Laporan Tugas Kecil 1
IF2211 Strategi Algoritma: Penyelesaian Permainan *Queens* Linkedin

---

I. **Algoritma *Brute Force***

Algoritma *Brute Force* adalah algoritma strategi solusi langsung yang menyelesaikan persoalan secara lempeng. Algoritma ini bersifat sederhana, langsung, dan jelas caranya atau langkah-langkahnya mudah dipahami secara intuitif. Algoritma ini didasarkan pada pernyataan di dalam persoalan dan definisi/konsep yang dilibatkan dalam persoalan tersebut. Algoritma ini sering ditemui pada beberapa kasus berikut, mencari elemen terbesar/terkecil pada senarai, pencarian beruntun, dan masih banyak lagi.

Algoritma *Brute Force* juga dapat diimplementasikan pada game *Queens* Linkedin. Game ini termasuk jenis *board game* yang memiliki peraturan, antara lain:
- *Board* berukuran *NxN* berbentuk persegi.
- Warna pada *board* berjumlah N dengan saling terkoneksi antar kotaknya.
- Pada setiap baris, kolom, dan warna hanya ada satu *queen*.
- Serta, *queen* tidak boleh bertetangga (kotak bersentuhan) dengan *queen* lain baik horizontal, vertikal maupun diagonal.

Langkah yang saya ambil dalam penyelesaian game ini dengan algoritma *Brute Force* murni tanpa backtracking. Hal ini mengakibatkan konsumsi waktu yang sangat lama untuk board berukuran besar.

Algoritma *Brute Force* pada *Queens*:
1. Kelompokkan titik-titik yang berkorespondensi pada setiap warnanya.
2. Ambil kemungkinan pertama yaitu titik paling awal setiap warna yang terdaftar dalam kelompok tersebut.
3. Validasikan titik-titik pada kelompok tersebut sesuai peraturan yang berlaku.
4. Jika ada yang belum memenuhi, ambil kemungkinan kedua yaitu titik kedua (*Increment*) warna terakhir yang terdaftar dalam kelompok tersebut.
5. Lakukan langkah 3-4, sampai warna terakhir sudah mencapai titik terakhir lakukan langkah 4 pada warna sebelum terakhir dan reset warna terakhir menjadi titik pertama.
6. Lakukan terus menerus langkah 3-4-5, sampai validasi pada langkah ke-3 berhasil. Maka itu adalah kombinasi titik-titik pada setiap warna yang merupakan solusi jawabannya.

Pseudocode algoritma *Brute Force* pada *Queens*:

```
function validate_points_pure(combinations)-> boolean
{Validasikan titik-titik setiap i pada senarai combinations
sesuai peraturan}
Kamus Lokal
i, j: integer
q1, q2: point
Algoritma
i traversal[0..length(combinations)]
    j traversal[i+1 .. length(combinations)]
       q1 <- combinations[i]
       q2 <- combinations[j]
       if (abs(q1[0]-q2[0]) == 1 && abs(q1[1] - q2[1] == 1))
then {Memastikan tidak ada yang bertetangga (bersentuhan kotak)
antar titik satu sama lain dalam peranai titik}
          -> False
-> True
```

```
function play_pure(board: dictionary, n: integer) -> tuple
{ Mencari solusi penempatan Queen menggunakan pendekatan Brute
Force berbasis kelompok warna }
Kamus Lokal
   color_groups: dictionary { key: warna, value: list of point
}
   colors: list of string
   color_cell_lists: list of list of point
   combinations: list of integer { indeks sel yang dipilih
untuk tiap warna }
   solutions: list of point
   i, j, cnt: integer
   found: boolean
Algoritma
   { Pre-processing: Kelompokkan sel berdasarkan warna }
   color_groups <- group board by color
   colors <- sort(keys(color_groups))
   if (length(colors) != n) then -> null

   color_cell_lists <- list of points per color
   combinations <- [0, 0, ..., 0] { inisialisasi n elemen
dengan nilai 0 }
   cnt <- 0
   found <- True

   while (found) do
       cnt <- cnt + 1
```

```
      { Ambil titik koordinat berdasarkan indeks kombinasi saat
ini }
      solutions <- [color_cell_lists[i][combinations[i]] for i
in 0..n-1]

      { Validasi Baris, Kolom, dan Tetangga }
      if (unique(rows in solutions) and unique(cols in
solutions)) then
          if validate_position_pure(solutions) then
             ->(solutions, cnt)

      { Proses Increment / Mencari Kombinasi Berikutnya }
      i <- n - 1
      while (i >= 0 and combinations[i] ==
length(color_cell_lists[i]) - 1) do
          i <- i - 1

      if (i < 0) then
          found <- False { Semua kemungkinan habis }
      else
          combinations[i] <- combinations[i] + 1
          J traversal [i+1..n]
             combinations[j] <- 0

   -> null
```

II. *Source* program dalam Python

    Source program yang saya buat dalam bahasa Python dari opsi-opsi bahasa pemrograman lainnya. Model pengerjaan yang saya gunakan dengan struktur *model*, *view*, dan *control* (MVC). Setiap komponen model tersebut direpresentasikan pada file-file, yaitu *model* pada file `logic.py`, *view* dan *control* pada file `app.py`, serta fungsi pembantu pada file `utils.py`. Struktur folder yang saya gunakan sebagai berikut:

```
  Tucil1_13524075/
├── doc/              # Laporan Tugas Kecil
├── src/              # Source Code program
│   ├── app.py        # Main GUI Entry Point (Flet)
│   ├── logic.py      # Implementasi Algoritma
│   ├── utils.py      # Fungsi utilitas & validasi
│   └── image_process.py # Pengolahan Gambar (Bonus)
├── test/             # File uji kasus (.txt)
└── README.md
```

Source program lengkap dari setiap file yang digunakan ada pada rincian berikut,
`logic.py`

```python
import time ,  asyncio

def validate_position_pure(combinations):
    for i in range(len(combinations)):
        for j in range(i + 1, len(combinations)):
            q1, q2 = combinations[i], combinations[j]
            if abs(q1[0] - q2[0]) == 1 and abs(q1[1] - q2[1]) == 1:
                return False
    return True

async def play_pure(board, n, page, render_board):
    start = time.process_time()
    cnt = 0

    board_list = []
    for y in range(n):
        row = ""
        for x in range(n):
            row += board[(x, y)]
        board_list.append(row)

    color_groups = {}
    for pos, color in board.items():
        if color not in color_groups:
            color_groups[color] = []
        color_groups[color].append(pos)

    colors = sorted(color_groups.keys())

    if len(colors) != n:
        end = time.process_time()
        return None, end - start

    color_cell_lists = [color_groups[c] for c in colors]
```

```python
    group_sizes = [len(cells) for cells in color_cell_lists]


    combinations = [0 for _ in range(n)]


    found = True
    while found:
        cnt += 1

        solutions = [color_cell_lists[i][combinations[i]] for i in
range(n)]


        rows = [q[1] for q in solutions]
        if len(set(rows)) == n:
            cols = [q[0] for q in solutions]
            if len(set(cols)) == n:
                if validate_position_pure(solutions):
                    end = time.process_time()
                    render_board(board_list, solutions)
                    await asyncio.sleep(0)
                    return solutions, end - start, cnt

        if cnt % 1000 == 0:
            render_board(board_list, solutions)
            await asyncio.sleep(0)


        i = n - 1
        while i >= 0 and combinations[i] == group_sizes[i] - 1:
            i -= 1

        if i < 0:
            found = False
        else:
            combinations[i] += 1
            for j in range(i + 1, n):
                combinations[j] = 0
```

```python
        end = time.process_time()
        return None, end - start, cnt

def validate_position_bt(x, y, cc, storageColor, storagePoint,
storageRow,storageCol):
    if x in storageRow or y in storageCol:
        return False
    if cc in storageColor:
        return False
    neighbor = [(x-1, y-1), (x-1, y+1), (x+1, y-1), (x+1, y+1)]
    for n in neighbor:
        if n in storagePoint:
            return False
    return True

async def play_bt(board, n, page, render_board):
    storagePoint = set()
    storageRow =  set()
    storageCol =  set()
    storageColor =  set()
    queenPoints = []
    step = [0]
    board_list = []
    for y in range(n):
        row = ""
        for x in range(n):
            row += board[(x, y)]
        board_list.append(row)
    async def searchPoints(index,count):
            step[0]+=1
            if count == n:
                return True
            if index >= n*n:
                return False
            x = index % n
            y = index // n
            color = board[(x, y)]
            if step[0] % 1000 == 0:
                render_board(board_list, queenPoints)
```

```python
                    await asyncio.sleep(0)
                if validate_position_bt(x, y, color, storageColor,
storagePoint, storageRow, storageCol):
                    storagePoint.add((x, y))
                    storageRow.add(x)
                    storageCol.add(y)
                    storageColor.add(color)
                    queenPoints.append((x, y))

                    if await searchPoints(index + 1, count + 1):
                        return True

                    storagePoint.remove((x, y))
                    storageRow.remove(x)
                    storageCol.remove(y)
                    storageColor.remove(color)
                    queenPoints.pop()


            if await searchPoints(index + 1, count):
                return True

            return False

    start = time.process_time()
    if(await searchPoints(0,0)):
        end = time.process_time()
        return queenPoints, end-start, step[0]
    else:
        end = time.process_time()
        return None, end-start, step[0]
```

utils.py

```python
def is_connected(board):
    n = len(board)
```

```python
    colors = set(char for row in board for char in row)

    for color in colors:
        start_node = None
        color_cells_count = 0
        for r in range(n):
            for c in range(n):
                if board[r][c] == color:
                    color_cells_count += 1
                    if not start_node:
                        start_node = (r, c)

        queue = [start_node]
        seen_in_bfs = {start_node}
        while queue:
            curr_r, curr_c = queue.pop(0)
            for dr, dc in [(0,1), (0,-1), (1,0), (-1,0)]:
                nr, nc = curr_r + dr, curr_c + dc
                if 0 <= nr < n and 0 <= nc < n and \
                    board[nr][nc] == color and (nr, nc) not in seen_in_bfs:
                    seen_in_bfs.add((nr, nc))
                    queue.append((nr, nc))

        if len(seen_in_bfs) != color_cells_count:
            return False, f"Warna {color} terpecah/tidak menyatu!"

    return True, "Semua wilayah terkoneksi dengan baik."

def validate_board(board):
    n = len(board)
    for row in board:
        if len(row) != n:
            return False, "Papan tidak persegi!"

    warna = set(char for row in board for char in row)
    if len(warna) != n:
        return False,f"Jumlah warna ({len(warna)}) tidak sama dengan
ukuran papan {n}!"
```

```python
    valid, pesan = is_connected(board)
    if (not valid):
        return valid, pesan
    return True, 'Papan Valid!'

def file_to_board(path):
    try:
        with open (path) as f:
            return [line.strip() for line in f.readlines() if
line.strip()]
    except:
        print("Gagal membaca file :(")

def convert_dict(board, n):
    points = [(x, y) for x in range(n) for y in range(n) ]
    boardDict = {}
    for x in range(n*n):
        x_coor, y_coor = points[x]
        color = board[y_coor][x_coor]
        boardDict[(x_coor, y_coor)] = color
    return boardDict

def interface(board,solution, n, f):
    for  y in range(n):
        for x in range(n):
            if (x, y) in solution:
                print('#', end='', file=f)
            else:
                print(f'{board[(x,y)]}', end='', file=f)
        print(file=f)
    print(file=f)
```

image_process.py

```python
from PIL import Image, ImageDraw, ImageFont

def generate_color_map(characters):

    base_colors = [
        (239, 83, 80),
        (66, 165, 245),
        (102, 187, 106),
        (156, 39, 176),
        (255, 167, 38),
        (38, 198, 218),
        (236, 64, 122),
        (38, 166, 154),
        (212, 225, 87),
        (255, 193, 7),
        (92, 107, 192),
        (255, 87, 34),
        (156, 204, 101),
        (103, 58, 183),
        (255, 235, 59),
        (79, 195, 247),
        (141, 110, 99),
        (120, 144, 156),
        (229, 115, 115),
        (129, 199, 132),
        (171, 71, 188),
        (255, 183, 77),
        (77, 208, 225),
        (240, 98, 146),
        (77, 182, 172),
        (220, 231, 117),
    ]

    color_map = {}
    for i, char in enumerate(characters):
        color_map[char] = base_colors[i % len(base_colors)]

    return color_map
```

```python
def board_to_image(board_data, solution=None,
output_path="solution.png", cell_size=80, color_map=None):

    n = len(board_data)
    img_size = n * cell_size

    img = Image.new('RGB', (img_size, img_size), 'white')
    draw = ImageDraw.Draw(img)

    if color_map is None:
        unique_chars = sorted(set(''.join(board_data)))
        color_map = generate_color_map(unique_chars)

    try:
        font =
ImageFont.truetype("/usr/share/fonts/truetype/dejavu/DejaVuSans-Bold.ttf
", cell_size // 2)
        label_font =
ImageFont.truetype("/usr/share/fonts/truetype/dejavu/DejaVuSans.ttf",
cell_size // 4)
    except:
        try:
            font = ImageFont.truetype("arial.ttf", cell_size // 2)
            label_font = ImageFont.truetype("arial.ttf", cell_size // 4)
        except:
            font = ImageFont.load_default()
            label_font = ImageFont.load_default()

    for y in range(n):
        for x in range(n):
            char = board_data[y][x]
            color = color_map.get(char, (200, 200, 200))

            x1 = x * cell_size
            y1 = y * cell_size
            x2 = x1 + cell_size
            y2 = y1 + cell_size
```

```python
            draw.rectangle([x1, y1, x2, y2], fill=color,
outline='black', width=2)

            draw.text((x1 + 5, y1 + 5), char, fill='white',
font=label_font)

            if solution and (x, y) in solution:
                text = "Q"

                bbox = draw.textbbox((0, 0), text, font=font)
                text_width = bbox[2] - bbox[0]
                text_height = bbox[3] - bbox[1]

                text_x = x1 + (cell_size - text_width) // 2
                text_y = y1 + (cell_size - text_height) // 2

                draw.text((text_x, text_y), text, fill='white',
font=font)

    img.save(output_path)
    print(f"Image saved: {output_path}")
    return output_path
```

app.py

```python
import flet as ft
import logic, utils, image_process


async def main(page: ft.Page):
    page.title = "Queens Game by Brute Force Algorithm"
    page.theme_mode = ft.ThemeMode.LIGHT
    page.scroll = ft.ScrollMode.ADAPTIVE
    page.window_width = 1000
    page.window_height = 800

    state = {
        "board": None,
```

```python
        "n": 0,
        "boardDict": {},
        "is_running": False,
        "solution": None
    }

    status_text = ft.Text("Pilih file atau masukkan teks board...",
italic=True)
    step_counter = ft.Text("Langkah: 0", weight="bold")
    duration_text = ft.Text("Durasi: 0 ms")

    file_name_input = ft.TextField(label="Nama File", hint_text="Contoh:
board.txt", expand=True)
    manual_input = ft.TextField(label="Atau Paste Board di sini",
multiline=True, min_lines=3)

    pure_bt_switch = ft.Switch(label="Pure Brute Force (Tanpa
Backtrack)", value=True)

    grid_display = ft.Container(
        content=ft.Column(),
        alignment=ft.alignment.Alignment(0, 0),
        padding=20
    )

    def render_board(board_data, solution=None):
        n = len(board_data)
        state["n"] = n
        state["boardDict"] = utils.convert_dict(board_data, n)

        color_map = {
            'A': ft.Colors.RED_400,
            'B': ft.Colors.BLUE_400,
            'C': ft.Colors.GREEN_400,
            'D': ft.Colors.PURPLE_400,
            'E': ft.Colors.ORANGE_400,
            'F': ft.Colors.CYAN_400,
            'G': ft.Colors.PINK_400,
            'H': ft.Colors.TEAL_400,
```

```python
            'I': ft.Colors.LIME_400,
            'J': ft.Colors.AMBER_400,
            'K': ft.Colors.INDIGO_400,
            'L': ft.Colors.DEEP_ORANGE_400,
            'M': ft.Colors.LIGHT_GREEN_400,
            'N': ft.Colors.DEEP_PURPLE_400,
            'O': ft.Colors.YELLOW_400,
            'P': ft.Colors.LIGHT_BLUE_400,
            'Q': ft.Colors.BROWN_400,
            'R': ft.Colors.BLUE_GREY_400,
            'S': ft.Colors.RED_300,
            'T': ft.Colors.GREEN_300,
            'U': ft.Colors.PURPLE_300,
            'V': ft.Colors.ORANGE_300,
            'W': ft.Colors.CYAN_300,
            'X': ft.Colors.PINK_300,
            'Y': ft.Colors.TEAL_300,
            'Z': ft.Colors.LIME_300
        }

        rows = []
        state["grid_cells"] = []

        for y in range(n):
            row_controls = []
            for x in range(n):
                char = board_data[y][x]
                is_queen = solution and (x, y) in solution

                cell = ft.Container(
                    content=ft.Text("Q" if is_queen else "", size=20,
weight="bold", color="white"),
                    bgcolor=color_map[char],
                    width=45, height=45,
                    alignment=ft.alignment.Alignment(0, 0),
                    border=ft.border.all(1, "black12"),
                    border_radius=4
                )
                row_controls.append(cell)
```

```python
            rows.append(ft.Row(controls=row_controls, spacing=5,
alignment=ft.MainAxisAlignment.CENTER))

        grid_display.content = ft.Column(controls=rows, spacing=5,
horizontal_alignment=ft.CrossAxisAlignment.CENTER)
        page.update()

    def load_board(e):
        try:
            if file_name_input.value:
                board = utils.file_to_board(file_name_input.value)
            else:
                board = [line.strip() for line in
manual_input.value.split("\n") if line.strip()]

            valid, msg = utils.validate_board(board)
            status_text.value = msg
            if valid:
                state["board"] = board
                render_board(board)
            page.update()
        except Exception as ex:
            status_text.value = f"Error: {str(ex)}"
            page.update()

    async def start_game(e):
        if not state["board"]:
            status_text.value = "Load board dulu!"
            page.update()
            return

        state["is_running"] = True
        status_text.value = "Sedang mencari solusi..."
        page.update()

        if pure_bt_switch.value:
            answer, duration, step = await
logic.play_pure(state["boardDict"], state["n"], page, render_board)
        else:
```

```python
            answer, duration, step = await
logic.play_bt(state["boardDict"], state["n"], page, render_board)

        if answer is None:
            status_text.value = "Tidak ada jawabannya, berikan board
lain ya :)"
            state["solution"] = None
        else:
            status_text.value = "Solusi Ditemukan!"
            state["solution"] = answer
            render_board(state["board"], answer)

        duration_text.value = f"Permainan berlangsung selama {duration *
1000:.2f} ms"
        step_counter.value = f"Permainan berlangsung dengan banyaknya
konfigurasi: {step}"
        state["is_running"] = False
        page.update()

    def export_to_txt(e):
        if not state['board']:
            status_text.value="Tidak ada board untuk diekspor!"
            page.update()
            return
        try:
            solution = state.get('solution')
            board = state.get('board')
            n = state.get('n')

            import datetime
            timestamp =
datetime.datetime.now().strftime("%Y%m%d_%H%M%S")
            output_path = f"queens_solution_{timestamp}.txt"


            with open(output_path, 'w') as f:
                for row in board:
                    f.write(row + '\n')
```

```python
                f.write('\n')

            if solution:
                f.write("Solution:\n")
                result = [list(row) for row in board]
                for (x, y) in solution:
                    result[y][x] = '#'

                for row in result:
                    f.write(''.join(row) + '\n')
            else:
                f.write("Tidak/Belum ada solusi yang memenuhi. Harap
jalankan programnya terlebih dahulu untuk mengetahui.\n")

        status_text.value = f"File disimpan: {output_path}"
        page.update()
    except Exception as ex:
        status_text.value = f"Error ekspor: {str(ex)}"
        page.update()



    def export_to_image(e):
        if not state["board"]:
            status_text.value = "Tidak ada board untuk diekspor!"
            page.update()
            return

        try:
            solution = state.get('solution')
            color_map = state.get('color_map')

            import datetime
            timestamp =
datetime.datetime.now().strftime("%Y%m%d_%H%M%S")
            output_path = f"queens_solution_{timestamp}.png"

            saved_path = image_process.board_to_image(state["board"],
solution, output_path, color_map=color_map)
```

```python
            status_text.value = f"Gambar disimpan: {saved_path}"
            page.update()

        except Exception as ex:
            status_text.value = f"Error ekspor: {str(ex)}"
            page.update()

    page.add(
        ft.Row([
            ft.Column([
                ft.Text("Queens Visualizer", size=28, weight="bold"),
                grid_display,
            ], expand=2,
horizontal_alignment=ft.CrossAxisAlignment.CENTER),

            ft.Column([
                ft.Card(
                    content=ft.Container(
                        content=ft.Column([
                            ft.Text("Konfigurasi Input", weight="bold"),
                            ft.Row([file_name_input,
ft.IconButton(ft.Icons.REFRESH, on_click=load_board)]),
                            manual_input,
                            ft.ElevatedButton("Load Board",
on_click=load_board, icon=ft.Icons.UPLOAD),

                        ]), padding=15
                    )
                ),
                ft.Card(
                    content=ft.Container(
                        content=ft.Column([
                            ft.Text("Kontrol Algoritma", weight="bold"),
                            pure_bt_switch,
                            ft.Divider(),
                            status_text,
                            duration_text,
                            step_counter,
```

```
                                ft.FilledButton("MULAI PENCARIAN",
icon=ft.Icons.PLAY_ARROW,
                                            on_click=start_game,
width=300, height=50),
                            ft.OutlinedButton("Ekspor ke Gambar",
icon=ft.Icons.DOWNLOAD,
                                            on_click=export_to_image,
width=300),
                            ft.OutlinedButton("Ekspor ke File Txt",
icon=ft.Icons.DOWNLOAD,
                                            on_click=export_to_txt,
width=300),
                        ]), padding=15
                    )
                )
            ], expand=1)
        ], expand=True)
    )

if __name__ == "__main__":
    ft.app(target=main, view=ft.AppView.WEB_BROWSER, port=8550)
```
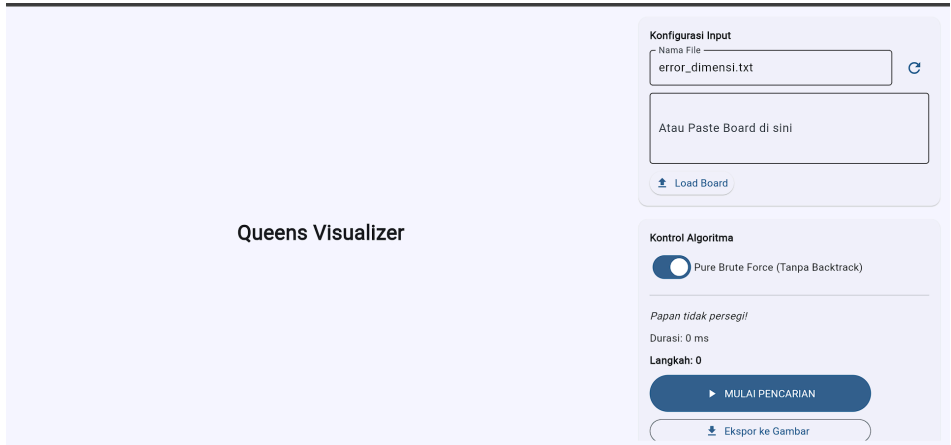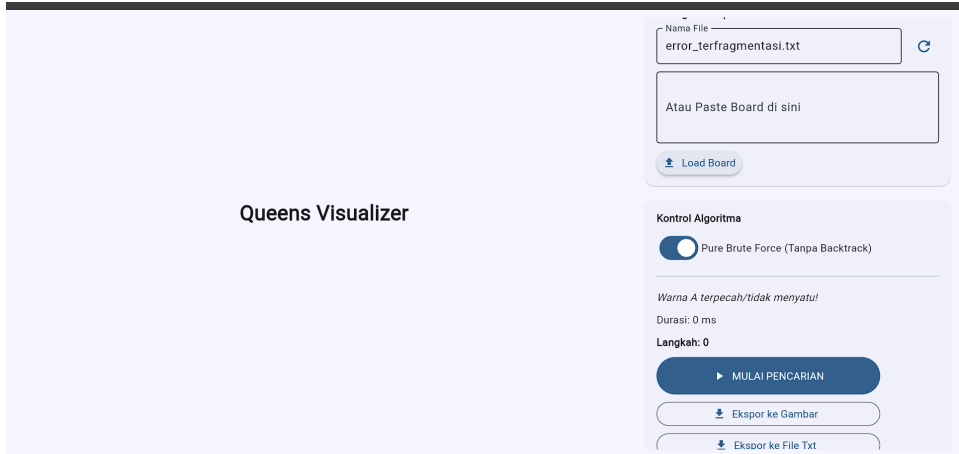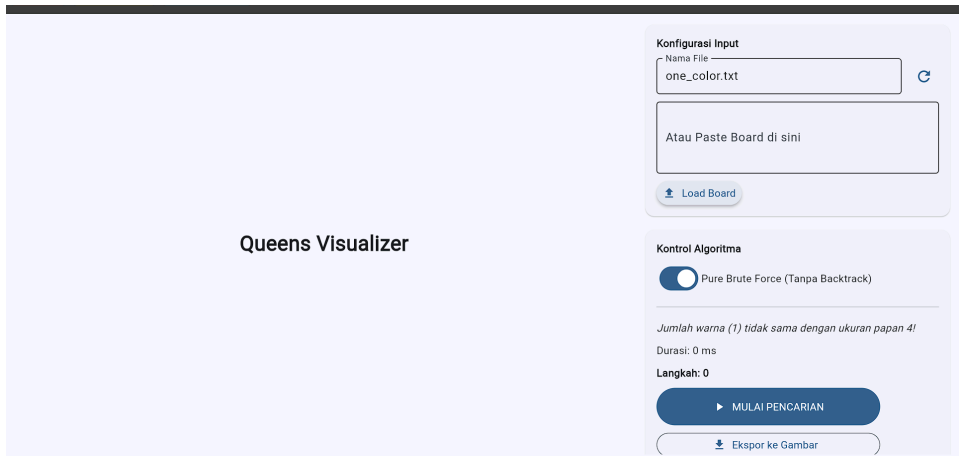
III.   Tangkapan layar studi kasus
1. Uji Kasus error dimensi

Input	Output
AAAAA BBBBB CCCC DDDDD EEEEE	

Queens Visualizer

Konfigurasi Input
Nama File
error_dimensi.txt

Atau Paste Board di sini

⬆ Load Board

Kontrol Algoritma
Pure Brute Force (Tanpa Backtrack)

Papan tidak persegi!
Durasi: 0 ms
Langkah: 0

▶ MULAI PENCARIAN

⬇ Ekspor ke Gambar
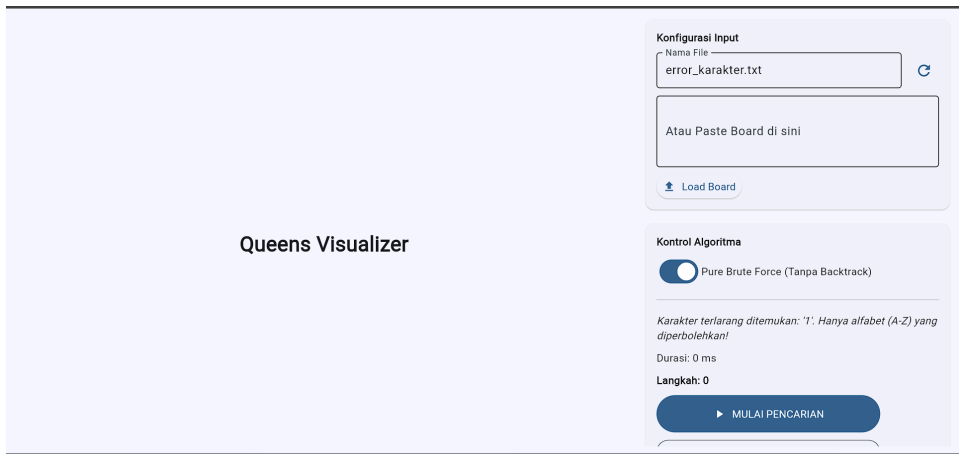
2. Uji Kasus error terfragmentasi warna

Input	Output
AAAAA ABBBC ABACC ABCCC DDEEE	

3. Uji Kasus error jumlah warna
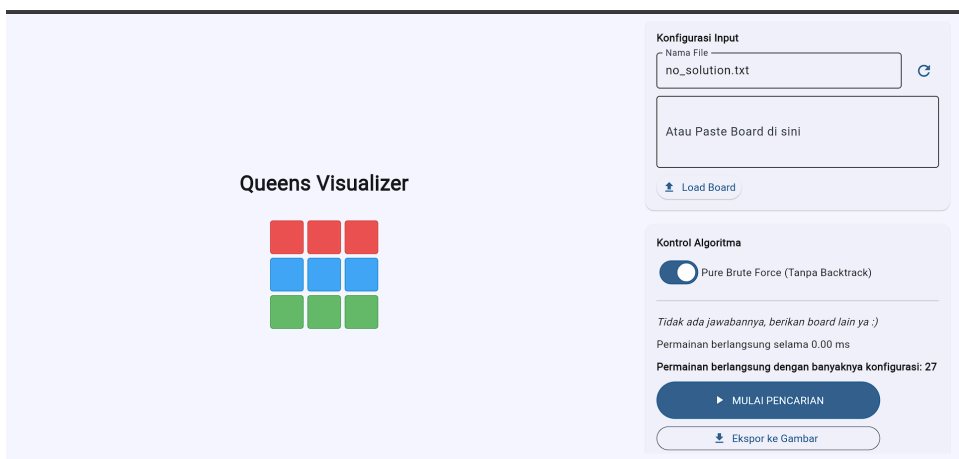
Input	Output
AAAA AAAA AAAA AAAA	

4. Uji Kasus error karakter

Input	Output

AAA1 BB#B CC2C DDDD	

5. Uji Kasus tidak ada solusi

Input	Output
AAA BBB CCC	

6. Uji Kasus normal ukuran 8x8

Input	Output

AABBBBBC
AABBBBCC
AAABBBCC
AAADDBCC
EAAAAACC
EEEEAFFF
EEEAAGFF
HHEEEGGG

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

7. Uji Kasus normal ukuran 9x9

Input	Output

AAABBCCCD
ABBBBCECD
ABBBDCECD
AAABDCCCD
BBBBDDDDD
FGGGDDHDD
FGIGDDHDD
FGIGDDHDD
FGGGDDHHH

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

IV. Pranala GitHub Repository

Pranala GitHub Repository: https://github.com/hakamavicena/Tucil1_13524075

V. Lampiran

No	Poin	Ya	Tidak
1	Program berhasil di kompilasi 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	✓	

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

13524075 - Hakam Avicena Musain