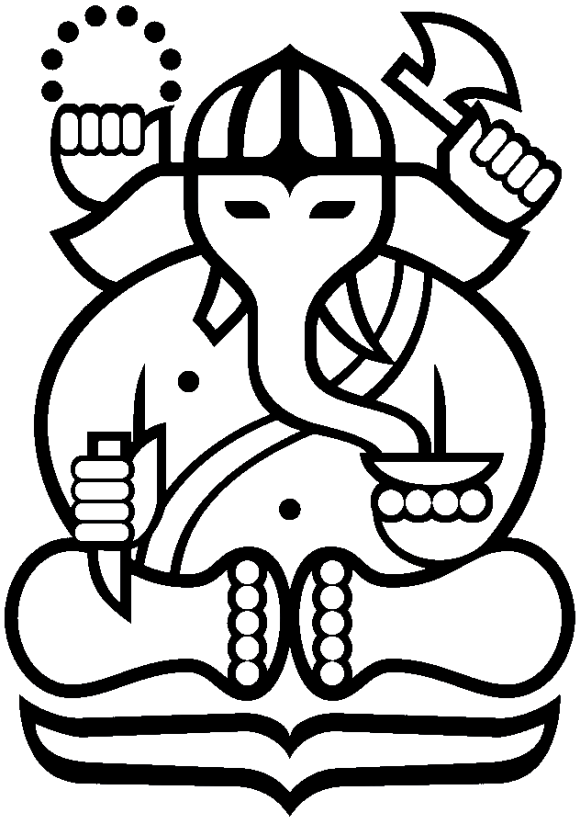
Laporan Tugas Kecil Penyelesaian IQ Puzzler Pro dengan Algoritma Brute Force

Tugas Kecil 1 - IF2211 Strategi Algoritma Semester II Tahun 2024/2025



Disusun oleh

Frederiko Eldad Mugiyono - 13523147

Daftar Isi

Daftar Gambar

1. Pendahuluan

Algoritma **brute force** merupakan salah satu metode dasar dalam penyelesaian masalah komputasi, di mana solusi ditemukan dengan mencoba semua kemungkinan hingga diperoleh hasil yang benar. Meskipun seringkali kurang efisien dibandingkan metode lain, pendekatan ini tetap penting karena mudah dipahami dan dapat diterapkan dalam berbagai kasus.

Laporan ini disusun sebagai bentuk dokumentasi dari pengerjaan tugas dalam mata kuliah **IF2211 Strategi Algoritma**, yang bertujuan untuk mengimplementasikan dan menganalisis algoritma brute force dalam menyelesaikan suatu permasalahan. Melalui tugas ini, telah dilakukan eksplorasi terhadap konsep brute force, penerapannya dalam kode program, serta pengujian untuk mengevaluasi kinerjanya. Laporan ini juga berisi pembahasan mengenai langkah-langkah algoritma yang digunakan, implementasi program, serta hasil pengujian dengan berbagai skenario input.

2. Penjelasan Algoritma Brute Force

Algoritma Brute Force digunakan dalam tugas ini untuk menyelesaikan permasalahan penyusunan kepingan puzzle pada papan permainan. Algoritma ini bekerja dengan mencoba semua kemungkinan penempatan kepingan pada papan hingga menemukan solusi yang valid.

Fungsi utama dari algoritma ini adalah mencari solusi dengan menempatkan kepingan secara berulang melalui eksplorasi posisi, rotasi, dan flipping, serta melakukan backtracking ketika percobaan penempatan tidak menghasilkan solusi yang valid.

2.1. Inisialisasi Algoritma

- Kelas **BruteForceAlgorithm** menerima objek **Board** sebagai papan permainan dan daftar **Piece** sebagai kumpulan kepingan puzzle yang akan dicoba untuk ditempatkan.
- Variabel **visited** digunakan untuk menghitung jumlah kemungkinan yang telah diuji selama pencarian solusi.

Potongan kode dari kelas BruteForceAlgorithm

```
public class BruteForceAlgorithm {
    private Board board;
    private List<Piece> pieces;
    public long visited = 0; // Counter

    public BruteForceAlgorithm(Board board, List<Piece> pieces)
{
        this.board = board;
        this.pieces = pieces;
    }
…
```

2.2. Fungsi solve(int pieceIndex)

- **Basis Rekursi**: Jika semua kepingan sudah dipasang (**pieceIndex == pieces.size()**) dan solusi valid (**board.isSolutionValid()**), maka solusi ditemukan.
- **Eksplorasi Kemungkinan**: Untuk setiap kepingan puzzle:
    - Dicoba dengan dua orientasi: **tanpa flipping** dan **dengan flipping secara vertikal**.
    - Dicoba dalam **4 kemungkinan rotasi** (0°, 90°, 180°, 270°).
    - Dicoba ditempatkan pada setiap posisi di papan permainan.

- Jika berhasil ditempatkan, fungsi dipanggil secara rekursif untuk kepingan berikutnya (**solve(pieceIndex + 1)**).
- Jika solusi tidak ditemukan, **backtracking** dilakukan dengan menghapus kepingan tersebut.

Kode dari fungsi solve(int pieceIndex)

```
public boolean solve(int pieceIndex) {
        if (pieceIndex >= pieces.size()) {
            return true; // Ini bisa
        }

        Piece piece = pieces.get(pieceIndex);

        // Flip vertical
        for (int flip = 0; flip < 2; flip++){
            // Rotasi piece (0°, 90°, 180°, 270°)
            for (int rotation = 0; rotation < 4; rotation++) {
                char[][] shape = piece.getShapeArray();

                // Nyoba masang ya ges ya
                for (int row = 0; row < board.getHeight();
row++) {
                    for (int col = 0; col < board.getWidth();
col++) {
                        visited++; // Tambah 1

                        if (canPlacePiece(shape, row, col)) {
                            placePiece(shape, row, col,
piece.getId());

                            if (solve(pieceIndex + 1)) {
                                return true; // Bisa gan
                            }

                            removePiece(shape, row, col); //
Backtrack
                        }
                    }
                }
                piece = piece.rotate90();
            }
            piece = piece.flipVertical();
        }

        return false; // Udah gabisa
    }
```

2.3. Fungsi canPlacePiece(char[][] shape, int startRow, int startCol)

- Memeriksa apakah kepingan dapat ditempatkan di posisi (**startRow, startCol**).
- Mengecek apakah kepingan melampaui batas papan permainan.
- Memastikan bahwa bagian kepingan tidak menimpa kepingan lain di papan.

Kode dari fungsi canPlacePiece(char[][] shape, int startRow, int startCol)

```
private boolean canPlacePiece(char[][] shape, int startRow, int
startCol) {
        int shapeHeight = shape.length;
        int shapeWidth = shape[0].length;

        if (startRow + shapeHeight > board.getHeight() ||
startCol + shapeWidth > board.getWidth()) {
            return false; // Nembus wak
        }

        for (int i = 0; i < shapeHeight; i++) {
            for (int j = 0; j < shapeWidth; j++) {
                if (shape[i][j] != ' ' &&
board.getGrid()[startRow + i][startCol + j] != 'X') {
                    return false; // Nempel wak
                }
            }
        }
        return true;
    }
```

2.4. Fungsi placePiece(char[][] shape, int startRow, int startCol, char pieceId)

- Menempatkan kepingan di papan dengan mengisi sel yang sesuai dengan **ID kepingan**.

Kode dari fungsi placePiece(char[][] shape, int startRow, int startCol, char pieceId)

```
private void placePiece(char[][] shape, int startRow, int
```

```
startCol, char pieceId) {
        for (int i = 0; i < shape.length; i++) {
            for (int j = 0; j < shape[0].length; j++) {
                if (shape[i][j] != ' ') {
                    board.getGrid()[startRow + i][startCol + j]
= pieceId;
                }
            }
        }
    }
```

2.5. Fungsi removePiece(char[][] shape, int startRow, int startCol)

- Menghapus kepingan dari papan untuk **melakukan backtracking** saat percobaan gagal.

Kode dari removePiece(char[][] shape, int startRow, int startCol)

```
private void removePiece(char[][] shape, int startRow, int
startCol) {
        for (int i = 0; i < shape.length; i++) {
            for (int j = 0; j < shape[0].length; j++) {
                if (shape[i][j] != ' ') {
                    board.getGrid()[startRow + i][startCol + j]
= 'X'; // Reset to X (empty)
                }
            }
        }
    }
```

2.6. Kompleksitas Algoritma

- Kompleksitas waktu mendekati **O(n!)** dalam kasus terburuk karena mencoba semua kemungkinan penempatan.

3. Source Code

Bagian ini menjelaskan struktur proyek, kode utama yang mengandung logika penyelesaian puzzle menggunakan algoritma brute force, serta implementasi antarmuka pengguna.

3.1. Struktur Proyek

Proyek ini memiliki struktur direktori dan file sebagai berikut:

```
Tucil1_13523147/
├── bin/
│   └── IQPuzzlePro.jar
├── doc/
├── src/
│   ├── BruteForceAlgorithm.java
│   ├── Board.java
│   ├── Piece.java
│   ├── Puzzle.java
│   └── Main.java
├── test/
└── README.md
```

Penjelasan masing-masing direktori dan file:

- **bin/**: Berisi file JAR hasil kompilasi, yaitu IQPuzzlePro.jar, yang dapat dieksekusi untuk menjalankan aplikasi.
- **doc/**: Direktori untuk dokumentasi proyek dalam bentuk **.pdf**.
- **src/**: Berisi kode sumber utama proyek.
    - **BruteForceAlgorithm.java**: Implementasi algoritma brute force untuk menyelesaikan puzzle.
    - **Board.java**: Representasi papan permainan dan operasi terkait.
    - **Piece.java**: Representasi kepingan puzzle beserta rotasi dan flipping.
    - **Puzzle.java**: Parser untuk membaca data kepingan dari file input.
    - **Main.java**: Entry point program untuk menjalankan solver dan antarmuka pengguna.
- **test/**: Direktori untuk file uji coba dalam bentuk **.txt** dan hasil dalam bentuk **.txt** dan **.png**.
- **README.md**: File readme yang mencakup deskripsi, persyaratan, instalasi, dan cara penggunaan aplikasi.

3.2. Main.java

Main.java merupakan entry point dari aplikasi yang bertanggung jawab untuk membaca input, memprosesnya menggunakan algoritma brute force, dan menampilkan hasil penyelesaian puzzle. Berikut adalah penjelasan struktur dan fungsionalitas utama dalam **Main.java**:

3.2.1. Membaca dan Menyiapkan Input

Program dimulai dengan membaca file berekstensi .txt dari pengguna yang berisi representasi kepingan puzzle. Data ini kemudian diolah untuk membentuk objek-objek **Piece** yang akan digunakan dalam algoritma penyelesaian.

3.2.2. Inisialisasi Papan Permainan

Objek **Board** dibuat untuk merepresentasikan papan permainan tempat kepingan puzzle akan ditempatkan. Ukuran papan dapat dikonfigurasi berdasarkan kebutuhan.

3.2.3. Menjalankan Algoritma Brute Force

Objek **BruteForceAlgorithm** dibuat dengan parameter papan permainan dan daftar kepingan puzzle. Algoritma ini mencoba setiap kemungkinan peletakan kepingan dengan mencakup:

- Rotasi (0°, 90°, 180°, 270°)
- Pembalikan secara vertikal
- Pencarian posisi yang memungkinkan di papan

3.2.4. Menampilkan dan Menyimpan Hasil

Setelah algoritma dijalankan, program menampilkan hasil penyelesaian, baik dalam bentuk visualisasi papan permainan maupun statistik seperti jumlah kemungkinan yang diuji dan waktu eksekusi dalam milidetik. Pengguna juga dapat menyimpan hasil tersebut ke dalam file **.txt** untuk dokumentasi atau **.png** sebagai gambar visualisasi solusi.

3.2.5. Penanganan Kesalahan

Jika file input tidak ditemukan atau tidak valid, program akan menampilkan pesan kesalahan yang sesuai agar pengguna dapat memperbaiki inputnya.

Kode dari Main

```
import javafx.application.Application;
import javafx.embed.swing.SwingFXUtils;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.BorderPane;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.paint.Color;
import javafx.scene.control.Alert;
import javafx.scene.control.Button;
import javafx.scene.control.TextArea;
import javafx.scene.control.ToolBar;
import javafx.scene.image.WritableImage;
import javafx.stage.FileChooser;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

import javax.imageio.ImageIO;

public class Main extends Application {
    private TextArea boardDisplay;
    private Board board;
    private List<Piece> pieces;
    private Canvas canvas;
    private static File lastDirectory = null;

    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("IQ Puzzle Pro Solver");

        BorderPane root = new BorderPane();
        canvas = new Canvas(640, 340);
        canvas.setVisible(false);

        boardDisplay = new TextArea();
        boardDisplay.setEditable(false);
        boardDisplay.setStyle("-fx-control-inner-background:
white; -fx-text-fill: black; -fx-font-family: 'Arial';");
        boardDisplay.setPrefHeight(100);
        boardDisplay.setMaxHeight(100);
```

```
        boardDisplay.setMinHeight(100);
        root.setCenter(canvas);
        root.setBottom(boardDisplay);

        Button loadFileButton = new Button("Load Puzzle File");
        Button saveButton = new Button("Export Solution");

        loadFileButton.setOnAction(e ->
loadPuzzleFile(primaryStage));
        saveButton.setOnAction(e -> saveFile(primaryStage,
canvas));
        loadFileButton.setStyle("-fx-text-fill: black;
-fx-font-family: 'Arial';");
        saveButton.setStyle("-fx-text-fill: black;
-fx-font-family: 'Arial';");

        ToolBar toolBar = new ToolBar(loadFileButton,
saveButton);
        root.setTop(toolBar);

        Scene scene = new Scene(root, 640, 480, Color.WHITE);
        primaryStage.setScene(scene);
        primaryStage.show();

        GraphicsContext gc = canvas.getGraphicsContext2D();
        gc.setFill(Color.WHITE);
        gc.fillRect(0, 0, canvas.getWidth(),
canvas.getHeight());
    }


    private void loadPuzzleFile(Stage stage) {
        FileChooser fileChooser = new FileChooser();
        fileChooser.setTitle("Open Puzzle File");

        if (lastDirectory != null && lastDirectory.exists()) {
            fileChooser.setInitialDirectory(lastDirectory);
        }

        fileChooser.getExtensionFilters().add(new
FileChooser.ExtensionFilter("Text Files", "*.txt"));

        File file = fileChooser.showOpenDialog(stage);

        if (file != null) {
            clearBoard();
            lastDirectory = file.getParentFile();
            try (Scanner scanner = new Scanner(file)) {
```

```
                int height = scanner.nextInt();
                int width = scanner.nextInt();
                int P = scanner.nextInt();
                scanner.nextLine();
                scanner.nextLine();

                board = new Board();
                board.genBoard(height, width);

                List<String> inputLines = new ArrayList<>();
                while (scanner.hasNextLine()) {
                    inputLines.add(scanner.nextLine());
                }

                pieces = Puzzle.parsePieces(inputLines);
                if (!Puzzle.isValid(pieces, P)) {
                    boardDisplay.setText(Puzzle.errorMsg(pieces,
P));
                    return;
                }
                solvePuzzle();
            } catch (FileNotFoundException e) {
                boardDisplay.setText("File not found!");
            }
        }
    }

    private void saveFile(Stage stage, Canvas canvas) {
        FileChooser fileChooser = new FileChooser();
        fileChooser.setTitle("Save Solution File");

        if (lastDirectory != null && lastDirectory.exists()) {
            fileChooser.setInitialDirectory(lastDirectory);
        }

        FileChooser.ExtensionFilter pngFilter = new
FileChooser.ExtensionFilter("PNG Image (*.png)", "*.png");
        FileChooser.ExtensionFilter txtFilter = new
FileChooser.ExtensionFilter("Text File (*.txt)", "*.txt");
        fileChooser.getExtensionFilters().addAll(pngFilter,
txtFilter);

        File file = fileChooser.showSaveDialog(stage);

        if (file != null) {
            lastDirectory = file.getParentFile();
            String extension = getFileExtension(file);
```

```
            if ("png".equalsIgnoreCase(extension)) {
                saveToPng(canvas, file);
            }
            else if ("txt".equalsIgnoreCase(extension)) {
                saveToText(board, file);
            }
            else {
                showError("Unsupported file type! Please choose
PNG or TXT.");
            }
        }
    }

    private void saveToPng(Canvas canvas, File file) {
        WritableImage image = new WritableImage((int)
canvas.getWidth(), (int) canvas.getHeight());
        canvas.snapshot(null, image);
        try {
            ImageIO.write(SwingFXUtils.fromFXImage(image, null),
"png", file);
        }
        catch (IOException ex) {
            ex.printStackTrace();
        }
    }

    private void saveToText(Board board, File file) {
        String content = formatBoard(board);
        try {
            Files.write(Paths.get(file.toURI()),
content.getBytes());
        }
        catch (IOException ex){
            ex.printStackTrace();
        }
    }

    private String getFileExtension(File file) {
        String name = file.getName();
        int lastDot = name.lastIndexOf('.');
        return (lastDot > 0) ? name.substring(lastDot + 1) : "";
    }

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

```
        alert.showAndWait();
    }

    private void solvePuzzle() {
        BruteForceAlgorithm solver = new
BruteForceAlgorithm(board, pieces);
        long startTime = System.currentTimeMillis();
        boolean solutionFound = solver.solve(0);
        long endTime = System.currentTimeMillis();
        long elapsedTimeMs = (endTime - startTime);

        if (solutionFound && board.isSolutionValid()) {
            boardDisplay.setText("Solution found.");
            drawBoard();
        } else {
            boardDisplay.setText("No solution found.");
            drawNoSolutionBoard();
        }
        boardDisplay.appendText("\nTotal possibilities visited:
" + solver.visited);
        boardDisplay.appendText("\nTime taken: " + elapsedTimeMs
+ " ms");
    }

    private String formatBoard(Board board) {
        StringBuilder sb = new StringBuilder();
        for (char[] row : board.getGrid()) {
            for (char cell : row) {
                sb.append(cell).append(" ");
            }
            sb.append("\n");
        }
        return sb.toString();
    }

    private void clearBoard(){
        GraphicsContext gc = canvas.getGraphicsContext2D();
        gc.clearRect(0, 0, canvas.getWidth(),
canvas.getHeight());
    }

    private void drawBoard() {
        canvas.setVisible(true);
        int cellSize = 40;
        int width = board.getWidth();
        int height = board.getHeight();

        canvas.setWidth(width * cellSize);
```

```
        canvas.setHeight(height * cellSize);

        GraphicsContext gc = canvas.getGraphicsContext2D();
        gc.setFill(Color.WHITE);
        gc.fillRect(0, 0, canvas.getWidth(),
canvas.getHeight());

        char[][] grid = board.getGrid();

        for (int row = 0; row < height; row++) {
            for (int col = 0; col < width; col++) {
                char piece = grid[row][col];
                gc.setFill(getPieceColor(piece));
                gc.fillRect(col * cellSize, row * cellSize,
cellSize, cellSize);

                gc.setStroke(Color.BLACK);
                gc.strokeRect(col * cellSize, row * cellSize,
cellSize, cellSize);

                gc.setFill(Color.BLACK);
                gc.setFont(javafx.scene.text.Font.font("Arial",
16));
                gc.fillText(String.valueOf(piece), col *
cellSize + cellSize / 3.5, row * cellSize + cellSize / 1.7);
            }
        }
    }

    private void drawNoSolutionBoard() {
        canvas.setVisible(true);
        GraphicsContext gc = canvas.getGraphicsContext2D();
        gc.setFill(Color.RED);
        gc.setFont(javafx.scene.text.Font.font("Arial", 72));
        gc.fillText("X", canvas.getWidth() / 2 - 20,
canvas.getHeight() / 2);
    }

    private Color getPieceColor(char piece) {
        int hash = (piece * 37) % 360;
        return Color.hsb(hash, 0.6, 0.7);
    }

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

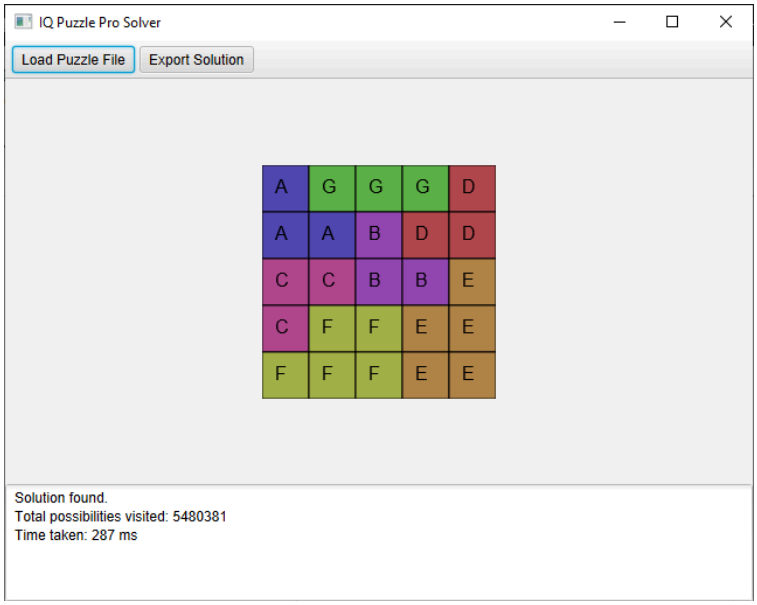
4. Uji Coba

Program diuji dengan menggunakan masukan berupa file berformat **.txt**, yang memiliki struktur sebagai berikut:

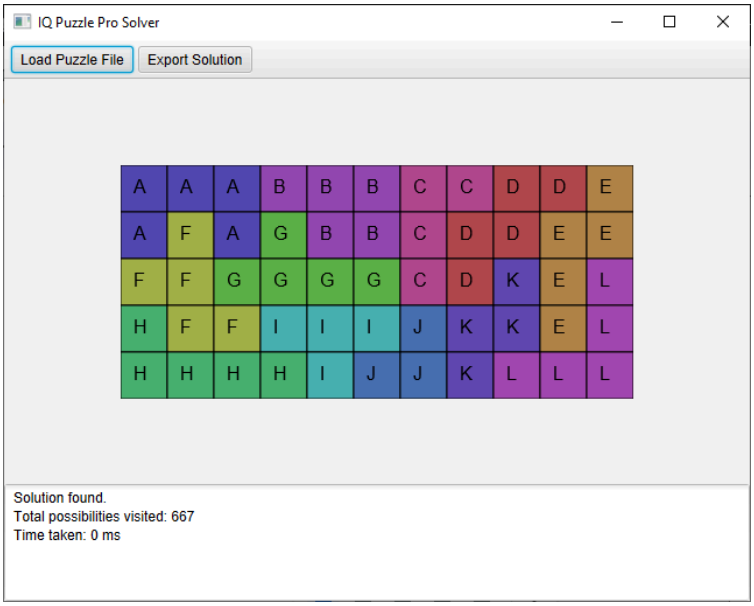
```
N M P
S
puzzle_1_shape
puzzle_2_shape
…
puzzle_P_shape
```

Dengan N dan M sebagai dimensi papan (N×M), P sebagai jumlah blok puzzle, S sebagai jenis kasus yang hanya bernilai DEFAULT, serta setiap blok puzzle direpresentasikan oleh P huruf kapital unik (A-Z).

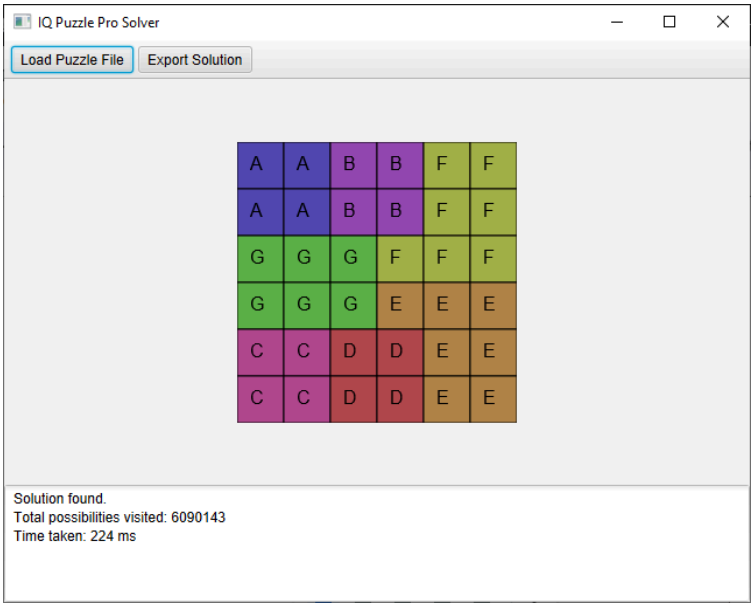
4.1. Tangkapan Layar Uji Coba



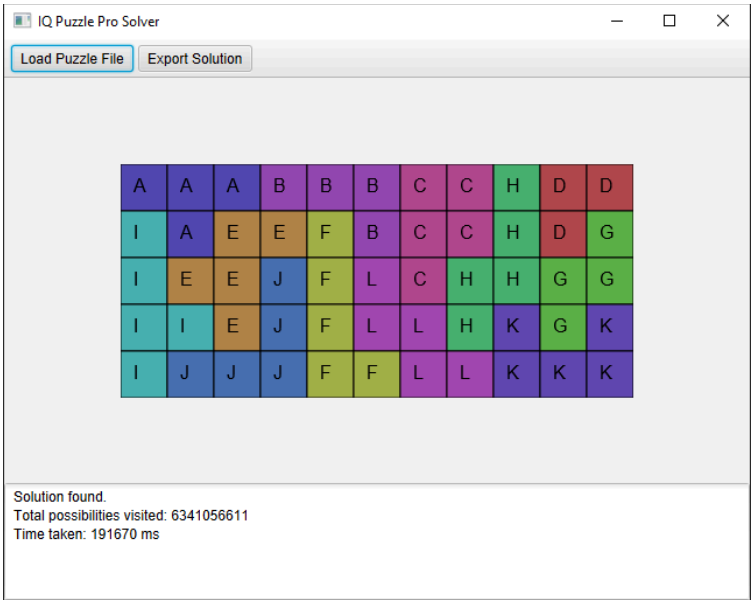
Gambar 4.1.1. Tangkapan Layar Hasil Uji Coba 1



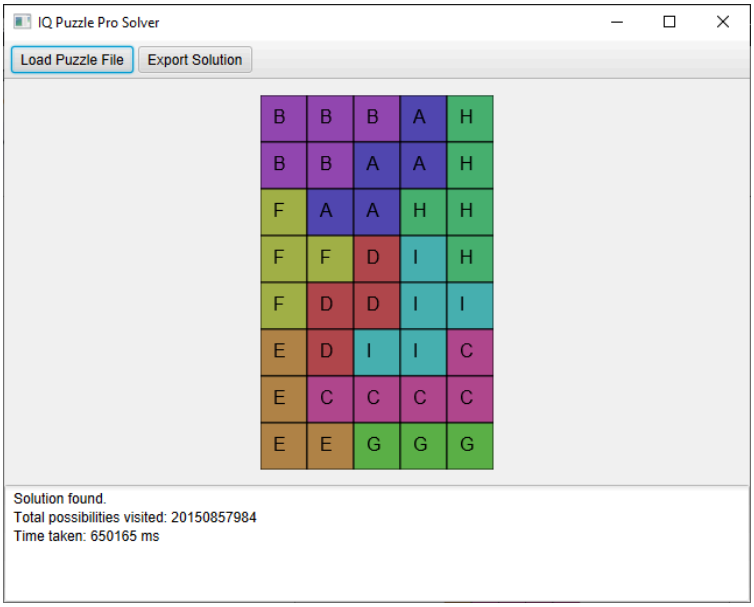
Gambar 4.1.2. Tangkapan Layar Hasil Uji Coba 2



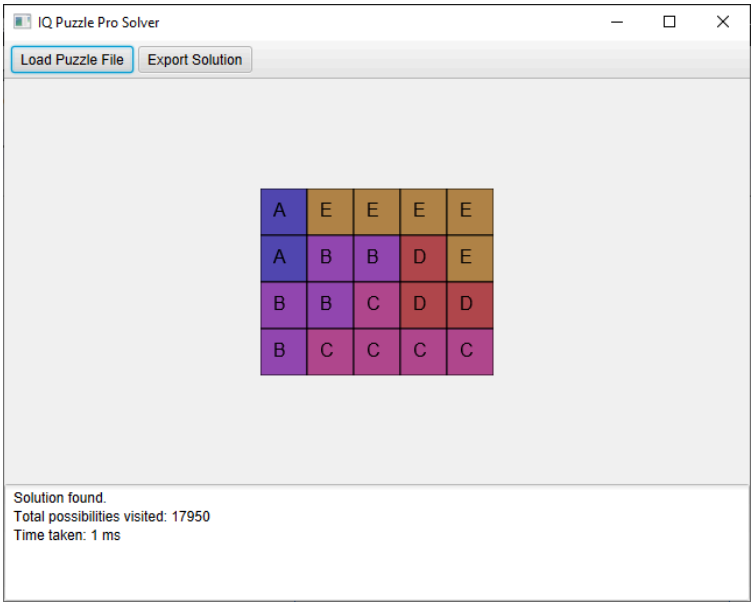
Gambar 4.1.3. Tangkapan Layar Hasil Uji Coba 3



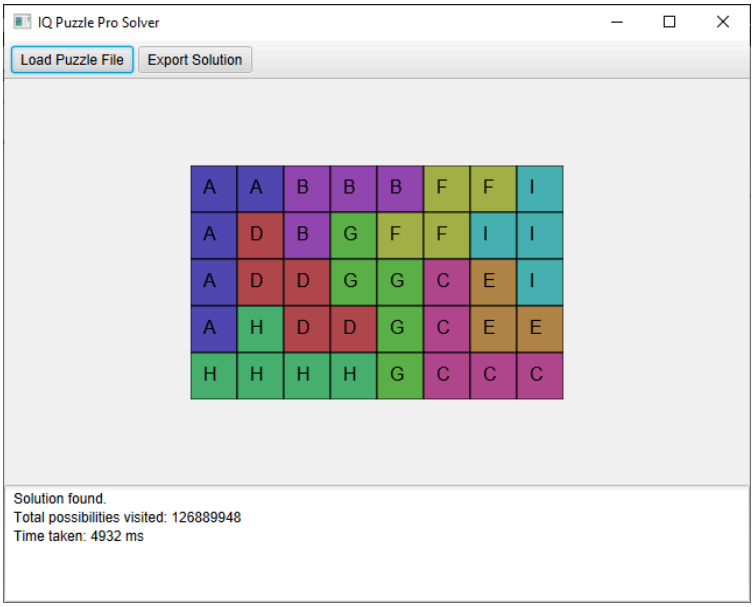
Gambar 4.1.4. Tangkapan Layar Hasil Uji Coba 4



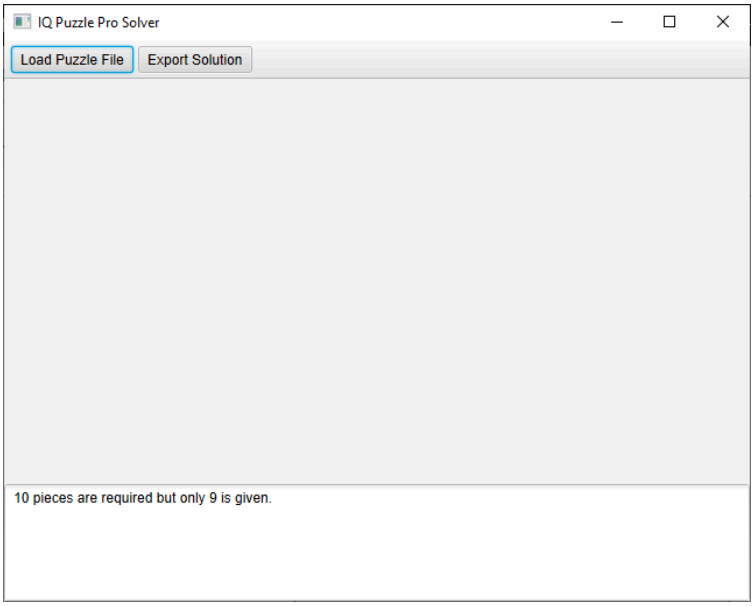
Gambar 4.1.5. Tangkapan Layar Hasil Uji Coba 5



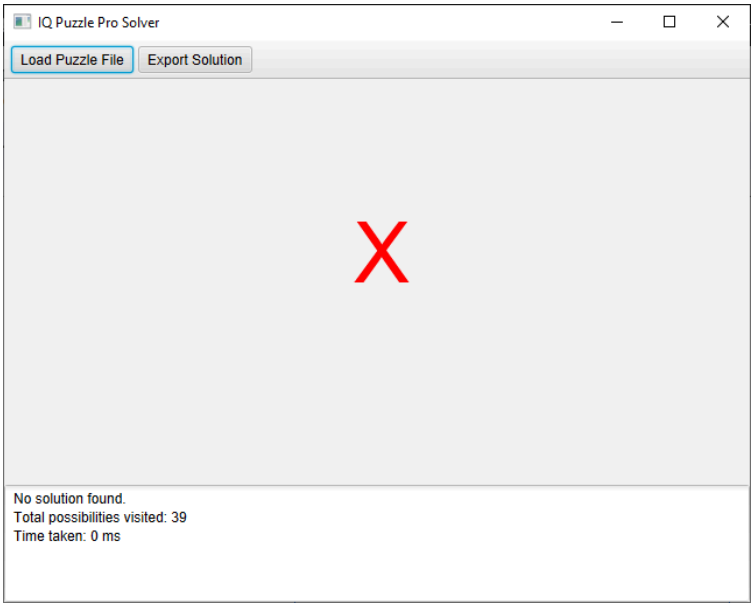
Gambar 4.1.6. Tangkapan Layar Hasil Uji Coba 6



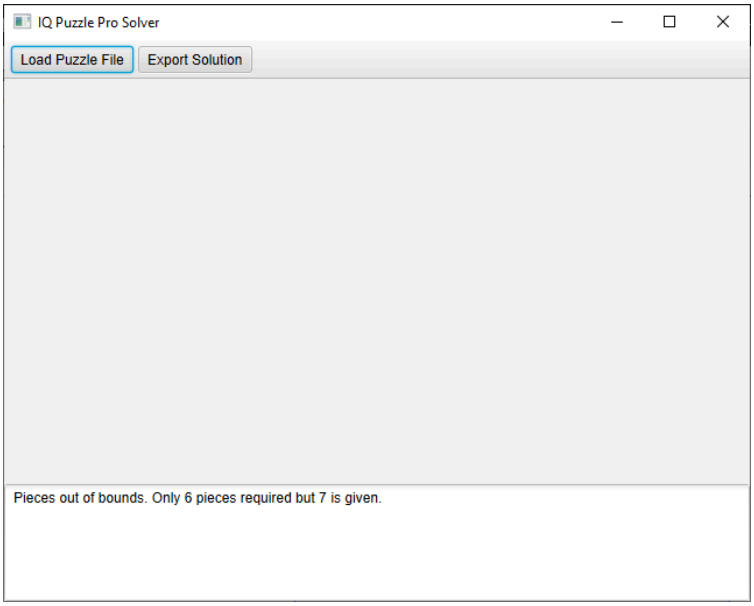
Gambar 4.1.7. Tangkapan Layar Hasil Uji Coba 7



Gambar 4.1.8. Tangkapan Layar Hasil Uji Coba 8



Gambar 4.1.9. Tangkapan Layar Hasil Uji Coba 9



Gambar 4.1.10. Tangkapan Layar Hasil Uji Coba 10

5. Pranala Repository

Tucil1_13523147

6. Lampiran

No	Poin	Ya	Tidak
1	Program berhasil dikompilasi 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	✔	
7	Program dapat menyelesaikan kasus konfigurasi *custom*		✔
8	Program dapat menyelesaikan kasus konfigurasi Piramida (3D)		✔
9	Program dibuat oleh saya sendiri	✔	