Tugas Kecil 1 IF2211 Strategi Algoritma

Penyelesaian Permainan IQ Puzzler Pro dengan Algoritma *Brute Force*

Disusun oleh:

13523077 – Albertus Christian Poandy

INSITIUT TEKNOLOGI BANDUNG
2024

DAFTAR ISI

DAFTAR ISI **2**
BAB I DESKRIPSI MASALAH DAN ALGORITMA **3**
1.1 Algoritma Brute Force 3
1.2 Permainan IQ Puzzler Pro 3
1.3 Algoritma Penyelesaian Permainan IQ Puzzler Pro dengan Pendekatan *Brute Force* 3
BAB II IMPLEMENTASI ALGORITMA DALAM BAHASA JAVA **5**
2.1 Kelas Block 5
2.2 Kelas Solver 5
BAB III SOURCE CODE PROGRAM **7**
3.1 Repository Program 7
3.2 Source Code Program 7
BAB IV MASUKAN DAN KELUARAN PROGRAM **29**
4.1 Kasus uji 1 – DEFAULT Solved 29
4.2 Kasus uji 2 – CUSTOM Solved 30
4.3 Kasus uji 3 – PYRAMID Solved 31
4.4 Kasus uji 4 – DEFAULT Not Solvable 32
4.5 Kasus uji 5 – CUSTOM Not Solvable 33
4.6 Kasus uji 6 – PYRAMID Not Solvable 34
4.7 Kasus uji 7 – PYRAMID Solved 35
BAB V LAMPIRAN **37**
REFERENSI **38**

BAB I

DESKRIPSI MASALAH DAN ALGORITMA

1.1 Algoritma Brute Force

Algoritma *brute force* adalah metode pendekatan pemecahan suatu masalah yang lurus atau lempeng (*straightforward*). Algoritma ini memecahkan persoalan secara sederhana, langsung, serta dengan cara yang jelas (*obvious*) dan mudah dipahami. Algoritma ini juga sering disebut algoritma naif (*naïve algorithm*).

Algoritma *brute force* umumnya bukanlah algoritma yang "cerdas" dan tidak sangkil, karena algoritma ini membutuhkan biaya komputasi yang besar dan waktu yang lama dalam penyelesaiannya. Oleh karena itu, algoritma brute force lebih cocok digunakan untuk persoalan yang memiliki ukuran masukan kecil, mengingat implementasinya yang juga relatif lebih mudah dan sederhana. Walaupun bukan merupakan pendekatan yang sangkil, hampir seluruh persoalan dapat dipecahkan dengan algoritma *brute force*.

1.2 Permainan IQ Puzzler Pro

IQ Puzzler Pro adalah permainan papan yang diproduksi oleh *Smart Games*. Dalam permainan ini, pemain bertugas mengisi papan dengan menggunakan semua blok puzzle yang tersedia tanpa terkecuali. Setiap blok memiliki bentuknya masing-masing, dan tantangannya terletak pada bagaimana pemain dapat menyusun blok-blok tersebut hingga papan terisi sepenuhnya tanpa ada ruang kosong yang tersisa.

Komponen utama dalam permainan ini terdiri dari papan (*board*) dan blok puzzle. Papan berfungsi sebagai area permainan di mana pemain harus menyusun blok-blok dengan tepat. Sementara itu, blok puzzle merupakan elemen yang harus digunakan untuk mengisi papan. Setiap blok dapat diputar serta dicerminkan sedemikian rupa sehingga bentuknya dapat sesuai dengan ruang yang tersedia di papan.

Untuk tugas ini, permainan dimulai dengan papan kosong, dan pemain harus menempatkan setiap blok tanpa ada yang bertumpang tindih, kecuali pada tantangan mode piramida yang memungkinkan adanya tumpukan antarblok. Permainan ini dianggap selesai jika seluruh blok telah digunakan dan papan benar-benar terisi. Untuk mode piramida, permainan selesai jika seluruh blok telah digunakan dan terbentuk konfigurasi piramida dengan alas persegi. Pada konfigurasi ini, blok puzzle bisa saja disusun dengan rotasi yang tidak umum pada kasus 2 dimensi.

1.3 Algoritma Penyelesaian Permainan IQ Puzzler Pro dengan Pendekatan *Brute Force*

Untuk menemukan Solusi dari permainan IQ Puzzler Pro secara algoritmik, digunakan pendekatan *brute force*. Adapun algoritma *brute force* yang digunakan untuk menyelesaikan permainan ini adalah sebagai berikut.

1. Representasikan papan sebagai sebuah matriks dengan jumlah baris dan kolom sesuai dengan masukan yang diberikan. Blok puzzle juga direpresentasikan sebagai sebuah matriks sesuai dengan panjang dan lebar blok.

2. Program akan mencari seluruh kemungkinan peletakan blok puzzle pada papan dengan metode *backtracking*.
3. Tempatkan blok puzzle pada posisi pertama (kiri atas) di papan. Periksa apakah penempatan blok tersebut valid (dengan memeriksa apakah blok puzzle menimpa blok yang lain atau terdapat bagian yang keluar dari papan)
4. Jika tidak ada pelanggaran, tempatkan blok puzzle pada posisi tersebut dan lanjut pada blok puzzle berikutnya.
5. Jika ditemukan pelanggaran, coba lakukan rotasi pada blok puzzle dan tempatkan lagi pada posisi yang sama.
6. Jika tidak ada satupun kemungkinan rotasi valid untuk diletakkan di papan, coba lakukan *mirroring* pada blok lalu coba lagi seluruh kemungkinan rotasi.
7. Jika masih tidak ada kemungkinan yang valid, coba lanjut ke posisi selanjutnya pada papan, dan ulangi langkah rotasi dan *mirroring*.
8. Jika tidak ada satupun posisi penempatan yang valid setelah mencoba seluruh posisi pada papan, maka pencarian menggunakan blok tersebut dihentikan dan kembali ke blok sebelumnya. Blok sebelumnya kemudian coba dirotasi, yang dilanjutkan dengan *mirroring*, kemudian dipindahkan (langkah 5–7).
9. Seluruh proses (langkah 3–8) diulang hingga seluruh blok berhasil ditempatkan pada papan.
10. Pada mode piramida, langkah-langkah algoritma tetap sama, hanya saja representasi papan dan blok puzzle dibuat menjadi matriks 3 dimensi, dan seluruh percobaan peletakan dan rotasi ditingkatkan pada konteks 3 dimensi.

Berikut adalah *pseudocode* untuk langkah penyelesaian di atas (pada 2 dimensi).

```
procedure solve()
    if (solveHelper(0, 0, 0, 0)) then
        {berhasil menemukan solusi}
    else
        {tidak menemukan solusi}

function solveHelper(block_idx, curr_row, curr_col, rotate_type: integer) ->
boolean
    if (rotate_type = 8) then
        return solveHelper(block_idx, curr_row, curr_col+1, 0)
    if (curr_col >= board.num_col) then
        return solveHelper(block_idx, curr_row+1, 0, 0)
    if (curr_row >= board.num_row) then
        return false
    if (block_idx >= blocks.length) then
        return isMapFull()

    curr_block <- blocks[block_idx]
    { lakukan rotasi/mirroring pada block sesuai dengan rotate_type }
    for i <- 1 to (rotate_type mod 4) do
        curr_block.rotate()
    if (rotate_type >= 4) then
        curr_block.mirror()
    { jika bisa diletakkan pada posisi (curr_row, curr_col), langsung diletakkan}
    if (place(curr_block, curr_row, curr_col)) then
        { lanjut ke block selanjutnya }
        if (solveHelper(curr_block+1, 0, 0, 0) and isMapFull()) then
            return true
        else
            {jika konfigurasi tidak valid untuk blok selanjutnya, undo peletakan}
            unplace(curr_block, curr_row, curr_col)

    return solveHelper(block_idx, curr_row, curr_col, rotate_type+1)
```

BAB II

IMPLEMENTASI ALGORITMA DALAM BAHASA JAVA

Dalam pembuatan program, digunakan bahasa pemrograman *Java*. Untuk proses implementasi yang lebih baik dan terstruktur, digunakan pendekatan *object oriented*. Terdapat dua kelas utama yang berperan dalam menyelesaikan persoalan, yaitu kelas *Block* dan kelas *Solver*.

2.1 Kelas Block

Kelas ini merupakan kelas yang menjadi representasi objek-objek block puzzle. Beberapa fungsi/metode utama pada kelas ini adalah:

Metode	Deskripsi
`void mirror()`	Melakukan *mirroring* pada blok puzzle
`void rotate()`	Melakukan rotasi 90 derajat pada blok puzzle
`void lift()`	Melakukan rotasi pada tingkat 3 dimensi, yaitu seperti 'mengangkat' ujung matriks persegi blok 90 derajat ke arah ujung diagonal lainnya.
`void mirrorLift()`	Melakukan *mirroring* pada blok yang telah di-*lift*

2.2 Kelas Solver

Kelas ini merupakan kelas yang bertanggung jawab untuk menemukan solusi dari persoalan, yaitu menggunakan algoritma dengan pendekatan *brute force*.

Metode	Deskripsi
`boolean place(Block block, int row, int col)`	Mencoba meletakkan *block* pada indeks (*row*, *col*) pada matriks. Apabila peletakan valid, mengembalikan *true* dan meletakkan *block* pada posisi tersebut. Jika peletakan tidak valid, mengembalikan *false*.
`void unplace(Block block, int row, int col)`	Melakukan *undo* peletakan *block* sebelumnya pada indeks (*row*, *col*). Fungsi ini digunakan ketika telah meletakkan *block* sebelumnya tetapi posisi peletakan *block* tak bersesuaian dengan konfigurasi peletakan blok-blok lainnya.
`boolean isMapFull()`	Melakukan pengecekan apakah papan telah terisi penuh oleh blok-blok puzzle yang ada
`boolean isPyramidSolved()`	Melakukan pengecekan apakah telah terbentuk konfigurasi piramida khusus untuk mode piramida

`boolean solveHelper(int blockidx, int i, int j, int rotate)`	Fungsi utama untuk menemukan solusi menggunakan pendekatan brute force, yang mengembalikan *true* jika berhasil menemukan solusi dan *false* jika sebaliknya.
`void solve()`	Fungsi yang akan menghasilkan output berdasarkan hasil dari mememanggil fungsi *solveHelper*.

BAB III

SOURCE CODE PROGRAM

3.1 Repository Program

Repository program dapat diakses melalui tautan berikut:
https://github.com/albertchriss/Tucil1_13523077

3.2 Source Code Program

3.2.1 *Block.java*

```
package puzzle;

public class Block {
    public int[][][] block;
    private int color, size, width, length;

    // Constructor
    public Block(String[] input) {
        int length = input.length, width = input[0].length();
        for (int i = 1; i < length; i++){
            width = Math.max(width, input[i].length());
        }
        this.length = length;
        this.width = width;
        this.size = Math.max(length, width)*2;
        this.block = new int[size][size][size];
        this.color = (int) (input[0].charAt(0));

        for (int i = 0; i < this.size; i++) {
            if (i >= length){
                for (int j = 0; j < this.size; j++)
                    this.block[i][j][0] = 0;
                continue;
            }
            String line = input[i];
            for (int j = 0; j < this.size; j++) {
                if (j >= line.length()) {
                    this.block[i][j][0] = 0;
                    continue;
                }

                char c = line.charAt(j);
                if (c == ' ') {
                    this.block[i][j][0] = 0;
                } else {
                    this.block[i][j][0] = this.color;
                }

                for (int k = 1; k < this.size; k++){
                    this.block[i][j][k] = 0;
                }
            }
        }

        pushLeft();
        pushUp();
    }

    public Block(Block input){
```

```
        this.size = input.size;
        this.color = input.color;
        this.block = new int[this.size][this.size][this.size];
        for (int i = 0; i < this.size; i++){
            for (int j = 0; j < this.size; j++){
                for (int k = 0; k < this.size; k++){
                    this.block[i][j][k] = input.block[i][j][k];
                }
            }
        }
    }

    // getter function
    public int getSize(){
        return this.size;
    }
    public int getColor(){
        return this.color;
    }
    public int getLuas(){
        int luas = 0;
        for (int i = 0; i < this.size; i++){
            for (int j = 0; j < this.size; j++){
                if (this.block[i][j][0] != 0){
                    luas++;
                }
            }
        }
        return luas;
    }

    public void mirror(){
        for (int i = 0; i < this.size; i++){
            for (int j = 0; j < this.size / 2; j++){
                int temp = this.block[i][j][0];
                this.block[i][j][0] = this.block[i][this.size - j - 1][0];
                this.block[i][this.size - j - 1][0] = temp;
            }
        }
        pushLeft();
        pushUp();
    }

    public void rotate() {
        int[][][] temp = new int[this.size][this.size][this.size];
        for (int k = 0; k < this.size; k++){
            for(int i = 0; i < this.size; i++){
                for (int j = 0; j < this.size; j++){
                    temp[i][j][k] = this.block[this.size - j - 1][i][k];
                }
            }
        }
        for (int k = 0; k < this.size; k++){
            for (int i = 0; i < this.size; i++){
                for (int j = 0; j < this.size; j++){
                    this.block[i][j][k] = temp[i][j][k];
                }
            }
        }
        pushLeft();
        pushUp();
    }

    public void lift(){
        for (int i = 0; i < this.size/2; i++){
            for (int j = 0; j < this.size/2; j++){
```

```
                int temp = this.block[i][j][i+j];
                this.block[i][j][i+j] = this.block[i][j][0];
                this.block[i][j][0] = temp;
            }
        }
        pushBottom();
        pushLeft();
        pushUp();
    }

    public void mirrorLift(){
        for (int k = 0; k < this.size/2; k++){
            for (int i = 0; i < k+1; i++){
                for (int j = 0; j < (k+1)/2; j++){
                    int temp = this.block[i][j][k];
                    this.block[i][j][k] = this.block[i][k-j][k];
                    this.block[i][k-j][k] = temp;
                }
            }
        }
        for (int k = this.size/2+1; k < this.size; k++){
            for (int i = k - this.size/2; i < this.size/2; i++){
                for (int j = k - this.size/2; j < k/2; j++){
                    int temp = this.block[i][j][k-1];
                    this.block[i][j][k-1] = this.block[i][k - j - 1][k-1];
                    this.block[i][k - j - 1][k-1] = temp;
                }
            }
        }
    }

    public void print3D() {
        for (int k = this.size-1; k >= 0; k--){
            System.out.println("Layer " + k);
            for (int i = 0; i < this.size/2; i++){
                for (int j = 0; j < this.size/2; j++){
                    if (this.block[i][j][k] == 0) System.out.print("X");
                    else
                    System.out.print((char) this.block[i][j][k]);
                }
                System.out.println();
            }
        }
    }

    public void print(){
        for (int i = 0; i < this.size; i++){
            for (int j = 0; j < this.size; j++){
                System.out.print((char) this.block[i][j][0] + " ");
            }
            System.out.println();
        }
    }

    /**
     * Merapatkan konten block ke kiri
     */
    private void pushLeft() {
        boolean empty = true;
        do{
            for (int i = 0; i < this.size; i++) {
                for (int k = 0; k < this.size; k++){
                    if (block[i][0][k] != 0) {
                        empty = false;
                        break;
                    }
```

```
                }
            }
            if (empty){
                for (int k = 0; k < this.size; k++){
                    for (int i = 0; i < this.size; i++) {
                        for (int j = 0; j < this.size - 1; j++) {
                                block[i][j][k] = block[i][j + 1][k];
                        }
                        block[i][this.size - 1][k] = 0;
                    }
                }
            }
        } while (empty);
    }

    /**
     * Merapatkan konten block ke kiri
     */
    private void pushUp() {
        boolean empty = true;
        do{
            for (int j = 0; j < this.size; j++) {
                for (int k = 0; k < this.size; k++){
                    if (block[0][j][k] != 0) {
                        empty = false;
                        break;
                    }
                }
            }
            if (empty){
                for (int k = 0; k < this.size; k++){
                    for (int i = 0; i < this.size - 1; i++) {
                        for (int j = 0; j < this.size; j++) {
                            block[i][j][k] = block[i + 1][j][k];
                        }
                    }
                    for (int j = 0; j < this.size; j++) {
                        block[this.size - 1][j][k] = 0;
                    }
                }
            }
        } while (empty);
    }

    private void pushBottom(){
        boolean empty = true;
        do {
            for (int i = 0; i < this.size; i++){
                for (int j = 0; j < this.size; j++){
                    if (this.block[i][j][0] != 0) {
                        empty = false;
                        break;
                    }
                }
            }
            if (empty){
                for (int k = 0; k < this.size-1; k++){
                    for (int i = 0; i < this.size; i++){
                        for (int j = 0; j < this.size; j++){
                            this.block[i][j][k] = this.block[i][j][k+1];
                        }
                    }
                }
                for (int i = 0; i < this.size; i++){
                    for (int j = 0; j < this.size; j++){
                        this.block[i][j][this.size-1] = 0;
```

```
                    }
                }
            }
        } while (empty);

    }

    public String getBlockAsString(){
        boolean aktif = false;
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < this.length; i++){
            for (int j = this.width-1; j >= 0; j--){
                if (this.block[i][j][0] > 0){
                    sb.append((char) this.block[i][j][0]);
                    aktif = true;
                }
                else{
                    if (aktif) sb.append(" ");
                }
            }
            sb.append("\n");
        }
        return sb.toString();
    }
}
```

3.2.2 *Solver.java*

```
package puzzle;

public class Solver {
    private int[][][] map;
    private Block[] blocks;
    private int width, length, height = 1, counter, luas = 0;
    private boolean validMap = true;
    private String errorMsg, mode;
    private boolean[] placed;
    private java.time.Duration duration;
    public boolean isSolved;
    public String solveMsg;

    public Solver(int length, int width, String[] input, Block[] blocks){
        this.length = length;
        this.width = width;
        if (this.length != input.length){
            this.validMap = false;
            this.errorMsg = "Invalid map!";
        }
        for (int i = 0; i < this.length; i++){
            if (input[i].length() != width){
                this.validMap = false;
                this.errorMsg = "Invalid map!";
                break;
            }
        }
        this.map = new int[this.length][this.width][this.height];
        if (validMap){
            for(int i = 0; i < this.length; i++){
                String line = input[i];
                for (int j = 0; j < this.width; j++){
                    if (line.charAt(j) == '.'){
                        this.map[i][j][0] = -1;
                    }
                    else if (line.charAt(j) == 'X'){
                        this.map[i][j][0] = 0;
                        this.luas++;
```

```
                    }
                    else{
                        this.validMap = false;
                        this.errorMsg = "Invalid map! Map must only contain \'.\' or
\'X\' character!";
                    }

                }
            }
        }
        this.blocks = blocks;
        this.mode = "CUSTOM";

        this.placed = new boolean[blocks.length];
        for (int i = 0; i < placed.length; i++){
            placed[i] = false;
        }
    }

    public Solver(int length, int width, Block[] blocks){
        this.length = length;
        this.width = width;
        this.map = new int[length][width][this.height];
        for (int i = 0; i < length; i++){
            for (int j = 0; j < width; j++){
                map[i][j][0] = 0;
            }
        }
        this.blocks = blocks;
        this.luas = length * width;
        this.mode = "DEFAULT";
        this.placed = new boolean[blocks.length];
        for (int i = 0; i < placed.length; i++){
            placed[i] = false;
        }
    }

    public Solver(int length, int width, int height, Block[] blocks){
        this.length = length;
        this.width = width;
        if (length != width){
            this.validMap = false;
            this.errorMsg = "Invalid map for pyramid! Row must be equal to column!";
        }
        this.height = height;
        this.map = new int[length][width][height];
        for (int k = 0; k < height; k++){
            for (int i = 0; i < length; i++){
                for (int j = 0; j < width; j++){
                    if (i >= k && j >= k){
                        map[i][j][k] = 0;
                        this.luas++;
                    }
                    else{
                        map[i][j][k] = -1;
                    }
                }
            }
        }
        this.blocks = blocks;
        this.mode = "PYRAMID";
        this.placed = new boolean[blocks.length];
        for (int i = 0; i < placed.length; i++){
            placed[i] = false;
        }
    }
```

```
    public int[][][] getMap(){
        return this.map;
    }

    public int getWidth(){
        return this.width;
    }

    public int getLength(){
        return this.length;
    }

    public int getHeight(){
        return this.height;
    }

    public int getCounter(){
        return this.counter;
    }

    public int getDuration(){
        if (duration == null) return 0;
        return (int) duration.toMillis();
    }

    // normal
    private boolean place(Block block, int row, int col){
        boolean ok = true;
        for (int i = 0; i < block.getSize(); i++){
            for (int j = 0; j < block.getSize(); j++){
                int mapi = i + row, mapj = j + col;
                if (mapi >= this.length || mapj >= this.width){
                    if (block.block[i][j][0] > 0)
                        ok = false;
                    continue;
                }
                if (block.block[i][j][0] > 0 && (this.map[mapi][mapj][0] > 0 ||
this.map[mapi][mapj][0] == -1)){
                    ok = false;
                }
            }
        }
        if (!ok) return false;
        for (int i = 0; i < block.getSize(); i++){
            for (int j = 0; j < block.getSize(); j++){
                int mapi = i + row, mapj = j + col;
                if (mapi < this.length && mapj < this.width && block.block[i][j][0] > 0)
                    this.map[mapi][mapj][0] = block.block[i][j][0];
            }
        }
        return true;
    }

    // pyramid
    private boolean place(Block block, int row, int col, int height){
        boolean ok = true;
        for (int i = 0; i < block.getSize(); i++){
            for (int j = 0; j < block.getSize(); j++){
                for (int k = 0; k < block.getSize(); k++){
                    int mapi = i + row, mapj = j + col, mapk = k + height;
                    if (mapi >= this.length || mapj >= this.width || mapk >=
this.height){
                        if (block.block[i][j][k] > 0)
                            ok = false;
                        continue;
```

```
                    }
                    if (block.block[i][j][k] > 0 && (this.map[mapi][mapj][mapk] > 0 ||
this.map[mapi][mapj][mapk] == -1)){
                        ok = false;
                    }
                }
            }
        }
        if (!ok) return false;
        for (int i = 0; i < block.getSize(); i++){
            for (int j = 0; j < block.getSize(); j++){
                for (int k = 0; k < block.getSize(); k++){
                    int mapi = i + row, mapj = j + col, mapk = k + height;
                    if (mapi < this.length && mapj < this.width && mapk < this.height &&
block.block[i][j][k] > 0)
                        this.map[mapi][mapj][mapk] = block.block[i][j][k];
                }
            }
        }
        return true;
    }

    // normal
    private void unplace(Block block, int row, int col){
        for (int i = 0; i < block.getSize(); i++){
            for (int j = 0; j < block.getSize(); j++){
                int mapi = i + row, mapj = j + col;
                if (mapi < this.length && mapj < this.width && block.block[i][j][0]> 0)
                    this.map[mapi][mapj][0] = 0;
            }
        }
    }

    // pyramid
    private void unplace(Block block, int row, int col, int height){
        for (int i = 0; i < block.getSize(); i++){
            for (int j = 0; j < block.getSize(); j++){
                for (int k = 0; k < block.getSize(); k++){
                    int mapi = i + row, mapj = j + col, mapk = k + height;
                    if (mapi < this.length && mapj < this.width && mapk < this.height &&
block.block[i][j][k] > 0)
                        this.map[mapi][mapj][mapk] = 0;
                }
            }
        }
    }

    private boolean isMapFull(){
        for (int i = 0; i < this.length; i++){
            for (int j = 0; j < this.width; j++){
                for (int k = 0; k < this.height; k++){
                    if (this.map[i][j][k] == 0) return false;
                }
            }
        }
        return true;
    }

    private boolean isPyramidSolved(){
        for (int k = 0; k < this.height; k++){
            for (int i = k; i < this.length; i++){
                for (int j = k; j < this.width; j++){
                    if (this.map[i][j][k] == 0) return false;
                }
            }
        }
```

```
        return true;
    }

    private boolean solveHelper(int blockidx, int i, int j, int rotate){
        this.counter++;
        if (rotate == 8) return solveHelper(blockidx, i, j+1, 0);
        if (j >= this.width) return solveHelper(blockidx, i+1, 0, rotate);
        if (i >= this.length) return false;
        if (blockidx >= blocks.length) return isMapFull();
        Block currBlock = new Block(blocks[blockidx]);
        // rotate block
        for (int temp = 0; temp < rotate%4; temp++) currBlock.rotate();
        if (rotate >= 4) currBlock.mirror();

        if (place(currBlock, i, j)){
            if (solveHelper(blockidx+1, 0, 0, 0) && isMapFull()) return true;

            else unplace(currBlock, i, j);
        }
        return solveHelper(blockidx, i, j, rotate+1);
    }

    private boolean solveHelper(int blockidx, int i, int j, int k, int rotate){
        this.counter++;
        if (rotate == 24) return solveHelper(blockidx, i, j, k+1, 0);
        if (k >= this.height) return solveHelper(blockidx, i, j+1, 0, rotate);
        if (j >= this.width) return solveHelper(blockidx, i+1, 0, 0, rotate);
        if (i >= this.length) return false;
        if (blockidx >= blocks.length) return isPyramidSolved();
        Block currBlock = new Block(blocks[blockidx]);
        // rotate block
        int lift = rotate / 8, rotate2 = rotate % 8;
        for (int temp = 0; temp < rotate2%4; temp++) currBlock.rotate();
        if (rotate2 >= 4) currBlock.mirror();
        if (lift >= 1) currBlock.lift();
        if (lift == 2) currBlock.mirrorLift();



        if (place(currBlock, i, j, k)){
            if (solveHelper(blockidx+1, 0, 0, 0, 0) && isPyramidSolved()) return true;
            else unplace(currBlock, i, j, k);
        }
        return solveHelper(blockidx, i, j, k, rotate+1);
    }

    public void solve(){
        if (!validMap){
            // System.out.println(errorMsg);
            this.isSolved = false;
            this.solveMsg = errorMsg;
            return;
        }

        int totalluas = 0;
        for (int i = 0; i < blocks.length; i++){
            totalluas += blocks[i].getLuas();
        }
        if (totalluas != this.luas){
            this.isSolved = false;
            this.solveMsg = "Not Solvable! Total area of blocks must be equal to the
area of the map!";
            return;
        }

        this.counter = 0;
```

```
        java.time.LocalTime time = java.time.LocalTime.now();
        if (this.mode.equals("PYRAMID")){
            if (solveHelper(0, 0, 0, 0, 0)){
                System.out.println();
                this.printPyramid();
                this.isSolved = true;
            }
            else{
                this.isSolved = false;
                this.solveMsg = "Not Solvable!";
            }
        }
        else {
            if (solveHelper(0, 0, 0, 0)){
                this.isSolved = true;
                System.out.println();
                this.print();
            }
            else{
                this.isSolved = false;
                this.solveMsg = "Not Solvable!";
            }
        }
        this.duration = java.time.Duration.between(time, java.time.LocalTime.now());
        System.out.println("\nBanyak kasus yang ditinjau: " + counter);
        System.out.println("\nWaktu pencarian: " + duration.toMillis() + " ms\n");
    }
    public void print(){
        for (int i = 0; i < this.length; i++){
            for (int j = 0; j < this.width; j++){
                if (map[i][j][0] > 0)
                    colorPrint((char) this.map[i][j][0]);
                else
                    System.out.print(" ");
            }
            System.out.println();
        }
    }

    public void printPyramid(){
        for (int k = this.height-1; k >= 0; k--){
            for (int i = k; i < this.length; i++){
                for (int j = k; j < this.width; j++){
                    if (map[i][j][k] > 0)
                        colorPrint((char) this.map[i][j][k]);
                    else
                        System.out.print(" ");
                }
                System.out.println();
            }
            System.out.println();
        }
    }

    public String getOutput(){
        StringBuilder sb = new StringBuilder();

        if (this.mode.equals("PYRAMID")){
            for (int k = this.height-1; k >= 0; k--){
                for (int i = k; i < this.length; i++){
                    for (int j = k; j < this.width; j++){
                        if (map[i][j][k] > 0)
                            sb.append((char) this.map[i][j][k]);
                        else
                            sb.append(" ");
                    }
```

```
                    sb.append("\n");
                }
                sb.append("\n");
            }
        }
        else{
            for (int i = 0; i < this.length; i++){
                for (int j = 0; j < this.width; j++){
                    if (map[i][j][0] > 0)
                        sb.append((char) this.map[i][j][0]);
                    else
                        sb.append(" ");
                }
                sb.append("\n");
            }
        }
        sb.append("\nBanyak kasus yang ditinjau: " + counter + "\n");
        sb.append("\nWaktu pencarian: " + duration.toMillis() + " ms\n");
        return sb.toString();
    }

    private void colorPrint(char c){
        int colorCode = (c - 'A') + 1;
        System.out.print("\u001B[38;5;" + colorCode + "m" + c + "\u001B[0m");
    }

}
```

3.2.3 *FileReader.java*

```
package puzzle;

import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.List;

public class FileReader {

    public int row, col, numBlocks;
    public String[] map;
    public Block[] blocks;
    public boolean validInput;
    public String mode, errorMsg;

    public void readInput(Path filePath){
        this.validInput = true;
        try {
            List<String> lines = Files.readAllLines(filePath);
            String[] firstLine = lines.get(0).split(" ");
            if (firstLine.length != 3){
                this.validInput = false;
                this.errorMsg = "Invalid input! First line must contain 3 integers
only!";
                return;
            }
            for (int i = 0; i < firstLine.length; i++){
                if (!isNumeric(firstLine[i])){
                    this.validInput = false;
                    this.errorMsg = "Invalid input! First line must contain 3
integers!";
                    return;
                }
            }
            this.row = Integer.parseInt(firstLine[0]);
```

```
            this.col = Integer.parseInt(firstLine[1]);
            this.numBlocks = Integer.parseInt(firstLine[2]);

            this.mode = lines.get(1);
            if (!this.mode.equals("DEFAULT") && !this.mode.equals("CUSTOM") &&
!this.mode.equals("PYRAMID")){
                this.validInput = false;
                this.errorMsg = "Invalid mode! Mode must be either DEFAULT, CUSTOM, or
PYRAMID";
                return;
            }

            int startBlockIdx = 2;
            if (this.mode.equals("CUSTOM")){
                List<String> mapLines = lines.subList(2, 2+this.row);
                startBlockIdx += this.row;
                if (!parseMap(mapLines, mode)){
                    return;
                }
            }

            List<String> blockLines = lines.subList(startBlockIdx, lines.size());
            if (!parseBlocks(blockLines)){
                this.validInput = false;
                this.errorMsg = "Invalid blocks input!";
                return;
            }
        }
        catch (IOException e){
            this.validInput = false;
            this.errorMsg = ("Error reading file: " + e.getMessage());
        }
    }

    public void readInput(String row, String col, String mode, String numBlocks, String
blocks, String map){
        this.validInput = true;
        if (!isNumeric(row) || !isNumeric(col) || !isNumeric(numBlocks)){
            this.validInput = false;
            this.errorMsg = "Invalid input! Row, column, and number of blocks must be
integers!";
            return;
        }
        this.row = Integer.parseInt(row);
        this.col = Integer.parseInt(col);
        if (this.row < 1 || this.col < 1){
            this.validInput = false;
            this.errorMsg = "Invalid input! Row and column must be greater than 0!";
            return;
        }
        this.numBlocks = Integer.parseInt(numBlocks);
        if (this.numBlocks < 1){
            this.validInput = false;
            this.errorMsg = "Invalid input! Number of blocks must be greater than 0!";
            return;
        }
        this.mode = mode;

        if (mode.equals("CUSTOM")){
            List<String> mapLines = java.util.Arrays.asList(map.split("\n"));
            if (!parseMap(mapLines, mode))
                return;
        }

        List<String> blockLines = java.util.Arrays.asList(blocks.split("\n"));
        if (!parseBlocks(blockLines)){
```

```
            return;
        }
    }

    private boolean parseMap(List<String> lines, String mode){
        if (lines == null || lines.size() == 0){
            this.validInput = false;
            this.errorMsg = "Invalid map! Map must not be empty!";
            return false;
        }
        if (this.row != lines.size()){
            this.validInput = false;
            this.errorMsg = "Invalid map! Number of rows must does not match!";
            return false;
        }
        for (int i = 0; i < lines.size(); i++){
            if (this.col != lines.get(i).length()){
                this.validInput = false;
                this.errorMsg = "Invalid map! Number of columns does not match!";
                return false;
            }
        }

        if (mode.equals("PYRAMID")) {
            if (this.row != this.col){
                this.validInput = false;
                this.errorMsg = "Invalid map for pyramid! Row must be equal to Column!";
                return false;
            }
            this.map = new String[this.row];
        }
        else{
            this.map = new String[this.row];
        }

        for(int i = 0; i < this.row; i++){
            String line = lines.get(i);
            for (int j = 0; j < this.col; j++){
                if(line.charAt(i) != 'X' && line.charAt(i) != '.'){
                    this.validInput = false;
                    this.errorMsg = "Invalid map! Map must only contain \'.\' or \'X\'
character!";
                }
            }
            this.map[i] = line;
        }

        return true;
    }

    private boolean parseBlocks(List<String> lines){
        if (lines == null || lines.size() == 0){
            this.validInput = false;
            this.errorMsg = "Invalid blocks input! Blocks must not be empty!";
            return false;
        }
        boolean[] used = new boolean[26];
        for (int i = 0; i < 26; i++){
            used[i] = false;
        }
        this.blocks = new Block[this.numBlocks];
        List<String> temp = new java.util.ArrayList<>();
        int j = 0;
        for (int i = 0; i < lines.size(); i++){
            String blockLine = lines.get(i);
            if (blockLine.length() == 0) continue;
```

```
            if (!isUppercase(blockLine)) {
                this.validInput = false;
                this.errorMsg = "Invalid blocks input! Block must only contain uppercase
letter!";
                return false;
            }
            if (!allSameChar(blockLine)) {
                this.validInput = false;
                this.errorMsg = "Invalid blocks input! Block must only contain one type
of character!";
                return false;
            }

            if (temp.size() == 0 || getChar(temp.get(0)) == getChar(blockLine)){
                temp.add(blockLine);
            }
            else{
                if (j >= this.numBlocks){
                    this.validInput = false;
                    this.errorMsg = "Invalid blocks input! Number of blocks does not
match!";
                    return false;
                }

                String[] block = new String[temp.size()];
                for (int k = 0; k < temp.size(); k++){
                    block[k] = temp.get(k);
                }
                this.blocks[j] = new Block(block);
                if (used[this.blocks[j].getColor() - 'A']){
                    this.validInput = false;
                    this.errorMsg = "Invalid blocks input! Duplicate block color!";
                    return false;
                }
                used[this.blocks[j].getColor() - 'A'] = true;
                j++;
                temp.clear();
                temp.add(blockLine);
            }
        }
        if (temp.size() > 0){
            if (j >= this.numBlocks) {
                this.validInput = false;
                this.errorMsg = "Invalid blocks input! Number of blocks does not
match!";
                return false;
            }
            String[] block = new String[temp.size()];
            for (int k = 0; k < temp.size(); k++){
                block[k] = temp.get(k);
            }
            this.blocks[j] = new Block(block);
            if (used[this.blocks[j].getColor() - 'A']){
                this.validInput = false;
                this.errorMsg = "Invalid blocks input! Duplicate block color!";
                return false;
            }
            used[this.blocks[j].getColor() - 'A'] = true;
            temp.clear();
            j++;
        }

        if (j != this.numBlocks){
            this.validInput = false;
            this.errorMsg = "Invalid blocks input! Number of blocks does not match!";
            return false;
```

```
        }
        return true;
    }

    public String getMapAsString(){
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < this.row; i++){
            sb.append(this.map[i]);
            sb.append("\n");
        }
        return sb.toString();
    }

    public String getBlocksAsString(){
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < this.numBlocks; i++){
            sb.append(this.blocks[i].getBlockAsString());
        }
        return sb.toString();
    }

    private char getChar(String s){
        for (int i = 0; i < s.length(); i++){
            if (Character.isLetter(s.charAt(i))) return s.charAt(i);
        }
        return ' ';
    }

    private boolean allSameChar(String s){
        char c = getChar(s);
        for (int i = 0; i < s.length(); i++){
            if (Character.isLetter(s.charAt(i)) && s.charAt(i) != c) return false;
        }
        return true;
    }

    private boolean isUppercase(String s){
        for (int i = 0; i < s.length(); i++){
            if (!Character.isUpperCase(s.charAt(i)) && s.charAt(i) != ' ') return false;
        }
        return true;
    }

    private boolean isNumeric(String str) {
        if (str == null) return false;
        try {
            Integer.parseInt(str);
            return true;
        } catch (NumberFormatException e) {
            return false;
        }
    }

}
```

3.2.4 *MainController.java*

```
package com.myapp;

import java.io.File;
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;
import javafx.application.Platform;
import javafx.collections.FXCollections;
```

```java
import javafx.concurrent.Task;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.geometry.Pos;
import java.awt.image.BufferedImage;
import javafx.scene.SnapshotParameters;
import javafx.scene.control.Button;
import javafx.scene.control.ChoiceBox;
import javafx.scene.control.ScrollPane;
import javafx.scene.control.TextArea;
import javafx.scene.control.TextField;
import javafx.scene.image.PixelFormat;
import javafx.scene.image.WritableImage;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.Pane;
import javafx.scene.layout.StackPane;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.scene.text.Font;
import javafx.scene.text.Text;
import javafx.scene.text.TextAlignment;
import javafx.scene.text.TextBoundsType;
import javafx.stage.FileChooser;
import javafx.stage.Stage;
import javax.imageio.ImageIO;
import puzzle.FileReader;
import puzzle.Solver;

public class MainController {

    @FXML
    private Text alertMsg;

    @FXML
    private TextArea blocksInput;

    @FXML
    private TextField colInput;

    @FXML
    private Text filename;

    @FXML
    private ScrollPane gridContainer;

    @FXML
    private TextArea mapInput;

    @FXML
    private Pane mapInputContainer;

    @FXML
    private ChoiceBox<String> modeInput;

    @FXML
    private TextField numBlocksInput;

    @FXML
    private TextField rowInput;

    @FXML
    private Button saveButtonPng;

    @FXML
    private Button saveButtonTxt;
```

```
    private VBox container;

    private FileChooser fileChooser = new FileChooser();
    private File inputFile;
    private FileReader reader = new FileReader();
    private Solver solver;
    private final Map<Character, Color> colorMap = new HashMap<>();

    @FXML
    public void initialize(){
        modeInput.setItems(FXCollections.observableArrayList("DEFAULT", "CUSTOM",
"PYRAMID"));
        modeInput.setValue("DEFAULT");
        modeInput.getSelectionModel().selectedItemProperty().addListener((observable,
oldValue, newValue) -> {
            if ("CUSTOM".equals(newValue)) {
                mapInputContainer.setDisable(false);
            } else {
                mapInput.clear();
                mapInputContainer.setDisable(true);
            }
        });
        mapInputContainer.setDisable(true);
        initColor();
        saveButtonPng.setDisable(true);
        saveButtonTxt.setDisable(true);
    }

    @FXML
    void onClickUploadFile(ActionEvent event) {
        alertMsg.setText("");
        alertMsg.setFill(Color.RED);

        File initialDirectory = new File("./test");
        if (initialDirectory.exists()) {
            fileChooser.setInitialDirectory(initialDirectory);
        }
        fileChooser.setTitle("Open Text File");
        fileChooser.getExtensionFilters().clear();
        fileChooser.getExtensionFilters().add(new FileChooser.ExtensionFilter("Text
Files", "*.txt"));

        inputFile = fileChooser.showOpenDialog(new Stage());

        if (inputFile == null) {
            return;
        }

        filename.setText(inputFile.getName());

        reader.readInput(inputFile.toPath());
        if (!reader.validInput){
            alertMsg.setText(reader.errorMsg);
            return;
        }

        rowInput.setText(reader.row + "");
        colInput.setText(reader.col + "");
        numBlocksInput.setText(reader.numBlocks + "");
        modeInput.setValue(reader.mode);
        if (reader.mode.equals("CUSTOM")){
            mapInput.setText(reader.getMapAsString());
        }
        blocksInput.setText(reader.getBlocksAsString());
    }
```

```
    @FXML
    void onClickSolve(ActionEvent event) {
        saveButtonTxt.setDisable(true);
        saveButtonPng.setDisable(true);
        gridContainer.setContent(null);
        alertMsg.setText("");
        alertMsg.setFill(Color.RED);
        reader.readInput(rowInput.getText(), colInput.getText(), modeInput.getValue(),
numBlocksInput.getText(), blocksInput.getText(), mapInput.getText());

        if (!reader.validInput){
            alertMsg.setText(reader.errorMsg);
            return;
        }
        if (reader.mode.equals("DEFAULT")){
            solver = new Solver(reader.row, reader.col, reader.blocks);
        }
        else if (reader.mode.equals("CUSTOM")){
            solver = new Solver(reader.row, reader.col, reader.map, reader.blocks);
        }
        else{
            solver = new Solver(reader.row, reader.col, reader.row, reader.blocks);
        }
        container = new VBox(5);
        container.setAlignment(Pos.CENTER);

        Task<Void> solveTask = new Task<Void>() {
            @Override
            protected Void call() {
                if (reader.mode.equals("DEFAULT")) {
                    solver = new Solver(reader.row, reader.col, reader.blocks);
                }
                else if (reader.mode.equals("CUSTOM")) {
                    solver = new Solver(reader.row, reader.col, reader.map,
reader.blocks);
                }
                else {
                    solver = new Solver(reader.row, reader.col, reader.row,
reader.blocks);
                }

                solver.solve();  // Solve on background thread
                return null;
            }
        };

        // When solver.solve() finishes, update the UI on the JavaFX Application Thread
        solveTask.setOnSucceeded(e -> {
            if (!solver.isSolved) {
                alertMsg.setFill(Color.RED);
                alertMsg.setText(solver.solveMsg + "\nBanyak kasus ditinjau: " +
solver.getCounter() + "                    Waktu Pencarian: " + solver.getDuration() +
" ms");
            }
            else {
                container.getChildren().clear();
                for (int k = solver.getHeight() - 1; k >= 0; k--) {
                    GridPane grid = createGrid(k);
                    container.getChildren().add(grid);
                }

                gridContainer.setContent(container);
                container.setPrefHeight(Math.max(gridContainer.getHeight(),
container.getHeight()));
                container.setPrefWidth(Math.max(gridContainer.getWidth(),
```

```
container.getWidth()));

                alertMsg.setFill(Color.GREEN);
                alertMsg.setText("Banyak kasus ditinjau: " + solver.getCounter() +
"                   Waktu Pencarian: " + solver.getDuration() + " ms");
                saveButtonPng.setDisable(false);
                saveButtonTxt.setDisable(false);
            }
        });

        solveTask.setOnFailed(e -> {
            alertMsg.setText("Error: Failed to solve.");
        });

        Task<Void> setCounter = new Task<>() {
            @Override
            protected Void call() {
                alertMsg.setFill(Color.BLACK);
                while (!solveTask.isDone()) {
                    // Update the UI safely
                    Platform.runLater(() -> {
                        container.getChildren().clear();
                        for (int k = solver.getHeight() - 1; k >= 0; k--) {
                            GridPane grid = createGrid(k);
                            container.getChildren().add(grid);
                        }

                        gridContainer.setContent(container);
                        container.setPrefHeight(Math.max(gridContainer.getHeight(),
container.getHeight()));
                        container.setPrefWidth(Math.max(gridContainer.getWidth(),
container.getWidth()));
                        alertMsg.setText("Banyak kasus ditinjau: " +
solver.getCounter());
                    });

                    try {
                        Thread.sleep(100); // Small delay to prevent excessive UI
updates
                    } catch (InterruptedException e) {
                        Thread.currentThread().interrupt();
                    }
                }
                return null;
            }
        };

        new Thread(solveTask).start();
        new Thread(setCounter).start();
    }

    @FXML
    void onClickSaveResPng(ActionEvent event) {
        // Set default file name and extension
        fileChooser.setInitialFileName("output.png");
        fileChooser.getExtensionFilters().clear();
        fileChooser.getExtensionFilters().add(new FileChooser.ExtensionFilter("PNG
Files", "*.png"));
        File initialDirectory = new File(".");
        if (initialDirectory.exists()) {
            fileChooser.setInitialDirectory(initialDirectory);
        }
        // Show save dialog
        File file = fileChooser.showSaveDialog(new Stage());

        if (file != null){
```

```
            // Ensure JavaFX thread
            Platform.runLater(() -> {
                try {
                    // Capture snapshot
                    WritableImage image = container.snapshot(new SnapshotParameters(),
null);

                    // Convert to BufferedImage manually
                    int width = (int) image.getWidth();
                    int height = (int) image.getHeight();
                    BufferedImage bufferedImage = new BufferedImage(width, height,
BufferedImage.TYPE_INT_ARGB);

                    int[] buffer = new int[width * height];
                    image.getPixelReader().getPixels(0, 0, width, height,
PixelFormat.getIntArgbInstance(), buffer, 0, width);
                    bufferedImage.setRGB(0, 0, width, height, buffer, 0, width);

                    // Save as PNG
                    ImageIO.write(bufferedImage, "png", file);
                    System.out.println("Image saved to: " + file.getAbsolutePath());
                }
                catch (IOException e) {
                    e.printStackTrace();
                }
            });
        }
    }

    @FXML
    void onClickSaveResTxt(ActionEvent event) {
        // Set default file name and extension
        fileChooser.setInitialFileName("output.txt");
        fileChooser.getExtensionFilters().clear();
        fileChooser.getExtensionFilters().add(new FileChooser.ExtensionFilter("Text
Files", "*.txt"));
        File initialDirectory = new File(".");
        if (initialDirectory.exists()) {
            fileChooser.setInitialDirectory(initialDirectory);
        }
        // Show save dialog
        File file = fileChooser.showSaveDialog(new Stage());

        if (file != null){
            // Ensure JavaFX thread
            String output = solver.getOutput();
            Platform.runLater(() -> {
                try {
                    // Write to file
                    java.nio.file.Files.write(file.toPath(), output.getBytes());
                    System.out.println("Text saved to: " + file.getAbsolutePath());
                }
                catch (IOException e) {
                    e.printStackTrace();
                }
            });
        }
    }

    private GridPane createGrid(int height){
        int size=25, width = solver.getWidth() - height, length = solver.getLength() -
height;
        GridPane grid = new GridPane();
        grid.setHgap(5); grid.setVgap(5);
        grid.setPadding(new javafx.geometry.Insets(5));
        grid.setAlignment(Pos.CENTER);
```

```
        grid.setStyle("-fx-background-color: rgb(120,120,120); -fx-background-radius:
10px;");
        grid.setMaxWidth((size + 5)*width+5);
        grid.setMaxHeight((size + 5)*length+5);
        for (int row = height; row < solver.getLength(); row++) {
            for (int col = height; col < solver.getWidth(); col++) {
                char letter;
                if (solver.getMap()[row][col][height] > 0) {
                    letter = (char) solver.getMap()[row][col][height];
                }
                else if (solver.getMap()[row][col][height] == -1){
                    letter = '-';
                }
                else{
                    letter = '\0';
                }
                Color color = colorMap.getOrDefault(letter, Color.GRAY); // Default to
gray if unknown
                Rectangle rect = new Rectangle(size, size);
                rect.setFill(color);
                rect.setArcHeight(size * 0.9);
                rect.setArcWidth(size * 0.9);
                // Create the text
                if (letter == '-') letter = '\0';
                Text text = new Text(letter + "");
                text.setFill(Color.WHITE);
                text.setFont(new Font(size * 0.5));
                text.setTextAlignment(TextAlignment.CENTER);
                text.setBoundsType(TextBoundsType.VISUAL);
                text.setStyle("-fx-font-weight: bold;");

                // Stack rectangle and text together
                StackPane stack = new StackPane();
                stack.getChildren().addAll(rect, text);
                stack.setAlignment(Pos.CENTER);
                grid.add(stack, col - height, row - height);
            }
        }
        return grid;
    }

    private void initColor(){
        colorMap.put('-', Color.rgb(120, 120, 120));
        colorMap.put('A', Color.BLUE);
        colorMap.put('B', Color.RED);
        colorMap.put('C', Color.GREEN);
        colorMap.put('D', Color.ORANGE);
        colorMap.put('E', Color.PURPLE);
        colorMap.put('F', Color.YELLOWGREEN);
        colorMap.put('G', Color.CYAN);
        colorMap.put('H', Color.MAGENTA);
        colorMap.put('I', Color.BROWN);
        colorMap.put('J', Color.PINK);
        colorMap.put('K', Color.BLACK);
        colorMap.put('L', Color.LIGHTBLUE);
        colorMap.put('M', Color.LIMEGREEN);
        colorMap.put('N', Color.GOLD);
        colorMap.put('O', Color.INDIGO);
        colorMap.put('P', Color.DARKGREEN);
        colorMap.put('Q', Color.DARKRED);
        colorMap.put('R', Color.DARKBLUE);
        colorMap.put('S', Color.DARKORANGE);
        colorMap.put('T', Color.DARKVIOLET);
        colorMap.put('U', Color.TEAL);
        colorMap.put('V', Color.SKYBLUE);
        colorMap.put('W', Color.SILVER);
```

```
        colorMap.put('X', Color.MIDNIGHTBLUE);
        colorMap.put('Y', Color.SALMON);
        colorMap.put('Z', Color.OLIVE);
    }

}
```

3.2.5 *App.java*

```
package com.myapp;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.image.Image;
import javafx.stage.Stage;

import java.io.IOException;

/**
 * JavaFX App
 */
public class App extends Application {

    private static Scene scene;

    @Override
    public void start(Stage stage) throws IOException {
        scene = new Scene(loadFXML("main-controller"));
        stage.setScene(scene);
        stage.setResizable(false);
        stage.setTitle("IQ Puzzle Pro Solver");
        stage.getIcons().add(new Image(("file:./src/main/resources/icon.jpg"));
        stage.show();
    }

    static void setRoot(String fxml) throws IOException {
        scene.setRoot(loadFXML(fxml));
    }

    private static Parent loadFXML(String fxml) throws IOException {
        FXMLLoader fxmlLoader = new FXMLLoader(App.class.getResource(fxml + ".fxml"));
        return fxmlLoader.load();
    }

    public static void main(String[] args) {
        launch();
    }
}
```

BAB IV

MASUKAN DAN KELUARAN PROGRAM

4.1 Kasus uji 1 – DEFAULT Solved

Masukan:

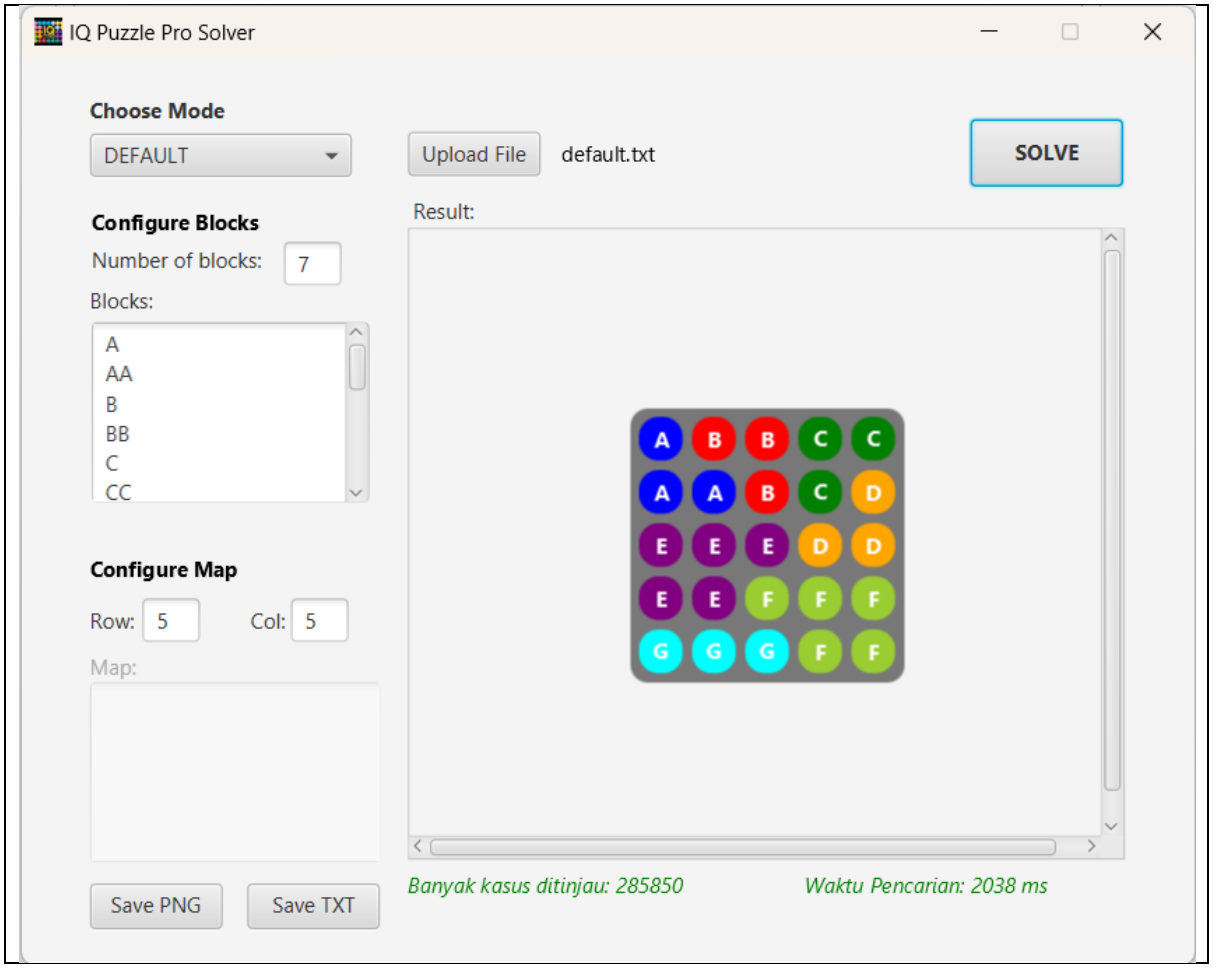
```
5 5 7
DEFAULT
A
AA
B
BB
C
CC
D
DD
EE
EE
E
FF
FF
F
GGG
```

Keluaran:

```
ABBCC
AABCD
EEEDD
EEFFF
GGGFF

Banyak kasus yang ditinjau: 285850

Waktu pencarian: 2038 ms
```



4.2 Kasus uji 2 – CUSTOM Solved

Masukan:

```
5 7 5
CUSTOM
...X...
.XXXXX.
XXXXXXX
.XXXXX.
...X...
A
AAA
BB
BBB
CCCC
 C
D
EEE
E
```

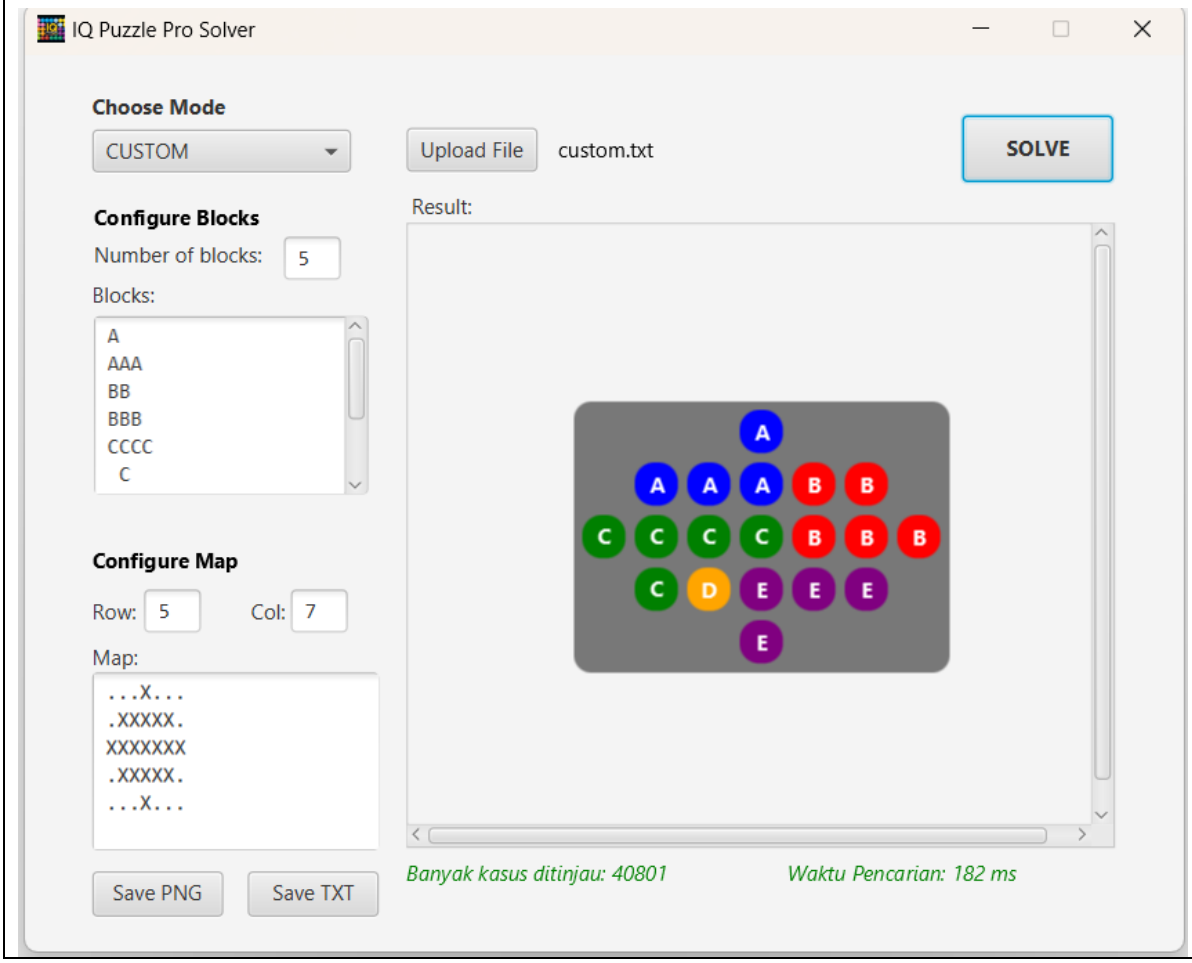
Keluaran:

```
   A
 AAABB
CCCCBBB
 DCEEE
```

```
   E

Banyak kasus yang ditinjau: 40796

Waktu pencarian: 271 ms
```



4.3 Kasus uji 3 – PYRAMID Solved

Masukan:

```
3 3 6
PYRAMID
A
AA
BB
CC
DDD
E
EE
F
```

Keluaran:

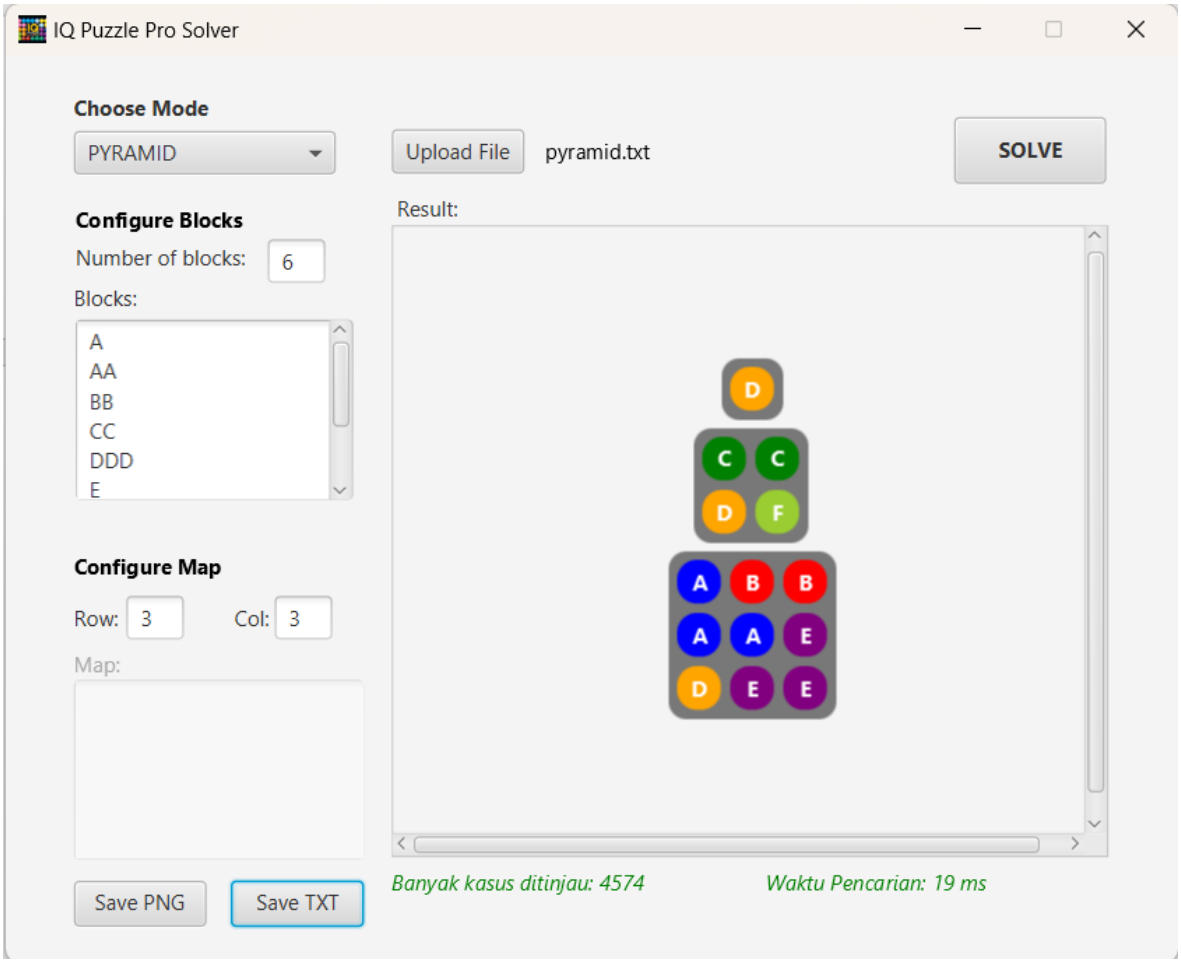
```
D

CC
DF
```

```
ABB
AAE
DEE


Banyak kasus yang ditinjau: 4574

Waktu pencarian: 19 ms
```

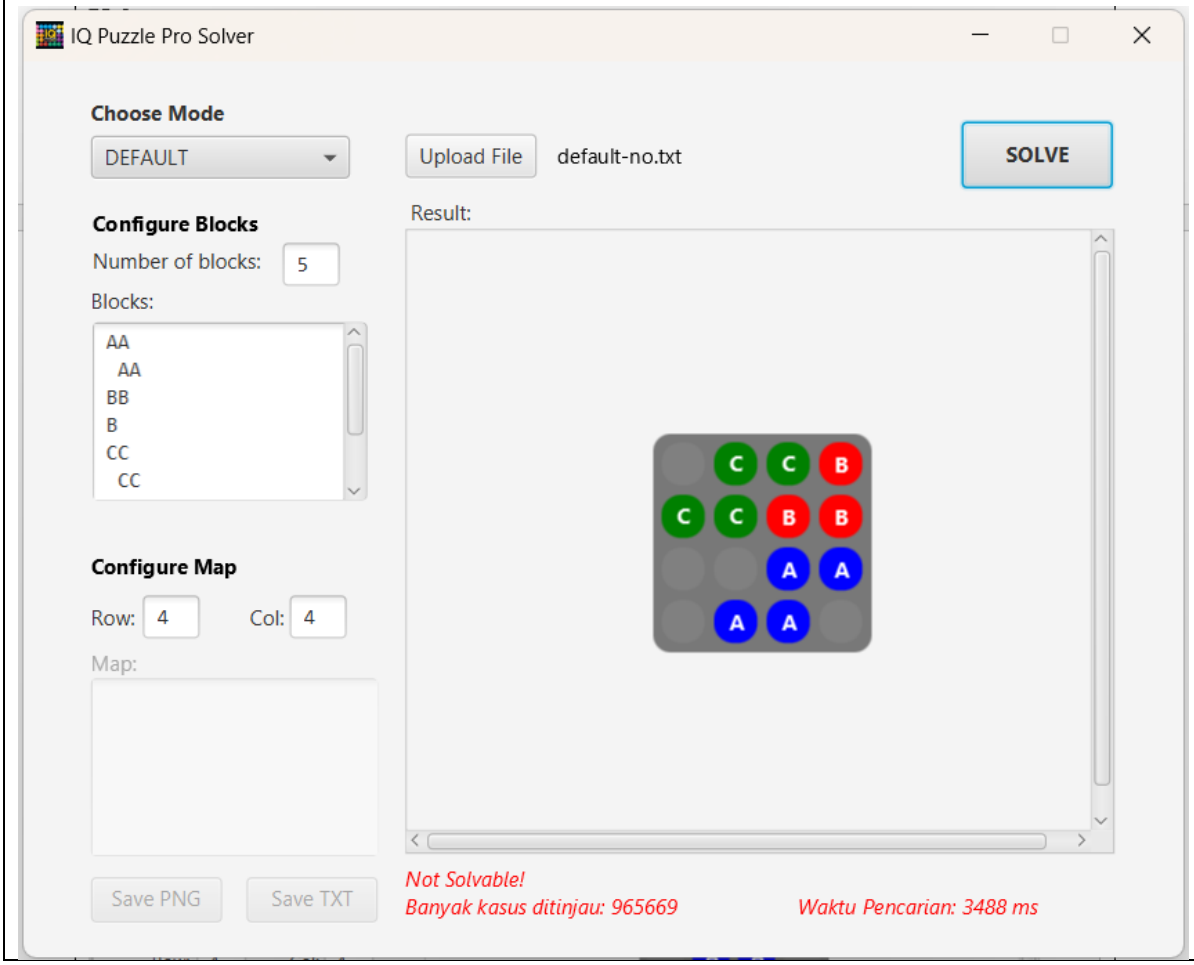


4.4 Kasus uji 4 – DEFAULT Not Solvable

Masukan:

```
4 4 5
DEFAULT
AAA
BB
B
CCC
CC
DDDDD
EE
EE
```

Keluaran:

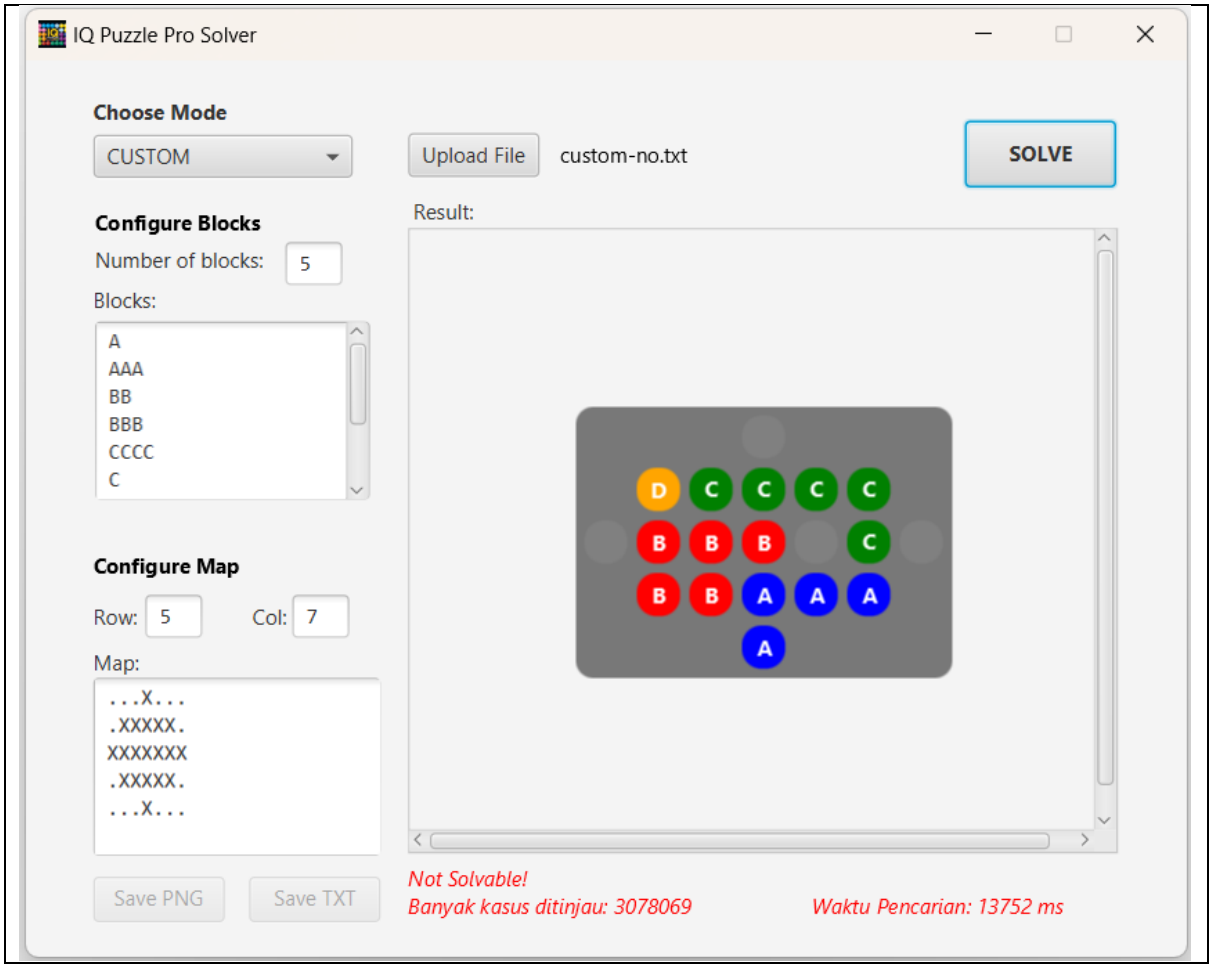


4.5 Kasus uji 5 – CUSTOM Not Solvable

Masukan:

```
5 7 5
CUSTOM
...X...
.XXXXX.
XXXXXXX
.XXXXX.
...X...
A
AAA
BB
BBB
CCCC
C
D
EEE
E
```

Keluaran:

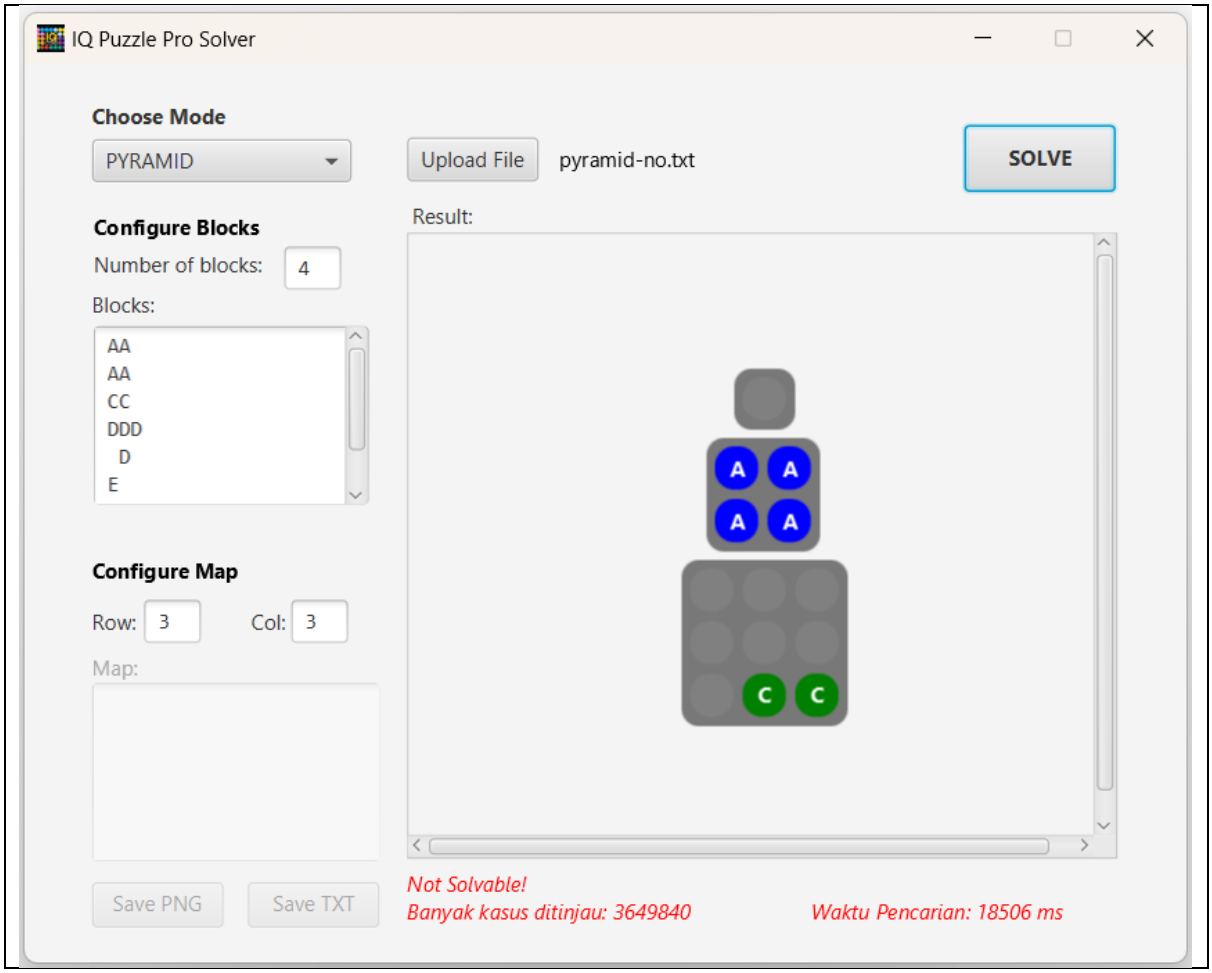


4.6 Kasus uji 6 – PYRAMID Not Solvable

Masukan:

```
3 3 4
PYRAMID
AA
AA
CC
DDD
 D
E
EEE
```

Keluaran:



4.7 Kasus uji 7 – PYRAMID Solved

Masukan:

```
4 4 11
PYRAMID
A
AA
BB
CC
DDD
E
EEE
FF
GG
 GG
HH
H
H
II
I
J
KK
```

Keluaran:

```
I
```

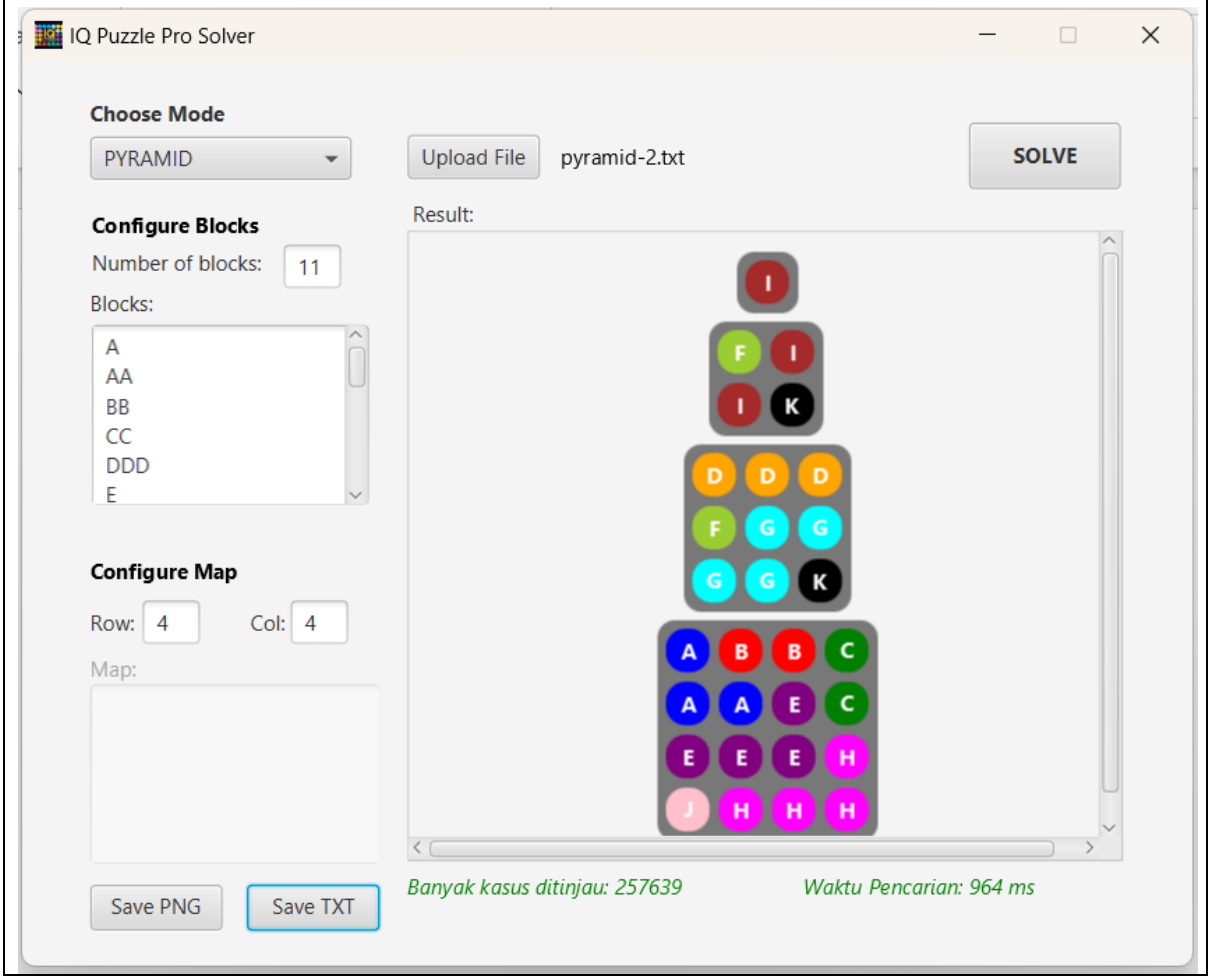
```
FI
IK

DDD
FGG
GGK

ABBC
AAEC
EEEH
JHHH


Banyak kasus yang ditinjau: 257639

Waktu pencarian: 964 ms
```



BAB V

LAMPIRAN

No	Poin	Ya	Tidak
1	Program berhasil dikompilasi tanpa kesalahan	✓	
2	Program berhasil dijalankan	✓	
3	Solusi yang diberikan program benar dan mematuhi aturan permainan	✓	
4	Program dapat membaca masukan berkas .txt serta menyimpan solusi dalam berkas .txt	✓	
5	Program memiliki *Graphical User Interface* (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	✓	

REFERENSI

R. Munir, *Algoritma Brute Force – Bagian 1*, Institut Teknologi Bandung, 2025. [Online]. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/02-Algoritma-Brute-Force-(2025)-Bag1.pdf.