

Tugas Kecil 1 Strategi Algoritma IF2211
Penyelesaian IQ Puzzler Pro dengan Algoritma Brute Force



Disusun oleh:

13523033 - Alvin Christopher Santausa

Daftar Isi

Daftar Isi	2
BAB I LATAR BELAKANG MASALAH.....	3
1.1. IQ Puzzler Pro	3
1.2. Algoritma Brute Force dan Implementasinya pada IQ Puzzler Pro	3
BAB II IMPLEMENTASI	5
2.1. Implementasi Algoritma <i>Brute Force</i> untuk Penyelesaian IQ Puzzler Pro	5
2.3. Penjelasan Source Code Algoritma Brute Force.....	7
2.4. Tampilan Aplikasi	11
BAB III PENGUJIAN	13
3.1. Pengujian 1.....	13
3.2. Pengujian 2.....	13
3.3. Pengujian 3.....	13
3.4. Pengujian 4.....	14
3.5. Pengujian 5.....	14
3.6. Pengujian 6.....	15
3.7. Pengujian 7.....	15
3.8. Pengujian 8.....	15
3.9. Pengujian 9.....	16
LAMPIRAN.....	17

BAB I

LATAR BELAKANG MASALAH

1.1. IQ Puzzler Pro



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 untuk mengisi seluruh papan dengan piece (blok puzzle) yang 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.

1.2. Algoritma Brute Force dan Implementasinya pada IQ Puzzler Pro

1	-2	5	-3	4	-1	6
---	----	---	----	---	----	---

1						
1	-2					
1	-2	5				
1	-2	5	-3			
1	-2	5	-3	4		
1	-2	5	-3	4	-1	
1	-2	5	-3	4	-1	6

Gambar 2 Algoritma Brute Force

(Sumber: <https://algodaily.com/lessons/kadanes-algorithm-explained/brute-force-approach>)

Algoritma *brute force*, sesuai namanya adalah sebuah metode yang lempang (*straightforward*) untuk memecahkan sebuah persoalan dengan mengandalkan komputasi murni dengan mencoba semua kemungkinan tanpa menggunakan teknik canggih untuk meningkatkan efficiency. *Brute force* biasanya didasarkan pada pernyataan pada persoalan (*problem statement*) dan definisi konsep yang dilibatkan. Berdasarkan pernyataan persoalan, contoh penggunaan algoritma *brute force* adalah untuk mencari elemen terbesar/terkecil dan untuk pencarian beruntun (*sequential search*). Sedangkan, berdasarkan definisi konsep yang terlibat, contohnya adalah menghitung a^n , menghitung $n!$, mengalikan dua buah matriks, tes bilangan prima, dan algoritma pengurutan *brute force* (*bubble sort* dan *selection sort*).

Pada IQ Puzzler Pro, algoritma *brute force* dapat digunakan dengan membuat semua variasi dari setiap blok puzzle dengan merotasi dan mencerminkannya sehingga 1 blok puzzle memiliki 8 buah variasi. Kemudian mencoba semua kemungkinan cara untuk menempatkan setiap blok puzzle di papan permainan. Jika tidak bisa diletakkan di tengah percobaan, maka potongan terakhir yang ditempatkan akan dihapus dan kemudian mencoba variasi bentuk lain atau posisi lain di papan permainan. Jika semua potongan puzzle berhasil di pasang tanpa mentissakan tempat kosong pada papan permainan, maka Solusi ditemukan. Jika semua blok puzzle telah ditempatkan tetapi masih ada tempat kosong ataupun jika papan penuh tetapi masih ada blok puzzle yang masih belum ditempatkan (berlebih) maka solusi tidak ditemukan.

BAB II

IMPLEMENTASI

2.1. Implementasi Algoritma *Brute Force* untuk Penyelesaian IQ Puzzler Pro

Sebelum menerapkan algoritma *brute force* untuk menyelesaikan permainan, semua variasi dari sebuah blok puzzle harus didapatkan dengan memutar dan mencerminkan blok puzzle tersebut. Fungsi `getAllOrientations()` akan membuat sebuah list kemudian memasukkan semua kemungkinan variasi blok puzzle ke dalam list tersebut. Kemungkinan-kemungkinan variasi blok puzzle adalah rotasi 0°, rotasi 0° dicerminkan, rotasi 90°, rotasi 90° dicerminkan, rotasi 180°, rotasi 180° dicerminkan, rotasi 270°, dan rotasi 270° dicerminkan. (Pada fungsi `getAllOrientations()` di bawah, diasumsikan fungsi `flip`, `rotate`, dan `add` sudah terdefinisi).

```
function getAllOrientations() → list of Piece  
{Menghasilkan semua orientasi dari sebuah potongan puzzle, termasuk  
rotasi dan flip}
```

Deklarasi:

`orientations` : list of `Piece`

`current` : `Piece`

`i` : `integer`

Algoritma:

`orientations` ← empty list

`current` ← this

for `i` ← 0 to 3 do

add `current` to `orientations`

add `flip(current)` to `orientations`

`current` ← `rotate(current)`

endfor

return `orientations`

Setelah mendapatkan semua kemungkinan variasi dari setiap blok puzzle, maka algoritma *brute force* dapat diterapkan dengan meniterasi setiap kemungkinan posisi x dan y pada papan, serta kemungkinan variasi setiap blok puzzle dengan menggunakan rekursi,

```

function Solve(input index: integer) → boolean
{Menyelesaikan IQ Puzzler Pro dengan algoritma brute force}

```

Deklarasi:

```

piece, p : Piece
x, y : integer

```

Algoritma:

```

if isBoardFull() and index < size of pieces then
    return false
endif

if index = size of pieces and not isBoardFull() then
    return false
endif

if index = size of pieces then
    print "Final Solution:"
    call printBoard()
    isSolved ← true
    return true
endif

piece ← pieces[index]

for x ← 0 to boardRows - 1 do
    for y ← 0 to boardCols - 1 do
        for each p in getAllOrientations(piece) do
            totalCaseChecked ← totalCaseChecked + 1
            if placePiece(p, x, y) then
                if Solve(index + 1) then
                    return true
                endif
            endif
            removePiece(p, x, y) {Backtracking}
        endif
    endfor
endfor

return false
end function

```

Pada Fungsi solve() di atas, dilakukan

1. Mencoba semua kemungkinan posisi x (perulangan/for loop 1) dan posisi y (perulangan/for loop 2), serta semua kemungkinan variasi dari sebuah blok puzzle (perulangan/for loop 3).
2. Jika potongan puzzle bisa dipasang, dilanjutkan ke potongan puzzle berikutnya dengan memanggil fungsi solve() secara rekursif untuk index blok puzzle berikutnya.
3. Jika potongan puzzle tidak bisa dipasang, maka blok puzzle terakhir yang dipasang akan dihapus dan variasi bentuk lain atau posisi lain akan dicoba.
4. Jika papan permainan sudah terpenuhi dan semua blok puzzle telah terpakai, maka solusi ditemukan. Namun, jika semua blok puzzle telah terpakai tetapi masih ada tempat

kosong pada papan permainan (kasus 1) ataupun papan permainan telah terpenuhi tetapi masih ada blok puzzle yang tersisa (kasus 2), maka solusi tidak ditemukan.

5. Jumlah kasus yang diuji juga disimpan dalam variable `totalCaseChecked` untuk melihat berapa kasus yang diuji secara *bruteforce*.

2.3. Penjelasan Source Code Algoritma Brute Force

Source Code yang berhubungan dengan IQ Puzzler Pro dan penyelesaiannya dengan algoritma *brute force* akan dijelaskan pada bagian ini. Repository proyek dapat diakses pada link github yang dilampirkan pada bagian lampiran

1. Class Piece

```
1 import java.util.ArrayList;
2 import java.util.List;
3
4 public class Piece {
5     public char[][] shape;
6     public int width;
7     public int height;
8
9     public Piece(char[][] shape) {
10         this.shape = shape;
11         this.height = shape.length;
12         this.width = shape[0].length;
13     }
14
15     public Piece rotate() {
16         char[][] newShape = new char[width][height];
17         for (int i = 0; i < width; i++) {
18             for (int j = 0; j < height; j++) {
19                 newShape[i][j] = shape[height - j - 1][i];
20             }
21         }
22         return new Piece(newShape);
23     }
24
25     public Piece flip() {
26         char[][] newShape = new char[height][width];
27         for (int i = 0; i < height; i++) {
28             for (int j = 0; j < width; j++) {
29                 newShape[i][j] = shape[i][width - j - 1];
30             }
31         }
32         return new Piece(newShape);
33     }
34
35     public List<Piece> getAllOrientations() {
36         List<Piece> orientations = new ArrayList<>();
37         Piece current = this;
38         for (int i = 0; i < 4; i++) {
39             orientations.add(current);
40             orientations.add(current.flip());
41             current = current.rotate();
42         }
43         return orientations;
44     }
45
46     public void print() {
47         System.out.println("Size: " + height + "x" + width);
48         for (int i = 0; i < height; i++) {
49             System.out.print(" ");
50             for (int j = 0; j < width; j++) {
51                 System.out.print(shape[i][j] + " ");
52             }
53             System.out.println();
54         }
55     }
56 }
57
```

Atribut

- a. `shape (char[][])`
menyimpan bentuk blok puzzle dalam bentuk matriks karakter
- b. `width (int)`
menyimpan lebar (jumlah kolom) dari blok puzzle.
- c. `height (int)`
menyimpan tinggi (jumlah baris) dari blok puzzle.

Method

- `Piece(char[][] shape)`
Menginisialisasi objek `Piece` dengan bentuk, `width`, dan `height` tertentu
- `Piece rotate()`
Merotasi blok puzzle searah jarum jam (90 derajat) dan mengembalikannya
- `Piece flip()`
Mencerminkan blok puzzle dan mengembalikannya
- `List<Piece> getAllOrientations()`
Menghasilkan semua kemungkinan orientasi potongan, termasuk rotasi 90°, 180°, 270°, dan pencerminan dari masing-masing rotasi.

2. Class Board

[illegible]

Atribut

- a. rows (int)
menyimpan jumlah baris papan permainan.

- b. cols (int)
menyimpan jumlah kolom papan permainan
- c. board (char[][])
menyimpan papan permainan sebagai matriks karakter
- d. colorMap
Menyimpan peta warna ANSI untuk setiap karakter yang digunakan dalam papan permainan

Method

- a. Board(int rows, int cols)
Menginisialisasi papan permainan dengan ukuran yang diberikan dan mengisi semua sel dengan '\$'
- b. boolean placePiece(Piece piece, int x, int y)
menempatkan blok puzzle di posisi x,y jika memungkinkan. Jika berhasil, mengubah karakter pada matriks papan permainan.
- c. void removePiece(Piece piece, int x, int y)
Menghapus piece dari papan dengan mengembalikan sel pada matriks ke '\$'
- d. boolean canPlace(Piece piece, int x, int y)
Mengecek apakah sebuah blok puzzle dapat ditempatkan pada x,y. jika bisa akan mengembalikan true
- e. void printBoard()
Mencetak papan ke konsol/terminal sesuai warna karakter yang ada.

3. Class Solver

```

1  import java.util.List;
2
3  public class Solver {
4      public Board board;
5      public List<Piece> pieces;
6      public boolean isSolved;
7      public int totalCaseChecked;
8
9      public Solver(Board board, List<Piece> pieces) {
10         this.board = board;
11         this.pieces = pieces;
12         this.isSolved = false;
13         this.totalCaseChecked = 0;
14     }
15
16     public boolean isBoardFull() {
17         for (int i = 0; i < board.rows; i++) {
18             for (int j = 0; j < board.cols; j++) {
19                 if (board.board[i][j] == '$') {
20                     return false;
21                 }
22             }
23         }
24         return true;
25     }
26
27     public boolean solve(int index) {
28         if (isBoardFull() && index < pieces.size()) {
29             return false;
30         }
31
32         if (index == pieces.size() && !isBoardFull()) {
33             return false;
34         }
35
36         if (index == pieces.size()) {
37             System.out.println("\nFinal Solution:");
38             board.printBoard();
39             isSolved = true;
40             return true;
41         }
42
43         Piece piece = pieces.get(index);
44         for (int x = 0; x < board.rows; x++) {
45             for (int y = 0; y < board.cols; y++) {
46                 for (Piece p : piece.getAllOrientations()) {
47                     totalCaseChecked++;
48                     if (board.placePiece(p, x, y)) {
49                         if (solve(index + 1)) {
50                             return true;
51                         }
52                         board.removePiece(p, x, y);
53                     }
54                 }
55             }
56         }
57         return false;
58     }
59 }

```

Atribut

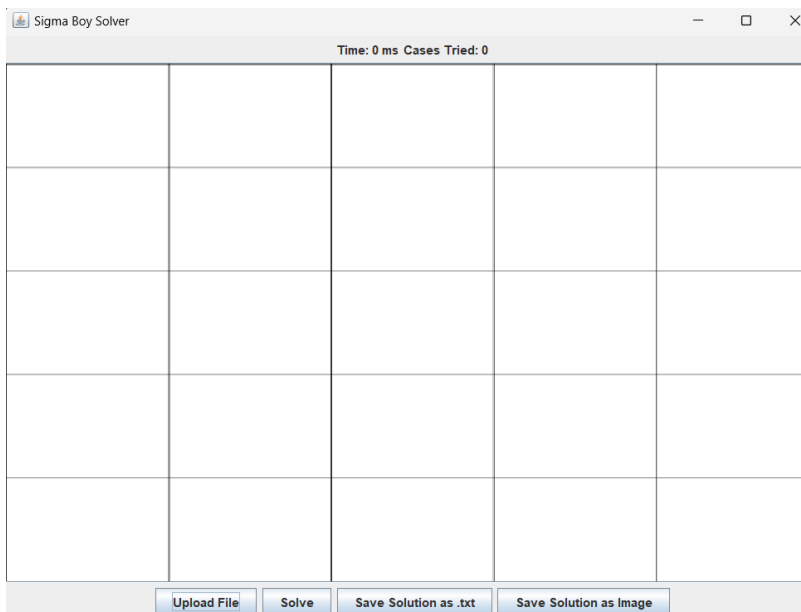
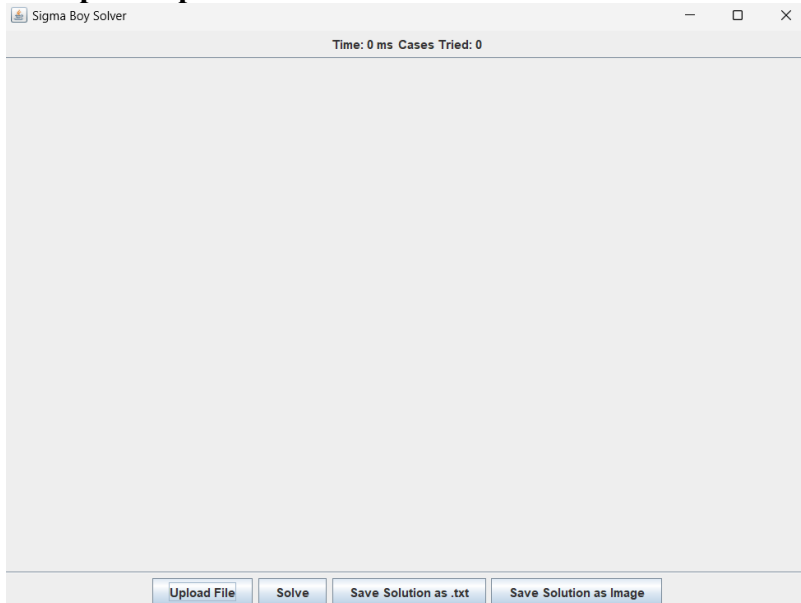
- board (Board)
menyimpan papan permainan yang akan diisi oleh blok puzzle
- pieces (List<Piece>)
Menyimpan daftar potongan yang akan ditempatkan di papan
- isSolved (Boolean)
menandakan apakah solusi telah ditemukan
- totalCaseChecked (int)
Menyimpan jumlah kasus yang diperiksa selama pencarian solusi


Method

- Solver(Board board, List<Piece> pieces)
Menginisialisasi solver dengan papan permainan dan blok puzzle yang diberikan, serta menginisialisasi isSolved ke false dan totalCaseChecked ke 0
- boolean isBoardFull()
Mengecek apakah semua sel di papan sudah terisi (tidak ada ruang kosong lagi/'\$')

- c. `boolean solve(int index)`
Mencari solusi dengan algoritma *brute force* dengan menempatkan setiap blok puzzle di semua kemungkinan posisi dan orientasi.

2.4. Tampilan Aplikasi



 Sigma Boy Solver

—

□

×

Time: 119 ms Cases Tried: 490490

A	B	B	C	C
A	A	B	C	D
E	E	E	D	D
E	E	F	F	F
G	G	G	F	F

Upload File

Solve

Save Solution as .txt

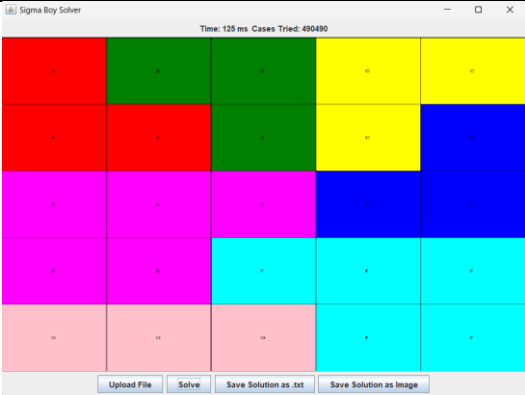
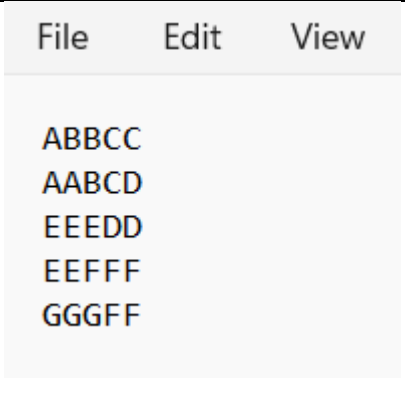
Save Solution as Image

BAB III PENGUJIAN

3.1. Pengujian 1

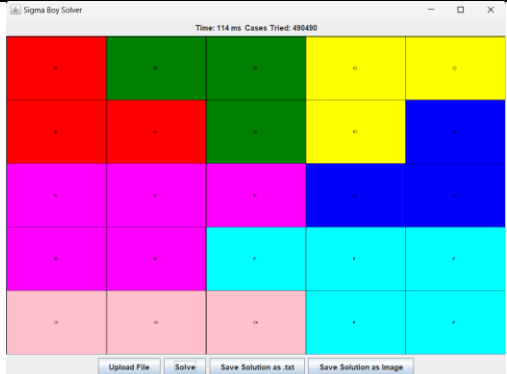
Input File	Output File Image	Output File .txt
3 3 6 DEFAULT A AA A B C D E F		

3.2. Pengujian 2

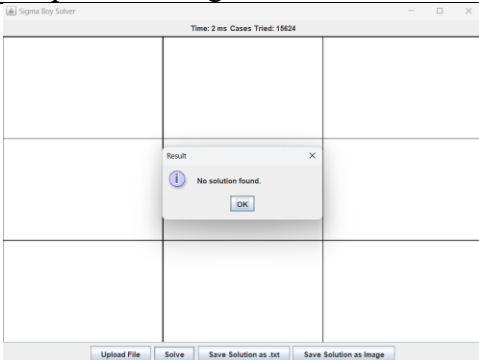
Input File	Output File Image	Output File .txt
5 5 7 DEFAULT A AA B BB C CC D DD EE EE E FF FF F GGG		

3.3. Pengujian 3

Input File	Output File Image	Output File .txt
------------	-------------------	------------------

5 5 7 DEFAULT A AA B BB C CC D DD EE EE E FF FF F GGG		File Edit View ABBCC AABCD EEEDD EEFFF GGGFF
---	--	---

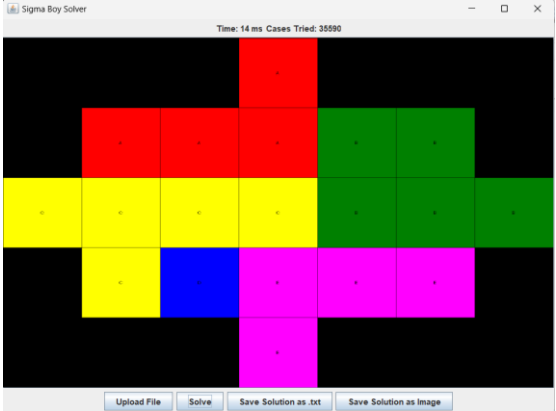
3.4. Pengujian 4

Input File	Output File Image	Output File .txt
3 3 3 DEFAULT AAA CCC BB		- (solution not found)

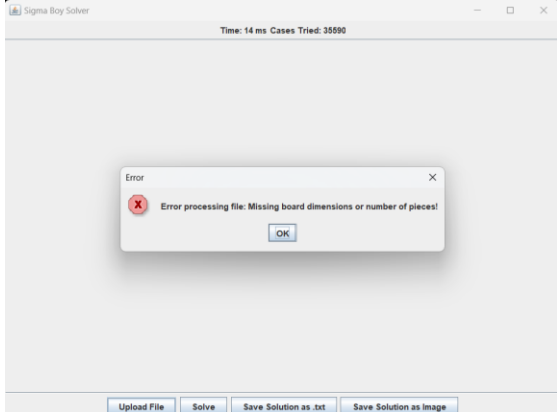
3.5. Pengujian 5

Input File	Output File Image	Output File .txt
4 4 8 DEFAULT A AA B BB C CC D DD E EE F FF		- (invalid number of pieces)

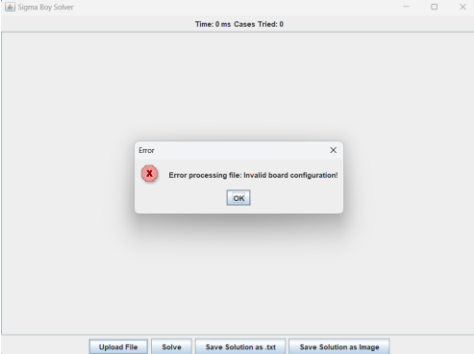
3.6. Pengujian 6

Input File	Output File Image	Output File .txt
5 7 5 CUSTOM ...X... .XXXXX. XXXXXXX X .XXXXX. ...X... A AAA BB BBB CCCC C D EEE E		File Edit View ...A... .AAABB. CCCCBBB .CDEEE. ...E...

3.7. Pengujian 7

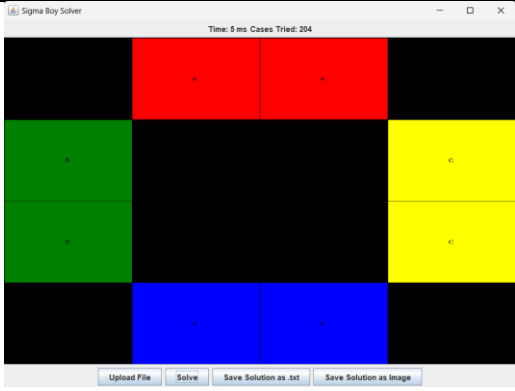
Input File	Output File Image	Output File .txt
HAHA		- (invalid dimensions)

3.8. Pengujian 8

Input File	Output File Image	Output File .txt
6 8 6 CUSTOM ..XX.... .XXXXXX. XXXXXXXXX .XXXXXX. ..XX.... A AA BBB		- (invalid board configuration)

BB CCCC C D EEE E		
----------------------------------	--	--

3.9. Pengujian 9

Input File	Output File Image	Output File .txt
4 4 4 CUSTOM .XX. X..X X..X .XX. AA B B C C DD		<div>File Edit View</div> .AA. B..C B..C .DD.

LAMPIRAN

Referensi

1. <https://www.freecodecamp.org/news/brute-force-algorithms-explained/>
2. [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2017-2018/Algoritma-Brute-Force-\(2016\).pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2017-2018/Algoritma-Brute-Force-(2016).pdf)

Link Repository

https://github.com/Incheon21/Tucil1_13523033

Tabel Evaluasi Program

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