

Laporan Tugas Kecil 1
IF2211 Strategi Algoritma



Dipersiapkan oleh:

Fachriza Ahmad Setiyono (13523162)

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
JL. GANESA 10, BANDUNG 40132

2025

DAFTAR ISI

DAFTAR ISI	2
DESKRIPSI MASALAH	3
1.1 Deskripsi Masalah	3
ALGORITMA PROGRAM	4
2.1 Algoritma Brute Force.....	4
IMPLEMENTASI PROGRAM.....	6
3.1 lib.....	6
3.1.1 config.....	6
3.1.2 GUI.....	7
3.2 puzzle.....	8
3.2.1 Block.java.....	8
3.2.2 Board.java.....	9
3.2.3 Game.java.....	11
3.2.4 Solver.java.....	11
3.3 App.java	14
EKSPERIMEN	15
4.1 Default, 2x2, 2 Blok, Tidak ada solusi	15
4.2 Default, 3x5, 5 Blok, Kekurangan blok	15
4.3 Default, 2x2, 2 Blok, Blok berlebih.....	16
4.4 Default, 5x5, 7 Blok, Sukses	16
4.5 Default, 5x11, 12 Blok, Sukses	17
4.6 Custom, 5x7, 5 Blok, Sukses.....	18
4.7 Custom, 5x10, 7 Blok, Sukses	18
LAMPIRAN	19

BAB I

DESKRIPSI MASALAH

1.1 Deskripsi Masalah



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

Komponen penting dari permainan IQ Puzzler Pro terdiri dari:

1. Papan (Board)

Board merupakan komponen utama yang menjadi tujuan permainan dimana pemain harus mampu mengisi seluruh area papan menggunakan blok-blok yang telah disediakan.

2. Blok (Block / 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.

Pada laporan tugas kecil ini akan dibahas algoritma dan implementasi program untuk menemukan 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.

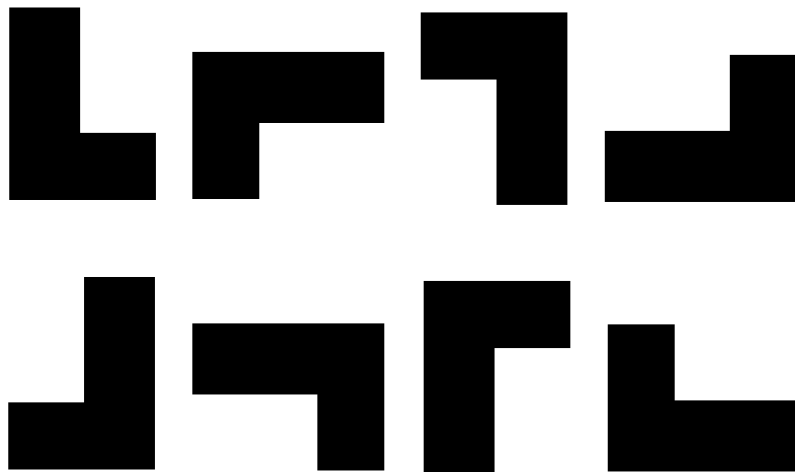
BAB II

ALGORITMA PROGRAM

2.1 Algoritma Brute Force

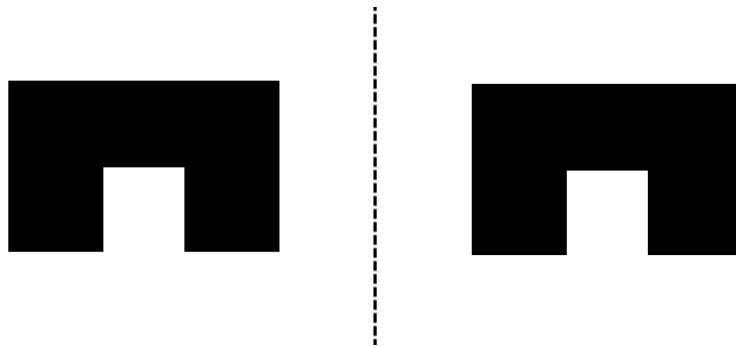
Pada permainan ini, algoritma brute force bisa digunakan untuk mencari salah satu solusi dari permainan dengan mencoba semua kemungkinan peletakan blok yang valid. Dengan algoritma ini, solusi dari suatu input pasti bisa ditemukan jika input tersebut valid. Jika input tidak valid, maka program akan memberi tahu pengguna bahwa input tersebut tidak memiliki solusi, baik karena memang tidak ada kombinasi peletakan blok yang memenuhi syarat kemenganan puzzle, blok yang kekurangan, ataupun blok yang berlebihan. Algoritma brute force dalam program ini diterapkan sebagai berikut:

1. Untuk semua blok yang dimuat dari input, buat semua variasi bentuk. Variasi bentuk berupa bentuk yang dicerminkan atau dirotasi dari bentuk semula. Umumnya, ada 8 total variasi dari suatu bentuk (termasuk bentuk semula) untuk penempatan di permukaan 2 dimensi.



(Semua variasi penempatan blok pada permukaan 2 dimensi)

Namun untuk menghemat waktu komputasi, variasi bentuk yang sama dengan suatu variasi bentuk lainnya bisa dianggap satu. Contohnya, bentuk yang memiliki simetri di sumbu y jika dicerminkan terhadap sumbu y akan memiliki bentuk yang sama, maka variasi ini tidak perlu dianggap. Perubahan ini tidak akan merubah hasil dari algoritma brute force yang akan dilakukan, dengan kata lain, hal ini adalah suatu bentuk optimisasi (yaitu dengan mengurangi kasus yang bersifat redundan) dan bukan heuristik.



(Blok dengan simetri sumbu-y)

2. Setelah semua input diproses oleh program, algoritma brute force lalu dilakukan dengan fungsi rekursif sebagai berikut:
 - a. Basis fungsi ada 2, yaitu ketika program telah menemukan bahwa blok puzzle memiliki jumlah lebih banyak dari yang dibutuhkan. Kasus ini terdeteksi saat fungsi ingin memasang suatu blok, tapi tidak ada *grid* papan yang kosong (puzzle sudah selesai). Basis kedua adalah ketika semua blok sudah digunakan atau terpasang pada puzzle. Basis ini memiliki 2 kemungkinan lagi, yaitu kasus jika papan terisi sepenuhnya yang berarti puzzle telah diselesaikan dengan solusi yang valid, dan kasus papan tidak terisi penuh yang berarti puzzle tidak bisa diselesaikan karena kurangnya blok puzzle yang tersedia.
 - b. Rekurens fungsi dapat dijabarkan sebagai berikut:
 - i. Ambil blok *B* dari *stack* blok.
 - ii. Untuk seluruh indeks *i, j* yang valid di *grid* papan:
 1. jika *grid* kosong maka untuk seluruh variasi blok *B*, dilakukan percobaan pemasangan blok di indeks *grid* tersebut. Jika blok *B* berhasil dipasang, maka lakukan pemanggilan rekursi. Jika hasil kembali fungsi adalah *true*, maka kembalikan *true*. Jika *false*, maka copot variasi blok *B* dari papan.
 2. Jika *grid* tidak kosong, maka lanjut ke indeks *grid* selanjutnya.
 - iii. Kembalikan blok *B* ke *stack*.
 - iv. Jika sama sekali tidak terjadi pemasangan blok di tahap (ii), maka program menyatakan bahwa blok puzzle berlebih.
 - v. Kembalikan nilai *false*.

Dari penjabaran fungsi tersebut, terlihat bahwa algoritma yang digunakan merupakan algoritma brute force, dimana program mencoba semua kemungkinan yang valid. Pada setiap pemanggilan fungsi, program akan mencoba untuk memasang seluruh varian blok di seluruh posisi papan. Untuk penyimpanan blok, *stack* digunakan karena ketika blok tidak jadi dipasang, maka blok bisa disimpan kembali, blok sebelumnya pada papan diubah, dan blok tadi bisa diambil lagi dengan mudah. Pengecekan kondisi penyelesaian puzzle tidak dilakukan pada setiap iterasi untuk alasan performa, tetapi pengecekan dilakukan hanya pada saat kondisi-kondisi tertentu yaitu ketika *stack* kosong atau penempatan blok tidak terjadi. Sekali lagi, langkah ini tidak memengaruhi keakuratan hasil dari algoritma dan hanya merupakan bentuk optimisasi.

BAB III

IMPLEMENTASI PROGRAM

Program diimplementasikan dalam bahasa pemrograman Java versi 21. Paradigma pemrograman yang digunakan berupa pemrograman berorientasi objek. Seluruh program berada dalam *package* `com.fachriza.iqpuzzlesolver`. Selain dijalankan pada terminal, program juga bisa dijalankan menggunakan *Graphical User Interface* (GUI). Selama pembuatan program, kompatibilitas antar *Linux* dan *Windows* juga telah diperhatikan.

Berikut adalah penjabaran dari program. Agar lebih mudah, setiap subjudul berikut menggambarkan jalur dari *package* program. Hanya kode yang dirasa penting saja yang akan dibahas disini.

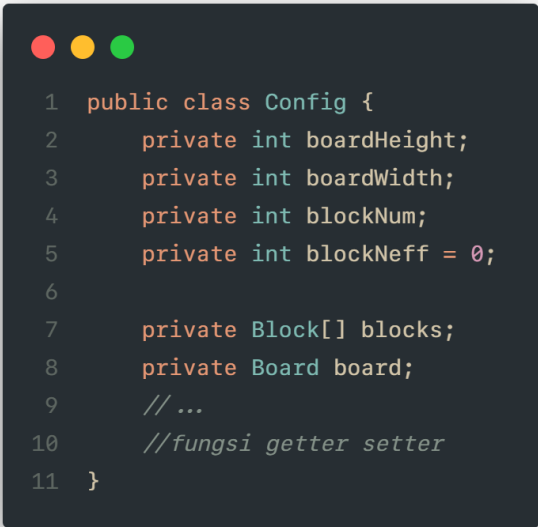
3.1 lib

Untuk membantu keberjalanan program utama seperti membanca input, memproses warna, dan operasi I/O lainnya, maka diperlukan beberapa kelas bantuan.

3.1.1 config

3.1.1.1 Config.Java

Config adalah kelas representasi dari file input. Kelas ini menyimpan atribut terkait permainan puzzle serta fungsi *getter setter* untuk tiap atribut.



```
1 public class Config {
2     private int boardHeight;
3     private int boardWidth;
4     private int blockNum;
5     private int blockNeff = 0;
6
7     private Block[] blocks;
8     private Board board;
9     // ...
10    //fungsi getter setter
11 }
```

3.1.1.1 ConfigHandler.Java

ConfigHandler adalah kelas yang bertugas untuk membaca file teks input. Kelas ini menggunakan *state machine* yang menggambarkan bagian input yang sedang dibaca agar lebih mudah membaca teks input.

```

1  public class ConfigHandler {
2      enum ConfigState {
3          NUMBERS,
4          TYPE,
5          CUSTOM,
6          BLOCKS,
7          FINISHED
8      }
9
10     private Config config;
11
12     public ConfigHandler(String fileName) throws Exception {
13         Path filePath = Paths.get("config", fileName);
14         if (!Files.exists(filePath))
15             throw new FileNotFoundException(String.join(" ", filePath.toString(), "not found!"));
16
17         try (BufferedReader reader = Files.newBufferedReader(filePath)) {
18             // baca file
19             // ...
20         }
21     }
22 }

```

3.1.2 GUI

3.1.2.1 MainFrame.Java

Kelas ini digunakan untuk implementasi GUI. GUI yang dibuat hanya berupa satu *window* utama yang terdiri dari tombol untuk memilih dan memroses file teks input, dan solusi dari puzzle yang digambar menggunakan pustaka bawaan Graphics2D. Kode untuk menggambar solusi dari puzzle adalah sebagai berikut:

```

1  int height = getHeight() / config.getHeight();
2  int width = getWidth() / config.getWidth();
3
4  int blockSize = Math.min(height, width);
5  int paddingX = getWidth() - blockSize * config.getWidth();
6  int paddingY = getHeight() - blockSize * config.getHeight();
7  int startX = paddingX / 2;
8  int startY = paddingY / 2;
9  Font font = new Font("Arial", Font.BOLD, 24);
10 g2d.setFont(font);
11 for (int i = 0; i < config.getHeight(); i++) {
12     for (int j = 0; j < config.getWidth(); j++) {
13         int elem = config.getBoard().getElement(j, i);
14         if (elem < 65)
15             continue;
16         int sphereX = j * blockSize + startX;
17         int sphereY = i * blockSize + startY;
18         int diameter = blockSize + 1;
19
20         g2d.setColor(com.fachriza.iqpuzzlersolver.lib.Color.colorMap
21             .get(com.fachriza.iqpuzzlersolver.lib.Color.colorTable[elem - 65]));
22         g2d.fillOval(sphereX, sphereY, diameter, diameter);
23         if (elem == 65 || elem == 73 || elem == 81) {
24             g2d.setColor(Color.WHITE);
25         } else {
26             g2d.setColor(Color.BLACK);
27         }
28
29         FontMetrics fm = g2d.getFontMetrics();
30         int textWidth = fm.stringWidth(".");
31         int textHeight = fm.getAscent();
32
33         int textX = sphereX + (diameter - textWidth) / 2 - 5;
34         int textY = sphereY + (diameter + textHeight) / 2 - 5;
35
36         g2d.drawString(String.valueOf((char) elem), textX, textY);
37     }
38 }
39 }

```

3.2 puzzle

Package ini berisi seluruh komponen utama dari permainan puzzle, yaitu blok, papan, proses permainan, dan *solver*—tempat kode brute force diimplementasikan—.

3.2.1 Block.java

Block adalah kelas untuk merepresentasikan setiap objek blok yang ada di permainan. Representasi dari bentuk blok adalah dengan kumpulan koordinat, dalam kasus ini adalah dalam bentuk senarai titik koordinat. Kumpulan koordinat ini juga menyimpan seluruh variasi bentuk dari blok tersebut.


```

1 public class Block {
2
3     private final byte ID;
4     private List<List<Point>> variants = new ArrayList<>();
5
6     public Block(byte ID) {
7         this.ID = ID;
8     }
9
10    public void addCoordinates(int x, int y) {
11        if (variants.isEmpty()) {
12            variants.add(new ArrayList<Point>());
13        }
14        variants.get(0).add(new Point(x, y));
15    }
16    // ...
17 }

```

Di kelas ini juga diimplementasikan fungsi untuk membuat seluruh varian bentuk blok. Seluruh varian blok dapat dicapat dengan mencerminkan bentuk awal dengan sumbu-x, sumbu-y, sumbu-x, dan sumbu-x lalu sumbu-y. Kemudian, untuk setiap varian yang telah dibentuk, dilakukan pencerminan terhadap garis $y=x$. Disini, variasi bentuk yang sama dengan bentuk lainnya tidak akan ditambahkan ke kumpulan varian blok tersebut.

```

1 private boolean isSymmetrical(List<Point> modified, int offsetX, int offsetY) {
2     // Mengecek apakah bentuk ini sudah ada sebelumnya
3 }
4
5 private Point calculateMinMaxOffset(List<Point> coordinates) {
6     // Menghitung offset agar hasil pencerminan kembali ke titik semula
7 }
8
9 public void generateVariants() {
10    if (variants.isEmpty()) {
11        return;
12    }
13    List<Point> original = variants.get(0);
14    Point offsets = calculateMinMaxOffset(original);
15    boolean isSymmetrical = false;
16
17    List<Point> modified = new ArrayList<>();
18    for (Point point : original) {
19        modified.add(new Point(point.y, point.x));
20    }
21    // System.out.println("x");
22    if (!isSymmetrical(modified, 0, 0)) {
23        variants.add(modified);
24    }
25
26    modified = new ArrayList<>();
27    for (Point point : original) {
28        modified.add(new Point(-point.x, point.y));
29    }
30    // System.out.println("y");
31    if (!isSymmetrical(modified, offsets.x, 0)) {
32        variants.add(modified);
33    }
34    modified = new ArrayList<>();
35    for (Point point : original) {
36        modified.add(new Point(point.y, -point.x));
37    }
38    variants.add(modified);
39    } else {
40        isSymmetrical = true;
41    }
42 }

```

```

43 modified = new ArrayList<>();
44 for (Point point : original) {
45     modified.add(new Point(point.x, -point.y));
46 }
47 // System.out.println("y");
48 if (!isSymmetrical(modified, 0, offsets.y)) {
49     variants.add(modified);
50     modified = new ArrayList<>();
51     for (Point point : original) {
52         modified.add(new Point(-point.y, point.x));
53     }
54     variants.add(modified);
55 } else {
56     isSymmetrical = true;
57 }
58
59 if (!isSymmetrical) {
60     modified = new ArrayList<>();
61     for (Point point : original) {
62         modified.add(new Point(-point.x, -point.y));
63     }
64     // System.out.println("x");
65     if (!isSymmetrical(modified, offsets.x, offsets.y)) {
66         variants.add(modified);
67         modified = new ArrayList<>();
68         for (Point point : original) {
69             modified.add(new Point(-point.y, -point.x));
70         }
71         variants.add(modified);
72     }
73 }
74
75 for (List<Point> variant : variants) {
76     for (Point point : variant) {
77         if (point.x == 0 && point.y == 0) {
78             variant.remove(point);
79             break;
80         }
81     }
82 }

```

3.2.2 Board.java

Board adalah representasi dari papan permainan puzzle. *Grid* puzzle dimodelkan dengan matriks *byte*. *Byte* dipilih karena data berukuran 8-bit sudah cukup untuk merepresentasikan ID blok yang hanya berjumlahkan 26 (A-Z).

```

1  public class Board {
2      private byte[][] grid;
3      private final int height;
4      private final int width;
5      private boolean isColorEnabled = true;
6
7      public Board(int N, int M) {
8          height = N;
9          width = M;
10         grid = new byte[height][width];
11     }
12     // ...
13 }

```

Board juga memiliki fungsi untuk meletakkan dan menghilangkan blok dari papan. Untuk alasan performa, fungsi placeBlock() dijalankan tanpa adanya pengecekan jika blok bisa diletakkan atau tidak. Jadi agar tidak merusak blok lain yang telah diletakkan, pengecekan terjadi beriringan dengan pemasangan. Jika salah satu koordinat dari blok tidak bisa diletakkan, maka penempatan diberhentikan dan seluruh koordinat blok yang telah dipasang dilepaskan lagi.

```

1  public boolean placeBlock(int centerX, int centerY, Block block, int variantIdx) {
2
3      List<Point> placed = new ArrayList<Point>();
4      for (Point points : block.getCoordinates(variantIdx)) {
5          int posX = centerX + points.x;
6          int posY = centerY + points.y;
7
8          if (!isCoordinateValid(posX, posY) || getElement(posX, posY) != 0) {
9              for (Point point : placed) {
10                 setElement(point.x, point.y, (byte) 0);
11             }
12             return false;
13         }
14
15         setElement(posX, posY, block.getID());
16         placed.add(new Point(posX, posY));
17     }
18     return true;
19 }
20
21 public void removeBlock(int centerX, int centerY, Block block, int variantIdx) {
22     for (Point points : block.getCoordinates(variantIdx)) {
23         int posX = centerX + points.x;
24         int posY = centerY + points.y;
25
26         if (isCoordinateValid(posX, posY)
27             && (getElement(posX, posY) == 0 || getElement(posX, posY) == block.getID())) {
28             setElement(posX, posY, (byte) 0);
29         }
30     }
31 }

```

3.2.3 Game.java

Objek dari kelas ini menggambarkan proses dari satu permainan puzzle yang berlangsung.

```
1 public class Game {
2     private Scanner scanner;
3     private ConfigHandler configHandler;
4     private Solver solver;
5
6     public Game(Scanner scanner) {
7         this.scanner = scanner;
8     }
9     public void start() {
10        try {
11            solver = new Solver(configHandler.getConfig());
12            solver.solve();
13
14            System.out.println("");
15            System.out.println(solver.getResult());
16
17            if (scanner != null)
18                SaveHandler.promptSave(scanner, solver);
19        } catch (Exception e) {
20            System.err.println(e.getMessage());
21        }
22    }
23    // ...
24 }
```

3.2.4 Solver.java

Objek Solver bertujuan untuk menyelesaikan permainan puzzle dengan algoritma brute force. Algoritma brute force diimplementasikan dalam fungsi rekursif `tryPlaceBlock()`. Konstruktor objek ini menerima objek `Config` lalu memasukkan seluruh blok puzzle ke *stack*. *State machine* juga digunakan untuk menggambarkan kondisi permainan puzzle.

```

1  public class Solver {
2      public enum SolverState {
3          NOT_STARTED,
4          SUCCESS,
5          FAIL_NO_SOLUTION,
6          FAIL_OVER_PIECE,
7          FAIL_LESS_PIECE
8      }
9
10     private long timeElapsedInNs;
11     private int cases = 0;
12
13     private Config config;
14     private SolverState state = SolverState.NOT_STARTED;
15
16     private Deque<Block> stack = new ArrayDeque<Block>();
17
18
19     public Solver(Config config) {
20         this.config = config;
21
22         for (Block block : config.getBlocks()) {
23             stack.offer(block);
24         }
25     }
26     // ...
27 }

```

Objek ini mempunyai fungsi `solve()` untuk memulai proses penyelesaian puzzle. Sebelum fungsi dipanggil, objek menyimpan waktu saat itu menggunakan `System.nanoTime()`. Ini digunakan untuk mencari tahu berapa lama waktu yang diperlukan untuk mencari solusi puzzle.

```

1  public SolverState solve() {
2      if (state != SolverState.NOT_STARTED) {
3          return state;
4      }
5      state = SolverState.FAIL_NO_SOLUTION;
6
7      // start brute-force
8      long startTime = System.nanoTime();
9      tryPlaceBlock();
10     long endTime = System.nanoTime();
11
12     timeElapsedInNs = endTime - startTime;
13
14     return state;
15
16 }

```

Sesuai dengan yang telah dijelaskan pada bab sebelumnya, fungsi `tryPlaceBlock()` merupakan fungsi rekursif brute force untuk menyelesaikan puzzle. Ada 2 basis dan bagian rekurens.

```
1 private boolean tryPlaceBlock() {
2     // base cases
3     if (state == SolverState.FAIL_OVER_PIECE) {
4         return false;
5     }
6
7     Board board = config.getBoard();
8
9     if (stack.isEmpty()) {
10         if (board.isSolved()) {
11             state = SolverState.SUCCESS;
12         } else {
13             // number of blocks "beads" should be equal to the number of grids
14             state = SolverState.FAIL_LESS_PIECE;
15         }
16         // early return
17         return true;
18     }
19
20     // recursion case
21     Block block = stack.pop();
22     boolean boardIsFull = true;
23
24     for (int i = 0; i < config.getHeight(); i++) {
25         for (int j = 0; j < config.getWidth(); j++) {
26
27             if (board.getElement(j, i) != 0) {
28                 continue;
29             }
30
31             // manually set the center ID because we removed it from the blocks coordinates
32             board.setElement(j, i, block.getID());
33
34             boardIsFull = false;
35
36             for (int k = 0; k < block.getVariantsNum(); k++) {
37                 if (board.placeBlock(j, i, block, k)) {
38                     // increment cases count if placement was successful
39                     cases++;
40                     if (tryPlaceBlock())
41                         return true;
42                     board.removeBlock(j, i, block, k);
43                 }
44             }
45             // manually remove the center
46             board.setElement(j, i, (byte) 0);
47         }
48     }
49     stack.push(block);
50     if (boardIsFull) {
51         // no available grid to even place the center of the block
52         // early return
53         state = SolverState.FAIL_OVER_PIECE;
54         return true;
55     }
56     return false;
57 }
```

Terdapat satu mikro-optimalisasi yang dilakukan disini untuk mempercepat waktu komputasi, yaitu dengan tidak menyimpan koordinat lokal (0,0) tiap blok. Koordinat poros tiap blok diletakkan di *grid* sebelum memanggil fungsi `board.placeBlock()` dan dilepas setelah semua varian blok dicoba. Ini mencegah pemasangan dan pencopotan berulang kali di koordinat poros dan dapat menghemat waktu beberapa milisekon.

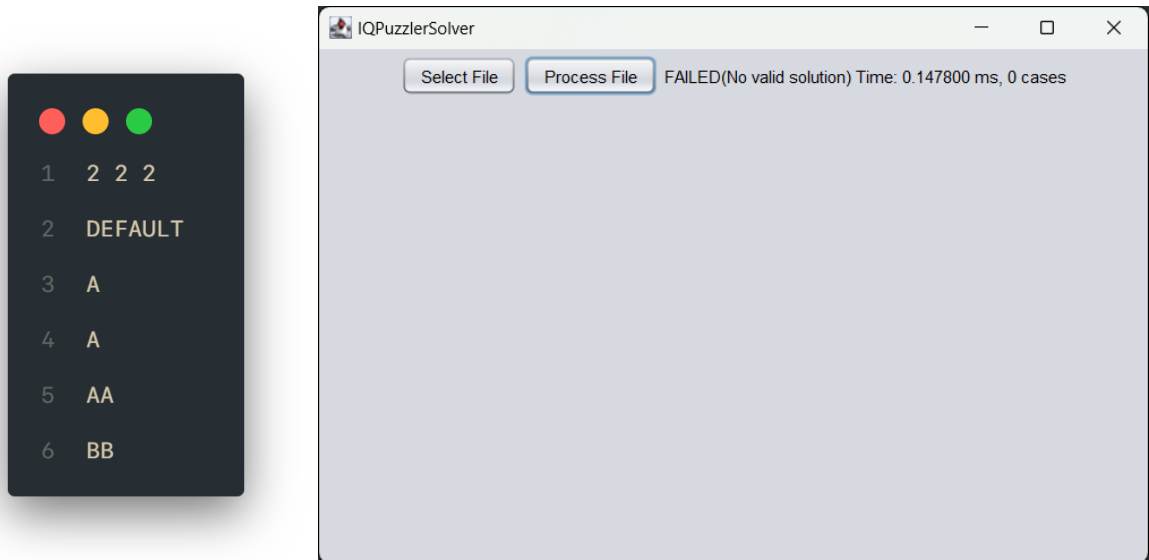
3.3 App.java

App.java berfungsi sebagai tempat masuk program. Kelas ini memiliki metode statik `main()` yang dipanggil ketika program dijalankan. Metode ini juga membaca argumen penjalanan program. Jika program dijalankan dengan argumen GUI, maka program akan dijalankan dengan GUI.

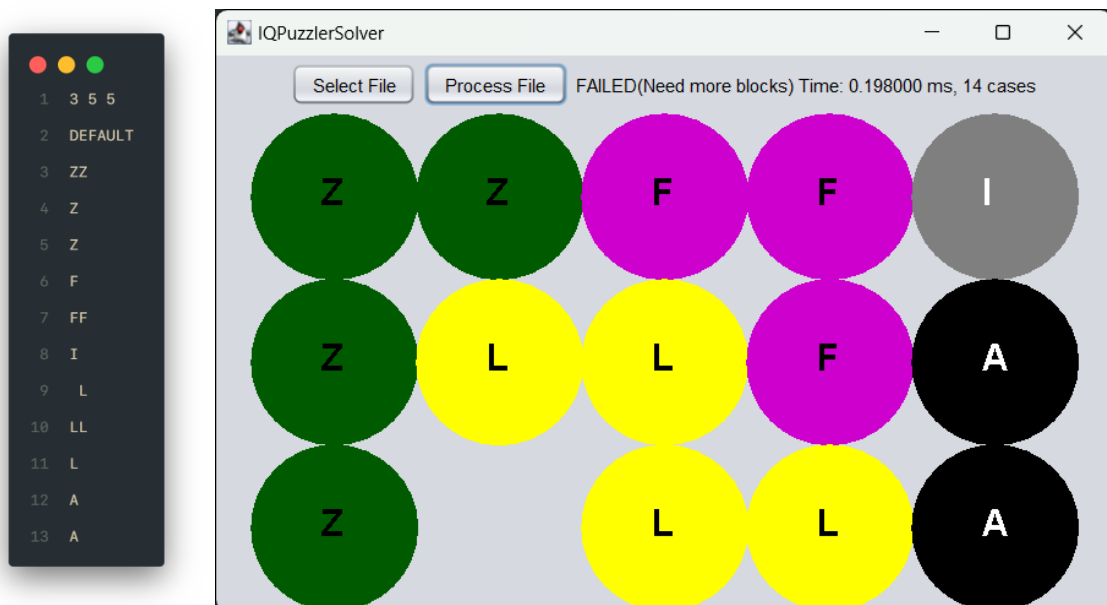
```
1 public class App {
2     public static void main(String[] args) {
3         if (args.length == 0) {
4             try (Scanner scanner = new Scanner(System.in)) {
5                 Game game = new Game(scanner);
6                 game.readConfig();
7                 game.start();
8             } catch (Exception e) {
9                 e.printStackTrace();
10            }
11        } else if (args[0].equals("GUI")) {
12            MainFrame mainFrame = new MainFrame();
13        } else {
14            System.err.println("Invalid arguments");
15        }
16    }
17 }
18
```

BAB IV EKSPERIMEN

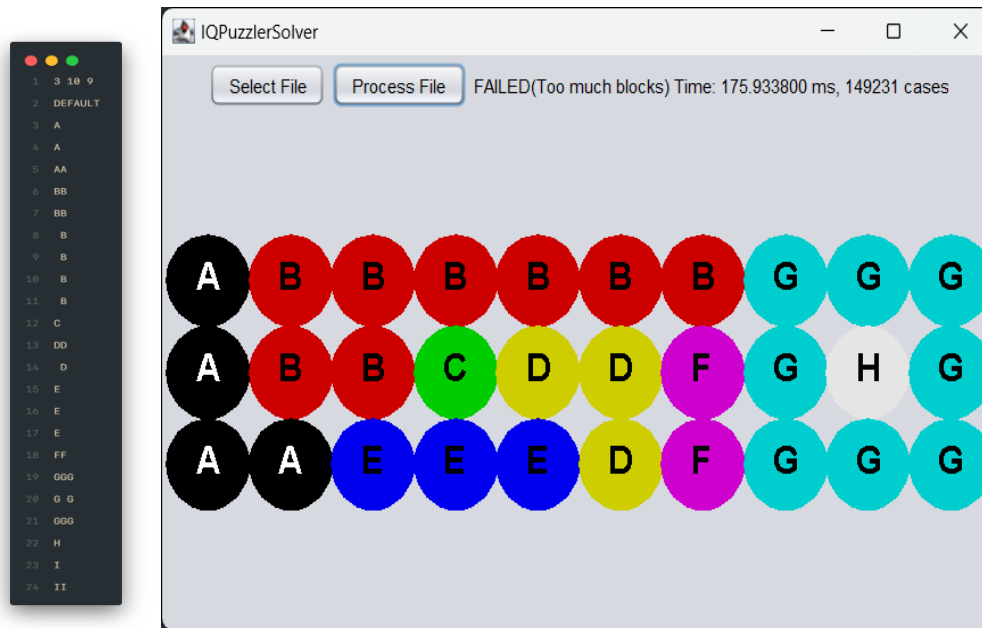
4.1 Default, 2x2, 2 Blok, Tidak ada solusi



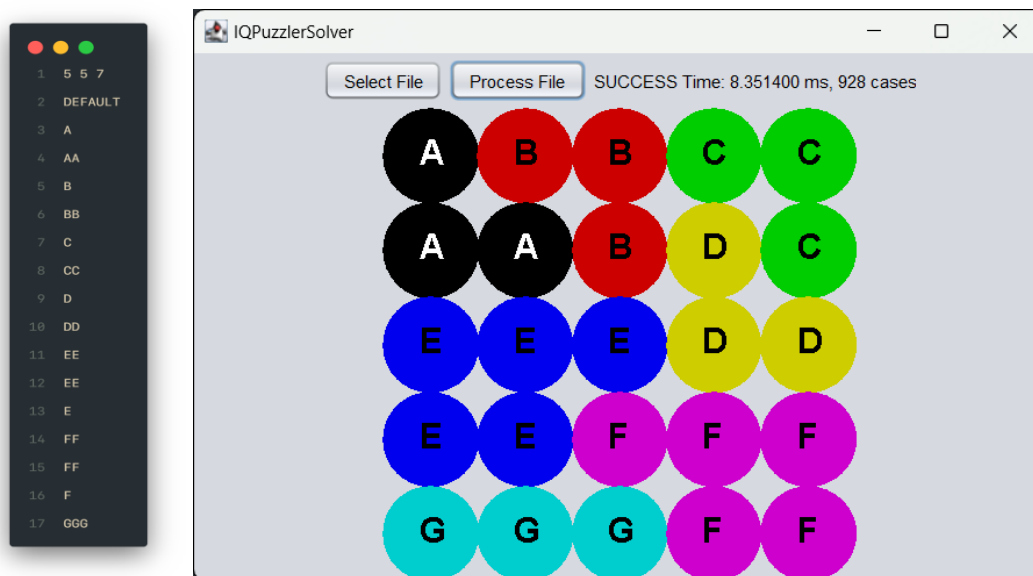
4.2 Default, 3x5, 5 Blok, Kekurangan blok



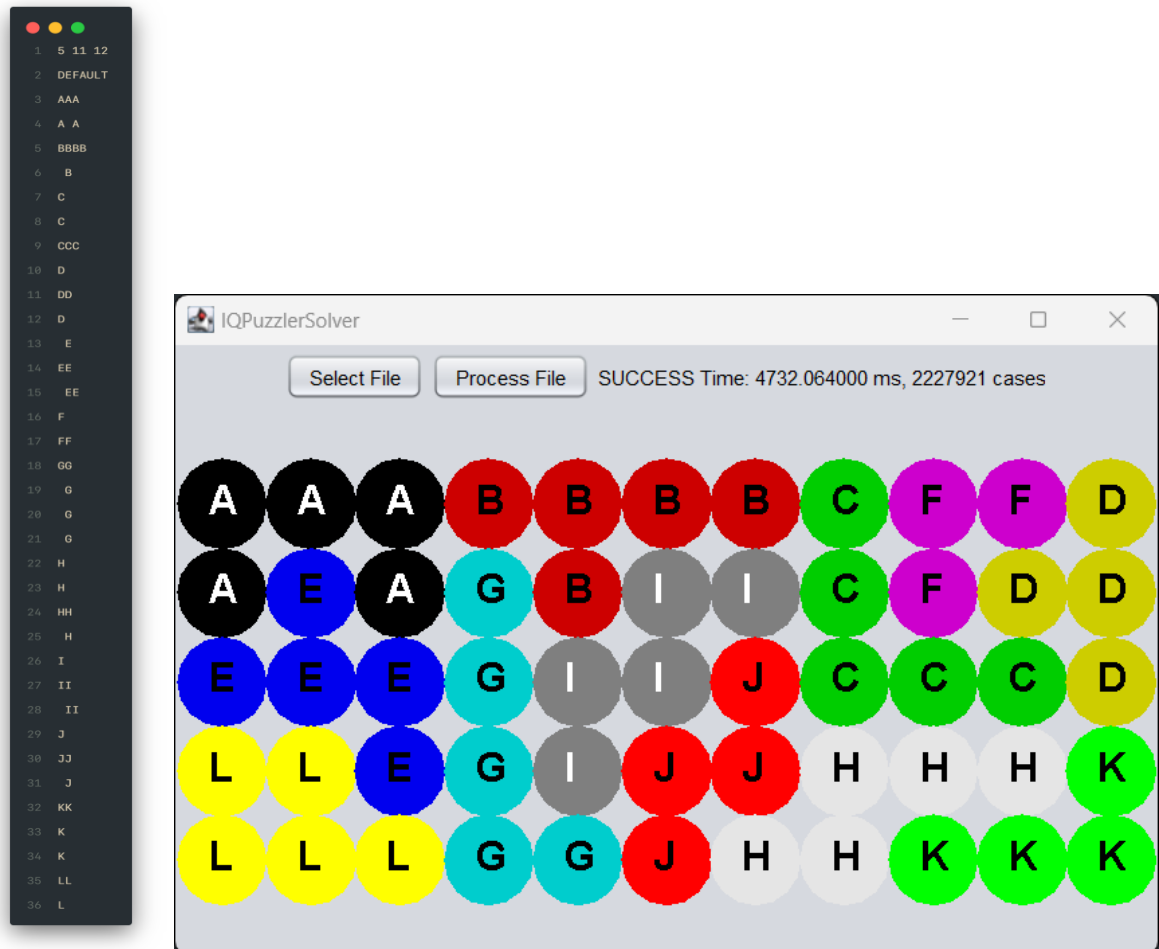
4.3 Default, 2x2, 2 Blok, Blok berlebih



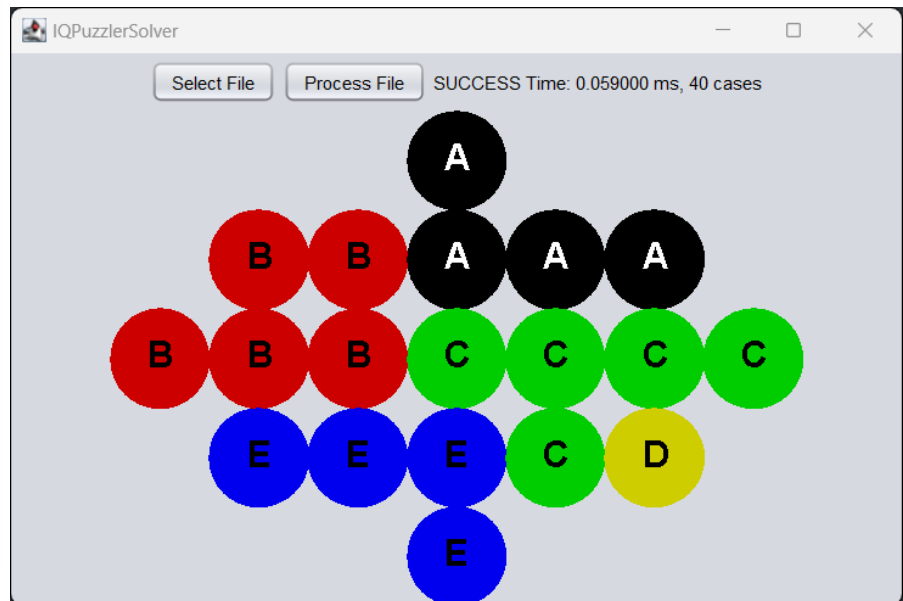
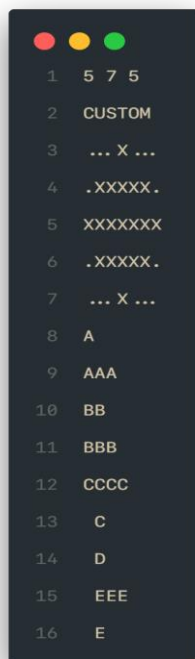
4.4 Default, 5x5, 7 Blok, Sukses



4.5 Default, 5x11, 12 Blok, Sukses



4.6 Custom, 5x7, 5 Blok, Sukses



4.7 Custom, 5x10, 7 Blok, Sukses



BAB VI

LAMPIRAN

Tabel Pengerjaan

No	Poin	Ya	Tidak
1	Program berhasil dikompilasi tanpa kesalahan	v	
2	Program berhasil berjalan	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	

Pranala repositori

https://github.com/L4mbads/Tucil1_13523162