LAPORAN TUGAS KECIL 1
IF2211 STRATEGI ALGORITMA

Penyelesaian IQ Puzzler Pro dengan Algoritma Brute Force

Disusun oleh:
Bob Kunanda 13523086

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
2025

Daftar Isi

BAGIAN 1

ALGORITMA BRUTE FORCE

Algoritma *Brute Force* merupakan algoritma yang memiliki cara penyelesaian sangat sederhana karena algoritma tersebut mempetimbangkan semua kemungkinan untuk menyelesaikan suatu permasalahan. Permasalahan yang butuh diselesaikan pada laporan ini adalah penyelesaian *IQ Puzzler Pro* dengan algoritma Brute Force yang memiliki penyelesaian yang serupa dengan *Exhaustive Search*, berikut adalah algoritma *Brute Force* yang dibuat:

1. Ubah semua *piece* menjadi koordinat dengan x pada indeks 0 dan y pada indeks 1 cth: aaa -> menjadi [[0,0],[1,0],[2,0]].
2. Lalu iterasi semua kemungkinan titik pada papan dengan cara iterasi secara horizontal lalu iterasi secara vertikal.
3. Pada titik tersebut uji semua kemungkinan penempatan *piece* tersebut dengan urutan perpindahan *pivot*, rotasi, pencerminan, dan rotasi. Perpindahan pivot dilakukan dengan cara mengubah salah satu titik dari suatu piece menjadi [0,0] sebagai pivot dan membuat yang lain mengikutinya secara relatif. Hal tersebut dilakukan agar dapat valid ketika digunakan dengan rumus transformasi x = -y, y = x. Lalu pencerminan dilakukan karena suatu *piece* dapat 2 sisi penempatannya.
4. Jika *piece* valid untuk dimasukkan, maka papan akan dipasangkan *piece* tersebut.
5. Jika tidak ada penempatan *piece* yang cocok, lanjut pengujian ke *piece* selanjutnya yang belum dipakai dan jika semua *piece* tidak bisa digunakan, lakukan *backtrack* yang menurunkan indeks titik pengujian dan penghapusan *piece* dari papan .
6. Pengujian tersebut dilakukan terus sampai terdapat suatu solusi atau semua kemungkinan *piece* sudah diujikan namun tidak ditemukan solusi.

Setelah *txt* diproses, hasil akan ditampilkan jika ada pada GUI dengan tampilan berbeda warna pada setiap karakter serta tertampil *steps* yang dilakukan serta *runtime*. Pengguna lalu dapat menyimpan hasil dalam bentuk *txt*.

BAGIAN 2

SOURCE PROGRAM

Projek ini menggunakan bahasa *Java*, dengan *library*:

1. *import java.util.Scanner*
2. *import java.io.File*
3. *import java.util.ArrayList*
4. *import java.io.FileNotFoundException*
5. *import java.util.concurrent.atomic.AtomicInteger*
6. *import javax.swing*
7. *import java.awt*
8. *import java.io*

Penyelesaian algoritma menggunakan *file* bernama *functions.java* yang berisi fungsi:

- *isFull*
- *rotate*
- *printBoard*
- *isFit*
- *placePiece*
- *removePiece*
- *ispapanFull*
- *areAllUsed*
- *solvePuzzle*
- *tryPlace*
- *mirror*
- *setPivot*
- *puzzleSolver*

Tampilan GUI menggunakan *file* bernama *PuzzleSolverGUI.java* yang fungsi:

- *PuzzleSolverGUI*
- *chooseFile*
- *getBoardHTML*
- *solvePuzzle*
- *saveSolution*
- *resultToBoard*

Berikut *source* program yang digunakan beserta penjelasannya:

1. *ispapanFull* digunakan untuk menguji apakah papan sudah terisi penuh.

```java
public static boolean isBoardFull(char[][] board) {
    for (int r = 0; r < board.length; r++) {
        for (int c = 0; c < board[0].length; c++) {
            if (board[r][c] == ' ') return false;
        }
    }
    return true;
}
```

2. *rotate* digunakan untuk rotasi suatu *piece* sebanyak 90 derajat.

```java
public static void rotate(ArrayList<ArrayList<Integer>> coordinates) {
    for (int i = 0; i < coordinates.size(); i++) {
        int x = coordinates.get(i).get(index:0);
        int y = coordinates.get(i).get(index:1);
        coordinates.get(i).set(index:0, -y);
        coordinates.get(i).set(index:1, x);
    }
}
```

3. *printBoard* digunakan untuk menampilkan papan dengan warna berbeda pada terminal.

```java
public static String printBoard(char[][] board) {
    StringBuilder result = new StringBuilder();
    String RESET = "\033[0m";
    String[] COLORS =
    {   "\033[31m", "\033[32m", "\033[33m", "\033[34m",
        "\033[35m", "\033[36m", "\033[37m", "\033[90m",
        "\033[91m", "\033[92m", "\033[93m", "\033[94m",
        "\033[95m", "\033[96m", "\033[97m", "\033[1;3;31m",
        "\033[1;3;32m", "\033[1;3;33m", "\033[1;3;34m",  "\033[1;3;35m",
        "\033[1;3;36m",  "\033[1;3;37m", "\033[1;3;90m",  "\033[1;3;91m",
        "\033[1;3;92m",  "\033[1;3;93m"
    };

    for (int i = 0; i < board.length; i++) {
        for (int j = 0; j < board[0].length; j++) {
            char piece = board[i][j];

            if (piece == ' ' || piece == '.') {
                System.out.print(RESET + ' ');
                result.append(c:' ');
            }
            else {
                int colorIndex = (piece - 'A') % COLORS.length;
                System.out.print(COLORS[colorIndex] + piece + RESET);
                result.append(piece);
            }
        }
        result.append(c:'\n');
        System.out.println();
    }
    return result.toString();
}
```

4. *isFit* digunakan untuk menguji apakah suatu *piece* dapat menempati suatu tempat pada papan.

```java
public static boolean isFit(ArrayList<ArrayList<Integer>>  coordinates, char[][] board, int indexX, int indexY) {
    for (int i = 0; i < coordinates.size(); i++) {
        int x = coordinates.get(i).get(index:0) + indexX;
        int y = coordinates.get(i).get(index:1) + indexY;
        if (x < 0 || y < 0 || x >= board.length || y >= board[0].length) {
            return false;
        }
        else{
            if(board[x][y] != ' '){
                return false;
            }
        }
    }
    return true;
}
```

5. *placePiece* digunakan untuk menaruh *piece* pada papan.

```java
public static void placePiece(ArrayList<ArrayList<Integer>> coordinates, char[][] board, int indexX, int indexY, char piece) {
    for (int i = 0; i < coordinates.size(); i++) {
        int x = coordinates.get(i).get(index:0);
        int y = coordinates.get(i).get(index:1);
        board[x + indexX][y + indexY] = piece;
    }
}
```

6. *removePiece* digunakan untuk menghapus suatu *piece* dari papan.

```java
public static void removePiece(ArrayList<ArrayList<Integer>> coordinates, char[][] board, int indexX, int indexY) {
    for (int i = 0; i < coordinates.size(); i++) {
        int x = coordinates.get(i).get(index:0);
        int y = coordinates.get(i).get(index:1);
        board[x + indexX][y + indexY] = ' ';
    }
}
```

7. *areAllUsed* digunakan untuk mengecek apakah semua *piece* sudah digunakan.

```java
public static boolean areAllUsed(ArrayList<Boolean> isUsed) {
    for (boolean used : isUsed) {
        if (!used) return false;
    }
    return true;
}
```

8. *solvePuzzle* merupakan fungsi utama dimana *Brute Force* digunakan untuk mencari jawaban yang cocok.

```java
public static boolean solvePuzzle(ArrayList<Character> characters, ArrayList<ArrayList<ArrayList<Integer>>> pieces, char[][] board, ArrayList<Boolean> isUsed, int row, int col, AtomicInteger steps) {
    if (areAllUsed(isUsed)) return isBoardFull(board);
    if (row >= board.length) return false;
    int nextRow = (col + 1 == board[0].length) ? row + 1 : row;
    int nextCol = (col + 1) % board[0].length;

    if (board[row][col] != ' ') return solvePuzzle(characters, pieces, board, isUsed, nextRow, nextCol,steps);

    boolean foundValidPlacement = false;
    for (int i = 0; i < pieces.size(); i++) {
        if (isUsed.get(i)) continue;

        ArrayList<ArrayList<Integer>> piece = pieces.get(i);

        for (int j = 0; j < piece.size(); j++) {
            setPivot(piece, j);

            for (int rotation = 0; rotation < 4; rotation++) {
                rotate(piece);

                if (tryPlace(characters, pieces, piece, board, isUsed, row, col, i, steps)) {
                    foundValidPlacement = true;
                    return true;
                }

                mirror(piece);
                if (tryPlace(characters, pieces, piece, board, isUsed, row, col, i, steps)) {
                    foundValidPlacement = true;
                    return true;
                }
                mirror(piece);
            }
        }
    }
```

9. *tryPlace* digunakan untuk mencoba penempatan *piece* dan untuk *backtrack*.

```java
public static boolean tryPlace(ArrayList<Character> characters, ArrayList<ArrayList<ArrayList<Integer>>> pieces, ArrayList<ArrayList<Integer>> piece, char[][] board, ArrayList<Boolean> isUsed, int row, int col,
    steps.incrementAndGet();
    if (isFit(piece, board, row, col)) {
        placePiece(piece, board, row, col, characters.get(pieceIndex));
        isUsed.set(pieceIndex, element:true);

        int nextRow = (col + 1 == board[0].length) ? row + 1 : row;
        int nextCol = (col + 1) % board[0].length;

        if (solvePuzzle(characters, pieces, board, isUsed, nextRow, nextCol, steps)) return true;

        removePiece(piece, board, row, col);
        isUsed.set(pieceIndex, element:false);
    }
    return false;
}
```

10. *mirror* digunakan untuk pencerminan *piece*.

```java
public static void mirror(ArrayList<ArrayList<Integer>> coordinates) {
    for (int i = 0; i < coordinates.size(); i++) {
        int x = coordinates.get(i).get(index:0);
        coordinates.get(i).set(index:0, -x);
    }
}
```

11. *setPivot* digunakan untuk mengubah *pivot* dari suatu *piece*.

```java
public static void setPivot(ArrayList<ArrayList<Integer>> coordinates, int index) {
    int xPivot = coordinates.get(index).get(index:0);
    int yPivot = coordinates.get(index).get(index:1);
    for (int i = 0; i < coordinates.size(); i++) {
        coordinates.get(i).set(index:0, coordinates.get(i).get(index:0) - xPivot);
        coordinates.get(i).set(index:1, coordinates.get(i).get(index:1) - yPivot);
    }
}
```

12. *puzzleSolver* digunakan untuk keseluruhan penyelesaiannya mulai dari *parsing txt* sampai mendapatkan papan yang terisi penempatan semua *piece* serta *runtime* dan *steps* yang dibutuhkan.

```java
public static String puzzleSolver (String pathName, int[] totalSteps, int[] runtime){
    long startTime = System.nanoTime();
    ArrayList<ArrayList<ArrayList<Integer>>> pieces = new ArrayList<>();
    ArrayList<ArrayList<Integer>> currentPiece = new ArrayList<>();
    ArrayList<ArrayList<Character>> tempBoard = new ArrayList<>();
    ArrayList<Boolean> isUsed = new ArrayList<>();
    ArrayList<Character> characters = new ArrayList<>();
    int N = 0;
    int M = 0;
    int P = 0;
    String type = "";


    try {
        // INPUT N M P
        File file = new File(pathName);
        Scanner fileScanner = new Scanner(file);
        String line = fileScanner.nextLine();
        char[] lineArray = line.toCharArray();
        String[] parts = line.trim().split(regex:"\\s+");
        N = Integer.parseInt(parts[0]);
        M = Integer.parseInt(parts[1]);
        P = Integer.parseInt(parts[2]);

        //VALIDATE N M P
        if  (N < 1 || M < 1 || P < 1 || P > 26){
            System.out.println(x:"Invalid input.");
            System.exit(status:0);
        }

        //INPUT S
        line = fileScanner.nextLine();
        type = line;
```

```
if  (line.trim().equals(anObject:"CUSTOM")){
    for(int i = 0; i < N; i++){
        line = fileScanner.nextLine();
        lineArray = line.toCharArray();
        ArrayList<Character> row = new ArrayList<>();
        for(int j = 0; j < M; j++){
            row.add(lineArray[j]);
        }
        tempBoard.add(row);
    }
}
else    {
    if(!line.trim().equals(anObject:"DEFAULT")){
        System.out.println(x:"Invalid input.");
        System.exit(status:0);
    }
}

//INPUT PIECES
int x = 0;
int y = 0;
char currentChar = ' ';

while (fileScanner.hasNextLine()) {
    line = fileScanner.nextLine();
    lineArray = line.toCharArray();

    while (x < lineArray.length && lineArray[x] == ' ') {
        x++;
    }

    if (x < lineArray.length && currentChar != lineArray[x]) {
        if (!currentPiece.isEmpty()) {
```

```
                if (!currentPiece.isEmpty()) {
                    pieces.add(currentPiece);
                    isUsed.add(e:false);
                    characters.add(currentChar);
                }
                currentPiece = new ArrayList<>();
                y = 0;
                currentChar = lineArray[x];
            }

            for (int i = x; i < lineArray.length; i++){
                if  (lineArray[i] != ' '){
                    ArrayList<Integer> coordinate = new ArrayList<>();
                    coordinate.add(x);
                    coordinate.add(y);
                    currentPiece.add(coordinate);
                }
                x++;
            }
            y++;
            x = 0;
        }
        pieces.add(currentPiece);
        isUsed.add(e:false);
        characters.add(currentChar);
        fileScanner.close();

    } catch (FileNotFoundException e) {
        System.out.println("File not found: " + e.getMessage());
    }
```

```
char[][] board = new char[N][M];
if  (type.equals(anObject:"DEFAULT")){
    for (int i = 0; i < N; i++){
        for (int j = 0; j < M; j++){
            board[i][j] = ' ';
        }
    }
}
if  (type.equals(anObject:"CUSTOM")){
    for (int i = 0; i < N; i++){
        for (int j = 0; j < M; j++){
            if  (tempBoard.get(i).get(j) == 'X'){
                board[i][j] = ' ';
            }
            else{
                board[i][j] = tempBoard.get(i).get(j);
            }
        }
    }
}

AtomicInteger steps = new AtomicInteger(initialValue:0);
boolean solved = solvePuzzle(characters, pieces, board, isUsed, row:0, col:0, steps);
long endTime = System.nanoTime();
long duration = (endTime - startTime) / 1_000_000;
totalSteps[0] = steps.get();
runtime[0] = (int) duration;
if(solved)  {
    System.out.println("Banyak kasus yang ditinjau: " + steps);
    System.out.println("Waktu pencarian: " + duration + " ms");
    return printBoard(board);
}
```

```
            else{
                System.out.println(x:"No solution found.");
                System.out.println("Banyak kasus yang ditinjau: " + steps);
                System.out.println("Waktu pencarian: " + duration + " ms");
                return "No solution found.";

            }
        }
    }
```

13. *PuzzleSolverGUI* digunakan untuk keseluruhan GUI.

```
public PuzzleSolverGUI() {
    setTitle(title:"Puzzle Solver by Bob Kunanda");
    setSize(width:700, height:600);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setLocationRelativeTo(c:null);
    setLayout(new BorderLayout());

    // Upload Button
    uploadButton = new JButton(text:"Upload Puzzle File");
    uploadButton.setFont(new Font(name:"Arial", Font.BOLD, size:16));
    uploadButton.addActionListener(e -> chooseFile());

    // Solve Button
    solveButton = new JButton(text:"Solve Puzzle");
    solveButton.setFont(new Font(name:"Arial", Font.BOLD, size:16));
    solveButton.setEnabled(b:false);
    solveButton.addActionListener(e -> solvePuzzle());

    // Save Button
    saveButton = new JButton(text:"Save Solution");
    saveButton.setFont(new Font(name:"Arial", Font.BOLD, size:16));
    saveButton.setEnabled(b:false);
    saveButton.addActionListener(e -> saveSolution());

    // Text Area for Output
    textArea = new JTextArea(rows:10, columns:50);
    textArea.setFont(new Font(name:"Monospaced", Font.PLAIN, size:16));
    textArea.setEditable(b:false);
    JScrollPane scrollPane = new JScrollPane(textArea);

    // Board Display
    boardDisplay = new JEditorPane();
    boardDisplay.setContentType(type:"text/html");
    boardDisplay.setEditable(b:false);
    boardDisplay.setText(t:"<html><pre style='font-size:16px;'>No board loaded</pre></html>");
```

```
        JScrollPane boardScroll = new JScrollPane(boardDisplay);

        // Info Labels
        stepLabel = new JLabel(text:"Steps Checked: 0");
        timeLabel = new JLabel(text:"Runtime: 0 ms");

        // Button Panel
        JPanel buttonPanel = new JPanel();
        buttonPanel.add(uploadButton);
        buttonPanel.add(solveButton);
        buttonPanel.add(saveButton);

        // Info Panel
        JPanel infoPanel = new JPanel(new GridLayout(rows:1, cols:2));
        infoPanel.add(stepLabel);
        infoPanel.add(timeLabel);

        add(buttonPanel, BorderLayout.NORTH);
        add(scrollPane, BorderLayout.WEST);
        add(boardScroll, BorderLayout.CENTER);
        add(infoPanel, BorderLayout.SOUTH);

        setVisible(b:true);
    }
```

14. *chooseFile* digunakan untuk mendapatkan *file*.

```
private void chooseFile() {
    JFileChooser fileChooser = new JFileChooser();
    fileChooser.setDialogTitle(dialogTitle:"Select a Puzzle File");
    fileChooser.setFileFilter(new javax.swing.filechooser.FileNameExtensionFilter(description:"Text Files", ...extensions:"txt"));

    int result = fileChooser.showOpenDialog(this);
    if (result == JFileChooser.APPROVE_OPTION) {
        selectedFile = fileChooser.getSelectedFile();
        textArea.setText("Selected File: " + selectedFile.getAbsolutePath() + "\n");
        solveButton.setEnabled(b:true);
        saveButton.setEnabled(b:false);
    }
}
```

15. *getpapanHTML* untuk menampilkan hasil papan.

```java
public static String getBoardHTML(char[][] board) {
    StringBuilder sb = new StringBuilder();
    sb.append(str:"<html><pre style='font-size:16px; font-family:monospace;'>");

    String[] COLORS = {
        "red", "green", "yellow", "blue", "purple", "cyan", "darkblue", "gray",
        "darkred", "darkgreen", "gold", "navy", "magenta", "teal", "silver",
        "brown", "lime", "orange", "indigo", "violet", "aqua", "darkgray"
    };

    for (char[] row : board) {
        for (char piece : row) {
            if (piece == ' ' || piece == '.') {
                sb.append(str:" ");
            } else {
                int colorIndex = (piece - 'A') % COLORS.length;
                sb.append("<font color='" + COLORS[colorIndex] + "'>" + piece + "</font>");
            }
        }
        sb.append(str:"<br>");
    }

    sb.append(str:"</pre></html>");
    return sb.toString();
}
```

16. *solvePuzzle* digunakan untuk memulai penyelesaian *puzzle*.

```java
private void solvePuzzle() {
    if (selectedFile == null) {
        JOptionPane.showMessageDialog(this, message:"No file selected!", title:"Error", JOptionPane.ERROR_MESSAGE);
        return;
    }

    textArea.append(str:"\nSolving Puzzle...\n");

    int[] totalSteps = {0};
    int[] runtime = {0};

    SwingWorker<String, Void> worker = new SwingWorker<>() {
        @Override
        protected String doInBackground() {
            return functions.puzzleSolver(selectedFile.getAbsolutePath(), totalSteps, runtime);
        }

        @Override
        protected void done() {
            try {
                lastSolution = get();
                textArea.append("\n" + lastSolution + "\n");

                stepLabel.setText("Steps Checked: " + totalSteps[0]);
                timeLabel.setText("Runtime: " + runtime[0] + " ms");

                boardDisplay.setText(getBoardHTML(resultToBoard(lastSolution)));

                saveButton.setEnabled(b:true);
            } catch (Exception ex) {
                JOptionPane.showMessageDialog(parentComponent:null, message:"Error solving puzzle!", title:"Error", JOptionPane.ERROR_MESSAGE);
            }
        }
    };

    worker.execute();
}
```

17. *saveSolution* digunakan untuk menyimpan solusi dalam *txt*.

```java
private void saveSolution() {
    if (lastSolution == null || lastSolution.isEmpty()) {
        JOptionPane.showMessageDialog(this, message:"No solution to save!", title:"Error", JOptionPane.ERROR_MESSAGE);
        return;
    }

    try {
        File solutionsDir = new File(pathname:"test/solutions");
        if (!solutionsDir.exists()) {
            solutionsDir.mkdir();
        }
        String fileName = selectedFile.getName().replace(target:".txt", replacement:"_solution.txt");
        File outputFile = new File(solutionsDir, fileName);

        try (BufferedWriter writer = new BufferedWriter(new FileWriter(outputFile))) {
            writer.write(lastSolution);
        }

        JOptionPane.showMessageDialog(this, "Solution saved to: " + outputFile.getAbsolutePath(),
                title:"Success", JOptionPane.INFORMATION_MESSAGE);
    } catch (IOException e) {
        JOptionPane.showMessageDialog(this, message:"Error saving solution!", title:"Error", JOptionPane.ERROR_MESSAGE);
    }
}
```

18. *resultToBoard* digunakan untuk mengubah hasil yang berupa *string* menjadi matriks.

```java
private char[][] resultToBoard(String result) {
    String[] lines = result.split(regex:"\n");
    char[][] board = new char[lines.length][];
    for (int i = 0; i < lines.length; i++) {
        board[i] = lines[i].toCharArray();
    }
    return board;
}
```

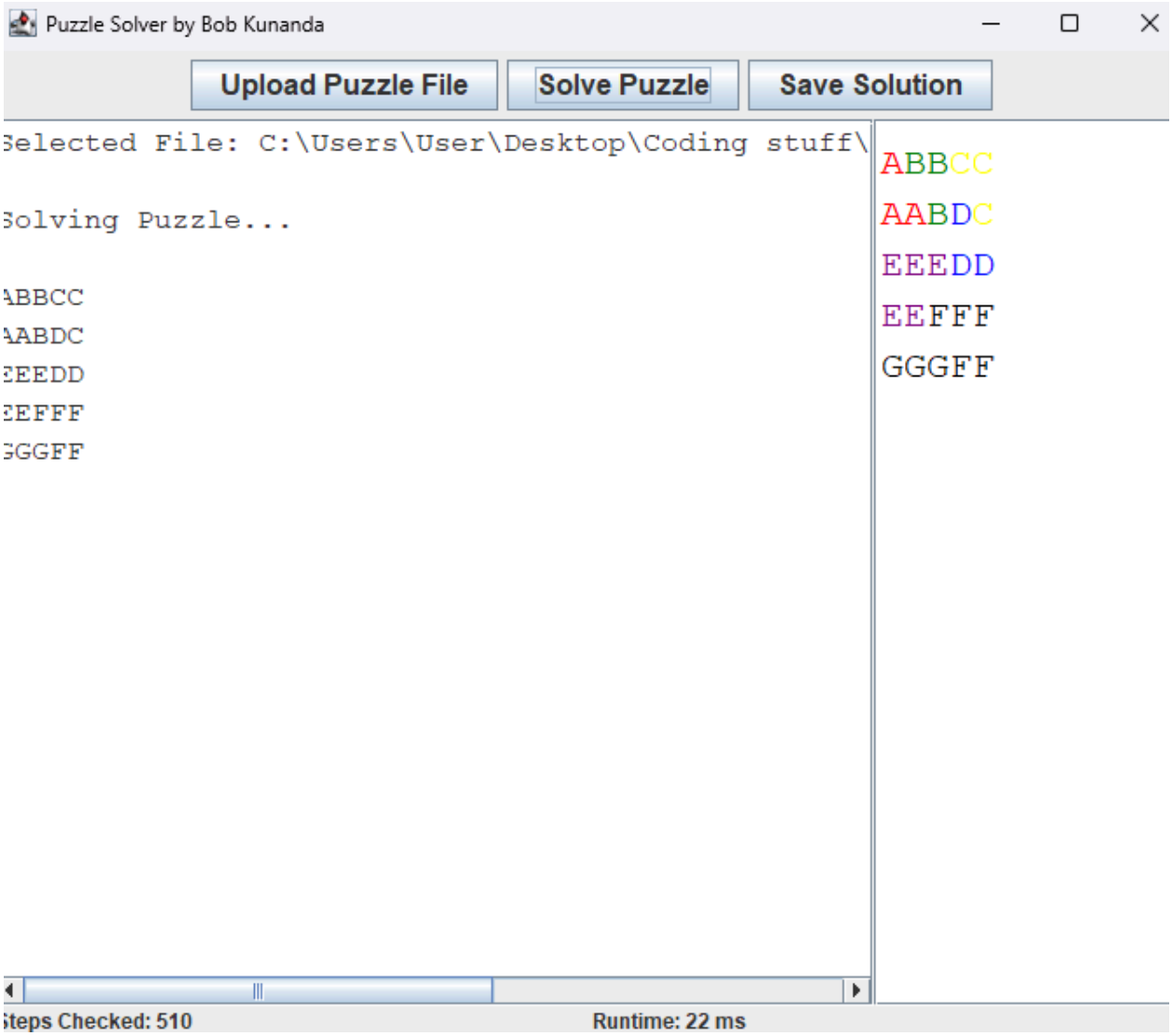
BAGIAN 3

TAMPILAN LAYAR HASIL TEST

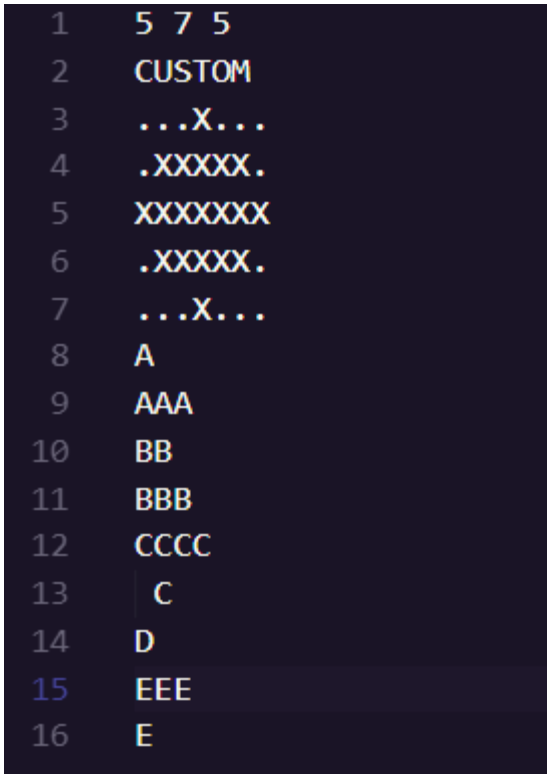
1. *Input*: (Soal dari spek)

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

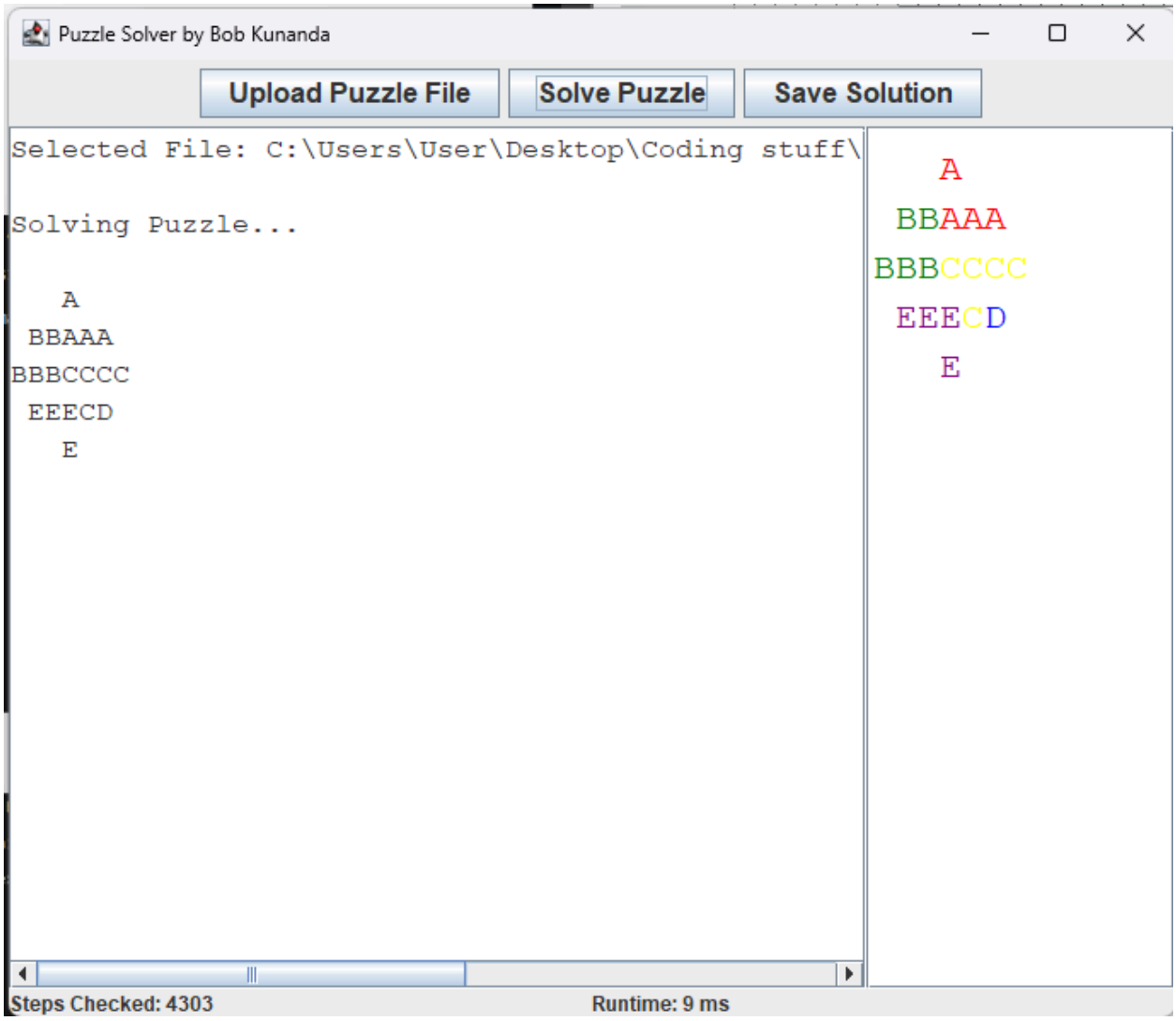
Output:


Puzzle Solver by Bob Kunanda
Upload Puzzle File
Solve Puzzle
Save Solution
Selected File: C:\Users\User\Desktop\Coding stuff\
Solving Puzzle...
ABBCC
AABDC
EEEDD
EEFFF
GGGFF
ABBCC
AABDC
EEEDD
EEFFF
GGGFF
Steps Checked: 510
Runtime: 22 ms

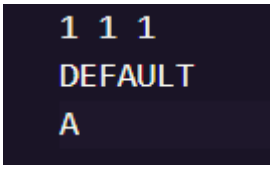
2. *Input*: (Soal dari spek (*custom*))



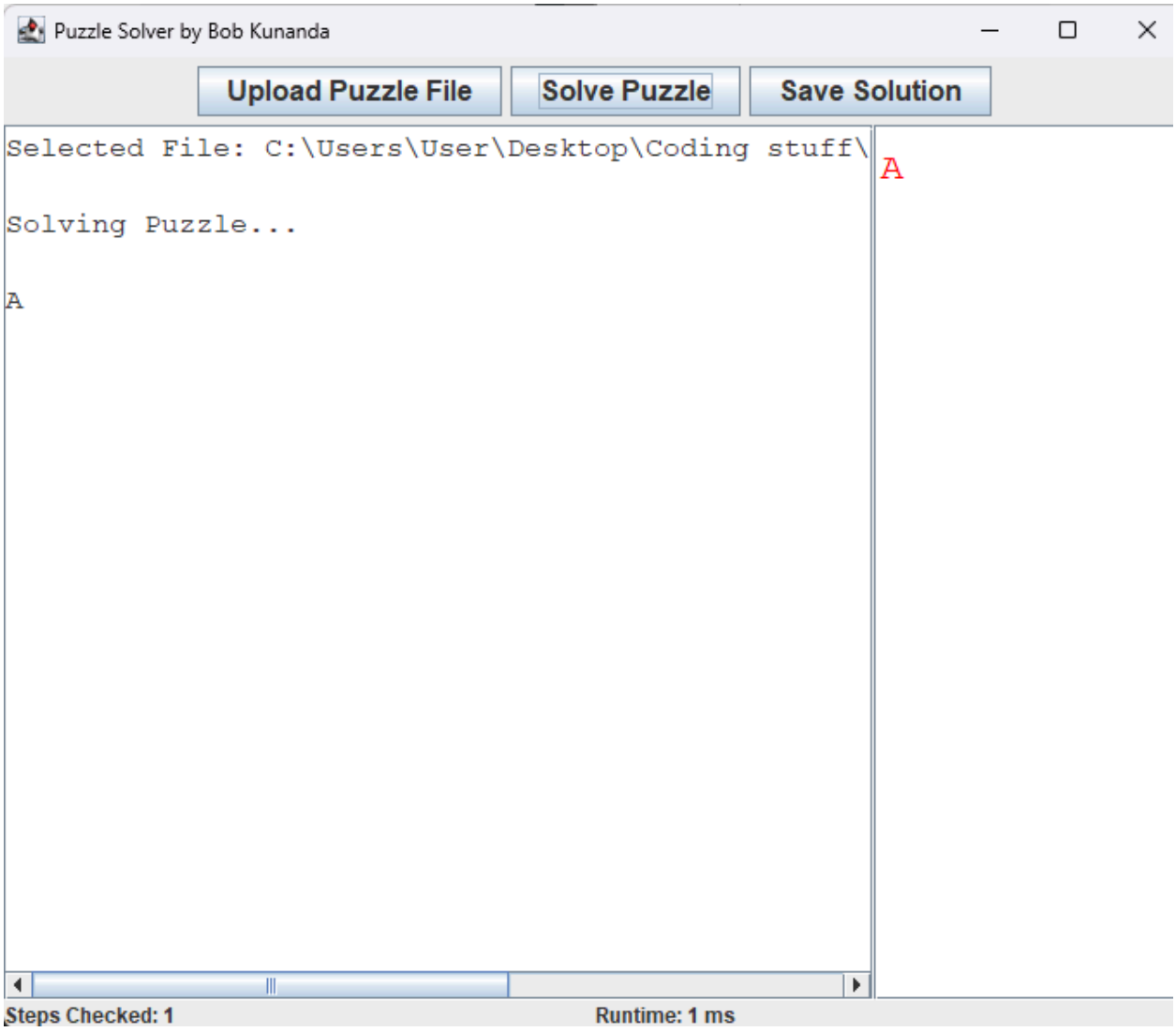
Output:



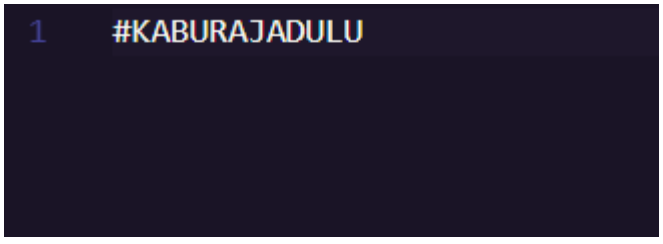
3. *Input*:(test 1 *piece*)



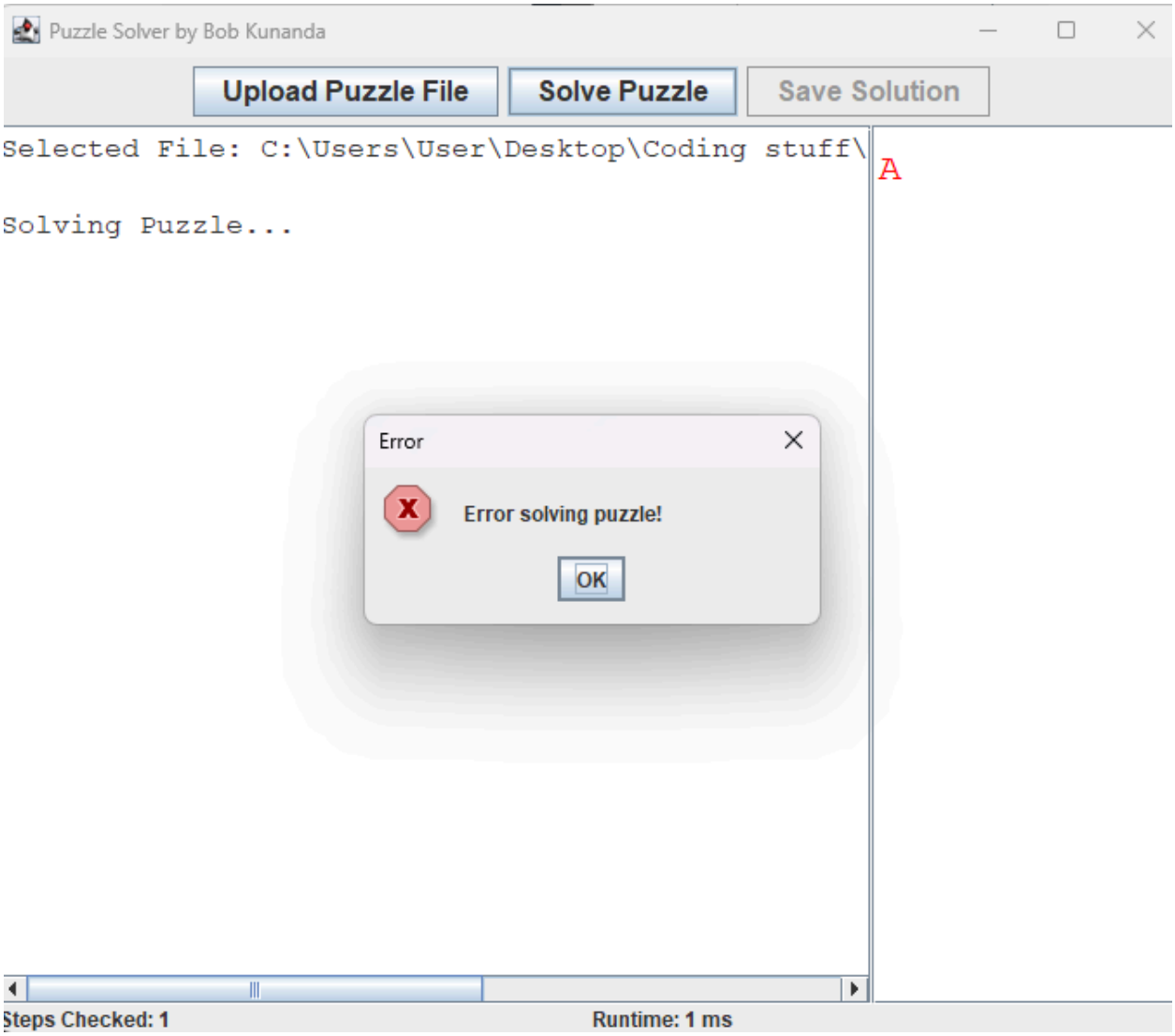
Output:



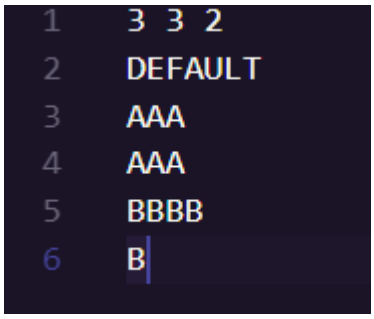
4. *Input*:(*test* salah format)



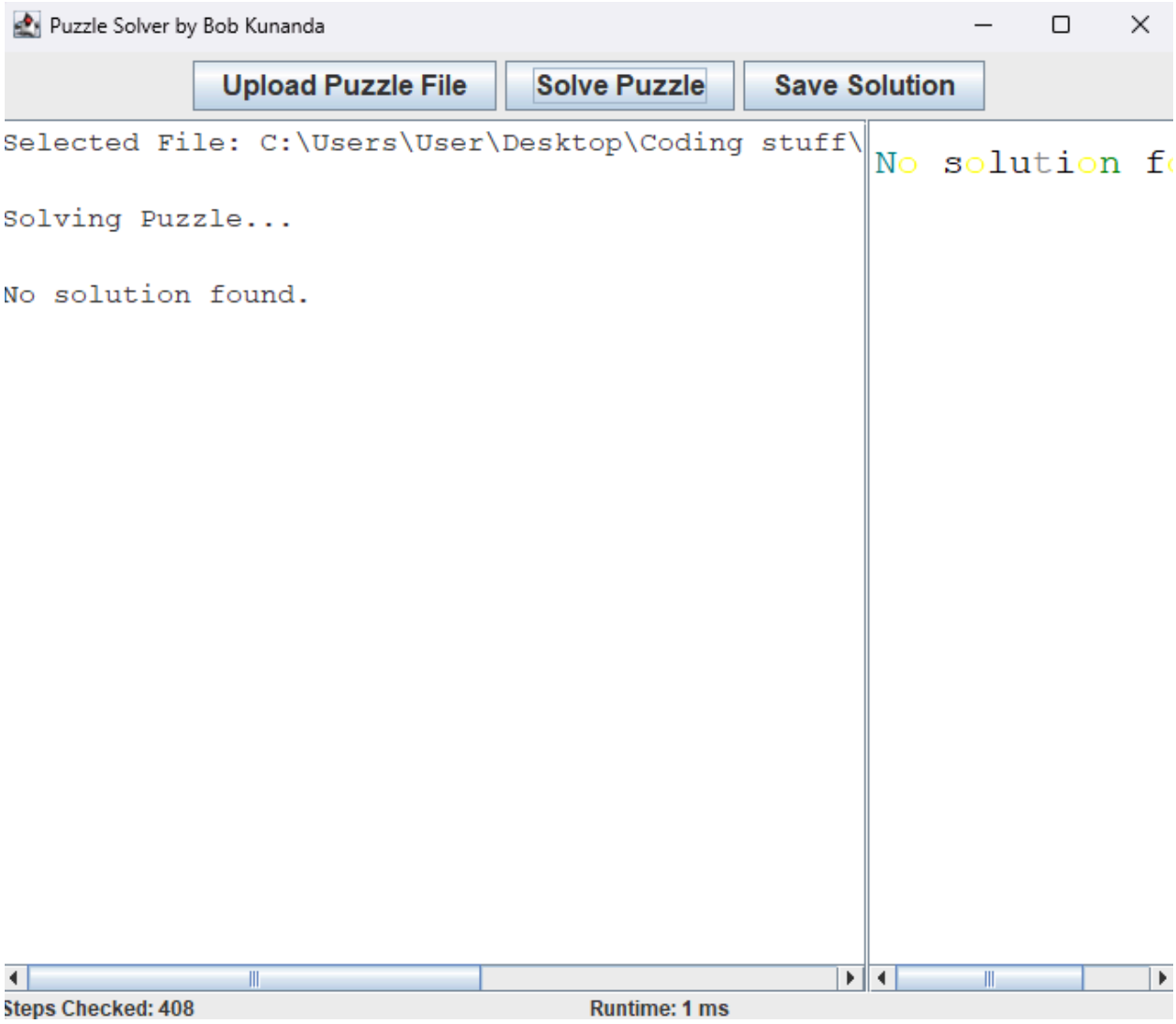
Output:



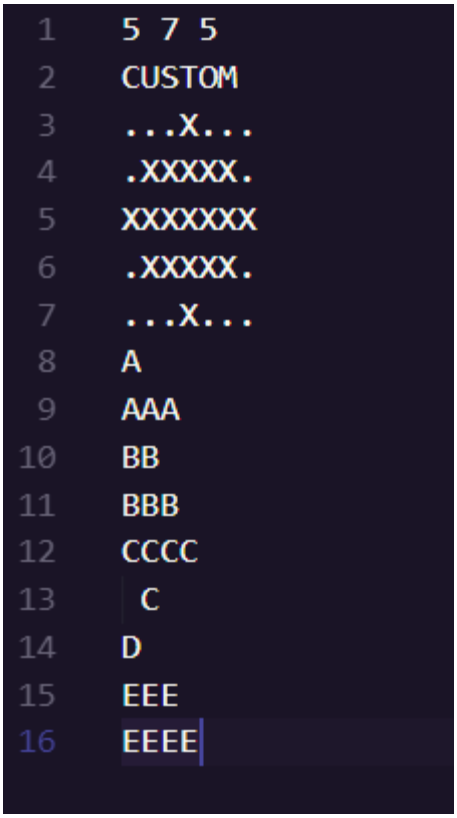
5. *Input*: (*Test no solution*)



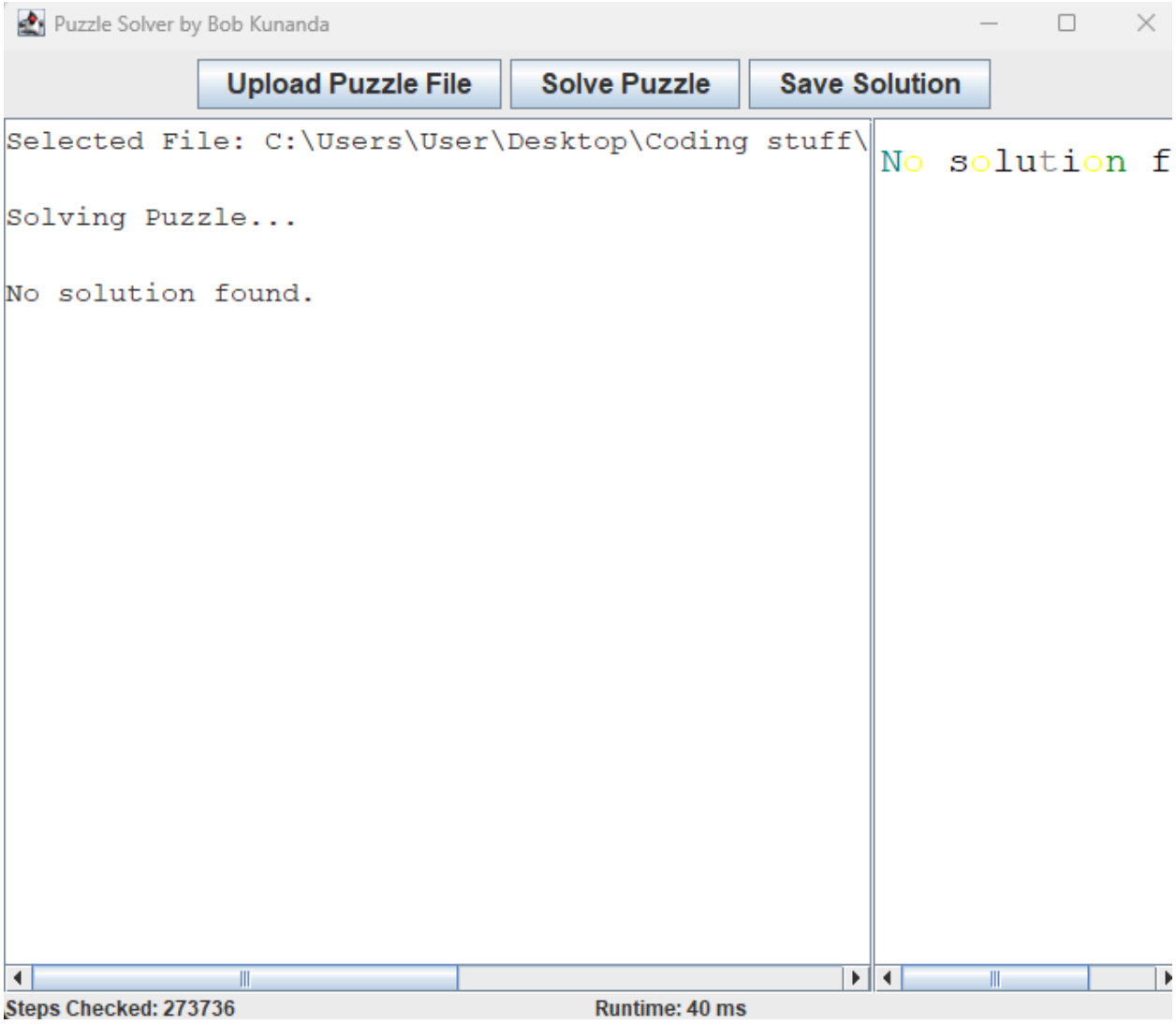
Output:


Puzzle Solver by Bob Kunanda
Upload Puzzle File
Solve Puzzle
Save Solution
Selected File: C:\Users\User\Desktop\Coding stuff\
Solving Puzzle...
No solution found.
No solution f
Steps Checked: 408
Runtime: 1 ms

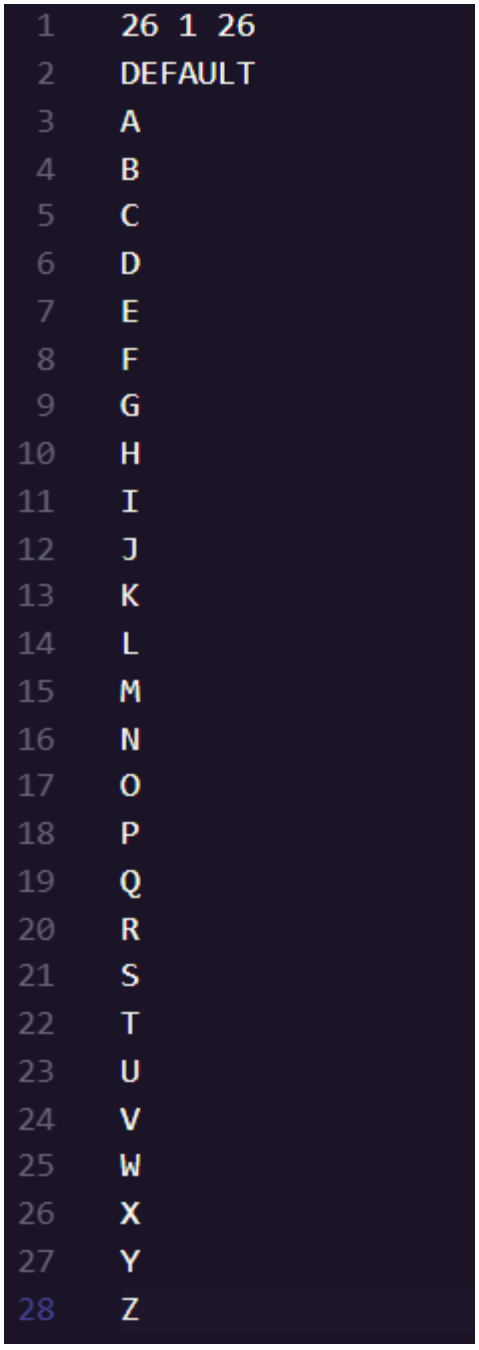
6. *Input*: (*Test no solution (custom)*)



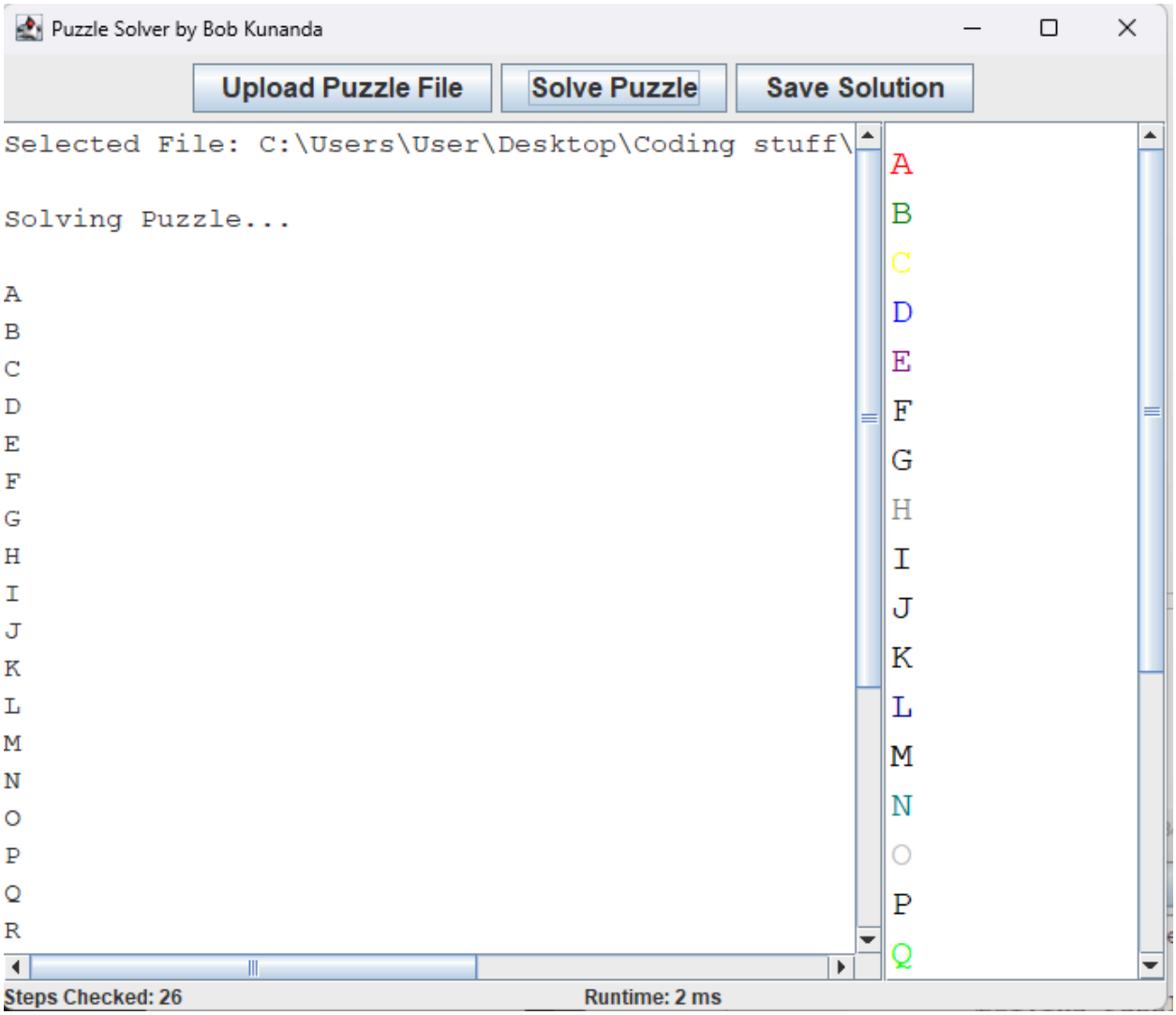
Output:


Puzzle Solver by Bob Kunanda
Upload Puzzle File
Solve Puzzle
Save Solution
Selected File: C:\Users\User\Desktop\Coding stuff\
Solving Puzzle...
No solution found.
No solution f
Steps Checked: 273736
Runtime: 40 ms

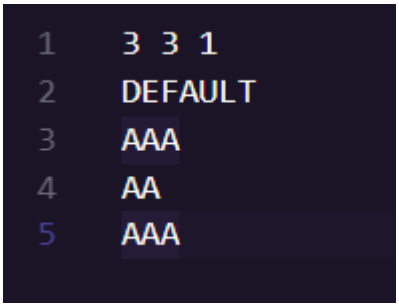
7. *Input*: (*Test all char*)



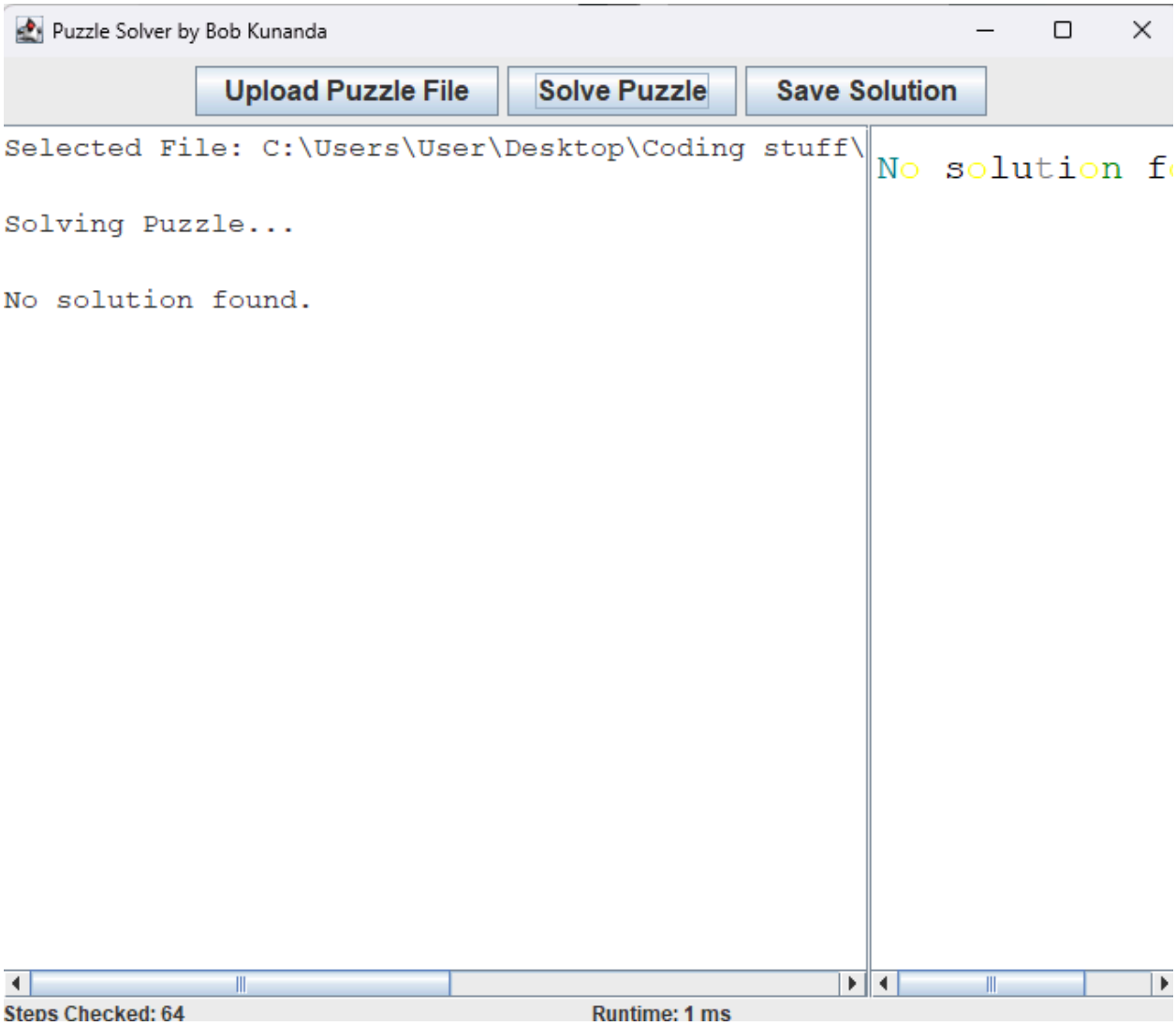
Output:



8. *Input*: (*test no solution (not enough piece)*)



Output:


Puzzle Solver by Bob Kunanda
Upload Puzzle File
Solve Puzzle
Save Solution
Selected File: C:\Users\User\Desktop\Coding stuff\
Solving Puzzle...
No solution found.
No solution f
Steps Checked: 64
Runtime: 1 ms

LAPORAN

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	☑	☐

Repo Github : https://github.com/BobSwagg13/Tucil1_13523086/tree/v1.0