Tugas Kecil 1 IF2211 Strategi Algoritma

Penyelesaian Permainan Queens Linkedin dengan Algoritma Brute Force

Dosen Pengajar: Prof. Dr. Ir. Rinaldi, M.T.

Disusun oleh:

13524071 - Kalyca Nathania Benedicta Manullang

Kelas 01

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2026

DAFTAR ISI

BAB I
DESKRIPSI MASALAH DAN ALGORITMA

1.1. Algoritma Brute Force

Algoritma *Brute Force* merupakan salah satu pendekatan paling dasar dalam penyelesaian masalah komputasi yang dilakukan dengan cara mengeksplorasi seluruh kemungkinan solusi yang ada secara sistematis hingga diperoleh solusi yang memenuhi seluruh syarat permasalahan. Pendekatan ini tidak melibatkan strategi khusus untuk mengurangi ruang pencarian, tetapi memeriksa setiap kandidat solusi secara langsung berdasarkan aturan yang telah ditentukan.

Pendekatan Brute Force sering disebut sebagai metode *straightforward* atau metode langsung karena proses penyelesaiannya bersifat eksplisit, sederhana, dan mudah dipahami. Dalam banyak kasus, algoritma ini menjadi pilihan awal dalam perancangan solusi karena tidak memerlukan analisis yang kompleks dan relatif mudah untuk diimplementasikan.

Dalam perancangan algoritma Brute Force, terdapat dua komponen penting yang harus diperhatikan. Pertama adalah perumusan masalah (*problem statement*) yang jelas sehingga seluruh kemungkinan solusi dapat didefinisikan dengan tepat. Kedua adalah pendefinisian ruang solusi**,** yaitu himpunan semua kandidat solusi yang mungkin muncul dari permasalahan tersebut. Setelah ruang solusi ditentukan, algoritma Brute Force akan melakukan iterasi terhadap setiap kandidat dan mengevaluasinya berdasarkan kriteria validitas yang telah ditetapkan. Sebagai contoh, pada permasalahan pencarian pasangan titik dengan jarak terdekat (*closest pair problem*), algoritma Brute Force bekerja dengan menghitung jarak Euclidean antara setiap pasangan titik yang ada. Semua hasil perhitungan tersebut kemudian dibandingkan untuk menentukan pasangan dengan jarak minimum. Jika terdapat n buah titik, jumlah pasangan yang perlu diperiksa adalah $\frac{n(n-1)}{2}$ sehingga kompleksitas waktu algoritma ini berada pada orde $O(n^2)$. Hal ini menunjukkan bahwa jumlah operasi meningkat secara kuadratik seiring bertambahnya ukuran *input*.

Karakteristik utama algoritma Brute Force adalah efisiensinya yang rendah jika diterapkan pada permasalahan dengan ukuran *input* yang besar. Karena seluruh kemungkinan solusi diperiksa satu per satu, algoritma ini sering kali memerlukan waktu eksekusi yang sangat lama serta sumber daya komputasi yang besar sehingga algoritma Brute Force sering dikategorikan sebagai algoritma naif, terutama jika dibandingkan dengan algoritma lain yang telah menerapkan teknik optimasi. Meskipun demikian, algoritma Brute Force memiliki keunggulan yang tidak dapat diabaikan. Salah satu kelebihan utamanya adalah jaminan kebenaran solusi karena tidak ada kemungkinan solusi yang terlewatkan dalam proses pencarian.

Algoritma Brute Force banyak digunakan dalam berbagai permasalahan komputasi dasar, seperti pencarian elemen dalam struktur data, pengurutan sederhana, pencocokan *string*, serta operasi matriks. Selain itu, metode ini sering diterapkan dalam permasalahan kombinatorial, seperti pemecahan teka-teki, penyusunan konfigurasi tertentu, dan permainan berbasis aturan yang ketat. Dalam kasus-kasus tersebut, algoritma Brute Force berfungsi sebagai pendekatan awal untuk mengevaluasi seluruh kemungkinan solusi sebelum mempertimbangkan metode yang lebih efisien. Namun, keterbatasan algoritma Brute Force menjadi semakin jelas ketika ukuran *input* bertambah besar, terutama pada permasalahan dengan pertumbuhan ruang solusi secara eksponensial atau faktorial. Dalam kondisi tersebut, penggunaan algoritma Brute Force sebagai solusi utama menjadi tidak praktis. Oleh karena itu, dalam praktiknya, algoritma ini sering digunakan sebagai *baseline solution* yang kemudian dibandingkan atau dikembangkan lebih lanjut menggunakan teknik optimasi, seperti *divide and conquer, greedy algorithm*, atau *dynamic programming.*

1.2. Queens Linkedin

Permainan Queens Linkedin merupakan sebuah permainan logika berbasis papan yang terinspirasi dari permasalahan klasik *N-Queens Problem* dengan penambahan batasan berupa pembagian papan ke dalam beberapa wilayah (*region*). Tujuan utama dari permainan ini adalah menempatkan sejumlah ratu (*queens*) pada papan berukuran $N \times N$ sedemikian rupa sehingga seluruh aturan permainan terpenuhi. Dalam permainan ini, setiap baris dan setiap kolom pada papan harus ditempati oleh tepat satu ratu. Selain itu, tidak diperbolehkan adanya dua *queen* yang berada pada sel yang saling bersentuhan secara diagonal maupun vertikal pada baris yang berdekatan sesuai aturan Queens Linkedin. Aturan tambahan yang membedakan permainan Queens Linkedin dari versi klasik adalah adanya pembagian papan ke dalam beberapa region yang ditandai dengan simbol tertentu. Setiap region hanya boleh mengandung satu ratu sehingga ratu-ratu yang ditempatkan harus berada pada region yang berbeda.

Papan permainan direpresentasikan dalam bentuk matriks dua dimensi, yaitu setiap sel memiliki label region berupa huruf kapital. Label tersebut menunjukkan bahwa sel-sel dengan huruf yang sama tergolong ke dalam satu region yang sama. Jumlah region pada papan dibatasi dan tidak melebihi jumlah maksimum huruf kapital alfabet, yaitu 26 region. Pembagian region ini menambah tingkat kompleksitas permainan karena selain memperhatikan baris, kolom, dan diagonal, pemain juga harus memastikan bahwa setiap ratu berada pada region yang unik. Dengan adanya berbagai batasan tersebut, tidak semua konfigurasi papan memiliki solusi yang valid. Pada beberapa konfigurasi tertentu, tidak mungkin untuk menempatkan ratu sehingga seluruh aturan terpenuhi secara bersamaan. Oleh karena itu, permainan Queens Linkedin tidak hanya menuntut pencarian solusi, tetapi juga kemampuan untuk menentukan apakah suatu konfigurasi papan memiliki solusi atau tidak. Permainan ini sangat cocok dijadikan studi

kasus dalam penerapan algoritma pencarian solusi, khususnya algoritma Brute Force. Hal ini disebabkan oleh sifat permasalahan yang kombinatorial, yaitu jumlah kemungkinan penempatan ratu meningkat secara signifikan seiring dengan bertambahnya ukuran papan. Setiap konfigurasi kandidat solusi harus diperiksa terhadap seluruh aturan permainan sehingga proses validasi menjadi bagian penting dari penyelesaian permasalahan.

1.3. Algoritma Penyelesaian Queens Linkedin dengan Algoritma Brute Force

Pada program penulis, permasalahan Queens Linkedin diselesaikan menggunakan algoritma brute force dengan pendekatan enumerasi seluruh kemungkinan penempatan *queen* pada papan permainan.

```
procedure ReadBoardFromFile(input filename : string)
{Membaca file masukan dan membentuk papan permainan beserta region}

Deklarasi:
    br          : BufferedReader
    line        : string
    lines       : array of string
    board       : array of array of char
    uniqueReg   : set of char
    N           : integer
    i, j        : integer
    c           : char

Algoritma:
    open file filename for reading
    initialize lines as empty array

    while not end of file do
        read line
        if trim(line) = "" then
            raise error "Terdapat baris kosong"
        endif
        add trim(line) to lines
    endwhile

    if size(lines) = 0 then
        raise error "File kosong"
    endif

    N ← size(lines)

    if length(lines[0]) ≠ N then
        raise error "Board harus berbentuk persegi NxN"
    endif

    initialize board[N][N]
    initialize uniqueReg as empty set

    for i ← 0 to N−1 do
        line ← lines[i]

        if length(line) ≠ N then
            raise error "Panjang baris tidak konsisten"
        endif

        for j ← 0 to N−1 do
            c ← line[j]

            if c not in 'A'..'Z' then
                raise error "Region harus berupa huruf A-Z"
            endif

            board[i][j] ← c
            add c to uniqueReg
        endfor
    endfor

    if size(uniqueReg) > 26 then
        raise error "Jumlah region melebihi 26"
    endif

    if size(uniqueReg) ≠ N then
        raise error "Jumlah region harus sama dengan ukuran board"
    endif

    close file

end procedure
```

```
function NextPermutation(input/output arr : array of integer) → boolean
{Menghasilkan permutasi berikutnya secara leksikografis dengan memodifikasi arr secara
langsung. Mengembalikan true jika permutasi berikutnya berhasil dibuat dan false jika arr
sudah merupakan permutasi terakhir}

Deklarasi:
    i, j, left, right, temp : integer
    n : integer

Algoritma:
    n ← length(arr)

    if n ≤ 1 then
        return false
    endif

    i ← n − 2

    while i ≥ 0 and arr[i] ≥ arr[i + 1] do
        i ← i − 1
    endwhile

    if i < 0 then
        return false
    endif

    j ← n − 1
    while arr[j] ≤ arr[i] do
        j ← j − 1
    endwhile

    {swap arr[i] dan arr[j]}
    temp ← arr[i]
    arr[i] ← arr[j]
    arr[j] ← temp

    {reverse bagian kanan}
    left ← i + 1
    right ← n − 1

    while left < right do
        temp ← arr[left]
        arr[left] ← arr[right]
        arr[right] ← temp

        left ← left + 1
        right ← right − 1
    endwhile

    return true
end function
```

```
function SolveQueensBruteForce(input board : Board, input N : integer) → boolean
{Menyelesaikan permainan Queens Linkedin menggunakan metode brute force dengan membangkitkan
dan memeriksa semua permutasi posisi queen}

Deklarasi:
    queens         : array [0..N−1] of integer
                     {queens[i] menyatakan kolom queen pada baris ke-i}
    iterationCount : long
    i              : integer

Algoritma:

    {Inisialisasi permutasi pertama}
    for i ← 0 to N−1 do
        queens[i] ← i
    endfor

    iterationCount ← 0

    while true do

        iterationCount ← iterationCount + 1

        {Periksa apakah konfigurasi saat ini valid}
        if IsValidPlacement(queens, board, N) then
            return true
        endif

        {Bangkitkan permutasi berikutnya}
        if NextPermutation(queens) = false then
            break
        endif

    endwhile

    {Tidak ditemukan solusi}
    return false

end function
```

```
function IsValidPlacement(input queens : array of integer,
                          input board  : Board,
                          input N      : integer) → boolean
{Memeriksa apakah konfigurasi queen valid berdasarkan aturan adjacency dan region}

Deklarasi:
    {tidak ada}

Algoritma:

    if CheckAdjacent(queens, N) = false then
        return false
    endif

    if CheckRegion(queens, board, N) = false then
        return false
    endif

    return true

end function
```

```
function CheckAdjacent(input queens : array of integer,
                       input N      : integer) → boolean
{Memastikan queen pada baris yang berdekatan tidak saling bersentuhan secara vertikal
maupun diagonal}

Deklarasi:
    i : integer

Algoritma:
    for i ← 0 to N-2 do

        if abs(queens[i] - queens[i+1]) ≤ 1 then
            return false
        endif

    endfor

    return true

end function
```

```
function CheckRegion(input queens : array of integer,
                     input board  : Board,
                     input N      : integer) → boolean
{Memastikan setiap queen berada pada region yang berbeda}

Deklarasi:
    usedRegion : set of char
    i          : integer
    r          : char

Algoritma:
    initialize usedRegion as empty set

    for i ← 0 to N-1 do

        r ← board.getRegion(i, queens[i])

        if r in usedRegion then
            return false
        endif

        add r to usedRegion

    endfor

    return true

end function
```

1. Algoritma dimulai dengan membaca file input menggunakan kelas `Board` yang berisi representasi papan berukuran $N \times N$ dengan setiap sel memiliki label region berupa huruf A sampai Z. File dibaca baris demi baris menggunakan `BufferedReader`, kemudian setiap baris divalidasi. Setelah validasi berhasil, ukuran papan disimpan dalam variabel `size` dan struktur region disimpan dalam array dua dimensi `regions`. Tahap ini merupakan tahap inisialisasi masalah, yaitu data masukan direpresentasikan dalam struktur data yang sesuai agar dapat diproses oleh algoritma. Sebelum algoritma brute force dijalankan, persoalan harus direpresentasikan dalam bentuk yang memungkinkan semua kandidat solusi dapat dibangkitkan secara sistematis.
2. Setelah papan berhasil dimuat, kelas `QueensSolver` menginisialisasi sebuah array satu dimensi bernama `queens` dengan panjang N, yaitu indeks array merepresentasikan baris dan

nilai pada indeks tersebut merepresentasikan kolom tempat *queen* ditempatkan pada baris tersebut. Array ini diinisialisasi dengan nilai awal {0, 1, 2, ..., N−1} yang merupakan permutasi pertama secara leksikografis. Representasi ini memastikan bahwa setiap *queen* berada pada baris yang berbeda dan kolom yang berbeda karena setiap nilai kolom muncul tepat satu kali dalam array. Dengan demikian, setiap kemungkinan konfigurasi penempatan *queen* dapat direpresentasikan sebagai permutasi dari array tersebut. Tahap ini merupakan bagian dari pemodelan ruang solusi yang merupakan langkah penting dalam brute force karena seluruh kandidat solusi akan dibangkitkan secara sistematis dari representasi ini.

3. Algoritma kemudian mulai membangkitkan semua kemungkinan konfigurasi *queen* menggunakan metode `PermutationGenerator.nextPermutation()`. Metode ini menghasilkan permutasi berikutnya secara sistematis dalam urutan leksikografis hingga seluruh permutasi telah dihasilkan. Setiap permutasi merepresentasikan satu kandidat solusi yang mungkin. Karena jumlah permutasi dari N elemen adalah $N!$, algoritma secara sistematis mengenumerasi hingga $N!$ kemungkinan konfigurasi atau hingga solusi ditemukan. Proses enumerasi seluruh ruang solusi ini merupakan inti dari algoritma brute force. Proses pembangkitan seluruh permutasi ini secara langsung mencerminkan prinsip exhaustive search karena algoritma tidak melewatkan kemungkinan solusi apa pun.
4. Algoritma kemudian melakukan evaluasi untuk menentukan apakah konfigurasi tersebut merupakan solusi yang valid menggunakan metode `isValid()` yang memanggil metode `checkAdjacent()` dan `checkRegion()`. Pemeriksaan dilakukan berdasarkan representasi solusi yang digunakan, yaitu setiap *queen* sudah pasti berada pada baris yang berbeda karena direpresentasikan sebagai sebuah permutasi. Oleh karena itu, metode `checkAdjacent()` hanya memeriksa queen pada baris yang berurutan sesuai dengan definisi *constraint* yang diberikan pada permasalahan, yaitu larangan adanya dua *queen* yang berada pada baris bertetangga dengan selisih kolom kurang dari atau sama dengan satu. Jika selisih kolom antara queen pada baris ke-i dan baris ke-$i+1$ adalah 0 atau 1, konfigurasi dinyatakan tidak valid. Selanjutnya, metode `checkRegion()` memeriksa apakah terdapat lebih dari satu *queen* yang ditempatkan pada region yang sama dengan menggunakan struktur data *HashSet* untuk mencatat region yang telah digunakan. Jika suatu region telah digunakan sebelumnya, konfigurasi dinyatakan tidak valid. Tahap ini merupakan tahap pengujian kandidat solusi dalam algoritma brute force.
5. Jika konfigurasi queen yang sedang diperiksa tidak memenuhi salah satu *constraint*, algoritma akan mengabaikan konfigurasi tersebut dan melanjutkan ke permutasi berikutnya menggunakan metode `nextPermutation()`. Sebaliknya, jika konfigurasi memenuhi seluruh constraint, konfigurasi tersebut dinyatakan sebagai solusi yang valid dan proses pencarian dihentikan. Seluruh proses pencarian dilakukan tanpa menggunakan teknik optimasi, seperti *pruning*,

heuristik, atau *backtracking*. Pembatasan satu *queen* pada setiap baris dan kolom diperoleh secara langsung dari representasi solusi dalam bentuk permutasi, bukan melalui eliminasi kandidat solusi selama proses pencarian.

6. Selama proses pencarian berlangsung, algoritma mencatat jumlah kandidat solusi yang telah diperiksa menggunakan variabel `iterationCount`. Setiap iterasi merepresentasikan satu permutasi yang diuji. Selain itu, algoritma juga memberikan pembaruan secara berkala kepada antarmuka pengguna menggunakan mekanisme listener dan SwingWorker sehingga pengguna dapat melihat perkembangan proses pencarian secara *real-time*. Pembaruan tampilan GUI dilakukan secara periodik setiap 250 milidetik untuk menampilkan konfigurasi *queen* terbaru, jumlah iterasi yang telah diperiksa, serta waktu eksekusi sementara. Interval ini dipilih agar tampilan tetap responsif tanpa mengganggu kinerja utama algoritma brute force. Pencatatan iterasi ini juga menunjukkan secara nyata besarnya ruang solusi yang harus diperiksa dalam brute force.
7. Setelah solusi ditemukan, algoritma menampilkan solusi tersebut pada antarmuka pengguna menggunakan metode `displayBoard()` dalam kelas `GUI`. Metode ini menggambar papan berukuran N × N menggunakan komponen `JPanel` dan `JLabel`, yaitu setiap sel diwarnai berdasarkan region-nya dan posisi *queen* ditampilkan menggunakan simbol ♛. Selain itu, program juga menampilkan jumlah iterasi dan waktu eksekusi pada status label. Tahap ini merupakan tahap presentasi solusi, yaitu menampilkan hasil solusi kepada pengguna dalam bentuk visual yang mudah dipahami.
8. Selain ditampilkan pada layar, program juga menyediakan fitur untuk menyimpan solusi menggunakan metode `saveSolution()`. Pengguna dapat memilih untuk menyimpan solusi dalam format TXT atau PNG. Pada format TXT, solusi disimpan dalam bentuk teks, yaitu simbol '#' yang merepresentasikan posisi *queen* dan huruf lainnya merepresentasikan region. Pada format PNG, solusi disimpan sebagai gambar menggunakan kelas `BufferedImage` dan `Graphics2D`, yaitu papan yang digambar lengkap dengan warna region dan simbol *queen*. Tahap ini merupakan tahap akhir dari proses penyelesaian masalah, yaitu menyimpan solusi yang telah ditemukan agar dapat digunakan kembali atau dianalisis lebih lanjut.
9. Proses brute force berakhir ketika solusi ditemukan atau ketika semua permutasi telah diperiksa. Jika solusi ditemukan, algoritma mengembalikan solusi tersebut dan menghentikan pencarian. Jika semua permutasi telah diperiksa dan tidak ditemukan solusi, algoritma menyimpulkan bahwa tidak ada solusi yang memenuhi *constraint*. Pendekatan ini menjamin kebenaran hasil karena semua kemungkinan solusi telah diperiksa.
10. Kompleksitas waktu algoritma ini didominasi oleh jumlah permutasi yang harus diperiksa, yaitu sebanyak $N!$. Untuk setiap permutasi, algoritma melakukan pemeriksaan validitas

menggunakan metode `checkAdjacent()` dan `checkRegion()` yang masing-masing membutuhkan waktu O(N) sehingga kompleksitas waktu dalam kasus terburuk adalah O($N \times N!$). Pada kasus terbaik, algoritma dapat berhenti lebih awal jika solusi ditemukan sebelum seluruh permutasi diperiksa. Kompleksitas yang sangat besar ini merupakan karakteristik khas dari algoritma brute force.

BAB II
IMPLEMENTASI ALGORITMA DALAM BAHASA JAVA

2.1. File GUI.java

File GUI.java berisi kelas GUI yang bertanggung jawab atas pembuatan dan pengelolaan antarmuka grafis program menggunakan Java Swing. Kelas ini berfungsi sebagai penghubung antara pengguna dan proses penyelesaian algoritma. Proses pencarian solusi dijalankan menggunakan SwingWorker pada background *thread* agar GUI tetap responsif dan tidak mengalami *freeze* selama proses brute force berlangsung.

Nama Method	Deskripsi
`GUI()`	*Method* konstruktor untuk membuat dan menampilkan antarmuka grafis. GUI terdiri dari beberapa komponen utama, yaitu tombol kontrol di bagian atas, papan permainan di bagian tengah, dan label status di bagian bawah.
`loadAndSolve()`	*Method* yang dijalankan ketika tombol *Load & Solve* ditekan. Method ini membuka dialog pemilihan file, membaca file papan permainan, dan menjalankan algoritma brute force. Proses pembaruan GUI dilakukan secara periodik setiap 250 milidetik menggunakan mekanisme `publish()` dan `process()` pada SwingWorker.
`displayBoard(Board board, int[] queens)`	*Method* untuk memperbarui tampilan papan permainan berdasarkan konfigurasi *queen* saat ini atau solusi akhir. *Queen* ditampilkan menggunakan simbol ♛.
`generateRegionColors(Board board)`	*Method* untuk menghasilkan warna unik pada setiap region papan agar visualisasi solusi lebih mudah dipahami.
`saveSolution()`	*Method* untuk menampilkan dialog pemilihan format penyimpanan solusi (TXT atau PNG).

`saveAsTXT(File file)`	*Method* untuk menyimpan solusi dalam format teks (.txt).
saveAsPNG(File file)	Method untuk menyimpan solusi dalam format gambar (.png) dengan tampilan *grid* dan simbol *queen*.

GUI menampilkan peringatan apabila ukuran papan terlalu besar karena dapat menyebabkan waktu komputasi yang sangat lama akibat kompleksitas algoritma brute force.

2.2. File Board.java

File Board.java berisi kelas Board yang bertanggung jawab untuk membaca dan memvalidasi *file input* serta menyimpan informasi papan permainan dan region.

Nama Method	Deskripsi
`Board(String filePath)`	Konstruktor yang menerima path *file input* dan memanggil *method* `readFromFile`.
`readFromFile(String filePath)`	*Method* untuk membaca *file input*, memvalidasi format *file*, ukuran papan, konsistensi panjang baris, karakter region (A–Z), serta jumlah region sebelum papan digunakan oleh algoritma.
`getSize()`	*Method* untuk mengembalikan ukuran papan permainan (N).
`getRegion(int row, int col)`	*Method* untuk mengembalikan karakter region pada posisi tertentu di papan.
`getRegions()`	*Method* untuk mengembalikan seluruh matriks region papan.

Validasi *input* meliputi pemeriksaan bahwa *file* tidak kosong, tidak mengandung baris kosong, papan berbentuk persegi ($N \times N$), panjang setiap baris konsisten, setiap sel hanya berisi karakter huruf A–Z, jumlah region tidak melebihi 26, dan jumlah region unik sama dengan ukuran papan (N).

2.3. File PermutationGenerator.java

File PermutationGenerator.java berisi kelas PermutationGenerator yang digunakan untuk menghasilkan permutasi leksikografis dari array posisi queen. Kelas ini merupakan komponen utama dalam implementasi brute force karena digunakan untuk membangkitkan seluruh kandidat solusi secara sistematis melalui permutasi sebagai bagian dari exhaustive search.

Nama Method	Deskripsi
`nextPermutation(int[] arr)`	*Method* untuk menghasilkan permutasi berikutnya secara leksikografis. Mengembalikan *false* jika seluruh permutasi telah dievaluasi.
`swap(int[] arr, int i, int j)`	*Method* pembantu untuk menukar dua elemen pada array.
`reverse(int[] arr, int start, int end)`	*Method* pembantu untuk membalik urutan elemen array dari indeks tertentu.

Kelas ini digunakan oleh QueensSolver untuk mengevaluasi seluruh kemungkinan penempatan queen.

2.4. File QueensSolver.java

File QueensSolver.java berisi kelas QueensSolver yang mengimplementasikan algoritma brute force untuk menyelesaikan permasalahan Queens Linkedin.

Nama Method	Deskripsi
`QueensSolver(Board board)`	Konstruktor yang menginisialisasi papan permainan, ukuran papan, array queen, serta variabel iterasi.
`solve(UpdateListener listener)`	Method utama untuk menjalankan algoritma brute force dengan mengevaluasi seluruh permutasi posisi queen dan melakukan *live update* ke GUI.
isValid()	Method untuk memeriksa apakah konfigurasi *queen*

	valid berdasarkan aturan *adjacency* dan region. *Method* ini memanggil `checkAdjacent()` untuk memastikan *queen* tidak saling menyerang pada posisi berdekatan serta `checkRegion()` untuk memastikan setiap *queen* berada pada region yang berbeda.
`checkAdjacent()`	*Method* untuk memastikan tidak ada *queen* yang saling menyerang pada posisi berdekatan (*adjacent*). *Method* ini memeriksa selisih posisi kolom antara *queen* pada baris yang berdekatan untuk memastikan *queen* tidak saling menyerang atau bersentuhan secara *adjacency* sesuai aturan permainan.
`checkRegion()`	*Method* untuk memastikan setiap *queen* berada pada region yang berbeda.
`getIterationCount()`	*Method* untuk mengembalikan jumlah iterasi yang telah dilakukan selama proses pencarian solusi.
`getQueens()`	*Method* untuk mengembalikan konfigurasi *queen* terakhir atau solusi yang ditemukan.
`isFoundSolution()`	*Method* untuk mengembalikan status apakah solusi berhasil ditemukan.

2.5. File Main.java

File Main.java berisi kelas Main yang berfungsi sebagai titik masuk (*entry point*) program.

Nama Method	Deskripsi
`main(String[] args)`	Method utama untuk menjalankan program dengan menampilkan antarmuka grafis (GUI).

BAB III

SOURCE CODE PROGRAM

3.1. Repositori GitHub

Repositori program dapat diakses melalui tautan GitHub berikut: https://github.com/kalycanbnctaa/Tucil1_13524071.

3.2. *Source Code* Program

3.2.1. GUI.java

```java
import javax.swing.*;
import javax.swing.filechooser.FileNameExtensionFilter;
import java.awt.*;
import java.awt.image.BufferedImage;
import java.io.*;
import java.util.HashMap;
import java.util.Map;
import javax.imageio.ImageIO;

public class GUI {

    private JFrame frame;
    private JLabel statusLabel;
    private JPanel boardPanel;
    private JButton loadButton;
    private JButton saveButton;

    private Board currentBoard;
    private int[] currentSolution;

    public GUI() {
        frame = new JFrame(title: "Queens Solver - Brute Force");
        frame.setSize(width: 800, height: 800);
        frame.setLocationRelativeTo(c: null);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLayout(new BorderLayout());

        loadButton = new JButton(text: "Load & Solve");
        saveButton = new JButton(text: "Save Solution");
        saveButton.setEnabled(b: false);

        JPanel topPanel = new JPanel();
        topPanel.add(loadButton);
        topPanel.add(saveButton);

        statusLabel = new JLabel(text: "Load a board file (.txt).", SwingConstants.CENTER);
        boardPanel = new JPanel();
```

```
        loadButton.addActionListener(e -> loadAndSolve());
        saveButton.addActionListener(e -> saveSolution());

        frame.add(topPanel, BorderLayout.NORTH);
        frame.add(boardPanel, BorderLayout.CENTER);
        frame.add(statusLabel, BorderLayout.SOUTH);

        frame.setVisible(b: true);
    }

    private void loadAndSolve() {

        showInfo(message: "Silakan pilih file input dengan format .txt");

        JFileChooser chooser = new JFileChooser();
        chooser.setFileFilter(new FileNameExtensionFilter(
                description: "Text Files (*.txt)", ...extensions: "txt"
        ));

        if (chooser.showOpenDialog(frame) == JFileChooser.APPROVE_OPTION) {

            File selectedFile = chooser.getSelectedFile();
            String fileName = selectedFile.getName().toLowerCase();

            if (!fileName.endsWith(suffix: ".txt")) {
                showError(message: "File input harus berformat .txt");
                return;
            }

            try {
                currentBoard = new Board(selectedFile.getAbsolutePath());

                if (currentBoard.getSize() > 10) {
                    showWarning(message: "N besar dapat menyebabkan waktu komputasi sangat lama.");
                }
```

```
                QueensSolver solver = new QueensSolver(currentBoard);

                loadButton.setEnabled(b: false);
                saveButton.setEnabled(b: false);
                statusLabel.setText(text: "Solving...");

                SwingWorker<Boolean, int[]> worker = new SwingWorker<>() {

                    long startTime;

                    @Override
                    protected Boolean doInBackground() {
                        startTime = System.currentTimeMillis();
                        return solver.solve((queensSnapshot, iteration) -> publish(queensSnapshot));
                    }

                    @Override
                    protected void process(java.util.List<int[]> chunks) {
                        int[] latest = chunks.get(chunks.size() - 1);
                        displayBoard(currentBoard, latest);
                        statusLabel.setText("Iterations: " + solver.getIterationCount());
                    }

                    @Override
                    protected void done() {
                        long endTime = System.currentTimeMillis();
                        loadButton.setEnabled(b: true);

                        try {
                            boolean solved = get();
                            if (solved) {
                                currentSolution = solver.getQueens().clone();
                                displayBoard(currentBoard, currentSolution);
                                statusLabel.setText(
                                        "Solved! Iterations: " + solver.getIterationCount()
```

```
                                    "Solved! Iterations: " + solver.getIterationCount()
                                            + " | Time: " + (endTime - startTime) + " ms"
                            );
                            saveButton.setEnabled(b: true);
                        } else {
                            showWarning(message: "Tidak ditemukan solusi.");
                            statusLabel.setText(
                                    "No solution found. Iterations: " + solver.getIterationCount()
                                            + " | Time: " + (endTime - startTime) + " ms"
                            );
                        }
                    } catch (Exception e) {
                        showError(message: "Terjadi error saat proses solving.");
                    }
                }
            };

            worker.execute();

        } catch (IOException ex) {
            currentBoard = null;
            currentSolution = null;

            boardPanel.removeAll();
            boardPanel.revalidate();
            boardPanel.repaint();

            statusLabel.setText(text: "Input tidak valid.");

            saveButton.setEnabled(b: false);

            showError("Error membaca file:\n" + ex.getMessage());
        }
```

```
        }
    }

    private void displayBoard(Board board, int[] queens) {
        boardPanel.removeAll();

        int size = board.getSize();
        boardPanel.setLayout(new GridLayout(size, size));
        Map<Character, Color> colorMap = generateRegionColors(board);

        for (int i = 0; i < size; i++) {
            for (int j = 0; j < size; j++) {

                JLabel cell = new JLabel(text: "", SwingConstants.CENTER);
                cell.setOpaque(isOpaque: true);
                cell.setBorder(BorderFactory.createLineBorder(Color.BLACK));
                cell.setBackground(colorMap.get(board.getRegion(i, j)));

                if (queens[i] == j) {
                    cell.setText(text: "♛");
                    cell.setFont(new Font(name: "Serif", Font.BOLD, size > 8 ? 24 : 36));
                }

                boardPanel.add(cell);
            }
        }

        boardPanel.revalidate();
        boardPanel.repaint();
    }

    private Map<Character, Color> generateRegionColors(Board board) {
        Map<Character, Color> map = new HashMap<>();
        char[][] regions = board.getRegions();

```

```
        final float GOLDEN_RATIO = 0.61803398875f;

        for (char[] row : regions) {
            for (char c : row) {
                if (!map.containsKey(c)) {
                    int idx = c - 'A';
                    float hue = (idx * GOLDEN_RATIO) % 1f;
                    Color color = Color.getHSBColor(hue, s: 0.6f, b: 0.95f);
                    map.put(c, color);
                }
            }
        }
        return map;
    }


    private void saveSolution() {

        if (currentSolution == null || currentBoard == null) {
            showError(message: "No solution available to save!");
            return;
        }

        String[] options = {"Save as TXT", "Save as PNG"};
        int choice = JOptionPane.showOptionDialog(
                frame,
                message: "Pilih format penyimpanan:",
                title: "Save Solution",
                JOptionPane.DEFAULT_OPTION,
                JOptionPane.QUESTION_MESSAGE,
                icon: null,
                options,
                options[0]
        );
```

```
        if (choice == -1) return;

        JFileChooser chooser = new JFileChooser();
        if (chooser.showSaveDialog(frame) == JFileChooser.APPROVE_OPTION) {
            try {
                File file = chooser.getSelectedFile();
                if (choice == 0) {
                    saveAsTXT(new File(file.getAbsolutePath() + ".txt"));
                } else {
                    saveAsPNG(new File(file.getAbsolutePath() + ".png"));
                }
                showInfo(message: "Solution saved successfully!");
            } catch (IOException e) {
                showError(message: "Error saving file.");
            }
        }
    }

    private void saveAsTXT(File file) throws IOException {
        try (PrintWriter pw = new PrintWriter(file)) {
            int size = currentBoard.getSize();

            for (int i = 0; i < size; i++) {
                for (int j = 0; j < size; j++) {

                    if (currentSolution[i] == j) {
                        pw.print(s: "#");
                    } else {
                        pw.print(currentBoard.getRegion(i, j));
                    }
                }
                pw.println();
            }
        }
    }
```

```
    private void saveAsPNG(File file) throws IOException {
        int cell = 60;
        int size = currentBoard.getSize();
        BufferedImage img = new BufferedImage(
                size * cell, size * cell, BufferedImage.TYPE_INT_ARGB
        );

        Graphics2D g = img.createGraphics();
        Map<Character, Color> colors = generateRegionColors(currentBoard);

        for (int i = 0; i < size; i++) {
            for (int j = 0; j < size; j++) {
                g.setColor(colors.get(currentBoard.getRegion(i, j)));
                g.fillRect(j * cell, i * cell, cell, cell);
                g.setColor(Color.BLACK);
                g.drawRect(j * cell, i * cell, cell, cell);

                if (currentSolution[i] == j) {
                    g.setFont(new Font(name: "Serif", Font.BOLD, size: 40));
                    g.drawString(str: "♛", j * cell + 15, i * cell + 45);
                }
            }
        }

        g.dispose();
        ImageIO.write(img, formatName: "png", file);
    }

    private void showError(String message) {
        JOptionPane.showMessageDialog(
                frame,
                message,
                title: "Error",
                JOptionPane.ERROR_MESSAGE
        );
    }
```

```

    private void showWarning(String message) {
        JOptionPane.showMessageDialog(
                frame,
                message,
                title: "Warning",
                JOptionPane.WARNING_MESSAGE
        );
    }

    private void showInfo(String message) {
        JOptionPane.showMessageDialog(
                frame,
                message,
                title: "Information",
                JOptionPane.INFORMATION_MESSAGE
        );
    }
}
```

3.2.2. Board.java

src > Board.java > Board > readFromFile(String)

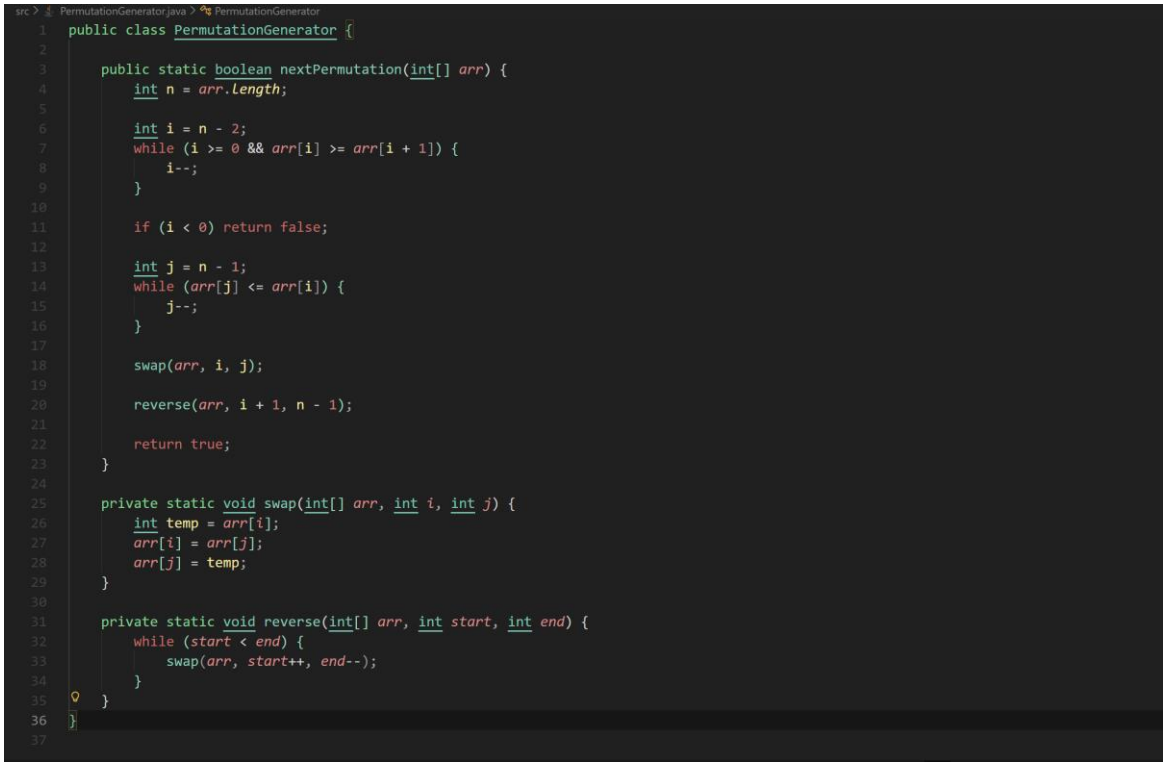
```java
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.HashSet;
import java.util.Set;
import java.util.ArrayList;
import java.util.List;

public class Board {

    private int size;
    private char[][] regions;

    public Board(String filePath) throws IOException {
        readFromFile(filePath);
    }

    private void readFromFile(String filePath) throws IOException {

        try (BufferedReader br = new BufferedReader(new FileReader(filePath))) {

            List<String> lines = new ArrayList<>();
            String line;
            int lineNumber = 0;

            // Baca semua baris
            while ((line = br.readLine()) != null) {
                lineNumber++;

                if (line.trim().isEmpty()) {
                    throw new IOException(
                        "Terdapat baris kosong pada baris " + lineNumber
                    );
                }

                lines.add(line.trim());
            }
```

```java
            if (lines.isEmpty()) {
                throw new IOException(message: "File kosong.");
            }

            size = lines.size();
            int rowLength = lines.get(index: 0).length();

            if (rowLength == 0) {
                throw new IOException(message: "Baris tidak boleh kosong.");
            }

            if (rowLength != size) {
                throw new IOException(
                    "Board harus berbentuk persegi (NxN). " +
                    "Ditemukan " + size + " baris dan panjang " + rowLength
                );
            }

            regions = new char[size][size];

            // Isi regions dan validasi karakter
            for (int i = 0; i < size; i++) {

                String currentLine = lines.get(i);

                if (currentLine.length() != rowLength) {
                    throw new IOException(
                        "Panjang baris tidak konsisten pada baris " + (i + 1)
                    );
                }

                for (int j = 0; j < size; j++) {

                    char c = currentLine.charAt(j);
```

```
                if (c < 'A' || c > 'Z') {
                    throw new IOException(
                        "Region harus berupa huruf A-Z. Ditemukan '" + c +
                        "' pada baris " + (i + 1) +
                        ", kolom " + (j + 1)
                    );
                }

                regions[i][j] = c;
            }
        }

        // Hitung jumlah region unik
        Set<Character> uniqueRegions = new HashSet<>();

        for (int i = 0; i < size; i++) {
            for (int j = 0; j < size; j++) {
                uniqueRegions.add(regions[i][j]);
            }
        }

        if (uniqueRegions.size() > 26) {
            throw new IOException(
                "Jumlah region melebihi 26 (" +
                uniqueRegions.size() +
                "). Maksimal 26 region yang diperbolehkan."
            );
        }

        if (uniqueRegions.size() != size) {
            throw new IOException(
                "Jumlah region harus sama dengan ukuran board (" +
                size + "), tetapi ditemukan " +
                uniqueRegions.size() + " region unik."
            );
```

```
            }
        }
    }

    // GETTER

    public int getSize() {
        return size;
    }

    public char getRegion(int row, int col) {
        return regions[row][col];
    }

    public char[][] getRegions() {
        return regions;
    }
}
```

3.2.3. PermutationGenerator.java



```java
public class PermutationGenerator {

    public static boolean nextPermutation(int[] arr) {
        int n = arr.length;

        int i = n - 2;
        while (i >= 0 && arr[i] >= arr[i + 1]) {
            i--;
        }

        if (i < 0) return false;

        int j = n - 1;
        while (arr[j] <= arr[i]) {
            j--;
        }

        swap(arr, i, j);

        reverse(arr, i + 1, n - 1);

        return true;
    }

    private static void swap(int[] arr, int i, int j) {
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }

    private static void reverse(int[] arr, int start, int end) {
        while (start < end) {
            swap(arr, start++, end--);
        }
    }
}
```

3.2.4. QueensSolver.java

```java
import java.util.Set;
import java.util.HashSet;

public class QueensSolver {

    public interface UpdateListener {
        void onUpdate(int[] queensSnapshot, long iteration);
    }

    private Board board;
    private int size;
    private int[] queens;
    private long iterationCount;
    private boolean foundSolution;

    private long lastUpdateTime = 0;
    private static final long UPDATE_INTERVAL = 250;

    public QueensSolver(Board board) {
        this.board = board;
        this.size = board.getSize();
        this.queens = new int[size];

        for (int i = 0; i < size; i++) {
            queens[i] = i;
        }

        this.iterationCount = 0;
        this.foundSolution = false;
    }

    public boolean solve(UpdateListener listener) {

        do {
            iterationCount++;

            // Live update berdasarkan waktu
            long now = System.currentTimeMillis();
            if (listener != null && now - lastUpdateTime >= UPDATE_INTERVAL) {
                listener.onUpdate(queens.clone(), iterationCount);
                lastUpdateTime = now;
            }

            if (isValid()) {
                foundSolution = true;

                if (listener != null) {
                    listener.onUpdate(queens.clone(), iterationCount);
                }

                return true;
            }

        } while (PermutationGenerator.nextPermutation(queens));

        return false;
    }

    private boolean isValid() {
        return checkAdjacent() && checkRegion();
    }

    private boolean checkAdjacent() {
    for (int i = 0; i < size - 1; i++) {
        int colDiff = Math.abs(queens[i] - queens[i + 1]);
        if (colDiff <= 1) {
            return false;
        }
    }
    return true;
}
```

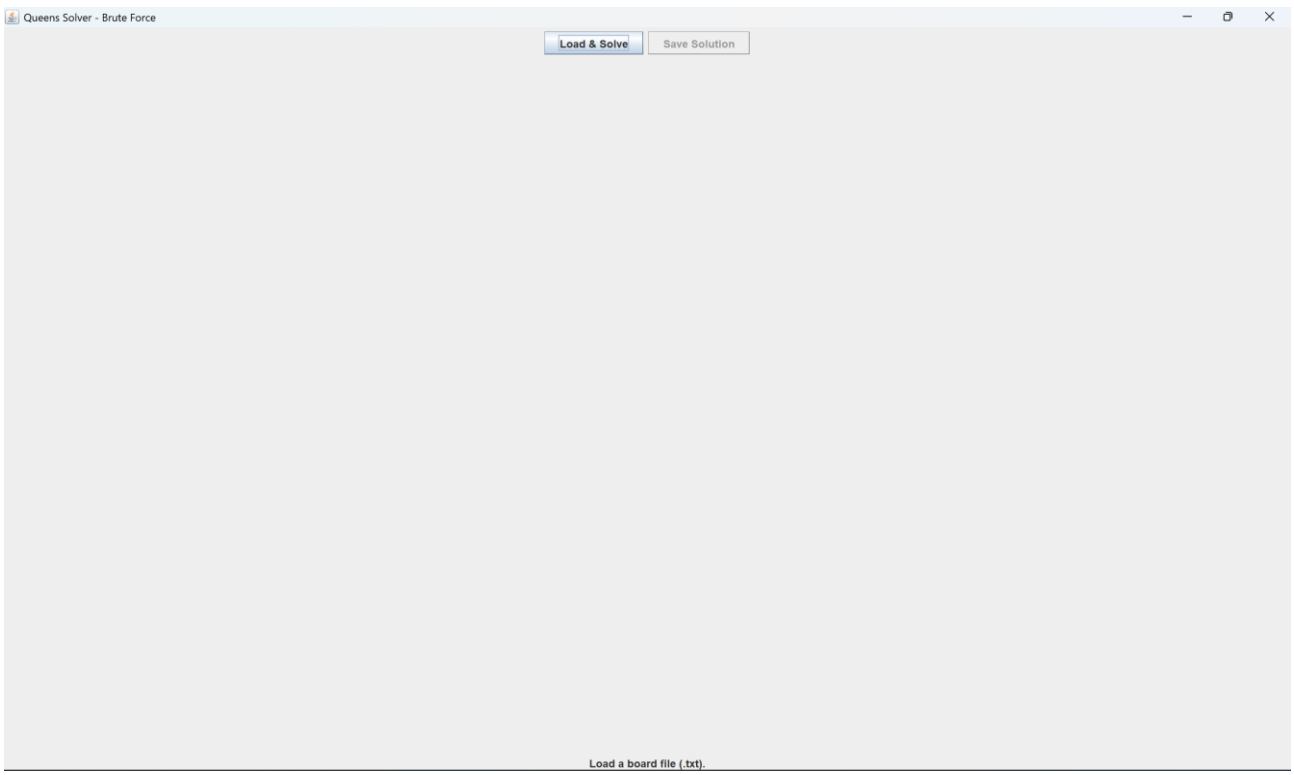
3.2.5. Main.java

```
72
73      private boolean checkRegion() {
74          Set<Character> usedRegions = new HashSet<>();
75
76          for (int i = 0; i < size; i++) {
77              char region = board.getRegion(i, queens[i]);
78              if (usedRegions.contains(region)) {
79                  return false;
80              }
81              usedRegions.add(region);
82          }
83
84          return true;
85      }
86
87      public long getIterationCount() {
88          return iterationCount;
89      }
90
91      public int[] getQueens() {
92          return queens;
93      }
94
95      public boolean isFoundSolution() {
96          return foundSolution;
97      }
98  }
```

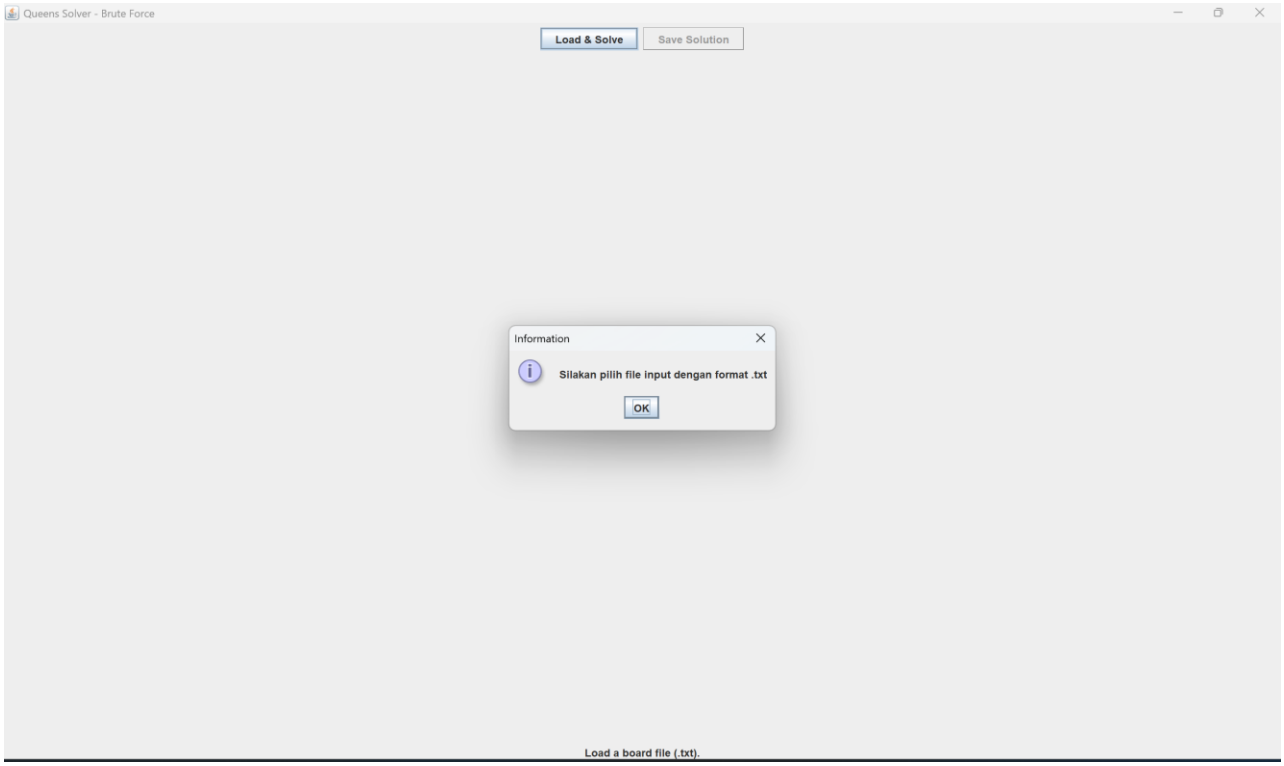
BAB IV

INPUT DAN *OUTPUT* PROGRAM

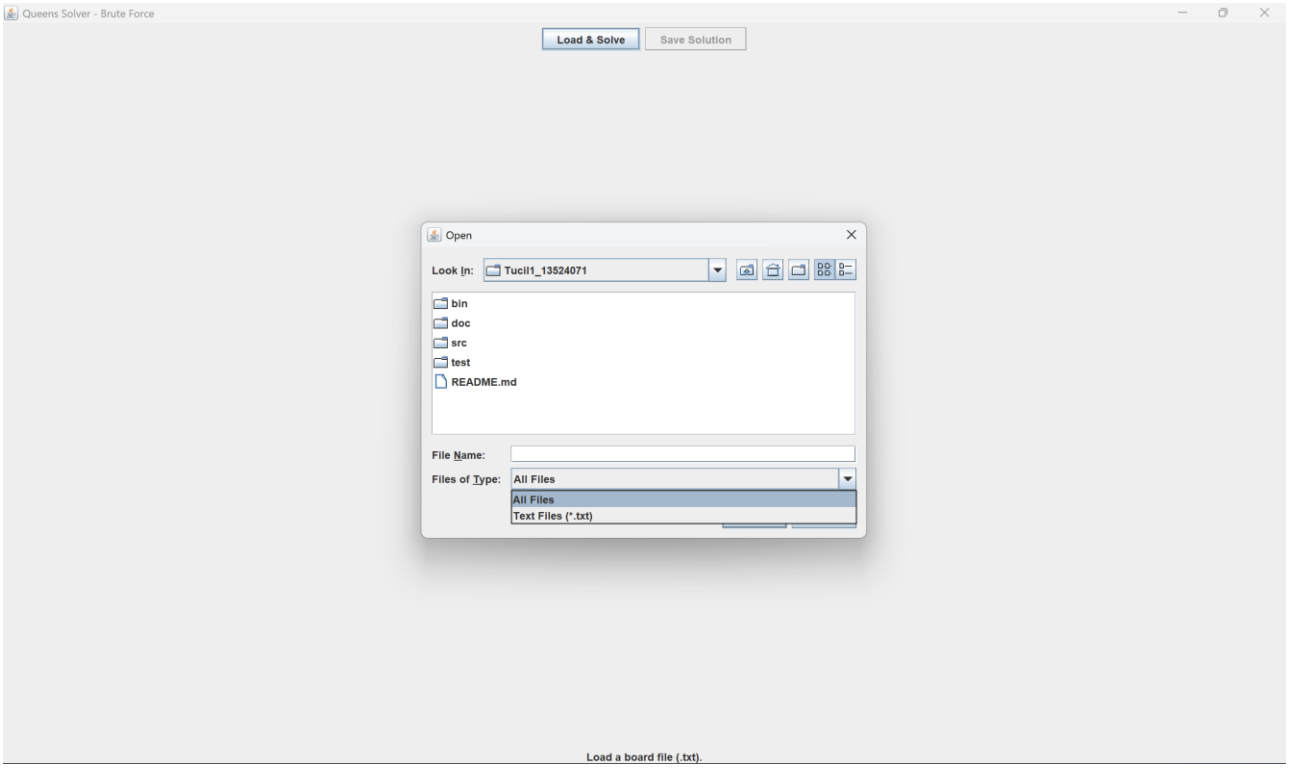
Tampilan awal GUI menampilkan jendela aplikasi Queens Solver – Brute Force dengan antarmuka yang masih kosong karena belum ada *file input* yang dimuat. Pada bagian atas terdapat dua tombol, yaitu tombol Load & Solve yang digunakan untuk memilih file papan dan menjalankan proses pencarian solusi, serta tombol Save Solution yang masih dalam keadaan nonaktif karena belum ada solusi yang tersedia. Bagian tengah berupa area papan (*board* panel) yang kosong dan akan digunakan untuk menampilkan papan dan posisi *queen* setelah *file* dimuat. Pada bagian bawah terdapat status label yang menampilkan pesan "Load a board file (.txt)." sebagai petunjuk kepada pengguna untuk memulai dengan memuat *file input*.



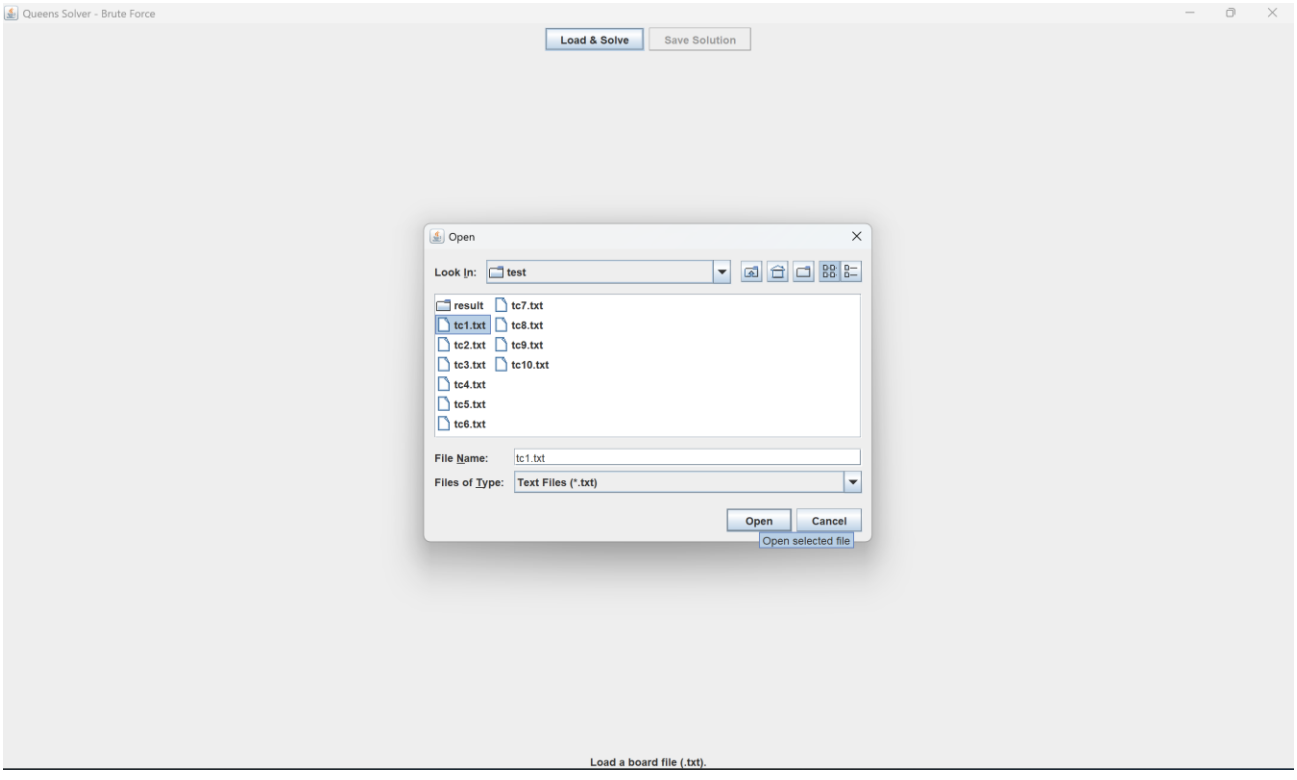
Setelah pengguna menekan tombol Load & Solve, sistem menampilkan jendela dialog informasi yang berisi pesan "Silakan pilih file input dengan format .txt" sebagai petunjuk agar pengguna memilih *file input* yang sesuai dengan format yang telah ditentukan. Pengguna dapat menekan tombol OK untuk melanjutkan ke proses pemilihan *file* melalui *file chooser*.



Setelah pengguna menekan tombol OK, sistem menampilkan jendela *file chooser* yang memungkinkan pengguna memilih *file input* dari direktori komputer. Opsi All Files menampilkan semua *file* yang ada di direktori tanpa memandang ekstensi. Sementara itu, opsi Text Files (*.txt) hanya menampilkan *file* yang memiliki ekstensi .txt sehingga membantu pengguna memilih *file* yang sesuai dengan format *input* yang didukung oleh program. Penggunaan filter ini mengurangi risiko pengguna memilih *file* dengan format yang tidak valid.



Pengguna kemudian menekan tombol Open untuk memuat *file* yang dipilih ke dalam program. Setelah *file* dipilih, program akan membaca isi *file* dan menjalankan algoritma brute force untuk mencari solusi.



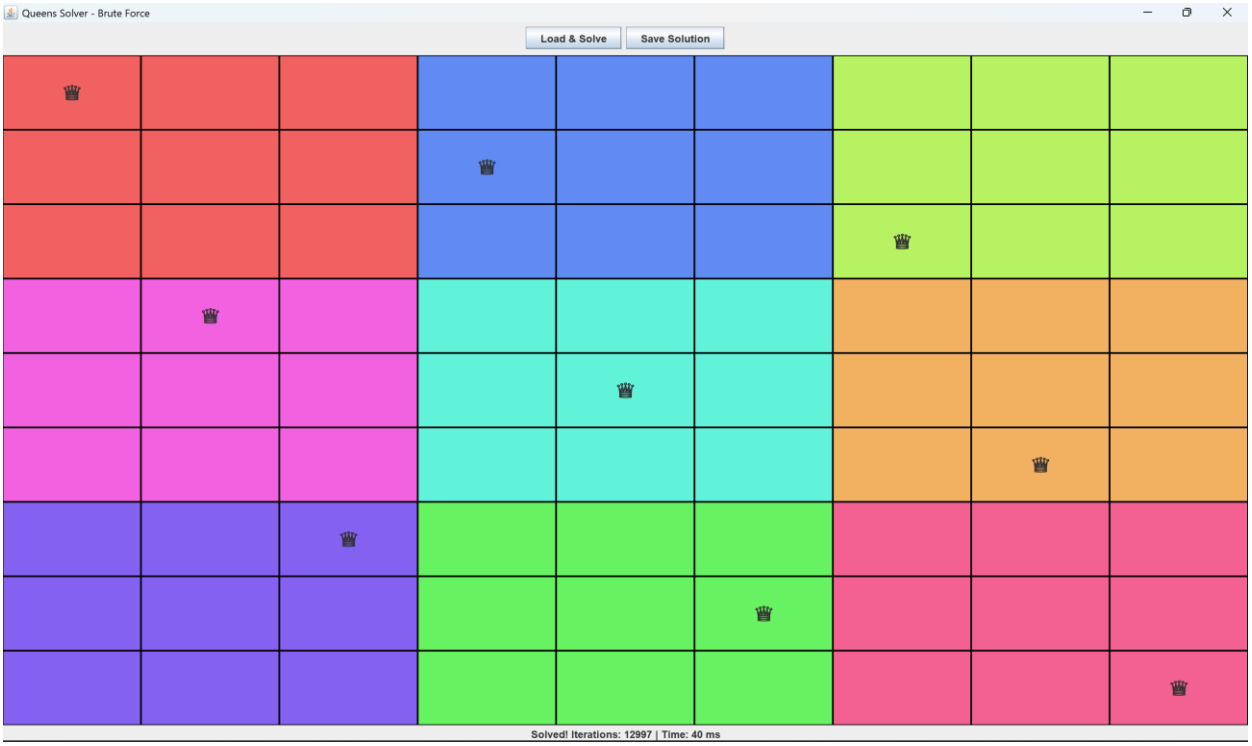
4.1. *Test Case* 1: *Board* Valid, Solusi Ada

tc1.txt

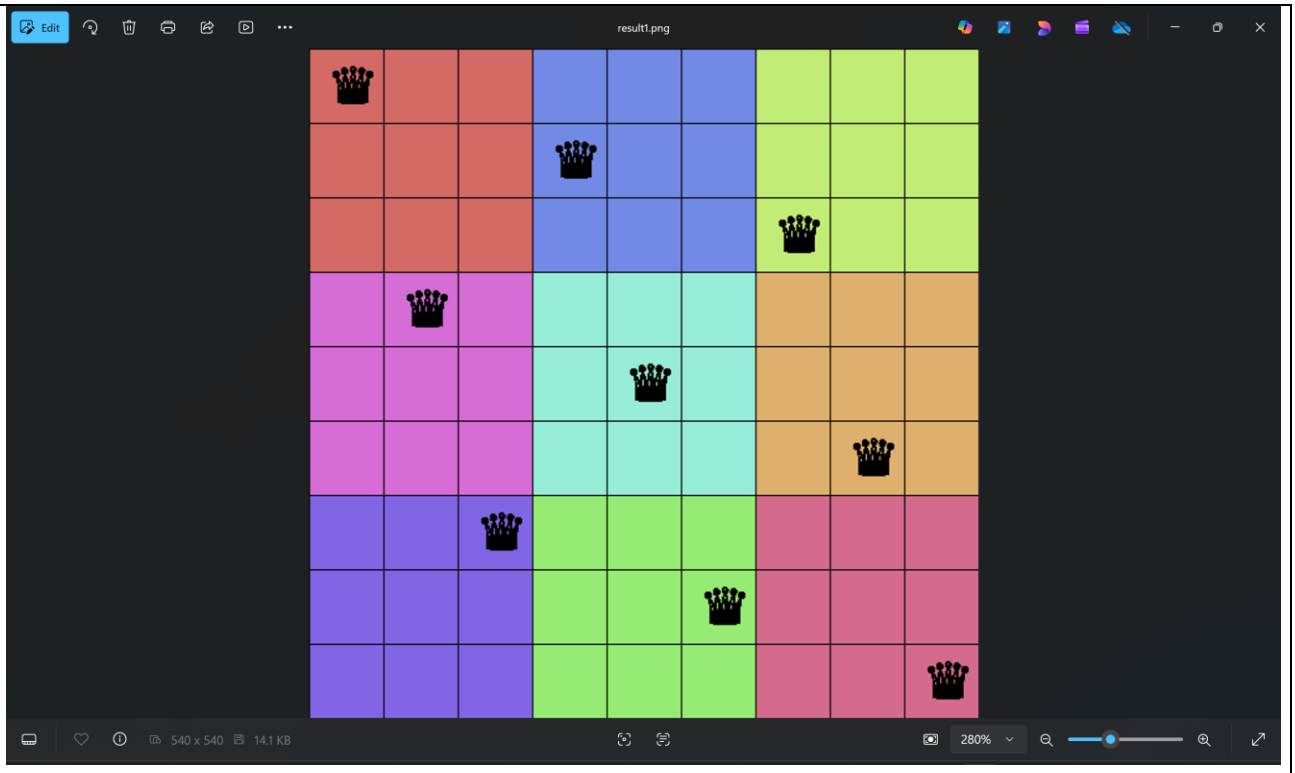
test > tc1.txt

```
AAABBBCCC
AAABBBCCC
AAABBBCCC
DDDEEEFFF
DDDEEEFFF
DDDEEEFFF
GGGHHHIII
GGGHHHIII
GGGHHHIII
```

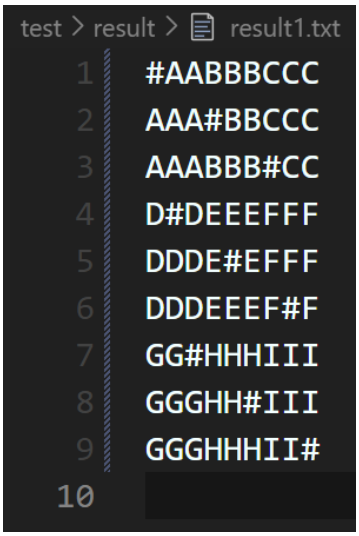
Output penyelesaian tc1.txt pada GUI



Output setelah disimpan sebagai gambar (.png)



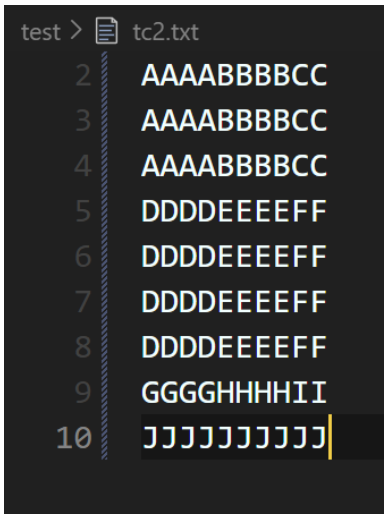
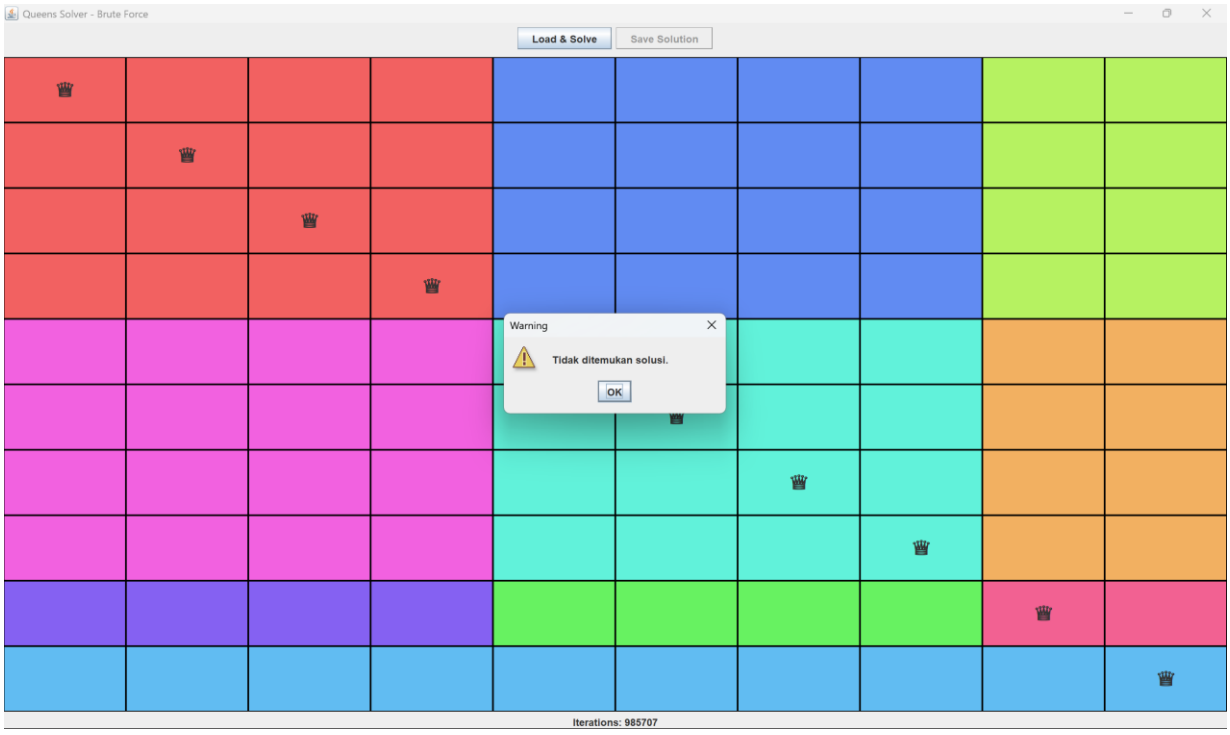
Output setelah disimpan sebagai text (.txt)

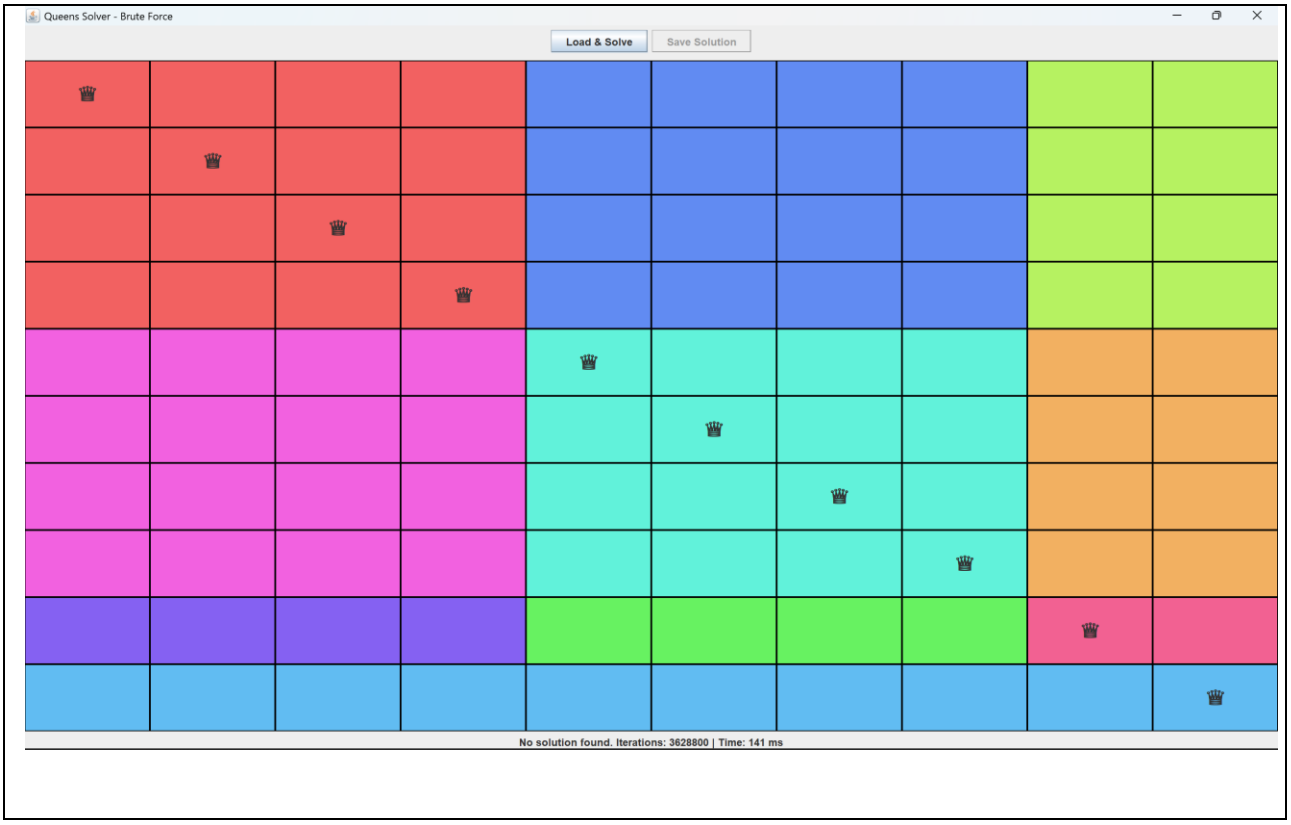


Analisis:

Board berukuran 9×9 dengan tepat 9 region unik (A–I). Semua *constraint* terpenuhi sehingga konfigurasi solusi ada.

4.2. *Test Case* 2: *Board* Valid, Solusi Tidak Ada

tc2.txt

Output penyelesaian tc2.txt pada GUI




Analisis:

Untuk *board* berukuran 10×10, algoritma brute force memeriksa seluruh kemungkinan sebanyak 10! = 3.628.800 permutasi. Program berhasil menyelesaikan pencarian dalam waktu 141 ms dan menyimpulkan bahwa tidak terdapat solusi yang memenuhi seluruh *constraint*. Tombol "Save Solution" hanya diaktifkan ketika solusi berhasil ditemukan. Jika tidak terdapat solusi, tombol tetap dinonaktifkan untuk mencegah penyimpanan data yang tidak valid.

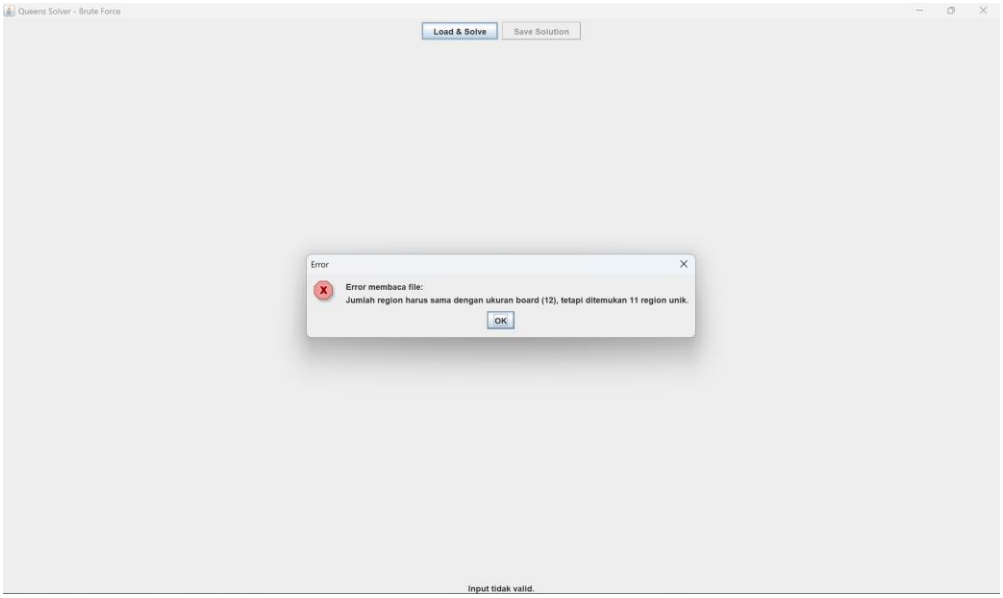
4.3. *Test Case* 3: Invalid karena Jumlah Region < N

tc3.txt

test > tc3.txt

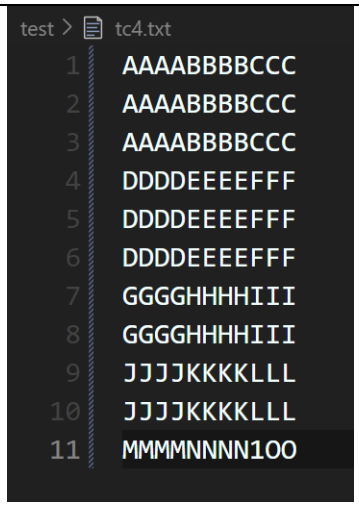
```
1   AAAAAAAAAAAA
2   BBBBBBBBBBBB
3   CCCCCCCCCCCC
4   DDDDDDDDDDDD
5   EEEEEEEEEEEE
6   FFFFFFFFFFFF
7   GGGGGGGGGGGG
8   HHHHHHHHHHHH
9   IIIIIIIIIIII
10  JJJJJJJJJJJJ
11  KKKKKKKKKKKK
12  KKKKKKKKKKKK
```

Output penyelesaian tc3.txt pada GUI

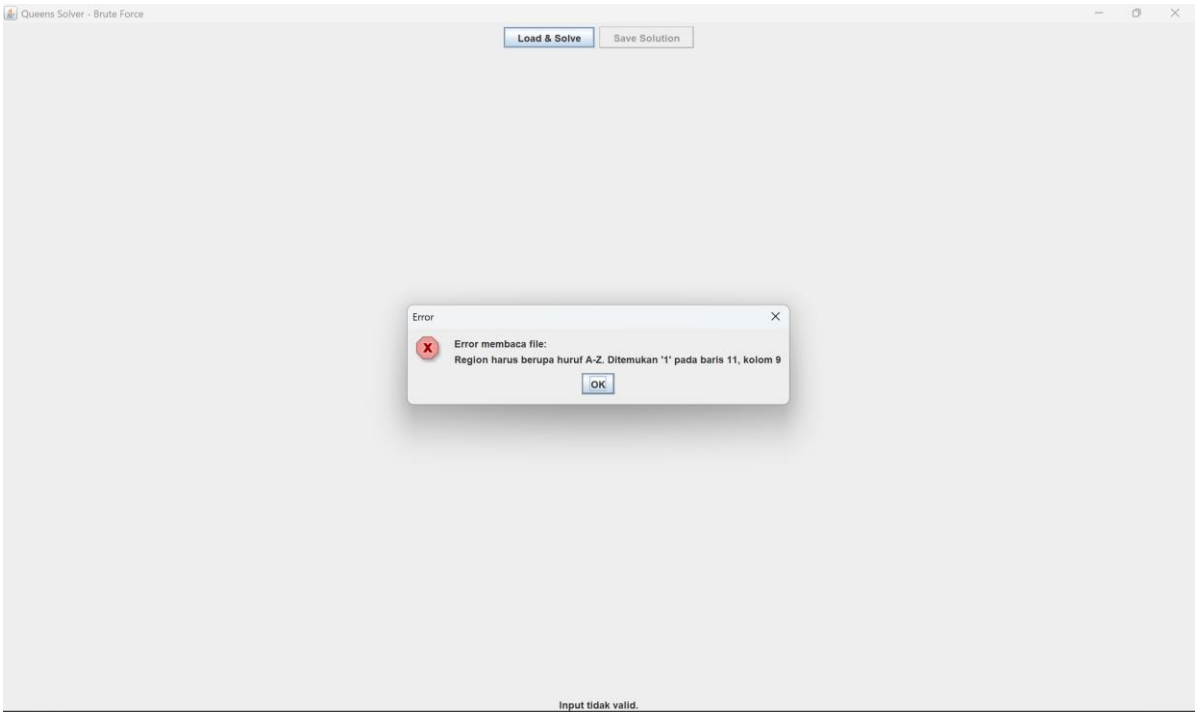


4.4. *Test Case* 4: Invalid karena Karakter Illegal

tc4.txt

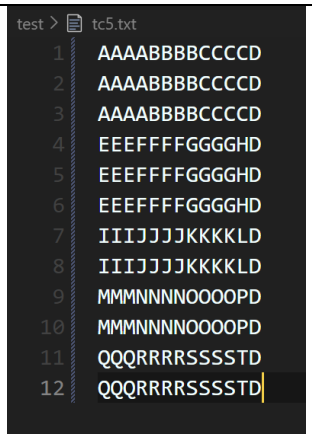


Output penyelesaian tc4.txt pada GUI

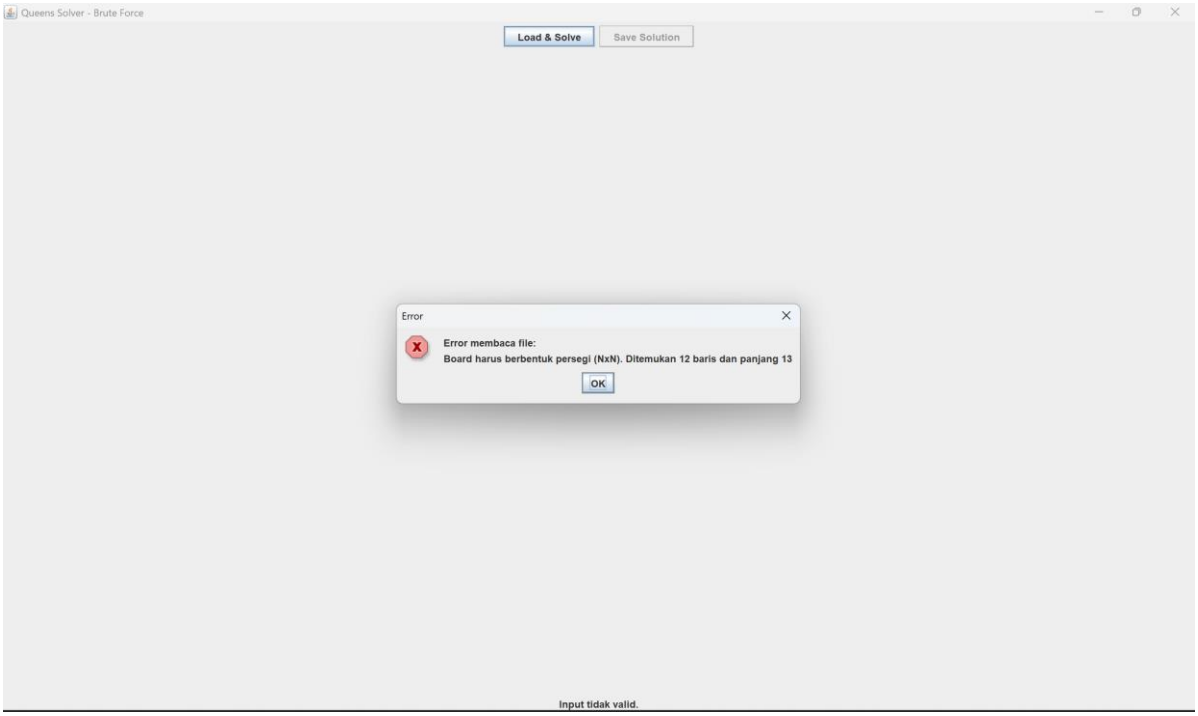


4.5. *Test Case* 5: Invalid karena Ukuran *Board* Bukan Persegi

tc5.txt



Output penyelesaian tc5.txt pada GUI



Program hanya menampilkan kesalahan ukuran board yang tidak berbentuk persegi (NxN) karena proses validasi *input* dilakukan secara berurutan dan eksekusi langsung dihentikan ketika kesalahan pertama ditemukan. Pemeriksaan jumlah region yang harus sama dengan ukuran *board* baru dilakukan setelah validasi bentuk persegi berhasil. Karena *input* sudah gagal pada tahap pengecekan persegi, program tidak melanjutkan ke tahap validasi tersebut.

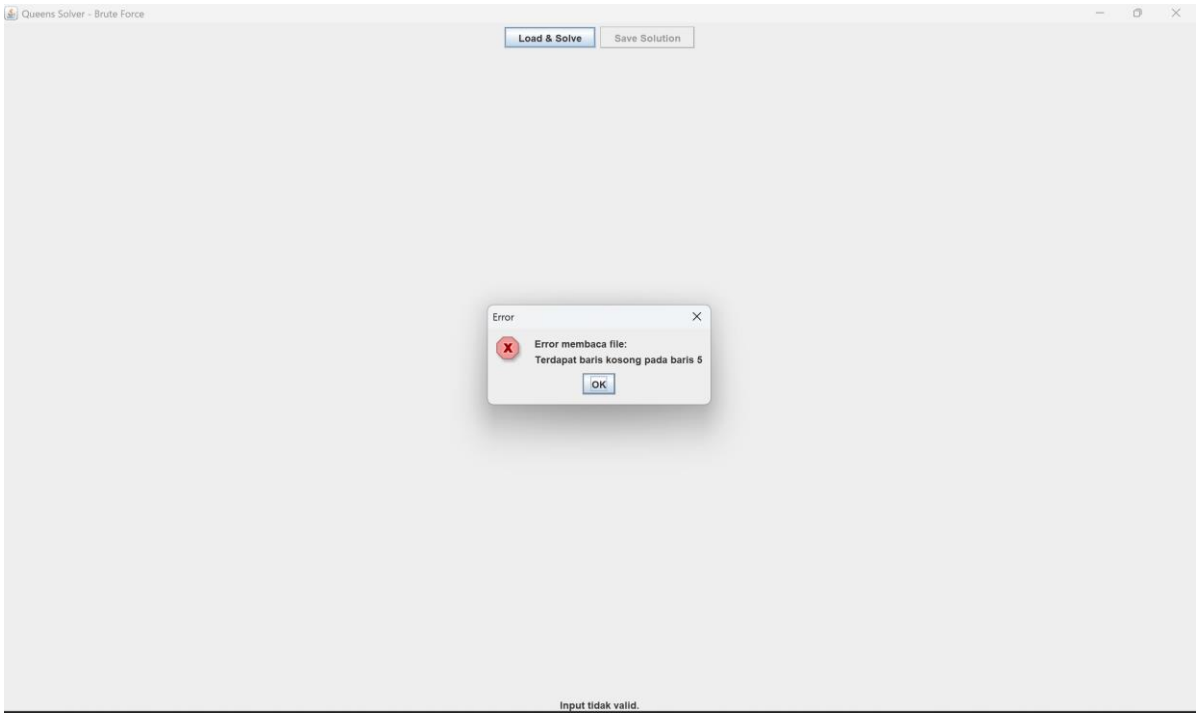
4.6. *Test Case* 6: Invalid karena Terdapat Baris Kosong

tc6.txt

test > tc6.txt

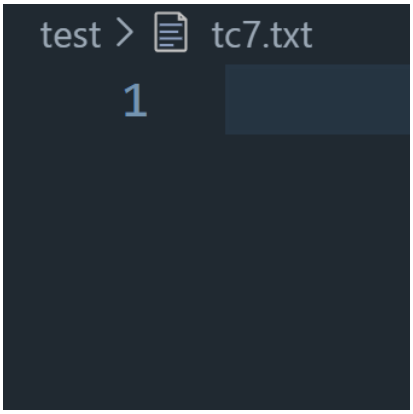
```
AAAABBBBCCCCDD
AAAABBBBCCCCDD
AAAABBBBCCCCDD
AAAABBBBCCCCDD

EEEEFFFFGGGGHH
EEEEFFFFGGGGHH
EEEEFFFFGGGGHH
EEEEFFFFGGGGHH
IIIIJJJJKKKKLL
IIIIJJJJKKKKLL
MMMMNNNNOOOOPP
MMMMNNNNOOOOPP
QQQQRRRRSSSSTT
QQQQRRRRSSSSTT
```

Output penyelesaian tc6.txt pada GUI

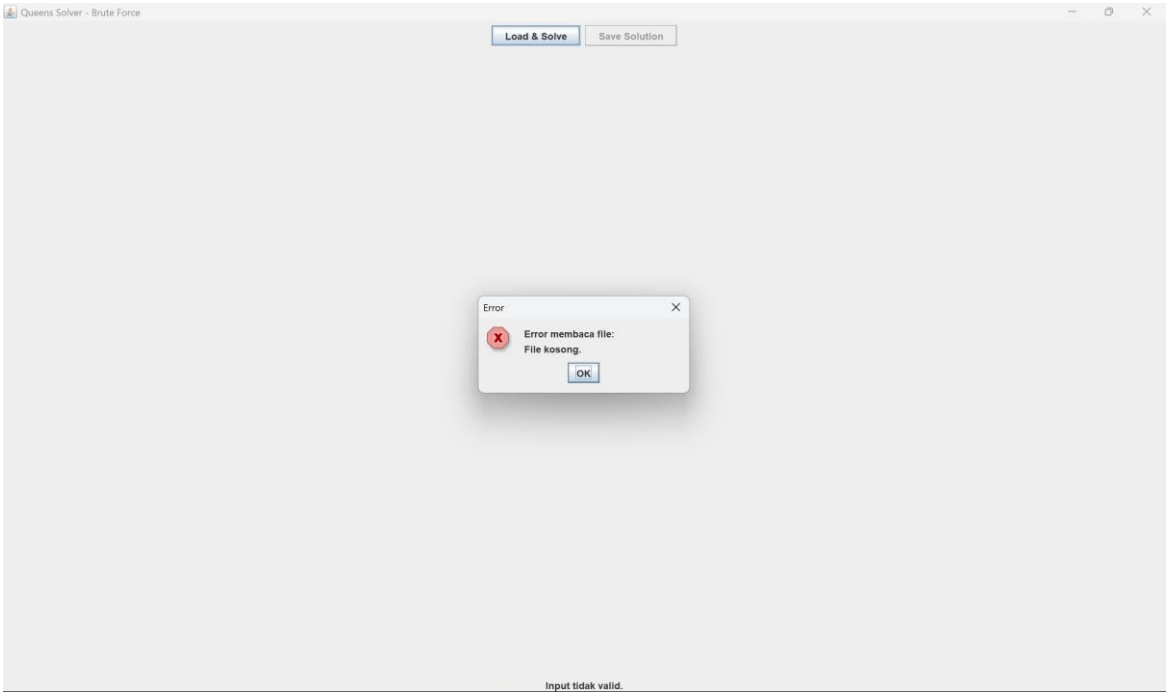


4.7. *Test Case* 7: Invalid karena Isi *File* Kosong

tc7.txt



Output penyelesaian tc7.txt pada GUI



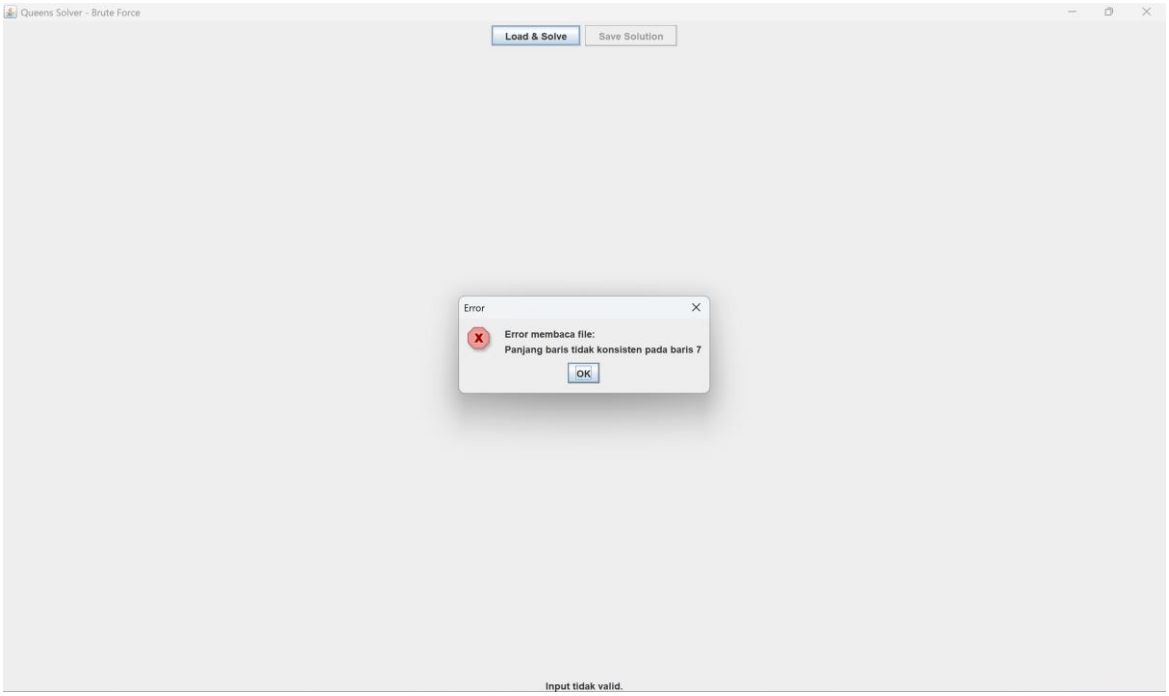
4.8. *Test Case* 8: Invalid karena Tidak Semua Baris Konsisten Panjangnya

tc8.txt

test > tc8.txt

```
1   ABCDEFGHIJKLM
2   ABCDEFGHIJKLM
3   ABCDEFGHIJKLM
4   ABCDEFGHIJKLM
5   ABCDEFGHIJKLM
6   ABCDEFGHIJKLM
7   ABCDEFGHIJKL
8   ABCDEFGHIJKLM
9   ABCDEFGHIJKLM
10  ABCDEFGHIJKLM
11  ABCDEFGHIJKLM
12  ABCDEFGHIJKLM
13  ABCDEFGHIJKLM
```

Output penyelesaian tc8.txt pada GUI



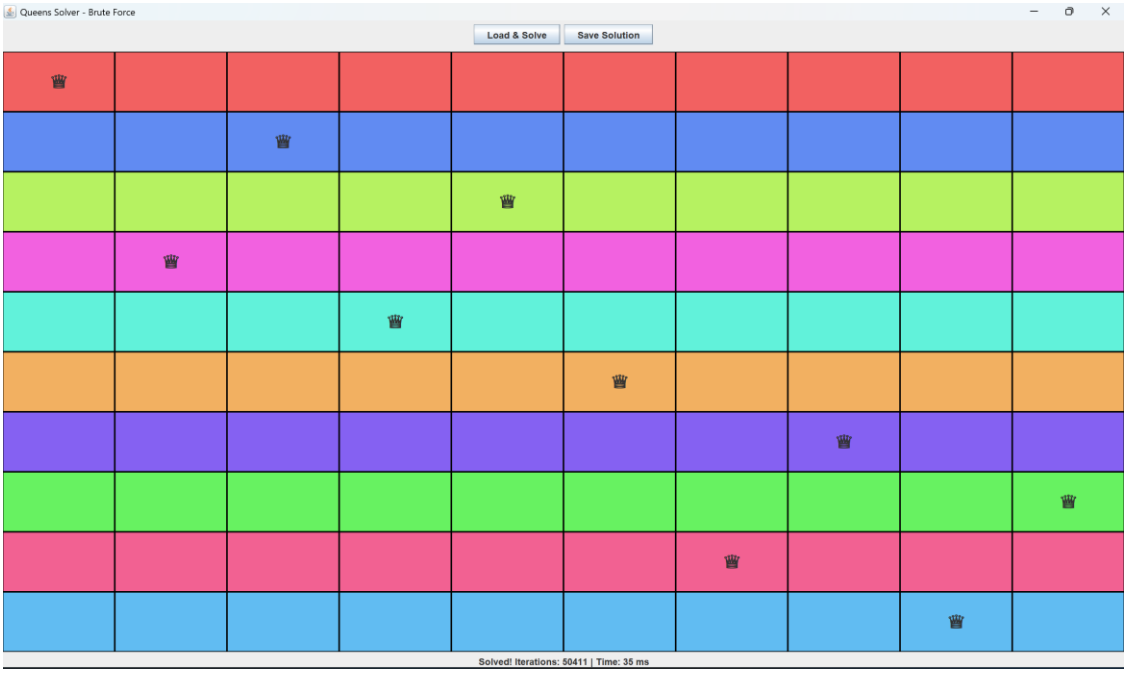
4.9. *Test Case* 9: Strip Horizontal

tc9.txt

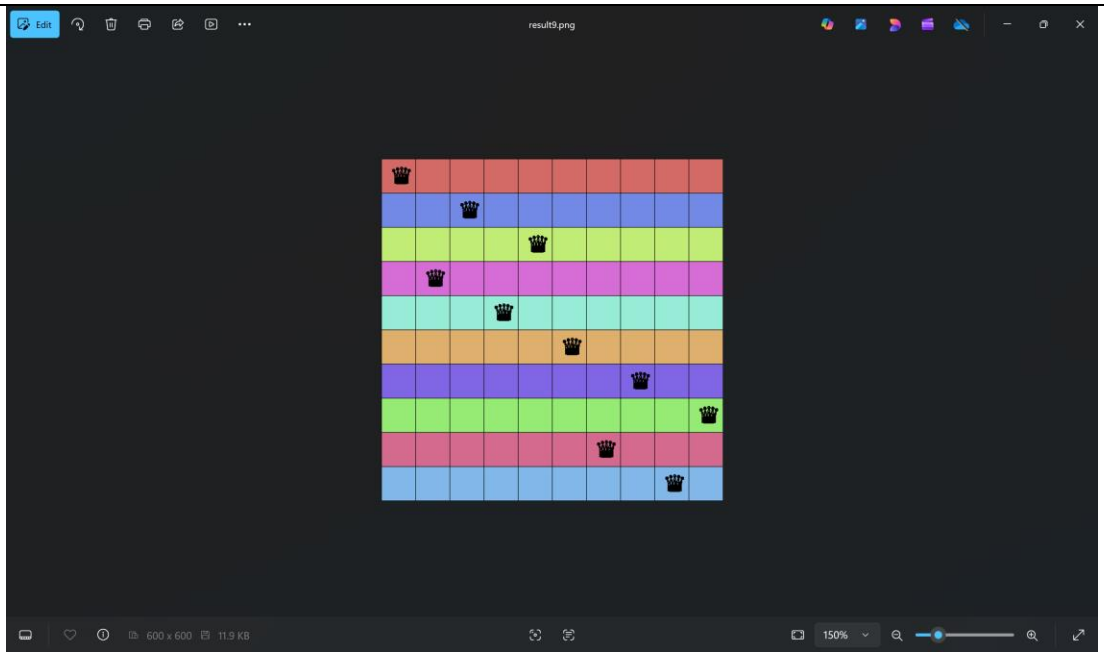
test > tc9.txt

```
1  AAAAAAAAAA
2  BBBBBBBBBB
3  CCCCCCCCCC
4  DDDDDDDDDD
5  EEEEEEEEEE
6  FFFFFFFFFF
7  GGGGGGGGGG
8  HHHHHHHHHH
9  IIIIIIIIII
10 JJJJJJJJJJ
```

Output penyelesaian tc9.txt pada GUI



Output setelah disimpan sebagai gambar (.png)



Output setelah disimpan sebagai text (.txt)

test > result > result9.txt

```
1  #AAAAAAAAA
2  BB#BBBBBBB
3  CCCC#CCCCC
4  D#DDDDDDDD
5  EEE#EEEEEE
6  FFFFF#FFFF
7  GGGGGGG#GG
8  HHHHHHHHH#
9  IIIIII#III
10 JJJJJJJJ#J
11
```

4.10. *Test Case* 10: Tangga

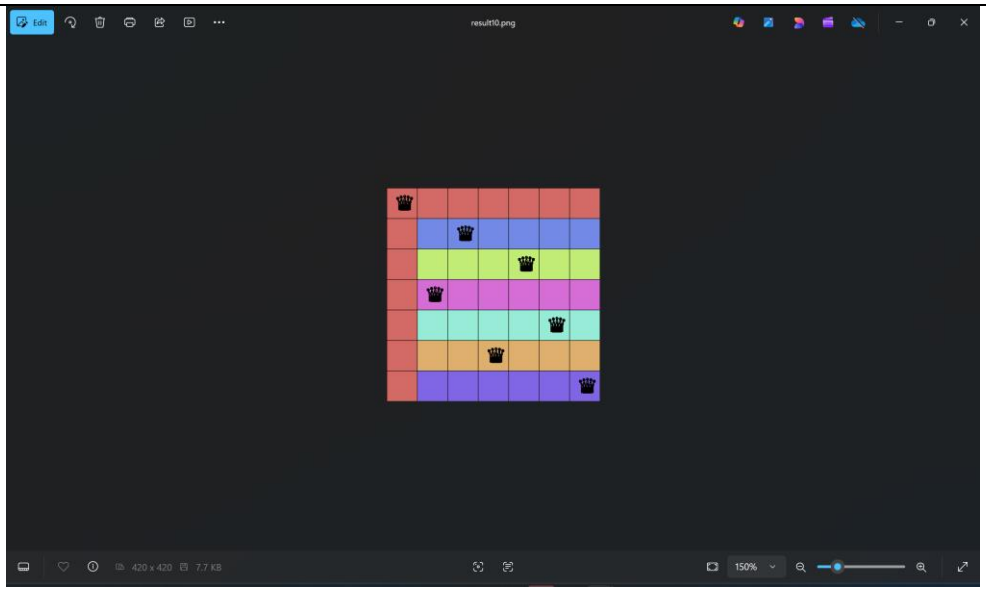
tc10.txt

test > tc10.txt

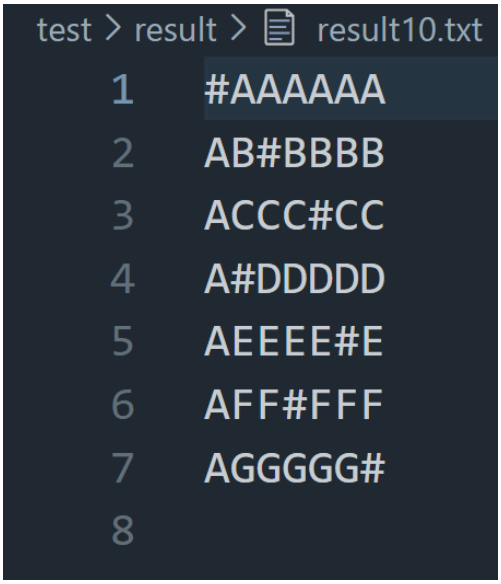
```
1 AAAAAAA
2 ABBBBBB
3 ACCCCCC
4 ADDDDDD
5 AEEEEEE
6 AFFFFFF
7 AGGGGGG
```

Output penyelesaian tc10.txt pada GUI

Output setelah disimpan sebagai gambar (.png)



Output setelah disimpan sebagai text (.txt)



BAB V
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 *Graphical User Interface* (GUI)	✓	
6	Program dapat menyimpan solusi dalam bentuk file gambar	✓	

DAFTAR PUSTAKA

Jain, S. *Brute Force Approach and Its Pros and Cons.* GeeksforGeeks. 2024. Available: Brute Force Approach and its pros and cons - GeeksforGeeks [Accessed: February 14, 2026].

Java Tutorial. W3Schools. n.d. 2026. Available: https://www.w3schools.com/java/default.asp [Accessed: February 15, 2026].

Munir, R. *Algoritma Brute Force (Bagian 1)*. Institut Teknologi Bandung. 2026. Available: 02-Algoritma-Brute-Force-(2026)-Bag1.pdf [Accessed: February 14, 2026].

Munir, R. *Algoritma Brute Force (Bagian 2)*. Institut Teknologi Bandung. 2026. Available: 03-Algoritma-Brute-Force-(2026)-Bag2.pdf [Accessed: February 14, 2026].

Munir, R. *Pengantar Strategi Algoritma*. Institut Teknologi Bandung. 2026. Available: 01-Pengantar-Strategi-Algoritma-(2026).pdf [Accessed: February 14, 2026].

Swing Introduction - Tpoint Tech. JavaTpoint. n.d. 2026. Available: https://www.tpointtech.com/java-swing [Accessed: February 15, 2026].

PERNYATAAN PRIBADI

Tugas ini disusun sepenuhnya tanpa bantuan kecerdasan buatan (*Generative AI*), melainkan hasil pemikiran dan analisis mandiri.

Kalyca Nathania B. Manullang