Laporan Tugas Kecil 1 Strategi Algoritma

NATANAEL I MANURUNG (13524021)

A. Representasi data

Input program diterima berupa file text yang berisikan alfabet A-Z disusun secara demikian. Input akan dicek dengan aturan:
Untuk input berisikan n warna (atau huruf di kasus ini) maka harus ada n x n karakter. Kesalahan input akan mengakibatkan program tidak berjalan.
Jika input benar, nantinya akan dikemas program ke dalam bentuk dictionary dengan karakter A-Z sebagai keynya, dengan value yang associated berupa *list of tuples* dimana tuple merepresentasikan koordinat (x,y) dari karakter tersebut.

B. Algoritma Brute Force

Strategi brute force yang digunakan adalah exhaustive search. Exhaustive search merujuk bagaimana sebuah algoritma dalam pencarian sebuah solusi melakukan pengecekan semua kemungkinan/hasil yang mungkin, lalu melakukan pengecekan terhadap setiap kemungkinan untuk menemukan kemungkinan kemungkinan yang benar. Implementasinya di dalam projek ini secara dasar memiliki langkah langkah sedemikian:

1. Generasi permutasi
2. Cek kesesuaian permutasi dengan aturan *Queens*
3. Jika sesuai, masukkan ke dalam list solusi

Sesuai langkah langkah berikut, ada 2 langkah utama yang dapat dijelaskan lebih dalam lagi

A1. Generasi permutasi

```
void search() {
    if (permutation.size() == n) {
        // process permutation
    } else {
        for (int i = 0; i < n; i++) {
            if (chosen[i]) continue;
            chosen[i] = true;
            permutation.push_back(i);
            search();
            chosen[i] = false;
            permutation.pop_back();
        }
    }
}
```

Source: https://cses.fi/book/book.pdf
Kode diatas mengenerasi permutasi dari angka 0 - n. Dengan dasar yang sama dapat dilakukan generasi permutasi atas koordinat untuk per warnanya.
Contoh untuk 3 warna (merah, kuning, hijau):

1. Pilih salah satu koordinat warna merah
2. Pilih salah satu koordinat warna kuning
3. Pilih salah satu koordinat warna hijau
4. Didapat satu permutasi, lakukan pengecekan terhadap permutasi (dibahas nanti)
5. Lanjutkan hingga semua option hijau habis
6. Pilih koordinat kuning lainnya
7. Pilih koordinat hijau lainnya
8. Ulangi sampai semua kombinasi kuning dan hijau habis, dimana setiap full permutasi dicapai dilakukan pengecekan terhadap permutasi
9. Pilih koordinat merah lainnya
10. Pilih koordinat kuning lainnya
11. Pilih koordinat hijau lainnya
12. Ulangi sampai semua permutasi dicapai dan dicek

Dengan langkah langkah dan prinsip tersebut, didapatlah kode berikut:

```
def solve():
    solution = []

    def solve2():
        temp = len(solution)
        if temp == len(colors):
            check(solution.copy())
        else:
            for i in coords[colors[temp]]:
                solution.append(i)
                solve2()
                solution.pop()
```

A2. Pengecekan permutasi

```
rows = set()
cols = set()
valid = True
# check per baris dan kolom
for x, y in case:
    if x in cols or y in rows:
        valid = False
        break
    cols.add(x)
    rows.add(y)
# check radius 1 (8 arah sekitar)
for i in range(len(case)):
    x1, y1 = case[i]
    for j in range(i+1, len(case)):
        x2, y2 = case[j]
        if abs(x1 - x2) <= 1 and abs(y1 - y2) <= 1:
            valid = False
            break
    if not valid:
        break
```

Pengecekan permutasi dilakukan dengan penambahan setiap titik x dan y dari permutasi yang dicek kedalam sebuah set, memastikan sebelum setiap penambahan adanya proses pengecekan apakah nilai tersebut sudah ada di set rows atau cols (jika x di rows, dan jika y di cols). Jika ada maka aturan *Queens* sudah tidak terpenuhi dan solusi permutasi dinyatakan salah.

Setelah itu dilakukan pengecekan terhadap adakah jarak radius 1 dari tiap titik solusi. Pengecekan ini dilakukan dengan mengecek jarak nilai x dan jarak nilai y setiap titik tidak lebih dari 1, dimulai dari titik pertama terhadap semua titik di depannya, titik kedua dan semua titik di depannya, dan berlanjut hingga pengecekan titik n-1 dan titik n.

C. Source Code

```
import time
import tkinter as tk
from tkinter import filedialog, scrolledtext
from collections import defaultdict
from PIL import Image, ImageDraw

# Globals
matrice = []
coords = defaultdict(list)
colors = []
ans = []
iter_count = 0
current_index = 0
cell = 40
visualization_enabled = False
visualization_delay = 10

# ALGORITMA UTAMA
def solve():
    solution = []

    def solve2():
        temp = len(solution)
        if temp == len(colors):
            check(solution.copy())
        else:
            for i in coords[colors[temp]]:
                solution.append(i)
                solve2()
                solution.pop()
    solve2()
def check(case):
    global iter_count, ans

    if visualization_enabled:
        render_current_check(case, is_checking=True)

    rows = set()
```

```
    cols = set()
    valid = True
    for x, y in case:
        if x in cols or y in rows:
            valid = False
            break
        cols.add(x)
        rows.add(y)
    for i in range(len(case)):
        x1, y1 = case[i]
        for j in range(i+1, len(case)):
            x2, y2 = case[j]
            if abs(x1 - x2) <= 1 and abs(y1 - y2) <= 1:
                valid = False
                break
        if not valid:
            break
    if visualization_enabled:
        render_current_check(case, is_checking=False,
is_valid=valid)
        root.update()
        time.sleep(visualization_delay / 1000.0)

    if valid:
        ans.append(case.copy())

    if visualization_enabled or iter_count % 500 == 0:
        output_text.insert(tk.END, f"Checking iteration
#{iter_count}\n")
        output_text.see(tk.END)
        if not visualization_enabled:
            root.update()

    iter_count += 1

def create_text_output(solution):
    new_board = [list(row) for row in matrice]
    for x, y in solution:
        new_board[y][x] = '#'
    return "\n".join("".join(row) for row in new_board)
```

```
def generate_color_map():
    palette = [
        "#FF0000","#0000FF","#008000","#FFFF00","#FFA500","#800080",
        "#00FFFF","#FF00FF","#FFC0CB","#A52A2A","#00FF00","#808080",
        "#8B0000","#00008B","#006400","#FFD700","#FF4500","#4B0082",
        "#40E0D0","#C71585","#FF69B4","#8B4513","#32CD32","#2F4F4F",
        "#DC143C","#1E90FF"
    ]
    letters = sorted(set("".join(matrice)))
    cmap = {}
    for i, letter in enumerate(letters):
        cmap[letter] = palette[i % len(palette)]
    return cmap

def render_current_check(case, is_checking=False, is_valid=False):
    canvas.delete("all")
    size = len(matrice)
    for y in range(size):
        for x in range(size):
            letter = matrice[y][x]
            color = color_map[letter]
            canvas.create_rectangle(
                x*cell, y*cell,
                (x+1)*cell, (y+1)*cell,
                fill=color
            )

    # Yellow Lagi ngecek | Hijau Valid | Merah Gagal
    if is_checking:
        queen_color = "#FFFF00"
        status_text = "Checking..."
    elif is_valid:
        queen_color = "#00FF00"
        status_text = "VALID"
    else:
        queen_color = "#FF0000"
        status_text = "INVALID"

    for x, y in case:
```

```
        canvas.create_oval(
            x*cell+10, y*cell+10,
            x*cell+30, y*cell+30,
            fill=queen_color,
            outline="black",
            width=2
        )
    root.title(f"Queens Solver - Iteration #{iter_count} -
{status_text}")


def render_solution():
    global current_index
    if not ans:
        return
    canvas.delete("all")
    size = len(matrice)
    solution = ans[current_index]
    for y in range(size):
        for x in range(size):
            letter = matrice[y][x]
            color = color_map[letter]

            canvas.create_rectangle(
                x*cell, y*cell,
                (x+1)*cell, (y+1)*cell,
                fill=color
            )
    for x, y in solution:
        canvas.create_oval(
            x*cell+10, y*cell+10,
            x*cell+30, y*cell+30,
            fill="black"
        )
    root.title(f"Solution {current_index+1}/{len(ans)}")

def next_solution():
    global current_index
    if not ans:
        return
```

```
    current_index = (current_index + 1) % len(ans)
    render_solution()

def choose_file():
    global matrice, coords, colors, ans, iter_count, current_index,
color_map
    filename = filedialog.askopenfilename(filetypes=[("Text
Files","*.txt")])
    if not filename:
        return
    output_text.delete("1.0", tk.END)
    try:
        with open(filename, "r") as f:
            matrice = [line.strip() for line in f if line.strip() !=
""]
    except:
        output_text.insert(tk.END, "ERROR: File cannot be
opened.\n")
        return
    n = len(matrice)
    if n == 0:
        output_text.insert(tk.END, "ERROR: Empty input.\n")
        return
    for row in matrice:
        if len(row) != n:
            output_text.insert(tk.END, "ERROR: Invalid board (Must
be NxN).\n")
            return
    coords = defaultdict(list)
    for y in range(n):
        for x in range(n):
            coords[matrice[y][x]].append((x, y))
    colors = list(coords.keys())
    if len(colors) != n:
        output_text.insert(tk.END, "ERROR: Too many colours
(>n).\n")
        return
    ans = []
    iter_count = 0
    current_index = 0
```

```
    color_map = generate_color_map()

    output_text.insert(tk.END, "Solving...\n")
    start = time.time()
    solve()
    end = time.time()
    output_text.insert(tk.END,
"-------------------------------------\n")
    output_text.insert(tk.END, f"Valid solutions: {len(ans)}\n")
    output_text.insert(tk.END, f"Iterations: {iter_count}\n")
    output_text.insert(tk.END, f"Time: {(end-start)*1000:.2f}
ms\n\n")
    if len(ans) == 0:
        output_text.insert(tk.END, "No valid solutions.\n")
        return
    render_solution()

def save_results():
    if not ans:
        output_text.insert(tk.END, "Tidak ada solusi untuk
disimpan.\n")
        return
    filename = filedialog.asksaveasfilename(
        defaultextension=".txt",
        filetypes=[("Text Files","*.txt")]
    )
    if not filename:
        return
    try:
        with open(filename, "w") as f:
            for i, solution in enumerate(ans):
                f.write(f"--- Solution {i+1} ---\n")
                f.write(create_text_output(solution))
                f.write("\n\n")
        output_text.insert(tk.END, "Results saved successfully.\n")
    except:
        output_text.insert(tk.END, "ERROR: Results failed to be
saved.\n")
```

```
def export_images():
    if not ans:
        output_text.insert(tk.END, "No solutions to export.\n")
        return

    filename = filedialog.asksaveasfilename(
        defaultextension=".png",
        filetypes=[("PNG Image", "*.png"), ("JPEG Image", "*.jpg")]
    )
    if not filename:
        return

    try:
        n = len(matrice)
        board_size = n * cell
        padding = 20
        label_height = 30
        num_solutions = len(ans)
        cols = min(4, num_solutions)
        rows = (num_solutions + cols - 1) // cols

        img_width = cols * board_size + (cols + 1) * padding
        img_height = rows * (board_size + label_height) + (rows + 1)
* padding
        composite = Image.new('RGB', (img_width, img_height),
'white')
        draw = ImageDraw.Draw(composite)

        for idx, solution in enumerate(ans):
            row = idx // cols
            col = idx % cols
            x_offset = col * board_size + (col + 1) * padding
            y_offset = row * (board_size + label_height) + (row + 1)
* padding
            for y in range(n):
                for x in range(n):
                    letter = matrice[y][x]
                    color_hex = color_map[letter]
                    color_rgb = tuple(int(color_hex[i:i+2], 16) for
i in (1, 3, 5)) # HEX to RGB
```

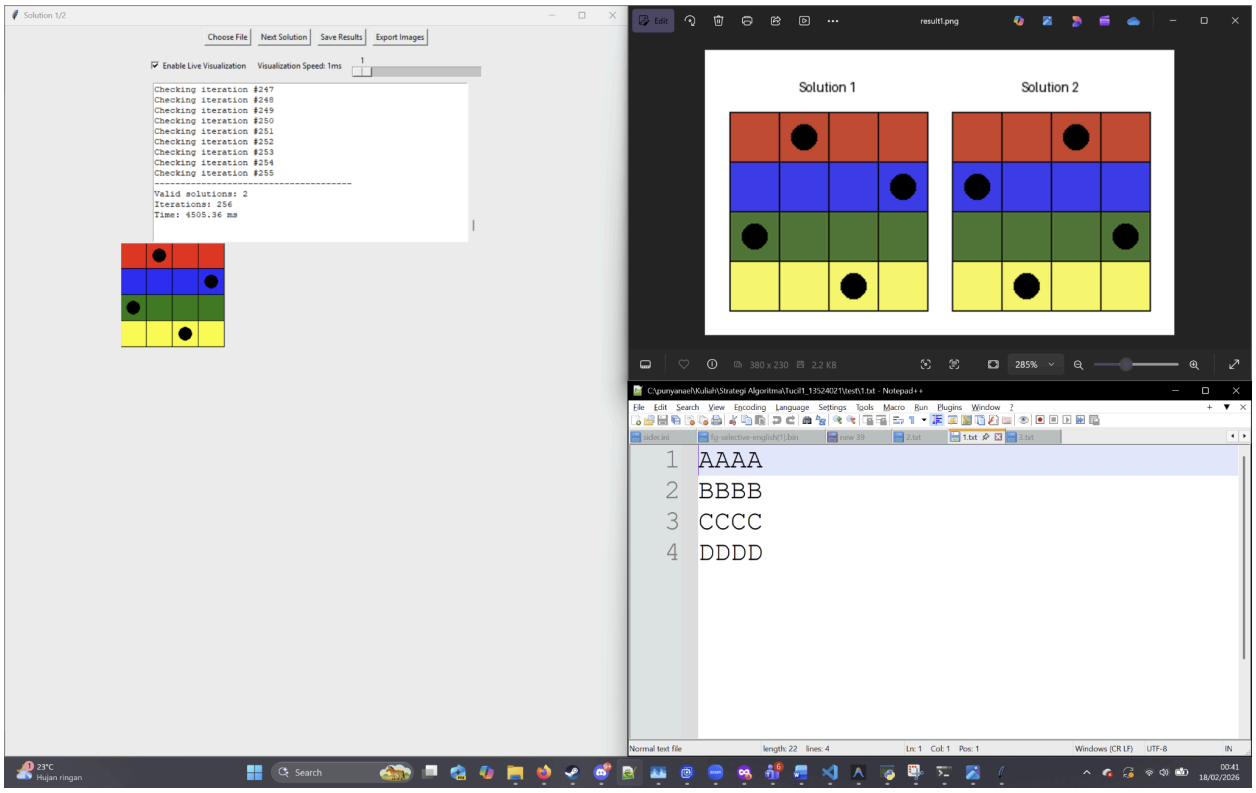
```
                    draw.rectangle(
                        [x_offset + x*cell, y_offset + label_height
+ y*cell,
                         x_offset + (x+1)*cell, y_offset +
label_height + (y+1)*cell],
                        fill=color_rgb,
                        outline='black'
                    )

            #Titik sama label
            for x, y in solution:
                draw.ellipse(
                    [x_offset + x*cell + 10, y_offset + label_height
+ y*cell + 10,
                     x_offset + x*cell + 30, y_offset + label_height
+ y*cell + 30],
                    fill='black',
                    outline='black'
                )
            draw.text(
                (x_offset + board_size // 2, y_offset + 10),
                f"Solution {idx + 1}",
                fill='black',
                anchor='mm'
            )

        composite.save(filename)
        output_text.insert(tk.END, f"Exported {num_solutions}
solution(s) to image successfully.\n")
    except Exception as e:
        output_text.insert(tk.END, f"ERROR: Failed to export images
- {str(e)}\n")




def toggle_visualization():
    global visualization_enabled
    visualization_enabled = viz_var.get()
    if visualization_enabled:
```

```
        output_text.insert(tk.END, "Live visualization enabled\n")
    else:
        output_text.insert(tk.END, "Live visualization disabled\n")
    output_text.see(tk.END)
def update_speed(val):
    global visualization_delay
    visualization_delay = int(float(val))
    speed_label.config(text=f"Visualization Speed:
{visualization_delay}ms")

# Main
root = tk.Tk()
root.title("Flag Placement Solver")
control_frame = tk.Frame(root)
control_frame.pack(pady=5)

btn = tk.Button(control_frame, text="Choose File",
command=choose_file)
btn.pack(side=tk.LEFT, padx=5)
next_btn = tk.Button(control_frame, text="Next Solution",
command=next_solution)
next_btn.pack(side=tk.LEFT, padx=5)
save_btn = tk.Button(control_frame, text="Save Results",
command=save_results)
save_btn.pack(side=tk.LEFT, padx=5)
export_btn = tk.Button(control_frame, text="Export Images",
command=export_images)
export_btn.pack(side=tk.LEFT, padx=5)

viz_frame = tk.Frame(root)
viz_frame.pack(pady=5)
viz_var = tk.BooleanVar(value=False)
viz_check = tk.Checkbutton(
    viz_frame,
    text="Enable Live Visualization",
    variable=viz_var,
    command=toggle_visualization
)
viz_check.pack(side=tk.LEFT, padx=5)
```

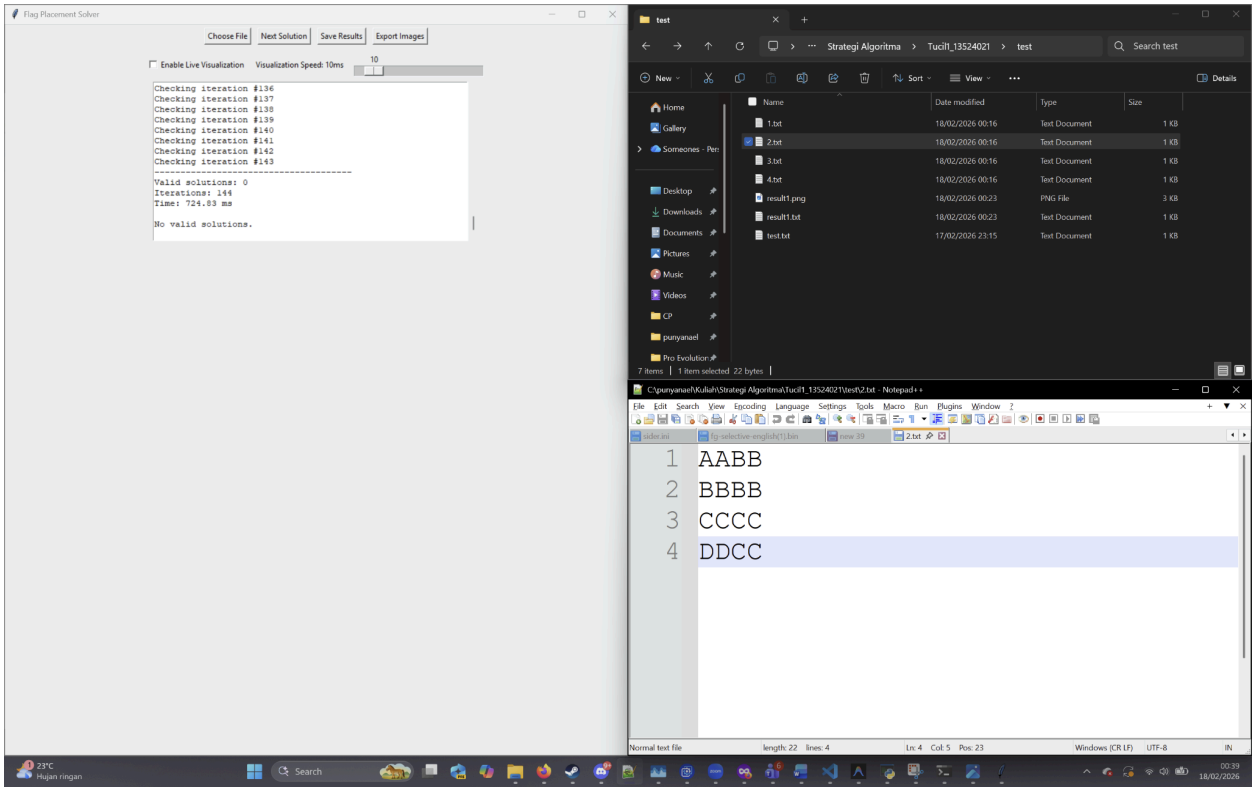
```
speed_label = tk.Label(viz_frame, text=f"Visualization Speed:
{visualization_delay}ms")
speed_label.pack(side=tk.LEFT, padx=5)
speed_slider = tk.Scale(
    viz_frame,
    from_=1,
    to=100,
    orient=tk.HORIZONTAL,
    command=update_speed,
    length=200
)
speed_slider.set(visualization_delay)
speed_slider.pack(side=tk.LEFT, padx=5)
output_text = scrolledtext.ScrolledText(root, width=60, height=15)
output_text.pack()

canvas = tk.Canvas(root, width=600, height=600)
canvas.pack()
root.mainloop()
```

D. Test Cases

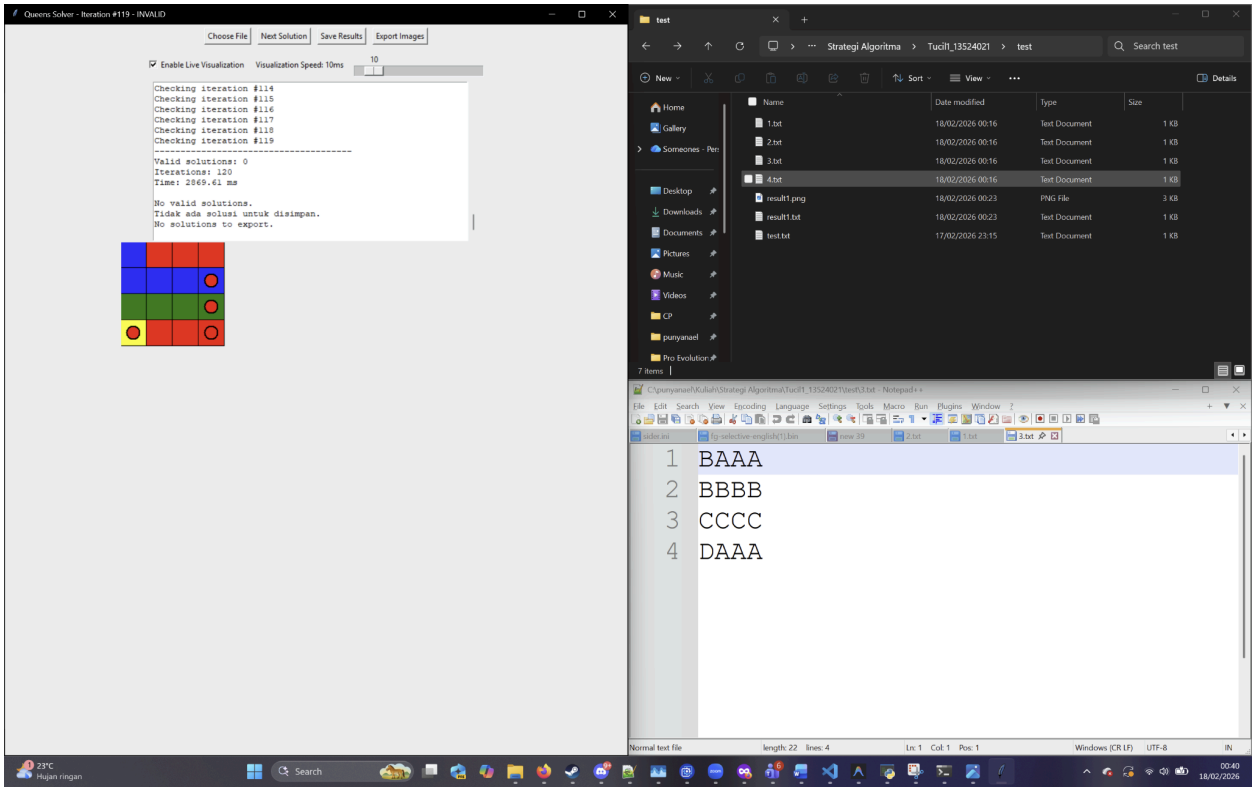
TEST #1



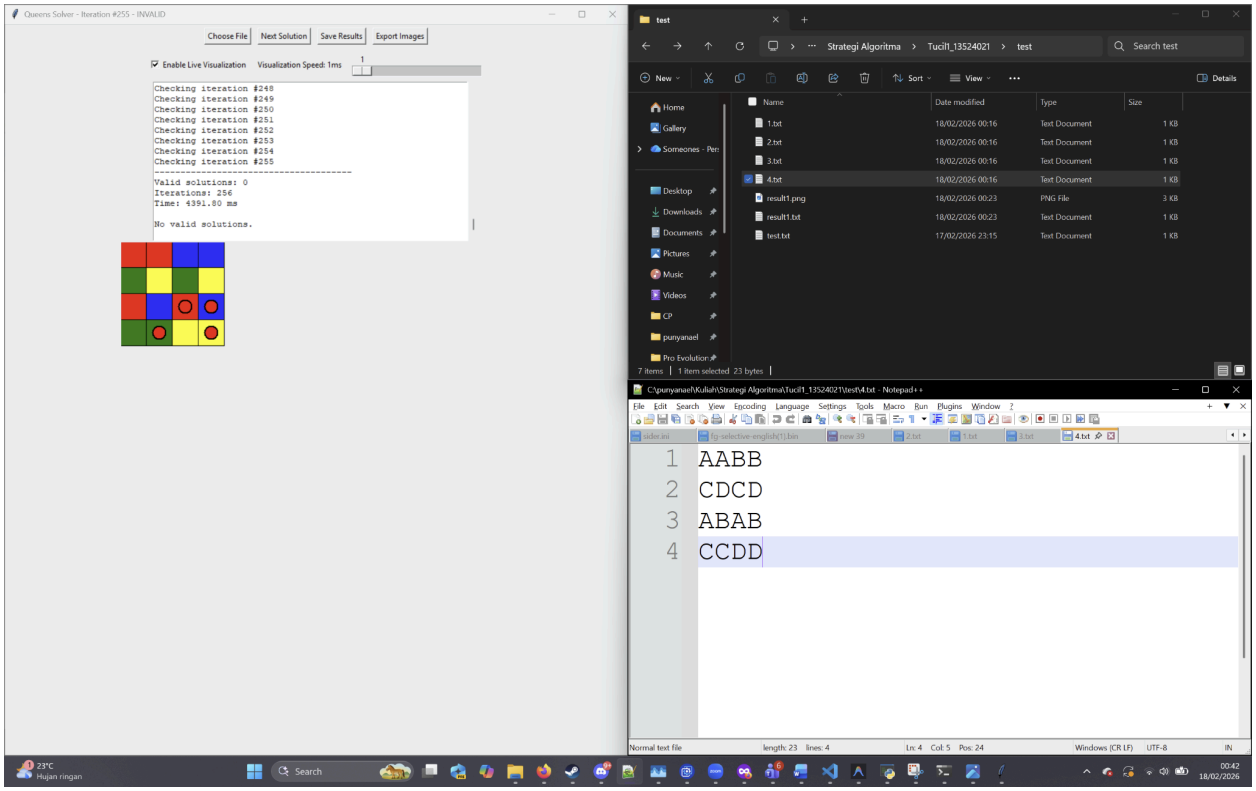
TEST #2


Flag Placement Solver
Choose File
Next Solution
Save Results
Export Images
Enable Live Visualization
Visualization Speed: 10ms
Checking iteration #136
Checking iteration #137
Checking iteration #138
Checking iteration #139
Checking iteration #140
Checking iteration #141
Checking iteration #142
Checking iteration #143
Valid solutions: 0
Iterations: 144
Time: 724.83 ms
No valid solutions.
Strategi Algoritma › Tucil1_13524021 › test
1.txt
2.txt
3.txt
4.txt
result1.png
result1.txt
test.txt
AABB
BBBB
CCCC
DDCC

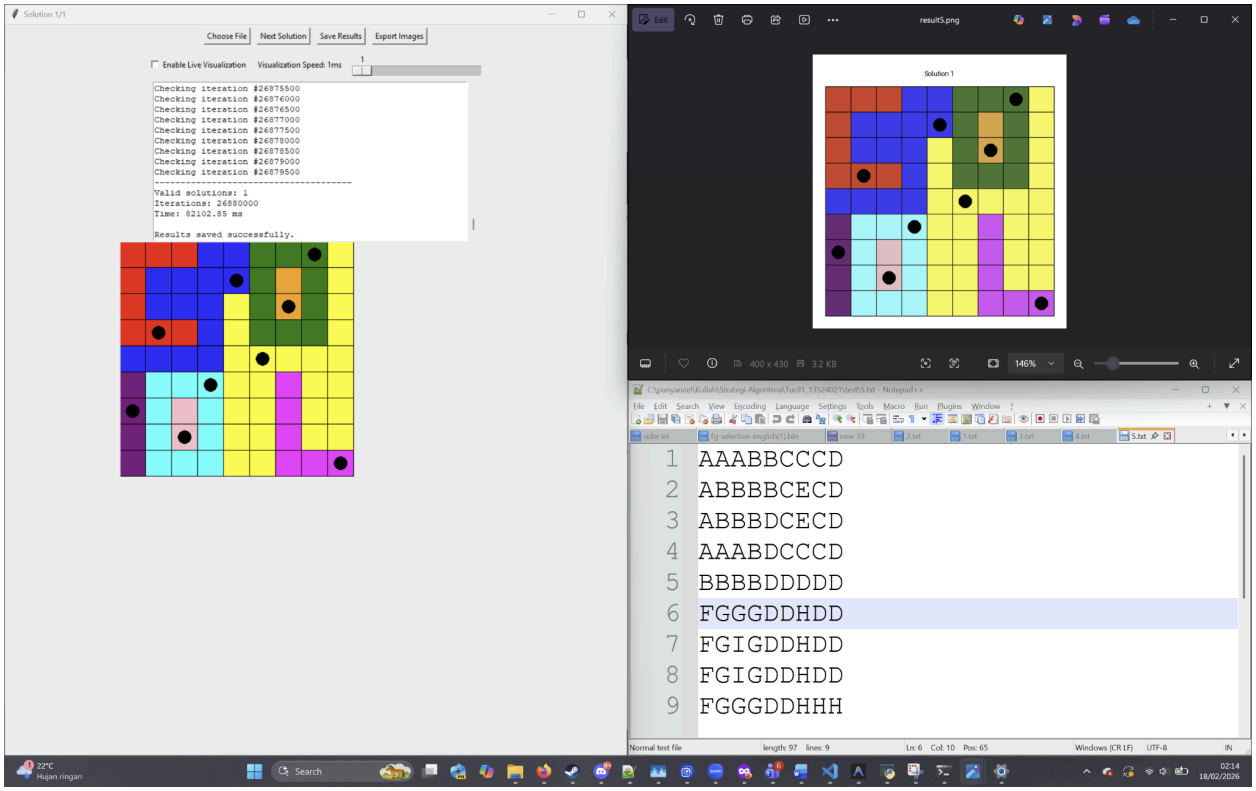
TEST #3


Queens Solver - Iteration #119 - INVALID
Choose File
Next Solution
Save Results
Export Images
Enable Live Visualization
Visualization Speed: 10ms
Checking iteration #114
Checking iteration #115
Checking iteration #116
Checking iteration #117
Checking iteration #118
Checking iteration #119
Valid solutions: 0
Iterations: 120
Time: 2869.61 ms
No valid solutions.
Tidak ada solusi untuk disimpan.
No solutions to export.
Strategi Algoritma › Tucil1_13524021 › test
1.txt
2.txt
3.txt
4.txt
result1.png
result1.txt
test.txt
BAAA
BBBB
CCCC
DAAA

TEST #4


Choose File
Next Solution
Save Results
Export Images
Enable Live Visualization
Visualization Speed: 1ms
Checking iteration #248
Checking iteration #249
Checking iteration #250
Checking iteration #251
Checking iteration #252
Checking iteration #253
Checking iteration #254
Checking iteration #255
Valid solutions: 0
Iterations: 256
Time: 4391.80 ms
No valid solutions.
AABB
CDCD
ABAB
CCDD

TEST #5


Choose File
Next Solution
Save Results
Export Images
Enable Live Visualization
Visualization Speed: 1ms
Checking iteration #26875500
Checking iteration #26876000
Checking iteration #26876500
Checking iteration #26877000
Checking iteration #26877500
Checking iteration #26878000
Checking iteration #26878500
Checking iteration #26879000
Checking iteration #26879500
Valid solutions: 1
Iterations: 26880000
Time: 82102.85 ms
Results saved successfully.
Solution 1
AAABBCCCD
ABBBBCECD
ABBBDCECD
AAABDCCCD
BBBBDDDDD
FGGGDDHDD
FGIGDDHDD
FGIGDDHDD
FGGGDDHHH

E. Repo

Link: https://github.com/SomeonesDads/Tucil1_13524021

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

Natanael I. Manurung