Tugas Kecil 1 IF2211 Strategi Algoritma

Penyelesaian IQ Puzzler Pro dengan Algoritma Brute Force

Disusun Oleh:

Heleni Gratia Meitrina Tampubolon (13523107)

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2024

DAFTAR ISI

BAB I

DESKRIPSI MASALAH DAN ALGORITMA

1.1. Algoritma Brute Force

Algoritma brute force merupakan metode penyelesaian masalah yang menerapkan eksplorasi sistematis terhadap setiap kemungkinan solusi hingga solusi yang benar ditemukan. Pendekatan ini dikenal sebagai metode langsung (*straightforward*) karena proses pemecahannya bersifat eksplisit, sederhana, serta mudah dipahami. Namun, kelemahan utama dari algoritma brute force adalah kompleksitasnya yang tinggi, terutama untuk masalah berskala besar, karena setiap kemungkinan solusi diperiksa secara menyeluruh sehingga memerlukan waktu eksekusi yang panjang dan sumber daya komputasi yang besar.

Dalam penyusunan algoritma brute force, terdapat dua aspek utama yang perlu diperhatikan, yaitu perumusan permasalahan (*problem statement*) serta definisi atau konsep yang digunakan dalam penyelesaiannya. Salah satu contoh penerapan algoritma brute force adalah pada permasalahan pencarian pasangan titik dengan jarak terdekat (*closest pairs problem*). Dalam kasus ini, algoritma brute force menghitung jarak Euclidean antara setiap pasangan titik, menyimpan hasil perhitungan, dan memilih pasangan dengan jarak terpendek. Kompleksitas algoritma ini dalam notasi Big-O adalah $O(n^2)$, karena apabila terdapat *n* titik, maka setiap titik akan dibandingkan dengan n-1 titik lainnya, dan proses ini dilakukan untuk seluruh titik dalam himpunan tersebut.

Algoritma brute force memiliki beberapa karakteristik yang membedakannya dari metode lain. Secara umum, algoritma ini tidak bersifat efisien karena memerlukan biaya komputasi yang besar serta waktu eksekusi yang lama, sehingga sering disebut sebagai algoritma naif. Pendekatan brute force lebih sesuai digunakan untuk permasalahan dengan jumlah masukan kecil karena hanya mencari solusi berdasarkan pola yang lebih sederhana tanpa menerapkan teknik optimasi tambahan. Meskipun demikian, hampir semua permasalahan dapat diselesaikan menggunakan algoritma brute force, sehingga metode ini sering digunakan sebagai pendekatan dasar sebelum menerapkan algoritma yang lebih optimal.

Keunggulan utama dari algoritma brute force adalah kepastian dalam menemukan solusi yang tepat, karena seluruh kemungkinan solusi diperiksa tanpa ada yang terlewatkan. Selain itu, pendekatan ini memiliki struktur yang sederhana dan mudah dipahami, sehingga lebih mudah diimplementasikan dibandingkan dengan metode yang lebih kompleks. Algoritma brute force juga sering digunakan dalam berbagai tugas komputasi standar, seperti pencarian, pengurutan, pencocokan string, serta perkalian matriks. Beberapa contoh penerapan lainnya meliputi pencarian substring dalam suatu teks, pemecahan sandi dengan metode brute force, serta penyelesaian permasalahan kombinatorial seperti teka-teki Sudoku atau permainan catur.

Namun, di balik kesederhanaannya, algoritma brute force memiliki beberapa kelemahan yang signifikan. Salah satu kelemahan utamanya adalah efisiensi yang rendah, terutama untuk

permasalahan dengan ukuran masukan yang besar, sehingga dalam banyak kasus tidak dapat diterima sebagai solusi yang praktis. Selain itu, metode ini kurang konstruktif dan tidak sefleksibel strategi penyelesaian masalah lainnya yang menggunakan teknik optimasi, seperti pemrograman dinamis, algoritma *greedy*, atau *divide and conquer*. Oleh karena itu, meskipun algoritma brute force sering digunakan sebagai solusi awal dalam eksplorasi suatu permasalahan, umumnya diperlukan pendekatan yang lebih efisien untuk menangani kasus dengan skala yang lebih besar.

1.2. IQ Puzzler Pro

IQ Puzzler Pro merupakan permainan papan yang diproduksi oleh perusahaan Smart Games. Permainan ini dirancang untuk mengasah keterampilan berpikir logis dan pemecahan masalah dengan cara menyusun blok puzzle pada papan permainan hingga seluruh area papan terisi penuh tanpa ada ruang yang tersisa. Permainan ini dapat dimainkan dalam berbagai tingkat kesulitan, mulai dari pemula hingga tingkat lanjutan, yang menuntut pemain untuk menemukan kombinasi penempatan blok yang tepat.

Komponen utama dalam permainan IQ Puzzler Pro terdiri dari papan permainan (*board*) dan blok puzzle (*piece*). Papan permainan ber*method* sebagai area utama tempat pemain menyusun blok-blok puzzle. Pada beberapa varian permainan, papan ini memiliki grid dua dimensi atau bahkan mode tiga dimensi yang memungkinkan variasi tantangan yang lebih kompleks. Blok puzzle adalah elemen yang digunakan untuk mengisi papan hingga penuh. Setiap blok memiliki bentuk yang unik, dan semua blok harus digunakan agar puzzle dapat terselesaikan dengan benar. Blok-blok ini dapat diputar atau dibalik untuk menemukan posisi yang sesuai dengan bentuk papan permainan.

Permainan ini menuntut pemain untuk mempertimbangkan berbagai kemungkinan kombinasi penyusunan blok, menjadikannya tantangan yang sesuai untuk pengembangan keterampilan spasial, berpikir strategis, serta daya ingat. Kesulitan dalam permainan IQ Puzzler Pro terletak pada banyaknya kemungkinan penempatan blok yang harus dicoba sebelum menemukan solusi yang tepat. Oleh karena itu, metode penyelesaian seperti algoritma brute force dapat diterapkan dalam eksplorasi seluruh kemungkinan solusi secara sistematis. Dalam konteks penelitian ini, permainan IQ Puzzler Pro dapat dianalisis sebagai sebuah permasalahan komputasi yang memerlukan pendekatan sistematis untuk menemukan solusi optimal.

1.3. Algoritma Penyelesaian IQ Puzzler Pro dengan Algoritma Brute Force

Penyelesaian permasalahan permainan IQ Puzzler Pro dapat dilakukan menggunakan algoritma brute force. Berikut merupakan detail dari langkah-langkah penyelesaiannya.

```
procedure readInputFile(input filename: string)
{Membaca file masukan dan menyimpan data papan permainan serta potongan puzzle}

Deklarasi:
    br : BufferedReader
    st : StringTokenizer
```

```
      N, M, P : integer
      caseType : string
      board : array [1..N, 1..M] of char
      pieces : list of PuzzlePiece
      shape : list of string
      line : string
      currentPiece, firstLetter : char

  Algoritma:
      open file filename for reading
      read line from file → st
      N ← convert to integer(st.nextToken())
      M ← convert to integer(st.nextToken())
      P ← convert to integer(st.nextToken())
      caseType ← trim(next line) { membaca tipe kasus }

      initialize board[N][M] with '.'

      initialize pieces as an empty list
      initialize shape as an empty list

      read next line from file → line
      if line is null then return blank

      currentPiece ← findFirstLetter(line)

      while line is not null do
          firstLetter ← findFirstLetter(line) {mencari pieces (huruf) yang lain}

          if firstLetter ≠ currentPiece then
              if shape is not empty then
                       add new PuzzlePiece(shape, currentPiece) to pieces (simpan piece
puzzle)
              endif

              shape ← empty list
              currentPiece ← firstLetter
          endif

          add line to shape
          read next line from file → line
      endwhile

      if shape is not empty then
          add new PuzzlePiece(shape, currentPiece) to pieces
      endif

      close file

      if size of pieces ≠ P then
          call showDifferentTotalPiecesWarning()
      endif
  end procedure
```

1. Program ini menampilkan antarmuka grafis (GUI) yang terdiri dari empat button utama, yaitu Upload File (.txt), Result, Save as PNG, dan Save as TXT. Pada Upload File, pengguna diharapkan mengunggah file dalam format .txt yang berisi informasi terkait ukuran papan permainan (N × M), jumlah potongan puzzle (P), jenis kasus (S), serta konfigurasi bentuk blok puzzle yang dilambangkan dengan karakter huruf. Pembacaan file ini dilakukan melalui *method* `readInputFile`, yang menerima parameter berupa string yang merepresentasikan file path.

Nilai N, M, dan P akan disimpan sebagai tipe data integer, sedangkan S disimpan sebagai string. Sementara itu, bentuk blok puzzle akan disimpan dalam sebuah list bersarang. Setiap potongan puzzle yang ditandai dengan perbedaan huruf akan dikelompokkan dalam sebuah list menggunakan *method* `PuzzlePiece`. Data ini disusun berdasarkan isi setiap baris dari awal hingga akhir blok puzzle terkait, yang kemudian disimpan dalam bentuk matriks char[][] (pieces). Selain itu, dilakukan juga deklarasi board, yaitu sebuah matriks berukuran N × M dan berisi inisialisasi '.' yang akan digunakan sebagai tempat penyusunan puzzle selama proses penyelesaian.

Pada *method* `PuzzlePiece`, dilakukan penyesuaian terhadap isi setiap potongan puzzle. Jika terdapat spasi di awal atau jika lebar karakter dalam suatu baris lebih pendek dibandingkan dengan baris lain dalam blok puzzle yang sama, maka sistem akan menambahkan titik (.) hingga ukuran matriks sesuai dengan jumlah baris dikali jumlah karakter terbanyak dalam suatu baris. Penyesuaian ini bertujuan untuk mempermudah proses transformasi bentuk puzzle selama pencarian solusi.

```
  procedure Solve(input index: integer) → boolean
  {Menyusun puzzle dengan pendekatan rekursif menggunakan brute-force}

  Deklarasi:
      piece, transformed : PuzzlePiece
      transformations : list of PuzzlePiece
      i, j : integer

  Algoritma:
      if isBoardFullyFilled() then
          if index < size of pieces then
                   call showExcessPiecesWarning() {Menampilkan pesan kesalahan jika ada
potongan berlebih}
          endif
          return true
      endif

      if index = size of pieces then
          return false
      endif

      piece ← pieces[index]
      transformations ← getAllTransformations(piece)

      for each transformed in transformations do
          for i ← 0 to N-1 do
              for j ← 0 to M-1 do
                  if canPlacePiece(transformed, i, j) then
                      placePiece(transformed, i, j)
                      if Solve(index + 1) then
                          return true
                      endif
                      removePiece(transformed, i, j)
                           iterationCount ← iterationCount + 1 {Kombinasi semua potongan
gagal}
                  endif
              endfor
          endfor
      endfor

      return false
end procedure
```

2. Setelah seluruh input berhasil dibaca, dilanjutkan dengan menekan button solve maka algoritma brute force akan mengevaluasi semua kemungkinan kombinasi pieces menggunakan *method* `solve`. Pencarian solusi dimulai dari piece pertama yang disimpan (diakses melalui indeks). Pada tahap awal, dilakukan pengecekan apakah board sudah penuh. Jika board sudah terisi sepenuhnya, proses pencarian dihentikan karena solusi telah ditemukan. Selain itu, sistem juga menangani kasus di mana board sudah penuh, tetapi masih terdapat pieces yang tersisa. Jika kondisi ini terjadi, program akan menganggap bahwa input tidak valid dan tidak ada solusi yang ditemukan.

   Pencarian solusi dilakukan melalui tiga perulangan bersarang:

   a. Perulangan pertama: Mengevaluasi setiap bentuk transformasi dari piece (rotasi dan refleksi).
   b. Perulangan kedua: Melakukan iterasi dari baris pertama hingga terakhir pada board.
   c. Perulangan ketiga: Melakukan iterasi pada setiap kolom dalam board.

   Perulangan kedua dan ketiga bertujuan untuk memastikan setiap koordinat dalam board diakses. Pada setiap iterasi, dilakukan pengecekan menggunakan *method* `canPlacePiece` untuk menentukan apakah piece dengan transformasi tertentu dapat ditempatkan di koordinat tersebut. Jika posisi memungkinkan (tidak tumpang tindih dan hanya mengisi area kosong, ditandai dengan karakter '.'), maka piece akan diletakkan di board dan proses dilanjutkan dengan pengecekan untuk piece berikutnya menggunakan rekursi.

   Jika sebuah piece tidak dapat ditempatkan atau posisinya tidak sesuai, maka piece tersebut akan dihapus dari board, dan sistem akan mencoba bentuk transformasi lain dari piece yang sama (*backtracking*). Setiap kali terjadi kegagalan dalam kombinasi piece, jumlah iterasi akan dihitung sebagai bagian dari total jumlah kasus yang ditinjau.

   Jika seluruh board berhasil diisi dengan pieces, *method* `solve` akan mengembalikan nilai true, menandakan bahwa solusi telah ditemukan. Solusi ini kemudian akan ditampilkan pada GUI. Selain itu, meskipun solusi telah ditemukan, jumlah iterasi tetap akan ditambah satu karena solusi tersebut juga dihitung sebagai salah satu kasus yang dievaluasi.

3. Jika ditemukan solusi yang memungkinkan, GUI akan menampilkan board yang telah terisi dengan kombinasi pieces yang tepat. Setiap piece akan ditampilkan menggunakan huruf yang berbeda, dengan warna unik untuk memudahkan identifikasi.
   Selain itu, status label di bagian bawah GUI akan menampilkan informasi sebagai berikut:

a. Jumlah iterasi yang dilakukan selama pencarian solusi.

b. Waktu eksekusi yang dibutuhkan untuk menemukan solusi.

Jika setelah mengevaluasi seluruh kemungkinan kombinasi tidak ditemukan solusi yang valid, status label akan menampilkan pemberitahuan bahwa tidak ada solusi yang ditemukan.

4. Pengguna dapat menyimpan hasil kombinasi board menggunakan tombol "Save as PNG" atau "Save as TXT", sesuai dengan format yang diinginkan:

   a. Jika memilih "Save as PNG", maka *method* saveToImage akan dijalankan. Hasilnya akan disimpan dalam format gambar (.png), di mana representasi huruf pada board akan ditampilkan sebagai bulatan, menyerupai tampilan dalam game IQ Puzzler Pro.

   b. Jika memilih "Save as TXT", maka *method* saveToTextFile akan dijalankan. Hasilnya akan disimpan dalam format teks biasa (.txt), menampilkan kombinasi huruf sesuai dengan board tanpa modifikasi visual tambahan.

BAB II

IMPLEMENTASI ALGORITMA DALAM BAHASA JAVA

2.1. File GUI.java

File ini berisi class GUI yang terdiri dari beberapa fungsi (*method*) untuk mengatur tampilan GUI.

Nama Method	Deskripsi
main()	Parameter: String[] args Method utama yang akan dijalankan untuk menampilkan antarmuka grafis (GUI).
GUI()	Parameter: - Method untuk mmebuat dan menampilkan antarmuka (GUI) menggunakan package Java Swing. Terdiri dari 3 komponen utama: -Button, terletak di bagian atas dan terdiri dari 4 button (Upload File (.txt), Result, Save as PNG, dan Save as TXT) -Board, di bagian tengah yang akan menampilkan solusi puzzle dalam bentuk grid dan warna yang berbeda untuk tiap piecesnya.

	-Label Status, di bagian bawah untuk memberikan informasi tentang status proses, jumlah iterasi, dan waktu yang dibutuhkan.
updateBoardDisplay()	Parameter: - Method untuk memperbarui tampilan board dengan hasil solusi puzzle. Dilakukan dengan menghapus tampilan board sebelumnya kemudian menampilkan karakter pada setiap sel di grid board beserta warna acak yang dimiliki tiap pieces.

2.2. File PuzzleSolver.java

File ini berisi class PuzzleSolver yang terdiri dari beberapa fungsi (method) yang menerapkan algoritma brute force dalam pencarian solusi pada permasalahan yang diberikan.

Nama Method	Deskripsi
PuzzleSolver()	Parameter: String filename Method untuk membaca nama file input.
readInputFile()	Parameter: String filename Method untuk memproses file input dan menyimpan informasi, berupa integer (N M P), string (Tipe Board), dan susunan puzzle. Susunan puzzle akan disimpan dalam sebuah list (shape) dengan tiap isi listnya berisi matriks yang diproses dengan method PuzzlePiece. Penambahan pieces akan dilakukan dengan mengecek perbedaan karakter huruf (dikelompokkan). Susunan board juga diinisialisasi dengan '.' Sesuai dengan ukuran board yang diminta pengguna.
findFirstLetter()	Parameter: String line Method untuk mengembalikan informasi karakter yng merupakan huruf pertama pada suatu line.
solve()	Parameter: int index

	Method untuk mengembalikan nilai boolean pengecekan apakah solusi ditemukan dengan melakukan algoritma brute force.
isBoardFullyFilled()	Parameter: - Method untuk mengembalikan true jika seluruh board sudah terisi dengan potongan puzzle (ditemukan solusi).
canPlacePiece()	Parameter: PuzzlePiece piece, int row, int col Method untuk mengecek apakah suatu piece dapat ditempatkan pada posisi board yang diinginkan (row x col). Piece dapat ditempatkan jika board masih berisi '.' pada posisi tersebut.
placePiece()	Parameter: PuzzlePiece piece, int row, int col Method untuk memperbarui board bila piece dapat diletakkan pada posisi yang diinginkan. Posisi tersebut akan diisi sesuai dengan bentuk piece (diisi dengan huruf).
removePiece()	Parameter: PuzzlePiece piece, int row, int col Method untuk menghapus suatu piece dari board karena kombinasi tersebut tidak memberikan solusi yang tepat.
showDifferentTotalPiecesWarning()	Parameter: - Method untuk menampilkan pop up bila jumlah pieces (P) tidak sesuai dengan potongan puzzle yang ada.
solvePuzzle()	Parameter: - Method untuk mengembalikan nilai Boolean sesuai solusi ditemukan atau tidak. Dilakukan pengecekan secara terurut (dimulai dari index/puzzle pertama) dengan memanggil method solve.
getIterationCount()	Parameter: - Method untuk mengembalikan nilai pada variable iterationCount untuk digunakan pada class GUI.
getBoard()	Parameter: -

	Method untuk mengembalikan nilai variable board untuk digunakan pada class GUI.
saveToImage()	Parameter: String filePath Method untuk menyimpan informasi board (solusi) dengan tampilan image (.png). Tampilan tersebut diatur agar berbentuk grid yang didalamnnya piece akan diwakili dengan bulatan bewarna yang tiap piecesnya berbeda satu sama lain.
saveToTextFile	ParameterL String fileParh Method untuk menyimpan informasi board (solusi) dengan tampilan text (.txt). Isi file tersebut berupa susunan pieces yang direpresentasikan dengan huruf yang bersesuaian.
Jika ingin menampilkan pada terminal:	
printBoard()	Parameter: - Method untuk menampilaj isi board pada terminal dengan susunan pieces (huruf) yang memiliki warna acak (berbeda satu sama lain). Pada method ini juga ditampikan opsi untuk menyimpan gambar (.png) atau tidak.

2.3. File PuzzlePiece.java

File ini berisi class PuzzlePiece yang terdiri dari beberapa fungsi (*method*) yang menerapkan algoritma untuk proses pengaturan penyimpanan pieces dan transformasinya.

Nama Method	Deskripsi
PuzzlePiece()	Parameter: List<String> shapeLines, char label Method untuk menyimpan pieces dalam matriks yang berukuran sesuai dengan banyaknya line dan maksimal karakter dari suatu line. Jika pada suatu line memiliki jumlah karakter yang lebih sedikit, maka akan ditambahkan '.'. Hal ini untuk memudahkan proses tranformasi.
getAllTransformation()	Parameter: -

	Method untuk menyimpan ke dalam list seluruh tranformasi, berupa rotasi (perulangan 4 kali, 90°, 180°, 270°, 360°, dan refleksi kemudian diulang dirotasikan yang terjadi pada tiap piece.
rotate()	Parameter: char[][] firstform Method untuk mengembalikan bentuk rotasi 90° pada piece yang dituju.
toString()	Parameter: char[][] arr Method untuk mengubah shape menjadi string agar dapat dilakukan pengecekan apakah bentuk tersebut unik atau tidak.
reflect()	Parameter: char[][] firstform Method untuk mengembalikan bentuk refleksi pada piece yang dituju.
convertToList()	Parameter: char[][] shapeArray Method untuk mengembalikan bentuk list setelah dicek unik atau tidak agar dapat diproses untuk dilakukan transformasi.
equals()	Parameter: obj Method untuk mengembalikan nilai boolean apakah piece yang dituju sama dengan piece lainnya (shapenya sama).
hashCode()	Parameter: - Method untuk menghasilkan nilai hash berdasarkan bentuk shape.

2.4. File Main.java (Opsional)

File ini berisi class Main yang terdiri dari method utama jika ingin menjalankan program pada CLI (Command Line Interface/Terminal).

Nama Method	Deskripsi
main()	Parameter: String[] args Method utama yang akan dijalankan untuk menjalankan program pada CLI.

BAB III

SOURCE CODE PROGRAM

3.1 Repositori Github

Repositori program dapat diakses melalui tautan GitHub berikut: https://github.com/mineraleee/Tucil1_13523107

3.2. Source Code Program

3.2.1. GUI.java

```java
import javax.swing.*;
import java.awt.*;
import java.io.*;
import java.util.Map;
import java.util.HashMap;
import java.util.Random;

public class GUI {
    private JFrame frame;
    private JLabel statusLabel;
    private File selectedFile;
    private String filePath = "";
    private boolean result = false;
    private JPanel boardPanel;
    private PuzzleSolver solver;

    public GUI() {
        frame = new JFrame(title:"IQ Puzzler Pro Solver!");
        frame.setSize(width:700,height:600);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLayout(new BorderLayout());

        JPanel buttonPanel = new JPanel();
        buttonPanel.setLayout(new FlowLayout(FlowLayout.CENTER, hgap:15, vgap:10));

        JButton loadButton = new JButton(text:"Load File Puzzle (.txt)");
        JButton solveButton = new JButton(text:"Solve");
        JButton savePNGButton = new JButton(text:"Save as PNG");
        JButton saveTXTButton = new JButton(text:"Save as TXT");
        statusLabel = new JLabel(text:"Select a puzzle file.", SwingConstants.CENTER);
        statusLabel.setFont(new Font(name:"Arial", Font.PLAIN, size:14));

        boardPanel = new JPanel();
        boardPanel.setLayout(new GridLayout(rows:1, cols:1)); // Default -> blank
        boardPanel.setBorder(BorderFactory.createTitledBorder(title:"Solved Puzzle"));
```

```
        // Select File Button
        loadButton.addActionListener(e -> {    The value of the lambda parameter e is not used
            JFileChooser fileChooser = new JFileChooser();
            int returnValue = fileChooser.showOpenDialog(frame);
            if (returnValue == JFileChooser.APPROVE_OPTION) {
                selectedFile = fileChooser.getSelectedFile();
                filePath = selectedFile.getAbsolutePath();
                statusLabel.setText("Selected file: " + selectedFile.getName());
            }
        });

        // Solve Button
        solveButton.addActionListener(e -> {    The value of the lambda parameter e is not used
            if (selectedFile == null) {
                statusLabel.setText(text:"Please load a puzzle file first!");
                return;
            }

            statusLabel.setText(text:"Solving...");

            try {
                long startTime = System.currentTimeMillis();
                solver = new PuzzleSolver(filePath); // use for save
                result = solver.solvePuzzle(); // solve the puzzle
                if (result) {
                    int iterationCount = solver.getIterationCount();
                    SwingUtilities.invokeLater(this::updateBoardDisplay); //show board
                    long endTime = System.currentTimeMillis();
                    long duration = endTime-startTime;
                    statusLabel.setText("<html>Solved in " + iterationCount + " iterations!<br>Time: " + duration +"ms </html>");
                } else {
                    int iterationCount = solver.getIterationCount();
                    long endTime = System.currentTimeMillis();
                    long duration = endTime-startTime;
                    statusLabel.setText("<html>No solution found in " + iterationCount + " iterations!<br>Time: " + duration +"ms </html>");
                }
            } catch (IOException ex) {
                statusLabel.setText("Error: " + ex.getMessage());
                ex.printStackTrace();
            }
```

```
        // Save as PNG Button
        savePNGButton.addActionListener(e -> {    The value of the lambda parameter e is not used
            if (!result || solver == null) {
                JOptionPane.showMessageDialog(frame, message:"No solution available to save!",
                    title:"Error", JOptionPane.ERROR_MESSAGE);
                return;
            }

            JFileChooser fileChooser = new JFileChooser();
            fileChooser.setDialogTitle(dialogTitle:"Save as PNG");

            int userSelection = fileChooser.showSaveDialog(frame);
            if (userSelection == JFileChooser.APPROVE_OPTION) {
                File fileToSave = fileChooser.getSelectedFile();
                String savePath = fileToSave.getAbsolutePath();

                if (!savePath.toLowerCase().endsWith(suffix:".png")) {
                    savePath += ".png";
                }

                solver.saveToImage(savePath);
                statusLabel.setText("Image saved: " + new File(savePath).getName());
            }
        });

        // Save as TXT Button
        saveTXTButton.addActionListener(e -> {    The value of the lambda parameter e is not used
            if (!result || solver == null) {
                JOptionPane.showMessageDialog(frame, message:"No solution available to save!",
                    title:"Error", JOptionPane.ERROR_MESSAGE);
                return;
            }

            JFileChooser fileChooser = new JFileChooser();
            fileChooser.setDialogTitle(dialogTitle:"Save as TXT");

            int userSelection = fileChooser.showSaveDialog(frame);
            if (userSelection == JFileChooser.APPROVE_OPTION) {
                File fileToSave = fileChooser.getSelectedFile();
                String savePath = fileToSave.getAbsolutePath();
```

```
120                 if (!savePath.toLowerCase().endsWith(suffix:".txt")) {
121                     savePath += ".txt";
122                 }
123 
124                 solver.saveToTextFile(savePath);
125                 statusLabel.setText("File saved: " + new File(savePath).getName());
126             }
127         });
128 
129         buttonPanel.add(loadButton);
130         buttonPanel.add(solveButton);
131         buttonPanel.add(savePNGButton);
132         buttonPanel.add(saveTXTButton);
133 
134         frame.add(buttonPanel, BorderLayout.NORTH);
135         frame.add(boardPanel, BorderLayout.CENTER);
136         frame.add(statusLabel, BorderLayout.SOUTH);
137         frame.setVisible(b:true);
138     }
139 
140     private void updateBoardDisplay() {
141         boardPanel.removeAll();
142         char[][] board = solver.getBoard(); // get the board from solver code
143 
144         int rows = board.length;
145         int cols = board[0].length;
146         boardPanel.setLayout(new GridLayout(rows, cols));
147 
148         Map<Character, Color> colorMap = new HashMap<>();
149         Random random = new Random();
150 
151         // variant color
152         for (char[] row : board) {
153             for (char c : row) {
154                 if (c != '.' && !colorMap.containsKey(c)) {
155                     colorMap.put(c, new Color(random.nextInt(bound:256), random.nextInt(bound:256), random.nextInt(bound:256))); // 3 -> RGB
156                 }
157             }
158         }
```

```
160         // Make the grid
161         for (int i = 0; i < rows; i++) {
162             for (int j = 0; j < cols; j++) {
163                 char c = board[i][j];
164                 JLabel cellLabel = new JLabel(String.valueOf(c), SwingConstants.CENTER);
165                 cellLabel.setOpaque(isOpaque:true);
166                 cellLabel.setBorder(BorderFactory.createLineBorder(Color.BLACK));
167 
168                 if (c=='.') {
169                     cellLabel.setBackground(Color.WHITE);
170                 } else {
171                     cellLabel.setBackground(colorMap.get(c));
172                 }
173                 boardPanel.add(cellLabel);
174             }
175         }
176         boardPanel.revalidate();
177         boardPanel.repaint();
178     }
179 
        Run | Debug
180     public static void main(String[] args) {
181         SwingUtilities.invokeLater(GUI::new);
182     }
183 }
184 
```

3.2.2. PuzzleSolver.java

```java
import java.util.List;
import java.util.ArrayList;
import java.util.Random;
import java.util.Scanner;
import java.util.StringTokenizer;
import java.util.Arrays;
import java.util.Map;
import java.util.HashMap;
import java.io.*;
import javax.imageio.ImageIO;
import javax.swing.JOptionPane;
import java.awt.*;
import java.awt.image.BufferedImage;

public class PuzzleSolver {
    private int N, M, P;
    private char[][] board;
    private List<PuzzlePiece> pieces;
    private int iterationCount = 0;
    private String caseType;    The value of the field PuzzleSolver.caseType is not used
    private Scanner scanner;    The value of the field PuzzleSolver.scanner is not used

    public PuzzleSolver(String filename) throws IOException {
        scanner = new Scanner(System.in);
        readInputFile(filename);
    }

    private void readInputFile(String filename) throws IOException {
        BufferedReader br = new BufferedReader(new FileReader(filename));
        StringTokenizer st = new StringTokenizer(br.readLine());
        N = Integer.parseInt(st.nextToken());
        M = Integer.parseInt(st.nextToken());
        P = Integer.parseInt(st.nextToken());
        caseType = br.readLine().trim(); //DEFAULT

        board = new char[N][M];
        for (char[] row : board) Arrays.fill(row, val:'.');

        pieces = new ArrayList<>();
        List<String> shape = new ArrayList<>();
```

```java
        String line = br.readLine();
        if (line== null) return;  // exit if the file blank    Resource leak: 'br' is not closed at this location

        char currentPiece = findFirstLetter(line); // take the first char in line

        while (line != null) {
            char firstLetter = findFirstLetter(line);

            if (firstLetter != currentPiece) {
                // save the previous line
                if (!shape.isEmpty()) {
                    pieces.add(new PuzzlePiece(shape, currentPiece));
                }

                shape = new ArrayList<>();
                currentPiece = firstLetter; // update the new char
            }

            shape.add(line); // newline
            line = br.readLine();
        }

        // Last piece
        if (!shape.isEmpty()) {
            pieces.add(new PuzzlePiece(shape, currentPiece));
        }
        br.close();
        if (pieces.size() != P){
            showDifferentTotalPiecesWarning();
        }
    }

    private char findFirstLetter(String line) {
        for (char c : line.toCharArray()) {
            if (Character.isLetter(c)) {
                return c; // return the first letter
            }
        }
        return '.'; // Default
    }
```

```java
    private boolean solve(int index) {
        if (isBoardFullyFilled()) {
            // still there's other pieces but the board already fullfill
            if (index < pieces.size()) {
                //showExcessPiecesWarning(); // show the pop up cause there's extra pieces
                return false;
            }
            return true;
        }

        if (index == pieces.size()) return false;

        PuzzlePiece piece = pieces.get(index);
        List<PuzzlePiece> transformations = piece.getAllTransformations();

        for (PuzzlePiece transformed : transformations) {
            for (int i = 0;i < N;i++) {
                for (int j = 0; j < M; j++) {
                    if (canPlacePiece(transformed,i,j)) {
                        placePiece(transformed,i,j);
                        if (solve(index + 1))return true;
                        removePiece(transformed,i,j);
                        iterationCount++; //the combination all of pieces fail
                    }
                }
            }
        }
        return false;
    }

    private boolean isBoardFullyFilled() {
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < M; j++) {
                if (board[i][j] == '.') {
                    return false;
                }
            }
        }
        return true; //board is completely filled
    }
```

```java
    private boolean canPlacePiece(PuzzlePiece piece, int row, int col) {
        for (int i = 0; i < piece.shape.length; i++) {
            for (int j = 0; j < piece.shape[i].length; j++) {
                if (piece.shape[i][j] != '.') {
                    int r = row + i, c = col + j;
                    if (r >= N || c >= M || board[r][c] != '.') return false;
                }
            }
        }
        return true;
    }

    private void placePiece(PuzzlePiece piece, int row, int col) {
        for (int i = 0; i < piece.shape.length; i++) {
            for (int j = 0; j < piece.shape[i].length; j++) {
                if (piece.shape[i][j] != '.') {
                    board[row + i][col + j] = piece.label;
                }
            }
        }
    }

    private void removePiece(PuzzlePiece piece, int row, int col) {
        for (int i = 0; i < piece.shape.length; i++) {
            for (int j = 0; j < piece.shape[i].length; j++) {
                if (piece.shape[i][j] != '.') {
                    board[row + i][col + j] = '.';
                }
            }
        }
    }

    // private void showExcessPiecesWarning() {
    //     JOptionPane.showMessageDialog(null,"There are extra puzzle pieces!","Warning!!!",JOptionPane.WARNING_MESSAGE);
    // }

    private void showDifferentTotalPiecesWarning() {
        JOptionPane.showMessageDialog(parentComponent:null,message:"The number of pieces doesn't match!",title:"Warning!!!",JOptionPane.WARNING_MESSAGE);
    }
```

```java
    public boolean solvePuzzle() {
        if (solve(index:0)) {
            iterationCount++; // add the right combination
            //printBoard(); //uncomment if wanna run in terminal
            //System.out.println("Iterations: " + iterationCount);
            return true;
        } return false;
    }

    //uncomment if u wanna run in terminal
    // private void printBoard() {
    //     Map<Character, String> colorMap = new HashMap<>();
    //     Random random = new Random();

    //     for (char[] row : board) {
    //         for (char c : row) {
    //             if (c != '.' && !colorMap.containsKey(c)) {
    //                 int colorCode = 16 + random.nextInt(240);
    //                 colorMap.put(c, "\u001B[38;5;" + colorCode + "m");
    //             }
    //         }
    //     }

    //     for (char[] row : board) {
    //         for (char c : row) {
    //             if (c == '.') {
    //                 System.out.print("\u001B[37m" + c + " ");
    //             } else {
    //                 System.out.print(colorMap.get(c) + c + " ");
    //             }
    //         }
    //         System.out.println("\u001B[0m");
    //     }

    //     System.out.print("Do you want to save the solution as an image? (yes/no): ");
    //     String response = scanner.nextLine().trim().toLowerCase();
    //     System.out.print("Input the filename: ");
    //     String filename = scanner.nextLine().trim().toLowerCase();
    //     String name = "../test/"+ filename + ".png";
```

```java
    //     if (response.equals("yes") || response.equals("y")) {
    //         saveToImage(name);
    //     } else if (response.equals("no") || response.equals("n")) {
    //         System.out.println("Solution not saved.");
    //     } else {
    //         System.out.println("Invalid input. Please enter yes or no.");
    //     }
    // }

    public int getIterationCount(){
        return iterationCount;
    }

    public char[][] getBoard() {
        return board;
    }

    //the form will be rounded/circle
    public void saveToImage(String filePath) {
        int cellSize = 70;
        int width = M * cellSize;
        int height = N * cellSize;
        BufferedImage image = new BufferedImage(width, height, BufferedImage.TYPE_INT_RGB);
        Graphics2D g = image.createGraphics();

        //background
        g.setColor(Color.WHITE);
        g.fillRect(x:0, y:0, width, height);

        // mapping unique color
        Map<Character, Color> colorMap = new HashMap<>();
        Random random = new Random();

        for (char[] row : board) {
            for (char c : row) {
                if (c != '.' && !colorMap.containsKey(c)) {
                    colorMap.put(c, new Color(random.nextInt(bound:256), random.nextInt(bound:256), random.nextInt(bound:256)));
                }
            }
        }
```

```
        // grid
        int padding =5;
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < M; j++) {
                char c = board[i][j];
                if (c != '.') {
                    g.setColor(colorMap.get(c));
                    g.fillOval(j * cellSize + padding / 2, i * cellSize + padding / 2, cellSize - padding, cellSize - padding);
                }
                g.setColor(Color.BLACK);
                g.drawRect(j * cellSize, i * cellSize, cellSize, cellSize);
            }
        }
        g.dispose();

        // save as png
        try {
            File outputFile = new File(filePath);
            ImageIO.write(image, formatName:"png", outputFile);
            System.out.println("Image saved to: " +filePath);
        } catch (IOException e) {
            e.printStackTrace();
            System.err.println("Error saving image: " +e.getMessage());
        }
    }

    //save as txt
    public void saveToTextFile(String filePath) {
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(filePath))) {
            for (char[] row:board) {
                writer.write(new String(row)); //convert char[] to String
                writer.newLine(); //newline
            }
            System.out.println("Board saved to: " +filePath);
        } catch (IOException e) {
            e.printStackTrace();
            System.err.println("Error saving board: " +e.getMessage());
        }
    }
}
```

3.2.3. PuzzlePiece.java

```
import java.util.*;

public class PuzzlePiece {
    char[][] shape;
    char label;

    //save the pieces
    public PuzzlePiece(List<String> shapeLines, char label) {
        this.label = label;

        // max width
        int maxChars = 0;
        for (String line : shapeLines) {
            maxChars = Math.max(maxChars, line.length());
        }

        this.shape = new char[shapeLines.size()][maxChars];

        for (int i = 0;i<shapeLines.size();i++) {
            String line = shapeLines.get(i);
            for (int j = 0; j < maxChars; j++) {
                if (j < line.length()) {
                    char currentChar = line.charAt(j);
                    this.shape[i][j] = (currentChar == ' ')?'.' : currentChar;
                } else {
                    this.shape[i][j] = '.'; //will fill remaining space with dots --> matriks (2D)
                }
            }
        }
    }

    // generate all possible transformations (rotations and reflections)
    public List<PuzzlePiece> getAllTransformations() {
        Set<String> uniqueShapes = new HashSet<>();
        List<PuzzlePiece> transformations = new ArrayList<>();
        char[][] currentShape = this.shape;

        for (int i = 0;i < 4;i++) { // rotate 4 times -> 90,180,270,360
            String shapeStr = toString(currentShape);
            if (uniqueShapes.add(shapeStr)) {
                transformations.add(new PuzzlePiece(convertToList(currentShape), label));
            }
```

```
            char[][] reflected = reflect(currentShape);
            shapeStr = toString(reflected);
            if (uniqueShapes.add(shapeStr)) {
                transformations.add(new PuzzlePiece(convertToList(reflected), label));
            }

            currentShape = rotate(currentShape); //rotate for next iteration
        }

        return transformations;
    }

    //convert char[][] to String for uniqueness check
    private String toString(char[][] arr) {
        StringBuilder sb = new StringBuilder();
        for (char[] row : arr) {
            sb.append(row).append(str:"\n");
        }
        return sb.toString();
    }

    //rotate the shape 90° clockwise
    private char[][] rotate(char[][] firstform) {
        int rows = firstform.length;
        int cols = firstform[0].length;
        char[][] rotated = new char[cols][rows];

        for (int i = 0;i < rows;i++) {
            for (int j = 0;j < cols;j++) {
                rotated[j][rows - i - 1] = firstform[i][j];
            }
        }
        return rotated;
    }

    //reflect the shape horizontally
    private char[][] reflect(char[][] firstform) {
        int rows = firstform.length;
        int cols = firstform[0].length;
        char[][] flipped = new char[rows][cols];
```

```
        for (int i = 0; i < rows; i++) {
            for (int j = 0;j < cols;j++) {
                flipped[i][cols - j - 1] = firstform[i][j];
            }
        }
        return flipped;
    }

    //cnvert char[][] to List<String> (storing transformations)
    private List<String> convertToList(char[][] shapeArray) {
        List<String> result = new ArrayList<>();
        for (char[] row : shapeArray) {
            result.add(new String(row));
        }
        return result;
    }

    //make sure there's no duplicate form
    @Override
    public boolean equals(Object obj) {
        if (this == obj) return true;
        if (obj == null||getClass()!=obj.getClass()) return false;
        PuzzlePiece other = (PuzzlePiece) obj;
        return Arrays.deepEquals(this.shape, other.shape);
    }

    @Override
    public int hashCode() {
        return Arrays.deepHashCode(shape);
    }
}
```

3.2.4. Main.java (opsional, jika ingin menampilkan di CLI)

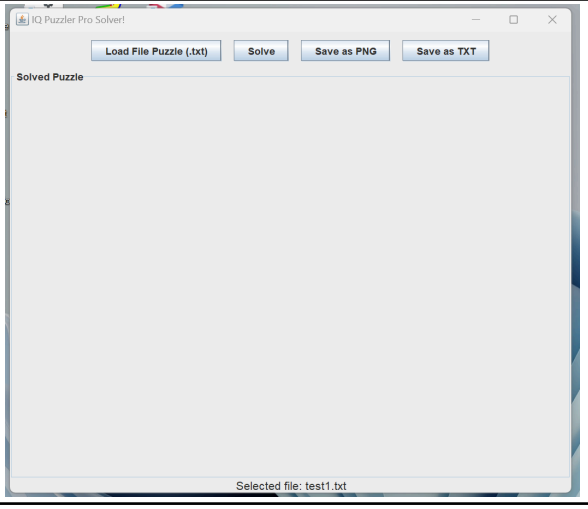
```java
import java.util.Scanner;
import java.io.IOException;

public class Main {
    Run | Debug
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print(s:"Enter test case file path: ");
        String filename = scanner.nextLine();
        long startTime = System.currentTimeMillis();
        String filePath = "../test/" + filename;

        try {
            PuzzleSolver solver = new PuzzleSolver(filePath);
            solver.solvePuzzle();
        } catch (IOException e) {
            System.out.println("Error reading file: " + e.getMessage());
        }
        scanner.close();
        long endTime = System.currentTimeMillis();
        System.out.println("Execution time: " + (endTime-startTime) + " ms");
    }
}
```

BAB IV

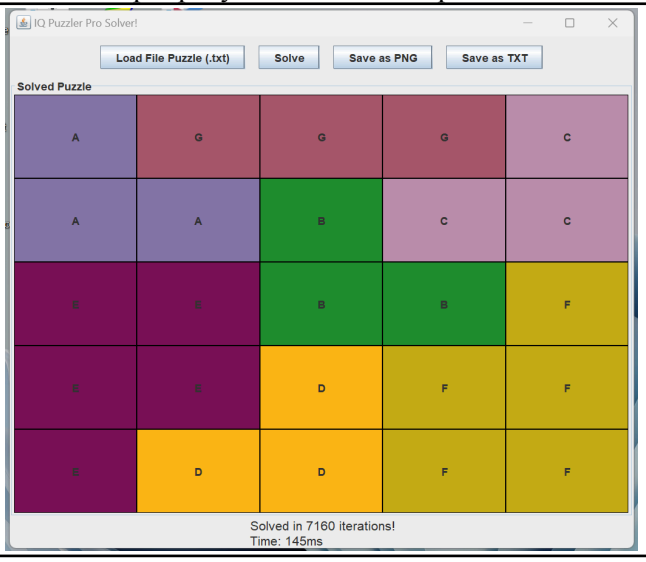
MASUKAN DAN LUARAN PROGRAM

4.1 Test Case 1

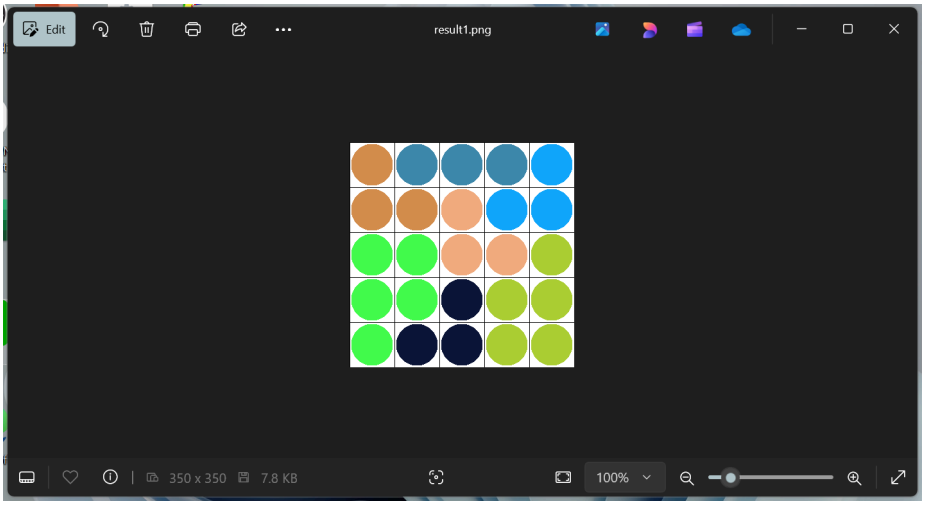
test1.txt
test > test1.txt 5 5 7 DEFAULT A AA B BB C CC D DD EE EE E FF FF F GGG

Tampilan GUI setelah file diupload


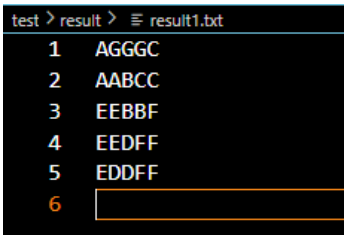
Output penyelesaian test1.txt pada GUI



Output setelah disimpan sebagai gambar (.png)

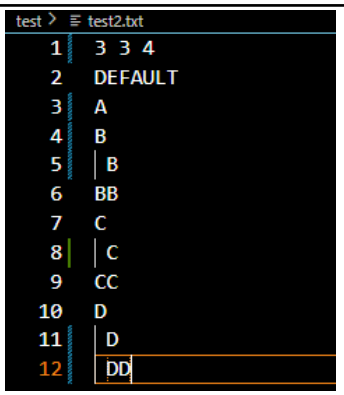


Output setelah disimpan sebagai text (.txt)

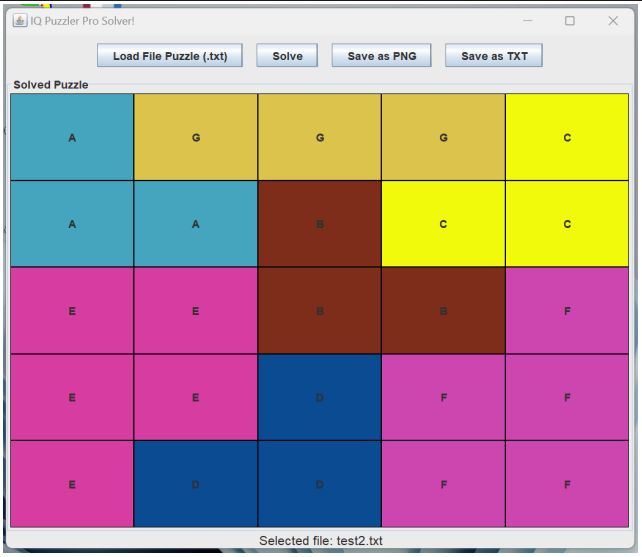


4.2 Test Case 2 (Input Tidak Valid)

test2.txt

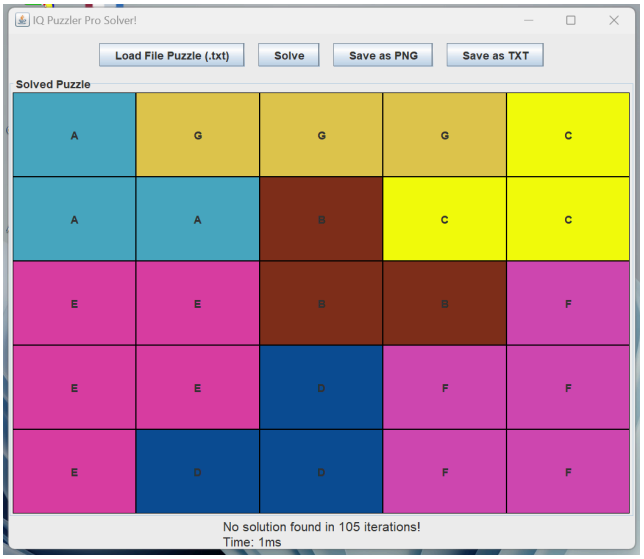


Tampilan GUI setelah file diupload

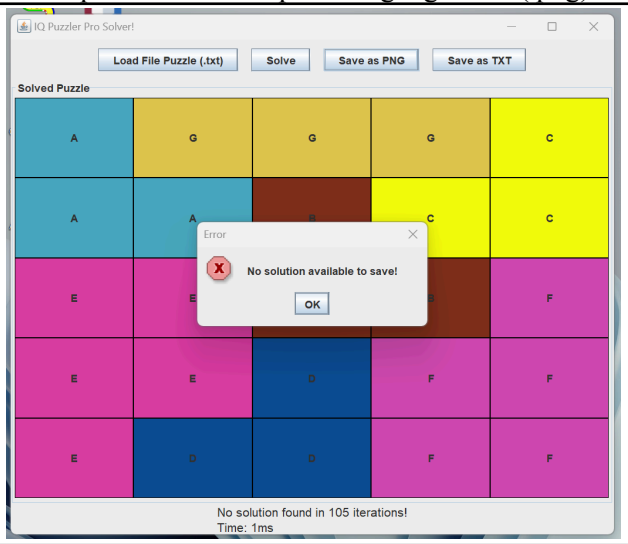


(masih tertampil output sebelumnya, tetapi status label pada bagian bawah sudah berubah)

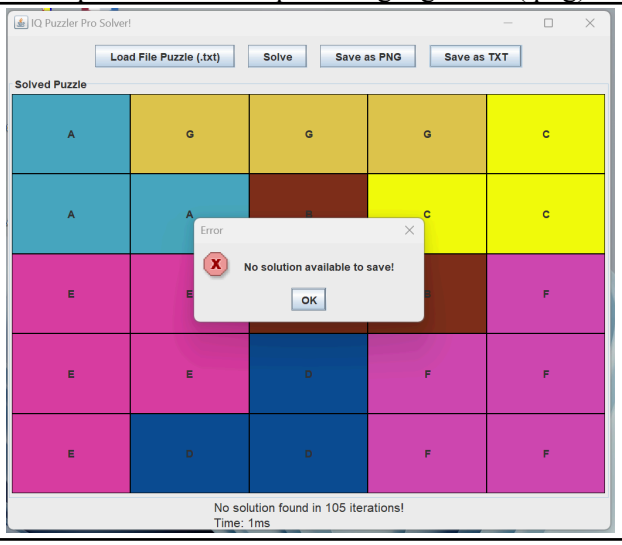
Output penyelesaian test2.txt pada GUI



Output setelah disimpan sebagai gambar (.png)



Output setelah disimpan sebagai gambar (.png)



4.3 Test Case 3 (Jumlah Pieces pada P Kurang)

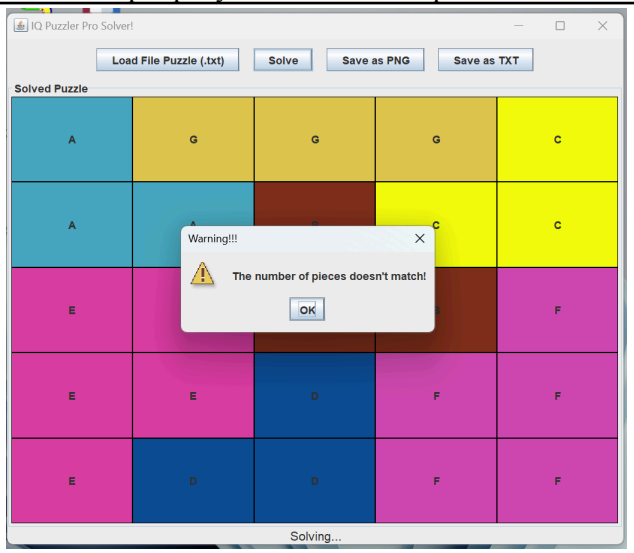
test3.txt

```
test > test3.txt
1   5 11 7
2   DEFAULT
3   AA
4   A
5   A
6   BBB
7   BB
8   C
9   CC
10  | C
11  D
12  DD
13  | DD
14  EE
15  | EEE
16  FF
17  F
18  F
19  F
20  GGGG
21  | G
22  | H
23  HHH
24  H
25  I
26  II
27  JJJ
28  J J
29  | K
30  KKK
31  LLL
32  L
33  L
```

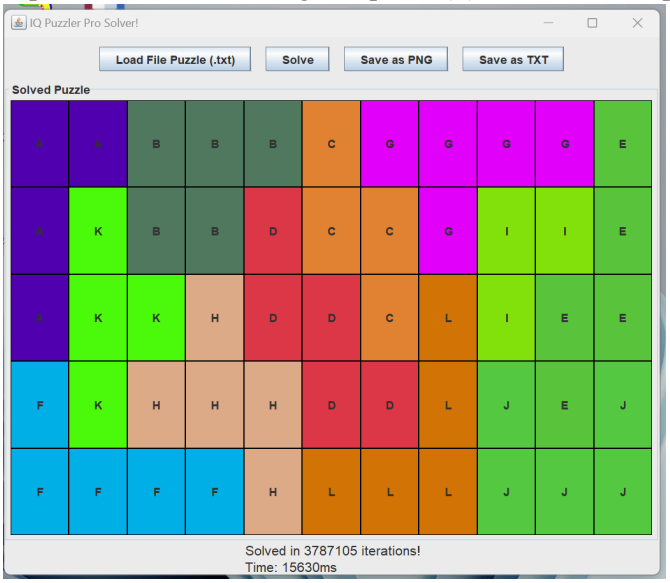
Tampilan GUI setelah file diupload

(masih tertampil output sebelumnya, tetapi status label pada bagian bawah sudah berubah)

Output penyelesaian test3.txt pada GUI

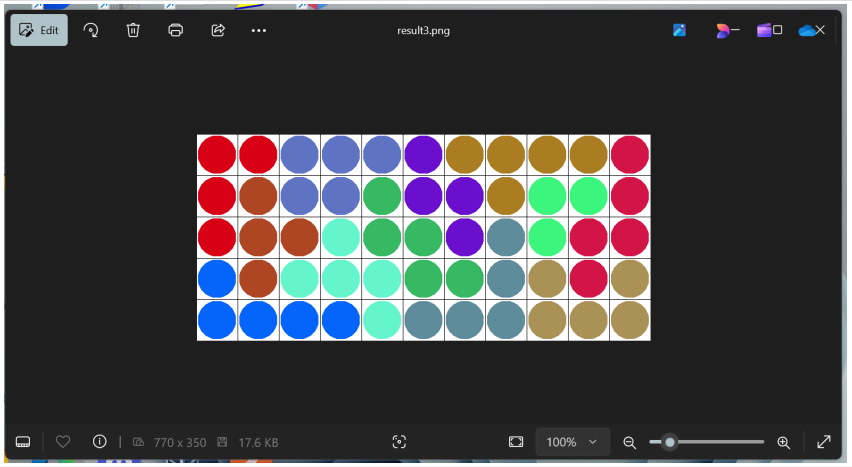


(Jika jumlah pieces tidak sesuai dengan inputan (P) maka akan tampil pop up)



(Waktu yang tampil menjadi bergantung pada kecepatan menekan tombol 'OK' pada pop up, karena setelahnya dilakukan proses solve)

Output setelah disimpan sebagai gambar (.png)



Output setelah disimpan sebagai gambar (.png)

test > result > result3.txt

```
1  AABBBCGGGGE
2  AKBBDCCGIIE
3  AKKHDDCLIEE
4  FKHHHDDLJEJ
5  FFFFHLLLJJJ
6
```

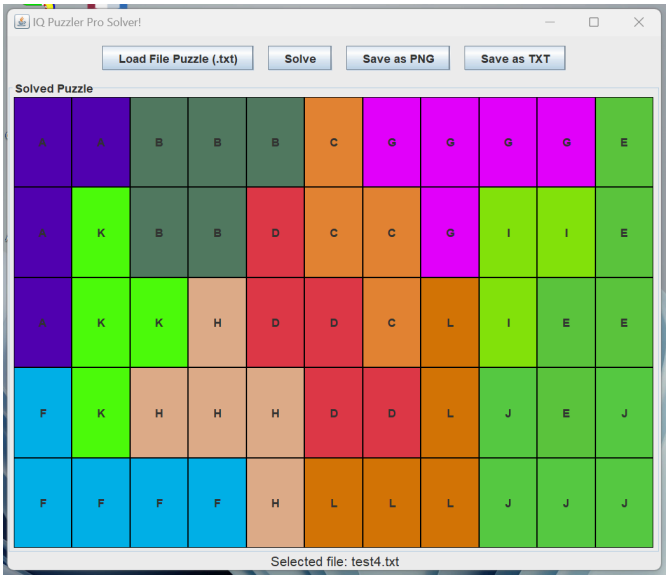
4.4 Test Case 4

test4.txt

test > test4.txt

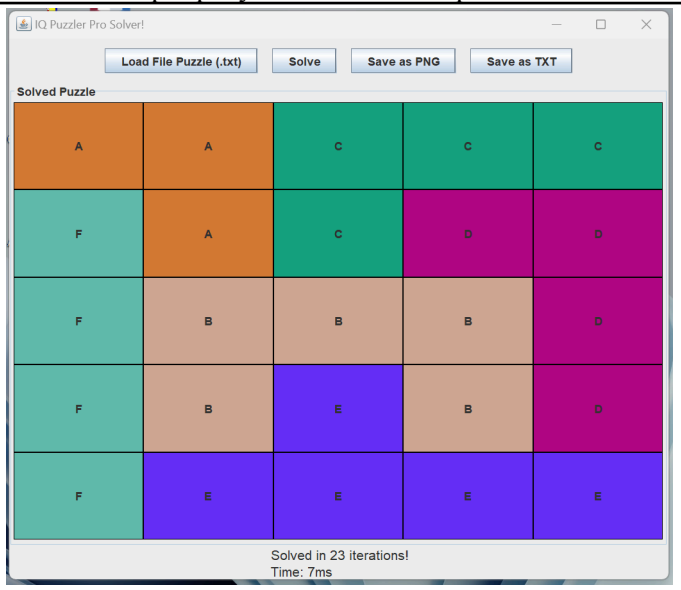
```
1  5 5 6
2  DEFAULT
3  AA
4   A
5  CCC
6  C
7  BBB
8  B B
9  DDD
10 D
11  E
12 EEEE
13 FFFF
```

Tampilan GUI setelah file diupload

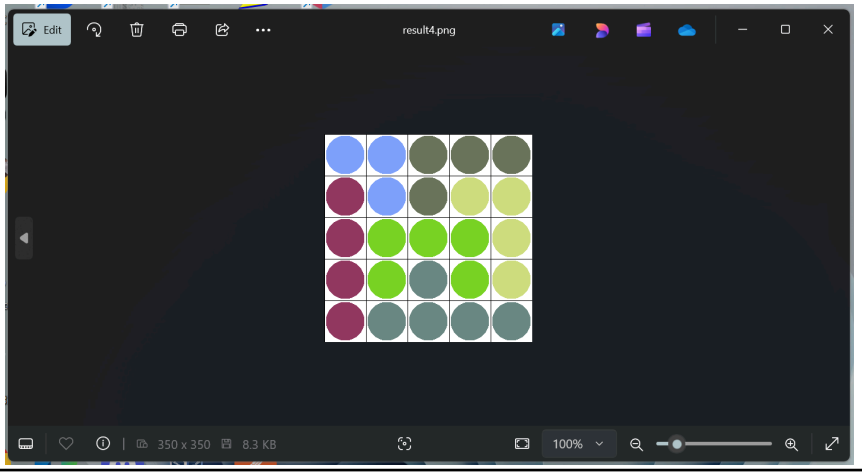


(masih tertampil output sebelumnya, tetapi status label pada bagian bawah sudah berubah)

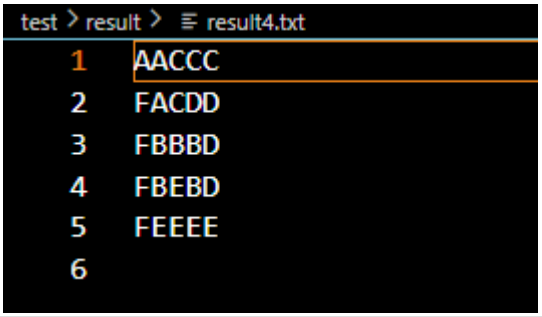
Output penyelesaian test4.txt pada GUI



Output setelah disimpan sebagai gambar (.png)

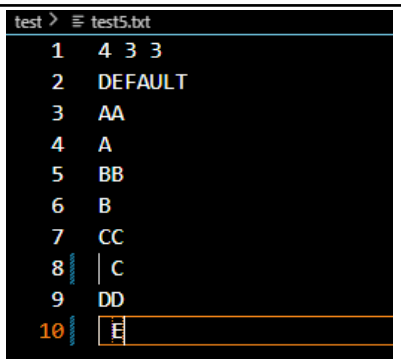


Output setelah disimpan sebagai gambar (.png)

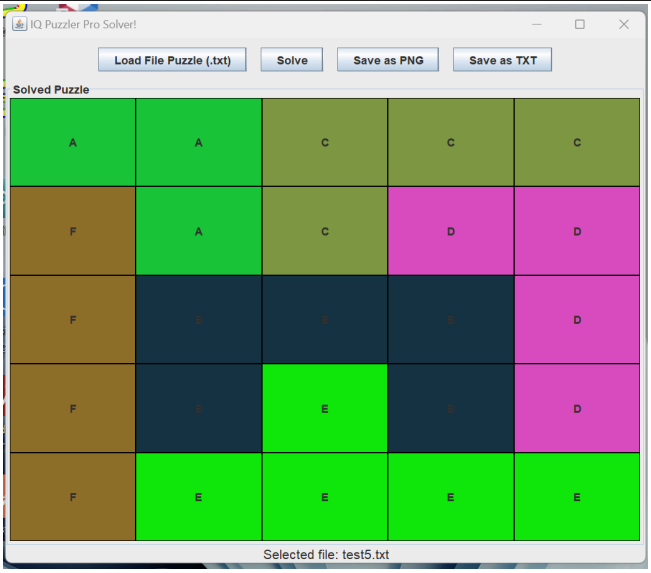


4.5 Test Case 5 (Jumlah Pieces Berlebih)

test5.txt

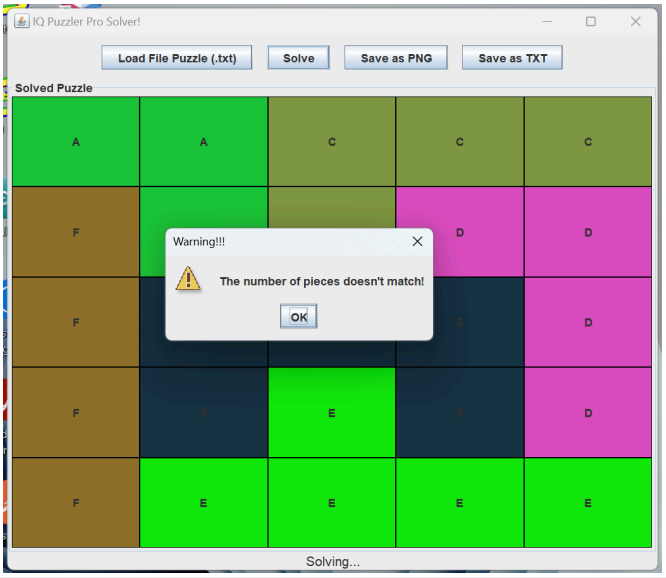


Tampilan GUI setelah file diupload

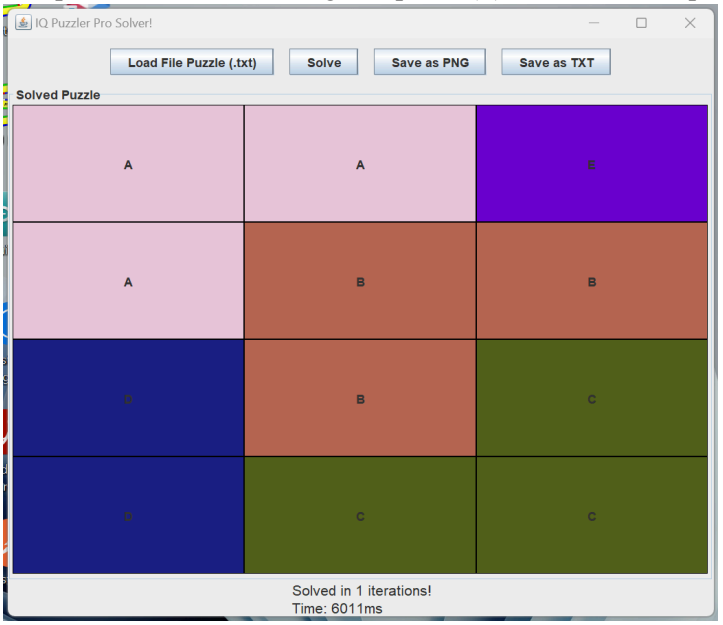


(masih tertampil output sebelumnya, tetapi status label pada bagian bawah sudah berubah)

Output penyelesaian test5.txt pada GUI

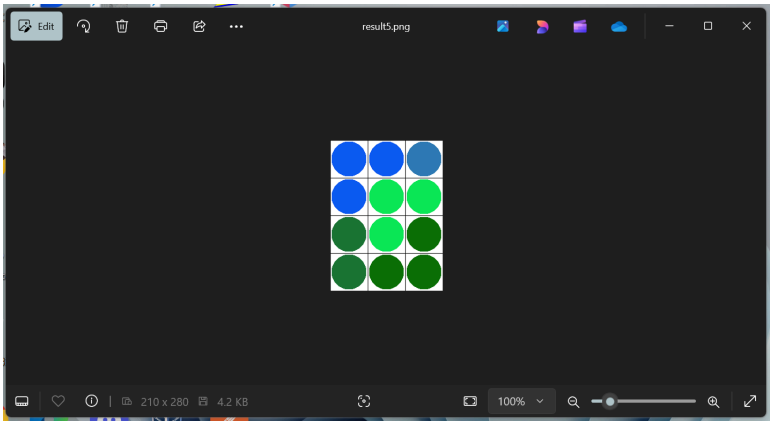


(Jika jumlah pieces tidak sesuai dengan inputan (P) maka akan tampil pop up)

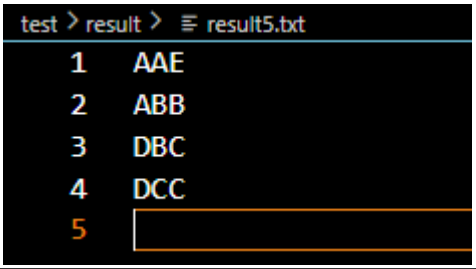


(Waktu yang tampil menjadi bergantung pada kecepatan menekan tombol ‘OK’ pada pop up, karena setelahnya dilakukan proses solve)

Output setelah disimpan sebagai gambar (.png)



Output setelah disimpan sebagai gambar (.png)

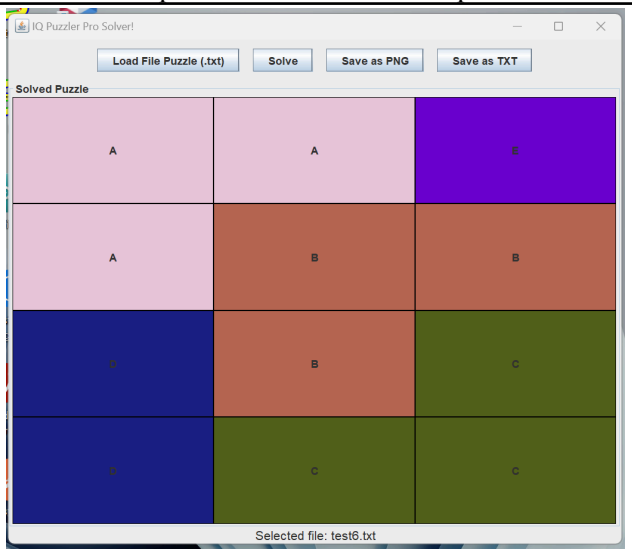


4.6 Test Case 6 (Pieces Berlebih pada Board, Dianggap Tidak Valid)

test6.txt

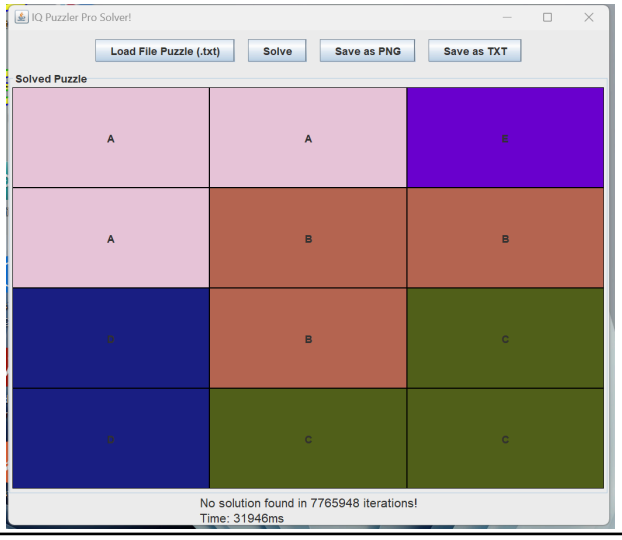
```
test > test6.txt
1   5 5 8
2   DEFAULT
3   A
4   AA
5   B
6   BB
7   C
8   CC
9   D
10  DD
11  EE
12  EE
13  E
14  FF
15  FF
16  F
17  GGG
18  HHH
```

Tampilan GUI setelah file diupload

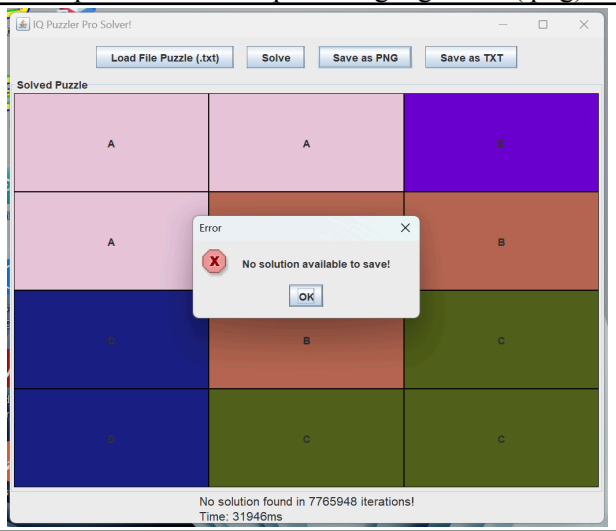


(masih tertampil output sebelumnya, tetapi status label pada bagian bawah sudah berubah)

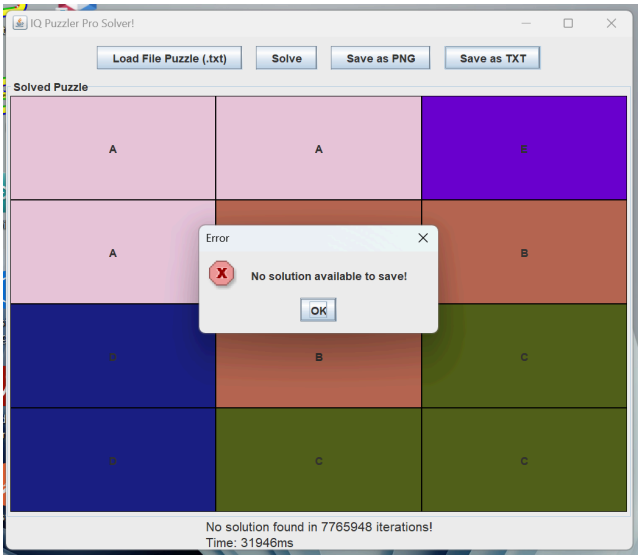
Output penyelesaian test6.txt pada GUI



Output setelah disimpan sebagai gambar (.png)

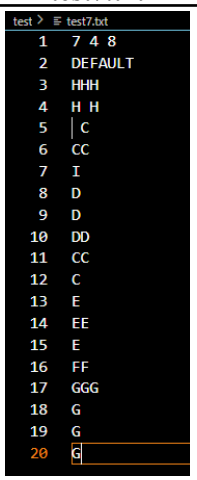


Output setelah disimpan sebagai text (.txt)

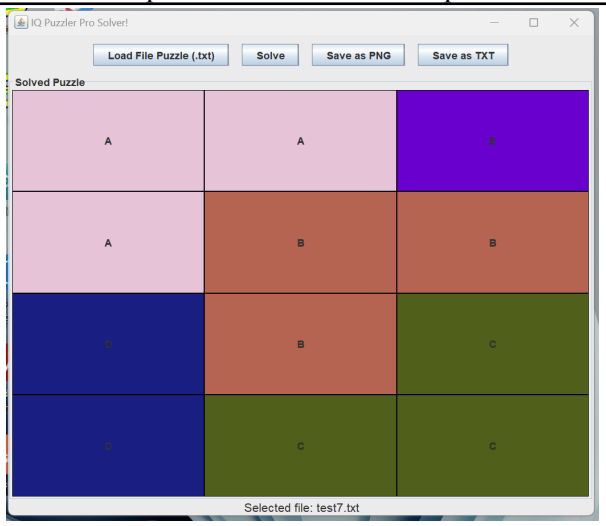


4.7 Test Case 7

test7.txt

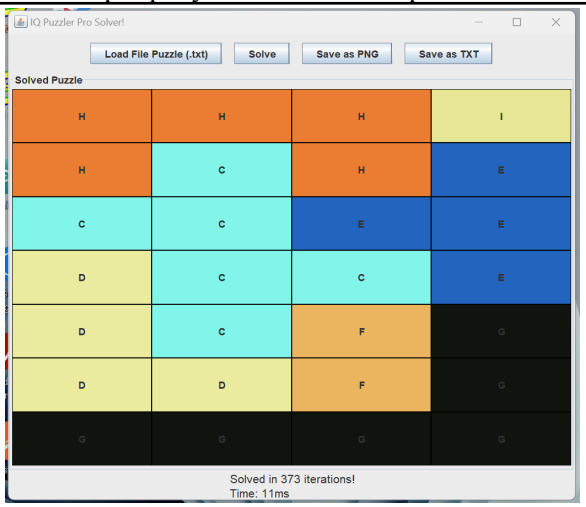


Tampilan GUI setelah file diupload

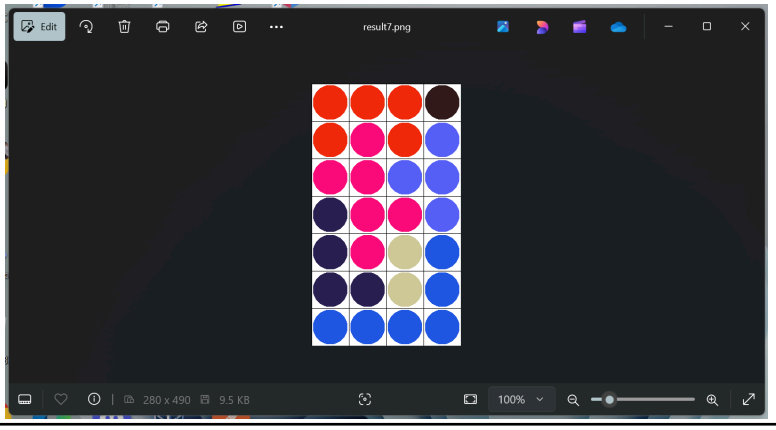


(masih tertampil output sebelumnya, tetapi status label pada bagian bawah sudah berubah)

Output penyelesaian test7.txt pada GUI



Output setelah disimpan sebagai gambar (.png)



Output setelah disimpan sebagai text (.txt)

test > result > result7.txt

```
1  HHHI
2  HCHE
3  CCEE
4  DCCE
5  DCFG
6  DDFG
7  GGGG
8
```

4.8 Test Case 8

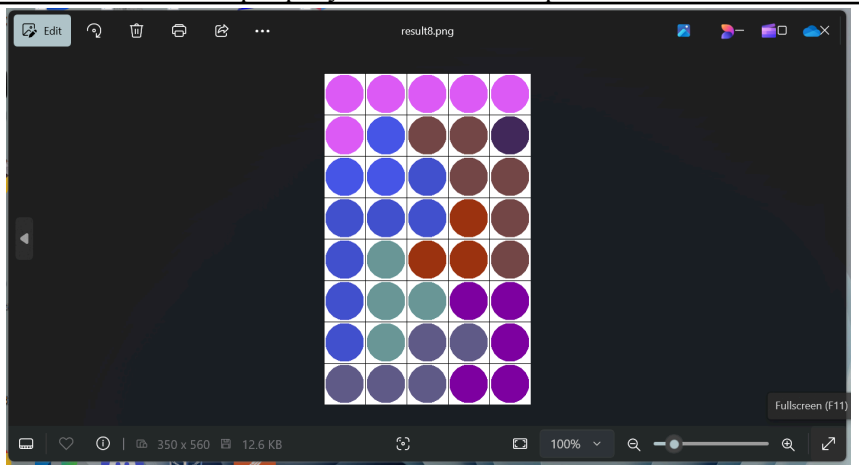
test8.txt

test > test8.txt

```
1  8 5 9
2  DEFAULT
3  LLLLL
4  L
5  C
6  CC
7  HHH
8  H H
9  I
10 DDD
11  DD
12    D
13 E
14 EE
15 E
16 AA
17 A
18 F
19 FF
20 F
21 F
22  G
23 GGG
24 G
25 G
26 G
```

Output penyelesaian test8.txt pada CLI

```
PS D:\STIMA\Tucil1_13523107\src> java Main
Enter test case file path: test8.txt
L L L L L
L C D D I
C C G D D
G G G A D
G E A A D
G E E H H
G E F F H
F F F H H
Do you want to save the solution as an image? (yes/no): yes
Input the filename: result8
Image saved to: ../test/result/result8.png
Iterations: 1909922
Execution time: 75791 ms
PS D:\STIMA\Tucil1_13523107\src>
```

Output penyelesaian test8.txt pada GUI


BAB V

LAMPIRAN

No	Poin	Ya	Tidak
1	Program berhasil dikompilasi tanpa kesalahan	✓	
2	Program berhasil dijalankan	✓	
3	Solusi yang diberikan program benar dan mematuhi aturan permainan	✓	
4	Program dapat membaca masukan berkas .txt serta menyimpan solusi dalam berkas .txt	✓	
5	Program memiliki *Graphical User Interface* (GUI)	✓	
6	Program dapat menyimpan solusi dalam bentuk file gambar	✓	
7	Program dapat menyelesaikan konfigurasi custom		✓
8	Program dapat menyelesaikan kasus konfigurasi piramida (3D)		✓
9	Program dibuat oleh saya sendiri	✓	

DAFTAR PUSTAKA

Jain, S. (2024, January 18). *Brute Force Approach and its pros and cons*. GeeksforGeeks. Retrieved February 21, 2025, from https://www.geeksforgeeks.org/brute-force-approach-and-its-pros-and-cons/

Java Tutorial. (n.d.). W3Schools. Retrieved February 17, 2025, from https://www.w3schools.com/java/default.asp

Munir, R. (n.d.). *Algoritma Brute Force (Bagian 1)*. Strategi Algoritma. Retrieved 02 21, 2025, from https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/02-Algoritma-Brute-Force-(2025)-Bag1.pdf

Swing Introduction - Tpoint Tech. (n.d.). JavaTpoint. Retrieved February 20, 2025, from https://www.tpointtech.com/java-swing