Tugas Kecil 1 IF2211 Strategi Algoritma

Penyelesaian IQ Puzzler Pro dengan Algoritma Brute Force

Oleh

Bryan P. Hutagalung (18222130)

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika – Institut Teknologi Bandung

Jl. Ganesha 10, Bandung 40132

2025

Daftar Isi

Daftar Gambar

Bab I

Pendahuluan

1.1 Deskripsi Tugas

Gambar 1.1.1 Permainan Game IQ Puzzler Pro

IQ Puzzler Pro adalah permainan papan yang diproduksi oleh perusahaan Smart Games. Tujuan dari permainan ini adalah pemain harus dapat mengisi seluruh papan dengan piece (blok puzzle) yang telah tersedia.

Komponen penting dari permainan IQ Puzzler Pro terdiri dari:

1. Board (Papan) – Board merupakan komponen utama yang menjadi tujuan permainan dimana pemain harus mampu mengisi seluruh area papan menggunakan blok-blok yang telah disediakan.
2. Blok/Piece – Blok adalah komponen yang digunakan pemain untuk mengisi papan kosong hingga terisi penuh. Setiap blok memiliki bentuk yang unik dan semua blok harus digunakan untuk menyelesaikan puzzle.

Tugasnya adalah menemukan cukup satu solusi dari permainan IQ Puzzler Pro dengan menggunakan algoritma Brute Force, atau menampilkan bahwa solusi tidak ditemukan jika tidak ada solusi yang mungkin dari puzzle.

1.2 Ilustrasi Kasus

Diberikan sebuah wadah berukuran 11 x 5 serta 12 buah blok puzzle dan beberapa blok telah ditempatkan dengan bentuk sebagai berikut.

Gambar 1.2.1 Awal Permainan Game IQ Puzzler Pro

Pemain berusaha untuk mengisi bagian papan yang kosong dengan menggunakan blok yang tersedia.

PERHATIAN: Untuk tucil ini permainan diawali dengan papan kosong

Permainan dinyatakan selesai jika pemain mampu mengisi seluruh papan dengan blok (dalam tugas kecil ini ada kemungkinan pemain tidak dapat mengisi seluruh papan).

Agar lebih jelas, amati video cara bermain berikut: https://youtube.com/shorts/MWiPAS3wfGM?feature=shared.

BAB II
Hasil dan Pembahasan

2.1 Repository Program

https://github.com/nathangalung/Tucil1_18222130.git

2.2 Reader

Kelas Reader dalam kode di atas bertanggung jawab untuk membaca dan memproses file konfigurasi yang berisi informasi puzzle. Kelas ini sangat penting dalam konteks sistem permainan puzzle karena ia menginterpretasikan file input yang memuat data penting, seperti dimensi puzzle, jenis puzzle, konfigurasi papan, dan definisi blok puzzle. Kelas ini dirancang untuk mematuhi prinsip Single Responsibility Principle, yaitu fokus hanya pada pemrosesan dan pembacaan file puzzle, sehingga memudahkan pemeliharaan dan pengembangan kode lebih lanjut.

Atribut kelas:

1. **Filename**
   Menyimpan path atau lokasi file yang akan diproses.
2. **puzzleType**
   Menyimpan tipe puzzle yang dapat berupa "DEFAULT" atau "CUSTOM".
3. **rows dan cols**
   Menyimpan dimensi dari papan permainan, yaitu jumlah baris dan kolom yang ada dalam puzzle.
4. **numBlocks**
   Menyimpan jumlah blok yang harus ditempatkan dalam puzzle.
5. **customConfig**
   Merupakan array dua dimensi (2D array) yang digunakan untuk menyimpan konfigurasi khusus jika tipe puzzle adalah "CUSTOM".

Konstruktor kelas Reader(String filename) menerima satu parameter, yaitu path ke file input puzzle. Konstruktor ini menginisialisasi filename, memungkinkan objek Reader untuk membuka dan membaca file yang telah ditentukan.

Metode:

1. **readDimensions()**

   Metode ini bertugas untuk membaca baris pertama file, yang mengandung dimensi puzzle, yaitu jumlah baris, kolom, dan blok. Prosesnya dimulai dengan membuka file menggunakan Scanner, kemudian membaca dan memisahkan baris pertama file menjadi tiga komponen yang masing-masing di-convert menjadi tipe data integer (untuk rows, cols, dan numBlocks). Tipe puzzle kemudian dibaca pada baris kedua. Akhirnya, metode ini mengembalikan array yang berisi tiga nilai integer tersebut.

2. **readPuzzleType()**

   Metode ini memastikan bahwa nilai puzzleType hanya dibaca satu kali. Jika puzzleType belum diinisialisasi, metode ini membuka file, melewati dua baris pertama (yang berisi dimensi dan tipe puzzle), lalu membaca tipe puzzle dan menyimpannya ke dalam atribut puzzleType. Nilai tipe puzzle ini kemudian dikembalikan.

3. **readCustomConfig()**

   Metode ini digunakan khusus untuk puzzle dengan tipe "CUSTOM". Metode ini pertama-tama memeriksa apakah puzzleType adalah "CUSTOM". Jika ya, maka metode ini akan membuka file dan membaca konfigurasi papan yang berupa karakter (misalnya, matriks berupa X, O, atau spasi). File dibaca baris per baris dan setiap baris diubah menjadi array karakter, yang kemudian disimpan dalam customConfig. Konfigurasi ini akhirnya dikembalikan dalam bentuk array 2D jika puzzle adalah "CUSTOM", atau null jika tipe puzzle lain.

4. **readBlocks()**

   Metode ini mengimplementasikan algoritma untuk membaca blok-blok dalam puzzle. Proses dimulai dengan membuka file dan melewati baris-baris yang berisi dimensi dan tipe puzzle. Jika puzzle berjenis "CUSTOM", maka baris yang berisi konfigurasi papan akan dilewati. Selanjutnya, metode ini

akan membaca baris-baris sisa yang berisi definisi blok. StringBuilder digunakan untuk menyatukan baris yang membentuk satu blok. Ketika ditemukan perubahan ID blok, blok yang telah terkumpul ditambahkan ke dalam daftar ArrayList<Block>. Setiap blok yang berhasil dibaca kemudian dibungkus dalam objek Block dan dimasukkan ke dalam array list. Blok terakhir juga ditambahkan jika masih ada data yang tersisa dalam StringBuilder.

5. **findBlockId(String line)**

   Metode ini bertugas untuk mengekstrak ID blok dari setiap baris. ID blok ditentukan oleh karakter pertama yang bukan spasi dalam baris. Jika tidak ada karakter selain spasi, maka ID blok akan dianggap sebagai spasi (' '). Metode ini mengembalikan karakter ID blok yang ditemukan.

Kode ini juga menggunakan mekanisme exception handling untuk menangani kemungkinan kegagalan saat mengakses file. Misalnya, jika file tidak ditemukan, akan dilemparkan FileNotFoundException. Penutupan Scanner dilakukan secara konsisten setelah pembacaan file selesai, menggunakan pola penutupan yang tepat untuk menghindari kebocoran sumber daya.

Secara keseluruhan, kelas Reader bertindak sebagai komponen yang bertanggung jawab untuk memuat dan mengonversi data dari file konfigurasi puzzle menjadi struktur data yang dapat diproses oleh aplikasi. Metode-metode yang ada memastikan bahwa data puzzle dibaca dengan benar, baik itu dimensi, tipe, konfigurasi khusus, maupun definisi blok-blok yang membentuk puzzle tersebut. Kelas ini mempermudah aplikasi dalam mengambil data dari file eksternal dan mengolahnya lebih lanjut untuk permainan atau simulasi puzzle.

```
package file;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.Scanner;
```

```java
import puzzle.Block;

public class Reader {
    private String filename;
    private String puzzleType;
    private int rows, cols, numBlocks;
    private char[][] customConfig;

    public Reader(String filename) {
        this.filename = filename;
    }

    public int[] readDimensions() throws FileNotFoundException {
        File file = new File(filename);
        Scanner scanner = new Scanner(file);
        String[] dimensions = scanner.nextLine().split(" ");
        rows = Integer.parseInt(dimensions[0]);
        cols = Integer.parseInt(dimensions[1]);
        numBlocks = Integer.parseInt(dimensions[2]);
        puzzleType = scanner.nextLine();
        scanner.close();
        return new int[]{rows, cols, numBlocks};
    }

    public String readPuzzleType() throws FileNotFoundException {
        if (puzzleType == null) {
            File file = new File(filename);
            Scanner scanner = new Scanner(file);
            scanner.nextLine();
            puzzleType = scanner.nextLine();
            scanner.close();
        }
        return puzzleType;
    }

    public char[][] readCustomConfig() throws FileNotFoundException {
        if (!puzzleType.equals("CUSTOM")) {
            return null;
        }

        File file = new File(filename);
```

```
        Scanner scanner = new Scanner(file);
        scanner.nextLine();
        scanner.nextLine();

        char[][] config = new char[rows][cols];
        for (int i = 0; i < rows; i++) {
            String line = scanner.nextLine();
            config[i] = line.toCharArray();
        }

        scanner.close();
        this.customConfig = config;
        return config;
    }

    public ArrayList<Block> readBlocks() throws FileNotFoundException {
        File file = new File(filename);
        Scanner scanner = new Scanner(file);
        scanner.nextLine();
        scanner.nextLine();

        if (puzzleType.equals("CUSTOM")) {
            for (int i = 0; i < rows; i++) {
                scanner.nextLine();
            }
        }

        ArrayList<Block> blocks = new ArrayList<>();
        StringBuilder currentBlock = new StringBuilder();
        String line;
        Character currentId = null;

        while (scanner.hasNextLine()) {
            line = scanner.nextLine();
            if (!line.isEmpty()) {
                char blockId = findBlockId(line);
                if (currentId == null) {
                    currentId = blockId;
                    currentBlock.append(line).append("\n");
                } else if (blockId == currentId) {
                    currentBlock.append(line).append("\n");
```

```
                } else {
                    blocks.add(new Block(currentBlock.toString()));
                    currentBlock = new StringBuilder();
                    currentBlock.append(line).append("\n");
                    currentId = blockId;
                }
            }
        }

        if (currentBlock.length() > 0) {
            blocks.add(new Block(currentBlock.toString()));
        }

        scanner.close();
        return blocks;
    }

    private char findBlockId(String line) {
        for (char c : line.toCharArray()) {
            if (c != ' ') {
                return c;
            }
        }
        return ' ';
    }
}
```

2.3 Writer

Kelas Writer adalah komponen penting dalam sistem penyelesaian puzzle yang bertugas untuk mengelola output solusi puzzle, baik itu dalam bentuk file teks maupun tampilan visual di console. Kelas ini mengimplementasikan dua fungsi utama: menulis solusi puzzle ke dalam file teks dan menampilkan solusi yang sudah diselesaikan di console dengan pewarnaan yang memudahkan pembacaan solusi.

Atribut outputPath digunakan untuk menyimpan path file output yang akan menyimpan solusi puzzle. Konstruktornya menerima parameter inputPath yang berfungsi sebagai path file input, kemudian mengubahnya menjadi path output dengan mengganti bagian direktori "input" menjadi "output/txt". Hal ini

memungkinkan pengorganisasian file output yang terstruktur dengan baik dan mudah diakses.

Salah satu fitur unik dari kelas Writer adalah penggunaan sistem pewarnaan menggunakan kode ANSI untuk membedakan setiap blok puzzle (dalam hal ini, blok-blok puzzle diwakili dengan karakter A-Z). Ada 26 warna berbeda yang dapat digunakan, yang dikelompokkan menjadi kategori-kategori sebagai berikut:

1. **Warna Dasar (A–F)**
   Merah, hijau, kuning, biru, ungu, dan cyan.
2. **Warna Cerah (G–L)**
   Versi terang dari warna dasar.
3. **Warna Latar (M–R)**
   Latar belakang dengan warna dasar.
4. **Warna Latar Cerah (S–X)**
   Latar belakang dengan warna terang.
5. **Warna Khusus (Y–Z)**
   Putih terang dan putih biasa.

Setiap blok puzzle yang teridentifikasi dengan huruf A-Z akan diberi warna sesuai dengan urutan dalam array COLORS. Hal ini membuat tampilan puzzle di console lebih mudah dibaca dan dipahami, terutama ketika puzzle melibatkan banyak blok yang berbeda.

Metode writeSolution menangani dua aspek output, yaitu:

1. **Penyimpanan ke dalam File**
   - Fungsi ini menulis konfigurasi grid solusi puzzle (berupa array dua dimensi grid) ke dalam file output, satu baris per satu baris.
   - Setelah itu, informasi mengenai waktu eksekusi (timeMs) dan jumlah iterasi (iterations) dicatat dalam file.
   - Penggunaan try-with-resources memastikan bahwa PrintWriter yang digunakan untuk menulis file ditutup dengan aman, mencegah kebocoran memori.
2. **Output ke Console**

- Fungsi ini juga menampilkan solusi puzzle ke console dengan pewarnaan blok puzzle sesuai dengan identitas bloknya menggunakan printColoredSolution.
- Di samping itu, informasi statistik performa, seperti waktu pencarian dan jumlah iterasi, juga ditampilkan di console agar pengguna dapat menilai efektivitas algoritma yang digunakan.

Metode printColoredSolution bertugas untuk mencetak solusi puzzle di console dengan pewarnaan berdasarkan identitas bloknya. Setiap sel pada grid solusi diproses satu per satu. Jika karakter pada sel tersebut adalah titik (.), maka tidak ada pewarnaan yang diterapkan. Untuk karakter lain (blok puzzle), warna yang sesuai dipilih dari array COLORS berdasarkan kode ASCII karakter tersebut. Setiap warna diterapkan ke karakter sesuai dengan urutan dalam array, yang mewakili blok A hingga Z. Setelah pewarnaan, format warna kembali ke normal dengan menggunakan kode RESET.

Metode ini juga memastikan bahwa format grid tetap konsisten, dengan pemisahan baris yang tepat dan pewarnaan yang jelas, menjadikannya mudah dipahami oleh pengguna.

Keunggulan Desain:

1. **Modularitas**

   Fungsi penyimpanan dan tampilan solusi dipisahkan dengan jelas. writeSolution bertanggung jawab untuk menulis ke file, sementara printColoredSolution mengelola tampilan di console.

2. **Keamanan Resource**

   Penggunaan try-with-resources dalam writeSolution memastikan bahwa resource seperti PrintWriter ditutup dengan benar, menghindari potensi kebocoran memori.

3. **User Experience**

   Dengan pewarnaan blok puzzle yang diterapkan di console, pengguna dapat lebih mudah membaca dan memverifikasi solusi puzzle. Sistem pewarnaan juga membuat solusi lebih menarik dan lebih jelas.

4. **Maintainability dan Ekstensibilitas**

   Struktur kode yang bersih dan terorganisir memungkinkan pengembang untuk menambah format output baru di masa depan, serta memperkenalkan metode baru untuk penanganan file output tanpa mengubah struktur yang sudah ada.

5. **Robust Error Handling**

   Penanganan IOException yang tepat memastikan bahwa jika terjadi kesalahan dalam operasi file, kesalahan tersebut dapat dikelola dengan baik tanpa menyebabkan crash aplikasi.

Kelas Writer memainkan peran krusial dalam menyediakan solusi yang dapat dibaca baik dalam file maupun di console. Dengan penggunaan pewarnaan ANSI yang cerdas dan pemisahan fungsi yang jelas, kelas ini menyediakan cara yang efektif untuk menampilkan solusi puzzle serta informasi performa secara menarik dan terstruktur. Desainnya yang modular dan aman juga memastikan kemudahan pemeliharaan dan pengembangan lebih lanjut.

```
package file;

import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import puzzle.Board;

public class Writer {
    private String outputPath;

    // ANSI color codes for 26 different colors (A-Z)
    public static final String RESET = "\u001B[0m";
    public static final String[] COLORS = {
        "\u001B[31m",    // RED (A)
        "\u001B[32m",    // GREEN (B)
        "\u001B[33m",    // YELLOW (C)
        "\u001B[34m",    // BLUE (D)
        "\u001B[35m",    // PURPLE (E)
```

```
        "\u001B[36m",     // CYAN (F)
        "\u001B[91m",     // BRIGHT RED (G)
        "\u001B[92m",     // BRIGHT GREEN (H)
        "\u001B[93m",     // BRIGHT YELLOW (I)
        "\u001B[94m",     // BRIGHT BLUE (J)
        "\u001B[95m",     // BRIGHT MAGENTA (K)
        "\u001B[96m",     // BRIGHT CYAN (L)
        "\u001B[41m",     // RED BACKGROUND (M)
        "\u001B[42m",     // GREEN BACKGROUND (N)
        "\u001B[43m",     // YELLOW BACKGROUND (O)
        "\u001B[44m",     // BLUE BACKGROUND (P)
        "\u001B[45m",     // PURPLE BACKGROUND (Q)
        "\u001B[46m",     // CYAN BACKGROUND (R)
        "\u001B[101m",    // BRIGHT RED BACKGROUND (S)
        "\u001B[102m",    // BRIGHT GREEN BACKGROUND (T)
        "\u001B[103m",    // BRIGHT YELLOW BACKGROUND (U)
        "\u001B[104m",    // BRIGHT BLUE BACKGROUND (V)
        "\u001B[105m",    // BRIGHT MAGENTA BACKGROUND (W)
        "\u001B[106m",    // BRIGHT CYAN BACKGROUND (X)
        "\u001B[97m",     // BRIGHT WHITE (Y)
        "\u001B[37m"      // WHITE (Z)
    };

    public Writer(String inputPath) {
        this.outputPath = inputPath.replace("input", "output/txt");
    }

     public void writeSolution(Board board, long timeMs, long
  iterations, String message) throws IOException {
                try (PrintWriter writer = new PrintWriter(new
  FileWriter(outputPath))) {
            char[][] grid = board.getGrid();

            // Write to file
            for (char[] row : grid) {
                writer.println(new String(row));
            }
            writer.println("\n" + message);
            writer.printf("Search duration: %d ms\n", timeMs);
            writer.printf("Number of iterations: %d\n", iterations);
```

```
            // Print to console with colors
            System.out.println("\nAttempted solution:");
            printColoredSolution(board);
            System.out.println("\n" + message);
            System.out.printf("Search duration: %d ms\n", timeMs);
                      System.out.printf("Number  of  iterations:  %d\n",
      iterations);
        }
    }

    public static void printColoredSolution(Board board) {
        char[][] grid = board.getGrid();
        System.out.println();
        for (char[] row : grid) {
            for (char c : row) {
                if (c == '.') {
                    System.out.print('.');
                } else {
                    int colorIndex = c - 'A';
                      if (colorIndex >= 0 && colorIndex < COLORS.length)
      {
                                System.out.print(COLORS[colorIndex] + c +
      RESET);
                    } else {
                        System.out.print(c);
                    }
                }
            }
            System.out.println();
        }
    }
}
```

2.4 Block

Kelas Block adalah komponen fundamental dalam sistem puzzle solver yang berfungsi untuk merepresentasikan setiap potongan puzzle. Kelas ini mengelola bentuk dan transformasi potongan puzzle dengan menggunakan sistem koordinat dua dimensi, yang memungkinkan manipulasi fleksibel sesuai dengan kebutuhan

dalam proses penyelesaian puzzle. Potongan puzzle ini dapat mengalami berbagai transformasi seperti rotasi atau pencerminan (flip), yang diatur dalam kelas ini melalui serangkaian metode yang memungkinkan perubahan bentuk potongan secara efisien dan konsisten.

Kelas Block memiliki dua atribut utama:

1. **id (char)**

   Merupakan identifikator unik untuk setiap potongan puzzle, yang berupa karakter dari A-Z. Ini digunakan untuk membedakan setiap potongan puzzle.
2. **coordinates (List<int[]>)**

   Menyimpan daftar koordinat yang mendefinisikan bentuk dari potongan puzzle. Setiap elemen dalam coordinates adalah sebuah array dengan dua nilai, yang merepresentasikan posisi baris (row) dan kolom (column) dalam grid.

Konstruktor kelas Block menerima parameter pattern, yaitu sebuah string yang merepresentasikan bentuk potongan puzzle dalam format multi-baris. Dalam proses konstruksi:

- **Inisialisasi ArrayList**

  ArrayList coordinates diinisialisasi untuk menyimpan posisi relatif dari potongan puzzle.
- **Identifikasi ID Potongan**

  Konstruktor memproses baris pertama dari pola (pattern) untuk menemukan karakter non-spasi pertama, yang digunakan sebagai ID potongan.
- **Pemetaan Karakter ke Koordinat**

  Setiap karakter dalam pola yang sama dengan ID potongan dipetakan ke koordinat dua dimensi (i, j) yang mewakili posisi baris dan kolom pada grid.
- **Normalisasi Koordinat**

  Setelah pemetaan selesai, metode normalizeCoordinates dipanggil untuk menyelaraskan posisi potongan dengan menggeser koordinat relatif sehingga potongan puzzle memiliki posisi yang standar, terlepas dari lokasi aslinya dalam grid.

Metode transformasi:

1. **copy()**

   Metode ini digunakan untuk membuat salinan independen dari potongan puzzle. Proses duplikasi ini mencakup:

   - Duplikasi ID potongan.
   - Deep copy dari koordinat untuk menghindari shared reference, sehingga perubahan pada objek salinan tidak memengaruhi objek asli.
   - Membuat objek Block baru dengan koordinat yang sama, memastikan independensi antara objek lama dan salinan.

2. **rotate90()**

   Metode ini melakukan rotasi 90 derajat searah jarum jam terhadap potongan puzzle. Rotasi dilakukan dengan cara mengubah setiap koordinat (x, y) menjadi (y, -x), sesuai dengan matriks rotasi 90 derajat. Setelah rotasi, normalizeCoordinates dipanggil untuk memastikan koordinat tetap terstandardisasi dan bentuk potongan tetap konsisten.

3. **flipHorizontal()**

   Metode ini melakukan pencerminan horizontal pada potongan puzzle dengan cara membalikkan nilai koordinat kolom (x) menjadi negatif. Seperti halnya rotasi, setelah pencerminan dilakukan, normalizeCoordinates dipanggil untuk menyesuaikan posisi potongan.

4. **normalizeCoordinates()**

   Metode ini bertanggung jawab untuk menormalisasi posisi koordinat. Proses ini dimulai dengan mencari nilai minimum untuk baris (minRow) dan kolom (minCol) dari seluruh koordinat yang ada. Kemudian, semua koordinat digeser relatif terhadap titik minimum ini, sehingga posisi potongan selalu dimulai dari titik koordinat (0, 0), memberikan konsistensi dalam representasi potongan puzzle.

Metode akses:

- **getId()**

  Mengembalikan ID potongan puzzle yang unik (karakter A-Z).
- **getCoordinates()**

  Mengembalikan daftar koordinat yang mendefinisikan bentuk potongan puzzle.

Keunggulan implementasi:

1. **Enkapsulasi Data**

   Atribut-atribut id dan coordinates dilindungi dengan akses terkontrol melalui metode akses publik (getId dan getCoordinates), yang menjaga integritas data.
2. **Pemeliharaan Bentuk**

   Dengan adanya metode normalizeCoordinates, bentuk potongan puzzle dijaga konsistensinya meskipun telah dilakukan transformasi seperti rotasi atau pencerminan.
3. **Fleksibilitas**

   Kelas Block mendukung berbagai bentuk potongan puzzle, baik yang sederhana maupun yang kompleks, dengan memungkinkan rotasi dan pencerminan potongan.
4. **Efisiensi Memori**

   Penggunaan struktur data seperti ArrayList<int[]> yang dinamis memungkinkan pengelolaan koordinat secara efisien, terutama jika jumlah koordinat berubah setelah transformasi.
5. **Keamanan Operasi**

   Dengan menggunakan deep copy dalam metode copy(), objek baru yang dihasilkan tidak akan mempengaruhi objek asli, menjaga keamanan data dan mencegah efek samping yang tidak diinginkan.
6. **Konsistensi**

   Metode normalizeCoordinates memastikan bahwa potongan puzzle selalu memiliki posisi yang standar, tidak peduli berapa kali potongan tersebut telah diputar atau dicerminkan.

Kelas Block berfungsi sebagai representasi dari setiap potongan puzzle dalam sistem puzzle solver. Dengan atribut-atribut yang mendefinisikan ID dan koordinat potongan, serta metode untuk memanipulasi dan mentransformasikan potongan melalui rotasi dan pencerminan, kelas ini memberikan fleksibilitas dan kemudahan dalam menangani potongan puzzle. Proses normalisasi koordinat memastikan bahwa potongan-potongan puzzle tetap konsisten dan terstandardisasi, yang penting untuk menyelesaikan puzzle dengan akurat. Kelas ini mendukung manipulasi yang efisien dan menjaga integritas data melalui enkapsulasi dan deep copy, menjadikannya komponen yang krusial dalam sistem solver puzzle.

```
package puzzle;

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

public class Block {
    private char id;
    private List<int[]> coordinates;

    public Block(String pattern) {
        this.coordinates = new ArrayList<>();
        String[] lines = pattern.trim().split("\n");

        // Find first non-space character as ID
        for (int i = 0; i < lines[0].length(); i++) {
            char c = lines[0].charAt(i);
            if (c != ' ') {
                this.id = c;
                break;
            }
        }

        // Store coordinates for each character in the pattern
```

```
        for (int i = 0; i < lines.length; i++) {
            String line = lines[i];
            for (int j = 0; j < line.length(); j++) {
                if (line.charAt(j) == id) {
                    coordinates.add(new int[]{i, j});
                }
            }
        }
        normalizeCoordinates();
    }

    public Block copy() {
        Block newBlock = new Block(String.valueOf(id));
        newBlock.coordinates = new ArrayList<>();
        for (int[] coord : coordinates) {
            newBlock.coordinates.add(new int[]{coord[0], coord[1]});
        }
        return newBlock;
    }

    public void rotate90() {
        for (int[] coord : coordinates) {
            int temp = coord[0];
            coord[0] = coord[1];
            coord[1] = -temp;
        }
        normalizeCoordinates();
    }

    public void flipHorizontal() {
        for (int[] coord : coordinates) {
            coord[1] = -coord[1];
        }
        normalizeCoordinates();
    }

    private void normalizeCoordinates() {
```

```
        int minRow = Integer.MAX_VALUE;
        int minCol = Integer.MAX_VALUE;

        for (int[] coord : coordinates) {
            minRow = Math.min(minRow, coord[0]);
            minCol = Math.min(minCol, coord[1]);
        }

        for (int[] coord : coordinates) {
            coord[0] -= minRow;
            coord[1] -= minCol;
        }
    }

    public char getId() {
        return id;
    }

    public List<int[]> getCoordinates() {
        return coordinates;
    }
}
```

2.5 Board

Kelas Board merupakan representasi dari papan permainan puzzle IQ Pro, yang mengelola status dan logika penempatan blok dalam permainan. Kelas ini mendukung dua mode permainan yang berbeda, yaitu DEFAULT dan CUSTOM, yang memengaruhi cara penempatan blok dan interaksi dengan elemen-elemen lain dalam grid. Dalam implementasinya, kelas Board bertanggung jawab untuk memvalidasi penempatan blok, menempatkan atau menghapus blok dari grid, serta mengakses dimensi papan dan jenis puzzle yang sedang dimainkan.

Atribut kelas:

- **rows (int)**

  Menyimpan jumlah baris dalam grid papan permainan.

- **cols (int)**

  Menyimpan jumlah kolom dalam grid papan permainan.
- **grid (char[][])**

  Matriks 2D yang merepresentasikan status papan permainan. Setiap elemen grid menyimpan karakter yang menunjukkan apakah sel tersebut kosong ('.') atau terisi oleh ID blok tertentu.
- **customConfig (char[][])**

  Menyimpan konfigurasi khusus untuk mode CUSTOM, yang berisi informasi tambahan yang memengaruhi cara blok dapat ditempatkan pada papan.
- **isCustomMode (boolean)**

  Flag yang menandakan apakah permainan sedang berada dalam mode CUSTOM atau tidak.
- **puzzleType (String)**

  Menyimpan jenis puzzle yang digunakan, yang dapat berupa "DEFAULT" atau "CUSTOM", menentukan logika yang digunakan untuk penempatan blok.

Konstruktor kelas Board bertanggung jawab untuk menginisialisasi papan permainan. Berikut langkah-langkah dalam konstruksinya:

1. **Menyimpan dimensi papan**

   Dimensi papan ditentukan oleh parameter rows (baris) dan cols (kolom), yang diterima dalam konstruktor.
2. **Menentukan mode permainan**

   Mode permainan ditentukan berdasarkan apakah ada konfigurasi khusus yang diberikan dalam parameter customConfig. Jika customConfig bernilai null, maka mode yang digunakan adalah DEFAULT, sedangkan jika ada konfigurasi yang diberikan, maka mode yang digunakan adalah CUSTOM.
3. **Inisialisasi grid kosong**

   Grid diinisialisasi dengan karakter '.' pada setiap selnya, menandakan bahwa seluruh papan awalnya kosong.

4. **Menyimpan konfigurasi khusus**

   Jika mode CUSTOM dipilih, konfigurasi khusus disalin ke atribut customConfig.

Metode penempatan blok:

1. **canPlace(Block block, int row, int col)**

   Metode ini memeriksa apakah sebuah blok dapat ditempatkan pada posisi tertentu di papan, baik dalam mode DEFAULT maupun CUSTOM:

   a. **Mode DEFAULT**

      Metode ini memeriksa dua hal utama:

      - Batas papan

        Memastikan koordinat baru dari setiap titik pada blok berada di dalam batas grid papan.
      - Tumpang tindih

        Memastikan bahwa blok tidak menumpuk dengan blok lain yang sudah ada di grid (yaitu, memastikan bahwa sel yang akan ditempati blok kosong, ditandai dengan '.').
      - Jika kedua syarat ini terpenuhi untuk semua koordinat blok, metode akan mengembalikan true, menunjukkan bahwa blok dapat ditempatkan.

   b. **Mode CUSTOM**

      Selain memeriksa batas papan dan tumpang tindih yang sama seperti pada mode DEFAULT, mode CUSTOM juga mensyaratkan bahwa blok harus menyentuh minimal satu sel yang bertanda 'X'. Hal ini dilakukan untuk memastikan bahwa penempatan blok sesuai dengan aturan khusus yang diterapkan pada konfigurasi permainan.

2. **place(Block block, int row, int col)**

   Metode ini menempatkan blok pada papan di posisi yang ditentukan dengan mengisi sel-sel yang sesuai di grid dengan ID blok tersebut. ID blok diambil dari karakter yang disimpan dalam objek Block. Setiap koordinat blok akan dipetakan ke posisi dalam grid berdasarkan parameter row dan col.

3. **remove(Block block, int row, int col)**

   Metode ini menghapus blok dari papan dengan mengembalikan karakter di sel-sel yang ditempati oleh blok menjadi '.'. Proses ini sebaliknya dari metode place, di mana koordinat blok dikurangi dan grid dikembalikan ke kondisi kosong.

   Metode akses:

- **getRows()**

  Mengembalikan jumlah baris (rows) pada papan.
- **getCols()**

  Mengembalikan jumlah kolom (cols) pada papan.
- **getGrid()**

  Mengembalikan grid papan saat ini dalam bentuk array 2D char[][].
- **getPuzzleType()**

  Mengembalikan jenis puzzle yang sedang dimainkan, yaitu "DEFAULT" atau "CUSTOM".

  Keunggulan implementasi

1. **Fleksibilitas Mode**

   Kelas Board mendukung dua mode permainan yang berbeda, yaitu DEFAULT dan CUSTOM, memberikan kemampuan untuk menyesuaikan aturan permainan berdasarkan jenis puzzle yang dimainkan.
2. **Validasi Komprehensif**

   Metode canPlace melakukan pemeriksaan yang mendalam terhadap validitas penempatan blok, termasuk memeriksa batas papan, tumpang tindih dengan blok lain, dan aturan khusus untuk mode CUSTOM.
3. **Enkapsulasi**

   Atribut-atribut kelas seperti grid, rows, cols, dan customConfig dilindungi dengan akses terkontrol, menjamin integritas data dan memastikan hanya metode yang relevan yang dapat memanipulasi data tersebut.

4. **Efisiensi**

   Penggunaan array 2D untuk menyimpan status grid memungkinkan representasi yang efisien dan mudah diakses dari sudut pandang penyelesaian puzzle. Akses dan manipulasi sel grid dilakukan dengan cepat menggunakan indeks array.

5. **Pemisahan Concern**

   Kelas Board memisahkan logika penempatan blok dari manipulasi grid, yang memastikan desain yang bersih dan memudahkan pengembangan lebih lanjut, seperti penambahan logika permainan baru.

Kelas Board menyediakan dasar yang kuat untuk representasi papan permainan dalam aplikasi puzzle IQ Pro. Dengan mendukung dua mode permainan yang berbeda, serta logika penempatan blok yang canggih dan validasi yang komprehensif, kelas ini memungkinkan pembuatan permainan puzzle yang fleksibel dan dapat disesuaikan. Dengan menggunakan array 2D untuk menyimpan status grid dan enkapsulasi yang baik untuk atribut kelas, implementasi ini menjaga efisiensi dan integritas data, sekaligus mendukung pengembangan lebih lanjut untuk algoritma penyelesaian puzzle yang lebih kompleks.

```
package puzzle;

public class Board {
    private int rows;
    private int cols;
    private char[][] grid;
    private char[][] customConfig;
    private boolean isCustomMode;
    private String puzzleType;

      public Board(int rows, int cols, char[][] customConfig, String
      puzzleType) {
        this.rows = rows;
        this.cols = cols;
        this.customConfig = customConfig;
        this.isCustomMode = (customConfig != null);
        this.puzzleType = puzzleType;
```

```
        this.grid = new char[rows][cols];

        // Initialize grid with empty spaces
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                grid[i][j] = '.';
            }
        }
    }

    public int getRows() {
        return rows;
    }

    public int getCols() {
        return cols;
    }

    public char[][] getGrid() {
        return grid;
    }

    public boolean canPlace(Block block, int row, int col) {
        // For DEFAULT mode or when customConfig is null
        if (!isCustomMode || customConfig == null) {
            for (int[] coord : block.getCoordinates()) {
                int newRow = row + coord[0];
                int newCol = col + coord[1];

                // Check bounds and overlap
                if (newRow < 0 || newRow >= rows ||
                    newCol < 0 || newCol >= cols ||
                    grid[newRow][newCol] != '.') {
                    return false;
                }
            }
            return true;
        }
        // For CUSTOM mode with customConfig
        else if (puzzleType.equals("CUSTOM")) {
            boolean touchesX = false;
```

```
            for (int[] coord : block.getCoordinates()) {
                int newRow = row + coord[0];
                int newCol = col + coord[1];

                if (newRow < 0 || newRow >= rows ||
                    newCol < 0 || newCol >= cols ||
                    grid[newRow][newCol] != '.') {
                    return false;
                }

                if (customConfig[newRow][newCol] == 'X') {
                    touchesX = true;
                }
            }
            return touchesX;
        }
        // Default case - cannot place block
        return false;
    }

    public void place(Block block, int row, int col) {
        for (int[] coord : block.getCoordinates()) {
            grid[row + coord[0]][col + coord[1]] = block.getId();
        }
    }

    public void remove(Block block, int row, int col) {
        for (int[] coord : block.getCoordinates()) {
            grid[row + coord[0]][col + coord[1]] = '.';
        }
    }

    public String getPuzzleType() {
        return puzzleType;
    }
}
```

2.6 Solver

Kelas Solver dalam kode di atas bertugas untuk menyelesaikan puzzle dengan memanfaatkan algoritma brute force yang menggabungkan teknik backtracking. Kelas ini dirancang untuk menangani dua mode permainan: DEFAULT dan CUSTOM, yang masing-masing memiliki logika penyelesaian yang sedikit berbeda. Kelas ini berfungsi untuk mencari solusi penempatan blok-blok pada papan permainan dengan cara menguji berbagai kemungkinan posisi, rotasi, dan orientasi blok hingga solusi yang valid ditemukan.

Solver berfungsi sebagai inti dari proses pencarian solusi untuk puzzle. Kelas ini bertanggung jawab untuk menentukan apakah blok dapat ditempatkan pada papan dalam berbagai orientasi dan posisi. Untuk mode DEFAULT, penyelesaian dilakukan dengan mencoba setiap posisi yang memungkinkan untuk setiap blok dalam urutan tertentu, dan menggunakan backtracking untuk mencoba solusi yang mungkin jika solusi sebelumnya gagal. Sementara itu, untuk mode CUSTOM, solver memiliki tambahan kondisi khusus yang memastikan bahwa blok yang ditempatkan harus menyentuh minimal satu sel 'X' dari konfigurasi khusus yang diberikan.

Komponen utama:

1. **Atribut Kelas**

   Atribut-atribut dalam kelas Solver menyimpan berbagai informasi yang diperlukan selama proses pencarian solusi:

   - **board**

     Sebuah objek Board yang mewakili papan permainan.
   - **blocks**

     List yang berisi blok-blok puzzle yang akan dipasang di papan.
   - **iterations**

     Menghitung jumlah iterasi yang dilakukan selama pencarian solusi.
   - **solved**

     Menyimpan status apakah puzzle telah berhasil diselesaikan atau belum.

- **customConfig**

  Menyimpan konfigurasi khusus (jika ada) untuk puzzle dalam bentuk matriks karakter.
- **puzzleType**

  Menyimpan tipe puzzle (DEFAULT atau CUSTOM) yang menentukan cara penyelesaian puzzle.

2. **Konstruktor**

   Konstruktor kelas Solver menerima dimensi papan, daftar blok, konfigurasi khusus, dan tipe puzzle sebagai parameter. Konstruktor ini menginisialisasi objek Board, mengurutkan blok berdasarkan ukuran (dari terbesar ke terkecil), dan mempersiapkan state awal untuk pencarian solusi.
3. **Metode solve()**

   Metode utama yang memulai proses penyelesaian puzzle. Berdasarkan tipe puzzle (DEFAULT atau CUSTOM), metode ini memanggil metode penyelesaian yang sesuai, yaitu solveDefault() untuk tipe DEFAULT dan solveCustom() untuk tipe CUSTOM.

   Algoritma penyelesaian **Brute Force:**

1. **Penyelesaian Mode DEFAULT**
   - Backtracking digunakan untuk memeriksa semua kemungkinan penempatan blok di papan. Pada setiap iterasi, blok akan dicoba pada posisi yang berbeda dan dalam berbagai rotasi (0°, 90°, 180°, 270°) dan orientasi (flip horizontal).
   - Proses ini dimulai dengan mencoba penempatan blok pertama, dan jika berhasil, akan melanjutkan ke blok berikutnya. Jika ada penempatan yang tidak valid, sistem akan kembali ke langkah sebelumnya untuk mencoba posisi lainnya.
   - Pencarian ini terus dilakukan hingga semua blok terpasang di papan atau hingga tidak ada solusi yang memungkinkan.
2. **Penyelesaian Mode CUSTOM**

- Pada mode CUSTOM, solver melakukan verifikasi tambahan untuk memastikan bahwa blok yang ditempatkan menyentuh minimal satu sel 'X' pada konfigurasi khusus.
- Algoritma ini memeriksa apakah semua sel dengan 'X' pada customConfig sudah terisi. Jika sudah, solver berhenti dan puzzle dianggap terpecahkan.
- Penyelesaian dilakukan dengan mencoba posisi dan orientasi yang mungkin untuk setiap blok, sambil memastikan blok memenuhi kondisi khusus yang diberikan oleh konfigurasi.

3. **Metode Pendukung**
    - **isCustomConfigFilled()**

      Metode ini memeriksa apakah semua sel dengan 'X' pada konfigurasi khusus telah terisi. Jika tidak, solver akan terus mencari solusi.
    - **sortBlocksBySize()**

      Mengurutkan blok berdasarkan ukuran untuk memprioritaskan blok yang lebih besar. Ini adalah optimasi yang membantu mempercepat pencarian solusi dengan menempatkan blok yang lebih besar terlebih dahulu, sehingga mengurangi kemungkinan kesalahan penempatan.
    - **tryPlaceBlock()**

      Metode ini mencoba menempatkan blok pada posisi tertentu di papan dan melanjutkan ke blok berikutnya jika penempatan berhasil.
4. **Fitur Tambahan**
    - Penghitungan Dimensi

      Metode getBlockHeight() dan getBlockWidth() digunakan untuk menghitung dimensi efektif setiap blok agar dapat diposisikan dengan benar pada papan.
    - Backtracking dengan Copy Blok

      Setiap kali mencoba penempatan blok, blok disalin menggunakan metode copy(). Hal ini memastikan bahwa blok yang diuji tidak merusak status blok lainnya, mendukung pencarian solusi tanpa mengubah blok yang sudah dipasang.

- Pengecekan Batas Papan
  Pengecekan dilakukan untuk memastikan blok tidak keluar dari batas papan dan tidak tumpang tindih dengan blok lain.

Keunggulan implementasi:

- **Fleksibilitas Mode**
  Dapat menangani kedua tipe puzzle, DEFAULT dan CUSTOM, dengan mekanisme penyelesaian yang sesuai.
- **Optimasi dengan Pengurutan Blok**
  Pengurutan blok berdasarkan ukuran memungkinkan pencarian solusi yang lebih efisien, terutama saat mencoba memasang blok besar terlebih dahulu.
- **Backtracking yang Efisien**
  Dengan menggunakan backtracking dan percabangan rotasi/orientasi, solver dapat mengeksplorasi semua kemungkinan penempatan blok secara sistematis.
- **Pengecekan Kondisi Khusus untuk Mode CUSTOM**
  Dengan menambahkan kondisi khusus pada mode CUSTOM, solver memastikan bahwa solusi memenuhi syarat tambahan, seperti menyentuh sel 'X'.

Akses state:

- **getIterations()**
  Menyediakan akses ke jumlah iterasi yang dilakukan selama proses pencarian solusi.
- **getBoard()**
  Menyediakan akses ke objek Board yang merepresentasikan papan permainan.
- **isSolved()**
  Mengembalikan status apakah puzzle telah berhasil diselesaikan.

Kelas Solver ini menawarkan solusi yang komprehensif dan efisien untuk menyelesaikan puzzle dengan berbagai konfigurasi dan mode permainan. Dengan menggabungkan teknik backtracking, pengecekan batas, dan rotasi/orientasi blok, kelas ini mampu menyelesaikan puzzle dengan berbagai kemungkinan yang sangat besar. Penggunaan pengurutan blok berdasarkan ukuran dan pengoptimalan pengecekan kondisi memastikan algoritma ini bekerja dengan efisien meskipun puzzle yang diselesaikan memiliki kompleksitas yang tinggi.

```
package puzzle;

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

public class Solver {
    private Board board;
    private List<Block> blocks;
    private long iterations;
    private boolean solved;
    private char[][] customConfig;
    private String puzzleType;
    private static final int NO_SOLUTION = 0;
    private static final int PUZZLE_BIGGER = 1;
    private static final int BOARD_BIGGER = 2;
    private int solutionStatus = NO_SOLUTION;

      public Solver(int rows, int cols, List<Block> blocks, char[][]
      customConfig, String puzzleType) {
        this.puzzleType = puzzleType;
        this.board = new Board(rows, cols, customConfig, puzzleType);
        this.blocks = sortBlocksBySize(blocks);
        this.iterations = 0;
        this.solved = false;
        this.customConfig = customConfig;
    }

    private boolean isCustomConfigFilled() {
        if (customConfig == null) return true;
```

```
        for (int i = 0; i < board.getRows(); i++) {
            for (int j = 0; j < board.getCols(); j++) {
                 if (customConfig[i][j] == 'X' && board.getGrid()[i][j]
  == '.') {
                    return false;
                }
            }
        }
        return true;
    }

    public boolean solve() {
              return  puzzleType.equals("CUSTOM")  ?  solveCustom(0)  :
  solveDefault(0);
    }

    private boolean solveDefault(int blockIndex) {
        iterations++;

        // Case 1: All blocks placed but board not full
        if (blockIndex >= blocks.size()) {
            if (!isBoardComplete()) {
                solutionStatus = BOARD_BIGGER;
                return false;
            }
            solved = true;
            return true;
        }

        // Case 2: Board full but blocks remain
        if (isBoardComplete() && blockIndex < blocks.size()) {
            solutionStatus = PUZZLE_BIGGER;
            return false;
        }

        Block currentBlock = blocks.get(blockIndex).copy();
        int maxRow = board.getRows() - getBlockHeight(currentBlock);
        int maxCol = board.getCols() - getBlockWidth(currentBlock);

        for (int row = 0; row <= maxRow; row++) {
            for (int col = 0; col <= maxCol; col++) {
```

```
                // Try original orientation
                  if (tryPlaceBlock(currentBlock, row, col, blockIndex))
    {
                      return true;
                }

                // Try flipped orientation
                currentBlock.flipHorizontal();
                  if (tryPlaceBlock(currentBlock, row, col, blockIndex))
    {
                      return true;
                }
                currentBlock.flipHorizontal();

                // Try 90-degree rotation
                currentBlock.rotate90();
                  if (tryPlaceBlock(currentBlock, row, col, blockIndex))
    {
                      return true;
                }

                // Try flipped 90-degree rotation
                currentBlock.flipHorizontal();
                  if (tryPlaceBlock(currentBlock, row, col, blockIndex))
    {
                      return true;
                }
            }
        }

        return false;
    }

    private boolean solveCustom(int blockIndex) {
        iterations++;

        if (blockIndex == 0) {
            int totalPieceSize = 0;
            for (Block block : blocks) {
                totalPieceSize += block.getCoordinates().size();
            }
```

```
        int totalXPositions = 0;
        for (int i = 0; i < board.getRows(); i++) {
            for (int j = 0; j < board.getCols(); j++) {
                if (customConfig[i][j] == 'X') {
                    totalXPositions++;
                }
            }
        }

        if (totalPieceSize > totalXPositions) {
            solutionStatus = PUZZLE_BIGGER;
            return false;
        }
    }

    if (blockIndex >= blocks.size()) {
        if (!isCustomConfigFilled()) {
            return false;
        }
        solved = true;
        return true;
    }

    Block currentBlock = blocks.get(blockIndex).copy();
    int maxRow = board.getRows() - getBlockHeight(currentBlock);
    int maxCol = board.getCols() - getBlockWidth(currentBlock);

    for (int row = 0; row <= maxRow; row++) {
        for (int col = 0; col <= maxCol; col++) {
            // Try original orientation
            if (board.canPlace(currentBlock, row, col)) {
                board.place(currentBlock, row, col);
                if (solveCustom(blockIndex + 1)) {
                    return true;
                }
                board.remove(currentBlock, row, col);
            }

            // Try flipped orientation
            currentBlock.flipHorizontal();
```

```
                if (board.canPlace(currentBlock, row, col)) {
                    board.place(currentBlock, row, col);
                    if (solveCustom(blockIndex + 1)) {
                        return true;
                    }
                    board.remove(currentBlock, row, col);
                }
                currentBlock.flipHorizontal();

                // Try rotations
                for (int i = 0; i < 3; i++) {
                    currentBlock.rotate90();
                    if (board.canPlace(currentBlock, row, col)) {
                        board.place(currentBlock, row, col);
                        if (solveCustom(blockIndex + 1)) {
                            return true;
                        }
                        board.remove(currentBlock, row, col);
                    }

                    // Try flipped rotation
                    currentBlock.flipHorizontal();
                    if (board.canPlace(currentBlock, row, col)) {
                        board.place(currentBlock, row, col);
                        if (solveCustom(blockIndex + 1)) {
                            return true;
                        }
                        board.remove(currentBlock, row, col);
                    }
                    currentBlock.flipHorizontal();
                }
            }
        }
        return false;
    }

    private boolean isBoardComplete() {
        char[][] grid = board.getGrid();
        for (int i = 0; i < board.getRows(); i++) {
            for (int j = 0; j < board.getCols(); j++) {
                if (grid[i][j] == '.') {
```

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

    private List<Block> sortBlocksBySize(List<Block> blocks) {
        List<Block> sorted = new ArrayList<>(blocks);
                                            sorted.sort((a,      b)      ->
      Integer.compare(b.getCoordinates().size(),
      a.getCoordinates().size()));
        return sorted;
    }

     private boolean tryPlaceBlock(Block block, int row, int col, int
      blockIndex) {
        if (board.canPlace(block, row, col)) {
            board.place(block, row, col);
            if (solveDefault(blockIndex + 1)) {
                return true;
            }
            board.remove(block, row, col);
        }
        return false;
    }

    private int getBlockHeight(Block block) {
        int maxRow = 0;
        for (int[] coord : block.getCoordinates()) {
            maxRow = Math.max(maxRow, coord[0]);
        }
        return maxRow + 1;
    }

    private int getBlockWidth(Block block) {
        int maxCol = 0;
        for (int[] coord : block.getCoordinates()) {
            maxCol = Math.max(maxCol, coord[1]);
        }
        return maxCol + 1;
```

```
    }

    public long getIterations() {
        return iterations;
    }

    public Board getBoard() {
        return board;
    }

    public boolean isSolved() {
        return solved;
    }

    public int getSolutionStatus() {
        return solutionStatus;
    }
}
```

2.7 Main

Kelas Main yang tercantum di atas adalah bagian dari aplikasi berbasis JavaFX yang digunakan untuk menyelesaikan puzzle dalam bentuk tiga dimensi (3D). Aplikasi ini, yang diberi nama "IQ Puzzler Pro Solver", memungkinkan pengguna untuk memvisualisasikan puzzle dalam ruang 3D, berinteraksi dengan elemen-elemen puzzle, serta melihat solusi yang ditemukan oleh algoritma penyelesaian puzzle. Kelas ini bertanggung jawab atas berbagai tugas utama, seperti menampilkan puzzle dalam bentuk 3D, membuat elemen-elemen grafis untuk puzzle (seperti grid dan bola), menghubungkan elemen-elemen puzzle, serta menangani interaksi pengguna dengan antarmuka grafis (GUI).

Penjelasan detil kode

1. **Metode displayPuzzle()**

   Fungsi ini bertugas untuk meng-update tampilan puzzle setiap kali puzzle baru dipilih atau diproses.

   Langkah pertama adalah membersihkan semua komponen puzzle yang ada di puzzleGroup dan addedConnections (yang mencatat koneksi antar

elemen puzzle).

Kemudian, grid puzzle (char[][] grid) diambil dari objek currentBoard, dan posisi elemen-elemen puzzle diatur dengan spasi yang ditentukan oleh SPHERE_RADIUS * 2.5.

Puzzle kemudian diproses dengan memanggil metode-metode untuk membuat elemen-elemen visual, seperti papan utama, bola (sphere), dan koneksi antar elemen puzzle. Semua elemen ini ditambahkan ke grup puzzle (puzzleGroup), yang selanjutnya akan ditampilkan di antarmuka pengguna.

2. **Metode createBoardGroup()**

   Fungsi ini membuat grup utama yang berisi papan puzzle dan elemen-elemen lainnya, seperti indentasi pada papan dan tepi melengkung.

   Box digunakan untuk membuat papan utama dengan dimensi yang dihitung berdasarkan ukuran grid puzzle. Papan ini diberi material berwarna abu-abu gelap menggunakan PhongMaterial.

   createIndentations() dan createCurvedEdge() adalah dua metode tambahan yang digunakan untuk membuat indentasi di papan (dimana elemen puzzle akan ditempatkan) dan tepi melengkung pada papan.

3. **Metode createIndentations()**

   Fungsi ini mengiterasi grid puzzle dan menambahkan elemen-elemen indentasi dalam bentuk bola (Sphere) di posisi yang sesuai.

   Bola ini digunakan untuk menunjukkan lokasi dimana elemen puzzle dapat ditempatkan di papan. Jika posisi grid berisi karakter selain titik (.), maka sebuah bola diletakkan di tempat tersebut dengan warna material yang berbeda.

4. **Metode createCurvedEdge()**

   Fungsi ini membuat tepi melengkung pada papan puzzle menggunakan objek Cylinder untuk memberikan tampilan visual yang lebih menarik. Tepi melengkung ini diberi material abu-abu gelap yang sesuai dengan tampilan keseluruhan papan.

5. **Metode createPieceSpheres()**

   Fungsi ini bertugas untuk membuat bola (sphere) yang mewakili

elemen-elemen puzzle yang ada di grid.

Setiap elemen puzzle diberi warna berdasarkan karakter pada grid menggunakan PhongMaterial. Bola-bola ini diletakkan pada posisi yang dihitung berdasarkan indeks grid, dan mereka ditambahkan ke puzzleGroup agar bisa ditampilkan.

Elemen puzzle dikelompokkan dalam map berdasarkan karakter mereka (misalnya, 'A', 'B', dll.), yang kemudian memungkinkan untuk menghubungkan bola-bola yang memiliki karakter yang sama.

6. **Metode createConnections()**

   Fungsi ini bertanggung jawab untuk membuat koneksi antara elemen-elemen puzzle yang berdekatan dan memiliki karakter yang sama.

   Menggunakan metode checkAndCreateConnection(), fungsi ini memeriksa setiap elemen puzzle untuk melihat apakah ada elemen puzzle yang berdekatan (baik secara horizontal atau vertikal) dan memiliki karakter yang sama. Jika ada, maka koneksi dibuat antara elemen-elemen tersebut menggunakan objek Cylinder.

   Setiap koneksi diberi warna yang sama dengan karakter elemen yang terhubung, dan diposisikan dan diputar agar sesuai dengan posisi dua bola yang terhubung.

7. **Metode checkAndCreateConnection()**

   Fungsi ini mengecek apakah dua posisi (di grid puzzle) berisi karakter yang sama. Jika demikian, maka metode ini akan membuat koneksi di antara kedua posisi tersebut. Fungsi ini memanfaatkan createConnection() untuk membuat objek koneksi yang sesuai.

8. **Metode createConnection()**

   Fungsi ini membuat objek Cylinder yang menghubungkan dua elemen puzzle berdasarkan posisi mereka.

   Koneksi ini kemudian diposisikan dan diputar agar sejajar dengan dua elemen yang terhubung. Koneksi diberi material dengan warna yang sesuai dengan karakter puzzle yang terhubung.

9. **Metode resetView()**

   Fungsi ini digunakan untuk mereset tampilan 3D dari puzzle. Semua rotasi dan skala diatur ulang ke nilai default, dengan tampilan rotasi X dan Y diatur agar puzzle kembali ke posisi awalnya.

10. **Metode showInfo() dan showError()**

    Kedua metode ini digunakan untuk menampilkan jendela dialog kepada pengguna untuk memberikan informasi atau menunjukkan pesan kesalahan.
    showInfo() digunakan untuk menampilkan pesan informasi, sedangkan showError() digunakan untuk menampilkan pesan kesalahan.

11. **Metode updateStatus()**

    Fungsi ini digunakan untuk memperbarui status aplikasi dengan menampilkan waktu pencarian solusi dan jumlah iterasi yang dilakukan oleh algoritma penyelesaian.
    Setelah perhitungan selesai, label status di antarmuka pengguna akan diupdate dengan informasi ini.

12. **Metode main()**

    Fungsi ini adalah titik awal dari aplikasi JavaFX. Di dalamnya, metode launch(args) dipanggil untuk memulai aplikasi JavaFX.

Informasi tambahan:

1. **Integrasi dengan Sistem Lain**

   Kelas Main berinteraksi dengan sistem lain seperti Solver, Reader, dan Writer, meskipun detail implementasi mereka tidak termasuk dalam kode yang diberikan. Namun, dapat diperkirakan bahwa:

   - Solver bertanggung jawab untuk menyelesaikan puzzle berdasarkan grid yang diberikan.
   - Reader mungkin digunakan untuk membaca data puzzle dari file atau sumber lain.
   - Writer digunakan untuk menulis hasil penyelesaian puzzle atau status aplikasi ke file.

2. **Desain Antarmuka Pengguna**

- Aplikasi ini mengandalkan JavaFX untuk antarmuka pengguna, memanfaatkan 3D Graphics (seperti Sphere, Box, Cylinder) untuk menampilkan elemen-elemen puzzle dalam ruang tiga dimensi.
- Fungsi-fungsi seperti rotate, scale, dan translate digunakan untuk memanipulasi posisi, rotasi, dan ukuran objek dalam ruang 3D, memberikan pengalaman visual yang menarik bagi pengguna.

3. **Fitur Interaktif**
    - Pengguna dapat berinteraksi dengan puzzle, memanipulasi tampilan 3D untuk memeriksa elemen-elemen puzzle dari berbagai sudut.
    - Koneksi antar elemen puzzle dapat dilihat dengan jelas, memberikan gambaran yang lebih jelas tentang cara solusi dihasilkan.

Dengan demikian, kelas Main.java ini menyediakan fungsionalitas dasar yang diperlukan untuk memvisualisasikan dan menyelesaikan puzzle 3D secara interaktif dalam aplikasi IQ Puzzler Pro Solver.

```
package main;

import javafx.application.Application;
import javafx.application.Platform;
import javafx.geometry.Pos;
import javafx.geometry.Insets;
import javafx.scene.*;
import javafx.scene.control.*;
import javafx.scene.layout.*;
import javafx.scene.paint.Color;
import javafx.scene.paint.PhongMaterial;
import javafx.scene.shape.*;
import javafx.scene.transform.Rotate;
import javafx.scene.transform.Scale;
import javafx.stage.Stage;
import javafx.stage.Modality;
import javafx.embed.swing.SwingFXUtils;
import javafx.scene.image.WritableImage;
import javax.imageio.ImageIO;
```

```
import java.io.File;
import java.util.*;
import file.*;
import puzzle.*;

public class Main extends Application {
    static {
        // Environment properties for WSL
        System.setProperty("prism.order", "sw");
        System.setProperty("prism.text", "t2k");
        System.setProperty("prism.forceGPU", "false");
        System.setProperty("glass.platform", "gtk");
        System.setProperty("javafx.platform", "gtk");
        System.setProperty("embedded", "false");
        System.setProperty("java.awt.headless", "false");
        System.setProperty("DISPLAY", ":0");
    }
    // Constants
    private static final double SPHERE_RADIUS = 20.0;

    // UI Components
    private Group puzzleGroup;
    private TextField fileNameField;
    private ProgressIndicator loadingIndicator;
    private Board currentBoard;
    private HBox buttonControls;
    private Label statusLabel = new Label("Enter a filename to start");
    private Label timeLabel = new Label();
    private Label iterationsLabel = new Label();
    private Label solutionFoundLabel = new Label("Solution found!");
    private VBox fileNameContainer;
    private VBox solutionContainer;
    private Group solutionFoundGroup;

    // 3D Controls
    private double mousePosX, mousePosY, mouseOldX, mouseOldY;
    private final Rotate rotateX = new Rotate(0, Rotate.X_AXIS);
    private final Rotate rotateY = new Rotate(0, Rotate.Y_AXIS);
    private double scaleValue = 1.0;
    private final Scale scaleTransform = new Scale(1, 1, 1);
```

```
    // State tracking
    private long solveTime, iterationCount, finalIterations;
    private Set<String> addedConnections = new HashSet<>();

    // Colors for puzzle pieces
    private static final Color[] PIECE_COLORS = {
        Color.RED, Color.GREEN, Color.YELLOW, Color.BLUE, Color.PURPLE,
      Color.CYAN,
        Color.rgb(255, 50, 50), Color.rgb(50, 255, 50), Color.rgb(255,
      255, 50),
        Color.rgb(50, 50, 255), Color.rgb(255, 50, 255), Color.rgb(50,
      255, 255),
        Color.rgb(180, 0, 0), Color.rgb(0, 180, 0), Color.rgb(180, 180,
      0),
        Color.rgb(0, 0, 180), Color.rgb(180, 0, 180), Color.rgb(0, 180,
      180),
                Color.rgb(255, 100, 100), Color.rgb(100, 255, 100),
      Color.rgb(255, 255, 100),
                Color.rgb(100, 100, 255), Color.rgb(255, 100, 255),
      Color.rgb(100, 255, 255),
        Color.WHITE, Color.LIGHTGRAY
    };

    @Override
    public void start(Stage primaryStage) {
        BorderPane root = new BorderPane();

        // Create input controls at top
        HBox inputControls = new HBox(10);
        fileNameField = new TextField();
        fileNameField.setPromptText("Enter test case filename");
        Button loadButton = new Button("Load & Solve");
        loadingIndicator = new ProgressIndicator();
        loadingIndicator.setVisible(false);
           inputControls.getChildren().addAll(fileNameField, loadButton,
      loadingIndicator);
        inputControls.setPadding(new Insets(10));
        inputControls.setAlignment(Pos.CENTER);

        // Create button controls
        buttonControls = new HBox(10);
```

```
        Button saveButton = new Button("Save Solution");
        saveButton.setOnAction(e -> showSaveOptions());
        Button resetViewButton = new Button("Reset View");
        resetViewButton.setOnAction(e -> resetView());
                        buttonControls.getChildren().addAll(saveButton,
    resetViewButton);
        buttonControls.setPadding(new Insets(10));
        buttonControls.setVisible(false);
        buttonControls.setAlignment(Pos.CENTER);

        // Create 3D scene container
        SubScene puzzleScene = createPuzzleScene();

        // Layout setup
        VBox topContent = new VBox(10);
        topContent.getChildren().addAll(inputControls, buttonControls);
        topContent.setAlignment(Pos.CENTER);
        root.setTop(topContent);
        root.setCenter(puzzleScene);

        // Mouse control for rotation
        puzzleScene.setOnMousePressed(me -> {
            mouseOldX = me.getSceneX();
            mouseOldY = me.getSceneY();
        });

        puzzleScene.setOnMouseDragged(me -> {
            mousePosX = me.getSceneX();
            mousePosY = me.getSceneY();
                    rotateX.setAngle(rotateX.getAngle()  -  (mousePosY  -
    mouseOldY)  / 2);
                    rotateY.setAngle(rotateY.getAngle()  +  (mousePosX  -
    mouseOldX)  / 2);
            mouseOldX = mousePosX;
            mouseOldY = mousePosY;
        });

        // Setup load button action
        loadButton.setOnAction(e -> {
            try {
                String filename = fileNameField.getText();
```

```
                if (!filename.isEmpty()) {
                    loadingIndicator.setVisible(true);
                    statusLabel.setText("Solving puzzle...");
                                Thread solverThread = new Thread(() ->
    solvePuzzle(filename));
                    solverThread.start();
                }
            } catch (Exception ex) {
                showError("Error: " + ex.getMessage());
                loadingIndicator.setVisible(false);
            }
        });

        // Scene setup
        Scene scene = new Scene(root, 1000, 800);
        primaryStage.setTitle("IQ Puzzler Pro Solver");
        primaryStage.setScene(scene);
        primaryStage.show();

        // Handle window close
        primaryStage.setOnCloseRequest(e -> {
            Platform.exit();
            System.exit(0);
        });
    }

    // Update showSaveOptions() method:
    private void showSaveOptions() {
        if (currentBoard == null) {
            showError("No solution to save!");
            return;
        }

        // Create dialog
        Stage saveDialog = new Stage();
        saveDialog.setTitle("Save Solution");
        saveDialog.initModality(Modality.APPLICATION_MODAL);

        VBox dialogVbox = new VBox(10);
        dialogVbox.setPadding(new Insets(10));
        dialogVbox.setAlignment(Pos.CENTER);
```

```
        Button txtButton = new Button("Save as TXT");
        Button imgButton = new Button("Save as PNG");

        txtButton.setOnAction(e -> {
            saveAsTxt();
            saveDialog.close();
        });

        imgButton.setOnAction(e -> {
            saveAsImage();
            saveDialog.close();
        });

        dialogVbox.getChildren().addAll(txtButton, imgButton);

        Scene dialogScene = new Scene(dialogVbox, 200, 100);
        saveDialog.setScene(dialogScene);
        saveDialog.show();
    }

    private void saveAsTxt() {
        try {
                              String  txtPath  =  "test/output/txt/"  +
      fileNameField.getText().replace(".txt", "") + ".txt";
            new File(txtPath).getParentFile().mkdirs();
            Writer writer = new Writer(txtPath);
             String message = statusLabel.getText(); // Get the current
      status message
                         writer.writeSolution(currentBoard,  solveTime,
      iterationCount, message);
                System.out.println("\nAttempted solution saved as text
      file: " + txtPath);
            showInfo("Text solution saved successfully!");
        } catch (Exception e) {
                System.out.println("\nError saving text solution: " +
      e.getMessage());
            showError("Error saving text solution: " + e.getMessage());
        }
    }
```

```
    private void saveAsImage() {
        try {
            resetView();
                                String imgPath = "test/output/png/" +
      fileNameField.getText().replace(".txt", "") + ".png";
            new File(imgPath).getParentFile().mkdirs();

                                        SubScene subScene = (SubScene)
      puzzleGroup.getScene().getRoot().lookup("#puzzleSubScene");
            WritableImage image = subScene.snapshot(null, null);
            File file = new File(imgPath);
             ImageIO.write(SwingFXUtils.fromFXImage(image, null), "png",
      file);

                    System.out.println("\nSolution saved as image: " +
      imgPath);
            showInfo("Image saved successfully!");
        } catch (Exception e) {
                        System.out.println("\nError saving image: " +
      e.getMessage());
            showError("Error saving image: " + e.getMessage());
        }
    }

    private SubScene createPuzzleScene() {
        puzzleGroup = new Group();
                   puzzleGroup.getTransforms().addAll(rotateX, rotateY,
      scaleTransform);

        // Create 3D scene
        Group root3D = new Group();
        AmbientLight ambient = new AmbientLight(Color.WHITE);

        PointLight light1 = new PointLight(Color.WHITE);
        light1.setTranslateZ(-1000);
        light1.setTranslateX(500);
        light1.setTranslateY(-500);

        PointLight light2 = new PointLight(Color.WHITE);
        light2.setTranslateZ(-1000);
        light2.setTranslateX(-500);
```

```
        light2.setTranslateY(500);

        // Initialize containers
        fileNameContainer = new VBox(5);
        statusLabel.setTextFill(Color.WHITE);
        fileNameContainer.getChildren().add(statusLabel);
        fileNameContainer.setPadding(new Insets(20));
        fileNameContainer.setVisible(true);

        solutionContainer = new VBox(5);
        timeLabel.setTextFill(Color.WHITE);
        iterationsLabel.setTextFill(Color.WHITE);
                        solutionContainer.getChildren().addAll(timeLabel,
    iterationsLabel);
        solutionContainer.setPadding(new Insets(20));
        solutionContainer.setVisible(false);

        solutionFoundLabel.setTextFill(Color.WHITE);
        solutionFoundGroup = new Group(solutionFoundLabel);
        solutionFoundGroup.setVisible(false);

        // Create separate Groups for positioning
        Group fileNameGroup = new Group(fileNameContainer);
        Group solutionStatsGroup = new Group(solutionContainer);

        // Position each text group independently
        fileNameGroup.setTranslateX(-85);
        fileNameGroup.setTranslateY(-25);

        solutionFoundGroup.setTranslateX(-45);
        solutionFoundGroup.setTranslateY(-200);

        solutionStatsGroup.setTranslateX(50);
        solutionStatsGroup.setTranslateY(125);

        // Everything to root3D
        root3D.getChildren().addAll(
            puzzleGroup,
            ambient,
            light1,
            light2,
```

```
            fileNameGroup,
            solutionFoundGroup,
            solutionStatsGroup
        );

        // Create larger subscene with black background
            SubScene subScene = new SubScene(root3D, 1000, 700, true,
    SceneAntialiasing.BALANCED);
        subScene.setId("puzzleSubScene");
        subScene.setFill(Color.rgb(30, 30, 30));

        // Setup camera
        PerspectiveCamera camera = new PerspectiveCamera(true);
        camera.setTranslateZ(-800);
        camera.setTranslateY(0);
        camera.setTranslateX(0);
        camera.setNearClip(0.1);
        camera.setFarClip(2000.0);
        camera.setFieldOfView(30);

        subScene.setCamera(camera);

        // Zoom control
        subScene.setOnScroll(event -> {
            double delta = event.getDeltaY() * 0.002;
                 scaleValue = Math.max(0.5, Math.min(2.0, scaleValue +
    delta));
            scaleTransform.setX(scaleValue);
            scaleTransform.setY(scaleValue);
            scaleTransform.setZ(scaleValue);
            event.consume();
        });

        return subScene;
    }

    private void solvePuzzle(String filename) {
        try {
            // Clear previous puzzle display first
            Platform.runLater(() -> {
                puzzleGroup.getChildren().clear();
```

```
            buttonControls.setVisible(false);
            fileNameContainer.setVisible(true);
            solutionFoundGroup.setVisible(false);
            solutionContainer.setVisible(false);
            loadingIndicator.setVisible(true);
            statusLabel.setText("Solving puzzle...");
        });

        // Process filename
        final String inputPath = filename.contains("test/input/") ?
            filename : "test/input/" + filename;
        final String fullPath = inputPath.endsWith(".txt") ?
            inputPath : inputPath + ".txt";

        // Check if file exists
        File file = new File(fullPath);
        if (!file.exists()) {
            Platform.runLater(() -> {
                statusLabel.setText("File not found: " + fullPath);
                fileNameContainer.setVisible(true);
                solutionFoundGroup.setVisible(false);
                solutionContainer.setVisible(false);
                loadingIndicator.setVisible(false);
            });
            return;
        }

        // When starting to solve
        Platform.runLater(() -> {
            currentBoard = null;
            statusLabel.setText("Solving puzzle...");
            fileNameContainer.setVisible(true);
            solutionFoundGroup.setVisible(false);
            solutionContainer.setVisible(false);
        });

        try {
            Reader reader = new Reader(fullPath);
            int[] dimensions = reader.readDimensions();
            String puzzleType = reader.readPuzzleType();
            char[][] customConfig = null;
```

```
            if (puzzleType.equals("CUSTOM")) {
                customConfig = reader.readCustomConfig();
            }
            ArrayList<Block> blocks = reader.readBlocks();

             if (blocks.isEmpty() || dimensions[2] != blocks.size())
{
                    throw new IllegalArgumentException("Invalid file
structure: Incomplete or malformed puzzle definition");
            }

            // Solve puzzle
            long startTime = System.currentTimeMillis();
                        Solver solver = new Solver(dimensions[0],
dimensions[1], blocks, customConfig, puzzleType);
            boolean hasSolution = solver.solve();
            long endTime = System.currentTimeMillis();
            final long duration = endTime - startTime;
            finalIterations = solver.getIterations();

            if (hasSolution) {
                currentBoard = solver.getBoard();
                System.out.println("\nSolution found!");
                System.out.println("\nPuzzle solution:");
                Writer.printColoredSolution(currentBoard);
                     System.out.printf("\nSearch duration: %d ms\n",
duration);
                     System.out.printf("Number of iterations: %d\n",
finalIterations);

                Platform.runLater(() -> {
                    statusLabel.setText("Solution found!");
                    solutionFoundLabel.setText("Solution found!");
                    fileNameContainer.setVisible(false);
                    solutionFoundGroup.setVisible(true);
                    solutionContainer.setVisible(true);
                    displayPuzzle();
                    updateStatus(duration, finalIterations);
                    loadingIndicator.setVisible(false);
                    buttonControls.setVisible(true);
                });
```

```
            } else {
                String message;
                switch (solver.getSolutionStatus()) {
                    case 1: // PUZZLE_BIGGER
                         message = "No solution found: Puzzle pieces
> Board space!";
                        break;
                    case 2: // BOARD_BIGGER
                         message = "No solution found: Puzzle pieces
< Board space!";
                        break;
                    default:
                            message = "No solution found: Cannot fit
pieces perfectly!";
                }

                currentBoard = solver.getBoard();
                final String finalMessage = message;

                Platform.runLater(() -> {
                    statusLabel.setText(finalMessage);
                    switch (solver.getSolutionStatus()) {
                        case 1: // PUZZLE_BIGGER
                             solutionFoundLabel.setText("No solution
found: Puzzle > Board!");
                            break;
                        case 2: // BOARD_BIGGER
                             solutionFoundLabel.setText("No solution
found: Puzzle < Board!");
                            break;
                        default:
                             solutionFoundLabel.setText("No solution
found: Cannot fit perfectly!");
                    }
                    fileNameContainer.setVisible(false);
                    solutionFoundGroup.setVisible(true);
                    solutionContainer.setVisible(true);
                    displayPuzzle();
                    updateStatus(duration, finalIterations);
                    loadingIndicator.setVisible(false);
                    buttonControls.setVisible(true);
```

```
                    });

                    System.out.println("\n" + message);
                }
                                    } catch (NumberFormatException |
    ArrayIndexOutOfBoundsException e) {
                Platform.runLater(() -> {
                    statusLabel.setText("Invalid file structure");
                    fileNameContainer.setVisible(true);
                    solutionFoundGroup.setVisible(false);
                    solutionContainer.setVisible(false);
                    loadingIndicator.setVisible(false);
                });
                System.out.println("\nInvalid file structure");
            }
        } catch (Exception e) {
            Platform.runLater(() -> {
                statusLabel.setText("Error: " + e.getMessage());
                fileNameContainer.setVisible(true);
                solutionFoundGroup.setVisible(false);
                solutionContainer.setVisible(false);
                loadingIndicator.setVisible(false);
            });
            System.out.println("\nError: " + e.getMessage());
        }
    }

    private void displayPuzzle() {
        puzzleGroup.getChildren().clear();
        addedConnections.clear();

        if (currentBoard == null) return;

        char[][] grid = currentBoard.getGrid();
        double spacing = SPHERE_RADIUS * 2.5;

        // Create board components with updated offsets
        double xOffset = 25.0;
        double yOffset = 15.0;

        // Create board group
```

```
        Group boardGroup = createBoardGroup(grid, spacing, xOffset,
    yOffset);
      puzzleGroup.getChildren().add(boardGroup);

      // Create spheres with connections
                    Map<Character, List<Sphere>> pieceSpheres =
    createPieceSpheres(grid, spacing, xOffset, yOffset);
            createConnections(grid, spacing, pieceSpheres, xOffset,
    yOffset);

      // Update view
      resetView();
      buttonControls.setVisible(true);
  }

   private Group createBoardGroup(char[][] grid, double spacing,
    double xOffset, double yOffset) {
      double boardWidth = grid[0].length * spacing * 1.2;
      double boardHeight = grid.length * spacing * 1.2;
      double boardDepth = spacing * 0.5;
      double curveRadius = SPHERE_RADIUS * 1.2;

      // Create main board
      Box mainBoard = new Box(boardWidth, boardHeight, boardDepth);
        PhongMaterial boardMaterial = new PhongMaterial(Color.rgb(40,
    40, 40));
      mainBoard.setMaterial(boardMaterial);
      mainBoard.setTranslateZ(-boardDepth/2);

      // Create indentations
       Group indentations = createIndentations(grid, spacing, xOffset,
    yOffset);

      // Create curved edge
          Cylinder curve = createCurvedEdge(curveRadius, boardWidth,
    boardHeight);

      Group boardGroup = new Group(mainBoard, indentations, curve);
      boardGroup.setTranslateZ(-spacing * 1.5);

      return boardGroup;
```

```
    }

    private Group createIndentations(char[][] grid, double spacing,
    double xOffset, double yOffset) {
      Group indentations = new Group();
      for (int row = 0; row < grid.length; row++) {
          for (int col = 0; col < grid[0].length; col++) {
              if (grid[row][col] != '.') {
                    double x = (col - grid[0].length/2.0) * spacing +
    xOffset;
                            double y = ((grid.length - 1 - row) -
    grid.length/2.0) * spacing + yOffset;

                Sphere indent = new Sphere(SPHERE_RADIUS * 0.9);
                                PhongMaterial indentMaterial = new
    PhongMaterial(Color.rgb(30, 30, 30));
                indent.setMaterial(indentMaterial);
                indent.setTranslateX(x);
                indent.setTranslateY(y);
                indent.setTranslateZ(SPHERE_RADIUS/2);
                indentations.getChildren().add(indent);
              }
          }
      }
      return indentations;
    }

     private Cylinder createCurvedEdge(double radius, double width,
    double height) {
      Cylinder curve = new Cylinder(radius, width);
      curve.setRotate(90);
      curve.setTranslateY(height/2);
      curve.setTranslateZ(-radius);
        PhongMaterial curveMaterial = new PhongMaterial(Color.rgb(50,
    50, 50));
      curve.setMaterial(curveMaterial);
      return curve;
    }

    private Map<Character, List<Sphere>> createPieceSpheres(char[][]
    grid, double spacing, double xOffset, double yOffset) {
```

```
        Map<Character, List<Sphere>> pieceSpheres = new HashMap<>();

        for (int row = grid.length - 1; row >= 0; row--) {
            for (int col = 0; col < grid[0].length; col++) {
                if (grid[row][col] != '.') {
                    Sphere sphere = new Sphere(SPHERE_RADIUS * 0.9);
                    PhongMaterial material = new PhongMaterial();

    material.setDiffuseColor(PIECE_COLORS[grid[row][col] - 'A']);
                    material.setSpecularColor(Color.WHITE);
                    material.setSpecularPower(32);
                    sphere.setMaterial(material);

                      double x = (col - grid[0].length/2.0) * spacing +
    xOffset;
                               double y = ((grid.length - 1 - row) -
    grid.length/2.0) * spacing + yOffset;
                    sphere.setTranslateX(x);
                    sphere.setTranslateY(y);
                    sphere.setTranslateZ(0);

                    puzzleGroup.getChildren().add(sphere);
                      pieceSpheres.computeIfAbsent(grid[row][col], k ->
    new ArrayList<>()).add(sphere);
                }
            }
        }
        return pieceSpheres;
    }

     private void createConnections(char[][] grid, double spacing,
    Map<Character, List<Sphere>> pieceSpheres,
                                   double xOffset, double yOffset) {
        for (int row = 0; row < grid.length; row++) {
            for (int col = 0; col < grid[0].length; col++) {
                if (grid[row][col] != '.') {
                    // Check right and down connections
                     checkAndCreateConnection(grid, row, col, row, col +
    1, spacing, pieceSpheres, xOffset, yOffset);
                      checkAndCreateConnection(grid, row, col, row + 1,
    col, spacing, pieceSpheres, xOffset, yOffset);
```

```
                }
            }
        }
    }

    private void checkAndCreateConnection(char[][] grid, int row1, int
     col1, int row2, int col2,
                                          double spacing, Map<Character,
     List<Sphere>> pieceSpheres,
                                         double xOffset, double yOffset)
     {
        if (row2 >= 0 && row2 < grid.length && col2 >= 0 && col2 <
     grid[0].length &&
            grid[row1][col1] == grid[row2][col2]) {

            String connectionKey = String.format("%d,%d-%d,%d", row1,
     col1, row2, col2);
            if (!addedConnections.contains(connectionKey)) {
                createConnection(row1, col1, row2, col2,
     grid[row1][col1], spacing, xOffset, yOffset);
                addedConnections.add(connectionKey);
            }
        }
    }

    private void createConnection(int row1, int col1, int row2, int
     col2, char pieceId,
                                  double spacing, double xOffset, double
     yOffset) {
        double x1 = (col1 - currentBoard.getGrid()[0].length/2.0) *
     spacing + xOffset;
        double y1 = ((currentBoard.getGrid().length - 1 - row1) -
     currentBoard.getGrid().length/2.0) * spacing + yOffset;
        double x2 = (col2 - currentBoard.getGrid()[0].length/2.0) *
     spacing + xOffset;
        double y2 = ((currentBoard.getGrid().length - 1 - row2) -
     currentBoard.getGrid().length/2.0) * spacing + yOffset;

        Cylinder connection = new Cylinder(SPHERE_RADIUS * 0.4,
            Math.sqrt(Math.pow(x2-x1, 2) + Math.pow(y2-y1, 2)));
```

```
        // Position and rotate the connection
        double angle = Math.toDegrees(Math.atan2(y2-y1, x2-x1));
        connection.getTransforms().addAll(
            new Rotate(90, Rotate.Z_AXIS),
            new Rotate(-angle, Rotate.Z_AXIS)
        );

        connection.setTranslateX((x1 + x2)/2);
        connection.setTranslateY((y1 + y2)/2);

        PhongMaterial material = new PhongMaterial();
        material.setDiffuseColor(PIECE_COLORS[pieceId - 'A']);
        connection.setMaterial(material);

        puzzleGroup.getChildren().add(connection);
    }

    private void resetView() {
        rotateX.setAngle(180);
        rotateY.setAngle(0);
        scaleValue = 1.0;
        scaleTransform.setX(scaleValue);
        scaleTransform.setY(scaleValue);
        scaleTransform.setZ(scaleValue);
    }

    private void showInfo(String message) {
        Alert alert = new Alert(Alert.AlertType.INFORMATION);
        alert.setTitle("Information");
        alert.setHeaderText(null);
        alert.setContentText(message);
        alert.showAndWait();
    }

    private void updateStatus(long duration, long iterations) {
        this.solveTime = duration;
        this.iterationCount = iterations;
        Platform.runLater(() -> {
              timeLabel.setText(String.format("Search duration: %d ms",
      duration));
```

```
                        iterationsLabel.setText(String.format("Number  of
      iterations: %d", iterations));

            // Ensure labels are visible
            statusLabel.setVisible(true);
            timeLabel.setVisible(true);
            iterationsLabel.setVisible(true);
        });
    }

    private void showError(String message) {
        Alert alert = new Alert(Alert.AlertType.ERROR);
        alert.setTitle("Error");
        alert.setHeaderText(null);
        alert.setContentText(message);
        alert.showAndWait();
    }

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

2.8 Hasil Eksperimen dan Analisis

2.7.1 Test Case 1 (Default)

a. Input

```
5 5 7
DEFAULT
A
AA
B
BB
C
CC
D
```

```
DD
EE
EE
E
FF
FF
F
GGG
```

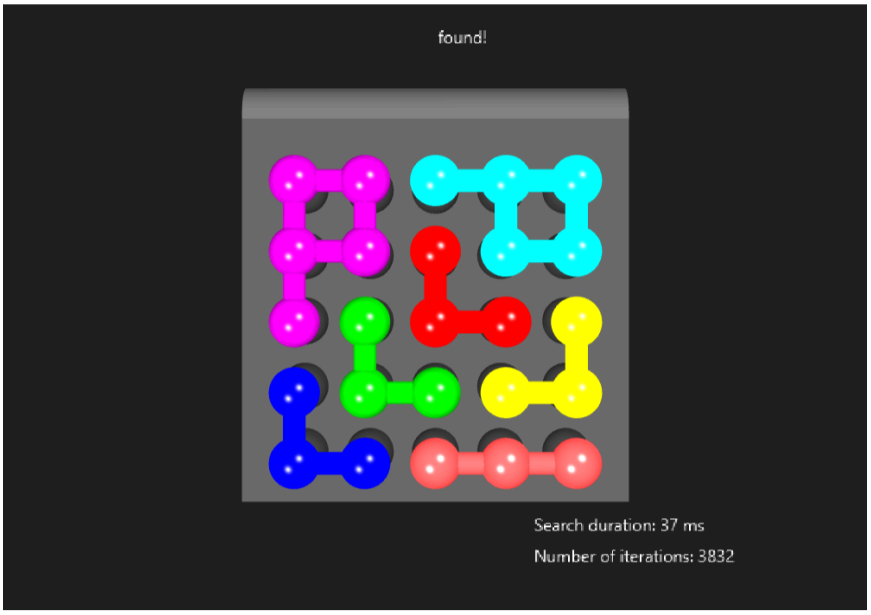
b. Output

1. TXT

```
EEFFF
EEAFF
EBAAC
DBBCC
DDGGG

Search duration: 37 ms
Number of iterations: 3832
```

2. PNG



Gambar 2.7.1 Output Test Case 1

2.7.2 Test Case 2 (Default)

a. Input

```
5 11 12
DEFAULT
AA
A
AA
BB
BB
B
C
C
C
DD
DD
EE
EE
E
FF
FF
F
F
G
G
GGG
H
HH
H
II
III
I
J
JJ
K
```

```
KK
K
K
LLLL
```

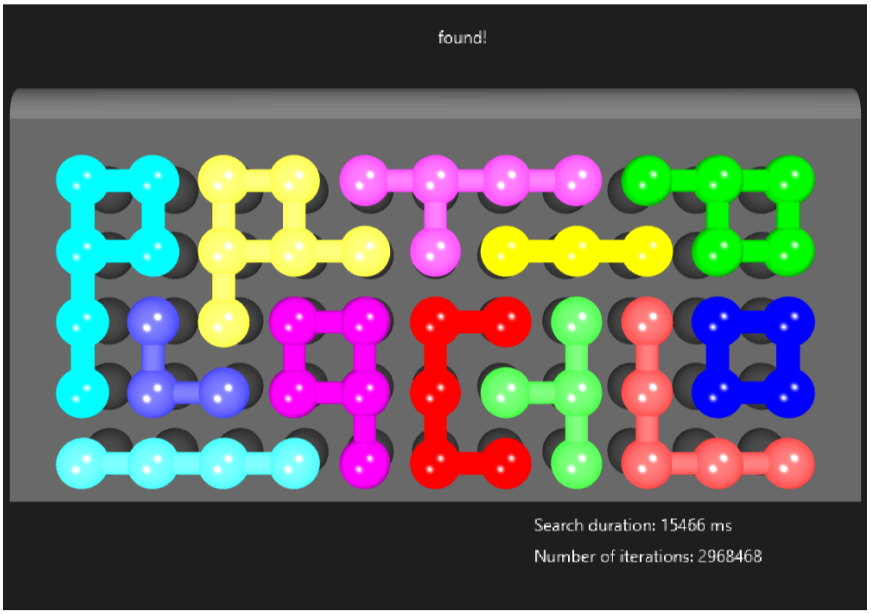
b. Output

1. TXT

```
FFIIKKKKBBB
FFIIIKCCCBB
FJIEEAAHGDD
FJJEEAHHGDD
LLLLEAAHGGG

Search duration: 15466 ms
Number of iterations: 2968468
```

2. PNG



Gambar 2.7.2 Output Test Case 2

2.7.3 Test Case 3 (Default)

a. Input

```
5 10 10
DEFAULT
A
A
AA
AA
 B
BBB
CCCC
C
DD
DDD
EEEE
 E
F
F
FF
F
GGGG
  GG
H
H
H
HH
I
II
I
J J
JJJ
```

b. Output

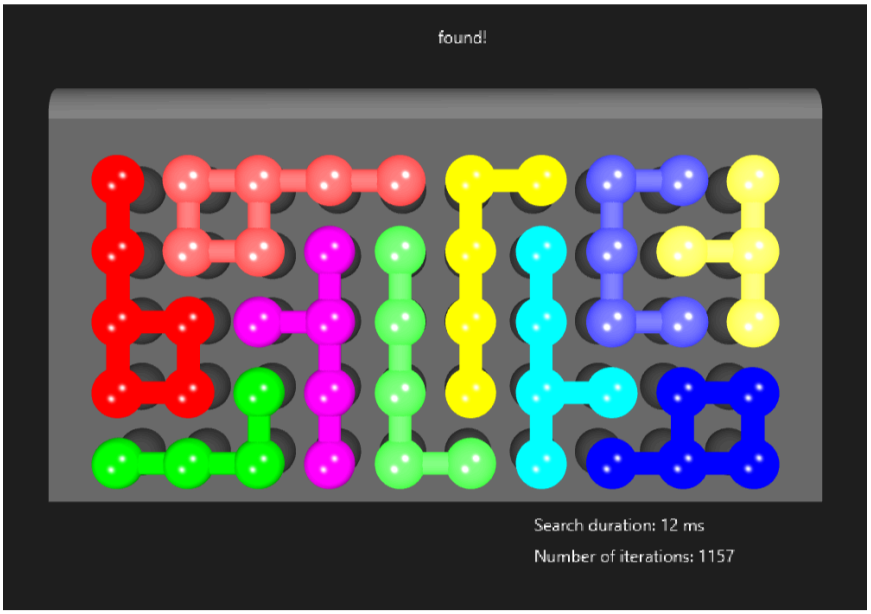
1. TXT

```
AGGGGCCJJI
AGGEHCFJII
AAEEHCFJJI
AABEHCFFDD
```

```
BBBEHHFDDD

Solution found!
Search duration: 12 ms
Number of iterations: 1157
```

2. PNG



Gambar 2.7.3 Output Test Case 3

2.7.4 Test Case 4 (Custom)

a. Input

```
5 7 5
CUSTOM
...X...
.XXXXX.
XXXXXXX
.XXXXX.
...X...
A
AAA
BB
```

```
BBB
CCCC
 C
D
EEE
E
```

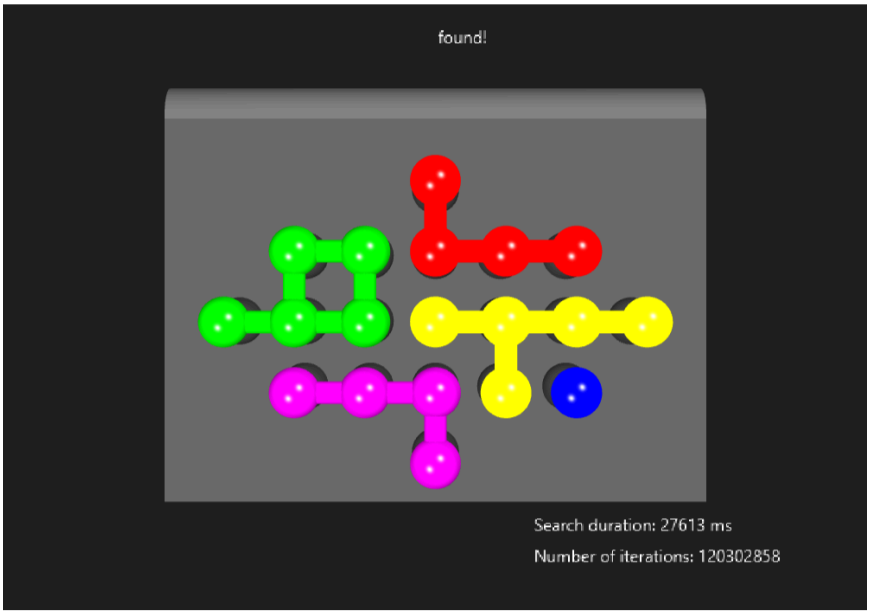
b. Output

1. TXT

```
...A...
.BBAAA.
BBBCCCC
.EEECD.
...E...

Search duration: 27613 ms
Number of iterations: 120302858
```

2. PNG



Gambar 2.7.4 Output Test Case 4

2.7.5 Test Case 5 (Custom)

a. Input

```
3 3 3
CUSTOM
XXX
.X.
XXX
A
AA
B
CCC
```

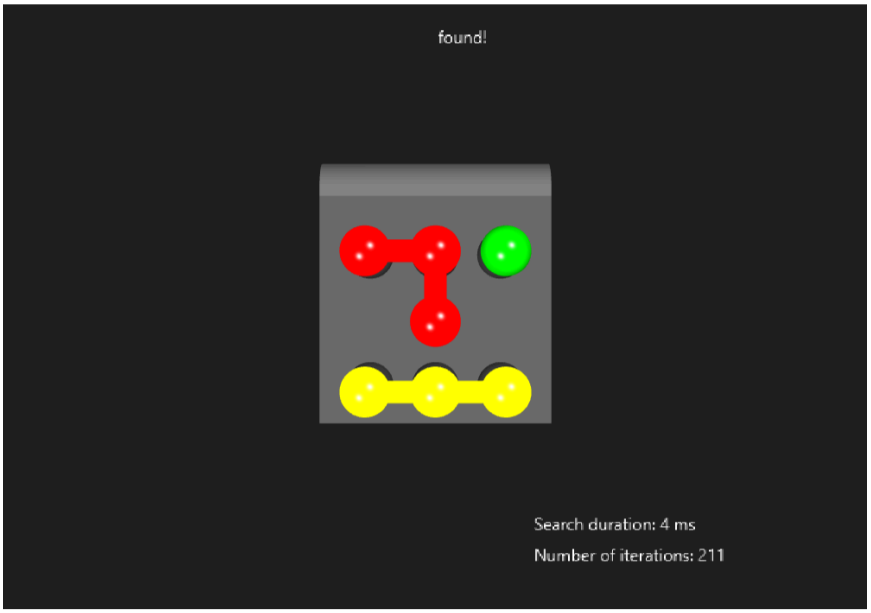
b. Output

1. TXT

```
AAB
.A.
CCC

Solution found!
Search duration: 4 ms
Number of iterations: 211
```

2. PNG



Gambar 2.7.5 Output Test Case 5

2.7.6 Test Case 6 (Custom)

a. Input

```
5 5 6
CUSTOM
XXXXX
X...X
XXXXX
X...X
XXXXX
A
BBBB
B
CCC
  C
  C
 D
DDDD
E
F
F
```

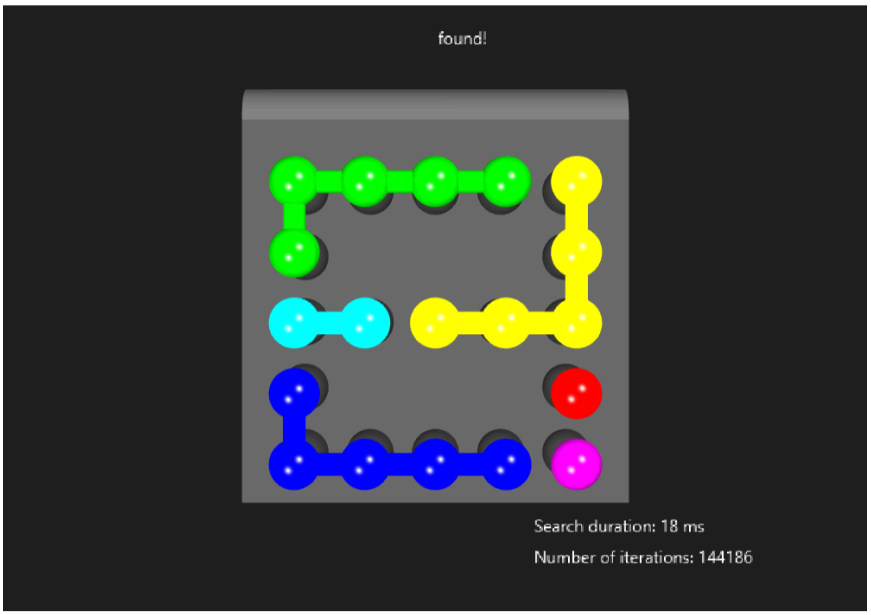
b. Output

1. TXT

```
BBBBC
B...C
FFCCC
D...A
DDDDE

Solution found!
Search duration: 18 ms
Number of iterations: 144186
```

2. PNG



Gambar 2.7.6 Output Test Case 6

2.7.7 Test Case 7 (Not Found)

a. Input

```
2 2 1
DEFAULT
```

```
A
```

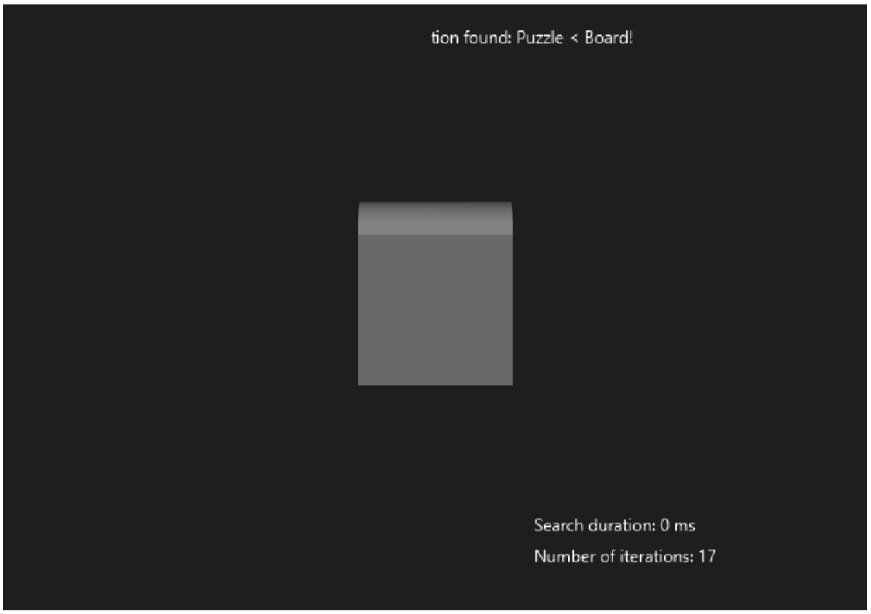
b. Output

1. TXT

```
..
..

No solution found: Puzzle pieces < Board space!
Search duration: 0 ms
Number of iterations: 17
```

2. PNG



Gambar 2.7.7 Output Test Case 7

2.7.8 Test Case 8 (Not Found)

a. Input

```
2 2 3
DEFAULT
A
```

```
B
BB
C
```

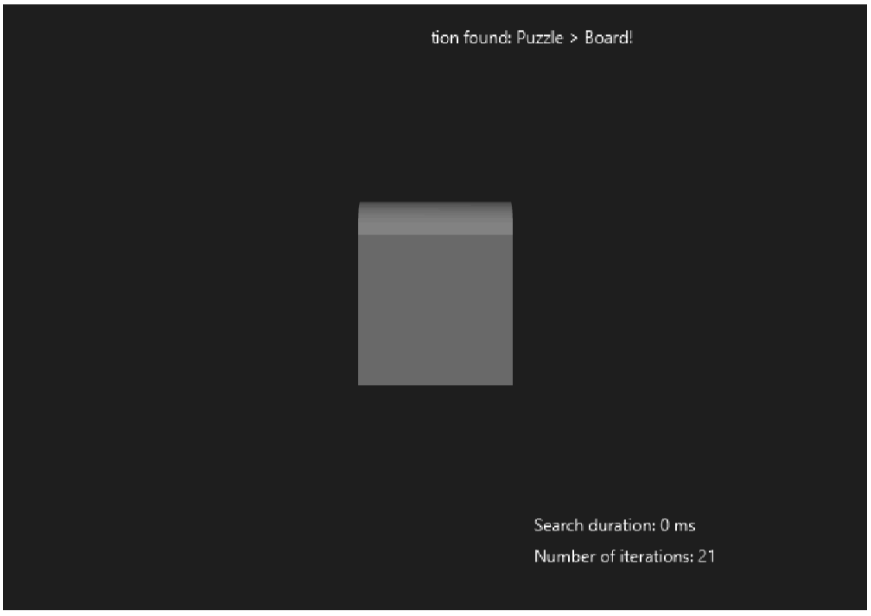
b. Output

1. TXT

```
..
..

No solution found: Puzzle pieces > Board space!
Search duration: 0 ms
Number of iterations: 21
```

2. PNG



Gambar 2.7.8 Output Test Case 8

2.7.9 Test Case 9 (Not Found)

a. Input

```
5 5 4
```

```
CUSTOM
..X..
.XXX.
XXXXX
.XXX.
..X..
A
AA
BB
 C
CCC
DD
DD
```

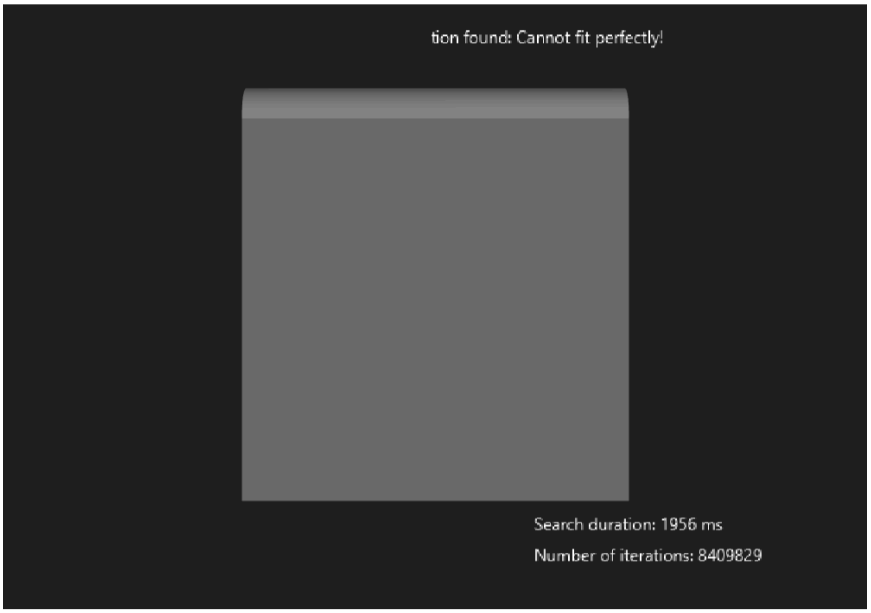
b. Output

1. TXT

```
.....
.....
.....
.....
.....

No solution found: Cannot fit pieces perfectly!
Search duration: 1956 ms
Number of iterations: 8409829
```

2. PNG



Gambar 2.7.9 Output Test Case 9

BAB III
Penutup

3.1 Kesimpulan

Implementasi solver untuk permainan IQ Puzzler Pro telah berhasil dikembangkan dengan fitur-fitur berikut:

1. **Algoritma Penyelesaian**
    - Menggunakan pendekatan Brute Force dengan optimasi Backtracking
    - Mendukung dua mode permainan: DEFAULT dan CUSTOM
    - Mampu mendeteksi berbagai kasus tidak ada solusi
2. **Antarmuka Pengguna**
    - Visualisasi 3D interaktif menggunakan JavaFX
    - Kontrol kamera (rotasi dan zoom) yang intuitif
    - Tampilan status dan statistik yang informatif
    - Sistem penyimpanan solusi dalam format teks dan gambar
3. **Performa**
    - Mampu menyelesaikan puzzle sederhana dalam hitungan milidetik
    - Dapat menangani puzzle kompleks dengan jutaan iterasi
    - Memberikan feedback real-time tentang proses penyelesaian

3.2 Saran

Tugas kecil ini telah berhasil mencapai tujuannya dalam mengimplementasikan solver untuk IQ Puzzler Pro dengan antarmuka yang user-friendly dan performa yang baik. Pengembangan lebih lanjut dapat fokus pada optimasi dan penambahan fitur untuk meningkatkan kegunaan dan pengalaman pengguna.

1. Optimasi Algoritma
    - Implementasi parallel processing untuk meningkatkan kecepatan

- Penambahan heuristik untuk mengurangi ruang pencarian
- Optimasi penggunaan memori untuk puzzle berukuran besar

2. Peningkatan Fitur
    - Penambahan animasi proses penyelesaian
    - Implementasi undo/redo untuk manipulasi puzzle
    - Fitur pembangkit puzzle acak
    - Mode interaktif untuk menempatkan blok secara manual
3. Penyempurnaan UI/UX
    - Penambahan tema gelap/terang
    - Customizable warna blok puzzle
    - Peningkatan responsivitas antarmuka
    - Tutorial atau panduan penggunaan terintegrasi
4. Pengembangan Teknis
    - Implementasi caching untuk meningkatkan performa
    - Penambahan unit testing untuk menjamin kualitas kode
    - Dokumentasi kode yang lebih komprehensif
    - Optimasi penggunaan resource sistem

Referensi

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to algorithms* (3rd ed.). MIT Press.

Vos, J. (2014). *Deep dive into JavaFX 3D*. Oracle Technology Network. https://www.oracle.com/technical-resources/articles/java/deep-dive-javafx-3d.html

Oracle. (2023). *JavaFX documentation*. Oracle. https://docs.oracle.com/en/java/javase/17/javafx/index.html

Dechter, R. (2003). *Constraint processing*. Morgan Kaufmann.

Russell, S., & Norvig, P. (2020). *Artificial intelligence: A modern approach* (4th ed.). Pearson.