Laporan Tugas Kecil 1 IF2211 Strategi Algoritma

Semester II tahun 2024/2025

Penyelesaian IQ Puzzler Pro dengan Algoritma Brute Force

13523089 - Ahmad Ibrahim

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2024

1. Algoritma

Algoritma *Brute Force* yang digunakan adalah mengeksplorasi seluruh permutasi dari posisi dan arah dari seluruh blok yang diberikan. Tinggi papan adalah N dan lebar papan adalah M. Jumlah blok adalah P, dengan P maksimal adalah 26, yaitu banyak dari semua alfabet. Terdapat 8 kemungkinan arah blok, yaitu posisi awal, rotasi 90 derajat, rotasi 180 derajat, rotasi 270 derajat, dan refleksi dari keempat posisi tersebut. Jadi kompleksitas yang didapatkan dari algoritma Brute Force-nya adalah $O(8^P NM)$ atau bisa juga ditulis $O(2^P NM)$.

Langkah-langkah yang dilakukan untuk melakukan algoritma *brute force* di atas adalah sebagai berikut:

1. Menentukan arah semua blok dengan membentuk sebuah *list of integer* yang pada awalnya berisi $\{0, 0, \dots, 0\}$ sepanjang P, yaitu list untuk menentukan arah dari setiap blok.
2. Membuat matriks boolean bernama *visited* yang akan digunakan untuk mencatat apakah blok ke-i pernah dimulai pada koordinat tersebut.
3. Iterasi seluruh papan dengan mencoba untuk meletakkan dan mencatat koordinat sebuah blok.
4. Setelah semua iterasi terjadi, melakukan pengecekan apakah masih terdapat kekosongan di papan. Jika masih terdapat kekosongan, maka dilakukan iterasi kembali dengan kombinasi arah blok selanjutnya seperti contoh $\{0, 0, \dots, 1\}$ hingga maksimal $\{7, 7, \dots, 7\}$. Jika kombinasi telah mencapai maksimal dan masih terdapat kekosongan di papan, maka solusi tidak ditemukan.

2. Source Code

App.java

```
package com.gui;

import java.io.IOException;

import atlantafx.base.theme.PrimerDark;
import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.image.Image;
import javafx.stage.Stage;

/**
 * JavaFX App
 */
public class App extends Application {

    private static Scene scene;

    @Override
    public void start(@SuppressWarnings("exports") Stage stage)
throws IOException {
        Application.setUserAgentStylesheet(new
PrimerDark().getUserAgentStylesheet());

        scene = new Scene(loadFXML("solver"), 1080, 900);
        stage.getIcons().add(new
Image(getClass().getResourceAsStream("/icons/images.jpg")));
        stage.setScene(scene);
        stage.setTitle("IQ PUZZLER PRO SOLVER");
        stage.centerOnScreen();
        stage.show();
```

```
    }

    static void setRoot(String fxml) throws IOException {
        scene.setRoot(loadFXML(fxml));
    }

    private static Parent loadFXML(String fxml) throws IOException
{
        FXMLLoader fxmlLoader = new
FXMLLoader(App.class.getResource(fxml + ".fxml"));
        return fxmlLoader.load();
    }

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

}
```

Solver.java

```
package com.gui;

import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Random;
import java.util.Scanner;
import java.awt.image.BufferedImage;
import java.awt.image.DataBufferInt;

import javax.imageio.ImageIO;
```

```java
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.fxml.FXML;
import javafx.geometry.VPos;
import javafx.scene.control.ComboBox;
import javafx.scene.control.TextArea;
import javafx.scene.control.TextField;
import javafx.scene.image.WritableImage;
import javafx.scene.control.Label;
import javafx.scene.paint.Color;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;
import javafx.scene.text.TextAlignment;
import javafx.stage.FileChooser;
import javafx.stage.Stage;
import javafx.scene.SnapshotParameters;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.control.Button;

public class Solver {

    @FXML
    private ComboBox<String> CaseField;
    @FXML
    private TextField NField;
    @FXML
    private TextField MField;
    @FXML
    private TextField PField;
    @FXML
    private Label ErrorField;
    @FXML
    private Label Announcement;
    @FXML
    private Canvas Image;
    @FXML
    private TextArea PiecesField;
    @FXML
```

```
    private TextArea CustomField;
    @FXML
    private Button SolveButton;
    @FXML
    private Button SaveButton;

    FileChooser fc = new FileChooser();

    private static int N, M, P;
    private static final char EMPTY = '_';
    private static final char PADDING = '.';
    private static char[][] board;
    private static final List<Piece> pieces = new ArrayList<>();

    private static int totalIterations = 0;

    static class Piece {
        char symbol;
        int[][] shape;

        Piece(char symbol, int[][] shape) {
            this.symbol = symbol;
            this.shape = shape;
        }

        Piece transform(int type) {
            int[][] tempShape =
Arrays.stream(this.shape).map(int[]::clone).toArray(int[][]::new);
            Piece newPiece = new Piece(symbol, tempShape);

            switch (type) {
                case 1: {
                    for (int[] coord : newPiece.shape) {
                        int temp = coord[0];
                        coord[0] = coord[1];
                        coord[1] = -temp;
                    }
                    break;
                }
```

```
                case 2: {
                    for (int[] coord : newPiece.shape) {
                        coord[0] *= -1;
                        coord[1] *= -1;
                    }
                    break;
                }
                case 3: {
                    for (int[] coord : newPiece.shape) {
                        int temp = coord[1];
                        coord[1] = coord[0];
                        coord[0] = -temp;
                    }
                    break;
                }
                case 4: {
                    for (int[] coord : newPiece.shape) {
                        coord[1] = -coord[1];
                    }
                    break;
                }
                case 5: {
                    for (int[] coord : newPiece.shape) {
                        int temp = coord[0];
                        coord[0] = coord[1];
                        coord[1] = temp;
                    }
                    break;
                }
                case 6: {
                    for (int[] coord : newPiece.shape) {
                        coord[0] *= -1;
                    }
                    break;
                }
                case 7: {
                    for (int[] coord : newPiece.shape) {
                        int temp = coord[1];
                        coord[1] = -coord[0];
```

```
                        coord[0] = -temp;
                    }
                    break;
                }
            }

            return newPiece;
        }
    }

    private boolean parse() {
        try {
            board = new char[N][M];
            int totalArea = N * M;
            for (int i = 0; i < N; i++) {
                Arrays.fill(board[i], EMPTY);
            }

            String caseType = CaseField.getValue();
            if (caseType.equals("CUSTOM")) {
                totalArea = 0;
                String[] customBoard =
CustomField.getText().split("\n");
                for (int i = 0; i < N; i++) {
                    String line = customBoard[i];

                    for (int j = 0; j < M; j++) {
                        if (line.charAt(j) == 'X') {
                            board[i][j] = EMPTY;
                            totalArea++;
                        } else {
                            board[i][j] = PADDING;
                        }
                    }
                }
            }


            String[] piecesData =
```

```
PiecesField.getText().split("\n");
            int area = 0;
            int idx = 0;
            for (int i = 0; i < P; i++) {
                List<int[]> shapeList = new ArrayList<>();

                char symbol = (char) (i + 'A');
                for (int j = 0; j < piecesData[idx].length(); j++)
{
                    if (piecesData[idx].charAt(j) >= 'A' &&
piecesData[idx].charAt(j) <= 'Z') {
                        symbol = piecesData[idx].charAt(j);
                        break;
                    }
                }

                int row = 0;
                readPiece:
                do {
                    boolean isFirst = true;

                    for (int col = 0; col <
piecesData[idx].length(); col++) {
                        if (piecesData[idx].charAt(col) == symbol)
{
                            isFirst = false;
                            shapeList.add(new int[]{row, col});
                            area++;
                        } else if (piecesData[idx].charAt(col) >=
'A' && piecesData[idx].charAt(col) <= 'Z' && isFirst) {
                            break readPiece;
                        } else if (piecesData[idx].charAt(col) !=
' ') {
                            ErrorField.setText("Karakter tidak
valid.");
                            return false;
                        }
                    }
                    row++;
```

```
                } while (++idx < piecesData.length);

                int[][] shape = shapeList.toArray(new int[0][]);
                pieces.add(new Piece(symbol, shape));
            }

            if (area != totalArea) {
                ErrorField.setText("Jumlah area tidak sesuai.");
                return false;
            }

            return true;
        } catch (Exception e) {
            ErrorField.setText("Error: " + e.getMessage());
        }
        return false;
    }

    private static boolean nextTransformation(int[] trans) {
        int i = trans.length - 1;
        while (i >= 0) {
            if (trans[i] < 7) {
                trans[i]++;
                return true;
            }
            trans[i] = 0;
            i--;
        }
        return false;
    }

    private static boolean place(char[][] localBoard, Piece piece,
int row, int col) {
        for (int[] coord : piece.shape) {
            int r = row + coord[0];
            int c = col + coord[1];
            if (r < 0 || r >= N || c < 0 || c >= M ||
localBoard[r][c] != EMPTY) {
                return false;
```

```
            }
        }
        for (int[] coord : piece.shape) {
            localBoard[row + coord[0]][col + coord[1]] =
piece.symbol;
        }
        return true;
    }

    private static void removePiece(char[][] localBoard, Piece
piece, int row, int col) {
        for (int[] coord : piece.shape) {
            int r = row + coord[0];
            int c = col + coord[1];
            if (r < 0 || r >= N || c < 0 || c >= M ||
localBoard[r][c] == PADDING) {
                continue;
            }
            if (localBoard[r][c] == piece.symbol) {
                localBoard[r][c] = EMPTY;
            }
        }
    }

    private static void printBoard(char[][] localBoard) {
        String[] style = {
            "",
            "\u001B[1m",
            "\u001B[3m",
            "\u001B[1;3m",
        };

        String[] backgrounds = {
            "\u001B[41m",
            "\u001B[42m",
            "\u001B[43m",
            "\u001B[44m",
            "\u001B[45m",
            "\u001B[46m",
```

```
            "\u001B[101m",
            "\u001B[102m",
            "\u001B[103m",
            "\u001B[104m",
            "\u001B[105m",
            "\u001B[106m",
            "\u001B[107m"
        };

        String reset = "\u001B[0m";

        for (char[] row : localBoard) {
            for (char cell : row) {
                if (cell >= 'A' && cell <= 'Z') {
                    String color = style[(cell - 'A') / 12] +
backgrounds[(cell - 'A') % 12];
                    System.out.print(color + cell + reset);
                } else {
                    System.out.print(cell);
                }
            }
            System.out.println();
        }
    }

    private static boolean check(char[][] localBoard) {
        for (char[] row : localBoard) {
            for (char cell : row) {
                if (cell == EMPTY) {
                    return false;
                }
            }
        }
        return true;
    }

    private static boolean solve() {
        int[] perm = new int[P];
```

```
        do {
            boolean[][][] visited = new boolean[P][N][M];

            for (int r = 0; r < N; r++) {
                for (int c = 0; c < M; c++) {
                    char[][] tempBoard =
Arrays.stream(board).map(char[]::clone).toArray(char[][]::new);

                    totalIterations++;

                    for (int i = 0; i < P; i++) {
                        for (int row = 0; row < N; row++) {
                            boolean isPlaced = false;

                            for (int col = 0; col < M; col++) {
                                Piece newPiece =
pieces.get(i).transform(perm[i]);
                                if (!visited[i][row][col] &&
!place(tempBoard, newPiece, row, col)) {
                                    removePiece(tempBoard,
newPiece, row, col);
                                } else {
                                    visited[i][row][col] = true;
                                    isPlaced = true;
                                    break;
                                }
                            }

                            if (isPlaced) break;
                        }
                    }

                    if (check(tempBoard)) {
                        board = tempBoard;
                        return true;
                    }
                }
            }
        } while (nextTransformation(perm));
```

```
        return false;
    }

    private void createImage(Canvas canvas, char[][] board) {
        GraphicsContext gc = canvas.getGraphicsContext2D();
        int cellSize = 50;
        int width = board[0].length * cellSize;
        int height = board.length * cellSize;

        canvas.setWidth(width);
        canvas.setHeight(height);

        Map<Character, Color> colorMap = new HashMap<>();
        Random rand = new Random();

        for (char[] row : board) {
            for (char c : row) {
                if (c != EMPTY && c != PADDING &&
!colorMap.containsKey(c)) {
                    colorMap.put(c, Color.color(rand.nextDouble(),
rand.nextDouble(), rand.nextDouble()));
                }
            }
        }

        for (int y = 0; y < board.length; y++) {
            for (int x = 0; x < board[0].length; x++) {
                Color color;
                if (board[y][x] == EMPTY) {
                    color = Color.WHITE;
                } else if (board[y][x] == PADDING) {
                    color = Color.LIGHTGRAY;
                } else {
                    color = colorMap.get(board[y][x]);
                }

                gc.setFill(color);
                gc.fillRect(x * cellSize, y * cellSize, cellSize,
```

```
cellSize);

                gc.setStroke(Color.BLACK);
                gc.setLineWidth(1);
                gc.strokeRect(x * cellSize, y * cellSize,
cellSize, cellSize);

                if (board[y][x] != EMPTY && board[y][x] !=
PADDING) {
                    gc.setFill(Color.BLACK);
                    gc.setFont(Font.font("Arial", FontWeight.BOLD,
20));
                    gc.setTextAlign(TextAlignment.CENTER);
                    gc.setTextBaseline(VPos.CENTER);

                    double textX = (x * cellSize) + (cellSize /
2);
                    double textY = (y * cellSize) + (cellSize /
2);
                    gc.fillText(String.valueOf(board[y][x]),
textX, textY);
                }
            }
        }
    }

    private void saveImage(Canvas canvas, String filename) {
        WritableImage writableImage = new WritableImage((int)
canvas.getWidth(), (int) canvas.getHeight());
        canvas.snapshot(new SnapshotParameters(), writableImage);

        int width = (int) canvas.getWidth();
        int height = (int) canvas.getHeight();
        BufferedImage bufferedImage = new BufferedImage(width,
height, BufferedImage.TYPE_INT_ARGB);

        int[] buffer = new int[width * height];
        writableImage.getPixelReader().getPixels(0, 0, width,
height, javafx.scene.image.PixelFormat.getIntArgbInstance(),
```

```
buffer, 0, width);

        int[] data = ((DataBufferInt)
bufferedImage.getRaster().getDataBuffer()).getData();
        System.arraycopy(buffer, 0, data, 0, buffer.length);

        File file = new File(filename);
        try {
            ImageIO.write(bufferedImage, "png", file);
            System.out.println("Gambar berhasil disimpan: " +
file.getAbsolutePath());
        } catch (IOException e) {
            System.out.println("Gagal menyimpan gambar: " +
e.getMessage());
        }
    }

    @FXML
    private void driver() {
        N = Integer.parseInt(NField.getText());
        M = Integer.parseInt(MField.getText());
        P = Integer.parseInt(PField.getText());

        if (parse()) {
            System.out.println("Start\n");
            long startTime = System.currentTimeMillis();
            boolean isSolved = solve();
            long endTime = System.currentTimeMillis();
            long duration = endTime - startTime;

            if (isSolved) {
                SaveButton.setDisable(false);
                printBoard(board);
                createImage(Image, board);
                Announcement.setText("Waktu pencarian: " +
duration + " ms || Banyak kasus yang ditinjau: " +
totalIterations);
            } else {
                System.out.println("Tidak ada solusi.");
```

```
            }

            System.out.println("\nWaktu pencarian: " + duration +
" ms");
            System.out.println("\nBanyak kasus yang ditinjau: " +
totalIterations);
            totalIterations = 0;
        } else {
            System.out.println("error");
        }
    }

    @FXML
    private void inputFile() {
        SaveButton.setDisable(true);

        fc.getExtensionFilters().add(new
FileChooser.ExtensionFilter("Text Files", "*.txt"));
        File inputFile = fc.showOpenDialog(new Stage());
        if (inputFile != null) {
            try (Scanner sc = new Scanner(inputFile)) {
                if (sc.hasNextLine()) {
                    String[] dimensions = sc.nextLine().split("
");
                    NField.setText(dimensions[0]);
                    MField.setText(dimensions[1]);
                    PField.setText(dimensions[2]);
                }

                if (sc.hasNextLine()) {
                    String caseType = sc.nextLine();
                    if (caseType.equals("CUSTOM")) {
                        StringBuilder customBoard = new
StringBuilder();
                        for (int i = 0; i <
Integer.parseInt(NField.getText()); i++) {
                            String line = sc.nextLine();
                            customBoard.append(line).append("\n");
                            for (char c : line.toCharArray()) {
```

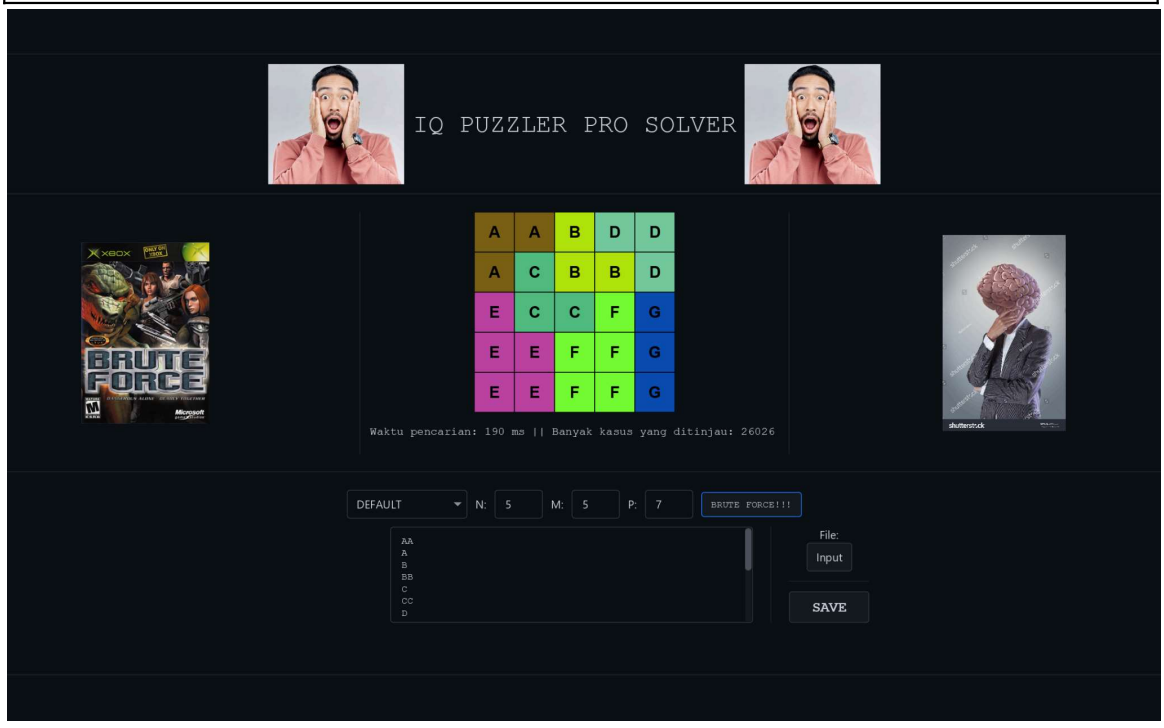
```
                                if (c != 'X' && c != '.') {
                                    System.out.print(c);
                                    ErrorField.setText("Karakter
tidak valid.");
                                    return;
                                }
                            }
                        }
                        CaseField.setValue("CUSTOM");

CustomField.setText(customBoard.toString());
                    } else if (caseType.equals("DEFAULT")) {
                        CaseField.setValue("DEFAULT");
                    } else {
                        ErrorField.setText("Case tidak valid.");
                        return;
                    }
                }

                StringBuilder inputData = new StringBuilder();
                while (sc.hasNextLine()) {
                    String line = sc.nextLine();
                    inputData.append(line).append("\n");
                    for (char c : line.toCharArray()) {
                        if ((c < 'A' || c > 'Z') && c != ' ') {
                            ErrorField.setText("Karakter tidak
valid: " + c);
                            return;
                        }
                    }
                }
                PiecesField.setText(inputData.toString());
            } catch (IOException e) {
                ErrorField.setText("Gagal membaca file: " +
e.getMessage());
            }
        }
    }
```

```
    @FXML
    private void initialize() {
        ObservableList<String> caseOptions =
FXCollections.observableArrayList(
            "DEFAULT", "CUSTOM"
        );
        CaseField.setItems(caseOptions);
        CaseField.setValue("CUSTOM");

        Announcement.setText("");
        ErrorField.setText("");
        SaveButton.setDisable(true);

        SaveButton.setOnAction(e -> {
            File outputFile = fc.showSaveDialog(new Stage());
            if (outputFile != null) {
                saveImage(Image, outputFile.getAbsolutePath());
            }
        });

        CaseField.setOnAction(e -> {
            String selectedCase = CaseField.getValue();

            boolean isCUSTOM = "CUSTOM".equals(selectedCase);
            CustomField.setVisible(isCUSTOM);

            System.out.println("Case dipilih: " + selectedCase);
        });
    }
}
```

3. Uji Kasus

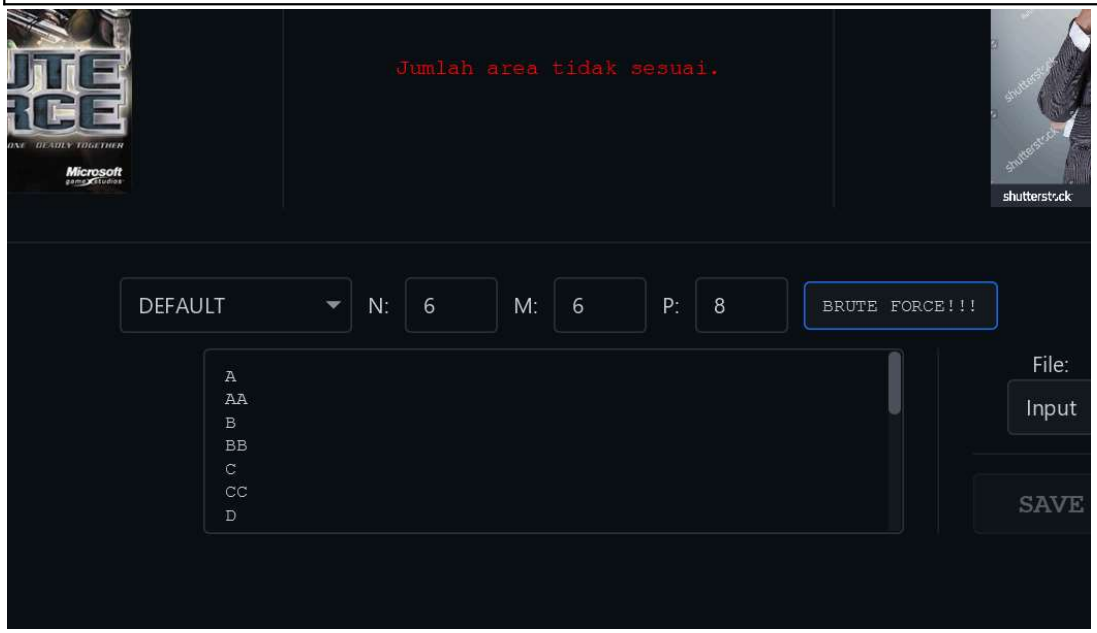
a. Uji Kasus 1

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



b. Uji Kasus 2

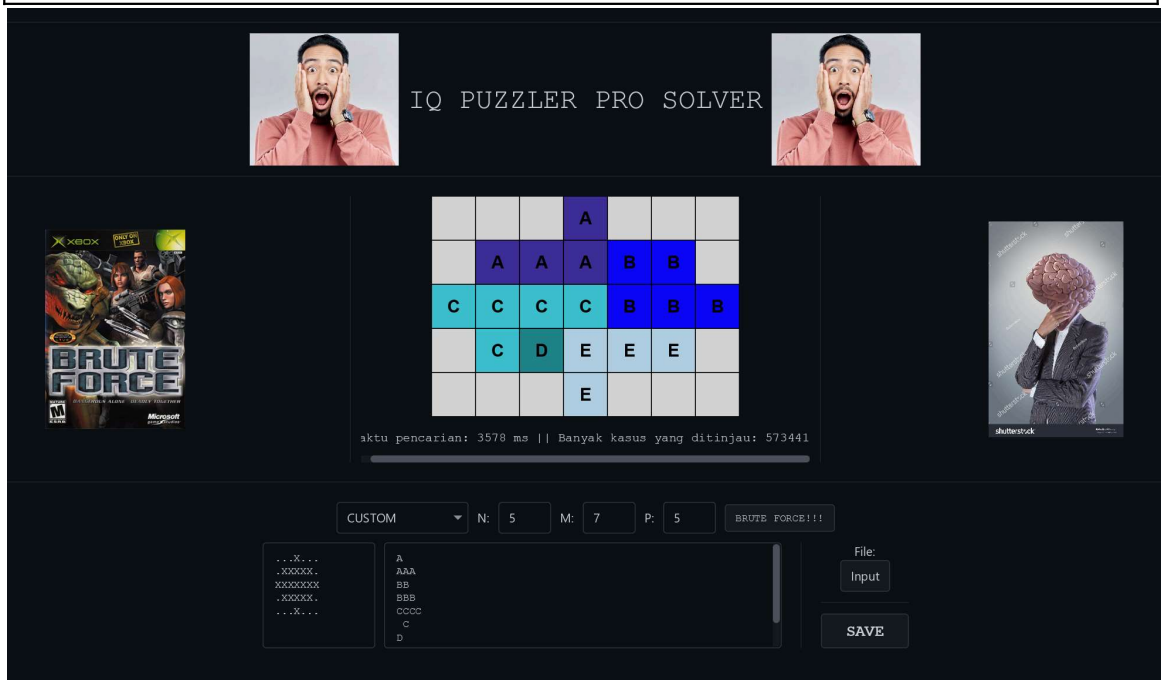
2.txt
6 6 8 DEFAULT A AA B BB C CC D DD EE EE E FF FF F GGG H H H H H



(jumlah area tidak sesuai)

c. Uji Kasus 3

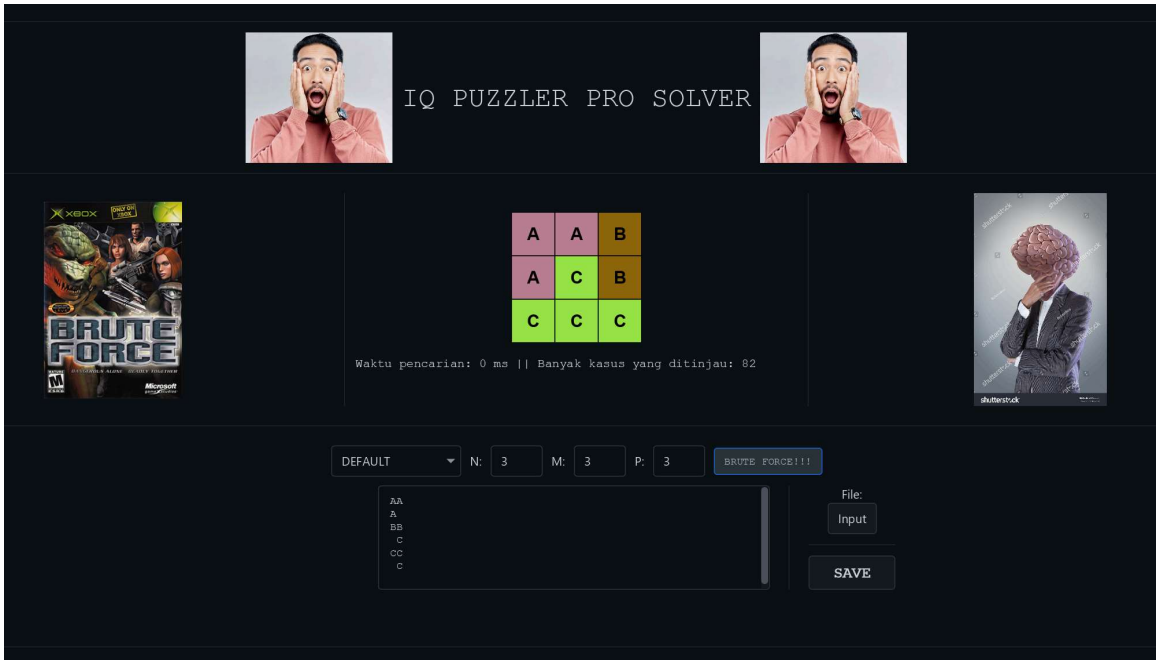
3.txt
5 7 5 CUSTOM ...X... .XXXXX. XXXXXXX .XXXXX. ...X... A AAA BB BBB CCCC C D EEE E



d. Uji Kasus 4

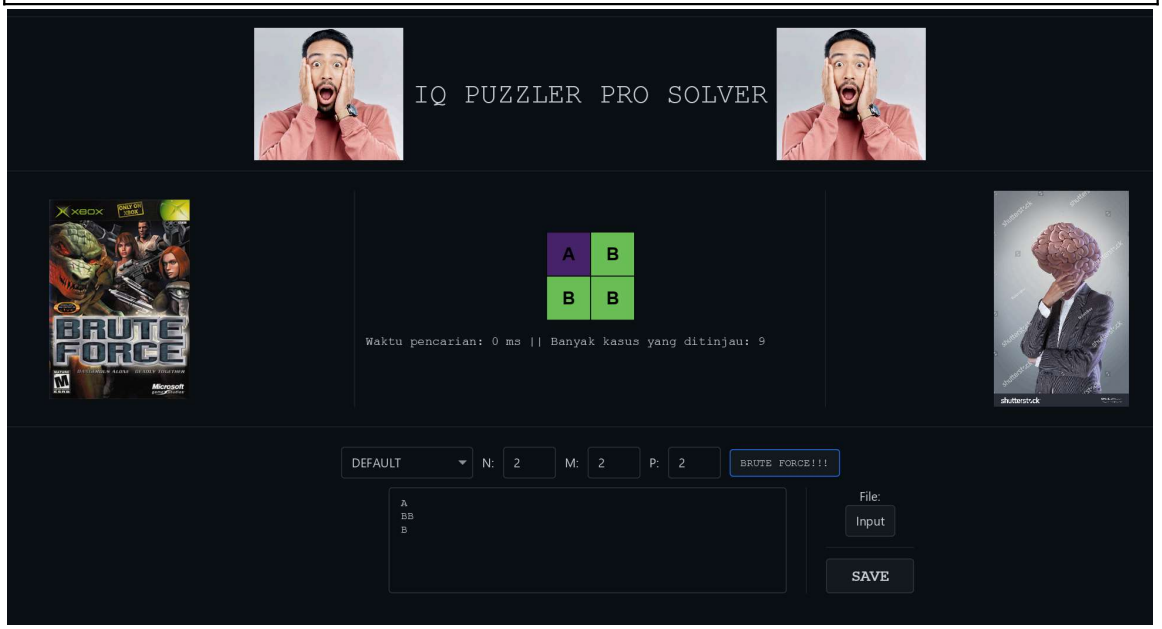
4.txt

3 3 3
DEFAULT
AA
A
BB
  C
CC
  C



e. Uji Kasus 5

5.txt
2 2 2 DEFAULT A BB B

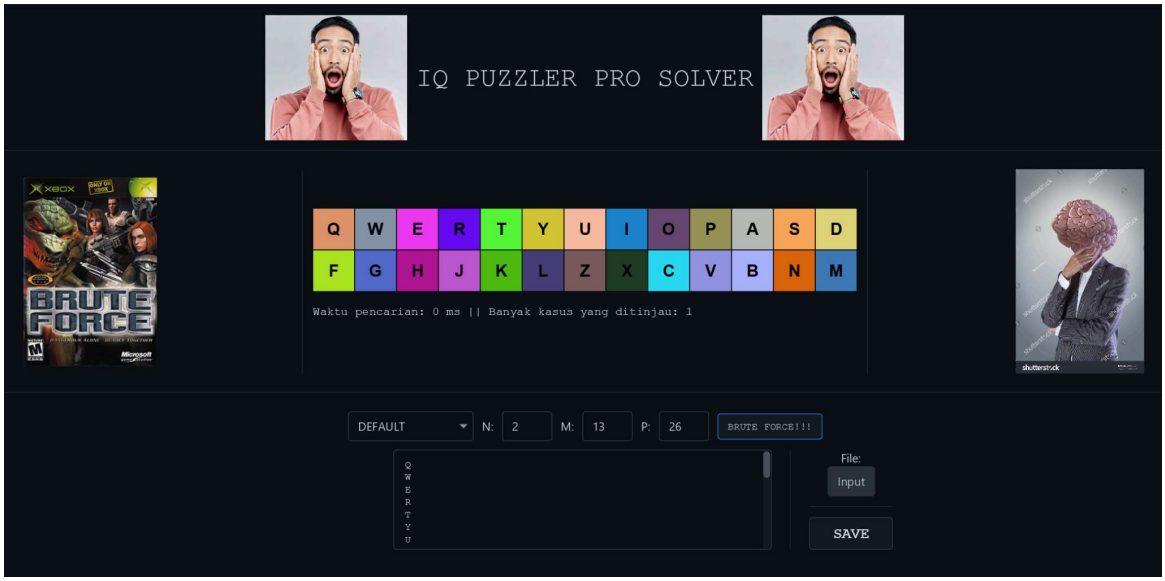


f. Uji Kasus 6

6.txt

```
2 13 26
DEFAULT
Q
W
E
R
T
Y
U
I
O
P
A
S
D
F
G
H
J
K
L
```

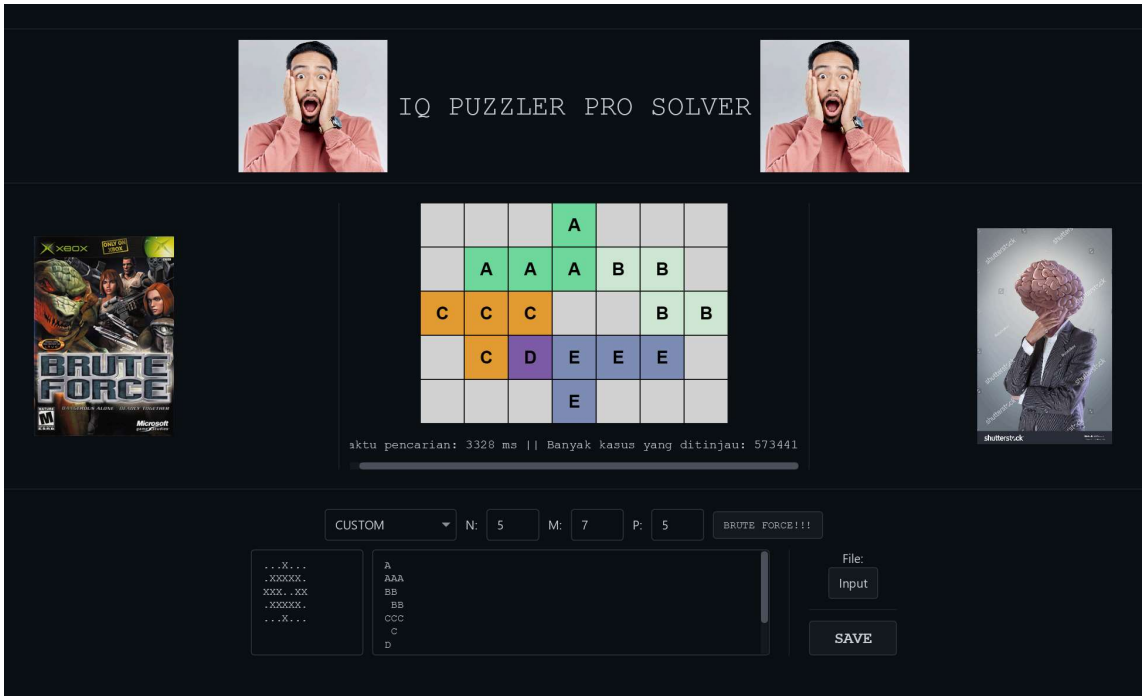
```
Z
X
C
V
B
N
M
```



g. Uji Kasus 7

7.txt

```
5 7 5
CUSTOM
...X...
.XXXXX.
XXX..XX
.XXXXX.
...X...
A
AAA
BB
 BB
CCC
 C
D
EEE
E
```



4. Link Repository

https://github.com/aibrahim185/strategi-algoritma_tucil-1_13523089