LAPORAN TUGAS KECIL 1

IF2211 STRATEGI ALGORITMA

"Penyelesaian Permainan Queens Linkedin

Menggunakan Algoritma BruteForce"

Oleh:

Juan Oloando Simanungkalit (13524032)

PROGRAM STUDI TEKNIK INFORMATIKA

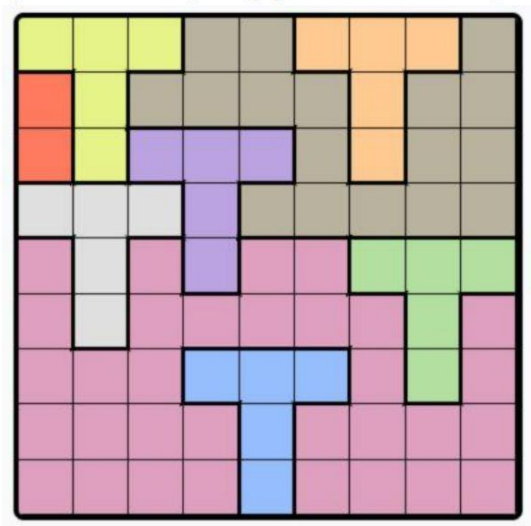
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

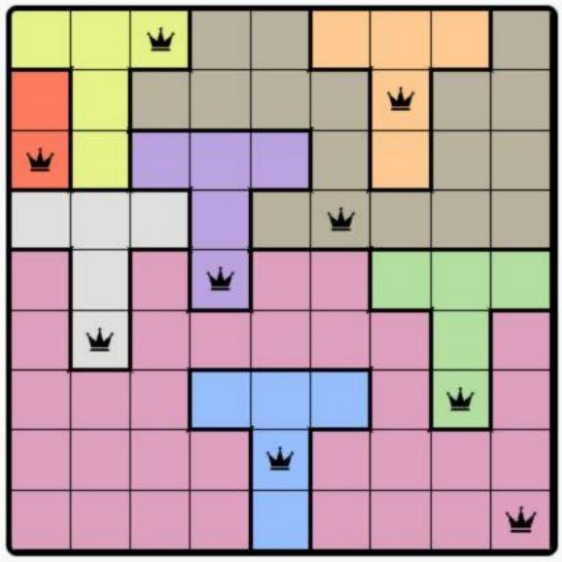
BAB I
DESKRIPSI PERSOALAN

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. Persoalannya adalah membuat program yang dapat menemukan satu solusi penempatan queen pada suatu papan berwarna yang diberikan, atau menampilkan bahwa tidak ada solusi yang valid. Program melakukan pencarian solusi menggunakan algoritma ***brute force***.

Diberikan papan kosong sebagai berikut,



dengan menggunakan algoritma ***brute force*** maka akan dicari solusi berdasarkan aturan main Queens sehingga hasilnya adalah sebagai berikut.



BAB II
DASAR TEORI

Algoritma *brute force* adalah metode sederhana untuk menyelesaikan masalah dengan mencoba semua kemungkinan solusi satu per satu hingga menemukan yang benar. Algoritma ini menyelesaikan persoalan secara lampang atau *straightforward*.

Proses ini dilakukan tanpa mempertimbangkan cara yang lebih efisien atau pola tertentu. Misalnya, dalam konteks pemecahan kata sandi atau enkripsi, algoritma ini akan mencoba setiap kombinasi karakter hingga kombinasi yang sesuai ditemukan. *Brute force* memiliki tujuan utama untuk mengeksplorasi semua kemungkinan dalam ruang solusi secara menyeluruh, sehingga solusi terbaik atau jawaban pasti akan ditemukan, meskipun membutuhkan waktu yang lama dan memakan sumber daya komputasi yang besar.

Pada umumnya, algoritma *brute force* dapat langsung digunakan dengan didasarkan pada pertanyataan di dalam persoalan (*problem statement*) serta definisi/konsep yang dilibatkan di dalam persoalan tersebut.

BAB III
DESAIN

3.1 Tech Stack

Aplikasi ini dibangun menggunakan bahasa Python dengan menggunakan beberapa library tambahan sebagai pendukung dalam pembuatan visualisasi, yaitu GUI. Arsitektur dari aplikasi ini terdiri atas beberapa komponen berikut.

1. Bahasa Pemrograman : Python 3.13.7
2. Framework : Tkinter
3. Library : Pillow (PIL), os, time,dll.

3.2 Struktur Program

Program disusun dalam struktur sesuai dengan spesifikasi tugas kecil yang tersusun atas beberapa folder, diantaranya sebagai berikut.

1. src/
    a) src/main/ : Berisi GUI dan main program
    b) src/bruteforce/ : Berisi algoritma brute force
    c) src/util/ : Berisi helper untuk menyimpan solusi
    d) src/validasiGrid/ : Berisi helper untuk validasi input
2. doc/ : Berisi laporan tugas kecil dalam bentuk PDF
3. test/ : Berisi *test case* serta solusinya

3.3 Cara Mengompilasi dan Menjalankan Program

Untuk melakukan kompilasi dan menjalankan program jalankan perintah berikut pada terminal.

1. Masuk ke folder project, lalu pindah direktori menuju “src”c

```
cd src
```

2. Jika belum memiliki library Pillow, maka install terlebih dahulu

```
pip install Pillow
```

3. Jalankan perintah berikut

```
python main.py
```

BAB IV
ALGORITMA

4.1 Konsep Dasar Brute Force Pada Algoritma

Algoritma yang digunakan pada tugas kecil ini adalah *brute force* murni dengan pendekatan rekursif. Pendekatan ini menoba seluruh kemungkinan penempatan queen pada papan berukuran N x N, kemudian baru validasi solusi setelah menempatkan N buah Queen pada bidang. Untuk kompleksitas waktu dari metode ini adalah O(N^N) karena setiap kolom memiliki N baris.

4.2 Langkah-langkah Algoritma Brute Force

1.Inisialisasi Solver (Suatu array untuk menyimpan posisi Queens)

Pada fungsi ini waktu mulai dicatat dan dibuat array sepanjang N dengan nilai awal -1. Index dari array melambangkan kolom dan valuenya merepresentasikan baris dimana Queen ditempatkan.

```
posisi = [-1] * self.n
```

2.Rekursi per Kolom

Jadi, prinsip dari algoritma ini adalah setiap kolom memiliki satu Queen . Algoritma berjalan dari kolom ke kolom lain. Jika col == N maka semua Queens sudah ditempatkan dan masuk ke tahap validasi.

```
bruteForce(col, posisi)
```

3.Coba semua baris untuk satu kolom

Tiap kolom melakukan hal berikut, yaitu meletakkan Queen di semua kemungkinan baris. Setiap iterasi dicatat.

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

4.Rekursi ke Kolom berikutnya

Setelah satu posisi dicoba maka akan pindah ke Kolom berikutnya.

```
bruteForce(col + 1, posisi)
```

5.Mundur jika tidak ada solusi

Jika memang sudah tidak ada solusi maka mundur ke kolom sebelumnya lalu coba alternatif lain.

```
posisi[col] = -1
return False
```

BAB V
SOURCE CODE

5.1 bruteforce.py

```
import time

# Fungsi untuk menentukan iterasi live update brute force
def liveUpdate(n):
    if (n<3):
        return 1
    elif (n < 5):
        return 10
    elif n < 9:
        return 10000
    else:
        return 1000000

# Kelas untuk menyimpan informasi yang dibutuhkan dalam
menyelesaikan permaianan Queens
class QueensSolution:

    def __init__(self,grid, updatePapan):
        self.grid   = grid
        self.n      = len(grid)
        self.updatePapan = updatePapan
        self.iterasi = 0
        self.waktuMulai = None
        self.solusi = []
        self.updateInterval = liveUpdate(self.n)
        self.Warna = sorted(set(cell for row in grid for
cell in row)) #nyimpen warna unik

    def solve(self):
        self.waktuMulai = time.time()
        posisi = [-1] * self.n # -1 kalau blm diisi
        found = self.bruteForce(0,posisi)
        waktuEks = int((time.time() - self.waktuMulai) *
1000)
        return found, self.solusi, self.iterasi, waktuEks

    def bruteForce(self, col, posisi):
        if col == self.n:
            self.iterasi += 1

            if self.iterasi % self.updateInterval == 0:
                queens = []
                for c in range(self.n):
                    queens.append((posisi[c],c)) #posisi
yang ada queen nya (row,col)
```

```
                self.updatePapan(queens, self.iterasi)

            if self.isValid(posisi):
                self.solusi = []
                for c in range(self.n):
                    self.solusi.append((posisi[c],c))
                return True
            else:
                return False

        for row in range(self.n):
            posisi[col] = row
            self.iterasi += 1

            if self.iterasi % self.updateInterval == 0:
                queens = []
                for c in range(col + 1):
                    queens.append((posisi[c],c))
                self.updatePapan(queens, self.iterasi)

            if self.bruteForce(col+1, posisi):
                return True

        posisi[col] = -1
        return False

    def isValid(self, posisi):
        n = self.n
        warnaUsed = set()
        for c in range(n):
            r = posisi[c]
            warnaReg = self.grid[r][c]
            if warnaReg in warnaUsed:
                return False
            warnaUsed.add(warnaReg)

        #tiap warna hanya satu queen
        if warnaUsed != set(self.Warna):
            return False

        #tiap row hanya satu queen
        if len(set(posisi)) != n:
            return False

        #ga bole ada diagonal
        for col1 in range(n):
            for col2 in range (col1 + 1, n):
                if abs(posisi[col1] - posisi[col2]) <= 1
and (col2-col1) == 1:
                    return False
```

```
        return True
```

Pada fungsi **liveUpdate** berguna untuk menentukan iterasi sesuai dengan jumlah N. Hal ini bertujuan untuk mengoptimalkan hasil output supaya tidak terlalu berat. Fungsi **isValid** berguna untuk mengecek apakah posisi Queens sudah sesuai aturan atau belum. Karena sejak awal sudah menempatkan satu Queen tiap kolom, maka hanya ada tiga kondisi yang di cek, yaitu 1. Apakah ada Queen pada baris yang sama? 2. Apakah setiap warna sudah ditempati oleh satu Queen? Dan 3. Apakah ada Queen yang letaknya saling diagonal?

5.2 util.py

```
import os

def namaFileOutput(directory, namaFile, tipeFile):
    fileOutput = os.path.join(directory,
f"{namaFile}.{tipeFile}")
    if not os.path.exists(fileOutput):
        return fileOutput
    counter = 1
    while True:
        fileOutput = os.path.join(directory,
f"{namaFile}_{counter}.{tipeFile}")
        if not os.path.exists(fileOutput):
            return fileOutput
        counter += 1

def saveAsTxt(grid, solusi, saveDir="."):
    if not grid:
        return None
    os.makedirs(saveDir, exist_ok=True)

    n = len(grid)
    output = [row[:] for row in grid]
    posisiQueen = set(solusi)

    for r in range(n):
        for c in range(n):
            if (r, c) in posisiQueen:
                output[r][c] = "#"

    result = "\n".join("".join(row) for row in output)
    filepath = namaFileOutput(saveDir, "Solusi-Queens",
"txt")

    with open(filepath, "w") as f:
        f.write(result)
```

```
    return filepath

def saveAsImage(widget, saveDir="."):
    try:
        from PIL import ImageGrab
        os.makedirs(saveDir, exist_ok=True)
        widget.update_idletasks()

        x = widget.winfo_rootx()
        y = widget.winfo_rooty()
        w = x + widget.winfo_width()
        h = y + widget.winfo_height()

        img = ImageGrab.grab((x, y, w, h))
        filepath = namaFileOutput(saveDir, "Solusi-
Queens", "png")
        img.save(filepath)
        return True, filepath
    except ImportError:
        return False, "Pillow belum diinstall. Silakan
jalankan: pip install Pillow"
    except Exception as e:
        return False, str(e)
```

Fungsi **namaFileOutput** berguna untuk menghasilkan nama file yang berbeda setiap kali save image atau .txt. Parameternya adalah directory file, nama default file, dan jenis filenya apa, antara .txt atau image. Fungsi **saveAsTxt** menyimpan hasil dalam bentuk file .txt dimana Queen direpresentasikan dengan symbol “#”. Lalu, **saveAsImage** menyimpan hasil dalam bentuk image. Untuk parameternya kurang lebih sama yaitu directory tempat penyimpanan file.

5.3 validasiGrid.py

```
def validasiGrid(grid):
    n = len(grid)

    #ukuran grid harus NxN
    if any(len(row)!=n for row in grid):
        return False, "Input Invalid: grid tidak berukuran
NxN"

    #jumlah warna harus sama dengan N
    warnaUnik = set(cell for row in grid for cell in row)
    if len(warnaUnik)!= n:
        return False, f"Input Invalid: jumlah warna unik
yang diharapkan {n} buah, yang diinput {len(warnaUnik)}
buah "
```

```
    return True, "Valid"
```

Ada dua fungsi utama dalam file ini, yaitu melakukan validasi apakah dimensi file input sudah sama atau belum dan validasi jumlah warna sudah sesuai dengan jumlah N (dimensi, Panjang atau lebar) atau belum.

BAB VI
TEST CASE

6.1 Test Case 1 (Grid 3x3)

Input:

AAB

ABB

ACC

Output:

6.2 Test Case 2 (Grid 5x5, Jumlah warna < N)

Input:

AABBB

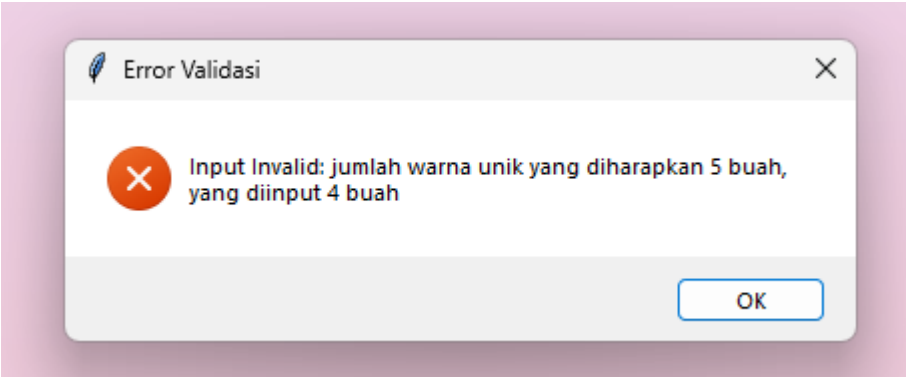
AABBB

CCDDD

CCDDD

CCDDD

Output:



6.3 Test Case 3 (Grid 6x6)

Input:

AAABBC

AAABBC

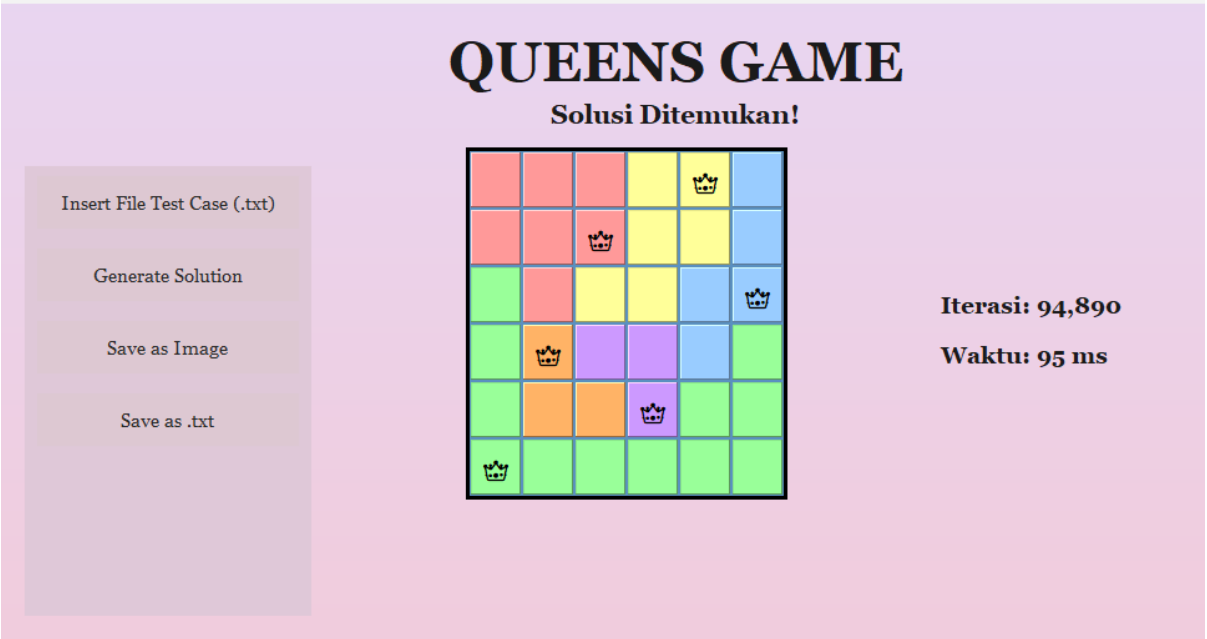
DABBCC

DEFFCD

DEEFDD

DDDDDD

Output:



6.4 Test Case 4 (Grid 9x9)

Input:

AAABBCCCD

ABBBBCECD

ABBBDCECD

AAABDCCCD

BBBBDDDDD

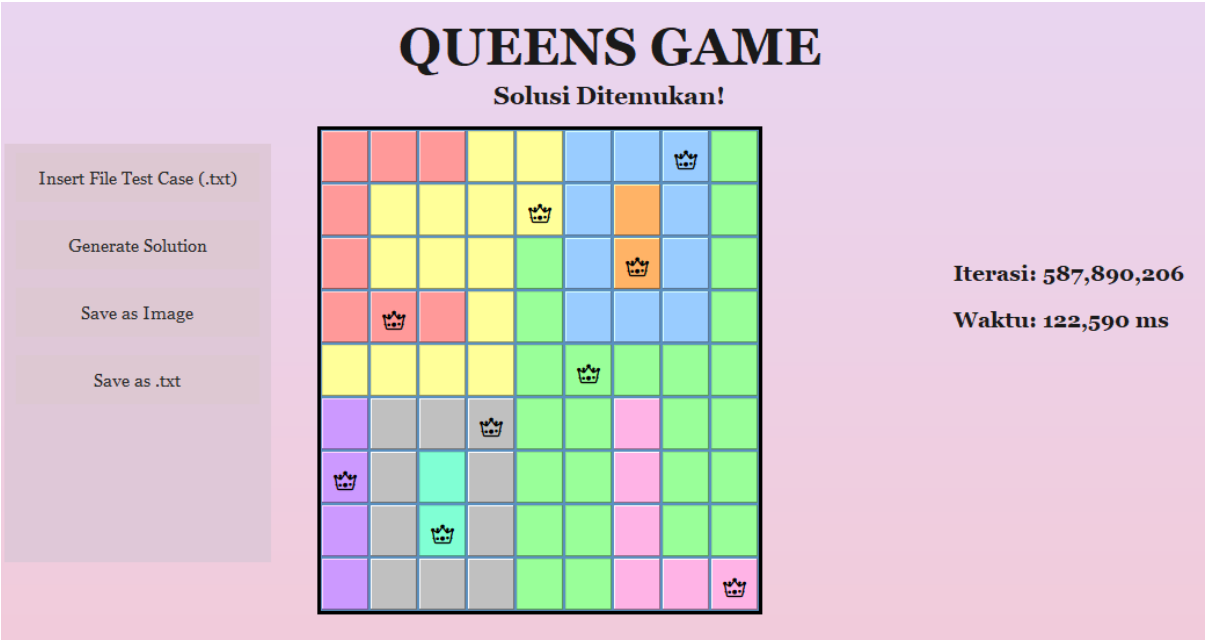
FGGGDDHDD

FGIGDDHDD

FGIGDDHDD

FGGGDDHHH

Output:



6.5 Test Case 5 (Grid 10x10)

Input:

AAABBCCCD

ABBBBCECD

ABBBDCECD

AAABDCCCD

BBBBDDDDD

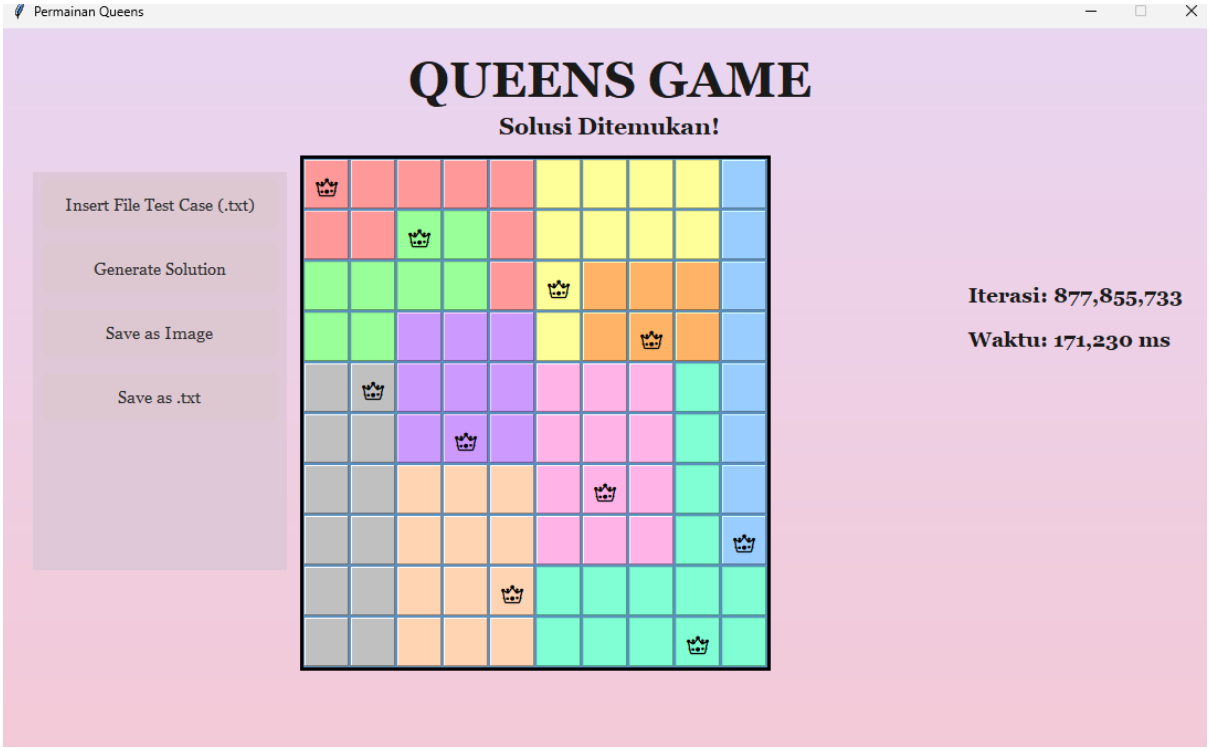
FGGGDDHDD

FGIGDDHDD

FGIGDDHDD

FGGGDDHHH

Output:



BAB VII
REPOSITORY GITHUB

Kode program lengkap tersedia pada link berikut:

https://github.com/juanoloando/Tucil1_13524032.git

BAB VIII
LAMPIRAN

8.1 Pernyataan Tidak Melakukan Kecurangan

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

Juan Oloando Simanungkalit (13524032)

8.2 Checklist Penilaian

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	✓	