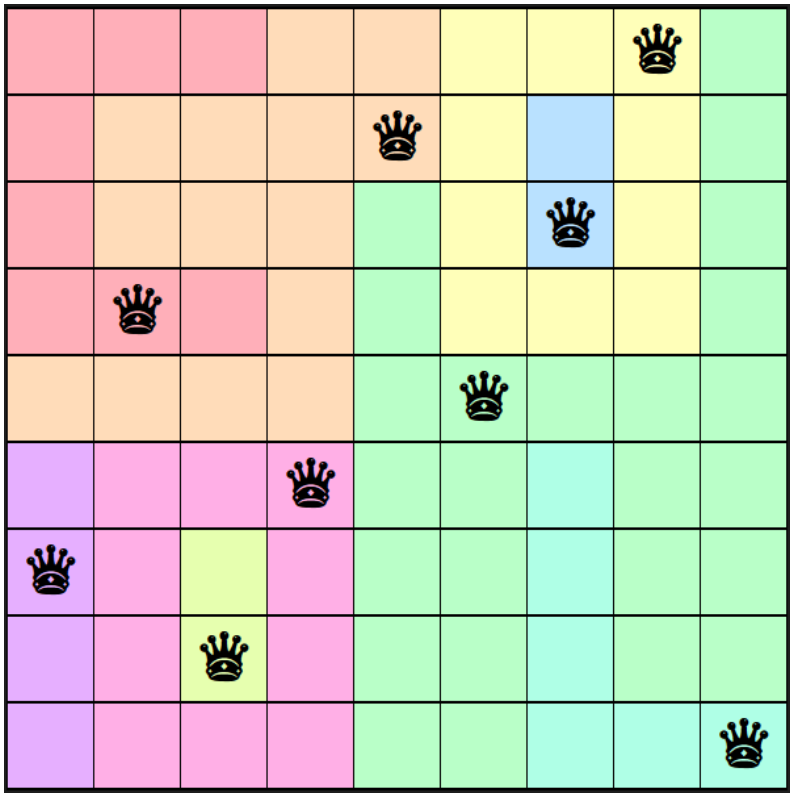
Laporan Tugas Kecil 1

IF 2211 Strategi Algoritma

Penyelesaian Permainan Queens LinkedIn



Oleh:

Wafiq Hibban Robbany / 13524016

13524016@std.stei.itb.ac.id

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

17 Februari 2026

Daftar Isi

Deskripsi Tugas **2**

Desain dan Implementasi **3**

2.1 Algoritma Brute Force 3

2.2 Struktur Program 5

2.3 Source Code Program 5

2.3.1 main.py 5

2.3.2 solver.py 14

Eksperimen **17**

3.1 Test Case #1 17

3.2 Test Case #2 17

3.3 Test Case #3 18

3.4 Test Case #4 19

3.5 Test Case #5 19

3.6 Test Case #6 20

Lampiran **22**

Deskripsi Tugas

Permainan Queens adalah sebuah teka-teki logika menempatkan sejumlah Ratu di atas papan berukuran N x N yang telah terbagi ke dalam beberapa wilayah warna (*region*). Meskipun memiliki kemiripan dengan persoalan *N-Queens*, permainan ini memiliki batasan tambahan terkait pembagian wilayah pada papan yang harus diperhatikan dalam penentuan solusi.

Aturan penempatan Ratu dalam permainan ini adalah sebagai berikut:

- Setiap baris dan setiap kolom hanya boleh ditempati oleh tepat satu Ratu.
- Setiap wilayah warna (*region*) hanya boleh ditempati oleh tepat satu Ratu.
- Dua buah Ratu tidak diperbolehkan saling bersentuhan, baik secara vertikal, horizontal, maupun diagonal pada kotak yang bersebelahan.

Tugas utama penulis adalah merancang dan mengimplementasikan program untuk mencari solusi konfigurasi papan Queens tersebut. Program dibangun menggunakan algoritma *Brute Force* bersifat murni untuk mengeksplorasi kemungkinan penempatan Ratu secara sistematis hingga ditemukan susunan yang memenuhi seluruh batasan aturan yang telah ditetapkan.

Desain dan Implementasi

2.1 Algoritma Brute Force

Algoritma *Brute Force* murni adalah pendekatan pemecahan masalah yang bekerja secara langsung dan naif dengan membangkitkan seluruh kemungkinan solusi, lalu mengujinya satu per satu terhadap kriteria permasalahan. Dalam penyelesaian permainan Queens ini, penulis menggunakan desain *Brute Force* dengan metode *Exhaustive Search* berbasis permutasi. Algoritma didesain dengan menempatkan satu Ratu pada setiap baris secara berurutan, seraya memastikan setiap Ratu menempati kolom yang berbeda. Proses penempatan ini terus berjalan hingga ke-N Ratu selesai diletakkan untuk membentuk suatu konfigurasi papan utuh. Setelah satu konfigurasi utuh terbentuk, barulah program mengujinya untuk memastikan tidak ada pelanggaran terhadap aturan wilayah warna maupun aturan ratu yang saling bersentuhan.

Terkait pemenuhan spesifikasi tugas, penulis berargumen bahwa implementasi algoritma ini tetap berada dalam koridor *Brute Force* murni dan tidak tergolong sebagai algoritma heuristik. Penggunaan strategi penempatan tepat satu Ratu pada setiap baris dan kolom yang berbeda bukanlah sebuah tebakan untuk memotong ruang pencarian secara prematur, melainkan murni merupakan penegakan aturan mutlak dari permainan itu sendiri. Algoritma tidak menggunakan fungsi evaluasi apa pun di tengah jalan, melainkan secara buta dan sistematis membangkitkan seluruh permutasi baris dan kolom yang mungkin, sebelum akhirnya melakukan pengujian kebenaran di akhir tahap.

Adapun tahapan pencarian solusi yang diimplementasikan dalam kode program adalah sebagai berikut:

1. Penulis menginisialisasi daftar kolom yang tersedia. Penempatan dimulai dari baris pertama atau indeks nol.
2. Pada baris yang sedang dievaluasi, algoritma memilih satu kolom dari daftar tersebut untuk diletakkan Ratu, lalu menghapus kolom itu sementara agar tidak digunakan oleh baris selanjutnya.
3. Algoritma memanggil fungsinya sendiri secara rekursif untuk menempatkan Ratu di baris berikutnya menggunakan sisa kolom yang ada. Tahap ini menyusun kombinasi posisi dan diteruskan hingga papan terisi penuh oleh N buah Ratu.
4. Setelah papan terisi penuh dan membentuk satu kandidat solusi lengkap, algoritma memanggil fungsi validasi. Fungsi ini memeriksa apakah ada konfigurasi yang melanggar aturan bersentuhan (termasuk diagonal) dan aturan satu Ratu per wilayah warna.
5. Jika hasil validasi benar, program menjadikan papan tersebut sebagai solusi akhir dan menghentikan pencarian.

2.2 Struktur Program

Program penyelesaian permainan Queens ini dikembangkan menggunakan bahasa pemrograman Python. Untuk antarmuka pengguna grafis (GUI), penulis menggunakan *library* PyQt6, sedangkan untuk fitur pembacaan papan dari gambar, penulis memanfaatkan *library* OpenCV.

Berikut adalah struktur *tree* dari direktori program yang telah dibangun:

```
Tucil1_13524016/
├── bin/
│   └── QueensSolver.exe
├── doc/
├── src/
│   ├── main.py
│   └── solver.py
├── test/
│   ├── tc/
│   └── solution/
└── README.md
```

2.3 Source Code Program

2.3.1 main.py

main.py merupakan implementasi antarmuka pengguna grafis (GUI). File ini bertugas menangani interaksi dengan pengguna serta menampilkan visualisasi papan permainan dan hasil pencarian solusi ke layar.

```python
import sys
import time
import copy
import cv2
import numpy as np
from PyQt6.QtWidgets import (QApplication, QMainWindow, QWidget,
QVBoxLayout,
                             QPushButton, QFileDialog, QLabel,
QMessageBox,
                             QCheckBox, QGroupBox, QSpinBox,
QHBoxLayout, QInputDialog)
from PyQt6.QtGui import QPainter, QColor, QFont
from PyQt6.QtCore import Qt, QRect, QThread, pyqtSignal
from solver import placeQueens, readFile, copyBoard
import solver

def imageToBoard(image_path, N):
    """
    Membaca gambar input dan mengubahnya menjadi matriks karakter A-Z
    """
    img = cv2.imread(image_path)
    if img is None:
        return None

    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    h, w, _ = img.shape
    cell_h = h / N
    cell_w = w / N

    color_list = []
    unique_colors = []
    for r in range(N):
        row_chars = []
        for c in range(N):
            cx = int((c + 0.5) * cell_w)
            cy = int((r + 0.5) * cell_h)
            avg_color = img[cy, cx]

            char_assigned = None
```

```
for i in range(len(unique_colors)):
                uc = unique_colors[i]
                dist = np.linalg.norm(avg_color - uc)
                if dist < 40:
                    char_assigned = chr(65 + i)
                    break
            if char_assigned is None:
                unique_colors.append(avg_color)
                char_assigned = chr(65 + len(unique_colors) - 1)
            row_chars.append(char_assigned)
        color_list.append(row_chars)

    return color_list

class BoardCanvas(QWidget):
    def __init__(self):
        super().__init__()
        self.color_list = None
        self.solution_board = None
        self.N = 0
        self.setMinimumSize(400, 400)
        self.palette = [
            QColor("#FFB3BA"), QColor("#FFDFBA"), QColor("#FFFFBA"),
QColor("#BAFFC9"),
            QColor("#BAE1FF"), QColor("#E6B3FF"), QColor("#FFB3E6"),
QColor("#B3FFE6"),
            QColor("#E6FFB3"), QColor("#B3B3FF"), QColor("#FFD1DC"),
QColor("#C1E1C1"),
            QColor("#F4C2C2"), QColor("#FDFD96"), QColor("#AEC6CF"),
QColor("#77DD77"),
            QColor("#CFCFC4"), QColor("#B39EB5"), QColor("#FFB7B2"),
QColor("#E2F0CB"),
            QColor("#CBAACB"), QColor("#F1CBFF"), QColor("#D0F0C0"),
QColor("#FDEEEA"),
            QColor("#E0BBE4"), QColor("#D4F0F0")
        ]

    def update_board(self, color_list, solution_board, N):
        self.color_list = color_list
        self.solution_board = solution_board
        self.N = N
        self.update()
```

```
def paintEvent(self, e):
        if not self.color_list or self.N == 0:
            return

        painter = QPainter(self)
        cell_size = min(self.width() // self.N, self.height() // self
.N)
        x_offset = (self.width() - (cell_size * self.N)) // 2
        y_offset = (self.height() - (cell_size * self.N)) // 2

        for r in range(self.N):
            for c in range(self.N):
                char_color = self.color_list[r][c]
                color_idx = (ord(char_color.upper()) - 65) % len(self
.palette)
                x_pos = x_offset + (c * cell_size)
                y_pos = y_offset + (r * cell_size)

                painter.setBrush(self.palette[color_idx])
                painter.setPen(QColor(0, 0, 0))
                painter.drawRect(x_pos, y_pos, cell_size, cell_size)
                if self.solution_board and self.solution_board[r][c]
== '#':
                    painter.setFont(QFont("Arial", cell_size // 2,
QFont.Weight.Bold))
                    painter.setPen(QColor(0, 0, 0))
                    rect = QRect(x_pos, y_pos, cell_size, cell_size)
                    painter.drawText(rect, Qt.AlignmentFlag.
AlignCenter, "♛")

class SolverWorker(QThread):
    update_signal = pyqtSignal(list, int)
    finished_signal = pyqtSignal(bool, int, float, list)

    def __init__(self, board, color_list, N, live, interval):
        super().__init__()
        self.board = copyBoard(board)
        self.color_list = color_list
        self.N = N
        self.live_update = live
        self.interval = interval
```

```
def run(self):
        start_time = time.time()
        def live_update(current_board, cases):
            if self.live_update and (cases % self.interval == 0):
                self.update_signal.emit(copyBoard(current_board),
cases)
                time.sleep(0.001)
        is_solved = placeQueens(self.board, self.color_list, self.N,
live_update)
        end_time = time.time()
        exec_time = (end_time - start_time) * 1000
        self.finished_signal.emit(is_solved, solver.total_cases,
exec_time, self.board)

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("LinkedIn Queens Solver by 13524016")
        self.resize(600, 700)

        main_widget = QWidget()
        self.setCentralWidget(main_widget)
        layout = QVBoxLayout(main_widget)

        self.btn_open = QPushButton("Input .txt file")
        self.btn_open.clicked.connect(self.load_file)
        layout.addWidget(self.btn_open)

        self.btn_image = QPushButton("Input as Image")
        self.btn_image.clicked.connect(self.input_image)
        layout.addWidget(self.btn_image)
        self.canvas = BoardCanvas()
        layout.addWidget(self.canvas)

        settings_group = QGroupBox("Settings")
        settings_layout = QHBoxLayout()
        self.cb_animate = QCheckBox("Show Live Update Process")
        self.cb_animate.setChecked(True)
        self.lbl_interval = QLabel("Show every:")
```

```
self.spin_interval = QSpinBox()
        self.spin_interval.setRange(1, 100000)
        self.spin_interval.setValue(500)
        self.spin_interval.setSuffix(" case(s)")
        self.cb_animate.toggled.connect(self.spin_interval.setEnabled
)
        settings_layout.addWidget(self.cb_animate)
        settings_layout.addWidget(self.lbl_interval)
        settings_layout.addWidget(self.spin_interval)
        settings_layout.addStretch()
        settings_group.setLayout(settings_layout)
        layout.addWidget(settings_group)

        self.lbl_status = QLabel("Please input the board file..")
        self.lbl_status.setAlignment(Qt.AlignmentFlag.AlignCenter)
        layout.addWidget(self.lbl_status)

        self.btn_solve = QPushButton("Solve")
        self.btn_solve.setEnabled(False)
        self.btn_solve.clicked.connect(self.solve)
        layout.addWidget(self.btn_solve)

        self.btn_save = QPushButton("Save as Image")
        self.btn_save.setEnabled(False)
        self.btn_save.clicked.connect(self.save_image)
        layout.addWidget(self.btn_save)

        self.btn_save_txt = QPushButton("Save as .txt")
        self.btn_save_txt.setEnabled(False)
        self.btn_save_txt.clicked.connect(self.save_txt)
        layout.addWidget(self.btn_save_txt)

        self.color_list = None
        self.solution_board = None
        self.N = 0
```

```
def load_file(self):
        file_path, _ = QFileDialog.getOpenFileName(self, "
Choose .txt File", "", "Text Files (*.txt)")
        if file_path:
            self.color_list, self.N = readFile(file_path)
            if self.color_list:
                self.solution_board = copyBoard(self.color_list)
                self.canvas.update_board(self.color_list, self.
solution_board, self.N)
                self.lbl_status.setText(f"{self.N}x{self.N}
 Board loaded. Ready to solve.")
                self.btn_solve.setEnabled(True)
            else:
                QMessageBox.warning(self, "Error", "
The file should contain an N×N grid composed of alphabetic character
s.
")

    def input_image(self):
        file_path, _ = QFileDialog.getOpenFileName(self, "
Choose Image", "", "Images (*.png *.jpg *.jpeg)")
        if file_path:
            N, ok = QInputDialog.getInt(self, "Board Size", "
Enter the board size (N):", 9, 4, 26, 1)
            if ok:
                try:
                    self.color_list = imageToBoard(file_path, N)
                    if self.color_list is None:
                        QMessageBox.warning(self, "Error", "
Failed to load image")
                        return
                    self.N = N
                    self.solution_board = copyBoard(self.color_list)
                    self.canvas.update_board(self.color_list, self.
solution_board, self.N)
                    self.lbl_status.setText(f"{self.N}x{self.N}
 Board loaded. Ready to solve.")
                    self.btn_solve.setEnabled(True)
                except Exception as e:
                    QMessageBox.warning(self, "Error", f
"Error accured:\n{e}")
```

```
def solve(self):
        live = self.cb_animate.isChecked()
        interval = self.spin_interval.value()
        self.btn_solve.setEnabled(False)
        self.btn_open.setEnabled(False)
        self.btn_image.setEnabled(False)
        self.lbl_status.setText("Finding Solution..")
        self.worker = SolverWorker(self.solution_board, self.
color_list, self.N, live, interval)
        self.worker.update_signal.connect(self.on_live_update)
        self.worker.finished_signal.connect(self.on_solve_finished)
        self.worker.start()

    def on_live_update(self, current_board, cases):
        self.solution_board = current_board
        self.canvas.update_board(self.color_list, self.solution_board
, self.N)
        self.lbl_status.setText(f"Evaluating case #{cases}...")

    def on_solve_finished(self, is_solved, total_cases, exec_time,
final_board):
        self.solution_board = final_board
        self.canvas.update_board(self.color_list, self.solution_board
, self.N)
        if is_solved:
            self.lbl_status.setText(f
"Solution found! Execution Time: {exec_time:.2f} ms | Cases checked:
{total_cases}")
            self.btn_save.setEnabled(True)
            self.btn_save_txt.setEnabled(True)
        else:
            self.lbl_status.setText(f
"No solution exists. Execution Time: {exec_time:.2f}
 ms | Cases checked: {total_cases}")
            self.btn_save.setEnabled(False)
            self.btn_save_txt.setEnabled(False)
        self.btn_solve.setEnabled(True)
        self.btn_open.setEnabled(True)
        self.btn_image.setEnabled(True)
```

```
def save_image(self):
        file_path, _ = QFileDialog.getSaveFileName(self, "
Save as Image", "solution.png", "Images (*.png *.jpg *.jpeg)")
        if file_path:
            pixmap = self.canvas.grab()
            if pixmap.save(file_path):
                QMessageBox.information(self, "Success", f
"Image successfully saved to:\n{file_path}")
            else:
                QMessageBox.warning(self, "Error", "
Failed to save the image. Please make sure the file extension is corr
ect (.png/.jpg).
")

    def save_txt(self):
        file_path, filter_type = QFileDialog.getSaveFileName(self, "
Save as .txt File", "solution.txt", "Text Files (*.txt)")
        if file_path:
            try:
                with open(file_path, 'w') as f:
                    for r in range(self.N):
                        row_string = ""
                        for c in range(self.N):
                            if self.solution_board[r][c] == '#':
                                row_string += "#"
                            else:
                                row_string += f"{self.color_list[r][c
]}"
                        f.write(row_string.strip() + "\n")

                QMessageBox.information(self, "Success", f
"Text file successfully saved to:\n{file_path}")
            except Exception as e:
                QMessageBox.warning(self, "Error", f"Failed to save:
{e}")

if __name__ == '__main__':
    app = QApplication(sys.argv)
    window = MainWindow()
    window.show()
    sys.exit(app.exec())
```

2.3.2 solver.py

solver.py berperan sebagai modul back-end yang menangani seluruh logika komputasi program. Modul ini memuat implementasi algoritma Brute Force penempatan posisi ratu beserta fungsi validasi terhadap aturan wilayah warna dan persentuhan ratu. Selain algoritma inti, file ini juga dilengkapi dengan fungsi utilitas pendukung, yaitu read_file untuk memproses data papan dari file input teks ke dalam bentuk matriks, serta copy_board untuk menduplikasi papan selama proses pencarian solusi berlangsung.

```python
import os
total_cases = 0

def isValid(queen_pos, color_list, N):
    """
    Validasi ratu dengan aturan:
    - Satu queen pada tiap daerah warna
    - Tidak ada queen yang saling berdekatan baik vertikal, horizonta
l, maupun diagonal
    """
    used_colors = set()
    for r in range(N):
        c = queen_pos[r]
        color = color_list[r][c]
        if color in used_colors:
            return False
        used_colors.add(color)
        if r > 0:
            prev_c = queen_pos[r-1]
            if abs(c-prev_c) <= 1:
                return False

    return True
```

```python
def placeQueens(board, color_list, N, live_update=None, queen_pos=
None, available_pos=None):
    """
    Meletakkan berbagai kemungkinan ratu pada baris dan kolom berbeda
    """
    global total_cases
    if queen_pos == None:
        queen_pos = []
        total_cases = 0
    if available_pos == None:
        available_pos = list(range(N))

    if len(queen_pos) == N:
        total_cases += 1
        if live_update:
            for r in range(N):
                for c in range(N):
                    board[r][c] = '.'
            for r in range(N):
                board[r][queen_pos[r]] = '#'
            live_update(board,total_cases)
        if isValid(queen_pos, color_list, N):
            for r in range(N):
                for c in range(N):
                    board[r][c] = '.'
            for r in range(N):
                board[r][queen_pos[r]] = '#'
            return True
        return False

    for row in range(len(available_pos)):
        col = available_pos[row]
        next_pos = queen_pos + [col]
        next_available = available_pos[:row] + available_pos[row+1:]
        if placeQueens(board, color_list, N, live_update,next_pos,
next_available):
            return True
    return False
```

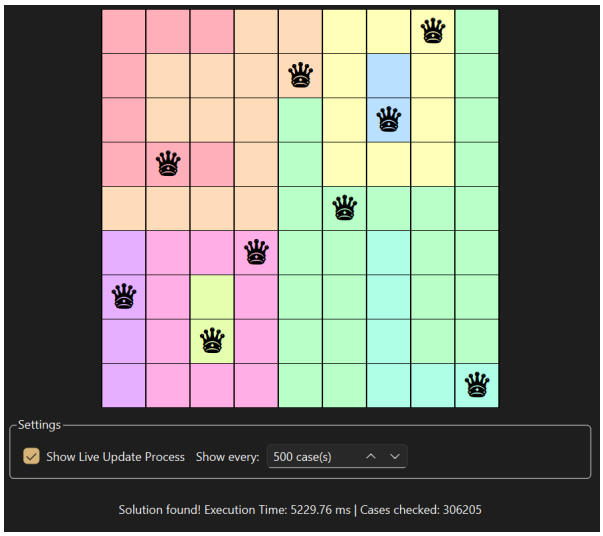
```python
def readFile(path):
    """
    Membaca konten file txt
    Mengembalikan tuple -> (colorList, N)
    colorList: 2D List berisi region warna
    N: Ukuran grid
    """
    if not os.path.exists(path):
        return None, 0
    color_list = []
    with open(path) as f:
        for line in f:
            noSpace = line.strip()
            if noSpace:
                color_list.append(list(noSpace))
    N = len(color_list)
    if N == 0:
        return None, 0
    for i in range(N):
        if len(color_list[i]) != N:
            return None, 0

    return color_list, N

def copyBoard(color_list):
    """
    Membuat salinan colorList untuk menyimpan pencarian solusi
    """
    board = []
    for row in color_list:
        board.append(row.copy())
    return board
```

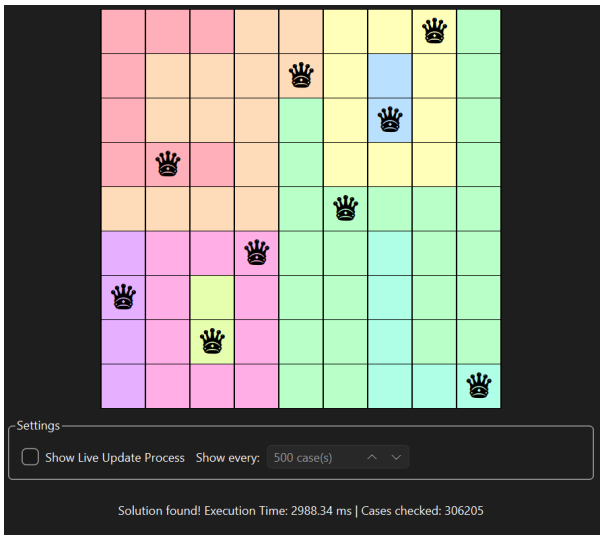
Eksperimen

3.1 Test Case #1

Input	Output
Input .txt file AAABBCCCD ABBBBCECD ABBBDCECD AAABDCCCD BBBBDDDDD FGGGDDHDD FGIGDDHDD FGIGDDHDD FGGGDDHHH	Live Update Enabled 

3.2 Test Case #2

Input	Output

Input .txt file	Live Update Disabled
AAABBCCCD ABBBBCECD ABBBDCECD AAABDCCCD BBBBDDDDD FGGGDDHDD FGIGDDHDD FGIGDDHDD FGGGDDHHH	

3.3 Test Case #3

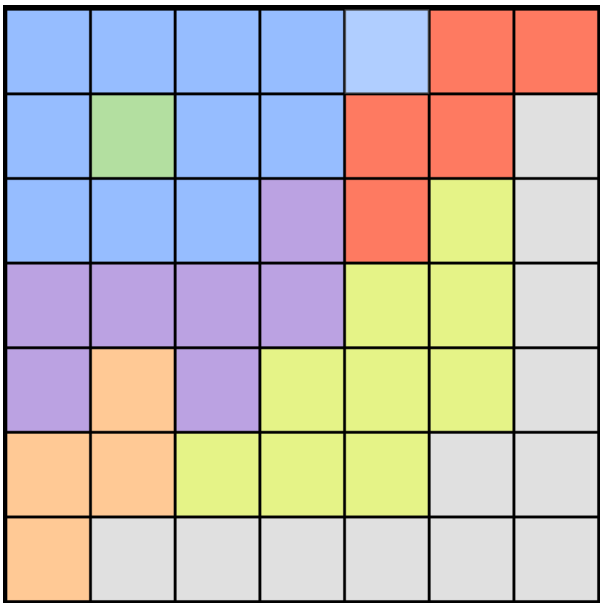
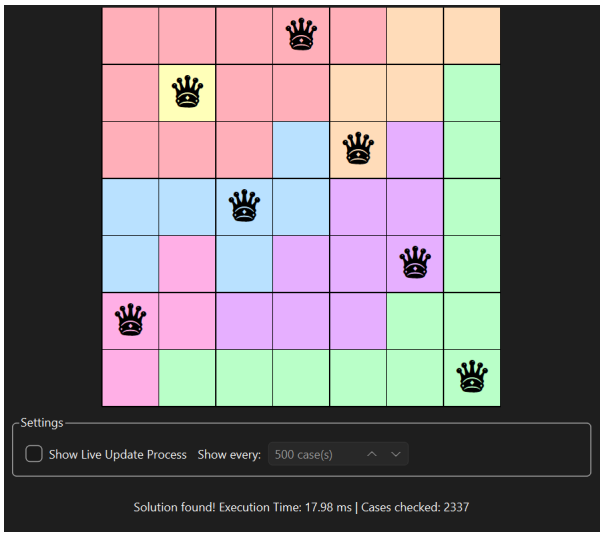
Input	Output
Input .txt file AAABBCCCD ABBBBCECD ABBBDCECD AAABDCCCD BBBBDDDDD FGGGDDHDD FGIGDDHDD FGIGDDHDD FGGGDDHHH	Output as image 

3.4 Test Case #4

Input	Output
Input .txt file AAABBCCCD ABBBBCECD ABBBDCECD AAABDCCCD BBBBDDDDD FGGGDDHDD FGIGDDHDD FGIGDDHDD FGGGDDHHH	Output as text file AAABBCC#D ABBB#CECD ABBBDC#CD A#ABDCCCD BBBBD#DDD FGG#DDHDD #GIGDDHDD FG#GDDHDD FGGGDDHH#

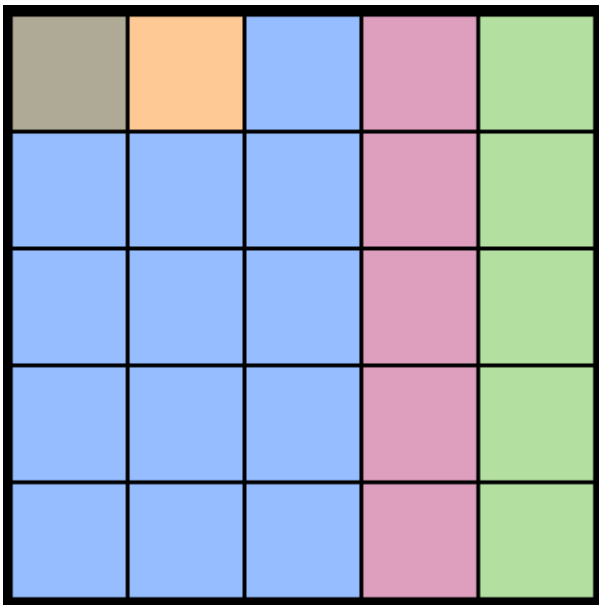
3.5 Test Case #5

Input	Output

Input as image	
	

3.6 Test Case #6

Input	Output

Input as image	No Solution
	

Lampiran

Tautan Repository Github: github.com/wafhr/Tucil1_13524016

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	✔	

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

Wafiq Hibban Robbany