

IF2211 Strategi Algoritma
Penyelesaian IQ Puzzler Pro dengan Algoritma Brute Force

Laporan Tugas Kecil 1

Disusun untuk memenuhi tugas mata kuliah Strategi Algoritma
pada Semester 4 (empat) Tahun Akademik 2024/2025



Disusun oleh:

Brian Ricardo Tamin (13523126)

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

BANDUNG 2025

DAFTAR ISI

BAB I DESKRIPSI MASALAH.....	3
BAB II SPESIFIKASI TUGAS	4
2.1. Input.....	4
2.2. Output:	5
BAB III TEORI SINGKAT	6
BAB IV SOURCE CODE	7
1. Source Code Overview.....	7
4.1.1. Library	7
4.1.2. Struktur.....	7
2. Cara Kerja Algoritma	8
3. Kompleksitas Algoritma.....	10
4.3.1 Best Case	10
4.3.1 Best Case	10
BAB V TEST CASE	11
BAB VI KESIMPULAN & SARAN	17
a. Kesimpulan	17
b. Saran	17
BAB VI DAFTAR PUSTAKA	18
1. Sumber Pustaka	18
2. Lampiran.....	18

BAB I

DESKRIPSI MASALAH

IQ Puzzler Pro adalah permainan papan yang bertujuan untuk mengisi seluruh papan permainan dengan blok-blok puzzle yang telah disediakan. Permainan ini memerlukan keterampilan logika dan strategi, di mana setiap blok memiliki bentuk yang unik dan harus digunakan secara keseluruhan untuk menyelesaikan puzzlenya.

Dalam tugas ini, permasalahan yang dihadapi adalah menemukan satu solusi yang memungkinkan untuk mengisi papan berukuran $M \times N$ dengan menggunakan n blok puzzle. Pada mulanya papan grid kosong sehingga block puzzle harus disesuaikan dengan menggunakan algoritma tertentu untuk mengisi kekosongan hingga penuh.

Kekompleksitasan iq puzzler pro ini mencakup:

1. Setiap blok memiliki bentuk yang berbeda, sehingga memerlukan pendekatan sistematis untuk menguji setiap kemungkinan penempatan.
2. Ada kemungkinan bahwa tidak ada solusi yang tersedia untuk konfigurasi tertentu, sehingga algoritma harus mampu mendeteksi dan melaporkan jika solusi memang tidak ditemukan.

Untuk menyelesaikan masalah ini, digunakan algoritma Brute Force, di mana semua kombinasi penempatan blok akan diuji satu per satu hingga ditemukan solusi yang memenuhi kondisi atau dikonfirmasi bahwa solusi tidak ada. Algoritma ini dipilih karena meskipun kurang efisien, ia menjamin semua kemungkinan diperiksa sehingga tidak ada solusi yang terlewat.

Dengan demikian, tujuan utama dari tugas ini adalah untuk mengembangkan algoritma yang mampu menyelesaikan permainan IQ Puzzler Pro atau memberikan verifikasi jika tidak ada solusi yang mungkin.



Gambar 1 Permainan IQ Puzzler Pro

(Sumber: <https://www.smartgames.eu/uk/one-player-games/iq-puzzler-pro-0>)

BAB II

SPESIFIKASI TUGAS

- Buatlah program sederhana dengan menggunakan bahasa pemrograman Java untuk mengimplementasikan *algoritma Brute Force* untuk mencari salah satu solusi dari persoalan iq puzzler pro.
- Seluruh input akan diterima melalui file dengan ekstensi .txt dimana suatu file.txt mengandung seluruh input requirement.
- Output akan dibagi menjadi 2, pada kasus ini output pada CLI yaitu jumlah percobaan untuk tiap kasus hingga solusi ditemukan, hasil solusi grid, output apakah solusi dapat ditemukan, dan waktu eksekusi. Disisi lain output juga dapat dalam bentuk file.txt atau file.png yaitu hasil solusi pada grid yang disimpan pada suatu file text ataupun gambar.

2.1. Input

- ‘M’ merupakan tinggi dari grid puzzle
- ‘N’ merupakan lebar dari grid puzzle
- ‘P’ merupakan jumlah block puzzle yang akan dimasukkan kedalam grid puzzle
- ‘S’ merupakan mode game yang akan mempengaruhi bentuk grid puzzle
- “GGG” sebagai bentuk representasi dari suatu block dengan identitasnya sebagai block ‘G’. (salah satu contoh input block)

```
N M P
S
puzzle_1_shape
puzzle_2_shape
...
puzzle_P_shape
```

Contoh:

```
5 5 7
DEFAULT
A
AA
B
BB
C
CC
D
DD
EE
EE
E
FF
FF
F
GGG
```

2.2. Output:

- “Attempts” : Jumlah percobaan yang telah dilalui hingga jawaban ditemukan
- “Solution” : Status apakah solusi ditemukan (Found!/not Found!)
- “Time” : Waktu eksekusi algoritma hingga solusi ditemukan
- Grid puzzle berupa matrix of text ataupun image hasil solusi

```
<grid solution>

Waktu pencarian: x ms

Banyak kasus yang ditinjau: n

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

Contoh:

```
AGGGD
AABDD
CCBBF
CEEFF
EEEFF
```

```
Waktu pencarian: 604 ms

Banyak kasus yang ditinjau: 7387

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

BAB III

TEORI SINGKAT

Algoritma brute force adalah metode sederhana untuk menyelesaikan masalah dengan mencoba setiap kemungkinan solusi satu per satu hingga menemukan jawaban yang benar atau paling mendekati. Cara ini cukup mudah dipahami dan diterapkan karena tidak memerlukan pengetahuan khusus atau asumsi tentang masalah tersebut. Namun, kelemahannya adalah metode ini bisa memakan banyak waktu dan sumber daya, terutama jika jumlah kemungkinan solusinya sangat besar atau rumit.

Dalam dunia komputasi, khususnya hacking, algoritma brute force sering digunakan. Hacking sendiri adalah upaya untuk masuk atau mengambil alih sistem komputer yang memiliki perlindungan keamanan. Salah satu caranya adalah dengan memecahkan kode pengaman, seperti password atau kunci enkripsi. Algoritma brute force bekerja dengan mencoba semua kombinasi karakter, mulai dari yang paling sederhana hingga yang paling rumit, sampai akhirnya menemukan kombinasi yang sesuai dengan kode yang sedang dicari.

Kelebihan utama dari algoritma brute force adalah kemampuannya untuk menjamin solusi yang ditemukan benar atau optimal, selama memiliki cukup waktu dan sumber daya. Metode ini juga tidak bergantung pada informasi tambahan atau asumsi tertentu, sehingga cocok untuk menyelesaikan masalah yang belum jelas atau pasti. Meski begitu, kekurangannya adalah algoritma ini sangat tidak efisien karena memeriksa semua kemungkinan tanpa strategi atau batasan tertentu. Efektivitasnya sangat dipengaruhi oleh seberapa besar dan rumit ruang solusi, serta panjang dan tingkat kesulitan kode yang dicoba dipecahkan.

BAB IV

SOURCE CODE

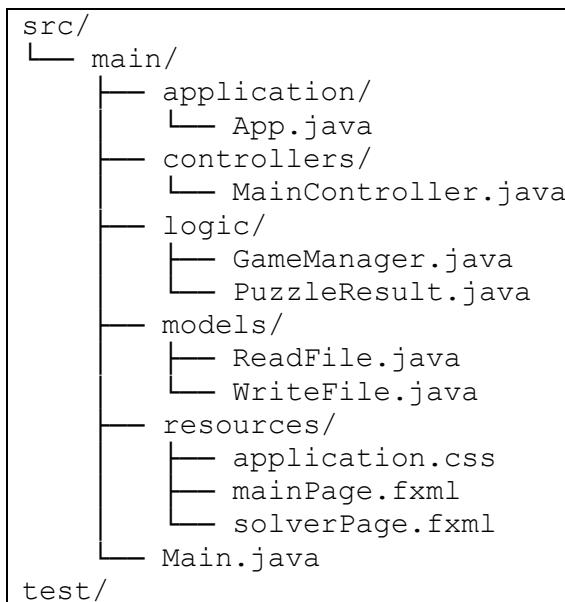
1. Source Code Overview

Tucil ini dikembangkan menggunakan Java (JDK 17+) dan Maven untuk mengelola dependensi serta build proyek. Antarmuka grafis (GUI) dibuat dengan JavaFX untuk visualisasi board, sedangkan FXML memisahkan logika aplikasi dari desain UI.

4.1.1. Library

- SwingFXUtils untuk menyimpan gambar solusi (PNG/JPEG)
- Javax.imageio untuk proses ekspor gambar
- JavaFX.concurrent (Task) untuk menjalankan algoritma secara *concurrent* agar GUI tidak *freeze*
- JavaFX.scene.control untuk komponen UI seperti label dan notifikasi

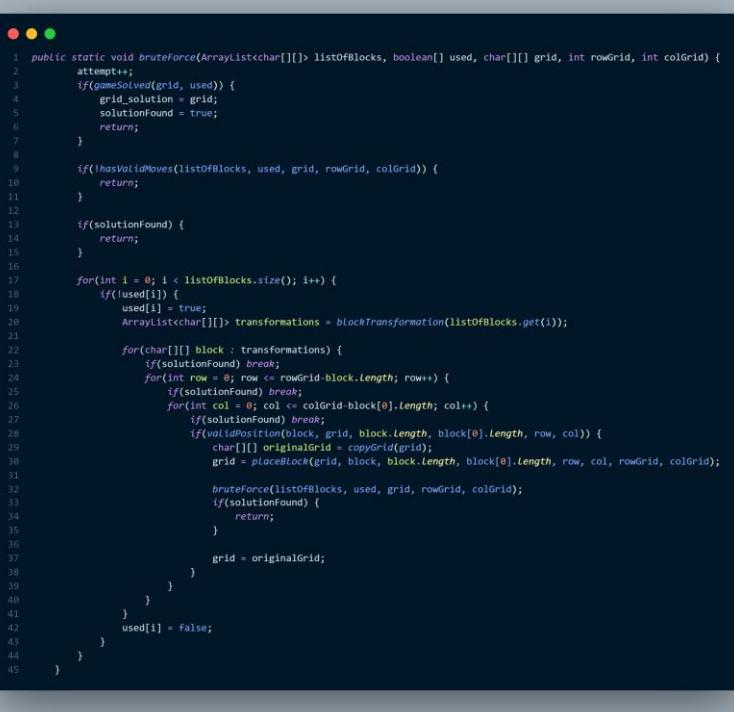
4.1.2. Struktur



- App.java : Menginisialisasi GUI dengan GUI
- MainController.java : berfungsi sebagai penghubung core program dengan GUI
- GameManager.java : core program dimana algoritma dan logika permainan berada
- PuzzleReesult.java : menerima return dari GameManager.java untuk dikirim ke MainController.java
- ReadFile.java : membaca file .txt sebagai input
- WriteFile.java : menulis solusi pada file .txt
- Application.css : styling GUI
- MainPage.fxml solverPage.fxml : html GUI

2. Cara Kerja Algoritma

Algoritma ini menggunakan metode brute force backtracking untuk menyelesaikan teka-teki penyusunan blok. Berikut adalah alur kerja secara bertahap:

source code	cara kerja
 <pre>1 public static PuzzleResult runSolver(String file) { 2 filename = file; 3 4 int[] boardSpecs = ReadFile.BoardSpecs(filename); 5 String mode = ReadFile.configuration(filename); 6 int N = boardSpecs[0]; 7 int M = boardSpecs[1]; 8 9 char[][][] shapes = ReadFile.Shapes(filename, boardSpecs[2]); 10 ArrayList<char[][]> listOfShapes = new ArrayList<>(); 11 for(char[][] block : shapes) { 12 listOfShapes.add(block); 13 } 14 15 char[][] puzzle_board = new char[N][M]; 16 puzzle_board = initialize_board(puzzle_board, N, M); 17 18 if(mode.equals("DEFAULT")){ 19 attempt = 0; 20 solutionFound = false; 21 startTime = System.nanoTime(); 22 bruteForce(listOfShapes, new boolean[listOfShapes.size()], puzzle_board, N, M); 23 endTime = System.nanoTime(); 24 } 25 26 long time = (endTime - startTime) / 1_000_000; 27 28 if(solutionFound){ 29 System.out.println("Solution Found!\n"); 30 print_board(grid_solution); 31 System.out.println("\nEstimated time: " + time + "ms"); 32 System.out.println("Total attempts: " + attempt); 33 } 34 else { 35 System.out.println("No solution!"); 36 System.out.println("Attempts: " + attempt); 37 } 38 39 40 return new PuzzleResult(solutionFound, attempt, time, grid_solution); 41 }</pre>	Algoritma dimulai dengan pengambilan input dari <code>runSolver()</code> . Fungsi ini membaca ukuran papan NXM, bentuk blok, dan mode konfigurasi dari file menggunakan kelas <code>ReadFile</code> . Kemudian sebelum memasuki fungsi iterasi bruteforce, program menginisialisasi papan kosong yang berisi karakter '.' (sel kosong) menggunakan fungsi <code>initialize_board()</code> .
 <pre>1 public static void bruteForce(ArrayList<char[][]> listOfBlocks, boolean[] used, char[][] grid, int rowGrid, int colGrid) { 2 attempt++; 3 if(gameSolved(grid, used)) { 4 grid_solution = grid; 5 solutionFound = true; 6 return; 7 } 8 9 if(!hasValidMoves(listOfBlocks, used, grid, rowGrid, colGrid)) { 10 return; 11 } 12 13 if(solutionFound) { 14 return; 15 } 16 17 for(int i = 0; i < listOfBlocks.size(); i++) { 18 if(used[i]) { 19 used[i] = true; 20 ArrayList<char[][]> transformations = blockTransformation(listOfBlocks.get(i)); 21 22 for(char[][] block : transformations) { 23 if(solutionFound) break; 24 for(int row = 0; row <= rowGrid-block.length; row++) { 25 if(solutionFound) break; 26 for(int col = 0; col <= colGrid-block[0].length; col++) { 27 if(solutionFound) break; 28 if(validPosition(block, grid, block.length, block[0].length, row, col)) { 29 char[][] originalGrid = copyGrid(grid); 30 grid = placeBlock(grid, block, block.length, block[0].length, row, col, rowGrid, colGrid); 31 32 bruteForce(listOfBlocks, used, grid, rowGrid, colGrid); 33 if(solutionFound) { 34 return; 35 } 36 37 grid = originalGrid; 38 39 } 40 } 41 } 42 used[i] = false; 43 } 44 } 45 } 46 }</pre>	Pada fungsi <code>bruteForce()</code> , digunakan fungsi iterasi rekursif untuk mencapai solusi yang benar pada grid, basis dari fungsi rekurens tersebut adalah jika game sudah selesai yaitu pada fungsi <code>gameSolved()</code> , atau pada fungsi <code>!isValidMoves()</code> dimana jika tidak ada blok yang dapat ditaruh lagi pada grid, maka iterasi di selesaikan. Algoritma melakukan iterasi melalui semua blok yang belum ditempatkan dengan menggunakan semua kemungkinan rotasi dan pencerminan blok menggunakan fungsi <code>blockTransformation()</code> lalu menempatkan setiap blok yang telah ditransformasi pada

setiap posisi valid di papan. Setelah menempatkan satu blok, maka memanggil rekursi untuk menempatkan blok berikutnya. Jika papan sudah penuh, rekursi berhenti dan solusi ditemukan, jika sebuah gerakan tidak valid, algoritma melakukan backtracking dan mengembalikan keadaan papan ke sebelumnya dan mencoba opsi berikutnya.



```

1 public static boolean validPosition(char[][] block, char[][] grid, int blockRow, int blockCol, int gridRowPosition, int gridColPosition) {
2     if (gridRowPosition + blockRow > grid.length || gridColPosition + blockCol > grid[0].length) {
3         return false;
4     }
5
6     for (int i = 0; i < blockRow; i++) {
7         for (int j = 0; j < blockCol; j++) {
8             if (block[i][j] != '.') {
9                 if (grid[gridRowPosition + i][gridColPosition + j] != '.') {
10                     return false;
11                 }
12             }
13         }
14     }
15     return true;
16 }
```

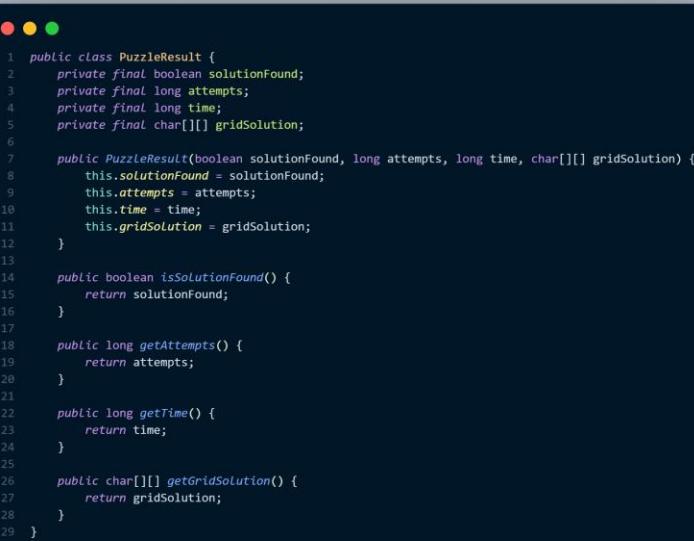
Pada fungsi `validPosition()`, sebelum menempatkan blok, algoritma ini memeriksa apakah blok sesuai dalam batas papan & memastikan blok tidak bertumpukan dengan blok yang sudah ada.



```

1 public static boolean gameSolved(char[][] grid, boolean[] used) {
2     for (char[] row : grid) {
3         for (char c : row) {
4             if (c == '.') {
5                 return false;
6             }
7         }
8     }
9     for (boolean blockUsed : used) {
10         if (!blockUsed) {
11             return false;
12         }
13     }
14     return true;
15 }
```

Pada fungsi `gameSolved()`, sebuah blok ditempatkan, algoritma memeriksa apakah papan tidak memiliki sel kosong dan semua blok yang telah digunakan setidaknya satu kali.



```

1 public class PuzzleResult {
2     private final boolean solutionFound;
3     private final long attempts;
4     private final long time;
5     private final char[][] gridSolution;
6
7     public PuzzleResult(boolean solutionFound, long attempts, long time, char[][] gridSolution) {
8         this.solutionFound = solutionFound;
9         this.attempts = attempts;
10        this.time = time;
11        this.gridSolution = gridSolution;
12    }
13
14    public boolean isSolutionFound() {
15        return solutionFound;
16    }
17
18    public long getAttempts() {
19        return attempts;
20    }
21
22    public long getTime() {
23        return time;
24    }
25
26    public char[][] getGridSolution() {
27        return gridSolution;
28    }
29 }
```

Pada class `PuzzleResult`, adalah class untuk menerima hasil return dari fungsi `runSolver()`, yaitu fungsi iterasi bruteforce yang berasal dari class `GameManager`, fungsi ini tidak berdampak pada flow algoritma, tetapi untuk memperoleh value yang akan digunakan pada controller GUI.

3. Kompleksitas Algoritma

Algoritma brute force yang digunakan dalam program ini memiliki kompleksitas waktu yang bergantung pada beberapa faktor, yaitu:

- n = jumlah blok yang tersedia
- r = jumlah baris pada grid (papan puzzle)
- c = jumlah kolom pada grid
- t = jumlah transformasi setiap blok (rotasi dan pencerminan)

Proses utama dari algoritma ini adalah menempatkan semua blok ke dalam grid dengan mencoba semua kemungkinan posisi dan transformasi. Pada setiap langkah, algoritma akan

- melakukan iterasi melalui semua blok yang belum digunakan $O(n)$
- mencoba setiap transformasi dari blok (rotasi dan pencerminan) $O(t)$
- Mengecek setiap posisi yang memungkinkan pada grid, yaitu $O(r * c)$
- Memeriksa apakah posisi tersebut valid $O(k^2)$

Sehingga, kompleksitas waktu total algoritma ini kira-kira adalah:

$$O(n * t * r * c * k^2)$$

- n untuk iterasi setiap blok
- t untuk setiap transformasi blok (pada kasus ini 8: 4 rotasi + 4 pencerminan)
- $r * c$ untuk seluruh posisi grid
- k^2 untuk validas apakah blok dapat ditempatkan di posisi tertentu

4.3.1 Best Case

Kasus terbaik (Best Case) terjadi ketika solusi ditemukan pada iterasi awal, yaitu saat blok-blok pertama yang dicoba sudah membentuk solusi yang valid. Dalam hal ini, algoritma akan berhenti segera setelah meneumkan solusi. Kompleksitas waktu untuk kasus terbaik mendekati

$$O(1)$$

Ini karena algoritma memiliki kondisi keluar (solutionFound) yang langsung mengakhiri proses begitu solusi ditemukan.

4.3.1 Best Case

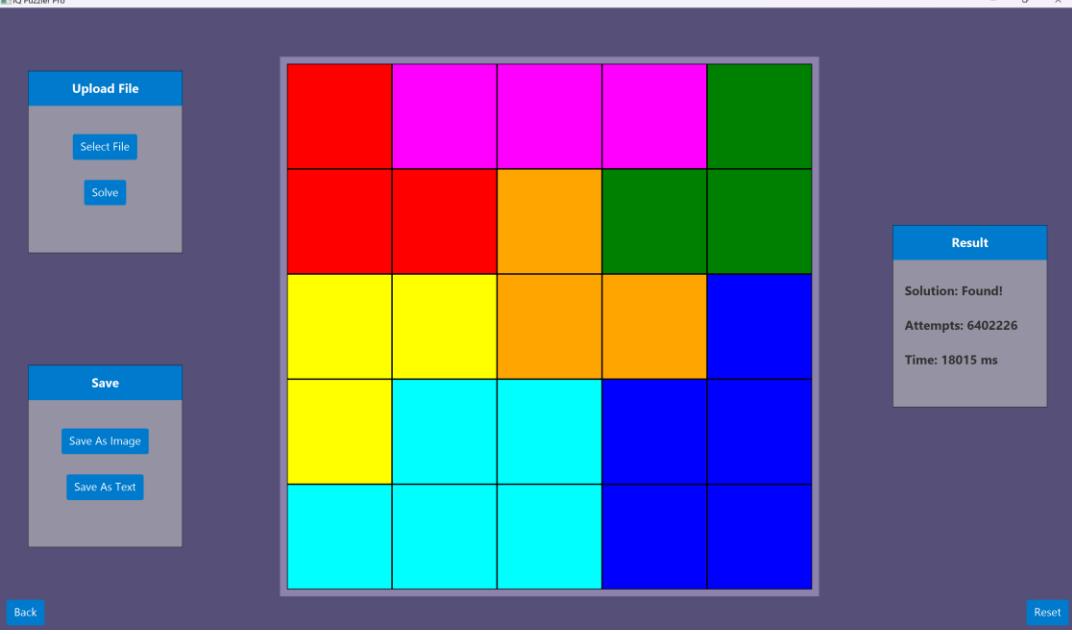
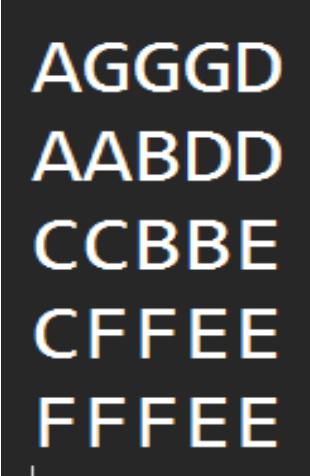
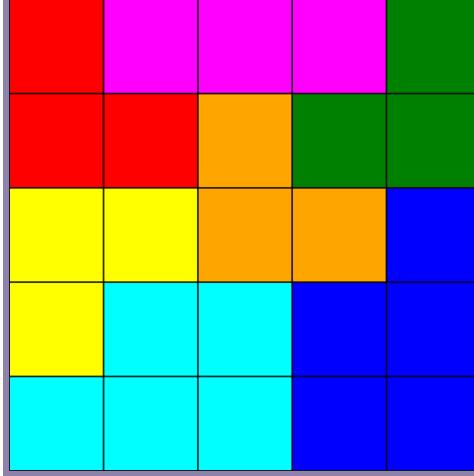
Kasus terburuk (Worst Case) terjadi ketika algoritma harus mencoba semua kemungkinan kombinasi blok dan transformasinya, sampai akhirnya menemukan solusi pada iterasi terakhir atau bahkan tidak menemukan solusi sama sekali. Kompleksitas waktunya dalam kasus ini adalah:

$$O(n! * t * r * c * k^2)$$

- n muncul karena algoritma brute force mencoba semua kemungkinan urutan penempatan blok (permutasi).
- t untuk transformasi blok (rotasi + pencerminan)
- $r * c$ untuk semua posisi grid
- k^2 untuk memvalidasi posisi setiap blok

BAB V

TEST CASE

Input	Output
5 5 7 DEFAULT A AA B BB C CC D DD EE EE E FF FF F GGG	<p>IQ Puzzler Pro</p>  <p>Upload File</p> <p>Select File</p> <p>Solve</p> <p>Save</p> <p>Save As Image</p> <p>Save As Text</p> <p>Back</p> <p>Reset</p> <p>Result</p> <p>Solution: Found!</p> <p>Attempts: 6402226</p> <p>Time: 18015 ms</p>  <p>Testcase1.txt</p>  <p>Testcase1.png</p>

5 11 12

DEFAULT

AAA

A A

BBB

BB

CC

C

C

DD

DD

D

E

EE

E

E

F

FF

G

GGGG

H

HHHH

III

I

J

JJ

K

KK

K

L

L

LLL

IQ Puzzler Pro

Upload File
Select File
Solve

Save
Save As Image
Save As Text

Back Reset

Result

Solution: Found!
Attempts: 16
Time: 0 ms

AAABBBCCDDEE
AFAGBBCDDEE
FFGGGGCDKEL
HFFIIIIJKKEL
HHHHIJJJKLLL

Testcase2.txt

Testcase2.png

8 12 9

DEFAULT

BBBB

BBBBB

BBBBB

BBBBB

BBBBB

BBBBBBBBBB

BBB BBB

BBB BBB

I

I

I

I II

IIII

AAA

AAA

D

DD

DD

DD

EEE

E

E

GG

GG

H

H

HHHH

HH

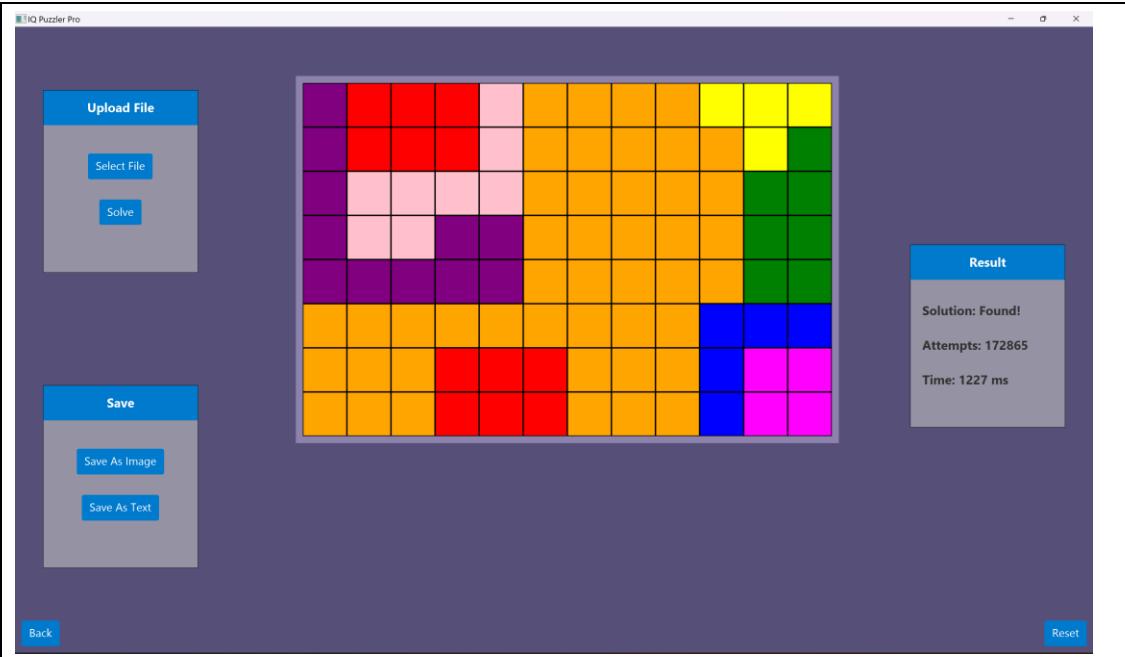
CCC

C

KK

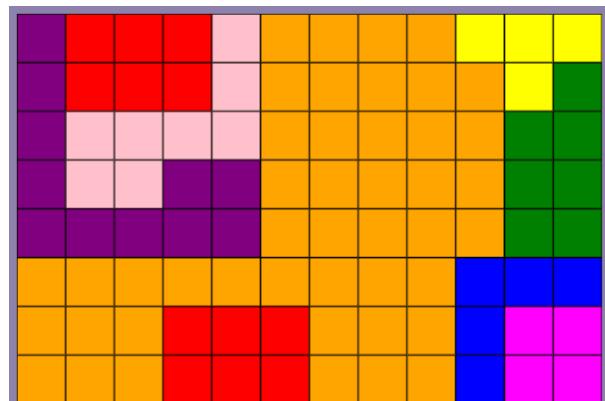
KK

KK

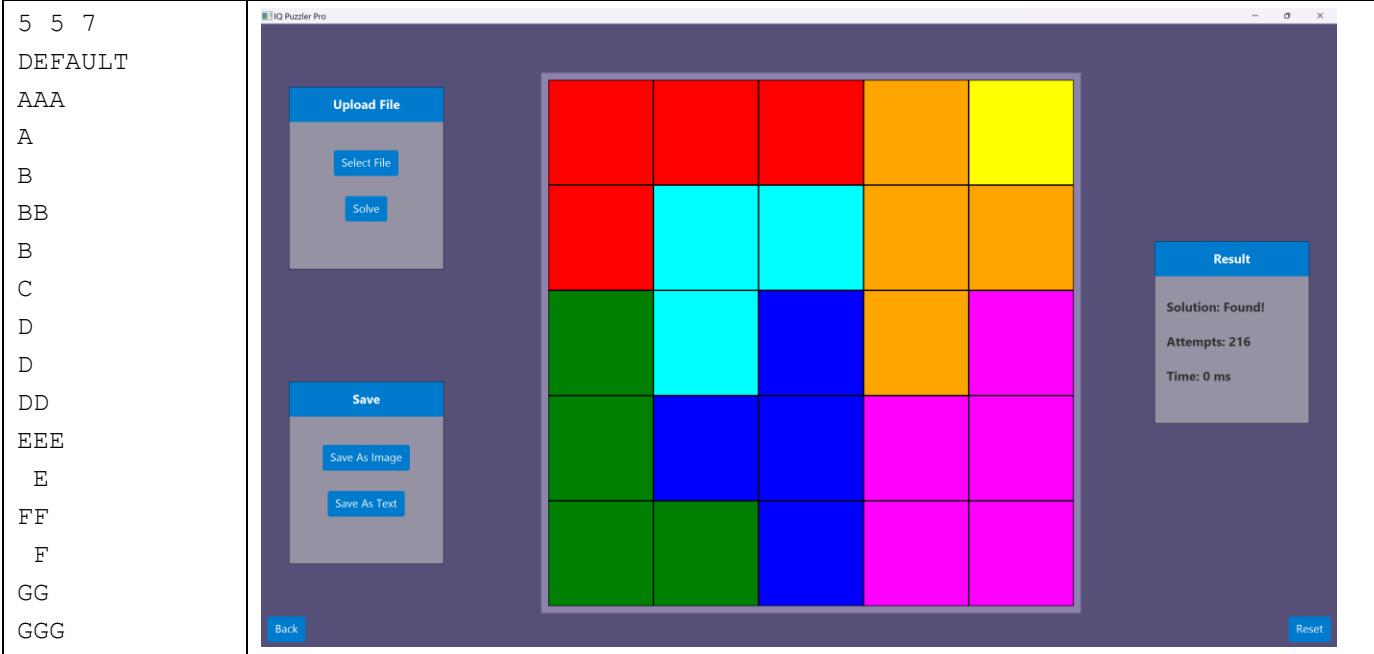


IAAAHBBBBCCC
IAAAHBBBBBCD
IHHHHBBBBBDD
IHIIIBBBBBDD
IIIIIBBBBBDD
BBBBBBBBBEEE
BBBKKKBBBEGG
BBBKKKBBBEGG

Testcase3.txt

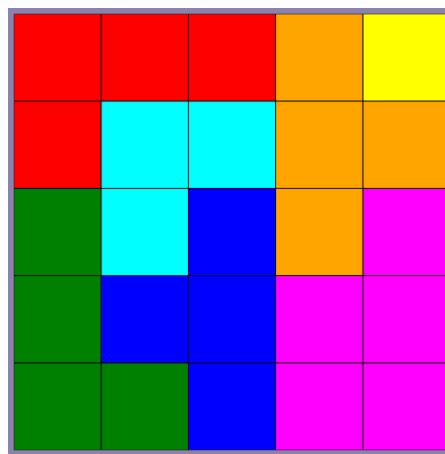


Testcase3.png

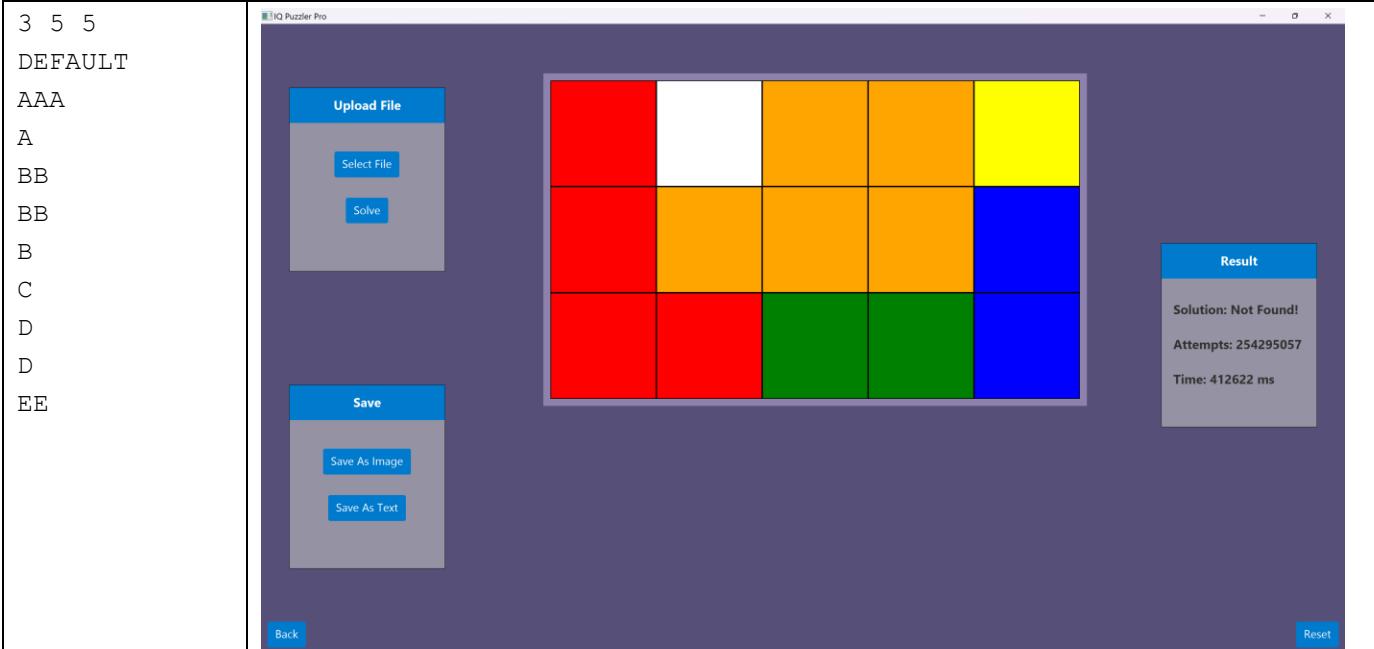


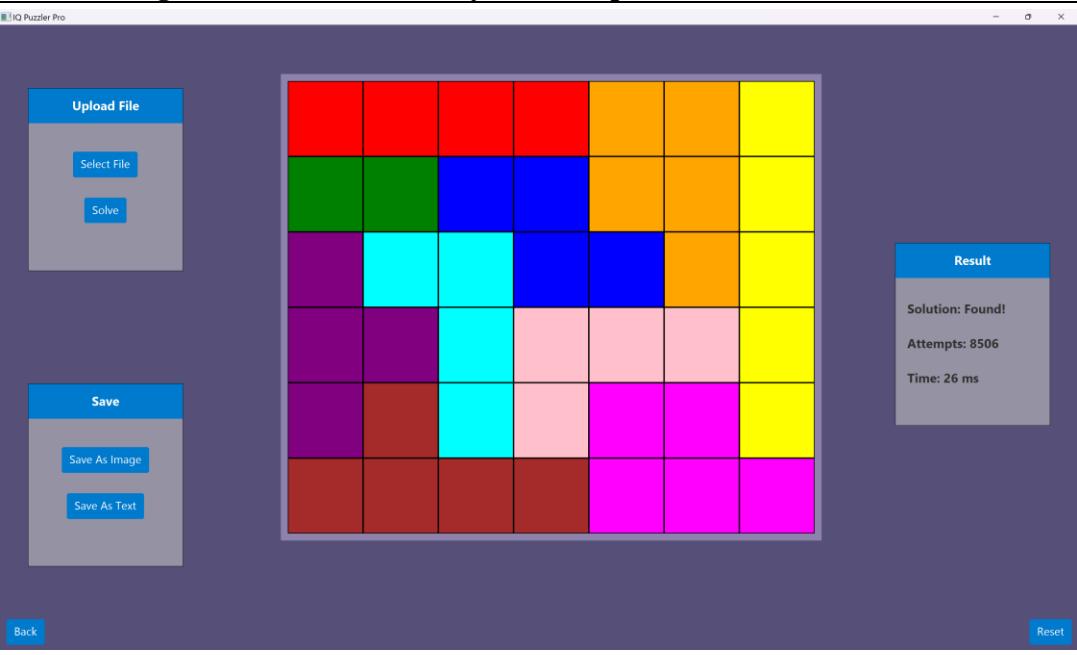
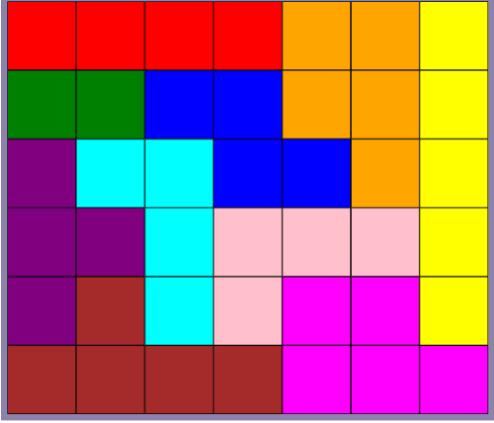
AAABC
AFFBB
DFEBG
DEEGG
DDEGG

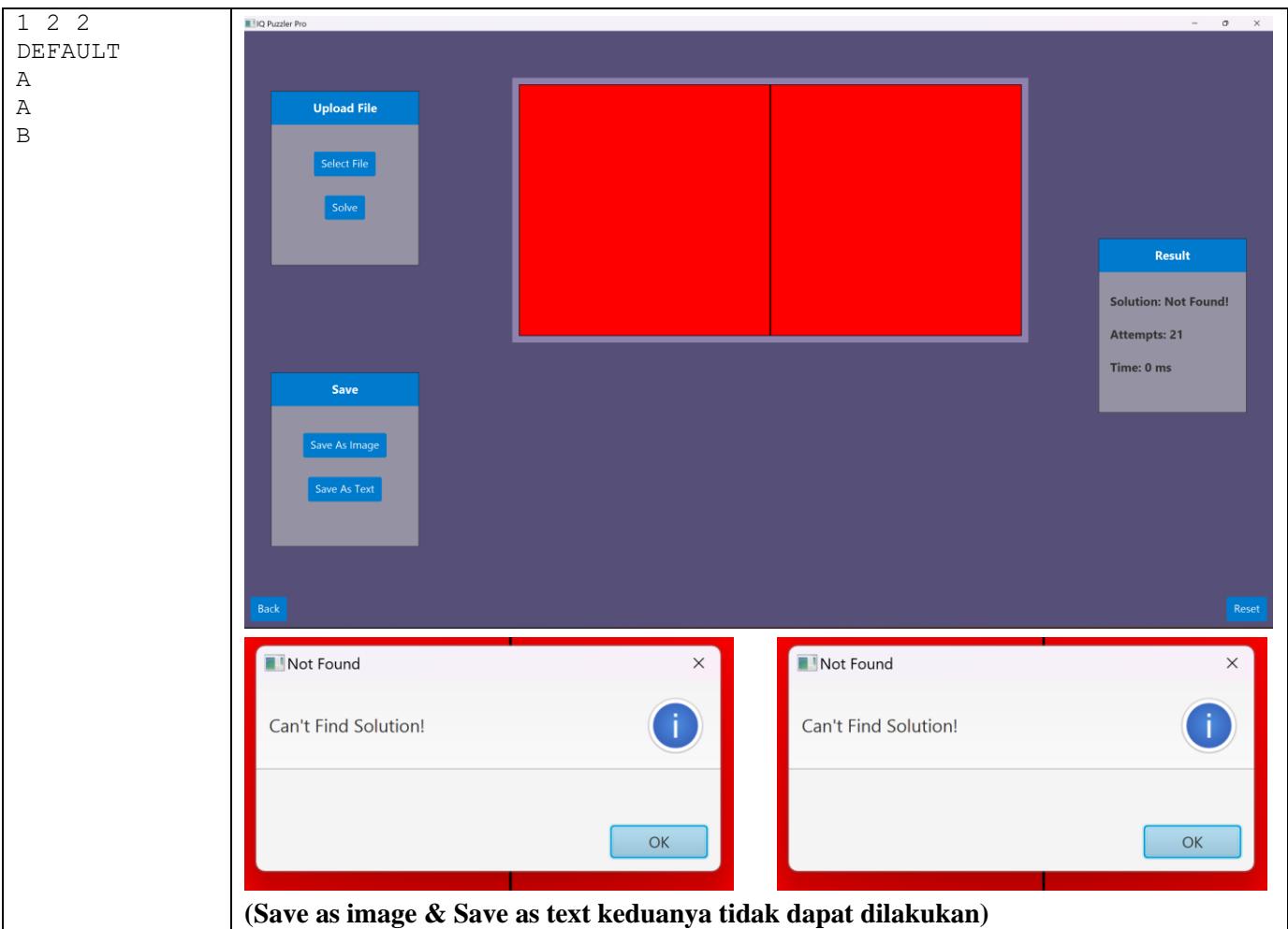
Testcase4.txt



Testcase4.png



		
	<p>(Save as image & Save as text keduanya tidak dapat dilakukan)</p>	
6 7 10 DEFAULT AAAA BB BB B C C C C C DD EE EE FF F F GG GG G HHH H III I J JJJJ		
	<p>Testcase6.txt</p>	<p>Testcase6.png</p>



BAB VI

KESIMPULAN & SARAN

a. Kesimpulan

Algoritma yang digunakan adalah brute force dengan backtracking untuk menyelesaikan teka-teki penyusunan blok. Meskipun metode ini sederhana dan menjamin menemukan solusi jika ada, kompleksitas waktunya sangat tinggi, terutama dalam kasus terburuk. Oleh karena itu, efektivitas algoritma ini sangat bergantung pada optimisasi seperti pruning jalur yang tidak valid.

b. Saran

Tugas kecil ini mewajibkan penggunaan algoritma bruteforce murni dimana dalam beberapa kasus rekursi yang terjadi melibatkan banyak sekali iterasi sehingga waktu eksekusi menjadi lama, untuk tugas kecil selanjutnya sebaiknya penggunaan algoritma diperbolehkan untuk mengimplementasikan beberapa optimalisasi bantuan agar laptop tidak cepat panas dan tidak ngeframe, karena tugas kecil kali ini benar-benar membunuh laptop saya hingga blue screen berkali-kali.

BAB VI

DAFTAR PUSTAKA

1. Sumber Pustaka

- [1] A. Pinchuk, "IQ Puzzler PRO - Solution and research," GitHub. [Online]. Available: <https://github.com/antonpinchuk/iq-puzzler-pro-solution>. [Accessed: Feb. 24, 2025].
- [2] Q. A. Wei, "IQPuzzlerPro-Solver: An efficient solver for the IQ Puzzler Pro game," GitHub repository, 2024. [Online]. Available: <https://github.com/QiuHongAnnaWei/IQPuzzlerPro-Solver> [Accessed: Feb. 24, 2025].

2. Lampiran

1. Github: https://github.com/brii26/Tucil1_13523126

2. Tabel Poin

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	✓	