

Laporan Tugas Kecil 1

IF2211 Strategi Algoritma

Penyelesaian Permainan Queens LinkedIn

Dosen Pengampu: Dr. Ir. Rinaldi, M.T.

Kelas: K-01



Disusun oleh:

Stanislaus Ardy Bramantyo

18223057

Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

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

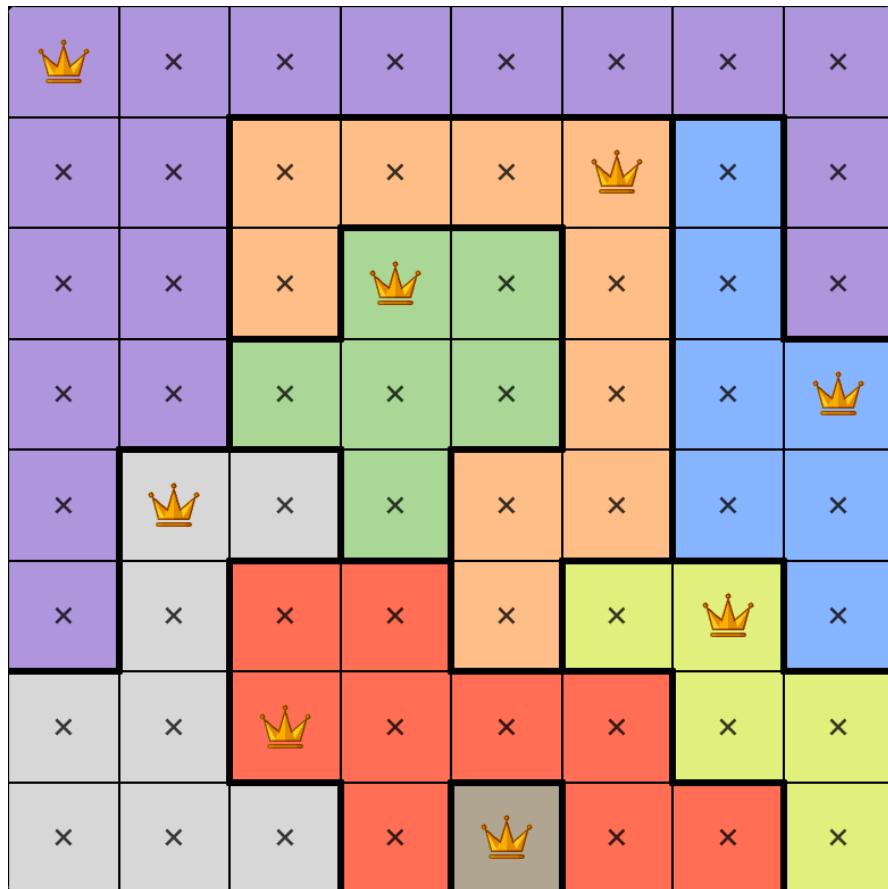


Stanislaus Ardy Bramantyo

Daftar Isi

Daftar Isi.....	3
Permasalahan.....	4
Rancangan Solusi.....	5
A. Eksplorasi Posisi Queen.....	5
1. Instansiasi seluruh kemungkinan posisi sebuah queen.....	5
2. Instansiasi seluruh variabel eksplorasi.....	6
3. Eksplorasi dan pencatatan kombinasi queen.....	6
B. Validasi setiap Kombinasi Posisi Queen.....	8
1. Validasi secara Horizontal (rows).....	8
2. Validasi secara Vertikal (cols):.....	8
3. Validasi secara Diagonal:.....	9
4. Validasi secara Warna:.....	9
C. Menggabungkan dalam Exhaustive Search.....	10
1. Instansiasi seluruh posisi queen yang memungkinkan.....	10
2. Mempersiapkan variabel waktu & iterasi untuk pencatatan, serta interval untuk live update.....	10
3. Iterasi untuk validasi setiap kombinasi.....	10
Pengerjaan Bonus Persoalaan.....	12
A. GUI (Graphical User Interface).....	12
1. Membentuk frame dari GUI.....	12
2. Header Component Algorithms.....	13
3. Body Component Algorithms.....	14
4. Proses Update Algorithm.....	15
B. Output as Image.....	15
Keperluan Tambahan.....	16
Code of Program.....	16
Testing of Program.....	22
Completion Table of Program.....	26

Permasalahan



Gambar 1. Permainan Queens LinkedIn

Queens adalah gim logika yang tersedia pada situs jejaring profesional LinkedIn. Tujuan dari gim ini adalah menempatkan queen pada sebuah papan persegi berwarna sehingga terdapat hanya satu queen pada tiap baris, kolom, dan daerah warna. Selain itu, satu queen tidak dapat ditempatkan bersebelahan dengan queen lainnya, termasuk secara diagonal.

Rancangan Solusi

Untuk permasalahan **Queens Linkedin** tersebut, terdapat sebuah pendekatan sederhana berjenis *bruteforce*. Pendekatan tersebut secara ringkas akan **mengeksplorasi** seluruh *possible combination* dari posisi setiap Queen yang ada di dalam papan warna tersebut.

Untuk memberi gambaran terhadap solusi, saya akan menampilkan beberapa bagian kecil langkah yang akan diambil dari algoritma yang telah dikembangkan dengan penjelasan dari setiap langkahnya yang disertai dengan implementasinya dalam algoritma. Sebagai catatan, kode yang ditampilkan akan menjadi sebagian kecil dari seluruh kode yang ada di dalam file, terutama di bagian yang penting saja. Ini juga berlaku kepada **komen** yang dinilai oleh penulis dirasa penting untuk dimasukan ke laporan.

Sebagai analisis, algoritma yang diajukan untuk solusi ini sekiranya memiliki Time Complexity sebesar $O(n^2 C_n \cdot n^2)$. Hal tersebut dapat dibagi menjadi dua bagian. Pertama, eksplorasi memakan $n^2 C_n$ yang dikarenakan ini bersifat setiap kombinasi posisi queen dari papan warna $N \times N$. Kedua, validasi memakan n^2 karena dalam proses validasi diagonal, algoritmanya memanfaatkan *double for loop*.

A. Eksplorasi Posisi Queen.

Algoritma, sekiranya, akan dimulai saat mengeksplorasi seluruh kombinasi posisi Queen yang memungkinkan tanpa ada duplikasi posisi.

1. Instansiasi seluruh kemungkinan posisi **sebuah queen**.

```
posisi_Q_papan = []
for i in range(self.n):
    for j in range(self.n):
        posisi_Q_papan.append((i,j))
```

Output dari line of code diatas sekiranya akan berbentuk seperti ini:

```
([0,0], [0,1], [0,2], ..., [0,n-1],
 [1, 0], [1, 1], ..., [n-1, n-1])
```

INGAT! Bahwa setiap possible posisi ditulis dalam sebuah indices.

2. Instansiasi seluruh variabel eksplorasi

```
num_of_posisi = len(posisi_Q_papan)
queen = list(range(self.n))

array_of_combination = []
```

n: Jumlah possible posisi queen.

queen: List queen yang ada sebanyak N.

combination: List yang berisikan kombinasi posisi queen.

3. Eksplorasi dan pencatatan kombinasi queen.

```
while True:
    temp = []
    for i in range(self.n):
        temp.append(posisi_Q_papan[queen[i]])
    array_of_combination.append(temp)

    idx = self.n - 1
    while idx >= 0:
        if queen[idx] != idx + (num_of_posisi - self.n):
            break
        else:
            idx -= 1

        if idx < 0: # udah mentok
            break

        else:
            queen[idx] += 1 # maju
            if idx != (self.n - 1):
                for i in range(idx+1, self.n):
                    queen[i] = queen[i-1] + 1
```

- Algoritma di atas akan **mencatat** setiap eksplorasi kombinasi posisi queen yang ada pada setiap loopnya.
- Pada setiap langkah eksplorasi, queen di ujung kanan($idx=self.n-1$) akan menambah stepnya sebanyak 1 langkah ($queen[idx] += 1$).
 - ★ Penambahan 1 untuk melangkah dapat dilakukan karena posisi bersifat *indices*.

```

queens_position = [
    (0,0), (0,1), (0,2), (0,3), # idx 0-3
    (1,0), (1,1), (1,2), (1,3), # idx 4-7
    (2,0), (2,1), (2,2), (2,3), # idx 8-11
    (3,0), (3,1), (3,2), (3,3) # idx 12-15
]
# I-1
queen = [0, 1, 2, 3]
queen[0] = 0 → position[0] = (0,0)
queen[1] = 1 → position[1] = (0,1)
queen[2] = 2 → position[2] = (0,2)
queen[3] = 3 → position[3] = (0,3) # Fokus pada ini

# I-2
queen = [0, 1, 2, 4]
queen[0] = 0 → position[0] = (0,0)
queen[1] = 1 → position[1] = (0,1)
queen[2] = 2 → position[2] = (0,2)
queen[3] = 4 → position[4] = (1,0) # Perubahan ada di sini!

# I-3
queen = [0, 1, 2, 5]
queen[0] = 0 → position[0] = (0,0)
queen[1] = 1 → position[1] = (0,1)
queen[2] = 2 → position[2] = (0,2)
queen[3] = 5 → position[5] = (1,1) # Perubahan ada di sini!

```

- Jika dalam eksplorasinya *queen* sampai di **ujung kanan** papan warna, maka kita pindahkan *queen* dibelakangnya satu kali dan mengeksplorasi ulang dengan *queen* ujung kanan (*if idx !=(self.n-1)*).

Saat memindahkan *queen* dibelakangnya, *queen* setelahnya menjadi di depannya kembali.

```

# I-1
queen = [0, 1, 2, 14]
queen[0] = 0 → position[0] = (0,0)
queen[1] = 1 → position[1] = (0,1)
queen[2] = 2 → position[2] = (0,2)
queen[14] = 14 → position[14] = (3,2)

# I-2
queen = [0, 1, 2, 15]
queen[0] = 0 → position[0] = (0,0)
queen[1] = 1 → position[1] = (0,1)
queen[2] = 2 → position[2] = (0,2) # Fokus pada ini
queen[15] = 15 → position[15] = (3,3) # Fokus pada ini

# I-3
queen = [0, 1, 3, 4]

```

```

queen[0] = 0 → position[0] = (0,0)
queen[1] = 1 → position[1] = (0,1)
queen[3] = 3 → position[3] = (0,3) # Perubahan ada di sini!
queen[4] = 4 → position[4] = (1,0) # Perubahan ada di sini!

```

- Setiap eksplorasi yang dilakukan (langkahnya) akan dicatat untuk nanti diverifikasi.

B. Validasi setiap Kombinasi Posisi Queen

Algoritma bagian ini akan mengecek keunikan posisi *queen* secara Horizontal, Vertikal, Diagonal, dan Warna.

1. Validasi secara Horizontal (rows)

```

horizontal=[]
horizontal_unik=set()
for i in queen_combination:
    horizontal.append(i)
    horizontal_unik.add(i)
if len(horizontal) != len(horizontal_unik):
    return False

```

- Validasi akan mengecek apakah jumlah *queen* sama dengan jumlah *queen* yang unik pada garis Horizontal dari *queen* ke-*n*.

2. Validasi secara Vertikal (cols):

```

vertikal=[]
vertikal_unik=set()
for i in queen_combination:
    vertikal.append(i)
    vertikal_unik.add(i)
if len(vertikal) != len(vertikal_unik):
    return False

```

- Validasi akan mengecek apakah jumlah *queen* sama dengan jumlah *queen* yang unik pada garis Vertikal dari *queen* ke-*n*.

3. Validasi secara Diagonal:

```
for ref in range(len(queen_combination)):
    for ref_kanan in range(ref+1, len(queen_combination)):
        x_r, y_r = queen_combination[ref]
        x_rk, y_rk = queen_combination[ref_kanan]
        if (abs(x_r-x_rk) <=1) and (abs(y_r-y_rk) <=1):
            return False
```

- Kita akan mengiterasikan setiap pasangan queen (dari kiri ke kanan) dan mengecek jaraknya.
- Kita akan menolak jika mereka diagonal dengan ditandakan jarak x dan jarak y sama.

4. Validasi secara Warna:

```
warna=[]
warna_unik=set()
for i in queen_combination:
    warna.append(self.board[i[0]][i[1]])
    warna_unik.add(self.board[i[0]][i[1]])
if len(warna) != len(warna_unik):
    return False
```

- Perlu diingat, untuk **queen** disini **berbeda** dengan yang **sebelumnya**. queens sendiri berupa **list** yang **berisikan posisi** queen ke- n
- **warna** akan mengecek warna queen tersebut dengan melihat dari posisinya (**lookup**).

```
self.colorMap = [
    ['R', 'R', 'B', 'G'],
    ['B', 'G', 'R', 'R'],
    ['G', 'B', 'B', 'R'],
    ['R', 'G', 'G', 'B']
]

queens = [(0,0), (1,2), (2,1), (3,3)]

queen = (0,0) → colorMap[0][0] = 'R'
queen = (1,2) → colorMap[1][2] = 'R'
queen = (2,1) → colorMap[2][1] = 'B'
queen = (3,3) → colorMap[3][3] = 'B'
```

C. Menggabungkan dalam Exhaustive Search

1. Instansiasi seluruh posisi *queen* yang memungkinkan.

```
posisi_Q_papan = []
for i in range(self.n):
    for j in range(self.n):
        posisi_Q_papan.append((i,j))
```

2. Mempersiapkan variabel waktu & iterasi untuk pencatatan, serta interval untuk *live update*.

```
time_start = time.time()
iterasi = 0
interval = max(1000, len(array_of_combination) // 120)
```

3. Iterasi untuk validasi setiap kombinasi.

```
for combination in array_of_combination:
    iterasi += 1
    if iterasi % interval == 0:
        print(f"\nIterasi ke-{iterasi}")
    result = []
    for i in range(self.n):
        row = []
        for j in range(self.n):
            row.append(self.board[i][j])
        result.append(row)
    for i, j in combination:
        result[i][j] = '#' # Queen

    for i in result:
        print(' '.join(i))

    time_end = (time.time() - time_start) * 1000
    if self.gui:
        self.gui(self.board_copy, result, time_end, iterasi, True)
    time.sleep(0.15)
```

```

        if self.validate(combination) == True:
            print(f"\nIterasi ke-{iterasi}")
            result = []
            for i in range(self.n):
                row = []
                for j in range(self.n):
                    row.append(self.board[i][j])
                result.append(row)
            for i, j in combination:
                result[i][j] = '#'

            for i in result:
                print(' '.join(i))
            time_end = (time.time()-time_start) * 1000
            print(f"\nWaktu pencarian: {time_end:.2f} ms")
            print(f"Banyak kasus yang ditinjau: {iterasi} kasus")
            f= open('solution.txt', 'w')
            for i in result:
                f.write(''.join(i)+'\n')
            f.close()

            if self.gui:
                self.gui(self.board_copy, result, time_end, iterasi, True)
            return True

        time_end = (time.time()-time_start) * 1000
        print(f"\n Tidak ada solusi")
        print(f"Waktu pencarian: {time_end:.2f} ms")
        print(f"Banyak kasus yang ditinjau: {iterasi} kasus")
        if self.gui:
            self.gui(self.board_copy, None, time_end, iterasi, False)
        return False
    
```

- Kita akan melakukan **validasi** dari **setiap kombinasi** posisi queen yang telah di eksplorasi. Setiap kombinasi untuk diverifikasi dianggap sebagai **iterasi**.
- Jika **ditengah** validasi terdapat **kebenaran**, maka akan diberhentikan dan menampilkan hasilnya.
- Dalam implementasi, akan melakukan *live update* setiap beberapa persen dari keseluruhan langkah.

Pengerjaan Bonus Persoalaan

A. GUI (*Graphical User Interface*)

Dalam penggerjaan *bonus* yang pertama, yaitu GUI, saya memanfaatkan *library* `tkinter` dan `threading`. Secara berurutan, *library* yang saya gunakan berguna untuk menggambarkan GUI itu sendiri serta “memperlancar” GUI itu sendiri berjalan.

1. Membentuk *frame* dari GUI.

```
header = tk.Frame(root) # header, untuk functional button
header.pack(fill=tk.X)

up_file = tk.Button(header, text="Upload File", command=self.upload,
bg='light blue', fg='black', padx=20)
up_file.pack(side=tk.LEFT)

solve_btn = tk.Button(header, text="Solusi!", command=self.solve,
bg='green', fg='white', padx=20)
solve_btn.pack(side=tk.LEFT, padx=5)

body=tk.Frame(root, bg='black') # body, buiat tampilin gambar
body.pack(fill=tk.BOTH, expand=True)
kontainer_input = tk.Frame(body, bg='black')
kontainer_input.pack(side=tk.LEFT, fill=tk.BOTH, expand=True, padx=5)
self.input_board = tk.Canvas(kontainer_input, bg='beige', width=400,
height=400)
self.input_board.pack(pady=5)
kontainer_output = tk.Frame(body, bg='black')
kontainer_output.pack(side=tk.LEFT, fill=tk.BOTH, expand=True, padx=5)
self.output_board = tk.Canvas(kontainer_output, bg='beige', width=400,
height=400)
self.output_board.pack(pady=5)

footer= tk.Frame(root, bg='light blue', pady=15) # footer, buat stats
aja
footer.pack(fill=tk.X)
self.stats = tk.Label(footer, text="stats: ", bg='light blue',
fg='black')
self.stats.pack()
```

- Menyiapkan beberapa komponen dari GUI, yaitu Header (Menampilkan Upload & Solution Button), Body (Menampilkan input & output), dan Footer (Menampilkan Stats).

2. Header Component Algorithms

```

def upload(self):
    file_input = filedialog.askopenfilename(filetypes=[("Text Files", "*.txt")], )
    self.input_board.delete("all")
    self.output_board.delete("all")

    if file_input:
        try:
            self.file_path=file_input
            self.algorithm=exhaustive_search(file_input, self.update)
            self.gambar(self.algorithm.board_copy, self.algorithm.board, self.input_board)

        except ValueError:
            messagebox.showerror("File Invalid")
            self.file_path=None
            self.algorithm=None

def solve(self):
    if self.algorithm:
        self.root.update()
        t = threading.Thread(target=exhaustive_search(self.file_path,
                                                       self.update).algoritma_ES)
        t.daemon =True
        t.start()

    else:
        messagebox.showwarning("File Invalid!")
        return

```

- Upload akan membuka windows yang memungkinkan user untuk memilih file untuk input ke dalam program.
- Input akan ada validasi pada awal instansiasi exhaustive_search yang mengecek apakah jumlah warna = N .

3. Body Component Algorithms

```
def gambar(self, board_awal, board_akhir, kontainer):
kontainer.delete("all")
if board_akhir:

    n=len(board_akhir)
    ukuran=min(500//n, 50)
    for i in range(n):
        for j in range(n):
            x_1=(j*ukuran) + 5
            y_1=(i*ukuran) + 5
            x_2=(x_1+ukuran)
            y_2=(y_1+ukuran)

            warna_akhir= board_akhir[i][j]
            if warna_akhir=='#': # Queen
                warna_awal= board_awal[i][j]
                kontainer.create_rectangle(x_1,y_1,x_2,y_2,
fill=self.pallete.get(warna_awal, 'white'))
                kontainer.create_text((x_1+x_2)/2, (y_1+y_2)/2, text='Q', fill='white',
font=('Arial', 18, 'bold'))

            else: #Warna biasa
                kontainer.create_rectangle(x_1,y_1,x_2,y_2,
fill=self.pallete.get(warna_akhir, 'white'))
```

- Penggambaran kotak dilakukan satu per satu menyesuaikan konten dari kotak tersebut (apakah warna saja, atau Queen[#]).
- Untuk ukuran hingga saat ini digambarkan dengan ukuran 50px paling minimal.

4. Proses Update Algorithm

```
def update(self, board_awal, board_akhir, time, iterasi, found):
    if found:
        self.gambar(board_awal, board_akhir, self.output_board)
        self.stats.config(text=f"Waktu pencarian: {time:.2f} ms & Banyak tinjauan: {iterasi} kasus")
        self.root.after(40, lambda: self.screenshot(self.output_board))

    else:
        self.output_board.delete("all")
        self.output_board.create_text(200, 200, text="Tidak ada solusi!", fill='red')
        self.stats.config(text=f"Waktu pencarian: {time:.2f} ms & Banyak tinjauan: {iterasi} kasus")
```

- Menggambar kotak *output* menyesuaikan hasil algoritma, apakah menemukan solusinya atau tidak.

B. Output as Image

```
def screenshot(self, kontainer):
    x=kontainer.winfo_rootx()
    y=kontainer.winfo_rooty()
    x_1=x+ kontainer.winfo_width()
    y_1=y+ kontainer.winfo_height()
    ImageGrab.grab(bbox=(x,y,x_1,y_1)).save("solution.png")
```

- Saya memanfaatkan Library PIL (Python Imaging Library) dan mengambil bagian ImageGrab.
- Secara ringkas, algoritma ini akan me-screenshot di sebuah kotak yang disesuaikan dengan koordinat dimana kotak *output* itu berada.
- Kelemahan *approach* ini adalah tidak bisa pindah ke windows lain karena akan screenshot windows lain jadinya (menyesuaikan apa yang ada di layar saja).

Keperluan Tambahan

Code of Program

```
import time
import tkinter as tk
from tkinter import filedialog, messagebox
import threading
from PIL import ImageGrab

class exhaustive_search:
    def __init__(self, file_input, gui):
        f = open(file_input, 'r')
        konten_file = f.readlines()
        f.close()

        self.board = []
        for line in konten_file:
            line = line.strip()
            if line != "":
                self.board.append(list(line))

        self.n = len(self.board)

        warna_unik = set()
        for i in range(self.n):
            for j in range(self.n):
                warna_unik.add(self.board[i][j]) # Karena bersifat set, maka tidak akan menambahkan warna yang sama

        if len(warna_unik) != self.n:
            raise ValueError("Input Invalid!")

    else:
        self.board_copy = self.board

    self.gui = gui

    def algoritma_ES(self):
        posisi_Q_papan = []
        for i in range(self.n):
            for j in range(self.n):
                posisi_Q_papan.append((i,j))

        num_of_posisi = len(posisi_Q_papan)
        queen = list(range(self.n))

        array_of_combination = []
        while True:
            temp = []
```

```

        for i in range(self.n):
            temp.append(posisi_Q_papan[queen[i]])
        array_of_combination.append(temp) # Untuk mencatat kombinasi dari setiap queen

        idx = self.n - 1
        while idx >= 0:
            if queen[idx] != idx + (num_of_posisi - self.n):
                break
            else:
                idx -= 1

        if idx < 0: # udah mentok
            break

        else:
            queen[idx] += 1 # maju
            if idx != (self.n - 1):
                for i in range(idx+1, self.n): # reset kalau bukan ujung aja
                    queen[i] = queen[i-1] + 1

    time_start = time.time()
    iterasi = 0
    interval = max(1000, len(array_of_combination) // 120) # 10% kurleb ceunah

    for combination in array_of_combination:
        iterasi += 1
        if iterasi % interval == 0:
            print(f"\nIterasi ke-{iterasi}")
        result = []
        for i in range(self.n):
            row = []
            for j in range(self.n):
                row.append(self.board[i][j])
            result.append(row)
        for i, j in combination:
            result[i][j] = '#' # Queen

        for i in result:
            print(' '.join(i))

        time_end = (time.time()-time_start) * 1000
        if self.gui:
            self.gui(self.board_copy, result, time_end, iterasi, True)
        time.sleep(0.15)

        if self.validate(combination) == True:
            print(f"\nIterasi ke-{iterasi}")
            result = []
            for i in range(self.n):
                row = []
                for j in range(self.n):
                    row.append(self.board[i][j])
                result.append(row)
            for i, j in combination:

```

```

        result[i][j] = '#'

    for i in result:
        print(' '.join(i))
    time_end = (time.time()-time_start) * 1000
    print(f"\nWaktu pencarian: {time_end:.2f} ms")
    print(f"Banyak kasus yang ditinjau: {iterasi} kasus")
    f= open('solution.txt', 'w')
    for i in result:
        f.write('.join(i)+'\n')
    f.close()

    if self.gui:
        self.gui(self.board_copy, result, time_end, iterasi, True)
    return True

    time_end = (time.time()-time_start) * 1000
    print(f"\n Tidak ada solusi")
    print(f"Waktu pencarian: {time_end:.2f} ms")
    print(f"Banyak kasus yang ditinjau: {iterasi} kasus")
    if self.gui:
        self.gui(self.board_copy, None, time_end, iterasi, False)
    return False

def validate(self, queen_combination):
    # 1. Horizontal
    horizontal=[]
    horizontal_unik=set()
    for i in queen_combination:
        horizontal.append(i)
        horizontal_unik.add(i)
    if len(horizontal) != len(horizontal_unik):
        return False

    #2. Vertikal
    vertikal=[]
    vertikal_unik=set()
    for i in queen_combination:
        vertikal.append(i)
        vertikal_unik.add(i)
    if len(vertikal) != len(vertikal_unik):
        return False

    #3. Diagonal
    for ref in range(len(queen_combination)):
        for ref_kanan in range(ref+1, len(queen_combination)):
            x_r, y_r = queen_combination[ref]
            x_rk, y_rk = queen_combination[ref_kanan]
            if (abs(x_r-x_rk) <=1) and (abs(y_r-y_rk) <=1):
                return False

    #4. Warna
    warna=[]

```

```

warna_unik=set()
for i in queen_combination:
    warna.append(self.board[i[0]][i[1]])
    warna_unik.add(self.board[i[0]][i[1]])
if len(warna) != len(warna_unik):
    return False

return True

class gui:
    def __init__(self, root):
        self.root = root
        self.root.geometry("800x600")

        header = tk.Frame(root) # header, untuk functional button
        header.pack(fill=tk.X)
        up_file = tk.Button(header, text="Upload File", command=self.upload, bg='light blue', fg='black',
        padx=20)
        up_file.pack(side=tk.LEFT)
        solve_btn =tk.Button(header, text="Solusi!", command=self.solve, bg='green', fg='white', padx=20)
        solve_btn.pack(side=tk.LEFT, padx=5)

        body=tk.Frame(root, bg='black') # body, buat tampilin gambar
        body.pack(fill=tk.BOTH, expand=True)
        kontainer_input = tk.Frame(body, bg='black')
        kontainer_input.pack(side=tk.LEFT, fill=tk.BOTH, expand=True, padx=5)
        self.input_board = tk.Canvas(kontainer_input, bg='beige', width=400, height=400)
        self.input_board.pack(pady=5)
        kontainer_output = tk.Frame(body, bg='black')
        kontainer_output.pack(side=tk.LEFT, fill=tk.BOTH, expand=True, padx=5)
        self.output_board = tk.Canvas(kontainer_output, bg='beige', width=400, height=400)
        self.output_board.pack(pady=5)

        footer= tk.Frame(root, bg='light blue', pady=15) # footer, buat stats aja
        footer.pack(fill=tk.X)
        self.stats = tk.Label(footer, text="stats: ", bg='light blue', fg='black')
        self.stats.pack()

        self.file_path = None
        self.algorithm = None

        self.pallete = { # me ji ku hi bi ni u
            'A': "#750707",
            'B': "#976810",
            'C': "#85852c",
            'D': '#008000',
            'E': "#28a228",
            'F': '#000080',
            'G': "#187b71",
            'H': "#5f047d",
            'I': "#a84576",
        }
    }

    def upload(self):

```

```

file_input = filedialog.askopenfilename(filetypes=[("Text Files", "*.txt")], )
self.input_board.delete("all")
self.output_board.delete("all")

if file_input:
    try:
        self.file_path=file_input
        self.algorithm=exhaustive_search(file_input, self.update)
        self.gambar(self.algorithm.board_copy, self.algorithm.board, self.input_board)

    except ValueError:
        messagebox.showerror("File Invalid")
        self.file_path=None
        self.algorithm=None

def solve(self):
    if self.algorithm:
        self.root.update()
        t = threading.Thread(target=exhaustive_search(self.file_path, self.update).algoritma_ES)
        t.daemon =True
        t.start()

    else:
        messagebox.showwarning("File Invalid!")
        return

def update(self, board_awal, board_akhir, time, iterasi, found):
    if found:
        self.gambar(board_awal, board_akhir, self.output_board)
        self.stats.config(text=f"Waktu pencarian: {time:.2f} ms & Banyak tinjauan: {iterasi} kasus")
        self.root.after(40, lambda: self.screenshot(self.output_board))

    else:
        self.output_board.delete("all")
        self.output_board.create_text(200, 200, text="Tidak ada solusi!", fill='red')
        self.stats.config(text=f"Waktu pencarian: {time:.2f} ms & Banyak tinjauan: {iterasi} kasus")

def gambar(self, board_awal, board_akhir, kontainer):
    kontainer.delete("all")
    if board_akhir:

        n=len(board_akhir)
        ukuran=min(500//n, 50)
        for i in range(n):
            for j in range(n):
                x_1=(j*ukuran) + 5
                y_1=(i*ukuran) + 5
                x_2=(x_1+ukuran)
                y_2=(y_1+ukuran)

                warna_akhir= board_akhir[i][j]
                if warna_akhir=='#': # Queen
                    warna_awal= board_awal[i][j]

```

```

        kontainer.create_rectangle(x_1,y_1,x_2,y_2, fill=self.pallete.get(warna_awal,
'white'))
        kontainer.create_text((x_1+x_2)/2, (y_1+y_2)/2, text='Q', fill='white',
font=('Arial', 18, 'bold'))

    else: #Warna biasa
        kontainer.create_rectangle(x_1,y_1,x_2,y_2, fill=self.pallete.get(warna_akhir,
'white'))

    def screenshot(self, kontainer):
        x=kontainer.winfo_rootx()
        y=kontainer.winfo_rooty()
        x_1=x+ kontainer.winfo_width()
        y_1=y+ kontainer.winfo_height()
        ImageGrab.grab(bbox=(x,y,x_1,y_1)).save("solution.png")

if __name__ == "__main__":
    import sys
    if len(sys.argv) > 1:
        file=sys.argv[1]
        solusi = exhaustive_search(file_input=file).algoritma_ES()
    else:
        root= tk.Tk()
        app= gui(root)
        root.mainloop()

```

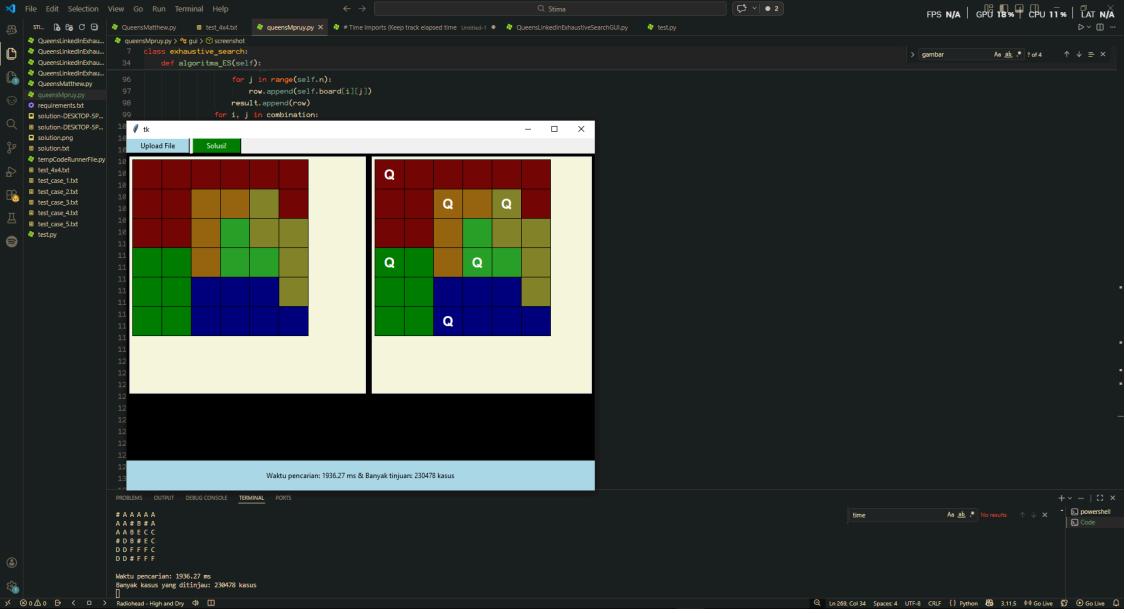
Testing of Program

1. Test Case #1

(<https://medium.com/@jeremy.ockenden/solving-linkedins-queens-puzzle-in-python-3f0d53925842>)

[INPUT Map]

```
AAAAAA
AABBCA
AABECC
DDBEEC
DDFFFC
DDFFFF
```



[OUTPUT Map]

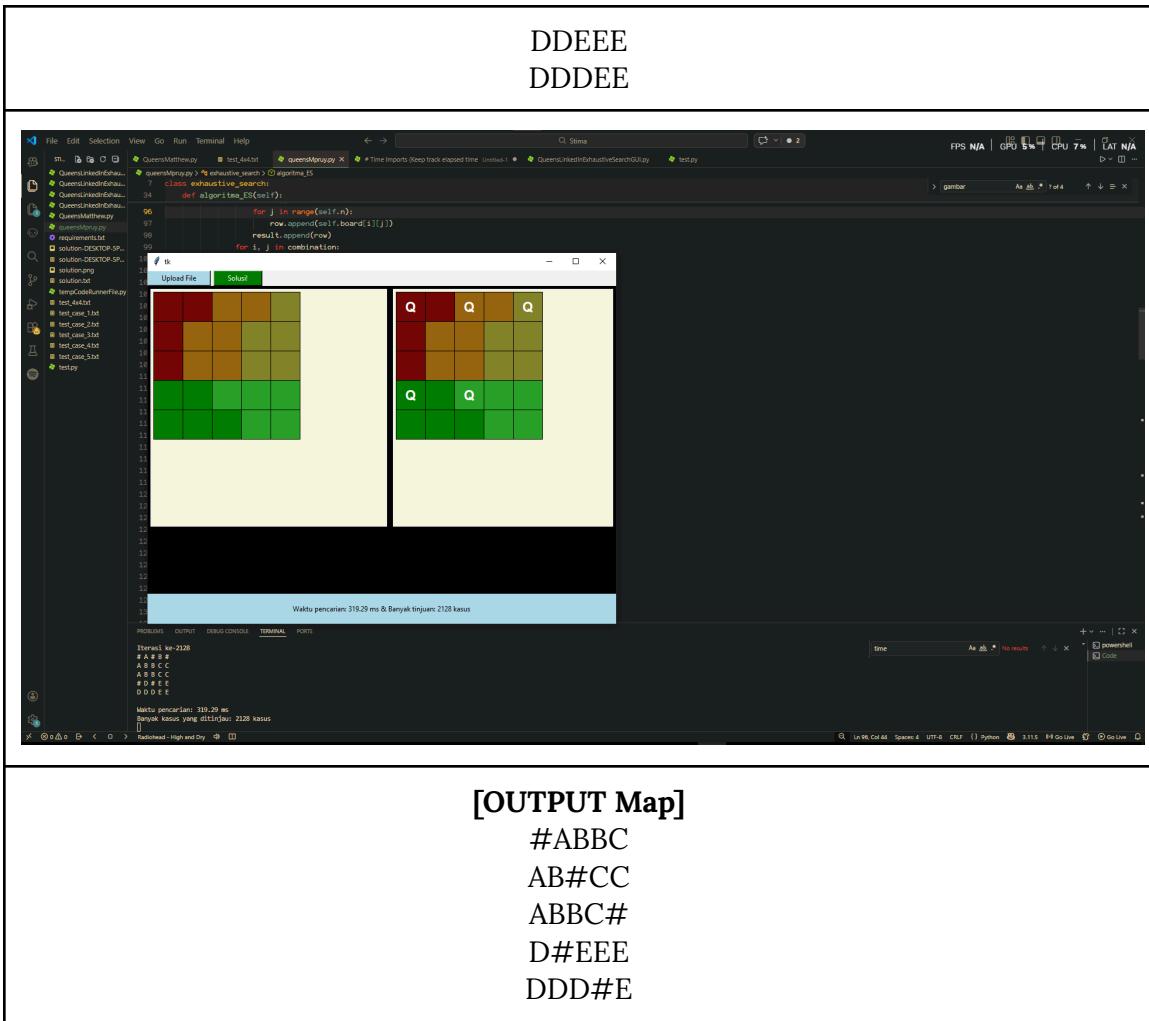
```
#AAAAA
AA#BCA
AABEC#
DDB#EC
D#FFFC
DDFF#F
```

2. Test Case #2

(https://github.com/RebeccaTadesse/LinkedIn-Queens-Solver/blob/main/sample_boards/b1.png)

[INPUT Map]

```
AABBC
ABBCC
ABBCC
```



[OUTPUT Map]

#ABBC
AB#CC
ABBC#
D#EEE
DDD#E

3. Test Case #3 (<https://stealthygaming.com/linkedin-queens-answer-today/>)

[INPUT Map]

AAABC
BBBBB
DCCB
DDCCCC
EDCF
EDFFFFF
EEEFGGG

The screenshot shows a terminal window with several tabs open. The current tab displays a 8x8 grid representing a chessboard. The grid has alternating colors (light and dark squares). On the board, there are three queens (Q) placed at (row 1, column 2), (row 2, column 1), and (row 3, column 4). There are also other pieces represented by letters: B (Bishop), C (King), D (Double Queen), E (EPA), F (Fool), G (Guard), and H (Horse). Below the grid, a message indicates the search took 9445.49 ms and found 3116200 cases. The terminal also shows some code snippets, including a requirement file and a solution file.

```
Waktu pencarian: 9445.49 ms & Banyak tingen: 3116200 kasus
time.sleep(0.15)
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
BBBBB#CCC
#B#C#C#C#
D#C#C#C#
L#E#F#F#
E#E#F#G#
Maktu pencariin: 9445.49 ms
Banyak kasus yang ditujuin: 3116200 kasus
[]
```

4. Test Case #4 (<https://imiron.io/post/linkedin-queens/>)

[INPUT Map]

ABBBBB

CDCBC

CDCBC

CCCEE

CCCCC

— 1 —

```
Waktu pencarian: 329.30 ms & Banyak tinjauan: 2772 kasus
time.sleep(0.15)
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Iterasi ke-2772
# # # #
C C # #
# C C #
C # C #
C C C E
C C C C

Waktu pencarian: 329.30 ms
Jumlah kasus yang ditinjau: 2772 kasus
[]
```

5. Test Case #5 (<https://tryhardguides.com/linkedin-queens-answer-today/>)

[INPUT Map]

AAAAAAA

BBBBBBBEE

BCBBDDI

CCBBDEI

FCBBBDDD

The screenshot shows a terminal window with the following content:

```

[OUTPUT Map]
AAA#AAA
B#BBBBB
BCBB#DD
CCBBD#
#CBBDD
FCBBD#
CC#BDDD

```

Below the terminal window, the output map is displayed as:

```

AAA#AAA
B#BBBBB
BCBB#DD
CCBBD#
#CBBDD
FCBBD#
CC#BDDD

```

Completion Table of Program

No	Poin	Ya	Tidak
1	Program berhasil di kompilasi tanpa kesalahan	X	
2	Program berhasil dijalankan	X	
3	Solusi yang diberikan program benar dan mematuhi aturan permainan	X	
4	Program dapat membaca masukan berkas .txt serta menyimpan solusi dalam berkas .txt	X	

5	Program memiliki Graphical User Interface (GUI)	X	
6	Program dapat menyimpan solusi dalam bentuk file gambar	X	