Laporan Tugas Kecil 1 IF-2211 Strategi Algoritma
Semester II Tahun Akademik 2025/2026

Penyelesaian Permainan Queens Linkedin

Disusun Oleh:

Muhammad Daffa Arrizki

Yanma - 13524133 K-03

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

2025

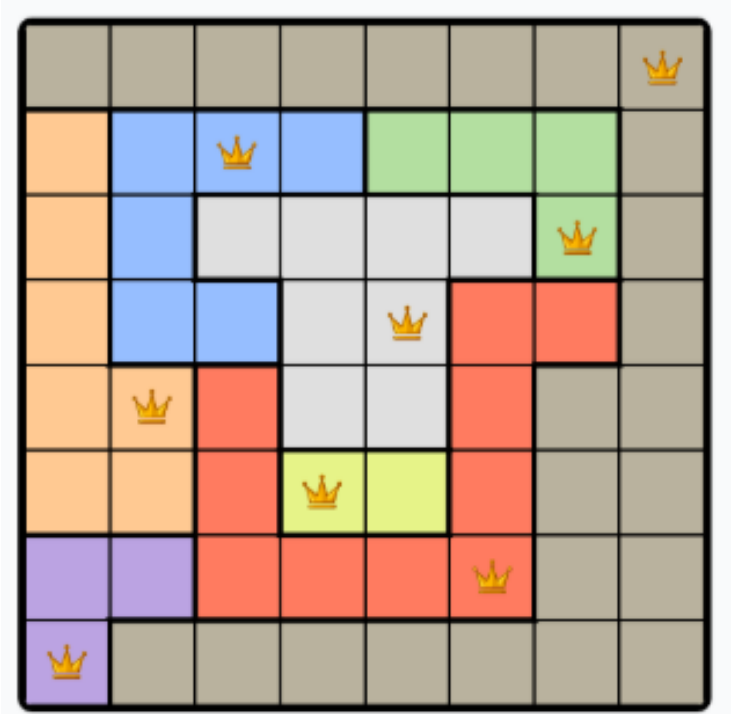
BAB I

Deskripsi Masalah

1.1 Deskripsi Tugas

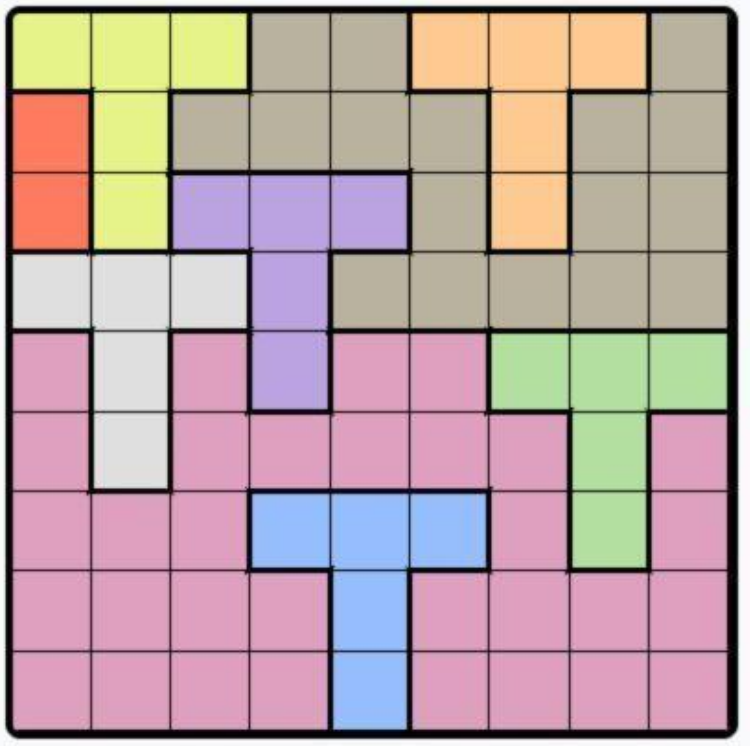
Queens adalah gim logika yang tersedia pada situs jejaring profesional LinkedIn. Tujuan dari gim ini adalah menempatkan *queen* pada sebuah papan persegi berwarna sehingga terdapat hanya satu *queen* pada tiap baris, kolom, dan daerah warna. Selain itu, satu *queen* tidak dapat ditempatkan bersebelahan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

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 pencarian solusi menggunakan algoritma ***brute force***.

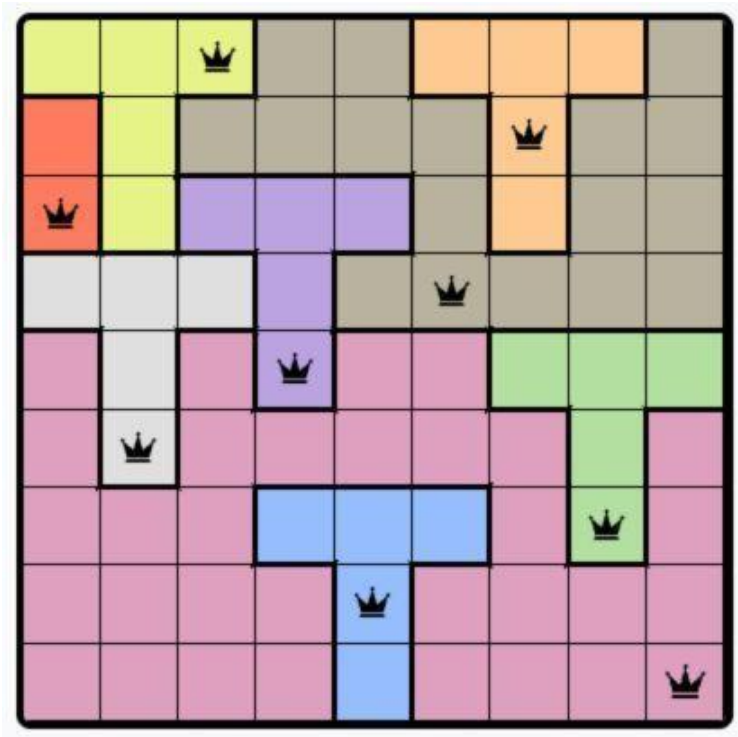


1.2 Ilustrasi Kasus

Diberikan papan sebagai berikut. **Untuk tugas ini, papan selalu dimulai kosong**.



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



BAB II

Algoritma Bruteforce dalam Penyelesaian Queens Linkedin

2.1. Board

Untuk merepresentasikan persoalan *N-Queens* yang memiliki batasan wilayah berwarna (*colored regions*), *board* dirancang menggunakan struktur matriks dan himpunan. *Board* ini memisahkan antara data statis (peta wilayah) dan data dinamis (posisi ratu) untuk memudahkan proses validasi.

2.1.1. Representa *Board*

Papan permainan dimodelkan sebagai grid dua dimensi berukuran *N x N*. Setiap sel pada grid memiliki dua atribut utama:

1. **Warna:** Melabeli setiap sel dengan karakter tertentu yang merepresentasikan wilayah warnanya. Data ini bersifat *immutable* (tidak berubah) selama pencarian solusi berlangsung.
2. **Ratu:** Menandakan status keberadaan ratu pada sel tersebut (apakah terisi atau kosong). Ini adalah data *mutable* yang akan terus berubah seiring proses pencarian solusi.

2.1.2. Pengecekan Validasi Ratu

Sebuah konfigurasi papan dinyatakan sebagai solusi yang sah (*valid*) jika dan hanya jika memenuhi seluruh kendala berikut:

1. Jumlah total ratu yang ditempatkan pada papan harus tepat berjumlah *N*.
2. Tidak diperbolehkan adanya dua ratu atau lebih yang menempati sel dengan label wilayah yang sama.
3. Tidak diperbolehkan adanya dua ratu yang saling bersinggungan, baik secara horizontal, vertikal, maupun diagonal.
4. Tidak diperbolehkan adanya dua ratu yang berada pada kolom yang sama.

2.2. Algoritma Solver

Strategi utama yang digunakan adalah brute force dengan runut balik (backtracking) iteratif dengan gagasan pemodelan berikut:

1. Representasi Solusi: satu baris satu ratu

   Solusi tidak dibangkitkan sebagai kombinasi bebas seluruh sel papan, tetapi sebagai pemetaan baris-ke-kolom. Artinya, untuk setiap baris hanya dipilih satu kolom kandidat ratu. Pendekatan ini exhaustive terhadap ruang solusi yang dimodelkan, namun lebih cepat dibandingkan dengan kombinasi bebas seluruh sel papan.
2. Ruang Pencarian Bertahap

   Pencarian dilakukan dari baris pertama hingga baris terakhir:
   - Pada baris aktif, algoritma mencoba kolom secara berurutan (kiri ke kanan).
   - Setiap penempatan dicatat sebagai state saat ini.
   - Jika masih ada baris berikutnya, algoritma maju ke baris selanjutnya.
3. Pengujian Validitas Konfigurasi

   Pengujian kendala dilakukan ketika konfigurasi sudah lengkap (*N* ratu telah ditempatkan), meliputi:
   - Jumlah ratu tepat N,
   - Tidak ada ratu pada baris yang sama,
   - Tidak ada ratu pada kolom yang sama,
   - Tidak ada ratu pada region/warna yang sama,
   - Tidak ada dua ratu saling bertetangga (8 arah).
   - Jika semua terpenuhi, pencarian berhenti dan solusi dinyatakan ditemukan.
4. Mekanisme Backtracking Iteratif

   Jika pada suatu baris semua kolom sudah dicoba namun tidak menghasilkan solusi:
   - algoritma mundur ke baris sebelumnya,
   - membatalkan penempatan terakhir pada baris tersebut,
   - melanjutkan percobaan ke kolom berikutnya.
   - Proses maju-mundur ini berulang sampai solusi ditemukan atau seluruh kemungkinan habis dieksplorasi.
5. Kondisi Berhenti
   - Berhasil: ditemukan satu konfigurasi valid.
   - Gagal: telah kembali melewati baris awal (tidak ada lagi state yang bisa dieksplorasi).

BAB III

Implementasi Program Dengan Java

Algoritma *Brute Force* dan struktur data yang telah dirancang pada Bab II diimplementasikan menggunakan bahasa pemrograman Java. Pendekatan yang digunakan adalah *Object-Oriented Programming* (OOP), di mana logika permainan dan algoritma pencarian dipisahkan ke dalam kelas-kelas yang berbeda untuk menjaga modularitas kode.

Terdapat dua kelas utama yang menjadi inti (*core logic*) dari program ini, yaitu **Kelas Board** sebagai representasi model data, dan **Kelas Solver** sebagai mesin eksekusi algoritma.

3.1 Board.java

```
public class Board {
    private int size;
    private char[][] color;
    private boolean[][] queens;
    private Map<Character, List<Cell>> cells;

    public Board(int size){
        this.size = size;
        this.color = new char[size][size];
        this.queens = new boolean[size][size];
        this.cells = new HashMap<>();
    }

    public void addCellToRegion(int r, int c, char color) {
        this.color[r][c] = color;
        this.cells.putIfAbsent(color, new ArrayList<>());
        this.cells.get(color).add(new Cell(r, c, color));
    }
    public List<Character> getAllRegions() {
        return new ArrayList<>(cells.keySet());
    }

    public char getRegion(int row, int col) {
        return color[row][col];
    }
```

```
public int getSize(){
    return this.size;
}

public void placeQueen(int row, int col){
    this.queens[row][col] = true;
}

public void removeQueen(int row, int col){
    this.queens[row][col] = false;
}
public boolean[][] getQueens() {
    return queens;
}

public synchronized boolean[][] copyQueens() {
    int n = queens.length;
    boolean[][] copy = new boolean[n][n];
    for (int row = 0; row < n; row++) {
        System.arraycopy(queens[row], 0, copy[row], 0, n);
    }
    return copy;
}

public boolean isValid(){
    int count = 0;
    for (int r = 0; r < size; r++) {
    for (int c = 0; c < size; c++) {
            if (queens[r][c]) count++;
        }
    }
    if (count != size) return false;
    for (int r1 = 0; r1 < size; r1++){
        for(int c1 = 0; c1 < size; c1++){
            if(queens[r1][c1]) {
                for(int r2 = 0; r2 < size; r2++){
                    for(int c2 = 0; c2 < size; c2++){
                        if (r1 == r2 && c1 == c2) continue;
```

```
                    if (queens[r2][c2]) {
                        if (r1 == r2) return false;
                        if (c1 == c2) return false;
                        if (color[r1][c1] == color[r2][c2]) return false;

                        int selisihRow = Math.abs(r1 - r2);
                        int selisihCol = Math.abs(c1 - c2);
                        if (selisihRow <= 1 && selisihCol <=1) return false;

                    }
                }
            }
        }
    }
    return true;
}
```

Kelas Board berfungsi sebagai struktur data utama yang menyimpan keadaan permainan. Kelas ini mengenkapsulasi atribut-atribut dimensi papan, peta wilayah, serta posisi ratu saat ini.

Atribut Utama:

- private int size: Menyimpan dimensi papan (*N*).
- private char[][] color: Menyimpan region/warna tiap sel.
- private boolean[][] queens: Matriks *boolean* yang menandai keberadaan ratu. Nilai true menandakan adanya ratu pada posisi tersebut.
- private Map<Character, List<Cell>> cells: daftar sel berdasarkan region.

Konstruktor Board(int size) menginisialisasi seluruh matriks dan peta wilayah. Untuk mengisi data wilayah dari masukan pengguna, digunakan metode addCellToRegion(int r, int c, char color) yang secara simultan memperbarui color dan cells.

Manajemen posisi ratu dilakukan melalui metode placeQueen dan removeQueen yang mengubah status boolean pada koordinat tertentu. Selain itu, terdapat metode utilitas copyQueens() yang melakukan *deep copy* terhadap matriks ratu, yang berguna untuk keperluan visualisasi pada GUI tanpa mengganggu proses pencarian utama.

Bagian paling penting dari kelas ini adalah metode boolean isValid(). Metode ini melakukan verifikasi menyeluruh terhadap konfigurasi papan saat ini. Langkah validasi yang dilakukan meliputi:

1. **Pengecekan Kuantitas**: Memastikan jumlah ratu yang terpasang (count) sama dengan size.

2. **Pengecekan Konflik**: Menggunakan iterasi bersarang (*nested loops*) untuk membandingkan setiap pasangan ratu *(r1, c1)* dan *(r2, c2)*. Validasi mengembalikan false jika:
    - Terdapat dua ratu pada baris atau kolom yang sama.
    - Terdapat dua ratu pada wilayah warna yang sama (color identik).
    - Terdapat dua ratu yang saling bertetangga dalam jarak 1 petak.

3.2 Solver.java

```
public class Solver {
    private Board board;
    private boolean found;
    private long totalCase;

    public Solver(Board board) {
        this.board = board;
        this.found = false;
        this.totalCase = 0;
    }

    public void solve(){
        found = false;
        totalCase = 0;

        int n = board.getSize();
        boolean[][] q = board.getQueens();
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (q[i][j]) {
                    board.removeQueen(i, j);
                }
            }
        }

        int[] p = new int[n];
        Arrays.fill(p, -1);

        int i = 0;
        while (i >= 0) {
            boolean adv = false;
            int s = p[i] + 1;
```

```
            for (int j = s; j < n; j++){
                totalCase++;

                board.placeQueen(i, j);
                p[i] = j;

                if (i == n - 1) {
                    if (board.isValid()){
                        found = true;
                        return;
                    }
                    board.removeQueen(i, j);
                    p[i] = -1;
                    continue;
                }

                i++;
                p[i] = -1;
                adv = true;
                break;
            }

            if (!adv){
                p[i] = -1;
                i--;
                if (i >= 0){
                    int k = p[i];
                    if (k >= 0) {
                        board.removeQueen(i, k);
                    }
                }
            }
        }
    }
```

Kelas Solver (terlampir pada Solver.java) bertanggung jawab untuk menjalankan algoritma pencarian solusi. Kelas ini memisahkan logika pencarian dari struktur data papan, menerima objek Board sebagai dependensi melalui konstruktornya.

Atribut Utama:

- private Board board: Referensi ke objek papan yang akan diselesaikan.
- private long totalCase: Penghitung jumlah iterasi atau konfigurasi yang telah diperiksa.
- private boolean found: Status solusi ditemukan/tidak.

Inti dari kelas ini adalah metode solve(), yang mengimplementasikan algoritma *Iterative Backtracking*. Berbeda dengan pendekatan rekursif, metode ini menggunakan larik integer int[] p sebagai representasi tumpukan (*stack*) posisi kolom ratu pada setiap baris.

Alur eksekusi metode solve() adalah sebagai berikut:

1. **Reset**: Papan dikosongkan dan larik p diinisialisasi dengan nilai -1.
2. **Iterasi (While Loop)**: Program memasuki perulangan utama selama indeks baris i >= 0.
3. **Ekspansi**: Pada baris aktif, program mencoba menempatkan ratu pada kolom selanjutnya yang tersedia. Variabel totalCase dinaikkan setiap kali penempatan dilakukan.
4. **Validasi Akhir**:
    - Jika program mencapai baris terakhir, metode memanggil board.isValid() untuk memeriksa keabsahan solusi total.
    - Jika valid, bendera found diatur menjadi true dan proses berhenti.
    - Jika tidak, program melakukan *backtrack*.
5. **Backtracking**: Jika tidak ada kolom yang valid pada baris saat ini, indeks baris dikurangi (i--), dan ratu pada baris sebelumnya dihapus untuk memungkinkan eksplorasi cabang lain.

BAB IV

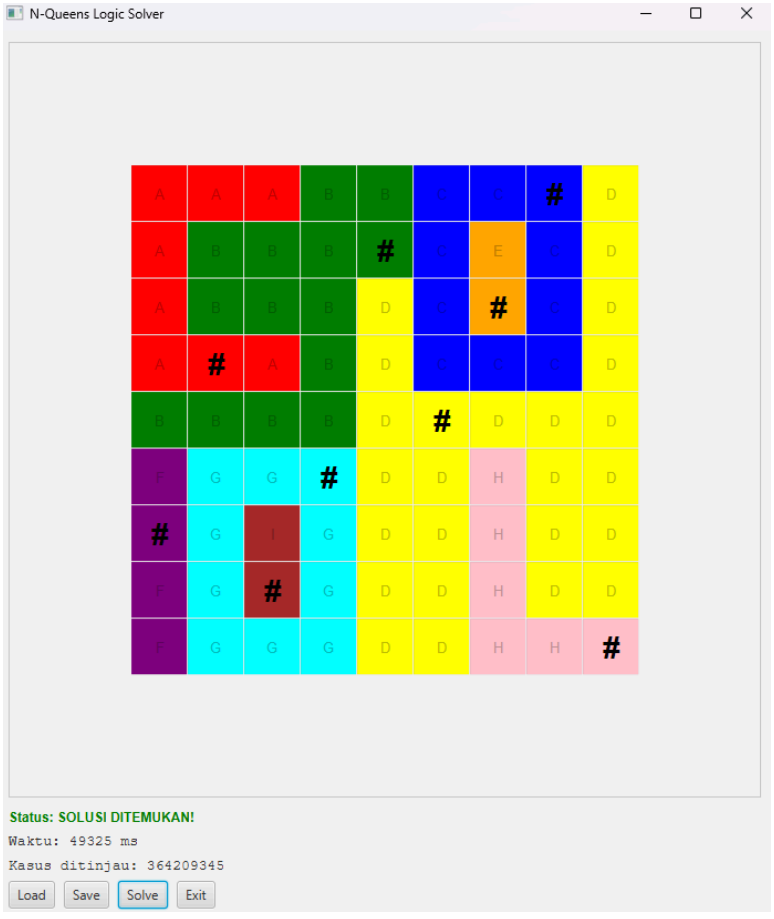
Eksperimen

4.1 Valid

Kondisi diterima hanya menerima konfigurasi berukuran NxN

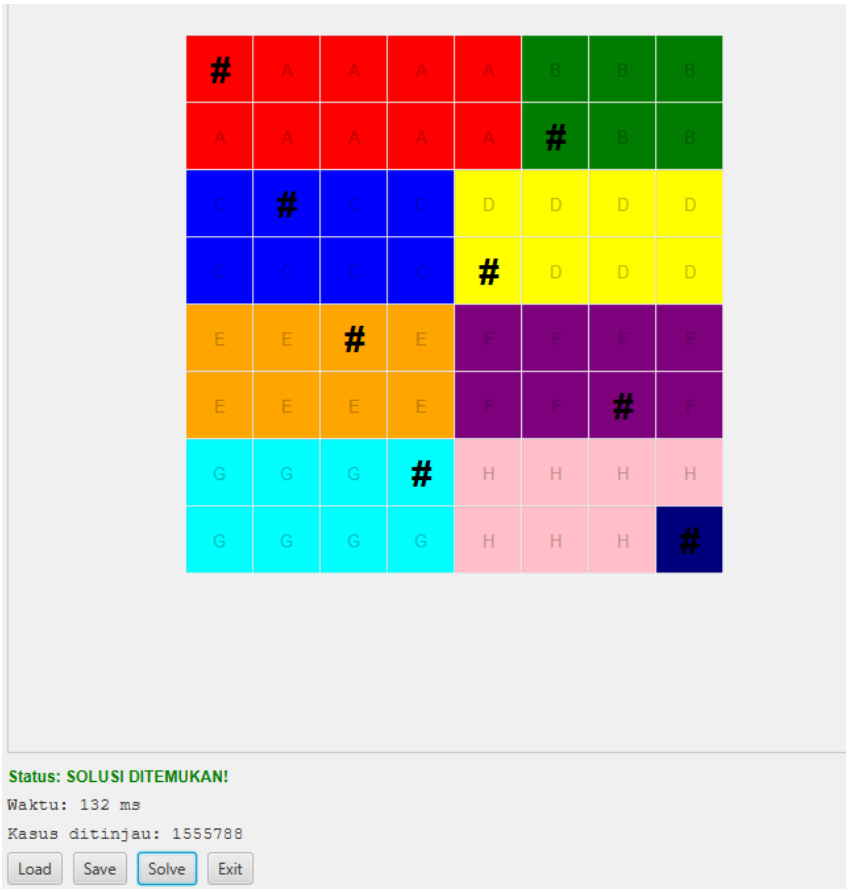
a. 9x9

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



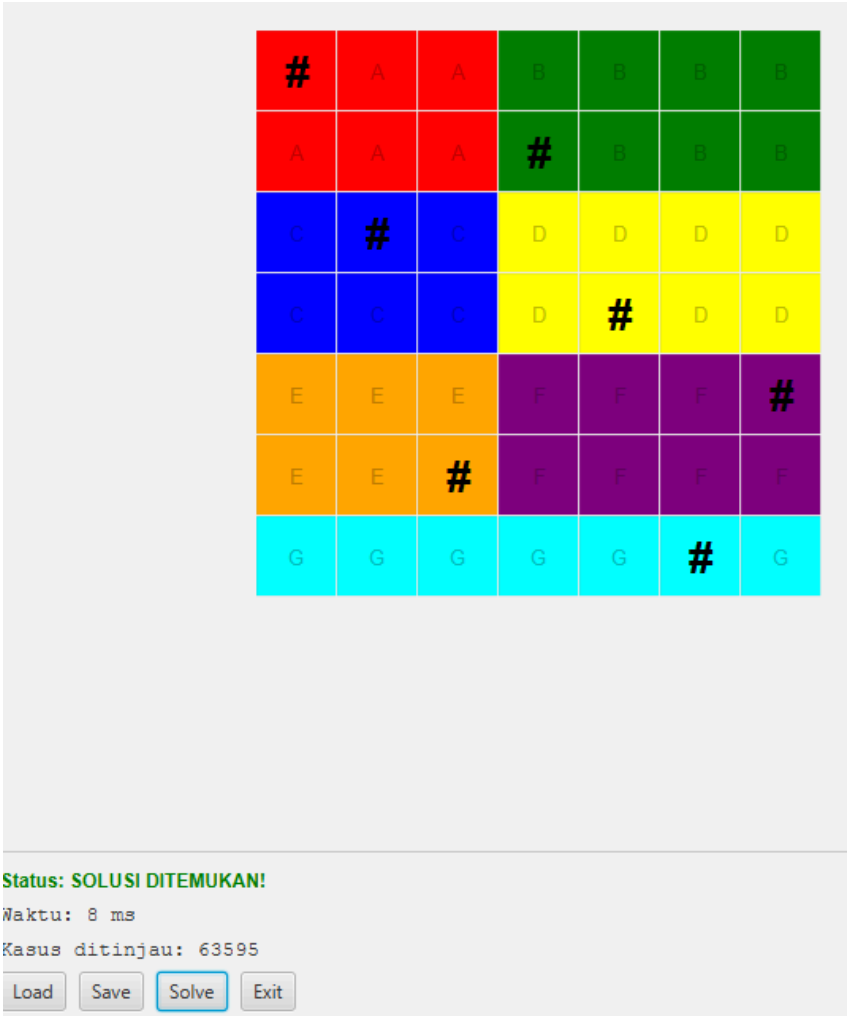
b. 8x8

```
AAAAABBB
AAAAABBB
CCCCDDDD
CCCCDDDD
EEEEFFFF
EEEEFFFF
GGGGHHHH
GGGGHHHL
```



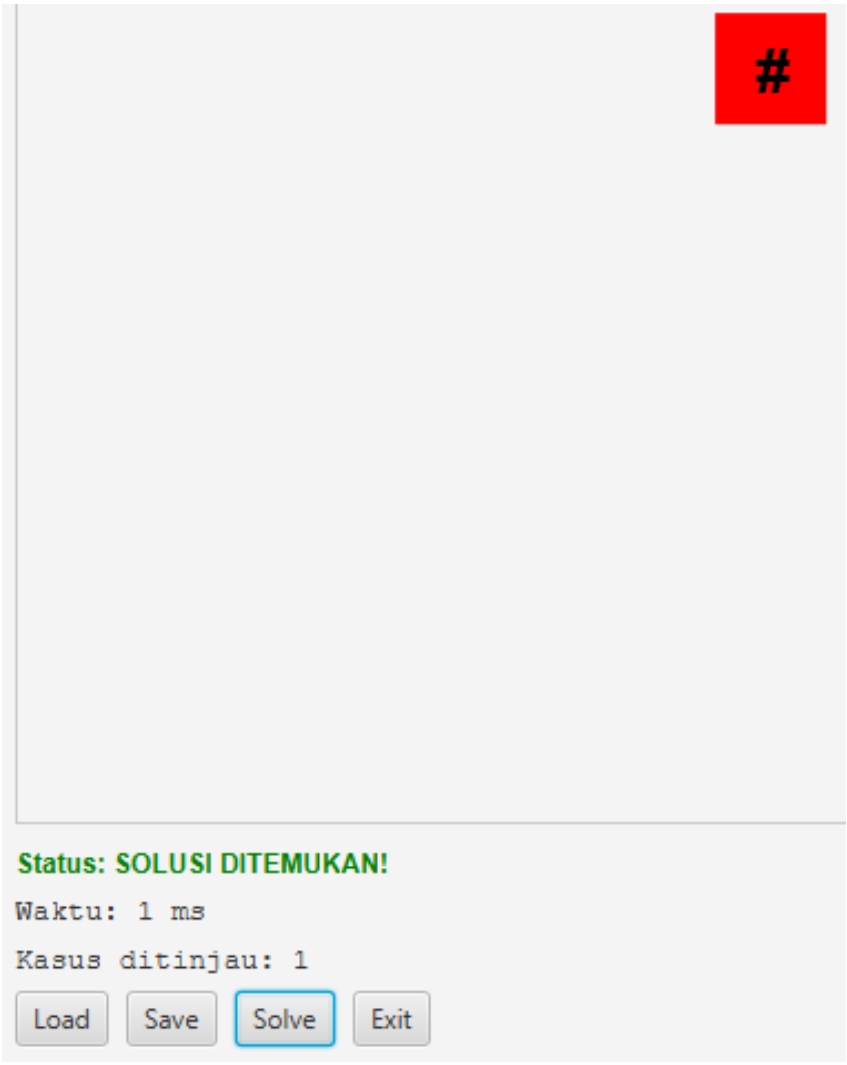
c. 7x7

```
AAABBBB
AAABBBB
CCCDDDD
CCCDDDD
EEEFFFF
EEEFFFF
GGGGGGG
```


#AABBBBB
AAA#BBB
C#CDDDD
CCCD#DD
EEEFFF#
EE#FFFF
GGGGG#G
Status: SOLUSI DITEMUKAN!
Waktu: 8 ms
Kasus ditinjau: 63595
Load
Save
Solve
Exit

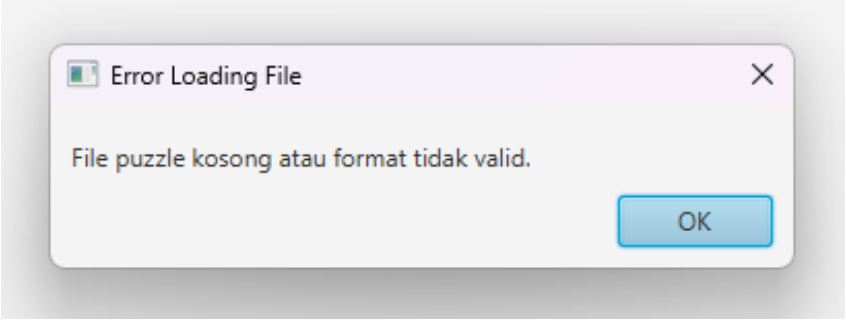
d. 1x1

A



4.2 Tidak Valid

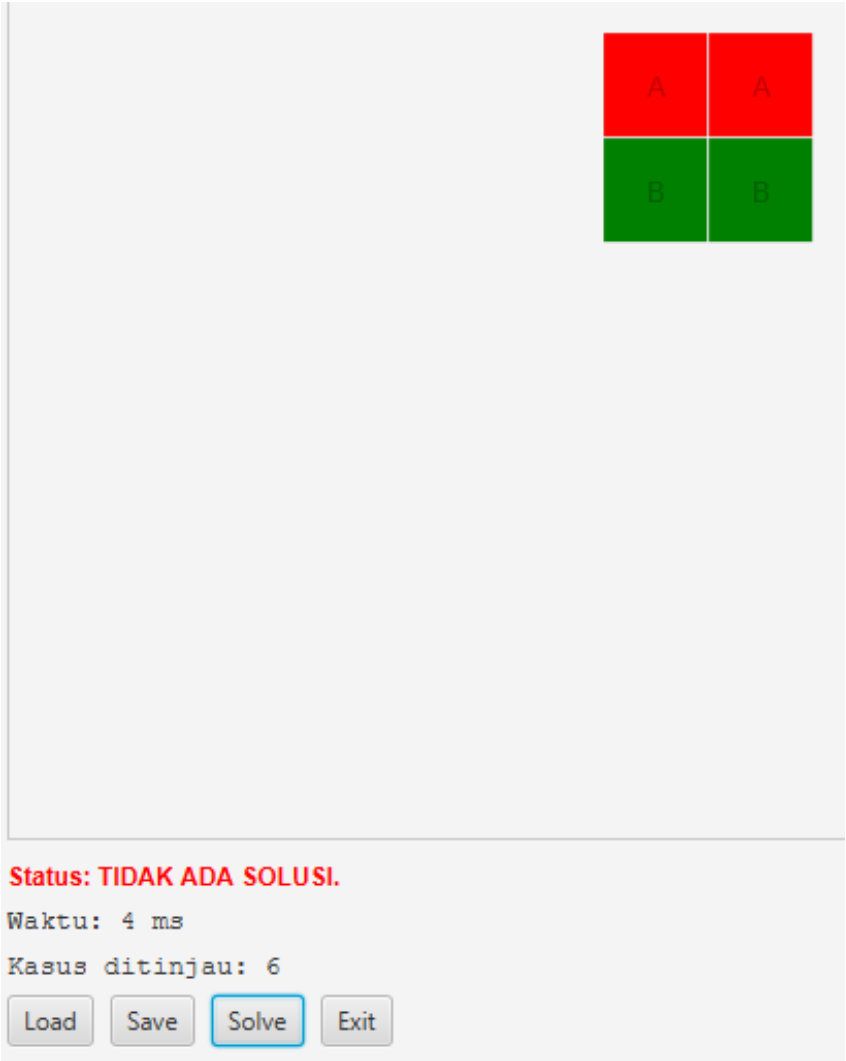
a. File Kosong



b. 2x2 atau Tidak ada solusi

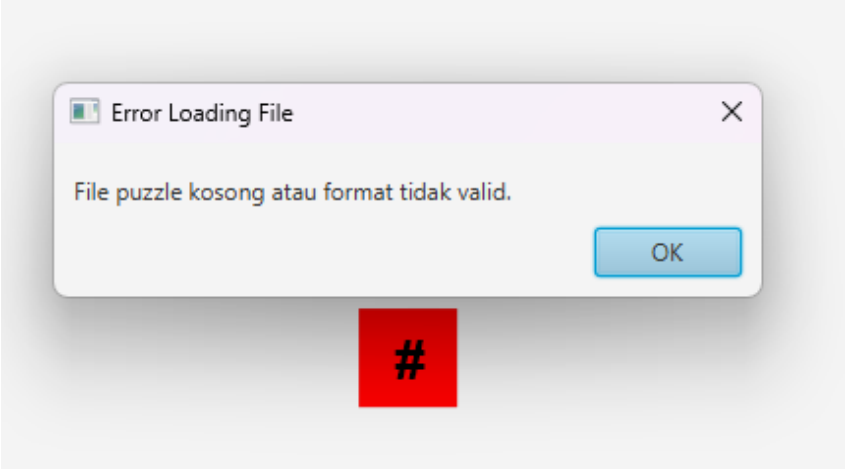
AA

```
BB
```



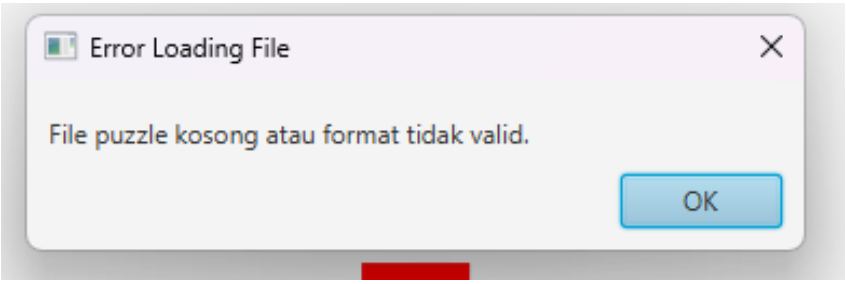
c. Ada warna yang kosong

```
AAABB B
ACCCB B
ACDDB B
ECDDF F
EEDFF F
EEEFF F
```



d. 6x7

```
AAABBB
ACCCBB
ACDDBB
ECDDFF
EEDFFF
EEEFFF
FFFFFF
```



BAB 5

PENUTUP

5.1 Tautan

https://github.com/daypft/tucil1_13524133

5.2 Lampiran

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 (*Generatuve AI)*. Melainkan hasil pemikiran dan Analisis Sendiri.

Muhammad Daffa Arrizki Yanma
13524133