Tugas Kecil 1 Strategi Algoritma IF2211

Penyelesaian IQ Puzzler Pro dengan Algoritma Brute Force

Disusun oleh:

13523003 – Dave Daniell Yanni

Daftar Isi

Bab 1. Latar Belakang

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 2. Algoritma

Algoritma solver disimpan pada folder utils.
Menerima board kosong yang berupa char[][], dan casesTried yang awalnya 0. allPieces adalah List<List<Piece>> yaitu List yang berisi List yang dalamnya ada sebuah Piece dan permutasi lainnya, rotasi dan mirror.

```
    Solver solver = new Solver(board, 0);
    Solver result = solver.solveBoard(board, allPieces);
    char[][] solution = result.board;
    long cases = result.casesTried;
```

```
package com.utils;

import java.util.Arrays;
import java.util.List;

public class Solver {
    public char[][] board;
    public long casesTried;

    public Solver(char[][] board, long casesTried) {
        this.board = board;
        this.casesTried = casesTried;
    }

    public Solver solveBoard(char[][] board, List<List<Piece>> allPiecesList) {
        char[][] workingBoard = new char[board.length][board[0].length];
        for (int i = 0; i < board.length; i++) {
            workingBoard[i] = Arrays.copyOf(board[i], board[i].length);
        }
        boolean[] usedPieces = new boolean[allPiecesList.size()];
        long[] caseCounter = new long[1];
        boolean solved = solve(workingBoard, allPiecesList, usedPieces, 0, caseCounter);
        return new Solver(solved ? workingBoard : board, caseCounter[0]);
    }
```

```java
    private boolean solve(char[][] board, List<List<Piece>> allPieces, boolean[] usedPieces, int
piecesPlaced,
            long[] caseCounter) {
        if (piecesPlaced == allPieces.size()) {
            return true;
        }
        int boardLength = board.length;
        int boardWidth = board[0].length;
        for (int pieceIndex = 0; pieceIndex < allPieces.size(); pieceIndex++) {
            if (usedPieces[pieceIndex])
                continue;
            for (Piece piece : allPieces.get(pieceIndex)) {
                char[][] pieceMatrix = piece.getPiece();
                char pieceChar = ' ';
                for (char[] row : pieceMatrix) {
                    for (char c : row) {
                        if (c != ' ') {
                            pieceChar = c;
                            break;
                        }
                    }
                    if (pieceChar != ' ')
                        break;
                }
                for (int row = 0; row <= board.length - pieceMatrix.length; row++) {
                    for (int col = 0; col <= board[0].length - pieceMatrix[0].length; col++) {
                        caseCounter[0]++;
                        boolean canPlace = true;
                        if (pieceMatrix.length + row > boardLength || pieceMatrix[0].length + col >
boardWidth) {
                            canPlace = false;
                        }
                        for (int i = 0; i < pieceMatrix.length && canPlace; i++) {
                            for (int j = 0; j < pieceMatrix[0].length && canPlace; j++) {
                                if (pieceMatrix[i][j] != ' ' && board[row + i][col + j] != ' ') {
                                    canPlace = false;
                                }
                            }
```

```
                }
                if (canPlace) {
                    for (int i = 0; i < pieceMatrix.length; i++) {
                        for (int j = 0; j < pieceMatrix[0].length; j++) {
                            if (pieceMatrix[i][j] != ' ') {
                                board[row + i][col + j] = pieceChar;
                            }
                        }
                    }
                    usedPieces[pieceIndex] = true;
                    if (solve(board, allPieces, usedPieces, piecesPlaced + 1, caseCounter)) {
                        return true;
                    }
                    for (int i = 0; i < pieceMatrix.length; i++) {
                        for (int j = 0; j < pieceMatrix[0].length; j++) {
                            if (pieceMatrix[i][j] != ' ') {
                                board[row + i][col + j] = ' ';
                            }
                        }
                    }
                    usedPieces[pieceIndex] = false;
                }
            }
        }
    }
    return false;
  }
}
```

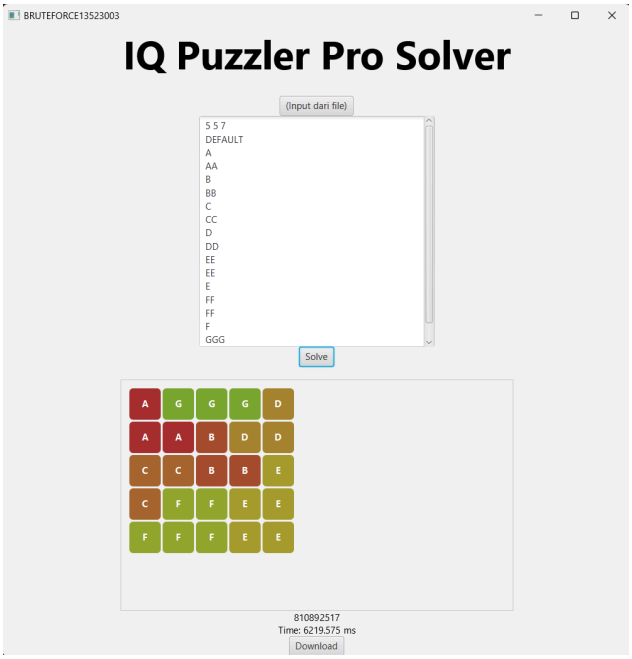
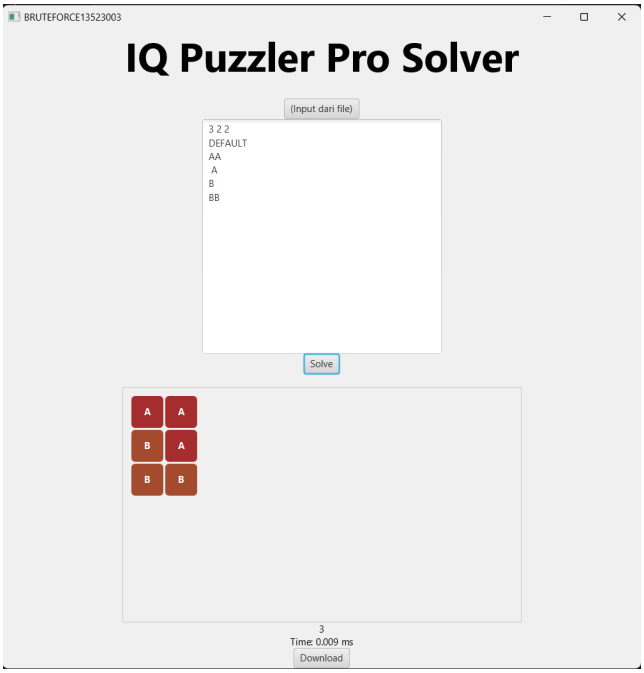
Algoritma melakukan rekursi dan mencoba setiap posisi. Jika tidak dapat diletak maka akan coba posisi berikut, jika dapat diletak maka akan panggil Solve lagi untuk menyelesaikan sisa board yang belum terisi. Solve menerima board yaitu board yang akan diisi, allPieces yaitu List<list<Piece>> yang berisi semua piece, usedPieces sebuah array boolean yang menandai piece mana yang sudah dipakai, piecesPlaced yaitu jumlah piece yang sudah dipakai, dan caseCounter untuk menghitup jumlah cases.

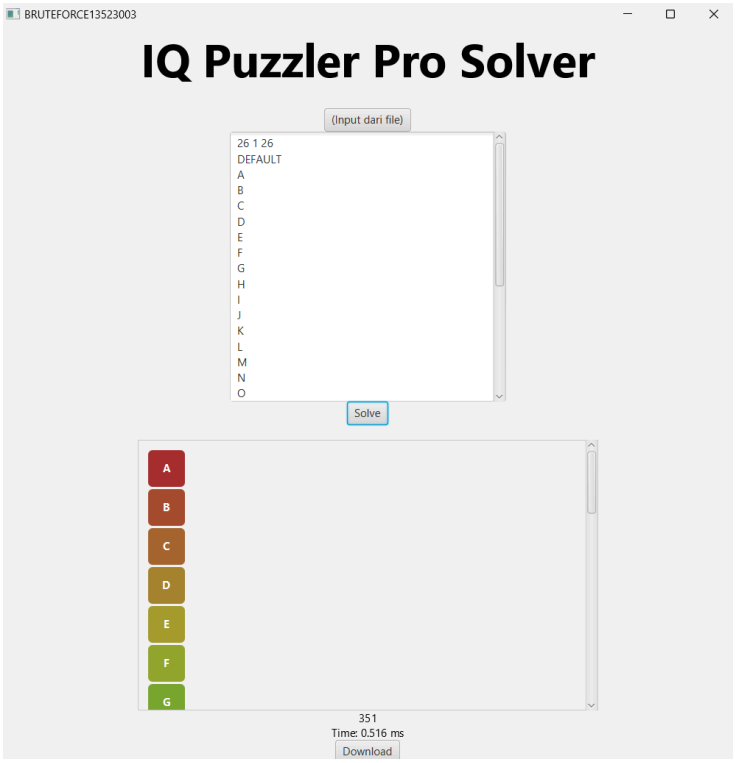
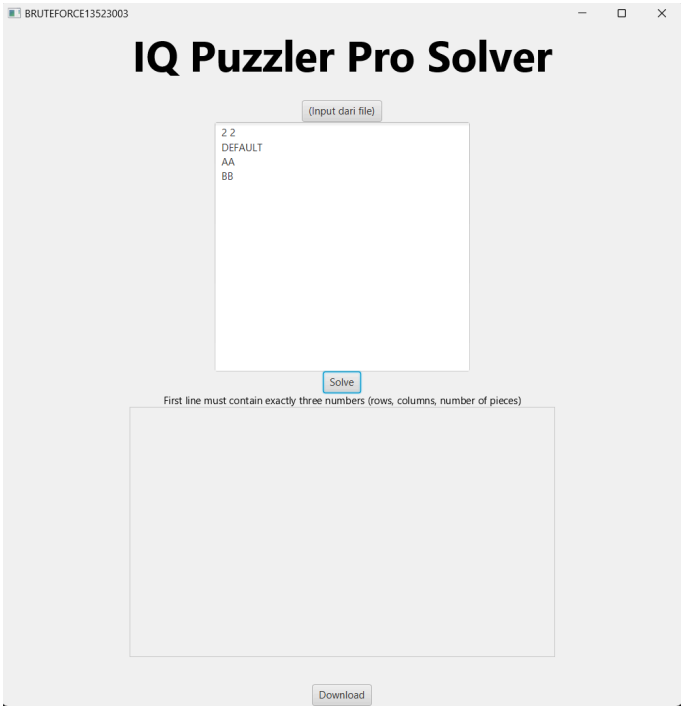
Alur program:

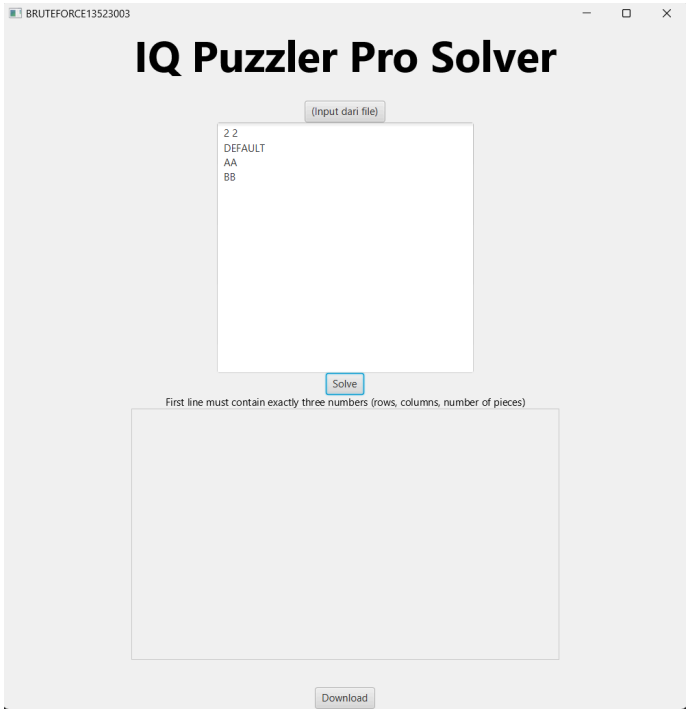
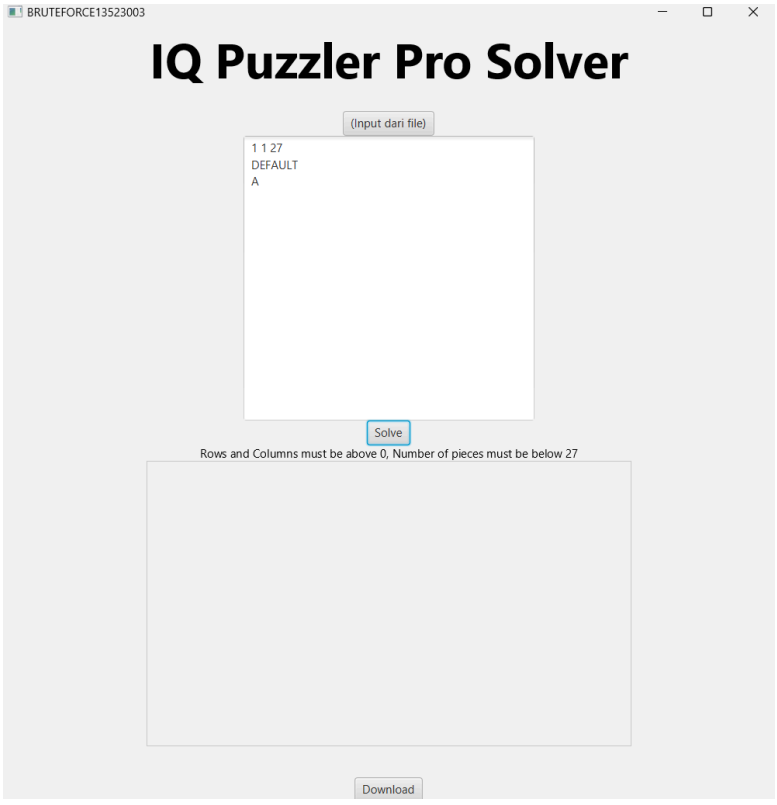
1. Cek apakah seluruh piece sudah placed
2. For loop, skip pieces yang sudah pernah dipakai

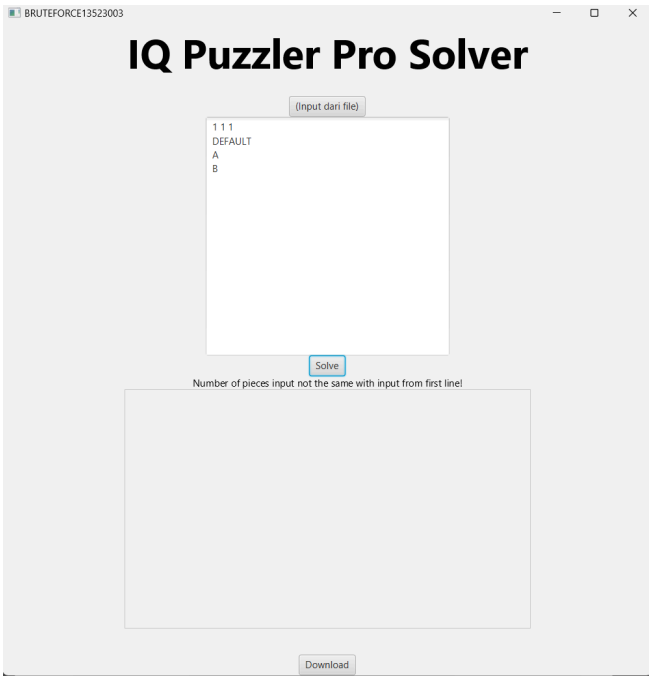
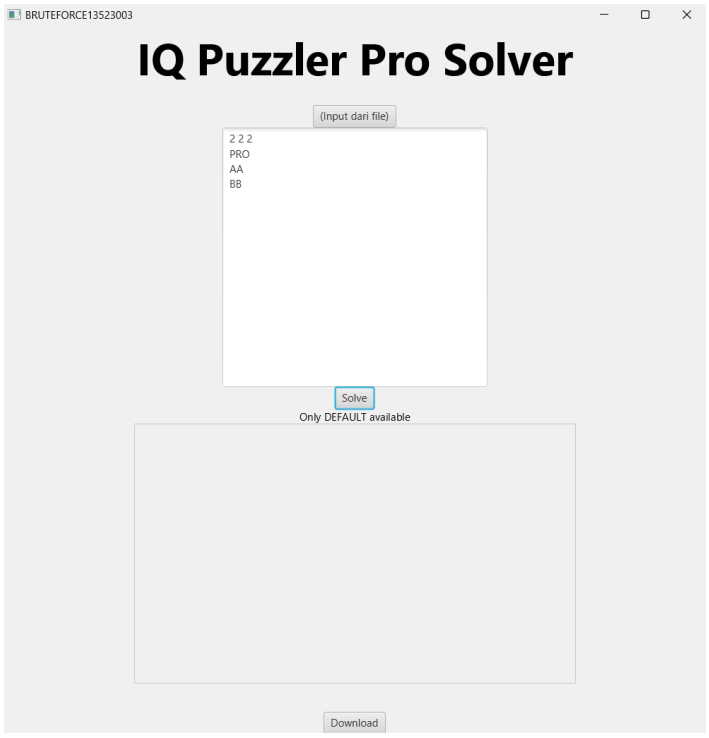
3. For loop List<Piece>
4. Jika dapat diletak maka akan diletak
5. Rekursi Solve dengan sisa board
6. Jika dapat Solve maka return true
7. Jika tidak maka piece yang diletak akan dihapus, board akan kembali ke ‘ ‘

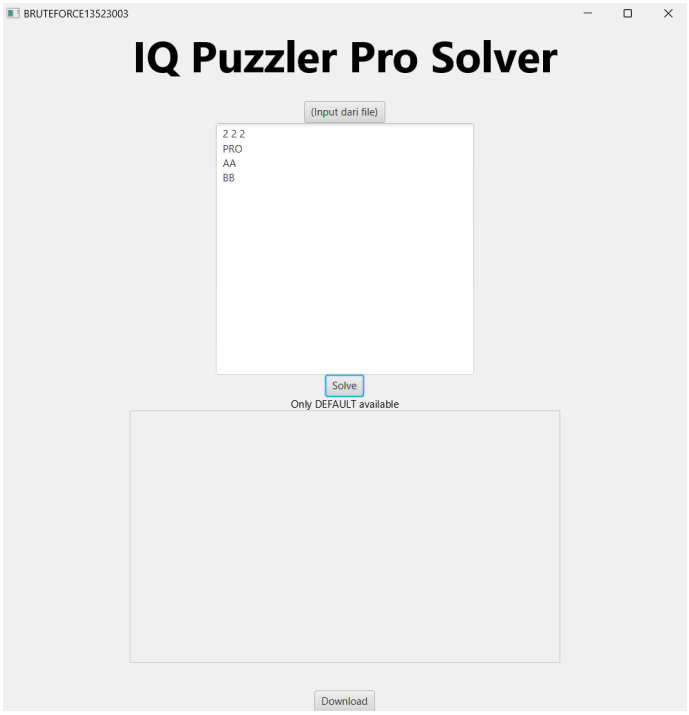
Bab 3. Pengujian

Test	I/O
1	
2	

3	 BRUTEFORCE13523003 IQ Puzzler Pro Solver (Input dari file) 26 1 26 DEFAULT A B C D E F G H I J K L M N O Solve A B C D E F G 351 Time: 0.516 ms Download
4	 BRUTEFORCE13523003 IQ Puzzler Pro Solver (Input dari file) 2 2 DEFAULT AA BB Solve First line must contain exactly three numbers (rows, columns, number of pieces) Download

5	 BRUTEFORCE13523003 IQ Puzzler Pro Solver (Input dari file) 2 2 DEFAULT AA BB Solve First line must contain exactly three numbers (rows, columns, number of pieces) Download
6	 BRUTEFORCE13523003 IQ Puzzler Pro Solver (Input dari file) 1 1 27 DEFAULT A Solve Rows and Columns must be above 0, Number of pieces must be below 27 Download

7	 BRUTEFORCE13523003 IQ Puzzler Pro Solver (Input dari file) 1 1 1 DEFAULT A B Solve Number of pieces input not the same with input from first line! Download
8	 BRUTEFORCE13523003 IQ Puzzler Pro Solver (Input dari file) 2 2 2 PRO AA BB Solve Only DEFAULT available Download

9	 BRUTEFORCE13523003 IQ Puzzler Pro Solver (Input dari file) 2 2 2 PRO AA BB Solve Only DEFAULT available Download

Lampiran

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

Link github: https://github.com/d2v6/Tucil1_13523003