

IF2211 Strategi Algoritma
Laporan Tugas Kecil 1
Penyelesaian IQ Puzzler Pro dengan Algoritma Brute
Force



Disusun oleh:
Daniel Pedrosa Wu – 13523099

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

DAFTAR ISI

DAFTAR ISI	2
BAB I: DESKRIPSI MASALAH DAN ALGORITMA BRUTE FORCE.....	3
1.1. Permainan IQ Puzzler Pro.....	3
1.2. Algoritma Brute Force dalam Penyelesaian Permainan IQ Puzzler Pro	4
BAB II: IMPLEMENTASI DALAM BAHASA JAVA	7
2.1. Spesifikasi Kelas	7
2.2. Detail Implementasi	9
2.3. Source Code	13
BAB III: EKSPERIMEN	30
BAB IV: LAMPIRAN	43

BAB I:

DESKRIPSI MASALAH DAN ALGORITMA BRUTE FORCE

1.1. Permainan IQ Puzzler Pro



Gambar 1. Permainan IQ Puzzler Pro
(Sumber: <https://www.smartgamesusa.com>)

IQ Puzzler Pro adalah permainan papan yang diproduksi oleh Perusahaan Smart Games. Dalam permainan ini, pemain ditugaskan untuk mengisi seluruh papan dengan menggunakan *piece* (blok puzzle) yang tersedia. Setiap blok terdiri dari bentuk yang beragam-ragam dan pemain harus menyusun blok-blok tersebut sedemikian sehingga papan terisi penuh tanpa ada ruang kosong yang tersedia.

Komponen dari permainan IQ Puzzler Pro terdiri dari *board* (papan) dan *piece* (blok). Papan menjadi lapangan permainan yang merupakan komponen utama dalam permainan IQ Puzzler Pro. Pemain harus mampu mengisi seluruh area papan dengan menggunakan blok-blok yang tersedia. Blok adalah komponen yang digunakan untuk mengisi papan. Setiap blok memiliki bentuknya masing-masing dan dapat diorientasikan sedemikian rupa agar sesuai dengan ruang yang tersedia di papan.

Untuk tujuan tugas ini, permainan akan dimulai dengan papan yang kosong. Pemain dapat meletakkan blok sedemikian sehingga tidak ada blok yang bertumpang tindih. Permainan dianggap selesai jika dan hanya jika papan terisi penuh dan seluruh blok puzzle berhasil diletakkan. Pada tugas ini, cukup satu solusi dari permainan IQ Puzzler Pro ditemukan dengan

menggunakan algoritma, sesuai dengan syarat yang telah ditentukan sebelumnya, dan jika tidak ada solusinya, maka pernyataan bahwa solusi tidak ditemukan akan ditampilkan.

1.2. Algoritma Brute Force dalam Penyelesaian Permainan IQ Puzzler Pro

Algoritma Brute Force adalah pendekatan pemecahan masalah yang memecahkan suatu masalah secara langsung, sederhana, dan mudah dipahami dengan menguji semua kemungkinan solusi hingga menemukan solusi yang benar atau mengonfirmasi bahwa tidak ada solusi yang benar. Pendekatan ini adalah pendekatan yang lurus atau lempang dan sering dianggap tidak efisien. Walaupun begitu, pendekatan ini hampir selalu menemukan solusi karena menelusuri semua kemungkinan.

Karena sifatnya yang naif, algoritma Brute Force memerlukan biaya komputasi yang besar dan waktu penyelesaian yang lama. Oleh karena itu, penggunaan algoritma Brute Force lebih cocok digunakan untuk masalah yang sederhana atau masalah dengan ruang solusi yang kecil atau terbatas sehingga jumlah kemungkinan yang ditelusuri masih dalam batas yang wajar. Pada masalah yang lebih kompleks, algoritma Brute Force juga dapat digunakan sebagai tolak ukur untuk mengukur kesanggupan algoritma lain.

Solusi dari permainan IQ Puzzler Pro dapat ditemukan secara algoritmik dengan pendekatan *brute force*. Pada pengerjaan tugas kecil ini, papan permainan akan direpresentasikan oleh suatu matriks berukuran $N \times M$. Pada matriks ini, bagian yang dapat diisi ditandai oleh karakter ‘.’. Setiap blok pun akan direpresentasikan oleh suatu matriks dengan ukuran dan bentuknya tersendiri. Algoritma yang digunakan dalam tugas ini adalah sebagai berikut:

- A. Tempatkan suatu blok pada posisi pertama (sudut kiri atas) di matriks papan permainan.
- B. Periksa apakah penempatan blok tersebut sah (tidak bertumpang tindih dengan blok lain dan berada pada posisi yang dapat diisi, serta tidak melampaui batas papan).
- C. Jika blok tersebut tidak sah, maka ubah orientasi dari blok tersebut (baik dengan memutar blok sebesar 90° atau dengan mencerminkan blok tersebut). Setelah orientasinya diubah, periksa apakah penempatan blok tersebut sudah sah. Proses ini dilanjutkan hingga ditemukan suatu orientasi yang sah atau semua orientasi telah diuji.
- D. Jika tidak ada orientasi yang sah setelah semua kemungkinan diuji, maka pindah ke posisi berikutnya dan ulangi dari langkah B.
- E. Jika ditemukan orientasi yang sah, maka tempatkan blok pada posisi tersebut lalu lanjut ke blok lainnya.
- F. Jika pada suatu blok tidak dapat diletakkan di posisi manapun dengan orientasi manapun, maka dilakukan proses *backtracking*. Blok tersebut ditinggalkan dan diganti dengan blok sebelumnya. Blok tersebut akan diuji orientasi hingga ditemukan orientasi yang sah. Jika tidak ada orientasi yang sah, maka berpindah ke posisi selanjutnya dan kembali ke langkah B.
- G. Proses ini akan berulang hingga ditemukan satu solusi yang sah, diketahui bahwa pencarian pasti gagal (papan terisi dengan penuh tetapi masih ada blok yang tersisa), atau telah diuji semua kemungkinan.

Algoritma ini dapat direpresentasikan lebih jelas dalam bentuk *pseudocode* sebagai berikut:

```
function findSolution(idx):
    if earlyStop is true:
        return false

    if board is full AND idx < total pieces:
        set earlyStop to true
        return false

    if idx ≥ total pieces:
        return whether board is full

    currentPiece ← get piece with index idx

    for each row i in board:
        for each column j in board:
            if earlyStop is true:
                Return false

            for each variant of currentPiece:
                if piece can be placed at (i, j):
                    place piece at (i, j)

                    if findSolution(idx + 1) returns true:
                        return true

                    remove piece from (i, j)

    return false
```

Algoritma ini diimplementasikan secara rekursif. Ada 3 hal yang menjadi *basis* dalam proses rekursi ini, yakni:

- A. Apakah *flag* *earlyStop* hidup, yang akan langsung menghentikan pencarian.
- B. Jika tidak, akan dievaluasi apakah papan penuh tetapi tidak semua blok terpakai, yang akan menghidupkan *flag* *earlyStop* dan langsung menghentikan pencarian.
- C. Jika tidak, akan dievaluasi apakah semua blok telah terpakai, yang akan memeriksa apakah papan sudah terisi penuh lalu mengembalikan hasilnya.

```
if earlyStop is true:
    return false

if board is full AND idx < total pieces:
    set earlyStop to true
    return false

if idx ≥ total pieces:
    return whether board is full
```

Kedua kasus pertama bertujuan untuk menghentikan pencarian ketika telah diketahui bahwa pencarian tidak memungkinkan solusi. Hal ini terjadi ketika papan sudah terisi penuh, namun masih ada blok yang tersisa.

```
for each row i in board:
    for each column j in board:
        if earlyStop is true:
            Return false
```

```
for each variant of currentPiece:
```

Seperti yang telah dijelaskan sebelumnya, akan diambil suatu potongan dan diperiksa di setiap posisi pada papan. Ini dilakukan melalui *nested for-loop* yang akan menelusuri matriks secara traversal baris-kolom. Di setiap posisi, akan diperiksa apakah *flag earlyStop* hidup. Ini bertujuan untuk menghentikan program jika flag tersebut hidup di tengah proses *looping* ini. Di setiap posisi, diperiksa apakah orientasi suatu blok dapat diletakkan di posisi tersebut. Jika tidak bisa, maka akan diubah orientasinya. Ini dilakukan melalui *for-loop* yang memeriksa tiap variasi atau orientasi sebanyak sekali dari blok yang sudah ditinjau.

```
if piece can be placed at (i, j):  
    place piece at (i, j)  
  
    if findSolution(idx + 1) returns true:  
        return true  
  
remove piece from (i, j)
```

Jika suatu orientasi dapat diletakkan di posisi tersebut, maka potongan tersebut akan diletakkan dan berpindah ke blok berikutnya. Proses ini dilakukan dengan memanggil fungsi secara rekursif namun untuk blok selanjutnya. Blok ini pun akan mengikuti proses yang sama dengan blok sebelumnya dan akan berlanjut hingga mencapai salah satu kasus basis. Kasus basis akan menentukan apakah susunan sudah benar. Jika sudah benar, maka akan terjadi propagasi keberhasilan yang menyatakan bahwa solusi sudah tercapai. Jika belum benar, maka akan ada dua kemungkinan yaitu pencarian tidak memungkinkan solusi atau tidak ada posisi yang sah untuk blok terakhir dengan susunan blok saat ini. Jika pencarian tidak memungkinkan solusi, maka program akan berhenti sementara pada kasus yang lain akan berlanjut ke *backtracking* untuk lanjut mencoba mencari solusi.

BAB II:

IMPLEMENTASI DALAM BAHASA JAVA

2.1. Spesifikasi Kelas

Program ini diimplementasikan dengan menggunakan bahasa pemrograman Java. Java adalah bahasa pemrograman yang berparadigma berorientasi objek. Yang dimaksud dengan paradigma berorientasi objek adalah pendekatan yang memodelkan sistem sebagai kumpulan dari objek yang memiliki data (*attribute*) dan perilaku (*method*) tersendiri, serta interaksi antar mereka. Objek dibuat berdasarkan kelas, yang berperan sebagai *blueprint* dalam mendefinisikan suatu objek. Pada program ini, terdapat 3 kelas utama yang berperan dalam implementasi algoritma *brute force* untuk menyelesaikan permainan IQ Puzzler Pro, yakni:

2.1.1. Kelas Piece

Kelas Piece adalah kelas yang merepresentasikan setiap blok puzzle.

Berikut ini adalah atribut-atribut dari kelas Piece:

Atribut	Penjelasan
List<char[][]> variants	Menyimpan setiap orientasi unik dari suatu blok.

Berikut ini adalah metode-metode dari kelas Piece:

Metode	Penjelasan
char[][] rotate(char[][] base)	Merotasi suatu array 2D base sebesar 90° berlawanan arah jarum jam.
char[][] flip(char[][] base)	Mencerminkan suatu array 2D base secara horizontal.
boolean checkDupes (char[][] shape, Set<String> unique)	Memeriksa apakah array 2D shape sudah berada di dalam himpunan string unique.
void generateVariants(char[][] base)	Menghasilkan variasi atau orientasi lain dari array 2D dimensi base dengan menggunakan metode rotate atau flip.
char[][] trimPiece(char[][] piece)	Memotong bagian kosong pada ujung array 2D piece.

2.1.2. Kelas Board

Kelas Board adalah kelas yang merepresentasikan papan permainan IQ Puzzler Pro

Berikut ini adalah atribut-atribut dari kelas Board:

Atribut	Penjelasan
int rows	Menyimpan jumlah baris pada papan permainan.
int cols	Menyimpan jumlah kolom pada papan permainan.

<code>char[][] originalGrid</code>	Menyimpan bentuk papan beserta isinya sebelum modifikasi.
<code>char[][] grid</code>	Menyimpan bentuk papan beserta isinya.
<code>List<Piece> pieces</code>	Menyimpan sebuah <i>list</i> yang berisi setiap bentuk blok.
<code>int pieceCount</code>	Menyimpan jumlah blok.
<code>Map<Character, String> colorMap</code>	Mengasosiasikan suatu warna dengan suatu blok

Berikut ini adalah metode-metode dari kelas Board:

Metode	Penjelasan
<code>boolean isGridFull()</code>	Memeriksa apakah semua posisi pada papan permainan telah terisi.
<code>void displayBoard()</code>	Menampilkan papan permainan.
<code>void resetGrid()</code>	Mengembalikan papan ke bentuk semula.

2.1.3. Kelas Solver

Kelas Solver adalah kelas yang bertujuan untuk menemukan solusi dari permainan IQ Puzzler Pro.

Berikut ini adalah atribut-attribut dari kelas Solver:

Atribut	Penjelasan
<code>Board board</code>	Menyimpan papan permainan.
<code>long iterationCount</code>	Menyimpan jumlah iterasi yang ditinjau.
<code>double executionTime</code>	Menyimpan waktu eksekusi program.
<code>boolean earlyStop</code>	Flag yang menandai kapan program harus berhenti lebih awal.

Berikut ini adalah metode-metode dari kelas Solver:

Metode	Penjelasan
<code>boolean canPlacePiece(int r, int c, char[][] variant)</code>	Memeriksa apakah array 2D <i>variant</i> dapat diletakkan pada papan di posisi (r, c).
<code>void placePiece(int r, int c, char[][] variant)</code>	Meletakkan array 2D <i>variant</i> pada papan di posisi (r, c).
<code>void removePiece(int r, int c, char[][] variant)</code>	Menghapus array 2D <i>variant</i> dari papan di posisi (r, c).
<code>boolean findSolution(int idx)</code>	Metode utama untuk menentukan solusi secara rekursif.
<code>boolean solve(int idx)</code>	Metode untuk memanggil metode <code>findSolution</code> dan menyimpan hasil.

2.2. Detail Implementasi

Membangkitkan Semua Variasi dari Suatu Blok

```
private void generateVariants(char[][] base) {
    Set<String> unique = new LinkedHashSet<>();

    char[][] currentShape = base;
    if (checkDupes(currentShape, unique)) {
        variants.add(currentShape);
    }

    for (int i = 0; i < 3; i++) {
        currentShape = rotate(currentShape);
        if (checkDupes(currentShape, unique)) {
            variants.add(currentShape);
        }
    }

    char[][] flippedShape = flip(base);
    if (checkDupes(flippedShape, unique)) {
        variants.add(flippedShape);
    }

    currentShape = flippedShape;

    for (int i = 0; i < 3; i++) {
        currentShape = rotate(currentShape);
        if (checkDupes(currentShape, unique)) {
            variants.add(currentShape);
        }
    }
}
```

Metode `generateVariants` berfungsi untuk membangkitkan semua variasi dari orientasi suatu blok. Variasi-variasi ini disimpan di dalam sebuah atribut dalam kelas `Piece`, yakni `variants`. Metode `rotate` berfungsi untuk merotasikan blok sebesar 90° berlawanan jarum jam dan metode `flip` berfungsi untuk mencerminkan blok secara horizontal. Dengan menggunakan kedua metode ini, akan dihasilkan maksimal 8 variasi yang berbeda dari suatu blok (termasuk variasi blok awal). Agar tidak mubazir, digunakan suatu set `unique` yang berfungsi menyimpan variasi yang sudah ada. Keunikan dari tiap variasi diperiksa melalui metode `checkDupes` yang berfungsi mengembalikan suatu nilai boolean mengenai keunikan variasi serta menambahkan variasi ke dalam set `unique`.

Memangkas Matriks

```
private char[][] trimPiece(char[][] piece) {
    int rows = piece.length;
    int cols = piece[0].length;

    int minRows = rows, maxRows = 0, minCols = cols, maxCols = 0;

    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            if (piece[i][j] != ' ') {
                minRows = Math.min(minRows, i);
            }
        }
    }
}
```

```

        maxRows = Math.max(maxRows, i);
        minCols = Math.min(minCols, j);
        maxCols = Math.max(maxCols, j);
    }
}

int newRows = maxRows - minRows + 1;
int newCols = maxCols - minCols + 1;

char[][] trimmedPiece = new char[newRows][newCols];
for (int i = 0; i < newRows; i++) {
    for (int j = 0; j < newCols; j++) {
        trimmedPiece[i][j] = piece[minRows + i][minCols + j];
    }
}

return trimmedPiece;
}

```

Metode `trimPiece` berfungsi untuk memangkas atau menghapus bagian kosong di sekitar suatu matriks yang merepresentasikan suatu blok. Pada implementasi metode ini, ada 3 tahapan yang dilalui, yakni:

- A. Menentukan batas atas, bawah, kiri, dan kanan yang berisi elemen non-kosong.
- B. Membentuk matriks baru dengan ukuran yang bersesuaian dengan batas yang telah ditentukan.
- C. Menyalin elemen-elemen yang relevan dari matriks yang lama ke matriks yang baru.

Tahapan ini penting untuk meningkatkan efisiensi penyimpanan dan pemrosesan, serta menghindari kesalahan dalam penempatan blok.

Ukuran matriks yang lebih kecil akan memakan memori yang lebih kecil dibanding matriks yang lebih besar. Operasi-operasi yang dilakukan pada matriks yang lebih kecil juga jauh lebih cepat dibanding pada matriks yang besar. Ini dapat berdampak signifikan pada algoritma ketika permainan harus mengevaluasi banyak kemungkinan posisi dan orientasi blok dalam pencarian solusi. Selain itu, dengan memangkas bagian kosong, penempatan blok di papan permainan menjadi lebih akurat karena hanya bagian yang benar-benar berisi elemen yang dipertimbangkan. Hal ini mengurangi peluang kesalahan dalam validasi posisi dan memastikan algoritma dapat berjalan dengan lebih efisien.

Memeriksa Apakah Papan Permainan Telah Terisi Penuh

```

public boolean isGridFull() {
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            if (grid[i][j] == '.') {
                return false;
            }
        }
    }
    return true;
}

```

Metode `isGridFull` berfungsi untuk memeriksa apakah papan permainan sudah terisi penuh oleh blok. Metode ini bekerja dengan melakukan traversal baris-kolom di matriks papan permainan. Jika ditemukan elemen yang belum terisi (ditandai oleh karakter '.'), maka metode akan mengembalikan nilai `false`, dan akan mengembalikan nilai `true` jika tidak ditemukan karakter '.'. Metode ini sangat penting karena akan digunakan dalam basis proses rekursi pencarian solusi.

Memeriksa Apakah Suatu Blok dapat Diletakkan di Suatu Posisi

```
private boolean canPlacePiece(int r, int c, char[][] variant) {
    int rows = variant.length;
    int cols = variant[0].length;

    if (r + rows > board.getRows() || c + cols > board.getCols()) {
        return false;
    }

    for (int i = r; i < r + rows; i++) {
        for (int j = c; j < c + cols; j++) {
            if (variant[i - r][j - c] != ' ' && board.getElmt(i, j) != '.') {
                return false;
            }
        }
    }

    return true;
}
```

Metode `canPlacePiece` berfungsi untuk memeriksa apakah suatu blok dapat diletakkan di suatu posisi. Posisi ini ditentukan oleh koordinat (`r`, `c`), yang merepresentasikan sudut kiri atas dari area tempat blok akan coba ditempatkan. Pertama-tama, metode ini akan memeriksa apakah peletakan blok di posisi (`r`, `c`) akan melampaui batas permainan. Jika tidak, maka metode akan melakukan traversal baris-kolom terhadap matriks yang merepresentasikan blok. Jika setiap bagian dari blok dapat diletakkan pada papan, maka metode akan mengembalikan nilai `true`.

Meletakkan Blok

```
private void placePiece(int r, int c, char[][] variant) {
    int rows = variant.length;
    int cols = variant[0].length;

    for (int i = r; i < r + rows; i++) {
        for (int j = c; j < c + cols; j++) {
            if (variant[i - r][j - c] != ' ') {
                board.setElmt(i, j, variant[i - r][j - c]);
            }
        }
    }
}
```

Metode `placePiece` berfungsi untuk meletakkan blok ke dalam papan permainan. Metode ini dipanggil setelah sudah dipastikan bahwa blok `variant` sah diletakkan di posisi

(r, c) oleh metode `canPlacePiece`. Metode ini bekerja dengan menggantikan karakter pada papan mulai dari posisi (r, c) dengan karakter yang ada pada blok `variant`. Ini dilakukan dengan melakukan traversal baris-kolom pada papan permainan.

Menghapus Blok

```
private void removePiece(int r, int c, char[][] variant) {
    int rows = variant.length;
    int cols = variant[0].length;

    for (int i = r; i < r + rows; i++) {
        for (int j = c; j < c + cols; j++) {
            if (variant[i - r][j - c] != ' ') {
                board.setElmt(i, j, '.');
            }
        }
    }
}
```

Metode `removePiece` berfungsi untuk menghapus suatu blok dari papan permainan. Metode ini bekerja dengan mengganti karakter pada papan dengan karakter '.' Sesuai dengan bentuk blok `variant` yang direpresentasikan oleh suatu matriks. Ini dilakukan dengan melakukan traversal baris-kolom pada papan permainan.

Mencari Solusi

```
public boolean findSolution(int idx) {
    if (earlyStop) {
        return false;
    }

    if (board.isGridFull() && idx < board.getPieceCount()) {
        earlyStop = true;
        return false;
    }
    if (idx >= board.getPieceCount()) {
        return board.isGridFull();
    }

    Piece currentPiece = board.getPieces().get(idx);
    for (int i = 0; i < board.getRows(); i++) {
        for (int j = 0; j < board.getCols(); j++) {
            if (earlyStop) {
                return false;
            }
            for (char[][] currentVariant : currentPiece.getVariants()) {
                iterationCount++;
                if (canPlacePiece(i, j, currentVariant)) {
                    placePiece(i, j, currentVariant);
                    if (findSolution(idx + 1)) {
                        return true;
                    }
                    removePiece(i, j, currentVariant);
                }
            }
        }
    }
}
```

```
    return false;
}
```

Metode `findSolution` menyelesaikan permainan IQ Puzzler Pro secara rekursif. Selayaknya suatu implementasi rekursif, metode ini memiliki basis dan rekurens. Sebelum mendalami proses tersebut, akan dijelaskan mengenai penggunaan boolean `earlyStop`. Boolean ini berfungsi sebagai salah satu *pruning* yang diimplementasikan dalam algoritma ini. Tujuannya adalah menghentikan rekursi jika telah diketahui bahwa pencarian tidak akan menghasilkan solusi terhadap permainan. Ini terjadi ketika papan terisi penuh namun masih terdapat blok yang belum terpakai. Dalam kata lain, total luas dari semua blok melebihi total luas dari papan permainan sehingga proses rekursi tidak akan pernah mencapai solusi yang sah. Secara *default*, boolean ini bernilai `false`, artinya belum melakukan penghentian lebih awal. Di setiap langkah rekursi, algoritma akan selalu mengecek nilai `earlyStop`. Jika papan telah terisi penuh namun belum semua blok digunakan, maka boolean akan bernilai `true` dan rekursi akan “dihentikan”.

Setelah boolean `earlyStop`, yang menjadi *basis* sebenarnya dalam rekursi ini adalah kasus dimana semua blok telah digunakan. Kasus ini diperiksa dengan memeriksa apakah nilai `idx` lebih besar sama dengan jumlah blok total. Jika iya, maka dia akan memanggil metode `isGridFull` untuk memeriksa apakah papan permainan juga terisi penuh. Ini disebut sebagai basis sebenarnya karena kasus ini hanya tercapai ketika rekursi secara alami mencapai titik di mana semua blok telah ditempatkan.

Bagian yang menjadi rekurens adalah proses di mana algoritma mencoba menempatkan setiap blok pada setiap posisi yang sah pada papan permainan. Rekurens ini terjadi ketika metode `findSolution` dipanggil secara rekursif dengan indeks berikutnya (`idx + 1`). Suatu blok akan diperiksa pada tiap posisi papan permainan secara traversal baris-kolom. Setiap orientasi dari blok akan dicoba hingga tercapai orientasi yang bisa diletakkan di suatu posisi. Metode kemudian akan memanggil dirinya sendiri pada indeks berikutnya (blok berikutnya). Jika pemanggilan rekursi mengembalikan nilai `true`, maka rekursi akan berhenti dengan mengembalikan `true` secara berantai (propagasi keberhasilan). Jika tidak, maka algoritma akan melakukan *backtracking* dan mencoba posisi lain hingga solusi ditemukan atau dipastikan tidak ada solusi.

2.3. Source Code

Piece.java

```
package piece;

import java.util.*;

public class Piece {
    private List<char[][]> variants;

    private char[][] rotate(char[][] base) {
        int rows = base.length;
        int cols = base[0].length;

        char[][] rotatedShape = new char[cols][rows];
```

```

        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                rotatedShape[j][rows - i - 1] = base[i][j];
            }
        }

        return rotatedShape;
    }

    private char[][] flip(char[][] base) {
        int rows = base.length;
        int cols = base[0].length;

        char[][] flippedShape = new char[rows][cols];

        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                flippedShape[i][cols - j - 1] = base[i][j];
            }
        }

        return flippedShape;
    }

    private boolean checkDupes (char[][] shape, Set<String> unique) {
        return unique.add(Arrays.deepToString(shape));
    }

    private void generateVariants(char[][] base) {
        Set<String> unique = new LinkedHashSet<>();

        char[][] currentShape = base;
        if (checkDupes(currentShape, unique)) {
            variants.add(currentShape);
        }

        for (int i = 0; i < 3; i++) {
            currentShape = rotate(currentShape);
            if (checkDupes(currentShape, unique)) {
                variants.add(currentShape);
            }
        }

        char[][] flippedShape = flip(base);
        if (checkDupes(flippedShape, unique)) {
            variants.add(flippedShape);
        }

        currentShape = flippedShape;

        for (int i = 0; i < 3; i++) {
            currentShape = rotate(currentShape);
            if (checkDupes(currentShape, unique)) {
                variants.add(currentShape);
            }
        }
    }

    private char[][] trimPiece(char[][] piece) {
        int rows = piece.length;
        int cols = piece[0].length;

        int minRows = rows, maxRows = 0, minCols = cols, maxCols = 0;

        for (int i = 0; i < rows; i++) {

```

```

        for (int j = 0; j < cols; j++) {
            if (piece[i][j] != ' ') {
                minRows = Math.min(minRows, i);
                maxRows = Math.max(maxRows, i);
                minCols = Math.min(minCols, j);
                maxCols = Math.max(maxCols, j);
            }
        }

        int newRows = maxRows - minRows + 1;
        int newCols = maxCols - minCols + 1;

        char[][] trimmedPiece = new char[newRows][newCols];
        for (int i = 0; i < newRows; i++) {
            for (int j = 0; j < newCols; j++) {
                trimmedPiece[i][j] = piece[minRows + i][minCols + j];
            }
        }

        return trimmedPiece;
    }

    public Piece(List<String> shapeStrings) {
        this.variants = new ArrayList<>();
        int rows = shapeStrings.size();
        int cols = 0;

        for(String shapeString : shapeStrings){
            cols = Math.max(cols, shapeString.length());
        }

        char[][] base = new char[rows][cols];
        for (int i = 0; i < rows; i++) {
            int lineLength = shapeStrings.get(i).length();
            for (int j = 0; j < cols; j++) {
                if (j >= lineLength) {
                    base[i][j] = ' ';
                } else {
                    base[i][j] = shapeStrings.get(i).charAt(j);
                }
            }
        }

        generateVariants(trimPiece(base));
    }

    public List<char[][]> getVariants() {
        return variants;
    }
}

```

Board.java

```

package board;

import java.util.*;
import piece.Piece;

public class Board {
    private int rows;
    private int cols;
    private char[][] originalGrid;

```

```

private char[][] grid;
private List<Piece> pieces;
private int pieceCount;
private static Map<Character, String> colorMap = new HashMap<>() {{
    put('A', "\033[38;2;255;0;0m"); // Red (255, 0, 0)
    put('B', "\033[38;2;0;255;0m"); // Green (0, 255, 0)
    put('C', "\033[38;2;0;0;255m"); // Blue (0, 0, 255)
    put('D', "\033[38;2;255;255;0m"); // Yellow (255, 255, 0)
    put('E', "\033[38;2;255;0;255m"); // Magenta (255, 0, 255)
    put('F', "\033[38;2;0;255;255m"); // Cyan (0, 255, 255)
    put('G', "\033[38;2;128;0;0m"); // Dark Red (128, 0, 0)
    put('H', "\033[38;2;0;128;0m"); // Dark Green (0, 128, 0)
    put('I', "\033[38;2;0;0;128m"); // Dark Blue (0, 0, 128)
    put('J', "\033[38;2;128;128;0m"); // Olive (128, 128, 0)
    put('K', "\033[38;2;128;0;128m"); // Purple (128, 0, 128)
    put('L', "\033[38;2;0;128;128m"); // Teal (0, 128, 128)
    put('M', "\033[38;2;255;165;0m"); // Orange (255, 165, 0)
    put('N', "\033[38;2;75;0;130m"); // Indigo (75, 0, 130)
    put('O', "\033[38;2;139;69;19m"); // Saddle Brown (139, 69, 19)
    put('P', "\033[38;2;255;192;203m"); // Pink (255, 192, 203)
    put('Q', "\033[38;2;173;216;230m"); // Light Blue (173, 216, 230)
    put('R', "\033[38;2;0;100;0m"); // Dark Green (0, 100, 0)
    put('S', "\033[38;2;165;42;42m"); // Brown (165, 42, 42)
    put('T', "\033[38;2;240;230;140m"); // Khaki (240, 230, 140)
    put('U', "\033[38;2;46;139;87m"); // Sea Green (46, 139, 87)
    put('V', "\033[38;2;255;215;0m"); // Gold (255, 215, 0)
    put('W', "\033[38;2;0;191;255m"); // Deep Sky Blue (0, 191, 255)
    put('X', "\033[38;2;138;43;226m"); // Blue Violet (138, 43, 226)
    put('Y', "\033[38;2;127;255;212m"); // Aquamarine (127, 255, 212)
    put('Z', "\033[38;2;218;112;214m"); // Orchid (218, 112, 214)
}};

public Board(char[][] board, int rows, int cols) {
    this.rows = rows;
    this.cols = cols;
    this.originalGrid = new char[rows][cols];
    this.grid = new char[rows][cols];
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            this.originalGrid[i][j] = board[i][j];
            this.grid[i][j] = board[i][j];
        }
    }
    this.pieces = null;
}

public void setPieces(List<Piece> pieces, int P) {
    this.pieces = pieces;
    this.pieceCount = P;
}

public int getRows() {
    return rows;
}

public int getCols() {
    return cols;
}

public char[][] getGrid() {
    return grid;
}

public char getElmt(int r, int c) {
    return grid[r][c];
}

```



```

    public List<Piece> getPieces() {
        return pieces;
    }

    public int getPieceCount() {
        return pieceCount;
    }

    public void setElmt(int r, int c, char x) {
        grid[r][c] = x;
    }

    public boolean isGridFull() {
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                if (grid[i][j] == '.') {
                    return false;
                }
            }
        }
        return true;
    }

    public void displayBoard() {
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                if (getElmt(i, j) == ' ') {
                    System.out.print('.');
                } else if (getElmt(i, j) == '.') {
                    System.out.print('?');
                } else {
                    char elmt = getElmt(i, j);
                    String color = colorMap.getOrDefault(elmt, "");
                    System.out.print(color + elmt + "\033[0m");
                }
            }
            System.out.println();
        }
    }

    public void resetGrid() {
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                grid[i][j] = originalGrid[i][j];
            }
        }
    }
}

```

Solver.java

```

package solver;

import board.Board;
import piece.Piece;

public class Solver {
    private Board board;
    private long iterationCount;
    private double executionTime;
    private boolean earlyStop;
}

```

```

private boolean canPlacePiece(int r, int c, char[][] variant) {
    int rows = variant.length;
    int cols = variant[0].length;

    if (r + rows > board.getRows() || c + cols > board.getCols()) {
        return false;
    }

    for (int i = r; i < r + rows; i++) {
        for (int j = c; j < c + cols; j++) {
            if (variant[i - r][j - c] != ' ' && board.getElmt(i, j) != '.') {
                return false;
            }
        }
    }

    return true;
}

private void placePiece(int r, int c, char[][] variant) {
    int rows = variant.length;
    int cols = variant[0].length;

    for (int i = r; i < r + rows; i++) {
        for (int j = c; j < c + cols; j++) {
            if (variant[i - r][j - c] != ' ') {
                board.setElmt(i, j, variant[i - r][j - c]);
            }
        }
    }
}

private void removePiece(int r, int c, char[][] variant) {
    int rows = variant.length;
    int cols = variant[0].length;

    for (int i = r; i < r + rows; i++) {
        for (int j = c; j < c + cols; j++) {
            if (variant[i - r][j - c] != ' ') {
                board.setElmt(i, j, '.');
            }
        }
    }
}

public boolean findSolution(int idx) {
    if (earlyStop) {
        return false;
    }

    if (board.isGridFull() && idx < board.getPieceCount()) {
        earlyStop = true;
        return false;
    }

    if (idx >= board.getPieceCount()) {
        return board.isGridFull();
    }

    Piece currentPiece = board.getPieces().get(idx);
    for (int i = 0; i < board.getRows(); i++) {
        for (int j = 0; j < board.getCols(); j++) {
            if (earlyStop) {
                return false;
            }
            for (char[][] currentVariant : currentPiece.getVariants()) {

```

```

        iterationCount++;
        if (canPlacePiece(i, j, currentVariant)) {
            placePiece(i, j, currentVariant);
            if (findSolution(idx + 1)) {
                return true;
            }
            removePiece(i, j, currentVariant);
        }
    }
}
return false;
}

public boolean solve(int idx) {
    long startTime = System.nanoTime();
    boolean result = findSolution(idx);
    long endTime = System.nanoTime();
    executionTime = (endTime - startTime) / 1_000_000.0;

    return result;
}

public long getIterationCount() {
    return iterationCount;
}

public double getExecutionTime() {
    return executionTime;
}

public Solver(Board board) {
    this.board = board;
    this.iterationCount = 0;
    this.executionTime = 0;
    this.earlyStop = false;
}
}

```

Parser.java

```

package parser;

import java.util.*;
import java.io.*;
import board.Board;
import piece.Piece;

public class Parser {
    public static Board parseFile(String filename) {
        try(BufferedReader br = new BufferedReader(new FileReader(filename))) {
            String line;

            do {
                line = br.readLine();
                if (line == null) {
                    throw new IllegalArgumentException("File does not contain valid
board dimensions and/or piece numbers.");
                }
            }
        }
    }
}

```

```

        line = line.trim();
    } while (line.isEmpty());

    String[] firstLine = line.split("\\s+");
    if (firstLine.length != 3) {
        throw new IllegalArgumentException("First line must have exactly 3
arguments (N, M and P).");
    }

    int N = Integer.parseInt(firstLine[0]);
    int M = Integer.parseInt(firstLine[1]);
    int P = Integer.parseInt(firstLine[2]);

    if (N <= 0 || M <= 0) {
        throw new IllegalArgumentException("Board size (N x M) must be positive
nonzero integers.");
    }

    if (P < 1 || P > 26) {
        throw new IllegalArgumentException("P must be between 1 and 26.");
    }

    do {
        line = br.readLine();
        if (line == null) {
            throw new IllegalArgumentException("File does not contain a valid
board type.");
        }
        line = line.trim();
    } while (line.isEmpty());

    String boardType = line;
    if (!boardType.equals("CUSTOM") && !boardType.equals("DEFAULT")) {
        throw new IllegalArgumentException("Board type must be 'DEFAULT' or
'CUSTOM'.");
    }

    char[][] grid = new char[N][M];
    if (boardType.equals("CUSTOM")) {
        do {
            line = br.readLine();
            if (line == null) {
                throw new IllegalArgumentException("File does not contain a
valid custom board.");
            }
            line = line.trim();
        } while (line.isEmpty());

        if (line == null || line.length() != M) {
            throw new IllegalArgumentException("Invalid CUSTOM board.");
        }

        for (char c : line.toCharArray()) {
            if (c != '.' && c != 'X') {
                throw new IllegalArgumentException("Invalid CUSTOM board.");
            }
        }

        char[] temp = line.toCharArray();
        for (int i = 0; i < M; i++) {
            if (temp[i] == 'X') {
                grid[0][i] = '.';
            } else {
                grid[0][i] = ' ';
            }
        }
    }
}

```

```

        for (int i = 1; i < N; i++) {
            line = br.readLine();
            if (line == null || line.length() != M) {
                throw new IllegalArgumentException("Invalid CUSTOM board.");
            }

            for (char c : line.toCharArray()) {
                if (c != '.' && c != 'X') {
                    throw new IllegalArgumentException("Invalid CUSTOM board.");
                }
            }

            temp = line.toCharArray();
            for (int j = 0; j < M; j++) {
                if (temp[j] == 'X') {
                    grid[i][j] = '.';
                } else {
                    grid[i][j] = ' ';
                }
            }
        }
    } else {
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < M; j++) {
                grid[i][j] = '.';
            }
        }
    }

    Map<Character, List<String>> pieceMap = new LinkedHashMap<>();
    char prevPiece = ' ';
    while ((line = br.readLine()) != null) {
        if (line.isEmpty()) {
            continue;
        }
        char firstChar = ' ';
        for (char c : line.toCharArray()) {
            if (Character.isLetter(c)) {
                firstChar = c;
                break;
            }
        }
        if (firstChar == ' ' || firstChar < 'A' || firstChar > 'Z') {
            throw new IllegalArgumentException("Each pieces must be within A-
Z.");
        }

        for (char c : line.toCharArray()) {
            if (c != firstChar && c != ' ') {
                throw new IllegalArgumentException("Invalid piece found.");
            }
        }

        if (prevPiece != firstChar && pieceMap.containsKey(firstChar)) {
            throw new IllegalArgumentException("Each piece must be
contiguous.");
        }

        pieceMap.putIfAbsent(firstChar, new ArrayList<>());
        pieceMap.get(firstChar).add(line);

        prevPiece = firstChar;
    }

    if (pieceMap.size() != P) {

```

```

        throw new IllegalArgumentException("The number of pieces must be equal
to " + P + ".");
    }

    List<Piece> pieces = new ArrayList<>();
    for (List<String> piece : pieceMap.values()) {
        pieces.add(new Piece(piece));
    }

    Board board = new Board(grid, N, M);
    board.setPieces(pieces, P);
    return board;
} catch (Exception e) {
    throw new RuntimeException(e.getMessage());
}
}
}

```

ImageGenerator.java

```

package image;

import java.awt.*;
import java.awt.image.BufferedImage;
import java.io.*;
import java.util.*;

public class ImageGenerator {
    private char[][] grid;

    private static Map<Character, Color> colorMap = new HashMap<>() {{
        put(' ', new Color(255, 255, 255)); // White (255, 255, 255)
        put('.', new Color(0, 0, 0)); // Black (0, 0, 0)
        put('A', new Color(255, 0, 0)); // Red (255, 0, 0)
        put('B', new Color(0, 255, 0)); // Green (0, 255, 0)
        put('C', new Color(0, 0, 255)); // Blue (0, 0, 255)
        put('D', new Color(255, 255, 0)); // Yellow (255, 255, 0)
        put('E', new Color(255, 0, 255)); // Magenta (255, 0, 255)
        put('F', new Color(0, 255, 255)); // Cyan (0, 255, 255)
        put('G', new Color(128, 0, 0)); // Dark Red (128, 0, 0)
        put('H', new Color(0, 128, 0)); // Dark Green (0, 128, 0)
        put('I', new Color(0, 0, 128)); // Dark Blue (0, 0, 128)
        put('J', new Color(128, 128, 0)); // Olive (128, 128, 0)
        put('K', new Color(128, 0, 128)); // Purple (128, 0, 128)
        put('L', new Color(0, 128, 128)); // Teal (0, 128, 128)
        put('M', new Color(255, 165, 0)); // Orange (255, 165, 0)
        put('N', new Color(75, 0, 130)); // Indigo (75, 0, 130)
        put('O', new Color(139, 69, 19)); // Saddle Brown (139, 69, 19)
        put('P', new Color(255, 192, 203)); // Pink (255, 192, 203)
        put('Q', new Color(173, 216, 230)); // Light Blue (173, 216, 230)
        put('R', new Color(0, 100, 0)); // Dark Green (0, 100, 0)
        put('S', new Color(165, 42, 42)); // Brown (165, 42, 42)
        put('T', new Color(240, 230, 140)); // Khaki (240, 230, 140)
        put('U', new Color(46, 139, 87)); // Sea Green (46, 139, 87)
        put('V', new Color(255, 215, 0)); // Gold (255, 215, 0)
        put('W', new Color(0, 191, 255)); // Deep Sky Blue (0, 191, 255)
        put('X', new Color(138, 43, 226)); // Blue Violet (138, 43, 226)
        put('Y', new Color(127, 255, 212)); // Aquamarine (127, 255, 212)
        put('Z', new Color(218, 112, 214)); // Orchid (218, 112, 214)
    }};

    public BufferedImage generateImage(File file) {

```

```

        int cellSize = 64;
        int rows = grid.length;
        int cols = grid[0].length;
        int height = rows * cellSize;
        int width = cols * cellSize;

        BufferedImage image = new BufferedImage(width, height,
BufferedImage.TYPE_INT_RGB);
        Graphics2D graphics = image.createGraphics();

        graphics.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);

        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                char elmt = grid[i][j];
                graphics.setColor(colorMap.getOrDefault(elmt, Color.WHITE));
                graphics.fillRect(cellSize * j, cellSize * i, cellSize, cellSize);

                graphics.setColor(Color.BLACK);
                graphics.drawRect(cellSize * j, cellSize * i, cellSize, cellSize);
                if (elmt != ' ' && elmt != '.') {
                    graphics.setFont(new Font("Monospaced", Font.BOLD, 20));
                    FontMetrics metrics = graphics.getFontMetrics();

                    int textWidth = metrics.stringWidth(String.valueOf(elmt));
                    int textHeight = metrics.getAscent();

                    int x = (cellSize * j) + (cellSize - textWidth) / 2;
                    int y = (cellSize * i) + ((cellSize - textHeight) / 2) + textHeight;

                    graphics.setColor(Color.BLACK);
                    graphics.drawString(String.valueOf(elmt), x, y);
                }
            }
        }

        graphics.dispose();

        return image;
    }
    public ImageGenerator(char[][] grid) {
        this.grid = grid;
    }
}

```

Main.java

```

import java.awt.image.BufferedImage;
import java.io.*;
import java.util.*;
import javax.imageio.ImageIO;
import parser.Parser;
import board.Board;
import solver.Solver;
import image.ImageGenerator;

public class Main {
    public static void main(String[] args) {
        if (args.length != 1) {
            System.out.println("Usage: java Main <file>");
        }
    }
}

```

```

        return;
    }

    String filename = "../input/" + args[0];

    try {
        Board board = Parser.parseFile(filename);
        Solver solver = new Solver(board);
        boolean solution = solver.solve(0);

        if (solution) {
            System.out.println("Solution found:");
            board.displayBoard();
        } else {
            System.out.println("No solution found:");
            board.resetGrid();
            board.displayBoard();
        }
        System.out.println("Execution time: " + solver.getExecutionTime() + " ms");
        System.out.println("Cases iterated: " + solver.getIterationCount());

        Scanner scanner = new Scanner(System.in);
        String ans;

        while (true) {
            System.out.print("Would you like to save the solution? (yes or no): ");
            ans = scanner.nextLine().trim().toLowerCase();

            if (ans.equals("yes") || ans.equals("no")) {
                break;
            }
            System.out.println("Please enter 'yes' or 'no'.");
        }

        if (ans.equals("yes")) {
            String outputFileName;

            while (true) {
                System.out.print("Enter the filename: ");
                outputFileName = scanner.nextLine().trim();

                if (isValidFilename(outputFileName)) {
                    break;
                }
                System.out.println("Avoid the following characters: \\ / : * ? \\" > |");
            }

            File testDirectory = new File("../test");
            if (!testDirectory.exists()) {
                testDirectory.mkdir();
            }

            File outputText = new File(testDirectory, outputFileName + ".txt");
            try(BufferedWriter bw = new BufferedWriter(new FileWriter(outputText)))
            {
                for (int i = 0; i < board.getRows(); i++) {
                    for (int j = 0; j < board.getCols(); j++) {
                        if (board.getElmt(i, j) == ' ') {
                            bw.write('.');
                        } else if (board.getElmt(i, j) == '.') {
                            bw.write('?');
                        } else {
                            bw.write(board.getElmt(i, j));
                        }
                    }
                }
            }
        }
    }

```



```

        bw.newLine();
    }
    bw.write("Execution time: " + solver.getExecutionTime() + " ms");
    bw.newLine();
    bw.write("Cases iterated: " + solver.getIterationCount());

    System.out.println("The solution is saved at: " +
outputText.getAbsolutePath());
    } catch (IOException e) {
        System.out.println("Failed to save the solution.");
    }

    File outputImage = new File(testDirectory, outputFileName + ".png");
    ImageGenerator imageGenerator = new ImageGenerator(board.getGrid());

    BufferedImage image = imageGenerator.generateImage(outputImage);
    try {
        ImageIO.write(image, "png", outputImage);
        System.out.println("The image is saved at: " +
outputText.getAbsolutePath());
    } catch (IOException e) {
        System.out.println("Failed to save the image.");
    }
}

scanner.close();

} catch (Exception e) {
    System.out.println("Error: " + e.getMessage());
}
}

private static boolean isValidFilename(String filename) {
    return filename.matches("^[^\\\\\\/:*?\"<>|]+$");
}
}

```

MainGUI.java

```

import java.awt.*;
import java.awt.datatransfer.*;
import java.awt.dnd.*;
import java.awt.image.BufferedImage;
import java.io.*;
import javax.imageio.ImageIO;
import javax.swing.*;
import parser.Parser;
import board.Board;
import solver.Solver;
import image.ImageGenerator;
import java.util.List;

public class MainGUI {
    private JFrame frame;
    private JTextArea fileContentArea;
    private JTextArea resultTextArea;
    private JLabel imageLabel;
    private JButton saveTxtButton;
    private JButton savePngButton;
    private File selectedFile;
    private Board board;
    private Solver solver;
}

```

```

private BufferedImage generatedImage;

public MainGUI() {
    frame = new JFrame("IQ Puzzle Pro Solver - 13523099");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(900, 600);
    frame.setLayout(new BorderLayout());

    JPanel leftPanel = new JPanel(new BorderLayout());
    JButton uploadButton = new JButton("Upload File");
    fileContentArea = new JTextArea(10, 20);
    fileContentArea.setEditable(false);
    JScrollPane fileScrollPane = new JScrollPane(fileContentArea);
    leftPanel.add(fileScrollPane, BorderLayout.CENTER);
    leftPanel.add(uploadButton, BorderLayout.SOUTH);
    new DropTarget(fileContentArea, new FileDropHandler());

    JPanel centerPanel = new JPanel(new BorderLayout());
    imageLabel = new JLabel("", SwingConstants.CENTER);
    centerPanel.add(imageLabel, BorderLayout.CENTER);

    JPanel rightPanel = new JPanel(new BorderLayout());

    JPanel resultPanel = new JPanel(new BorderLayout());
    resultTextArea = new JTextArea("The result will be displayed here");
    resultTextArea.setEditable(false);
    resultPanel.add(new JScrollPane(resultTextArea), BorderLayout.CENTER);

    JPanel buttonPanel = new JPanel(new GridLayout(1, 3));
    JButton runButton = new JButton("Solve");
    saveTxtButton = new JButton("Save as TXT");
    savePngButton = new JButton("Save as PNG");
    saveTxtButton.setEnabled(false);
    savePngButton.setEnabled(false);

    buttonPanel.add(runButton);
    buttonPanel.add(saveTxtButton);
    buttonPanel.add(savePngButton);

    rightPanel.add(resultPanel, BorderLayout.NORTH);
    rightPanel.add(buttonPanel, BorderLayout.SOUTH);

    frame.add(leftPanel, BorderLayout.WEST);
    frame.add(centerPanel, BorderLayout.CENTER);
    frame.add(rightPanel, BorderLayout.EAST);

    uploadButton.addActionListener(_ -> uploadFile());
    runButton.addActionListener(_ -> runSolver());
    saveTxtButton.addActionListener(_ -> saveAsText());
    savePngButton.addActionListener(_ -> saveAsImage());

    frame.setVisible(true);
}

private void uploadFile() {
    JFileChooser fileChooser = new JFileChooser(new File("../input"));
    fileChooser.setDialogTitle("Select Input File");
    fileChooser.setFileFilter(new
java.swing.filechooser.FileNameExtensionFilter("Text Files", "txt"));

    if (fileChooser.showOpenDialog(frame) == JFileChooser.APPROVE_OPTION) {
        handleFileSelection(fileChooser.getSelectedFile());
    }
}

private void handleFileSelection(File file) {

```

```

        if (file == null || !file.getName().endsWith(".txt")) {
            fileContentArea.setText("Invalid file. Please upload a .txt file.");
            return;
        }

        selectedFile = file;
        fileContentArea.setText("");
        resultTextArea.setText("The result will be displayed here");
        imageLabel.setIcon(null);
        saveTxtButton.setEnabled(false);
        savePngButton.setEnabled(false);

        try (BufferedReader br = new BufferedReader(new FileReader(selectedFile))) {
            String line;
            while ((line = br.readLine()) != null) {
                fileContentArea.append(line + "\n");
            }
        } catch (IOException e) {
            fileContentArea.setText("Failed to read file.");
        }
    }

    private void runSolver() {
        if (selectedFile == null) {
            resultTextArea.setText("Upload a file first.");
            return;
        }

        try {
            board = Parser.parseFile(selectedFile.getAbsolutePath());
            solver = new Solver(board);
            boolean solution = solver.solve(0);

            if (solution) {
                resultTextArea.setText("Solution found:\nExecution time: " +
                    solver.getExecutionTime() + " ms\nCases iterated: " + solver.getIterationCount());
            } else {
                resultTextArea.setText("No solution found:\nExecution time: " +
                    solver.getExecutionTime() + " ms\nCases iterated: " + solver.getIterationCount());
                board.resetGrid();
            }

            generateImage();
            displayImage();
            saveTxtButton.setEnabled(true);
            savePngButton.setEnabled(true);

        } catch (Exception e) {
            resultTextArea.setText("Error: " + e.getMessage());
        }
    }

    private void generateImage() {
        try {
            ImageGenerator imageGenerator = new ImageGenerator(board.getGrid());
            generatedImage = imageGenerator.generateImage(null);
        } catch (Exception e) {
            resultTextArea.setText("Failed to generate image.");
        }
    }

    private void displayImage() {
        if (generatedImage != null) {
            imageLabel.setIcon(new ImageIcon(generatedImage));
        }
    }
}

```

```

private void saveAsText() {
    JFileChooser fileChooser = new JFileChooser(new File("../test"));
    fileChooser.setDialogTitle("Save as TXT");
    fileChooser.setFileFilter(new
javax.swing.filechooser.FileNameExtensionFilter("Text Files", "txt"));

    if (fileChooser.showSaveDialog(frame) == JFileChooser.APPROVE_OPTION) {
        File file = new File(fileChooser.getSelectedFile() + ".txt");
        try (BufferedWriter bw = new BufferedWriter(new FileWriter(file))) {
            for (int i = 0; i < board.getRows(); i++) {
                for (int j = 0; j < board.getCols(); j++) {
                    if (board.getElmt(i, j) == ' ') {
                        bw.write('.');
                    } else if (board.getElmt(i, j) == '.') {
                        bw.write('?');
                    } else {
                        bw.write(board.getElmt(i, j));
                    }
                }
                bw.newLine();
            }

            bw.write("Execution time: " + solver.getExecutionTime() + " ms");
            bw.newLine();
            bw.write("Cases iterated: " + solver.getIterationCount());

            resultTextArea.setText("The solution is saved at: " +
file.getAbsolutePath());
        } catch (IOException e) {
            resultTextArea.setText("Failed to save the solution.");
        }
    }
}

private void saveAsImage() {
    if (generatedImage == null) {
        resultTextArea.setText("No image to save.");
        return;
    }

    JFileChooser fileChooser = new JFileChooser(new File("../test"));
    fileChooser.setDialogTitle("Save as PNG");
    fileChooser.setFileFilter(new
javax.swing.filechooser.FileNameExtensionFilter("PNG Images", "png"));

    if (fileChooser.showSaveDialog(frame) == JFileChooser.APPROVE_OPTION) {
        File outputImage = new File(fileChooser.getSelectedFile() + ".png");

        try {
            ImageIO.write(generatedImage, "png", outputImage);
            resultTextArea.setText("The image is saved at: " +
outputImage.getAbsolutePath());
        } catch (IOException e) {
            resultTextArea.setText("Failed to save the image.");
        }
    }
}

private class FileDropHandler extends DropTargetAdapter {
    @Override
    public void drop(DropTargetDropEvent event) {
        try {
            event.acceptDrop(DnDConstants.ACTION_COPY);
            Transferable transferable = event.getTransferable();

```

```
        Object data =
transferable.getTransferData(DataFlavor.javaFileListFlavor);
        if (data instanceof List<?>) {
            List<?> genericList = (List<?>) data;
            if (!genericList.isEmpty() && genericList.get(0) instanceof File) {
                handleFileSelection((File) genericList.get(0));
            }
        }
    } catch (Exception ex) {
        fileContentArea.setText("Drag and drop failed.");
    }
}

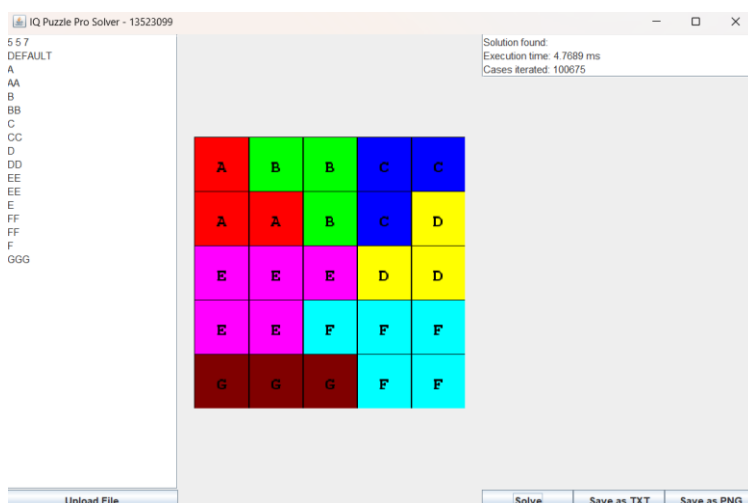
public static void main(String[] args) {
    SwingUtilities.invokeLater(MainGUI::new);
}
}
```

BAB III: EKSPERIMEN

Test Case 1: DEFAULT (memiliki solusi)

Input	Output																									
5 5 7 DEFAULT A AA B BB C CC D DD EE EE E FF FF F GGG	<div>Solution found: ABBCC AABCD EEEDD EEFFF GGGFF Execution time: 4.8016 ms Cases iterated: 100675 Would you like to save the solution? (yes or no): <input type="text"/></div> <table><tr><td>A</td><td>B</td><td>B</td><td>C</td><td>C</td></tr><tr><td>A</td><td>A</td><td>B</td><td>C</td><td>D</td></tr><tr><td>E</td><td>E</td><td>E</td><td>D</td><td>D</td></tr><tr><td>E</td><td>E</td><td>F</td><td>F</td><td>F</td></tr><tr><td>G</td><td>G</td><td>G</td><td>F</td><td>F</td></tr></table>	A	B	B	C	C	A	A	B	C	D	E	E	E	D	D	E	E	F	F	F	G	G	G	F	F
A	B	B	C	C																						
A	A	B	C	D																						
E	E	E	D	D																						
E	E	F	F	F																						
G	G	G	F	F																						

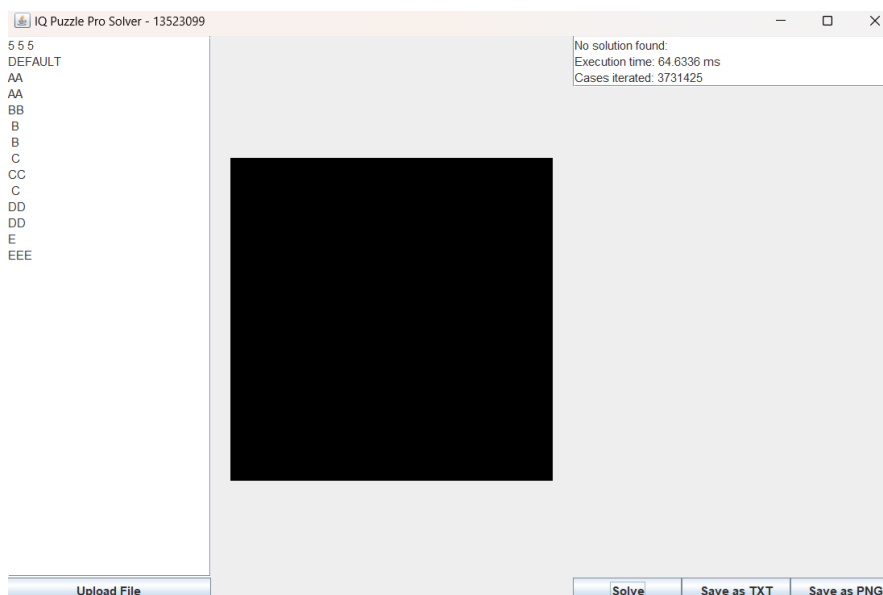
Tampilan GUI:



Test Case 2: DEFAULT (tidak memiliki solusi)

Input	Output
5 5 5 DEFAULT AA AA BB B B C CC C DD DD E EEE	<pre> No solution found: ????? ????? ????? ????? ????? ????? Execution time: 68.4283 ms Cases iterated: 3731425 Would you like to save the solution? (yes or no): </pre>

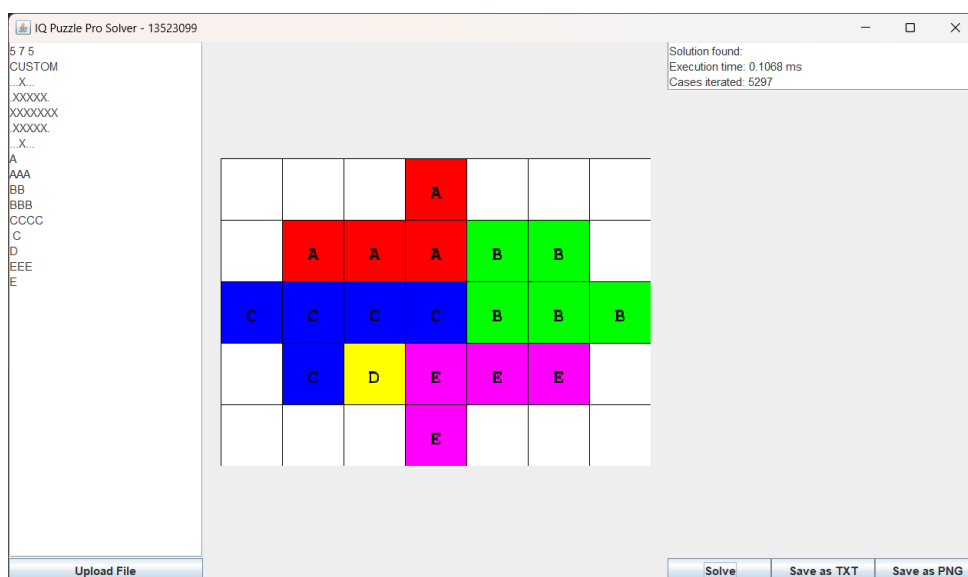
Tampilan GUI:



Test Case 3: CUSTOM (memiliki solusi)

Input	Output																																			
5 7 5 CUSTOM ...X... .XXXXX. XXXXXXXX .XXXXX. ...X... A AAA BB BBB CCCC C D EEE E	<div>Solution found: ...A... .AAABBB. CCCCBBB .CDEEE. ...E... Execution time: 0.8852 ms Cases iterated: 5297 Would you like to save the solution? (yes or no): <input type="checkbox"/></div> <table><tr><td></td><td></td><td></td><td>A</td><td></td><td></td><td></td></tr><tr><td></td><td>A</td><td>A</td><td>A</td><td>B</td><td>B</td><td></td></tr><tr><td>C</td><td>C</td><td>C</td><td>C</td><td>B</td><td>B</td><td>B</td></tr><tr><td></td><td>C</td><td>D</td><td>E</td><td>E</td><td>E</td><td></td></tr><tr><td></td><td></td><td></td><td>E</td><td></td><td></td><td></td></tr></table>				A					A	A	A	B	B		C	C	C	C	B	B	B		C	D	E	E	E					E			
			A																																	
	A	A	A	B	B																															
C	C	C	C	B	B	B																														
	C	D	E	E	E																															
			E																																	

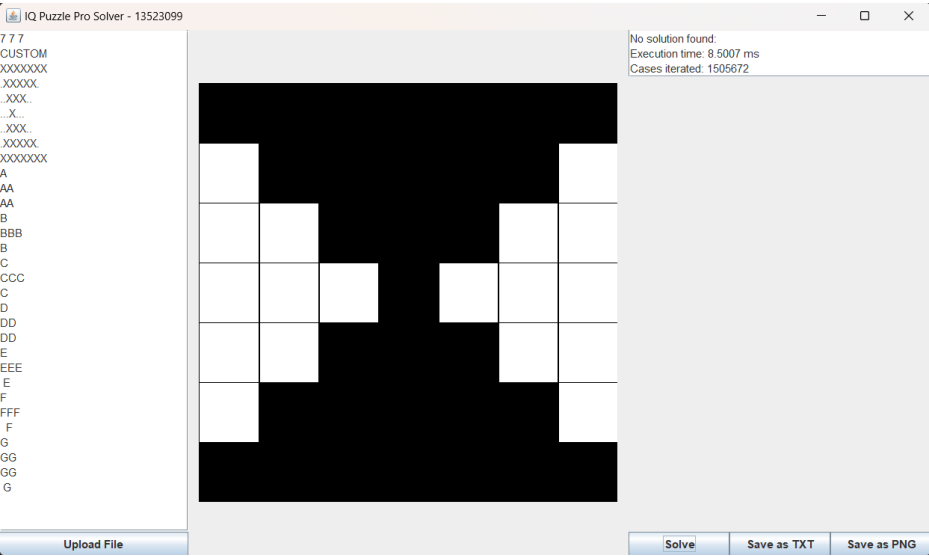
Tampilan GUI:



Test Case 4: CUSTOM (tidak memiliki solusi)

Input	Output
7 7 7 CUSTOM XXXXXXXX .XXXXX. ..XXX.. ...X... ..XXX.. .XXXXX. XXXXXXXX A AA AA B BBB B C CCC C D DD DD E EEE E F FFF F G GG GG G	<pre> No solution found: ??????? .?????. ..???.. ...?... ..???.. .?????. ??????? Execution time: 22.3634 ms Cases iterated: 1505672 Would you like to save the solution? (yes or no): </pre>

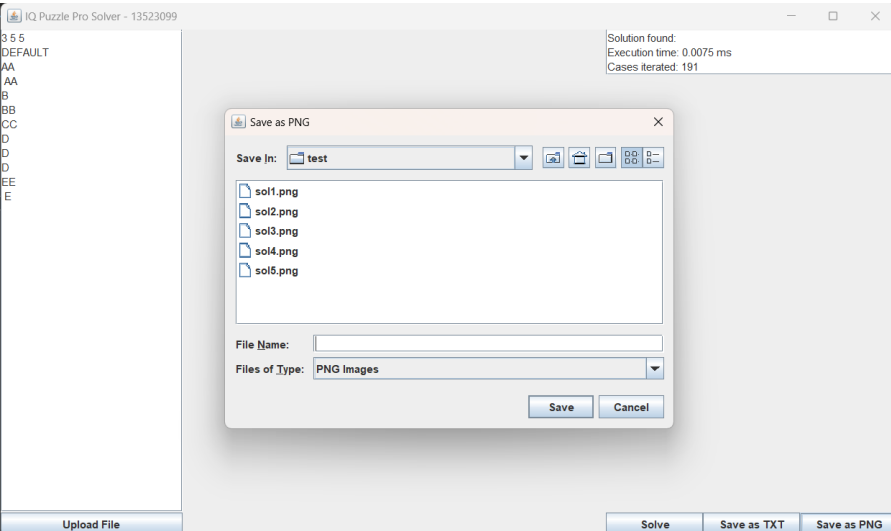
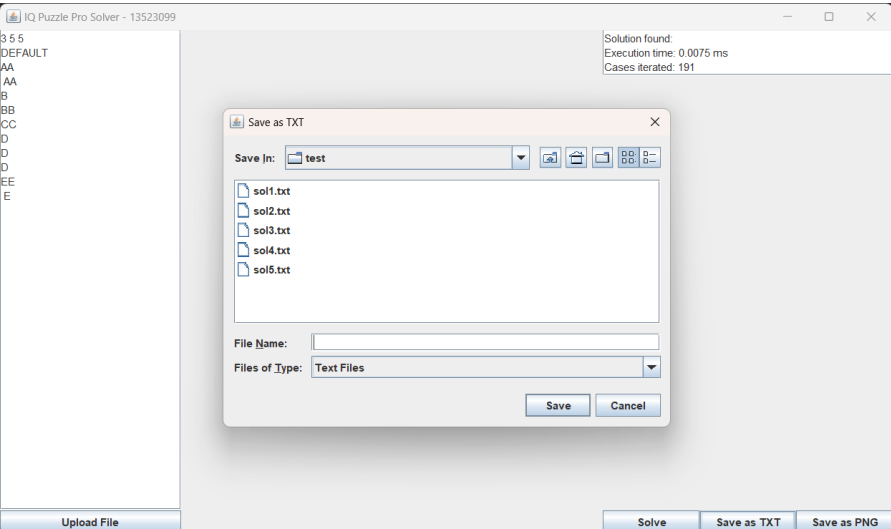
Tampilan GUI:



Test Case 5: SAVETOFILE/DEFAULT (memiliki solusi)

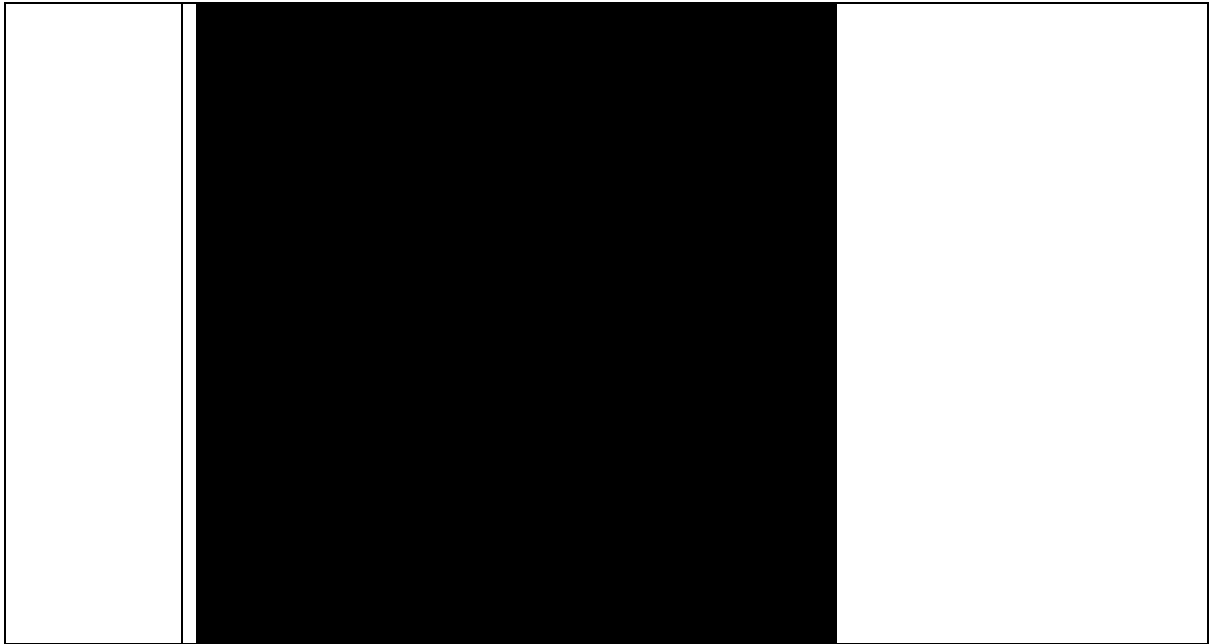
Input	Output															
3 5 5 DEFAULT AA AA B BB CC D D D EE E	<div>Solution found: AABBC EAABC EEDDD Execution time: 0.0792 ms Cases iterated: 191 Would you like to save the solution? (yes or no): yes Enter the filename: sol5 The solution is saved at: C:\Users\ASUS\Desktop\Project\Tugas-Kecil-1-Stima-IF2211\bin\..\test\sol5.txt The image is saved at: C:\Users\ASUS\Desktop\Project\Tugas-Kecil-1-Stima-IF2211\bin\..\test\sol5.txt</div> <div>AABBC EAABC EEDDD Execution time: 0.0792 ms Cases iterated: 191</div> <table><tr><td>A</td><td>A</td><td>B</td><td>B</td><td>C</td></tr><tr><td>E</td><td>A</td><td>A</td><td>B</td><td>C</td></tr><tr><td>E</td><td>E</td><td>D</td><td>D</td><td>D</td></tr></table>	A	A	B	B	C	E	A	A	B	C	E	E	D	D	D
A	A	B	B	C												
E	A	A	B	C												
E	E	D	D	D												

Tampilan GUI:

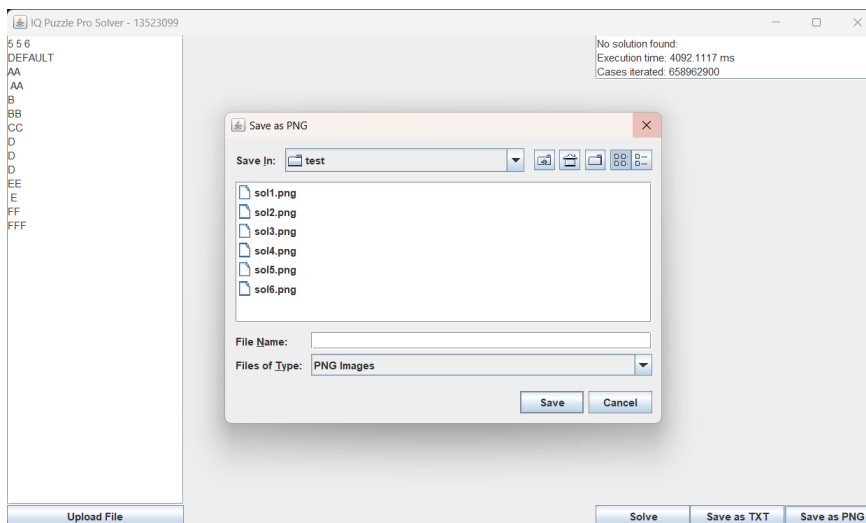
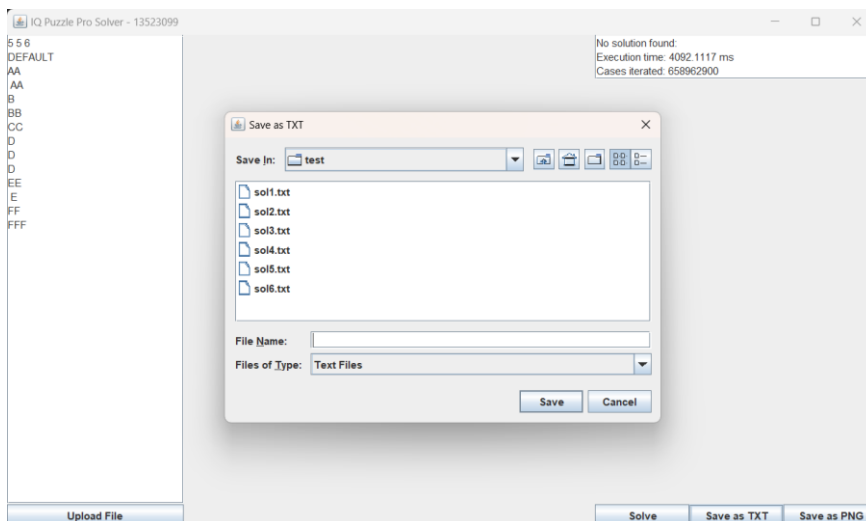


Test Case 6: SAVETOFILE/DEFAULT (tidak memiliki solusi)

Input	Output
5 5 6 DEFAULT AA AA B BB CC D D D EE E FF FFF	<div>No solution found: ????? ????? ????? ????? ????? Execution time: 3913.6104 ms Cases iterated: 658962900 Would you like to save the solution? (yes or no): yes Enter the filename: sol6 The solution is saved at: C:\Users\ASUS\Desktop\Project\Tugas-Kecil-1-Stima-IF2211\bin\..\test\sol6.txt The image is saved at: C:\Users\ASUS\Desktop\Project\Tugas-Kecil-1-Stima-IF2211\bin\..\test\sol6.txt</div> <div>????? ????? ????? ????? ????? ????? Execution time: 3913.6104 ms Cases iterated: 658962900</div>



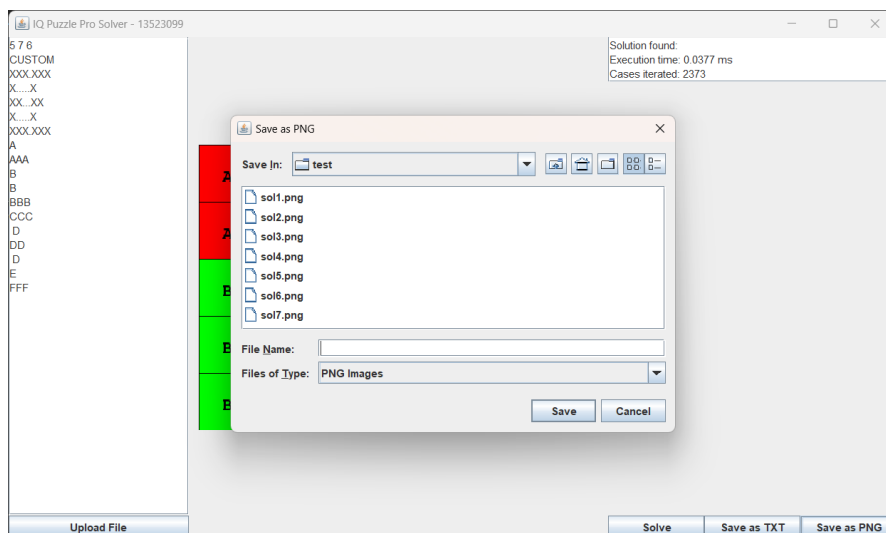
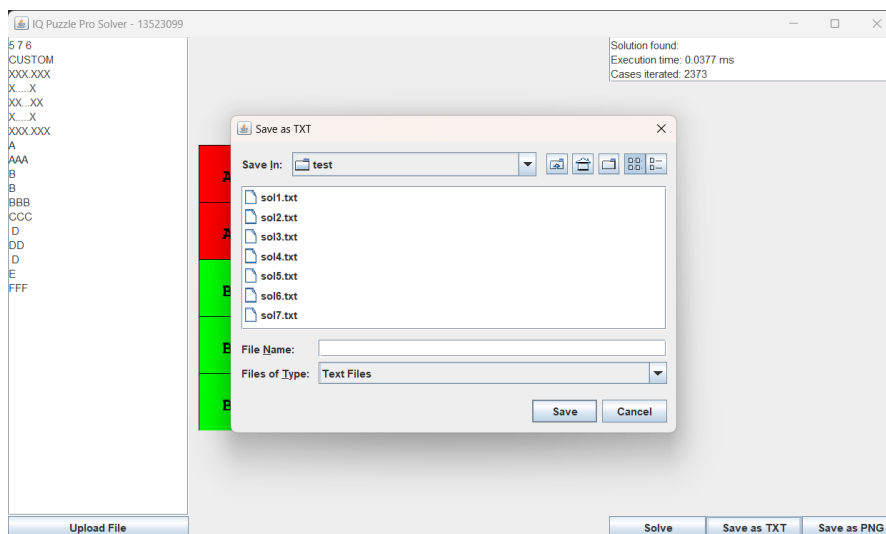
Tampilan GUI:



Test Case 7: SAVETOFILE/CUSTOM (memiliki solusi)

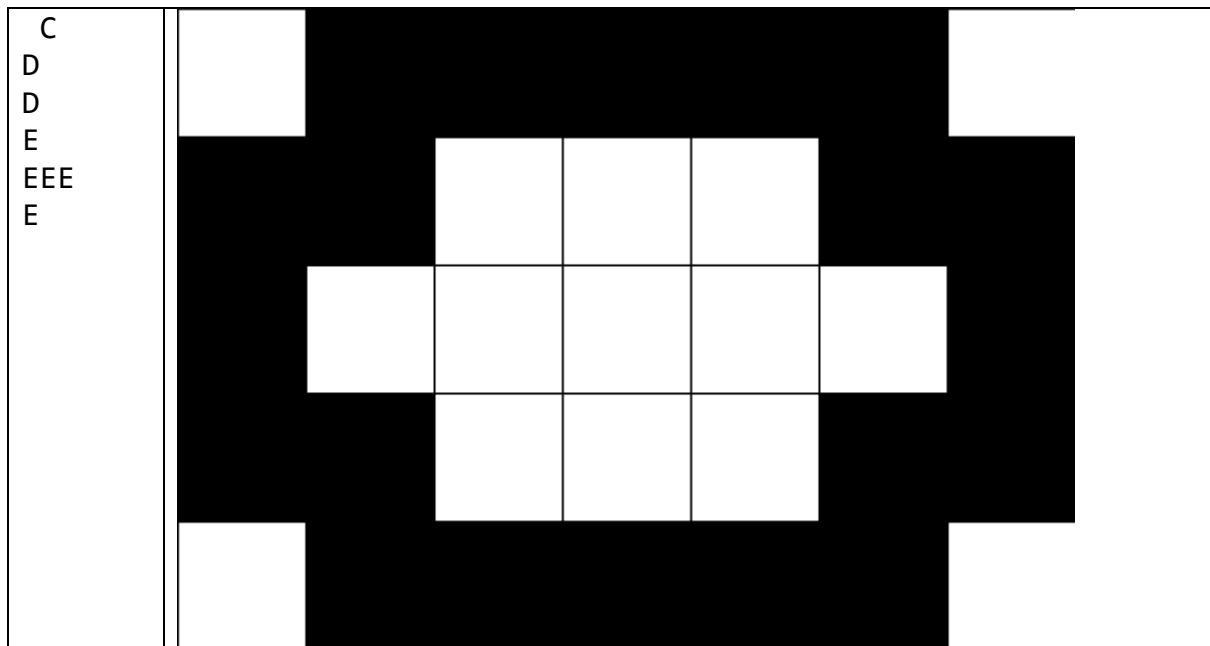
Input	Output																																			
5 7 6 CUSTOM XXX.XXX X.....X XX...XX X.....X XXX.XXX A AAA B B BBB CCC D DD D E FFF	<div>Solution found: AAA.CCC A.....D BE...DD B.....D BBB.FFF Execution time: 0.4946 ms Cases iterated: 2373 Would you like to save the solution? (yes or no): yes Enter the filename: sol7 The solution is saved at: C:\Users\ASUS\Desktop\Project\Tugas-Kecil-1-Stima-IF2211\bin\..\test\sol7.txt The image is saved at: C:\Users\ASUS\Desktop\Project\Tugas-Kecil-1-Stima-IF2211\bin\..\test\sol7.txt</div> <div>AAA.CCC A.....D BE...DD B.....D BBB.FFF Execution time: 0.4946 ms Cases iterated: 2373</div> <table><tr><td>A</td><td>A</td><td>A</td><td></td><td>C</td><td>C</td><td>C</td></tr><tr><td>A</td><td></td><td></td><td></td><td></td><td></td><td>D</td></tr><tr><td>B</td><td>E</td><td></td><td></td><td></td><td>D</td><td>D</td></tr><tr><td>B</td><td></td><td></td><td></td><td></td><td></td><td>D</td></tr><tr><td>B</td><td>B</td><td>B</td><td></td><td>F</td><td>F</td><td>F</td></tr></table>	A	A	A		C	C	C	A						D	B	E				D	D	B						D	B	B	B		F	F	F
A	A	A		C	C	C																														
A						D																														
B	E				D	D																														
B						D																														
B	B	B		F	F	F																														

Tampilan GUI:

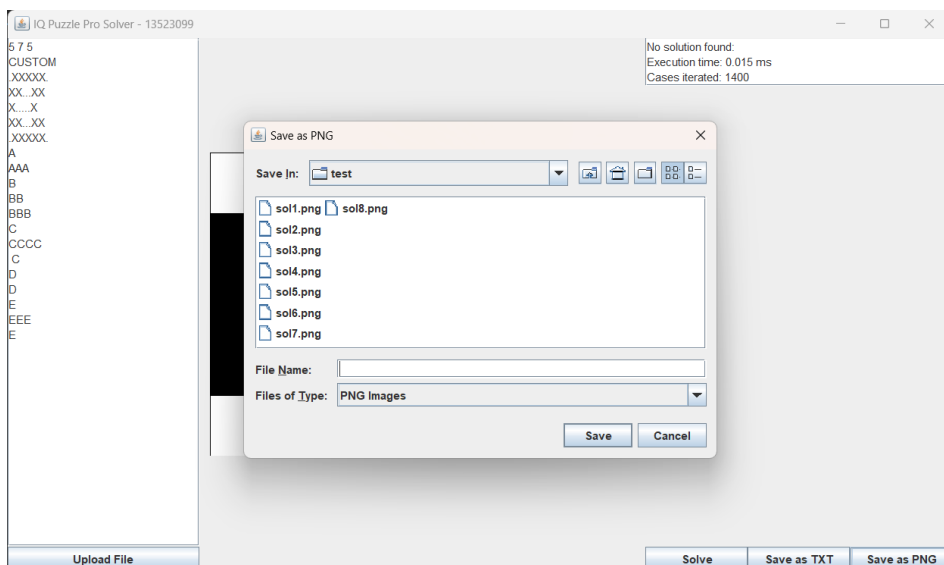
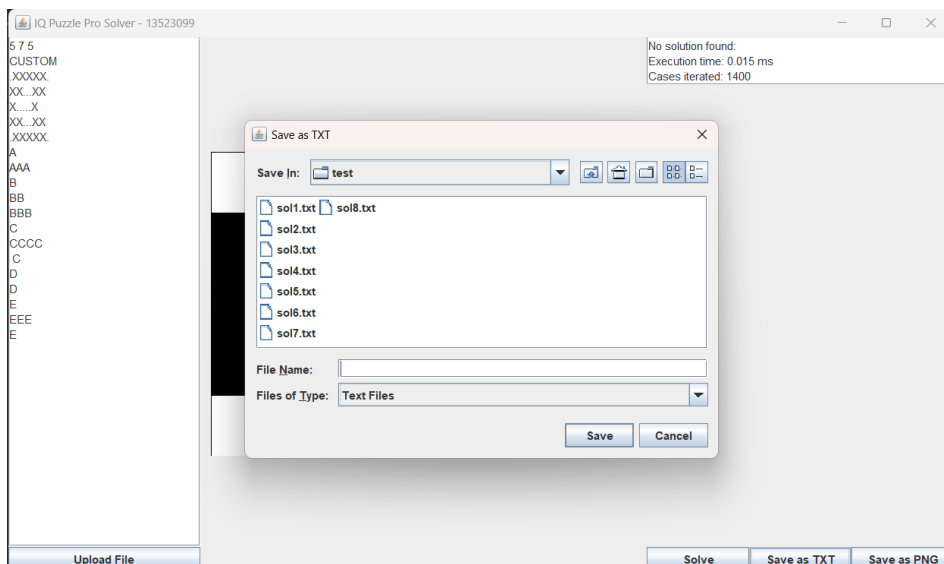


Test Case 8: SAVETOFILE/CUSTOM (tidak memiliki solusi)

Input	Output
5 7 5 CUSTOM .XXXXX. XX...XX X.....X XX...XX .XXXXX.	No solution found: ..????? ??...?? ?......? ??...?? ..????? Execution time: 0.3359 ms Cases iterated: 1400 Would you like to save the solution? (yes or no): yes Enter the filename: sol8 The solution is saved at: C:\Users\ASUS\Desktop\Project\Tugas-Kecil-1-Stima-IF2211\bin\..\test\sol8.txt The image is saved at: C:\Users\ASUS\Desktop\Project\Tugas-Kecil-1-Stima-IF2211\bin\..\test\sol8.txt
A AAA B BB BBB C CCCC	..????? ??...?? ?......? ??...?? ..????? Execution time: 0.3359 ms Cases iterated: 1400



Tampilan GUI:



Error Handling:

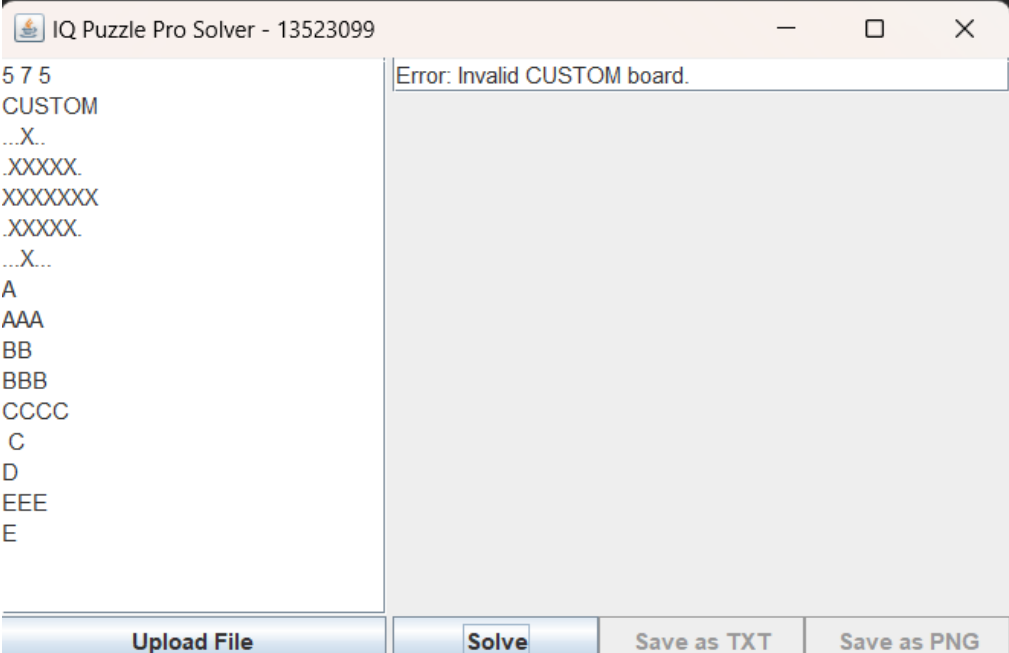
Test Case 9: Invalid Arguments

Input	Output
5 5 DEFAULT A AA B BB C CC D DD EE EE E FF FF F GGG	<div> PS C:\Users\ASUS\Desktop\Project\Tugas-Kecil-1-Stima-IF2211\bin> java Main test9.txt Error: First line must have exactly 3 arguments (N, M and P). </div> <div> <div> <div> IQ Puzzle Pro Solver - 13523099 </div> <div> 5 5 DEFAULT A AA B BB C CC D DD EE EE E FF FF F GGG </div> <div> Error: First line must have exactly 3 arguments (N, M and P). </div> </div> <div> Upload File Solve Save as TXT Save as PNG </div> </div>

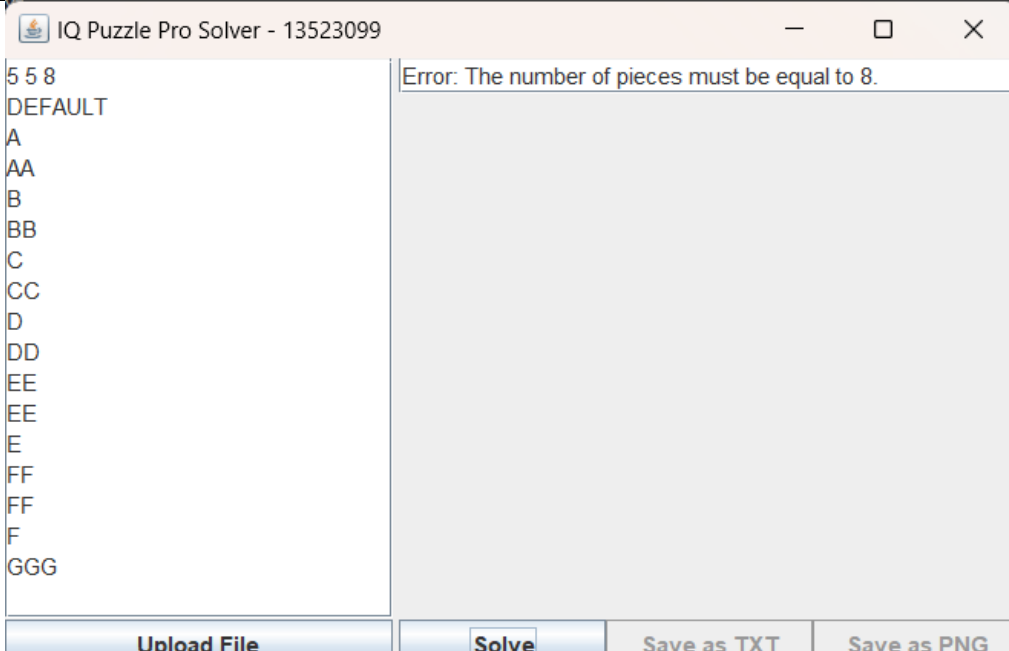
Test Case 10: Invalid Board Dimensions

Input	Output
0 -1 7 DEFAULT A AA B BB C CC D DD EE EE E FF FF F GGG	<div> PS C:\Users\ASUS\Desktop\Project\Tugas-Kecil-1-Stima-IF2211\bin> java Main test10.txt Error: Board size (N x M) must be positive nonzero integers. </div> <div> <div> <div> IQ Puzzle Pro Solver - 13523099 </div> <div> 0 -1 7 DEFAULT A AA B BB C CC D DD EE EE E FF FF F GGG </div> <div> Error: Board size (N x M) must be positive nonzero integers. </div> </div> <div> Upload File Solve Save as TXT Save as PNG </div> </div>

Test Case 11: Invalid CUSTOM Board

Input	Output
5 7 5 CUSTOM ...X.. .XXXXX. XXXXXXXX .XXXXX. ...X... A AAA BB BBB CCCC C D EEE E	<pre>PS C:\Users\ASUS\Desktop\Project\Tugas-Kecil-1-Stima-IF2211\bin> java Main test11.txt Error: Invalid CUSTOM board.</pre> 

Test Case 12: Invalid Number of Pieces

Input	Output
5 5 8 DEFAULT A AA B BB C CC D DD EE EE E FF FF F GGG	<pre>PS C:\Users\ASUS\Desktop\Project\Tugas-Kecil-1-Stima-IF2211\bin> java Main test12.txt Error: The number of pieces must be equal to 8.</pre> 

Test Case 13: Non-Contiguous Pieces

Input	Output
5 5 7 DEFAULT A AA B BB C CC D DD EE FF FF F GGG EE E	<div><div>PS C:\Users\ASUS\Desktop\Project\Tugas-Kecil-1-Stima-IF2211\bin> java Main test13.txt</div><div>Error: Each piece must be contiguous.</div></div> <div><div>IQ Puzzle Pro Solver - 13523099</div><div><div>5 5 7 DEFAULT A AA B BB C CC D DD EE EE FF FF F GGG EE E</div><div>Error: Each piece must be contiguous.</div></div><div><div>Upload File</div><div>Solve</div><div>Save as TXT</div><div>Save as PNG</div></div></div>

BAB IV:
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 <i>Graphical User Interface</i> (GUI)	✓	
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	✓	

Source code dari program ini tersimpan dalam repositori yang dapat diakses melalui tautan:
https://github.com/DanielDPW/Tucil1_13523099