LAPORAN TUGAS KECIL 1
IF2211 STRATEGI ALGORITMA

PENYELESAIAN PERMAINAN QUEENS LINKEDIN DENGAN ALGORITMA *BRUTE FORCE*

Disusun oleh:

Amanda Aurellia Salsabilla - 13524131

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2026

DAFTAR ISI

BAB 1
ALGORITMA

Algoritma *brute force* merupakan cara yang paling naif dan intuitif untuk memecahkan sebuah permasalahan. Dalam konteks Tugas Kecil 1 ini, algoritma brute force digunakan untuk menyelesaikan permainan mengasah otak yang diciptakan oleh LinkedIn, yaitu Queens. Permainan Queens menuntut pemain untuk meletakkan Queen di papan persegi yang terdiri dari blok-blok warna yang berbeda. Penempatan Queen harus mengikuti aturan berikut:

1. Hanya terdapat satu Queen di setiap baris
2. Hanya terdapat satu Queen di setiap kolom
3. Hanya terdapat satu Queen di setiap warna
4. Tidak boleh ada Queen yang bertetangga/bersebelahan (termasuk secara diagonal)

Dengan melibatkan rekursivitas, maka setiap kombinasi kemungkinan penempatan Queen akan dicari, divalidasi, dan ditetapkan sebagai solusi apabila valid. Jika tidak valid, maka akan digunakan pendekatan backtracking.

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

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

1. Input yang berasal dari file .txt akan diproses terlebih dahulu oleh *parser.py* dengan membaca semua baris, membersihkannya dari spasi/enter, serta menghapus baris yang kosong apabila ada. Setelah itu, dilakukan validasi terhadap panjang setiap baris, yang mana apabila tidak sama dengan N, maka akan menampilkan pesan error. Apabila lolos pemeriksaan, maka akan ditampilkan dimensi dari papan (N x N). Kumpulan baris yang sebelumnya berbentuk string akan diubah menjadi list of characters untuk masing-masing huruf (yang menandakan warna).
2. Pencarian solusi dilakukan secara rekursif oleh fungsi generate. Basisnya adalah ketika rekursi sampai ke baris paling bawah (papan telah penuh). Dalam proses percobaan semua kemungkinan kombinasi, dilakukan visualisasi (live update) setiap 100.000 iterasi dengan

menampilkan papan yang terdiri dari huruf-huruf input dan tanda ‘#’ sebagai Queen.

3. Fungsi *check_entire_board* bertugas untuk memeriksa apakah kombinasi peletakan Queens dari papan saat ini valid. Pengecekan dilakukan setelah seluruh baris terisi. Pada pelaksanaannya, list *current_placement* menyimpan posisi Queen dengan menggunakan indeks sebagai penanda baris dan value sebagai kolom.
4. Pengecekan constraints dijalani secara bertahap. Pengecekan kolom dilakukan dengan mengonversi list menjadi set lalu membandingkan panjang set dengan N. Apabila panjang set tidak sama dengan N, maka terdapat kolom yang duplikat dan tidak memenuhi constraints. Pengecekan warna dilakukan dengan memastikan apakah warna tersebut sudah terdaftar di list *used_colors* atau belum. Terakhir, pengecekan ketetanggaan dilakukan dengan membandingkan baris posisi saat ini dengan baris sebelumnya dan menghitung selisih kolom dari kedua baris tersebut. Apabila selisih kolom <= 1, maka Queen saling bertetangga.
5. Setelah kombinasi yang memenuhi constraints ditemukan, maka pencarian langsung dihentikan dan solusi akhir, waktu (dalam ms), serta jumlah iterasi ditampilkan. Terdapat opsi untuk menyimpan solusi yang telah didapatkan, baik dalam bentuk text maupun image.
6. Apabila problem yang diinput tidak memiliki solusi, maka program tetap akan mencari kemungkinan kombinasi terlebih dahulu sebelum menyatakan gagal mencari solusi.

Walaupun bisa dipastikan dapat memberikan jawaban, algoritma *brute force* bukanlah pilihan terbaik untuk memecahkan permasalahan Queens ini karena memakan waktu yang cukup lama dan kompleksitas dari algoritma ini akan bertumbuh secara eksponensial seiring dengan diperbesarnya ukuran papan dan jumlah warna yang digunakan. Program ini dapat mengeluarkan output yang sama melalui CLI dan GUI. Pada GUI, telah dimanfaatkan konsep multithreading demi meringankan beban kerja komputer dalam mencari solusi sekaligus menampilkan gambar.

BAB 2
SOURCE CODE

Bahasa yang digunakan dalam pembuatan program ini adalah bahasa Python. Program disimpan dalam folder src. Di dalam folder src, terdapat file *parser.py*, *solver.py*, *main.py* untuk CLI, dan *gui.py* untuk GUI. Seluruh test case diletakkan di folder test, yang di dalamnya terdiri atas subfolder input dan output. Folder input berisi file .txt, sementara folder output digunakan untuk menyimpan solusi, baik sebagai text maupun image.

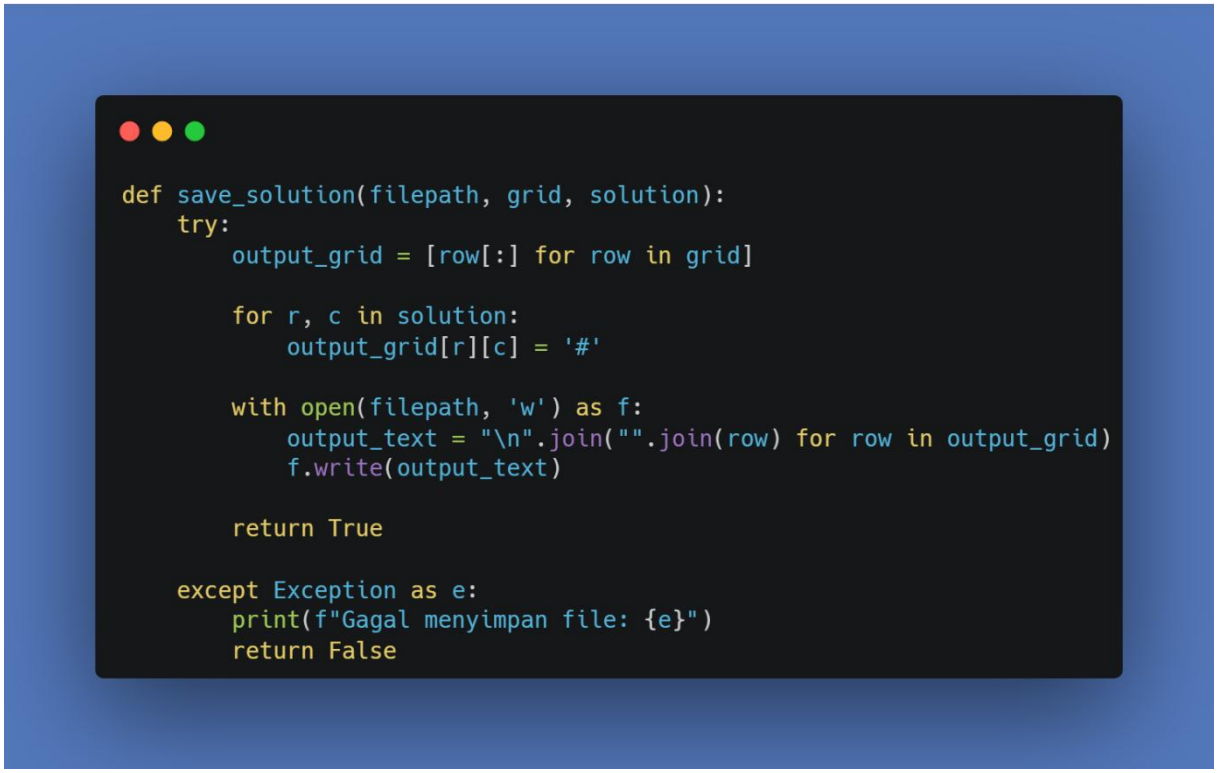
Algoritma inti terdapat dalam fungsi *solve_queens* yang terdiri dari fungsi *check_entire_board* dan *generate* untuk mencari solusi dari kemungkinan-kemungkinan kombinasi posisi Queen dan penempatannya pada papan. Berikut adalah source code dari fungsi-fungsi tersebut.

```python
def solve_queens(grid, N, visualize_callback=None):
    current_placement = [0] * N
    solution = []
    iterations = 0

    def check_entire_board(placement):
        if len(set(placement)) != N:
            return False

        used_colors = set()
        for r in range(N):
            c = placement[r]
            color_char = grid[r][c]

            if color_char in used_colors:
                return False
            used_colors.add(color_char)

        for r in range(1, N):
            curr_col = placement[r]
            prev_col = placement[r-1]

            if abs(curr_col - prev_col) <= 1:
                return False

        return True

      def generate(row):
        nonlocal iterations

        if row == N:
            iterations += 1

            if visualize_callback and iterations % 100000 == 0:
                 temp_sol = [(r, current_placement[r]) for r in range(N)]
                 visualize_callback(temp_sol, iterations)

            if check_entire_board(current_placement):
                nonlocal solution
                solution = [(r, current_placement[r]) for r in range(N)]
                return True

            return False

        for col in range(N):
            current_placement[row] = col

            if generate(row + 1):
                return True

        return False

    found = generate(0)

    if found:
        return solution, iterations
    else:
        return None, iterations
```

Untuk fungsi bantu seperti *parser*, berikut kode programnya.

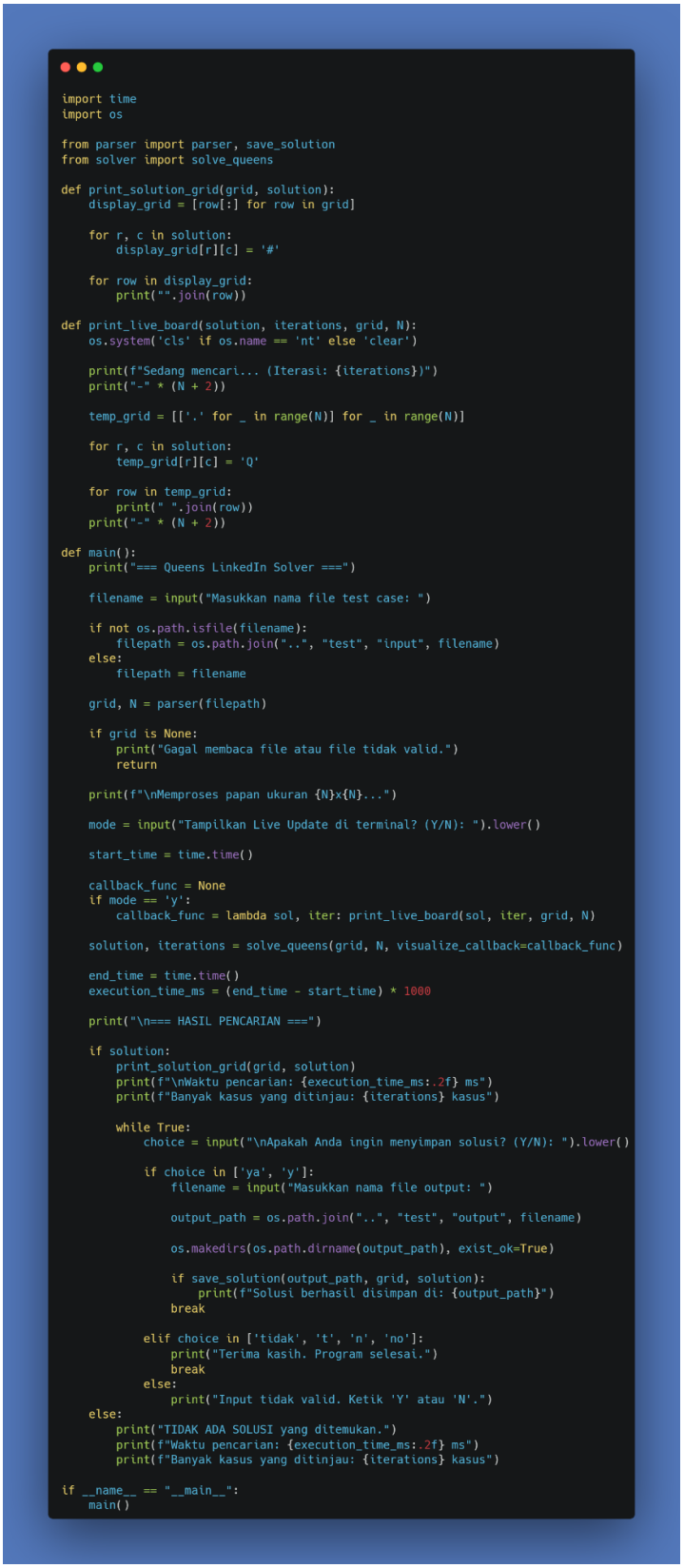
```python
def parser(filepath):
    grid = []

    if not os.path.exists(filepath):
        print(f"Error: File '{filepath}' tidak ditemukan.")
        return (None, None)

    try:
        with open(filepath, 'r') as f:
            lines = [line.rstrip() for line in f.readlines()]

            lines = [line for line in lines if line]

            if not lines:
                print("Error: File kosong.")
                return (None, None)

            N = len(lines)

            for i, line in enumerate(lines):
                if len(line) != N:
                    print(f"Error Validasi: Baris ke-{i+1} memiliki panjang {len(line)}, seharusnya {N}")
                    print("Input harus berupa papan persegi (N x N).")
                    return (None, None)

                grid.append(list(line))

            return (grid, N)

    except Exception as e:
        print(f"Terjadi kesalahan saat membaca file: {e}.")
        return (None, None)
```

Terdapat juga fungsi *save_solution* yang digunakan untuk menyimpan solusi apabila program dijalankan dengan CLI.



```python
def save_solution(filepath, grid, solution):
    try:
        output_grid = [row[:] for row in grid]

        for r, c in solution:
            output_grid[r][c] = '#'

        with open(filepath, 'w') as f:
            output_text = "\n".join("".join(row) for row in output_grid)
            f.write(output_text)

        return True

    except Exception as e:
        print(f"Gagal menyimpan file: {e}")
        return False
```

Apabila program ini dijalankan melalui CLI, maka program dijalankan melalui program *main.py* yang terdiri dari fungsi *print_solution_grid* untuk menampilkan solusi akhir, *print_live_board* untuk live update, dan fungsi *main* yang bertugas untuk menjalankan semua pekerjaan. Kode program dapat dilihat pada tangkapan layar berikut.



```python
import time
import os

from parser import parser, save_solution
from solver import solve_queens

def print_solution_grid(grid, solution):
    display_grid = [row[:] for row in grid]

    for r, c in solution:
        display_grid[r][c] = '#'

    for row in display_grid:
        print("".join(row))

def print_live_board(solution, iterations, grid, N):
    os.system('cls' if os.name == 'nt' else 'clear')

    print(f"Sedang mencari... (Iterasi: {iterations})")
    print("-" * (N + 2))

    temp_grid = [['.' for _ in range(N)] for _ in range(N)]

    for r, c in solution:
        temp_grid[r][c] = 'Q'

    for row in temp_grid:
        print(" ".join(row))
    print("-" * (N + 2))

def main():
    print("=== Queens LinkedIn Solver ===")

    filename = input("Masukkan nama file test case: ")

    if not os.path.isfile(filename):
        filepath = os.path.join("..", "test", "input", filename)
    else:
        filepath = filename

    grid, N = parser(filepath)

    if grid is None:
        print("Gagal membaca file atau file tidak valid.")
        return

    print(f"\nMemproses papan ukuran {N}x{N}...")

    mode = input("Tampilkan Live Update di terminal? (Y/N): ").lower()

    start_time = time.time()

    callback_func = None
    if mode == 'y':
        callback_func = lambda sol, iter: print_live_board(sol, iter, grid, N)

    solution, iterations = solve_queens(grid, N, visualize_callback=callback_func)

    end_time = time.time()
    execution_time_ms = (end_time - start_time) * 1000

    print("\n=== HASIL PENCARIAN ===")

    if solution:
        print_solution_grid(grid, solution)
        print(f"\nWaktu pencarian: {execution_time_ms:.2f} ms")
        print(f"Banyak kasus yang ditinjau: {iterations} kasus")

        while True:
            choice = input("\nApakah Anda ingin menyimpan solusi? (Y/N): ").lower()

            if choice in ['ya', 'y']:
                filename = input("Masukkan nama file output: ")

                output_path = os.path.join("..", "test", "output", filename)

                os.makedirs(os.path.dirname(output_path), exist_ok=True)

                if save_solution(output_path, grid, solution):
                    print(f"Solusi berhasil disimpan di: {output_path}")
                break

            elif choice in ['tidak', 't', 'n', 'no']:
                print("Terima kasih. Program selesai.")
                break
            else:
                print("Input tidak valid. Ketik 'Y' atau 'N'.")
    else:
        print("TIDAK ADA SOLUSI yang ditemukan.")
        print(f"Waktu pencarian: {execution_time_ms:.2f} ms")
        print(f"Banyak kasus yang ditinjau: {iterations} kasus")

if __name__ == "__main__":
    main()
```

Sedangkan, apabila program ingin dijalankan melalui GUI, maka dapat menggunakan program gui.py. Program juga mendukung masukan berupa file .txt ataupun gambar dengan format JPEG/JPG/PNG di GUI. Namun, karena kedua kode program tersebut terlalu panjang, maka kode program tidak akan ditampilkan pada bab ini. Untuk selengkapnya, dapat merujuk ke link repository Github yang terdapat di lampiran.

File *imageloader.py* berisi fungsi *rgb_dist* untuk menghitung jarak euclidean antara dua warna di gambar input, *merge_positions* untuk menggabungkan garis yang berdekatan, *pick_best_periodic_lines* untuk filter garis grid supaya jaraknya konsisten, *crop_inner_board* untuk menghapus frame hitam, *detect_grid_lines* untuk melihat outline kotak di papan, *sample_cell_color* untuk mengambil warna rata-rata dari kotak, *cluster_colors* untuk pengelompokan warna, *assign_letter* untuk mapping warna ke huruf, dan *process_image* sebagai fungsi utama untuk mengolah gambar masukan.

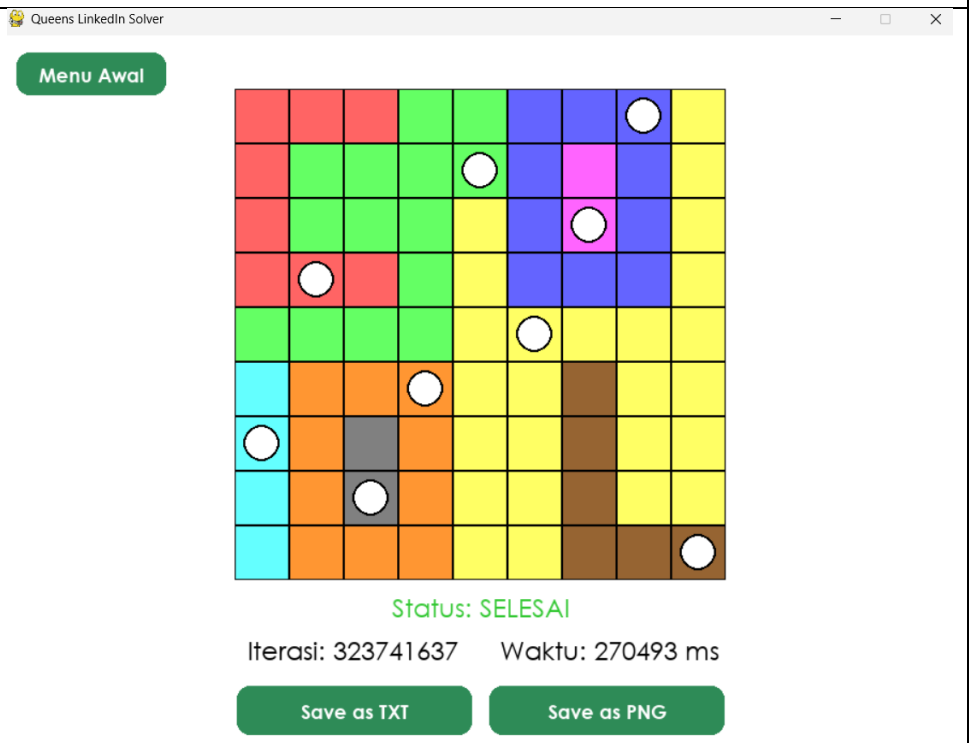
Pada dasarnya, *gui.py* menggunakan fungsi-fungsi seperti *layout_menu* dan *layout_solver* untuk kedua halaman, *load_input* untuk membuka file dialog dan menerima masukan file .txt ataupun image, *reset_state* untuk tombol kembali ke menu awal, *start_solver* ketika gambar berhasil diload, *update_visual* untuk memperlihatan live update, *run_solver* untuk menjalankan solver di thread lain, beberapa fungsi *draw* untuk masing-masing assets, dan fungsi *run* sebagai loop utama.

Program ini dapat menerima input file baik melalui CLI maupun melalui GUI, keduanya diproses dengan mekanisme yang serupa. Untuk penggunaan melalui CLI, program dapat dijalankan melalui src/main.py dan jika ingin melalui GUI dapat dijalankan melalui src/gui.py. Untuk file solusi akan disimpan di luar kedua folder, yaitu di folder test/output.

BAB 3

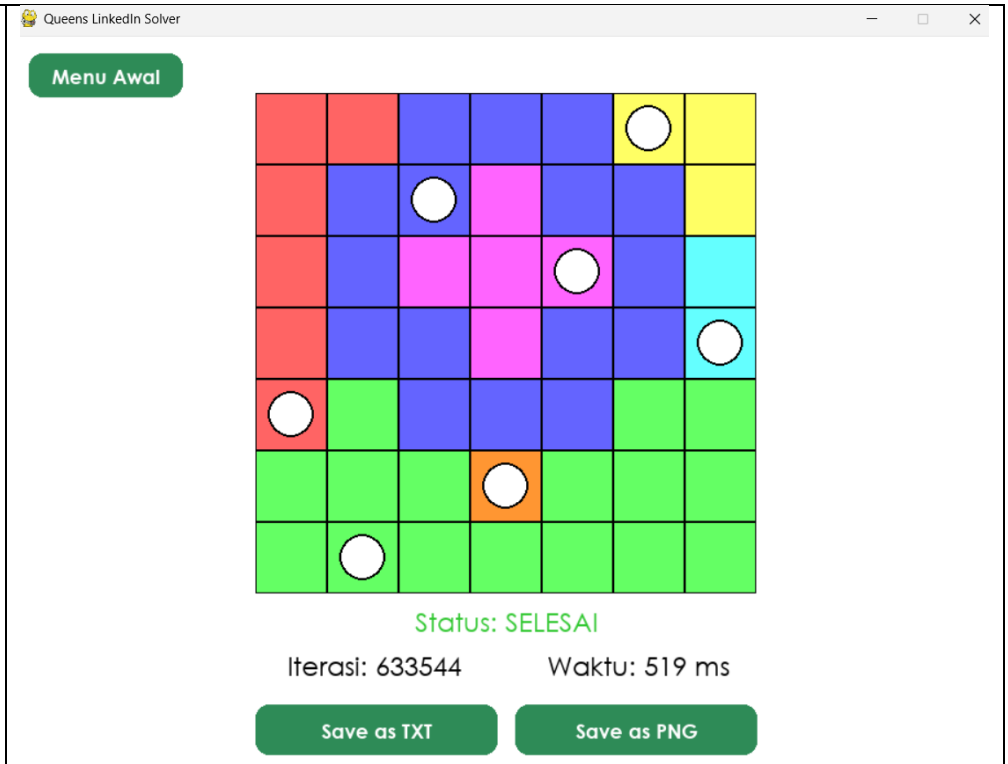
TEST CASE

Test Case 1

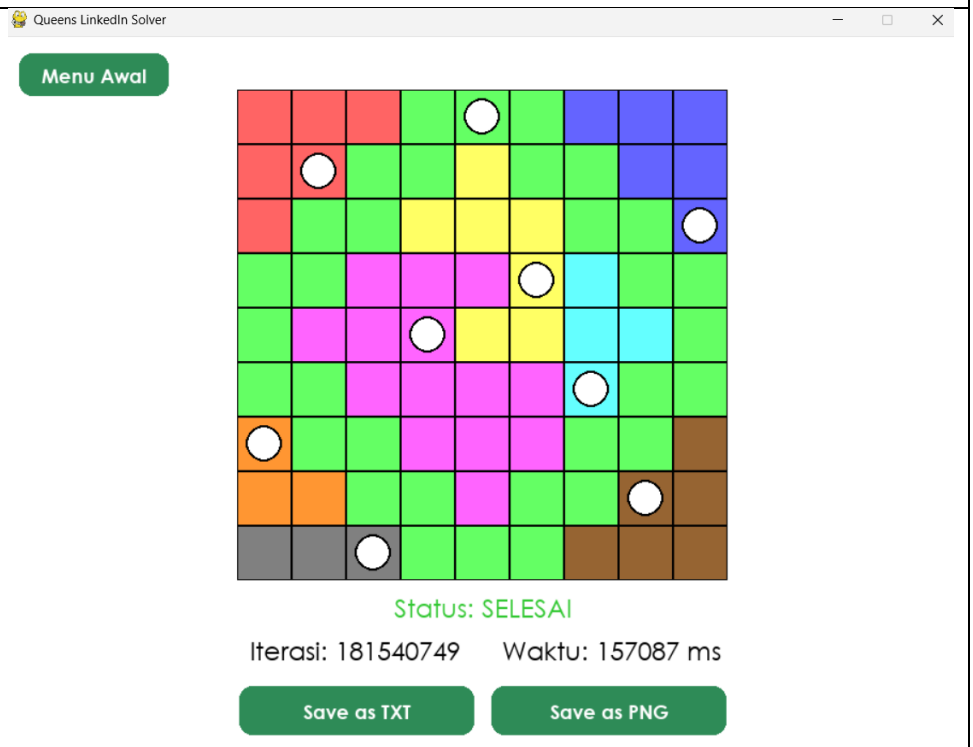
Input	Output
AAABBCCCD ABBBBCECD ABBBDCECD AAABDCCCD BBBBDDDDD FGGGDDHDD FGIGDDHDD FGIGDDHDD FGGGDDHHH (sebagai .txt)	

Test Case 2

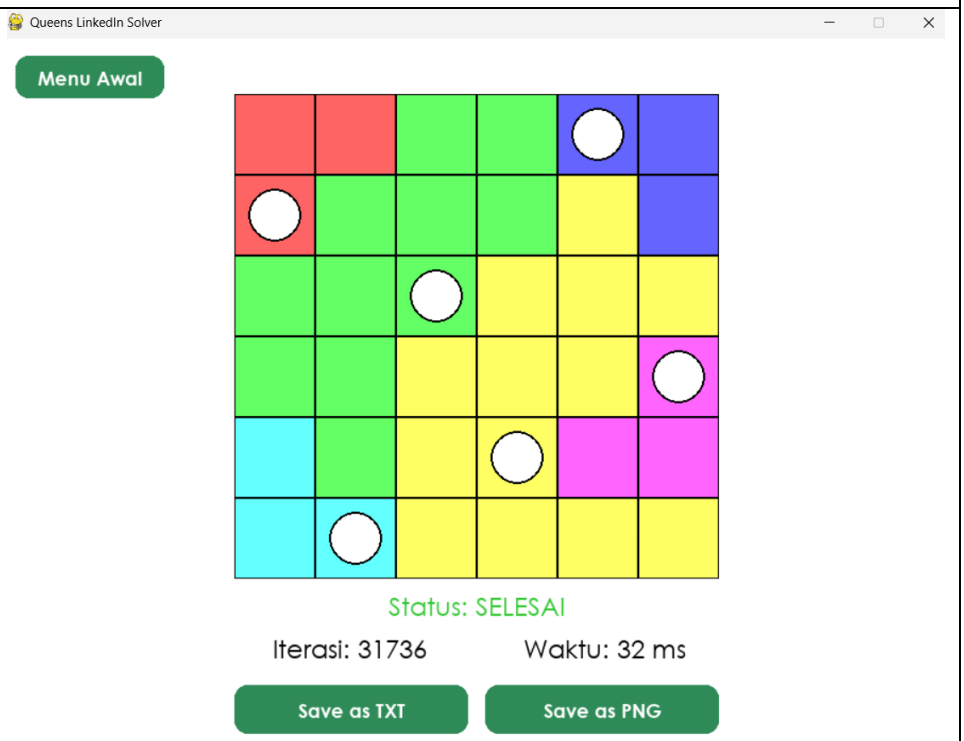
Input	Output

AACCCDD ACCECCD ACEEECF ACCECCF ABCCCBB BBBGBBB BBBBBBB (sebagai .txt)	

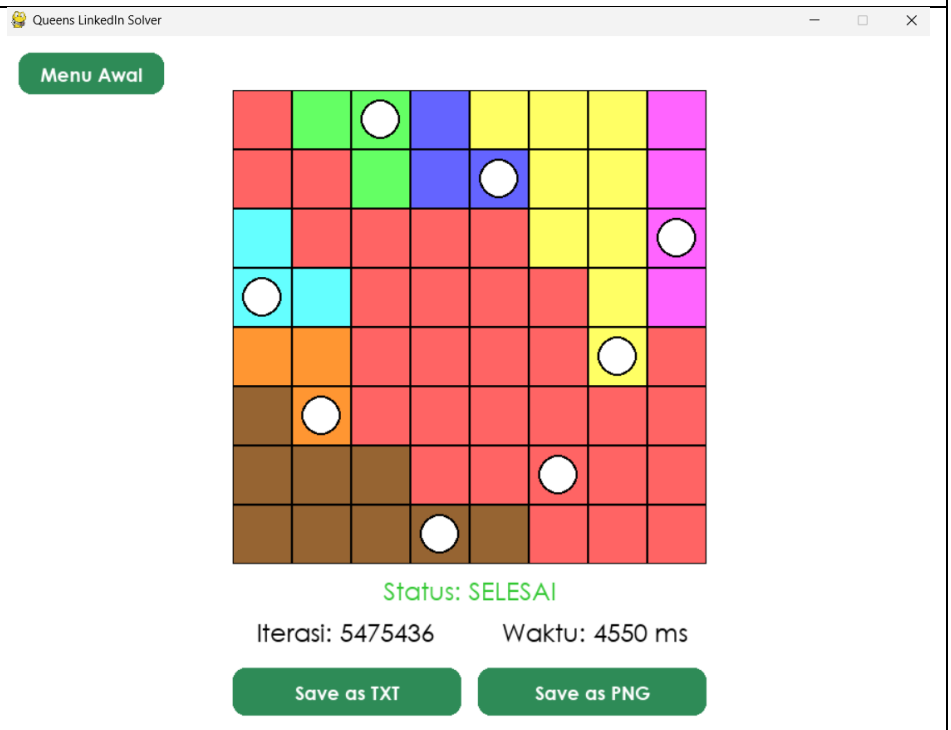
Test Case 3

Input	Output
AAABBBCCC AABBDBBCC ABBDDDBBC BBEEEDFBB BEEEDDFFB BBEEEEFBB GBBEEEBBH GGBBEBBHH IIIBBBHHH (sebagai .txt)	

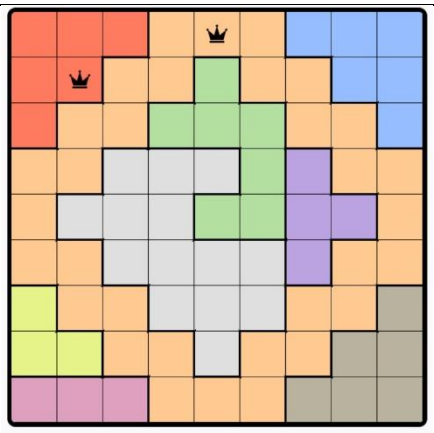
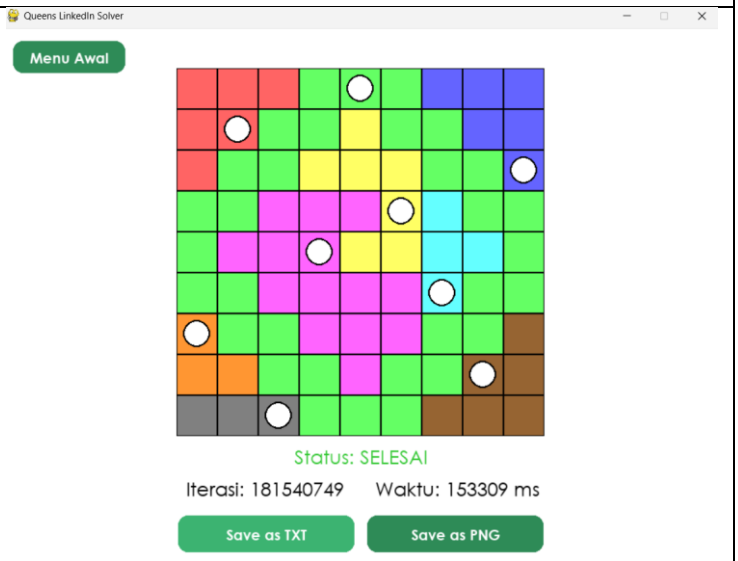
Test Case 4

Input	Output
AABBCC ABBBDC BBBDDD BBDDDE FBDDEE FFDDDD (sebagai .txt)	

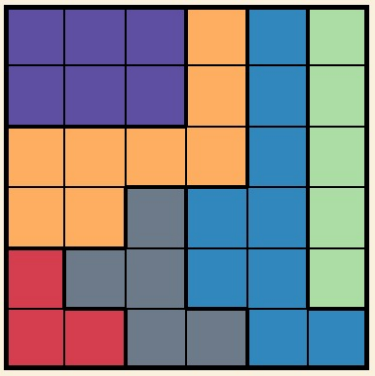
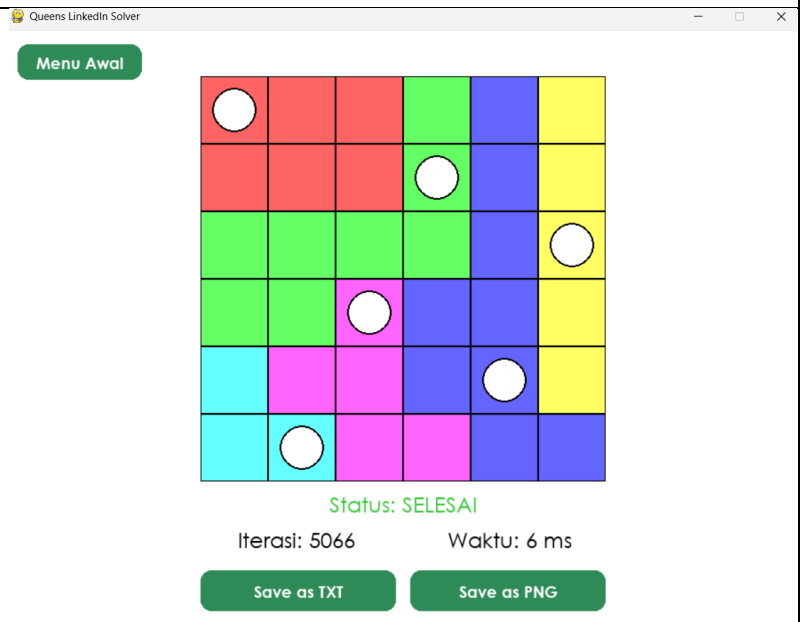
Test Case 5

Input	Output
ABBCDDDE AABCCDDE FAAAADDE FFAAAADE GGAAAADA HGAAAAAA HHHAAAAA HHHHHAAA (sebagai .txt)	

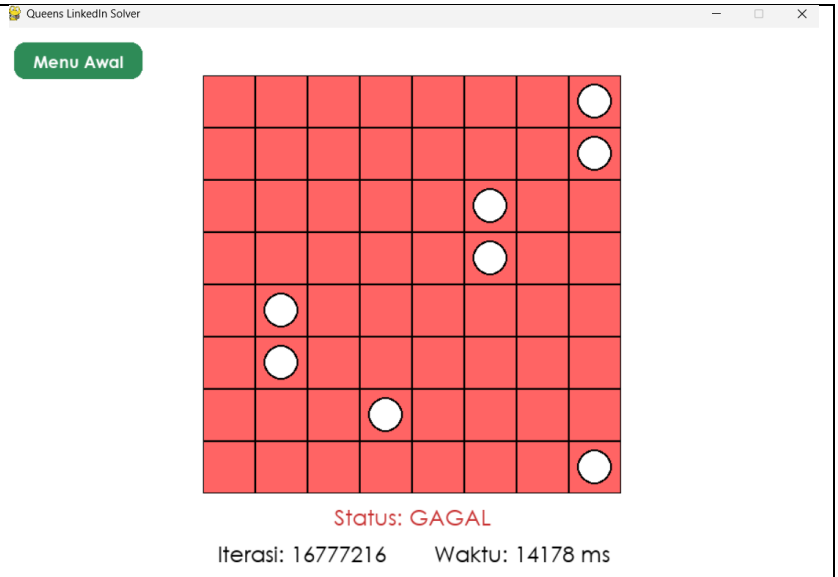
Test Case 6

Input	Output
	

Test Case 7

Input	Output
	

Test Case 8

AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA (sebagai .txt)	

LAMPIRAN

Pernyataan Tidak Melakukan Kecurangan

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

Amanda Aurellia Salsabilla

Link Repository Github

https://github.com/bvzooka/Tucil1_13524131.git

Tabel Checklist

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	✓	