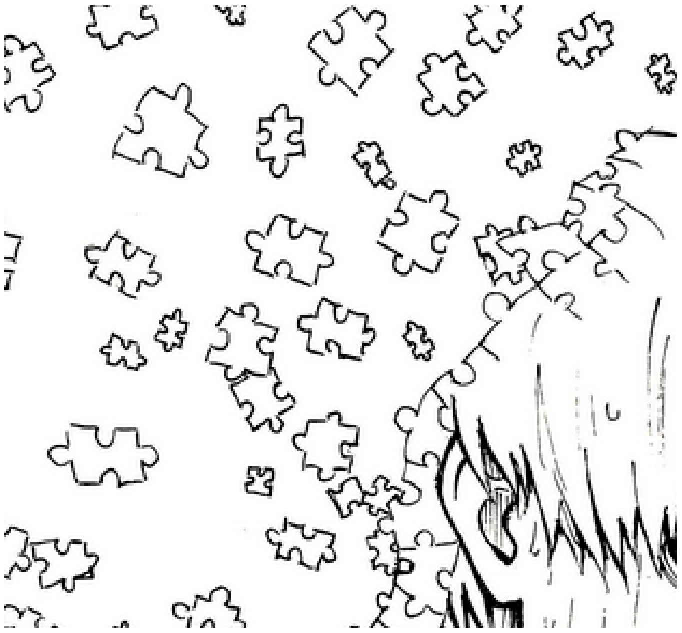
LAPORAN TUGAS KECIL 1
IF2211 - STRATEGI ALGORITMA
Semester II tahun 2024/2025
Penyelesaian IQ Puzzler Pro dengan Algoritma Brute Force



Disusun oleh :

Muhammad Hazim Ramadhan Prajoda 13523009

Dosen Pengajar :

Dr. Nur Ulfa Maulidevi, S.T, M.

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2025

Daftar Isi

Bab I Deskripsi Masalah

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.

Permainan dimulai dengan papan yang kosong. Pemain dapat meletakkan blok puzzle sedemikian sehingga tidak ada blok yang bertumpang tindih (kecuali dalam kasus 3D). Setiap blok puzzle dapat dirotasikan maupun dicerminkan. Puzzle dinyatakan selesai jika dan hanya jika papan terisi penuh dan seluruh blok puzzle berhasil diletakkan.

Tugas anda 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.

(Sumber: https://www.smartgamesusa.com)

Bab II Teori Brute Force

Dalam mencari solusi untuk permainan ini, program menggunakan algoritma Brute Force yang artinya program akan mencoba semua kemungkinan penempatan blok puzzle pada papan. Untuk setiap blok, program mencoba semua posisi yang mungkin (koordinat x,y pada papan), semua rotasi yang mungkin (4 rotasi: 0°, 90°, 180°, 270°), dan semua pencerminan yang mungkin (normal dan mirror)

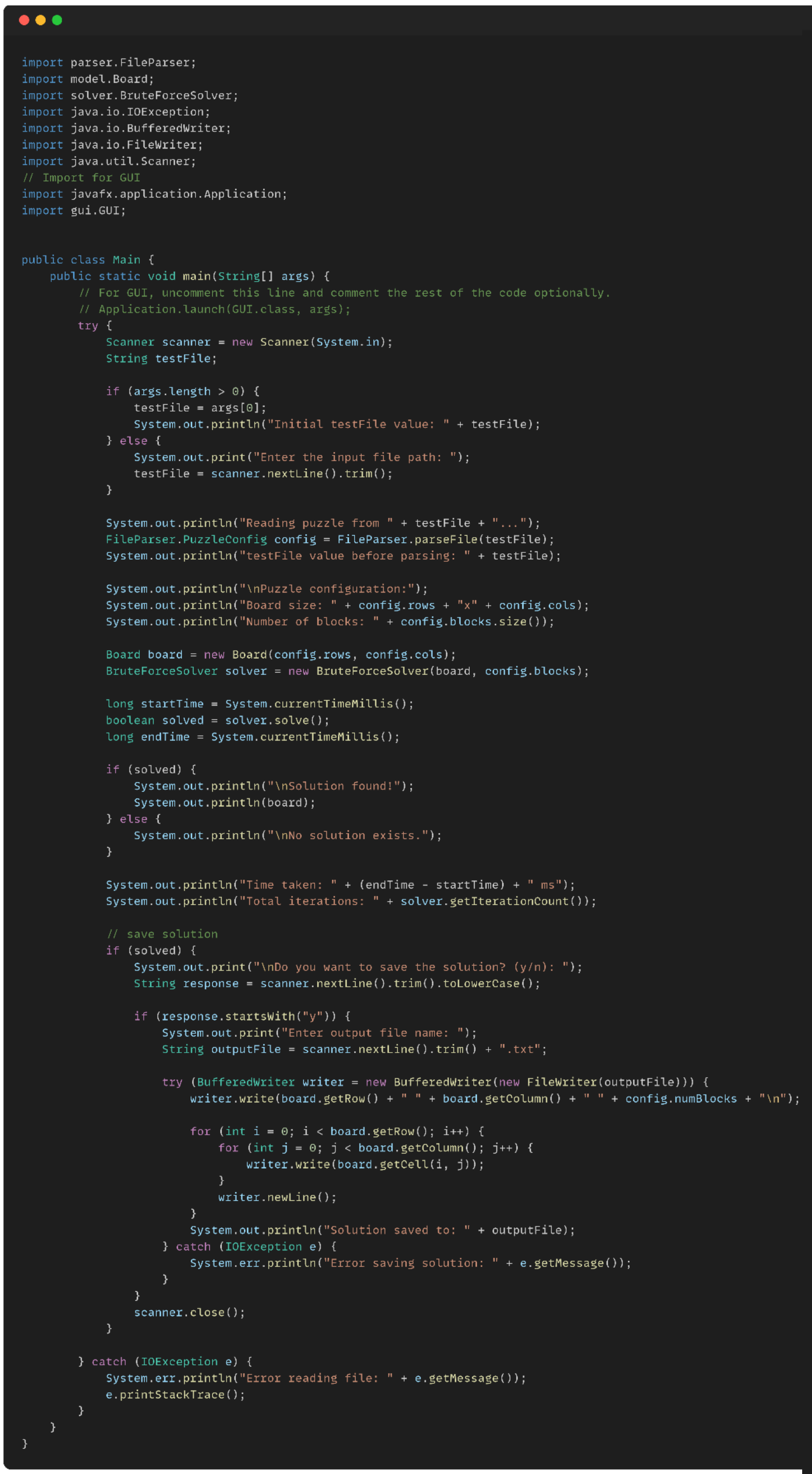
Langkah-langkah Brute Force:

1. **Program mulai dengan papan kosong dan daftar blok puzzle yang tersedia.**
2. **Untuk setiap blok puzzle, program akan melakukan:**
    - Tandai blok sebagai "digunakan" dalam array used[].
    - Coba setiap rotasi dari blok (0°, 90°, 180°, 270°).
    - Untuk setiap rotasi, coba dua orientasi (normal dan mirror).
    - Untuk setiap orientasi, coba setiap posisi pada papan (x,y).
3. **Pada setiap percobaan penempatan blok:**
    - Cek apakah blok bisa ditempatkan di posisi tersebut.
    - Jika bisa, letakkan blok di posisi tersebut.
    - Lanjut ke blok berikutnya secara rekursif.
    - Jika tidak menuju solusi, hapus blok (backtracking) dan coba posisi lain.
4. **Program berhenti jika:**
    - Solusi ditemukan (semua blok berhasil ditempatkan).
    - Tidak ada solusi yang mungkin setelah mencoba semua kombinasi.
5. **Optimisasi yang ditambahkan:**
    - Cek apakah jumlah sel kosong cukup untuk sisa blok.
    - Hentikan pencarian di cabang yang tidak mungkin menuju solusi.

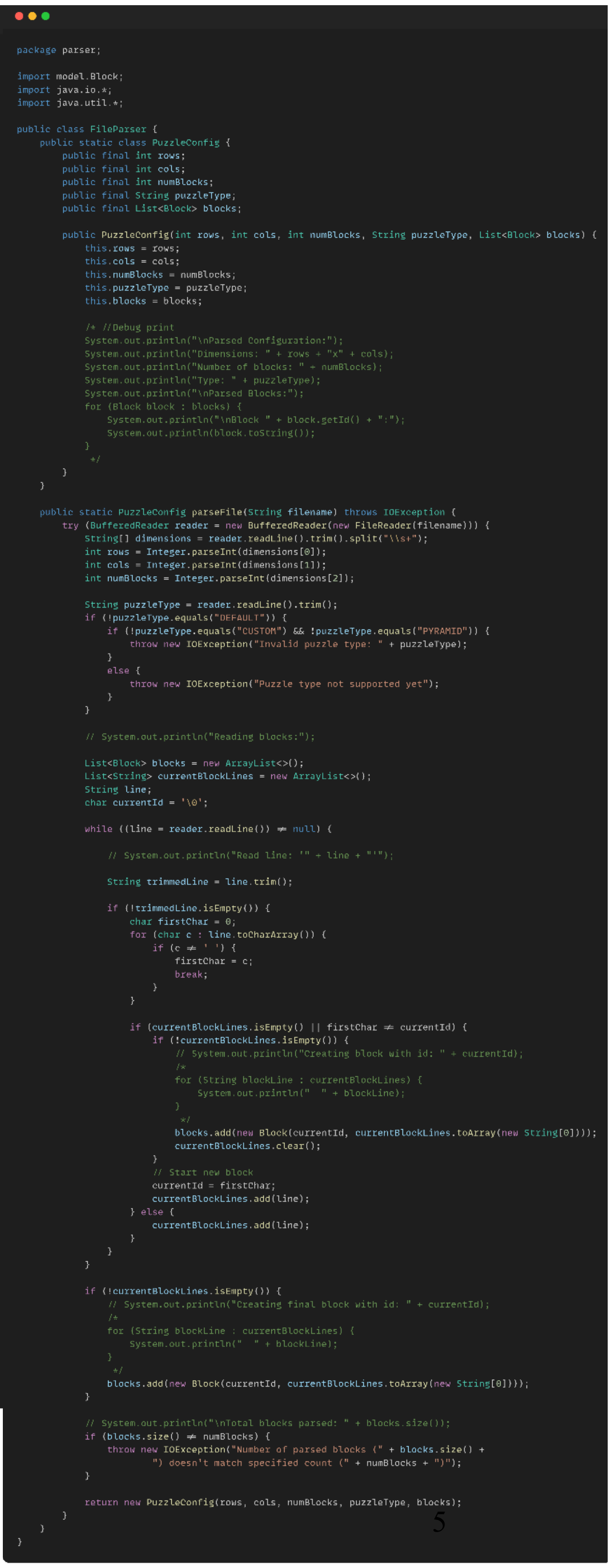
Bab III Source Code

bisa dilihat juga di link repository pada lampiran

Main.java - FileParser.java

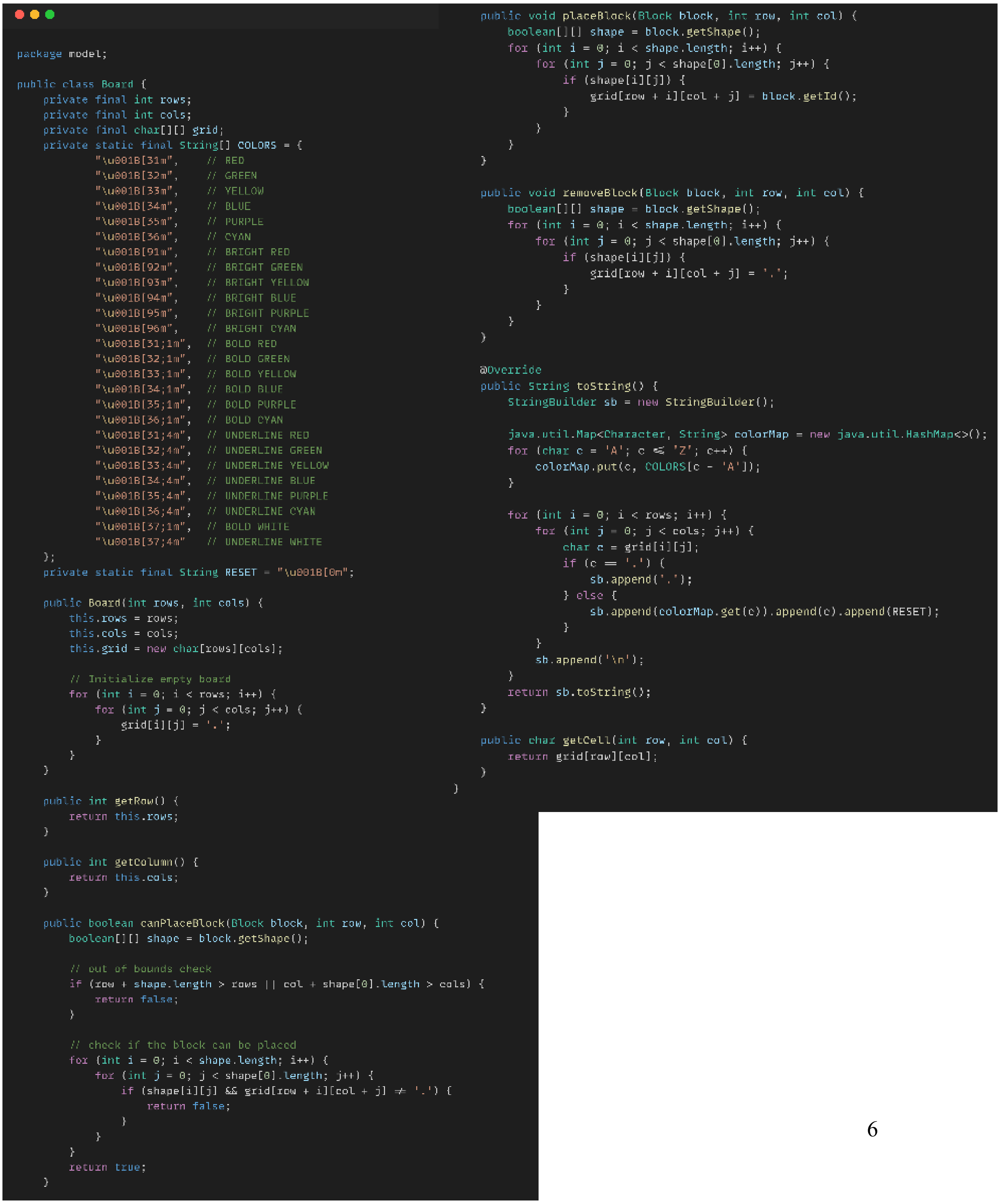


```java
import parser.FileParser;
import model.Board;
import solver.BruteForceSolver;
import java.io.IOException;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.util.Scanner;
// Import for GUI
import javafx.application.Application;
import gui.GUI;


public class Main {
    public static void main(String[] args) {
        // For GUI, uncomment this line and comment the rest of the code optionally.
        // Application.launch(GUI.class, args);
        try {
            Scanner scanner = new Scanner(System.in);
            String testFile;

            if (args.length > 0) {
                testFile = args[0];
                System.out.println("Initial testFile value: " + testFile);
            } else {
                System.out.print("Enter the input file path: ");
                testFile = scanner.nextLine().trim();
            }

            System.out.println("Reading puzzle from " + testFile + "...");
            FileParser.PuzzleConfig config = FileParser.parseFile(testFile);
            System.out.println("testFile value before parsing: " + testFile);

            System.out.println("\nPuzzle configuration:");
            System.out.println("Board size: " + config.rows + "x" + config.cols);
            System.out.println("Number of blocks: " + config.blocks.size());

            Board board = new Board(config.rows, config.cols);
            BruteForceSolver solver = new BruteForceSolver(board, config.blocks);

            long startTime = System.currentTimeMillis();
            boolean solved = solver.solve();
            long endTime = System.currentTimeMillis();

            if (solved) {
                System.out.println("\nSolution found!");
                System.out.println(board);
            } else {
                System.out.println("\nNo solution exists.");
            }

            System.out.println("Time taken: " + (endTime - startTime) + " ms");
            System.out.println("Total iterations: " + solver.getIterationCount());

            // save solution
            if (solved) {
                System.out.print("\nDo you want to save the solution? (y/n): ");
                String response = scanner.nextLine().trim().toLowerCase();

                if (response.startsWith("y")) {
                    System.out.print("Enter output file name: ");
                    String outputFile = scanner.nextLine().trim() + ".txt";

                    try (BufferedWriter writer = new BufferedWriter(new FileWriter(outputFile))) {
                        writer.write(board.getRow() + " " + board.getColumn() + " " + config.numBlocks + "\n");

                        for (int i = 0; i < board.getRow(); i++) {
                            for (int j = 0; j < board.getColumn(); j++) {
                                writer.write(board.getCell(i, j));
                            }
                            writer.newLine();
                        }
                        System.out.println("Solution saved to: " + outputFile);
                    } catch (IOException e) {
                        System.err.println("Error saving solution: " + e.getMessage());
                    }
                }
                scanner.close();
            }

        } catch (IOException e) {
            System.err.println("Error reading file: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```



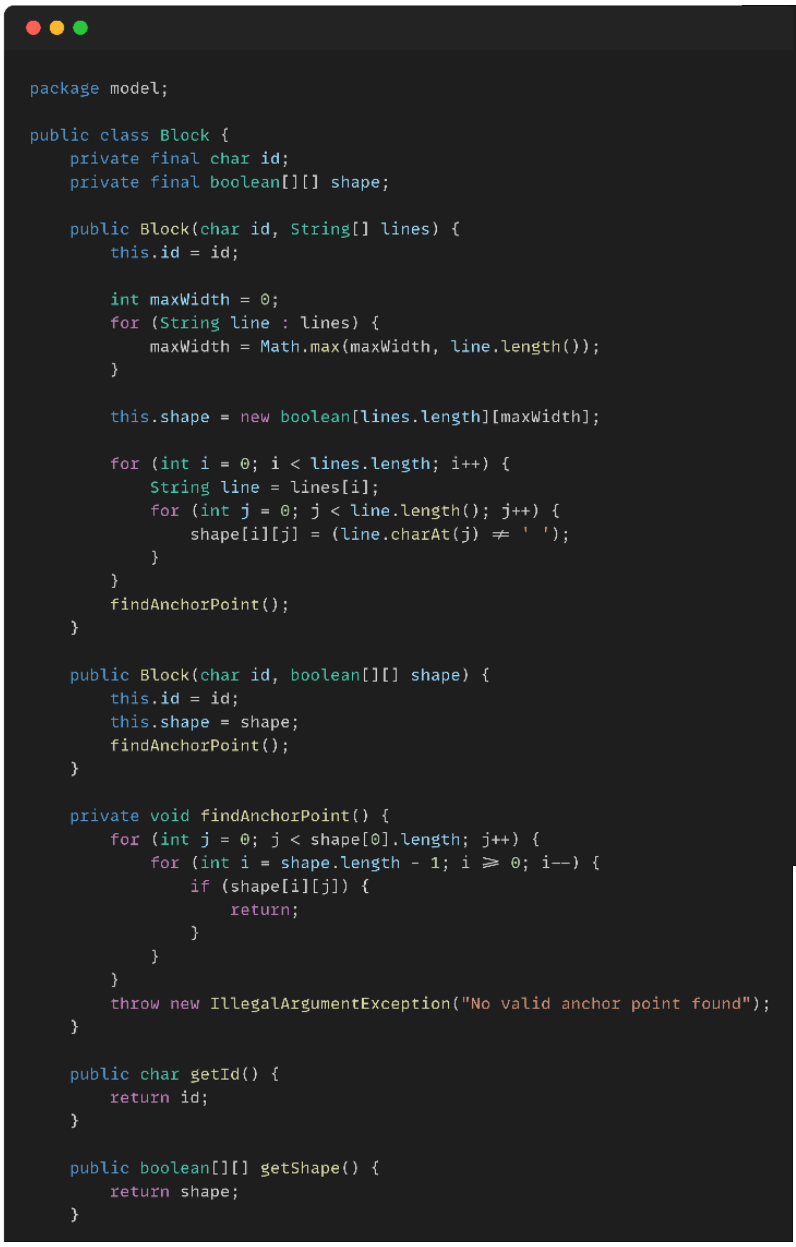
```java
package parser;

import model.Block;
import java.io.*;
import java.util.*;

public class FileParser {
    public static class PuzzleConfig {
        public final int rows;
        public final int cols;
        public final int numBlocks;
        public final String puzzleType;
        public final List<Block> blocks;

        public PuzzleConfig(int rows, int cols, int numBlocks, String puzzleType, List<Block> blocks) {
            this.rows = rows;
            this.cols = cols;
            this.numBlocks = numBlocks;
            this.puzzleType = puzzleType;
            this.blocks = blocks;

            /* //Debug print
            System.out.println("\nParsed Configuration:");
            System.out.println("Dimensions: " + rows + "x" + cols);
            System.out.println("Number of blocks: " + numBlocks);
            System.out.println("Type: " + puzzleType);
            System.out.println("\nParsed Blocks:");
            for (Block block : blocks) {
                System.out.println("\nBlock " + block.getId() + ":");
                System.out.println(block.toString());
            }
             */
        }
    }

    public static PuzzleConfig parseFile(String filename) throws IOException {
        try (BufferedReader reader = new BufferedReader(new FileReader(filename))) {
            String[] dimensions = reader.readLine().trim().split("\\s+");
            int rows = Integer.parseInt(dimensions[0]);
            int cols = Integer.parseInt(dimensions[1]);
            int numBlocks = Integer.parseInt(dimensions[2]);

            String puzzleType = reader.readLine().trim();
            if (!puzzleType.equals("DEFAULT")) {
                if (!puzzleType.equals("CUSTOM") && !puzzleType.equals("PYRAMID")) {
                    throw new IOException("Invalid puzzle type: " + puzzleType);
                }
                else {
                    throw new IOException("Puzzle type not supported yet");
                }
            }

            // System.out.println("Reading blocks:");

            List<Block> blocks = new ArrayList<>();
            List<String> currentBlockLines = new ArrayList<>();
            String line;
            char currentId = '\0';

            while ((line = reader.readLine()) != null) {

                // System.out.println("Read line: '" + line + "'");

                String trimmedLine = line.trim();

                if (!trimmedLine.isEmpty()) {
                    char firstChar = 0;
                    for (char c : line.toCharArray()) {
                        if (c != ' ') {
                            firstChar = c;
                            break;
                        }
                    }

                    if (currentBlockLines.isEmpty() || firstChar != currentId) {
                        if (!currentBlockLines.isEmpty()) {
                            // System.out.println("Creating block with id: " + currentId);
                            /*
                            for (String blockLine : currentBlockLines) {
                                System.out.println("  " + blockLine);
                            }
                             */
                            blocks.add(new Block(currentId, currentBlockLines.toArray(new String[0])));
                            currentBlockLines.clear();
                        }
                        // Start new block
                        currentId = firstChar;
                        currentBlockLines.add(line);
                    } else {
                        currentBlockLines.add(line);
                    }
                }
            }

            if (!currentBlockLines.isEmpty()) {
                // System.out.println("Creating final block with id: " + currentId);
                /*
                for (String blockLine : currentBlockLines) {
                    System.out.println("  " + blockLine);
                }
                 */
                blocks.add(new Block(currentId, currentBlockLines.toArray(new String[0])));
            }

            // System.out.println("\nTotal blocks parsed: " + blocks.size());
            if (blocks.size() != numBlocks) {
                throw new IOException("Number of parsed blocks (" + blocks.size() +
                        ") doesn't match specified count (" + numBlocks + ")");
            }

            return new PuzzleConfig(rows, cols, numBlocks, puzzleType, blocks);
        }
    }
}
```

Board.java

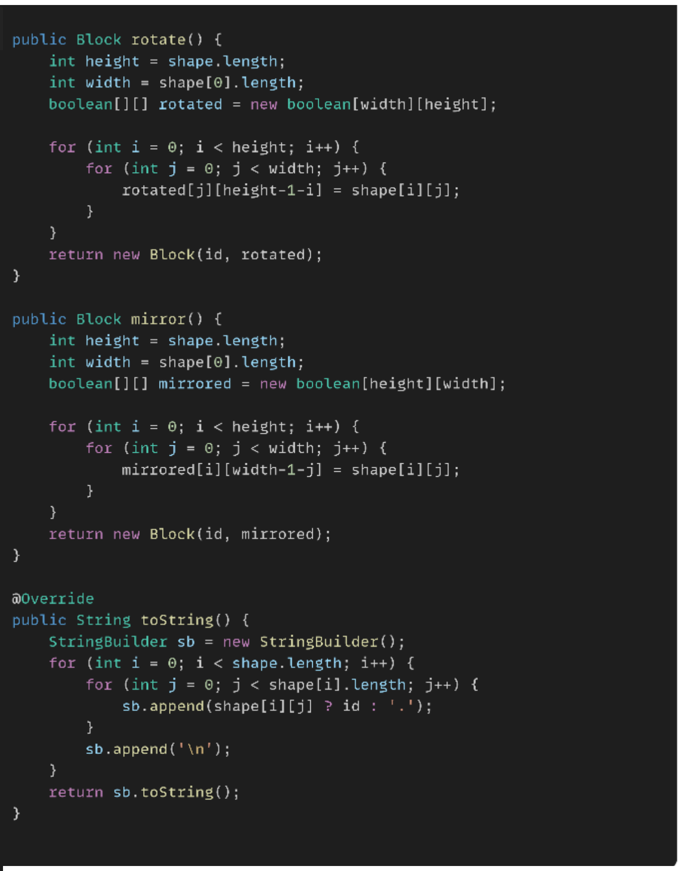


```java
package model;

public class Board {
    private final int rows;
    private final int cols;
    private final char[][] grid;
    private static final String[] COLORS = {
            "\u001B[31m",     // RED
            "\u001B[32m",     // GREEN
            "\u001B[33m",     // YELLOW
            "\u001B[34m",     // BLUE
            "\u001B[35m",     // PURPLE
            "\u001B[36m",     // CYAN
            "\u001B[91m",     // BRIGHT RED
            "\u001B[92m",     // BRIGHT GREEN
            "\u001B[93m",     // BRIGHT YELLOW
            "\u001B[94m",     // BRIGHT BLUE
            "\u001B[95m",     // BRIGHT PURPLE
            "\u001B[96m",     // BRIGHT CYAN
            "\u001B[31;1m",   // BOLD RED
            "\u001B[32;1m",   // BOLD GREEN
            "\u001B[33;1m",   // BOLD YELLOW
            "\u001B[34;1m",   // BOLD BLUE
            "\u001B[35;1m",   // BOLD PURPLE
            "\u001B[36;1m",   // BOLD CYAN
            "\u001B[31;4m",   // UNDERLINE RED
            "\u001B[32;4m",   // UNDERLINE GREEN
            "\u001B[33;4m",   // UNDERLINE YELLOW
            "\u001B[34;4m",   // UNDERLINE BLUE
            "\u001B[35;4m",   // UNDERLINE PURPLE
            "\u001B[36;4m",   // UNDERLINE CYAN
            "\u001B[37;1m",   // BOLD WHITE
            "\u001B[37;4m"    // UNDERLINE WHITE
    };
    private static final String RESET = "\u001B[0m";

    public Board(int rows, int cols) {
        this.rows = rows;
        this.cols = cols;
        this.grid = new char[rows][cols];

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

    public int getRow() {
        return this.rows;
    }

    public int getColumn() {
        return this.cols;
    }

    public boolean canPlaceBlock(Block block, int row, int col) {
        boolean[][] shape = block.getShape();

        // out of bounds check
        if (row + shape.length > rows || col + shape[0].length > cols) {
            return false;
        }

        // check if the block can be placed
        for (int i = 0; i < shape.length; i++) {
            for (int j = 0; j < shape[0].length; j++) {
                if (shape[i][j] && grid[row + i][col + j] != '.') {
                    return false;
                }
            }
        }
        return true;
    }

    public void placeBlock(Block block, int row, int col) {
        boolean[][] shape = block.getShape();
        for (int i = 0; i < shape.length; i++) {
            for (int j = 0; j < shape[0].length; j++) {
                if (shape[i][j]) {
                    grid[row + i][col + j] = block.getId();
                }
            }
        }
    }

    public void removeBlock(Block block, int row, int col) {
        boolean[][] shape = block.getShape();
        for (int i = 0; i < shape.length; i++) {
            for (int j = 0; j < shape[0].length; j++) {
                if (shape[i][j]) {
                    grid[row + i][col + j] = '.';
                }
            }
        }
    }

    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder();

        java.util.Map<Character, String> colorMap = new java.util.HashMap<>();
        for (char c = 'A'; c <= 'Z'; c++) {
            colorMap.put(c, COLORS[c - 'A']);
        }

        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                char c = grid[i][j];
                if (c == '.') {
                    sb.append('.');
                } else {
                    sb.append(colorMap.get(c)).append(c).append(RESET);
                }
            }
            sb.append('\n');
        }
        return sb.toString();
    }

    public char getCell(int row, int col) {
        return grid[row][col];
    }
}
```

Block.java

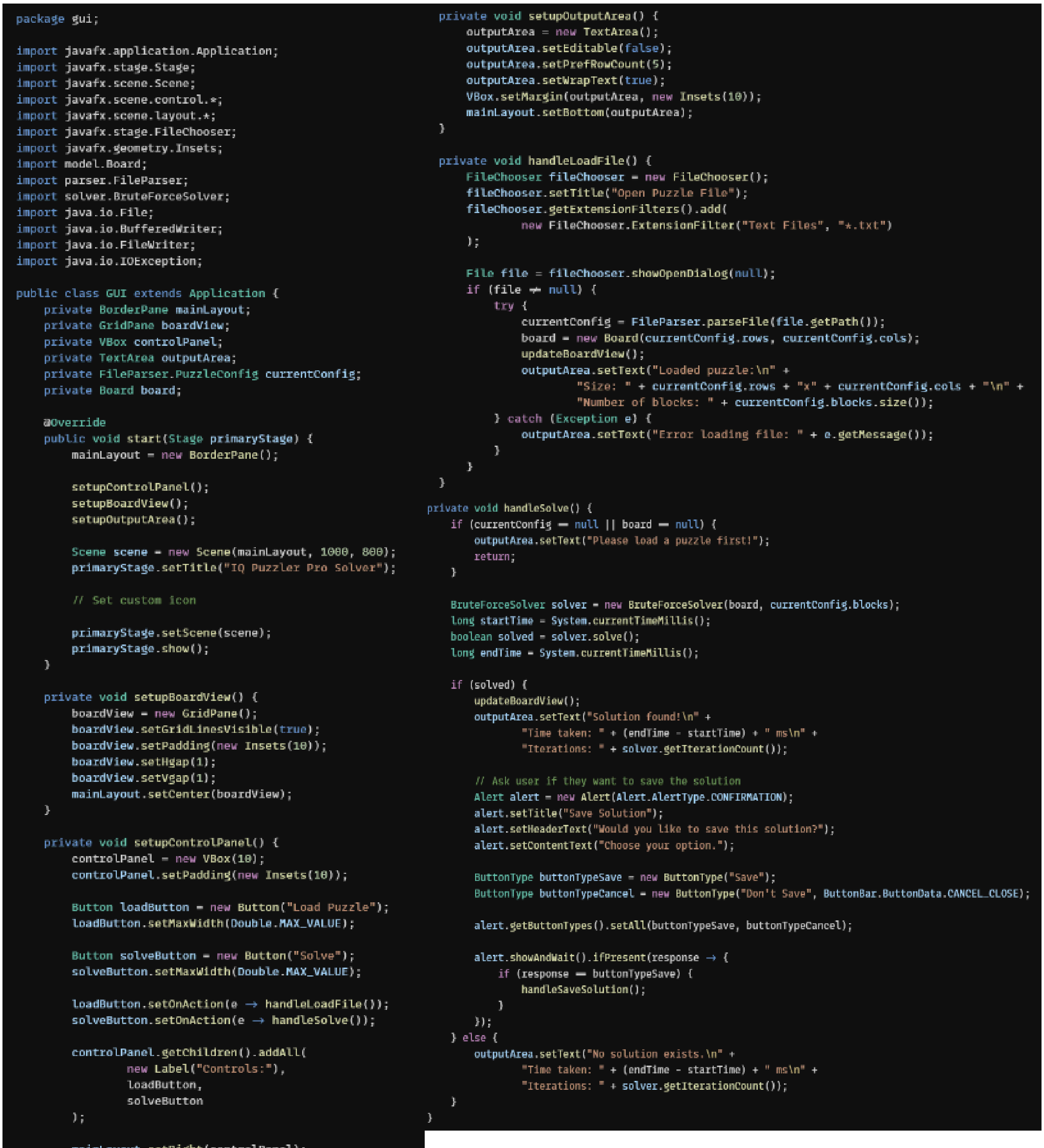


```java
package model;

public class Block {
    private final char id;
    private final boolean[][] shape;

    public Block(char id, String[] lines) {
        this.id = id;

        int maxWidth = 0;
        for (String line : lines) {
            maxWidth = Math.max(maxWidth, line.length());
        }

        this.shape = new boolean[lines.length][maxWidth];

        for (int i = 0; i < lines.length; i++) {
            String line = lines[i];
            for (int j = 0; j < line.length(); j++) {
                shape[i][j] = (line.charAt(j) ≠ ' ');
            }
        }
        findAnchorPoint();
    }

    public Block(char id, boolean[][] shape) {
        this.id = id;
        this.shape = shape;
        findAnchorPoint();
    }

    private void findAnchorPoint() {
        for (int j = 0; j < shape[0].length; j++) {
            for (int i = shape.length - 1; i ≥ 0; i--) {
                if (shape[i][j]) {
                    return;
                }
            }
        }
        throw new IllegalArgumentException("No valid anchor point found");
    }

    public char getId() {
        return id;
    }

    public boolean[][] getShape() {
        return shape;
    }
```



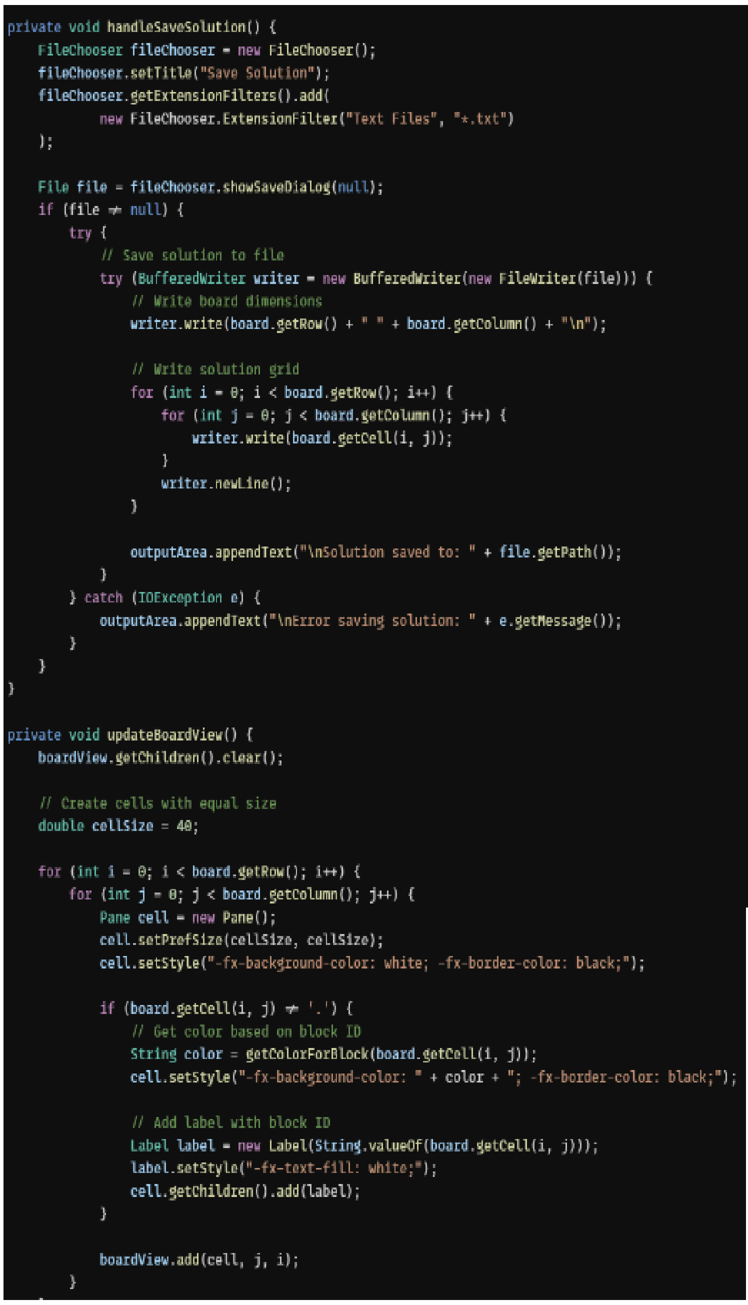
```java
public Block rotate() {
    int height = shape.length;
    int width = shape[0].length;
    boolean[][] rotated = new boolean[width][height];

    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            rotated[j][height-1-i] = shape[i][j];
        }
    }
    return new Block(id, rotated);
}

public Block mirror() {
    int height = shape.length;
    int width = shape[0].length;
    boolean[][] mirrored = new boolean[height][width];

    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            mirrored[i][width-1-j] = shape[i][j];
        }
    }
    return new Block(id, mirrored);
}

@Override
public String toString() {
    StringBuilder sb = new StringBuilder();
    for (int i = 0; i < shape.length; i++) {
        for (int j = 0; j < shape[i].length; j++) {
            sb.append(shape[i][j] ? id : '.');
        }
        sb.append('\n');
    }
    return sb.toString();
}
```

GUI.java

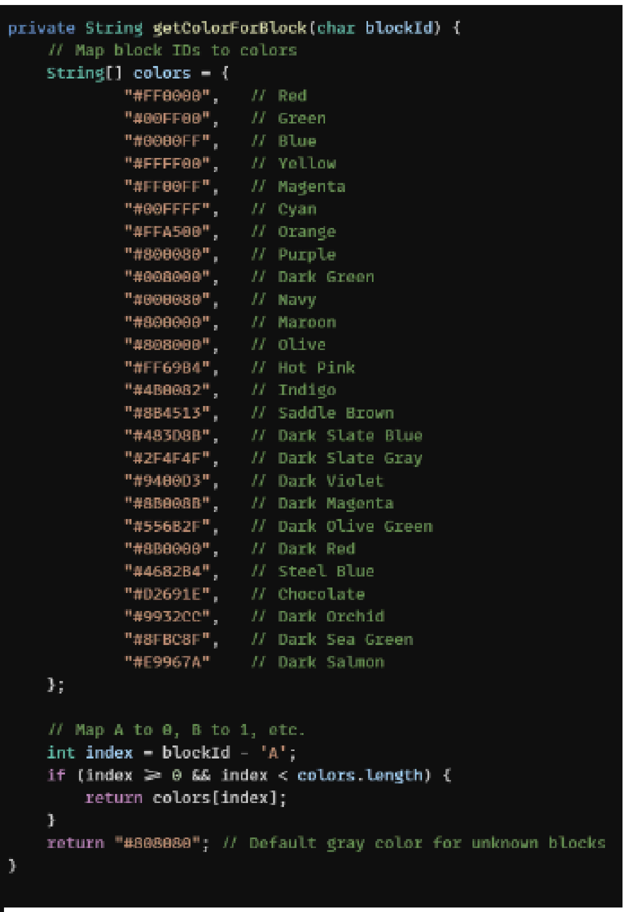


```java
package gui;

import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.*;
import javafx.stage.FileChooser;
import javafx.geometry.Insets;
import model.Board;
import parser.FileParser;
import solver.BruteForceSolver;
import java.io.File;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;

public class GUI extends Application {
    private BorderPane mainLayout;
    private GridPane boardView;
    private VBox controlPanel;
    private TextArea outputArea;
    private FileParser.PuzzleConfig currentConfig;
    private Board board;

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

        setupControlPanel();
        setupBoardView();
        setupOutputArea();

        Scene scene = new Scene(mainLayout, 1000, 800);
        primaryStage.setTitle("IQ Puzzler Pro Solver");

        // Set custom icon

        primaryStage.setScene(scene);
        primaryStage.show();
    }

    private void setupBoardView() {
        boardView = new GridPane();
        boardView.setGridLinesVisible(true);
        boardView.setPadding(new Insets(10));
        boardView.setHgap(1);
        boardView.setVgap(1);
        mainLayout.setCenter(boardView);
    }

    private void setupControlPanel() {
        controlPanel = new VBox(10);
        controlPanel.setPadding(new Insets(10));

        Button loadButton = new Button("Load Puzzle");
        loadButton.setMaxWidth(Double.MAX_VALUE);

        Button solveButton = new Button("Solve");
        solveButton.setMaxWidth(Double.MAX_VALUE);

        loadButton.setOnAction(e -> handleLoadFile());
        solveButton.setOnAction(e -> handleSolve());

        controlPanel.getChildren().addAll(
                new Label("Controls:"),
                loadButton,
                solveButton
        );

        mainLayout.setRight(controlPanel);
```

```java
    private void setupOutputArea() {
        outputArea = new TextArea();
        outputArea.setEditable(false);
        outputArea.setPrefRowCount(5);
        outputArea.setWrapText(true);
        VBox.setMargin(outputArea, new Insets(10));
        mainLayout.setBottom(outputArea);
    }

    private void handleLoadFile() {
        FileChooser fileChooser = new FileChooser();
        fileChooser.setTitle("Open Puzzle File");
        fileChooser.getExtensionFilters().add(
                new FileChooser.ExtensionFilter("Text Files", "*.txt")
        );

        File file = fileChooser.showOpenDialog(null);
        if (file != null) {
            try {
                currentConfig = FileParser.parseFile(file.getPath());
                board = new Board(currentConfig.rows, currentConfig.cols);
                updateBoardView();
                outputArea.setText("Loaded puzzle:\n" +
                        "Size: " + currentConfig.rows + "x" + currentConfig.cols + "\n" +
                        "Number of blocks: " + currentConfig.blocks.size());
            } catch (Exception e) {
                outputArea.setText("Error loading file: " + e.getMessage());
            }
        }
    }

private void handleSolve() {
    if (currentConfig == null || board == null) {
        outputArea.setText("Please load a puzzle first!");
        return;
    }

    BruteForceSolver solver = new BruteForceSolver(board, currentConfig.blocks);
    long startTime = System.currentTimeMillis();
    boolean solved = solver.solve();
    long endTime = System.currentTimeMillis();

    if (solved) {
        updateBoardView();
        outputArea.setText("Solution found!\n" +
                "Time taken: " + (endTime - startTime) + " ms\n" +
                "Iterations: " + solver.getIterationCount());

        // Ask user if they want to save the solution
        Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
        alert.setTitle("Save Solution");
        alert.setHeaderText("Would you like to save this solution?");
        alert.setContentText("Choose your option.");

        ButtonType buttonTypeSave = new ButtonType("Save");
        ButtonType buttonTypeCancel = new ButtonType("Don't Save", ButtonBar.ButtonData.CANCEL_CLOSE);

        alert.getButtonTypes().setAll(buttonTypeSave, buttonTypeCancel);

        alert.showAndWait().ifPresent(response -> {
            if (response == buttonTypeSave) {
                handleSaveSolution();
            }
        });
    } else {
        outputArea.setText("No solution exists.\n" +
                "Time taken: " + (endTime - startTime) + " ms\n" +
                "Iterations: " + solver.getIterationCount());
    }
}
```

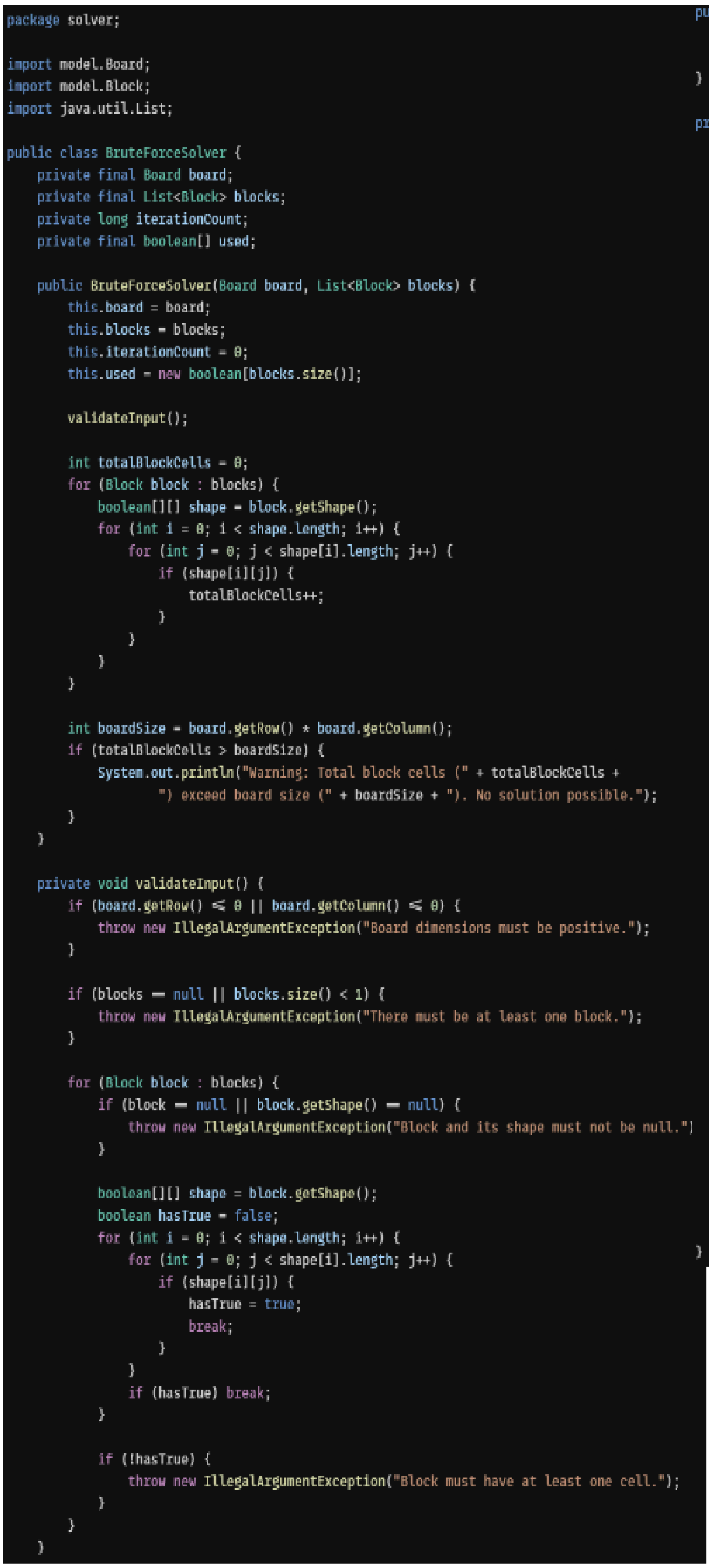


```java
private void handleSaveSolution() {
    FileChooser fileChooser = new FileChooser();
    fileChooser.setTitle("Save Solution");
    fileChooser.getExtensionFilters().add(
            new FileChooser.ExtensionFilter("Text Files", "*.txt")
    );

    File file = fileChooser.showSaveDialog(null);
    if (file ≠ null) {
        try {
            // Save solution to file
            try (BufferedWriter writer = new BufferedWriter(new FileWriter(file))) {
                // Write board dimensions
                writer.write(board.getRow() + " " + board.getColumn() + "\n");

                // Write solution grid
                for (int i = 0; i < board.getRow(); i++) {
                    for (int j = 0; j < board.getColumn(); j++) {
                        writer.write(board.getCell(i, j));
                    }
                    writer.newLine();
                }

                outputArea.appendText("\nSolution saved to: " + file.getPath());
            }
        } catch (IOException e) {
            outputArea.appendText("\nError saving solution: " + e.getMessage());
        }
    }
}

private void updateBoardView() {
    boardView.getChildren().clear();

    // Create cells with equal size
    double cellSize = 40;

    for (int i = 0; i < board.getRow(); i++) {
        for (int j = 0; j < board.getColumn(); j++) {
            Pane cell = new Pane();
            cell.setPrefSize(cellSize, cellSize);
            cell.setStyle("-fx-background-color: white; -fx-border-color: black;");

            if (board.getCell(i, j) ≠ '.') {
                // Get color based on block ID
                String color = getColorForBlock(board.getCell(i, j));
                cell.setStyle("-fx-background-color: " + color + "; -fx-border-color: black;");

                // Add label with block ID
                Label label = new Label(String.valueOf(board.getCell(i, j)));
                label.setStyle("-fx-text-fill: white;");
                cell.getChildren().add(label);
            }

            boardView.add(cell, j, i);
        }
```

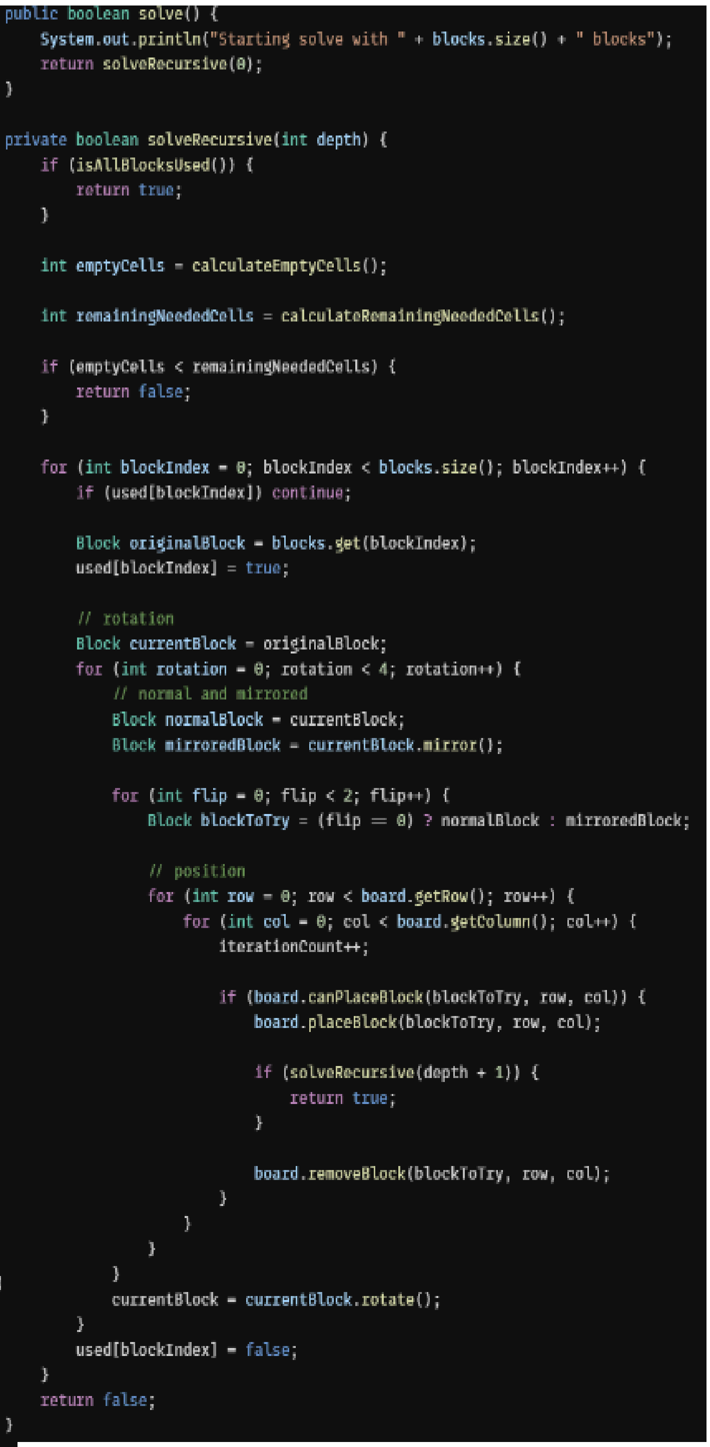


```java
private String getColorForBlock(char blockId) {
    // Map block IDs to colors
    String[] colors = {
            "#FF0000",   // Red
            "#00FF00",   // Green
            "#0000FF",   // Blue
            "#FFFF00",   // Yellow
            "#FF00FF",   // Magenta
            "#00FFFF",   // Cyan
            "#FFA500",   // Orange
            "#800080",   // Purple
            "#008000",   // Dark Green
            "#000080",   // Navy
            "#800000",   // Maroon
            "#808000",   // Olive
            "#FF69B4",   // Hot Pink
            "#4B0082",   // Indigo
            "#8B4513",   // Saddle Brown
            "#483D8B",   // Dark Slate Blue
            "#2F4F4F",   // Dark Slate Gray
            "#9400D3",   // Dark Violet
            "#8B008B",   // Dark Magenta
            "#556B2F",   // Dark Olive Green
            "#8B0000",   // Dark Red
            "#4682B4",   // Steel Blue
            "#D2691E",   // Chocolate
            "#9932CC",   // Dark Orchid
            "#8FBC8F",   // Dark Sea Green
            "#E9967A"    // Dark Salmon
    };

    // Map A to 0, B to 1, etc.
    int index = blockId - 'A';
    if (index ≥ 0 && index < colors.length) {
        return colors[index];
    }
    return "#808080"; // Default gray color for unknown blocks
}
```

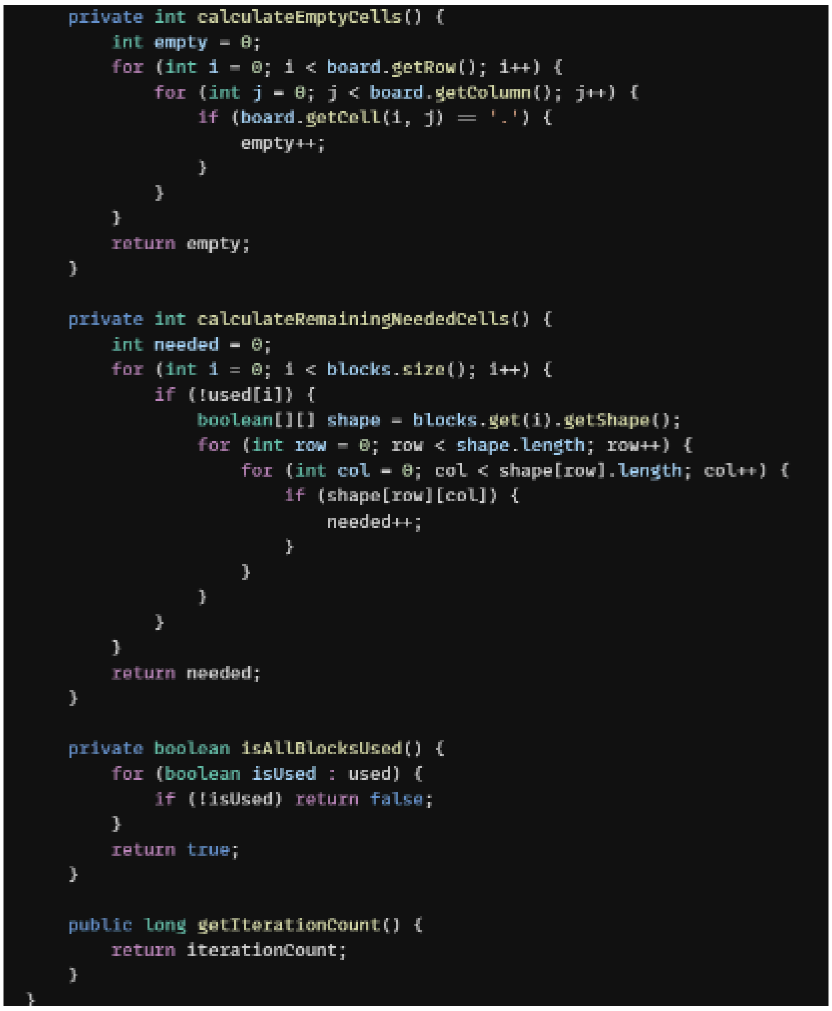
BruteForceSolver.java



```java
package solver;

import model.Board;
import model.Block;
import java.util.List;

public class BruteForceSolver {
    private final Board board;
    private final List<Block> blocks;
    private long iterationCount;
    private final boolean[] used;

    public BruteForceSolver(Board board, List<Block> blocks) {
        this.board = board;
        this.blocks = blocks;
        this.iterationCount = 0;
        this.used = new boolean[blocks.size()];

        validateInput();

        int totalBlockCells = 0;
        for (Block block : blocks) {
            boolean[][] shape = block.getShape();
            for (int i = 0; i < shape.length; i++) {
                for (int j = 0; j < shape[i].length; j++) {
                    if (shape[i][j]) {
                        totalBlockCells++;
                    }
                }
            }
        }

        int boardSize = board.getRow() * board.getColumn();
        if (totalBlockCells > boardSize) {
            System.out.println("Warning: Total block cells (" + totalBlockCells +
                    ") exceed board size (" + boardSize + "). No solution possible.");
        }
    }

    private void validateInput() {
        if (board.getRow() <= 0 || board.getColumn() <= 0) {
            throw new IllegalArgumentException("Board dimensions must be positive.");
        }

        if (blocks == null || blocks.size() < 1) {
            throw new IllegalArgumentException("There must be at least one block.");
        }

        for (Block block : blocks) {
            if (block == null || block.getShape() == null) {
                throw new IllegalArgumentException("Block and its shape must not be null.")
            }

            boolean[][] shape = block.getShape();
            boolean hasTrue = false;
            for (int i = 0; i < shape.length; i++) {
                for (int j = 0; j < shape[i].length; j++) {
                    if (shape[i][j]) {
                        hasTrue = true;
                        break;
                    }
                }
                if (hasTrue) break;
            }

            if (!hasTrue) {
                throw new IllegalArgumentException("Block must have at least one cell.");
            }
        }
    }
```



```java
public boolean solve() {
    System.out.println("Starting solve with " + blocks.size() + " blocks");
    return solveRecursive(0);
}

private boolean solveRecursive(int depth) {
    if (isAllBlocksUsed()) {
        return true;
    }

    int emptyCells = calculateEmptyCells();

    int remainingNeededCells = calculateRemainingNeededCells();

    if (emptyCells < remainingNeededCells) {
        return false;
    }

    for (int blockIndex = 0; blockIndex < blocks.size(); blockIndex++) {
        if (used[blockIndex]) continue;

        Block originalBlock = blocks.get(blockIndex);
        used[blockIndex] = true;

        // rotation
        Block currentBlock = originalBlock;
        for (int rotation = 0; rotation < 4; rotation++) {
            // normal and mirrored
            Block normalBlock = currentBlock;
            Block mirroredBlock = currentBlock.mirror();

            for (int flip = 0; flip < 2; flip++) {
                Block blockToTry = (flip == 0) ? normalBlock : mirroredBlock;

                // position
                for (int row = 0; row < board.getRow(); row++) {
                    for (int col = 0; col < board.getColumn(); col++) {
                        iterationCount++;

                        if (board.canPlaceBlock(blockToTry, row, col)) {
                            board.placeBlock(blockToTry, row, col);

                            if (solveRecursive(depth + 1)) {
                                return true;
                            }

                            board.removeBlock(blockToTry, row, col);
                        }
                    }
                }
            }
            currentBlock = currentBlock.rotate();
        }
        used[blockIndex] = false;
    }
    return false;
}
```

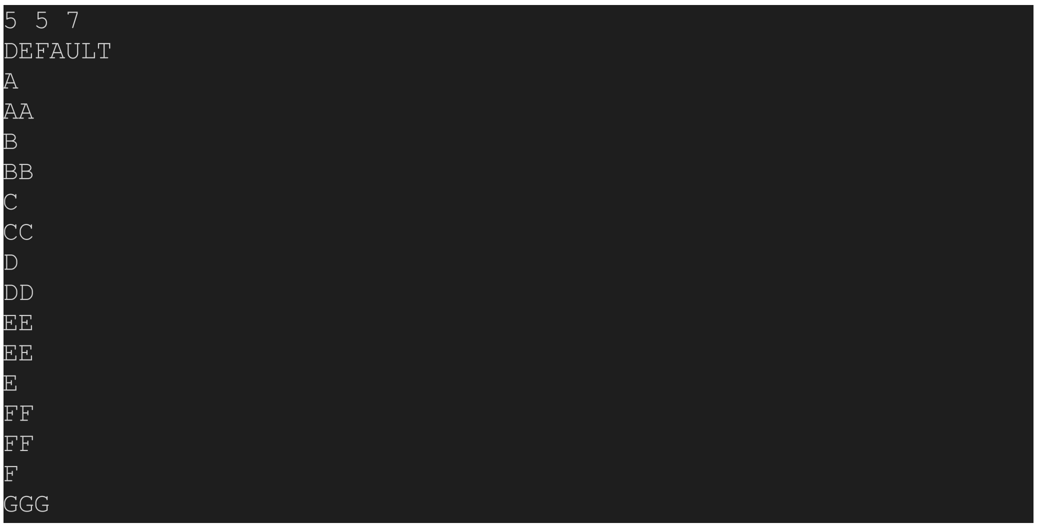


```java
    private int calculateEmptyCells() {
        int empty = 0;
        for (int i = 0; i < board.getRow(); i++) {
            for (int j = 0; j < board.getColumn(); j++) {
                if (board.getCell(i, j) == '.') {
                    empty++;
                }
            }
        }
        return empty;
    }

    private int calculateRemainingNeededCells() {
        int needed = 0;
        for (int i = 0; i < blocks.size(); i++) {
            if (!used[i]) {
                boolean[][] shape = blocks.get(i).getShape();
                for (int row = 0; row < shape.length; row++) {
                    for (int col = 0; col < shape[row].length; col++) {
                        if (shape[row][col]) {
                            needed++;
                        }
                    }
                }
            }
        }
        return needed;
    }

    private boolean isAllBlocksUsed() {
        for (boolean isUsed : used) {
            if (!isUsed) return false;
        }
        return true;
    }

    public long getIterationCount() {
        return iterationCount;
    }
}
```

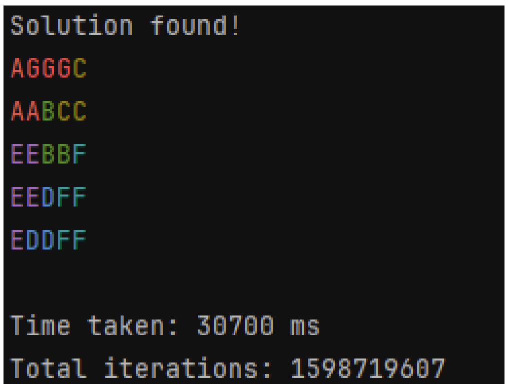
Bab IV Screenshot

1. Test1

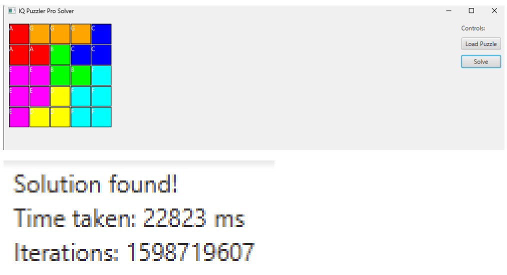
valid



```
5 5 7
DEFAULT
A
AA
B
BB
C
CC
D
DD
EE
EE
E
FF
FF
F
GGG
```



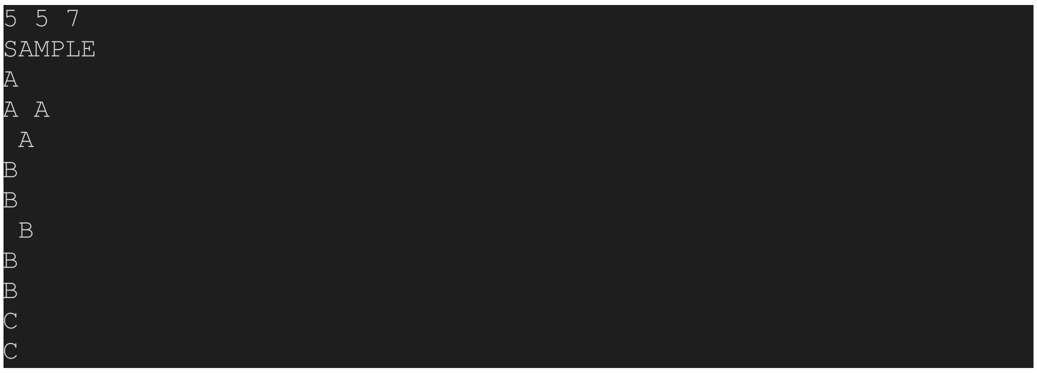
```
Solution found!
AGGGC
AABCC
EEBBF
EEDFF
EDDFF

Time taken: 30700 ms
Total iterations: 1598719607
```



Solution found!
Time taken: 22823 ms
Iterations: 1598719607

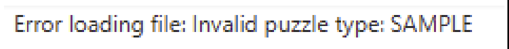
2. Test2

invalid puzzle type

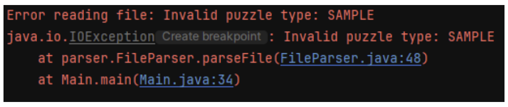


```
5 5 7
SAMPLE
A
A A
 A
B
B
 B
B
B
C
C
```

```
C
D
D
D
E
E
 E
FFF
 F
GG
G
```



```
Error loading file: Invalid puzzle type: SAMPLE
```

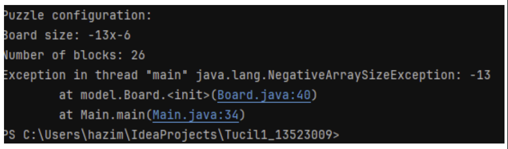


```
Error reading file: Invalid puzzle type: SAMPLE
java.io.IOException Create breakpoint : Invalid puzzle type: SAMPLE
    at parser.FileParser.parseFile(FileParser.java:48)
    at Main.main(Main.java:34)
```

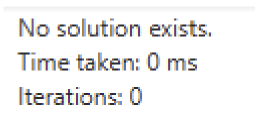
3. Test3

invalid board dimension

```
-5 -5 7
DEFAULT
A
AA
B
BB
C
CC
D
DD
EE
EE
E
FF
FF
F
GGG
```



```
Puzzle configuration:
Board size: -13x-6
Number of blocks: 26
Exception in thread "main" java.lang.NegativeArraySizeException: -13
        at model.Board.<init>(Board.java:40)
        at Main.main(Main.java:34)
PS C:\Users\hazim\IdeaProjects\Tucil1_13523009>
```

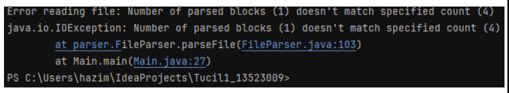


```
No solution exists.
Time taken: 0 ms
Iterations: 0
```

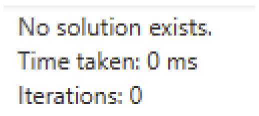
4. Test4

block count doesn't matched parsed blocks

```
4 4 4
DEFAULT
AA
AA
AA
AA
AA
AA
```



```
Error reading file: Number of parsed blocks (1) doesn't match specified count (4)
java.io.IOException: Number of parsed blocks (1) doesn't match specified count (4)
	at parser.FileParser.parseFile(FileParser.java:103)
	at Main.main(Main.java:27)
PS C:\Users\hazim\IdeaProjects\Tucil1_13523009>
```

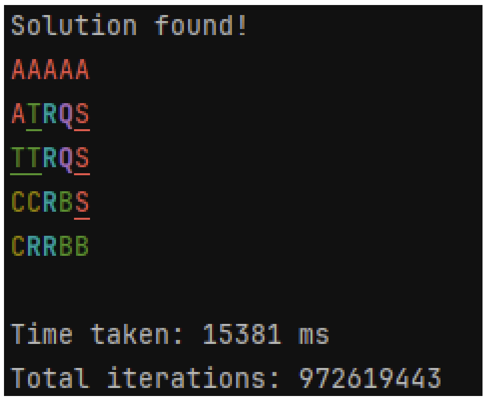


```
No solution exists.
Time taken: 0 ms
Iterations: 0
```

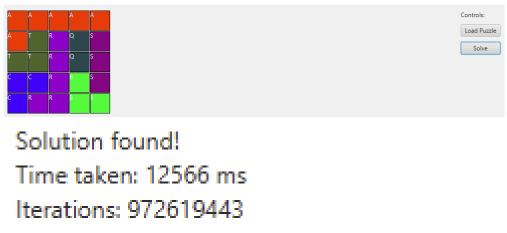
5. Test5

valid

```
5 5 7
DEFAULT
AAAAA
A
R
R
R
RR
QQ
B
BB
SSS
CC
C
TT
T
```



```
Solution found!
AAAAA
ATRQS
TTRQS
CCRBS
CRRBB

Time taken: 15381 ms
Total iterations: 972619443
```



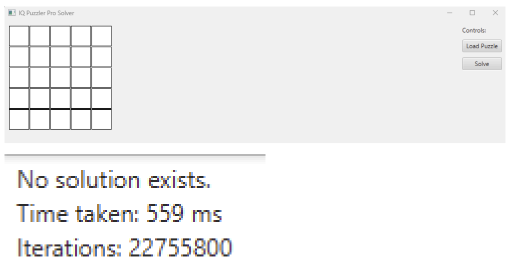
```
Solution found!
Time taken: 12566 ms
Iterations: 972619443
```

6. Test6

blocks are not connected

```
5 5 3
DEFAULT
A

B
BB
C
CC
```

```
No solution exists.
Time taken: 610 ms
Total iterations: 22755800
```



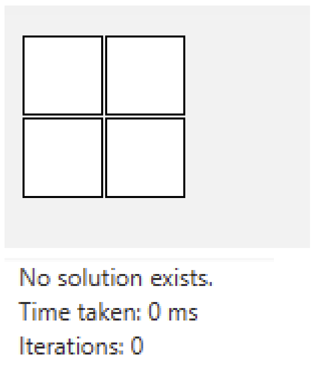
No solution exists.
Time taken: 559 ms
Iterations: 22755800

7. Test7

total block cell exceed board size

```
2 2 2
DEFAULT
AAAA
AAAA
B
BB
```

```
Warning: Total block cells (11) exceed board size (4). No solution possible.
Starting solve with 2 blocks

No solution exists.
Time taken: 0 ms
Total iterations: 0
```

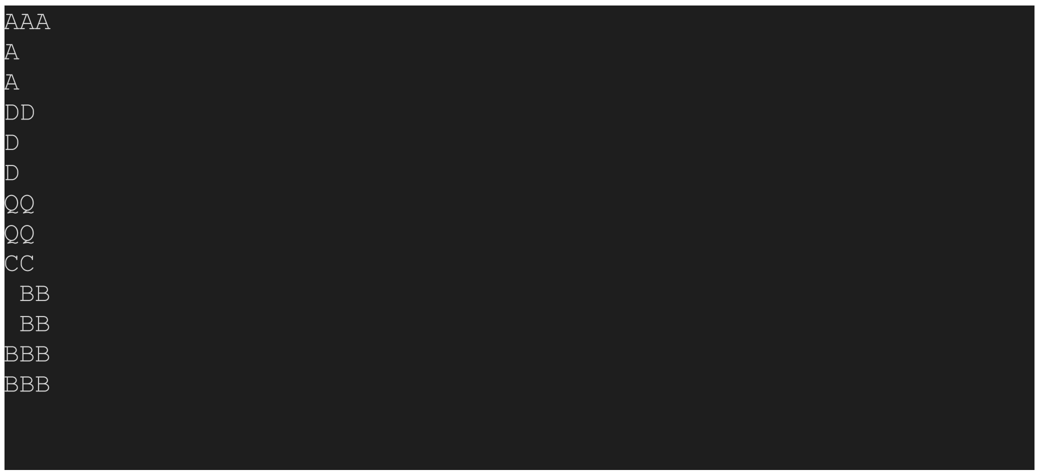


No solution exists.
Time taken: 0 ms
Iterations: 0

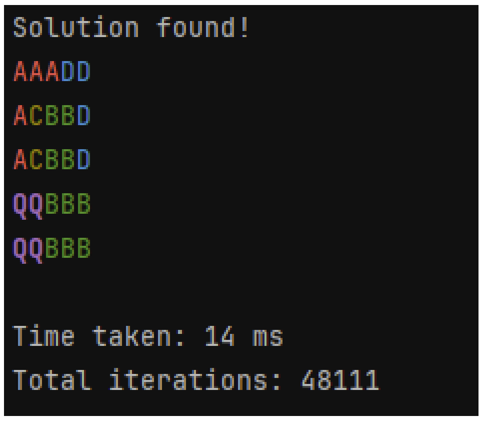
8. Test8

valid

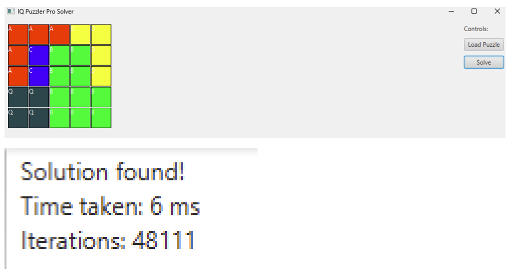
```
5 5 5
DEFAULT
```



```
AAA
A
A
DD
D
D
QQ
QQ
CC
 BB
 BB
BBB
BBB
```



```
Solution found!
AAADD
ACBBD
ACBBD
QQBBB
QQBBB

Time taken: 14 ms
Total iterations: 48111
```



Solution found!
Time taken: 6 ms
Iterations: 48111

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	✅	

Github Repository :

https://github.com/SayyakuHajime/Tucil1_13523009

“hari pertama sudah dikasih tucil…”