; j++) {
                    //
                    System.out.print(Ansi256.coloredText(String.valueOf(boardMatrix[i][j]),
                        pieceColors.get(boardMatrix[i][j]), background));
                    System.out.print(boardMatrix[i][j] + " ");
                }
                System.out.println();
            }
        }
    }

    public void printBoard(Map<Integer, Integer> pieceColors, String
empty, String wall, int background) {
        printBoard(false, "", pieceColors, empty, wall, background,
false);
    }

    public void printBoardStandard(Map<Integer, Integer> pieceColors) {

```

```

        printBoard(pieceColors, Ansi256.coloredText("■", 0, 0),
Ansi256.coloredText("■", 15, 15), 0);
    }

    public boolean isSolved() {
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                if (availablePlaces[i][j] == true) return false;
            }
        }
        return true;
    }

    public Piece2D[] getPieces() {
        return pieces.toArray(Piece2D[]::new);
    }

    public void printAvailablePlaces() {
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                System.out.print(availablePlaces[i][j] ? "1" : "0");
            }
            System.out.println();
        }
    }
}

```

3.2.2. Piece2D.java

```

package iqpuzzlerpro;
import java.util.*;

public class Piece2D {

    private final int id;
    private final String symbol;
    private final Vec2I[] shape;

```

```
private int state;

public Piece2D(int id, String symbol, Vec2I[] shape) {
    this.id = id;
    this.symbol = symbol;
    this.shape = shape;
    this.state = 0;
}

public int getId() {
    return id;
}

public String getSymbol() {
    return symbol;
}

public Vec2I[] getShape() {
    Vec2I[] shapeCopy = new Vec2I[shape.length];
    for (int i = 0; i < shape.length; i++) {
        shapeCopy[i] = new Vec2I(shape[i].x, shape[i].y);
    }
    return shapeCopy;
}

public void rotateOnce() {
    for (Vec2I v : shape) {
        v.rotateOnceAroundCenter(shape[0]);
    }
}

public boolean contains(Vec2I v) {
    for (Vec2I vec : shape) {
        if (vec.equals(v)) {
            return true;
        }
    }
    return false;
}
```

```

public void print(int rows, int cols, int offsetX, int offsetY) {
    printShape();
    Piece2D tempPiece = this.copy();
    tempPiece.setPosition(new Vec2I(offsetX, offsetY));
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            Vec2I v = new Vec2I(i, j);
            if (tempPiece.contains(v)) {
                System.out.print(symbol);
            } else {
                System.out.print(" ");
            }
        }
        System.out.println();
    }
}

public void printShape() {
    System.out.print("Piece " + id + " " + symbol + ": ");
    for (Vec2I v : shape) {
        System.out.print("(" + v.x + ", " + v.y + ") ");
    }
    System.out.println();
}

public Vec2I getPosition() {
    return shape[0].copy();
}

public final void setPosition(Vec2I position) {
    Vec2I offset = new Vec2I(shape[0].x, shape[0].y);
    for (Vec2I v : shape) {
        v.x -= offset.x;
        v.y -= offset.y;
    }
    for (Vec2I v : shape) {
        v.x += position.x;
        v.y += position.y;
    }
}

```



```

    }
}

public void setPosition(int x, int y) {
    setPosition(new Vec2I(x, y));
}

public void flipHorizontal() {
    for (Vec2I v : shape) {
        v.mirrorX(shape[0].x);
    }
}

public Piece2D copy() {
    Vec2I[] newShape = new Vec2I[shape.length];
    for (int i = 0; i < shape.length; i++) {
        newShape[i] = new Vec2I(shape[i].x, shape[i].y);
    }
    return new Piece2D(id, symbol, newShape);
}

public static Piece2D stringArrayToPiece2D(int newId, String
newSymbol, String[] array) {
    ArrayList<Vec2I> newShapeList = new ArrayList<>();
    for (int i = 0; i < array.length; i++) {
        String[] split = array[i].split("");
        for (int j = 0; j < split.length; j++) {
            if (!split[j].equals(" ")) {
                newShapeList.add(new Vec2I(i, j));
            }
        }
    }
    return new Piece2D(newId, newSymbol,
newShapeList.toArray(Vec2I[]::new));
}

public int getState() {
    return state;
}

```

```

    public void setState(int r, int c, int state) {
        setPosition(r, c);
        int rot = state % 4;
        int flip = state / 4;
        for (int i = 0; i < rot; i++) {
            rotateOnce();
        }
        if (flip == 1) flipHorizontal();
        this.state = state;
    }
}

```

3.2.3. Solver.java

```

package iqpuzzlerpro;
import java.io.IOException;
import java.util.*;

public class Solver {

    private static long iterations = 0;
    private static long startTime = 0;
    private static Map<Integer, Integer> colorMap;
    private static Board2D board;
    private static ArrayList<Piece2D> givenPieces;
    private static int rows;
    private static int cols;
    private static int numOfPieces;
    private static final int NUM_OF_ROT_STATES = 8;
    private static boolean colorMapSet = false;
    private static Result result;

    public static void setColorMap(String filename) throws IOException {
        try {
            colorMap = FileHandle.readColorMap(filename);
            colorMapSet = true;
        } catch (Exception e) {

```

```

        throw e;
    }
}

public static Result getResult() {
    return new Result(result.isSolved, result.board.copy(),
result.time, result.iterations);
}

public static boolean solve(ProgramInput programInput, Board2D
boardToSolve, boolean printResult, boolean printProgress) {
    givenPieces = programInput.pieces;
    numOfPieces = givenPieces.size();
    board = boardToSolve.copy();
    rows = programInput.rows;
    cols = programInput.cols;
    iterations = 0;
    result = new Result();

    if (!colorMapSet) {
        colorMap = new HashMap<>();
        for (Piece2D piece : givenPieces) {
            colorMap.put(piece.getId(), 15);
        }
    }

    startTime = System.nanoTime();
    boolean solved = attempt(0, 0, 0, 0, printProgress);
    long endTime = System.nanoTime();

    double timeTaken = (double) (endTime - startTime) / 1e6;

    if (solved) {
        result = new Result(true, board.copy(), timeTaken,
iterations);
    } else {
        result = new Result(false, board.copy(), timeTaken,
iterations);
    }
}

```

```

        if (printResult) {
            if (solved) {
                board.printBoardStandard(colorMap);
                System.out.printf("\nSolved in %.3f ms with %d
iterations\n", timeTaken, iterations);
            } else {
                // board.clearPieces();
                System.out.printf("\nCannot solve the board with given
pieces. Checked %d iterations in %.3f ms\n", iterations, timeTaken);
            }
        }
        return solved;
    }

    public static boolean solve(ProgramInput programInput, Board2D
boardToSolve) {
        return solve(programInput, boardToSolve, false, false);
    }

    public static boolean solve(ProgramInput programInput, Board2D
boardToSolve, boolean printResult) {
        return solve(programInput, boardToSolve, printResult, false);
    }

    private static void printProgress(long startTime) {
        long elapsedTime = System.nanoTime() - startTime;
        board.printBoardStandard(colorMap);
        System.out.printf("Iterations: %d | Time: %.3f ms\n",
iterations, elapsedTime / 1e9);
    }

    private static boolean attempt(int pieceIndex, int r, int c, int
rot, boolean printProgress) {
        if (pieceIndex == numOfPieces) {
            return board.isSolved();
        }

        if (printProgress) {

```

```

        if (iterations % 1000 == 0) {
            printProgress(startTime);
            System.err.printf("Piece %d %s on (%d, %d) with rot
%d\n", pieceIndex, givenPieces.get(pieceIndex).getSymbol(), r, c, rot);
            System.err.println("Hold Enter to continue...");
            Scanner sc = new Scanner(System.in);
            String input = sc.nextLine();
        }
    }

    iterations++;

    if (rot == NUM_OF_ROT_STATES){
        return attempt(pieceIndex, r, c+1, 0, printProgress);
    }
    if (c == cols) {
        return attempt(pieceIndex, r+1, 0, rot, printProgress);
    }
    if (r == rows) {
        return false;
    }

    Piece2D currentPiece = givenPieces.get(pieceIndex).copy();

    currentPiece.setState(r, c, rot);

    if (board.placePiece(currentPiece)) {
        // System.err.printf("Piece %d placed\n", pieceIndex);
        if (attempt(pieceIndex+1, 0, 0, 0, printProgress)) return
board.isSolved();
        else board.popLastPiece();
    }

    return attempt(pieceIndex, r, c, rot+1, printProgress);
}
}

```

3.2.4. Main.java

```
import iqpuzzlerpro.*;
import java.awt.*;
import java.io.File;
import java.util.*;

public class Main {

    public static boolean yesNoPrompt(String prompt) {
        System.err.println(prompt);
        String input;
        Scanner sc = new Scanner(System.in);
        input = sc.nextLine();
        if (input.toLowerCase().equals("y") ||
input.toLowerCase().equals("n")) {
            return input.toLowerCase().equals("y");
        } else {
            System.err.println("Invalid input. Defaulting to no.");
        }
        return false;
    }

    public static String pathPrompt(String prompt, String defaultPath) {
        System.err.println(prompt);
        String input;
        Scanner sc = new Scanner(System.in);
        input = sc.nextLine();
        File file = new File(input);
        while (!file.exists() && input.length() > 0) {
            System.err.println("File does not exist. Enter the path to
an existing file (default: " + defaultPath + "):");
            input = sc.nextLine();
            file = new File(input);
        }

        if (input.length() > 0) {
```

```

        return input;
    }
    return defaultPath;
}

public static void saveFileDialog(Result result) {
    Frame frame = new Frame();
    FileDialog fileDialog = new FileDialog(frame, "Save result to
file", FileDialog.SAVE);
    fileDialog.setFile("result.txt");
    fileDialog.setVisible(true);

    String directory = fileDialog.getDirectory();
    String filename = fileDialog.getFile();

    if (directory != null && filename != null) {
        if (!filename.toLowerCase().endsWith(".txt")) {
            filename += ".txt";
        }
        File file = new File(directory, filename);

        try {
            FileHandle.saveResult(result, file.getAbsolutePath());
            System.out.println("File saved successfully as " +
file.getAbsolutePath());
        } catch (Exception e) {
            System.err.println(e);
        }
    }
    frame.dispose();
}

public static void startProgram() {

    System.out.println("\nIQPuzzlerPro Solver!!!\n\n");

    ProgramInput programInput;
    String path = "test/input.txt";

```

```

        while (true) {
            try {
                path = pathPrompt("Enter the path to the test case file
(default:" + path + "):", path);
                programInput = FileHandle.readProgramInput(path);
                break;
            } catch (Exception e) {
                System.err.println(e.getMessage());
            }
        }

        programInput.print();
        System.err.println();

        String colorMapPath = "src/colors/colors.txt";
        Map<Integer, Integer> colorMap;

        while (true) {
            try {
                colorMapPath = pathPrompt("Enter the path to the color
map file (default:" + colorMapPath + "):", colorMapPath);
                colorMap = FileHandle.readColorMap(colorMapPath);
                Solver.setColorMap(colorMapPath);
                break;
            } catch (Exception e) {
                System.err.println(e.getMessage());
            }
        }

        boolean printResult = yesNoPrompt("Print result? (y/n):");
        boolean printProgress = yesNoPrompt("Print progress? (y/n):");

        Board2D board = new Board2D(programInput.rows,
programInput.cols, programInput.board.getEmptyBoard());

        Scanner sc = new Scanner(System.in);
        System.err.println("Press Enter to start...");
        String input = sc.nextLine();

```



```
        boolean solved = Solver.solve(programInput, board, printResult,
printProgress);

        if (solved) {
            System.err.println("Solved!");
        } else {
            System.err.println("Not solved.");
        }

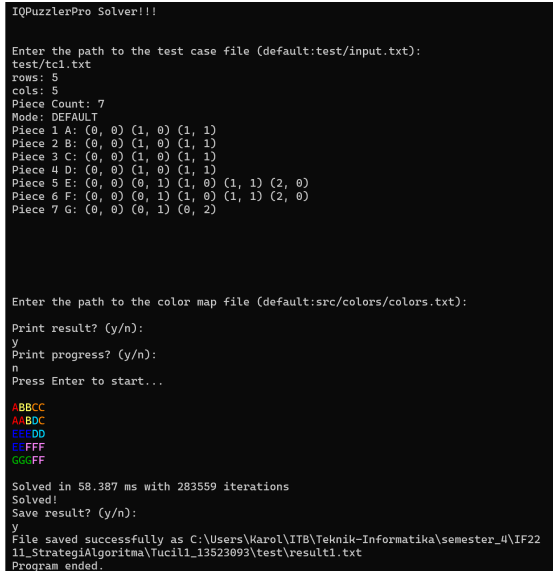
        boolean saveResult = yesNoPrompt("Save result? (y/n):");
        if (saveResult) {
            saveFileDialog(Solver.getResult());
        }
        System.err.println("Program ended.");
    }

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

BAB IV

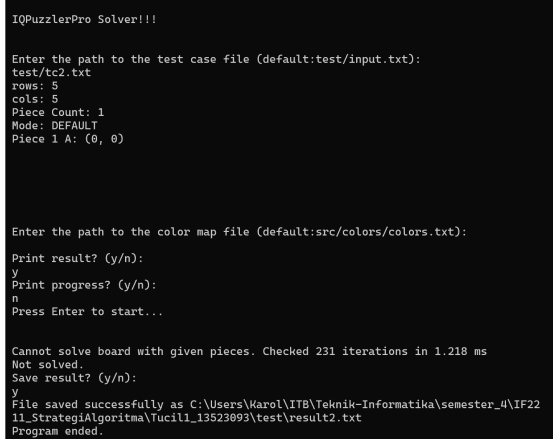
EKSPERIMEN

4.1. Test Case 1 - Kasus Normal

Masukan	Keluaran
5 5 7 DEFAULT A AA B BB C CC D DD EE EE E FF FF F GGG	 <pre> IQPuzzlerPro Solver!!! Enter the path to the test case file (default:test/input.txt): test/tc1.txt rows: 5 cols: 5 Piece Count: 7 Mode: DEFAULT Piece 1 A: (0, 0) (1, 0) (1, 1) Piece 2 B: (0, 0) (1, 0) (1, 1) Piece 3 C: (0, 0) (1, 0) (1, 1) Piece 4 D: (0, 0) (1, 0) (1, 1) Piece 5 E: (0, 0) (0, 1) (1, 0) (1, 1) (2, 0) Piece 6 F: (0, 0) (0, 1) (1, 0) (1, 1) (2, 0) Piece 7 G: (0, 0) (0, 1) (0, 2) Enter the path to the color map file (default:src/colors/colors.txt): Print result? (y/n): y Print progress? (y/n): n Press Enter to start... ABBCC AABDC EEEDD EEFFF GGGFF Solved in 58.387 ms with 283559 iterations Solved! Save result? (y/n): y File saved successfully as C:\Users\Warol\ITB\Teknik-Informatika\semester_4\IF22 11_StrategiAlgoritma\Tucil1_13523093\test\result1.txt Program ended. </pre> test/result1.txt Solved in 58.3869 ms with 283559 iterations 5 5 7 ABBCC AABDC EEEDD EEFFF GGGFF 1 2 2 3 3 1 1 2 4 3 5 5 5 4 4 5 5 6 6 6 7 7 7 6 6 1 A 0 0 0 2 B 0 1 5 3 C 0 3 5 4 D 1 3 0

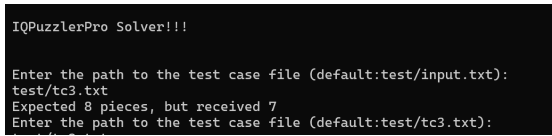
	5 E 2 0 5 6 F 3 4 3 7 G 4 0 0
--	--

4.2. Test Case 2 - Jumlah Lubang Board Lebih Banyak daripada Jumlah Segmen Piece

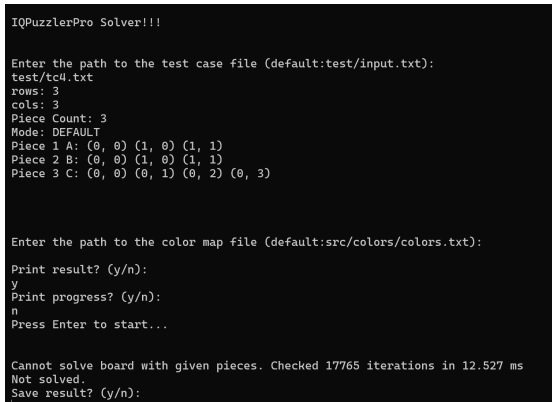
Masukan	Keluaran
5 5 1 DEFAULT A	 <pre> IQPuzzlerPro Solver!!! Enter the path to the test case file (default:test/input.txt): test/tc2.txt rows: 5 cols: 5 Piece Count: 1 Mode: DEFAULT Piece 1 A: (0, 0) Enter the path to the color map file (default:src/colors/colors.txt): Print result? (y/n): y Print progress? (y/n): n Press Enter to start... Cannot solve board with given pieces. Checked 231 iterations in 1.218 ms Not solved. Save result? (y/n): y File saved successfully as C:\Users\Warol\ITB\Teknik-Informatika\semester_4\IF22 11.StrategiAlgoritma\Lucili_13523093\test\result2.txt Program ended. </pre> test/result1.txt Cannot solve board with given pieces. Checked 231 iterations in 1.2178 ms 5 5 0 0

4.3. Test Case 3 - Jumlah Piece Tidak Sesuai

Masukan	Keluaran
---------	----------

5 5 8 DEFAULT A AA B BB C CC D DD EE EE E FF FF F GGG	 <p>Keterangan: program akan meminta terus path file dari pengguna hingga input valid atau program dimatikan.</p>
---	---

4.4. Test Case 4 - Jumlah Lubang Board Lebih Sedikit daripada Jumlah Segmen Piece

Masukan	Keluaran
3 3 3 DEFAULT A AA B BB CCCC	 <p>test/result4.txt Cannot solve board with given pieces. Checked 17765 iterations in 12.5265 ms 3 3 0 0 0 0 0 0 0 0 0 0</p>

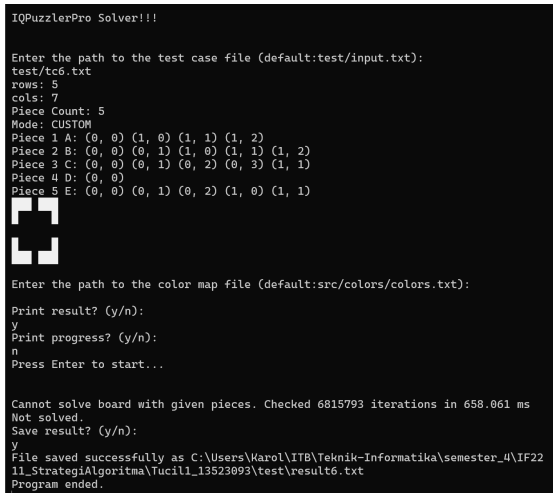
--	--

4.5. Test Case 5 - Board Custom Berhasil Diselesaikan

Masukan	Keluaran
<pre> 5 7 5 CUSTOM ...X... .XXXXX. XXXXXXXX .XXXXX. ...X... A AAA BB BBB CCCC C D EEE E </pre>	 <pre> test/result5.txt Solved in 32.9736 ms with 141014 iterations 5 7 5 ...A... .BBAAA. BBBCCCC .EEECD. ...E... -1 -1 -1 1 -1 -1 -1 -1 2 2 1 1 1 -1 2 2 2 3 3 3 3 -1 5 5 5 3 4 -1 -1 -1 -1 5 -1 -1 -1 1 A 0 3 0 2 B 1 2 6 3 C 2 3 0 4 D 3 5 0 5 E </pre>

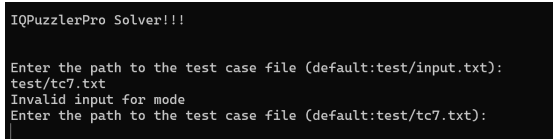
	3 3 6
--	-------

4.6. Test Case 6 - Board Custom Gagal Diselesaikan

Masukan	Keluaran
5 7 5 CUSTOM ...X... .XXXXX. XXXXXXXX .XXXXX. ...X... A AAA BB BBB CCCC C D EEE EE	 <pre> test/result6.txt Cannot solve board with given pieces. Checked 6815793 iterations in 658.0614 ms 5 7 0 -1 -1 -1 0 -1 -1 -1 -1 0 0 0 0 -1 0 0 0 0 0 0 -1 0 0 0 0 -1 </pre>

	-1 -1 -1 0 -1 -1 -1
--	---------------------

4.7. Test Case 7 - Mode Tidak Valid

Masukan	Keluaran
5 7 5 CUSTOMMMM ...X... .XXXXX. XXXXXXXXX .XXXXX. ...X... A AAA BB BBB CCCC C D EEE E	 <pre> IQPuzzlerPro Solver!!! Enter the path to the test case file (default:test/input.txt): test/tc7.txt Invalid input for mode Enter the path to the test case file (default:test/tc7.txt): </pre> <p>Keterangan: program akan meminta terus path file dari pengguna hingga input valid atau program dimatikan.</p>

BAB V
LAMPIRAN

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	✓	

BAB VI

REFERENSI

Munir, R. (2025). Algoritma brute force (Bagian 1) [Materi kuliah]. Institut Teknologi Bandung.
Diakses dari
[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/02-Algoritma-Brute-Force-\(2025\)-Bag1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/02-Algoritma-Brute-Force-(2025)-Bag1.pdf)