

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

**Penyelesaian IQ Puzzler Pro dengan
Algoritma Brute Force**



Dibuat oleh:

Filbert Engyo - 13523163

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2025

DAFTAR ISI

BAB I	
DESKRIPSI MASALAH.....	3
BAB II	
IMPLEMENTASI PROGRAM.....	5
BAB III	
EKSPERIMEN.....	10
1. Test Case 1.....	10
2. Test Case 2.....	10
3. Test Case 3.....	10
4. Test Case 4.....	10
5. Test Case 5.....	10
6. Test Case 6.....	10
7. Test Case 7.....	10

BAB I

DESKRIPSI MASALAH

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.



Gambar 1. Permainan IQ Puzzler Pro

(Sumber: <https://www.smartgamesusa.com/one-player-games/iq-puzzler-pro>)

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.

Pada laporan ini, akan dibahas implementasi program menggunakan bahasa pemrograman Java untuk menyelesaikan permainan ini dengan algoritma *brute force* yaitu mencoba segala kemungkinan peletakan blok pada *board* tanpa terjadi tumpukan atau tumpang tindih sehingga bisa memenuhi seluruh *board* dengan blok yang ada yang akhirnya menampilkan satu solusi akhir yang disertakan dengan durasi dan jumlah operasi yang dilakukan, atau tidak ada solusi.

BAB II

IMPLEMENTASI PROGRAM

Algoritma *brute force* diimplementasikan dalam bentuk fungsi *solve* yang sekaligus menjadi parameter hasil boolean dari didapatkannya solusi atau tidak, seperti gambar dibawah:



```
1 private static boolean solve(Board board, List<Block> blocks, int index)
2 {
3     if (index == blocks.size())
4     {
5         if (board.isFull())
6         {
7             board.printBoard();
8             return true;
9         }
10        return false;
11    }
12
13    Block block = blocks.get(index);
14    List<Block> orientations = block.getOrientations();
15
16    for (Block orient : orientations)
17    {
18        for (int i = 0; i < board.getN(); i++)
19        {
20            for (int j = 0; j < board.getM(); j++)
21            {
22                if (board.canPlaceBlock(orient, i, j))
23                {
24                    iterations++;
25                    board.placeBlock(orient, i, j, (char) ('A' + index));
26                    if (solve(board, blocks, index + 1)) return true;
27                    board.removeBlock(orient, i, j);
28                }
29            }
30        }
31    }
32
33    return false;
34 }
```

Gambar 2.1. Fungsi *Solve*

Fungsi *solve* mengimplementasikan algoritma *backtracking* yang dilakukan secara rekursif yaitu dengan mencari solusi melalui peletakan blok-blok dalam *board* dengan dirotasi dan/atau dicerminkan agar bisa mencoba kemungkinan-kemungkinan yang ada untuk memenuhi *board*. Fungsi menggunakan 3 parameter yaitu *Board* sebagai tempat yang dibuat dalam *Matrix of Character*, kemudian ada *List of Block* yaitu hasil bacaan dari input yang disimpan dalam sebuah *list*, lalu yang terakhir adalah *integer* index yang akan diinisialisasi dari 0 untuk mengakses blok dari *List of Block*. Sebelumnya, kelas *Reader* hanya terfokus pada bentuk dan bukan huruf.

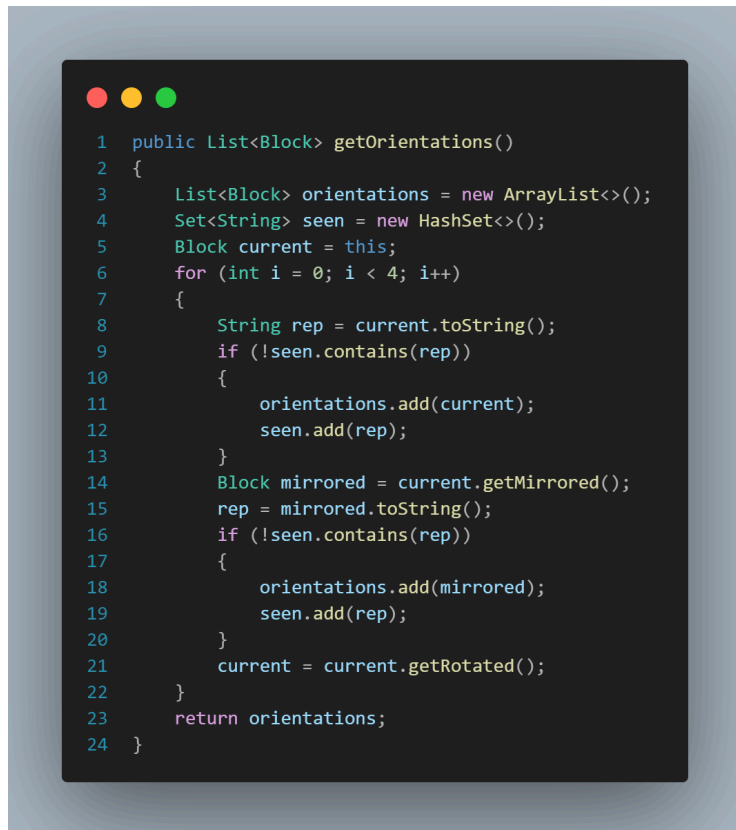
Berdasarkan parameter itu, algoritma dimulai dengan basis yaitu apabila indeks memiliki jumlah yang sama dengan ukuran blocks atau panjang *List of Block*, apabila sama dicek kembali apakah *board* sudah terisi dengan penuh dengan fungsi *isFull()* dari kelas *Board* yang apabila juga terpenuhi baru akan mengembalikan *true*, selain itu akan mengembalikan *false*.



```
1 public boolean isFull()
2 {
3     for (int i = 0; i < N; i++)
4     {
5         for (int j = 0; j < M; j++)
6         {
7             if (board[i][j] == '.') return false;
8         }
9     }
10    return true;
11 }
```

Gambar 2.2. Fungsi *isFull*

Memulai proses rekursif, blok akan diproses dari *List of Block* dengan menggunakan fungsi yang kemudian dibuat *List of Block* yang baru dengan fungsi *getOrientations()* dari kelas *Block* untuk membuat orientasi lain dari blok yang sedang diproses dengan dicerminkan dan/atau dirotasi, sehingga tercipta *List of Block* baru yang berisi segala orientasi dari blok.



```

1  public List<Block> getOrientations()
2  {
3      List<Block> orientations = new ArrayList<>();
4      Set<String> seen = new HashSet<>();
5      Block current = this;
6      for (int i = 0; i < 4; i++)
7      {
8          String rep = current.toString();
9          if (!seen.contains(rep))
10         {
11             orientations.add(current);
12             seen.add(rep);
13         }
14         Block mirrored = current.getMirrored();
15         rep = mirrored.toString();
16         if (!seen.contains(rep))
17         {
18             orientations.add(mirrored);
19             seen.add(rep);
20         }
21         current = current.getRotated();
22     }
23     return orientations;
24 }

```

Gambar 2.3. Fungsi *getOrientations*

Kemudian tahapan *loop* dimulai dengan meloop setiap orientasi dalam *List of Block* orientasi yang digunakan loop bersarang *i* dengan batas *N* sebagai panjang baris dan *j* dengan batas *M* sebagai panjang kolom untuk menguji setiap baris dan kolom pada *board*, lalu proses dimulai dengan penggunaan fungsi *canPlaceBlock()* dari kelas *Board* yang memastikan suatu titik atau sekitarnya tidak memiliki blok lain yang menyebabkan tumpang tindih.

```

1 public boolean canPlaceBlock(Block block, int row, int col)
2 {
3     char[][] shape = block.getShape();
4     int h = block.getHeight();
5     int w = block.getWidth();
6
7     if (row + h > N || col + w > M) return false;
8
9     for (int i = 0; i < h; i++)
10    {
11        for (int j = 0; j < w; j++)
12        {
13            if (shape[i][j] != '.' && board[row + i][col + j] != '.')
14            {
15                return false;
16            }
17        }
18    }
19    return true;
20 }
21

```

Gambar 2.4. Fungsi *canPlaceBlock*

Apabila posisi telah memenuhi syarat maka iterations akan bertambah satu (sebelumnya variabel iterations adalah *integer* yang dideklarasikan pada kelas Main untuk menghitung jumlah iterasi dari percobaan yang dilakukan), kemudian blok dipasang dengan fungsi *placeBlock()* dari kelas Board dengan orientasi yang sesuai, lalu tahap rekursif dilakukan yaitu dengan memanggil kembali fungsi *solve()* dengan parameter index yang diubah menjadi index+1 untuk mencoba blok selanjutnya sekaligus menentukan apakah solusi ditemukan, dan apabila solusi tidak ditemukan maka dilakukan proses *backtrack* yaitu dengan menghapus blok yang telah dipasang pada titik (i, j), kemudian dicoba pada posisi lain dengan fungsi *removeBlock()* dari kelas Board.


```

1 public void placeBlock(Block block, int row, int col, char blockType)
2 {
3     char[][] shape = block.getShape();
4     int h = block.getHeight();
5     int w = block.getWidth();
6
7     for (int i = 0; i < h; i++)
8     {
9         for (int j = 0; j < w; j++)
10        {
11            if (shape[i][j] != '.')
12            {
13                board[row + i][col + j] = blockType;
14            }
15        }
16    }
17 }

```

Gambar 2.5. Fungsi *placeBlock*

```

1 public void removeBlock(Block block, int row, int col)
2 {
3     char[][] shape = block.getShape();
4     int h = block.getHeight();
5     int w = block.getWidth();
6
7     for (int i = 0; i < h; i++) {
8         for (int j = 0; j < w; j++)
9         {
10            if (shape[i][j] != '.')
11            {
12                board[row + i][col + j] = '.';
13            }
14        }
15    }
16 }
17

```

Gambar 2.6. Fungsi *removeBlock*

Apabila semua kemungkinan posisi dan orientasi telah dicoba dan tetap tidak menemukan solusi, maka fungsi akan mengembalikan *false*.

BAB III

EKSPERIMEN

1. Test Case 1

```
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
```

Gambar 3.1.1. Input Test Case 1

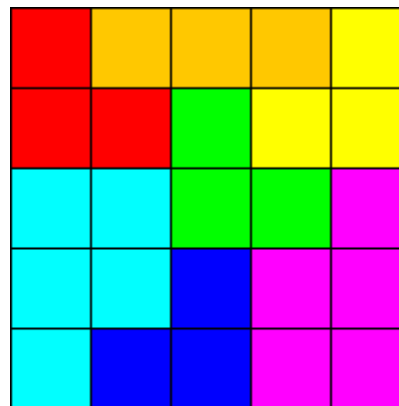
```
$ java -cp bin com.Main
Masukkan nama file test case (.txt): test1.txt
A G G G C
A A B C C
E E B B F
E E D F F
E D D F F

Waktu pencarian: 95.964 ms
Jumlah iterasi: 7166 kali
```

Gambar 3.1.2. Hasil Terminal Test Case 1

```
test > ≡ test1.txt
1  Bentuk Akhir Papan:
2  A G G G C
3  A A B C C
4  E E B B F
5  E E D F F
6  E D D F F
7
8
9  Waktu Pencarian: 114.156 ms
10 Jumlah Iterasi: 7166
11
```

Gambar 3.1.3. Hasil Simpan .txt Test Case 1



Waktu Pencarian: 114.156
Jumlah Iterasi: 7166

Gambar 3.1.4. Hasil Simpan Gambar Test Case 1

2. Test Case 2

```
input > test2.txt
1 5 11 12
2 DEFAULT
3 AAA
4 A A
5 BBBB
6 B
7 C
8 C
9 CCC
10 D
11 DD
12 D
13 E
14 EE
15 EE
16 F
17 FF
18 GG
19 G
20 G
21 G
22 H
23 H
24 HH
25 H
26 I
27 II
28 II
29 J
30 JJ
31 J
32 KK
33 K
34 K
35 LL
36 LLL
37
```

Gambar 3.2.1. Input Test Case 2

```

$ java -cp bin com.Main
Masukkan nama file test case (.txt): test2.txt
A A A B B B B C K K K
A J A D B F F C K G G
J J I D D F E C C C G
J I I D H H E E L L G
I I H H H E E L L L G

Waktu pencarian: 1557.465 ms
Jumlah iterasi: 228850 kali

```

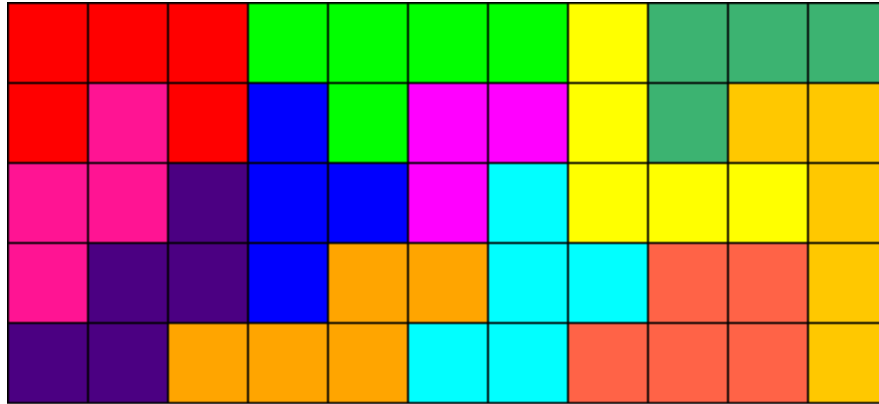
Gambar 3.2.2. Hasil Terminal Test Case 2

```

test > test2.txt
1  Bentuk Akhir Papan:
2  A A A B B B B C K K K
3  A J A D B F F C K G G
4  J J I D D F E C C C G
5  J I I D H H E E L L G
6  I I H H H E E L L L G
7
8
9  Waktu Pencarian: 1832.078 ms
10 Jumlah Iterasi: 228850
11

```

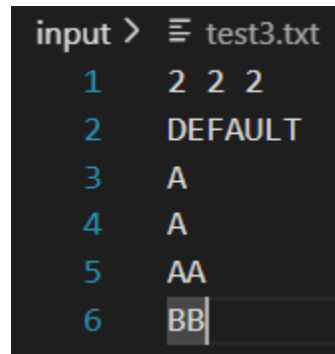
Gambar 3.2.3. Hasil Simpan .txt Test Case 2



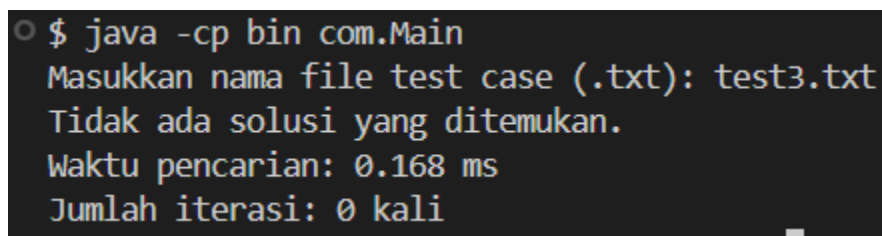
Waktu Pencarian: 1832.078 ms
Jumlah Iterasi: 228850

Gambar 3.2.4. Hasil Simpan Gambar Test Case 2

3. Test Case 3



Gambar 3.3.1. Input Test Case 3



Gambar 3.3.2. Hasil Terminal Test Case 3

```
test > ≡ test3.txt
1   Bentuk Akhir Papan:
2   . .
3   . .
4
5
6   Waktu Pencarian: 0.121 ms
7   Jumlah Iterasi: 0
8
```

Gambar 3.3.3. Hasil Simpan .txt Test Case 3

Tidak ada Waktu Per Jumlah Ite

Gambar 3.3.4. Hasil Simpan Gambar Test Case 3

4. Test Case 4

```
input > test4.txt
1 5 10 7
2 CUSTOM
3 ..XX.XXX..
4 X.X..XXX.X
5 X.XX.XXX.X
6 X.XX.XXXXX
7 XXXXXXXXXXX
8 K
9 K
10 KK
11 KK
12 CCC
13 C
14 D
15 DD
16 D
17 DD
18 F
19 FFFF
20 GGGG
21 G
22 H H
23 HHH
24 PP
25 PP
26 P
```

Gambar 3.4.1. Input Test Case 4

```
● $ java -cp bin com.Main
Masukkan nama file test case (.txt): test4.txt
Tidak ada solusi yang ditemukan.
Waktu pencarian: 0.853 ms
Jumlah iterasi: 4 kali
```

Gambar 3.4.2. Hasil Terminal Test Case 4


```
test > ≡ test4.txt
1  Bentuk Akhir Papan:
2  . . . . .
3  . . . . .
4  . . . . .
5  . . . . .
6  . . . . .
7
8
9  Waktu Pencarian: 1.127 ms
10 Jumlah Iterasi: 4
11 |
```

Gambar 3.4.3. Hasil Simpan .txt Test Case 4

Tidak ada solusi ditemukan!									

Waktu Pencarian: 1.127 ms
Jumlah Iterasi: 4

Gambar 3.4.4. Hasil Simpan Gambar Test Case 4

5. Test Case 5

```
input > ≡ test5.txt
1      5 11 12
2      DEFAULT
3      CC
4      C
5      CC
6      W
7      WW
8      WW
9      L
10     LLLL
11     Z
12     ZZ
13     Z
14     Z
15     C
16     C
17     CCC
18     TTT
19     T
20     M
21     MMMM
22     G
23     GG
24     GG
25     QQ
26     QQ
27     BB
28     B
29     PP
30     PPP
31     N
32     NNN
```

Gambar 3.5.1. Input Test Case 5

```

$ java -cp bin com.Main
Masukkan nama file test case (.txt): test5.txt
C C G W B B Q Q P P P
C G G W W B L Q Q P P
C C G G W W L L L L C
N N N Z Z T M M M M C
N Z Z Z T T T M C C C

Waktu pencarian: 851056.513 ms
Jumlah iterasi: 125470074 kali

```

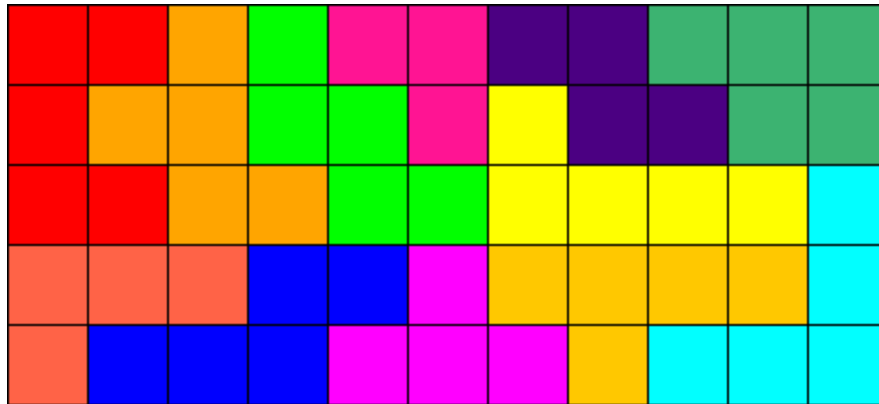
Gambar 3.5.2. Hasil Terminal Test Case 5

```

test > test5.txt
1  Bentuk Akhir Papan:
2  C C G W B B Q Q P P P
3  C G G W W B L Q Q P P
4  C C G G W W L L L L C
5  N N N Z Z T M M M M C
6  N Z Z Z T T T M C C C
7
8
9  Waktu Pencarian: 851056.513 ms
10 Jumlah Iterasi: 125470074
11

```

Gambar 3.5.3. Hasil Simpan .txt Test Case 5



Waktu Pencarian: 889033.161 ms

Jumlah Iterasi: 125470074

Gambar 3.5.3. Hasil Simpan Gambar Test Case 5

6. Test Case 6

Gambar 3.6.1. Input Test Case 6

```

$ java -cp bin com.Main
Masukkan nama file test case (.txt): test6.txt
A B C D E F
G H I J K L
M N O P Q R
S T U V W X
Y Z Z Z Z Z

Waktu pencarian: 9.099 ms
Jumlah iterasi: 26 kali

```

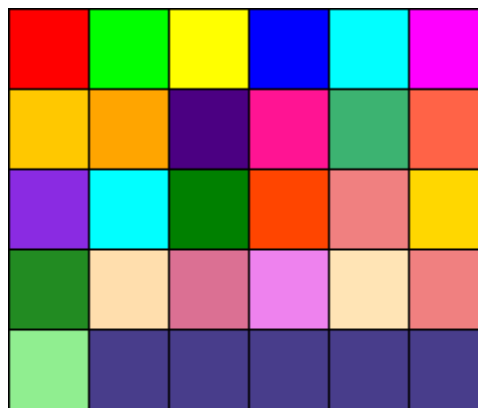
Gambar 3.6.2. Hasil Terminal Test Case 6

```

test > ≡ test6.txt
1  Bentuk Akhir Papan:
2  A B C D E F
3  G H I J K L
4  M N O P Q R
5  S T U V W X
6  Y Z Z Z Z Z
7
8
9  Waktu Pencarian: 15.573 ms
10 Jumlah Iterasi: 26
11

```

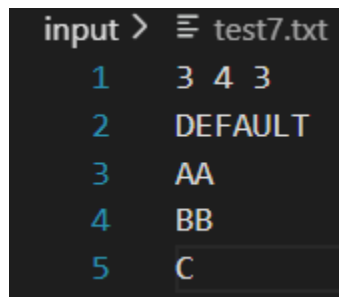
Gambar 3.6.3. Hasil Simpan .txt Test Case 6



Waktu Pencarian: 15.573 ms
Jumlah Iterasi: 26

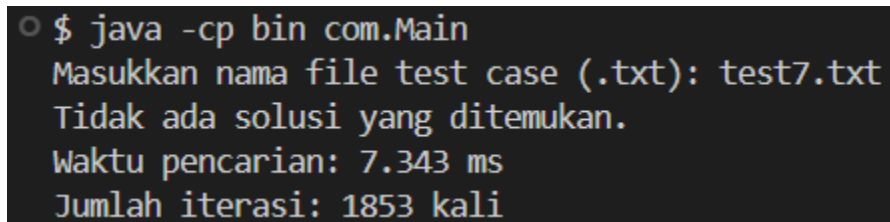
Gambar 3.6.4. Hasil Simpan Gambar Test Case 6

7. Test Case 7



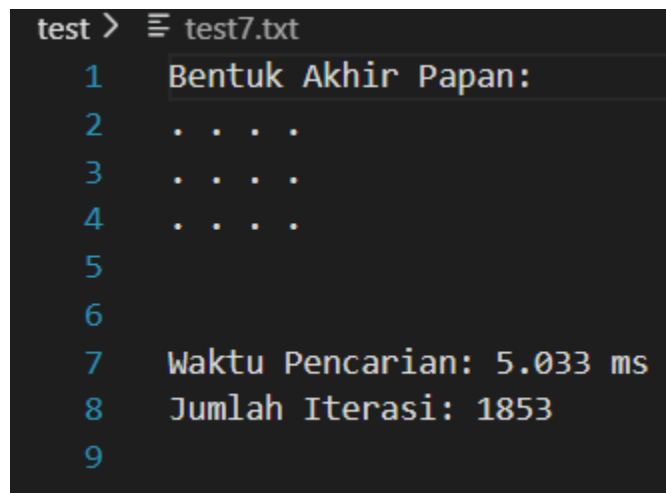
```
input > test7.txt
1 3 4 3
2 DEFAULT
3 AA
4 BB
5 C
```

Gambar 3.7.1. Input Test Case 7



```
$ java -cp bin com.Main
Masukkan nama file test case (.txt): test7.txt
Tidak ada solusi yang ditemukan.
Waktu pencarian: 7.343 ms
Jumlah iterasi: 1853 kali
```

Gambar 3.7.2. Hasil Terminal Test Case 7



```
test > test7.txt
1 Bentuk Akhir Papan:
2 . . . .
3 . . . .
4 . . . .
5
6
7 Waktu Pencarian: 5.033 ms
8 Jumlah Iterasi: 1853
9
```

Gambar 3.7.3. Hasil Simpan .txt Test Case 7

Tidak ada solusi diter			
Waktu Pencarian: 5.0			
Jumlah Iterasi: 1853			

Gambar 3.7.4. Hasil Simpan Gambar Test Case 7

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 <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	✓	

Link Repository GitHub: https://github.com/filbertengyo/Tucil1_13523163