

TUGAS KECIL 1

IF-2211 STRATEGI ALGORITMA



Dipersiapkan oleh:

Kenneth Poenadi - 13523040

Dosen Pengampu:

Dr. Nur Ulfa Maulidevi, S.T, M.Sc.

Dr. Ir. Rinaldi Munir, M.T.

**PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
JL. GANESA 10, BANDUNG 40132**

2025

BAB 1

ALGORTIMA BRUTE FORCE

1. Pengertian Algoritma Brute Force

Algoritma adalah metode yang terdiri dari serangkaian langkah yang terstruktur dan sistematis untuk menyelesaikan masalah dengan bantuan komputer (Jando & Nani, 2018). Terdapat berbagai macam algoritma yang digunakan zaman sekarang untuk menyelesaikan berbagai masalah yang ada di dunia pemrograman. Salah satu algoritma yang paling dasar adalah algoritma brute force, pendekatan yang lurus atau lempang (straightforward) untuk memecahkan suatu persoalan. Pendekatan ini mencari semua kemungkinan solusi untuk menemukan jawaban yang tepat. Meskipun sederhana dalam implementasi, algoritma ini memiliki kompleksitas waktu yang tinggi, biasanya eksponensial atau faktorial, yang membuatnya kurang efisien untuk masalah dengan ruang pencarian yang besar.

BAB 2

Notasi Algoritmik Program dan Penjelasan

1. Notasi Algoritmik Algoritma Brute Force

function solvePuzzle(List of Char remainingPieces, Matriks of Char board) → boolean

{I.S: remainingPieces dan board terdefinisi}

{F.S: Solusi papan puzzle atau "Tidak ada solusi"}

KAMUS LOKAL:

currentPiece : Matriks of Char

nextPieces : List of Char

rotatedPiece : Matriks of Char

rotation, i, j, N, M, P : integer

ALGORITMA:

if (remainingPieces.isEmpty()) then

if (IsBoardComplete() = true) then

 → true

else

 → false

currentPiece ← remainingPieces[0]

nextPieces ← remainingPieces[1...remainingPieces.size()-1]

pieceToTry <- currentPiece

mirroredPiece <- mirrorPiece(currentPiece);

```

    rotatedPiece ← currentPiece
    mirror traversal[0...1]
        if(mirror = 0) then
            currentOrientation <- pieceToTry
        else
            currentOrientation <- mirroredPiece
    rotation traversal[0...3]
    i traversal[0...N-1]
        j traversal[0...M-1]
            totalLangkah ← totalLangkah + 1
            if(taruhPiece(board, rotatedPiece, i, j) = true) then
                PlacePiece(board, rotatedPiece, i, j)

                if(solvePuzzle(nextPieces, board) = true) then
                    → true

                removePiece(board, rotatedPiece, i, j)

    rotatedPiece ← rotasipiece(rotatedPiece)

    → false

```

2. function taruhPiece(Matriks of Char board, Matriks of Char piece, integer startRow, integer startCol) -> boolean

{I.S: board, piece, startRow, dan startCol terdefinisi}

{F.S: mengembalikan true jika piece dapat ditempatkan pada posisi tersebut, false jika tidak}

KAMUS LOKAL:

pieceRows, pieceCols, i, j : integer

ALGORITMA:

pieceRows \leftarrow piece.length

pieceCols \leftarrow piece[0].length

if (startRow + pieceRows > board.length OR startCol + pieceCols > board[0].length)
then

→ false

i traversal [0...pieceRows-1]

j traversal [0...pieceCols-1]

if (piece[i][j] \neq '.' AND board[startRow + i][startCol + j] \neq '.') then

→ false

→ true

3. function placePiece(Matriks of Char board, Matriks of Char piece, integer startRow, integer startCol) -> void

{I.S: board, piece, startRow, dan startCol terdefinisi}

{F.S: piece ditempatkan pada board mulai dari posisi (startRow, startCol)}

KAMUS LOKAL:

i, j : integer

ALGORITMA:

```
i traversal [0...piece.length-1]
  j traversal [0...piece[0].length-1]
    if (piece[i][j] ≠ '.') then
      board[startRow + i][startCol + j] ← piece[i][j]
```

4. function removepiece(Matriks of Char board, Matriks of Char piece, integer startRow, integer startCol) -> void

{I.S: board dengan piece yang sudah ditempatkan, startRow, dan startCol terdefinisi}

{F.S: piece dihapus dari board}

KAMUS LOKAL:

i, j : integer

ALGORITMA:

```
i traversal [0...piece.length-1]
  j traversal [0...piece[0].length-1]
    if (piece[i][j] ≠ '.') then
      board[startRow + i][startCol + j] ← '.'
```

5. function saveiPiece(Matriks of Char piece) -> Matriks of Char

{I.S: piece terdefinisi}

{F.S: mengembalikan piece yang telah dirotasi 90 derajat searah jarum jam}

KAMUS LOKAL:

rows, cols, i, j : integer

rotated : Matriks of Char

ALGORITMA:

rows \leftarrow piece.length

cols \leftarrow piece[0].length

rotated[cols][rows] \leftarrow matriks karakter baru

i traversal [0...rows-1]

j traversal [0...cols-1]

rotated[j][rows - 1 - i] \leftarrow piece[i][j]

\rightarrow rotated

6. function mirrorPiece(Matriks of Char piece) \rightarrow Matriks Of Char

{I.S: piece terdefinisi}

{F.S: mengembalikan piece yang telah dicerminkan horizontal}

KAMUS LOKAL:

rows, cols, i, j : integer

mirrored : char[][]

ALGORITMA:

rows \leftarrow piece.length

cols \leftarrow piece[0].length

mirrored \leftarrow new char[rows][cols]

i traversal [0...rows-1]

j traversal [0...cols-1]

mirrored[i][cols - 1 - j] \leftarrow piece[i][j]

→ mirrored

7. function IsBoardComplete() -> boolean

{I.S: board terdefinisi}

{F.S: mengembalikan true jika board sudah terisi penuh, false jika masih ada posisi kosong}

KAMUS LOKAL:

i, j : integer

ALGORITMA:

i traversal [0...board.length-1]

j traversal [0...board[0].length-1]

if (board[i][j] = '.') then

→ false

→ true

CARA KERJA PROGRAM

Prosedur Awal:

{I.S: N, M, dan pieces terdefinisi}

{F.S: Menampilkan solusi puzzle jika ada}

KAMUS:

board : Matriks of Char[N][M]

totalLangkah : integer

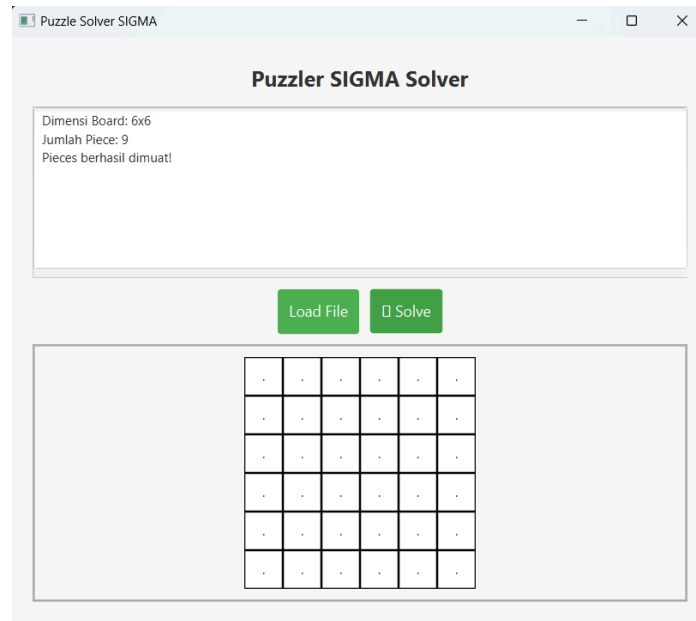
Hasil : boolean

ALGORITMA:

1. Inisialisasi board[N][M] dengan karakter '.'
2. totalLangkah \leftarrow 0
3. Hasil \leftarrow SolvePuzzle(pieces, board)
4. if (Hasil = true) then
 output "Puzzle berhasil diselesaikan!"
 output "Total langkah: " + totalLangkah
else
 output "Tidak ada solusi untuk puzzle ini."
 output "Total langkah: " + totalLangkah

2. Penjelasan Algoritma Brute Force

Program pastinya pertama dimulai dengan tampilan GUI , yang mana saya menggunakan JavaFX versi 23 pada kali ini, program pastinya meminta User untuk upload .txt yang berisikan puzzle yang akan diselesaikan, setelah berhasil mengupload barulah muncul grid, apabila jumlah puzzle yang diberikan pada txt dan puzzle yang diberikan berbeda maka akan muncul showAlert popup bahwa data yang diberikan salah. lalu apabila data benar.



Gambar 1. Contoh Berhasil Load Puzzle (Sumber: Diri Sendiri)

Pertama, program tentunya akan membaca jumlah board M dan N yang ada pada input txt pengguna, setelah mendapatkan M dan N, program akan membuat matriks yang merupakan M x N tersebut dan diisi dengan character ‘.’ yang menandakan bahwa matriks masih kosong, sedangkan puzzle *piece* nya dibuat juga dengan matriks-matriks yang merepresentasikan piece tersebut, berikut contohnya:

```
Piece 1:
[A, .]
[A, A]

Piece 2:
[B, .]
[B, B]

Piece 3:
[C, .]
[C, C]

Piece 4:
[D, .]
[D, D]

Piece 5:
[E, E]
[E, E]
[E, .]

Piece 6:
[F, F]
[F, F]
[F, .]

Piece 7:
[G, G, G]

Piece 1:
[A, A, A]
[., A, .]

Piece 2:
[., B, B]
[B, B, .]
[B, ., .]

Piece 3:
[C, C]
[C, C]
[C, C]
[C, .]

Piece 4:
[D, D, D]

Piece 5:
[E, E, E, E]
[., ., ., E]

Piece 6:
[F, ., ., F]
[F, F, F, F]

Piece 7:
[G, G, .]
[G, G, G]
[., G, .]
```

Gambar 2 Cara Program Memproses *Piece* (Pada Terminal VSCode) (Sumber: Diri Sendiri)

Kedua, setelah berhasil memetakan puzzle piece akan dimasukan ke dalam list of matriks char yang berisi *piece-piece* dari puzzle, program akan mulai masuk ke dalam loopingnya, yang dimulai dari

```
for (int mirror = 0; mirror < 2; mirror++) {
```

Yang berfungsi untuk melakukan mirroring dari puzzle piece kita, 0 sampai 2 karena mirror hanya bisa sekali saja, kedua kali untuk membalikan dia seperti normal, pertama dia akan masuk ke kondisi if else, apabila mirrornya 0 maka dia akan masuk ke **currentOrientation** menggunakan **pieceToTry** (piece original), sedangkan bila sudah tidak 0, maka akan menggunakan mirrored piecenya, setelah itu akan masuk ke looping berikutnya

```
for (int rotation = 0; rotation < 4; rotation++)
```

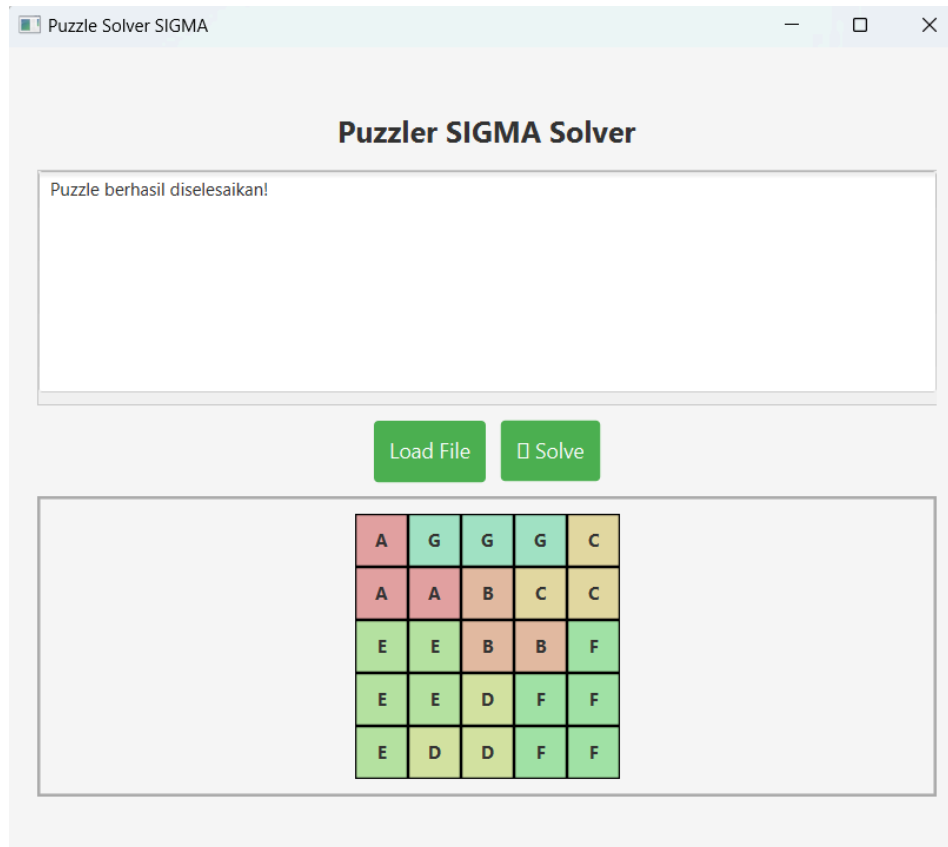
 yang berfungsi untuk melakukan rotasi 0 derajat, 90 derajat, 180 derajat, 270 derajat, selanjutnya

```
for (int i = 0; i < board.length; i++) {

    for (int j = 0; j < board[0].length; j++) {
```

Board.length adalah baris dan board[0].length adalah kolom, program tentunya dimulai dari koordinat 0,0 pada matriks board, program kemudian akan menaruh piece pertama pada koordinat 0,0 (taruhPiece pasti benar untuk piece pertama) , sebelumnya program akan mengecek terlebih dahulu dengan fungsi taruhPiece, fungsi taruhPiece sendiri berguna untuk mengecek apakah piece bisa ditaruh di posisi tertentu, taruhPiece menerima parameter matriks board, piece, startRow (indeks row sekarang), startCol (indeks col sekarang), pertama dia akan mengecek apakah piece memiliki bagian yang melebihi board, apabila iya maka taruhPiece akan langsung return false, sedangkan apabila tidak akan dicek apakah pada koordinat 0,0 memiliki tumpang tindih dengan piece awal? Apabila iya maka return false sedangkan bila tidak akan return true yang menandakan dapat dipasang. Lalu akan lanjut ke koordinat berikutnya yaitu koordinat 0,1 lalu apabila true dia akan langsung memanggil placepiece yang mana akan memetakan piece tersebut ke dalam matriks board. Fungsi placePiece akan mengambil karakter-karakter dari piece yang bukan '.' (yang berarti bagian dari piece) dan menempatkannya pada board sesuai dengan koordinat yang telah ditentukan. Setelah piece berhasil ditempatkan, program akan melakukan langkah rekursif dengan memanggil fungsi solvePuzzle(nextPiece (piece berikutnya), board) untuk sisa piece yang belum ditempatkan. Jika pemanggilan rekursif solvePuzzle mengembalikan nilai true, artinya solusi telah ditemukan dengan susunan piece saat ini, maka program akan mengembalikan nilai true dan proses selesai. Namun, jika mengembalikan false, artinya tidak ada solusi yang dapat dibentuk dengan penempatan piece saat ini, program akan melakukan backtracking. Backtracking dilakukan dengan memanggil fungsi removePiece yang akan menghapus piece yang baru saja ditempatkan dari board (mengembalikan posisi tersebut menjadi karakter '.'). Setelah itu, program akan melakukan rotasi dengan memanggil rotasipiece yang akan merotasi 90 derajat. Setiap kali mencoba menempatkan piece, variabel total langkah akan bertambah satu untuk mencatat jumlah kemungkinan yang telah dilakukan. Jika semua piece berhasil ditempatkan (remainingPieces kosong), program akan memanggil fungsi isBoardComplete untuk memastikan bahwa seluruh board terisi (tidak ada karakter '.' tersisa). Jika benar, program mengembalikan true, artinya puzzle telah diselesaikan dan jika tidak maka akan memanggil showAlert yang menandakan bahwa program gagal. Apabila sudah berhasil semua maka akan memanggil drawboard yang mana akan menggambarkan ke GUI solusi yang didapatkan, pengguna dapat memilih untuk menyimpan solusinya.

Secara keseluruhan program yang salah jalankan memiliki **kompleksitas big(O) = $O(4^P \times (N \times M)^P)$** yang menandakan bahwa kasus terburuk adalah program berjalan dengan kompleksitas eksponensial. 4 menandakan jumlah rotasi, sehingga kalau ada P pieces, maka di kasus terburuk, kita coba 4^P kemungkinan rotasi. Kita harus mencoba semua kemungkinan penempatan di board, yaitu $(N \times M)^P$ kombinasi, dan terakhir



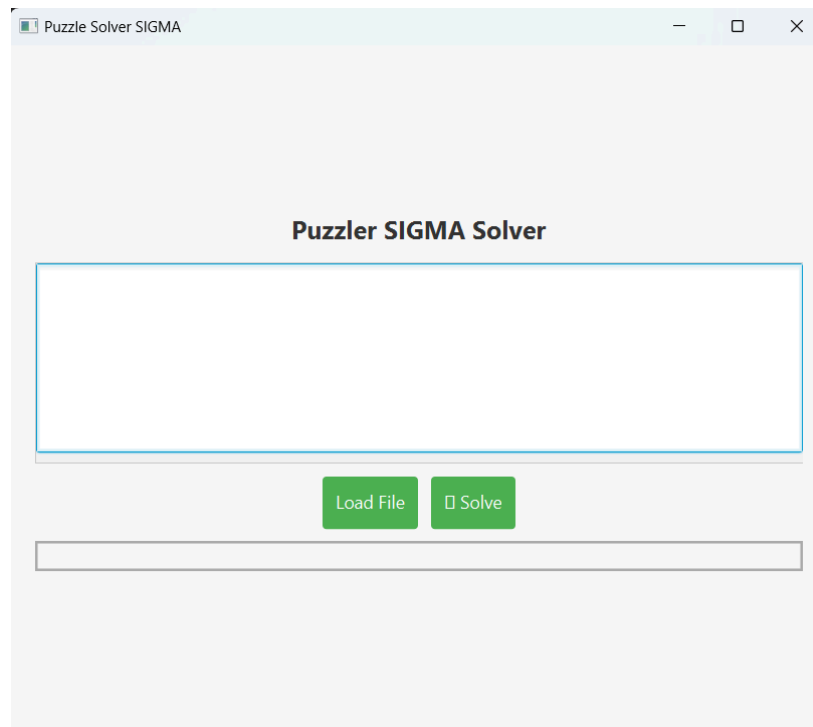
Gambar 3. Contoh Solusi dari Puzzle (Sumber: Diri Sendiri)

BAB 3

Demonstrasi dan Test Case

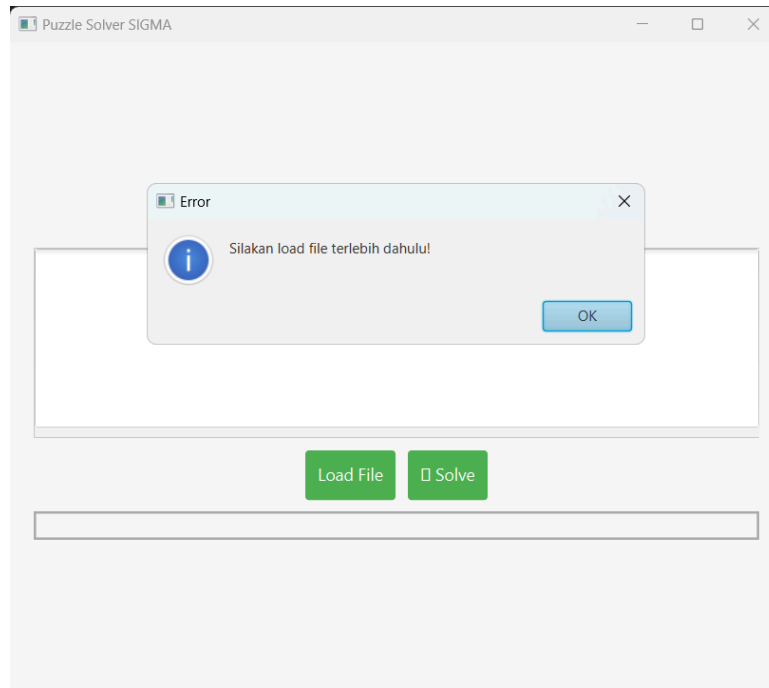
1. Demonstrasi

- Tampilan awal saat membuka program:



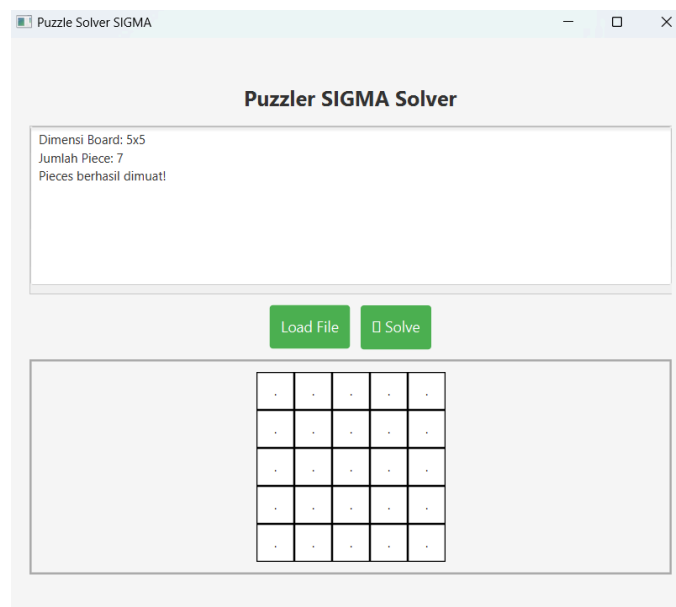
Gambar 4. Tampilan Awal GUI Program (Sumber: Diri Sendiri)

- Memencet tombol solve sebelum load file



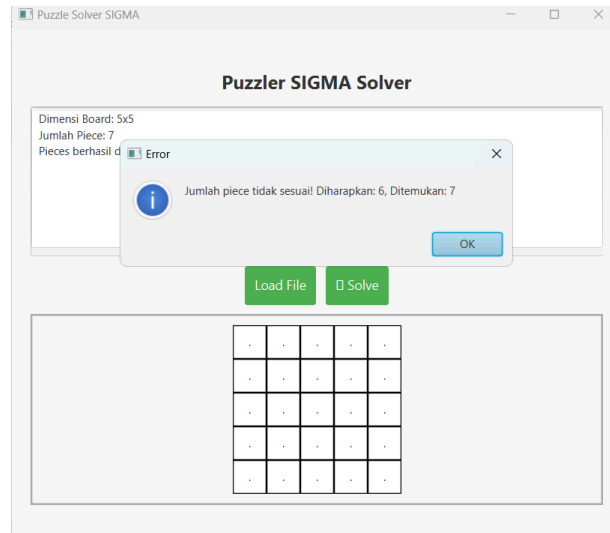
Gambar 5. Tampilan saat menekan tombol solve sebelum meload file apapun (Sumber: Diri Sendiri)

- Tampilan saat berhasil *load file*



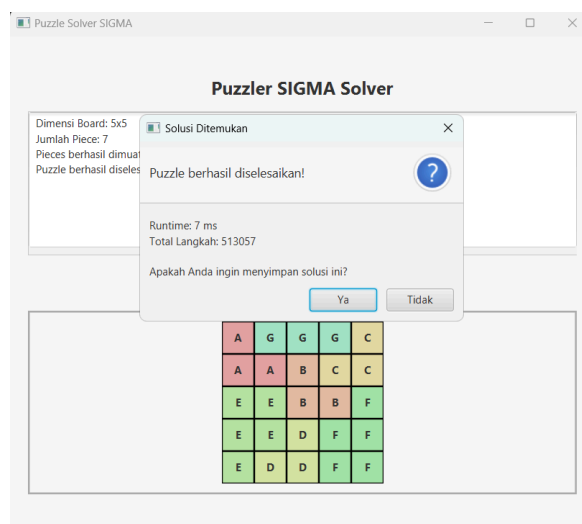
Gambar 6. Tampilan saat berhasil load file (Sumber: Diri Sendiri)

- Tampilan saat data file tidak sesuai



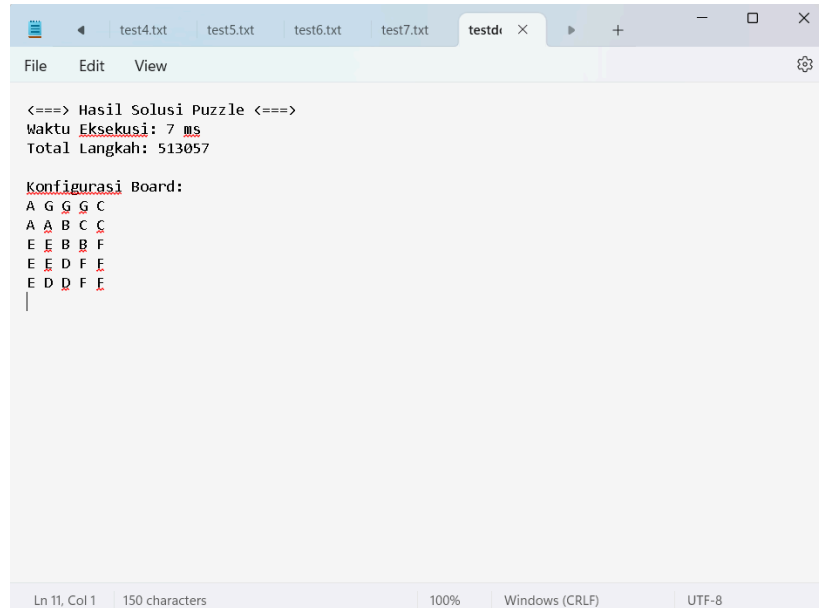
Gambar 7. Tampilan saat data file tidak sesuai (Sumber: Diri Sendiri)

- Tampilan saat puzzle berhasil di solve



Gambar 8. Tampilan saat ditemukan solusi dari Puzzle (Sumber: Diri Sendiri)

- Tampilan file.txt dan .png



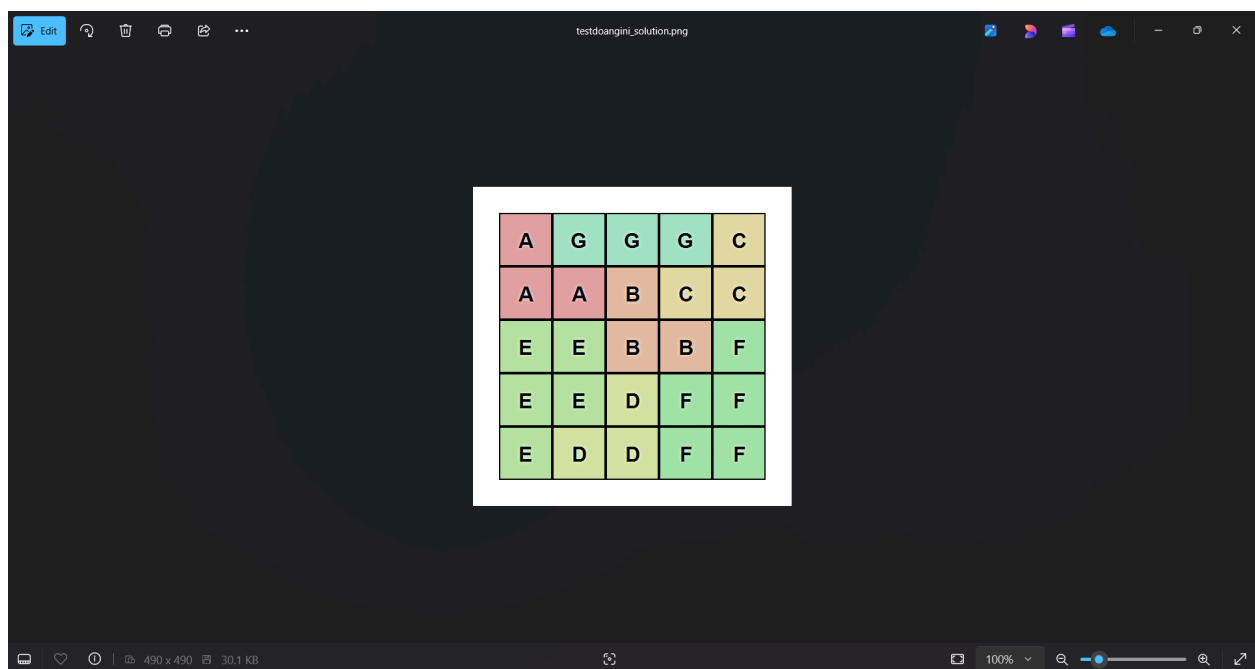
The screenshot shows a text editor window with the following text:

```
<==> Hasil Solusi Puzzle <==>
Waktu Eksekusi: 7 ms
Total Langkah: 513057

Konfigurasi Board:
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
|
```

The status bar at the bottom indicates: Ln 11, Col 1 | 150 characters | 100% | Windows (CRLF) | UTF-8

Gambar 9. Tampilan .txt solusi dari Puzzle (Sumber: Diri Sendiri)



Gambar 10. Tampilan .png solusi dari Puzzle (Sumber: Diri Sendiri)

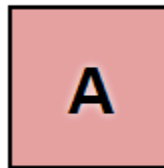
2. Test Case

2.1 Test Case 1

- Input:

1 1 1
DEFAULT
A

- Output:



Gambar 11. Tampilan .png solusi dari Test Case 1 (*Sumber: Diri Sendiri*)

<====> Hasil Solusi Puzzle <====>

Waktu Eksekusi: 0 ms

Total Langkah: 1

Konfigurasi Board:

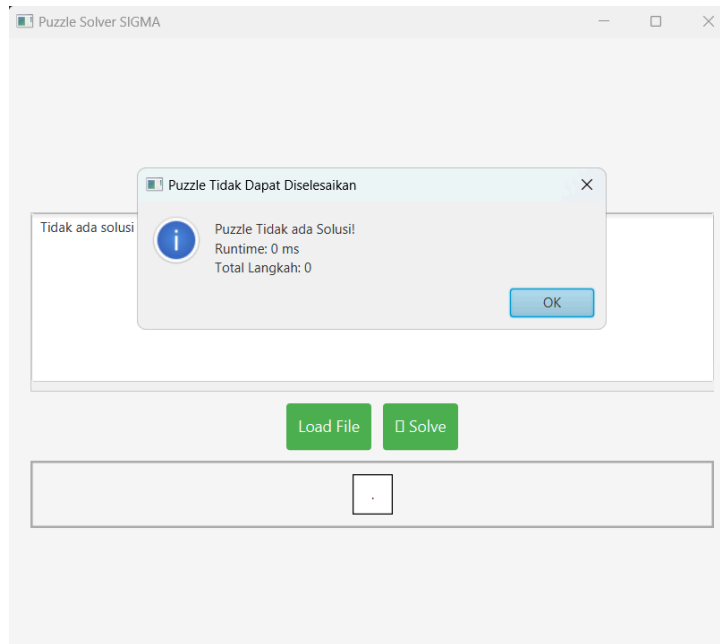
A

2.2 Test Case 2

- Input:

1 1 1
DEFAULT
AA
A

- Output:



Gambar 12. Tampilan .png solusi dari Test Case 2 (Sumber: Diri Sendiri)

2.3 Test Case 3

- Input:

2 2 3
 DEFAULT
 A
 BB
 C

- Output:

A	C
B	B

Gambar 13. Tampilan .png solusi dari Test Case 3 (*Sumber: Diri Sendiri*)

<===> Hasil Solusi Puzzle <===>

Waktu Eksekusi: 0 ms

Total Langkah: 6

Konfigurasi Board:

A C

B B

2.4 Test Case 4

- Input:

3 3 9

DEFAULT

A

B

C

D

E

F

G

H

I

- Output:

A	B	C
D	E	F
G	H	I

Gambar 14. Tampilan .png solusi dari Test Case 4 (*Sumber: Diri Sendiri*)

<====> Hasil Solusi Puzzle <====>

Waktu Eksekusi: 0 ms

Total Langkah: 45

Konfigurasi Board:

A B C

D E F

G H I

2.5 Test Case 5

- Input:

5 5 7

DEFAULT

A

AA

B

BB

C

CC

D

DD

EE

EE

E

FF

FF
F
GGG

- Output:

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

Gambar 15. Tampilan .png solusi dari Test Case 5 (*Sumber: Diri Sendiri*)

<===> Hasil Solusi Puzzle <===>

Waktu Eksekusi: 6 ms

Total Langkah: 513057

Konfigurasi Board:

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

2.6 Test Case 6

- Input:

6 6 9
DEFAULT

AA
 BBBB
 CC
 DD
 DDDDDD
 EEE
 FFF
 FFFFFFFF
 GG
 G
 H
 HH
 II

- Output:

A	A	B	B	B	B
C	C	G	G	H	H
D	D	G	I	I	H
D	D	D	D	D	D
F	F	F	E	E	E
F	F	F	F	F	F

Gambar 16. Tampilan .png solusi dari Test Case 6 (*Sumber: Diri Sendiri*)

<====> Hasil Solusi Puzzle <====>

Waktu Eksekusi: 0 ms

Total Langkah: 3785

Konfigurasi Board:

A A B B B B

C C G G H H

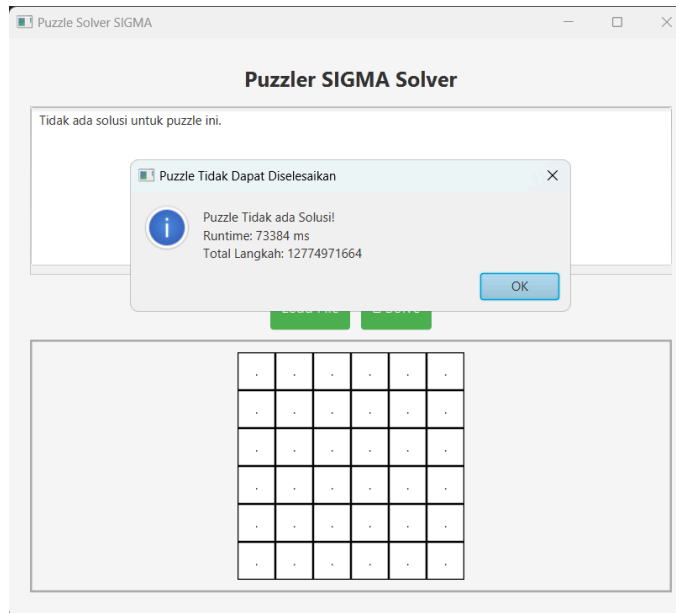
DDGIIH
DDDDDD
FFEEEE
FFFFFF

2.7 Test Case 7

- Input:

6 6 9
DEFAULT
AA
BBBB
CC
DD
DDDDDD
EEE
FFF
FFFFFF
GG
G
H
HH
I

- Output:



Gambar 17. Tampilan .png solusi dari Test Case 7 (Sumber: Diri Sendiri)

2.8 Test Case 8

- Input:

6 6 7
 DEFAULT
 AAA
 A
 BB
 BB
 B
 CC
 CC
 CC
 C
 DDD
 EEEE
 E
 F F
 FFFF
 GG
 GGG
 G

- Output:

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

Gambar 18. Tampilan .png solusi dari Test Case 8 (*Sumber: Diri Sendiri*)

<===> Hasil Solusi Puzzle <===>

Waktu Eksekusi: 2086 ms

Total Langkah: 312751815

Konfigurasi Board:

B A A A G G

B B A G G G

F B B F G C

F F F F C C

D D D E C C

E E E E C C

2.9 Test Case 9

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

Gambar 19. Tampilan .png solusi dari Test Case 9 (Sumber: Diri Sendiri)

<===> Hasil Solusi Puzzle <===>

Waktu Eksekusi: 145 ms

Total Langkah: 9359308

Konfigurasi Board:

A A A F F C

G A F F C C


G G F B B C

E E D B B B

E D D D B H

E E D D H H

REFERENSI

1. Munir, R. (2010). *Strategi Algoritma*. Bandung: Penerbit ITB.
<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/stmik.htm>
2.  Spesifikasi Tugas Kecil 1 Stima 2024/2025

LAMPIRAN

Repository source code program: https://github.com/KennethhPoenadi/Tucil1_13523040

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	

Tabel 1. Review Program (Sumber:  Spesifikasi Tugas Kecil 1 Stima 2024/2025)