

LAPORAN TUGAS KECIL
IF2211 Strategi Algoritma

Penyelesaian IQ Puzzler Pro dengan Algoritma Brute Force



Disusun oleh:

Andrew Tedjapratama (13523148)

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2025

DAFTAR ISI

DAFTAR ISI	1
BAB I	2
DESKRIPSI MASALAH	2
BAB II	3
SPESIFIKASI TUGAS	3
BAB III	5
TEORI SINGKAT	5
BAB IV	6
DESKRIPSI ALGORITMA	6
4.1 Inisialisasi Data	6
4.2 Proses Pencarian Solusi dengan Algoritma Brute Force	8
4.3 Penyelesaian dan Pengolahan Hasil	11
BAB V	12
IMPLEMENTASI BONUS	12
5.1 Graphical User Interface (GUI)	12
5.2 Output sebagai Gambar	13
BAB VI	14
PENGETESAN PROGRAM	14
Test Case 1	14
Test Case 2	15
Test Case 3	16
Test Case 4	17
Test Case 5	18
Test Case 6	19
Test Case 7	20
Test Case 8	21
Test Case 9	22
BAB VII	23
KESIMPULAN DAN SARAN	23
7.1 Kesimpulan	23
7.1 Saran	23
BAB VIII	24
LAMPIRAN	24
Lampiran	24
a. Link Repository GitHub:	24
b. Tabel Hasil	24

BAB I

DESKRIPSI MASALAH

IQ Puzzler Pro adalah sebuah permainan puzzle di sebuah papan yang diproduksi oleh perusahaan Smart Games. Permainan ini bertujuan agar pemain dapat mengisi papan dengan *piece* (blok puzzle) yang tersedia. Pada kasus atau masalah tugas ini, papan permainan awalnya kosong dan semua blok puzzle harus digunakan untuk menyelesaikan puzzle.

Komponen utama dari permainan IQ Puzzler Pro terdiri dari papan (*board*) sebagai area yang harus diisi dan blok puzzle (*pieces*) dengan yang dapat dirotasi atau dicerminkan sedemikian mungkin sehingga memungkinkan banyak orientasi unik dari blok puzzle tersebut. Oleh karena itu, tantangan utama dari permainan ini adalah menemukan konfigurasi yang tepat agar seluruh area papan terisi penuh dengan jumlah blok-blok yang telah disediakan. Suatu aturan penting dalam permainan ini adalah semua blok harus digunakan untuk menyelesaikan puzzle.

Tugas kecil 1 dari mata kuliah Strategi Algoritma ini meminta sebuah penyelesaian masalah dengan membuat suatu program IQ Puzzler Pro menggunakan algoritma *brute force* untuk menemukan satu solusi atau menyatakan tidak ada solusi. Permainan IQ Puzzler Pro bisa memiliki variasi dari kasus konfigurasi, yaitu antara DEFAULT, CUSTOM, atau PYRAMID yang menambah kompleksitas. Selain bentuk konfigurasi, ketika ukuran papan atau jumlah blok meningkat, waktu eksekusi juga dapat menjadi sangat lama.

Penyelesaian IQ Puzzler Pro ini menunjukkan relevansi strategi algoritma dan ilmu komputer dalam mengatasi masalah logika sehari-hari, bahkan dalam permainan puzzle yang sering ditemui dalam kehidupan nyata maupun dalam *games* di berbagai aplikasi. Algoritma *brute force* termasuk salah satu algoritma yang meskipun memakan waktu, termasuk salah satu metode yang terjamin karena mencoba setiap kombinasi yang mungkin hingga menemukan kombinasi yang benar, membuatnya efektif bahkan terhadap kombinasi rumit.

BAB II

SPESIFIKASI TUGAS

- Buatlah program sederhana dalam bahasa Java yang mengimplementasikan algoritma *Brute Force* yang harus bersifat “murni” untuk menemukan solusi tanpa memanfaatkan *heuristik*.
- Papan yang perlu diisi mulanya akan selalu kosong. Sebuah blok puzzle bisa aja dirotasi maupun dicerminkan sebelum diletakkan pada papan.

- **Input:**

Program akan membaca file *test case* yang berisi :

1. Baris pertama berisi papan ($N \times M$) terdiri atas dua buah variabel baris (N) dan kolom (M), diikuti dengan jumlah blok puzzle (P)
2. Baris kedua merupakan variabel string tipe konfigurasi papan (S) yang hanya bernilai salah satu diantara DEFAULT/CUSTOM/PYRAMID.
3. Baris ketiga dan selanjutnya merupakan blok puzzle dilambangkan oleh konfigurasi *Character* berupa huruf $A - Z$ dalam kapital sebanyak P buah huruf berbeda.

Untuk lebih jelas, file input *test case* berekstensi .txt yang memiliki format sebagai berikut:

```
N M P
S
puzzle_1_shape
puzzle_2_shape
...
```

Contoh Test case:

```
5 5 7
DEFAULT
A
AA
B
BB
C
CC
D
```

```
DD
EE
EE
E
FF
FF
F
GGG
```

Contoh output:

```
AGGGD
AABDD
CCBBF
CEEFF
EEEFF
```

Waktu pencarian: 600 ms

Banyak kasus yang ditinjau: 7123

Apakah anda ingin menyimpan solusi? (ya/tidak)

Pada output program di atas, dapat dilihat tampilan konfigurasi blok puzzle yang berhasil mengisi papan dengan setiap blok memiliki warna yang berbeda untuk menunjukan blok puzzle dengan jelas. Setelah itu, ditampilkan juga waktu pencarian atau *runtime* dalam *milisecond* (waktu saat pencarian algoritma saja). Banyak kasus atau jumlah iterasi dalam algoritma *brute force* juga ditampilkan dan akan diberikan opsi untuk menyimpan solusi atau tidak. Jika pengguna memilih “ya”, maka solusi akan disimpan ke file output berekstensi .txt. Setelah itu, dalam tugas ini juga diimplementasikan fitur menyimpan file sebagai gambar sehingga solusi dapat dilihat dalam format gambar (.png).

BAB III

TEORI SINGKAT

Algoritma *brute force* adalah pendekatan yang lempang untuk memecahkan suatu persoalan. Metode algoritma ini mencoba semua kemungkinan solusi secara sistematis hingga menemukan suatu kombinasi solusi yang benar atau menyatakan tidak ada solusi. Algoritma ini sangat mudah untuk dipahami dan diimplementasikan disebabkan sifatnya

Algoritma *brute force* yang dipakai pada implementasi IQ Puzzler Pro menggunakan sifatnya yang “murni” (tanpa heuristik). Heuristik merupakan salah satu pendekatan cerdas yang berkaitan dengan formulasi yang biasanya spekulatif. Pendekatan ini dapat mempercepat pencarian solusi engan aturan tertentu (misalnya prioritas penempatan blok yang besar). Pada tugas ini, pendekatan heuristik tidak digunakan agar algoritma tetap murni dan diberikan ruang untuk mengeksplor algoritma *brute force*. Namun, meskipun kelebihan dari algoritma *brute force* murni menjamin solusi jika ada, kekurangannya adalah memiliki kompleksitas waktu yang tinggi sehingga memakan waktu yang banyak.

Dalam percobaan ini, meskipun memfokuskan pada algoritma *brute force*, digunakan juga suatu algoritma lain yang cenderung sering dikaitkan dengan *brute force* yaitu *backtracking*. Pendekatan *backtracking* mencoba suatu solusi parsial dan mundur jika gagal atau tidak mendapatkan solusi pada iterasi berikut-berikutnya, lalu setelah mundur akan lanjut ke kemungkinan lain berikutnya. Oleh karena itu, pendekatan ini juga sangat membantu dalam optimalisasi pencarian solusi mempercepat algoritma *brute force* dalam menemukan suatu kombinasi yang tepat.

Dalam konteks IQ Puzzler Pro, metode *brute force* dipakai dengan mencoba kombinasi setiap penempatan blok puzzle pada papan dengan menggunakan semua orientasi unik yang dimiliki setelah dirotasi atau dicerminkan. Proses ini dilakukan dengan rekursif dan ketika sampai blok yang terakhir pada urutan tertentu sudah dicapai dan jika gagal, program akan mengembalikan ke langkah sebelumnya (*backtracking*, serta mencoba blok puzzle lainnya dengan orientasi yang berbeda. Pendekatan ini lanjut hingga papan terisi penuh atau semua kemungkinan habis (tidak ada solusi).

BAB IV

DESKRIPSI ALGORITMA

4.1 Inisialisasi Data

Program memulai dengan membaca dan memvalidasi data dari input file .txt (melalui class **ReadInput**) untuk mempersiapkan dimensi papan ($N \times M$), jumlah blok (P), tipe kasus papan (S), dan daftar blok puzzle (pieceList). Data-data yang telah diinput tersebut kemudian disimpan dalam class **FileData**. Selain itu, FileData juga menyimpan matriks setiap blok-blok puzzle beserta dengan **array dinamis yaitu pieceList**. Blok-blok puzzle diproses menjadi daftar objek dalam class **Piece**.

```
1 public class FileData {
2     private int N, M, P;
3     private String S;
4     private char[][] matrix;
5     private ArrayList<Piece> pieceList;
6
7     public FileData(int N, int M, int P, String S, char[][] matrix, ArrayList<Piece> pieceList) {
8         this.N = N;
9         this.M = M;
10        this.P = P;
11        this.S = S;
12        this.matrix = matrix;
13        this.pieceList = pieceList;
14    }
```

Gambar 4.1.1 Kelas FileData

```
1 public class Piece {
2     private char[][] shape;
3     private int rows;
4     private int cols;
5     private char id;
6     private List<Piece> allOrientations;
7
8     public Piece(char[][] shape, char id) {
9         this.shape = shape;
10        this.rows = shape.length;
11        this.cols = shape[0].length;
12        this.id = id;
13        this.allOrientations = null;
14    }
```

Gambar 4.1.2 Kelas Piece

Setiap Piece direpresentasikan sebagai **matriks karakter (char[][])** beserta dengan **id** nya berupa **karakter huruf itu sendiri (A-Z)**. Kelas **Piece** juga menyimpan suatu array berisi Piece yang menyimpan *allOrientations* (segala orientasi unik dari blok puzzle). Setelah itu, Piece yang tersimpan dalam **array dinamis** berisi setiap blok puzzle (pieceList).

Papan permainan direpresentasikan oleh instance dari kelas **Board**, menyimpan bentuk blok puzzle, baris dan kolom puzzle, id atau karakter puzzle, beserta orientasi setiap puzzle

(setelah dirotasi atau dicerminkan). Konstruktor dari board berisi FileData karena papan dibuat berdasarkan dimensi $N \times M$ yang diperoleh dari FileData. Papan board diinisialisasi kosong dengan setiap space kosong dalam matriks (`char[][] grid`) direpresentasikan dengan karakter '.' (titik), menandakan belum ada blok yang ditempatkan.

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

    public Board(FileData fileData) {
        this.rows = fileData.getN();
        this.cols = fileData.getM();
        this.grid = new char[rows][cols];

        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                grid[i][j] = '.';
            }
        }
    }
}
```

Gambar 4.1.3 Kelas Board

Class **Solver** dibuat untuk melakukan proses pencarian solusi. Variabel penting dalam kelas ini meliputi *attempt* (jumlah iterasi kasus), *runtime* (waktu pencarian, *endTime* dikurang *startTime*), serta *stopped* sebagai boolean volatil untuk menghentikan proses secara manual. Variabel *showTesting* dan variabel-variabel debug bersifat opsional untuk menunjukkan debug setiap iterasi papan di CLI/Terminal (fitur ini dapat diakses di GUI), sedangkan *debugWriteToFile*, *debugWriter*, dan *debugFile* berfungsi dalam pengetesan fitur save ke file saja.

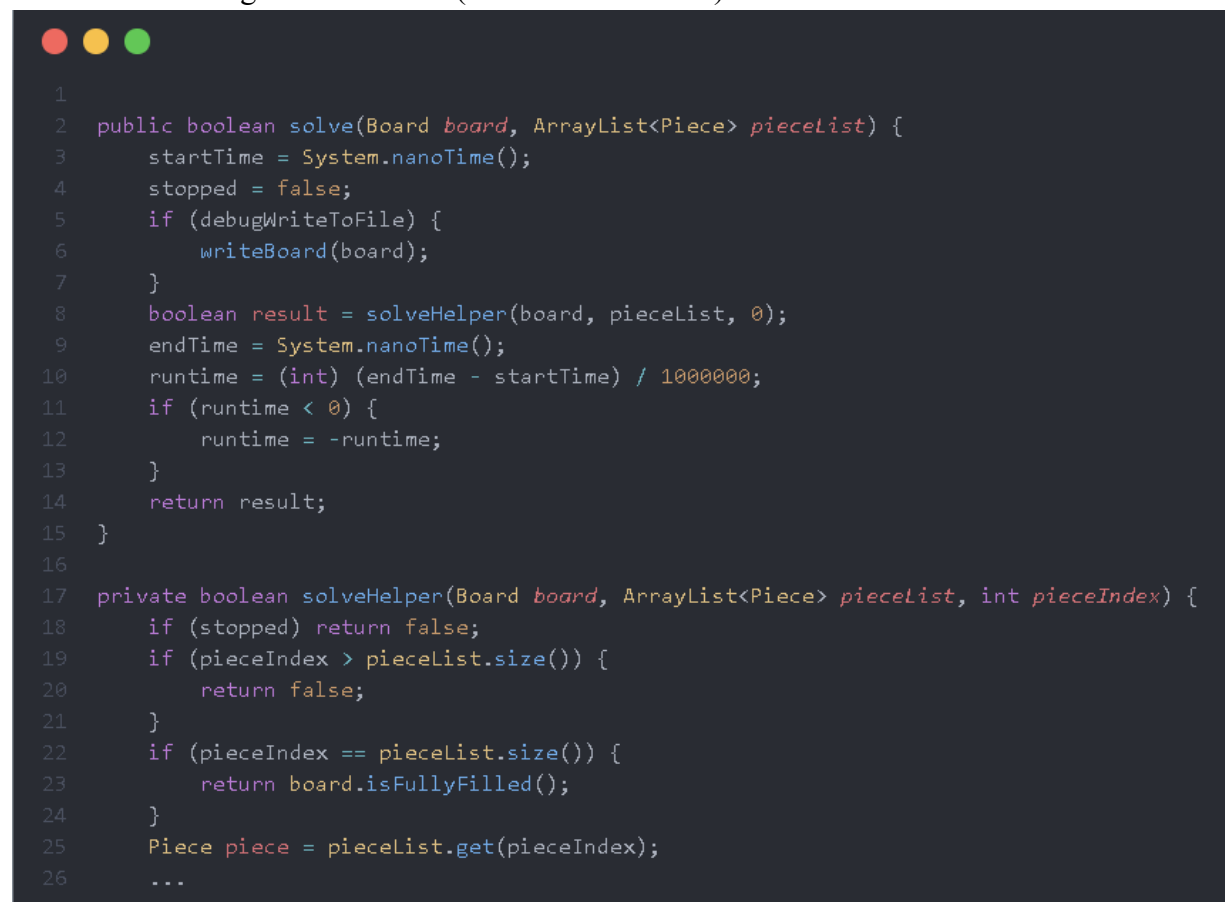
```
1 public class Solver {
2     private int attempt = 0;
3     private long startTime = 0;
4     private long endTime = 0;
5     private int runtime = 0;
6     private volatile boolean stopped;
7     private boolean showTesting = false;
8     private boolean debugWriteToFile = false;
9     private PrintWriter debugWriter;
10    private String debugFile;
11
12    public Solver() {
13        this.showTesting = false;
14    }
15
16    public Solver(boolean showTesting, boolean debugWriteToFile, String folderPath, String filePath) {
17        this.showTesting = showTesting;
18        this.debugWriteToFile = debugWriteToFile;
19
20        if (debugWriteToFile) {
21            try {
22                File folder = new File(folderPath);
23                if (!folder.exists()) {
24                    folder.mkdirs();
25                }
26
27                this.debugFile = folderPath + File.separator + filePath;
28                this.debugWriter = new PrintWriter(new FileWriter(debugFile));
29            } catch (IOException e) {
30                System.err.println("Error debug file: " + e.getMessage());
31                this.debugWriteToFile = false;
32            }
33        }
34    }
35}
```

Gambar 4.1.4 Kelas Solver

4.2 Proses Pencarian Solusi dengan Algoritma Brute Force

Metode **solve** dari kelas Solver dipanggil dengan parameter instance Board **board** (papan permainan) dan array dinamis **pieceList** (daftar blok). Pada awal metode ini, waktu mulai (**startTime**) dicatat untuk mengukur waktu pencarian. Selanjutnya, metode **solve** memanggil metode **solveHelper** yang merupakan fungsi rekursif utama dalam mencari solusi permainan. Metode ini bekerja secara rekursif untuk menempatkan blok satu per satu dengan beberapa kondisi:

1. Jika **stopped** bernilai true maka metode akan mengembalikan false dan berhenti mencari.
2. Jika **pieceIndex** lebih besar dari ukuran **pieceList** (jumlah blok puzzle lebih dari cukup), maka tidak valid dan mengembalikan false.
3. Jika **pieceIndex** sama dengan ukuran **pieceList**, artinya semua blok telah dicoba dan memeriksa apakah papan terisi penuh (memanggil **isFullyFilled()**), jika sudah penuh maka mengembalikan true (solusi ditemukan).

A screenshot of a code editor showing the implementation of the solve and solveHelper methods. The code is in Java and is displayed on a dark background with light-colored text. The solve method is a public boolean method that takes a Board object and an ArrayList of Piece objects as parameters. It initializes startTime, stopped, and debugWriteToFile, then calls solveHelper. The solveHelper method is a private boolean method that takes the same Board and ArrayList parameters, plus an int pieceIndex. It checks for stopped, pieceIndex > pieceList.size(), and pieceIndex == pieceList.size() conditions, and returns the result of board.isFullyFilled() or the result of placing a piece.

```
1
2 public boolean solve(Board board, ArrayList<Piece> pieceList) {
3     startTime = System.nanoTime();
4     stopped = false;
5     if (debugWriteToFile) {
6         writeBoard(board);
7     }
8     boolean result = solveHelper(board, pieceList, 0);
9     endTime = System.nanoTime();
10    runtime = (int) (endTime - startTime) / 1000000;
11    if (runtime < 0) {
12        runtime = -runtime;
13    }
14    return result;
15 }
16
17 private boolean solveHelper(Board board, ArrayList<Piece> pieceList, int pieceIndex) {
18     if (stopped) return false;
19     if (pieceIndex > pieceList.size()) {
20         return false;
21     }
22     if (pieceIndex == pieceList.size()) {
23         return board.isFullyFilled();
24     }
25     Piece piece = pieceList.get(pieceIndex);
26     ...
```

Gambar 4.2.1 Metode solve dan solveHelper (bagian kondisi)

Proses **solveHelper** dimulai dari blok pertama, dengan mengambil objek **Piece** pada indeks **pieceIndex** dari **pieceList**. Setelah itu, akan didapatkan semua kemungkinan orientasi blok puzzle menggunakan function **getAllOrientationsList** dari kelas **Piece** (rotasi 0°, 90°, 180°, 270°, serta cermin horizontal dan vertikal). Namun, pada program ini, hanya diimplementasikan cermin horizontal karena cermin vertikal tidak menghasilkan orientasi unik

diluar kombinasi rotasi dan cermin horizontal (rotasi dan flip satu saja sudah menghasilkan segala orientasi unik).

Pengecekan orientasi unik menggunakan HashSet sehingga tidak ada bentuk puzzle yang duplikat. Metode **getAllOrientations()** menghasilkan daftar objek Piece yang mewakili setiap orientasi unik, dihitung sebelumnya untuk efisiensi dan kecepatan.



```

1 public List<Piece> getAllOrientationsList() {
2     Set<String> uniqueOrientations = new HashSet<>();
3     List<Piece> orientations = new ArrayList<>();
4     Piece current = this;
5     for (int r = 0; r < 4; r++) {
6         addIfUnique(orientations, uniqueOrientations, current);
7
8         Piece flippedHorizontally = current.flipHorizontal();
9         addIfUnique(orientations, uniqueOrientations, flippedHorizontally);
10        current = current.rotate();
11    }
12    return orientations;
13 }
14 public void initializeOrientations() {
15     this.allOrientations = getAllOrientationsList();
16 }
17 public List<Piece> getAllOrientations() {
18     if (allOrientations == null) {
19         initializeOrientations();
20     }
21     return allOrientations;
22 }

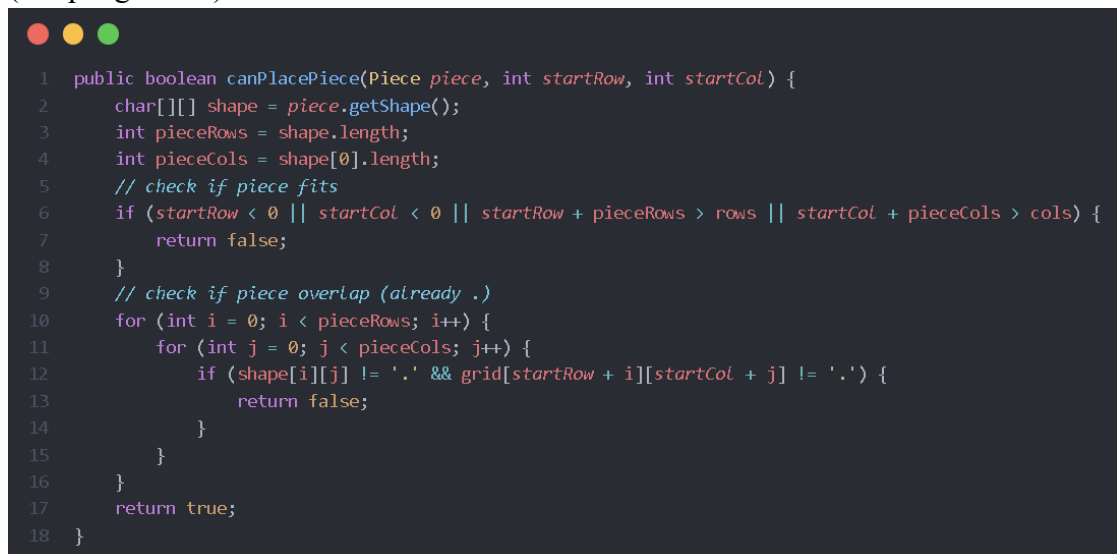
1 // ? HELPER
2 private void addIfUnique(List<Piece> orientations, Set<String>
3     uniqueOrientations, Piece piece) {
4     String rep = pieceToString(piece.getShape());
5     if (uniqueOrientations.add(rep)) {
6         orientations.add(piece);
7     }
8 }
9 private String pieceToString(char[][] shape) {
10    StringBuilder sb = new StringBuilder();
11    for (char[] row : shape) {
12        sb.append(Arrays.toString(row));
13    }
14    return sb.toString();
15 }

```

Gambar 4.2.2 Metode getAllOrientationsList dan addIfUnique untuk pencarian orientasi blok unik

Lengkapnya, **solveHelper** melakukan perulangan untuk setiap posisi di papan (i, j) dari $(0,0)$ hingga $(rows - 1, cols - 1)$, dan setiap orientasi dalam daftar orientasi:

1. **Diperiksa apakah orientasi blok puzzle dapat ditempatkan** dengan memanggil **canPlacePiece(orientation, i, j)** dari instance Board. Metode tersebut memverifikasi bahwa blok tidak melampaui batas papan dan tidak ada puzzle yang saling *overlap* (tumpang tindih).



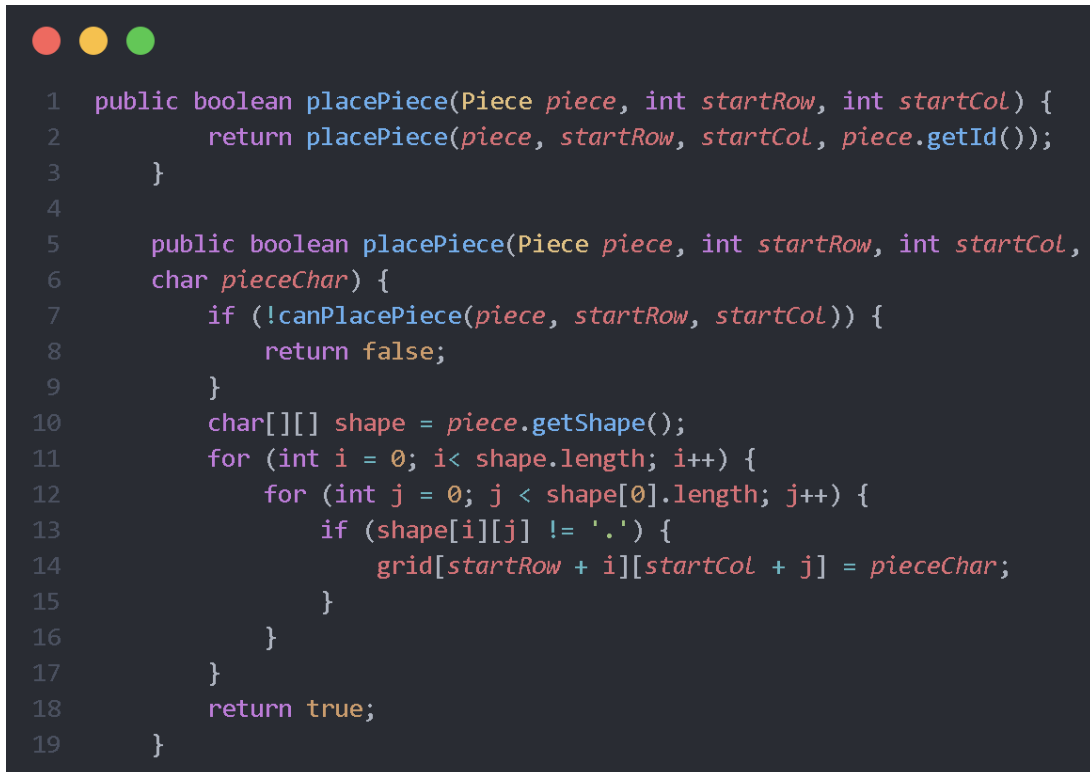
```

1 public boolean canPlacePiece(Piece piece, int startRow, int startCol) {
2     char[][] shape = piece.getShape();
3     int pieceRows = shape.length;
4     int pieceCols = shape[0].length;
5     // check if piece fits
6     if (startRow < 0 || startCol < 0 || startRow + pieceRows > rows || startCol + pieceCols > cols) {
7         return false;
8     }
9     // check if piece overlap (already .)
10    for (int i = 0; i < pieceRows; i++) {
11        for (int j = 0; j < pieceCols; j++) {
12            if (shape[i][j] != '.' && grid[startRow + i][startCol + j] != '.') {
13                return false;
14            }
15        }
16    }
17    return true;
18 }

```

Gambar 4.2.3 Metode canPlacePiece untuk mengecek apakah blok puzzle bisa ditempatkan di posisi tertentu di Board

2. Jika **canPlacePiece** mengembalikan true, maka akan ditempatkan blok puzzle dengan memanggil **placePiece(orientation, i, j)** pada instance Board, yang mengisi sel sesuai dalam matriks grid.



```
1 public boolean placePiece(Piece piece, int startRow, int startCol) {
2     return placePiece(piece, startRow, startCol, piece.getId());
3 }
4
5 public boolean placePiece(Piece piece, int startRow, int startCol,
6 char pieceChar) {
7     if (!canPlacePiece(piece, startRow, startCol)) {
8         return false;
9     }
10    char[][] shape = piece.getShape();
11    for (int i = 0; i < shape.length; i++) {
12        for (int j = 0; j < shape[0].length; j++) {
13            if (shape[i][j] != '.') {
14                grid[startRow + i][startCol + j] = pieceChar;
15            }
16        }
17    }
18    return true;
19 }
```

Gambar 4.2.4 Metode placePiece untuk menempatkan blok puzzle di posisi tertentu dalam board

3. **Tambahkan nilai attempt** (banyak kasus) sebanyak 1 setiap kali penempatan blok yang valid (**canPlacePiece** mengembalikan true) dilakukan, bukan hanya yang berhasil atau gagal di akhir saja.
4. Panggil solveHelper secara rekursif dengan pieceIndex ditambah 1
5. **Jika hasil panggilan rekursif solveHelper true**, artinya solusi ditemukan dan diakhiri pencarian. Solusi tersebut ditemukan ketika semua block dalam pieceList sudah ditempatkan dalam papan secara valid dan isFullyFilled() mengembalikan true, menandakan tidak ada sel kosong pada papan.
6. **Jika hasil solveHelper false**, artinya blok berikutnya tidak dapat ditempatkan dan papan belum penuh sehingga akan dilakukan *backtracking* dengan memanggil **removePiece(orientation,i,j)** pada instance Board. Hal ini mengembalikan sel-sel yang sebelumnya diisi menjadi '.' kondisi sebelum penempatan. *Backtracking* berlanjut hingga algoritma mencoba semua kombinasi posisi dan orientasi untuk blok sebelumnya, atau hingga semua kemungkinan tidak ditemukan solusi.

```

1 private boolean solveHelper(Board board, ArrayList<Piece> pieceList, int pieceIndex) {
2     if (stopped) return false;
3     if (pieceIndex > pieceList.size()) {
4         return false;
5     }
6     if (pieceIndex == pieceList.size()) {
7         return board.isFullyFilled();
8     }
9     Piece piece = pieceList.get(pieceIndex);
10    ...
11    for (int i = 0; i < board.getRows(); i++) {
12        for (int j = 0; j < board.getCols(); j++) {
13            for (Piece orientation: piece.getAllOrientations()) {
14                if (board.canPlacePiece(orientation, i, j)) {
15                    if (showTesting) {
16                        board.printBoard();
17                    }
18                    board.placePiece(orientation, i, j);
19                    attempt++;
20                    if (debugWriteToFile) {
21                        writeBoard(board);
22                    }
23                    if (solveHelper(board, pieceList, pieceIndex + 1)){
24                        return true;
25                    }
26                    // ! BACKTRACKING
27                    board.removePiece(orientation, i, j);
28                    if (showTesting) {
29                        board.printBoard();
30                    }
31
32
33                    if (debugWriteToFile) {
34                        writeBoard(board);
35                    }
36                }
37            }
38        }
39    }
40    return false;
41 }

```

Gambar 4.2.5 Metode solveHelper

4.3 Penyelesaian dan Pengolahan Hasil

Ketika solveHelper selesai (baik dengan solusi atau kegagalan total), akan dicatat waktu *endTime* dan dikurangkan dengan *startTime* mendapatkan *runtime* (waktu pencarian). **Jika hasil *runtime* negatif (karena overflow), akan diambil nilai absolutnya.** Hasil solveHelper kemudian dikembalikan ke metode solve.

Jika hasil true, maka matriks grid berisi **kombinasi solusi blok yang valid**. Sedangkan, jika hasilnya false, papan akan tetap dalam **keadaan awal (kosong)**, karena semua penempatan telah di batalkan oleh *backtracking*. Papan akhir, beserta dengan waktu pencarian serta jumlah kasus akan ditunjukkan. Tampilan akhir maupun hasil *output* akan divisualisasikan di dalam bab selanjutnya.

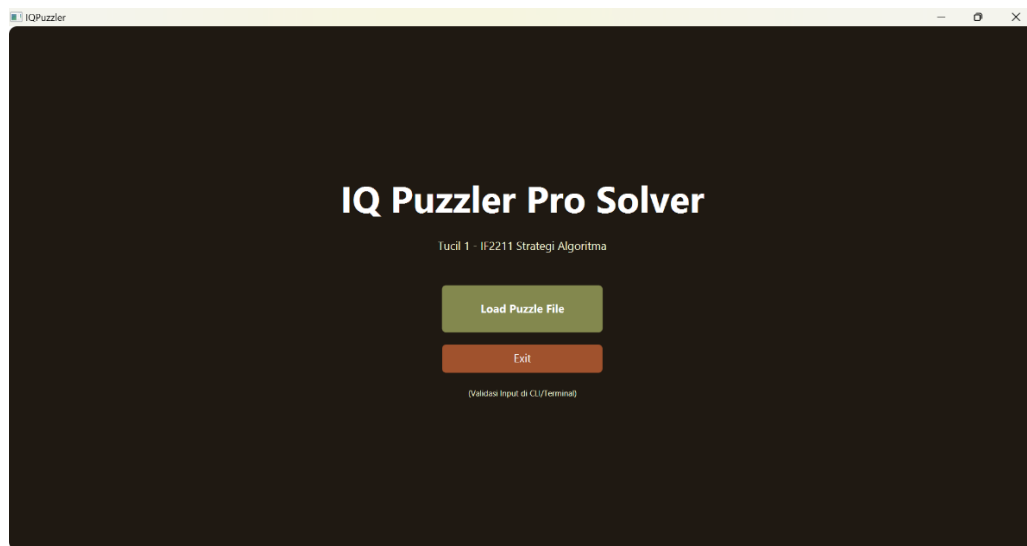
BAB V

IMPLEMENTASI BONUS

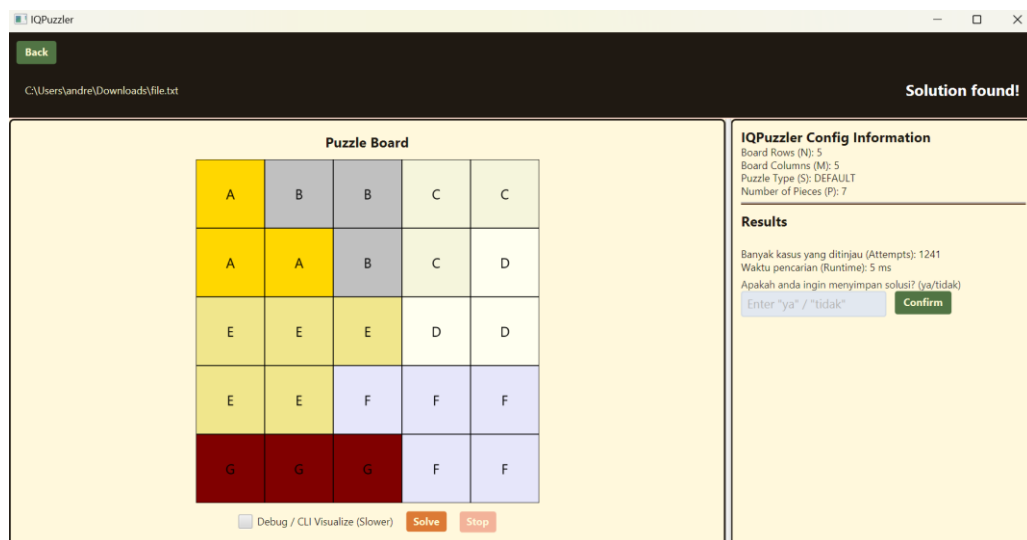
Selain spesifikasi wajib, dalam proyek ini dua fitur bonus diimplementasikan, yaitu Graphical User Interface (GUI) dan Output berupa Gambar. Implementasi ini memanfaatkan JavaFX sebagai framework untuk antarmuka dan penggunaan Java AWT dari JavaFX untuk output berupa gambar.

5.1 Graphical User Interface (GUI)

Fitur GUI bertujuan untuk memvisualisasikan papan puzzle yang terisi oleh blok-blok berwarna. Implementasi GUI memberikan pengalaman interaktif serta memungkinkan melihat proses penyelesaian, hasil akhir, dan statistik secara *real-time*.



Gambar 5.1.1 Title page IQ Puzzler Pro Solver



Gambar 5.1.2 GUI contoh tampilan solusi IQ Puzzler Pro

Visualisasi papan pada Gambar 5.1.2 memvisualisasikan papan terisi secara berwarna, meningkatkan pemahaman dan interaktivitas pengguna dibanding output CLI/terminal biasa. Ketika tombol Solve diklik, penggunaan Task dan Timer dari JavaFX menyediakan akses untuk memperbarui tampilan papan dan status (Attempts: ...) secara berkala selama proses berlangsung. Jika sudah selesai, akan muncul status “No solution!” jika tidak didapatkan solusi, dan “Solution found!” disertakan opsi untuk menyimpan solusi jika didapatkan solusi.

5.2 Output sebagai Gambar

Fitur ini memenuhi spesifikasi bonus dengan memberikan output solusi dalam format gambar (.png) yang mudah dilihat. Kedua output (baik solusi papan biasa maupun solusi gambar) dapat dilihat pada folder “test/” dengan solusi non-gambar disimpan dalam “test/output.txt” serta solusi gambar disimpan dalam “test/solution.png”.

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

Gambar 5.2.1 Solusi IQ Puzzler Pro dalam file .png


```
test > output.txt
1  ABBCC
2  AABCD
3  EEEDD
4  EEFFF
5  GGGFF
```

Gambar 5.2.2 Solusi IQ Puzzler Pro dalam file .txt

BAB VI

PENGETESAN PROGRAM

Test Case 1

Input	Output																																																		
5 5 7 DEFAULT A AA B BB C CC D DD EE EE E FF FF F GGG	<div><div><div><div>Back</div><div>C:\Users\landsef\Downloads\file.txt</div><div>Solution found!</div></div><div><div><div>Puzzle Board</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><div><div>Debug / CLI Visualize (Slower)</div><div>Solve</div><div>Stop</div></div></div></div><div><div><div>IQPuzzler Config Information</div><div>Board Rows (N): 5 Board Columns (M): 5 Puzzle Type (S): DEFAULT Number of Pieces (P): 7</div></div><div><div>Results</div><div>Banyak kasus yang ditinjau (Attempts): 1241 Waktu pencarian (Runtime): 1 ms Apakah anda ingin menyimpan solusi? (y/n/t/s/a)</div><div><div></div><div>Confirm</div></div></div></div></div><div>test/output.txt<div>test >  output.txt</div><div>1 ABBCC</div><div>2 AABCD</div><div>3 EEEDD</div><div>4 EEEFF</div><div>5 GGGFF</div></div><div>test/solution.png<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></div></div></div>	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
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																																															

Test Case 2

Input	Output
5 11 12 DEFAULT AAA A A BBB BB CC C C DD DD D E EE E E F FF FF G GGGG H HHHH III I J JJ K KK K L L LLL	<div> <div>Back</div> <div>C:\Users\andri\Downloads\file.txt</div> <div>Solution found!</div> <div> <div>Puzzle Board</div> </div> <div> <div>IQPuzzler Config Information</div> <div> Board Rows (N): 5 Board Columns (M): 11 Puzzle Type (S): DEFAULT Number of Pieces (P): 12 </div> </div> <div> <div>Results</div> <div> Banyak kasus yang ditinjau (Attempts): 1469463 Waktu pencarian (Runtime): 2113 ms Apakah anda ingin menyimpan solusi? (ya/tidak) </div> <div>Confirm</div> </div> </div>

test/output.txt

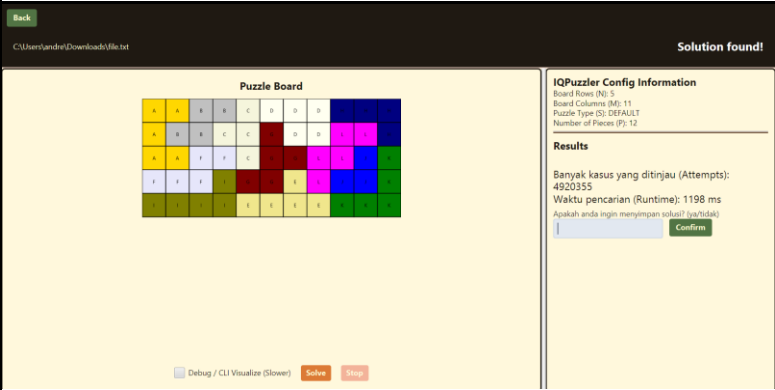
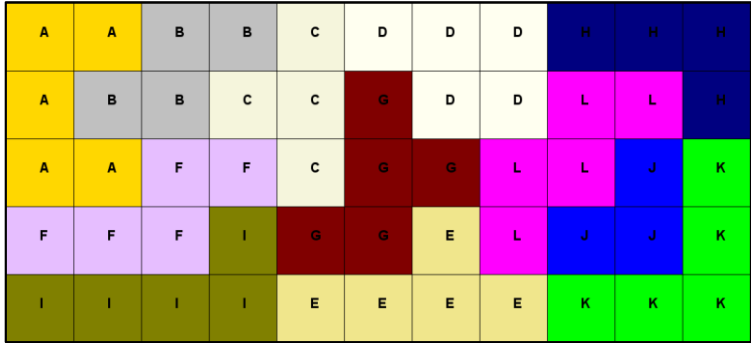
```

test > output.txt
1  AAABBBBCDDHH
2  AIAFBBCJDDH
3  LIIFFCJJDDH
4  LIFEEEEGKKH
5  LLLLEGGGGKK

```

test/solution.png

Test Case 3

Input	Output
5 11 12 DEFAULT AA A AA BB BB CCC C DDD DD E E EE E FF FFF GG GG G HH H H I IIII J JJ K K KKK L LL LL	<div>  </div> <div> test/output.txt <pre> test > output.txt 1 AABBCDDDDHHH 2 ABBCCGDDLH 3 AAFFCGLLJK 4 FFFIGGELJJK 5 IIIIEEEEKKK </pre> </div> <div> test/solution.png  </div>

Test Case 4

Input	Output
8 12 9 DEFAULT BBBB BBBBB BBBBB BBBBB BBBBB BBBBBBBBB BBB BBB BBB BBB I I I I II IIIII AAA AAA D DD DD DD EEE E E GG GG H H HHHH HH CCC C KK KK KK	<div> <div>Back</div> <div>C:\Users\andre\Downloads\file.txt</div> <div>Solution found!</div> </div> <div> <div>Puzzle Board</div> </div> <div> <div>IQPuzzler Config Information</div> <div>Board Rows (N): 8</div> <div>Board Columns (M): 12</div> <div>Puzzle Type (S): DEFAULT</div> <div>Number of Pieces (P): 9</div> </div> <div> <div>Results</div> <div>Banyak kasus yang ditinjau (Attempts): 307</div> <div>Waktu pencarian (Runtime): 0 ms</div> <div>Apakah anda ingin menyimpan solusi? (y/n)</div> <div>Enter "y" / "tidak"</div> <div>Confirm</div> </div>

test/output.txt

```

test > output.txt
1 IAAAHBBBBCCC
2 IAAAHBBBBBCD
3 IH HHHBBBBBDD
4 IH HI BBBBDD
5 I I I I BBBBDD
6 BBBB BBBB EEE
7 BBBKKKBBBEGG
8 BBBKKKBBBEGG


```

test/solution.png

Test Case 5

Input	Output																																										
6 7 10 DEFAULT AAAA BB BB B C C C C C DD EE EE FF F F GG GG G HHH H III I J JJJJ	<div><div>Back</div><div>C:\Users\andre\Downloads\Me.txt</div><div>Solution found!</div></div> <div><div>Puzzle Board</div><table><tr><td>A</td><td>A</td><td>A</td><td>A</td><td>B</td><td>B</td><td>C</td></tr><tr><td>D</td><td>D</td><td>E</td><td>B</td><td>B</td><td>B</td><td>C</td></tr><tr><td>F</td><td>E</td><td>E</td><td>I</td><td>G</td><td>G</td><td>C</td></tr><tr><td>F</td><td>E</td><td>I</td><td>I</td><td>G</td><td>G</td><td>C</td></tr><tr><td>F</td><td>F</td><td>H</td><td>I</td><td>G</td><td>J</td><td>C</td></tr><tr><td>H</td><td>H</td><td>H</td><td>J</td><td>J</td><td>J</td><td>J</td></tr></table><div><input type="checkbox"/> Debug / <input type="checkbox"/> Visualize (Slower) <input type="button" value="Solve"/> <input type="button" value="Stop"/></div></div> <div><div>IQPuzzler Config Information</div><div>Board Rows (N): 6 Board Columns (M): 7 Puzzle Type (S): DEFAULT Number of Pieces (P): 10</div><div>Results</div><div>Banyak kasus yang ditinjau (Attempts): 299 Waktu pencarian (Runtime): 1 ms Apakah anda ingin menyimpan solusi? (y/n): <input type="button" value="Confirm"/></div></div>	A	A	A	A	B	B	C	D	D	E	B	B	B	C	F	E	E	I	G	G	C	F	E	I	I	G	G	C	F	F	H	I	G	J	C	H	H	H	J	J	J	J
A	A	A	A	B	B	C																																					
D	D	E	B	B	B	C																																					
F	E	E	I	G	G	C																																					
F	E	I	I	G	G	C																																					
F	F	H	I	G	J	C																																					
H	H	H	J	J	J	J																																					
	<div>test/output.txt</div> <div>test > output.txt</div> <div>1 AAAABBC</div> <div>2 DDEBBBC</div> <div>3 FEEIGGC</div> <div>4 FEIIGGC</div> <div>5 FFHIGJC</div> <div>6 HHHJJJJ</div>																																										
	<div>test/solution.png</div> <table><tr><td>A</td><td>A</td><td>A</td><td>A</td><td>B</td><td>B</td><td>C</td></tr><tr><td>D</td><td>D</td><td>E</td><td>B</td><td>B</td><td>B</td><td>C</td></tr><tr><td>F</td><td>E</td><td>E</td><td>I</td><td>G</td><td>G</td><td>C</td></tr><tr><td>F</td><td>E</td><td>I</td><td>I</td><td>G</td><td>G</td><td>C</td></tr><tr><td>F</td><td>F</td><td>H</td><td>I</td><td>G</td><td>J</td><td>C</td></tr><tr><td>H</td><td>H</td><td>H</td><td>J</td><td>J</td><td>J</td><td>J</td></tr></table>	A	A	A	A	B	B	C	D	D	E	B	B	B	C	F	E	E	I	G	G	C	F	E	I	I	G	G	C	F	F	H	I	G	J	C	H	H	H	J	J	J	J
A	A	A	A	B	B	C																																					
D	D	E	B	B	B	C																																					
F	E	E	I	G	G	C																																					
F	E	I	I	G	G	C																																					
F	F	H	I	G	J	C																																					
H	H	H	J	J	J	J																																					


Test Case 6

Input	Output																																																																																				
<pre>7 6 7 DEFAULT AAAA A A BB BBB BB BBBB BBB C CC CC DD DD E EEE E FF G G G GGG</pre>	<div><div><div>Back</div><div>C:\Users\anket\Downloads\id3\id3.txt</div><div>Solution found!</div></div><div><div><div>Puzzle Board</div><table><tr><td>A</td><td>A</td><td>A</td><td>A</td><td>S</td><td>S</td></tr><tr><td>A</td><td>S</td><td>S</td><td>A</td><td>S</td><td>S</td></tr><tr><td>C</td><td>S</td><td>S</td><td>S</td><td>S</td><td>S</td></tr><tr><td>C</td><td>C</td><td>S</td><td>S</td><td>S</td><td>S</td></tr><tr><td>E</td><td>C</td><td>C</td><td>D</td><td>D</td><td></td></tr><tr><td>E</td><td>E</td><td>E</td><td>D</td><td>D</td><td></td></tr><tr><td>F</td><td>F</td><td>E</td><td></td><td></td><td></td></tr></table><div>Debug / CLI Visualizer (Slower) Solve Stop</div></div><div><div><div>IQPuzzler Config Information</div><div>Board Rows (R): 7 Board Columns (C): 6 Puzzle Type (S): DEFAULT Number of Pieces (P): 7</div><div><div>Results</div><div>Banyak kasus yang ditinjau (Attempts): 10 Waktu pencarian (Runtime): 0 ms Apakah anda ingin menyimpan solusi? (save/Save)</div><div>Confirm</div></div></div></div></div><div>test/output.txt<div>test >  output.txt</div><div>1 AAAABB</div><div>2 ABBABB</div><div>3 CBBBBB</div><div>4 CCBBBG</div><div>5 ECCDDG</div><div>6 EEEDDG</div><div>7 FFEGGG</div></div><div>test/solution.png<table><tr><td>A</td><td>A</td><td>A</td><td>A</td><td>B</td><td>B</td></tr><tr><td>A</td><td>B</td><td>B</td><td>A</td><td>B</td><td>B</td></tr><tr><td>C</td><td>B</td><td>B</td><td>B</td><td>B</td><td>B</td></tr><tr><td>C</td><td>C</td><td>B</td><td>B</td><td>B</td><td>G</td></tr><tr><td>E</td><td>C</td><td>C</td><td>D</td><td>D</td><td>G</td></tr><tr><td>E</td><td>E</td><td>E</td><td>D</td><td>D</td><td>G</td></tr><tr><td>F</td><td>F</td><td>E</td><td>G</td><td>G</td><td>G</td></tr></table></div></div>	A	A	A	A	S	S	A	S	S	A	S	S	C	S	S	S	S	S	C	C	S	S	S	S	E	C	C	D	D		E	E	E	D	D		F	F	E				A	A	A	A	B	B	A	B	B	A	B	B	C	B	B	B	B	B	C	C	B	B	B	G	E	C	C	D	D	G	E	E	E	D	D	G	F	F	E	G	G	G
A	A	A	A	S	S																																																																																
A	S	S	A	S	S																																																																																
C	S	S	S	S	S																																																																																
C	C	S	S	S	S																																																																																
E	C	C	D	D																																																																																	
E	E	E	D	D																																																																																	
F	F	E																																																																																			
A	A	A	A	B	B																																																																																
A	B	B	A	B	B																																																																																
C	B	B	B	B	B																																																																																
C	C	B	B	B	G																																																																																
E	C	C	D	D	G																																																																																
E	E	E	D	D	G																																																																																
F	F	E	G	G	G																																																																																

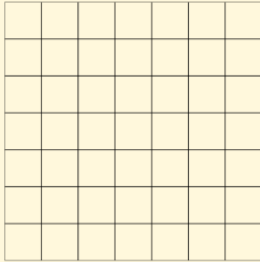
Test Case 7

Input	Output																																								
5 8 7 DEFAULT AA AA A AAA A B BB B B C CCC CCC DD EE EE E E FF F F F G GG GGG	<div><div>Back</div><div>C:\Users\andre\Downloads\llc.txt</div><div>Solution found!</div></div> <div><div>Puzzle Board</div><table><tr><td>A</td><td>A</td><td>B</td><td>B</td><td>B</td><td>B</td><td>C</td><td>D</td></tr><tr><td>E</td><td>A</td><td>A</td><td>B</td><td>A</td><td>C</td><td>C</td><td>D</td></tr><tr><td>E</td><td>E</td><td>A</td><td>A</td><td>A</td><td>G</td><td>C</td><td>C</td></tr><tr><td>F</td><td>E</td><td>E</td><td>E</td><td>A</td><td>G</td><td>G</td><td>C</td></tr><tr><td>F</td><td>F</td><td>F</td><td>F</td><td>G</td><td>G</td><td>G</td><td>C</td></tr></table><div><div>Debug / UI Visualize (Slower)</div><div>Solve</div><div>Stop</div></div></div> <div><div>IQPuzzler Config Information</div><div>Board Rows (N): 5 Board Columns (M): 8 Puzzle Type (S): DEFAULT Number of Pieces (P): 7</div><div>Results</div><div>Banyak kasus yang ditinjau (Attempts): 72 Waktu pencarian (Runtime): 0 ms Apakah anda ingin menyimpan solusi ini? (y/n): <div>Enter 'y' or 'n':</div><div>Confirm</div></div></div>	A	A	B	B	B	B	C	D	E	A	A	B	A	C	C	D	E	E	A	A	A	G	C	C	F	E	E	E	A	G	G	C	F	F	F	F	G	G	G	C
A	A	B	B	B	B	C	D																																		
E	A	A	B	A	C	C	D																																		
E	E	A	A	A	G	C	C																																		
F	E	E	E	A	G	G	C																																		
F	F	F	F	G	G	G	C																																		
	test/output.txt test > output.txt 1 AABBBBCD 2 EAABACCD 3 EAAAAGCC 4 FEEAGGC 5 FFFFGGGC																																								
	test/solution.png <table><tr><td>A</td><td>A</td><td>B</td><td>B</td><td>B</td><td>B</td><td>C</td><td>D</td></tr><tr><td>E</td><td>A</td><td>A</td><td>B</td><td>A</td><td>C</td><td>C</td><td>D</td></tr><tr><td>E</td><td>E</td><td>A</td><td>A</td><td>A</td><td>G</td><td>C</td><td>C</td></tr><tr><td>F</td><td>E</td><td>E</td><td>E</td><td>A</td><td>G</td><td>G</td><td>C</td></tr><tr><td>F</td><td>F</td><td>F</td><td>F</td><td>G</td><td>G</td><td>G</td><td>C</td></tr></table>	A	A	B	B	B	B	C	D	E	A	A	B	A	C	C	D	E	E	A	A	A	G	C	C	F	E	E	E	A	G	G	C	F	F	F	F	G	G	G	C
A	A	B	B	B	B	C	D																																		
E	A	A	B	A	C	C	D																																		
E	E	A	A	A	G	C	C																																		
F	E	E	E	A	G	G	C																																		
F	F	F	F	G	G	G	C																																		

Test Case 8

Input	Output																		
<pre>3 3 3 DEFAULT AA A B BB C C C</pre>	<div><div><div>Back</div><div>C:\Users\andri\Downloads\file.txt</div><div>Solution found!</div></div><div><div><div>Puzzle Board</div><table><tr><td>A</td><td>A</td><td>B</td></tr><tr><td>A</td><td>B</td><td>B</td></tr><tr><td>C</td><td>C</td><td>C</td></tr></table><div><div><input type="checkbox"/> Debug / CLI Visualize (Slower)</div><div><div>Solve</div><div>Stop</div></div></div></div></div><div><div><div>IQPuzzler Config Information</div><div>Board Rows (R): 3 Board Columns (M): 3 Puzzle Type (S): DEFAULT Number of Pieces (P): 3</div></div><div><div>Results</div><div>Banyak kasus yang ditinjau (Attempts): 3 Waktu pencarian (Runtime): 0 ms Apakah anda ingin menyimpan solusi? (ya/tidak)</div><div><div>Enter "ya" / "tidak"</div><div>Confirm</div></div></div></div></div> <div><pre>test/output.txt test >  output.txt 1 AAB 2 ABB 3 CCC</pre></div> <div><pre>test/solution.png</pre><table><tr><td>A</td><td>A</td><td>B</td></tr><tr><td>A</td><td>B</td><td>B</td></tr><tr><td>C</td><td>C</td><td>C</td></tr></table></div>	A	A	B	A	B	B	C	C	C	A	A	B	A	B	B	C	C	C
A	A	B																	
A	B	B																	
C	C	C																	
A	A	B																	
A	B	B																	
C	C	C																	

Test Case 9

Input	Output
7 7 5 DEFAULT AAA A BBB B CCCCC C C DDD D EEE E	<div> <div>Sack</div> <div>C:\Users\andri\Downloads\file.txt</div> <div>No solution!</div> <div> <div>Puzzle Board</div>  <div> <input type="checkbox"/> Debug / CLI Visualize (Slower) <div>Solve</div> <div>Stop</div> </div> </div> <div> <div>IQPuzzler Config Information</div> <div> Board Rows (N): 7 Board Columns (M): 7 Puzzle Type (S): DEFAULT Number of Pieces (P): 5 </div> </div> <div> <div>Results</div> <div> Banyak kasus yang ditinjau (Attempts): 118665048 Waktu pencarian (Runtime): 1571 ms </div> </div> </div> <div>TIDAK ADA SOLUSI</div>

BAB VII

KESIMPULAN DAN SARAN

7.1 Kesimpulan

Implementasi penyelesaian IQ Puzzler Pro dilakukan dalam Tugas Kecil 1 IF2211 Strategi Algoritma dengan menggunakan algoritma *brute force*. Program berhasil memenuhi spesifikasi wajib dengan menemukan solusi atau menyatakan tidak ada solusi untuk kasus DEFAULT. Setelah itu, fitur bonus seperti GUI dan penyimpanan gambar meningkatkan interaktivitas dan visualisasi yang menarik bagi pengguna.

Berdasarkan implementasi yang telah dibuat, dapat disimpulkan bahwa algoritma *brute force* sangat baik sebagai pendekatan pencarian solusi yang terjamin. Namun, keterbatasan *brute force* terlihat dalam waktu eksekusi yang lama pada papan yang besar, dicerminkan dengan kompleksitas besar dalam jumlah kasus iterasi (*attempts*) dan waktu pencarian (*runtime*). Pendekatan algoritma *backtracking* juga dipakai bersamaan dengan algoritma *brute force* dengan tujuan optimalisasi dan pencarian solusi yang benar.

7.1 Saran

Program dapat dikembangkan lebih lanjut, terutama dalam sisi optimalisasi ataupun menambahkan validasi yang lebih lengkap dan konsisten. Kedepannya, program juga dapat dikembangkan untuk konfigurasi tipe papan CUSTOM maupun PYRAMID agar dapat memecahkan masalah dengan *scope* yang lebih luas dan kompleks.

BAB VIII

LAMPIRAN

Lampiran

- a. Link Repository GitHub:

https://github.com/andrewtedja/Tucil1_13523148

- b. Tabel Hasil

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 custom		✓
8	Program dapat menyelesaikan kasus konfigurasi Piramida (3D)		✓
9	Program dibuat oleh saya sendiri	✓	