

**LAPORAN TUGAS KECIL 1
IF2211 STRATEGI ALGORITMA**

**PENYELESAIAN IQ PUZZLER PRO DENGAN
ALGORITMA BRUTE FORCE**



Disusun oleh:
Aliya Husna Fayyaza - 13523062

**PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
BANDUNG
2024**

DAFTAR ISI

BAB 1 ALGORITMA.....	3
BAB 2 SOURCE CODE.....	7
BAB 3 TEST CASE.....	12

BAB 1

ALGORITMA

Algoritma *brute force* merupakan cara paling intuitif untuk memecahkan semua permasalahan termasuk menemukan solusi atas berbagai kemungkinan bentuk puzzle dan papan dari permainan IQ Puzzle Pro. Pendekatan ini akan mencari seluruh kemungkinan kombinasi penempatan puzzle yang akan bervariasi karena perbedaan posisi peletakan puzzle dan posisi ditematkannya puzzle pada papan sampai akhirnya seluruh puzzle bisa dimuat dalam papan. Proses *brute force* ini juga melibatkan algoritma rekursif untuk mencari solusi dan *backtracking* dilakukan agar bisa mundur satu tahap jika ternyata solusi tidak ditemukan.

Meskipun menggunakan algoritma rekursif dan *backtracking*, algoritma ini tetap dikategorikan sebagai *brute force* murni karena tetap mencoba semua kemungkinan tanpa adanya optimasi yang mengeliminasi langkah-langkah yang tidak perlu berdasarkan teknik heuristik seperti penentuan prioritas peletakan puzzle atau menghindari kombinasi-kombinasi tertentu yang kecil kemungkinannya untuk menjadi sebuah solusi. *Backtracking* di sini tidak digunakan untuk mengurangi jumlah kemungkinan yang dieksplorasi, melainkan hanya untuk memungkinkan program mundur ke tahap sebelumnya ketika suatu kombinasi tidak valid. Dengan demikian, seluruh kombinasi yang memungkinkan tetap diuji satu per satu, yang merupakan prinsip dasar dari metode *brute force*.

Berikut penjelasan terkait tahapan-tahapan dalam algoritma *brute force* yang digunakan,

1. Input yang berasal dari file akan diproses terlebih dahulu dalam *parser.java* dengan cara mengambil informasi terkait lebar dan panjang papan, banyaknya puzzle yang ingin dimuat dalam papan, serta bentuk dari potongan-potongan puzzle melalui *method parseFile*. Selanjutnya, potongan-potongan puzzle akan disimpan dalam *array of matrix of character* bernama *blocks* menggunakan *method matrixin*. Untuk mempermudah proses selanjutnya, *space* kosong dalam *blocks* akan diisi dengan karakter titik sebagai penanda.

2. Matriks *board* berukuran *lebar* kali *panjang* digunakan untuk merepresentasikan papan permainan sesuai dengan input pengguna. Matriks ini awalnya diisi dengan karakter titik sebagai inisialisasi papan kosong.
3. Untuk mengakomodasi penemuan kombinasi-kombinasi peletakan puzzle, dibuat *method rotate* yang menggunakan *method rotate90* sebagai basisnya. Dalam *method-method* yang akan dijelaskan, *m* adalah banyak baris dan *n* adalah banyak kolom. Cara kerja dari *rotate90* adalah dengan memutar elemen matriks awal dari baris pertama ke kolom terakhir matriks *rotated*, baris kedua matriks awal ke kolom kedua terakhir matriks *rotated*, dan seterusnya yang dimuat dalam kode $rotated[j][m - 1 - i] = matrix[i][j]$. Lalu, *method* ini digunakan dalam *method rotate* yang akan mengulang *method rotate90* sebanyak yang dibutuhkan untuk mengakomodasi rotasi 180 dan 270 derajat dengan *looping* sebanyak derajat yang diinginkan dibagi dengan 90. Dibuat juga *method flipHor* untuk membalikkan matriks *blocks* secara horizontal dengan cara mengambil elemen dari sisi kanan matriks awal dan ditempatkan di sisi kiri matriks *flipped* sesuai urutan (pencerminan dengan sumbu vertikal di tengah), dimuat dalam kode $flipped[i][j] = matrix[i][n - 1 - j]$. Selanjutnya, dibuat juga fungsi *flipVer* untuk membalikkan matriks *blocks* secara vertikal dengan cara membalik sisi atas matriks awal ke bawah matriks *flipped* (pencerminan dengan sumbu horizontal di tengah), dimuat dalam kode $flipped[m - 1 - i][j] = matrix[i][j]$
4. Algoritma utama dari pencarian kombinasi yang mungkin menjadi solusi terletak di dalam *method solve* yang memiliki perulangan bersarang. *Loop* pertama digunakan untuk mencoba semua rotasi (0°, 90°, 180°, dan 270°) menggunakan *method rotate*, *loop* kedua digunakan untuk mencoba membalikkan matriks *blocks* secara horizontal (*j* == 0 untuk posisi matriks *blocks* tidak dibalik dan *j* == 1 untuk posisi dibalik), *loop* ketiga digunakan untuk mencoba membalikkan matriks *blocks* secara vertikal (*k* == 0 untuk posisi matriks *blocks* tidak dibalik dan *k* == 1 untuk posisi dibalik), dan *loop* terakhir digunakan untuk mencoba kemungkinan-kemungkinan bisa ditematkannya matriks *blocks* ke dalam papan sesuai dengan posisi pada setiap iterasinya (disimpan sebagai *finalBlock*). *x* mencoba semua baris pada papan sebagai titik awal dan *y* mencoba semua kolom pada papan sebagai titik awal. Kode *panjang* - *finalBlock.length* dan *lebar* - *finalBlock[0].length* memastikan penempatan matriks *blocks* tidak melebihi luas papan.

Selanjutnya, untuk memeriksa apakah matriks *blocks* bisa ditempatkan pada papan, digunakan *method muat* dan jika bisa ditempatkan, matriks *board* akan diisi menggunakan *method taro*. Setelah matriks *blocks* ditempatkan, dipanggil rekursi untuk matriks *blocks* selanjutnya dengan kode *solve(index + 1)*. Jika ternyata matriks *blocks* selanjutnya tidak bisa dipasang, matriks *blocks* yang sebelumnya dimasukkan akan dihapus dan tempat yang berkesesuaian di matriks *board* akan diganti dengan karakter titik.

5. Sebuah susunan puzzle mungkin tidak memiliki solusi, yaitu saat papan tidak penuh tapi semua puzzle sudah dimasukkan, saat belum semua puzzle dimasukkan tapi papan sudah penuh, maupun saat papan belum penuh, belum semua puzzle dimasukkan, tapi sudah tidak ada tempat yang sesuai untuk puzzle tersebut. Kode menangani tiga kasus dimana puzzle tidak memiliki solusi melalui sistem backtracking. Saat *index == blocks.size()* dan *!isBoardFull()* bernilai *true*, akan dikembalikan sebagai *false* untuk mencoba konfigurasi lain. Untuk kasus kedua, *method muat* akan mengembalikan *false* untuk semua posisi yang dicoba, menyebabkan *return false* dan *backtrack* ke konfigurasi sebelumnya. Sedangkan untuk kasus terakhir, *method solve* akan mencoba semua kemungkinan rotasi dan posisi untuk matriks *blocks* yang sedang diiterasi. Jika tidak ada yang berhasil (semua pemanggilan rekursif mengembalikan *false*), *method* akan *return false* untuk mencoba konfigurasi berbeda dari *blocks* sebelumnya. Jika semua kemungkinan sudah dicoba dan tetap tidak ada solusi, *main program* akan mencetak "Tidak ada solusinya."
6. Untuk memeriksa apakah sebuah matriks *blocks* bisa diletakkan di posisi tertentu pada papan, digunakan *method muat* yang menerima argumen matriks *blocks* serta koordinat titik awalnya. Pengecekan dilakukan dengan memastikan bahwa jika elemen matriks *blocks* yang ingin dimasukkan bukanlah karakter titik, maka posisi pada papan yang berkesesuaian haruslah kosong.
7. Untuk mempermudah penempatan matriks *blocks* ke dalam papan, digunakan *method taro* yang menerima argumen matriks *blocks*, koordinat titik awal, serta huruf yang menjadi identitas matriks *blocks* yang sedang diproses. Penempatan dimulai dengan memeriksa terlebih dahulu bahwa elemen matriks *blocks* yang akan ditempatkan bukanlah elemen berisi karakter titik, lalu secara iteratif mengisi papan dengan huruf matriks *blocks* sampai seluruh elemen matriks *blocks* termuat dalam matriks *board*.

8. Untuk pewarnaan solusi dari *blocks* yang sudah dimasukkan ke dalam *boards*, digunakan *method getColorForChar* sebagai pewarna tampilan solusi yang akan disimpan dalam bentuk gambar serta *method COLORS* digunakan sebagai pewarna tampilan solusi yang muncul di terminal.

Walaupun bisa dipastikan dapat memberikan jawaban, algoritma *brute force* bukanlah pilihan terbaik untuk memecahkan permasalahan IQ Puzzle Pro ini karena memakan waktu yang cukup lama dan kompleksitas dari algoritma ini akan bertumbuh secara eksponensial seiring dengan diperbesarnya ukuran papan dan jumlah potongan puzzle yang ingin dimasukkan. Jumlah waktu yang dibutuhkan juga akan tergantung dengan urutan *looping* pencarian kombinasi peletakan puzzle. Program ini dapat mengeluarkan output yang sama melalui CLI dan GUI, namun ada perbedaan sedikit yaitu waktu yang dibutuhkan GUI untuk menampilkan solusi sedikit lebih lama daripada CLI karena mencakup operasi-operasi yang berhubungan dengan menampilkan solusinya pada GUI.

BAB 2

SOURCE CODE

Bahasa yang digunakan dalam pembuatan program ini adalah bahasa Java. Program disimpan dalam folder src. Di dalam folder src, terdapat dua folder untuk mengakomodasi dua pilihan asal input, yaitu folder CLI untuk input dari terminal dan folder graphic untuk input melalui GUI. Untuk tiap folder, program terdiri dari dua file yaitu *parser.java* dan *solver.java*, namun ada tambahan *GUI.java* di dalam folder graphic untuk mengatur GUI dari program menggunakan *extension* Java Swing.

Algoritma inti terdapat dalam kelas *solver* yang memiliki *method* utama yaitu *solve* yang digunakan untuk mencari solusi dari kemungkinan-kemungkinan kombinasi posisi puzzle dan penempatannya pada papan dengan bantuan fungsi-fungsi rotasi dan pembalikan. Berikut adalah source code dari fungsi-fungsi tersebut:

```
1 public static boolean solve(int index) {
2     if (index == blocks.size()) {
3         if (!isBoardFull()) {
4             return false;
5         }
6         long waktuAkhir = System.currentTimeMillis();
7         System.out.println("Solusi ditemukan: ");
8         cetakBlock(board);
9         System.out.println("Waktu pencarian: " + (waktuAkhir - waktuMulai) + " ms");
10        System.out.println("Banyak kasus yang ditinjau: " + kasusDitinjau);
11
12        Scanner scanner = new Scanner(System.in);
13        System.out.print("Apakah anda ingin menyimpan solusi? (ya/tidak): ");
14        String input = scanner.nextLine().trim().toLowerCase();
15        if (input.equals("ya")) {
16            try (FileWriter writer = new FileWriter("solution.txt")) {
17                for (int i = 0; i < board.length; i++) {
18                    for (int j = 0; j < board[i].length; j++) {
19                        writer.write(board[i][j] + " ");
20                    }
21                    writer.write("\n");
22                }
23                System.out.println("Solusi berhasil disimpan ke solution.txt");
24                image(board, "solution.png");
25            } catch (IOException e) {
26                System.out.println("Gagal menyimpan solusi: " + e.getMessage());
27            }
28        }
29        return true;
30    }
31
32    char[][] currentBlock = blocks.get(index);
33    char blockChar = hurufs.get(index);
34
35    for (int i = 0; i < 4; i++) {
36        char[][] rotatedBlock = rotate(currentBlock, i*90);
37        for (int j = 0; j < 2; j++) {
38            char[][] flippedBlock = (j == 1) ? flipHor(rotatedBlock) : rotatedBlock;
39            for (int k = 0; k < 2; k++) {
40                char[][] finalBlock = (k == 1) ? flipVer(flippedBlock) : flippedBlock;
41                for (int x = 0; x <= panjang - finalBlock.length; x++) {
42                    for (int y = 0; y <= lebar - finalBlock[0].length; y++) {
43                        if (muat(finalBlock, x, y)) {
44                            taro(finalBlock, x, y, blockChar);
45                            kasusDitinjau++;
46                            if (solve(index + 1)) {
47                                return true;
48                            }
49                            taro(finalBlock, x, y, '.');
50                        }
51                    }
52                }
53            }
54        }
55    }
56    return false;
57 }
```

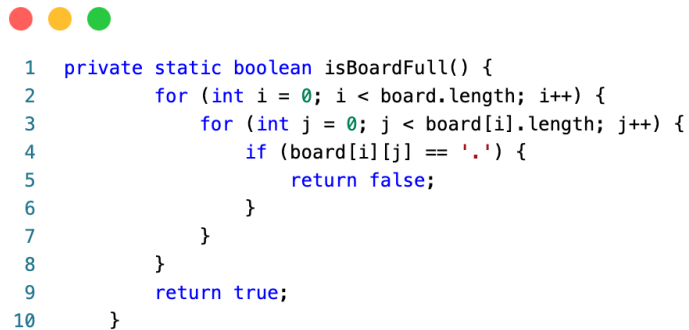
Selanjutnya, berikut adalah fungsi pembantu untuk merotasi dan membalikkan puzzle untuk mencari solusi.

```
1 private static char[][] rotate90(char[][] matrix) {
2     int m = matrix.length;
3     int n = matrix[0].length;
4     char[][] rotated = new char[n][m];
5     for (int i = 0; i < m; i++) {
6         for (int j = 0; j < n; j++) {
7             rotated[j][m - 1 - i] = matrix[i][j];
8         }
9     }
10    return rotated;
11 }
12
13 private static char[][] rotate(char[][] matrix, int degree) {
14     char[][] result = matrix;
15     for (int i = 0; i < degree / 90; i++) {
16         result = rotate90(result);
17     }
18    return result;
19 }
20
21 private static char[][] flipHor(char[][] matrix) {
22     int m = matrix.length;
23     int n = matrix[0].length;
24     char[][] flipped = new char[m][n];
25     for (int i = 0; i < m; i++) {
26         for (int j = 0; j < n; j++) {
27             flipped[i][j] = matrix[i][n - 1 - j];
28         }
29     }
30    return flipped;
31 }
32
33 private static char[][] flipVer(char[][] matrix) {
34     int m = matrix.length;
35     int n = matrix[0].length;
36     char[][] flipped = new char[m][n];
37     for (int i = 0; i < m; i++) {
38         for (int j = 0; j < n; j++) {
39             flipped[m - 1 - i][j] = matrix[i][j];
40         }
41     }
42    return flipped;
43 }
```

Dalam fungsi solve juga digunakan fungsi muat untuk memeriksa apakah puzzle bisa dimasukkan dalam papan dan fungsi taro untuk mengubah elemen matriks papan yang berkesesuaian dengan elemen matriks puzzle.

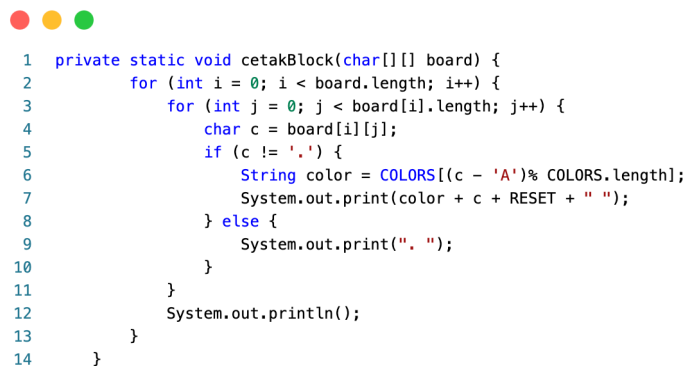
```
1 private static boolean muat(char[][] block, int x, int y){
2     for (int i = 0; i < block.length; i++){
3         for (int j = 0; j < block[i].length; j++){
4             if (block[i][j] != '.' && board[x + i][y + j] != '.') {
5                 return false;
6             }
7         }
8     }
9     return true;
10 }
11
12 private static void taro(char[][] block, int x, int y, char huruf){
13     for (int i = 0; i < block.length; i++){
14         for (int j = 0; j < block[i].length; j++){
15             if (block[i][j] != '.'){
16                 board[x+i][y+j] = huruf;
17             }
18         }
19     }
20 }
```


Selanjutnya, berikut adalah *method* *isBoardFull* yang digunakan untuk memeriksa apakah *board* sudah penuh terisi dengan *blocks*. *Method* ini digunakan untuk membantu proses penentuan ada atau tidaknya solusi.



```
1 private static boolean isBoardFull() {
2     for (int i = 0; i < board.length; i++) {
3         for (int j = 0; j < board[i].length; j++) {
4             if (board[i][j] == '.') {
5                 return false;
6             }
7         }
8     }
9     return true;
10 }
```

Terdapat pula *method* *cetakBlock* untuk menampilkan potongan-potongan puzzle serta papan solusinya jika ada. Untuk mengakomodasi pewarnaan teks tampilan solusi, digunakan *method* *COLORS* dan *RESET* (untuk memastikan bahwa teks selanjutnya tidak terkena efek dari pewarnaan teks sebelumnya) yang berbasis kode ANSI.



```
1 private static void cetakBlock(char[][] board) {
2     for (int i = 0; i < board.length; i++) {
3         for (int j = 0; j < board[i].length; j++) {
4             char c = board[i][j];
5             if (c != '.') {
6                 String color = COLORS[(c - 'A') % COLORS.length];
7                 System.out.print(color + c + RESET + " ");
8             } else {
9                 System.out.print(" ");
10            }
11        }
12        System.out.println();
13    }
14 }
```

Semua *method-method* ini akan dijalankan melalui *method main* sebagai berikut,

```
1 public static void main(String[] args) {
2     try (Scanner scanner = new Scanner(System.in)) {
3         System.out.print("Masukkan nama file: ");
4         String filename = scanner.nextLine();
5
6         if (!filename.toLowerCase().endsWith(".txt")) {
7             System.out.println("File tidak valid. File harus berformat .txt");
8             return;
9         }
10        File file = new File(filename);
11        if (!file.exists()) {
12            System.out.println("File tidak ditemukan di directory Tucil 1 IQ Puzzler Pro");
13            return;
14        }
15
16        parser p = new parser(filename);
17        hurufs = p.getHurufs();
18        solver.lebar = p.getLebar();
19        solver.panjang = p.getPanjang();
20        solver.jmlblok = p.getJmlblok();
21        solver.blocks = p.getBlocks();
22        solver.tipeKasus = p.getTipeKasus();
23
24        System.out.println("Ukuran Papan: " + lebar + " x " + panjang);
25        System.out.println("Jumlah Blok: " + jmlblok);
26        System.out.println("Tipe Kasus: " + tipeKasus);
27        System.out.println("\nBlok Puzzle:");
28
29        for (int i = 0; i < blocks.size(); i++) {
30            System.out.println("Blok ke-" + (i + 1) + ":");
31            cetakBlock(blocks.get(i));
32            System.out.println();
33        }
34
35        board = new char[panjang][lebar];
36        for (int i = 0; i < panjang; i++) {
37            for (int j = 0; j < lebar; j++) {
38                board[i][j] = '.';
39            }
40        }
41
42        waktuMulai = System.currentTimeMillis();
43        if (!solve(0)) {
44            System.out.println("Tidak ada solusinya.");
45        }
46    }
47    catch (IOException e) {
48        System.out.println("Error: " + e.getMessage());
49    }
50 }
```

Program ini dapat menerima input file baik melalui CLI maupun melalui GUI, keduanya diproses dengan mekanisme yang serupa. Untuk penggunaan melalui CLI, dapat dijalankan program `src/CLI/solver.java` dan jika ingin melalui GUI dapat dijalankan `src/graphic/GUI.java` dengan terlebih dahulu melakukan *compile* folder yang ingin dijalankan. Untuk file solusi akan disimpan di luar kedua folder.

BAB 3 TEST CASE

a. Test Case 1

Input:

5 5 7

DEFAULT

A

AA

B

BB

C

CC

D

DD

EE

EE

E

FF

FF

F

GGG

Output:

Membaca file: testcase.txt
Ukuran Papan: 5 x 5
Jumlah Blok: 7
Tipe Kasus: DEFAULT

Blok Puzzle:
Blok ke-1:

A .
A A

Blok ke-2:

B .
B B

Blok ke-3:

C .
C C

Blok ke-4:

D .
D D

Blok ke-5:

E E
E E
E .

Blok ke-6:

F F
F F
F .

Blok ke-7:

G G G

Solusi ditemukan:

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

Waktu pencarian: 1092 ms

Banyak kasus yang ditinjau: 211925

b. Test case 2

11 5 12

DEFAULT

AAA

A

A

BB

BB

B

C

CC

C

DD

DD

E

E

E

EE

FF

FFF

G

GG

HHH

H H

I

III

J

JJ

JJ

KKK

KK

LL

L

L

Membaca file: testcase.txt
Ukuran Papan: 11 x 5
Jumlah Blok: 12
Tipe Kasus: DEFAULT

Blok Puzzle:

Blok ke-1:

A A A
A . .
A . .

Blok ke-2:

. B B
B B .
B . .

Blok ke-3:

C .
C C
C .

Blok ke-4:

D D .
. D D

Blok ke-5:

E .
E .
E .
E E

Blok ke-6:

. . F F
F F F .

Blok ke-7:

G .
G G

Blok ke-8:

H H H
H . H

Blok ke-9:

. I . .
I I I I

Blok ke-11:

K K K
. K K

Blok ke-12:

L L
. L
. L

Solusi ditemukan:

A A A B B C D D H H E
A F B B J C C D D H E
A F B J J C K K H H E
G F F L J J K K I E E
G G F L L L K I I I I

Waktu pencarian: 67117 ms

Banyak kasus yang ditinjau: 9584007

Apakah anda ingin menyimpan solusi? (ya/tidak): tidak

c. Test case 3

Input:

3 3 3

DEFAULT

AAA

BBBBB

C

Output:

Membaca file: testcase.txt

Ukuran Papan: 3 x 3

Jumlah Blok: 3

Tipe Kasus: DEFAULT

Blok Puzzle:

Blok ke-1:

A A A

Blok ke-2:

B B B B B

Blok ke-3:

C

Tidak ada solusinya.

d. Test case 4

Input:

5 5 7

DEFAULT

AAA

B

B

C

CC

D

DD

EEE

E E

E

FF

F

F

GGG

G

Output:

Jumlah Blok: 7
Tipe Kasus: DEFAULT

Blok Puzzle:
Blok ke-1:

A A A

Blok ke-2:

B
B

Blok ke-3:

C .
C C

Blok ke-4:

D .
D D

Blok ke-5:

E E E
E . E
E . .

Blok ke-6:

F F
F .
F .

Blok ke-7:

G G G
. G .

Solusi ditemukan:

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

Waktu pencarian: 250 ms

Banyak kasus yang ditinjau: 43084

Apakah anda ingin menyimpan solusi? (ya/tidak): ya

Solusi berhasil disimpan ke solution.txt

Solusi berhasil disimpan sebagai gambar dalam solution.png

Solution.txt:

A A A B E

D D E B E

F D E E E

F G G G C

F F G C C

Solution.png:

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

e. Test case 5

Input:

5 5 7

DEFAULT

AAA

B

B

C

CC

D

DD

EEE

E E

E

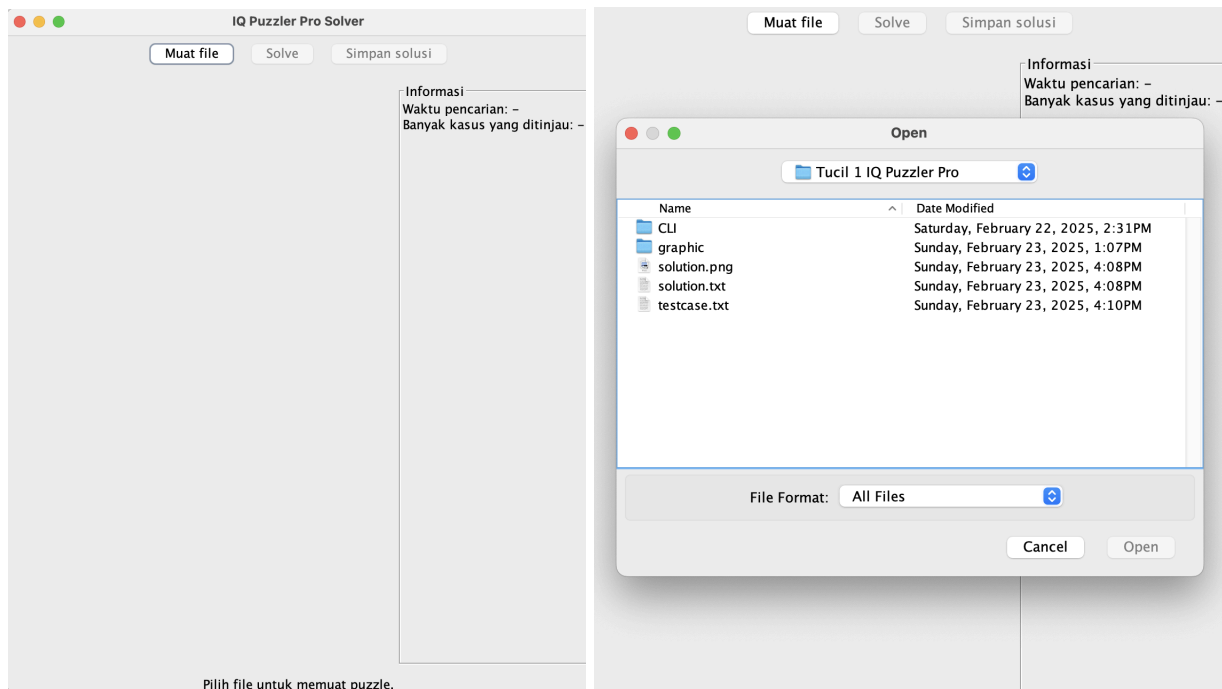
FF

F

F

GGG

G



Muat file Solve Simpan solusi

.
.
.
.
.

Informasi
Waktu pencarian: -
Banyak kasus yang ditinjau: -

File berhasil dimuat. Klik Solve and see the magic!

Muat file Solve Simpan solusi

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

Informasi
Waktu pencarian: 47 ms
Banyak kasus yang ditinjau: 92

Solusi ditemukan!

Muat file Solve Simpan solusi

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

Informasi
Waktu pencarian: 47 ms
Banyak kasus yang ditinjau: 92

Solusi ditemukan!

Simpan Solusi

Apakah anda ingin menyimpan solusi?

No Yes

Muat file Solve Simpan solusi

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

Informasi
Waktu pencarian: 47 ms
Banyak kasus yang ditinjau: 92

Solusi berhasil disimpan ke solution.txt dan solution.png

Berhasil Menyimpan

OK

Solution.txt:

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

Solution.png:

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

f. Test case 6

Input:

5 5 6

DEFAULT

AA

A

B

BB

BB

C

CC

DDD

D D

E

EE

E

F

F

FFF

Output:

Membaca file: testcase.txt
Ukuran Papan: 5 x 5
Jumlah Blok: 6
Tipe Kasus: DEFAULT

Blok Puzzle:
Blok ke-1:

A A
A .

Blok ke-2:

. B .
B B .
. B B

Blok ke-3:

C .
C C

Blok ke-4:

D D D
D . D

Blok ke-5:

. E
E E
. E

Blok ke-6:

. . F
. . F
F F F

Solusi ditemukan:

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

Waktu pencarian: 89 ms

Banyak kasus yang ditinjau: 14389

Apakah anda ingin menyimpan solusi? (ya/tidak): tidak

Test case 7

5 4 5

DEFAULT

AA

A

A

BB

BBB

C

CC

C

DDD

E

EE

E

Membaca file: testcase.txt
Ukuran Papan: 5 x 4
Jumlah Blok: 5
Tipe Kasus: DEFAULT

Blok Puzzle:

Blok ke-1:

A A
A .
A .

Blok ke-2:

B B .
B B B

Blok ke-3:

. C
C C
C .

Blok ke-4:

D D D

Blok ke-5:

E .
E E
. E

Solusi ditemukan:

A A C B B
A C C B B
A C E E B
D D D E E

Waktu pencarian: 36 ms

Banyak kasus yang ditinjau: 5378

Apakah anda ingin menyimpan solusi? (ya/tidak): tidak

LAMPIRAN

Link github: <https://github.com/aliyahusnaf/Tucil-1-Stima.git>

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	