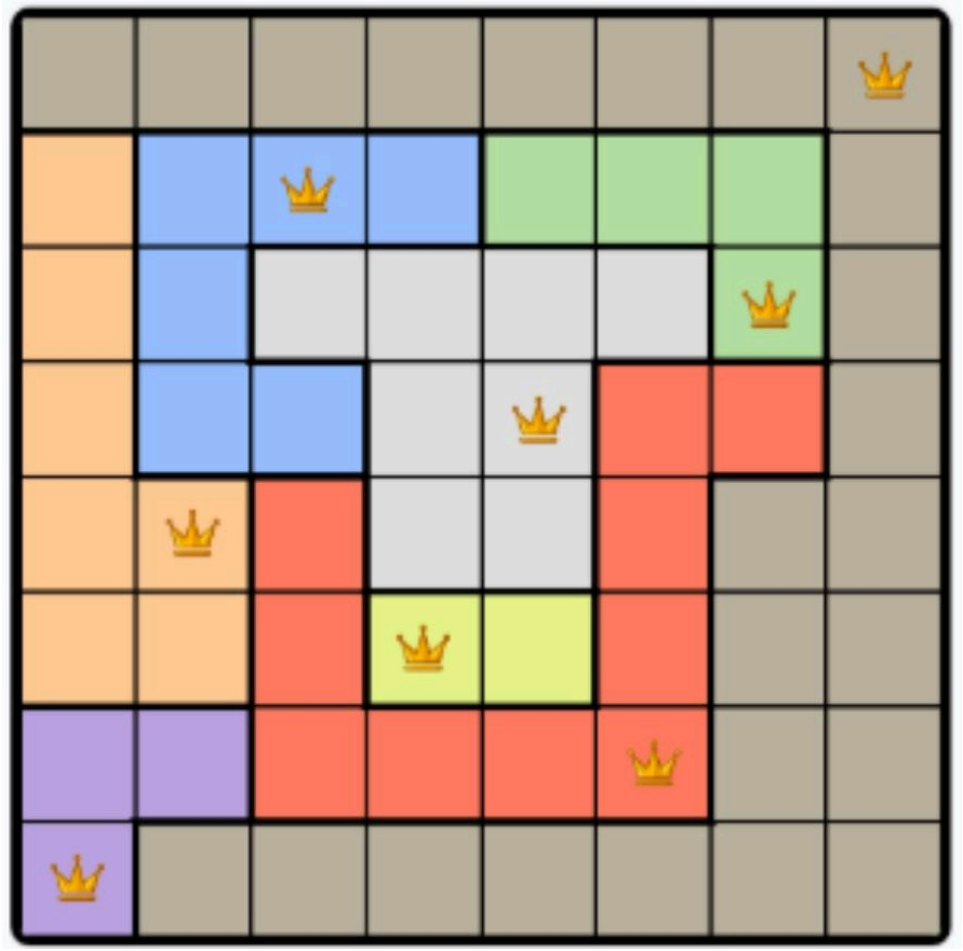
Laporan Tugas Kecil 1

IF2211 Strategi Algoritma

Penyelesaian Permainan Queens Linkedin Dengan Brute-Force

Semester II Tahun 2025/2026



Disusun oleh:

Bryan Pratama Putra Hendra - 13524067

Laboratorium Ilmu dan Rekayasa Komputasi

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

DAFTAR ISI

1. Pendahuluan

1.1 Deskripsi

Queens merupakan sebuah puzzle logika yang tersedia pada platform jejaring profesional LinkedIn. Dalam permainan ini, pemain ditantang untuk menempatkan queen pada papan persegi yang terdiri dari beberapa daerah warna sehingga memenuhi aturan berikut:

- Hanya satu queen berada pada setiap baris.
- Hanya satu queen berada pada setiap kolom.
- Hanya satu queen berada pada setiap daerah warna yang sama.
- Tidak ada dua queen yang ditempatkan bersebelahan, termasuk secara diagonal.

Permainan ini menarik untuk dijadikan studi kasus dalam pengembangan algoritma karena meskipun konsepnya sederhana, jumlah kemungkinan konfigurasi papan meningkat sangat cepat seiring bertambahnya ukuran papan. Oleh karena itu, implementasi algoritma yang efisien menjadi penting untuk menyelesaikan masalah ini secara komputasional.

1.2 Tujuan

Tujuan dari tugas ini adalah mengembangkan program yang mampu menemukan satu solusi valid dalam permainan Queens menggunakan algoritma *brute force*. Program harus dapat membaca konfigurasi papan awal dari file input, menempatkan queen sesuai aturan permainan, menampilkan hasil solusi, menampilkan *live update* dari proses pencarian dengan *brute force*, serta menghitung jumlah konfigurasi yang diperiksa dan waktu eksekusi. Selain itu, versi optimasi dari algoritma juga dikembangkan untuk meningkatkan efisiensi komputasi dan mendukung visualisasi proses pencarian solusi.

1.3 Ide Penerapan

Untuk menyelesaikan masalah ini, penulis mengembangkan solusi yang menggabungkan *backend* dengan bahasa pemrograman C++ dan *frontend* dengan bahasa pemrograman Python. *Backend* C++ bertanggung jawab untuk logika inti algoritma brute force, termasuk validasi posisi queen, backtracking, dan optimisasi. Bahasa Python digunakan sebagai *frontend* untuk visualisasi, dan menampilkan papan secara interaktif. Integrasi antara C++ dan Python dilakukan menggunakan binding, sehingga fungsi-fungsi backend dapat dipanggil langsung dari Python tanpa kehilangan performa komputasi.

Pendekatan ini memungkinkan:

- Pemanfaatan kecepatan eksekusi bahasa pemrograman C++ untuk pencarian solusi brute force.
- Kemudahan bahasa pemrograman Python dalam menampilkan GUI atau output interaktif.

2. Implementasi

Dalam permainan Queens, pendekatan brute force digunakan untuk menemukan solusi dengan cara mengeksplorasi seluruh kemungkinan susunan ratu pada papan berukuran N × N. Metode ini bekerja secara rekursif, dengan menempatkan satu ratu pada setiap baris secara bergantian, kemudian melakukan pemeriksaan aturan setiap kali papan diisi penuh. Aturan yang harus dipatuhi meliputi: hanya satu ratu per kolom, tidak ada dua ratu yang berdekatan baik secara horizontal, vertikal, maupun diagonal dalam jarak satu petak, dan setiap ratu berada pada daerah warna yang berbeda. Dengan strategi ini, algoritma dapat menjamin bahwa setiap konfigurasi yang diperiksa valid sesuai kriteria permainan, meskipun memerlukan eksplorasi kombinatorial yang ekstensif.

2.1 Brute Force

Algoritma *brute force* yang diimplementasikan penulis, yaitu melakukan enumerasi terhadap semua kemungkinan penempatan queen pada papan berdasarkan aturan permainan yang dijelaskan pada pendahuluan. Pada bagian ini, akan dijelaskan bagaimana cara fungsi *brute force* yang digunakan pada program ini bekerja.

Prosedur utama *brute force* terdapat pada fungsi *bruteforce(int row)* dan versi optimasinya *bruteforceopt(int row)*. Alur kerjanya secara sederhana adalah sebagai berikut:

- Pemanggilan pertama dilakukan dengan parameter row = 0, serta papan awal dari input pengguna dan array *currentPlacement* yang masih kosong.
- Algoritma mencoba menempatkan queen pada setiap kolom di baris saat ini (row), kemudian melakukan pemanggilan rekursif ke baris berikutnya (row + 1).
- Setelah semua baris terisi, dilakukan pengecekan apakah konfigurasi papan valid, termasuk aturan kolom unik, larangan queen berdekatan, dan aturan warna.
- Jika valid, array solution diupdate dengan posisi queen yang sah. Lalu dilihatkan kepada pengguna solusi dari board queens yang diberikan.

Untuk versi optimasi (*bruteforceopt*), algoritma memanfaatkan pengecekan sebagian kondisi selama proses rekursi, termasuk memeriksa kolom dan warna yang sudah terpakai, sehingga beberapa cabang yang jelas akan gagal dapat dilewati tanpa diuji.

Dalam implementasi *brute force* tanpa optimasi, kompleksitas waktu berada pada orde eksponensial O(n^n), karena setiap baris harus mencoba semua kolom yang tersedia berdasarkan konfigurasi baris sebelumnya. Optimasi yang diterapkan dapat mengurangi jumlah kasus yang diperiksa secara signifikan, tetapi kompleksitas terburuk tetap eksponensial.

2.2 Pseudocode

isValidPlacement dan *bruteforce* menunjukkan alur algoritma brute force murni untuk permainan Queens. Fungsi *isValidPlacement* digunakan untuk memeriksa apakah seluruh papan yang telah ditempati queen memenuhi aturan permainan, yaitu setiap kolom hanya berisi satu queen, queen tidak saling bersinggungan dalam radius satu petak, dan setiap queen berada pada warna yang berbeda. Sedangkan prosedur bruteforce melakukan penempatan queen secara rekursif, mencoba setiap kolom pada setiap baris, kemudian memanggil dirinya sendiri untuk baris berikutnya. Jika seluruh baris terisi, algoritma memanggil *isValidPlacement* untuk memastikan konfigurasi valid, lalu memperbarui papan solusi. Prosedur ini juga mendukung fitur live update dengan memanggil *stepCallback* secara periodik, sehingga proses brute force dapat divisualisasikan secara bertahap.

```
function isValidPlacement()
```

```
    for r1 = 0 to n - 1 do
        for r2 = r1 + 1 to n - 1 do
            c1 = currentplacement[r1]
            c2 = currentplacement[r2]

            if c1 = c2 then
                return false
            end if

            if abs(r1 - r2) ⩽ 1 and abs(c1 - c2) ⩽ 1 then
                return false
            end if

            if board[r1][c1] = board[r2][c2] then
                return false
            end if
        end for
    end for

  return true
end function
```

```
      procedure bruteforce(row)
          if found = true then
              return
          end if

          if row = n then
              countcases = countcases + 1

              if isValidPlacement() = true then
                  found = true
                  for r = 0 to n - 1 do
                      solution[r][currentplacement[r]] = '#'
                  end for

                  if stepcallback is defined then
                      call stepcallback(solution)
                  end if
              end if
              return
          end if

          for col = 0 to n - 1 do
```

```
            currentplacement[row] = col
            solution[row][col] = '#'
            cnt = cnt + 1

            if stepcallback is defined and cnt mod 50000 = 0 then
                call stepcallback(solution)
            end if

            call bruteforce(row + 1)

            if found = true then
                return
            end if

            solution[row][col] = '.'

            if stepcallback is defined and cnt mod 50000 = 0 then
                call stepcallback(solution)
            end if

        end for
    end procedure
```

Pada versi optimasi, prosedur *bruteforceopt* menggunakan pengecekan sebagian kondisi pada fungsi *isSafe* sebelum melakukan rekursi. Hal ini memungkinkan algoritma melewati beberapa cabang yang jelas akan gagal, sehingga mengurangi jumlah kasus yang perlu diperiksa. Selain itu, algoritma tetap menempatkan queen secara rekursif pada setiap baris, namun hanya melanjutkan ke baris berikutnya jika posisi kolom saat ini aman. Prosedur ini juga mendukung fitur live update dengan memanggil *stepCallback* secara periodik dan pemanggilan ini lebih cepat daripada yang *brute force* murni karena iterasi lebih sedikit, sehingga proses *brute force* dapat divisualisasikan secara bertahap. Dengan strategi ini, algoritma tetap ada unsur *brute force* namun lebih efisien dibandingkan versi tanpa optimasi.

```
    function isSafe(row, col)
        for r = 0 to row - 1 do
            c = currentplacement[r]

            if c = col then
                return false
            end if

            if abs(r - row) ⩽ 1 and abs(c - col) ⩽ 1 then
                return false
            end if

            if board[r][c] = board[row][col] then
```

```
            return false
        end if
    end for

    return true
end function
```

```
procedure bruteforceopt(row)
    if found = true then
        return
    end if

    if row = n then
        found = true
        if stepcallback is defined then
            call stepcallback(solution)
        end if
        return
    end if

    for col = 0 to n - 1 do
        countcases = countcases + 1
        cnt = cnt + 1

        if not isSafe(row, col) then
            continue
        end if

        currentplacement[row] = col
        solution[row][col] = '#'

        if stepcallback is defined and cnt mod 50 = 0 then
            call stepcallback(solution)
        end if

        call bruteforceopt(row + 1)

        if found = true then
            return
        end if

        solution[row][col] = '.'

        if stepcallback is defined and cnt mod 50 = 0 then
            call stepcallback(solution)
        end if
```

```
        end for
    end procedure
```

2.3 Struktur Proyek

```
├── assets/
│   └── ...
├── docs/
│   └── Laporan_Tucil1.pdf
├── src/
│   ├── app.py
│   ├── bindings.cpp
│   ├── board.cpp
│   ├── board.h
│   ├── queens.cpp
│   ├── queens.h
│   ├── queensSolver.cpp
│   └── queensSolver.h
├── test/
│   ├── input/
│   │   └── ...
│   └── output/
│       └── ...
├── CMakeList.txt
├── doc/
└── README.md
```

2.4 Source Code

Berikut merupakan source code dari program yang diimplementasikan untuk menyelesaikan permainan Queens.

2.4.1 board.h

```
#ifndef BOARD_H
#define BOARD_H

#include <iostream>
#include <vector>
#include <unordered_set>

using namespace std;

class Board
```

```
{
private:
    vector<vector<char>> board;
    int size = 0;

public:
    Board();
    void setBoard(const vector<vector<char>> &inputBoard);
    bool isSquare() const;
    bool isValidBoard() const;
    int getSize() const;
    char getElmt(int row, int col) const;
    const vector<vector<char>> &getBoard() const;
    int countDistinctColors() const;
};

#endif
```

2.4.2 board.cpp

```
#include "board.h"

Board::Board() {}

void Board::setBoard(const vector<vector<char>> &inputBoard)
{
    board = inputBoard;
    size = board.size();
}

bool Board::isSquare() const
{
    for (int i = 0; i < size; i++)
    {
        if ((int)board[i].size() != size) return false;
    }
    return true;
}

bool Board::isValidBoard() const
{
    if (size == 0) return false;
    if (!isSquare()) return false;

    return true;
}
```

```
int Board::getSize() const
{
   return size;
}

char Board::getElmt(int row, int col) const
{
   return board[row][col];
}

const vector<vector<char>> &Board::getBoard() const
{
   return board;
}

int Board::countDistinctColors() const
{
   std::unordered_set<char> colors;
   for (const auto &row : board)
   {
       for (char c : row) colors.insert(c);
   }
   return static_cast<int>(colors.size());
}
```

2.4.3 queens.h

```
#ifndef QUEENS_H
#define QUEENS_H

class Queen
{
private:
   int row;
   int col;

public:
   Queen();
   Queen(int r, int c);
   int getRow() const;
   int getCol() const;
   void setPosition(int r, int c);
};

#endif
```

2.4.4 queens.cpp

```
#include "queens.h"

Queen::Queen() : row(-1), col(-1) {}

Queen::Queen(int r, int c)
{
   row = r;
   col = c;
}

int Queen::getRow() const
{
   return row;
}

int Queen::getCol() const
{
   return col;
}

void Queen::setPosition(int r, int c)
{
   row = r;
   col = c;
}
```

2.4.5 queensSolver.h

```
#ifndef QUEENSSOLVER_H
#define QUEENSSOLVER_H

#include <chrono>
#include <vector>
#include <iostream>
#include <functional>
#include <thread>

#include "board.h"
#include "queens.h"
```

```
using namespace std;

struct SolveResult
{
   vector<vector<char>> solution;
   vector<vector<char>> board;
   long long countCases;
   long long durationMs;
   bool found;
};

class QueensSolver
{
private:
   const Board &board;
   vector<vector<char>> solution;
   vector<Queen> queens;
   vector<int> currentPlacement;
   function<void(const vector<vector<char>> &)> stepCallback;

   int n;
   long long countCases = 0, cnt = 0;
   bool found = false;
   bool optimized;

   bool isSafe(int row, int col);
   bool isValidPlacement();
   void bruteforceopt(int row);
   void bruteforce(int row);

public:
    QueensSolver(const Board &b, bool optimized, function<void(const
vector<vector<char>> &)> callback = nullptr);
   SolveResult solve();
};

#endif
```

2.4.6 queensSolver.cpp

```
#include "queensSolver.h"

QueensSolver::QueensSolver(const Board &b, bool optimized, function<void(const vector<vector<char>> &)> callback) : board(b), optimized(optimized), stepCallback(callback)
{
    n = board.getSize();
    solution = board.getBoard();
    queens.resize(n);
    currentPlacement.resize(n);
}

bool QueensSolver::isSafe(int row, int col)
{
    for (int r = 0; r < row; r++)
    {
        int c = currentPlacement[r];
        if (c == col) return false;

        if (abs(r - row) <= 1 && abs(c - col) <= 1) return false;

          if (board.getElmt(r, c) == board.getElmt(row, col)) return false;
    }

    return true;
}

bool QueensSolver::isValidPlacement()
{
    for (int r1 = 0; r1 < n; r1++)
    {
        for (int r2 = r1 + 1; r2 < n; r2++)
        {
            int c1 = currentPlacement[r1];
            int c2 = currentPlacement[r2];

            if (c1 == c2) return false;

               if (abs(r1 - r2) <= 1 && abs(c1 - c2) <= 1) return false;

                if (board.getElmt(r1, c1) == board.getElmt(r2, c2)) return false;
        }
    }
```

```
    return true;
}

void QueensSolver::bruteforceopt(int row)
{
    if (found) return;

    if (row == n)
    {
        found = true;
        if (stepCallback) stepCallback(solution);
        return;
    }

    for (int col = 0; col < n; col++)
    {
        countCases++;
        cnt++;

        if (!isSafe(row, col)) continue;

        currentPlacement[row] = col;
        solution[row][col] = '#';

        if (stepCallback && cnt % 50 == 0) stepCallback(solution);

        bruteforceopt(row + 1);

        if (found) return;

        solution[row][col] = '.';

        if (stepCallback && cnt % 50 == 0) stepCallback(solution);
    }
}

void QueensSolver::bruteforce(int row)
{
    if (found) return;

    if (row == n)
    {
        countCases++;

        if (isValidPlacement())
        {
            found = true;
```

```
                                for (int r = 0; r < n; r++)
solution[r][currentPlacement[r]] = '#';

                if (stepCallback) stepCallback(solution);
            }
            return;
        }

        for (int col = 0; col < n; col++)
        {
            currentPlacement[row] = col;
            solution[row][col] = '#';

            cnt++;

                        if (stepCallback && cnt % 50000 == 0)
stepCallback(solution);

            bruteforce(row + 1);

            if (found) return;

            solution[row][col] = '.';

                        if (stepCallback && cnt % 50000 == 0)
stepCallback(solution);
        }
    }

    SolveResult QueensSolver::solve()
    {
       int distinctColors = board.countDistinctColors();
       if (distinctColors ≠ n)
       {
            SolveResult result;
            result.countCases = 0;
            result.durationMs = 0;
            result.found = false;
            result.board = board.getBoard();
            result.solution = {};
            return result;
       }

       auto start = std::chrono::high_resolution_clock::now();

       if (optimized) bruteforceopt(0);
       else bruteforce(0);

       auto end = chrono::high_resolution_clock::now();
```

```
          auto duration = chrono::duration_cast<chrono::milliseconds>(end
- start);

          SolveResult result;
          result.countCases = countCases;
          result.durationMs = duration.count();
          result.found = found;
          result.board = board.getBoard();

          if (found) result.solution = solution;
          else result.solution = {};

          return result;
      }
```

2.4.7 bindings.cpp

```
      #include <pybind11/pybind11.h>
      #include <pybind11/stl.h>

      #include <fstream>
      #include <vector>
      #include <string>

      #include "board.h"
      #include "queensSolver.h"

      namespace py = pybind11;

      std::vector<std::vector<char>>   loadBoardFile(const   std::string&
filename)
      {
          std::ifstream file(filename);
              if  (!file.is_open())  throw  std::runtime_error("File  tidak
ditemukan");

          std::vector<std::vector<char>> tempBoard;
          std::string line;

          while (getline(file, line))
          {
              std::vector<char> row;
              for (char c : line) row.push_back(c);
              tempBoard.push_back(row);
          }
```

```
        file.close();
        return tempBoard;
    }

    SolveResult    solveBoard(const    std::vector<std::vector<char>>&
boardInput,
        bool optimized,
        py::function callback = py::none())
    {
        Board board;
        board.setBoard(boardInput);

        if (!board.isValidBoard()) throw std::runtime_error("Board tidak
valid");

            std::function<void(const  std::vector<std::vector<char>>&)>
stepCallback = nullptr;

        if (!callback.is_none())
        {
                                    stepCallback    =    [callback](const
std::vector<std::vector<char>>& state)
            {
                py::gil_scoped_acquire acquire;
                callback(state);
            };
        }

        QueensSolver solver(board, optimized, stepCallback);
        return solver.solve();
    }

    PYBIND11_MODULE(queens_solver, m)
    {
        py::class_<SolveResult>(m, "SolveResult")
            .def_readonly("solution", &SolveResult::solution)
            .def_readonly("countCases", &SolveResult::countCases)
            .def_readonly("durationMs", &SolveResult::durationMs)
            .def_readonly("board", &SolveResult::board)
            .def_readonly("found", &SolveResult::found);

        m.def("loadBoardFile", &loadBoardFile);
        m.def("solveBoard", &solveBoard,
            py::arg("boardInput"),
            py::arg("optimized") = true,
            py::arg("callback") = py::none());
    }
```

2.4.8 app.py

```
import tkinter as tk
import queens_solver
import os
import colorsys
import threading
from tkinter import filedialog, messagebox
from PIL import Image, ImageTk, ImageDraw

current_board = None
current_solution = None
current_input_path = None
current_result = None
BOARD_PIXEL_SIZE = 600

def live_update(state):
   canvas.delete("queen")

   n = len(state)
   cell_size = BOARD_PIXEL_SIZE / n

   for i in range(n):
       for j in range(n):
           if state[i][j] == '#':
               x = j * cell_size + cell_size / 2
               y = i * cell_size + cell_size / 2

               canvas.create_image(
                   x,
                   y,
                   image=queen_img,
                   tags="queen"
               )

   root.update_idletasks()

def generate_colors(n=26):
   colors = []
   for i in range(n):
       hue = i / n
       r, g, b = colorsys.hsv_to_rgb(hue, 0.5, 0.95)

colors.append(f"#{int(r*255):02x}{int(g*255):02x}{int(b*255):02x}")
       return colors
```

```
REGION_COLORS = generate_colors(26)

def load_image(path, width):
   img = Image.open(path).convert("RGBA")
   ratio = width / img.width
   height = int(img.height * ratio)
   img = img.resize((width, height), Image.LANCZOS)
   return ImageTk.PhotoImage(img)

def load_file():
   global current_board, current_input_path

   file_path = filedialog.askopenfilename(
       title="Pilih file input",
       filetypes=[("Text files", "*.txt")]
   )

   if not file_path:
       return

   try:
       current_board = queens_solver.loadBoardFile(file_path)
       current_input_path = file_path
       status_label.config(text="Board loaded", fg="black")
       draw_board_only(current_board)

   except Exception as e:
       messagebox.showerror("Error", str(e))

def show_solution():
   global current_board, current_solution, current_result

   if current_board is None:
       messagebox.showwarning("Warning", "No board to solve.")
       return

   def run_solver():
       global current_board, current_solution, current_result
       try:
           result = queens_solver.solveBoard(
               current_board,
               optimized_var.get(),
               live_update
           )

           current_solution = result.solution
           current_result = result

           if result.found:
```

```
                    status_label.config(
                                        text=f"Solusi ditemukan | Waktu:
{result.durationMs} ms | Cases: {result.countCases}",
                        fg="green"
                    )
                else:
                    status_label.config(
                                        text=f"Solusi tidak ditemukan | Waktu:
{result.durationMs} ms | Cases: {result.countCases}",
                        fg="red"
                    )

            except Exception as e:
                messagebox.showerror("Error", str(e))

        threading.Thread(target=run_solver).start()

    def draw_board_only(board):
        canvas.delete("all")

        n = len(board)
        cell_size = BOARD_PIXEL_SIZE / n

        canvas.config(width=BOARD_PIXEL_SIZE, height=BOARD_PIXEL_SIZE)

        for i in range(n):
            for j in range(n):
                letter = board[i][j]
                idx = ord(letter) - ord('A')
                color = REGION_COLORS[idx % 26]

                x1 = j * cell_size
                y1 = i * cell_size
                x2 = x1 + cell_size
                y2 = y1 + cell_size

                canvas.create_rectangle(
                    x1, y1, x2, y2,
                    fill=color,
                    outline="black",
                    width=2
                )

    def clear_board():
            global current_board, current_solution, current_result,
current_input_path

        current_board = None
        current_solution = None
```

```
        current_result = None
        current_input_path = None

        canvas.delete("all")
        status_label.config(text="Board cleared", fg="black")

    def save_as_txt():
        if current_solution is None or current_input_path is None:
            messagebox.showwarning("Warning", "No solution to save.")
            return

        os.makedirs("test/output", exist_ok=True)

                                                base_name              =
os.path.splitext(os.path.basename(current_input_path))[0]
                        output_path     =     os.path.join("test/output",
f"{base_name}_solution.txt")

        with open(output_path, "w") as f:
            for i, row in enumerate(current_board):
                line = ""
                for j, cell in enumerate(row):
                    if current_solution[i][j] == '#':
                        line += '#'
                    else:
                        line += cell
                f.write(line + "\n")

            f.write("\n")
                 f.write(f"Waktu  Pencarian:  {current_result.durationMs}
ms\n")
            f.write(f"Jumlah Cases: {current_result.countCases}")

        messagebox.showinfo("Success", f"Saved to {output_path}")

    def save_as_image():
        if current_solution is None or current_input_path is None:
            messagebox.showwarning("Warning", "No solution to save.")
            return

        os.makedirs("test/output", exist_ok=True)

                                                base_name              =
os.path.splitext(os.path.basename(current_input_path))[0]
                        output_path     =     os.path.join("test/output",
f"{base_name}_solution.png")

        n = len(current_board)
        board_size = BOARD_PIXEL_SIZE
```

```
        cell_size = board_size / n

        img = Image.new("RGB", (board_size, board_size), "#f2dfb2")
        draw = ImageDraw.Draw(img)

        queen_original = Image.open("assets/QLOGO.png").convert("RGBA")
        queen_size = int(cell_size * 0.7)
              queen_img_resized   =   queen_original.resize((queen_size,
queen_size), Image.LANCZOS)

        for i in range(n):
            for j in range(n):
                letter = current_board[i][j]
                idx = ord(letter) - ord('A')
                color = REGION_COLORS[idx % 26]

                x1 = j * cell_size
                y1 = i * cell_size
                x2 = x1 + cell_size
                y2 = y1 + cell_size

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

                if current_solution[i][j] == '#':
                    queen_x = int(x1 + (cell_size - queen_size) / 2)
                    queen_y = int(y1 + (cell_size - queen_size) / 2)
                        img.paste(queen_img_resized, (queen_x, queen_y),
queen_img_resized)

        img.save(output_path)
        messagebox.showinfo("Success", f"Saved to {output_path}")

    root = tk.Tk()
    root.title("Queens Solver by Bryan")
    root.geometry("1200x800")
    root.configure(bg="#f2dfb2")
    optimized_var = tk.BooleanVar(value=True)

    root.grid_columnconfigure(0, weight=0)
    root.grid_columnconfigure(1, weight=0)
    root.grid_columnconfigure(2, weight=1)
    root.grid_rowconfigure(0, weight=1)

    sidebar = tk.Frame(root, bg="#f2dfb2", width=380)
    sidebar.grid(row=0, column=0, sticky="ns")
    sidebar.grid_propagate(False)

    inner_sidebar = tk.Frame(sidebar, bg="#f2dfb2")
```

```
      inner_sidebar.pack(fill="both", expand=True)

      logo_img = load_image("assets/QUEENS.png", 320)
      queen_img = load_image("assets/QLOGO.png", 65)

      logo_label = tk.Label(inner_sidebar, image=logo_img, bg="#f2dfb2")
      logo_label.pack(pady=(20, 90))

      def image_button(parent, img, command):
         return tk.Button(
             parent,
             image=img,
             command=command,
             bd=0,
             highlightthickness=0,
             relief="flat",
             bg="#f2dfb2",
             activebackground="#f2dfb2",
             cursor="hand2"
         )

      load_btn_img = load_image("assets/LOADTXT.png", 240)
      solve_btn_img = load_image("assets/SOLVE.png", 240)
      save_txt_img = load_image("assets/SAVETXT.png", 240)
      save_img_img = load_image("assets/SAVEIMG.png", 240)
      clear_btn_img = load_image("assets/CLEAR.png", 240)

      button_frame = tk.Frame(inner_sidebar, bg="#f2dfb2")
      button_frame.pack()

      image_button(button_frame, load_btn_img, load_file).pack(pady=12)
      image_button(button_frame,                              solve_btn_img,
show_solution).pack(pady=(12, 4))

      # ===== CHECKBOX =====
      optimized_checkbox = tk.Checkbutton(
         button_frame,
         text="Optimized",
         variable=optimized_var,
         onvalue=True,
         offvalue=False,
         bg="#f2dfb2",
         activebackground="#f2dfb2",
         font=("Helvetica", 12, "bold"),
         fg="black"
      )
      optimized_checkbox.pack(pady=(0, 12))

      image_button(button_frame, save_txt_img, save_as_txt).pack(pady=12)
```

```
      image_button(button_frame,                                    save_img_img,
save_as_image).pack(pady=12)
      image_button(button_frame,                                  clear_btn_img,
clear_board).pack(pady=12)

      separator = tk.Frame(root, bg="#c5b58a", width=3)
      separator.grid(row=0, column=1, sticky="ns")

      board_area = tk.Frame(root, bg="#f2dfb2")
      board_area.grid(row=0, column=2, sticky="nsew")

      status_label = tk.Label(
         board_area,
         text="Load a board to begin",
         font=("Helvetica", 16, "bold"),
         bg="#f2dfb2",
         fg="black"
      )
      status_label.pack(pady=20)

      canvas = tk.Canvas(
         board_area,
         bg="#f2dfb2",
         highlightthickness=0
      )
      canvas.pack(expand=True)

      root.mainloop()
```

3. Pengujian

Untuk keperluan pengujian, penulis menyiapkan beberapa file input dalam format .txt. File-file tersebut disimpan pada direktori test/input dan digunakan untuk memvalidasi algoritma serta memeriksa hasil keluaran program.

3.1 Test 1 : Papan Berukuran 9 x 9

3.1.1 File Input

```
AAABBCCCD
ABBBBCECD
ABBBDCECD
AAABDCCCD
BBBBDDDDD
FGGGDDHDD
```

```
FGIGDDHDD
FGIGDDHDD
FGGGDDHHH
```

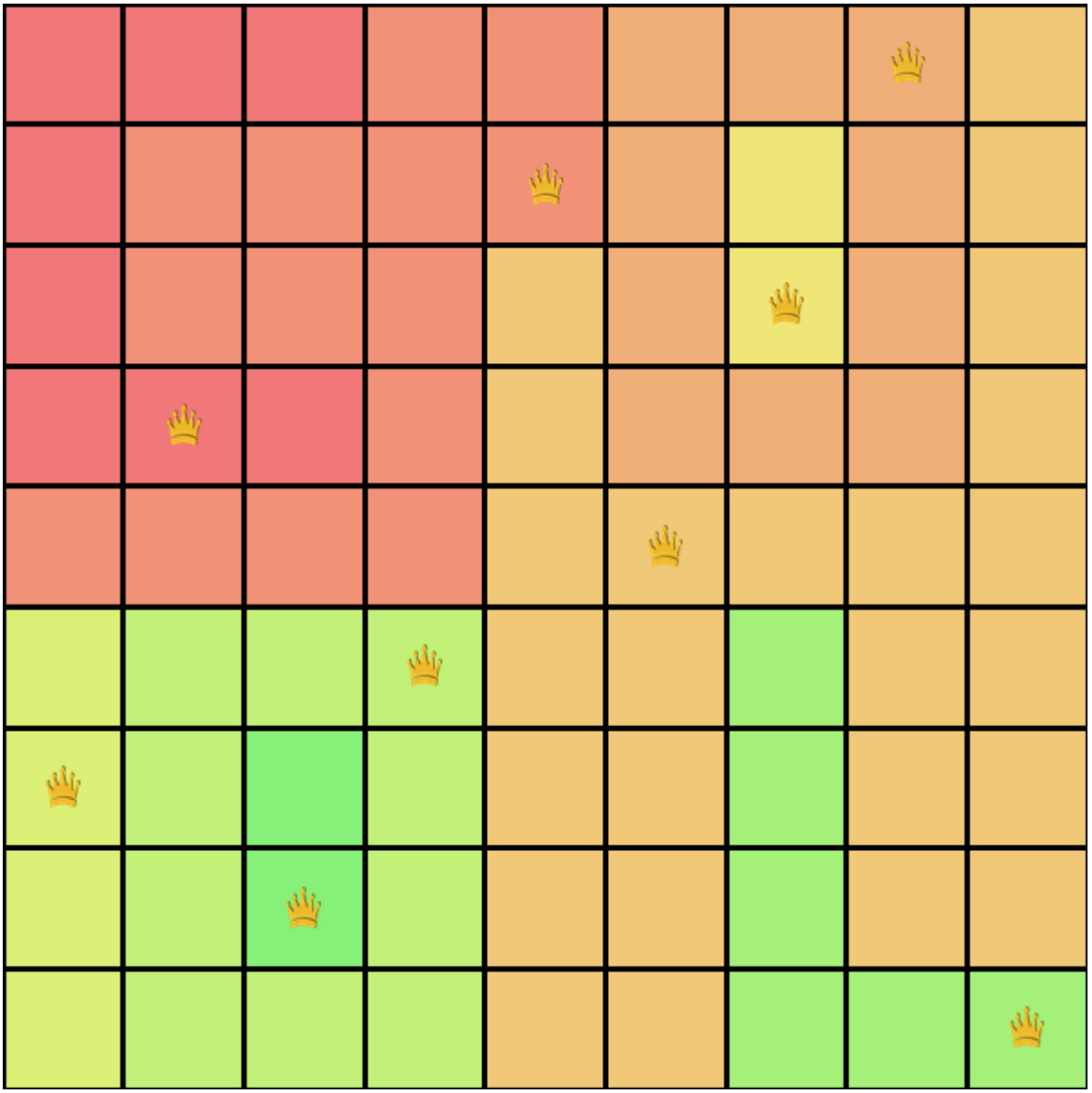
3.1.2 File Output

Output .txt

```
AAABBCC#D
ABBB#CECD
ABBBDC#CD
A#ABDCCCD
BBBBD#DDD
FGG#DDHDD
#GIGDDHDD
FG#GDDHDD
FGGGDDHH#

Waktu Pencarian: 2170 ms
Jumlah Cases: 323741637
```

Output Berupa gambar



3.2 Test 2 : Papan Berukuran 7 x 7

3.2.1 File Input

```
AAAAAFF
AAAAAFF
AAAAEEF
ABBAEEF
ABCCEEG
```

```
DBBBEEG
DDBBEEE
```

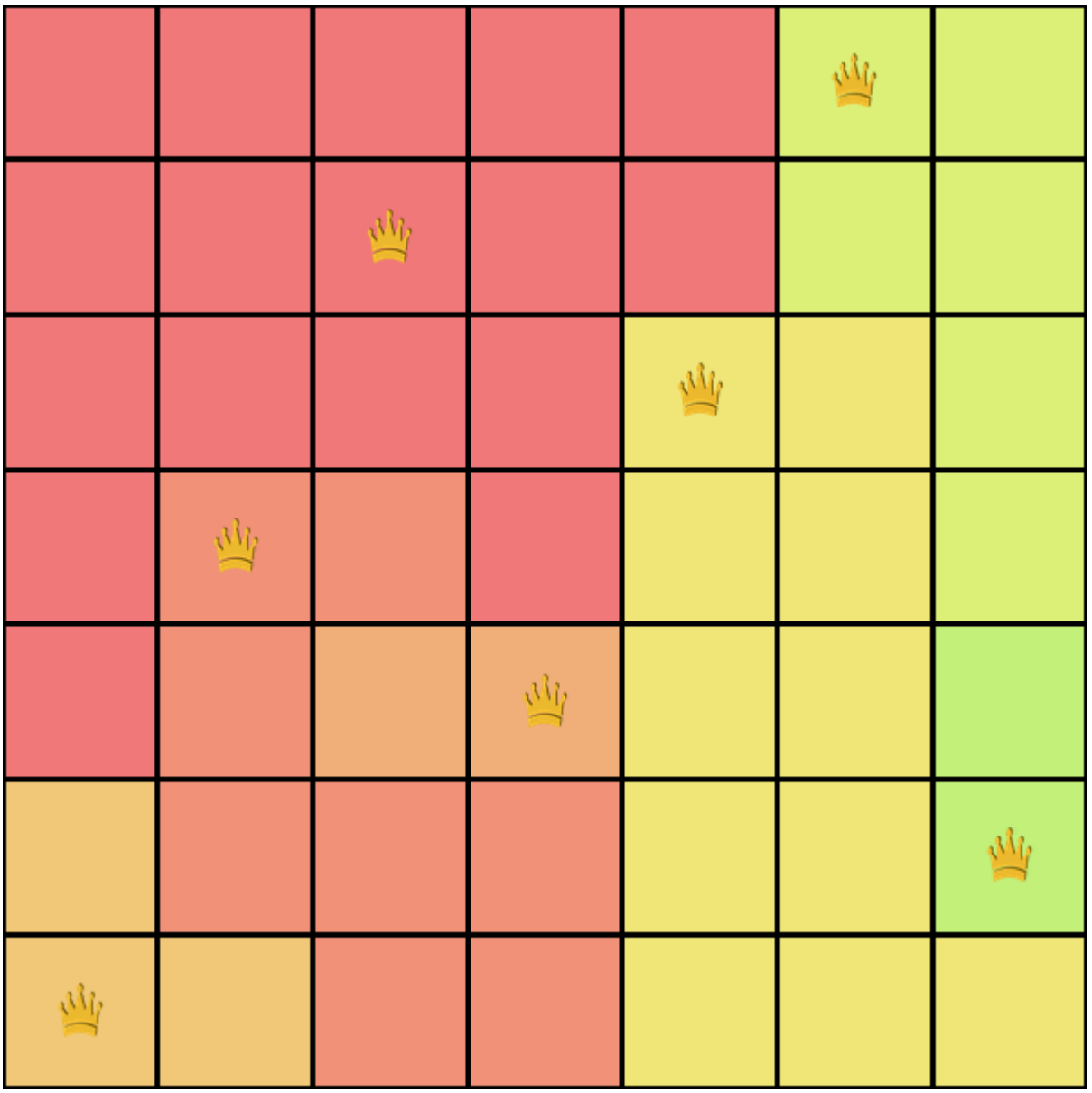
3.2.2 File Output

Output .txt

```
AAAAA#F
AA#AAFF
AAAA#EF
A#BAEEF
ABC#EEG
DBBBEE#
#DBBEEE

Waktu Pencarian: 33 ms
Jumlah Cases: 631996
```

Output Berupa gambar



3.3 Test 3 : Papan Berukuran 8 x 8

3.3.1 File Input

```
AAAABBBB
CAAABBDB
CAAABBDB
CAAABBBB
CCAEEBBB
CFFEEEGG
CFFEEEGG
```

```
HHFFEEEE
```

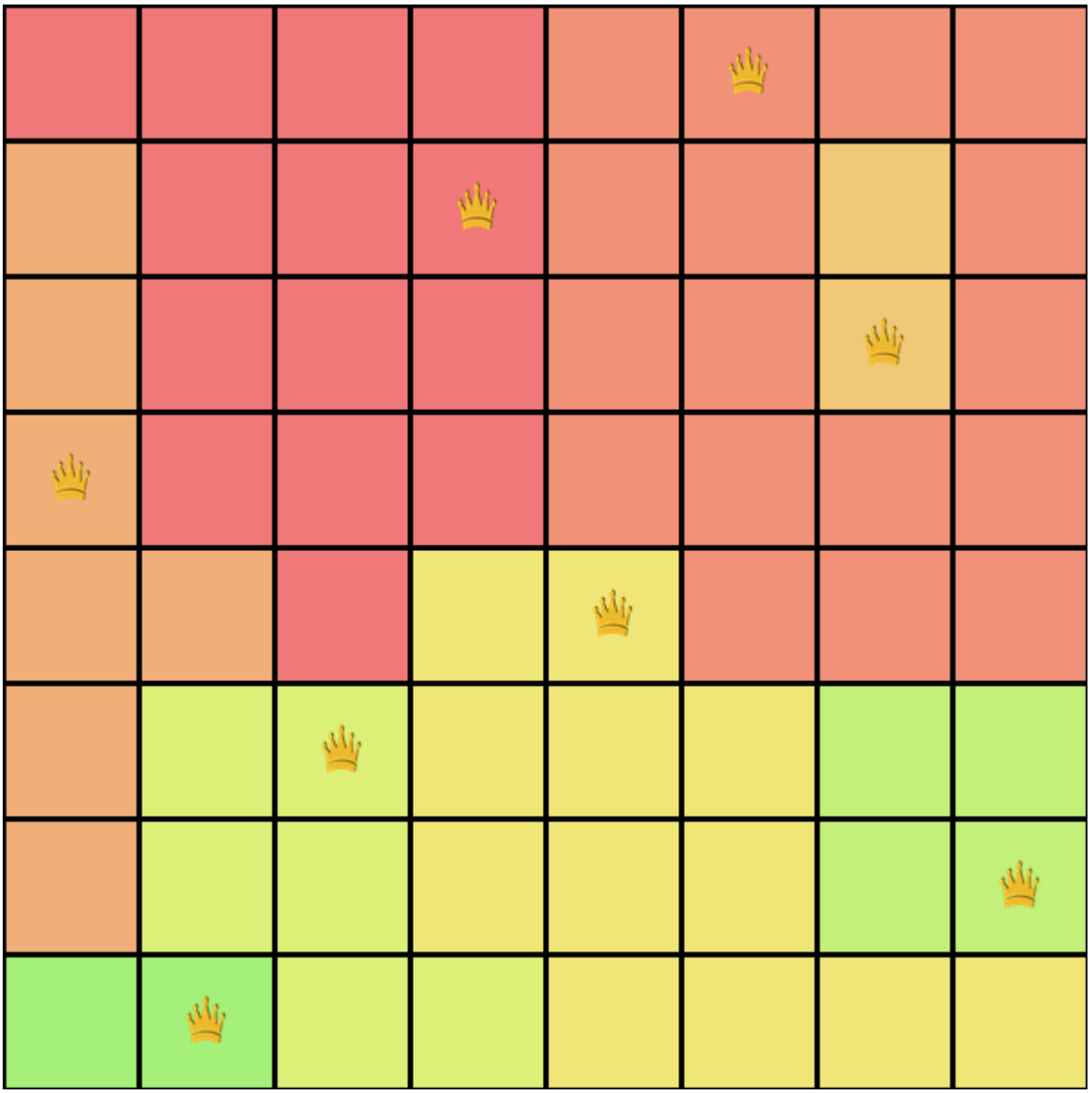
3.3.2 File Output

Output .txt

```
AAAAB#BB
CAA#BBDB
CAAABB#B
#AAABBBB
CCAE#BBB
CF#EEEGG
CFFEEEG#
H#FFEEEE

Waktu Pencarian: 94 ms
Jumlah Cases: 11471034
```

Output Berupa gambar



3.4 Test 4 : Papan Berukuran 26 x 26

3.4.1 File Input

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

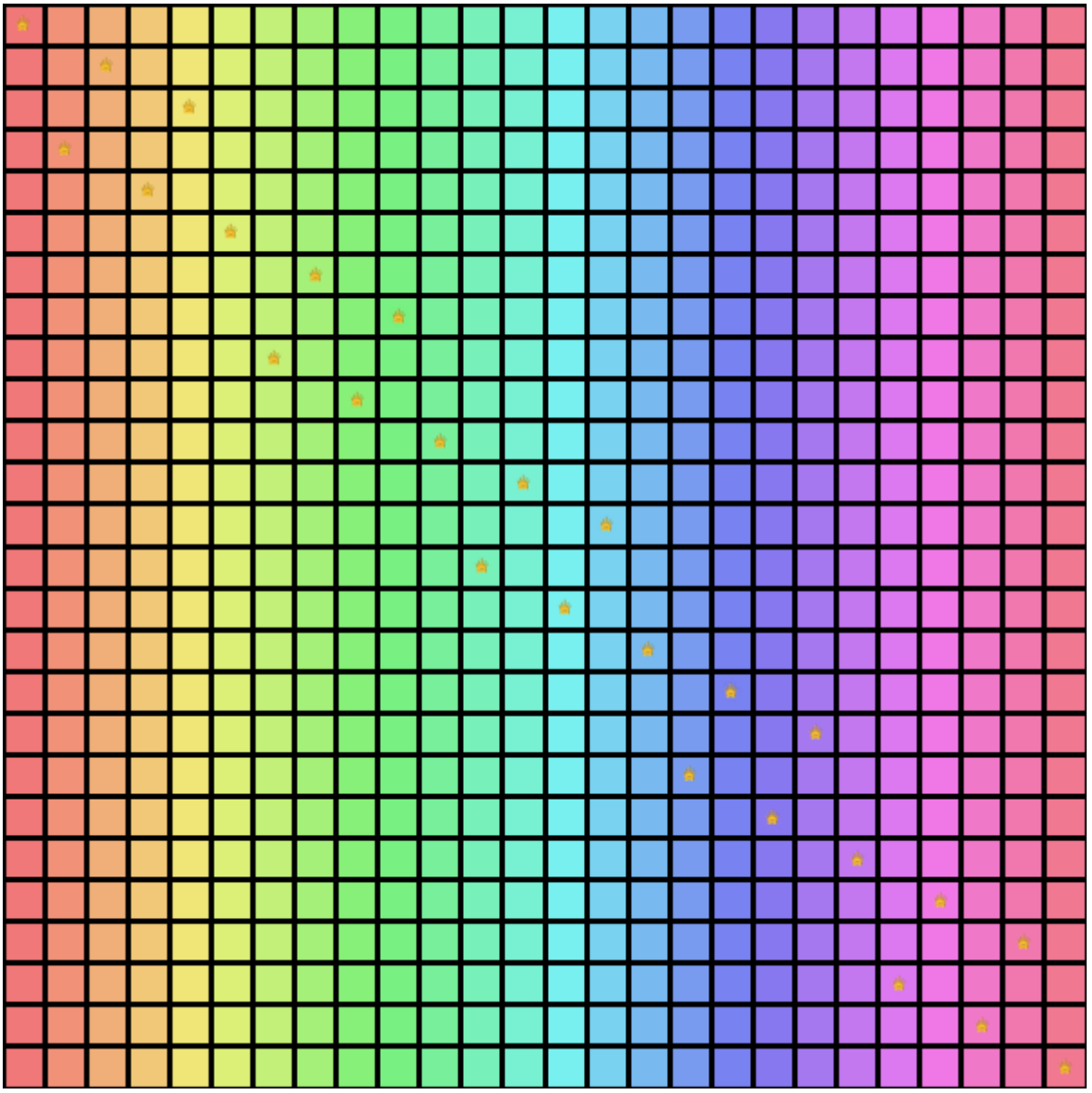
3.4.2 File Output

Output .txt

```
#BCDEFGHIJKLMNOPQRSTUVWXYZ
AB#DEFGHIJKLMNOPQRSTUVWXYZ
ABCD#FGHIJKLMNOPQRSTUVWXYZ
A#CDEFGHIJKLMNOPQRSTUVWXYZ
ABC#EFGHIJKLMNOPQRSTUVWXYZ
ABCDE#GHIJKLMNOPQRSTUVWXYZ
ABCDEFG#IJKLMNOPQRSTUVWXYZ
ABCDEFGHI#KLMNOPQRSTUVWXYZ
ABCDEF#HIJKLMNOPQRSTUVWXYZ
ABCDEFGH#JKLMNOPQRSTUVWXYZ
ABCDEFGHIJ#LMNOPQRSTUVWXYZ
ABCDEFGHIJKL#NOPQRSTUVWXYZ
ABCDEFGHIJKLMN#PQRSTUVWXYZ
ABCDEFGHIJK#MNOPQRSTUVWXYZ
ABCDEFGHIJKLM#OPQRSTUVWXYZ
ABCDEFGHIJKLMNO#QRSTUVWXYZ
ABCDEFGHIJKLMNOPQ#STUVWXYZ
ABCDEFGHIJKLMNOPQRS#UVWXYZ
ABCDEFGHIJKLMNOP#RSTUVWXYZ
ABCDEFGHIJKLMNOPQR#TUVWXYZ
ABCDEFGHIJKLMNOPQRST#VWXYZ
ABCDEFGHIJKLMNOPQRSTUV#XYZ
ABCDEFGHIJKLMNOPQRSTUVWX#Z
ABCDEFGHIJKLMNOPQRSTU#WXYZ
ABCDEFGHIJKLMNOPQRSTUVW#YZ
ABCDEFGHIJKLMNOPQRSTUVWXY#

Waktu Pencarian: 23 ms
Jumlah Cases: 351
```

Output Berupa gambar



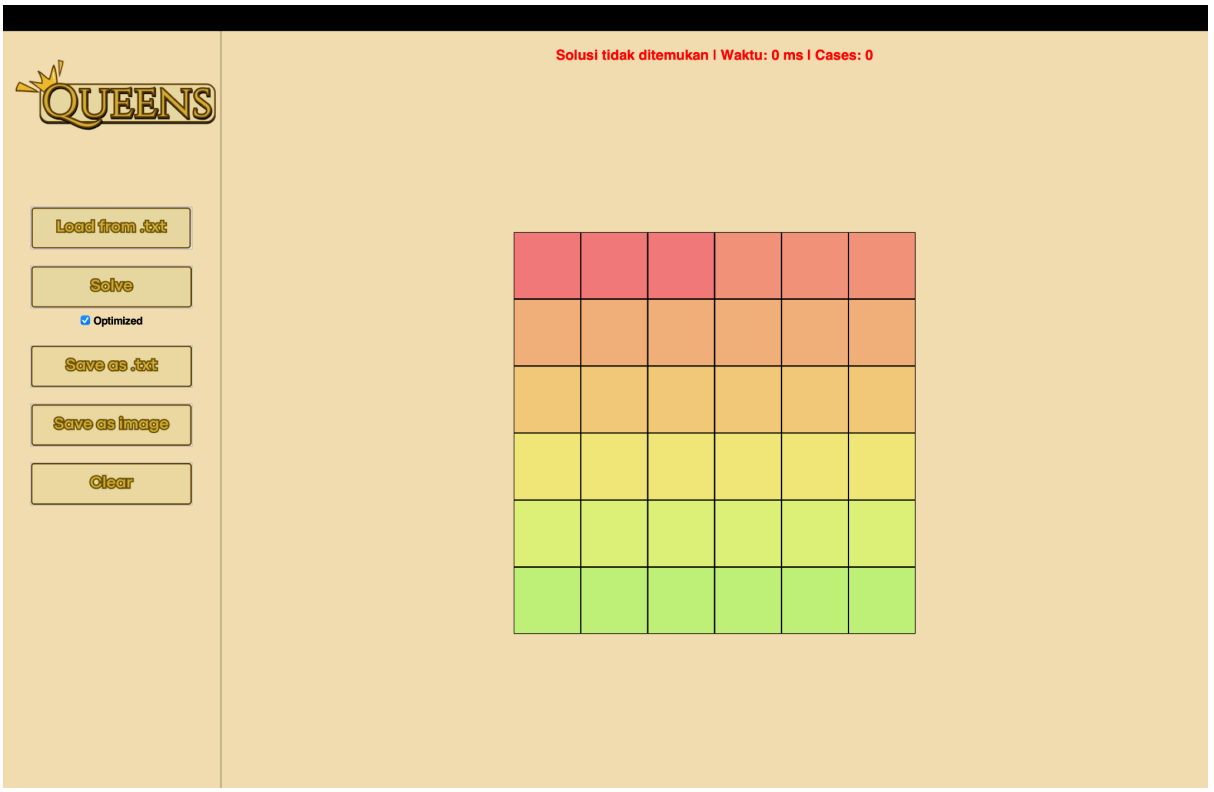
3.5 Test 5 : Papan Berukuran 6 x 6 tanpa Solusi

3.5.1 File Input

```
AAABBB
CCCCCC
DDDDDD
EEEEEE
FFFFFF
GGGGGG
```

3.5.2 File Output

Output Berupa gambar



3.6 Test 6 : Papan Berukuran 9 x 9 dengan Optimized Brute Force

3.5.1 File Input

```
AAABBCCCB
DABBBBCBB
DAEEEBCBB
FFFEBBBBB
GFGEGGHHH
GFGGGGGHG
GGGIIIGHG
GGGGIGGGG
GGGGIGGGG
```

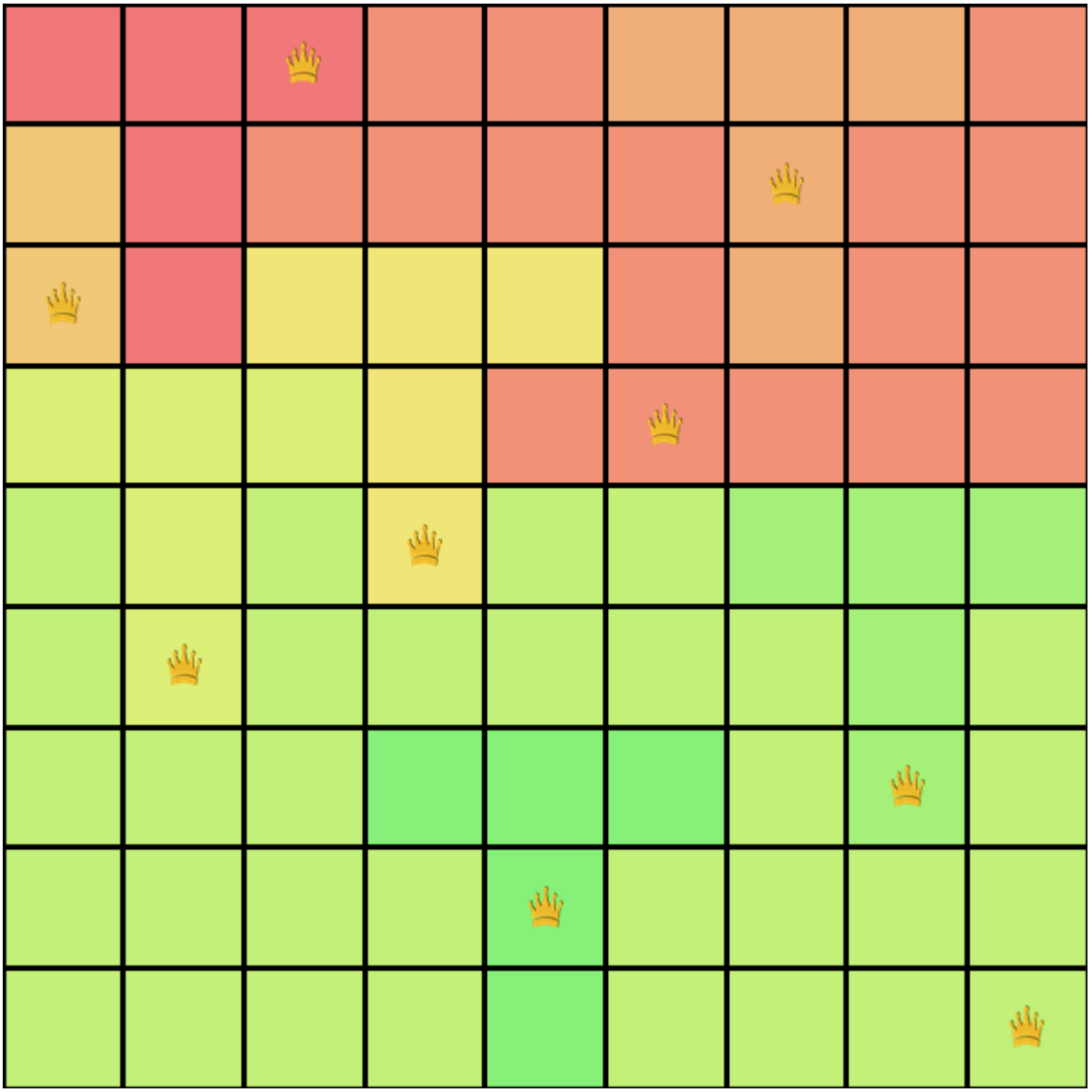
3.5.2 File Output

Output Berupa File .txt

```
AA#BBCCCB
DABBBB#BB
#AEEEBCBB
FFFEB#BBB
GFG#GGHHH
G#GGGGGHG
GGGIIIG#G
GGGG#GGGG
GGGGIGGG#

Waktu Pencarian: 2137 ms
Jumlah Cases: 24453
```

Output Berupa Gambar



4. Lampiran

Penerapan algoritma untuk solver dan GUI terdapat pada *repository* github penulis:
https://github.com/BryannPPH/Tucil1_13524067

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	✓	
7.	Program dapat menerima input sebuah gambar		✓

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

Bryan Pratama Putra Hendra