

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

Nathaniel Jonathan Rusli – 13523013

13523013@std.stei.itb.ac.id

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

Daftar Isi

IQ Puzzler Pro	2
Implementasi Brute Force	2
1. Pseudocode Implementasi	2
2. Penjelasan Implementasi	3
Source Code Program	5
1. BruteForceSolver.java	5
2. Board.java	6
3. Piece.java	8
4. FileIO.java	10
5. Main.java	13
Test Case: Input & Output	15
1. Terminal Input and Output	15
2. File and Image Output	17
Lampiran	19

IQ Puzzler Pro

IQ Puzzler Pro merupakan permainan puzzle strategi yang menguji kemampuan pemain dalam menyusun kepingan berbentuk unik ke dalam papan hingga seluruh ruang terisi tanpa tumpang tindih. Setiap kepingan memiliki bentuk dan orientasi tertentu yang dapat diputar dan dicerminkan untuk menyesuaikan dengan pola papan. Tantangan utama dalam permainan ini adalah menemukan cara penempatan kepingan yang tepat sehingga tidak ada ruang tersisa dan tidak ada kepingan yang bertumpuk.

Laporan ini dibuat sebagai penyelesaian Tugas Kecil 1 IF2211 Strategi Algoritma. Dalam tugas ini, saya mengembangkan program *solver* otomatis untuk IQ Puzzler Pro menggunakan algoritma *brute force* dengan pendekatan *backtracking*. Program ini dikembangkan dalam bahasa Java dan mampu membaca puzzle dari file input, mencari satu solusi yang valid, serta menyimpan hasilnya dalam bentuk file teks (.txt) dan gambar (.png).

Pendekatan *brute force* yang digunakan dalam program ini bekerja dengan mengeksplorasi seluruh kemungkinan peletakan kepingan, termasuk seluruh rotasi dan pencerminan yang diperbolehkan (*brute force* murni). Jika ditemukan solusi yang valid, program akan mencetak hasilnya, sedangkan jika tidak ada solusi, program akan menginformasikan bahwa puzzle tidak memiliki solusi.

Implementasi *Brute Force*

1. Pseudocode Implementasi

Berikut adalah pseudocode algoritma *brute force* murni (konvensi dalam kelas):

```
procedure solve():  
    isSolved ← backtrack(0)  
    if (isSolved) then  
        board.printBoard()  
    output("Puzzle tidak memiliki solusi!")  
  
function backtrack(int idx) → boolean:  
    iterCount ← iterCount + 1  
  
    if (idx = listOfPieces.size()) then  
        if (board.isFull) then  
            → true  
        → false
```

```

Piece piece ← listOfPieces.get(idx)
List<Piece> transformedPieces ← piece.getRotationsAndMirrors();

iterate i [0..7]
    Piece currentPiece ← rotatedAndMirroredPieces.get(i);

    iterate row [0..board.getRows()]
        iterate col [0..board.getCols()]
            if (board.canPlacePiece(currentPiece, row, col)) then
                board.placePiece(currentPiece, row, col);
                { Lanjutkan search }
                if (backtrack(idx + 1)) then
                    → true;
                { Rollback }
                board.removePiece(currentPiece, row, col);
→ false;

```

2. Penjelasan Implementasi

Pendekatan *brute force* dengan *backtracking* yang digunakan dalam program ini merupakan teknik pencarian rekursif yang mencoba semua kemungkinan secara sistematis. Pada setiap langkah, algoritma mencoba meletakkan kepingan pada seluruh posisi yang memungkinkan, lalu melanjutkan ke kepingan berikutnya dengan cara yang sama. Jika pada suatu titik tidak ditemukan solusi yang valid, maka program akan kembali ke posisi sebelumnya (*rollback*) dan mencoba alternatif lain hingga program kehabisan alternatif dan program menyatakan tidak terdapat solusi. Sebaliknya, program akan berhenti sesaat bertemu dengan salah satu solusi.

Proses ini dapat dibayangkan sebagai pohon keputusan (*state-space tree*), di mana setiap cabang mewakili suatu keputusan dalam peletakan kepingan. Jika suatu cabang gagal mencapai solusi, program akan memangkas cabang tersebut dan kembali ke cabang sebelumnya, lalu mencoba jalur lain. Selain itu, program ini juga menggunakan dua buah *constraint* (bukan heuristik) selaku bagaimana permainan puzzle ini dimainkan pada papan fisik:

1. *Piece* tidak boleh keluar dari papan
2. *Piece* tidak boleh menimpa *piece* lainnya

yang mana keduanya diperiksa menggunakan *function* `canPlacePiece()`.

Untuk memudahkan penulis dalam mengimplementasikan algoritma ini, penulis menggunakan pengembalian boolean (*true* atau *false*) untuk menandakan apakah solusi dari permainan ditemukan atau tidak. Bila solusi sudah ditemukan, maka program akan mengembalikan *true* hingga seluruh lapisan rekursi tertutup. Kedua potongan pseudocode ini merupakan *trigger* dari berhentinya algoritma:

<u>if</u> (board.isFull) <u>then</u> → <u>true</u>	<u>if</u> (backtrack(idx + 1)) <u>then</u> → <u>true</u> ;
---	---

Penulis juga menambahkan perhitungan iterasi untuk ditampilkan pada bagian *output* (disimpan pada atribut *iterCount*).

Perlu diperhatikan juga *object* penunjang implementasi algoritma ini dalam program: Piece dan Board (yang berfungsi sebagai komponen inti dalam proses pencarian solusi, akan dijelaskan lebih lanjut pada bagian **Source Code Program**)

Untuk merangkum penjelasan sebelumnya, berikut adalah langkah-langkah algoritma yang telah diimplementasikan:

1. Periksa apakah semua *piece* telah ditempatkan (*goal* dari *backtracking*)
 - Jika sudah, periksa apakah *board* sudah penuh
 - Jika penuh, solusi pun ditemukan (kembalikan *true*)
 - Jika tidak penuh, lakukan *rollback*
2. Ambil *piece* berikutnya, lalu coba tempatkan pada seluruh posisi *board* yang memungkinkan dengan:
 - Memastikan *piece* tidak keluar dari batas papan
 - Memastikan *piece* tidak menimpa *piece* lainnya

Jika valid atau *piece* dapat diletakkan, maka letakkan *piece* tersebut pada *board* dan lanjutkan proses ke *piece* selanjutnya secara rekursif.
3. Jika tidak terdapat solusi pada jalur saat ini, *rollback* (hapus *piece*) dan coba posisi lain
4. Ulangi proses hingga seluruh solusi ditemukan atau kemungkinan telah diuji (solusi tidak ditemukan).

Source Code Program

1. BruteForceSolver.java

Kelas BruteForceSolver bertanggung jawab untuk menjalankan algoritma *brute force* dengan *Backtracking* atau dalam kata lain merupakan implementasi dari penjelasan algoritma pada bagian **Implementasi Brute Force**.

Kelas ini memiliki metode `solve()` yang berperan sebagai *trigger* proses pencarian solusi, dan metode `backtrack(int idx)` yang menangani proses eksplorasi semua kemungkinan posisi dan orientasi kepingan pada papan.

```
// Method bantuan untuk memulai backtracking
public void solve() {
    boolean isSolved = backtrack(0);
    // Output hasil berdasarkan status solving
    if (isSolved) {
        board.printBoard();
        return;
    }
    System.out.println("Puzzle tidak memiliki solusi!");
}

// Method proses backtracking: mengembalikan true bila terdapat solusi
private boolean backtrack(int idx) {
    // Menambah hitungan iterasi
    iterCount++;

    // Basis: seluruh Piece sudah ditempatkan pada Board
    if (idx == listOfPieces.size()) {
        if (board.isFull()) {
            return true;
        }
        return false;
    }

    // Brute force seluruh kemungkinan transformasi Piece dan posisi pada Board
    Piece piece = listOfPieces.get(idx);
    List<Piece> rotatedAndMirroredPieces = piece.getRotationsAndMirrors();

    for (int i = 0; i < 8; i++) {
        Piece currentPiece = rotatedAndMirroredPieces.get(i);

        for (int row = 0; row < board.getRows(); row++) {
            for (int col = 0; col < board.getCols(); col++) {
                // Pengecekan penempatan piece selaku aturan IQ Puzzler Pro
                if (board.canPlacePiece(currentPiece, row, col)) {
                    board.placePiece(currentPiece, row, col);
                    // Melanjutkan proses searching (berhenti bila sudah menemukan solusi)
                    if (backtrack(idx + 1)) {
                        return true;
                    }
                    // Rollback: kembali pada level state-tree sebelumnya
                    board.removePiece(currentPiece, row, col);
                }
            }
        }
    }
}
```

```

    }
    }
}
return false;
}

```

2. Board.java

Kelas Board merepresentasikan papan permainan tempat kepingan-kepingan diletakkan. Board diimplementasikan sebagai grid dua dimensi bertipe data char (char[][] grid), di mana setiap sel berisi karakter kepingan yang telah ditempatkan atau karakter titik (.) yang menunjukkan ruang kosong.

```

public class Board {
    // Atribut Board
    private int rows, cols;
    private char[][] grid;

    // Konstruktor Board
    public Board(int rows, int cols) {
        this.rows = rows;
        this.cols = cols;
        grid = new char[rows][cols];

        // Mengisi grid dengan char '.' sebagai tanda kosong
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                grid[i][j] = '.';
            }
        }
    }
}

```

Kelas ini menyediakan beberapa metode utama:

- a. canPlacePiece(Piece piece, int row, int col): Memeriksa apakah kepingan dapat ditempatkan pada posisi tertentu.

```

public boolean canPlacePiece(Piece piece, int row, int col) {
    // Dimensi Piece
    char[][] pieceShape = piece.getShape();

    // Pengecekan
    for (int i = 0; i < pieceShape.length; i++) {
        for (int j = 0; j < pieceShape[i].length; j++) {
            // Tidak menimpa piece lain
            if (pieceShape[i][j] != '.') {
                // Tidak keluar dari Board
                if ((row + i < 0) || (col + j < 0) || (row + i >= rows) || (col + j >= cols)) {
                    return false;
                }
            }
        }
    }
}

```

```

    }
    // Cek apakah sel Board sudah ditempatkan atau belum
    if (grid[row + i][col + j] != '.') {
        return false;
    }
}
}
}
return true;
}

```

- b. `placePiece(Piece piece, int row, int col)`: Menempatkan kepingan ke papan jika valid.

```

public void placePiece(Piece piece, int row, int col) {
    // Dimensi Piece
    char[][] pieceShape = piece.getShape();

    // Menempatkan piece
    for (int i = 0; i < pieceShape.length; i++) {
        for (int j = 0; j < pieceShape[i].length; j++) {
            if (pieceShape[i][j] != '.') {
                grid[row + i][col + j] = pieceShape[i][j];
            }
        }
    }
}

```

- c. `removePiece(Piece piece, int row, int col)`: Menghapus kepingan dari papan (rollback) jika jalur pencarian tidak menghasilkan solusi.

```

public void removePiece(Piece piece, int row, int col) {
    // Dimensi Piece
    char[][] pieceShape = piece.getShape();

    // Menghapus piece
    for (int i = 0; i < pieceShape.length; i++) {
        for (int j = 0; j < pieceShape[i].length; j++) {
            if (pieceShape[i][j] != '.') {
                if (grid[row + i][col + j] == pieceShape[i][j]) {
                    grid[row + i][col + j] = '.';
                }
            }
        }
    }
}

```

d. `isFull()`: Memeriksa apakah seluruh papan telah terisi sebagai kondisi solusi.

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

e. `printBoard()`: Mencetak papan ke terminal dalam format grid (bewarna).

```
public void printBoard() {
    String defaultColor = "\033[0m";

    // Print masing-masing huruf sesuai warnanya
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            System.out.print(getColorCode(grid[i][j]) + grid[i][j] + defaultColor);
        }
        System.out.println();
    }
}
```

Dengan kelas ini, *solver* dapat berinteraksi dengan papan permainan dan memastikan bahwa semua aturan permainan dipatuhi.

3. Piece.java

Kelas `Piece` merepresentasikan kepingan puzzle yang digunakan dalam permainan. Setiap objek `Piece` menyimpan (berupa atribut):

1. Huruf label (misalnya A, B, C) untuk identifikasi.
2. Bentuk kepingan dalam bentuk array dua dimensi (`char[][] shape`).

```
public class Piece {
    // Atribut Piece
    private char label;
    private char[][] shape;

    // Konstruktor Piece
    public Piece(char label, char[][] shape) {
        this.label = label;
        this.shape = shape;
    }
}
```


Fitur utama dalam kelas ini antara lain:

1. Rotasi dan pencerminan kepingan, yang memungkinkan setiap kepingan diuji dalam 8 konfigurasi berbeda (4 rotasi + 4 pencerminan).
2. Metode `getRotationsAndMirrors()` yang mengembalikan daftar seluruh kemungkinan transformasi dari suatu kepingan.

```
public List<Piece> getRotationsAndMirrors() {
    // Pembuatan list shape hasil seluruh konfigurasi rotasi dan mirror
    List<Piece> transformedShapes = new ArrayList<>();

    // Proses loading seluruh konfigurasi rotasi dan pencerminan
    char[][] currentShape = shape;

    for (int i = 0; i < 4; i++) {
        currentShape = rotate(currentShape);
        transformedShapes.add(new Piece(label, currentShape));
        transformedShapes.add(new Piece(label, mirror(currentShape)));
    }
    return transformedShapes;
}

// Method untuk merotasi Piece
public char[][] rotate(char[][] currentShape) {
    // Dimensi shape
    int shapeRow = currentShape.length;
    int shapeCol = currentShape[0].length;

    // Pembuatan shape hasil rotasi
    char[][] rotatedShape = new char[shapeCol][shapeRow];

    for (int i = 0; i < shapeRow; i++) {
        for (int j = 0; j < shapeCol; j++) {
            rotatedShape[j][shapeRow - i - 1] = currentShape[i][j];
        }
    }
    return rotatedShape;
}

// Method untuk mencerminkan piece
public char[][] mirror(char[][] currentShape) {
    // Dimensi shape
    int shapeRow = currentShape.length;
    int shapeCol = currentShape[0].length;

    // Pembuatan shape hasil mirror
    char[][] mirroredShape = new char[shapeRow][shapeCol];

    for (int i = 0; i < shapeRow; i++) {
        for (int j = 0; j < shapeCol; j++) {
            mirroredShape[i][j] = currentShape[i][shapeCol - j - 1];
        }
    }
    return mirroredShape;
}
}
```

Dengan adanya kelas ini, program dapat secara dinamis menyesuaikan setiap kepingan untuk mencoba semua kemungkinan peletakan di papan permainan.

4. FileIO.java

Kelas FileIO bertanggung jawab untuk membaca input puzzle dari file, serta menyimpan hasil solusi dalam bentuk file teks (.txt) dan gambar (.png).

Metode utama yang terdapat dalam kelas ini:

1. `loadInputFile(String filename)`: Membaca file input dan menyusun objek Board serta daftar Piece berdasarkan data yang tersedia.

```
public static boolean loadInputFile(String filename) {
    try {
        BufferedReader reader = new BufferedReader(new FileReader(filename));

        // Membaca N M P: boardRows, boardCols, totalPieces
        String[] firstLine = reader.readLine().trim().split(" ");
        if (firstLine.length != 3) {
            reader.close();
            System.out.println("Error: Format baris pertama (N M P) salah");
            return false;
        }

        boardRows = Integer.parseInt(firstLine[0]); // N
        boardCols = Integer.parseInt(firstLine[1]); // M
        totalPieces = Integer.parseInt(firstLine[2]); // P

        // Validasi input N M P
        if (boardRows <= 0 || boardCols <= 0 || totalPieces <= 0 || totalPieces > 26) {
            reader.close();
            System.out.println("Error: Dimensi papan atau jumlah pieces tidak valid (maksimal 26)!");
            return false;
        }

        // Read mode (baris kedua) harus "DEFAULT"
        String mode = reader.readLine().trim();
        if (!mode.equals("DEFAULT")) {
            reader.close();
            System.out.println("Error: Mode solver harus DEFAULT!");
            return false;
        }

        // Construct Board
        board = new Board(boardRows, boardCols);
        pieces = new ArrayList<>();

        // Baca seluruh Piece
        List<String> pieceLines = new ArrayList<>();
        char currentLetter = '\0';
        String line;
```

```

while ((line = reader.readLine()) != null) {
    if (line.trim().isEmpty()) continue;

    // Mendeteksi char baru pertama (penanda Piece baru)
    char firstChar = detectFirstCharacter(line);

    if (firstChar != currentLetter) {
        if (!pieceLines.isEmpty()) {
            pieces.add(createPieceFromLines(currentLetter, pieceLines));
            pieceLines.clear();
        }
        currentLetter = firstChar;
    }

    // Validasi input hanya char 'A' - 'Z' dan ' '
    if (!line.matches("[A-Z ]+")) {
        reader.close();
        System.out.println("Error: Format blok tidak valid: " + line);
        return false;
    }

    pieceLines.add(line.replace(" ", "."));
}

if (!pieceLines.isEmpty()) {
    pieces.add(createPieceFromLines(currentLetter, pieceLines));
}

// Memastikan jumlah Piece sama dengan input P
if (pieces.size() != totalPieces) {
    reader.close();
    System.out.println("Error: Jumlah blok tidak sesuai (diharapkan: " +
        totalPieces + ", ditemukan: " + pieces.size() + ").");
    return false;
}

reader.close();
return true;
} catch (IOException e) {
    System.out.println("Error: File tidak ditemukan atau tidak bisa dibuka.");
    return false;
}
}

```

2. `saveSolutionToFile(String inputFilename)`: Menyimpan hasil solusi dalam format teks dengan grid yang sama seperti di terminal.

```

public static void saveSolutionToFile(String inputFilename) {
    if (board == null) {
        System.out.println("Error: Tidak ada papan yang dapat disimpan");
        return;
    }

    // Memastikan directory sudah ada
    File solutionDir = new File("test/solution_file/");
}

```

```

        if (!solutionDir.exists()) {
            solutionDir.mkdirs();
        }

        File file = new File(inputFilename);
        String baseName = file.getName().replace(".txt", "");
        String solutionFilename = "test/solution_file/solution_" + baseName + ".txt";

        try (BufferedWriter writer = new BufferedWriter(new FileWriter(solutionFilename))) {
            for (char[] row : board.getGrid()) {
                for (char cell : row) {
                    writer.write(cell + " ");
                }
                writer.newLine();
            }
            System.out.println("Solusi berhasil di simpan: " + solutionFilename);
            System.out.println();
        } catch (IOException e) {
            System.out.println("Error: Gagal menyimpan file solusi");
        }
    }
}

```

3. `saveSolutionAsImage(String inputFilename)`: Menghasilkan gambar solusi dengan warna berbeda untuk setiap kepingan dan menyimpannya dalam format .png.

```

public static void saveSolutionAsImage(String inputFilename) {
    if (board == null) {
        System.out.println("Error: Tidak ada papan yang dapat disimpan");
        return;
    }

    // Memastikan directory sudah ada
    File solutionDir = new File("test/solution_image/");
    if (!solutionDir.exists()) {
        solutionDir.mkdirs();
    }

    File file = new File(inputFilename);
    String baseName = file.getName().replace(".txt", "");
    String solutionFilename = "test/solution_image/solution_" + baseName + ".png";

    // Image size
    int cellSize = 50;
    int rows = board.getRows();
    int cols = board.getCols();
    int width = cols * cellSize;
    int height = rows * cellSize;

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

    // Menggambar papan
    char[][] grid = board.getGrid();
    Map<Character, Color> colorMap = new HashMap<>();
    Color[] availableColors = {

```

```

        Color.RED, Color.BLUE, Color.GREEN, Color.ORANGE, Color.MAGENTA, Color.CYAN,
        new Color(255, 165, 0), new Color(128, 0, 128), new Color(255, 215, 0),
        new Color(0, 128, 128), new Color(128, 128, 0), new Color(255, 105, 180)
    };

    int colorIndex = 0;
    for (char[] row : grid) {
        for (char cell : row) {
            if (cell != '.' && !colorMap.containsKey(cell)) {
                colorMap.put(cell, availableColors[colorIndex % availableColors.length]);
                colorIndex++;
            }
        }
    }

    for (int row = 0; row < rows; row++) {
        for (int col = 0; col < cols; col++) {
            char cell = grid[row][col];
            graphics.setColor(colorMap.getOrDefault(cell, Color.WHITE));
            graphics.fillRect(col * cellSize, row * cellSize, cellSize, cellSize);
            graphics.setColor(Color.BLACK);
            graphics.drawRect(col * cellSize, row * cellSize, cellSize, cellSize);
        }
    }

    graphics.dispose();

    // Simpan Image
    try {
        ImageIO.write(image, "png", new File(solutionFilename));
        System.out.println("Image solusi berhasil disimpan: " + solutionFilename);
        System.out.println();
    } catch (IOException e) {
        System.out.println("Error: Gagal menyimpan file solusi");
    }
}

```

Kelas ini memastikan bahwa program dapat membaca data puzzle dari file serta menyimpan hasil solusi dengan format yang mudah dianalisis.

5. Main.java

Kelas Main adalah entry point program, di mana semua komponen diinisialisasi dan solver dijalankan. Alur eksekusi dalam Main.java adalah sebagai berikut:

1. Membaca file input menggunakan `FileIO.loadInputFile(filename)`.
2. Menampilkan papan awal sebelum solver dijalankan.
3. Menjalankan algoritma brute force menggunakan `BruteForceSolver.solve()`.
4. Menampilkan hasil di terminal serta menyimpan solusi dalam file teks dan gambar jika solusi ditemukan.

```

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);

    // Input nama file
    System.out.print("Nama file input (.txt): ");
    String filename = scanner.nextLine();
    System.out.println();

    if (!FileIO.loadInputFile(filename)) {
        scanner.close();
        return;
    }

    // Brute force dan perhitungan waktu eksekusi
    BruteForceSolver solver = new BruteForceSolver(FileIO.board, FileIO.pieces);
    long startTime = System.currentTimeMillis();
    solver.solve();
    long endTime = System.currentTimeMillis();

    // Output hasil waktu pencarian dan jumlah iterasi
    System.out.println("\nWaktu pencarian: " + (endTime - startTime) + " ms");
    System.out.println();
    System.out.println("Banyak kasus yang ditinjau: " + solver.getIterationCount());
    System.out.println();

    // Output file .txt dan gambar bila terdapat solusi
    if (solver.getBoard().isFull()){
        // Output file .txt
        System.out.print("Apakah anda ingin menyimpan solusi (.txt)? (ya/tidak): ");
        String outputFile = scanner.nextLine().trim();

        if (outputFile.equalsIgnoreCase("ya")) {
            FileIO.saveSolutionToFile(filename);
        }

        // Output file gambar
        System.out.print("Apakah anda ingin menyimpan solusi (.png)? (ya/tidak): ");
        String outputImage = scanner.nextLine().trim();

        if (outputImage.equalsIgnoreCase("ya")) {
            FileIO.saveSolutionAsImage(filename);
        }
    }

    scanner.close();
}

```

Dengan adanya Main.java, semua komponen program dapat saling berinteraksi dengan baik untuk menyelesaikan puzzle secara otomatis.

Test Case: Input & Output

1. Terminal Input and Output

Test Case #1: Spesifikasi

Input	Output
<pre>test > test1.txt 1 5 5 7 2 DEFAULT 3 A 4 AA 5 B 6 BB 7 C 8 CC 9 D 10 DD 11 EE 12 EE 13 E 14 FF 15 FF 16 F 17 GGG</pre>	<pre>Nama file input (.txt): test/test1.txt AABBD AEBDD EECCF EECCF GGGFF Waktu pencarian: 52 ms Banyak kasus yang ditinjau: 1715 Apakah anda ingin menyimpan solusi (.txt)? (ya/tidak): ya Solusi berhasil di simpan: test/solution_file/solution_test1.txt Apakah anda ingin menyimpan solusi (.png)? (ya/tidak): ya Image solusi berhasil disimpan: test/solution_image/solution_test1.png</pre>

Test Case #2: Board 5 x 11 dengan 12 Piece

Input	Output
<pre>test > test2.txt 1 5 11 12 2 DEFAULT 3 AA 4 A 5 A 6 BB 7 BBB 8 C 9 CC 10 C 11 D 12 DD 13 DD 14 E 15 E 16 EE 17 E 18 FF 19 F 20 F 21 F 22 GGGG 23 G 24 H 25 HH 26 HH 27 I 28 II 29 JJJ 30 J J 31 K 32 KK 33 K 34 L 35 L 36 LLL</pre>	<pre>Nama file input (.txt): test/test2.txt AAABBCGGGG FFABBECCGD FJJBHELDK FJHHIELDKK FJHHIELLLK Waktu pencarian: 7910 ms Banyak kasus yang ditinjau: 2801760 Apakah anda ingin menyimpan solusi (.txt)? (ya/tidak): ya Solusi berhasil di simpan: test/solution_file/solution_test2.txt Apakah anda ingin menyimpan solusi (.png)? (ya/tidak): ya Image solusi berhasil disimpan: test/solution_image/solution_test2.png</pre>

Test Case #3: Board 6 x 6 dengan 7 Piece

Input	Output
<pre> test > test3.txt 1 6 6 7 2 DEFAULT 3 AAA 4 A 5 BB 6 BB 7 B 8 CC 9 CC 10 CC 11 C 12 DDD 13 EEEE 14 E 15 F F 16 FFFF 17 GG 18 GGG 19 G </pre>	<pre> Nama file input (.txt): test/test3.txt FFFFEE FGGFAE GGGAEE DGBBAE DBBCCC DBCCCC Waktu pencarian: 7195 ms Banyak kasus yang ditinjau: 2972988 Apakah anda ingin menyimpan solusi (.txt)? (ya/tidak): ya Solusi berhasil di simpan: test/solution_file/solution_test3.txt Apakah anda ingin menyimpan solusi (.png)? (ya/tidak): ya Image solusi berhasil disimpan: test/solution_image/solution_test3.png </pre>

Test Case #4: Board 5 x 5 dengan 5 Piece

Input	Output
<pre> test > test4.txt 1 5 5 5 2 DEFAULT 3 BB 4 SS 5 FF 6 WW 7 A AAA 8 A A 9 AAAAA 10 A A 11 AAA A </pre>	<pre> Nama file input (.txt): test/test4.txt ABAAA ABAFF AAAAA WWASA AAAAA Waktu pencarian: 4344 ms Banyak kasus yang ditinjau: 1964288 Apakah anda ingin menyimpan solusi (.txt)? (ya/tidak): ya Solusi berhasil di simpan: test/solution_file/solution_test4.txt Apakah anda ingin menyimpan solusi (.png)? (ya/tidak): ya Image solusi berhasil disimpan: test/solution_image/solution_test4.png </pre>

Test Case #5: Tidak ada solusi

Input	Output
-------	--------

<pre>test > test5.txt 1 3 3 2 2 DEFAULT 3 AAA 4 AAA 5 BBBB 6 B</pre>	<pre>Nama file input (.txt): test/test5.txt Puzzle tidak memiliki solusi! Waktu pencarian: 0 ms Banyak kasus yang ditinjau: 17</pre>
---	---

Test Case #6: Kesalahan pada format input

Input	Output
<pre>test > test6.txt 1 3 3 2 1 2 DEFAULT 3 ∨ AAA 4 AA 5 ∨ BBB 6 B</pre>	<pre>Nama file input (.txt): test/test6.txt Error: Format baris pertama (N M P) salah</pre>

Test Case #7: Salah input (jumlah Piece)

Input	Output
<pre>test > test7.txt 1 3 3 2 2 DEFAULT 3 AAA 4 AA 5 AAA</pre>	<pre>Nama file input (.txt): test/test7.txt Error: Jumlah blok tidak sesuai (diharapkan: 2, ditemukan: 1).</pre>

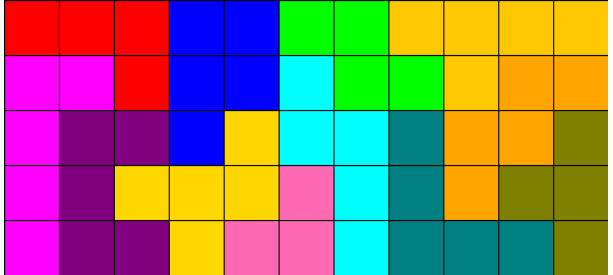
2. File and Image Output

Test Case #1: Spesifikasi

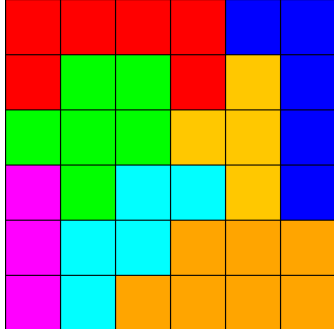
File Output	Image Output
-------------	--------------

<pre>test > solution_file > solution_test1.txt 1 A A B B D 2 A E B D D 3 E E C C F 4 E E C F F 5 G G G F F</pre>	
---	--

Test Case #2: Board 5 x 11 dengan 12 Piece

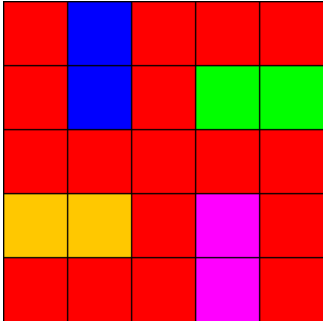
File Output	Image Output
<pre>test > solution_file > solution_test2.txt 1 A A A B B C C G G G G 2 F F A B B E C C G D D 3 F J J B H E E L D D K 4 F J H H H I E L D K K 5 F J J H I I E L L L K</pre>	

Test Case #3: Board 6 x 6 dengan 7 Piece

File Output	Image Output
<pre>test > solution_file > solution_test3.txt 1 F F F F E E 2 F G G F A E 3 G G G A A E 4 D G B B A E 5 D B B C C C 6 D B C C C C</pre>	

Test Case #4: Board 5 x 5 dengan 5 Piece

File Output	Image Output
-------------	--------------

<pre>test > solution_file > solution_test4.txt 1 A B A A A 2 A B A F F 3 A A A A A 4 W W A S A 5 A A A S A</pre>	
---	--

Lampiran

Repository: [OxNathaniel/Tucil1_13523013](https://github.com/OxNathaniel/Tucil1_13523013)

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	✓	
---	----------------------------------	---	--