

Laporan Tugas Kecil 1

IF2211 Strategi Algoritma



Disusun oleh:
Aryo Wisanggeni (13523100)

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA INSTITUT
TEKNOLOGI BANDUNG
2024

Daftar Isi

1. Algoritma	3
PseudoCode	3
Deskripsi Algoritma	4
2. Source Code	6
Struktur Program	6
Package Custom	8
File InputPuzzlerPro	8
File OutputPuzzlerPro	13
File InputFormat	17
Package HelperComp	18
File Input	18
File BruteForce	23
File OutputTxt	29
File InputImageGenerator	31
File OutputImage	36
Package GUI	38
File MainFrame	38
File Solution	46
3. Test Case	51
Test case 1 (default)	51
Test Case 2 (default)	53
Test Case 3 (default)	56
Test Case 4 (default)	59
Test Case 5 (default)	61
Test Case 6 (custom)	63
Test Case 7 (custom)	65
Test Case 8 (custom)	68
4. Lampiran	71

1. Algoritma

Penyelesaian puzzle IQ Puzzler Pro dengan algoritma brute force dibuat dengan bahasa pemrograman java. Proses input dan output diproses melalui GUI dengan library swing. Program di-compile menjadi class masing-masing di folder bin dan selanjutnya dijadikan file jar yang digunakan untuk eksekusi program.

PseudoCode

Mulai

Deklarasi piecesList sebagai List

Deklarasi puzzleBoard sebagai papan

Untuk setiap piece dalam piecesList (loop 1)

Tempatkan piece pada puzzleBoard

Jika piece tidak muat

Pindahkan posisi piece

Jika piece muat

Kembali ke langkah loop 1 untuk piece berikutnya

Jika semua posisi telah habis (langkah 4)

Kembali ke posisi awal dan putar potongan 90 derajat searah jarum jam

Lakukan langkah loop 1 lagi

Lakukan langkah 4 sebanyak 3 kali untuk mencoba semua rotasi potongan

Jika semua rotasi sudah dicoba

Refleksikan piece, kemudian lakukan langkah 4 lagi

Lakukan rotasi 3 kali dan coba semua posisi untuk setiap rotasi

Jika pada langkah proses atas papan penuh dan tidak ada piece yang tersisa

Kembalikan solusi sebagai berhasil

Jika pada langkah proses atas papan tidak penuh, tetapi tidak ada piece yang tersisa

Kembalikan bahwa tidak ada solusi yang ditemukan
Selesai

Deskripsi Algoritma

Algoritma bruteforce adalah algoritma yang menyelesaikan masalah dengan cara paling sederhana atau paling lempeng dan jelas dari awal. Keunikannya adalah dapat menyelesaikan semua masalah, hanya saja bisa sangat menghabiskan waktu.

Pada tugas kecil ini, puzzle diselesaikan sesuai dengan deskripsi pseudocode di atas, yaitu dengan cara rekursi untuk implementasi brute force. Hal ini dilakukan karena dari setiap kemungkinan piece kejadian berikutnya bisa bercabang banyak. Dengan demikian, untuk mencoba cabang-cabang kemungkinan konfigurasi piece puzzle pada papan, rekursi merupakan metode implementasi yang bagus untuk kasus ini.

Bruteforce yang diimplementasi adalah sebagai berikut.

Fungsi yang digunakan untuk implementasi menerima kondisi papan dan juga list dari piece yang tersisa.

Sebagai basis fungsi, ada dua kondisi yang mungkin terjadi:

1. List dari piece yang tersisa habis dan papan terpenuhi, maka return papan yang berisi solusinya dan jumlah hitungan kasus = 1
2. List dari piece yang tersisa habis dan papan tidak terpenuhi, maka return null (indikasi tidak ada solusi) dan jumlah hitungan kasus = 1

Sebagai rekursi, algoritmanya adalah sebagai berikut:

1. Ambil suatu piece dari list of pieces
2. Taruh piece di suatu tempat di papan, jika tidak bisa diletakkan, coba posisi berikutnya.
3. Jika piece dapat diletakkan ambil piece lagi dari list of pieces dan coba letakkan di papan lagi untuk semua posisi yang mungkin.
4. Jika posisi sudah dicoba dan tidak ada yang mungkin untuk meletakkan piece, rotasi piece sejauh 90 derajat searah jarum jam sebanyak 3 kali (untuk semua kombinasi bentuk yang berbeda dari satu piece) dan cek semua posisi lagi.
5. Jika masih belum menemukan posisi yang valid, cerminkan/balikkan piece dan cobakan semua posisi lagi dan rotasi 90 derajat searah jarum jam sebanyak 3 kali lagi.
6. Untuk setiap kasus piece tidak bisa diletakkan naikkan jumlah kasus sebanyak 1

7. Fungsi akan berhenti ketika semua kombinasi berakhir ke basis 2 yang berarti tidak ada solusi atau salah satu kombinasi mendapat basis 1 yang berarti solusi ditemukan dan fungsi langsung berhenti tanpa memproses kombinasi lain.

Algoritma ini menangani semua kasus sehingga pasti solusi akan ditemukan jika ada. Selain itu, algoritma ini juga akan menangani kasus jika papan terpenuhi dan terdapat piece terpisah. Hal ini bisa terjadi karena ketika papan penuh, karena belum memenuhi kondisi basis "list piece tersisa habis", piece berikutnya akan masih dicoba dimuatkan pada papan penuh yang akan berakhir dengan piece tidak dapat dimuatkan untuk setiap posisi. Ini akhirnya akan merambat ke piece lain yang sudah diletakkan di papan hingga memang ditemukan bahwa tidak ada kombinasi yang memenuhi.

2. Source Code

Struktur Program

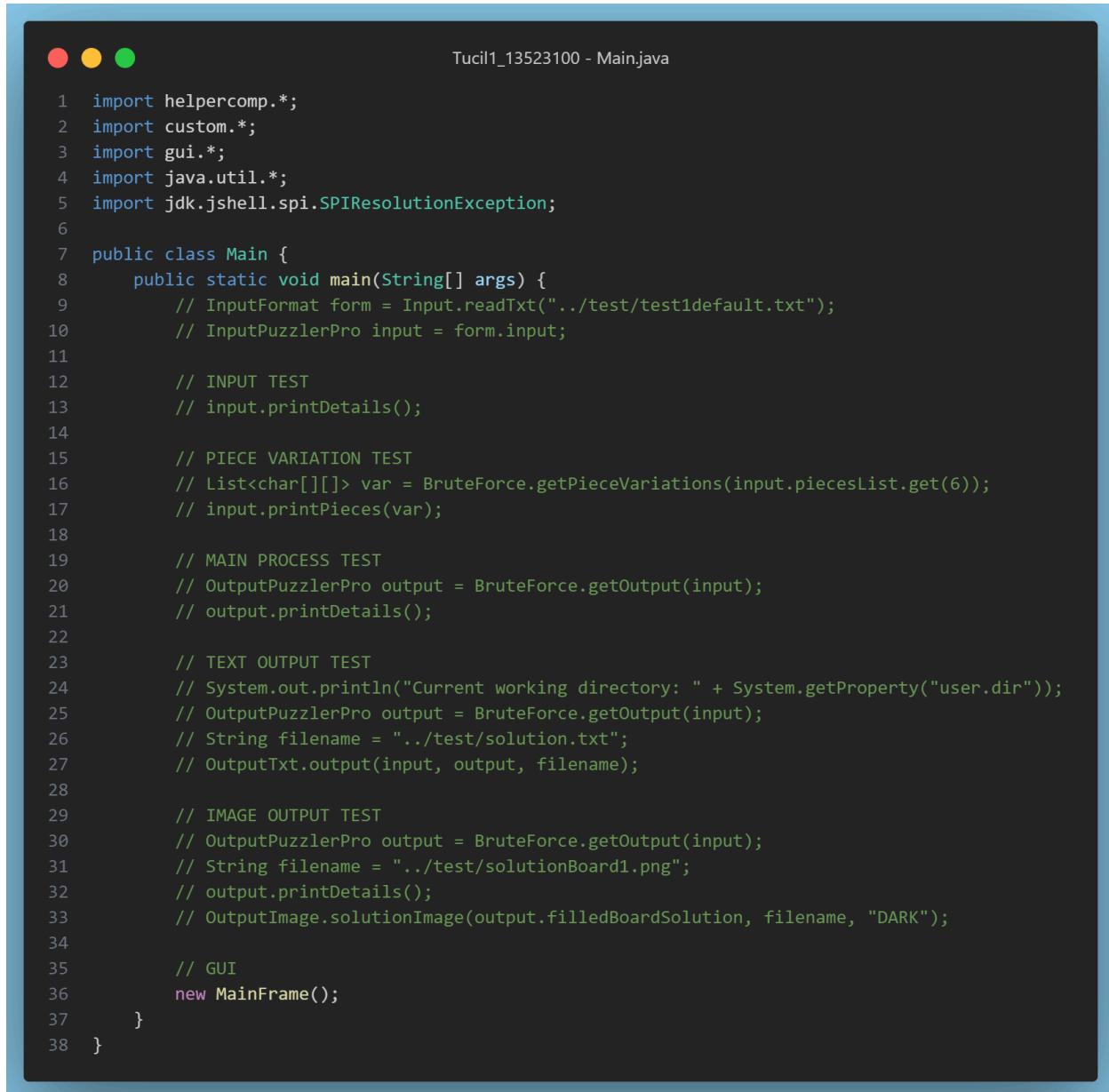
Struktur file src pada tugas kecil ini adalah sebagai berikut.

src	
custom	
InputFormat.java	3
InputPuzzlerPro.java	2
OutputPuzzlerPro.java	3
gui	
MainFrame.java	9+
Solution.java	9+
helpercomp	
BruteForce.java	3
Input.java	6
InputImageGenerator.java	7
OutputImage.java	2
OutputTxt.java	5
Main.java	5, M

Pada src, terdapat Main.java yang merupakan driver utama untuk menjalankan program termasuk GUI. Pada src juga terdapat tiga package buatan. Package pertama adalah **custom**, ini adalah package yang menyimpan class yang bersifat sebagai struct atau format input dan output, tetapi juga terdapat beberapa metode pada class-class dalam package tersebut. Package kedua adalah **helpercomp**, dalam package ini berisi program-program yang berfungsi untuk memroses input menjadi output atau dalam kata lain program-program pembantu yang

jika disatukan akan menjadi program Main. Package terakhir **gui**, sesuai namanya, adalah package yang mengendalikan Graphical User Interface (GUI) pada program ini, dengan MainFrame sebagai window awal untuk input dan Solution sebagai window yang muncul untuk menunjukkan hasil pemrosesan brute force.

Program **Main** yang berisi test-test per komponen sewaktu development hanya perlu memanggil GUI untuk memulai program pada versi final program ini



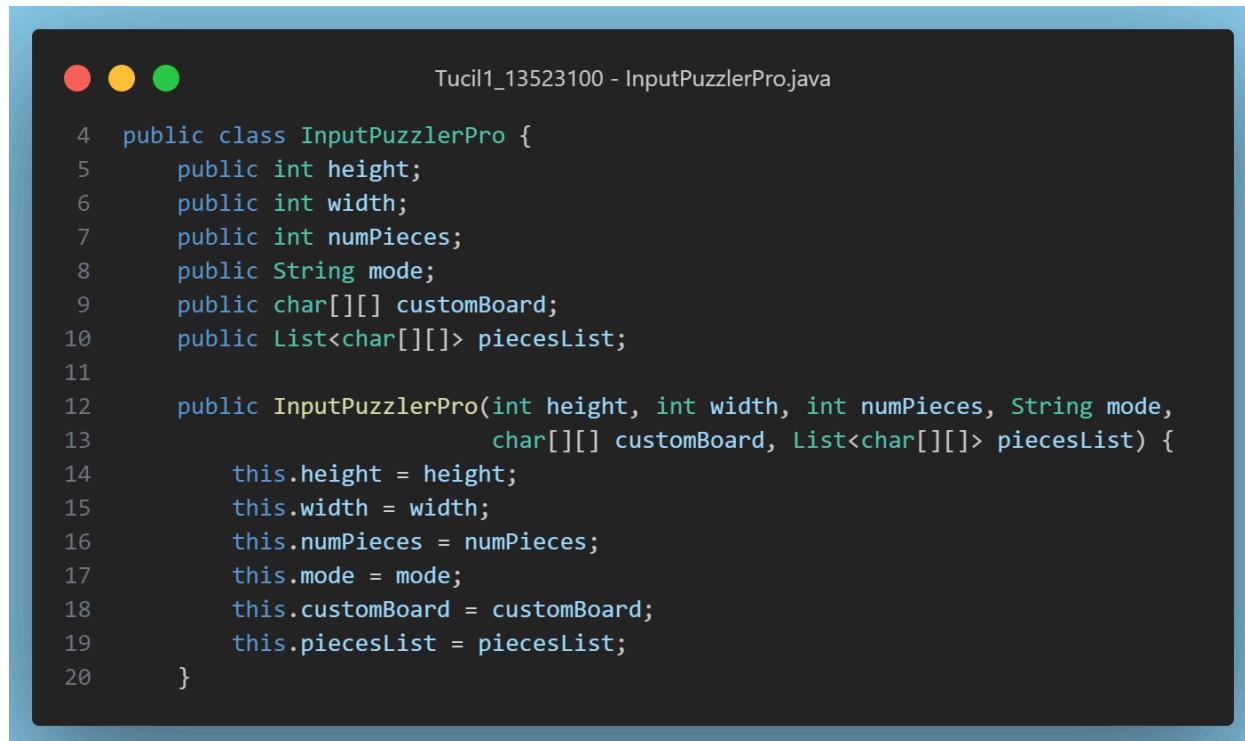
The screenshot shows a terminal window with a dark background and light-colored text. At the top left are three circular icons: red, yellow, and green. The title bar at the top right reads "Tucil1_13523100 - Main.java". The main area contains 38 numbered lines of Java code:

```
1 import helpercomp.*;
2 import custom.*;
3 import gui.*;
4 import java.util.*;
5 import jdk.jshell.spi.SPIResolutionException;
6
7 public class Main {
8     public static void main(String[] args) {
9         // InputFormat form = Input.readTxt("../test/test1default.txt");
10        // InputPuzzlerPro input = form.input;
11
12        // INPUT TEST
13        // input.printDetails();
14
15        // PIECE VARIATION TEST
16        // List<char[][]> var = BruteForce.getPieceVariations(input.piecesList.get(6));
17        // input.printPieces(var);
18
19        // MAIN PROCESS TEST
20        // OutputPuzzlerPro output = BruteForce.getOutput(input);
21        // output.printDetails();
22
23        // TEXT OUTPUT TEST
24        // System.out.println("Current working directory: " + System.getProperty("user.dir"));
25        // OutputPuzzlerPro output = BruteForce.getOutput(input);
26        // String filename = "../test/solution.txt";
27        // OutputTxt.output(input, output, filename);
28
29        // IMAGE OUTPUT TEST
30        // OutputPuzzlerPro output = BruteForce.getOutput(input);
31        // String filename = "../test/solutionBoard1.png";
32        // output.printDetails();
33        // OutputImage.solutionImage(output.filledBoardSolution, filename, "DARK");
34
35        // GUI
36        new MainFrame();
37    }
38 }
```

Package Custom

File InputPuzzlerPro

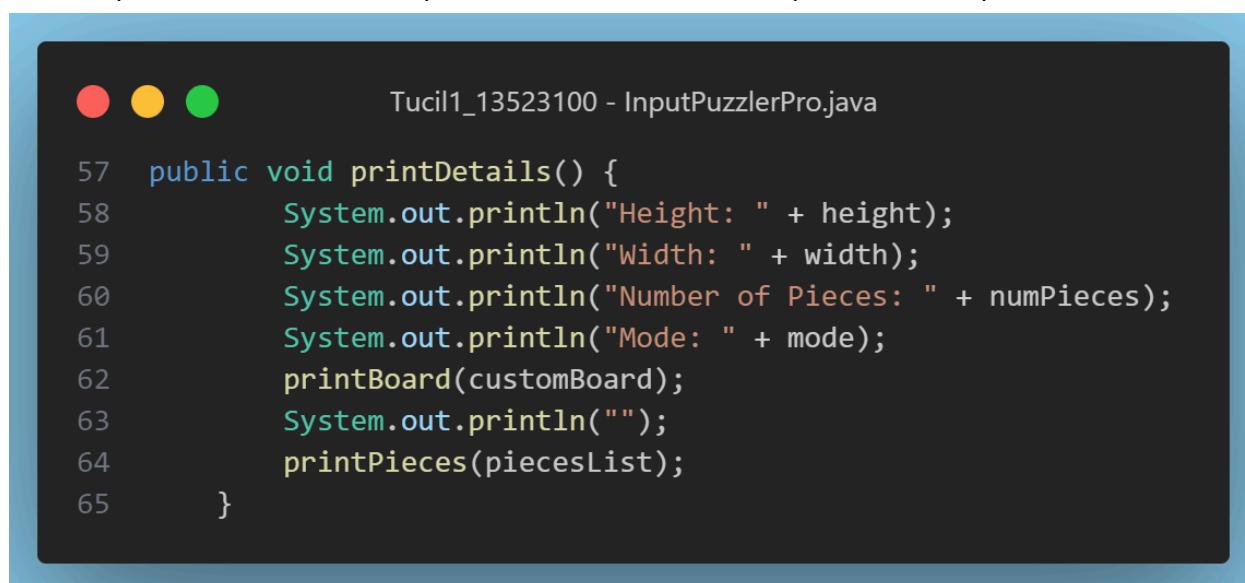
Atribut dan konstruktor InputPuzzlerPro



```
Tucil1_13523100 - InputPuzzlerPro.java

4  public class InputPuzzlerPro {
5      public int height;
6      public int width;
7      public int numPieces;
8      public String mode;
9      public char[][] customBoard;
10     public List<char[][]> piecesList;
11
12     public InputPuzzlerPro(int height, int width, int numPieces, String mode,
13                           char[][] customBoard, List<char[][]> piecesList) {
14         this.height = height;
15         this.width = width;
16         this.numPieces = numPieces;
17         this.mode = mode;
18         this.customBoard = customBoard;
19         this.piecesList = piecesList;
20     }
}
```

Metode printDetails untuk memperlihatkan semua atribut InputPuzzlerPro pada CLI



```
Tucil1_13523100 - InputPuzzlerPro.java

57    public void printDetails() {
58        System.out.println("Height: " + height);
59        System.out.println("Width: " + width);
60        System.out.println("Number of Pieces: " + numPieces);
61        System.out.println("Mode: " + mode);
62        printBoard(customBoard);
63        System.out.println("");
64        printPieces(piecesList);
65    }
}
```

File OutputPuzzlerPro

Atribut dan konstruktor OutputPuzzlerPro

```
Tucil1_13523100 - OutputPuzzlerPro.java

5  public class OutputPuzzlerPro {
6      public char[][] filledBoardSolution;
7      public long executionTime;
8      public int casesExplored;
9
10     public OutputPuzzlerPro(char[][] filledBoardSolution, long executionTime, int casesExplored) {
11         this.filledBoardSolution = filledBoardSolution;
12         this.executionTime = executionTime;
13         this.casesExplored = casesExplored;
14     }
}
```

Metode untuk menampilkan papan solusi

```
Tucil1_13523100 - OutputPuzzlerPro.java

32    public static void printSolutionBoard(char[][] boardSolution) {
33        if (boardSolution == null) {
34            System.out.println("Solution: No solution");
35        } else {
36            char[] chars = new char[26];
37            int charCount = 0;
38            System.out.println("Solution:");
39            for (int i = 0; i < boardSolution.length; i++) {
40                for (int j = 0; j < boardSolution[i].length; j++) {
41                    char currentChar = boardSolution[i][j];
42
43                    if (!Character.isUpperCase(currentChar)) {
44                        System.out.print(" ");
45                        continue;
46                    }
47
48                    if (getCharIndex(chars, currentChar) == -1) {
49                        // Add the character to the chars array
50                        chars[charCount] = currentChar;
51                        charCount++;
52                    }
53
54                    int charIndex = getCharIndex(chars, currentChar);
55
56                    if (charIndex != -1 && charIndex < ANSI_COLORS.length) {
57                        System.out.print(ANSI_COLORS[charIndex] + currentChar + " ");
58                    } else {
59                        // If no color exists for the charIndex, use the default color
60                        System.out.print("\u033[0m" + currentChar + " ");
61                    }
62                }
63                System.out.println();
64            }
65            System.out.print("\u033[0m");
66        }
67    }
```

Metode pembantu metode lain dan list ANSI_COLORS untuk print warna ke CLI

```
● ● ● Tucil1_13523100 - OutputPuzzlerPro.java

16  public static int getCharIndex(char[] chars, char c) {
17      for (int i = 0; i < chars.length; i++) {
18          if (chars[i] == c) {
19              return i;
20          }
21      }
22      return -1;
23  }
24
25  public static final String[] ANSI_COLORS = {"\u001b[38;2;0;255m", "\u001b[38;2;0;255;0m", "\u001b[38;2;255;128;0m", "\u001b[38;2;166;66;200m", "\u001b[38;2;101;55;0m",
26      "\u001b[38;2;128;128;128m", "\u001b[38;2;255;0;0m", "\u001b[38;2;128;0;0m", "\u001b[38;2;143;0;255m", "\u001b[38;2;205;127;50m",
27      "\u001b[38;2;255;255;204m", "\u001b[38;2;254;220;86m", "\u001b[38;2;0;0;128m", "\u001b[38;2;248;131;121m", "\u001b[38;2;224;176;255m",
28      "\u001b[38;2;255;204;153m", "\u001b[38;2;255;215;0m", "\u001b[38;2;0;255;255m", "\u001b[38;2;0;127;255m", "\u001b[38;2;0;128;128m",
29      "\u001b[38;2;112;130;56m", "\u001b[38;2;245;245;220m", "\u001b[38;2;54;69;79m", "\u001b[38;2;0;0;255m", "\u001b[38;2;36;122;253m",
30      "\u001b[38;2;255;209;226m"};
```

Metode untuk print semua atribut OutputPuzzlerPro

```
● ● ● Tucil1_13523100 - OutputPuzzlerPro.java

69  public void printDetails() {
70      printSolutionBoard(filledBoardSolution);
71      System.out.println("");
72      System.out.printf("Execution Time: %d ms%n", executionTime);
73      System.out.println("Number of cases checked: " + casesExplored);
74  }
```

Metode untuk memberi luaran sebagai .txt

```
Tucil1_13523100 - OutputPuzzlerPro.java

76  public void exportDetailsToTxt(String filename) {
77      try (BufferedWriter writer = new BufferedWriter(new FileWriter(filename))) {
78          // Write solution board to file
79          if (filledBoardSolution == null) {
80              System.out.println("Solution: No solution");
81              return;
82          } else {
83              writer.write("Solution:\n");
84              for (int i = 0; i < filledBoardSolution.length; i++) {
85                  for (int j = 0; j < filledBoardSolution[i].length; j++) {
86                      char currentChar = filledBoardSolution[i][j];
87
88                      if (!Character.isUpperCase(currentChar)) {
89                          writer.write(" ");
90                      } else {
91                          writer.write(currentChar + " ");
92                      }
93                  }
94                  writer.newLine();
95              }
96          }
97
98          writer.newLine();
99
100         // Write execution time and number of cases at the end
101         writer.write("Execution Time: " + executionTime + " ms\n");
102         writer.write("Number of cases checked: " + casesExplored + "\n");
103
104         System.out.println("Solution saved to " + filename);
105     } catch (IOException e) {
106         e.printStackTrace();
107         System.out.println("An error occurred while saving the solution.");
108     }
}
```

File InputFormat

Keseluruhan class InputFormat, class ini merupakan extension dari InputPuzzlerPro karena GUI yang memerlukan errorMessage yang terjadi pada proses input.

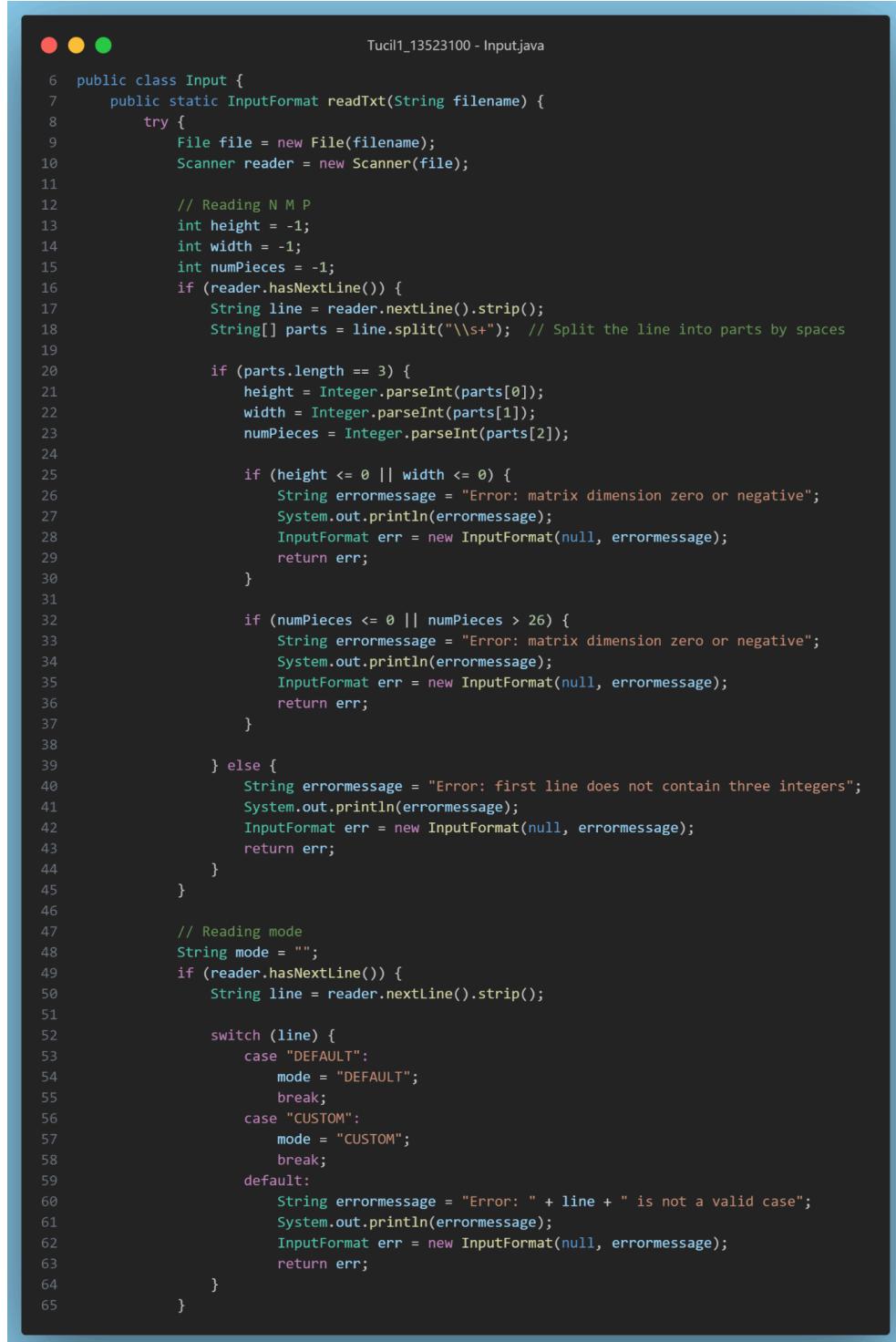
```
● ● ● Tucil1_13523100 - InputFormat.java

1 package custom;
2 import custom.InputPuzzlerPro;
3
4 public class InputFormat {
5     public InputPuzzlerPro input;
6     public String errorMessage;
7
8     public InputFormat(InputPuzzlerPro input, String errorMessage) {
9         this.input = input;
10        this.errorMessage = errorMessage;
11    }
12
13    public InputPuzzlerPro getInput() {
14        return input;
15    }
16
17    public String getErrorMessage() {
18        return errorMessage;
19    }
20}
21
```

Package HelperComp

File Input

Bagian pertama menginput width, height, jumlah piece, dan mode



The screenshot shows a Java code editor window with the title "Tucil1_13523100 - Input.java". The code is a class named "Input" with a static method "readTxt". The method reads three integers from a file: height, width, and numPieces. It then checks if these values are valid (non-negative). If they are, it reads a mode string from the next line. If the mode is "DEFAULT", it sets mode to "DEFAULT". If it's "CUSTOM", it sets mode to "CUSTOM". Otherwise, it prints an error message and returns an InputFormat object with an error message. The code uses try-with-resources to handle the file and scanner.

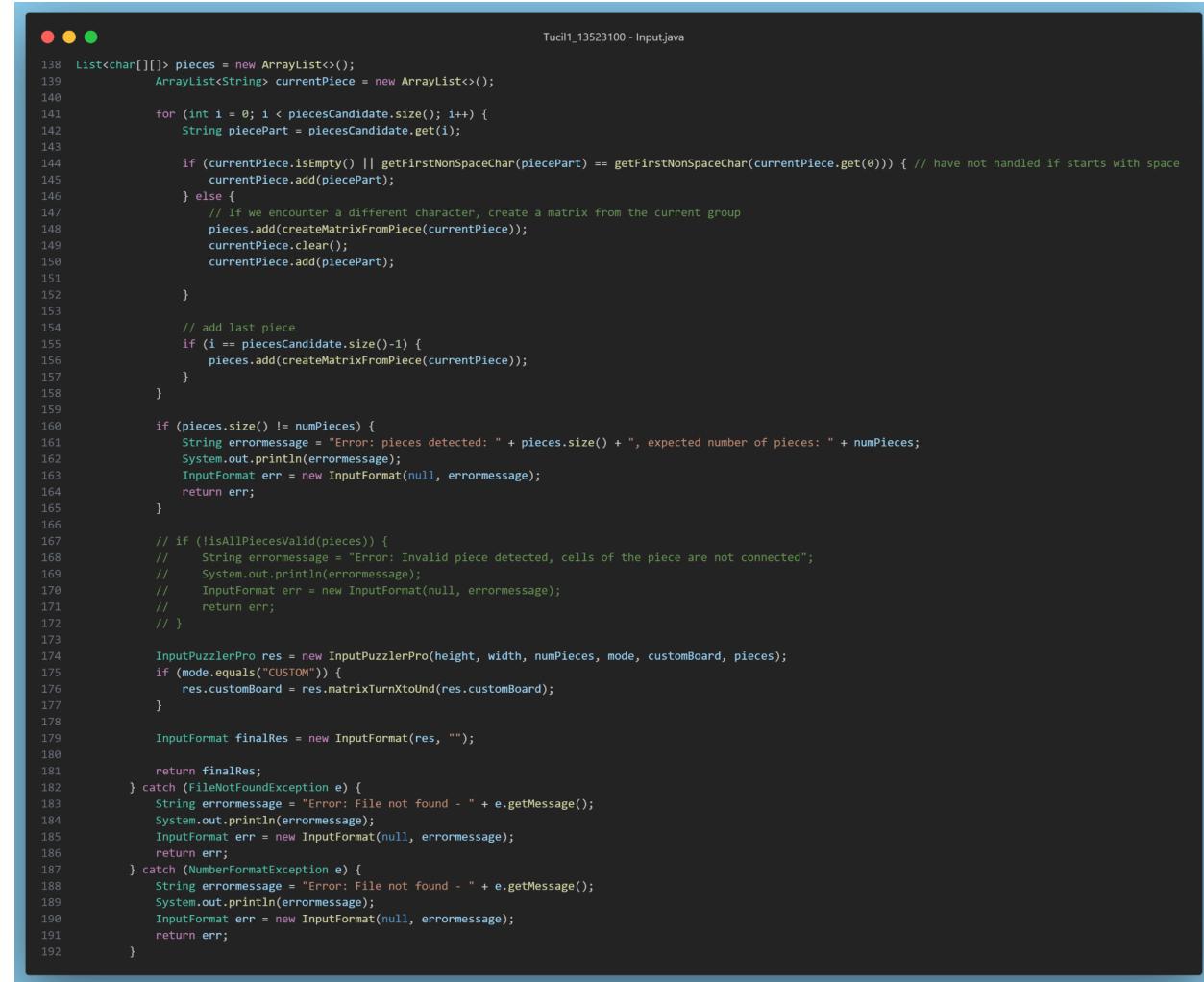
```
6  public class Input {
7      public static InputFormat readTxt(String filename) {
8          try {
9              File file = new File(filename);
10             Scanner reader = new Scanner(file);
11
12             // Reading N M P
13             int height = -1;
14             int width = -1;
15             int numPieces = -1;
16             if (reader.hasNextLine()) {
17                 String line = reader.nextLine().strip();
18                 String[] parts = line.split("\\s+");
19                 if (parts.length == 3) {
20                     height = Integer.parseInt(parts[0]);
21                     width = Integer.parseInt(parts[1]);
22                     numPieces = Integer.parseInt(parts[2]);
23
24                     if (height <= 0 || width <= 0) {
25                         String errormessage = "Error: matrix dimension zero or negative";
26                         System.out.println(errormessage);
27                         InputFormat err = new InputFormat(null, errormessage);
28                         return err;
29                     }
30
31                     if (numPieces <= 0 || numPieces > 26) {
32                         String errormessage = "Error: matrix dimension zero or negative";
33                         System.out.println(errormessage);
34                         InputFormat err = new InputFormat(null, errormessage);
35                         return err;
36                     }
37
38                 } else {
39                     String errormessage = "Error: first line does not contain three integers";
40                     System.out.println(errormessage);
41                     InputFormat err = new InputFormat(null, errormessage);
42                     return err;
43                 }
44             }
45
46             // Reading mode
47             String mode = "";
48             if (reader.hasNextLine()) {
49                 String line = reader.nextLine().strip();
50
51                 switch (line) {
52                     case "DEFAULT":
53                         mode = "DEFAULT";
54                         break;
55                     case "CUSTOM":
56                         mode = "CUSTOM";
57                         break;
58                     default:
59                         String errormessage = "Error: " + line + " is not a valid case";
60                         System.out.println(errormessage);
61                         InputFormat err = new InputFormat(null, errormessage);
62                         return err;
63                 }
64             }
65         }
66     }
```

Bagian kedua membaca papan custom dan piece-piece

```
● ● ● Tucilt_13523100 - Input.java

67 // Reading matrix if mode == "CUSTOM"
68     char[][] customBoard = null;
69     String lineAfterMatrix = "";
70
71     if (mode.equals("CUSTOM")) {
72         boolean inMatrix = false;
73         ArrayList<String> matrixLines = new ArrayList<>();
74
75         while (reader.hasNextLine()) {
76             String line = reader.nextLine();
77
78             if (line.contains("X") && line.strip().length() == width) {
79                 inMatrix = true;
80             }
81
82             if (inMatrix) {
83                 matrixLines.add(line.strip());
84                 inMatrix = false;
85             } else {
86                 if (matrixLines.isEmpty()) {
87                     String errormessage = "Error: no custom matrix provided OR matrix provided does not match given width";
88                     System.out.println(errormessage);
89                     InputFormat err = new InputFormat(null, errormessage);
90                     return err;
91                 }
92                 lineAfterMatrix = line;
93                 break;
94             }
95         }
96
97         customBoard = new char[matrixLines.size()][];
98         for (int i = 0; i < matrixLines.size(); i++) {
99             String row = matrixLines.get(i);
100            customBoard[i] = row.toCharArray();
101        }
102
103        if (customBoard.length != height) {
104            String errormessage = "Error: Board height does not match given height";
105            System.out.println(errormessage);
106            InputFormat err = new InputFormat(null, errormessage);
107            return err;
108        }
109    }
110
111    // Reading pieces
112    String line;
113    ArrayList<String> piecesCandidate = new ArrayList<>();
114
115    if (mode.equals("CUSTOM")) {
116        line = lineAfterMatrix;
117    } else {
118        line = reader.nextLine();
119    }
120
121    while (true) {
122        if (isPiecePartValid(line)){
123            piecesCandidate.add(line);
124        } else {
125            String errormessage = "Error: invalid piece detected -> has different characters in one row OR not a capital alphabet, error piece: " + line;
126            System.out.println(errormessage);
127            InputFormat err = new InputFormat(null, errormessage);
128            return err;
129        }
130
131        if (reader.hasNextLine()) {
132            line = reader.nextLine();
133        } else {
134            break;
135        }
136    }
}
```

Bagian ketiga, block yang dikomen adalah filter untuk mencegah piece yang tidak terhubung



```
138 List<char[][]> pieces = new ArrayList<>();
139     ArrayList<String> currentPiece = new ArrayList<>();
140
141     for (int i = 0; i < piecesCandidate.size(); i++) {
142         String piecePart = piecesCandidate.get(i);
143
144         if (currentPiece.isEmpty() || getFirstNonSpaceChar(piecePart) == getFirstNonSpaceChar(currentPiece.get(0))) { // have not handled if starts with space
145             currentPiece.add(piecePart);
146         } else {
147             // If we encounter a different character, create a matrix from the current group
148             pieces.add(createMatrixFromPiece(currentPiece));
149             currentPiece.clear();
150             currentPiece.add(piecePart);
151         }
152     }
153
154     // add last piece
155     if (i == piecesCandidate.size()-1) {
156         pieces.add(createMatrixFromPiece(currentPiece));
157     }
158 }
159
160 if (pieces.size() != numPieces) {
161     String errorMessage = "Error: pieces detected: " + pieces.size() + ", expected number of pieces: " + numPieces;
162     System.out.println(errorMessage);
163     InputFormat err = new InputFormat(null, errorMessage);
164     return err;
165 }
166
167 // if (!isValidAllPieces(pieces)) {
168 //     String errorMessage = "Error: Invalid piece detected, cells of the piece are not connected";
169 //     System.out.println(errorMessage);
170 //     InputFormat err = new InputFormat(null, errorMessage);
171 //     return err;
172 // }
173
174 InputPuzzlerPro res = new InputPuzzlerPro(height, width, numPieces, mode, customBoard, pieces);
175 if (mode.equals("CUSTOM")) {
176     res.customBoard = res.matrixTurnXtoUnd(res.customBoard);
177 }
178
179 InputFormat finalRes = new InputFormat(res, "");
180
181 return finalRes;
182 } catch (FileNotFoundException e) {
183     String errorMessage = "Error: File not found - " + e.getMessage();
184     System.out.println(errorMessage);
185     InputFormat err = new InputFormat(null, errorMessage);
186     return err;
187 } catch (NumberFormatException e) {
188     String errorMessage = "Error: File not found - " + e.getMessage();
189     System.out.println(errorMessage);
190     InputFormat err = new InputFormat(null, errorMessage);
191     return err;
192 }
```

Method pembantu input

```
Tucil1_13523100 - Input.java

195 public static char[][] createMatrixFromPiece(List<String> piece) {
196     int maxLength = piece.stream().mapToInt(String::length).max().orElse(0);
197     char[][] matrix = new char[piece.size()][maxLength];
198
199     for (int i = 0; i < piece.size(); i++) {
200         String piecePart = piece.get(i);
201         for (int j = 0; j < piecePart.length(); j++) {
202             matrix[i][j] = (piecePart.charAt(j) == ' ') ? '_' : piecePart.charAt(j);
203         }
204         for (int j = piecePart.length(); j < maxLength; j++) {
205             matrix[i][j] = '_';
206         }
207     }
208     return matrix;
209 }
210
211 public static boolean isPiecePartValid(String piecePart) {
212     if (piecePart.isEmpty()) {
213         return false;
214     }
215
216     char alph = getFirstNonSpaceChar(piecePart);
217
218     if (!Character.isUpperCase(alph)) {
219         return false; // If the first character is not a capital letter, return false
220     }
221
222     for (int i = 1; i < piecePart.length(); i++) {
223         if (piecePart.charAt(i) != alph && piecePart.charAt(i) != ' ') {
224             return false;
225         }
226     }
227
228     return true;
229 }
230
231 public static char getFirstNonSpaceChar(String line) {
232     for (int i = 0; i < line.length(); i++) {
233         char currentChar = line.charAt(i);
234
235         if (currentChar != ' ') {
236             return currentChar;
237         }
238     }
239
240     return '\0';
241 }
```

File BruteForce

Fungsi utama untuk memroses puzzle

```
Tucil1_13523100 - BruteForce.java

5  public class BruteForce {
6      public static OutputPuzzlerPro getOutput(InputFormat form) {
7          char[][] beginningMat;
8          if (form.input.mode.equals("DEFAULT")) {
9              beginningMat = createEmptyMatrix(form.input.height, form.input.width);
10         } else if (form.input.mode.equals("CUSTOM")) {
11             beginningMat = form.input.customBoard;
12         } else {
13             System.out.printf("Error: %s is not a valid case%n", form.input.mode);
14             return null;
15         }
16
17         long startTime = System.currentTimeMillis();
18         OutputPuzzlerPro res = processPuzzle(beginningMat, form.input.piecesList);
19         long endTime = System.currentTimeMillis();
20         long duration = endTime - startTime;
21         res.executionTime = duration;
22
23         return res;
24     }
}
```

Fungsi rekursi brute force

```
Tucil1_13523100 - BruteForce.java

26  public static OutputPuzzlerPro processPuzzle(char[][] currentMatrix, List<char[][]> pieces) {
27      if (pieces.isEmpty()) {
28          if (matrixIsFull(currentMatrix)) {
29              OutputPuzzlerPro res = new OutputPuzzlerPro(currentMatrix, 0, 1);
30              // res.printSolutionBoard(currentMatrix);
31              return res;
32          } else {
33              OutputPuzzlerPro res = new OutputPuzzlerPro(null, 0, 1);
34              // res.printSolutionBoard(currentMatrix);
35              return res;
36          }
37      }
38
39      char[][] currentPiece = pieces.get(0);
40      // printMatrix(currentPiece);
41      List<char[][]> currentVariations = getPieceVariations(currentPiece);
42      int countCases = 0;
43
44      // Try all piece variation with rotate and flip
45      for (int i = 0; i < currentVariations.size(); i++) {
46          int currentWidth = currentVariations.get(i)[0].length;
47          int currentHeight = currentVariations.get(i).length;
48
49          int height = currentMatrix.length - currentHeight + 1;
50          int width = currentMatrix[0].length - currentWidth + 1;
51
52          // if doesn't fit because larger than matrix, don't process because it fails here
53          if (height <= 0 || width <= 0) {
54              countCases += 1;
55              continue;
56          }
57
58          // Checking if the current piece position variation fits for every possible location in matrix
59          for (int j = 0; j < height; j++) {
60              for (int k = 0; k < width; k++) {
61                  boolean canFit = true;
62
63                  // determining if currentVariation fits
64                  for (int x = 0; x < currentHeight; x++) {
65                      for (int y = 0; y < currentWidth; y++) {
66                          if (currentVariations.get(i)[x][y] != '_' && currentMatrix[j + x][k + y] != '_') {
67                              canFit = false;
68                              break;
69                          }
70                      }
71                      if (!canFit) break;
72                  }
73
74                  // If it fits, place the variation into currentMatrix and continue to the next piece
75                  if (canFit) {
76                      char[][] newMatrix = placeVariation(currentMatrix, currentVariations.get(i), j, k);
77                      OutputPuzzlerPro res = processPuzzle(newMatrix, listWithoutFirstElement(pieces));
78                      if (res.filledBoardSolution != null) {
79                          res.casesExplored += countCases;
80                          return res;
81                      } else {
82                          countCases += res.casesExplored;
83                      }
84                  } else { // count case and continue to the next position
85                      countCases += 1;
86                  }
87              }
88          }
89          // if all positions fail, try another variation
90      }
91      // if all variation fail, return null matrix and how many cases were explored
92      OutputPuzzlerPro res = new OutputPuzzlerPro(null, 0, countCases);
93      return res;
94  }
```

Fungsi pertama akan membuat papan kosong dari input dan fungsi kedua adalah fungsi yang cek apakah papan sudah terpenuhi atau tidak



The screenshot shows a Java code editor with a dark theme. At the top, there are three colored circular icons (red, yellow, green) and the file name "Tucil1_13523100 - BruteForce.java". The code consists of two static methods:

```
96  public static char[][] createEmptyMatrix(int height, int width) {  
97      char[][] matrix = new char[height][width];  
98  
99      for (int i = 0; i < height; i++) {  
100          for (int j = 0; j < width; j++) {  
101              matrix[i][j] = '_';  
102          }  
103      }  
104  
105     return matrix;  
106 }  
107  
108  
109    public static boolean matrixIsFull(char[][] m) {  
110        for (int i = 0; i < m.length; i++) {  
111            for (int j = 0; j < m[0].length; j++) {  
112                if (m[i][j] == '_') {  
113                    return false;  
114                }  
115            }  
116        }  
117  
118        return true;  
119    }
```

Fungsi untuk mendapatkan variasi rotasi dan cermin dari suatu piece

```
● ● ● Tucil1_13523100 - BruteForce.java

121 public static List<char[][]> getPieceVariations(char[][] piece) {
122     List<char[][]> variations = new ArrayList<>();
123     variations.add(piece);
124
125     // flip
126     char[][] mirroredPieceHorizontal = new char[piece.length][piece[0].length];
127
128     for (int i = 0; i < piece.length; i++) {
129         for (int j = 0; j < piece[0].length; j++) {
130             mirroredPieceHorizontal[i][piece[0].length - j - 1] = piece[i][j];
131         }
132     }
133     variations.add(mirroredPieceHorizontal);
134
135     // rotate
136     char[][] oldVar = piece;
137     for (int i = 0; i < 3; i++) {
138         int rows = oldVar.length;
139         int cols = oldVar[0].length;
140         char[][] var = new char[cols][rows];
141
142         for (int j = 0; j < rows; j++) {
143             for (int k = 0; k < cols; k++) {
144                 var[k][rows - 1 - j] = oldVar[j][k];
145             }
146         }
147
148         variations.add(var);
149         oldVar = var;
150     }
151
152     oldVar = mirroredPieceHorizontal;
153     for (int i = 0; i < 3; i++) {
154         int rows = oldVar.length;
155         int cols = oldVar[0].length;
156         char[][] var = new char[cols][rows];
157
158         for (int j = 0; j < rows; j++) {
159             for (int k = 0; k < cols; k++) {
160                 var[k][rows - 1 - j] = oldVar[j][k];
161             }
162         }
163
164         variations.add(var);
165         oldVar = var;
166     }
167
168     return variations;
169 }
```

Fungsi untuk meletakkan piece pada matrix papan

```
Tucil1_13523100 - BruteForce.java

171 public static char[][] placeVariation(char[][] currentMatrix, char[][] currentVariation, int startX, int startY) {
172     char[][] newMatrix = new char[currentMatrix.length][currentMatrix[0].length];
173
174     for (int i = 0; i < currentMatrix.length; i++) {
175         System.arraycopy(currentMatrix[i], 0, newMatrix[i], 0, currentMatrix[i].length);
176     }
177
178     for (int i = 0; i < currentVariation.length; i++) {
179         for (int j = 0; j < currentVariation[0].length; j++) {
180             if (currentVariation[i][j] != '_') {
181                 newMatrix[startX + i][startY + j] = currentVariation[i][j];
182             }
183         }
184     }
185
186     return newMatrix;
187 }
```

Method untuk mengambil piece dari list

```
Tucil1_13523100 - BruteForce.java

220 public static List<char[][]> listWithoutFirstElement(List<char[][]> originalList) {
221     if (originalList.isEmpty()) {
222         return new ArrayList<>();
223     }
224
225     return new ArrayList<>(originalList.subList(1, originalList.size()));
226 }
```

File OutputTxt

Bagian pertama

```
Tucil1_13523100 - OutputTxt.java

5  public class OutputTxt {
6      public static void output (InputPuzzlerPro input, OutputPuzzlerPro output, String filename) {
7          try (BufferedWriter writer = new BufferedWriter(new FileWriter(filename))) {
8              // Write input board to file
9              char[][] inputBoard = input.createEmptyMatrix();
10             if (input.mode.equals("CUSTOM")) {
11                 if (input.customBoard == null){
12                     inputBoard = null;
13                 } else {
14                     inputBoard = input.matrixTurnXtoUnd(input.customBoard);
15                 }
16             }
17
18             if (inputBoard == null) {
19                 writer.write("Input Board: null");
20                 writer.newLine();
21             } else {
22                 writer.write("Input Board:");
23                 writer.newLine();
24                 for (int i = 0; i < inputBoard.length; i++) {
25                     for (int j = 0; j < inputBoard[i].length; j++) {
26                         if (inputBoard[i][j] != '_') {
27                             writer.write(" ");
28                         } else {
29                             writer.write(inputBoard[i][j] + " ");
30                         }
31                     }
32                     writer.newLine();
33                 }
34             }
35             writer.newLine();
36
37             // Write input pieces
38             if (input.piecesList.isEmpty()) {
39                 writer.write("Pieces List: empty");
40                 writer.newLine();
41             } else {
42                 writer.write("Pieces List:");
43                 writer.newLine();
44                 for (int index = 0; index < input.piecesList.size(); index++) {
45                     writer.write("Piece " + (index + 1) + ":");
46                     writer.newLine();
47                     if (input.piecesList.get(index) == null) {
48                         writer.write("null");
49                     } else {
50                         for (int i = 0; i < input.piecesList.get(index).length; i++) {
51                             for (int j = 0; j < input.piecesList.get(index)[i].length; j++) {
52                                 writer.write(input.piecesList.get(index)[i][j] + " ");
53                             }
54                             writer.newLine();
55                         }
56                     }
57                 }
58             }
59             writer.newLine();
}
```

Bagian kedua

```
● ○ ● Tucil1_13523100 - OutputTxt.java

37 // Write input pieces
38     if (input.piecesList.isEmpty()) {
39         writer.write("Pieces List: empty");
40         writer.newLine();
41     } else {
42         writer.write("Pieces List:");
43         writer.newLine();
44         for (int index = 0; index < input.piecesList.size(); index++) {
45             writer.write("Piece " + (index + 1) + ":");
46             writer.newLine();
47             if (input.piecesList.get(index) == null) {
48                 writer.write("null");
49             } else {
50                 for (int i = 0; i < input.piecesList.get(index).length; i++) {
51                     for (int j = 0; j < input.piecesList.get(index)[i].length; j++) {
52                         writer.write(input.piecesList.get(index)[i][j] + " ");
53                     }
54                     writer.newLine();
55                 }
56             }
57         }
58     }
59     writer.newLine();

60
61     // Write solution board to file
62     if (output.filledBoardSolution == null) {
63         writer.write("Solution: No solution");
64         writer.newLine();
65     } else {
66         writer.write("Solution:\n");
67         for (int i = 0; i < output.filledBoardSolution.length; i++) {
68             for (int j = 0; j < output.filledBoardSolution[i].length; j++) {
69                 char currentChar = output.filledBoardSolution[i][j];
70
71                 if (!Character.isUpperCase(currentChar)) {
72                     writer.write(" ");
73                 } else {
74                     writer.write(currentChar + " ");
75                 }
76             }
77             writer.newLine();
78         }
79     }
80     writer.newLine();

81
82     // Write execution time and number of cases at the end
83     writer.write("Execution Time: " + output.executionTime + " ms\n");
84     writer.write("Number of cases checked: " + output.casesExplored + "\n");
85
86     System.out.println("Solution saved to " + filename);
87 } catch (IOException e) {
88     e.printStackTrace();
89     System.out.println("An error occurred while saving the solution.");
90 }
91 }
```

File InputImageGenerator

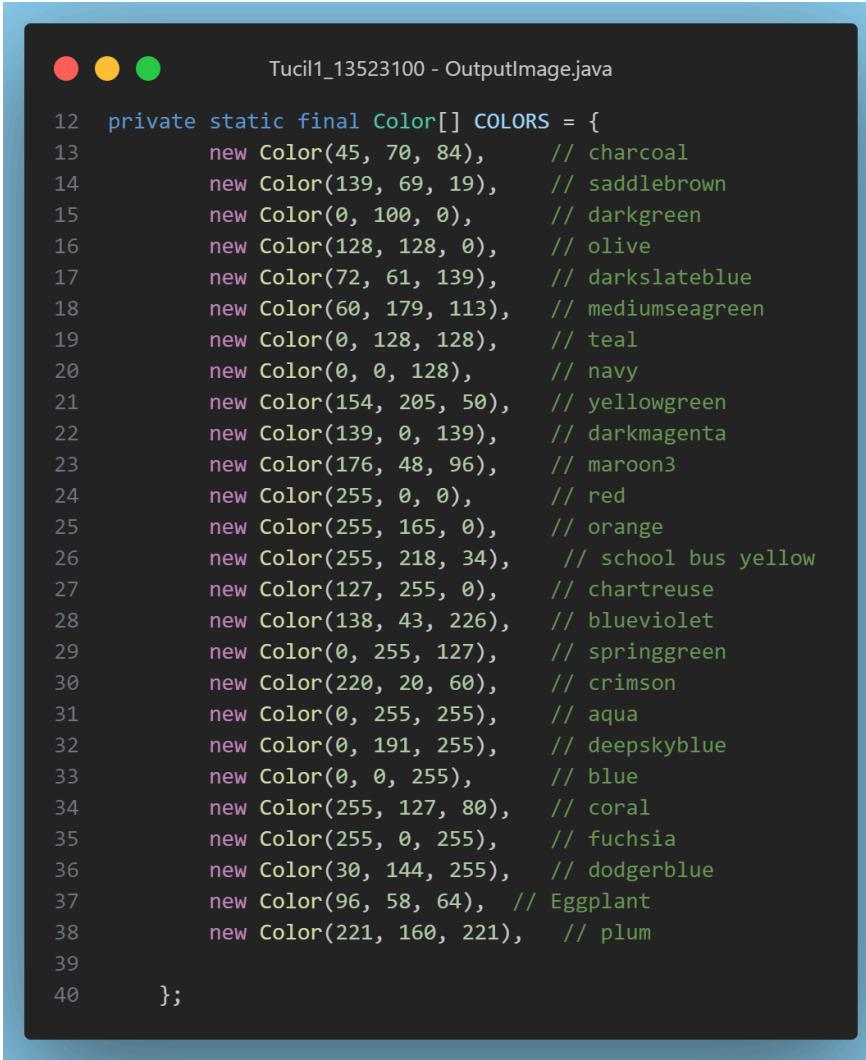
Fungsi untuk save input sebagai image untuk GUI

```
Tucil1_13523100 - InputImageGenerator.java

41 public static void saveInput (InputPuzzlerPro input) {
42     if (input != null) {
43         // setting up tempFolder
44         String currentDirectory = System.getProperty("user.dir");
45
46         File currentDirFile = new File(currentDirectory);
47         String fullPath = currentDirectory + File.separator + "temp";
48
49         File tempFolder = new File(fullPath);
50
51         // Create tempFolder if it does not exist
52         if (!tempFolder.exists()) {
53             if (tempFolder.mkdirs()) {
54                 System.out.println("Temporary folder created: " + tempFolder.getAbsolutePath());
55             } else {
56                 System.out.println("Failed to create temporary folder.");
57             }
58         } else {
59             System.out.println("Temporary folder already exists.");
60         }
61
62         // Generating empty board
63         if (input.mode.equals("DEFAULT")) {
64             String boardName = fullPath + File.separator + "emptyBoard.png";
65             char[][] formattedBoard = createEmptyMatrix(input.height, input.width);
66
67             boardGenerator(formattedBoard, boardName, "LIGHT");
68         } else if (input.mode.equals("CUSTOM")) {
69             String boardName = fullPath + File.separator + "emptyBoard.png";
70             char[][] formattedBoard = input.matrixTurnXtoUnd(input.customBoard);
71
72             boardGenerator(formattedBoard, boardName, "LIGHT");
73         }
74
75         // Generating pieces
76         for (int i = 0; i < input.piecesList.size(); i++) {
77             String pieceName = fullPath + File.separator + "piece" + (i+1) + ".png";
78
79             pieceGenerator(input.piecesList.get(i), pieceName, "LIGHT");
80         }
81     }
82 }
83 }
```

File OutputImage

List warna untuk gambar



The screenshot shows a code editor window with a dark theme. At the top, there are three circular icons: red, yellow, and green. The title bar reads "Tucil1_13523100 - OutputImage.java". The main area contains the following Java code:

```
12 private static final Color[] COLORS = {  
13     new Color(45, 70, 84),      // charcoal  
14     new Color(139, 69, 19),    // saddlebrown  
15     new Color(0, 100, 0),     // darkgreen  
16     new Color(128, 128, 0),   // olive  
17     new Color(72, 61, 139),   // darkslateblue  
18     new Color(60, 179, 113),  // mediumseagreen  
19     new Color(0, 128, 128),   // teal  
20     new Color(0, 0, 128),    // navy  
21     new Color(154, 205, 50),  // yellowgreen  
22     new Color(139, 0, 139),   // darkmagenta  
23     new Color(176, 48, 96),   // maroon3  
24     new Color(255, 0, 0),    // red  
25     new Color(255, 165, 0),   // orange  
26     new Color(255, 218, 34),  // school bus yellow  
27     new Color(127, 255, 0),   // chartreuse  
28     new Color(138, 43, 226),  // blueviolet  
29     new Color(0, 255, 127),   // springgreen  
30     new Color(220, 20, 60),   // crimson  
31     new Color(0, 255, 255),   // aqua  
32     new Color(0, 191, 255),   // deepskyblue  
33     new Color(0, 0, 255),    // blue  
34     new Color(255, 127, 80),  // coral  
35     new Color(255, 0, 255),   // fuchsia  
36     new Color(30, 144, 255),  // dodgerblue  
37     new Color(96, 58, 64),   // Eggplant  
38     new Color(221, 160, 221),  // plum  
39  
40 };
```

Fungsi menyimpan output sebagai image



```
Tucil1_13523100 - OutputImage.java

42 public static void solutionImage(char[][] solutionBoard, String filename, String mode) {
43     int cellSize = 80; // Size of each cell
44     int padding = 5; // Padding between cells
45     int edgePadding = 40; // Padding at edge of image
46     int font_size = 40; // Font size for letters inside of the cells
47
48     int rows = solutionBoard.length;
49     int cols = solutionBoard[0].length;
50
51     // Calculate the width and height of the image based on the matrix size
52     int imageWidth = (cols * cellSize) + ((cols - 1) * padding) + 2 * edgePadding;
53     int imageHeight = (rows * cellSize) + ((rows - 1) * padding) + 2 * edgePadding;
54
55     BufferedImage image = new BufferedImage(imageWidth, imageHeight, BufferedImage.TYPE_INT_ARGB);
56     Graphics2D g2d = image.createGraphics();
57     g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);
58     g2d.setRenderingHint(RenderingHints.KEY_INTERPOLATION, RenderingHints.VALUE_INTERPOLATION_BILINEAR);
59
60     if (mode.equals("DARK")) {
61         g2d.setColor(Color.BLACK);
62     } else {
63         g2d.setColor(Color.WHITE);
64     }
65
66     g2d.fillRect(0, 0, imageWidth, imageHeight);
67
68     // Set font and stroke for circle cell
69     g2d.setFont(new Font("Arial", Font.PLAIN, font_size));
70     g2d.setStroke(new BasicStroke(2));
71
72     // Calculate the starting positions to center the matrix
73     int startX = edgePadding;
74     int startY = edgePadding;
75
76     // + (imageWidth - (cols * (cellSize + padding) + edgePadding)) / 2;
77     // + (imageHeight - (rows * (cellSize + padding) + edgePadding)) / 2;
78
79     // Iterate through the matrix and draw circles for capital letters
80     for (int i = 0; i < rows; i++) {
81         for (int j = 0; j < cols; j++) {
82             char currentChar = solutionBoard[i][j];
83             int centerX = startX + j * (cellSize + padding) + cellSize / 2;
84             int centerY = startY + i * (cellSize + padding) + cellSize / 2;
85
86             // Draw a circle only for capital letters
87             if (Character.isUpperCase(currentChar)) {
88                 int charIndex = currentChar - 'A';
89                 g2d.setColor(COLORS[charIndex]);
90
91                 g2d.fillOval(centerX - cellSize / 2, centerY - cellSize / 2, cellSize, cellSize); // Draw circle
92                 g2d.setColor(Color.WHITE); // Draw letter in white inside the circle
93
94                 FontMetrics fontMetrics = g2d.getFontMetrics();
95                 int charWidth = fontMetrics.charWidth(currentChar);
96                 int charX = centerX - (charWidth / 2);
97                 int baselineY = centerY + (fontMetrics.getAscent() - fontMetrics.getDescent()) / 2;
98
99                 g2d.drawString(String.valueOf(currentChar), charX, baselineY); // Draw the letter
100
101             // Draw Links
102             // Up (i-1, j)
103             if (i > 0 && solutionBoard[i-1][j] == currentChar) {
104                 drawLinkBetweenCells(g2d, centerX, centerY - cellSize/2, centerX, centerY - (cellSize/2 + padding), COLORS[charIndex]);
105             }
106             // Down (i+1, j)
107             if (i < rows - 1 && solutionBoard[i+1][j] == currentChar) {
108                 drawLinkBetweenCells(g2d, centerX, centerY + cellSize/2, centerX, centerY + (cellSize/2 + padding), COLORS[charIndex]);
109             }
110             // Left (i, j-1)
111             if (j > 0 && solutionBoard[i][j-1] == currentChar) {
112                 drawLinkBetweenCells(g2d, centerX - cellSize/2, centerY, centerX - (cellSize/2 + padding), centerY, COLORS[charIndex]);
113             }
114             // Right (i, j+1)
115             if (j < cols - 1 && solutionBoard[i][j+1] == currentChar) {
116                 drawLinkBetweenCells(g2d, centerX + cellSize/2, centerY, centerX + (cellSize/2 + padding), centerY, COLORS[charIndex]);
117             }
118         } else {
119             if (mode.equals("DARK")) {
120                 g2d.setColor(Color.BLACK);
121             } else {
122                 g2d.setColor(Color.WHITE);
123             }
124             g2d.fillRect(centerX - cellSize / 2, centerY - cellSize / 2, cellSize, cellSize); // Clear the area
125         }
126     }
127 }
128
129 g2d.dispose();
130
131 try {
132     ImageIO.write(image, "PNG", new File(filename));
133     System.out.println("Image saved as " + filename);
134 } catch (IOException e) {
135     System.out.println("Error saving the image: " + e.getMessage());
136 }
137
138 }
```

Package GUI

File MainFrame

File ini menyimpan window awal ketika memulai program dan bertugas untuk menerima input dari user. Di bawah adalah fungsi untuk memanggil mainframe, selain untuk inisiasi window, juga terdapat fungsi untuk delete folder temp ketika program selesai digunakan

```
● ○ ● Tucil1_13523100 - MainFrame.java

14 public class MainFrame extends JFrame {
15
16     private JButton chooseFileButton;
17     private JButton uploadFileButton;
18     private JButton solveButton;
19     private File selectedFile = null;
20
21     public MainFrame() {
22         String currentDirectory = System.getProperty("user.dir");
23         File currentDirFile = new File(currentDirectory);
24         String tempString = currentDirFile + File.separator + "temp";
25
26         initialize();
27
28         this.addWindowListener(new WindowAdapter() {
29             @Override
30             public void windowClosing(WindowEvent e) {
31                 deleteTempFolder(new File(tempString));
32             }
33         });
34     }
```

Implementasi inisiasi MainFrame (1)

```
● ● ● Tucil1_13523100 - MainFrame.java

36 public void initialize() {
37     setTitle("Peak Brute Force IQ Puzzler Pro Solver (Def not brain rot fr fr)");
38     setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
39
40     getContentPane().setBackground(new Color(240, 240, 240));
41
42     JPanel mainPanel = new JPanel(new BorderLayout(20, 20));
43     mainPanel.setBorder(BorderFactory.createEmptyBorder(20, 20, 20, 20));
44     mainPanel.setBackground(new Color(230, 230, 230));
45
46     // Left panel (dynamic)
47     JPanel leftPanel = new JPanel(new BorderLayout(20, 20));
48     leftPanel.setBackground(new Color(230, 230, 230));
49
50     leftPanel.setLayout(new BoxLayout(leftPanel, BoxLayout.Y_AXIS));
51     leftPanel.setBackground(Color.WHITE);
52
53     JPanel topLeftPanel = new JPanel();
54     topLeftPanel.setLayout(new BoxLayout(topLeftPanel, BoxLayout.Y_AXIS));
55     topLeftPanel.setBackground(Color.WHITE);
56
57     JLabel boardLabel = new JLabel("Empty Board");
58     boardLabel.setAlignmentX(Component.CENTER_ALIGNMENT);
59     topLeftPanel.add(boardLabel);
60
61     JPanel emptyBox = new JPanel();
62     emptyBox.setAlignmentX(Component.CENTER_ALIGNMENT);
63     emptyBox.setBackground(Color.WHITE);
64
65     topLeftPanel.add(emptyBox);
66
67     JPanel bottomLeftPanel = new JPanel();
68     bottomLeftPanel.setLayout(new BoxLayout(bottomLeftPanel, BoxLayout.Y_AXIS));
69     bottomLeftPanel.setBackground(Color.WHITE);
70
71     JLabel piecesLabel = new JLabel("Pieces");
72     piecesLabel.setAlignmentX(Component.CENTER_ALIGNMENT);
73     bottomLeftPanel.add(piecesLabel);
74
75     JPanel[] imagePanels = new JPanel[26];
76     JLabel[] imageLabels = new JLabel[26];
77
78     JPanel gridPanel = new JPanel();
79     gridPanel.setLayout(new GridLayout(0, 7, 10, 10)); // 0 rows (dynamic), 6 columns, 10px gap
```

Implementasi inisiasi MainFrame (2)

```
● ● ● Tucil1_13523100 - MainFrame.java

81  for (int i = 0; i < 26; i++) {
82      imagePanels[i] = new JPanel();
83      imagePanels[i].setBackground(Color.WHITE);
84      imagePanels[i].setPreferredSize(new Dimension(150, 150));
85
86      imageLabels[i] = new JLabel("Piece " + (i + 1));
87      imageLabels[i].setHorizontalAlignment(SwingConstants.CENTER);
88      imageLabels[i].setVerticalAlignment(SwingConstants.CENTER);
89
90      imagePanels[i].add(imageLabels[i]);
91
92      gridPanel.add(imagePanels[i]);
93  }
94
95 bottomLeftPanel.add(gridPanel);
96
97 leftPanel.add(topLeftPanel);
98 leftPanel.add(bottomLeftPanel);
99
100 // marginPanel to match mainPanel padding
101 JPanel marginPanelLeft = new JPanel();
102 marginPanelLeft.setLayout(new BorderLayout());
103 marginPanelLeft.setBorder(BorderFactory.createEmptyBorder(0, 20, 0, 0));
104 marginPanelLeft.add(leftPanel, BorderLayout.CENTER);
105
106 mainPanel.add(marginPanelLeft, BorderLayout.CENTER);
107
108 // Right panel (buttons)
109 JPanel rightPanel = new JPanel();
110 rightPanel.setPreferredSize(new Dimension(300, 150));
111 rightPanel.setLayout(new BoxLayout(rightPanel, BoxLayout.Y_AXIS));
112 rightPanel.setBackground(new Color(230, 230, 230));
113
114 JPanel fixedPanel = new JPanel();
115 fixedPanel.setLayout(new BoxLayout(fixedPanel, BoxLayout.Y_AXIS));
116 fixedPanel.setBackground(Color.WHITE);
117
118 Border paddingBorder = BorderFactory.createEmptyBorder(20, 20, 20, 20);
119 fixedPanel.setBorder(paddingBorder);
```

Implementasi inisiasi MainFrame (3)

```
 121 // File buttons
 122     chooseFileButton = new JButton("Choose File");
 123     uploadFileButton = new JButton("Upload File");
 124
 125     chooseFileButton.setPreferredSize(new Dimension(200, 40));
 126     uploadFileButton.setPreferredSize(new Dimension(200, 40));
 127     chooseFileButton.setMaximumSize(new Dimension(200, 40));
 128     uploadFileButton.setMaximumSize(new Dimension(200, 40));
 129
 130     chooseFileButton.setBackground(Color.BLACK);
 131     uploadFileButton.setBackground(Color.BLACK);
 132     chooseFileButton.setFont(new Font("Arial", Font.PLAIN, 15));
 133     uploadFileButton.setFont(new Font("Arial", Font.PLAIN, 15));
 134     chooseFileButton.setForeground(Color.WHITE);
 135     uploadFileButton.setForeground(Color.WHITE);
 136     chooseFileButton.setFocusPainted(false);
 137     uploadFileButton.setFocusPainted(false);
 138
 139     chooseFileButton.setAlignmentX(Component.CENTER_ALIGNMENT);
 140     uploadFileButton.setAlignmentX(Component.CENTER_ALIGNMENT);
 141
 142     // Filename text
 143     JLabel fileName = new JLabel("No file selected", SwingConstants.CENTER);
 144     fileName.setAlignmentX(Component.CENTER_ALIGNMENT);
 145
 146     // Choose file button functionality
 147     chooseFileButton.addActionListener((ActionEvent e) -> {
 148         JFileChooser fileChooser = new JFileChooser();
 149         String currentDirectory = System.getProperty("user.dir");
 150         File currentDirFile = new File(currentDirectory);
 151
 152         File initialDirectory = new File(currentDirectory);
 153
 154         if (initialDirectory.exists() && initialDirectory.isDirectory()) {
 155             fileChooser.setCurrentDirectory(initialDirectory);
 156         }
 157
 158         FileNameExtensionFilter txtFilter = new FileNameExtensionFilter("Text Files (.txt)", "txt");
 159         fileChooser.setFileFilter(txtFilter);
 160
 161         int result = fileChooser.showOpenDialog(MainFrame.this);
 162
 163         if (result == JFileChooser.APPROVE_OPTION) {
 164             selectedFile = fileChooser.getSelectedFile();
 165             fileName.setText("Selected file: " + selectedFile.getName());
 166         } else {
 167             System.out.println("No file chosen.");
 168         }
 169     });

```

Implementasi inisiasi MainFrame (4)

```
Tucil1_13523100 - MainFrame.java

171  AtomicReference<InputFormat> formRef = new AtomicReference<>(null);
172
173      // Upload file button functionality
174      uploadFileButton.addActionListener((ActionEvent e) -> {
175          if (selectedFile != null) {
176              System.out.println("Uploading file: " + selectedFile.getAbsolutePath());
177              InputFormat form = Input.readTxt(selectedFile.getAbsolutePath());
178              formRef.set(form);
179              InputPuzzlerPro input = form.input;
180
181              if (input == null) {
182                  JOptionPane.showMessageDialog(this, form.errorMessage,
183                      "Gooner input detected!", JOptionPane.ERROR_MESSAGE);
184              } else {
185                  String currentDirectory = System.getProperty("user.dir");
186                  File currentDirFile = new File(currentDirectory);
187                  String tempString = currentDirFile + File.separator + "temp";
188                  File tempFolder = new File(tempString);
189
190                  if (tempFolder.exists()) {
191                      for (File file : tempFolder.listFiles()) {
192                          if (file.isDirectory()) {
193                              deleteDirectory(file);
194                          } else {
195                              file.delete();
196                          }
197                      }
198                      tempFolder.delete();
199                  }
200
201                  InputImageGenerator.saveInput(input);
202
203                  if (!tempFolder.exists()) {
204                      JOptionPane.showMessageDialog(this, "Input process failed, input image not created",
205                          "Error", JOptionPane.ERROR_MESSAGE);
206                  } else {
207                      File emptyBoardFile = new File(tempFolder, "emptyBoard.png");
208                      if (emptyBoardFile.exists()) {
209                          ImageIcon emptyBoardIcon = new ImageIcon(emptyBoardFile.getAbsolutePath());
210                          JLabel emptyBoardLabel = new JLabel(emptyBoardIcon);
211
212                          SwingUtilities.invokeLater(() -> {
213                              emptyBox.removeAll();
214                              emptyBox.add(emptyBoardLabel, BorderLayout.CENTER);
215                              emptyBox.revalidate();
216                              emptyBox.repaint();
217                          });
218                      }
219
220                      for (int i = 0; i < input.piecesList.size(); i++) {
221                          String pieceFileName = "piece" + (i + 1) + ".png";
222                          File pieceFile = new File(tempFolder, pieceFileName);
223
224                          if (pieceFile.exists()) {
225                              ImageIcon pieceIcon = new ImageIcon(pieceFile.getAbsolutePath());
226                              JLabel pieceLabel = new JLabel(pieceIcon);
227                              pieceLabel.setHorizontalTextPosition(SwingConstants.CENTER);
228                              pieceLabel.setVerticalTextPosition(SwingConstants.CENTER);
229
230                              int panelIndex = i;
231                              SwingUtilities.invokeLater(() -> {
232                                  JPanel panelToUpdate = imagePanels[panelIndex];
233                                  panelToUpdate.removeAll();
234                                  panelToUpdate.setLayout(new BorderLayout());
235                                  panelToUpdate.add(pieceLabel, BorderLayout.CENTER);
236                                  panelToUpdate.revalidate();
237                                  panelToUpdate.repaint();
238                              });
239                          }
240                      }
241                  }
242              }
243          } else {
244              JOptionPane.showMessageDialog(this, "No file selected to upload.",
245                  "Error", JOptionPane.ERROR_MESSAGE);
246          }
247      });


```

Implementasi inisiasi MainFrame (5)

```
Tucil1_13523100 - MainFrame.java

249     fixedPanel.add(chooseFileButton);
250     fixedPanel.add(Box.createVerticalStrut(10));
251     fixedPanel.add(uploadFileButton);
252     fixedPanel.add(Box.createVerticalStrut(10));
253     fixedPanel.add(fileName);
254
255     rightPanel.add(fixedPanel);
256
257     mainPanel.add(rightPanel, BorderLayout.EAST);
258     mainPanel.setBorder(BorderFactory.createEmptyBorder(20, 0, 20, 20));
259
260     // Bottom Section
261     JPanel bottomPanel = new JPanel(new BorderLayout());
262     bottomPanel.setBackground(new Color(230, 230, 230));
263
264     solveButton = new JButton("Solve");
265     solveButton.setPreferredSize(new Dimension(500, 50));
266     solveButton.setBackground(new Color(0, 122, 255));
267     solveButton.setForeground(Color.WHITE);
268     solveButton.setFocusPainted(false);
269     solveButton.setFont(new Font("Arial", Font.BOLD, 18));
270
271     // Solve button functionality
272     solveButton.addActionListener(new ActionListener() {
273         @Override
274         public void actionPerformed(ActionEvent e) {
275             if (selectedFile != null && formRef.get() != null) {
276                 InputFormat form = formRef.get();
277                 if (form.input == null) {
278                     JOptionPane.showMessageDialog(MainFrame.this, form.errorMessage,
279                         "Error", JOptionPane.ERROR_MESSAGE);
280                 } else {
281                     new Solution(form);
282                 }
283             } else {
284                 JOptionPane.showMessageDialog(MainFrame.this, "No file selected, can't solve",
285                     "Error", JOptionPane.ERROR_MESSAGE);
286             }
287         }
288     });
289 }
290
291     bottomPanel.add(solveButton);
292
293     JPanel marginPanelBottom = new JPanel();
294     marginPanelBottom.setLayout(new BorderLayout());
295     marginPanelBottom.setBorder(BorderFactory.createEmptyBorder(0, 20, 0, 0));
296
297     marginPanelBottom.add(bottomPanel, BorderLayout.CENTER);
298
299     mainPanel.add(marginPanelBottom, BorderLayout.SOUTH);
300
301     add(mainPanel);
302     setMinimumSize(new Dimension(1200, 800));
303     pack();
304
305     // Display the frame
306     setVisible(true);
```

Fungsi pembantu untuk file management



Tucil1_13523100 - MainFrame.java

```
309  private void deleteDirectory(File directory) {
310      if (directory.isDirectory()) {
311          String[] files = directory.list();
312          if (files != null) {
313              for (String file : files) {
314                  File currentFile = new File(directory, file);
315                  if (currentFile.isDirectory()) {
316                      deleteDirectory(currentFile);
317                  } else {
318                      currentFile.delete();
319                  }
320              }
321          }
322      }
323      directory.delete();
324  }
325
326  private void deleteTempFolder(File folder) {
327      if (folder.exists()) {
328          try {
329              Files.walk(folder.toPath())
330                  .map(java.nio.file.Path::toFile)
331                  .sorted((f1, f2) -> f2.compareTo(f1))
332                  .forEach(File::delete);
333              System.out.println("Temporary folder deleted successfully!");
334          } catch (IOException ex) {
335              System.err.println("Failed to delete temp folder: " + ex.getMessage());
336          }
337      }
338  }
```

File Solution

Fungsi yang dipanggil untuk membuka solution dan juga delete folder temp kalau program ditutup



```
Tucil1_13523100 - Solution.java

14 public Solution(InputFormat form) {
15     String currentDirectory = System.getProperty("user.dir");
16     File currentDirFile = new File(currentDirectory);
17     String tempString = currentDirFile + File.separator + "temp";
18
19     initialize(form);
20
21     this.addWindowListener(new WindowAdapter() {
22         @Override
23         public void windowClosing(WindowEvent e) {
24             deleteTempFolder(new File(tempString));
25         }
26     });
27 }
```

Implementasi Solution (1)

```
● ○ ● Tucil1_13523100 - Solution.java

29 public void initialize(InputFormat form) {
30     OutputPuzzlerPro output = BruteForce.getOutput(form);
31     output.printDetails();
32
33     // Temporary folder for saving the solution
34     String currentDirectory = System.getProperty("user.dir");
35     File currentDirFile = new File(currentDirectory);
36     String tempString = currentDirFile + File.separator + "temp";
37     File tempFolder = new File(tempString);
38
39     if (!tempFolder.exists()) {
40         if (tempFolder.mkdirs()) {
41             System.out.println("Temporary folder created: " + tempFolder.getAbsolutePath());
42         } else {
43             System.out.println("Failed to create temporary folder.");
44         }
45     } else {
46         System.out.println("Temporary folder already exists.");
47     }
48
49     // Define the file path for the solution image
50     String filename = tempFolder.getAbsolutePath() + File.separator + "solution.png";
51
52     JPanel topPanel = new JPanel();
53     topPanel.setLayout(new BoxLayout(topPanel, BoxLayout.Y_AXIS));
54     topPanel.setBackground(Color.WHITE);
55
56     JLabel boardLabel = new JLabel("Solution Board");
57     boardLabel.setFont(new Font("Arial", Font.BOLD, 24));
58     boardLabel.setAlignmentX(Component.CENTER_ALIGNMENT);
59     boardLabel.setBorder(BorderFactory.createEmptyBorder(20, 0, 0, 0));
60     topPanel.add(boardLabel);
61
62     // Check if the solution board is available
63     if (output.filledBoardSolution != null) {
64         OutputImage.solutionDisplayImage(output.filledBoardSolution, filename, "LIGHT");
65
66         ImageIcon boardImageIcon = new ImageIcon(filename);
67         JLabel boardImageLabel = new JLabel(boardImageIcon);
68
69         boardImageLabel.setAlignmentX(Component.CENTER_ALIGNMENT);
70         topPanel.add(boardImageLabel);
71     } else {
72         JPanel noSolutionPanel = new JPanel();
73         noSolutionPanel.setPreferredSize(new Dimension(400, 400));
74         noSolutionPanel.setBackground(Color.WHITE);
75         JLabel noSolutionLabel = new JLabel("No Valid Solution");
76         noSolutionLabel.setAlignmentX(Component.CENTER_ALIGNMENT);
77         noSolutionPanel.add(noSolutionLabel);
78         noSolutionPanel.setAlignmentX(Component.CENTER_ALIGNMENT);
79         topPanel.add(noSolutionPanel);
80     }
}
```

Implementasi Solution (2)

```
82 // Panel to house infoPanel
83 JPanel centerPanel = new JPanel();
84 centerPanel.setBackground(new Color(240, 240, 240));
85
86 // Execution time and number of cases explored stored in infoPanel
87 JPanel infoPanel = new JPanel();
88 infoPanel.setLayout(new BoxLayout(infoPanel, BoxLayout.Y_AXIS));
89 infoPanel.setBackground(Color.WHITE);
90 infoPanel.setBorder(BorderFactory.createEmptyBorder(20, 20, 20, 20));
91
92 infoPanel.setPreferredSize(new Dimension(600, 100));
93 infoPanel.setMaximumSize(new Dimension(600, 100));
94
95 // Create the execution time label
96 JLabel executionTimeLabel = new JLabel("Execution Time = " + output.executionTime + " ms");
97 executionTimeLabel.setFont(new Font("Arial", Font.BOLD, 20));
98 executionTimeLabel.setAlignmentX(Component.LEFT_ALIGNMENT);
99 infoPanel.add(executionTimeLabel);
100
101 infoPanel.add(Box.createVerticalStrut(10));
102
103 // Create the cases explored label
104 JLabel casesExploredLabel = new JLabel("Number of Cases Explored = " + output.casesExplored);
105 casesExploredLabel.setFont(new Font("Arial", Font.BOLD, 20));
106 casesExploredLabel.setAlignmentX(Component.LEFT_ALIGNMENT);
107 infoPanel.add(casesExploredLabel);
108
109 centerPanel.add(infoPanel);
110
111 // Bottom Panel with buttons
112 JPanel bottomPanel = new JPanel();
113 bottomPanel.setLayout(new GridLayout(1, 2, 10, 10)); // 1 row, 2 columns, 10px horizontal and vertical gap
114 bottomPanel.setBackground(Color.WHITE);
115 bottomPanel.setBorder(new EmptyBorder(20, 20, 20, 20));
116
117 // Button for saving the solution as text and image
118 JButton saveTextButton = new JButton("Save Solution Text");
119 JButton saveImageButton = new JButton("Save Solution Image");
120
121 saveTextButton.setMaximumSize(new Dimension(300, 100));
122 saveImageButton.setMaximumSize(new Dimension(300, 100));
123
124 saveTextButton.setBackground(Color.BLACK);
125 saveImageButton.setBackground(Color.BLACK);
126 saveTextButton.setFont(new Font("Arial", Font.PLAIN, 15));
127 saveImageButton.setFont(new Font("Arial", Font.PLAIN, 15));
128 saveTextButton.setForeground(Color.WHITE);
129 saveImageButton.setForeground(Color.WHITE);
130 saveTextButton.setFocusPainted(false);
131 saveImageButton.setFocusPainted(false);
```

Implementasi Solution (3)

```
Tucil1_13523100 - Solution.java

133 // Save solution text button functionality
134 saveTextButton.addActionListener(new ActionListener() {
135     @Override
136     public void actionPerformed(ActionEvent e) {
137         JFileChooser fileChooser = new JFileChooser(currentDirFile);
138         fileChooser.setDialogTitle("Choose a location to save the solution text");
139         fileChooser.setSelectedFile(new File("solution.txt"));
140
141         int returnValue = fileChooser.showSaveDialog(null);
142         if (returnValue == JFileChooser.APPROVE_OPTION) {
143             File selectedFile = fileChooser.getSelectedFile();
144             // Ensure the file has the ".txt" extension
145             if (!selectedFile.getName().endsWith(".txt")) {
146                 selectedFile = new File(selectedFile.getAbsolutePath() + ".txt");
147             }
148             String fname = selectedFile.getAbsolutePath();
149             OutputTxt.output(form.input, output, fname); // Save the text file
150         }
151     }
152 });
153 bottomPanel.add(saveTextButton);
154
155 // Save solution as image button functionality
156 saveImageButton.addActionListener(new ActionListener() {
157     @Override
158     public void actionPerformed(ActionEvent e) {
159         JFileChooser fileChooser = new JFileChooser(currentDirFile);
160         fileChooser.setDialogTitle("Choose a location to save the solution image");
161         fileChooser.setSelectedFile(new File("solution.png"));
162
163         int returnValue = fileChooser.showSaveDialog(null);
164         if (returnValue == JFileChooser.APPROVE_OPTION) {
165             File selectedFile = fileChooser.getSelectedFile();
166             // Ensure the file has the ".png" extension
167             if (!selectedFile.getName().endsWith(".png")) {
168                 selectedFile = new File(selectedFile.getAbsolutePath() + ".png");
169             }
170             String fname = selectedFile.getAbsolutePath();
171             OutputImage.solutionImage(output.filledBoardSolution, fname, "LIGHT"); // Save the image
172         }
173     }
174 });
175 bottomPanel.add(saveImageButton);
176
177 // mainPanel
178 JPanel mainPanel = new JPanel(new BorderLayout(20, 20));
179 mainPanel.setBackground(new Color(240, 240, 240));
180 mainPanel.setBorder(BorderFactory.createEmptyBorder(20, 20, 20, 20));
181
182 // Add all the components to the main panel
183 mainPanel.add(topPanel, BorderLayout.NORTH);
184 mainPanel.add(centerPanel, BorderLayout.CENTER);
185 mainPanel.add(bottomPanel, BorderLayout.SOUTH);
186
187 JScrollPane scrollPane = new JScrollPane(mainPanel);
188 scrollPane.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED);
189 scrollPane.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED);
190
191 setTitle("Peak Brute Force IQ Puzzler Pro Solver Solution (Def not brain rot fr fr)");
192 setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
193 getContentPane().setBackground(new Color(240, 240, 240));
194 add(mainPanel);
195 setMinimumSize(new Dimension(1200, 800));
196 pack();
197 setVisible(true);
```

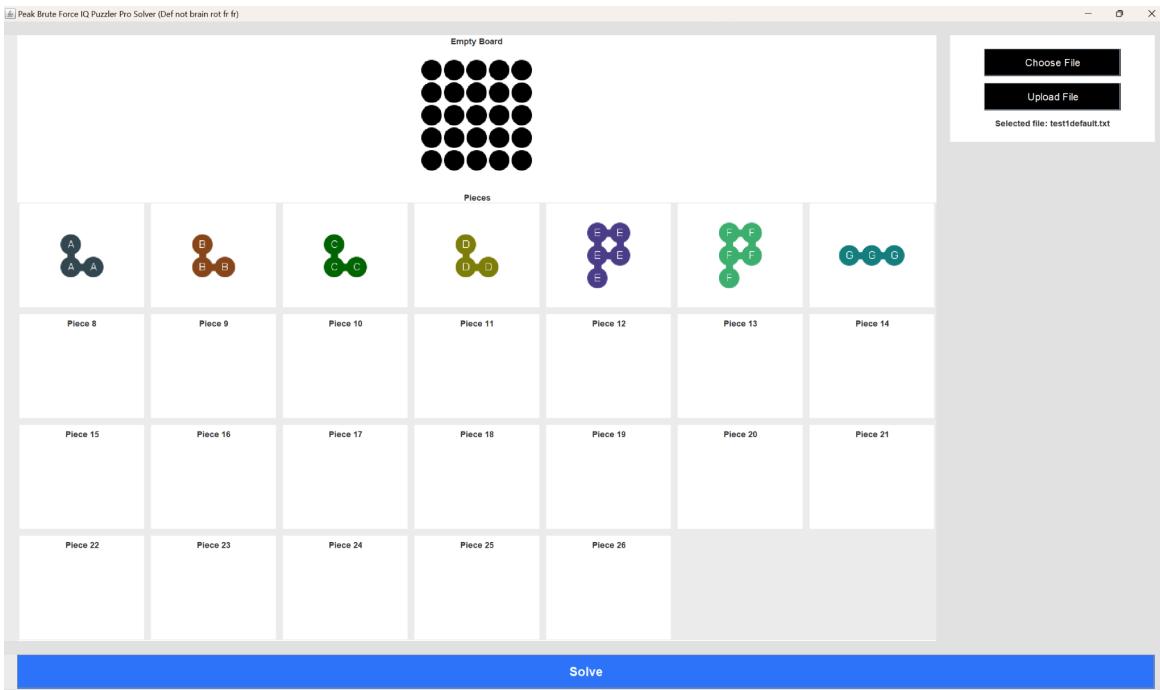
3. Test Case

Test Case 1 (default)

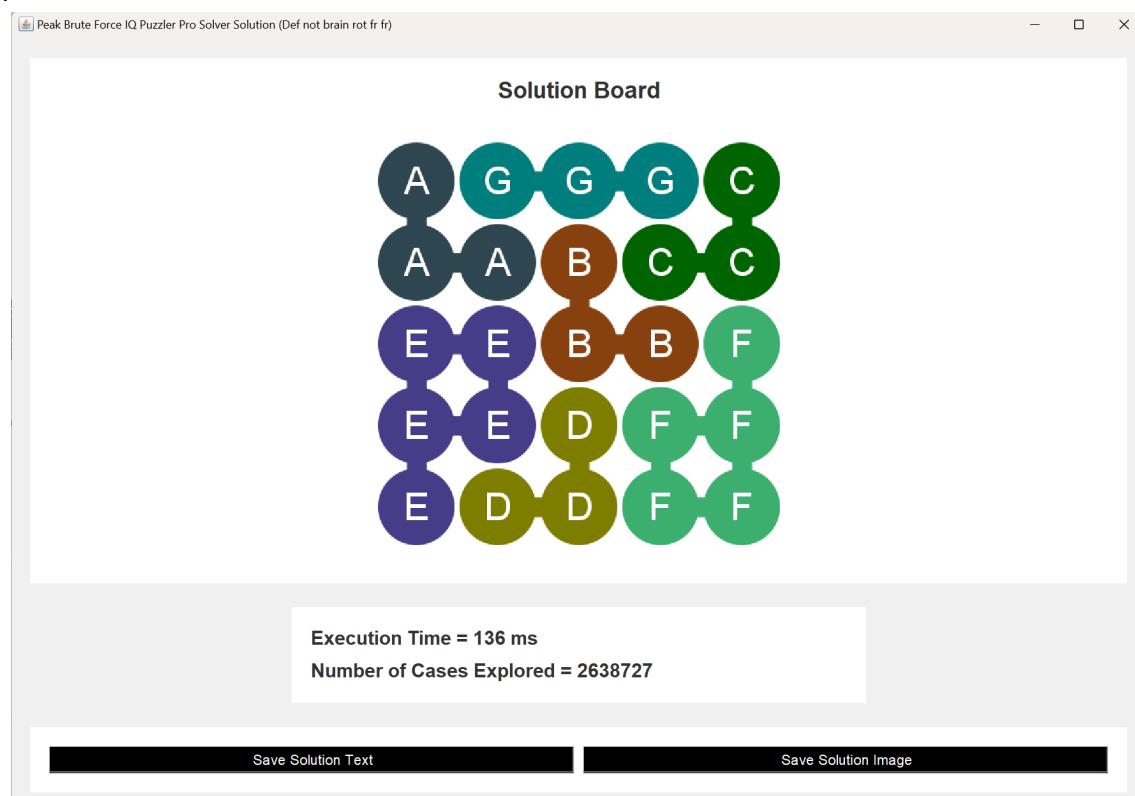
Input

```
test > test1default.txt
```

```
1 5 5 7
2 DEFAULT
3 A
4 AA
5 B
6 BB
7 C
8 CC
9 D
10 DD
11 EE
12 EE
13 E
14 FF
15 FF
16 F
17 GGG
```



Output



```
● ● ● Tucil1_13523100 - solution1default.txt
32 Solution:
33 A G G G C
34 A A B C C
35 E E B B F
36 E E D F F
37 E D D F F
38
39 Execution Time: 136 ms
40 Number of cases checked: 2638727
```

Test Case 2 (default)

Input

```
test > test2default.txt
```

```
1 5 11 12
```

```
2 DEFAULT
```

```
3 AA
```

```
4 A
```

```
5 A
```

```
6 BB
```

```
7 BBB
```

```
8 | C
```

```
9 CC
```

```
10 C
```

```
11 D
```

```
12 DD
```

```
13 | DD
```

```
14 E
```

```
15 E
```

```
16 EE
```

```
17 | E
```

```
18 FFFF
```

```
19 | F
```

```
20 GG
```

```
21 | G
```

```
22 | G
```

```
23 | G
```

```
24 H
```

```
25 HH
```

```
26 | I
```

```
27 II
```

```
28 | II
```

```
29 JJJ
```

```
30 J J
```

```
31 | K
```

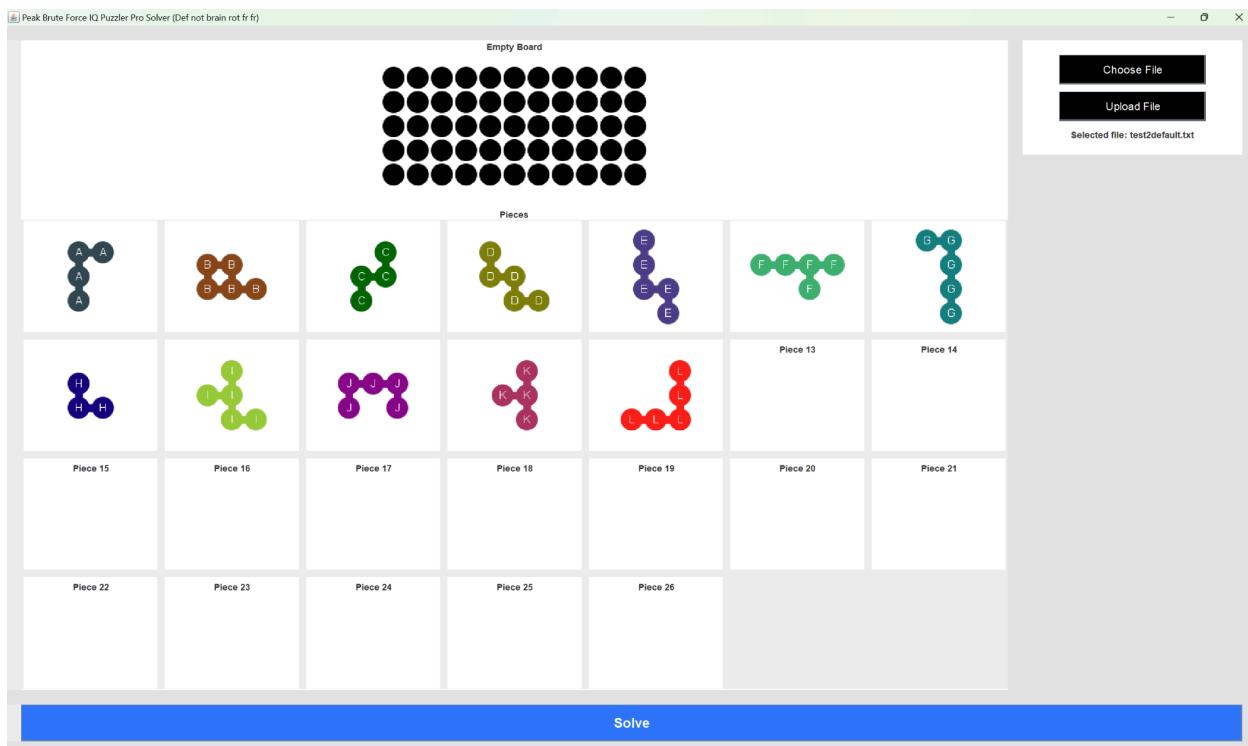
```
32 KK
```

```
33 | K
```

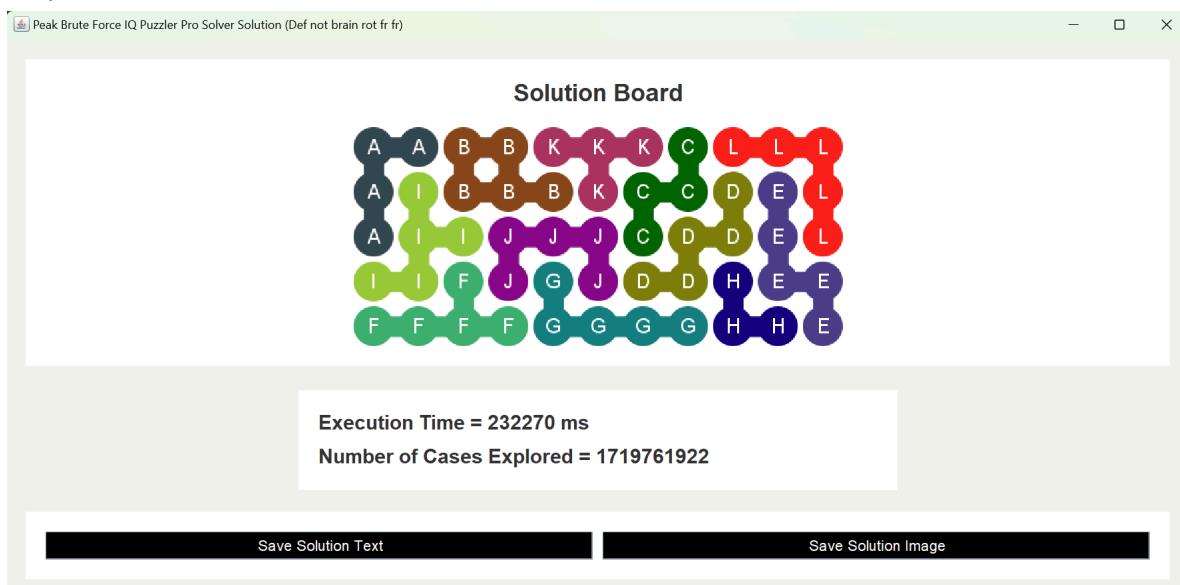
```
34 | L
```

```
35 | L
```

```
36 LLL
```



Output

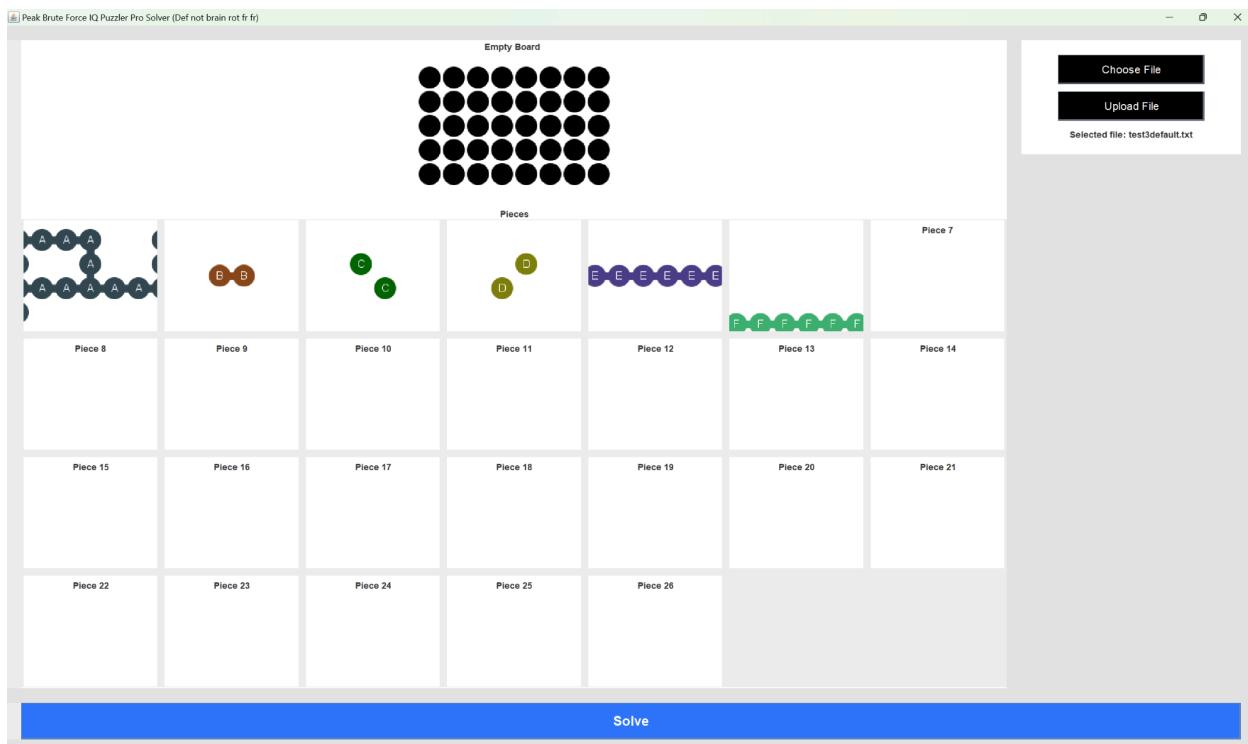


```
● ● ● Tucil1_13523100 - solution2default.txt

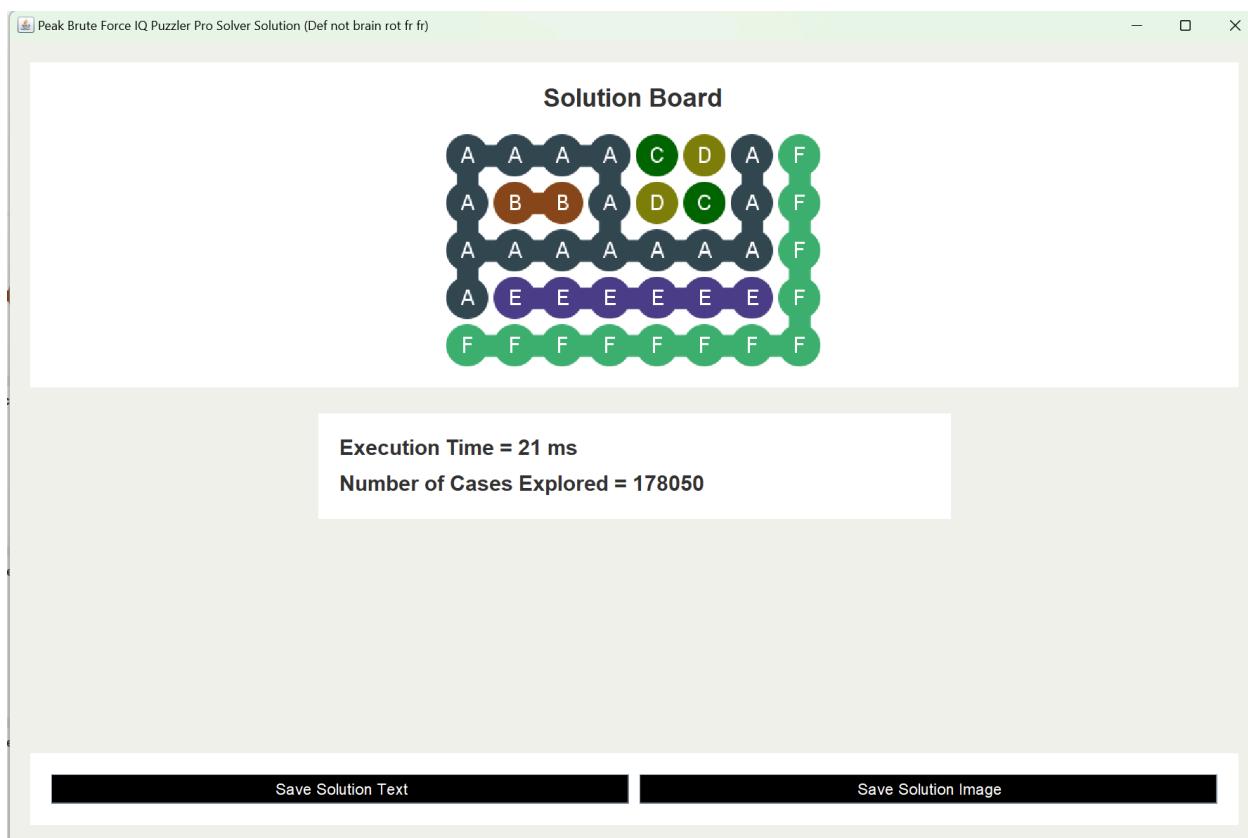
56 Solution:
57 A A B B K K K C L L L
58 A I B B B K C C D E L
59 A I I J J J C D D E L
60 I I F J G J D D H E E
61 F F F F G G G G H H E
62
63 Execution Time: 232270 ms
64 Number of cases checked: 1719761922
```

Test Case 3 (default)

```
test > test3default.txt
1 5 8 6
2 DEFAULT
3 AAAA A
4 A A A
5 AAAAAAA
6 A
7 BB
8 C
9 C
10 D
11 D
12 EEEEE
13 | F
14 | F
15 | F
16 | F
17 FFFFFFFF
```



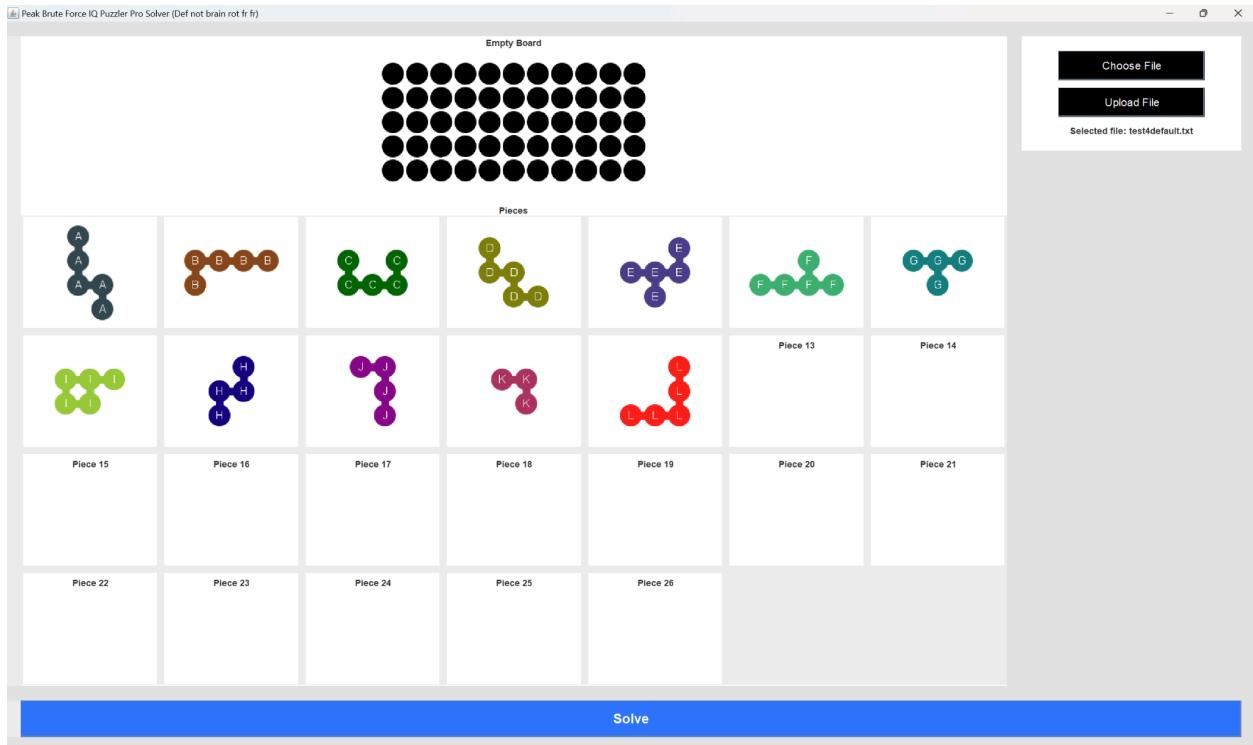
Output



● ● ● Tucil1_13523100 - solution3default.txt

31 Solution:
32 A A A A C D A F
33 A B B A D C A F
34 A A A A A A A F
35 A E E E E E E F
36 F F F F F F F F
37
38 Execution Time: 21 ms
39 Number of cases checked: 178050

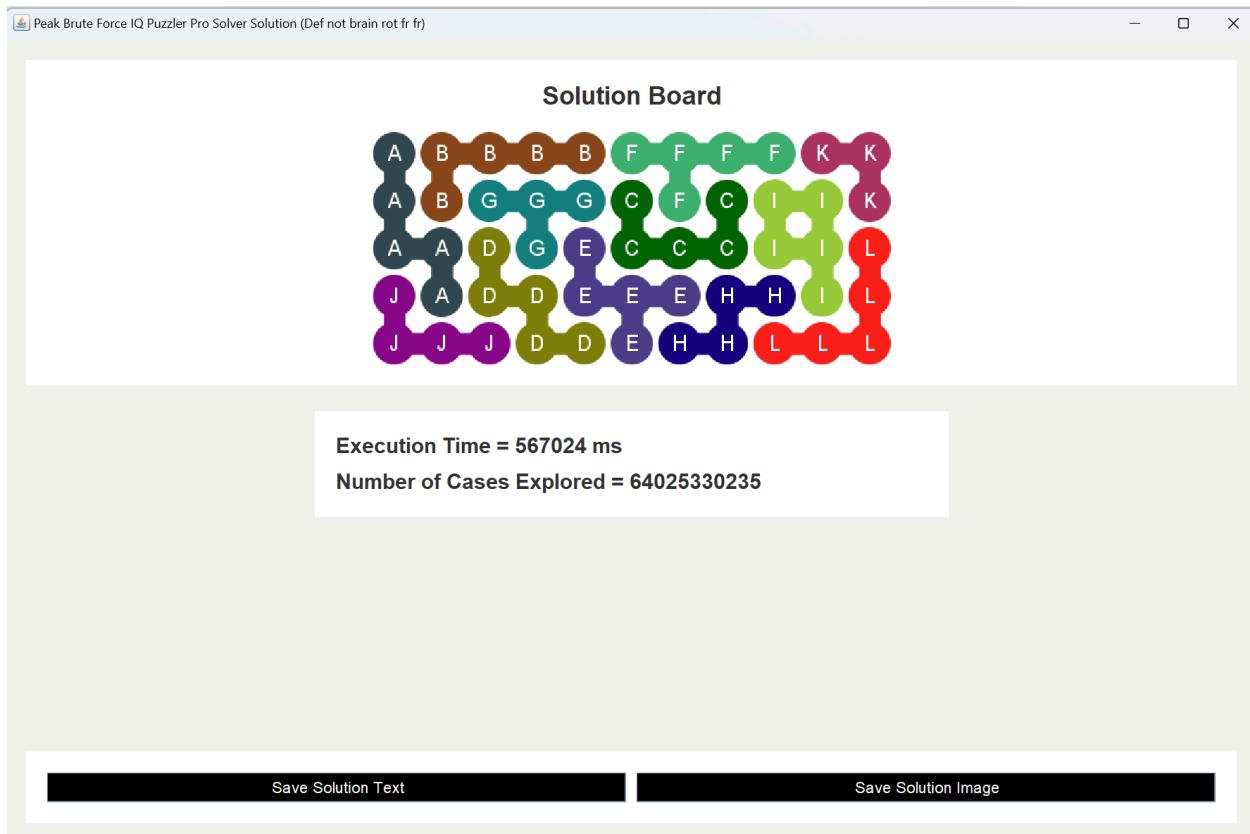
Test Case 4 (default)



test > test4default.txt

```
1 5 11 12
2 DEFAULT
3 A
4 A
5 AA
6 | A
7 BBBB
8 B
9 ▶ C C
10 CCC
11 D
12 | DD
13 | DD
14 | E
15 EEE
16 | E
17 | F
18 FFFF
19 GGG
20 | G
21 III
22 II
23 | H
24 HH
25 | H
26 JJ
27 | J
28 | J
29 KK
30 | K
31 | L
32 | L
33 LLL
```

Output



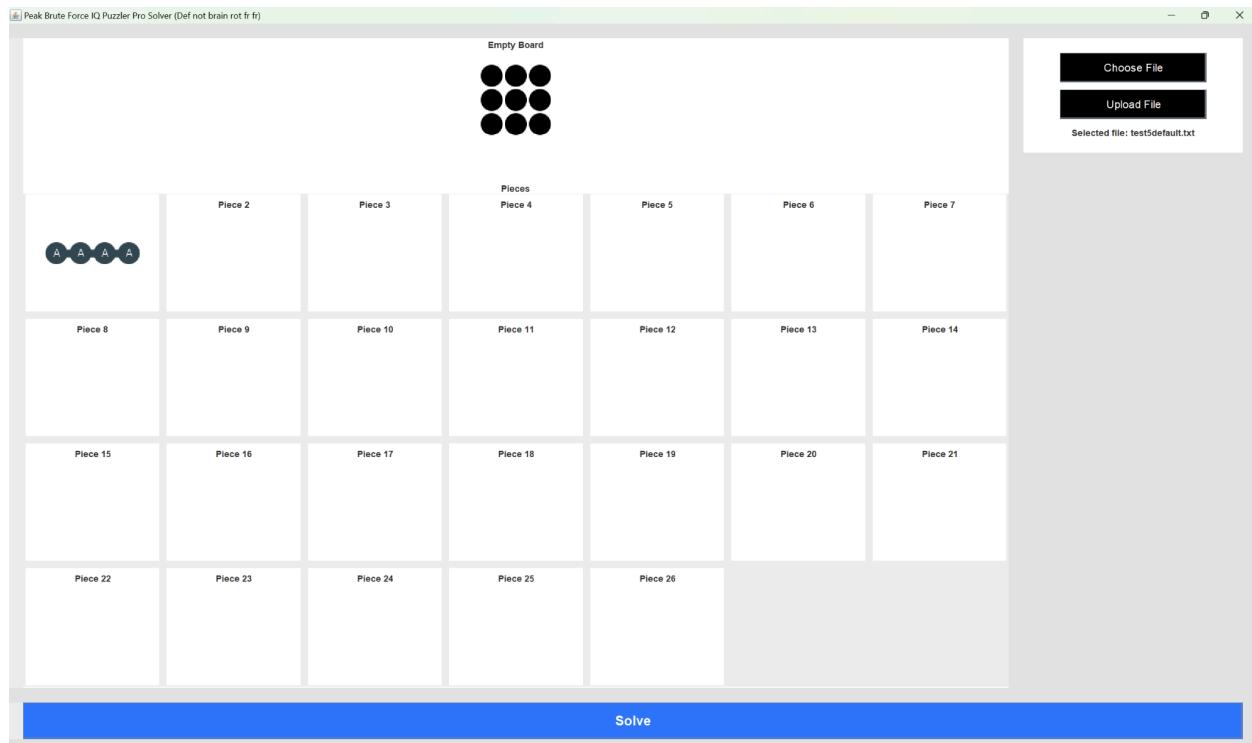
```
● ● ● Tucil1_13523100 - solution4default.txt

53 Solution:
54 A B B B B F F F F F K K
55 A B G G G C F C I I K
56 A A D G E C C C I I L
57 J A D D E E E H H I I L
58 J J J D D E H H L L L
59
60 Execution Time: 567024 ms
61 Number of cases checked: 64025330235
```

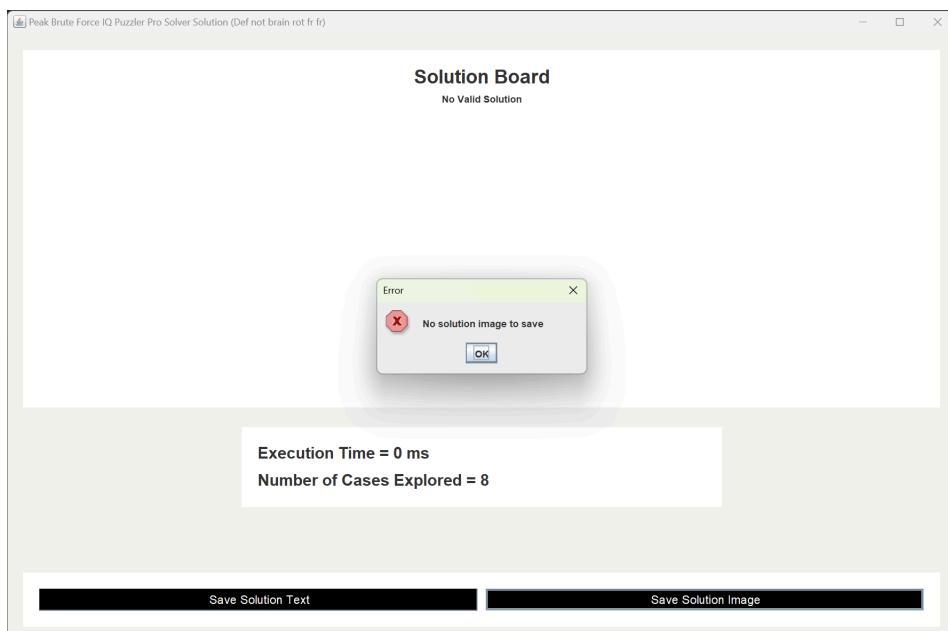
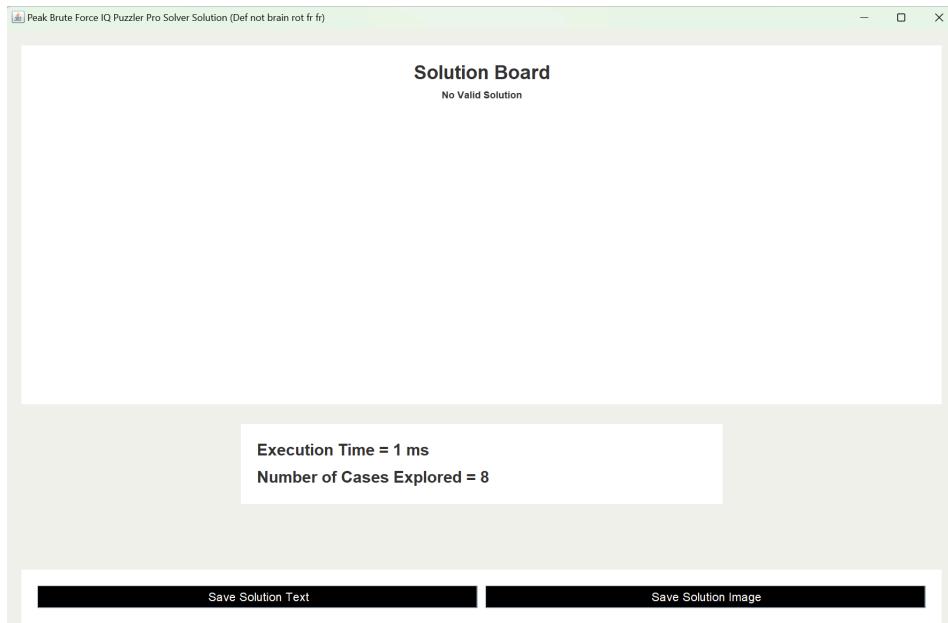
Test Case 5 (default)

Input

```
test > test5default.txt
1 3 3 1
2 DEFAULT
3 AAAA
```



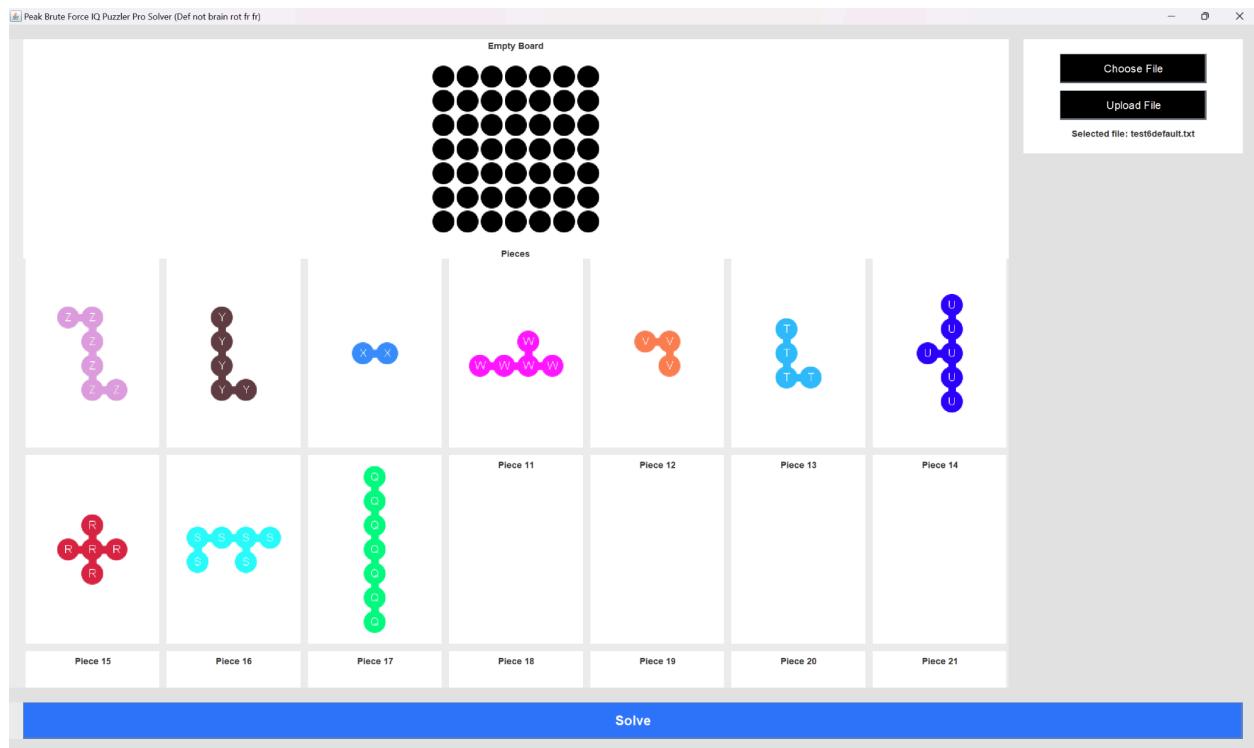
Output



```
● ○ ● Tucil1_13523100 ~ solution5default.txt
6  Pieces List:
7  Piece 1:
8  A A A A
9
10 Solution: No solution
11
12 Execution Time: 0 ms
13 Number of cases checked: 8
```

Test Case 6 (default)

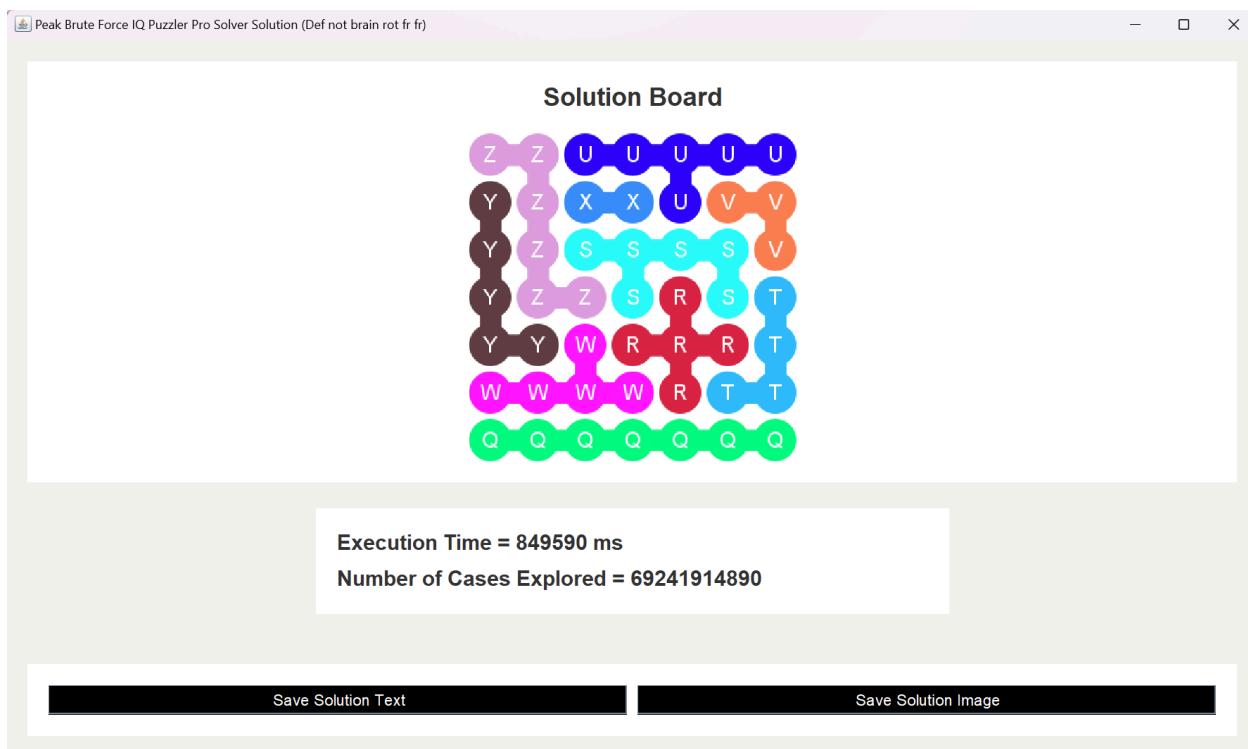
Input



test >  test6default.txt

```
1    7 7 10
2    DEFAULT
3    ZZ
4    |
5    Z
6    |
7    ZZ
8    Y
9    Y
10   YY
11   XX
12   |
13   WWWWW
14   VV
15   |
16   V
17   T
18   T
19   |
20   U
21   U
22   |
23   U
24   |
25   RRR
26   |
27   R
28   SSSS
29   S
30   Q
31   Q
32   Q
33   Q
34   Q
35   Q
```

Output

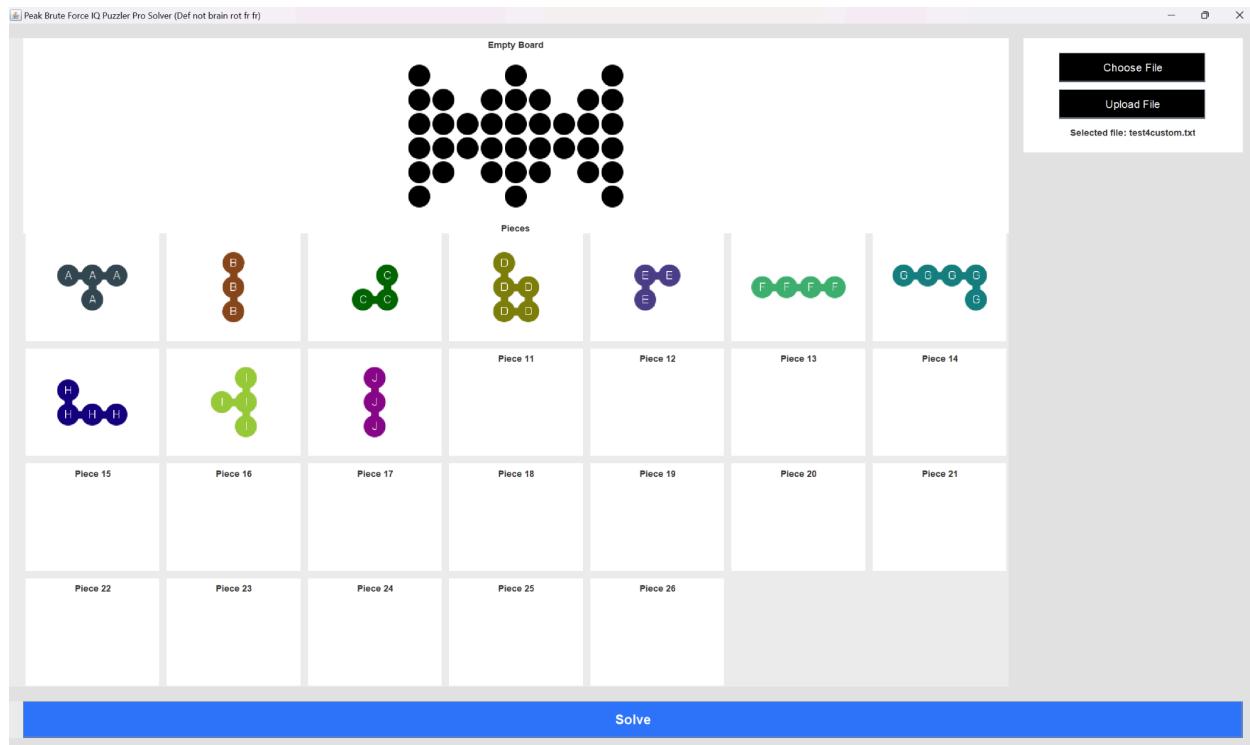


```
● ● ● Tucil1_13523100 - solution6default.txt

55 Solution:
56 Z Z U U U U U
57 Y Z X X U V V
58 Y Z S S S S V
59 Y Z Z S R S T
60 Y Y W R R R T
61 W W W W R T T
62 Q Q Q Q Q Q Q
63
64 Execution Time: 849590 ms
65 Number of cases checked: 69241914890
```

Test Case 7 (Custom)

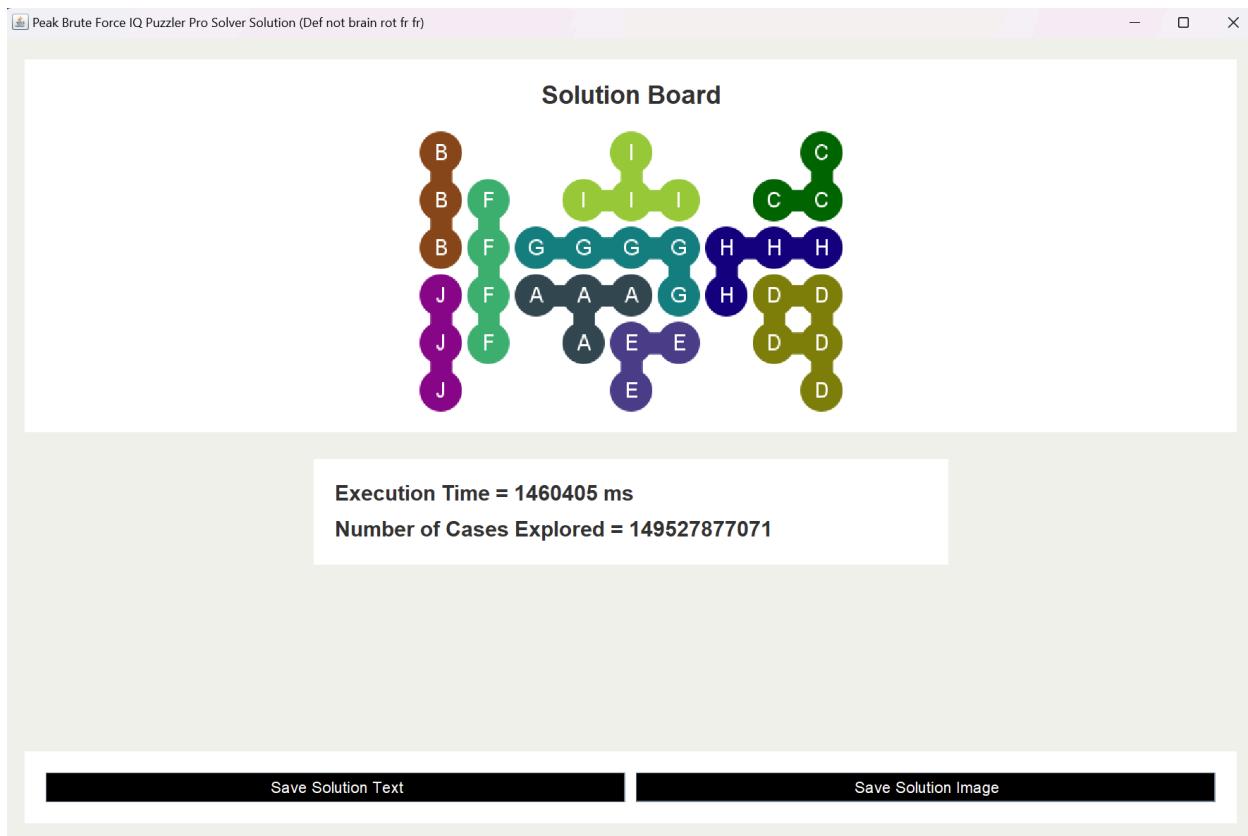
Input



test >  test4custom.txt

```
1   6 9 10
2   CUSTOM
3   X...X...X
4   XX.XXX.XX
5   XXXXXXXXXX
6   XXXXXXXXXX
7   XX.XXX.XX
8   X...X...X
9   AAA
10  | A
11  B
12  B
13  B
14  | C
15  CC
16  D
17  DD
18  DD
19  EE
20  E
21  FFFF
22  GGGG
23  | G
24  H
25  HHH
26  | I
27  II
28  | I
29  J
30  J
31  J
```

Output



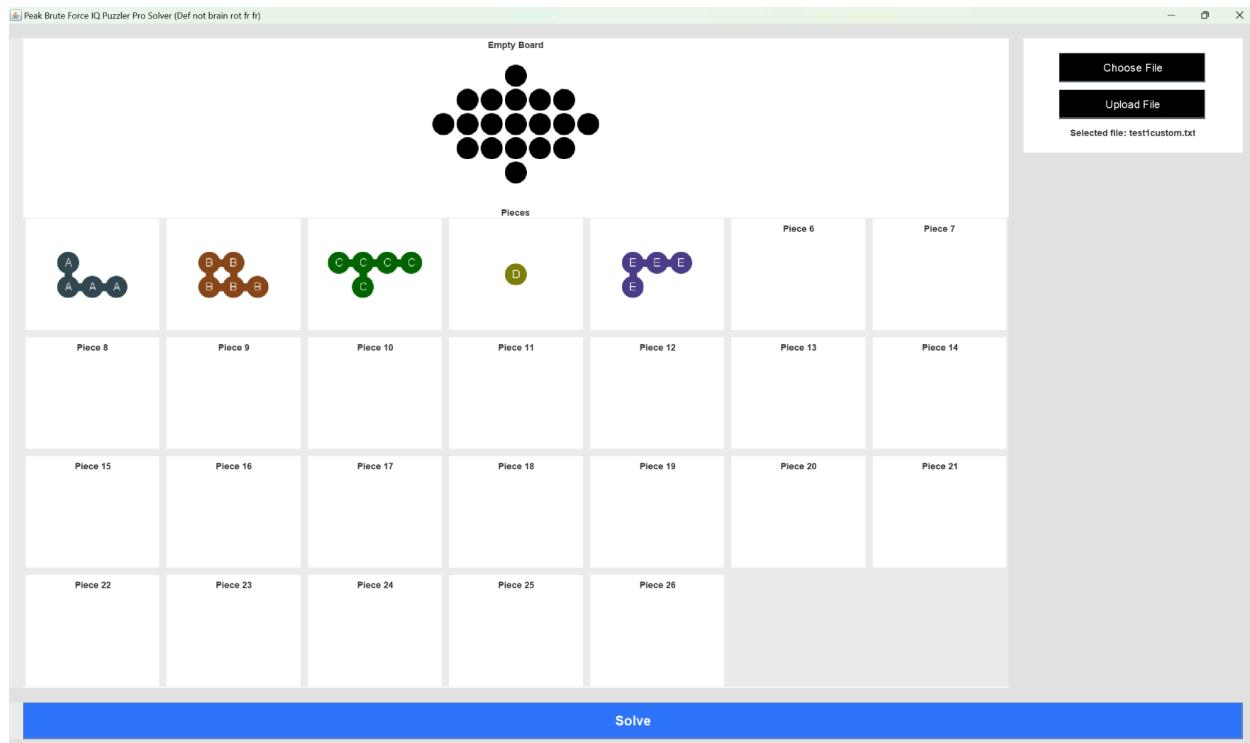
```
● ● ● Tucil1_13523100 - solution4custom.txt

44 Solution:
45 B I C
46 B F I I I C C
47 B F G G G G H H H
48 J F A A A G H D D
49 J F A E E D D
50 J E D
51
52 Execution Time: 1460405 ms
53 Number of cases checked: 149527877071
```

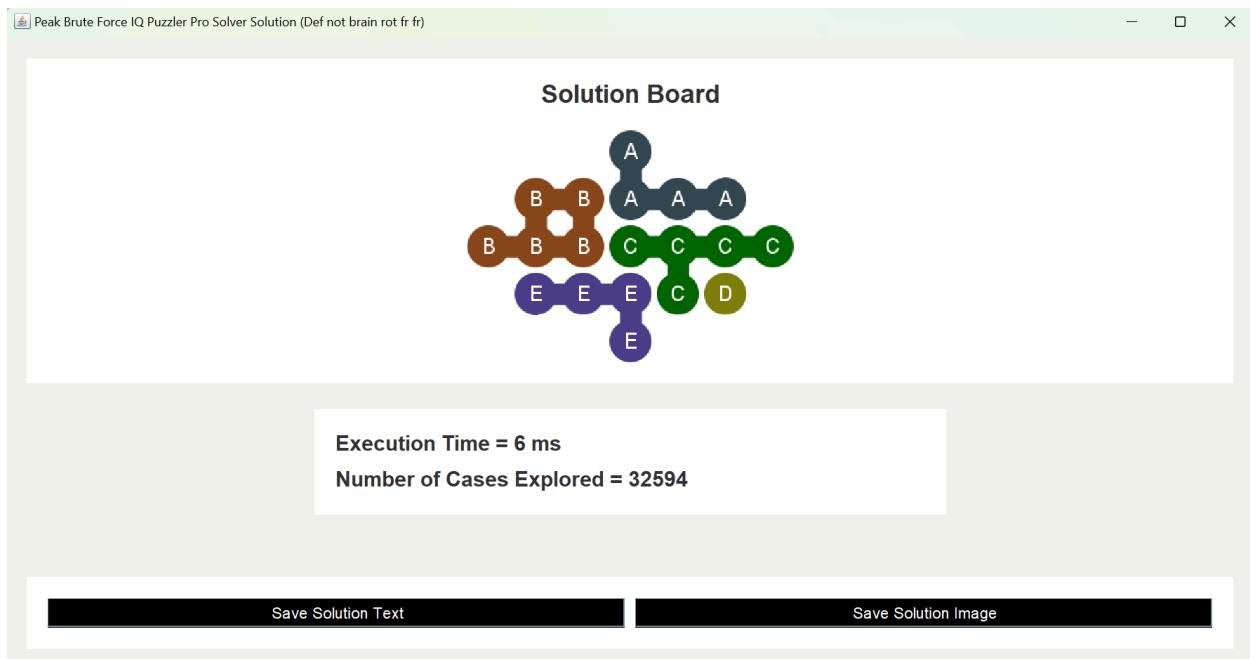
Test Case 8 (custom)

Input

```
test > test1custom.txt
1 5 7 5
2 CUSTOM
3 ...X...
4 .XXXXX.
5 XXXXXX
6 .XXXXX.
7 ...X...
8 A
9 AAA
10 BB
11 BBB
12 CCCC
13 | C
14 D
15 EEE
16 E
```



Output



```
● ● ● Tucil1_13523100 - solution1custom.txt

24 Solution:

25      A

26      B B A A A

27      B B B C C C C

28      E E E C D

29      E

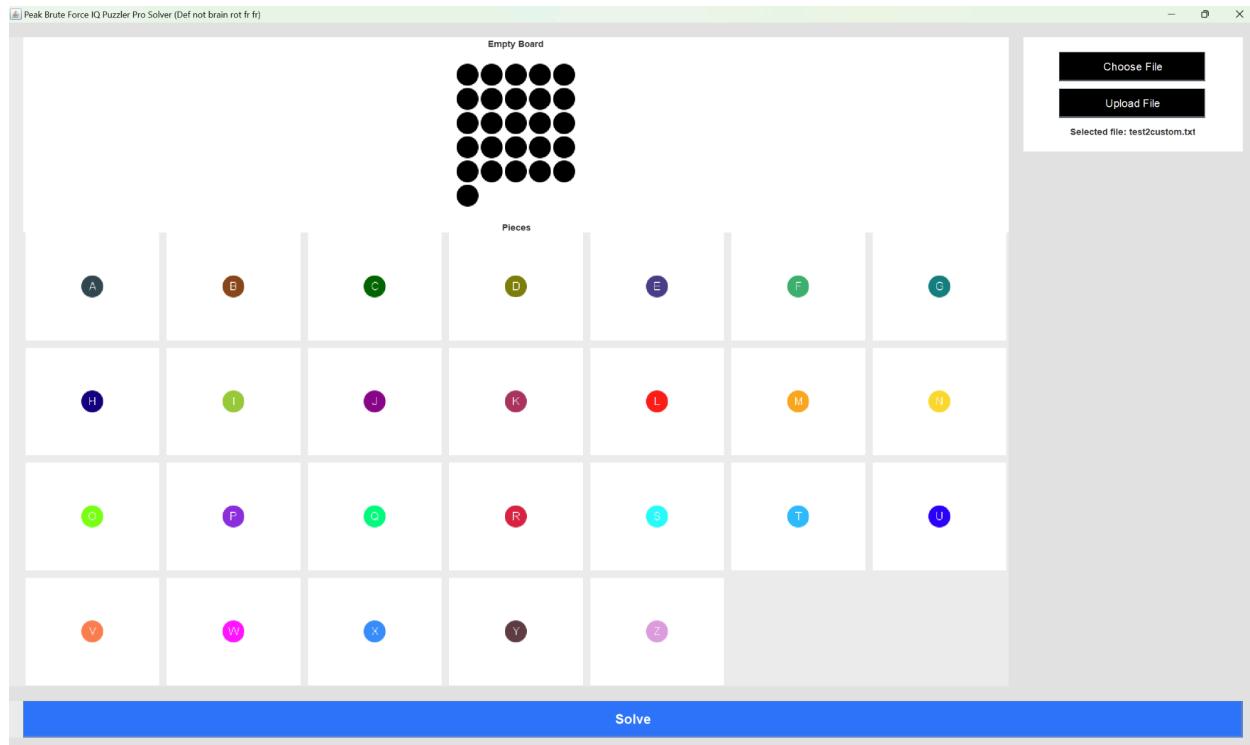
30

31 Execution Time: 6 ms

32 Number of cases checked: 32594
```

Test Case 9 (custom)

Input



```
test > test2custom.txt
```

```
1   6 5 26
2   CUSTOM
3   XXXXX
4   XXXXX
5   XXXXX
6   XXXXX
7   XXXXX
8   X....
9   A
10  B
11  C
12  D
13  E
14  F
15  G
16  H
17  I
18  J
19  K
20  L
21  M
22  N
23  O
24  P
25  Q
26  R
27  S
28  T
29  U
30  V
31  W
32  X
33  Y
34  Z
```

Output

Peak Brute Force IQ Puzzler Pro Solver Solution (Def not brain rot fr fr)

Solution Board

Execution Time = 1 ms
Number of Cases Explored = 326

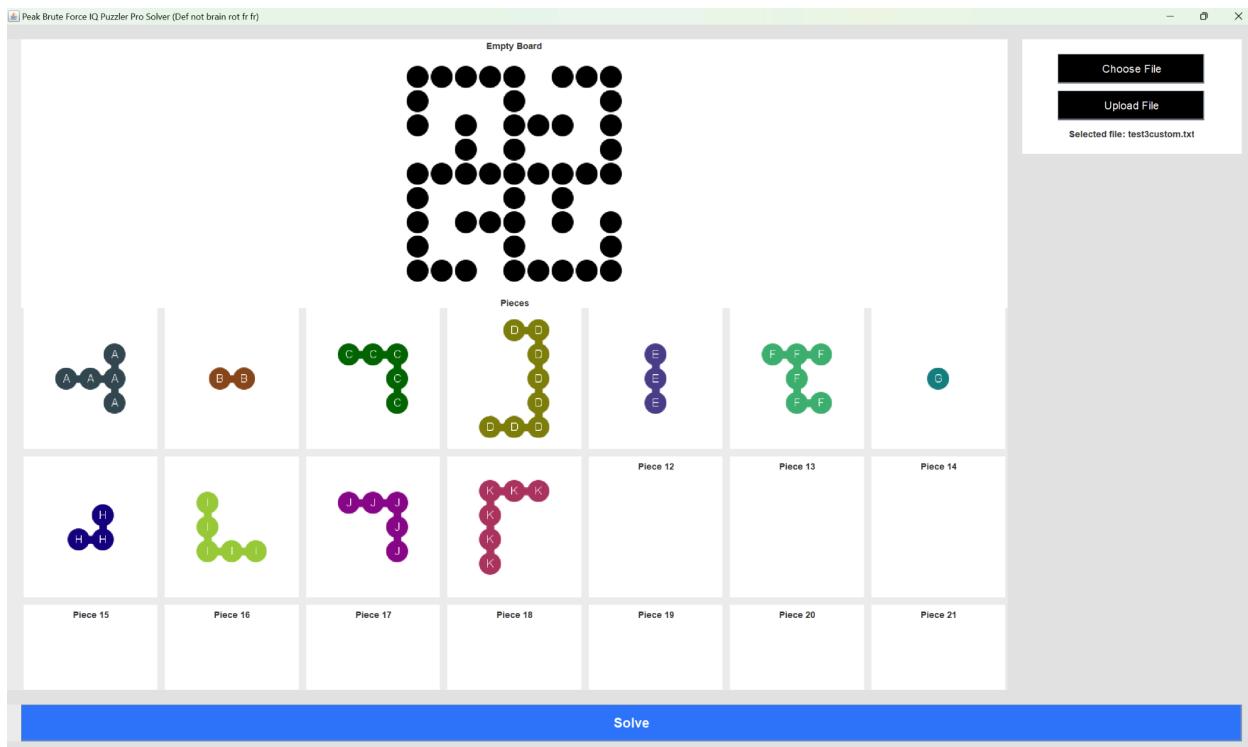
Save Solution Text Save Solution Image

```
63 Solution:  
64 A B C D E  
65 F G H I J  
66 K L M N O  
67 P Q R S T  
68 U V W X Y  
69 Z  
70  
71 Execution Time: 1 ms  
72 Number of cases checked: 326
```

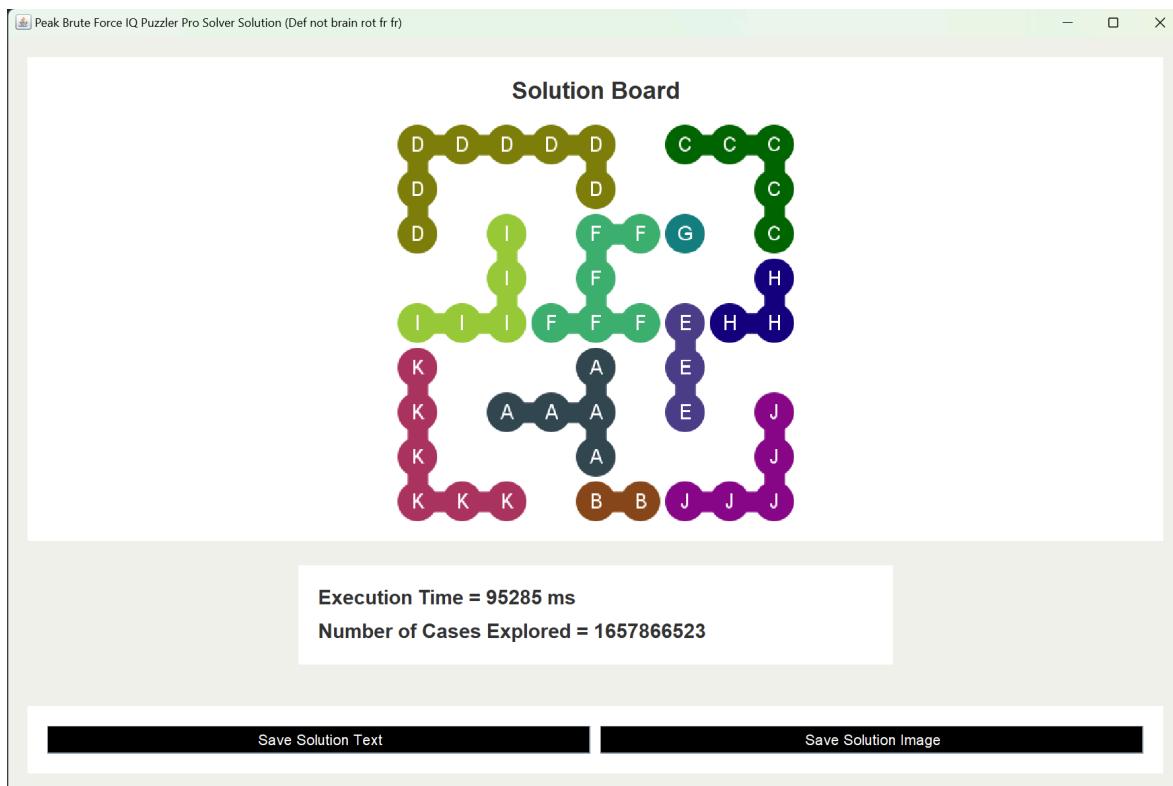
Test Case 10 (custom)

Input

```
test > test3custom.txt
1  9 9 11
2  CUSTOM
3  XXXXX.XXX
4  X...X...X
5  X.X.XXX.X
6  ..X.X...X
7  XXXXXXXXXX
8  X...X.X..
9  X.XXX.X.X
10 X...X...X
11 XXX.XXXXXX
12 | A
13 AAA
14 | A
15 BB
16 CCC
17 | C
18 | C
19 | DD
20 | D
21 | D
22 | D
23 DDD
24 E
25 E
26 E
27 FFFF
28 | F
29 | FF
30 | G
31 | H
32 HH
33 I
34 I
35 III
36 JJJ
37 | J
38 | J
39 KKK
40 K
41 K
42 K
```



Output



● ● ● Tucil1_13523100 - solution3custom.txt

```
56 Solution:  
57 D D D D D C C C  
58 D D C  
59 D I F F G C  
60 I F H  
61 I I I F F F E H H  
62 K A E  
63 K A A A E J  
64 K A J  
65 K K K B B J J J  
66  
67 Execution Time: 95285 ms  
68 Number of cases checked: 1657866523
```

4. Lampiran

Link Repository: https://github.com/Staryo40/Tucil1_13523100

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 <i>Graphical User Interface (GUI)</i>	✓	
6	Program dapat menyimpan solusi dalam bentuk file gambar	✓	
7	Program dapat menyelesaikan kasus konfigurasi <i>custom</i>	✓	
8	Program dapat menyelesaikan kasus konfigurasi Piramida (3D)		✓
9	Program dibuat oleh saya sendiri	✓	