

Laporan Tugas Kecil 1

IQ Puzzler Pro Solver

Algoritma Brute Force



Disusun oleh:

Guntara Hambali

13523114

**PROGRAM STUDI TEKNIK INFORMATIKA SEKOLAH TEKNIK ELEKTRO
DAN INFORMATIKA INSTITUT TEKNOLOGI BANDUNG JL. GANESA 10,
BANDUNG 40132**

Bab I

Algoritma

Berikut adalah algoritma atau langkah-langkah yang ditempuh untuk mendapatkan solusi pada IQ Puzzler pro. Secara umum, saya mengambil inspirasi dari penyelesaian permainan sudoku:

1. Tempatkan satu blok pada ujung kiri atas papan
2. Lalu, tempatkan satu blok lainnya setelah blok sebelumnya. Cek apakah blok tersebut tumpang-tindih dengan blok sebelumnya. Jika tidak tumpang-tindih, kembali ke langkah 1 dengan blok lain dan posisi setelahnya. Jika tumpang-tindih, ke langkah 3.
3. Coba variasi lain dari blok tersebut (rotasi dan pencerminan dengan total variasi sebanyak 8). Apabila blok tersebut sudah tidak tumpang-tindih, kembali ke langkah 1 dengan blok lain di posisi setelahnya. Jika masih tumpang-tindih, artinya blok tersebut tidak dapat digunakan di posisi tersebut. Dicoba menggunakan blok lain dan variasinya.
4. Jika blok lain masih belum bisa mengisi tempat tersebut, artinya harus dilakukan mundur satu blok ke blok sebelumnya dan ulangi dari langkah 1 hingga ditemukan solusinya.
5. Ulangi langkah-langkah di atas hingga papan terisi penuh dengan blok dan blok tidak tersisa lagi.

Bab II

Source Code

Berikut adalah implementasi algoritma brute force pada IQ Puzzler Pro solver yang saya buat menggunakan bahasa pemrograman java:

```
package puzzleSolver;
import java.awt.image.BufferedImage;
import java.io.File;
import java.util.*;
import java.util.Scanner;

class Solver {
    private final int boardWidth;
    private final int boardHeight;
    private final List<List<String>> rawPieces;
    private final char[][] board;
    private final List<Block> blocks = new ArrayList<>();
    private int casesConsidered = 0;
    private double executionTimeMs = 0;

    // ANSI color codes for different letters
    private static final String[] COLORS = {
        "\u001B[31m", // Red
        "\u001B[32m", // Green
        "\u001B[33m", // Yellow
        "\u001B[34m", // Blue
        "\u001B[35m", // Magenta
        "\u001B[36m", // Cyan
        "\u001B[37m", // White
        "\u001B[91m", // Bright Red
        "\u001B[92m", // Bright Green
        "\u001B[93m", // Bright Yellow
        "\u001B[94m", // Bright Blue
        "\u001B[95m", // Bright Magenta
        "\u001B[96m", // Bright Cyan
        "\u001B[97m" // Bright White
    };

    private static final String RESET = "\u001B[0m";

    public Solver(int boardWidth, int boardHeight, List<List<String>> rawPieces) {
        this.boardWidth = boardWidth;
        this.boardHeight = boardHeight;
        this.board = new char[boardHeight][boardWidth];
        this.rawPieces = rawPieces;
        initializeBoard();
        initializeBlocks(rawPieces);
    }
}
```

```

    }

    public char[][] getBoard() { return board; }
    public void setExecutionTimeMs(double executionTimeMs) { this.executionTimeMs =
executionTimeMs;}
    public double getExecutionTimeMs() { return executionTimeMs;}
    public int getCasesConsidered(){ return casesConsidered; }

    private void initializeBoard() {
        for (int i = 0; i < boardHeight; i++) {
            Arrays.fill(board[i], '.'); // Use '.' for empty spaces
        }
    }

    private void initializeBlocks(List<List<String>> rawPieces) {
        rawPieces.remove(0); // First element contains board dimensions
        for (List<String> rawPiece : rawPieces) {
            blocks.add(new Block(rawPiece));
        }
    }

    private boolean canPlacePiece(int[][] piece, int row, int col) {
        for (int i = 0; i < piece.length; i++) {
            for (int j = 0; j < piece[i].length; j++) {
                if (piece[i][j] == 1) {
                    int newRow = row + i;
                    int newCol = col + j;
                    if (newRow >= boardHeight || newCol >= boardWidth || board[newRow][newCol] !=
'.') {
                        return false;
                    }
                }
            }
        }
        return true;
    }

    private void placePiece(int[][] piece, int row, int col, int pieceIndex) {
        char pieceLetter = rawPieces.get(pieceIndex).get(0).charAt(0); //get unique letter form
input txt

        for (int i = 0; i < piece.length; i++) {
            for (int j = 0; j < piece[i].length; j++) {
                if (piece[i][j] == 1) {
                    board[row + i][col + j] = pieceLetter;
                }
            }
        }
    }

```

```

    }
}

private void removePiece(int[][] piece, int row, int col) {
    for (int i = 0; i < piece.length; i++) {
        for (int j = 0; j < piece[i].length; j++) {
            if (piece[i][j] == 1) {
                board[row + i][col + j] = '.'; // Reset cell
            }
        }
    }
}

private boolean isBoardFull() {
    for (char[] row : board) {
        for (char cell : row) {
            if (cell == '.') return false;
        }
    }
    return true;
}

public boolean solve(int pieceIndex) {
    casesConsidered++; // Count recursive calls

    // If all blocks are placed, board is full
    if (pieceIndex >= blocks.size()) {
        return isBoardFull();
    }
    Block block = blocks.get(pieceIndex);
    boolean placedAtLeastOnce = false;

    for (int[][] pieceVariant : block.getVariances()) {
        for (int row = 0; row < boardHeight; row++) {
            for (int col = 0; col < boardWidth; col++) {
                if (canPlacePiece(pieceVariant, row, col)) {
                    placedAtLeastOnce = true;
                    placePiece(pieceVariant, row, col, pieceIndex);

                    if (solve(pieceIndex + 1)) {
                        return true;
                    }

                    removePiece(pieceVariant, row, col);
                }
            }
        }
    }
}

```

```

    }
}

// If no placement, no solution
if (!placedAtLeastOnce) {
    return false;
}

return false;
}

public void solvePuzzle() {
    long startTime = System.nanoTime();

    if (solve(0)) {
        long endTime = System.nanoTime();
        executionTimeMs = (endTime - startTime) / 1_000_000.0;
        System.out.println("Solution found:");
        printBoard();
        System.out.println("Execution time: " + executionTimeMs + " ms");
        System.out.println("Total cases considered: " + casesConsidered);

        Scanner scanner = new Scanner(System.in);

        // Ask to save the solution as text
        System.out.print("Do you want to save the solution to a text file? (yes/no): ");
        String response = scanner.nextLine().trim().toLowerCase();
        if (response.equals("yes")) {
            IO.saveSolutionAsFile(this.board, executionTimeMs, casesConsidered);
            System.out.println("Solution saved to solution.txt.");
        } else {
            System.out.println("Solution not saved.");
        }

        // Ask to save the solution as image
        System.out.print("Do you want to save the solution as an image? (yes/no): ");
        response = scanner.nextLine().trim().toLowerCase();
        if (response.equals("yes")) {
            BufferedImage image = IO.makeImage(this.board);
            IO.saveSolutionAsImage(image, new File("solution.png"));
            System.out.println("Solution saved as solution.png.");
        } else {
            System.out.println("Solution image not saved.");
        }
    } else {
        System.out.println("No solution found.");
    }
}

```

}
}
}

Bab III

Test Case

Berikut merupakan beragam kasus uji yang ditujukan untuk menguji keberjalanan program dan fitur-fiturnya. Gambar dalam pdf mungkin terkompresi. Oleh karena itu, saya sediakan dokumen docx di repositori github.

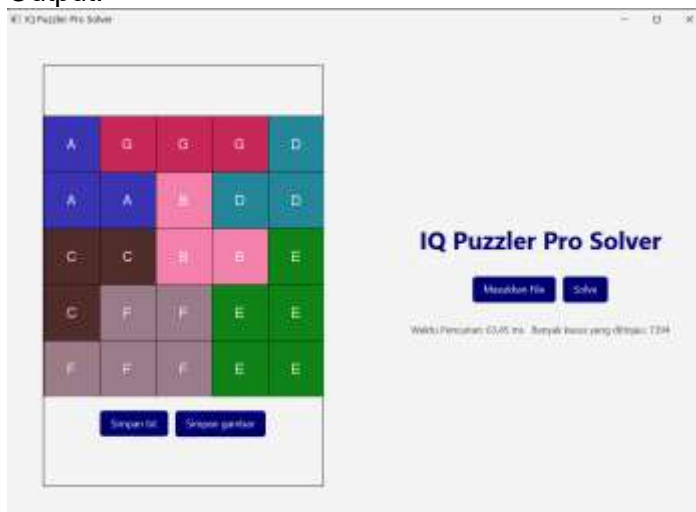
A. Kasus Uji 1

Input:



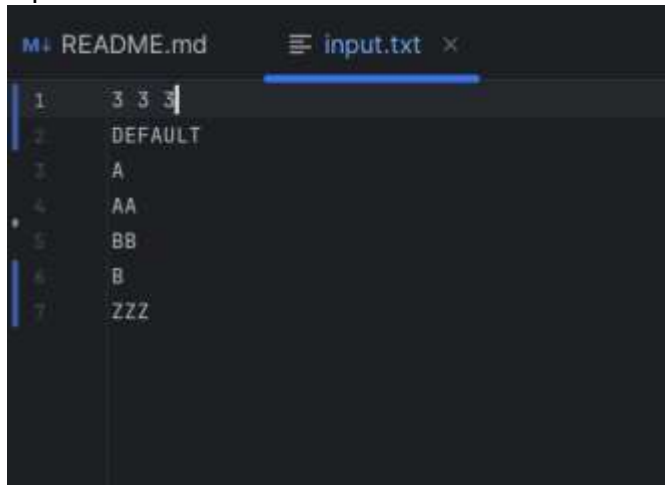
```
1 5 5 7
2 DEFAULT
3 A
4 AA
5 B
6 BB
7 C
8 CC
9 D
10 DD
11 EE
12 EE
13 E
14 FF
15 FF
16 F
17 GGG
```

Output:



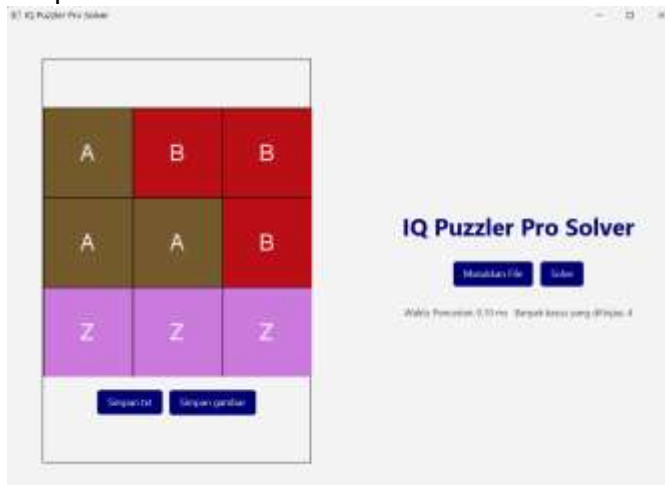
B. Kasus Uji 2

Input:



```
1 3 3 3
2 DEFAULT
3 A
4 AA
5 BB
6 B
7 ZZZ
```

Output:

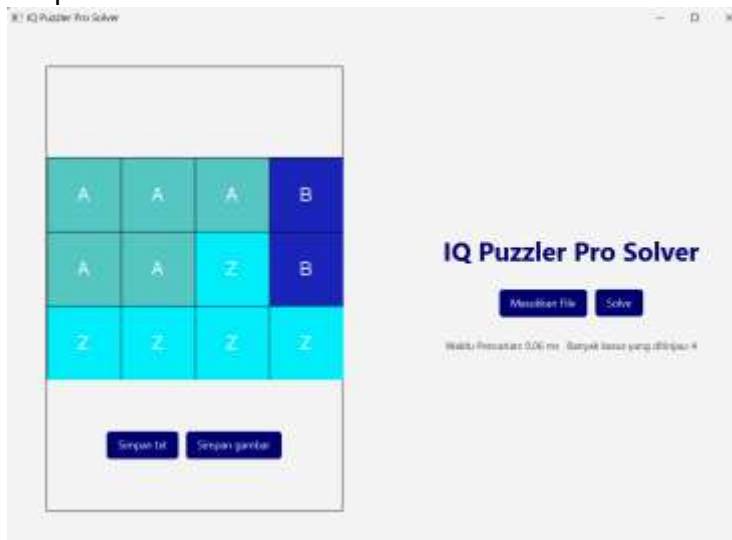


C. Kasus Uji 3

Input:

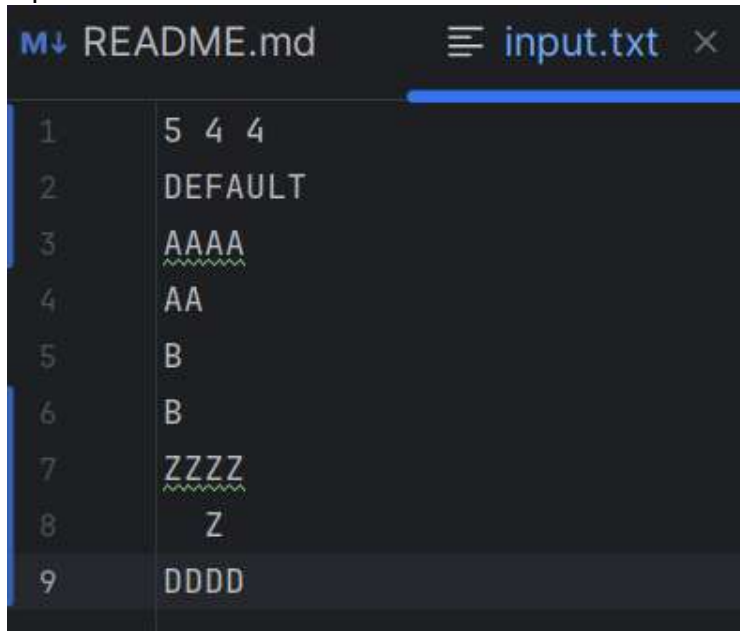
```
1 3 4 3
2 DEFAULT
3 AAA
4 AA
5 B
6 B
7 ZZZZ
8 Z
```

Output:



D. Kasus Uji 4

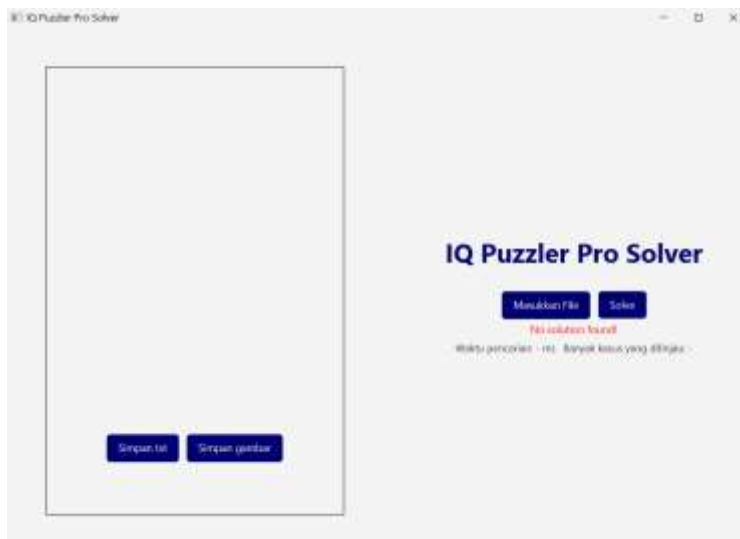
Input:



The screenshot shows a code editor with two tabs: 'README.md' and 'input.txt'. The 'input.txt' tab is active and contains the following text:

```
1 5 4 4
2 DEFAULT
3 AAAA
4 AA
5 B
6 B
7 ZZZZ
8 Z
9 DDDD
```

Output:



Masih tersisa satu blok sehingga solusi tidak ditemukan

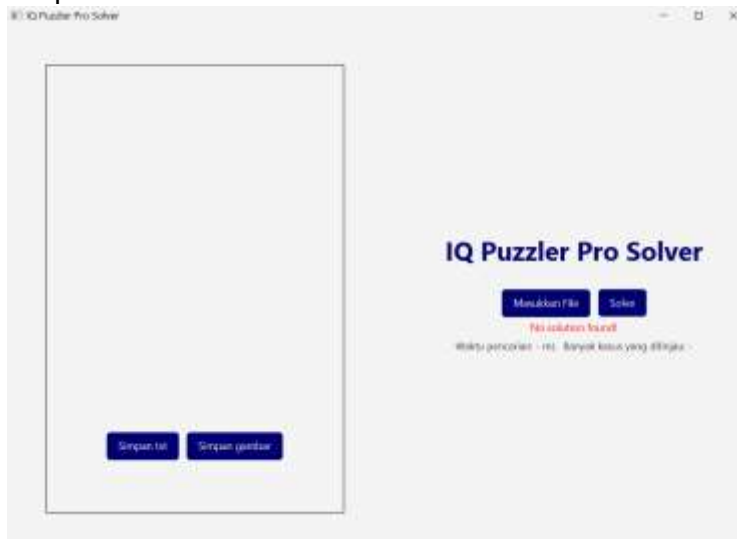
E. Kasus Uji 5

Input:



```
1 5 5 a
2 DEFAULT
3 A
4 AA
5 B
6 BB
7 C
8 CC
9 D
10 DD
11 EE
12 EE
13 E
14 FF
15 FF
16 F
17 GGG
18 HHHH
```

Output:



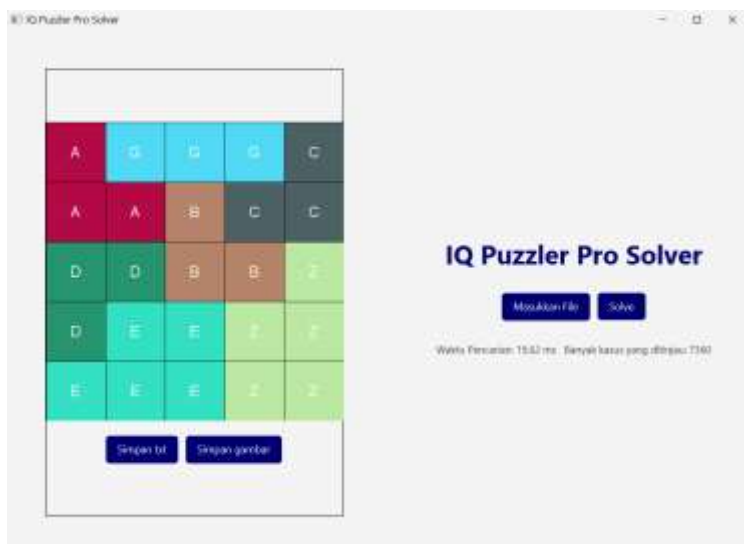
Masih tersisa satu blok sehingga solusi tidak ditemukan.

F. Kasus Uji 6

Input:

```
1 5 5 7
2 DEFAULT
3 A
4 AA
5 B
6 BB
7 CC
8 C
9 D
10 DD
11 E
12 EE
13 EE
14 ZZ
15 ZZ
16 Z
17 GGG
```

Output:

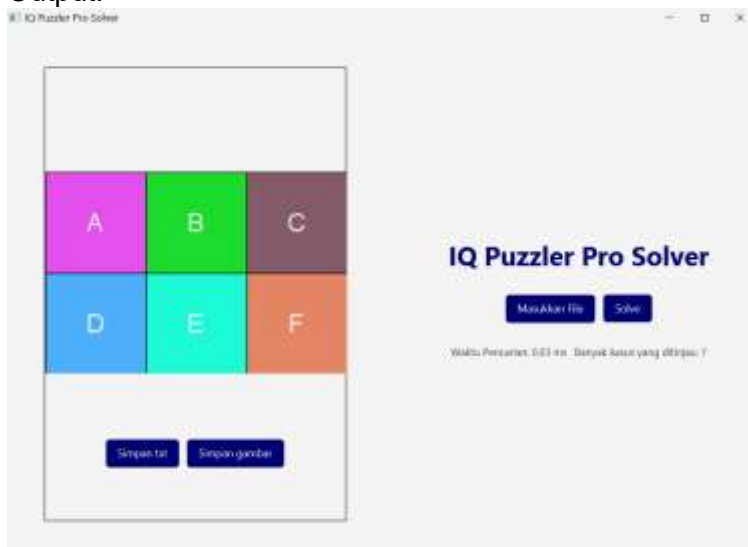


G. Kasus Uji 7

Input:

```
README.md  input.txt  IO.java
1  2 3 6
2  DEFAULT
3  A
4  B
5  C
6  D
7  E
8  F
```

Output:



Bab IV

Lampiran

Repositori program

https://github.com/guntarahmbl/Tucil1_13523114

Pernyataan

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