Laporan Tugas Kecil 1

IF2211 Strategi Algoritma

Queens Game Solver Using Brute Force Algorithm

Semester II Tahun 2025/2026

Samuelson Dharmawan Tanuraharja
13524001

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
2026

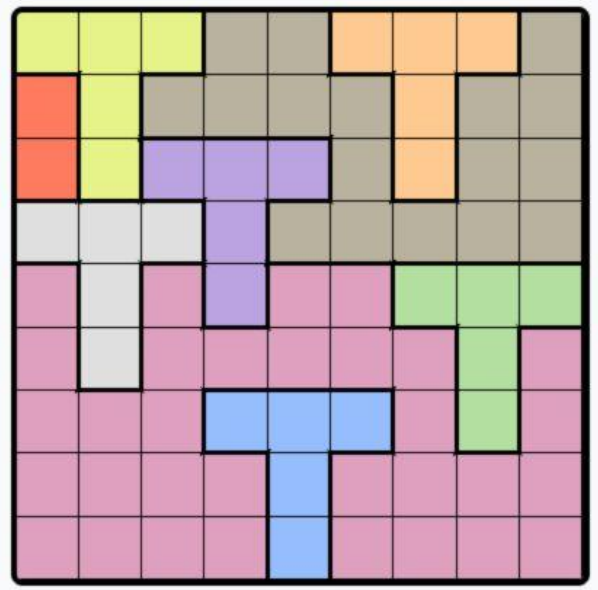
Daftar Isi

1 Deskripsi Penugasan

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

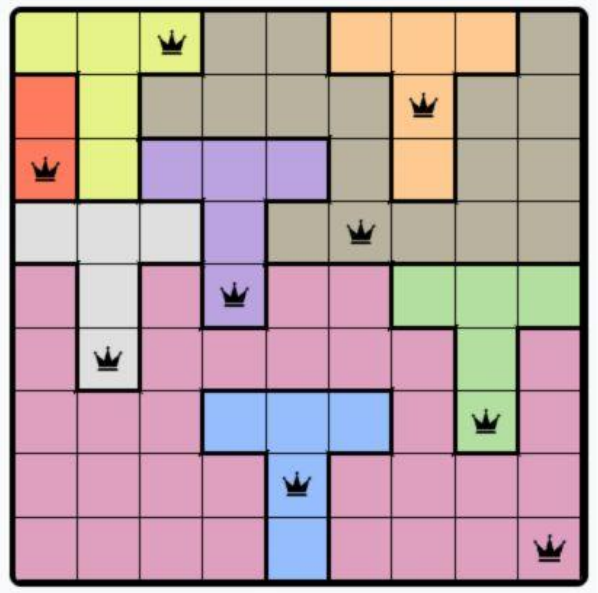
Tugas anda adalah membuat program yang dapat menemukan satu solusi penempatan *queen* pada suatu papan berwarna yang diberikan atau menampilkan bahwa tidak ada solusi yang valid. Program melakukan solusi menggunakan algoritma brute force.

Misal, diberikan papan sebagai berikut. Untuk tugas ini, papan selalu dimulai kosong.



Gambar 1: Papan Permainan Queens

Di bawah ini satu-satunya solusi valid. Perhatikan bahwa tiap baris, kolom, dan daerah warna sudah memiliki satu ditempati satu *queen*.



Gambar 2: Solusi Permainan Queen

2 Spesifikasi Tugas

2.1 Spesifikasi Wajib

- Buatlah sebuah program sederhana dalam bahasa Java, Python, C++, C, atau GO yang mengimplementasikan **algoritma *Brute Force*** untuk mencari solusi dalam permainan *Queens*.
- Algoritma *Brute Force* yang dimaksud harus bersifat murni, tidak menggunakan heuristik apapun.
- **Input:** program akan memberikan pengguna sebuah arahan pada terminal (jika tidak mengerjakan bonus) untuk memilih file test case berekstensi .txt, kemudian program membaca file test case tersebut yang berisi konfigurasi awal dari papan *Queens* yang akan diselesaikan. Perhatikan bahwa input dapat memiliki ukuran papan yang berbeda-beda, serta program harus dapat memvalidasi apakah input yang diberikan merupakan input valid atau bukan.

Berikut adalah contoh file .txt yang akan dijadikan sebagai input:

```
AAABBCCCD
ABBBBCECD
ABBBDCECD
AAABDCCCD
BBBBDDDDD
FGGGDDHDD
FGIGDDHDD
FGIGDDHDD
FGGGDDHHH
```

- **Output:**
  1. Menampilkan hasil akhir dari papan yang sudah terisi oleh *queens* dengan aturan yang benar.
  2. Menampilkan banyak konfigurasi atau iterasi yang ditinjau oleh algoritma.
  3. Menampilkan waktu eksekusi program dalam *millisecond* (cukup waktu pencarian dengan algoritma, tidak termasuk membaca dan menulis file).
  4. *Live Update:* program harus dapat memvisualisasikan proses *brute force* yang dilakukan.

Berikut adalah contoh output berdasarkan contoh input diatas:

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

Waktu pencarian: 120 ms
Banyak kasus yang ditinjau: 1000 kasus
Apakah Anda ingin menyimpan solusi? (Ya/Tidak)
```

2.2 Spesifikasi Bonus

Poin maksimal untuk bonus adalah 10.

- Membuat GUI (*Graphical User Interface* untuk program secara keseluruhan yang dapat memberi *input* dan memberi visualisasi (8 poin).
- *Input as image*, *input* dari program adalah sebuah gambar yang merupakan konfigurasi dari papan awal. (5 poin)
- *Output as image*, *output* dari hasil peletakan *queens* diberikan dalam bentuk gambar. (2 poin)

3 Dasar Teori

3.1 Brute Force Algorithm

Algoritma *Brute Force* adalah sebuah pendekatan penyelesaian masalah yang dilakukan secara lempang (*straightforward*). Algoritma ini didasarkan langsung pada pernyataan persoalan (*problem statement*) dan definisi konsep yang terlibat di dalamnya.

Karakteristik utama dari algoritma ini adalah memecahkan masalah dengan cara yang sederhana, langsung, dan jelas (*obvious way*). Seringkali pendekatan ini diasosiasikan dengan istilah "*Just do it!*" atau "*Just Solve it!*" karena langkah-langkah penyelesaiannya yang mudah dipahami tanpa memerlukan teknik optimasi yang rumit.

Dalam implementasinya, *Brute Force* bekerja dengan menelusuri atau mencacah seluruh kemungkinan solusi yang ada (misalnya melalui *exhaustive search*) untuk menemukan elemen yang memenuhi kriteria pencarian. Meskipun metode ini mungkin membutuhkan waktu komputasi yang besar untuk ukuran masukan yang besar, algoritma *Brute Force* memiliki kelebihan utama yaitu menjamin ditemukannya solusi (jika solusi tersebut memang ada) dan memastikan solusi yang ditemukan benar adanya.

3.2 Multithreading menggunakan QThread di Python

Dalam pengembangan aplikasi berbasis antarmuka pengguna grafis (*Graphical User Interface* atau GUI), responsivitas adalah aspek krusial. Secara *default*, aplikasi GUI berjalan pada satu *thread* utama (*main thread*) yang bertugas menangani penggambaran elemen visual dan merespons interaksi pengguna.

Jika sebuah proses komputasi yang berat (seperti algoritma *Brute Force* pada kasus ini) dijalankan langsung pada *main thread*, maka *user interface* aplikasi akan mengalami pembekuan (*freezing*) atau menjadi *not responding* hingga proses tersebut *crash*. Hal ini dikarenakan *main thread* sibuk melakukan perhitungan dan tidak sempat memproses *event* pembaruan tampilan antarmuka.

Untuk mengatasi hal tersebut, digunakan teknik *multithreading*. Di Python, khususnya dalam kerangka kerja PyQt5, kelas `QThread` menyediakan cara untuk menjalankan proses secara paralel di latar belakang. `QThread` memungkinkan pemisahan antara logika pemrosesan data (*worker thread*) dan logika tampilan (*GUI thread*). Komunikasi antara kedua *thread* ini dilakukan menggunakan mekanisme *Signals and Slots*, yang memungkinkan pengiriman data, seperti progres iterasi atau posisi *queen*, secara aman (*thread-safe*) untuk memperbarui tampilan secara *real-time* tanpa mengganggu responsivitas aplikasi.

3.3 Pengolahan Citra Digital menggunakan Pillow

Citra digital pada dasarnya adalah representasi numerik dari gambar dua dimensi. Sebuah citra dapat dipandang sebagai matriks di mana setiap elemennya disebut piksel (*picture element*). Setiap piksel memiliki nilai intensitas warna, yang pada format citra berwarna umumnya direpresentasikan dalam model warna RGB (*Red, Green, Blue*).

Dalam tugas ini, pemrosesan input berupa gambar dilakukan menggunakan pustaka **Pillow** (*Python Imaging Library Fork*). Pillow menyediakan kemampuan untuk memanipulasi citra, termasuk membaca file gambar, mengakses data piksel secara langsung, dan melakukan konversi format warna.

Mekanisme pembacaan papan permainan dari gambar melibatkan proses *sampling* warna pada koordinat piksel tertentu untuk memetakan area visual menjadi matriks logika program. Dengan membagi dimensi gambar berdasarkan ukuran papan $N \times N$, program dapat mengekstrak representasi warna dari setiap petak papan dan mengonversinya menjadi karakter unik yang merepresentasikan wilayah (*region*) warna tersebut.

4 Desain

4.1 Environment

Aplikasi ini dibangun menggunakan bahasa pemrograman Python. Pemilihan Python didasarkan pada kemudahan pengembangan, dukungan pustaka yang luas, serta portabilitas lintas platform. Berikut adalah rincian teknologi yang digunakan:

1. **Bahasa Pemrograman:** Python 3.14.
2. **Framework GUI:** PyQt5 (Python binding untuk Qt v5) digunakan untuk membangun antarmuka grafis yang responsif.

3. **Image Processing:** *Library* Pillow (PIL) digunakan untuk memproses *input* gambar dan mengekstrak warna piksel menjadi representasi *board matrix*.
4. **Concurrency:** Modul `QThread` dari PyQt5 digunakan untuk memisahkan proses algoritma *solver* dari *main thread* GUI, sehingga memungkinkan visualisasi *live update* tanpa menyebabkan aplikasi *crash*.

4.2 Struktur Program

Proyek ini disusun dengan struktur direktori sebagai berikut:

1. `src/`: Direktori utama yang berisi kode sumber program.
   - `gui.py`: Program utama (*entry point*) yang menangani logika antarmuka pengguna (*frontend*) dan interaksi pengguna.
   - `solver.py`: Implementasi kelas `QueensSolver` yang berisi logika algoritma *Brute Force* dan mekanisme *multithreading*.
   - `loader.py`: Modul untuk tahap pemrosesan file eksternal, baik format teks (`.txt`) maupun gambar, serta validasi input.
2. `test/`: Direktori berisi ragam *test case* untuk pengujian.
   - File berekstensi `.txt` untuk input berbasis karakter.
   - File gambar (`.png`, `.jpg`) untuk input berbasis visual.
3. `doc/`: Direktori yang berisi Laporan Akhir Tugas Kecil 1 IF2211 Strategi Algoritma 2026.
4. `requirements.txt`: Daftar dependensi pustaka Python yang diperlukan untuk menjalankan aplikasi.

Program dirancang dengan prinsip pemisahan tanggung jawab (*Separation of Concerns*):

- **GUI Layer (`gui.py`):** Bertanggung jawab menampilkan visualisasi papan dan menerima input. Komponen ini tidak melakukan perhitungan algoritma secara langsung.
- **Logic Layer (`solver.py`):** Bertanggung jawab mencari solusi menggunakan algoritma rekursif. Layer ini berkomunikasi dengan GUI melalui mekanisme *Signals and Slots*.
- **Data Layer (`loader.py`):** Bertanggung jawab memparsing data mentah dari file menjadi struktur data matriks yang siap diproses.

4.3 Cara Instalasi dan Menjalankan Program

Berikut adalah langkah-langkah untuk mengaktifkan *virtual environment* dan menjalankan program:

1. Persiapan *Virtual Environment*

Menyiapkan *virtual environment* menjadi tahap krusial demi menjaga program tidak terganggu karena perbedaan depedensi yang sudah terinstalasi di lingkungan Python global.

```
# Membuat virtual environment
python -m venv venv

# Mengaktifkan venv (Windows)
venv\Scripts\activate

# Mengaktifkan venv (Linux/macOS)
source venv/bin/activate
```

2. Instalasi Dependensi

Unduh dan pasang *library* yang dibutuhkan (PyQt5 dan Pillow) sekaligus menggunakan kode berikut ini.

```
pip install -r requirements.txt
```

3. Menjalankan Program

Jalankan file utama `gui.py` yang berada di dalam direktori `src`.

```
python src/gui.py
```

5 Implementasi Spesifikasi Wajib

5.1 Struktur Data

Untuk mendukung kinerja algoritma pencarian solusi, program menggunakan beberapa struktur data bawaan Python yang efisien, antara lain:

- `self.grid`: Representasi papan permainan menggunakan *List of Lists* (matriks 2D). Setiap elemen menyimpan karakter yang merepresentasikan warna wilayah.

```
self.grid = matriks_warna  # Tipe: List[List[str]]
```

- `self.occ_cols` dan `self.occ_colors`: Struktur data Himpunan (*Set*) digunakan untuk menyimpan daftar kolom dan warna yang sedang ditempati oleh *queen*. Penggunaan *Set* memungkinkan operasi pengecekan (*lookup*) dilakukan dalam waktu konstan $O(1)$.

```
self.occ_cols = set()
self.occ_colors = set()
```

- `self.queens_positions`: *List* yang menyimpan *tuple* koordinat $(baris, kolom)$ dari ratu yang sudah ditempatkan untuk keperluan validasi ketetanggaan (*adjacency*).

5.2 Algoritma Utama

Algoritma yang diimplementasikan adalah **Algoritma Brute Force** dengan pendekatan **Exhaustive Search**. Algoritma ini bekerja dengan cara mencoba menempatkan satu ratu pada setiap baris secara urut, mulai dari baris 0 hingga $N - 1$, dan memeriksa validitasnya terhadap aturan permainan (*constraints*). Pendekatan ini serupa dengan solusi Brute Force untuk kasus Sudoku yang dipaparkan pada PPT Algoritma Brute Force Bag. 1 IF2211 Strategi Algoritma 2026.

Pendekatan ini menjamin ditemukannya solusi (jika memang ada solusi) karena algoritma menelusuri ruang solusi secara menyeluruh tanpa menggunakan elemen heuristik atau tebakan acak. Langkah-langkah detailnya adalah sebagai berikut:

1. **Inisialisasi:** Program memulai pencarian dari baris pertama (`row = 0`).
2. **Iterasi Kolom:** Pada baris yang sedang aktif, algoritma mencoba menempatkan ratu pada setiap kolom secara berurutan (dari indeks 0 hingga $N - 1$).
3. **Validasi Kendala:** Setiap kali mencoba menempatkan ratu di posisi (row, col), program memanggil fungsi `is_safe()` untuk memeriksa:
   - Apakah kolom tersebut sudah ditempati?
   - Apakah warna di tile tersebut sudah memiliki ratu?
   - Apakah tile tersebut bersentuhan dengan ratu lain di sekitarnya?
4. **Penempatan dan Rekursi:**
   - Jika posisi valid, ratu ditempatkan, dan status data (`occ_cols`, `occ_colors`) diperbarui. Kemudian, algoritma melanjutkan pencarian ke baris berikutnya $(row + 1)$.
   - Jika posisi tidak valid, algoritma lanjut mencoba kolom berikutnya di baris yang sama.
5. **Mundur:** Jika pada suatu baris tidak ditemukan kolom yang valid (buntu), algoritma membatalkan langkah pada baris sebelumnya (menghapus ratu terakhir) dan mencoba kemungkinan kolom lain.
6. **Terminasi:** Algoritma berhenti ketika berhasil menempatkan ratu di semua baris $(row = N)$ yang berarti solusi ditemukan.

5.3 Implementasi dalam Python

Berikut adalah implementasi kode inti dari kelas `QueensSolver` di file `src/solver.py`.

```python
def solve(self, row):
    # Basis: Jika semua ratu berhasil ditempatkan (row == N), solusi ditemukan
    if row == self.N:
        return True

    # Coba tempatkan Queen di setiap kolom pada baris ini
    for col in range(self.N):
        self.iterations += 1

        # Cek validitas posisi berdasarkan aturan (Constraint Check)
        if self.is_safe(row, col):
            # 1. Tempatkan Ratu (Update State)
            self.solution.append((row, col))
            self.occ_cols.add(col)
            self.occ_colors.add(self.grid[row][col])
            self.queens_positions.append((row, col))

            # Visualisasi Live Update
            self.update_sig.emit(row, col, True)
            self.msleep(50)

            # 2. Lanjut ke baris berikutnya
            if self.solve(row + 1):
                return True # Solusi ditemukan di cabang ini

            # 3. Jika jalan buntu, batalkan langkah (Undo/Reset State)
            self.solution.pop()
            self.occ_cols.remove(col)
            self.occ_colors.remove(self.grid[row][col])
            self.queens_positions.remove((row, col))

            self.update_sig.emit(row, col, False)
            self.msleep(20)

    # Jika semua kolom dicoba dan gagal, kembalikan False (Mundur)
    return False
```

Deskripsi Alur Algoritma Pencarian:
Algoritma ini mengimplementasikan prinsip *exhaustive search* secara sistematis yang bekerja baris demi baris. Berikut adalah deskripsi langkah-langkahnya:

1. **Basis Rekursi:** Periksa apakah `row` saat ini sama dengan N. Jika ya, berarti seluruh N ratu telah berhasil ditempatkan tanpa melanggar aturan main. Algoritma mengembalikan `True` (solusi ditemukan).
2. **Enumerasi Kemungkinan:** Jika belum selesai, lakukan iterasi untuk setiap kemungkinan kolom `col` dari indeks 0 sampai $N - 1$ pada baris `row` saat ini.
3. **Rekursi Peletakkan Queen:** Untuk setiap kolom, panggil fungsi `is_safe(row, col)`.
   - Jika posisi **tidak aman**, abaikan dan lanjut ke kolom berikutnya (langsung memangkas cabang pencarian yang tidak valid).
   - Jika posisi **aman**, lakukan langkah berikut:
     a. **Maju:** Tempatkan ratu pada posisi tersebut. Tambahkan data ke `solution`, `occ_cols`, dan `occ_colors`.
     b. **Rekursif:** Panggil fungsi `solve` untuk baris berikutnya (`row + 1`).
     c. **Evaluasi Hasil Rekursif:**
        – Jika pemanggilan rekursif mengembalikan `True`, maka kembalikan `True` ke pemanggil sebelumnya (rekursikan keberhasilan).

– Jika pemanggilan rekursif mengembalikan `False` (jalan buntu), maka lakukan pembatalan langkah dengan melakukan penghapusan ratu dari posisi tersebut (kosongkan data dari `solution`, `occ_cols`, dll) agar dapat mencoba kemungkinan kolom lain pada iterasi berikutnya.

4. **Terminasi Kegagalan:** Jika seluruh kolom $0..N-1$ pada baris ini telah dicoba dan tidak ada yang menghasilkan solusi valid, kembalikan `False` (memberi sinyal *False* ke langkah sebelumnya untuk menginfokan bahwa konfigurasi yang diunggah tidak memiliki solusi).

5.4 Fungsi Pembantu

Untuk memastikan setiap langkah validasi berjalan sesuai aturan matematis yang ketat (non-heuristik), digunakan fungsi `is_safe`.

```
def is_safe(self, row, col):
    current_color = self.grid[row][col]

    # Cek 1: Warna sudah terpakai?
    if current_color in self.occ_colors:
        return False

    # Cek 2: Kolom sudah terpakai?
    if col in self.occ_cols:
        return False

    # Cek 3: Tetangga (Atas-Kiri, Atas, Atas-Kanan)
    check_directions = [(-1, -1), (-1, 0), (-1, 1)]
    for dr, dc in check_directions:
        nr, nc = row + dr, col + dc
        if 0 <= nr < self.N and 0 <= nc < self.N:
            if (nr, nc) in self.queens_positions:
                return False

    return True
```

Deskripsi Logika Validasi:
Fungsi `is_safe` bertugas memastikan bahwa penempatan ratu pada posisi (row, col) tidak melanggar aturan permainan. Pemeriksaan dilakukan secara *straightforward* dengan urutan sebagai berikut:

1. **Pemeriksaan *Colour Region*:** Algoritma memeriksa apakah warna pada petak `grid[row][col]` sudah terdaftar di dalam set `occ_colors`. Jika sudah ada, kembalikan `False`.

2. **Pemeriksaan Kolom:** Algoritma memeriksa apakah indeks kolom `col` sudah terdaftar di dalam himpunan `occ_cols`. Jika sudah ada, kembalikan `False`.

3. **Pemeriksaan Tetangga (Adjacency):** Algoritma memeriksa tiga posisi tetangga yang berada di baris atasnya, yaitu:

   - Diagonal kiri-atas $(row-1, col-1)$
   - Tepat di atas $(row-1, col)$
   - Diagonal kanan-atas $(row-1, col+1)$

   Jika salah satu dari posisi tersebut berisi ratu (terdaftar di `queens_positions`), maka posisi saat ini tidak memenuhi aturan (kembalikan `False`).

4. Jika lolos dari ketiga pemeriksaan di atas, posisi bisa ditempati oleh Queen karena tidak melanggar aturan (kembalikan `True`).

6 Implementasi Spesifikasi Bonus

6.1 Graphical User Interface (GUI)

Antarmuka pengguna dibangun menggunakan pustaka **PyQt5**. Struktur utama aplikasi didefinisikan dalam kelas `MainWindow` yang mewarisi `QMainWindow`. Komponen-komponen penyusun antarmuka diorganisir menggunakan *layout manager* sebagai berikut:

- `MainWindow`: Jendela utama aplikasi.
- `QWidget (Central)`: Kontainer utama yang menampung seluruh elemen visual.
- `QVBoxLayout (Root)`: Tata letak vertikal untuk menyusun *header*, papan permainan, dan panel statistik.
- `BoardWidget`: *Custom Widget* yang berfungsi sebagai kanvas untuk menggambar visualisasi papan dan ratu secara dinamis.
- `SliderDialog`: Dialog kustom (`QDialog`) yang muncul saat pengguna mengunggah gambar, berfungsi untuk meminta input ukuran papan (N) menggunakan *slider* ketika input terdeteksi dalam bentuk sebuah *image*.

6.1.1 Multithreading dan Responsivitas

Untuk memastikan GUI tidak membeku (*freeze*) saat algoritma *Brute Force* melakukan komputasi berat, aplikasi menerapkan *multithreading* menggunakan `QThread`.

1. **Worker Thread:** Kelas `QueensSolver` (turunan dari `QThread`) menjalankan logika pencarian solusi di latar belakang.
2. **Signal and Slot:** Komunikasi antara *worker thread* dan *GUI thread* dilakukan melalui mekanisme `pyqtSignal`.
   - `update_sig`: Dikirim setiap kali ratu ditempatkan atau dihapus untuk memicu penggambaran ulang (*repaint*) pada papan.
   - `finished_sig`: Dikirim saat pencarian selesai untuk menampilkan *popup* hasil.

6.1.2 Visualisasi Papan (Rendering)

Fungsi `paintEvent` di dalam kelas `BoardWidget` akan dipanggil secara otomatis oleh sistem Qt setiap kali ada permintaan pembaruan tampilan untuk memvisualisasikan papan warna.

```
def paintEvent(self, event):
    if self.N == 0: return

    painter = QPainter(self)
    painter.setRenderHint(QPainter.Antialiasing)

    # Hitung ukuran sel dinamis berdasarkan ukuran satu tile papan
    w = self.width()
    h = self.height()
    cell_size = min(w, h) // self.N

    # Menggambar Grid Warna
    for r in range(self.N):
        for c in range(self.N):
            char = self.grid[r][c]
            bg_color = self.colors.get(char, Qt.white)
            painter.setBrush(QBrush(bg_color))
            painter.setPen(QPen(Qt.black, 1))

            # Gambar kotak
            painter.drawRect(x, y, cell_size, cell_size)

    # Menggambar Queen Emoji
    font_size = int(cell_size * 0.6)
```

```
25      emoji_font = QFont("Segoe UI Emoji", font_size)
26      painter.setFont(emoji_font)
27
28      for (r, c) in self.queens:
29          # Align Center Queen Emoji (maaf tidak bisa di display character emoji
                queen-nya karena tidak kompatibel)
30          painter.drawText(x, y, cell_size, cell_size, Qt.AlignCenter, "<queen-
                emoji>")
```

Kode di atas akan selalu menggambar ulang seluruh papan secara *real-time* menggunakan primitif grafis (`drawRect`, `drawText`), sehingga visualisasi sangat ringan dan responsif terhadap perubahan ukuran tile papan warna.

6.2 Input as Image

Fitur bonus ini memungkinkan pengguna untuk mengunggah gambar papan permainan sebagai input. Implementasi dilakukan menggunakan pustaka **Pillow** pada modul `loader.py`.

Proses pengolahan citra dilakukan dengan langkah-langkah berikut:

1. **Deteksi Format:** Fungsi `read_file` mendeteksi ekstensi gambar (.png, .jpg) dan mengembalikan objek gambar.
2. **Input Ukuran (N):** Karena gambar tidak memiliki informasi jumlah baris dan kolom dari papan yang diunggah, aplikasi memunculkan sebuah komponen `SliderDialog` agar pengguna dapat menentukan dimensi papan ($N \times N$) secara interaktif.
3. **Sampling Warna:** Gambar dibagi menjadi $N \times N$ petak. Program mengambil sampel piksel di titik tengah setiap petak imajiner tersebut.
4. **Mapping Karakter:** Warna RGB yang unik dipetakan menjadi karakter alfabet ('A', 'B', dst.) untuk membentuk papan dalam representasi matriks karakter.

```
1  def image_to_grid(image, N):
2      width, height = image.size
3      cell_w = width / N
4      cell_h = height / N
5      matriks_warna = []
6
7      for r in range(N):
8          row = []
9          for c in range(N):
10             # Ambil sampel warna di tengah sel
11             center_x = int((c * cell_w) + (cell_w / 2))
12             center_y = int((r * cell_h) + (cell_h / 2))
13             pixel = image.getpixel((center_x, center_y))
14             row.append(pixel)
15         matriks_warna.append(row)
16
17     # Konversi RGB ke Karakter ('A', 'B', 'C', dst)
18     return rgb_to_chars(matriks_warna)
```

Algoritma ini memungkinkan program untuk membaca *screenshot* dari permainan Queens asli dan merepresentasikannya dalam struktur data internal yang dapat diselesaikan oleh algoritma *solver*.

7 Pengujian

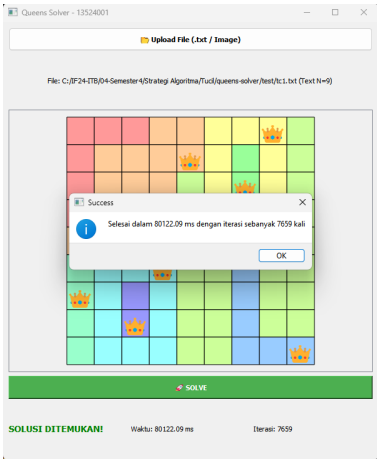
Untuk memastikan program berjalan dengan benar, dilakukan beberapa pengujian dengan beberapa kasus uji (*test case*). Kasus uji yang digunakan adalah sebagai berikut.

1. Kasus Uji 9×9 (file .txt)
2. Kasus Uji 8×8 (file *Image*)
3. Kasus Uji 3×3 (case no solution)
4. Kasus Uji 26×26 (case maksimum)
5. Kasus Uji Tidak Valid (file korup / binary salah)

7.1 Kasus Uji 1

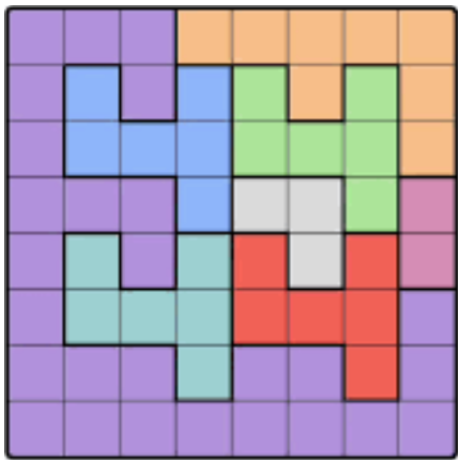
```
AAABBCCCD
ABBBBCECD
ABBBDCECD
AAABDCCCD
BBBBDDDDD
FGGGDDHDD
FGIGDDHDD
FGIGDDHDD
FGGGDDHHH
```

- Deskripsi: Papan 9×9
- Output:



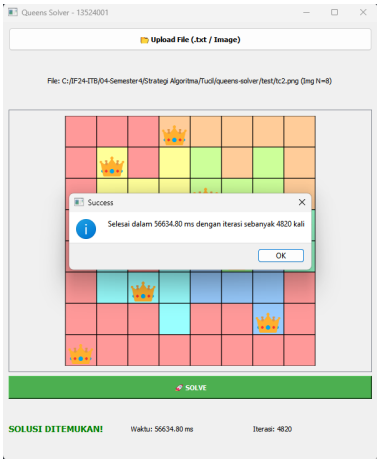
Gambar 3: Hasil Kasus Uji 1

7.2 Kasus Uji 2



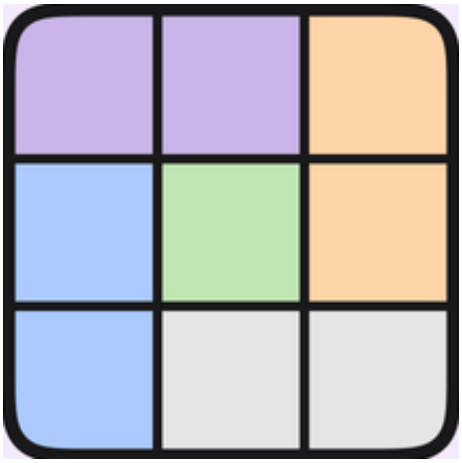
Gambar 4: Kasus Uji 2 (Input Gambar)

- Deskripsi: Papan 8×8
- Output:



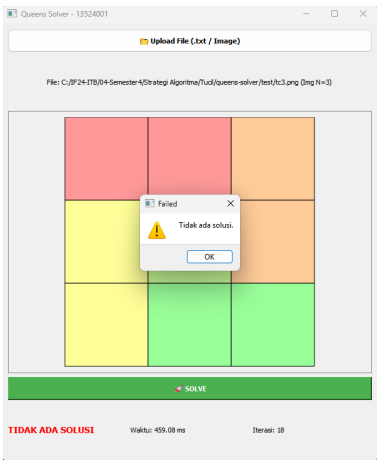
Gambar 5: Hasil Kasus Uji 2

7.3 Kasus Uji 3



Gambar 6: Kasus Uji 3

- Deskripsi: Papan 3×3 (kasus tidak ada solusi)
- Output:

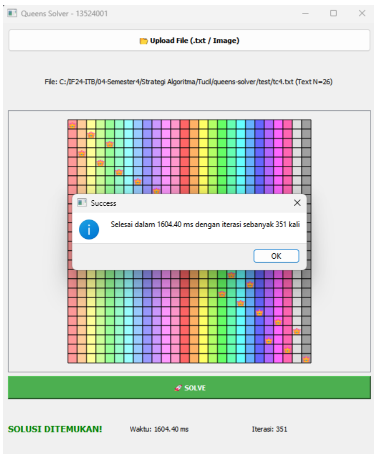


Gambar 7: Hasil Kasus Uji 3

7.4 Kasus Uji 4

```
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
```

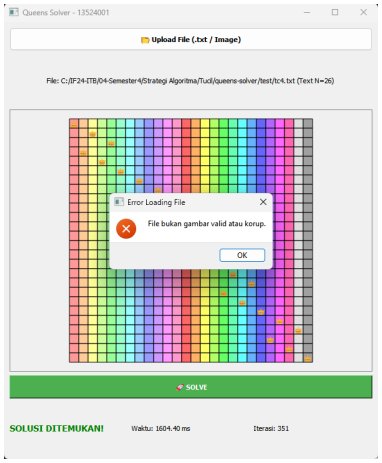
- Deskripsi: Papan 26×26 (case maksimum)
- Output:



Gambar 8: Hasil Kasus Uji 4

7.5 Kasus Uji 5

- Deskripsi: File korup / binary salah
- Output:



Gambar 9: Hasil Kasus Uji 5

7.6 Rangkuman Hasil Pengujian

Berikut adalah ringkasan hasil pengujian kasus uji yang sudah dilakukan beserta waktu penyelesaian dan banyak kasus yang ditinjau.

No	Deskripsi Kasus	Ukuran	Waktu	Banyak Kasus
1	Kasus Normal File .txt	9×9	$80,122.09$ ms	7659 kasus
2	Kasus Normal File Gambar	8×8	$56,634.80$ ms	4820 kasus
3	Kasus Gagal	3×3	459.08 ms	18 kasus
4	Kasus Maksimal	26×26	1604.40	351 kasus
5	File Korup	N/A	N/A	Input tidak valid

Tabel 1: Ringkasan Hasil Pengujian Kasus Uji

8 Penutup

8.1 Lampiran

Berikut adalah tautan yang merujuk pada repositori program Queens Solver saya disimpan. Link Github: https://github.com/sameuslondhrm/Tucil1_13524001.git
Rangkuman hasil program ini tertera pada tabel berikut.

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 membaca masukan dalam bentuk file gambar	✓	

Tabel 2: Ringkasan Keterselesaian Program

8.2 Pernyataan

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

Bandung, 18 Februari 2026

Samuelson Dharmawan Tanuraharja
13524001