Laporan Tugas Kecil 1
Mata Kuliah Strategi Algoritma

Audric Yusuf Maynard Simatupang - 13524010
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
cubapetuking@gmail.com, 13524010@std.stei.itb.ac.id

Bab 1

Pendahuluan

1.1. Latar Belakang

Queens (LinkedIn) merupakan permainan teka-teki logika pada papan berukuran N * N yang dibagi menjadi beberapa wilayah berwarna. Tujuan permainan ini adalah untuk menempatkan tepat N jumlah ratu pada papan tersebut dengan mengikuti aturan yang sudah ditetapkan. Pada umumnya, aturan yang berlaku pada permainan Queens adalah sebagai berikut :

- Setiap baris harus memiliki tepat satu ratu
- Setiap kolom harus memiliki tepat satu ratu
- Setiap wilayah berwarna harus memiliki tepat satu waktu
- Tidak ada dua buah ratu yang bersebelahan, termasuk secara diagonal

Permainan ini umumnya dipandang sebagai masalah pemenuhan kendala, yaitu pemain harus menentukan posisi ratu yang memenuhi segala larangan yang diberikan. Akan tetapi, berbeda dengan N-Queens klasik pada catur, permainan ini tidak melarang penempatan satu diagonal panjang, selama ratu tidak tepat bersebelahan.

Ada berbagai algoritma yang dapat digunakan untuk menyelesaikan permainan Queens. Namun, hal yang cukup mengejutkan adalah permainan ini dapat diselesaikan dengan cukup andal menggunakan algoritma Brute Force, salah satu algoritma termudah untuk diimplementasi. Oleh karena itu, penggunaan algoritma Brute Force untuk menyelesaikan permainan Queens menjadi pembahasan utama untuk laporan ini.

Bab 2

Dasar Teori

2.1. Algoritma Brute Force

Algoritma Brute Force merupakan pendekatan penyelesaian masalah dengan mencoba segala kemungkinan solusi pada ruang pencarian, kemudian mengambil solusi yang memenuhi kriteria yang ditetapkan. Algoritma ini umumnya relatif mudah untuk diimplementasi dan menjamin solusi ditemukan, apabila memang terdapat solusi pada permasalahan tersebut. Akan tetapi, algoritma ini umumnya memiliki kompleksitas waktu yang cukup tinggi sehingga masukan yang besar akan memakan waktu yang sangat banyak. Oleh karena itu, umumnya, Algoritma Brute Force dioptimalisasikan menggunakan beberapa teknik, seperti Backtracking, Pruning, dan lain lain.

Pada laporan ini, Algoritma Brute Force yang digunakan adalah Naive Brute Force. Algoritma ini mencari setiap kemungkinan solusi tanpa menggunakan heuristik maupun strategi optimasi untuk memperkecil ruang pencarian. Algoritma menghasilkan konfigurasi solusi lalu memvalidasikan solusi terhadap aturan permainan. Apabila konfigurasi tersebut sudah memenuhi aturan permainan, algoritma tersebut akan menyimpan solusi tersebut. Jika tidak, algoritma akan menghasilkan konfigurasi selanjutnya dan melakukan validasi ulang sampai menemukan konfigurasi yang valid.

Bab 3

Metodologi

3.1. Aturan Permainan

Pada program yang telah dibuat, ada beberapa aturan permainan yang dipatuhi oleh program. Aturan tersebut bisa dibagi menjadi 3 bagian, yaitu :

1. Aturan pemilihan warna
   Berikut adalah aturan pemilihan warna papan pada program yang telah dibuat :
   - Apabila terdapat papan berukuran N * N, jumlah wilayah berwarna harus berjumlah tepat N buah.
   - Tidak boleh ada bagian papan yang tidak memiliki warna.
   - Setiap wilayah berwarna harus kontinu, yang berarti tidak ada dua kelompok warna identik yang terpisah.
2. Aturan pembuatan papan
   Hanya terdapat satu aturan terkait pembuatan papan, yaitu papan harus berbentuk segi empat.
3. Aturan penempatan ratu
   Berikut adalah aturan penempatan ratu pada program yang telah dibuat :
   - Setiap baris harus memiliki tepat satu ratu.
   - Setiap kolom harus memiliki tepat satu ratu.
   - Setiap wilayah berwarna harus memiliki tepat satu waktu.
   - Tidak ada dua buah ratu yang bersebelahan, termasuk secara diagonal.

3.2. Implementasi Brute Force

3.2.1. Konsep Dasar

Tentu saja, konsep paling utamanya adalah untuk mencari segala kemungkinan konfigurasi posisi ratu. Akan tetapi, bagaimana cara program dapat mencari segala kemungkinan tersebut? Berikut adalah konsep dasar yang digunakan program untuk mencari segala kemungkinan konfigurasi posisi ratu.

Asumsikan terdapat papan berukuran 5 x 5. Pada papan ini, program harus meletakkan tepat 5 buah ratu. Anggap semua kotak pada papan tersebut diurutkan menjadi suatu baris lalu setiap kotak ditandai dengan angka dari 1 hingga 25. Ratu juga ditandai dengan angka 1 hingga 5. Ratu pun diletakkan di atas kotak 1 hingga kotak 5, dengan ratu 1 berada pada kotak 1, ratu 2 pada kotak 2, dan seterusnya. Ini merupakan konfigurasi pertama yang dihasilkan program. Untuk menghasilkan konfigurasi kedua, ratu ke-5 dipindahkan satu kotak ke kanan. Apabila ratu ke-5 sudah mencapai akhir baris kotak atau sudah menabrak ratu yang lain, ratu ke-4 dipindahkan satu kotak ke kanan dan ratu ke-5 dipindahkan ke sebelah

kanannya ratu ke-4. Begitu seterusnya sampai konfigurasi akhir, yaitu semua ratu sudah berada di sebelah kanan.

Pada setiap konfigurasi, baris kotak tersebut kemudian disusun kembali menjadi papan, lalu divalidasikan terhadap aturan permainan yang sudah ditetapkan.

3.2.2. Implementasi Algoritma Brute Force

Berikut adalah kode keseluruhan untuk algoritma Brute Force dan validasinya :

- Algoritma Brute Force

```
package com.mycompany.app.backend.data;

import java.util.function.LongConsumer;

public class BruteSolver {
    private BoardObject boardObject;
    private ColorObject colorObject;
    private SolutionObject solutionObject;
    private GameLogic gameLogic;

    public BruteSolver(BoardObject boardObject, ColorObject colorObject,
SolutionObject solutionObject) {
        this.boardObject = boardObject;
        this.colorObject = colorObject;
        this.solutionObject = solutionObject;
        this.gameLogic = new GameLogic(boardObject, colorObject);
    }

    public void exhaustiveBrute(LongConsumer tick, long period) {
        long startTimer = System.nanoTime();
        int[] queen = new int[boardObject.getBoardLength()];
        for (int i = 0 ; i < boardObject.getBoardLength() ; i++) {
            queen[i] = i + 1;
        }

        int tempIter = 0;
        while (true) {
            applyToBoard(queen);

            tempIter++;
            solutionObject.incrementIteration();

            if (tick != null && period > 0 && (tempIter % period == 0)) {
```

```
                tick.accept(tempIter);
            }

            if (gameLogic.isValidBoard()) {
                solutionObject.addSolution(queen.clone());
                break;
            }
            if (!nextCombination(queen, boardObject.getBoardLength())) break;
        }
        long endTimer = System.nanoTime();
        int timeTaken = (int) ((endTimer - startTimer) / 1000000);
        solutionObject.setTime(timeTaken);

        if (tick != null) tick.accept(tempIter);
    }

    public boolean nextCombination(int[] queen, int n) {
        int max = n*n;
        for (int i = queen.length - 1 ; i >= 0 ; i--) {
            int limit = max - (queen.length - 1 - i);

            if (queen[i] < limit) {
                queen[i]++;
                for (int j = i + 1 ; j < queen.length ; j++) {
                    queen[j] = queen[j-1] + 1;
                }
                return true;
            }
        }
        return false;
    }

    public void applyToBoard(int[] queen) {
        boardObject.resetBoard();
        for (int i = 0 ; i < queen.length ; i++) {
            int row = (queen[i] - 1)/boardObject.getBoardLength();
            int col = (queen[i] - 1)%boardObject.getBoardLength();
            boardObject.setQueen(row, col);

        }
    }

}
```

- Validasi

```
package com.mycompany.app.backend.data;

public class GameLogic {
    private BoardObject boardObject;
    private ColorObject colorObject;

    public GameLogic(BoardObject boardObject, ColorObject colorObject) {
        this.boardObject = boardObject;
        this.colorObject = colorObject;
    }

    public boolean checkOccupied(int x, int y) {
        if (boardObject.getTile(x, y) == 1) {
            return false;
        }
        return true;
    }

    public boolean checkAdjacent(int x, int y) {
        for (int i = x-1 ; i <= x+1 ; i++) {
            for (int j = y-1 ; j <= y+1 ; j++) {
                if (boardObject.getTile(i, j) == -1) {
                    continue;
                }
                if (boardObject.getTile(i, j) == 1) {
                    return false;
                }
            }
        }
        return true;
    }

    public boolean checkLine(int x, int y) {
        // check horizontal line first
        for (int i = 0 ; i < boardObject.getBoardLength() ; i++) {
            if (boardObject.getTile(x, i) == 1) {
                return false;
            }
        }

        // check vertical
        for (int i = 0 ; i < boardObject.getBoardLength() ; i++) {
            if (boardObject.getTile(i, y) == 1) {
                return false;
```

```
            }
        }

        return true;
    }

    public boolean checkColor(int x, int y) {
        int colorCode = colorObject.getColorTile(x, y);
        for (int i = 0 ; i < colorObject.getLength() ; i++) {
            for (int j = 0 ; j < colorObject.getLength() ; j++) {
                if (colorObject.getColorTile(i, j) == colorCode &&
boardObject.getTile(i, j) == 1) {
                    return false;
                }
            }
        }

        return true;
    }

    public boolean isValidQueen(int x, int y) {
        if (!checkOccupied(x, y)) return false;
        if (!checkAdjacent(x, y)) return false;
        if (!checkColor(x, y)) return false;
        if (!checkLine(x, y)) return false;
        return true;
    }

    public boolean isValidBoard() {
        for (int i = 0 ; i < boardObject.getBoardLength() ; i++) {
            for (int j = 0 ; j < boardObject.getBoardLength() ; j++) {
                if (boardObject.getTile(i, j) == 1) {
                    boardObject.deleteQueen(i, j);
                    if (!isValidQueen(i, j)) {
                        boardObject.setQueen(i, j);
                        return false;
                    }
                    boardObject.setQueen(i, j);
                }
            }
        }
        return true;
    }
}
```

Pada kedua kelas tersebut, terdapat beberapa fungsi-fungsi utama yang menjadi inti algoritma Brute Force.

Berikut adalah fungsi-fungsi penting pada kelas BruteSolver :

- exhaustiveBrute

```
public void exhaustiveBrute(LongConsumer tick, long period) {
        long startTimer = System.nanoTime();
        int[] queen = new int[boardObject.getBoardLength()];
        for (int i = 0 ; i < boardObject.getBoardLength() ; i++) {
            queen[i] = i + 1;
        }

        int tempIter = 0;
        while (true) {
            applyToBoard(queen);

            tempIter++;
            solutionObject.incrementIteration();

            if (tick != null && period > 0 && (tempIter % period == 0)) {
                tick.accept(tempIter);
            }

            if (gameLogic.isValidBoard()) {
                solutionObject.addSolution(queen.clone());
                break;
            }
            if (!nextCombination(queen, boardObject.getBoardLength()))
break;
        }
        long endTimer = System.nanoTime();
        int timeTaken = (int) ((endTimer - startTimer) / 1000000);
        solutionObject.setTime(timeTaken);

        if (tick != null) tick.accept(tempIter);
    }
```

Kode ini merupakan bagian utama Algoritma Brute Force yang digunakan oleh program. Tanggung jawab utama fungsi ini adalah sebagai inisiator, thread, dan juga manager kodenya. Fungsi ini memanggil fungsi lain untuk membuat kemungkinan konfigurasi solusi selanjutnya, kemudian memanggil fungsi untuk mengubah array queen menjadi papan lalu mengirimkan papan tersebut ke fungsi lain untuk divalidasi.

- nextCombination

```
public boolean nextCombination(int[] queen, int n) {
    int max = n*n;
    for (int i = queen.length - 1 ; i >= 0 ; i--) {
        int limit = max - (queen.length - 1 - i);

        if (queen[i] < limit) {
            queen[i]++;
            for (int j = i + 1 ; j < queen.length ; j++) {
                queen[j] = queen[j-1] + 1;
            }
            return true;
        }
    }
    return false;
}
```

Fungsi ini memiliki tanggung jawab untuk melakukan generasi kemungkinan konfigurasi solusi selanjutnya. Pertama, fungsi ini mencari tau batasan untuk setiap ratu, misalnya n = 5, maka batasan untuk setiap ratu adalah queen[0] = 21, queen[1] = 22, dan seterusnya. Lalu, program melakukan pengecekan terhadap posisi ratu. Jika masih berada di batas bawah limit, ratu akan di-increment, lalu akan masuk ke dalam for-loop. For-loop akan berjalan jika i = 3 ke bawah untuk memastikan ratu yang sudah mencapai limit akan ditarik kembali ke posisi yang paling kecil yang memungkinkan. Fungsi ini akan mengembalikan true jika menemukan konfigurasi posisi ratu dan mengembalikan false jika tidak lagi menemukan konfigurasi yang memungkinkan sehingga program akan berhenti.

- applyToBoard

```
public void applyToBoard(int[] queen) {
    boardObject.resetBoard();
    for (int i = 0 ; i < queen.length ; i++) {
        int row = (queen[i] - 1)/boardObject.getBoardLength();
        int col = (queen[i] - 1)%boardObject.getBoardLength();
        boardObject.setQueen(row, col);

    }
}
```

Fungsi ini bertanggung jawab untuk mengubah array queen menjadi papan yang bisa diproses oleh validator. Caranya adalah dengan membagi posisi ratu dengan panjang papan untuk mendapatkan baris dan melakukan operasi modulo posisi ratu dengan panjang papan untuk mendapatkan kolom.

Berikut adalah fungsi-fungsi penting pada kelas GameLogic :

- checkOccupied

```
public boolean checkOccupied(int x, int y) {
    if (boardObject.getTile(x, y) == 1) {
        return false;
    }
    return true;
}
```

Fungsi ini bertanggung jawab untuk mengecek apakah suatu kotak sudah ditempati sebuah ratu. Fungsi ini mengembalikan true jika kotak tersebut kosong.

- checkAdjacent

```
public boolean checkAdjacent(int x, int y) {
    for (int i = x-1 ; i <= x+1 ; i++) {
        for (int j = y-1 ; j <= y+1 ; j++) {
            if (boardObject.getTile(i, j) == -1) {
                continue;
            }
            if (boardObject.getTile(i, j) == 1) {
                return false;
            }
        }
    }
    return true;
}
```

Fungsi ini bertanggung jawab untuk mengecek apakah sudah ada ratu yang menempati kotak di sekitar kotak yang di-input. Fungsi ini mengembalikan true jika tidak ada ratu di sekitar kotak yang di-input.

- checkLine

```
public boolean checkLine(int x, int y) {
    // check horizontal line first
    for (int i = 0 ; i < boardObject.getBoardLength() ; i++) {
        if (boardObject.getTile(x, i) == 1) {
            return false;
        }
    }

    // check vertical
    for (int i = 0 ; i < boardObject.getBoardLength() ; i++) {
```

```
            if (boardObject.getTile(i, y) == 1) {
                return false;
            }
        }

        return true;
    }
```

Fungsi ini bertanggung jawab untuk mengecek apakah ada ratu yang sudah menempati kotak yang sebaris dan sekolom dengan kotak yang diinput. Fungsi ini mengembalikan true jika tidak ada ratu yang menempati kotak yang sebaris dan sekolom dengan kotak yang diinput.

- checkColor

```
    public boolean checkColor(int x, int y) {
        int colorCode = colorObject.getColorTile(x, y);
        for (int i = 0 ; i < colorObject.getLength() ; i++) {
            for (int j = 0 ; j < colorObject.getLength() ; j++) {
                if (colorObject.getColorTile(i, j) == colorCode &&
boardObject.getTile(i, j) == 1) {
                    return false;
                }
            }
        }

        return true;
    }
```

Fungsi ini bertanggung jawab untuk mengecek apakah ada ratu yang sudah menempati wilayah berwarna yang ditempati kotak yang diinput. Fungsi ini mengembalikan true jika tidak ada ratu yang menempati kotak yang sewarna dengan kotak yang diinput.

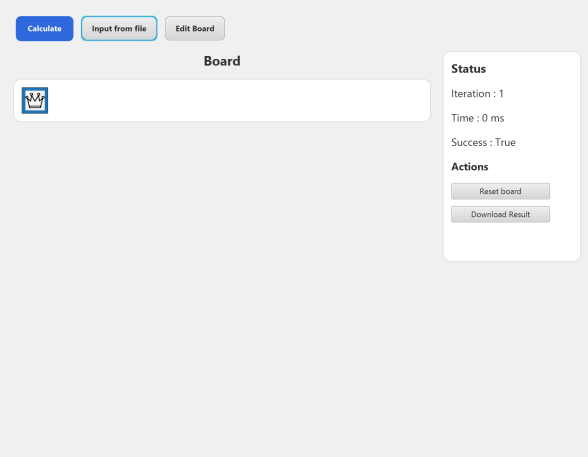
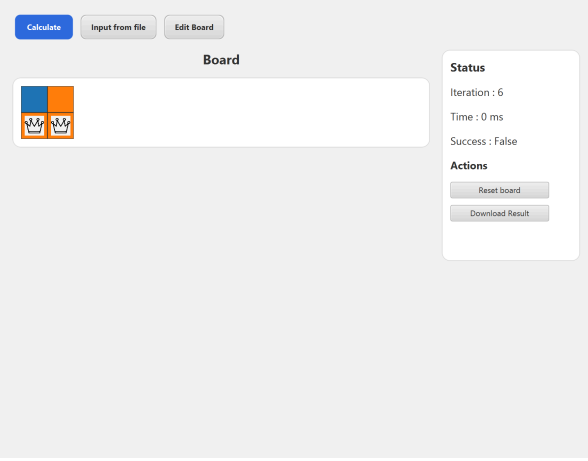
3.2.3. Implementasi GUI

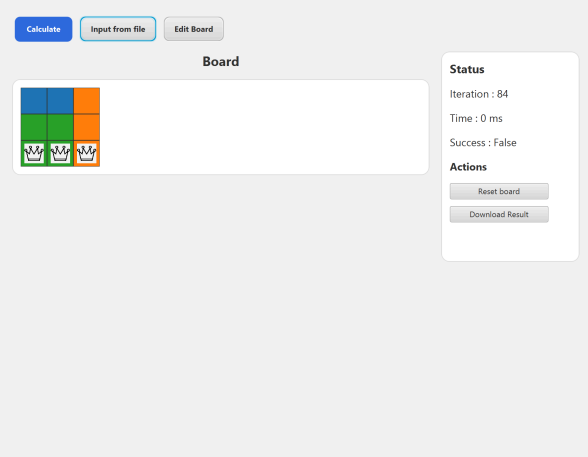
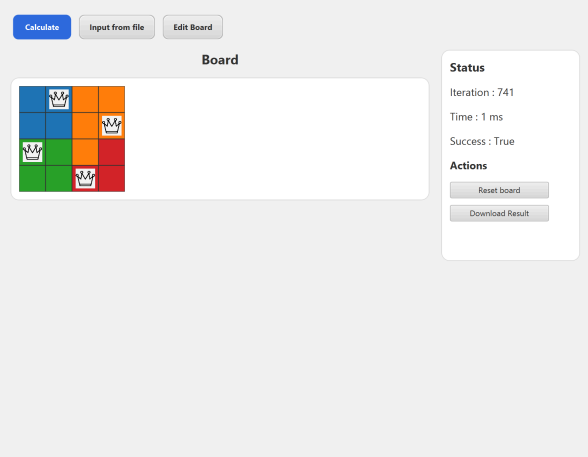
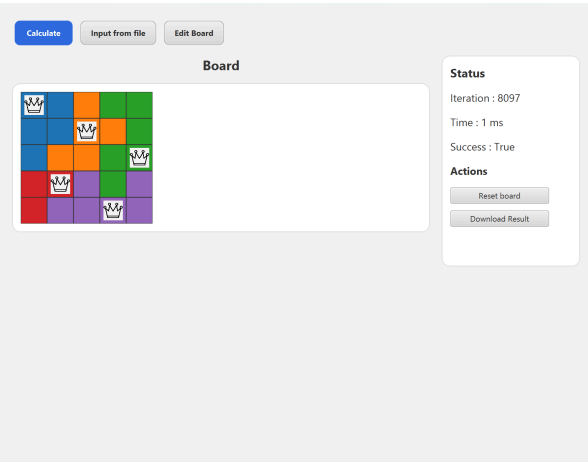
Implementasi GUI dilakukan dengan menggunakan JavaFX. Model yang digunakan adalah MVC (Model View Controller) untuk source code selengkapnya akan dilampirkan pada lampiran.

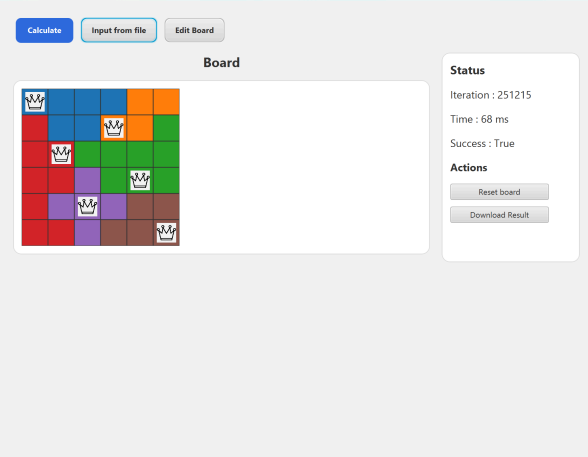
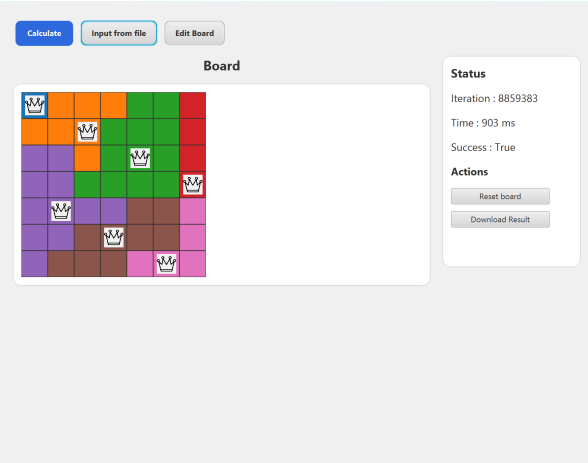
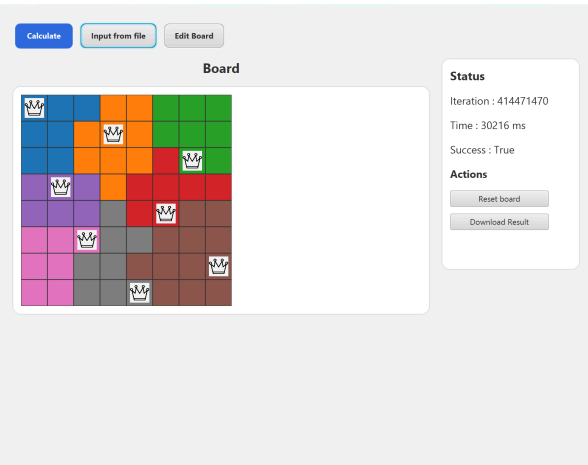
Bab 4
Hasil Percobaan

4.1. Hasil dan Analisis

Percobaan dilakukan menggunakan papan berukuran 1 x 1 hingga 8 x 8. Berikut adalah hasil percobaannya :

Ukuran	Input	Output	Analisis
1x1	A		Pada kasus ini, hanya terdapat satu kemungkinan, yaitu pada satu-satunya kotak yang tersedia. Program menemukan jawaban tersebut.
2x2	AB BB		Pada kasus ini, tidak terdapat solusi yang memenuhi aturan. Program berhenti pada kemungkinan terakhir, yaitu ratu pada baris terakhir. Status success menyatakan false yang berarti tidak ada solusi untuk papan ini.

3x3	AAB CCB CCB		Kasus ini sama seperti kasus kedua. Program juga menyatakan bahwa success adalah false yang berarti tidak ada solusi untuk papan ini.
4x4	AABB AABB CCBD CCDD		Pada kasus ini, program berhasil menemukan solusinya. Waktu yang dibutuhkan adalah 1 ms dan iterasi yang dibutuhkan adalah 741 iterasi.
5x5	AABCC AABBC ABBCC DDECE DEEEE		Pada kasus ini, program juga berhasil menemukan solusinya. Waktu yang dibutuhkan tetap 1 ms, tetapi iterasi yang dibutuhkan lebih banyak. Hal ini menunjukkan salah satu kelemahan Brute Force, yaitu kompleksitas yang lebih tinggi untuk data yang lebih besar.

6x6	AAAABB DAABBC DDCCCC DDECCC DEEEFF DDEFFF		Pada kasus ini, program juga menemukan sebuah solusi. Waktu dan iterasi yang dibutuhkan untuk menemukan solusi jauh lebih tinggi daripada uji kasus 5x5.
7x7	ABBBCCD BBBCCCD EEBCCCD EECCCCD EEEEFFG EEFFFFG EFFFGGG		Pada kasus ini, program juga menemukan sebuah solusi. Waktu dan iterasi yang dibutuhkan juga lebih tinggi daripada uji kasus sebelumnya.
8x8	AAABBCCC AABBBCCC AABBBDCC EEEBDDDD EEEHDDFF GGGHHFFF GGHHFFFF GGHHHFFF		Pada kasus ini, hal yang sama terjadi seperti uji kasus sebelumnya. Solusi ditemukan tetapi waktu dan iterasi jauh lebih tinggi.

4.2 Penarikan Kesimpulan

Uji kasus ini membuktikan dua pernyataan, yaitu kompleksitas algoritma Brute Force jauh lebih tinggi untuk masalah yang lebih besar serta algoritma Brute Force dapat diandalkan untuk menemukan solusinya. Kedua pernyataan tersebut didukung dengan data yang didapat, yaitu waktu dan iterasi selalu meningkat drastis ketika papan lebih besar, tetapi solusi selalu ditemukan jika papan tersebut memang memiliki solusi.

Bab 5

Penutup

5.1. Pranala GitHub

Berikut adalah link menuju GitHub program pada laporan ini :
https://github.com/alivovue/Tucil1_13524010.git

5.2. Tabel Penyelesaian

No	Poin	Ya	Tidak
1	Program berhasil di kompilasi tanpa kesalahan	✓	
2	Program berhasil dijalankan	✓	
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	✓	

5.3. Source Code

Berikut adalah lampiran source code secara lengkap :

1. Folder Controller
    - BoardUI.java

```
package com.mycompany.app.backend.controller;
import com.mycompany.app.backend.data.*;

import javafx.geometry.Pos;
```

```
import javafx.scene.layout.GridPane;
import javafx.scene.layout.StackPane;
import javafx.scene.image.ImageView;
import javafx.scene.image.Image;

public class BoardUI {
    private BoardObject boardObject;
    private ColorObject colorObject;
    private PaletteUI paletteUI;

    private final GridPane grid = new GridPane();
    private ImageView[][] queenIcons;
    private StackPane[][] colorCells;
    private boolean interactable;

    private static final String[] colorPalette = new String[] {
        "#1f77b4", "#ff7f0e", "#2ca02c", "#d62728", "#9467bd", "#8c564b",
        "#e377c2", "#7f7f7f", "#bcbd22", "#17becf", "#0066cc", "#ff9900",
        "#00cc66", "#cc0000", "#6600cc", "#996633", "#cc3399", "#5a5a5a",
        "#99cc00", "#00cccc", "#3366ff", "#ff3366", "#33ccff", "#ffcc00",
        "#66ff66", "#ff6666"
    };

    private final Image queenCrownImage = new
Image(getClass().getResourceAsStream("/queen_png.png"));

    public BoardUI(BoardObject boardObject, ColorObject colorObject, PaletteUI
paletteUI) {
        this.boardObject = boardObject;
        this.colorObject = colorObject;
        this.paletteUI = paletteUI;

        int n = colorObject.getLength();
        queenIcons = new ImageView[n][n];
        colorCells = new StackPane[n][n];

        this.interactable = false;

        buildBoardGridPane();
        refresh();
    }

    public GridPane getGrid() {
        return grid;
    }
```

```
    public void setInteractable(boolean set) {
        interactable = set;
    }

    public void buildBoardGridPane() {
        grid.getChildren().clear();
        int n = colorObject.getLength();
        double maxBoardPx = 520;
        double size = Math.floor(maxBoardPx / n);
        size = Math.max(14, Math.min(size, 40));

        queenIcons = new
ImageView[colorObject.getLength()][colorObject.getLength()];

        for (int i = 0 ; i < colorObject.getLength() ; i++) {
            for (int j = 0 ; j < colorObject.getLength() ; j++) {

                StackPane cell = new StackPane();
                cell.setMinSize(size, size);
                cell.setAlignment(Pos.CENTER);

                int code = colorObject.getColorTile(i, j);

                String background;
                if (code == 0) {
                    background = "#bdbdbd";
                }
                else {
                    background = colorPalette[(code - 1) %
colorPalette.length];
                }

                cell.setStyle(
                    "-fx-background-color: " + background + ";" +
                    "-fx-border-color: #333333;" +
                    "-fx-border-width: 0.6;"
                );

                ImageView icon = new ImageView(queenCrownImage);
                icon.setPreserveRatio(true);
                icon.setFitWidth(size * 0.75);
                icon.setFitHeight(size * 0.75);
                icon.setVisible(false);
```

```
                queenIcons[i][j] = icon;
                cell.getChildren().add(icon);

                if (interactable) {
                    final int row = i;
                    final int col = j;
                    cell.setOnMouseClicked(e -> {
                        if (paletteUI == null) return;
                        int selectedColor = paletteUI.getSelectedColor();
                        colorObject.setColor(row, col, selectedColor);
                        String bg;
                        if (selectedColor == 0) {
                            bg = "#bdbdbd";
                        }
                        else {
                            bg = colorPalette[(selectedColor - 1) %
colorPalette.length];
                        }
                        cell.setStyle(
                            "-fx-background-color: " + bg + ";" +
                            "-fx-border-color: #333333;" +
                            "-fx-border-width: 0.6;"
                        );
                    });
                }
                grid.add(cell, j, i);
            }
        }
    }

    public void refresh() {
        for (int i = 0 ; i < colorObject.getLength() ; i++) {
            for (int j = 0 ; j < colorObject.getLength() ; j++) {
                queenIcons[i][j].setVisible(boardObject.getTile(i, j) == 1);
            }
        }
    }
}
```

- ButtonsUI.java

```
package com.mycompany.app.backend.controller;

import java.io.File;
```

```
import com.mycompany.app.backend.data.*;

import javafx.geometry.Pos;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.Spinner;
import javafx.scene.control.SpinnerValueFactory;
import javafx.scene.control.Alert.AlertType;
import javafx.scene.image.WritableImage;
import javafx.scene.layout.HBox;
import javafx.stage.FileChooser;
import javafx.scene.SnapshotParameters;
import javafx.scene.control.Alert;
import javafx.stage.Window;

import javafx.embed.swing.SwingFXUtils;

import javax.imageio.ImageIO;
import java.awt.image.BufferedImage;

import javafx.application.Platform;
import javafx.concurrent.Task;

public class ButtonsUI {
    private BoardObject boardObject;
    private ColorObject colorObject;
    private BruteSolver bruteSolver;
    private SolutionObject solutionObject;
    private BoardUI boardUI;

    private ColorObject cacheColorObject;
    private Label iterationLabel;
    private Label timeLabel;
    private Label successLabel;

    public ButtonsUI(BoardObject boardObject, ColorObject colorObject,
SolutionObject solutionObject, BoardUI boardUI) {
        this.boardObject = boardObject;
        this.colorObject = colorObject;
        this.solutionObject = solutionObject;
        this.boardUI = boardUI;
        this.bruteSolver = new BruteSolver(boardObject, colorObject,
solutionObject);
        this.cacheColorObject = new ColorObject(colorObject.getLength());
```

```
    }

    public void setStatusLabel(Label iterationLabel, Label timeLabel, Label
successLabel) {
        this.iterationLabel = iterationLabel;
        this.timeLabel = timeLabel;
        this.successLabel = successLabel;
    }

    public Button calculateButton() {
        Button b = new Button("Calculate");
        b.setOnAction(e -> {
            if (!colorObject.isValidColor()) {
                errorMessage("Invalid Color", "Board is not fully colored or
more than 26 colors");
                return;
            }

            // check color number == n, ambil dari file reader
            boolean[] seen = new boolean[27];
            int jumlahWarna = 0;

            for (int i = 0 ; i < colorObject.getLength() ; i++) {
                for (int j = 0 ; j < colorObject.getLength() ; j++) {
                    if (!seen[colorObject.getColorTile(i, j)]) {
                        seen[colorObject.getColorTile(i, j)] = true;
                        jumlahWarna++;
                    }
                }
            }

            if (jumlahWarna != colorObject.getLength()) {
                errorMessage("Invalid Color", "Board size and color are not the
same amount");
                return;
            }


            solutionObject.resetSolutions();
            boardObject.resetBoard();
            boardUI.refresh();
            b.setDisable(true);

            Task<Void> task = new Task<>() {
                protected Void call() {
```

```
                    bruteSolver.exhaustiveBrute(it -> {
                        Platform.runLater(() -> {
                            boardUI.refresh();
                            if (iterationLabel != null)
iterationLabel.setText("Iteration : " + solutionObject.getIteration());
                            if (timeLabel != null) timeLabel.setText("Time : "
+ solutionObject.getTime() + " ms");
                            if (successLabel != null)
successLabel.setText("Success : " + (solutionObject.getSolutions().isEmpty() ?
"False" : "True"));
                        });
                    }, 5000);
                    Platform.runLater(() -> boardUI.refresh());
                    return null;
                }
            };

            task.setOnSucceeded(ev -> b.setDisable(false));
            task.setOnFailed(ev -> {
                b.setDisable(false);
                Throwable ex = task.getException();
                errorMessage("Brute force crashed", ex == null ? "Unknown
error" : ex.getMessage());
            });

            new Thread(task, "BruteThread").start();

        });
        return b;
    }

    public Button inputFromFile() {
        Button b = new Button("Input from file");
        b.setOnAction(e -> {
            Window window = b.getScene().getWindow();
            FileChooser fileChooser = new FileChooser();
            fileChooser.setTitle("Choose .txt file");
            fileChooser.getExtensionFilters().addAll(
                new FileChooser.ExtensionFilter("Text files (*.txt)", "*.txt")
            );
            File file = fileChooser.showOpenDialog(window);
            if (file == null) return;
            try {
                String path = file.getAbsolutePath();
                ColorObject temp = FileReader.loadColorBoard(path);
```

```
                this.colorObject.copyBoard(temp);
                this.boardObject.setSize(temp.getLength());
                boardUI.buildBoardGridPane();
            }
            catch (Exception ex) {
                errorMessage("File load error", ex.getMessage());
            }

        });
        return b;
    }

    public Button saveToFile() {
        Button b = new Button("Save to file");
        b.setOnAction(e -> {
            Window window = b.getScene().getWindow();

            FileChooser fileChooser = new FileChooser();
            fileChooser.setTitle("Save .txt file");
            fileChooser.getExtensionFilters().addAll(
                new FileChooser.ExtensionFilter("Text files (*.txt)", "*.txt")
            );
            fileChooser.setInitialFileName("hasil.txt");

            File file = fileChooser.showSaveDialog(window);
            if (file == null) return;

            try {
                String path = file.getAbsolutePath();
                if (!path.toLowerCase().endsWith(".txt")) {
                    path += ".txt";
                }

                int n = this.colorObject.getLength();

                try (java.io.BufferedWriter out =
                        java.nio.file.Files.newBufferedWriter(
                            java.nio.file.Path.of(path),
                            java.nio.charset.StandardCharsets.UTF_8
                        )) {

                    for (int i = 0; i < n; i++) {
                        StringBuilder row = new StringBuilder(n);
                        for (int j = 0; j < n; j++) {
```

```
                            int number = this.colorObject.getColorTile(i, j);

                            if (number < 1 || number > 26) {
                                throw new IllegalArgumentException("invalid
color number at (" + i + "," + j + "): " + number);
                            }

                            char c;
                            if (boardObject.getTile(i, j) == 1) {
                                c = '#';
                            }
                            else {
                                c = (char) ('A' + (number - 1));
                            }
                            row.append(c);
                        }
                        out.write(row.toString());
                        out.newLine();
                    }
                }

            } catch (Exception ex) {
                errorMessage("File save error", ex.getMessage());
            }
        });
        return b;
    }



    public Button resetBoardButton() {
        Button b = new Button("Reset board");
        b.setOnAction(e -> {
            boardObject.resetBoard();
            boardUI.refresh();
        });
        return b;
    }

    public Button resetColorButton() {
        Button b = new Button("Reset color");
        b.setOnAction(e -> {
            colorObject.resetColor();
            boardUI.buildBoardGridPane();
        });
```

```
        return b;
    }

    public Button downloadButton() {
        Button b = new Button("Download Result");
        b.setOnAction(e -> {
            if (solutionObject.getSolutions().isEmpty()) {
                errorMessage("No solution", "There is no solution yet, please
run the program first");
                return;
            }

            Window window = b.getScene().getWindow();
            FileChooser fileChooser = new FileChooser();
            fileChooser.setTitle("Save Image");
            fileChooser.getExtensionFilters().add(
                new FileChooser.ExtensionFilter("PNG Image (*.png)", "*.png")
            );
            fileChooser.setInitialFileName("board.png");

            File file = fileChooser.showSaveDialog(window);
            if (file == null) return;
            try {
                SnapshotParameters snapshotParameters = new
SnapshotParameters();
                WritableImage writableImage =
boardUI.getGrid().snapshot(snapshotParameters, null);
                BufferedImage bufferedImage =
SwingFXUtils.fromFXImage(writableImage, null);
                File result = file.getName().toLowerCase().endsWith(".png") ?
file : new File(file.getAbsolutePath() + ".png");
                ImageIO.write(bufferedImage, "png", result);
            }
            catch (Exception ex) {
                errorMessage("Save failed", ex.getMessage());
            }
        });
        return b;

    }

    public HBox boardSizeControl(int initialSize, int minSize, int maxSize) {
        Label label = new Label("Board size:");

        Spinner<Integer> spinner = new Spinner<>();
```

```
        spinner.setValueFactory(
            new SpinnerValueFactory.IntegerSpinnerValueFactory(minSize,
maxSize, initialSize)
        );
        spinner.setEditable(true);
        spinner.setPrefWidth(90);

        Button apply = new Button("Apply Size");
        apply.setOnAction(e -> {
            int n = spinner.getValue();
            boardObject.setSize(n);
            colorObject.setSize(n);
            boardUI.buildBoardGridPane();
        });

        HBox box = new HBox(8, label, spinner, apply);
        box.setAlignment(Pos.CENTER);
        return box;
    }

    private void errorMessage(String title, String message) {
        Alert alert = new Alert(AlertType.ERROR);
        alert.setTitle(title);
        alert.setHeaderText(null);
        alert.setContentText(message);
        alert.showAndWait();
    }

    // for edit UI

    public Button applyButton() {
        Button b = new Button("Apply Changes");
        b.setOnAction(e -> {
            System.out.println("changes applied");
        });
        return b;
    }

    public Button cancelButton() {
        Button b = new Button("Cancel Changes");
        b.setOnAction(e -> {
            System.out.println("changes canceled");
        });
        return b;
    }
```

```
    public Button editButton() {
        Button b = new Button("Edit Board");
        b.setOnAction(e -> {
            System.out.println("entering edit page");
        });
        return b;
    }
}
```

- PaletteUI.java

```
package com.mycompany.app.backend.controller;

import java.util.ArrayList;
import java.util.List;

import javafx.geometry.Pos;
import javafx.scene.control.Button;
import javafx.scene.layout.GridPane;

public class PaletteUI {
    private int selectedColor;
    private final List<Button> paletteButtons = new ArrayList<>(26);

    public PaletteUI() {
        this.selectedColor = 1;
    }

    public int getSelectedColor() {
        return selectedColor;
    }

    public GridPane colorPaletteBar() {
        GridPane grid = new GridPane();
        grid.setHgap(8);
        grid.setVgap(8);
        grid.setAlignment(Pos.TOP_LEFT);

        paletteButtons.clear();

        int columns = 6;

        for (int i = 1; i <= 26; i++) {
```

```
            String colorHex = getPaletteHex(i);
            Button b = createPaletteButton(colorHex, i);
            paletteButtons.add(b);

            int index = i - 1;
            int col = index % columns;
            int row = index / columns;

            grid.add(b, col, row);
        }

        updateHighlight();
        return grid;
    }


    private String getPaletteHex(int code) {
        String[] palette = new String[] {
            "#1f77b4", "#ff7f0e", "#2ca02c", "#d62728", "#9467bd", "#8c564b",
            "#e377c2", "#7f7f7f", "#bcbd22", "#17becf", "#0066cc", "#ff9900",
            "#00cc66", "#cc0000", "#6600cc", "#996633", "#cc3399", "#5a5a5a",
            "#99cc00", "#00cccc", "#3366ff", "#ff3366", "#33ccff", "#ffcc00",
            "#66ff66", "#ff6666"
        };
        return palette[(code - 1) % palette.length];
    }

    private Button createPaletteButton(String colorHex, int code) {
        Button b = new Button();
        b.setMinSize(32, 32);
        b.setPrefSize(32, 32);
        b.setMaxSize(32, 32);
        b.setStyle(baseStyle(colorHex, false));

        b.setOnAction(e -> {
            selectedColor = code;
            updateHighlight();
        });

        return b;
    }

    private void updateHighlight() {
        for (int i = 0; i < paletteButtons.size(); i++) {
            int code = i + 1;
```

```
            String hex = getPaletteHex(code);
            boolean selected = (code == selectedColor);
            paletteButtons.get(i).setStyle(baseStyle(hex, selected));
        }
    }

    private String baseStyle(String colorHex, boolean selected) {
        if (selected) {
            return ""
                + "-fx-background-color: " + colorHex + ";"
                + "-fx-border-color: black;"
                + "-fx-border-width: 3;"
                + "-fx-background-radius: 6;"
                + "-fx-border-radius: 6;"
                + "-fx-padding: 0;"
                + "-fx-effect: dropshadow(gaussian, rgba(0,0,0,0.35), 8, 0.25,
0, 0);";
        }

        return ""
            + "-fx-background-color: " + colorHex + ";"
            + "-fx-border-color: rgba(0,0,0,0.45);"
            + "-fx-border-width: 1;"
            + "-fx-background-radius: 6;"
            + "-fx-border-radius: 6;"
            + "-fx-padding: 0;";
    }
}
```

2. Folder Data
    - BoardObject.java

```
package com.mycompany.app.backend.data;

// board = 0 no queen
// board = 1 queen

public class BoardObject {
    private int[][] board;

    public BoardObject(int size) {
        board = new int[size][size];
        resetBoard();
    }
```

```java
    // getter functions

    public int[][] getBoard() {
        return board;
    }

    public int getTile(int x, int y) {
        if (x >= 0 && x < board.length && y >= 0 && y < board[0].length) {
            return board[x][y];
        }
        else {
            return -1;
        }
    }

    public int getBoardLength() {
        return board.length;
    }

    // setter functions

    public void setQueen(int x, int y) {
        if (x >= 0 && x < board.length && y >= 0 && y < board[0].length) {
            board[x][y] = 1;
        }
        else {
            throw new IllegalArgumentException("Invalid position");
        }
    }

    public void deleteQueen(int x, int y) {
        if (x >= 0 && x < board.length && y >= 0 && y < board[0].length) {
            board[x][y] = 0;
        }
        else {
            throw new IllegalArgumentException("Invalid position");
        }
    }

    public void resetBoard() {
        for (int i = 0 ; i < board.length ; i++) {
            for (int j = 0 ; j < board[i].length ; j++) {
                board[i][j] = 0;
            }
```

```
        }
    }

    public void setSize(int n) {
        this.board = new int[n][n];
    }


}
```

- BruteSolver.java

```
package com.mycompany.app.backend.data;

import java.util.function.LongConsumer;

public class BruteSolver {
    private BoardObject boardObject;
    private ColorObject colorObject;
    private SolutionObject solutionObject;
    private GameLogic gameLogic;

    public BruteSolver(BoardObject boardObject, ColorObject colorObject,
SolutionObject solutionObject) {
        this.boardObject = boardObject;
        this.colorObject = colorObject;
        this.solutionObject = solutionObject;
        this.gameLogic = new GameLogic(boardObject, colorObject);
    }

    public void exhaustiveBrute(LongConsumer tick, long period) {
        long startTimer = System.nanoTime();
        int[] queen = new int[boardObject.getBoardLength()];
        for (int i = 0 ; i < boardObject.getBoardLength() ; i++) {
            queen[i] = i + 1;
        }

        int tempIter = 0;
        while (true) {
            applyToBoard(queen);

            tempIter++;
            solutionObject.incrementIteration();
```

```
            if (tick != null && period > 0 && (tempIter % period == 0)) {
                tick.accept(tempIter);
            }

            if (gameLogic.isValidBoard()) {
                solutionObject.addSolution(queen.clone());
                break;
            }
            if (!nextCombination(queen, boardObject.getBoardLength())) break;
        }
        long endTimer = System.nanoTime();
        int timeTaken = (int) ((endTimer - startTimer) / 1000000);
        solutionObject.setTime(timeTaken);

        if (tick != null) tick.accept(tempIter);
    }

    public boolean nextCombination(int[] queen, int n) {
        int max = n*n;
        for (int i = queen.length - 1 ; i >= 0 ; i--) {
            int limit = max - (queen.length - 1 - i);

            if (queen[i] < limit) {
                queen[i]++;
                for (int j = i + 1 ; j < queen.length ; j++) {
                    queen[j] = queen[j-1] + 1;
                }
                return true;
            }
        }
        return false;
    }

    public void applyToBoard(int[] queen) {
        boardObject.resetBoard();
        for (int i = 0 ; i < queen.length ; i++) {
            int row = (queen[i] - 1)/boardObject.getBoardLength();
            int col = (queen[i] - 1)%boardObject.getBoardLength();
            boardObject.setQueen(row, col);

        }
    }

}
```

- ColorObject.java

```java
package com.mycompany.app.backend.data;

// 0 = default color

public class ColorObject {
    private int[][] color;

    public ColorObject(int size) {
        color = new int[size][size];
        resetColor();
    }

    // getter

    public int[][] getColor() {
        return color;
    }

    public int getColorTile(int x, int y) {
        if (x < color.length && x >= 0 && y < color[0].length && y >=0) {
            return color[x][y];
        }
        else {
            return -1;
        }
    }

    public int getLength() {
        return color.length;
    }

    // setter

    public void setColor(int x, int y, int colorNumber) {
        if (x < color.length && x >= 0 && y < color[0].length && y >= 0) {
            color[x][y] = colorNumber;
        }
        else {
            throw new IllegalArgumentException("invalid position");
        }
    }
```

```java
public void resetColor() {
    for (int i = 0 ; i < color.length ; i++) {
        for (int j = 0 ; j < color[i].length ; j++) {
            color[i][j] = 0;
        }
    }
}

public void copyBoard(ColorObject colorObject) {
    int length = colorObject.getLength();
    this.color = new int[length][length];
    for (int i = 0 ; i < color.length ; i++) {
        for (int j = 0 ; j < color.length ; j++) {
            this.color[i][j] = colorObject.getColorTile(i, j);
        }
    }
}

public ColorObject copyColorObject() {
    int n = getLength();
    ColorObject temp = new ColorObject(n);
    for (int i = 0 ; i < n ; i++) {
        for (int j = 0 ; j < n ; j++) {
            temp.setColor(i, j, this.color[i][j]);
        }
    }
    return temp;
}

public void setSize(int size) {
    this.color = new int[size][size];
}

// checking
public boolean isValidColor() {
    for (int i = 0 ; i < color.length ; i++) {
        for (int j = 0 ; j < color.length ; j++) {
            if (color[i][j] == 0 || color[i][j] > 26) {
                return false;
            }
        }
    }
    return true;
}
```

```
}
```

- FileReader.java

```
package com.mycompany.app.backend.data;

import java.nio.file.Files;
import java.nio.file.Path;
import java.util.List;

public class FileReader {
    public static ColorObject loadColorBoard(String filepath) throws Exception
{
        List<String> line = Files.readAllLines(Path.of(filepath));
        line.removeIf(s -> s.trim().isEmpty());

        if (line.size() == 0) throw new IllegalArgumentException("empty file");

        int n = line.size();

        for (int i = 0 ; i < line.size() ; i++) {
            if (line.get(i).length() != n) {
                throw new IllegalArgumentException("not square");
            }
        }

        boolean[] seen = new boolean[27];
        int jumlahWarna = 0;

        ColorObject color = new ColorObject(n);
        for (int i = 0 ; i < n ; i++) {
            String row = line.get(i);
            for (int j = 0 ; j < n ; j++) {
                char c = Character.toUpperCase(row.charAt(j));
                if (c < 'A' || c > 'Z') {
                    throw new IllegalArgumentException("not valid char");
                }

                int number = c - 'A' + 1;

                if (!seen[number]) {
                    seen[number] = true;
                    jumlahWarna++;
                }
```

```
                color.setColor(i, j, number);
            }
        }

        if (jumlahWarna != n) {
            throw new IllegalArgumentException("jumlah warna != n");
        }

        return color;
    }
}
```

- GameLogic.java

```
package com.mycompany.app.backend.data;

public class GameLogic {
    private BoardObject boardObject;
    private ColorObject colorObject;

    public GameLogic(BoardObject boardObject, ColorObject colorObject) {
        this.boardObject = boardObject;
        this.colorObject = colorObject;
    }

    public boolean checkOccupied(int x, int y) {
        if (boardObject.getTile(x, y) == 1) {
            return false;
        }
        return true;
    }

    public boolean checkAdjacent(int x, int y) {
        for (int i = x-1 ; i <= x+1 ; i++) {
            for (int j = y-1 ; j <= y+1 ; j++) {
                if (boardObject.getTile(i, j) == -1) {
                    continue;
                }
                if (boardObject.getTile(i, j) == 1) {
                    return false;
                }
            }
        }
```

```
        return true;
    }

    public boolean checkLine(int x, int y) {
        // check horizontal line first
        for (int i = 0 ; i < boardObject.getBoardLength() ; i++) {
            if (boardObject.getTile(x, i) == 1) {
                return false;
            }
        }

        // check vertical
        for (int i = 0 ; i < boardObject.getBoardLength() ; i++) {
            if (boardObject.getTile(i, y) == 1) {
                return false;
            }
        }

        return true;
    }

    public boolean checkColor(int x, int y) {
        int colorCode = colorObject.getColorTile(x, y);
        for (int i = 0 ; i < colorObject.getLength() ; i++) {
            for (int j = 0 ; j < colorObject.getLength() ; j++) {
                if (colorObject.getColorTile(i, j) == colorCode &&
boardObject.getTile(i, j) == 1) {
                    return false;
                }
            }
        }

        return true;
    }

    public boolean isValidQueen(int x, int y) {
        if (!checkOccupied(x, y)) return false;
        if (!checkAdjacent(x, y)) return false;
        if (!checkColor(x, y)) return false;
        if (!checkLine(x, y)) return false;
        return true;
    }

    public boolean isValidBoard() {
        for (int i = 0 ; i < boardObject.getBoardLength() ; i++) {
```

```
            for (int j = 0 ; j < boardObject.getBoardLength() ; j++) {
                if (boardObject.getTile(i, j) == 1) {
                    boardObject.deleteQueen(i, j);
                    if (!isValidQueen(i, j)) {
                        boardObject.setQueen(i, j);
                        return false;
                    }
                    boardObject.setQueen(i, j);
                }
            }
        }
        return true;
    }
}
```

- SolutionObject.java

```
package com.mycompany.app.backend.data;

import java.util.ArrayList;

// solutions is array containing the positions of queens in which columns in
each rows
// so, solution [1,2,3] means having a queen at row 1 column 1, row 2 column 2,
row 3 column 3

public class SolutionObject {
    private ArrayList<int[]> solutions;
    private int time;
    private long iteration;

    public SolutionObject() {
        solutions = new ArrayList<>();
        time = 0;
        iteration = 0;
    }

    // setter
    public void addSolution(int[] solution) {
        solutions.add(solution.clone());
    }

    public void setTime(int t) {
        time = t;
```

```
    }

    public void setIteration(int it) {
        iteration = it;
    }

    public void incrementIteration() {
        iteration++;
    }


    // getter
    public ArrayList<int[]> getSolutions() {
        return solutions;
    }

    public int getTime() {
        return time;
    }

    public long getIteration() {
        return iteration;
    }

    public void resetSolutions() {
        solutions.clear();
        time = 0;
        iteration = 0;
    }
}
```

3. Folder Frontend
    - EditMenu.java

```
package com.mycompany.app.frontend;

import com.mycompany.app.backend.controller.BoardUI;
import com.mycompany.app.backend.controller.ButtonsUI;
import com.mycompany.app.backend.controller.PaletteUI;
import com.mycompany.app.backend.data.BoardObject;
import com.mycompany.app.backend.data.ColorObject;
import com.mycompany.app.backend.data.SolutionObject;

import javafx.geometry.Insets;
```

```
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.HBox;
import javafx.scene.layout.StackPane;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

public class EditMenu {

    public Scene createScene(Stage stage, Scene mainScene, BoardObject board,
ColorObject originalColor, SolutionObject sols, BoardUI mainBoardUI) {
        ColorObject newColor = originalColor.copyColorObject();
        ColorObject cacheColor = newColor.copyColorObject();

        PaletteUI paletteUI = new PaletteUI();
        BoardUI editBoardUI = new BoardUI(board, newColor, paletteUI);
        editBoardUI.setInteractable(true);
        editBoardUI.buildBoardGridPane();
        editBoardUI.refresh();

        ButtonsUI buttonsUI = new ButtonsUI(board, newColor, sols,
editBoardUI);

        Label title = new Label("Edit Board (" + newColor.getLength() + "x" +
newColor.getLength() + ")");
        title.setStyle("-fx-font-size: 20px; -fx-font-weight: 900;");

        StackPane boardWrap = new StackPane(editBoardUI.getGrid());
        boardWrap.setAlignment(Pos.CENTER);
        boardWrap.setPadding(new Insets(14));
        boardWrap.setStyle(
            "-fx-background-color: white;" +
            "-fx-border-color: #d9d9d9;" +
            "-fx-border-radius: 12;" +
            "-fx-background-radius: 12;"
        );

        VBox center = new VBox(12, title, boardWrap);
        center.setAlignment(Pos.TOP_CENTER);
        center.setPadding(new Insets(6));
```

```
        Label settingsTitle = new Label("Board Settings");
        settingsTitle.setStyle("-fx-font-size: 16px; -fx-font-weight: 900;");

        HBox sizeControl = buttonsUI.boardSizeControl(newColor.getLength(), 1,
26);
        Button clearButton = buttonsUI.resetColorButton();

        VBox settingsBox = new VBox(10, settingsTitle, sizeControl,
clearButton);
        settingsBox.setAlignment(Pos.TOP_LEFT);
        settingsBox.setPadding(new Insets(12));
        settingsBox.setStyle(cardStyle());

        Label paletteTitle = new Label("Palette");
        paletteTitle.setStyle("-fx-font-size: 16px; -fx-font-weight: 900;");

        Label paletteHint = new Label("Pick a color, then click tiles to
paint.");
        paletteHint.setStyle("-fx-font-size: 12px; -fx-text-fill:
rgba(0,0,0,0.7);");

        GridPane paletteGrid = paletteUI.colorPaletteBar();

        VBox paletteBox = new VBox(10, paletteTitle, paletteHint, paletteGrid);
        paletteBox.setAlignment(Pos.TOP_LEFT);
        paletteBox.setPadding(new Insets(12));
        paletteBox.setStyle(cardStyle());

        Button applyButton = buttonsUI.applyButton();
        applyButton.setText("Apply Changes");
        applyButton.setStyle(
            "-fx-font-weight: 900;" +
            "-fx-padding: 10 16;" +
            "-fx-background-radius: 10;" +
            "-fx-border-radius: 10;" +
            "-fx-background-color: #2d6cdf;" +
            "-fx-text-fill: white;"
        );

        applyButton.setOnAction(e -> {
            originalColor.copyBoard(newColor);
            mainBoardUI.buildBoardGridPane();
            mainBoardUI.refresh();
            stage.setScene(mainScene);
        });
```

```
        Button cancelButton = buttonsUI.cancelButton();
        cancelButton.setText("Cancel");
        cancelButton.setStyle(
            "-fx-font-weight: 800;" +
            "-fx-padding: 10 16;" +
            "-fx-background-radius: 10;" +
            "-fx-border-radius: 10;"
        );

        cancelButton.setOnAction(e -> {
            newColor.copyBoard(cacheColor);
            editBoardUI.buildBoardGridPane();
            editBoardUI.refresh();
            stage.setScene(mainScene);
        });

        HBox actionRow = new HBox(10, cancelButton, applyButton);
        actionRow.setAlignment(Pos.CENTER_RIGHT);

        VBox sidebar = new VBox(12, settingsBox, paletteBox, actionRow);
        sidebar.setAlignment(Pos.TOP_LEFT);
        sidebar.setPadding(new Insets(6, 0, 0, 12));
        sidebar.setPrefWidth(320);


        BorderPane root = new BorderPane();
        root.setPadding(new Insets(16));
        root.setCenter(center);
        root.setRight(sidebar);

        return new Scene(root, 900, 700);
    }

    private String cardStyle() {
        return ""
            + "-fx-background-color: white;"
            + "-fx-border-color: #d9d9d9;"
            + "-fx-border-radius: 12;"
            + "-fx-background-radius: 12;";
    }
}
```

- MainMenu.java

```java
package com.mycompany.app.frontend;

import com.mycompany.app.backend.controller.BoardUI;
import com.mycompany.app.backend.controller.ButtonsUI;
import com.mycompany.app.backend.data.BoardObject;
import com.mycompany.app.backend.data.ColorObject;
import com.mycompany.app.backend.data.SolutionObject;

import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.HBox;
import javafx.scene.layout.StackPane;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

public class MainMenu {

    public Scene createScene(Stage stage) {
        int n = 8;

        BoardObject board = new BoardObject(n);
        ColorObject color = new ColorObject(n);
        SolutionObject sols = new SolutionObject();

        BoardUI boardUI = new BoardUI(board, color, null);
        ButtonsUI buttonsUI = new ButtonsUI(board, color, sols, boardUI);

        Label iterationLabel = new Label("Iteration : 0");
        Label timeLabel = new Label("Time : 0 ms");
        Label successLabel = new Label("Success : False");

        String statusStyle = "-fx-font-size: 16px; -fx-font-weight: 600;";
        iterationLabel.setStyle(statusStyle);
        timeLabel.setStyle(statusStyle);
        successLabel.setStyle(statusStyle);

        buttonsUI.setStatusLabel(iterationLabel, timeLabel, successLabel);

        BorderPane root = new BorderPane();
        root.setPadding(new Insets(16));
```

```
        Scene mainScene = new Scene(root, 900, 700);

        Button calculateButton = buttonsUI.calculateButton();
        Button inputFileButton = buttonsUI.inputFromFile();
        Button editButton = buttonsUI.editButton();
        Button resetBoardButton = buttonsUI.resetBoardButton();
        Button downloadButton = buttonsUI.downloadButton();
        Button saveToFileButton = buttonsUI.saveToFile();

        editButton.setOnAction(e -> {
            EditMenu editMenu = new EditMenu();
            Scene editScene = editMenu.createScene(stage, mainScene, board,
color, sols, boardUI);
            stage.setScene(editScene);
        });

        calculateButton.setStyle(
            "-fx-font-weight: 900;" +
            "-fx-padding: 10 18;" +
            "-fx-background-radius: 8;" +
            "-fx-border-radius: 8;" +
            "-fx-background-color: #2d6cdf;" +
            "-fx-text-fill: white;"
        );

        String secondaryStyle =
            "-fx-font-weight: 700;" +
            "-fx-padding: 10 16;" +
            "-fx-background-radius: 8;" +
            "-fx-border-radius: 8;";

        inputFileButton.setStyle(secondaryStyle);
        editButton.setStyle(secondaryStyle);

        resetBoardButton.setPrefWidth(150);
        downloadButton.setPrefWidth(150);
        saveToFileButton.setPrefWidth(150);

        HBox topBar = new HBox(12, calculateButton, inputFileButton,
editButton);
        topBar.setAlignment(Pos.CENTER_LEFT);
        topBar.setPadding(new Insets(10));
        root.setTop(topBar);

        Label boardTitle = new Label("Board");
```

```
        boardTitle.setStyle("-fx-font-size: 20px; -fx-font-weight: 900;");

        StackPane boardWrap = new StackPane(boardUI.getGrid());
        boardWrap.setAlignment(Pos.CENTER);
        boardWrap.setPadding(new Insets(12));
        boardWrap.setStyle(
            "-fx-background-color: white;" +
            "-fx-border-color: #d9d9d9;" +
            "-fx-border-radius: 12;" +
            "-fx-background-radius: 12;"
        );

        boardUI.getGrid().setHgap(0);
        boardUI.getGrid().setVgap(0);

        VBox centerCard = new VBox(12, boardTitle, boardWrap);
        centerCard.setAlignment(Pos.TOP_CENTER);
        centerCard.setPadding(new Insets(6));

        root.setCenter(centerCard);

        Label statusTitle = new Label("Status");
        statusTitle.setStyle("-fx-font-size: 18px; -fx-font-weight: 900;");

        Label actionsTitle = new Label("Actions");
        actionsTitle.setStyle("-fx-font-size: 16px; -fx-font-weight: 900;");

        VBox actionsBox = new VBox(10, resetBoardButton, downloadButton,
saveToFileButton);
        actionsBox.setAlignment(Pos.TOP_LEFT);

        VBox rightPanel = new VBox(
            12,
            statusTitle,
            iterationLabel,
            timeLabel,
            successLabel,
            actionsTitle,
            actionsBox
        );
        rightPanel.setAlignment(Pos.TOP_LEFT);
        rightPanel.setPadding(new Insets(12));
        rightPanel.setPrefWidth(210);
        rightPanel.setMaxHeight(320);
```

```
        rightPanel.setStyle(
            "-fx-background-color: white;" +
            "-fx-border-color: #d9d9d9;" +
            "-fx-border-radius: 12;" +
            "-fx-background-radius: 12;"
        );

        BorderPane.setAlignment(rightPanel, Pos.TOP_RIGHT);
        BorderPane.setMargin(rightPanel, new Insets(6, 0, 0, 12));

        root.setRight(rightPanel);

        return mainScene;
    }
}
```

4. Root :
    - App.java

```
package com.mycompany.app;

import com.mycompany.app.frontend.MainMenu;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.stage.Stage;

public class App extends Application {

    @Override
    public void start(Stage stage) {
        MainMenu menu = new MainMenu();
        Scene scene = menu.createScene(stage);
        stage.setTitle("Queens Solver");
        stage.setScene(scene);
        stage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

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

Audric Yusuf Maynard Simatupang