

Laporan Tugas Kecil 1
IF2211 - Strategi Algoritma
Penyelesaian Permainan *Queens* LinkedIn



Disusun oleh:

Nazwan Siddqi Muttaqin/18223066

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

2025

A. Penjelasan Algoritma

Dalam menyelesaikan permainan *Queens* dari LinkedIn ini, program menggunakan pendekatan algoritma *Brute Force* dengan teknik *Exhaustive Search*. Algoritma ini bekerja dengan cara men-*generate* seluruh kemungkinan konfigurasi penempatan queen pada board, kemudian melakukan pengujian dan validasi terhadap setiap konfigurasi tersebut untuk memverifikasi apakah seluruh aturan permainan terpenuhi.

1. Algoritma Utama

Pada pendekatan ini, algoritma mendefinisikan ruang pencarian seluas mungkin tanpa melakukan constraint terhadap posisi kolom. Algoritma bekerja dengan asumsi bahwa setiap *queen* pada baris ke- i dapat ditempatkan di kolom mana saja dari 0 hingga $N - 1$, tanpa mempedulikan apakah kolom tersebut sudah terisi oleh *queen* lain atau belum pada saat proses *generating*.

Langkah - langkah algoritma:

1. Representasi Solusi

Solusi direpresentasikan sebagai sebuah *array* satu dimensi $P = [c_0, c_1, \dots, c_{n-1}]$, di mana indeks i menyatakan baris ke- i dan nilai c_i menyatakan posisi kolom dari *queen* pada baris tersebut. Representasi ini lebih baik dibandingkan menggunakan 2D karena mempersempit ruang solusi tanpa menyalahi aturan *brute force* dengan teknik *Exhaustive Search* yang men-*generate* semua kemungkinan solusi yang ada.

2. Generate Candidate

Algoritma men-*generate* seluruh kombinasi penempatan yang mungkin dengan prinsip *Cartesian Product*. Untuk papan berukuran $N \times N$, setiap baris memiliki N pilihan kolom.

- Contoh kandidat yang di-*generate* termasuk konfigurasi yang melanggar *constraint* seperti $[0, 0, 0, 0]$
- Total kandidat solusi yang di-*generate* dan diperiksa adalah sebanyak N^N .

3. Validasi Konfigurasi

Setiap kandidat solusi yang di-*generate* akan divalidasi berdasarkan aturan khusus permainan *Queens*, yaitu:

- *Column Constraint*: Memeriksa apakah terdapat duplikasi nilai dalam array solusi. Jika ada (artinya satu kolom berisi > 1 ratu), solusi tidak valid.
- *Color Constraint*: Memastikan tidak ada dua ratu atau lebih yang menempati wilayah warna yang sama. Program memetakan setiap sel (baris, kolom) ke warna tertentu sesuai input, dan memeriksa duplikasi warna pada konfigurasi saat ini.

- *Neighbor/Touching Constraint*: Memastikan tidak ada dua ratu yang bersentuhan, baik secara horizontal, vertikal, maupun diagonal. Algoritma ini menerapkan optimasi dengan hanya membandingkan posisi *queen* pada baris ke- i dengan *queen* pada baris tepat di bawahnya ($i + 1$), bukan dengan cara membandingkan seluruh pasangan *queen* dengan *nested loop*. Hal ini didasarkan pada prinsip bahwa dua *queen* dapat bersentuhan jika selisih barisnya adalah 1. Jika selisih posisi kolom antara baris i dan $i + 1$ kurang dari 2 ($|c_i - c_{i+1}| < 2$), maka kedua *queen* tersebut dinyatakan bersentuhan (baik secara vertikal maupun diagonal), sehingga konfigurasi dinyatakan tidak valid.

4. Mekanisme Pencarian

Algoritma melakukan iterasi terhadap setiap kandidat solusi yang dihasilkan. Jika ditemukan sebuah konfigurasi yang memenuhi seluruh kriteria validasi di atas, program akan langsung menghentikan pencarian dan mengembalikan konfigurasi, jumlah iterasi, dan waktu eksekusi tersebut sebagai solusi. Sebaliknya, jika seluruh kandidat solusi tidak ditemukan konfigurasi yang valid, program akan menyimpulkan bahwa tidak solusi tidak ada.

5. Kompleksitas Algoritma

Kompleksitas waktu dari algoritma ini adalah $O(N \cdot N^N)$. Faktor N^N diperoleh dari banyaknya kemungkinan solusi yang di-*generate* dengan eksponensial. Sedangkan faktor N diperoleh dari fungsi cek_validasi, di mana algoritma melakukan iterasi tunggal sebanyak $N - 1$ kali untuk memeriksa constraint yang sudah didefinisikan.

2. Algoritma Optimasi

Sebagai pembanding kinerja, program juga menyertakan mode kedua yang menggunakan pendekatan Brute Force pada Ruang Permutasi. Berbeda dengan metode biasa, metode ini mendefinisikan ruang pencarian hanya pada himpunan permutasi dari angka kolom $[0, \dots, N - 1]$.

- Mekanisme: *Generate* kandida dilakukan dengan menukar urutan kolom yang unik. Kandidat seperti $[0, 0, 0, 0]$ tidak akan pernah di-*generate*.
- Dampak : *Constraint* "satu ratu per kolom" terpenuhi secara otomatis oleh struktur data permutasi, sehingga validasi kolom tidak lagi diperlukan. Validasi hanya dilakukan untuk Warna dan Tetangga.
- Kompleksitas: Ruang pencarian tereduksi drastis menjadi $N!$ (Faktorial).

$$O(N \cdot N!)$$

B. Source Code

Berikut adalah implementasi kode program utama yang ditulis dalam bahasa Python. Kode dibagi menjadi kelas *QueensSolver* yang menangani logika algoritma dan kelas *QueensApp* yang menangani GUI.

```
class QueenSolver:

import customtkinter as ctk
from tkinter import filedialog, messagebox
from PIL import Image, ImageDraw, ImageFont
import time
import os
import re
import sys

ctk.set_appearance_mode("dark")
ctk.set_default_color_theme("blue")

class QueensSolver:
    def __init__(self, grid_warna):
        self.grid_warna = grid_warna
        self.n = len(grid_warna)
        self.solusi = None
        self.jumlah_percobaan = 0

    def generate_permutasi(self, elemen):
        if len(elemen) <= 1:
            yield elemen
        else:
            for permutasi in self.generate_permutasi(elemen[1:]):
                for i in range(len(elemen)):
                    yield permutasi[:i] + [elemen[0]] + permutasi[i:]

    def generate_semua_kombinasi(self, panjang, current_posisi=[]):
```

```

        if len(current_posisi) == panjang:
            yield current_posisi
        else:
            for i in range(self.n):
                yield from self.generate_semua_kombinasi(panjang,
current_posisi + [i])

def cek_validasi(self, posisi_queens):
    self.jumlah_percobaan += 1
    warna_terpakai = []

    # Constraint Kolom
    if len(set(posisi_queens)) != self.n:
        return False

    # Constraint Warna
    for baris in range(self.n):
        kolom = posisi_queens[baris]
        warna = self.grid_warna[baris][kolom]

        if warna in warna_terpakai:
            return False
        warna_terpakai.append(warna)

    # Constraint Neighbor
    for i in range(self.n - 1):
        selisih_kolom = abs(posisi_queens[i] - posisi_queens[i+1])
        if selisih_kolom < 2:
            return False

    return True

def cari_solusi(self):
    generator = self.generate_semua_kombinasi(self.n)

```

```

        for kemungkinan in generator:
            is_valid = self.cek_validasi(kemungkinan)
            if is_valid:
                yield kemungkinan, True, True
                return
            else:
                yield kemungkinan, False, False

        yield None, False, True

def cari_solusi_optimal(self):
    angka_kolom = list(range(self.n))
    generator = self.generate_permutasi(angka_kolom)

    for kemungkinan in generator:
        is_valid = self.cek_validasi(kemungkinan)
        if is_valid:
            yield kemungkinan, True, True
            return
        else:
            yield kemungkinan, False, False

    yield None, False, True

```

1. `__init__(self, grid_warna)`
 - Fungsi: Konstruktor inisialisasi.
 - Penjelasan: Menyimpan data warna grid, menentukan ukuran papan berdasarkan panjang grid, dan mereset penghitung jumlah percobaan (`self.jumlah_percobaan`) ke 0.
2. `generate_permutasi(self, elemen)`
 - Fungsi: Generator Permutasi (untuk Mode Optimasi).
 - Penjelasan: Fungsi rekursif yang menghasilkan urutan acak unik dari angka-angka kolom.
3. `generate_semua_kombinasi(self, panjang, current_posisi=[])`
 - Fungsi: Generator Kombinasi Lengkap (untuk Mode Standar).

- Penjelasan: Fungsi rekursif yang menghasilkan semua kemungkinan kombinasi posisi kolom (termasuk yang berulang), sesuai prinsip Cartesian Product.
4. `cek_validasi(self, posisi_queens)`
- Fungsi: Validator Aturan.
 - Penjelasan: Memeriksa apakah kandidat solusi memenuhi 3 aturan:
 1. Kolom Unik: Tidak ada kolom yang kembar.
 2. Warna: Tidak ada dua ratu di wilayah warna yang sama.
 3. Neighbor: Tidak ada dua ratu yang bersentuhan (vertikal/diagonal) dengan mengecek selisih baris dan kolom.
 - Penjelasan: Memeriksa apakah kandidat solusi memenuhi 3 aturan:
5. `cari_solusi(self)`
- Fungsi: Eksekusi Brute Force Standar.
 - Penjelasan: Mengambil kandidat dari `generate_semua_kombinasi`, memvalidasinya satu per satu, dan menggunakan `yield` untuk mengirim update visualisasi ke GUI secara bertahap.
6. `cari_solusi_optimal(self)`
- Fungsi: Eksekusi Brute Force Optimasi.
 - Penjelasan: Mengambil kandidat dari `generate_permutasi`, memvalidasinya satu per satu. Proses ini jauh lebih cepat karena ruang pencarian lebih sedikit.

class QueensApp:

```
class QueensApp(ctk.CTk):  
    def __init__(self):  
        super().__init__()   
  
        self.title("Tucil 1 IF2211 - Queens Linkedin Solver")  
        self.geometry("900x600")  
  
        self.grid_data = []  
        self.tombol_grid = {}  
        self.solver = None  
        self.generator_solusi = None  
        self.start_time = 0  
        self.is_running = False  
        self.input_filename = None
```

```

self.base_path = self.get_base_path()
self.previous_queens = None

self.grid_columnconfigure(1, weight=1)
self.grid_rowconfigure(0, weight=1)
self.setup_sidebar()
self.setup_board_area()

def setup_sidebar(self):
    frame_kiri = ctk.CTkFrame(self, width=200, corner_radius=0)
    frame_kiri.grid(row=0, column=0, sticky="nsew")

    label_judul = ctk.CTkLabel(frame_kiri, text="Queens Linked in
Solver\nBrute Force", font=("Roboto Medium", 20))
    label_judul.pack(pady=20, padx=5)

    self.btn_load = ctk.CTkButton(frame_kiri, text="Pilih File .txt",
command=self.load_file, font=("Roboto", 16))
    self.btn_load.pack(pady=10, padx=20)
    self.btn_start = ctk.CTkButton(frame_kiri, text="Cari Solusi",
command=self.start_solusi, state="disabled", font=("Roboto", 16))
    self.btn_start.pack(pady=10, padx=20)
    self.btn_start_optimal = ctk.CTkButton(frame_kiri, text="Cari Solusi
Optimal", command=self.start_solusi_optimal, state="disabled", font=("Roboto",
16))
    self.btn_start_optimal.pack(pady=10, padx=20)
    self.label_status = ctk.CTkLabel(frame_kiri, text="Menunggu file",
font=("Roboto", 14))
    self.label_status.pack(pady=(20, 5))
    self.label_iterasi = ctk.CTkLabel(frame_kiri, text="Iterasi: 0",
font=("Roboto", 14))
    self.label_iterasi.pack(pady=5)
    self.label_waktu = ctk.CTkLabel(frame_kiri, text="Waktu: 0 ms",
font=("Roboto", 14))

```



```

        self.label_waktu.pack(pady=5)

    def setup_board_area(self):
        self.frame_kanan = ctk.CTkFrame(self)
        self.frame_kanan.grid(row=0, column=1, sticky="nsew", padx=20, pady=20)

    def load_file(self):
        file_path = filedialog.askopenfilename(filetypes=[("Text files",
        "*.txt")])
        if not file_path:
            return

        self.input_filename = os.path.basename(file_path)

        with open(file_path, 'r') as f:
            lines = f.readlines()

        self.grid_data = [list(line.strip()) for line in lines if line.strip()]
        self.n = len(self.grid_data)

        for line in self.grid_data:
            if len(line) != self.n:
                messagebox.showerror("Error", "File tidak valid: Grid harus
                persegi (NxN).")
                return

        self.gambar_papan()
        self.label_status.configure(text=f"File berhasil dimuat
        ({self.n}x{self.n})")
        self.btn_start.configure(state="normal")
        self.btn_start_optimal.configure(state="normal")

    def get_warna_region(self, huruf):
        warna_map = {

```

```

        'A': '#FFB5BA',
        'B': '#B5F0D5',
        'C': '#A8D8FF',
        'D': '#FFED99',
        'E': '#D9B3FF',
        'F': '#FFB380',
        'G': '#B8E994',
        'H': '#87E0D9',
        'I': '#FFA8CC',
        'J': '#C2A3FF',
        'K': '#FFDAA8',
        'L': '#FF9999',
        'M': '#99E6D9',
        'N': '#B3D9FF',
        'O': '#FFE5B4',
        'P': '#E6B3D6',
        'Q': '#D4FF99',
        'R': '#FFC2E0',
        'S': '#FFFAAA',
        'T': '#FFCC99',
        'U': '#C4E6E6',
        'V': '#D9D9D9',
        'W': '#A8E6CF',
        'X': '#B3B3FF',
        'Y': '#F5C2F5',
        'Z': '#99FFEB',
    }

    return warna_map.get(huruf.upper(), '#E8E8E8')

def gambar_papan(self):
    for widget in self.frame_kanan.wininfo_children():
        widget.destroy()

    self.tombol_grid = {}

```

```

for i in range(self.n):
    self.frame_kanan.grid_rowconfigure(i, weight=1)
    self.frame_kanan.grid_columnconfigure(i, weight=1)

for r in range(self.n):
    for c in range(self.n):
        huruf_region = self.grid_data[r][c]
        warna_bg = self.get_warna_region(huruf_region)

        kotak = ctk.CTkButton(self.frame_kanan, text=huruf_region,
fg_color=warna_bg, text_color=warna_bg, hover=False, corner_radius=0,
font=("Arial", 24, "bold"), border_width=0.5, border_color="black")
        kotak.grid(row=r, column=c, sticky="nsew")
        self.tombol_grid[(r, c)] = kotak

def start_solusi(self):
    if not self.grid_data:
        return

    self.solver = QueensSolver(self.grid_data)
    self.generator_solusi = self.solver.cari_solusi()

    self.is_running = True
    self.start_time = time.time()
    self.btn_start.configure(state="disabled")
    self.btn_load.configure(state="disabled")
    self.label_status.configure(text="Mencari solusi..",
text_color="#75BAFE")

    self.update_logika()

# Solusi optimal menggunakan permutasi untuk memastikan tidak ada duplikasi
kolom, sehingga lebih cepat menemukan solusi jika ada

```

```

def start_solusi_optimal(self):
    if not self.grid_data:
        return

    self.solver = QueensSolver(self.grid_data)
    self.generator_solusi = self.solver.cari_solusi_optimal()

    self.is_running = True
    self.start_time = time.time()
    self.btn_start.configure(state="disabled")
    self.btn_start_optimal.configure(state="disabled")
    self.btn_load.configure(state="disabled")
    self.label_status.configure(text="Mencari solusi..",
text_color="#75BAFE")

    self.update_logika()

def update_logika(self):
    if not self.is_running:
        return

    try:
        steps_per_frame = 100
        data_terakhir = None
        found = False
        finished = False

        for _ in range(steps_per_frame):
            data_terakhir, found, finished = next(self.generator_solusi)
            if finished:
                break

        waktu_berjalan = (time.time() - self.start_time) * 1000
        self.label_iterasi.configure(text=f"Iterasi:

```

```

{self.solver.jumlah_percobaan})
        self.label_waktu.configure(text=f"Waktu: {waktu_berjalan:.2f} ms")

        if data_terakhir:
            self.visualisasi_queens(data_terakhir)

        if finished:
            self.is_running = False
            self.btn_load.configure(state="normal")
            self.btn_start.configure(state="normal")
            self.btn_start_optimal.configure(state="normal")

            if found:
                self.label_status.configure(text="Solusi ditemukan!",
text_color="green")
                messagebox.showinfo("Sukses", f"Solusi ditemukan dalam
{self.solver.jumlah_percobaan} iterasi dan {waktu_berjalan:.2f} ms.")

                simpan = messagebox.askyesno("Simpan Solusi", "Apakah Anda
ingin menyimpan solusi?")
                if simpan:
                    self.simpan_solusi(data_terakhir, waktu_berjalan)
                else:
                    self.label_status.configure(text="Tidak ada solusi.",
text_color="red")
                    messagebox.showinfo("Info", "Algoritma selesai, tidak ada
solusi valid.")
            else:
                self.after(1, self.update_logika)

        except StopIteration:
            pass

    def get_base_path(self):

```

```

if getattr(sys, 'frozen', False):
    return os.path.dirname(os.path.dirname(sys.executable))
else:
    return os.path.dirname(os.path.dirname(os.path.abspath(__file__)))

def simpan_solusi(self, posisi_queens, waktu_ms):
    test_folder = os.path.join(self.base_path, "test")
    os.makedirs(test_folder, exist_ok=True)

    nomor = "1"
    if self.input_filename:
        nums = re.findall(r'\d+', self.input_filename)
        if nums:
            nomor = nums[0]

    filepath_txt = os.path.join(test_folder, f"output{nomor}.txt")

    with open(filepath_txt, 'w') as f:
        for r in range(self.n):
            for c in range(self.n):
                ada_queen = (c == posisi_queens[r])
                if ada_queen:
                    f.write("#")
                else:
                    huruf = self.grid_data[r][c]
                    f.write(huruf)
            f.write("\n")

    self.simpan_solusi_gambar(posisi_queens, nomor)

    messagebox.showinfo("Sukses", f"Solusi berhasil disimpan:\n- {filepath_txt}\n- {os.path.join(test_folder, f'output{nomor}.png')}")

def simpan_solusi_gambar(self, posisi_final, nomor):

```

```

if not posisi_final:
    return

test_folder = os.path.join(self.base_path, "test")
os.makedirs(test_folder, exist_ok=True)

cell_size = 100
img_size = self.n * cell_size

image = Image.new("RGB", (img_size, img_size), "white")
draw = ImageDraw.Draw(image)
font_queen = ImageFont.truetype("seguisym.ttf", size=int(cell_size *
0.6))

for r in range(self.n):
    for c in range(self.n):
        x0 = c * cell_size
        y0 = r * cell_size
        x1 = x0 + cell_size
        y1 = y0 + cell_size

        huruf = self.grid_data[r][c]
        warna_hex = self.get_warna_region(huruf)

        draw.rectangle([x0, y0, x1, y1], fill=warna_hex,
outline="black", width=1)

        if posisi_final[r] == c:
            center_x = x0 + (cell_size / 2)
            center_y = y0 + (cell_size / 2)
            draw.text((center_x, center_y), "👑", fill="black",
font=font_queen, anchor="mm")

filepath = os.path.join(test_folder, f"output{nomor}.png")

```

```

        image.save(filepath)

def visualisasi_queens(self, posisi_queens):
    if self.previous_queens:
        for r, old_c in enumerate(self.previous_queens):
            new_c = posisi_queens[r] if r < len(posisi_queens) else None

            if old_c != new_c and old_c is not None:
                tombol = self.tombol_grid[(r, old_c)]
                huruf_region = self.grid_data[r][old_c]
                tombol.configure(text="",
                                text_color=self.get_warna_region(huruf_region))

        for r, c in enumerate(posisi_queens):
            if c is not None and 0 <= c < self.n:
                tombol = self.tombol_grid[(r, c)]
                tombol.configure(text="♔", text_color="black", font=("Segoe UI
Symbol", 36, "bold"))
                self.previous_queens = posisi_queens.copy()

if __name__ == "__main__":
    app = QueensApp()
    app.mainloop()

```

1. `__init__(self)`

- Fungsi: Konstruktor Utama Aplikasi.
- Penjelasan: Menginisialisasi window, mengatur judul dan dimensi aplikasi, menyiapkan variabel state (seperti `grid_data`, `solver`), serta memanggil fungsi untuk menyusun layout antarmuka (sidebar dan `board_area`).

2. `setup_sidebar(self)`

- Fungsi: Inisialisasi Panel Kontrol.

- Penjelasan: Membuat panel di sisi kiri yang berisi elemen kontrol seperti tombol "Pilih File", tombol eksekusi untuk kedua mode algoritma ("Cari Solusi" dan "Cari Solusi Optimal"), serta label status informasi (iterasi dan waktu).
3. `setup_board_area(self)`
 - Fungsi: Inisialisasi Area Papan.
 - Penjelasan: Menyiapkan frame kontainer di sisi kanan tempat grid papan permainan akan dirender.
 4. `load_file(self)`
 - Fungsi: Penanganan Input File.
 - Penjelasan: Membuka dialog file untuk memilih file .txt, membaca isi grid, memvalidasi dimensi agar berbentuk persegi, dan memicu penggambaran papan awal.
 5. `get_warna_region(self, huruf)`
 - Fungsi: Pemetaan Warna.
 - Penjelasan: Mengonversi karakter wilayah (input huruf) menjadi kode warna heksadesimal spesifik untuk pewarnaan visual papan.
 6. `gambar_papan(self)`
 - Fungsi: Rendering Grid Awal.
 - Penjelasan: Membuat ulang widget tombol grid pada GUI berdasarkan data input yang baru dimuat, mewarnainya sesuai region, dan menyimpannya ke dalam dictionary `self.tombol_grid`.
 7. `start_solusi(self)`
 - Fungsi: Eksekusi Algoritma.
 - Penjelasan: Menginisialisasi QueensSolver dan memulai proses pencarian menggunakan metode `cari_solusi()`, serta mengaktifkan loop pembaruan UI.
 8. `start_solusi_optimal(self)`
 - Fungsi: Eksekusi Algoritma Optimasi.
 - Penjelasan: Menginisialisasi QueensSolver dan memulai proses pencarian menggunakan metode `cari_solusi_optimal()`, yang lebih efisien untuk input besar.
 9. `update_logika(self)`
 - Fungsi: Loop Animasi & Logika.

- Penjelasan: Fungsi rekursif yang mengambil langkah berikutnya dari generator algoritma, memperbarui label statistik, dan memanggil visualisasi papan. Fungsi ini juga menangani kondisi terminal (solusi ditemukan/tidak) dan dialog penyimpanan.

10. `get_base_path(self)`

- Fungsi: Utilitas Path Direktori.
- Penjelasan: Mendapatkan path direktori yang benar untuk operasi file, memastikan kompatibilitas saat dijalankan sebagai script Python maupun executable (.exe).

11. `simpan_solusi(self, ...)`

- Fungsi: Penyimpanan Output Teks.
- Penjelasan: Menulis representasi solusi akhir (grid karakter dengan simbol '#') ke dalam file .txt di direktori test/.

12. `simpan_solusi_gambar(self, ...)`


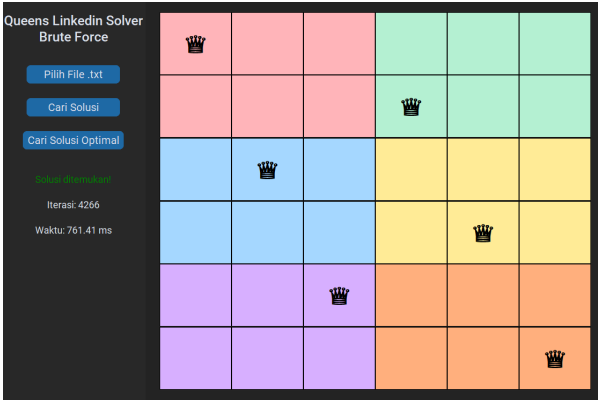
- Fungsi: Penyimpanan Output Gambar.
- Penjelasan: Menggunakan library PIL untuk menggambar ulang papan solusi secara grafis dan menyimpannya sebagai file .png.

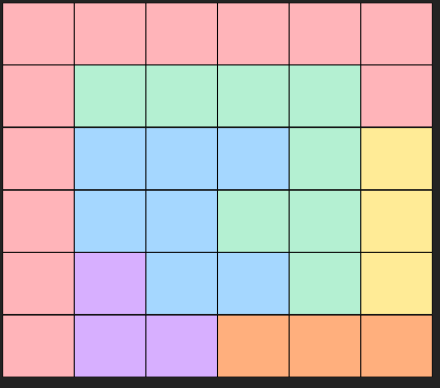
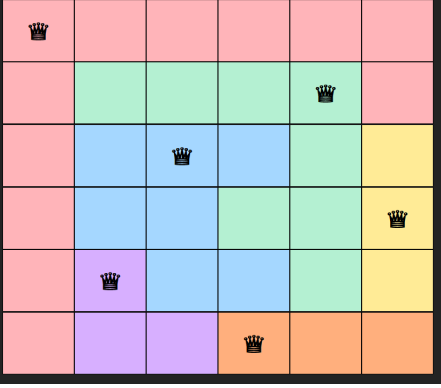
13. `visualisasi_queens(self, posisi_queens)`

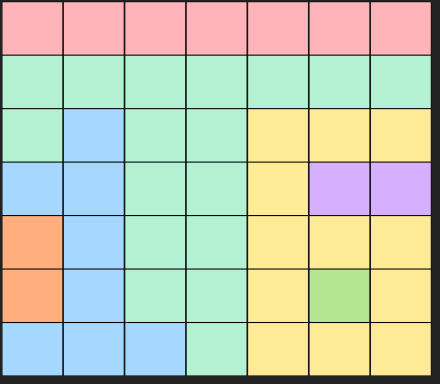
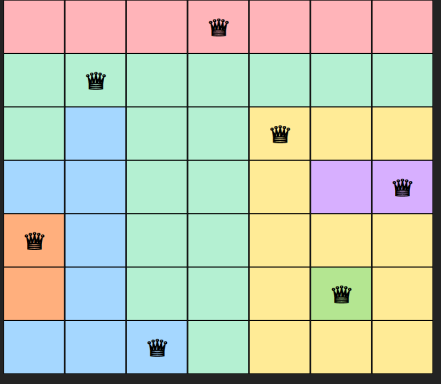
- Fungsi: Pembaruan Visual Papan.
- Penjelasan: Memperbarui posisi ikon ratu pada grid GUI secara real-time.

C. Hasil Program

Bagian ini memaparkan hasil eksekusi program terhadap beberapa *test cases* yang berbeda ukurannya. Setiap pengujian menampilkan visualisasi papan awal, visualisasi solusi akhir, serta data kinerja algoritma berupa waktu eksekusi dan jumlah iterasi.

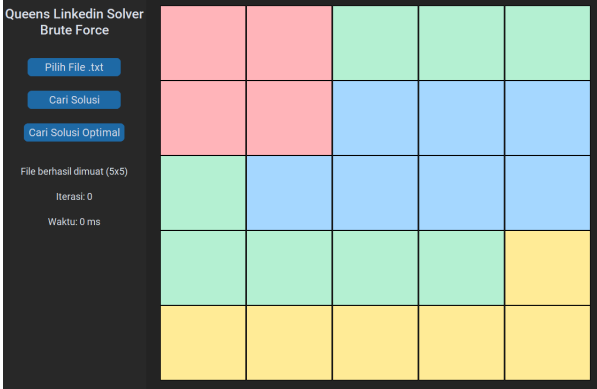
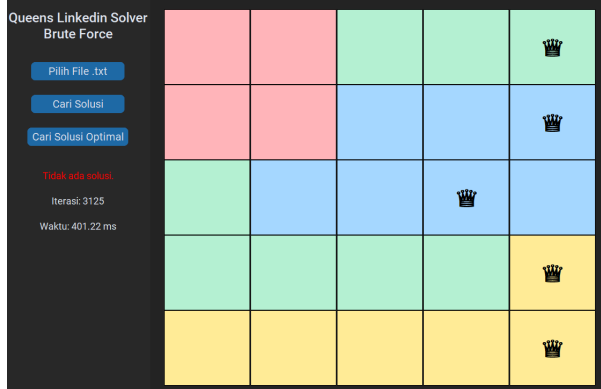
Percobaan 1	
Input	Output
 <p>AAABBB AAABBB CCCDDD CCCDDD EEEEFF EEEEFF</p>	 <p>#AABBB AAA#BB C#CDDD CCCD#D EE#FFF EEEEF#</p>
Iterasi : 4266	
Waktu : 761.41 ms	

Percobaan 2	
Input	Output
<div><div>Queens LinkedIn Solver Brute Force</div><div><div>Pilih File .txt</div><div>Cari Solusi</div><div>Cari Solusi Optimal</div></div><div>File berhasil dimuat (6x6)</div><div>Iterasi: 0</div><div>Waktu: 0 ms</div></div> 	<div><div>Queens LinkedIn Solver Brute Force</div><div><div>Pilih File .txt</div><div>Cari Solusi</div><div>Cari Solusi Optimal</div></div><div>File berhasil dimuat</div><div>Iterasi: 5806</div><div>Waktu: 1138.58 ms</div></div> 
AAAAAA ABBBBA ACCCBD ACCBBD AECCBD AEEFFF	#AAAAA ABBB#A AC#CBD ACCB#B A#CCBD AEE#FF
Iterasi : 5806	
Waktu : 1138.58 ms	

Percobaan 3	
Input	Output
<div><div>Queens LinkedIn Solver Brute Force</div><div><div>Pilih File .txt</div><div>Cari Solusi</div><div>Cari Solusi Optimal</div></div><div>File berhasil dimuat (7x7)</div><div>Iterasi: 0</div><div>Waktu: 0 ms</div></div> 	<div><div>Queens LinkedIn Solver Brute Force</div><div><div>Pilih File .txt</div><div>Cari Solusi</div><div>Cari Solusi Optimal</div></div><div>File berhasil dimuat</div><div>Iterasi: 381454</div><div>Waktu: 79636.58 ms</div></div> 
AAAAAAA BBBBBBB	AAA#AAA B#BBBBB

BCBBDDD CCBBDEE FCBBDDD FCBBGD CCCBDDD	BCBB#DD CCBBDE# #CBBD FCBB#D CC#BDD
Iterasi : 381454	
Waktu : 79636.58 ms	

Percobaan 4	
Input	Output
<div> <div> Queens LinkedIn Solver Brute Force <div> <div>Pilih File .txt</div> <div>Cari Solusi</div> <div>Cari Solusi Optimal</div> </div> <div> File berhasil dimuat (4x4) Iterasi: 0 Waktu: 0 ms </div> </div> <div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> </div> </div> <div> AAAA BBBB BCCC DDDC </div>	<div> <div> Queens LinkedIn Solver Brute Force <div> <div>Pilih File .txt</div> <div>Cari Solusi</div> <div>Cari Solusi Optimal</div> </div> <div> <div>Solusi Berhasil</div> Iterasi: 142 Waktu: 27.16 ms </div> </div> <div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> </div> </div> <div> AA#A #BBB BCC# D#DC </div>
Iterasi : 142	
Waktu : 27.16 ms	

Percobaan 5	
Input	Output
 <p>AABBB AACCC BCCCC BBBBB DDDDD</p>	 <p>Tidak ada solusi</p>
Iterasi : 3125 (Tidak ada solusi)	
Waktu : 401.22 ms (Tidak ada solusi)	

D. Link Repository

Kode program lengkap beserta instruksi penggunaan dan *file executable* dapat diakses melalui *link repository* berikut:

[NazwanSM/Tucil1_18223066](https://github.com/NazwanSM/Tucil1_18223066)

E. Surat Pernyataan

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



Nazwan Siddqi Muttaqin

F. Lampiran

No	Poin	Ya	Tidak
1	Program berhasil di kompilasi tanpa kesalahan	✓	
2	Program berhasil di jalankan	✓	
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	✓	