LAPORAN TUGAS KECIL 1
IF2211 STRATEGI ALGORITMA

PENYELESAIAN IQ PUZZLER PRO DENGAN ALGORITMA BRUTE-FORCE

Oleh:

Darrel Adinarya Sunanda
13523061

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2025

Daftar Isi

BAB I
Deskripsi Program

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 di mana 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**.

(Dikutip dari: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/Tucil1-Stima-2025.pdf)

Gambar I.1 Permainan IQ Puzzler Pro
(Sumber: https://www.smartgamesusa.com)

BAB II
Algoritma Brute-force

Pada program ini, digunakan algoritma **brute-force** untuk mencari solusi peletakan semua blok pada papan permainan. Algoritma ini mencoba semua kemungkinan posisi setiap blok dengan berbagai rotasi dan orientasi hingga menemukan susunan yang sesuai.

Adapun langkah-langkah yang digunakan dalam algoritma ini adalah sebagai berikut:

1. Inisialisasi dan Pengecekan Basis Rekursi
    - Program menerima papan permainan (board) dan daftar blok (blocks) yang harus ditempatkan.
    - Jika semua blok telah berhasil ditempatkan (index == blocks.size()), maka papan yang valid dikembalikan sebagai hasil.
2. Iterasi untuk Menempatkan Blok (**Brute-force**)
    - Program mengambil blok ke-index dari daftar blok.
    - Untuk setiap blok, program mencoba dua kemungkinan orientasi (normal dan terbalik) menggunakan operasi flip.
    - Setiap orientasi diuji dengan 4 kemungkinan rotasi (0°, 90°, 180°, dan 270°) menggunakan operasi rotate.
    - Untuk setiap rotasi, program mencoba menempatkan blok di setiap posisi (x, y) yang memungkinkan di papan.
3. Rekursi untuk Mencari Solusi
    - Jika blok berhasil ditempatkan, program akan memanggil kembali fungsi solve secara rekursif untuk menempatkan blok berikutnya.
    - Jika semua blok berhasil ditempatkan, papan yang valid akan dikembalikan sebagai solusi.
    - Jika tidak ada solusi ditemukan untuk posisi saat ini, program akan mencoba rotasi atau orientasi lain dari blok.
4. Backtracking
    - Jika suatu konfigurasi tidak menghasilkan solusi, program akan kembali ke langkah sebelumnya (backtrack) dan mencoba kemungkinan lain.
    - Jika setelah mencoba semua kemungkinan tidak ditemukan solusi, maka fungsi akan mengembalikan null.
    - Variabel iterationCount digunakan untuk menghitung jumlah percobaan yang dilakukan selama proses pencarian solusi.

Algoritma ini mencoba semua kemungkinan posisi untuk setiap blok dengan berbagai rotasi dan orientasi sehingga kompleksitasnya adalah sebuah polinom :

$$\boldsymbol{n \times 2 \times 4 \times W \times H}$$

Dengan W dan H sebagai lebar dan tinggi papan permainan.

BAB III
Source Code

```java
import javax.imageio.ImageIO;
import java.awt.*;
import java.awt.image.*;
import java.io.*;
import java.util.*;
import java.util.List;
```

Gambar III.1 Daftar import

```java
record Case(Board board, List<Block> blocks, String path) {}
```

Gambar III.2 Record Case

```java
class Block {
    public char[][] block;

    // Constructor
    public Block(String[] parts) {
        int width = parts[0].length();
        for (int i = 1; i < parts.length; i++) {
            if (parts[i].length() > width) {
                width = parts[i].length();
            }
        }

        int height = parts.length;
        this.block = new char[height][width];

        for (int i = 0; i < height; i++) {
            for (int j = 0; j < width; j++) {
                if (j >= parts[i].length() || parts[i].charAt(j) == ' ') {
                    block[i][j] = '_';
                } else {
                    block[i][j] = parts[i].charAt(j);
                }
            }
        }
    }

    // Get the width and height of the block
    public int width() { return block[0].length; }
    public int height() { return block.length; }

    // Rotate the block
    public void rotate() {
        char[][] temp = new char[width()][height()];
        for (int i = 0; i < height(); i++) {
            for (int j = 0; j < width(); j++) {
                temp[j][height() - i - 1] = block[i][j];
            }
        }
        block = temp;
    }

    // Flip the block
    public void flip() {
        char [][] temp = new char[height()][width()];
        for (int i = 0; i < height(); i++) {
            for (int j = 0; j < width(); j++) {
                temp[i][width() - j - 1] = block[i][j];
            }
        }
        block = temp;
    }

}
```

Gambar III.3 Kelas Block

```java
class Board {
    private final char[][] board;

    // Constructor
    public Board(int N, int M) {
        this.board = new char[N][M];
        for (char[] row : board) {
            Arrays.fill(row, '_');
        }
    }

    // Copy constructor
    public Board(Board board) {
        this.board = new char[board.board.length][board.board[0].length];
        for (int i = 0; i < board.board.length; i++) {
            System.arraycopy(board.board[i], 0, this.board[i], 0, board.board[i].length);
        }
    }

    // Get the width and height of the board
    public int width() { return board[0].length; }
    public int height() { return board.length; }

    // Print the board
    public void printBoard() {
        for (char[] row : board) {
            for (char c : row) {
                switch (c) {
                    case 'A' -> System.out.print("\u001B[31m" + c + " \u001B[0m"); // Red
                    case 'B' -> System.out.print("\u001B[32m" + c + " \u001B[0m"); // Green
                    case 'C' -> System.out.print("\u001B[33m" + c + " \u001B[0m"); // Yellow
                    case 'D' -> System.out.print("\u001B[34m" + c + " \u001B[0m"); // Blue
                    case 'E' -> System.out.print("\u001B[35m" + c + " \u001B[0m"); // Magenta
                    case 'F' -> System.out.print("\u001B[36m" + c + " \u001B[0m"); // Cyan
                    case 'G' -> System.out.print("\u001B[37m" + c + " \u001B[0m"); // White
                    case 'H' -> System.out.print("\u001B[90m" + c + " \u001B[0m"); // Bright Black
                    case 'I' -> System.out.print("\u001B[91m" + c + " \u001B[0m"); // Bright Red
                    case 'J' -> System.out.print("\u001B[92m" + c + " \u001B[0m"); // Bright Green
                    case 'K' -> System.out.print("\u001B[93m" + c + " \u001B[0m"); // Bright Yellow
                    case 'L' -> System.out.print("\u001B[94m" + c + " \u001B[0m"); // Bright Blue
                    case 'M' -> System.out.print("\u001B[95m" + c + " \u001B[0m"); // Bright Magenta
                    case 'N' -> System.out.print("\u001B[96m" + c + " \u001B[0m"); // Bright Cyan
                    case 'O' -> System.out.print("\u001B[97m" + c + " \u001B[0m"); // Bright White
                    case 'P' -> System.out.print("\u001B[1;31m" + c + " \u001B[0m"); // Bold Red
                    case 'Q' -> System.out.print("\u001B[1;32m" + c + " \u001B[0m"); // Bold Green
                    case 'R' -> System.out.print("\u001B[1;33m" + c + " \u001B[0m"); // Bold Yellow
                    case 'S' -> System.out.print("\u001B[1;34m" + c + " \u001B[0m"); // Bold Blue
                    case 'T' -> System.out.print("\u001B[1;35m" + c + " \u001B[0m"); // Bold Magenta
                    case 'U' -> System.out.print("\u001B[1;36m" + c + " \u001B[0m"); // Bold Cyan
                    case 'V' -> System.out.print("\u001B[1;37m" + c + " \u001B[0m"); // Bold White
                    case 'W' -> System.out.print("\u001B[1;90m" + c + " \u001B[0m"); // Bold Bright Black
                    case 'X' -> System.out.print("\u001B[1;91m" + c + " \u001B[0m"); // Bold Bright Red
                    case 'Y' -> System.out.print("\u001B[1;92m" + c + " \u001B[0m"); // Bold Bright Green
                    case 'Z' -> System.out.print("\u001B[1;93m" + c + " \u001B[0m"); // Bold Bright Yellow
                    default -> System.out.print(c + " ");
                }
            }
            System.out.println();
        }
    }


    // Place a block on the board
    public boolean placeBlock(Block block, int x, int y) {
        // Position check
        if (x < 0 || y < 0 || x + block.width() > board[0].length || y + block.height() > board.length) {
            return false;
        }

        // Intersection check
        for (int i = 0; i < block.height(); i++) {
            for (int j = 0; j < block.width(); j++) {
                if (block.block[i][j] != '_' && board[y + i][x + j] != '_') {
                    return false;
                }
            }
        }

        // Place the block
        for (int i = 0; i < block.height(); i++) {
            for (int j = 0; j < block.width(); j++) {
                if (block.block[i][j] != '_') {
                    board[y + i][x + j] = block.block[i][j];
                }
            }
        }
        return true;
    }

    public boolean isNotFull() {
        for (char[] rows : board) {
            for (char c : rows) {
                if (c == '_') {
                    return true;
                }
            }
        }
        return false;
    }

    public String[] toStrings() {
        String[] strings = new String[board.length];
        for (int i = 0; i < board.length; i++) {
            strings[i] = new String(board[i]);
        }
        return strings;
    }

    public char getBlockAt(int x, int y) {
        return board[y][x];
    }

}
```

Gambar III.4 Kelas Board

```
class Parser {

    private static char getFirstNonSpace(String str) {
        for (int i = 0; i < str.length(); i++) {
            if (str.charAt(i) != ' ') {
                return str.charAt(i);
            }
        }
        return ' ';
    }

    @SuppressWarnings({"BusyWait"})
    public static Case getCase() throws InterruptedException {

        while (true) {
            try {
                BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
                System.out.print("Masukkan path dari file txt: ");
                String path = br.readLine();

                BufferedReader fileReader = new BufferedReader(new FileReader(path));

                // Read N, M, P, S
                String line = fileReader.readLine();

                String[] dimensions = line.split(" ");
                if (dimensions.length != 3) { // Check if the dimensions are valid
                    throw new IllegalArgumentException("Invalid dimensions '" + line + "'!");
                }
                int N, M, P, S;
                try {
                    N = Integer.parseInt(dimensions[0]);
                    M = Integer.parseInt(dimensions[1]);
                    P = Integer.parseInt(dimensions[2]);
                } catch (NumberFormatException e) {
                    throw new IllegalArgumentException("Invalid dimensions '" + line + "'!");
                }

                // Input validation for N, M, P
                if (P < 1) {
                    throw new IllegalArgumentException("Number of blocks cannot be less than 1! (" + P + ")!");
                } if (P > 26) {
                    throw new IllegalArgumentException("Number of blocks > 26! (" + P + ")!");
                } if (N < 1 || M < 1) {
                    throw new IllegalArgumentException("Invalid board dimensions (" + N + "x" + M + ")!");
                }

                // Read the case type
                line = fileReader.readLine();
                switch (line) {
                    case "DEFAULT" -> S = 0;
                    case "CUSTOM" -> S = 1;
                    case "PYRAMID" -> S = 2;
                    default -> throw new IllegalArgumentException("Invalid case type '" + line + "'!");
                }

                // Exit for unimplemented pyramid case
                if (S == 2) {
                    System.err.println("Sorry! Haven't implemented pyramid cases yet. :(");
                    System.exit(0);
                }

                // Process the blocks
                int blockCount = 0;
                List<String> parts = new ArrayList<>();
                List<Block> blocks = new ArrayList<>();
                List<Character> letters = new ArrayList<>();
                for (int i = 'A'; i <= 'Z'; i++) {
                    letters.add((char) i);
                }

                // Process the custom case
                Block customBlock = null;
                if (S == 1) {
                    for (int i = 0; i < N; i++) {
                        line =  fileReader.readLine();
                        if (line.length() != M) {
                            throw new IllegalArgumentException("Invalid custom board dimensions at line " + (i+3) + "!");
                        }

                        char[] lineArr = line.toCharArray();
                        for (int j = 0; j < M; j++) {
                            if (lineArr[j] == 'X') {lineArr[j] = ' ';}
                            else if (lineArr[j] != '.') {
                                throw new IllegalArgumentException("Invalid custom configuration at '" + line + "'!");
                            }
                        }
                        line = new String(lineArr);
                        parts.add(line);
                    }
                    customBlock = new Block(parts.toArray(new String[0]));
                    parts.clear();
                }

                // Iterate over every line in the blocks
                line = fileReader.readLine();
                while (line != null) {
                    // Use the first non-whitespace character of the line as the block letter
                    char currentBlockLetter = getFirstNonSpace(line);
                    // Check if the block letter is a valid character
                    if (!letters.remove((Character) currentBlockLetter)) {
                        if (line.isEmpty()) {
                            throw new IllegalArgumentException("Problematic whitespaces present!");
                        } else {
                            throw new IllegalArgumentException("Invalid block part '" + line + "'!");
                        }
                    }
                    // Continue checking the block
                    while (line != null && getFirstNonSpace(line) == currentBlockLetter) {
                        // Iterate over every character in the line
                        for (int i = 0; i < line.length(); i++) {
                            // Check if the block part is valid (only contains the block letter or whitespace)
                            if (line.charAt(i) != currentBlockLetter && line.charAt(i) != ' ') {
                                throw new IllegalArgumentException("Invalid block part '" + line + "'!");
                            }
                        }
                        // Add the block part after checking the characters
                        parts.add(line);
                        line = fileReader.readLine();
                    }
                    // Add the block to the list after checking the block parts
                    blocks.add(new Block(parts.toArray(new String[0])));
                    blockCount++;
                    // Clear the block parts after adding the block and continue to the next block
                    parts.clear();
                }
                fileReader.close();
                // Check if the number of blocks is valid
                if (blockCount != P) {
                    throw new IllegalArgumentException("Invalid number of blocks (P=" + P + " vs " + blockCount + ")!");
                }

                // Create the board
                Board board = new Board(N,M);
                if (S == 1) { board.placeBlock(customBlock, 0, 0); }
                return new Case(board, blocks, path);
            }

            catch (IOException e) {
                System.err.println("\nThe system cannot find the file specified!");
                System.err.println("Please check the path and try again.");
            }
            catch (IllegalArgumentException e) {
                System.err.println("\n" + e.getMessage());
                System.err.println("Please check the file and try again.");
            }

            Thread.sleep(1000);
            System.out.print(".");
            Thread.sleep(1000);
            System.out.print(".");
            Thread.sleep(1000);
            System.out.print(".\n\n");
        }
    }

    public static void saveSolution(Board board, long time, int iterations, String path) {
        path = path.replace(".txt", "_solution.txt");
        try (FileWriter myWriter = new FileWriter(path)) {
            for (String row: board.toStrings()) {
                myWriter.write(row + "\n");
            }
            myWriter.write("\nWaktu pencarian: " + time + "ms\n");
            myWriter.write("\nBanyak kasus yang ditinjau: " + iterations);

        } catch (IOException e) {
            System.err.println("\nAn error occurred while trying to save the solution!");
        }
    }
}
```

Gambar III.5 Kelas Parser untuk membaca input file

```
class Painter {

    private static Color charColor(char letter) {
        return switch (letter) {
            case 'A' -> Color.RED;
            case 'B' -> Color.BLUE;
            case 'C' -> Color.GREEN;
            case 'D' -> Color.YELLOW;
            case 'E' -> Color.ORANGE;
            case 'F' -> Color.PINK;
            case 'G' -> Color.CYAN;
            case 'H' -> Color.MAGENTA;
            case 'I' -> Color.LIGHT_GRAY;
            case 'J' -> Color.DARK_GRAY;
            case 'K' -> new Color(128, 0, 0); // Maroon
            case 'L' -> new Color(0, 128, 0); // Dark Green
            case 'M' -> new Color(0, 0, 128); // Navy
            case 'N' -> new Color(128, 128, 0); // Olive
            case 'O' -> new Color(128, 0, 128); // Purple
            case 'P' -> new Color(0, 128, 128); // Teal
            case 'Q' -> new Color(255, 165, 0); // Orange
            case 'R' -> new Color(255, 105, 180); // Hot Pink
            case 'S' -> new Color(75, 0, 130); // Indigo
            case 'T' -> new Color(139, 69, 19); // Saddle Brown
            case 'U' -> new Color(210, 105, 30); // Chocolate
            case 'V' -> new Color(46, 139, 87); // Sea Green
            case 'W' -> new Color(0, 206, 209); // Dark Turquoise
            case 'X' -> new Color(255, 140, 0); // Dark Orange
            case 'Y' -> new Color(70, 130, 180); // Steel Blue
            case 'Z' -> new Color(123, 104, 238); // Medium Slate Blue
            default -> Color.BLACK;
        };
    }

    public static void saveImage(Board board, String path) {
        BufferedImage image = new BufferedImage(20*board.width(),
20*board.height(), BufferedImage.TYPE_INT_RGB);
        Graphics2D g = image.createGraphics();

        g.setColor(Color.WHITE);
        g.fillRect(0, 0, 20*board.width(), 20*board.height());

        for (int x = 0; x < board.width(); x++) {
            for (int y = 0; y < board.height(); y++) {
                g.setColor(charColor(board.getBlockAt(x, y)));
                g.fillRect(x*20, y*20, 20, 20);
            }
        }

        File output = new File(path.replace(".txt", "_solution.png"));
        try {
}
```

Gambar III.6 Kelas Painter untuk menghasilkan gambar dari solusi

```java
public class Main {

    private static int iterationCount = 0;

    private static Board solve(Board board, List<Block> blocks, int index) {
        if (index == blocks.size()) {
            return board;
        }

        Block block = blocks.get(index);
        for (int orientation = 0; orientation < 2; orientation++) {
            for (int rotation = 0; rotation < 4; rotation++) {
                for (int y = 0; y < board.height(); y++) {
                    for (int x = 0; x < board.width(); x++) {
                        iterationCount++;
                        Board newBoard = new Board(board);
                        if (newBoard.placeBlock(block, x, y)) {
                            Board result = solve(newBoard, blocks, index + 1);
                            if (result != null) {
                                return result;
                            }
                        }
                    }
                }
                block.rotate();
            }
            block.flip();
        }
        return null;
    }

    public static void main(String[] args) throws InterruptedException {

        System.out.println("\n=== Penyelesaian IQ Puzzler Pro dengan Algoritma Brute Force ===\n");

        Case initialState = Parser.getCase();

        long startTime = System.currentTimeMillis();
        Board solution = solve(initialState.board(), initialState.blocks(), 0);
        long endTime = System.currentTimeMillis();

        if (solution != null) {
            solution.printBoard();
            // Lazy solution compared to checking before brute-forcing
            if (solution.isNotFull()) {
                System.out.println("Papan tidak dapat terisi penuh dengan semua balok!");
            }
        } else {
            System.out.println("Tidak ada solusi yang ditemukan!");
        }

        System.out.println("\nWaktu pencarian: " + (endTime - startTime) + "ms");
        System.out.println("\nBanyak kasus yang ditinjau: " + iterationCount);
        if (solution == null) {System.exit(0);}

        Scanner sc = new Scanner(System.in);
        while (true) {
            System.out.print("\nApakah anda ingin menyimpan solusi");
            if (solution.isNotFull()) {System.out.print(" parsial");}
            System.out.print("? (ya/tidak): ");

            String input = sc.nextLine();
            if (input.equals("ya")) {
                Parser.saveSolution(solution, endTime - startTime, iterationCount, initialState.path());
                Painter.saveImage(solution, initialState.path());
                System.out.println("\nSolusi berhasil disimpan pada " +
                        initialState.path().replace(".txt", "_solution")
                        + "!\nTerima kasih telah menggunakan program ini! :3");
                System.exit(0);
            } else if (input.equals("tidak")) {
                System.out.println("\nSolusi tidak akan disimpan.\nTerima kasih telah menggunakan program ini! :3");
                System.exit(0);
            }
        }
    }
}
```

Gambar III.7 Kelas Main yang mengandung algoritma brute-force

BAB IV
Eksperimen

Berikut adalah beberapa contoh keluaran program pada berbagai uji kasus:

1. Masukan tidak valid

a) Masukan dengan dimensi invalid

Input

```
3 3
DEFAULT
A
A
BB
CCC
C
D
```

```
=== Penyelesaian IQ Puzzler Pro dengan Algoritma Brute Force ===

Masukkan path dari file txt: test/bad.txt

Invalid dimensions '3 3'!
Please check the file and try again.
...
```

b) Masukan dengan dimensi non-angka

Input

```
3 F 4
DEFAULT
A
A
BB
CCC
C
D
```

```
=== Penyelesaian IQ Puzzler Pro dengan Algoritma Brute Force ===

Masukkan path dari file txt: test/bad.txt

Invalid dimensions '3 F 4'!
Please check the file and try again.
...
```

c) Masukan dengan dimensi papan negatif

Input

```
3 -3 4
DEFAULT
A
A
BB
CCC
C
D
```

```
=== Penyelesaian IQ Puzzler Pro dengan Algoritma Brute Force ===

Masukkan path dari file txt: test/bad.txt

Invalid board dimensions (3x-3)!
Please check the file and try again.
...
```

d) Masukan dengan jumlah balok < 1

Input

```
3 3 0
DEFAULT
A
A
BB
CCC
C
D
```

```
=== Penyelesaian IQ Puzzler Pro dengan Algoritma Brute Force ===

Masukkan path dari file txt: test/bad.txt

Number of blocks cannot be less than 1! (0)!
Please check the file and try again.
...
```

e) Masukan dengan tipe kasus invalid

Input

```
3 3 4
AMIMIR
A
A
BB
CCC
C
D
```

```
=== Penyelesaian IQ Puzzler Pro dengan Algoritma Brute Force ===

Masukkan path dari file txt: test/bad.txt

Invalid case type 'AMIMIR'!
Please check the file and try again.
...
```

f) Masukan dengan balok terputus

Input

```
3 3 4
AMIMIR
A

A
BB
CCC
C
D
```

```
=== Penyelesaian IQ Puzzler Pro dengan Algoritma Brute Force ===

Masukkan path dari file txt: test/bad.txt

Problematic whitespaces present!
Please check the file and try again.
...
```

g) Masukan dengan balok tercampur

Input

```
3 3 4
DEFAULT
A
A
ABB
CCC
C
D
```

```
=== Penyelesaian IQ Puzzler Pro dengan Algoritma Brute Force ===

Masukkan path dari file txt: test/bad.txt

Invalid block part 'ABB'!
Please check the file and try again.
...
```

h) Masukan dengan jumlah balok ≠ P

Input

```
3 3 4
DEFAULT
A
A
BB
CCC
C
D
EE
EE
FF
 F
```

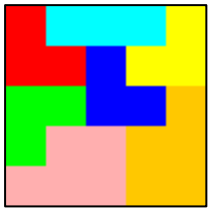
```
=== Penyelesaian IQ Puzzler Pro dengan Algoritma Brute Force ===

Masukkan path dari file txt: test/bad.txt

Invalid number of blocks (P=4 vs 6)!
Please check the file and try again.
...
```

2. Masukan valid

i) Masukan kasus DEFAULT pada spesifikasi

Input

```
5 5 7
DEFAULT
A
AA
B
BB
CC
C
 D
DD
EE
EE
E
FF
FF
F
GGG
```

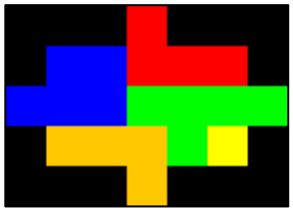


```
Masukkan path dari file txt: test/test.txt
A G G G D
A A B D D
C C B B E
C F F E E
F F F E E

Waktu pencarian: 1125ms

Banyak kasus yang ditinjau: 5198831
```

j) Masukan kasus CUSTOM pada spesifikasi

Input

```
5 7 5
CUSTOM
...X...
.XXXXX.
XXXXXXX
.XXXXX.
...X...
A
AAA
BB
BBB
CCCC
 C
D
EEE
E
```



```
Masukkan path dari file txt: test/custom.txt
. . . A . . .
. B B A A A .
B B B C C C C
. E E E C D .
. . . E . . .

Waktu pencarian: 44ms

Banyak kasus yang ditinjau: 165000
```

k) Masukan kasus papan non-persegi

Input

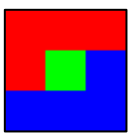
```
3 5 5
DEFAULT
AAAAA
BB
CC
 C
DD
D
E
 E
```

```
Masukkan path dari file txt: test/rect.txt
A A A A A
D B B E C
D D E C C

Waktu pencarian: 4ms

Banyak kasus yang ditinjau: 8866
```

l) Masukan kasus yang membutuhkan *flip*

Input

```
3 3 3
DEFAULT
AAA
A
B
BBB
C
```

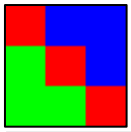


```
Masukkan path dari file txt: test/flip.txt
A A A
A C B
B B B

Waktu pencarian: 0ms

Banyak kasus yang ditinjau: 46
```

m) Masukan kasus dengan balok diagonal

Input

```
3 3 3
DEFAULT
A
 A
  A
BB
B
CC
 C
```

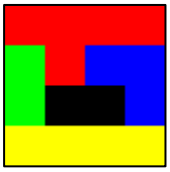


```
Masukkan path dari file txt: test/diagonal.txt
A B B
C A B
C C A

Waktu pencarian: 0ms

Banyak kasus yang ditinjau: 34
```

n) Masukan kasus dengan balok tidak mencukupi untuk mengisi papan

Input

```
4 4 4
DEFAULT
AAAA
 A
BB
 B
C
C
DDDD
```

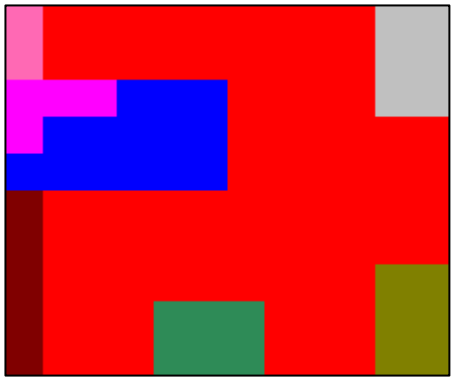


```
Masukkan path dari file txt: test/weird.txt
A A A A
C A B B
C _ _ B
D D D D
Papan tidak dapat terisi penuh dengan semua balok!

Waktu pencarian: 1ms

Banyak kasus yang ditinjau: 26

Apakah anda ingin menyimpan solusi parsial? (ya/tidak):
```

o) Masukan AMOGUS

Input

```
10 12 8
DEFAULT
AAAAAAAAA
AAAAAAAAA
   AAAA
   AAAAAA
   AAAAAA
AAAAAAAAAAA
AAAAAAAAAAA
AAAAAAAAA
AAA  AAA
AAA  AAA
HHH
H
  BBB
 BBBBB
BBBBBB
RR
KKKKK
VVV
VVV
III
III
NNN
NNN
```



```
Masukkan path dari file txt: test/sus.txt
R A A A A A A A A A I I
R A A A A A A A A A I I
H H H B B B A A A A I I
H B B B B B A A A A A A
B B B B B B A A A A A A
K A A A A A A A A A A A
K A A A A A A A A A A A
K A A A A A A A A A N N
K A A A V V V A A A N N
K A A A V V V A A A N N

Waktu pencarian: 3525ms

Banyak kasus yang ditinjau: 1664481
```

BAB V
Pranala Repositori

Berikut adalah tautan menuju repositori yang digunakan dalam pembuatan tugas ini:
https://github.com/Darsua/Tucil1_13523061

BAB VI
Referensi

Berikut adalah referensi laporan yang digunakan dalam pembuatan laporan ini:
https://github.com/mrsyaban/24-Solver/blob/main/doc/Laporan_Tucil1.pdf

BAB VII
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 *Graphical User Interface* (GUI)		
6	Program dapat menyimpan solusi dalam bentuk file gambar		
7	Program dapat menyelesaikan kasus konfigurasi *custom*		
8	Program dapat menyelesaikan kasus konfigurasi Piramida (3D)		
9	Program dibuat oleh saya sendiri		