

**Laporan Tugas Kecil 1 IF2211 Strategi Algoritma
Semester II tahun 2024/2025**

Penyelesaian IQ Puzzler Pro dengan Algoritma Brute Force



Disusun oleh:
Refki Alfarizi
13523002

**Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
2025**

Daftar Isi

Daftar Isi	1
Pendahuluan	2
Penyelesaian dan Program	3
Ide Awal Penyelesaian	3
Implementasi dan Penjelasan Program Utama	3
Implementasi Program Pembantu	5
Model Dasar Blok	5
Model Dasar Papan	6
Pengujian Kasus Uji	10
1. Default, 5x5, blok terurut secara abjad, terdapat solusi	10
2. Default, 8x8, blok tidak terurut secara abjad, terdapat solusi	10
3. Custom, 5x7, terdapat solusi	11
4. Default, 8x8, tidak terdapat solusi	12
5. Default, 5x7, terdapat duplikat	13
6. Custom, ukuran asli 5x7, tertulis 9x9	14
7. Default, 0x0, ukuran dan jumlah blok < 0	15
8. Default, 5x5, jumlah blok kurang dari P	15
9. Default, 5x5, jumlah blok lebih dari P	15
10. Default, 10x12, terdapat solusi	16
11. Default, 5x7, terdapat solusi	17
12. Tidak ada format	18
Lampiran	19
Tautan Repositori	19
Tabel Pernyataan	19

Pendahuluan



Gambar 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 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.

Penyelesaian dan Program

Ide Awal Penyelesaian

Penyelesaian IQ Puzzler Pro dapat dilakukan dengan metode *brute force*. Metode ini menyelesaikan masalah dengan cara mengeksplorasi setiap kemungkinan penempatan blok secara sistematis pada papan permainan. Proses tersebut diimplementasikan dengan mencoba menempatkan setiap blok pada berbagai posisi yang tersedia, satu per satu, untuk mencari konfigurasi yang memungkinkan seluruh papan terisi tanpa terjadi tumpang tindih.

Implementasi dan Penjelasan Program Utama

```
public class PuzzleSolver {  
  
    ...  
  
    public PuzzleSolver(Board board, List<Block> blocks) {  
  
        ...  
  
        this.blocksPermutation = new ArrayList<>();  
        for (Block block : blocks) {  
            this.blocksPermutation.add(Block.generateUnique(block));  
        }  
    }  
  
    private boolean solve(int index) {  
        if (impossible){  
            return false;  
        }  
        if (index == blocks.size()) {  
            impossible = !board.isFull();  
            return !impossible;  
        }  
  
        for (int i = 0; i < board.getRows(); i++) {  
            for (int j = 0; j < board.getCols(); j++) {  
                List<Block> uniqueBlocks = blocksPermutation.get(index);  
                for (Block b : uniqueBlocks) {  
                    casesExamined++;  
                    if (board.placeBlock(b, i, j)) {  
                        if (solve(index + 1)) {  
                            solved = true;  
                        }  
                    }  
                }  
            }  
        }  
    }  
}
```

```

        return true;
    } else {
        board.removeBlock(b, i, j);
    }
}
}
}
}
}
solved = false;
return false;
}
}

```

Berikut adalah penjelasan dari implementasi program utama tersebut.

1. Program membuat sebuah larik dua dimensi untuk menyimpan setiap kemungkinan variasi yang valid dari masing-masing blok. Variasi yang dimaksud adalah rotasi dan pencerminan.
2. Program memulai pencarian dengan mengiterasi setiap posisi yang ada pada papan.
3. Pada setiap posisi, program akan mencoba memasang variasi pertama dari blok pertama.
4. Jika blok dapat dipasang pada posisi tersebut, program akan memanggil dirinya sendiri (rekursif) untuk melakukan pemasangan dengan langkah-langkah yang sama namun untuk blok selanjutnya (bukan variasi).
5. Jika blok tidak dapat dipasang, program akan melanjutkan pencarian ke variasi selanjutnya.
6. Jika tidak ada variasi yang dapat dipasang pada papan, program akan mengembalikan *false* yang menandakan penempatan blok sebelumnya salah. Karena blok sebelumnya salah, blok tersebut akan dikembalikan dari papan dan program akan melanjutkan pencarian ke variasi selanjutnya pada blok sebelumnya tersebut (*backtracking*).
7. Ketika seluruh blok sudah terpasang pada papan, maka dapat dilakukan pengecekan untuk menarik kesimpulan. Jika papan terisi penuh, maka program akan mengembalikan *true* yang menandakan solusi sudah ditemukan dan proses *backtracking* akan selesai hingga penempatan balok pertama. Sebaliknya, jika papan tidak terisi penuh, dapat disimpulkan bahwa tidak akan ada solusi valid untuk konfigurasi tersebut. Untuk setiap penempatan, dapat dipastikan bahwa papan tidak akan pernah penuh karena luas total dari semua blok kurang dari luas papan tersebut (*pruning*).

Hal yang menarik dari langkah-langkah tersebut adalah optimisasi-optimisasi yang dapat dilakukan untuk mempercepat proses *brute force* tanpa menggunakan metode heuristik.

Seluruh optimisasi yang diimplementasikan terinspirasi dari buku “Competitive Programming 4 - Book 1”. Berikut adalah beberapa penjelasan dari optimisasi yang dilakukan.

1. Program tidak “melempar” kondisi papan saat rekursi. Teknik tersebut dilakukan karena alasan penggunaan memori dan waktu yang dibutuhkan untuk menyalin kondisi dari papan berulang kali. Selain itu, pendekatan rekursif yang secara konsep melakukan *Depth First Search* memungkinkan teknik tersebut dilakukan karena pada *Depth First Search* proses pencarian menelusuri satu “jalan” yang searah seperti “maju” dan “mundur” dan tidak berpindah-pindah “jalan”. Sehingga proses “maju” dan “mundur” itulah yang memungkinkan proses pasang dan lepas blok pada papan dapat dilakukan.
2. Program kembali ke keadaan sebelumnya (*backtracking*) jika memang tidak dapat melanjutkan pemasangan lagi atau suatu kesimpulan dapat ditarik (*pruning*). Teknik *pruning* digunakan karena program tidak perlu melanjutkan pencarian yang “tidak berguna” sehingga memotong jumlah kasus yang perlu ditinjau dan mempercepat pencarian.

Optimisasi-optimisasi tersebut bukanlah solusi heuristik karena tidak mengubah ruang solusi atau memberikan perkiraan solusi, melainkan hanya meningkatkan efisiensi dari pencarian *brute force*. Dengan kata lain, meskipun teknik-teknik seperti *backtracking* dan *pruning* memotong cabang-cabang pencarian, tetapi cabang yang dipotong hanyalah cabang yang tidak mungkin menghasilkan solusi sehingga seluruh kemungkinan solusi tetap diperiksa secara menyeluruh tanpa penurunan akurasi.

Implementasi Program Pembantu

Berikut adalah implementasi program dari fungsi/metode yang digunakan dalam membantu proses *brute force*.

Model Dasar Blok

```
public class Block {
    private boolean[][] grid;
    private char symbol;

    public Block rotate() {
        int rows = grid.length;
        int cols = grid[0].length;
        boolean[][] rotated = new boolean[cols][rows];
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                rotated[j][rows - 1 - i] = grid[i][j];
            }
        }
    }
}
```

```

    }

    }
    return new Block(symbol, rotated);
}

public Block mirror() {
    int rows = grid.length;
    int cols = grid[0].length;
    boolean[][] mirrored = new boolean[rows][cols];
    for (int i = 0; i < rows; i++) {
        mirrored[i] = grid[i].clone();
        for (int j = 0; j < cols / 2; j++) {
            boolean temp = mirrored[i][j];
            mirrored[i][j] = mirrored[i][cols - 1 - j];
            mirrored[i][cols - 1 - j] = temp;
        }
    }
    return new Block(symbol, mirrored);
}

public static List<Block> generateUnique(Block original) {
    Set<Block> uniqueSet = new HashSet<>();
    Block current = original;
    for (int i = 0; i < 4; i++) {
        uniqueSet.add(current);
        uniqueSet.add(current.mirror());
        current = current.rotate();
    }
    return new ArrayList<>(uniqueSet);
}

...
}

```

Model Dasar Papan

```

public class Board {
    private final int rows;
    private final int cols;
}

```

```

private final char[][] grid;

// Constants for empty cells and boundaries.
public static final char EMPTY = '#';
public static final char BLOCK_BOUNDARY = '.';

public boolean canPlace(Block block, int row, int col) {
    boolean[][] blockGrid = block.getGrid();
    int blockRows = blockGrid.length;
    int blockCols = blockGrid[0].length;

    for (int i = 0; i < blockRows; i++) {
        for (int j = 0; j < blockCols; j++) {
            if (blockGrid[i][j]) {
                int boardRow = row + i;
                int boardCol = col + j;
                // Check boundaries.
                if (boardRow < 0 || boardRow >= rows || boardCol < 0 ||
boardCol >= cols) {
                    return false;
                }
                // Check that the target cell is empty.
                if (grid[boardRow][boardCol] != EMPTY) {
                    return false;
                }
            }
        }
    }
    return true;
}

public boolean placeBlock(Block block, int row, int col) {
    if (!canPlace(block, row, col)) {
        return false;
    }
    boolean[][] blockGrid = block.getGrid();
    int blockRows = blockGrid.length;
    int blockCols = blockGrid[0].length;
    char symbol = block.getSymbol();

    for (int i = 0; i < blockRows; i++) {

```



```

        for (int j = 0; j < blockCols; j++) {
            if (blockGrid[i][j]) {
                grid[row + i][col + j] = symbol;
            }
        }
    }
    return true;
}

public boolean removeBlock(Block block, int row, int col) {
    boolean[][] blockGrid = block.getGrid();
    int blockRows = blockGrid.length;
    int blockCols = blockGrid[0].length;
    char symbol = block.getSymbol();

    // Verify that the block is present.
    for (int i = 0; i < blockRows; i++) {
        for (int j = 0; j < blockCols; j++) {
            if (blockGrid[i][j]) {
                if (grid[row + i][col + j] != symbol) {
                    return false;
                }
            }
        }
    }

    // Remove the block by setting the affected cells to EMPTY.
    for (int i = 0; i < blockRows; i++) {
        for (int j = 0; j < blockCols; j++) {
            if (blockGrid[i][j]) {
                grid[row + i][col + j] = EMPTY;
            }
        }
    }
    return true;
}

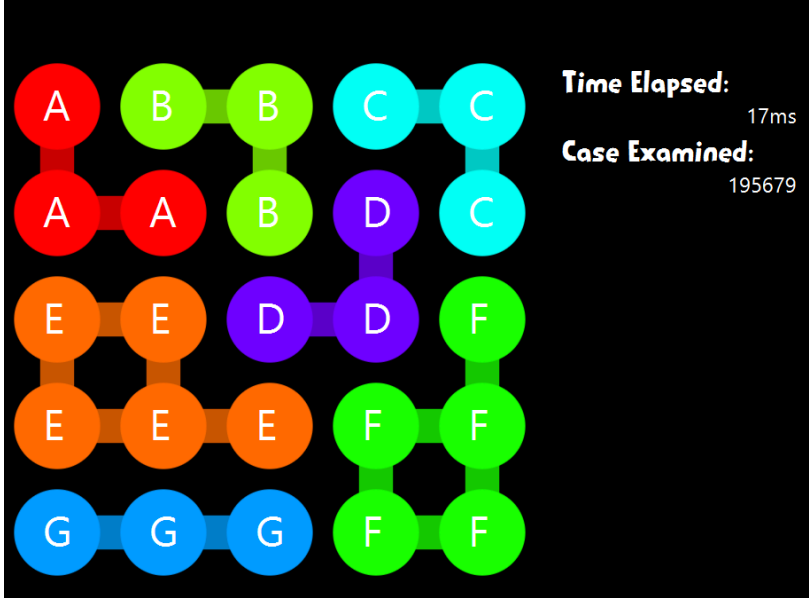
public boolean isFull() {
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            if (grid[i][j] == EMPTY) {
                return false;
            }
        }
    }
    return true;
}

```

```
        }  
    }  
}  
return true;  
}  
  
...  
  
}
```

Pengujian Kasus Uji

1. Default, 5x5, blok terurut secara abjad, terdapat solusi

Input	Output
5 5 7 DEFAULT A AA B BB C CC D DD EE EE E FF FF F GGG	 <p>Time Elapsed: 17ms Case Examined: 195679</p> <pre>≡ output_1.txt ×</pre> <pre>7 ~~~~ ~~~~ 6 ~~~~ ~~~~ 5 ~~~~ ~~~~ 4 ~~~~ ~~~~ 3 ~~~~ ~~~~ 2 Time Elapsed: 17ms 1 Case Examined: 195679</pre>

2. Default, 8x8, blok tidak terurut secara abjad, terdapat solusi

Input	Output
-------	--------

```

8 8 12
DEFAULT
AAA
A A
  A
J
J
JJ
  J
JJ
C
C
C
CCC
DDD
DDD
DDD
EE
  F
FFFF
  F
GG
G
G
HHHH
  H
  H
    II
IIIII
  II
LL
K
BBB
B B
  B

```

Time Elapsed: 520ms
Case Examined: 8297159

```

≡ output_2.txt ×
10  AJJJEECH
9   AJAJJJCH
8   AAACCCCH
7   DDDIIHHH
6   DDDIIIII
5   DDDIIFBB
4   LGFFFFKB
3   LGGGFBBB
2   Time Elapsed: 520ms
1   Case Examined: 8297159

```

3. Custom, 5x7, terdapat solusi

Input	Output
-------	--------

```

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

```

Time Elapsed: 0ms
Case Examined: 22211

≡ output_3.txt ×

```

7    ...A...
6    .AAABB.
5    CCCCBBB
4    .DCEEE.
3    ...E...
2    Time Elapsed: 0ms
1    Case Examined: 22211

```

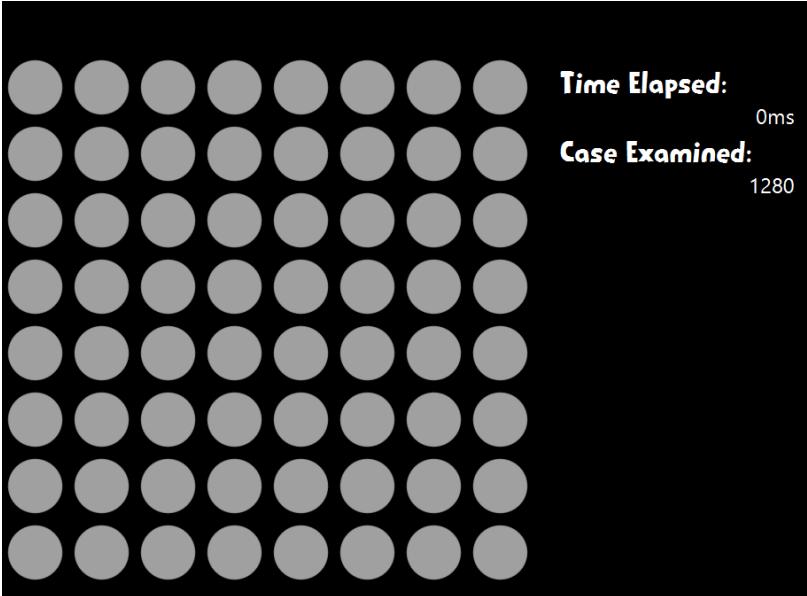
4. Default, 8x8, tidak terdapat solusi

Input	Output
-------	--------

```

8 8 3
DEFAULT
AA
A
    B
BBBB
    B
CCC
    C
    CC

```



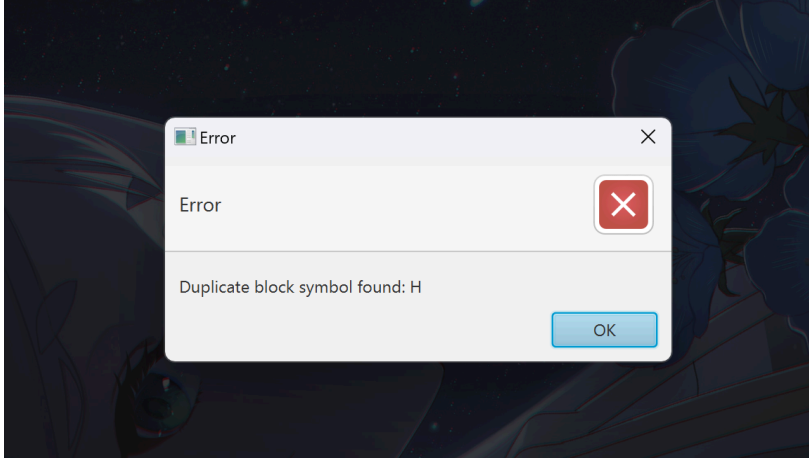
Time Elapsed:
0ms
Case Examined:
1280

≡ output_4.txt ×

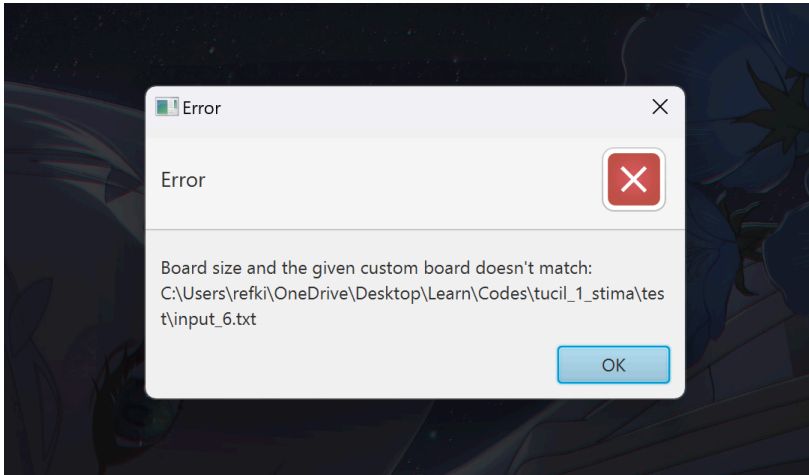
10	#####
9	#####
8	#####
7	#####
6	#####
5	#####
4	#####
3	#####
2	Time Elapsed: 0ms
1	Case Examined: 1280

5. Default, 5x7, terdapat duplikat

Input	Output
-------	--------

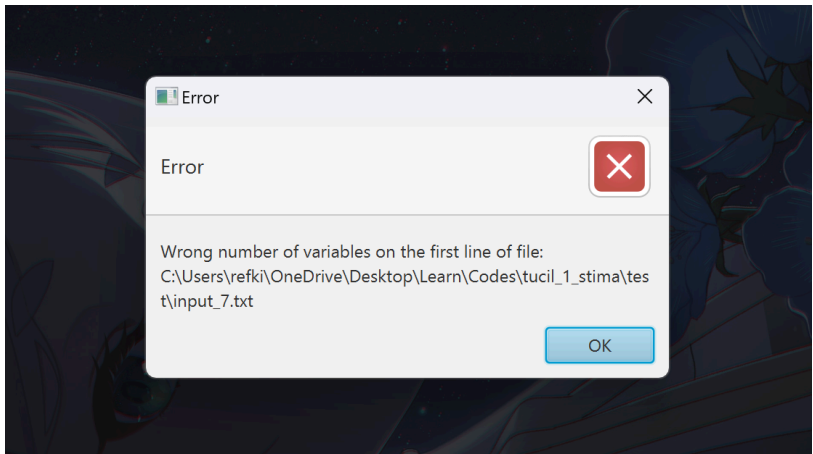
<pre> 5 7 9 DEFAULT HH AAA A A HHH H H B BB B B CC CC DD DD D EE E E FF F G G G G </pre>	
--	--

6. Custom, ukuran asli 5x7, tertulis 9x9

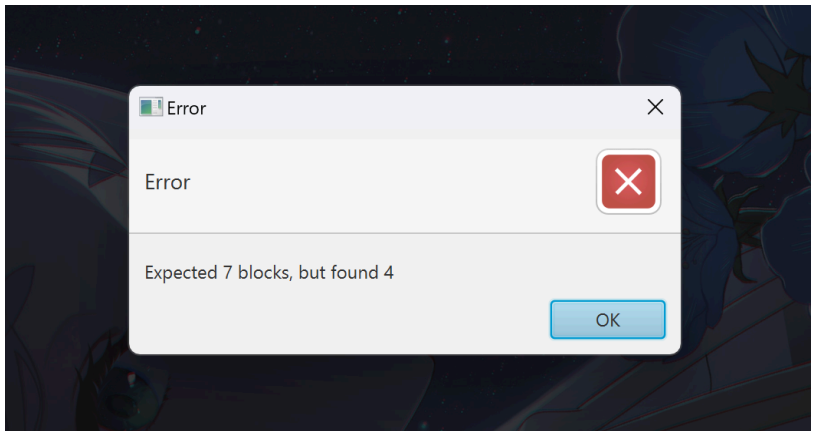
Input	Output
<pre> 9 9 5 CUSTOM ...X... .XXXXX. XXXXXXXX .XXXXX. ...X... A AAA BB BBB CCCC C D </pre>	

EEE E	
----------	--

7. Default, 0x0, ukuran dan jumlah blok < 0

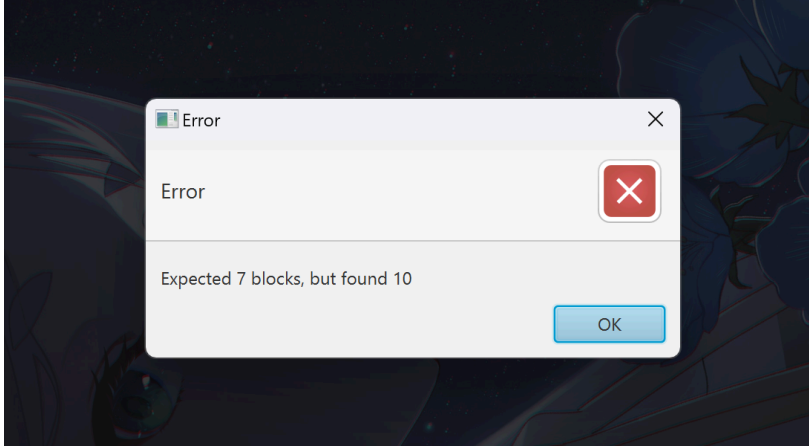
Input	Output
0 0 0 DEFAULT	

8. Default, 5x5, jumlah blok kurang dari P

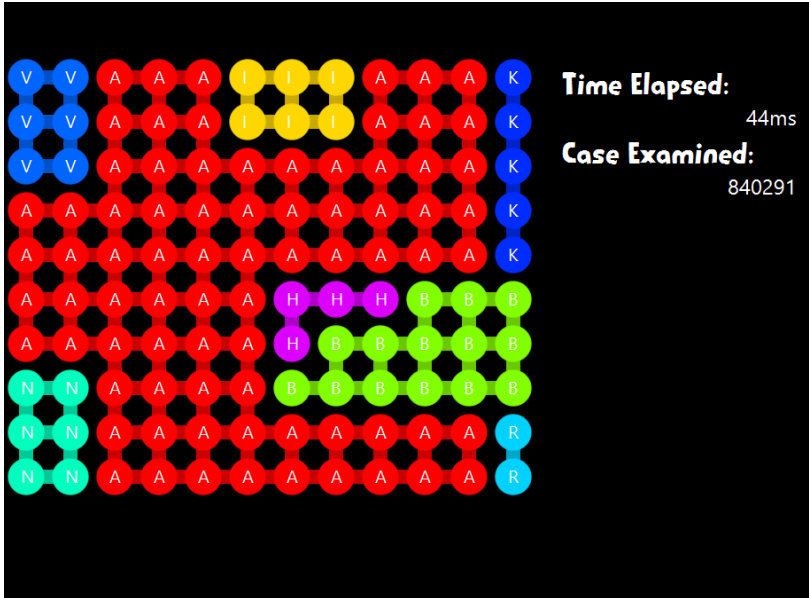
Input	Output
5 5 7 DEFAULT A AA B BB C CC D DD	

9. Default, 5x5, jumlah blok lebih dari P

Input	Output
-------	--------

<pre> 5 5 7 DEFAULT A AA B BB C CC D DD EEEE FFF F F FFFF GG HH II JJJJ </pre>	
---	--

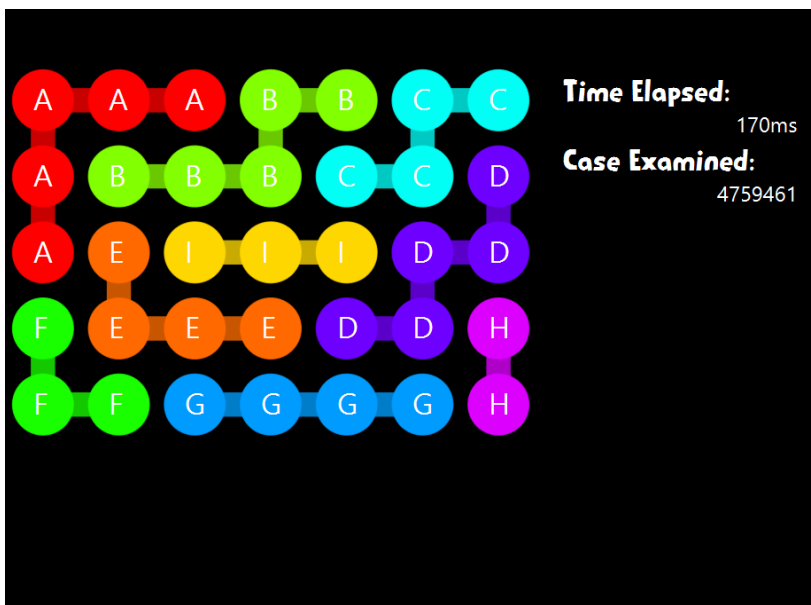
10.Default, 10x12, terdapat solusi

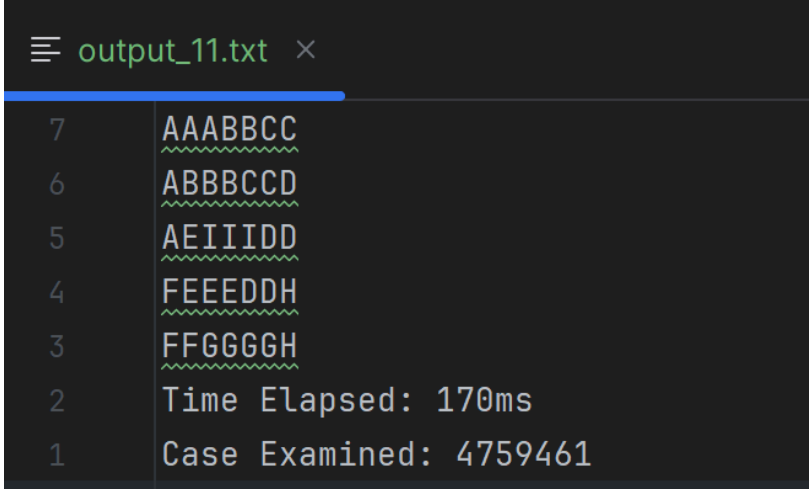
Input	Output
<pre> 10 12 8 DEFAULT AAAAA AAAAA AAAA AAAAA AAAAA AAAAAAAAA AAAAAAAAA AAAAAAAAA AAA AAA AAA AAA HHH H BBB BBBB BBBBB RR KKKKK VVV VVV III </pre>	

III
NNN
NNN

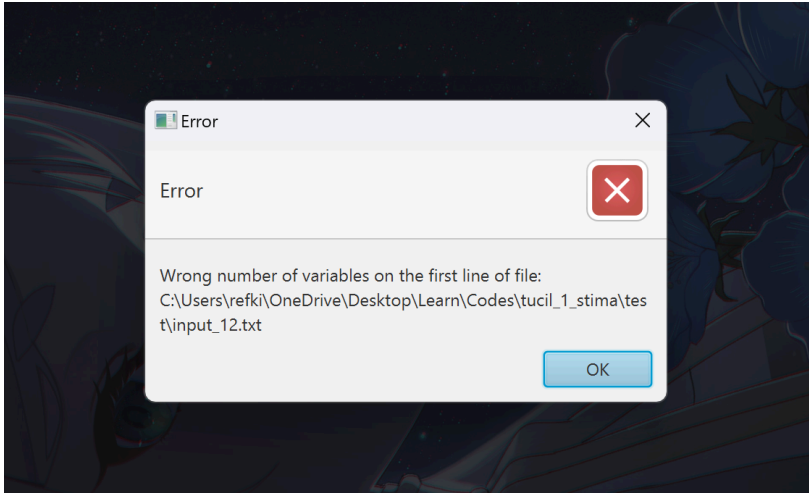
```
output_10.txt x
12  VVAAAIITIAAAK
11  VVAAAIITIAAAK
10  VVAAAAAAAAAAK
9   AAAAAAAAAAAK
8   AAAAAAAAAAAK
7   AAAAAHHHBBB
6   AAAAAHBBBBB
5   NNAAAABBBBBB
4   NNAAAAAAAAAR
3   NNAAAAAAAAAR
2   Time Elapsed: 44ms
1   Case Examined: 840291
```

11.Default, 5x7, terdapat solusi

Input	Output
5 7 9 DEFAULT AAA A A B BB B B CC CC DD DD D EE E E FF F G G G	 <p>Time Elapsed: 170ms Case Examined: 4759461</p>

G HH III	
----------------	--

12. Tidak ada format

Input	Output
Apresiasi setinggi-tingginya buat para asisten dan juga dosen!	

Lampiran

Tautan Repositori

https://github.com/I0stplains/Tucil1_13523002

Tabel Pernyataan

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	✓	