

Laporan Tugas Kecil 1

IF2211 Strategi Algoritma



Disusun oleh :

Ahsan Malik Al Farisi - 13523074

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
SEMESTER 2 TAHUN 2024/2025

Daftar Isi

Daftar Isi.....	1
BAB I.....	1
Deskripsi Masalah.....	1
BAB II.....	2
Teori Singkat.....	2
2.1. Algoritma Brute Force.....	2
BAB III.....	2
Implementasi.....	2
3.1. Konsep Algoritma.....	2
3.2. Implementasi.....	3
3.3. Source Code.....	4
BAB IV.....	10
Hasil Program.....	10
4.1. Case 1.....	10
4.2. Case 2.....	11
4.3. Case 3.....	12
4.4. Case 4.....	13
4.5. Case 5.....	14
4.6. Case 6.....	14
4.7. Case 7.....	15
Lampiran.....	16

BAB I

Deskripsi Masalah

IQ Puzzler Pro adalah permainan papan yang diproduksi oleh perusahaan Smart Games. Tujuan dari permainan ini adalah pemain harus dapat mengisi seluruh papan dengan piece (blok puzzle) yang telah tersedia.



Gambar 1. Permainan IQ Puzzler Pro

Komponen penting dari permainan IQ Puzzler Pro terdiri dari:

- Board (Papan) – Board merupakan komponen utama yang menjadi tujuan permainan dimana pemain harus mampu mengisi seluruh area papan menggunakan blok-blok yang telah disediakan.
- Blok/Piece – Blok adalah komponen yang digunakan pemain untuk mengisi papan kosong hingga terisi penuh. Setiap blok memiliki bentuk yang unik dan semua blok harus digunakan untuk menyelesaikan puzzle.

Permainan dimulai dengan papan yang kosong. Pemain dapat meletakkan blok puzzle sedemikian sehingga tidak ada blok yang bertumpang tindih (kecuali dalam kasus 3D). Setiap blok puzzle dapat dirotasikan maupun dicerminkan. Puzzle dinyatakan selesai jika dan hanya jika papan terisi penuh dan seluruh blok puzzle berhasil diletakkan.

Tugas anda adalah menemukan cukup satu solusi dari permainan IQ Puzzler Pro dengan menggunakan algoritma Brute Force, atau menampilkan bahwa solusi tidak ditemukan jika tidak ada solusi yang mungkin dari puzzle.

BAB II

Teori Singkat

2.1. Algoritma Brute Force

Algoritma brute force adalah algoritma yang memiliki pendekatan yang lurus atau lempang (straightforward) untuk memecahkan suatu persoalan. Pada dasarnya algoritma brute force bukanlah algoritma selalu optimal karena sangat mengandalkan kemampuan komputasi dalam pemecahan masalahnya, sehingga kata *force* dalam *brute force* cukup menjelaskan karakteristiknya. Selain itu algoritma ini kerap juga disebut algoritma naive. Walaupun begitu algoritma brute force adalah algoritma yang memiliki beberapa kelebihan, yaitu sederhana dan mudah diimplementasikan sehingga jarang ada persoalan yang tidak bisa diselesaikan oleh algoritma ini.

BAB III

Implementasi

3.1. Konsep Algoritma

Algoritma ini menggunakan rekursif dan backtracking dalam menyusun potongan ke dalam papan dengan mempertimbangkan semua kemungkinan variasi dan juga mencoba setiap titik kosong yang berada di papan.

- Program ini akan membaca file untuk kemudian diubah ke Array of String kemudian diubah ke List of Matrix Character untuk kemudian diubah ke Set of Coordinate. Sehingga program ini akan menggunakan koordinat sebagai dasar penempatan potongannya.
- Board akan dibentuk dalam sesuai dengan ukuran dalam bentuk Matrix of Character dan juga akan dibentuk List of Pieces yang merupakan kumpulan potongan dengan isinya, yaitu characterId, warna, dan Set of Coordinate.
- Ada basis rekursi di awal kode untuk mengecek jika semua potongan sudah dicoba untuk kemudian dicek lagi apakah papan sudah penuh. Jika penuh, maka solusi sudah ditemukan dan akan return true, jika tidak akan melanjutkan pencarian solusi lainnya.
- Program akan mengambil potongan sesuai indeks iterasi dan akan dilakukan iterasi dari koordinat kolom kosong yang pertama hingga paling akhir.
- Tiap potongan akan dicek apakah bisa ditempatkan di papan, jika bisa maka program akan memanggil fungsi solve(pieceIndex + 1) untuk melanjutkan potongan berikutnya.
- Jika variasi pertama potongan tidak bisa ditaruh di papan maka untuk setiap koordinat potongan akan dilakukan 2 iterasi:
 - Iterasi untuk membalikkan potongan:
 - flip = 0 Tidak dibalik

- flip = 1 Dibalik secara horizontal
 - flip = 2 Dibalik secara vertikal
- Iterasi untuk memutar potongan yang bersarang di dalam iterasi membalikkan potongan:
 - Mencoba 4 kemungkinan rotasi (0° , 90° , 180° , 270°).
- Jika setelah dirotasikan dan dibalikkan potongan masih tidak dapat ditaruh di papan maka program akan mencoba titik selanjutnya pada papan.
- Jika sudah pindah hingga ke titik akhir di papan dan masih tidak bisa, maka program akan kembali ke potongan sebelumnya (backtrack) untuk mencoba variasi membalikkan dan rotasi selanjutnya, juga mencoba menaruh di titik papan selanjutnya.

3.2. Implementasi

```
function solve(pieceIndex):
  if (pieceIndex >= pieces.size()) then
    return Board.isFull()

  piece ← pieces.get(pieceIndex)
  firstEmptyCol ← Board.findFirstEmptyCol()

  y traversal [board height - 1..0]:
    x traversal [0..board width - 1]:
      flip traversal [0 to 2]:
        flippedPiece ← piece

        if flip == 1 then
          flippedPiece ← flipPieceHorizontal()
        else if flip == 2 then
          flippedPiece ← flipPieceVertical()

        rotatedPiece ← flippedPiece

        rotation traversal [0..3]:
          iteration count ← iteration count + 1
          if (canPlacePiece(rotatedPiece, x, y)) then
            placePiece(rotatedPiece, x, y)

            if (solve(pieceIndex + 1))
              returns true then

              removePiece(rotatedPiece, x, y) #backtrack

            rotatedPiece ← rotatedPiece.rotate90Clockwise()

  return false
```

3.3. Source Code

```
1 public static void main(String[] args) throws Exception{
2     // Read File
3     String inputPath = "test\\input\\";
4     System.out.println("Masukkan nama file dari folder input (1.txt)");
5
6     try(Scanner scanner = new Scanner(System.in)){
7         String filePath = scanner.nextline();
8         inputPath = inputPath.concat(filePath);
9         String[] config = IO.readConfig(inputPath);
10
11         // Take element
12         String stringNMP = config[0];
13         // String boardtype = config[1];
14         String stringPieces = config[2];
15         // System.out.println(stringPieces);
16
17         // Convert to Matrix
18         String[] arrayPieces = stringPieces.split("\n");
19         List<Char[][]> matrixPieces = IO.piecesToMatrix(arrayPieces);
20
21         // Convert to Coordinate
22         List<Piece> pieces = new ArrayList<Piece>();
23         for(int i=0; i < matrixPieces.size(); i++){
24             Set<Coordinate> pieceCoordinate = new HashSet<Coordinate>();
25
26             boolean foundChar = false;
27             Character character = null;
28             int j = 0;
29             while(!foundChar){
30                 character = matrixPieces.get(i)[0][j];
31                 if(character == '.'){
32                     j++;
33                 }
34                 else{
35                     foundChar = true;
36                 }
37             }
38             character = matrixPieces.get(i)[0][j];
39
40             pieceCoordinate = Piece.matrixToCoordinate(matrixPieces.get(i));
41             Color color = IO.COLORS[i];
42             Piece newPiece = new Piece(pieceCoordinate, character, color);
43             pieces.add(i, newPiece);
44         }
45
46         // Setup Board
47         String[] splitNMP = stringNMP.split(" ");
48         if(splitNMP.length != 3){
49             System.err.println("Config is not valid");
50             System.exit(0);
51         }
52
53         int N = Integer.parseInt(splitNMP[0]);
54         int M = Integer.parseInt(splitNMP[1]);
55         int P = Integer.parseInt(splitNMP[2]);
56         if(N<0 || M<0 || P<0){
57             System.err.println("NMP is not valid");
58             System.exit(0);
59         }
60
61         new Board(N, M);
62         Solver.setPieces(pieces);
63
64         if(Solver.checkTotalPieceAgainstBoardSize()){
65             System.out.println("Unsolvable, Matrix piece is not equal to the board
66 size");
67             System.exit(0);
68         }
69
70         System.out.println("Searching for the solution, this may take a while...");
71         // Start Timer
72         long startTime = System.nanoTime();
73         // Solver
74         if(Solver.solve()){
75             System.out.println("Finished one solution found!");
76         }else{
77             System.out.println("Didn't found any solution");
78         }
79
80         // Finish Timer
81         long endTime = System.nanoTime();
82         long executionTime
83             = (endTime - startTime) / 1000000;
84         System.out.println("Execution time: " + executionTime + "ms");
85         System.out.println("Iteration : " + Solver.getIterationCount());
86
87         // Output File
88         Character[][] solutionMatrix = Board.getBoard();
89         Map<Character, String> colorMap = new HashMap<>();
90         for (char c = 'A'; c <= 'Z'; c++) {
91             colorMap.put(c, IO.TEXTCOLORS[c - 'A']);
92         }
93
94         Piece.printColorizedMatrix(solutionMatrix, colorMap);
95
96         while(true){
97             try {
98                 IO.saveMatrixToFile(solutionMatrix, pieces);
99                 break;
100             } catch (Exception e) {
101                 System.err.println(e);
102             }
103         }
104     }
105 }
106
107 }
108 catch(Exception e){
109     System.err.println(e);
110 }
111 }
112 }
```

Gambar 2. Main Code

```
1 public class Solver{
2     private static List<Piece> pieces;
3     static int iterationCount = 0;
4
5     public static void setPieces(List<Piece> pieceList) {
6         pieces = pieceList;
7     }
8
9     public static int getIterationCount(){
10         return iterationCount;
11     }
12
13     public static boolean solve(int pieceIndex) {
14         if (pieceIndex >= pieces.size()) {
15             return Board.isFull(); // kalau index sudah melebihi sizanya cek apakah boardnya sudah terisi penuh
16         }
17
18         Piece piece = pieces.get(pieceIndex);
19
20         Coordinate firstEmptyCol = Board.findFirstEmptyCol();
21         // System.out.println();
22
23         for (int y = Board.getHeight() - firstEmptyCol.getY() - 1; y >= 0; y--) {
24             for (int x = firstEmptyCol.getX(); x < Board.getWidth(); x++) {
25                 for(int flip = 0; flip < 3; flip++){
26                     Piece flippedPiece = piece;
27
28                     if(flip == 1){
29                         flippedPiece = flippedPiece.flipPieceHorizontal();
30                     }else if (flip==2){
31                         flippedPiece = flippedPiece.flipPieceVertical();
32                     }
33
34                     Piece rotatedPiece = flippedPiece;
35                     for (int rotation = 0; rotation < 4; rotation++) { // Coba semua rotasi
36                         iterationCount++;
37                         if (canPlacePiece(rotatedPiece, x, y)) {
38                             Board.placePiece(rotatedPiece, x, y);
39                             if (solve(pieceIndex + 1)) {
40                                 return true;
41                             }
42                         }
43                         Board.removePiece(rotatedPiece, x, y); // Backtrack
44                     }
45                     rotatedPiece = rotatedPiece.rotate90Clockwise();
46                 }
47             }
48         }
49     }
50
51     return false;
52 }
53
54 public static boolean canPlacePiece(Piece piece, int initX, int initY){
55     Set<Coordinate> pieceCoordinate = piece.getPieceShape();
56     for (Coordinate coordinate : pieceCoordinate) {
57         int x = initX + coordinate.getX();
58         int y = (Board.getHeight() - 1 - (initY + coordinate.getY()));
59         int boardWidth = Board.getWidth();
60         int boardHeight = Board.getHeight();
61         if (x<0 || x>= boardWidth || y < 0 || y >= boardHeight){
62             return false;
63         }
64         if(!Board.isEmpty(x, y)){
65             return false;
66         }
67         // System.out.println("Can place piece at X:" + x + "Y:" + y);
68     }
69     return true;
70 }
71
72 public static boolean checkTotalPieceAgainstBoardSize(){
73     int count = 0;
74     for (Piece piece : pieces) {
75         Set<Coordinate> tempCoordinate = piece.getPieceShape();
76         for (@SuppressWarnings("unused") Coordinate Coordinate : tempCoordinate) {
77             count++;
78         }
79     }
80     int boardsize = Board.getWidth() * Board.getHeight();
81     if((boardsize) == count){
82         return true;
83     }
84     else{
85         return false;
86     }
87 }
```

Gambar 3. Solver Code

```

1  public class Piece{
2      Set<Coordinate> shape;
3      Character character;
4      Color color;
5
6      public Piece(Set<Coordinate> shape, Character character, Color color){
7          this.shape = shape;
8          this.character = character;
9          this.color = color;
10     }
11
12     public Set<Coordinate> getPieceShape(){
13         return shape;
14     }
15     public Character getPieceCharacter(){
16         return character;
17     }
18     public Color getColor(){
19         return color;
20     }
21
22
23
24     public static Set<Coordinate> matrixToCoordinate(char[][] piece){
25         Set<Coordinate> pieceCoordinate = new HashSet<Coordinate>();
26         int height = piece.length;
27         int width = piece[0].length;
28         int x = 0;
29         int y = 0;
30
31         for(int r = 0; r < height; r++){
32             for(int c = 0; c < width; c++){
33                 if (piece[r][c] != '.'){
34                     x = c ;
35                     y = height - r - 1;
36                     Coordinate newCoordinate = new Coordinate(x, y);
37                     pieceCoordinate.add(newCoordinate);
38                 }
39             }
40         }
41         return pieceCoordinate;
42     }
43
44     public Piece rotate90Clockwise(){
45         Set<Coordinate> rotatedShape = new HashSet<>();
46         for (Coordinate coordinate : shape) {
47             Coordinate rotatedCoordinate = coordinate.rotate90Clockwise();
48             rotatedShape.add(rotatedCoordinate);
49         }
50         return new Piece(rotatedShape, character, color);
51     }
52
53     public Piece flipPieceHorizontal(){
54         Set<Coordinate> rotatedShape = new HashSet<>();
55         for (Coordinate coordinate : shape) {
56             Coordinate rotatedCoordinate = coordinate.flipHorizontal();
57             rotatedShape.add(rotatedCoordinate);
58         }
59         return new Piece(rotatedShape, character, color);
60     }
61
62     public Piece flipPieceVertical(){
63         Set<Coordinate> rotatedShape = new HashSet<>();
64         for (Coordinate coordinate : shape) {
65             Coordinate rotatedCoordinate = coordinate.flipVertical();
66             rotatedShape.add(rotatedCoordinate);
67         }
68         return new Piece(rotatedShape, character, color);
69     }
70
71     public Set<Coordinate> translateCoordinates(int dx, int dy){
72         Set<Coordinate> newShape = new HashSet<Coordinate>();
73         for(Coordinate coord : shape){
74             newShape.add(new Coordinate(coord.x + dx, coord.y + dy));
75         }
76         return newShape;
77     }

```

Gambar 4. Class Piece


```

1 public class Board{
2     private static Character[][] board;
3     private static int width;
4     private static int height;
5
6     public Board(int n, int m) {
7         height = n;
8         width = m;
9         board = new Character[n][m];
10        fillBoard(board, 'o'); // Fill board with default character
11    }
12
13    public static Character[][] getBoard() {
14        return board;
15    }
16    public static int getWidth() {
17        return width;
18    }
19    public static int getHeight() {
20        return height;
21    }
22
23    public static Coordinate findFirstEmptyCol() {
24        for (int y = 0; y < getHeight(); y++) {
25            for (int x = 0; x < getWidth(); x++) {
26                if (isEmpty(x, y)) {
27                    return new Coordinate(x, y); // Return the first row that contain
28                    // an empty space
29                }
30            }
31            return new Coordinate(0, 0); // If no empty space is found
32        }
33
34        public static boolean isEmpty(int x, int y){
35            return board[y][x] == 'o';
36        }
37
38        public static boolean isFull(){
39            for (Character[] characters : board) {
40                for (Character character : characters) {
41                    if(character == 'o'){
42                        return false;
43                    }
44                }
45            }
46            return true;
47        }
48
49        public static void fillBoard(Character[][] board, Character val){
50            int n = board.length;
51            int m = board[0].length;
52
53            for(int i = 0; i < n; i++){
54                for(int j = 0; j < m; j++){
55                    board[i][j] = val;
56                }
57            }
58        }
59
60        public static void placePiece(Piece piece, int startX, int startY) {
61            Set<Coordinate> pieceCoordinate = piece.getPieceShape();
62            for (Coordinate coord : pieceCoordinate) {
63                int x = startX + coord.getX();
64                int y = (Board.getHeight() - 1 - (startY + coord.getY()));
65                board[y][x] = piece.character;
66            }
67        }
68
69        public static void removePiece(Piece piece, int startX, int startY) {
70            Set<Coordinate> pieceCoordinate = piece.getPieceShape();
71            for (Coordinate coord : pieceCoordinate) {
72                int x = startX + coord.getX();
73                int y = (Board.getHeight() - 1 - (startY + coord.getY()));
74                board[y][x] = 'o';
75            }
76        }
77    }

```

Gambar 5. Class Board

```

1 public class Coordinate {
2     int x,y;
3
4     public Coordinate(int x, int y){
5         this.x = x ;
6         this.y = y;
7     }
8
9     public int getX(){
10        return x;
11    }
12    public int getY(){
13        return y;
14    }
15
16    public Coordinate rotate90Clockwise(){
17        return new Coordinate(y, -x);
18    }
19    public Coordinate flipHorizontal(){
20        return new Coordinate(x, -y);
21    }
22
23    public Coordinate flipVertical(){
24        return new Coordinate(-x, y);
25    }

```

Gambar 6. Class Coordinate

```

1 public static void saveMatrixToFile(Character[][] matrix, List<Piece> pieces) throws
IOException{
2     Scanner scanner = new Scanner(System.in);
3     String directoryPath = "test\\output\\";
4     String outputPath = "";
5     outputPath = outputPath.concat(directoryPath);
6
7     while (true) {
8         System.out.println("Save Hasil ke file?(Y/n)");
9         char save = scanner.nextLine().charAt(0);
10        if (save == 'Y' || save == 'y') {
11            System.out.println("Masukkan nama file e.g (output): ");
12            String filename = scanner.nextLine();
13            outputPath = outputPath.concat(filename);
14            String outputPathTXT = outputPath.concat(".txt");
15            String outputPathPNG = outputPath.concat(".png");
16
17            boolean isFileExists = checkFilePath(outputPathTXT);
18            if (isFileExists){
19                System.out.println("Terdapat file dengan nama yang sama apakah anda
ingin overwrite? (Y/N)");
20                char overwrite = scanner.next().charAt(0);
21                if (overwrite == 'Y' || overwrite == 'y'){
22                    writeMatrixToFile(matrix, outputPathTXT);
23                    matrixToImage(matrix, pieces, outputPathPNG);
24                    System.out.println("File berhasil di simpan pada " + outputPat
h);
25                }
26                break;
27            }
28            else if (overwrite == 'n' || overwrite == 'N') {
29                System.out.println("File tidak disimpan\n");
30                break;
31            }
32            else {
33                System.out.println("Masukkan 'Y' untuk ya atau 'n' untuk tidak
\n");
34                break;
35            }
36        }
37        else{
38            writeMatrixToFile(matrix, outputPathTXT);
39            matrixToImage(matrix, pieces, outputPathPNG);
40            System.out.println("File berhasil di simpan pada " + outputPath);
41        }
42        break;
43    }
44
45    else if (save == 'n' || save == 'N') {
46        System.out.println("File tidak disimpan.\n");
47        break; // Exit the loop without saving
48    }
49    else {
50        System.out.println("Masukkan 'Y' untuk ya atau 'n' untuk tidak\n");
51        break;
52    }
53    scanner.close();
54 }

```

Gambar 7. Code Save Matrix to File

```

1 public static String[] readConfig(String filepath) throws FileNotFoundException{
2     File file = new File(filepath);
3     Scanner scanner = new Scanner(file);
4     String config = "";
5     String boardType = "";
6     StringBuilder pieces = new StringBuilder();
7
8     try{
9         if (scanner.hasNextLine()){
10             config = scanner.nextLine();
11             if(scanner.hasNextLine()){
12                 boardType = scanner.nextLine();
13                 while(scanner.hasNextLine()){
14                     String line = scanner.nextLine();
15                     pieces.append(line).append("\n");
16                 }
17                 scanner.close();
18                 pieces.toString().trim();
19             }
20             else{
21                 System.err.println("Tidak terdapat board type");
22             }
23         }
24         else{
25             System.err.println("Tidak terdapat format NMP");
26         }
27     }
28     catch (Exception e){
29         System.err.println("Error: " + e);
30     }
31     // System.out.println(config);
32     // System.out.println(boardType);
33
34     scanner.close();
35     return new String[]{config, boardType, pieces.toString()};
36 }

```

Gambar 8. Code Read Config

```

1 public static void matrixToImage(Character[][] board, List<Piece> pieces, String out
2     putPath) {
3     int cellSize = 50;
4     int width = board[0].length * cellSize;
5     int height = board.length * cellSize;
6     BufferedImage image = new BufferedImage(width, height, BufferedImage.TYPE_INT_R
7     G);
8     Graphics2D g = image.createGraphics();
9
10    // background color
11    g.setFont(new Font("SansSerif", Font.PLAIN, 30));
12    g.setColor(Color.WHITE);
13    g.fillRect(0, 0, width, height);
14    g.drawImage(image, 0, 0 + height, width, -height, null);
15
16    for (int row = 0; row < board.length; row++) {
17        for (int col = 0; col < board[0].length; col++) {
18            char ch = board[row][col];
19            for (Piece piece : pieces) {
20                if(piece.getPieceCharacter() == ch){
21                    Color pieceColor = piece.getColor();
22                    g.setColor(pieceColor);
23                    break;
24                }
25            }
26            int xRect = col * cellSize;
27            int yRect = row * cellSize;
28            g.fillRoundRect(xRect, yRect, cellSize, cellSize, 15, 15 );
29            g.setColor(Color.WHITE);
30            g.drawRect(xRect, yRect, cellSize, cellSize);
31
32            int xChar = col * cellSize + 15;
33            int yChar = (row + 1) * cellSize - 10;
34            g.drawString(String.valueOf(ch), xChar, yChar);
35        }
36    }
37    g.dispose();
38
39    try {
40        File output = new File(outputPath);
41        System.out.println(outputPath);
42        ImageIO.write(image, "png", output);
43        System.out.println("Image saved: " + output.getAbsolutePath());
44    } catch (Exception e) {
45        e.printStackTrace();
46    }
47 }

```

Gambar 9. Code Image Output

BAB IV

Hasil Program

4.1. Case 1

```
5 5 7
DEFAULT
A
AA
B
BB
C
CC
D
DD
EE
EE
E
FF
FF
F
GGG
```

Gambar 10. Input 1.txt

```
1.txt
Searching for the solution, this may take a while...
Finished one solution found!
Execution time: 1ms
Iteration : 499
A A B B C
A D B C C
E D D F G
E E F F G
E E F F G
```

Gambar 11. Output text 1.txt



Gambar 12. Output image 1.png

4.2. Case 2

```
13 2 26  
DEFAULT  
A  
B  
C  
D  
E  
F  
G  
H  
I  
J  
K  
L  
M  
N  
O  
P  
Q  
R  
S  
T  
U  
V  
W  
X  
Y  
Z
```

Gambar 13. Input 4.txt

```
4.txt  
Searching for the solution, this may take a while...  
Finished one solution found!  
Execution time: 0ms  
Iteration : 26  
A B  
C D  
E F  
G H  
I J  
K L  
M N  
O P  
Q R  
S T  
U V  
W X  
Y Z
```

Gambar 14. Output 4.txt



Gambar 15. Output Image 4.png

4.3. Case 3

```

5 5 7
DEFAULT
| A
AA
B
BB
| C
CC
D
DD
EE
EE
E
FF
FF
F
GGG

```

Gambar 16. Input 2.txt

```

2.txt
Searching for the solution, this may take a while...
Finished one solution found!
Execution time: 1308ms
Iteration : 11312912
A B B C C
A A B D C
E E E D D
E E F F F
G G G F F

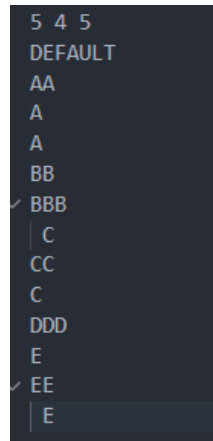
```

Gambar 17. Output 2.txt

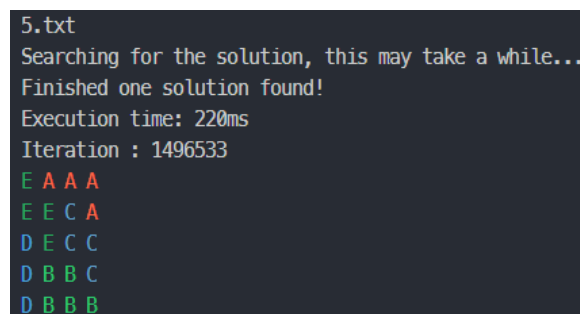


Gambar 18. Output Image 2.png

4.4. Case 4



Gambar 19. Input 5.txt



Gambar 20. Output 5.txt



Gambar 21. Output 5.png

4.5. Case 5

```
5 5 7
DEFAULT
A
A
B
BB
C
CC
D
DD
EE
EE
E
FF
FF
F
GGG
```

Gambar 22. Input 9.txt

```
Masukkan nama file dari folder input (1.txt)
9.txt
Unsolvable, Matrix piece is not equal to the board size
```

Gambar 23. Output 9.txt

4.6. Case 6

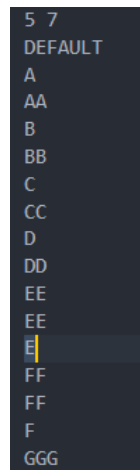
```
5 -1 7
DEFAULT
A
AA
B
BB
C
CC
D
DD
EE
EE
E
FF
FF
F
GGG
```

Gambar 24. Input 10.txt

```
Masukkan nama file dari folder input (1.txt)
10.txt
NMP is not valid
```

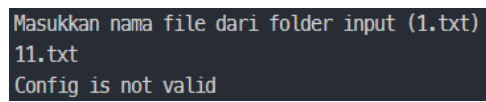
Gambar 25. Output 10.txt

4.7. Case 7



```
5 7
DEFAULT
A
AA
B
BB
C
CC
D
DD
EE
EE
E
FF
FF
F
GGG
```

Gambar 26. Input 11.txt



```
Masukkan nama file dari folder input (1.txt)
11.txt
Config is not valid
```

Gambar 27. Output 11.txt

Lampiran

Link repository: https://github.com/ahsuunn/Tucil1_13523074

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