

Tugas Kecil 3 IF2211 Strategi Algoritma
Penyelesaian Puzzle Rush Hour Menggunakan Algoritma
Pathfinding



Disusun oleh:

Bertha Soliany Frandi 13523026

Rafen Max Alessandro 13523031

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2025

DAFTAR ISI

DAFTAR ISI.....	2
BAB I.....	4
DESKRIPSI MASALAH.....	4
BAB II.....	6
ALGORITMA PATHFINDING.....	6
2.1 Algoritma UCS.....	6
2.2 Algoritma Greedy Best First Search.....	6
2.3 Algoritma A*.....	7
BAB III.....	9
ANALISIS ALGORITMA PATHFINDING.....	9
3.1 Definisi $f(n)$ dan $g(n)$	9
3.2 Nilai Heuristik pada Algoritma A*.....	10
3.3 Perbandingan Algoritma UCS dan BFS.....	11
3.4 Perbandingan Algoritma A* dan UCS.....	11
3.5 Analisis Solusi Greedy Best First Search.....	12
BAB IV.....	14
SOURCE CODE PROGRAM.....	14
4.1 Struktur Folder.....	14
4.2 Main.java.....	14
4.3 Algorithm.java.....	19
4.4 Board.java.....	22
4.5 Heuristic.java.....	31
4.6 InputOutput.java.....	34
4.7 Piece.java.....	43
4.8 Simpul.java.....	46
BAB V.....	48
MASUKAN DAN LUARAN PROGRAM.....	48
5.1 Program Utama.....	48
5.2 Validasi Input.....	49
5.3 Algoritma UCS.....	57
5.3.1 Test Case 1.....	57
5.3.2 Test Case 2.....	59
5.3.3 Test Case 3.....	61
5.3.4 Test Case 4.....	63
5.3.5 Test Case 5.....	65
5.4 Algoritma Greedy Best First Search.....	66

5.4.1 Test Case 1.....	66
5.4.2 Test Case 2.....	68
5.4.3 Test Case 3.....	70
5.4.4 Test Case 4.....	72
5.4.5 Test Case 5.....	74
5.5 Algoritma A*.....	75
5.5.1 Test Case 1.....	75
5.5.2 Test Case 2.....	76
5.5.3 Test Case 3.....	78
5.5.4 Test Case 4.....	80
5.5.5 Test Case 5.....	81
BAB VI.....	83
ANALISIS PERCOBAAN.....	83
BAB VII.....	86
KESIMPULAN.....	86
BAB VIII.....	87
LAMPIRAN.....	87
1. Referensi.....	87
2. Pranala repository GitHub.....	87
3. Tabel Checklist.....	87

BAB I

DESKRIPSI MASALAH

Rush Hour adalah sebuah permainan puzzle logika berbasis *grid* yang menantang pemain untuk menggeser kendaraan di dalam sebuah kotak (biasanya berukuran 6 x 6) agar mobil utama (umumnya berwarna merah) dapat keluar dari kemacetan melalui pintu keluar di sisi papan. Setiap kendaraan hanya bisa bergerak lurus ke depan atau ke belakang sesuai dengan orientasinya (horizontal atau vertikal), dan tidak dapat berputar. Tujuan utama dari permainan ini adalah memindahkan mobil merah ke pintu keluar dengan jumlah langkah seminimal mungkin.

Komponen penting dari permainan Rush Hour terdiri dari:

1. Papan

Papan merupakan tempat permainan dimainkan. Papan terdiri atas *cell*, yaitu sebuah poin singular pada papan. Sebuah *piece* akan menempati *cell-cell* pada papan. Permainan diawali dengan meletakkan *piece* di dalam papan dengan konfigurasi tertentu, terdiri atas lokasi dan orientasi dari *piece* tersebut, antara horizontal atau vertikal. Hanya *primary piece* yang dapat digerakkan keluar papan melewati pintu keluar. *Piece* yang bukan *primary piece* tidak dapat digerakkan keluar papan. Papan memiliki satu pintu keluar yang pasti berada di dinding papan dan sejajar dengan orientasi *primary piece*.

2. Piece

Piece adalah sebuah kendaraan di dalam papan. Setiap *piece* memiliki posisi, ukuran, dan orientasi. Orientasi sebuah *piece* hanya dapat berupa horizontal atau vertikal—tidak mungkin diagonal. *Piece* dapat memiliki beragam ukuran, yaitu jumlah *cell* yang ditempati oleh *piece*. Secara standar, variasi ukuran sebuah *piece* adalah 2-piece (menempati 2 *cell*) atau 3-piece (menempati 3 *cell*). Suatu *piece* tidak dapat digerakkan melewati/menembus *piece* yang lain.

3. Primary Piece

Primary piece adalah kendaraan utama yang harus dikeluarkan dari papan (biasanya berwarna merah). Hanya boleh terdapat satu *primary piece* pada sebuah permainan.

4. Pintu Keluar

Pintu keluar adalah tempat bagi *primary piece* digerakkan keluar untuk menyelesaikan permainan.

5. Gerakan

Gerakan adalah pergeseran *piece* di dalam permainan. *Piece* hanya dapat bergerak/bergeser lurus sesuai orientasinya (atas-bawah jika vertikal dan kiri-kanan jika horizontal). Suatu *piece* tidak dapat digerakkan melewati/menembus *piece* yang lain.

BAB II

ALGORITMA PATHFINDING

2.1 Algoritma UCS

Algoritma *Uniform Cost Search* atau UCS adalah salah satu algoritma pencarian yang sering digunakan untuk mencari *cost* terkecil dari *start node* ke *goal node*. Algoritma ini merupakan variasi dari algoritma Dijkstra dan sangat sesuai untuk kasus dimana setiap simpul mempunyai *cost* yang berbeda-beda. UCS menggunakan *priority queue* untuk menyimpan simpul, sehingga simpul dengan *cost* terendah akan ditelusuri terlebih dahulu. Terminasi pencarian akan terjadi ketika *goal node* menjadi simpul yang ditelusuri. Dengan itu, dipastikan bahwa algoritma UCS akan menghasilkan jalan ke *goal node* yang terendah.

Berikut adalah langkah-langkah algoritma UCS.

1. Memulai dari *start node*.
2. Memasukkan node ke dalam *priority queue* dengan *cost* 0.
3. *Dequeue* simpul dengan *cost* terkecil dari *queue*.
4. Jika simpul adalah *goal node*, hentikan pencarian dan *return* jalur yang telah dibangun serta total *cost*-nya.
5. Jika simpul belum pernah dikunjungi atau ditemukan simpul tersebut dengan biaya yang lebih murah, telusuri simpul tersebut dan catat semua tetangganya.
6. Hitung biaya total dari *start node* ke simpul tetangga tersebut
7. Masukkan simpul tetangga itu ke dalam *priority queue*.
8. Ulangi langkah ke-3 selama *queue* tidak kosong.
9. Proses berhenti ketika *goal node* diraih atau semua simpul telah diperiksa sehingga *priority queue* kosong. Jika ditemukan bahwa *priority queue* kosong, maka dinyatakan bahwa tidak terdapat solusi pencarian dari *start node* ke *goal node*.

2.2 Algoritma Greedy Best First Search

Algoritma *Greedy Best First Search* adalah algoritma pencarian yang mencari jalur dari *start node* ke *goal node* dengan memprioritaskan simpul yang terlihat lebih

menjanjikan, tidak melihat apakah simpul memiliki *cost* terpendek untuk ditelusuri seperti pada UCS. Algoritma ini bekerja dengan mengevaluasi *cost* dari setiap kemungkinan simpul selanjutnya dan menelusuri simpul dengan *cost* terendah. *Greedy Best First Search* menggunakan fungsi heuristik untuk melakukan evaluasi dan menentukan simpul yang paling menjanjikan. Evaluasi ini dinyatakan dalam fungsi $h(n)$ yang menghitung perkiraan *cost* dari jalur sekarang menuju *goal node*. Dengan demikian, jalur selanjutnya yang akan dipilih ditentukan berdasarkan nilai $h(n)$ paling kecil dari *node* yang belum ditelusuri, dan proses ini diulang sampai mencapai *goal node*. Pendekatan yang terus terang dari algoritma ini menjadikannya tergolong mudah untuk diimplementasikan, tetapi keberhasilan dalam menghasilkan jalur pencarian dengan *cost* paling terkecil hanya dapat diraih jika fungsi bersifat *admissible*, yaitu nilai heuristik $h(n)$ yang dihitung tidak lebih besar dari *cost* sebenarnya simpul tersebut ke *goal node*.

Berikut adalah langkah-langkah dari algoritma *Greedy Best First Search*.

1. Memulai dari *start node*.
2. Memasukkan *start node* ke *priority queue* dengan nilai $h(start)$ yang merupakan nilai heuristik dari simpul awal ke simpul tujuan.
3. Mengambil simpul dengan nilai heuristik terkecil.
4. Jika simpul adalah tujuan, hentikan pencarian dan *return* jalur.
5. Jika simpul belum pernah dikunjungi, telusuri semua tetangga dari simpul tersebut.
6. Menghitung nilai heuristik masing-masing tetangga.
7. Masukkan ke dalam *queue* berdasarkan nilai heuristiknya.
8. Ulangi langkah ke-3 selama *queue* tidak kosong.
9. Proses berhenti ketika *goal node* diraih atau semua simpul telah diperiksa sehingga *priority queue* kosong. Jika ditemukan bahwa *priority queue* kosong, maka dinyatakan bahwa tidak terdapat solusi pencarian dari *start node* ke *goal node*.

2.3 Algoritma A*

Algoritma A* adalah algoritma pencarian untuk menemukan jalur terpendek antara titik awal dan akhir dengan menggunakan kombinasi biaya sekarang dan perkiraan biaya ke depannya. Algoritma ini menggunakan fungsi $f(n) = g(n) + h(n)$. $g(n)$

adalah biaya dari awal ke simpul ke- n dan $h(n)$ adalah heuristik estimasi biaya dari simpul ke- n ke tujuan.

Berikut adalah langkah-langkah dari algoritma A*.

1. Memasukkan simpul awal ke *priority queue* dengan nilai $f(start) = g(start) + h(start)$.
2. Mengambil simpul dengan nilai $f(n)$ paling kecil.
3. Jika simpul adalah tujuan, hentikan pencarian dan *return* jalur serta biaya.
4. Jika simpul belum pernah ditelusuri atau ditemukan simpul tersebut dengan biaya lebih murah, telusuri tetangganya.
5. Menghitung $g(n)$ dan $h(n)$ untuk setiap tetangga.
6. Menambang simpul tetangga ke *queue* berdasarkan nilai $f(n)$.
7. Ulangi langkah ke-2 selama *queue* tidak kosong.
8. Proses berhenti ketika *goal node* diraih atau semua simpul telah diperiksa sehingga *priority queue* kosong. Jika ditemukan bahwa *priority queue* kosong, maka dinyatakan bahwa tidak terdapat solusi pencarian dari *start node* ke *goal node*.

BAB III

ANALISIS ALGORITMA PATHFINDING

3.1 Definisi $f(n)$ dan $g(n)$

Algoritma *pathfinding* menghitung *cost* sebagai bahan pertimbangan penelusuran simpul yang dievaluasi menggunakan fungsi $f(n)$. Fungsi $f(n)$ menyatakan fungsi evaluasi yang digunakan untuk menentukan urutan pembangkitan simpul. Nilai $f(n)$ yang lebih kecil akan memberikan prioritas lebih tinggi dalam *priority queue*, sehingga simpul tersebut akan lebih dahulu ditelusuri. Nilai dari $f(n)$ ditentukan berbeda-beda tergantung dengan algoritma yang digunakan, dengan setiap algoritma (UCS, *Greedy Best First Search*, dan A^*) memberikan definisi yang berbeda untuk $f(n)$.

UCS (*Uniform Cost Search*) termasuk kedalam *uninformed search*, yang mengelompokkan algoritma-algoritma yang tidak membutuhkan informasi tambahan dalam melakukan penelusuran. Pada UCS, $f(n)$ didefinisikan dengan $g(n)$, yaitu *cost* langkah dari *start node* menuju simpul yang sedang ditelusuri saat ini. Hal ini berbeda dengan *Greedy Best First Search* yang menggunakan heuristik dalam mendefinisikan $f(n)$. *Greedy Best First Search* melakukan estimasi *cost* dari simpul yang sedang ditelusuri saat ini menuju *goal node*. Karena informasi tersebut didapatkan melalui evaluasi terhadap simpul saat ini, *Greedy Best First Search* termasuk ke dalam *informed search*, yang mengelompokkan algoritma-algoritma yang membutuhkan informasi tambahan dalam melakukan penelusuran. Dengan demikian, *Greedy Best First Search* mendefinisikan dengan $h(n)$, yaitu *cost* simpul menuju *goal node* yang didapatkan melalui pemahaman heuristik.

Algoritma A^* merupakan algoritma yang mempersatukan kedua algoritma sebelumnya dengan mendefinisikan $f(n)$ sebagai jumlah dari $g(n)$ dan $h(n)$. Algoritma ini memperhitungkan *cost* langkah dari *start node* menuju simpul yang sedang ditelusuri saat ini dan estimasi *cost* dari simpul yang sedang ditelusuri saat ini menuju *goal node*, memberikan perhitungan lebih dalam untuk menentukan simpul yang ditelusuri.

Pada tugas kecil kali ini, $g(n)$ menyatakan jumlah langkah yang diambil dari kondisi *board* awal menuju simpul kondisi *board* yang sedang ditelusuri, dengan langkah

menyatakan pergerakan dari salah satu *piece*, seberapa jauh pergerakannya. Karena setiap simpul kondisi *board* merupakan hasil pergerakan salah satu *piece* dari kondisi *board* sebelumnya, maka nilai $g(n)$ akan terus diperbarui ketika simpul baru diproses. Lalu, nilai $h(n)$ menyatakan estimasi *cost* untuk mencapai *goal node* dari simpul yang sedang diproses ini, dan nilainya didapatkan berdasarkan suatu fungsi heuristik yang telah ditentukan sebelumnya.

3.2 Nilai Heuristik pada Algoritma A*

Algoritma A* bergantung pada fungsi heuristik untuk menentukan simpul selanjutnya yang akan ditelusuri. Fungsi heuristik yang digunakan harus bersifat *admissible*, yaitu untuk semua simpul n , berlaku $h(n) \leq h^*(n)$, dengan $h^*(n)$ adalah *cost* sebenarnya dari simpul n ke *goal node*. Oleh karena itu, heuristik yang digunakan tidak boleh melebih-lebihkan estimasi jarak dari simpul n ke *goal node*. Heuristik yang *admissible* menjadi dasar bagi algoritma A* untuk menemukan solusi optimal karena algoritma tidak akan mengabaikan jalur yang secara nyata lebih baik untuk dilewati (memiliki *cost* aktual lebih rendah dibandingkan evaluasi heuristik).

Heuristik utama yang digunakan pada tugas kecil ini adalah Manhattan Distance. Heuristik ini akan menghitung jarak *cell* dari *primary piece* ke pintu keluar, tidak menghitung halangan yang menghalangi *primary piece* untuk bergerak ke pintu keluar. Heuristik ini memenuhi sifat *admissible* karena setiap langkah yang diperlukan untuk memindahkan *primary piece* menuju pintu keluar setidaknya akan mengurangi Manhattan Distance sebesar satu nilai. Halangan *piece* lain yang menghalangi *primary piece* hanya akan menambah jumlah langkah yang sebenarnya dan tidak mungkin menguranginya. Oleh karena itu, Manhattan Distance selalu memberikan perkiraan biaya terendah atau paling tidak sama dengan biaya sebenarnya yang dibutuhkan, tidak melebih-lebihkan *cost* dan menjadikannya baik sebagai heuristik untuk algoritma A* (yang digunakan juga dalam algoritma *Greedy Best First Search*).

Heuristik kedua yang digunakan pada tugas kecil ini adalah menghitung jumlah *piece* yang menghalangi blok P (*primary piece*) ke pintu keluar. Heuristik ini cukup *admissible* dikarenakan jumlah blok yang menghalangi tidak dilebih-lebihkan dan sesuai

dengan kondisi pada papan. Jumlah piece yang menghalangi menandakan minimal blok yang harus dipindahkan untuk *primary piece* bisa menuju pintu keluar. Begitu juga jika memilih untuk menggabungkan kedua heuristik. Gabungan kedua heuristik tetap *admissible* karena kedua heuristik memiliki sifat *admissible* dan menggabungkan kedua tidak menggandakan atau melebih-lebihkan nilai heuristik.

3.3 Perbandingan Algoritma UCS dan BFS

Dalam konteks algoritma *pathfinding* untuk penyelesaian Rush Hour, algoritma UCS dan BFS dapat dikatakan dilakukan dengan tahapan yang sama. Hal ini karena graf yang dibentuk sebagai representasi kondisi *board* yang mungkin bukan merupakan *weighted graph*, sehingga diperlukan penginisialisasian nilai sebagai dasar *cost* untuk pemilihan simpul yang ditelusuri oleh algoritma UCS. Seperti yang telah dijelaskan sebelumnya, tugas kecil ini menggunakan jumlah langkah yang diambil, sehingga untuk setiap simpul kondisi *board*, simpul kondisi *board* yang dibangkitkan berikutnya akan memiliki nilai $cost + 1$ dari simpul kondisi *board* awal untuk setiap simpul kondisi *board* yang dibangkitkan. Karena program dimulai dari satu simpul sebagai *start node*, maka simpul kondisi *board* di level yang sama akan memiliki nilai *cost* yang sama, sehingga UCS akan melakukan penelusuran dalam satu level terlebih dahulu sebelum ke level berikutnya, sama seperti algoritma BFS.

UCS akan menghasilkan perbedaan proses penelusuran dengan algoritma BFS untuk graf dengan tipe *weighted graph*. Maka, untuk algoritma *pathfinding* penyelesaian Rush Hour yang menghasilkan graf dengan tipe *unweighted graph*, proses penelusurannya akan sama dengan graf BFS.

3.4 Perbandingan Algoritma A* dan UCS

Secara teoritis, algoritma A* cenderung lebih efisien dibandingkan UCS sebagai pengaruh dari penggunaan fungsi heuristik sebagai faktor evaluasi *cost* dalam melangsungkan penelusuran simpul yang memperkirakan *cost* tersisa menuju *goal node*. Penggunaan fungsi heuristik membantu untuk mengurangi ruang pencarian dari algoritma A* untuk hanya mempertimbangkan simpul kondisi *board* yang memiliki kemungkinan untuk menghasilkan solusi dengan *cost* terkecil. Hal ini berbeda dengan algoritma UCS yang hanya mempertimbangkan *cost* kumulatif terhadap *start node*. Dengan demikian,

semua *node*, termasuk yang memberikan jalur menuju *goal node* dengan *cost* lebih besar dibandingkan yang telah ditelusuri, akan tetap dipertimbangkan dan mengurangi efisiensi dari algoritma *path finding*.

Maka, dapat disimpulkan bahwa algoritma A* lebih berfokus dalam mencari solusi dengan *cost* terendah, dibandingkan dengan algoritma UCS yang lebih berfokus dalam melakukan penelusuran berdasarkan jarak ke *start node*, menjadikan proses eksplorasi lebih luas dan kurang terarah. Algoritma A* yang didasarkan terhadap informasi heuristik yang tepat akan secara efektif memangkas ruang pencarian dan mempercepat proses menemukan jalur terpendek dibandingkan algoritma UCS.

3.5 Analisis Solusi *Greedy Best First Search*

Algoritma *Greedy Best First Search* tidak dapat menjamin ditemukannya solusi optimal untuk penyelesaian Rush Hour karena algoritma ini hanya mengandalkan evaluasi fungsi heuristik sebagai dasar pemilihan simpul kondisi *board* yang akan ditelusuri, memprioritaskan pencarian bagi simpul kondisi *board* dengan yang dianggap paling dekat dengan *goal node*. Namun, karena hanya mengandalkan *cost* yang didapatkan secara heuristik, meskipun heuristik yang digunakan bersifat *admissible*, *Greedy Best First Search* dapat melewati jalur yang secara kumulatif memiliki *cost* lebih rendah, tetapi tidak memiliki *cost* heuristik yang menjanjikan.

Walaupun algoritma *pathfinding* akan selalu memberikan hasil (asal tersedia), *Greedy Best First Search* rentan untuk terjebak dalam lingkup *local minimum* atau *plateau*. *Greedy Best First Search* dapat membawa penelusuran pada keadaan seluruh simpul kondisi *board* memiliki nilai heuristik yang lebih buruk ketika terdapat jalur yang lebih baik secara global. Ketika algoritma berada pada keadaan ini, maka *Greedy Best First Search* akan terus menjelajahi area di sekitar *local minimum* hingga akhirnya mencapai *goal node* dengan jalur yang tidak optimal, atau berhenti di simpul *leaf* yang bukan *goal node*, yang berarti algoritma harus mengambil simpul baru ke dalam ruang pencarian dan memperpanjang waktu pencarian jalur.

Greedy Best First Search yang hanya berfokus pada nilai heuristik tidak mempertimbangkan *cost* aktual yang diperlukan untuk mencapai simpul kondisi *board*

tersebut, sehingga solusi yang dihasilkan oleh *Greedy Best First Search* tidak dapat dijamin optimal.

BAB IV

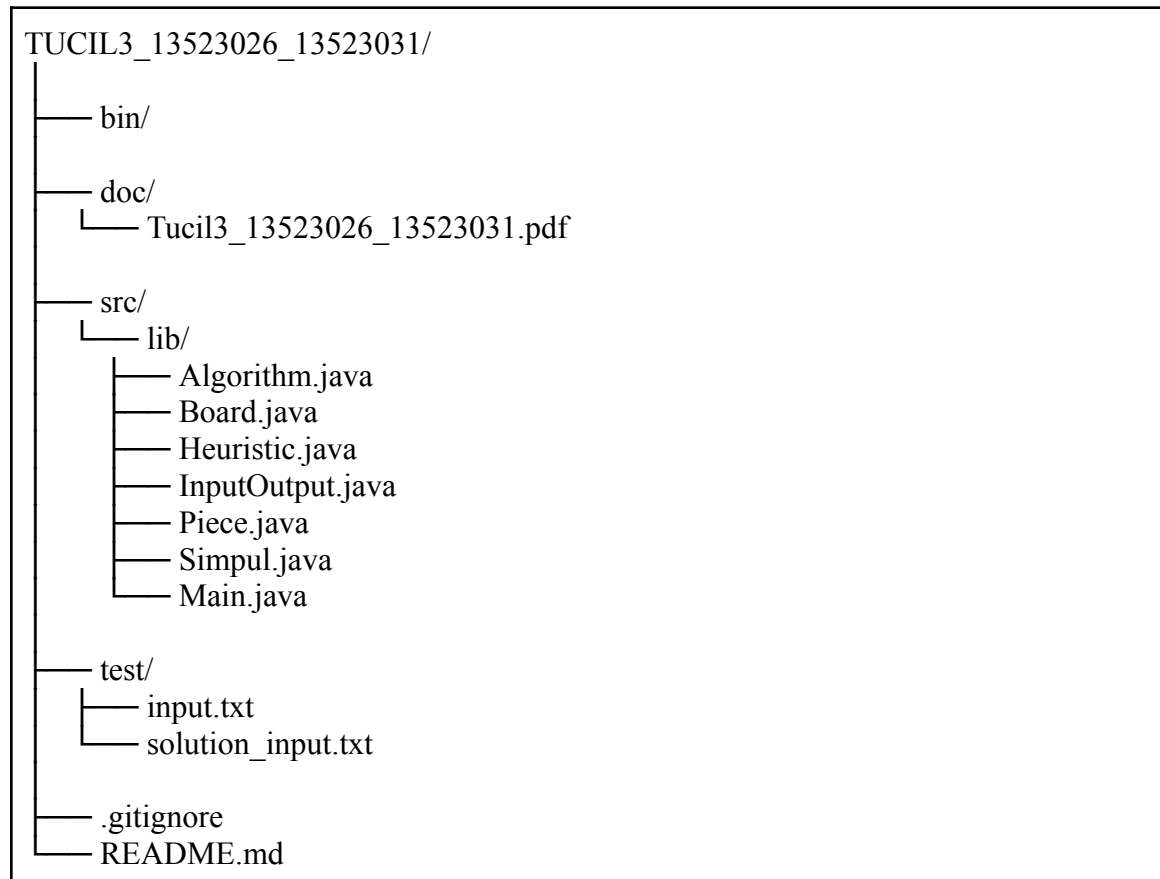
SOURCE CODE PROGRAM

Repositori program dapat diakses melalui pranala berikut:

https://github.com/BerthaSoliany/Tucil3_13523026_13523031.git

4.1 Struktur Folder

Berikut adalah struktur folder yang ada di *repository* GitHub kami.



4.2 Main.java

`Main.java` adalah titik awal program. Program akan dijalankan sesuai dengan isi dari `Main.java`. `Main.java` bertugas untuk menerima masukan dari pengguna dan memanggil fungsi-fungsi, seperti fungsi *solver* dan fungsi untuk *output*.

```
import java.io.File;
import java.io.IOException;
import java.util.Scanner;
```

```
import java.util.List;
import java.util.Map;

// import javax.swing.text.html.StyleSheet;

import lib.*;

public class Main {
    public static void main(String[] args) {
        System.out.println("");
        System.out.println("=====");
        System.out.println("Bertha Soliany Frandi dan Rafen Max Alessandro");
        System.out.println("Dibuat oleh: 13523026 - Bertha Soliany Frandi dan 13523031 - Rafen Max Alessandro");
        program();
    }

    public static void program() {
        try(Scanner sc = new Scanner(System.in)){
            System.out.println("\nMasukkan nama file (ex.txt) atau 'exit' untuk keluar: ");

            String fileName = sc.nextLine();
            File file = new File("test"+File.separator+fileName);

            if (fileName.equals("exit")){System.out.println("Bye bye~");System.exit(1);}

            while (fileName.isEmpty() || !fileName.contains(".txt") || !file.exists()){
                if (fileName.isEmpty()) {
```

```

        System.out.println("Kosong wak\n");
    } else if (!fileName.contains(".txt")){
        System.out.println("Nama file tidak valid. Pastikan nama
file diakhiri dengan .txt\n");
    } else if (!file.exists()){
        System.out.println("Tidak ada file dengan nama
"+fileName+" di folder test.\n");
    }

    System.out.println("Masukkan nama file (ex.txt) atau 'exit'
untuk keluar: ");

    fileName = sc.nextLine();
    file = new File("test"+File.separator+fileName);

    if (fileName.equals("exit")){System.out.println("Bye
bye~");System.exit(1);}
    }

    try{
        Board b = new Board();
        InputOutput.readFile(fileName, b);

        System.out.println("Berikut adalah board yang dimasukkan:
");

        b.displayBoard();

        System.out.println("Sekarang, pilih algoritma penyelesaian
yang akan digunakan: ");
        int pilihan=0;
        String pil;
        boolean validInput = false;
        while (!validInput) {
            System.out.println("1. Algoritma UCS");
            System.out.println("2. Algoritma Greedy Best First
Search");

            System.out.println("3. Algoritma A*");
            System.out.println("4. Keluar");
            System.out.println("Pilihan: ");
            pil = sc.nextLine();
            try {
                pilihan = Integer.parseInt(pil);
                if (pilihan >= 1 && pilihan <= 4) {
                    validInput = true;
                } else {
                    System.out.println("Pilihan tidak valid,
pastikan pilihan antara 1-4\n");

```



```

    }
    } catch (NumberFormatException e) {
        System.out.println("Input harus berupa angka.\n");
    }
}

int heuristic = 0;
switch (pilihan) {
    case 1:
        System.out.println("\nPilihan pertama dipilih:
menggunakan algoritma UCS");
        break;
    case 2:
        System.out.println("\nPilihan kedua dipilih:
menggunakan algoritma Greedy Best First Search");
        System.out.println("Pilih heuristik yang digunakan:
");
        System.out.println("1. Manhattan Distance ke pintu
keluar");
        System.out.println("2. Jumlah piece yang
menghalangi");
        System.out.println("3. Keduanya");
        System.out.println("4. Kembali ke menu utama");
        System.out.println("Pilihan: ");
        boolean validInput2 = false;
        while (!validInput2) {
            String choice = sc.nextLine();
            try {
                heuristic = Integer.parseInt(choice);
                if (heuristic >= 1 && heuristic <= 4) {
                    validInput2 = true;
                } else {
                    System.out.println("Pilihan tidak valid,
pastikan pilihan antara 1-4\n");
                }
            } catch (NumberFormatException e) {
                System.out.println("Input harus berupa
angka.\n");
            }
        }
        if (heuristic == 4) {
            program();
        } else {
            System.out.print("\nHeuristik yang dipilih: ");
            if (heuristic == 1) {
                System.out.println("Manhattan Distance ke
pintu keluar");
            }
        }
    }
}

```

```

        } else if (heuristic == 2) {
            System.out.println("Jumlah piece yang
menghalangi");

        } else if (heuristic == 3) {
            System.out.println("Keduanya");
        }
    }
    break;
case 3:
    System.out.println("\nPilihan ketiga dipilih:
menggunakan algoritma A*");
    System.out.println("Pilih heuristic yang digunakan:
");
    System.out.println("1. Manhattan Distance ke pintu
keluar");
    System.out.println("2. Jumlah piece yang
menghalangi");

    System.out.println("3. Keduanya");
    System.out.println("4. Kembali ke menu utama");
    System.out.println("Pilihan: ");
    boolean validInput3 = false;
    while (!validInput3) {
        String choice = sc.nextLine();
        try {
            heuristic = Integer.parseInt(choice);
            if (heuristic >= 1 && heuristic <= 4) {
                validInput3 = true;
            } else {
                System.out.println("Pilihan tidak valid,
pastikan pilihan antara 1-4\n");
            }
        } catch (NumberFormatException e) {
            System.out.println("Input harus berupa
angka.\n");
        }
    }
    if (heuristic == 4) {
        program();
    } else {
        System.out.print("\nHeuristic yang dipilih: ");
        if (heuristic == 1) {
            System.out.println("Manhattan Distance ke
pintu keluar");

        } else if (heuristic == 2) {
            System.out.println("Jumlah piece yang
menghalangi");

        } else if (heuristic == 3) {

```

```

        System.out.println("Keduanya");
    }
}
break;
case 4:
    System.out.println("Bye bye~");
    System.exit(1);
}

long startTime = System.nanoTime();
List<Simpul> output = Algorithm.search(b, pilihan,
heuristic);
Map<Simpul, Simpul> parent = Algorithm.getParent();
long endTime = System.nanoTime();
long duration = ((endTime-startTime)/1_000_000);
InputOutput.printOutput(output, duration, parent);
InputOutput.writeFile(fileName, b, output, duration,
pilihan, heuristic);
System.out.println("Berhasil menyimpan output ke dalam file
test\\solution_"+fileName+"\n");

    } catch (IOException e) {
        program();
    }
}
}
}
// javac -d bin -sourcepath src src/Main.java
// java -cp bin Main

```

4.3 Algorithm.java

Algorithm.java adalah *file* yang berisi kelas Algorithm. Kelas Algorithm berguna untuk menampung semua algoritma *pathfinding*. Fungsi dan/atau prosedur yang ada di kelas ini, dipanggil di Main.java sesuai dengan masukan pengguna.

```

package lib;

import java.util.*;

public class Algorithm {

    private static Map<Simpul, Simpul> parent = new HashMap<>();
    private static int visitedCount = 0;

```

```

    public static Map<Simpul, Simpul> getParent() {
        return parent;
    }

    public static int getVisitedCount() {
        return visitedCount;
    }

    public static List<Simpul> search(Board board, int algorithm, int
    heuristic) {
        PriorityQueue<Simpul> queue = new
    PriorityQueue<>(Comparator.comparingInt(Simpul::getCost));
        Set<Board> visited = new HashSet<>();
        Map<Board, Integer> costMap = new HashMap<>();

        Simpul initialSimpul = new Simpul(board, '-', 0, 0);
        queue.add(initialSimpul);
        visited.add(board);
        costMap.put(board, 0);

        while (!queue.isEmpty()) {
            Simpul currentSimpul = queue.poll();
            Board currentBoard = currentSimpul.getBoard();

            // Check if the exit is reached
            if (currentBoard.isWin()) {

                // Add the exit state to the result
                return reconstructPath(parent, currentSimpul);
            }
            visited.add(currentBoard);

            // Generate next states
            List<Simpul> neighbors = currentBoard.getNeighbors();

            for (Simpul neighbor : neighbors) {
                visitedCount++;
                Board neighborBoard = neighbor.getBoard();

                // kalau board sudah pernah diperiksa, maka cost saat ini
                pasti lebih besar, bisa langsung di skip
                if (!visited.contains(neighborBoard)) {
                    int newCost = costMap.get(currentBoard) + 1;
                    newCost = getHeuristicCost(neighborBoard, algorithm,
                    newCost, heuristic);

                    // !costMap.containsKey(neighborBoard) hanya guard untuk

```

```

memastikan costMap.get(neighborBoard) tidak null
        if (!costMap.containsKey(neighborBoard) || newCost <
costMap.get(neighborBoard)) {
            neighbor.setCost(newCost);

            queue.add(neighbor);
            visited.add(neighborBoard);
            costMap.put(neighborBoard, newCost);
            parent.put(neighbor, currentSimpul);
        }
    }
}

// tidak ditemukan solusi
return Collections.emptyList();
}

private static List<Simpul> reconstructPath(Map<Simpul, Simpul> parent,
Simpul exit) {
    LinkedList<Simpul> path = new LinkedList<Simpul>();
    while (exit != null) {
        path.addFirst(exit);
        exit = parent.get(exit);
    }
    return path;
}

private static int getHeuristicCost(Board board, int algorithm, int
UCSCost, int heuristic) {
    switch (algorithm) {
        case 1: // UCS
            return UCSCost;
        case 2: // greedy
            return Heuristic.getChoosenHeuristic(board, heuristic);
        case 3: // A*
            return UCSCost + Heuristic.getChoosenHeuristic(board,
heuristic);
        default:
            return 0;
    }
}
}

```

4.4 Board.java

Board.java adalah *file* yang berisi kelas Board. Kelas ini berguna sebagai representasi papan permainan yang memiliki berbagai *piece* termasuk *primary piece* dan pintu keluar yang dituju oleh *primary piece*.

```
package lib;

import java.util.Arrays;
import java.util.List;
import java.util.ArrayList;

public class Board {
    private int width; // lebar papan
    private int height; // tinggi papan
    private int totalPieces; // total pieces selain primary piece
    private int[] exitLoc; // pintu keluar
    private char[][] board; // untuk memetakan pieces (mainly untuk
    // kebutuhan output dan cek apakah keluar dr board atau enggak)
    private Piece[] pieces; //

    public Board(){}

    // copy constructor
    public Board(Board b){
        this.width = b.width;
        this.height = b.height;
        this.totalPieces = b.totalPieces;
        this.exitLoc = b.exitLoc.clone();
        this.board = new char[height][width];
        for (int i = 0; i < height; i++) {
            this.board[i] = b.board[i].clone();
        }
        this.pieces = new Piece[b.pieces.length];
        for (int i = 0; i < b.pieces.length; i++) {
            this.pieces[i] = new Piece(b.pieces[i]);
        }
    }

    public int getWidth(){return width;}
    public int getHeight(){return height;}
    public int getTotalPieces(){return totalPieces;}
    public int[] getExitLoc(){return exitLoc;}
    public char[][] getBoard(){return board;}
    public Piece[] getPieces(){return pieces;}
    public Piece getPiece(char id){
```

```

        for (int i = 0; i < pieces.length; i++) {
            if (pieces[i].getId() == id) {
                return pieces[i];
            }
        }
        return null;
    }

    public void setWidth(int w){width=w;}
    public void setHeight(int h){height=h;}
    public void setTotalPieces(int p){totalPieces=p;}
    public void setExitLoc(int[] loc){exitLoc=loc;}
    public void setBoard(char[][] b){board=b;}
    public void setPieces(Piece[] p){pieces=p;}

    public void printBoard(){
        System.out.println(Arrays.deepToString(board));
    }

    public void displayBoard() {
        String reset = "\u001B[0m";
        String yellow = "\u001B[33m";
        String green = "\u001B[32m";

        if (exitLoc[0]==-1){
            for (int i=0;i<width;i++){
                if (exitLoc[1]==i){
                    System.out.print(green+"K"+reset);
                } else {
                    System.out.print(" ");
                }
            }
            System.out.println();
            for (int i = 0; i < height; i++) {
                for (int j = 0; j < width; j++) {
                    if (board[i][j] == 'P') {
                        System.out.print(yellow + board[i][j] + reset);
                    } else {
                        System.out.print(board[i][j]);
                    }
                }
                System.out.println();
            }
        } else if (exitLoc[0]==height){
            for (int i = 0; i < height; i++) {
                for (int j = 0; j < width; j++) {
                    if (board[i][j] == 'P') {

```

```

        System.out.print(yellow + board[i][j] + reset);
    } else {
        System.out.print(board[i][j]);
    }
}
System.out.println();
}
for (int i=0;i<width;i++){
    if (exitLoc[1]==i){
        System.out.print(green+"K"+reset);
    } else {
        System.out.print(" ");
    }
}
System.out.println();
} else if (exitLoc[1] == -1) {
    for (int i = 0; i < height; i++) {
        if (i == exitLoc[0]) {
            System.out.print(green+"K"+reset);
        } else {
            System.out.print(" ");
        }
        for (int j = 0; j < width; j++) {
            if (board[i][j] == 'P') {
                System.out.print(yellow + board[i][j] + reset);
            } else {
                System.out.print(board[i][j]);
            }
        }
        System.out.println();
    }
} else if (exitLoc[1] == width) {
    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            if (board[i][j] == 'P') {
                System.out.print(yellow + board[i][j] + reset);
            } else {
                System.out.print(board[i][j]);
            }
        }
        if (i == exitLoc[0]) {
            System.out.print(green+"K"+reset);
        } else {
            System.out.print(" ");
        }
        System.out.println();
    }
}

```



```

    }
    System.out.println();
}

public void displayBoardForOutput(char id, Board b) {
    String reset = "\u001B[0m";
    String blueBg = "\u001B[44m";
    String yellow = "\u001B[33m";
    String green = "\u001B[32m";

    if (exitLoc[0]==-1){
        for (int i=0;i<width;i++){
            if (exitLoc[1]==i){
                System.out.print(green+"K"+reset);
            } else {
                System.out.print(" ");
            }
        }
        System.out.println();
        for (int i = 0; i < height; i++) {
            for (int j = 0; j < width; j++) {
                if (board[i][j] == id) {
                    if (id == 'P') {
                        System.out.print(blueBg + yellow + board[i][j] +
reset);

                    } else {
                        System.out.print(blueBg + board[i][j] + reset);
                    }
                } else if (board[i][j] == 'P') {
                    System.out.print(yellow + board[i][j] + reset);
                } else {
                    if (b.getBoard()[i][j] == id) {
                        System.out.print(blueBg + board[i][j] + reset);
                    } else {
                        System.out.print(board[i][j]);
                    }
                }
            }
        }
        System.out.println();
    }
} else if (exitLoc[0]==height){
    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            if (board[i][j] == id) {
                if (id == 'P') {
                    System.out.print(blueBg + yellow + board[i][j] +
reset);

```

```

        } else {
            System.out.print(blueBg + board[i][j] + reset);
        }
    } else if (board[i][j] == 'P') {
        System.out.print(yellow + board[i][j] + reset);
    } else {
        if (b.getBoard()[i][j] == id) {
            System.out.print(blueBg + board[i][j] + reset);
        } else {
            System.out.print(board[i][j]);
        }
    }
}
System.out.println();
}
for (int i=0;i<width;i++){
    if (exitLoc[1]==i){
        System.out.print(green+"K"+reset);
    } else {
        System.out.print(" ");
    }
}
System.out.println();
} else if (exitLoc[1] == -1) {
    for (int i = 0; i < height; i++) {
        if (i == exitLoc[0]) {
            System.out.print(green+"K"+reset);
        } else {
            System.out.print(" ");
        }
        for (int j = 0; j < width; j++) {
            if (board[i][j] == id) {
                if (id == 'P') {
                    System.out.print(blueBg + yellow + board[i][j] +
reset);

                } else {
                    System.out.print(blueBg + board[i][j] + reset);
                }
            } else if (board[i][j] == 'P') {
                System.out.print(yellow + board[i][j] + reset);
            } else {
                if (b.getBoard()[i][j] == id) {
                    System.out.print(blueBg + board[i][j] + reset);
                } else {
                    System.out.print(board[i][j]);
                }
            }
        }
    }
}

```

```

        }
        System.out.println();
    }
} else if (exitLoc[1] == width) {
    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            if (board[i][j] == id) {
                if (id == 'P') {
                    System.out.print(blueBg + yellow + board[i][j] +
reset);

                } else {
                    System.out.print(blueBg + board[i][j] + reset);
                }
            } else if (board[i][j] == 'P') {
                System.out.print(yellow + board[i][j] + reset);
            } else {
                if (b.getBoard()[i][j] == id) {
                    System.out.print(blueBg + board[i][j] + reset);
                } else {
                    System.out.print(board[i][j]);
                }
            }
        }
    }
    if (i == exitLoc[0]) {
        System.out.print(green+"K"+reset);
    } else {
        System.out.print(" ");
    }
    System.out.println();
}
}
System.out.println();
}

public void displayPiece(){
    Piece[] pieces = getPieces();
    if (pieces == null || pieces.length == 0) {
        System.out.println("Board tidak mempunyai piece.");
        return;
    }

    for (Piece p : pieces) {
        System.out.println("ID piece: " + p.getId());
        System.out.println("Dimensi piece: " + p.getHeight() + " x " +
p.getWidth());
        System.out.println("Lokasi:");
        for (int[] loc : p.getLocation()) {

```

```

        System.out.println(" (" + loc[0] + ", " + loc[1] + ")");
    }
    System.out.println();
}

}

public boolean isWin(){
    if (exitLoc[0]==-1){
        if (board[0][exitLoc[1]] == 'P'){
            return true;
        }
    } else if (exitLoc[0]==height){
        if (board[height-1][exitLoc[1]] == 'P'){
            return true;
        }
    } else if (exitLoc[1] == -1) {
        if (board[exitLoc[0]][0] == 'P'){
            return true;
        }
    } else if (exitLoc[1] == width) {
        if (board[exitLoc[0]][width-1] == 'P'){
            return true;
        }
    }
    return false;
}

public Piece removePiece(char id){
    // remove piece dari board
    Piece removedPiece = getPiece(id);
    for (int i = 0; i < removedPiece.getLocation().length; i++) {
        int x = removedPiece.getLocation()[i][0];
        int y = removedPiece.getLocation()[i][1];
        board[x][y] = '.';
    }

    return removedPiece;
}

public void addPiece(Piece piece){
    // add piece ke board
    char id = piece.getId();
    for (int i = 0; i < piece.getLocation().length; i++) {
        int x = piece.getLocation()[i][0];
        int y = piece.getLocation()[i][1];
        board[x][y] = id;
    }
}

```

```

        for (int i = 0; i < pieces.length; i++) {
            if (pieces[i].getId() == piece.getId()) {
                pieces[i] = piece;
                break;
            }
        }
    }

    public List<Integer> checkPossibleMove(char id){
        // cek piece bisa bergerak kemana
        Piece piece = getPiece(id);
        int i = piece.getLocation()[0][0];
        int j = piece.getLocation()[0][1];
        int length = piece.getLength();
        char orientation = piece.getOrientation();
        List<Integer> possibleMoves = new ArrayList<>();
        int move;

        if (orientation == 'h') {
            // check left
            move = -1;
            while (j + move >= 0 && board[i][j + move] == '.') {
                possibleMoves.add(move);
                move--;
            }
            // check right
            move = 1;
            while (j + length - 1 + move < width && board[i][j + length - 1
+ move] == '.') {
                possibleMoves.add(move);
                move++;
            }
        } else {
            // check up
            move = -1;
            while (i + move >= 0 && board[i + move][j] == '.') {
                possibleMoves.add(move);
                move--;
            }
            // check down
            move = 1;
            while (i + length - 1 + move < height && board[i + length - 1 +
move][j] == '.') {
                possibleMoves.add(move);
                move++;
            }
        }
    }

```

```

    }

    return possibleMoves;
}

public List<Simpul> getNeighbors(){
    List<Simpul> neighbors = new ArrayList<>();
    for (Piece piece : pieces) {
        List<Integer> possibleMoves = checkPossibleMove(piece.getId());
        List<Piece> possiblePieces =
piece.possiblePieceMoves(possibleMoves);
        for (Piece newPiece : possiblePieces) {
            Board newBoard = new Board(this);

            newBoard.removePiece(piece.getId());
            newBoard.addPiece(newPiece);

            int moveValue = 0;
            if (newPiece.getOrientation() == 'h') {
                moveValue = newPiece.getLocation()[0][1] -
piece.getLocation()[0][1];
            } else {
                moveValue = newPiece.getLocation()[0][0] -
piece.getLocation()[0][0];
            }

            neighbors.add(new Simpul(newBoard, newPiece.getId(),
moveValue, -1));
        }
    }
    return neighbors;
}

@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;
    Board other = (Board) o;

    // Compare pieces' positions
    if (!Arrays.equals(this.pieces, other.pieces)) return false;

    return true;
}

@Override
public int hashCode() {

```

```

        return Arrays.hashCode(pieces);
    }
}

```

Heuristic.java

Heurittic.java adalah file yang bertugas untuk menampung kode-kode untuk menghitung heuristik yang bisa digunakan untuk algoritma-algoritma yang diimplementasikan.

```

package lib;

public class Heuristic {
    // menghitung jarak blok ke pintu keluar
    public static int manhattanDistance(Board b, char id){
        Piece pieceP = b.getPiece(id);
        int[][] locs = pieceP.getLocation();
        int minDistance=Integer.MAX_VALUE;

        int exitX = b.getExitLoc()[1];
        int exitY = b.getExitLoc()[0];
        if (exitY == -1) {
            exitY = 0;
        } else if (exitY == b.getHeight()) {
            exitY = b.getHeight() - 1;
        }

        if (

```

4.5 Heuristic.java

Heurittic.java adalah *file* yang bertugas untuk menampung kode-kode untuk menghitung heuristik yang bisa digunakan untuk algoritma-algoritma yang diimplementasikan.

```

package lib;

import java.util.HashSet;
import java.util.Set;

public class Heuristic {
    // menghitung jarak blok ke pintu keluar
    public static int manhattanDistance(Board b, char id){
        Piece pieceP = b.getPiece(id);
        int[][] locs = pieceP.getLocation();
        int minDistance=Integer.MAX_VALUE;

```

```

        int exitX = b.getExitLoc()[1];
        int exitY = b.getExitLoc()[0];
        if (exitY == -1) {
            exitY = 0;
        } else if (exitY == b.getHeight()) {
            exitY = b.getHeight() - 1;
        }

        if (exitX == -1) {
            exitX = 0;
        } else if (exitX == b.getWidth()) {
            exitX = b.getWidth() - 1;
        }

        for(int[] loc : locs){
            int x = loc[1];
            int y = loc[0];
            int distance = Math.abs(x - exitX) + Math.abs(y - exitY);
            if (distance < minDistance){
                minDistance = distance;
            }
        }
        return minDistance;
    }

    // menghitung jumlah piece yang menghalangi
    public static int blockCount(Board b) {
        Piece p = b.getPiece('P');
        int[][] locs = p.getLocation();
        int exitX = b.getExitLoc()[1];
        int exitY = b.getExitLoc()[0];
        char[][] board = b.getBoard();
        int width = b.getWidth();
        int height = b.getHeight();

        Set<Character> blockers = new HashSet<>();

        boolean isHorizontal = p.getHeight() == 1;

        if (isHorizontal) {
            int y = locs[0][1];

            int minX = Integer.MAX_VALUE, maxX = Integer.MIN_VALUE;
            for (int[] loc : locs) {
                minX = Math.min(minX, loc[0]);
                maxX = Math.max(maxX, loc[0]);
            }

```



```

        if (exitX == width) {
            for (int x = maxX + 1; x < width; x++) {
                char c = board[y][x];
                if (c != '.' && c != 'P') blockers.add(c);
            }
        } else if (exitX == -1) {
            for (int x = minX - 1; x >= 0; x--) {
                char c = board[y][x];
                if (c != '.' && c != 'P') blockers.add(c);
            }
        }
    } else {
        int x = locs[0][0];

        int minY = Integer.MAX_VALUE, maxY = Integer.MIN_VALUE;
        for (int[] loc : locs) {
            minY = Math.min(minY, loc[1]);
            maxY = Math.max(maxY, loc[1]);
        }

        if (exitY == height) {
            for (int y = maxY + 1; y < height; y++) {
                char c = board[y][x];
                if (c != '.' && c != 'P') blockers.add(c);
            }
        } else if (exitY == -1) {
            for (int y = minY - 1; y >= 0; y--) {
                char c = board[y][x];
                if (c != '.' && c != 'P') blockers.add(c);
            }
        }
    }
}

// System.out.println("Jumlah blocker: " + blockers.size());
return blockers.size();
}

public static int getChoosenHeuristic(Board b, int choice) {
    if (choice == 1) {
        return manhattanDistance(b, 'P');
    } else if (choice == 2) {
        return blockCount(b);
    } else if (choice == 3) {
        return manhattanDistance(b, 'P') + blockCount(b);
    } else {
        return 0;
    }
}

```

```
    }  
  }  
}
```

4.6 InputOutput.java

InputOutput.java adalah *file* yang bertugas untuk mengatasi masukan dan keluaran program. Masukan dan keluaran yang dimaksud adalah untuk *Command Line Interface* (CLI) dan juga *file* dengan tipe .txt.

```
package lib;  
  
import java.io.BufferedReader;  
import java.io.FileReader;  
import java.io.FileWriter;  
import java.io.IOException;  
import java.util.*;  
  
public class InputOutput {  
    public static void readFile(String fileName, Board b) throws  
    IOException{  
        try {  
            fileName = "test\\" + fileName;  
            // BARIS X KOLOM  
            int A,B,N;  
  
            BufferedReader br = new BufferedReader(new  
            FileReader(fileName));  
            StringTokenizer st = new StringTokenizer(br.readLine());  
  
            if (st.countTokens() != 2){  
                br.close();  
                throw new IOException("Format ukuran papan tidak valid.");  
            }  
  
            try {  
                A = Integer.parseInt(st.nextToken());  
                B = Integer.parseInt(st.nextToken());  
            } catch (NumberFormatException e) {  
                br.close();  
                throw new IOException("Format salah: A dan B harus berupa  
angka.");  
            }  
            b.setWidth(B);  
        }  
    }  
}
```

```

        b.setHeight(A);

        System.out.println("A:"+A+" B:"+B); // debugging

        if (A<=0||B<=0){
            br.close();
            throw new IOException("Dimensi papan tidak valid.");
        }

        N = Integer.parseInt(br.readLine());
        b.setTotalPieces(N);

        System.out.println("Total piece selain primary piece: "+N+"\n");
// debugging

        char[][] board = new char[A][B];
        int[] exit = {-1,-1};

        List<String> lines = new ArrayList<>();
        String line;
        while ((line=br.readLine())!=null){
            lines.add(line);
        }

        int panjang, lebar=-1; // untuk mengecek kevalidan panjang dan
// lebar papan

        int row = 0;
        for (int i=0;i<lines.size();i++){
            String l = lines.get(i);
            // l = l.replaceAll(" ", "");
            if (l.trim().equals("K")){
                // K di plg atas, exit={-1,j}
                if (i == 0) {
                    exit[0] = -1;
                    exit[1] = l.indexOf('K');
                } else if (i == lines.size() - 1) { // K plg bawah,
// exit={B,j}
                    exit[0] = A;
                    exit[1] = l.indexOf('K');
                }
                panjang = lines.size()-1;
                if (panjang!=A){br.close();throw new IOException("Tinggi
papan "+panjang+" tidak sesuai.");}
                continue;
            }
        }

```

```

        l = l.replaceAll(" ", "");

        panjang = lines.size();
        if (panjang!=A){br.close();throw new IOException("Tinggi
papan "+panjang+" tidak sesuai.");}

        if (l.length()==B+1){

            // K plg kiri, exit={row,-1}
            if (l.charAt(0)=='K'){
                exit[0] = row;
                exit[1] = -1;
                l=lines.get(i).substring(1);
            } else if (l.charAt(l.length()-1)=='K'){ // K plg kanan,
exit={row,B}

                exit[0] = row;
                exit[1] = B;
                l = l.substring(0, B);
            }
            lebar = l.length();
            if (lebar != B){br.close();throw new IOException("Lebar
papan "+lebar+" tidak sesuai.");}
        } else {
            lebar = l.length();
            System.out.println(lebar);

            if (lebar != B){br.close();throw new IOException("Lebar
papan "+lebar+" tidak sesuai.");}
        }

        if (row<A){
            for (char c : l.toCharArray()){
                if ((c<'A' || c>'Z') || c=='K'){
                    l = l.replace(c, '.');
                }
            }
            board[row++] = l.substring(0, B).toCharArray();
        }
    }

    if (exit[0] == -1 && exit[1] == -1) {
        br.close();
        throw new IOException("Pintu keluar (K) tidak ditemukan atau
tidak valid.");
    }

    b.setExitLoc(exit);

```

```

        b.setBoard(board);

        Map<Character, List<int[]>> pieceMap = new HashMap<>();
        for (int i = 0; i < A; i++) {
            for (int j = 0; j < B; j++) {
                char c = board[i][j];
                if (c != '.' && (c >= 'A' || c <= 'Z') && c != 'K') {
                    pieceMap.putIfAbsent(c, new ArrayList<>());
                    pieceMap.get(c).add(new int[]{i, j});
                }
            }
        }

        Piece[] pieces = new Piece[pieceMap.size()];
        int idx = 0;
        for (Map.Entry<Character, List<int[]>> entry :
pieceMap.entrySet()) {
            char id = entry.getKey();
            List<int[]> locs = entry.getValue();

            int minY = Integer.MAX_VALUE, maxY = Integer.MIN_VALUE;
            int minX = Integer.MAX_VALUE, maxX = Integer.MIN_VALUE;
            for (int[] p : locs) {
                minY = Math.min(minY, p[0]);
                maxY = Math.max(maxY, p[0]);
                minX = Math.min(minX, p[1]);
                maxX = Math.max(maxX, p[1]);
            }

            int height = maxY - minY + 1;
            int width = maxX - minX + 1;

            int[][] loc = new int[locs.size()][2];
            for (int i = 0; i < locs.size(); i++) {
                loc[i][0] = locs.get(i)[0];
                loc[i][1] = locs.get(i)[1];
            }

            Piece piece = new Piece(id, loc, width, height);
            pieces[idx++] = piece;
        }

        b.setPieces(pieces);
        br.close();
        if (pieces.length != N+1){
            throw new IOException("Jumlah piece "+pieces.length+" tidak

```

```

sesuai dengan file input.");
    }
} catch (IOException e) {
    System.out.println("Gagal membaca file "+fileName);
    e.printStackTrace();
    throw e;
}

}

    public static void writeFile(String fileName, Board awal, List<Simpul>
output, long timeSpent){
    try{
        fileName = "test\\" + "solution_"+fileName;
        FileWriter save = new FileWriter(fileName);
        save.write("Solusi untuk "+fileName+"\n\n"); // fileName adalah
fileName input
        save.write("Dimensi papan: "+awal.getHeight()+" x
"+awal.getWidth()+"\n");
        save.write("Jumlah piece selain P:
"+awal.getTotalPieces()+"\n");
        save.write("\nWaktu penyelesaian: "+timeSpent+" ns\n");
        save.write("Total gerakan yang dilakukan:
"+output.size()+"\n\n");
        // print ke file untuk papan awal
        save.write("Papan Awal\n");
        if (awal.getExitLoc()[0]==-1){
            for (int i=0;i<awal.getWidth();i++){
                if (awal.getExitLoc()[1]==i){
                    save.write("K");
                } else {
                    save.write(" ");
                }
            }
            save.write("\n");
            for (int i = 0; i < awal.getHeight(); i++) {
                for (int j = 0; j < awal.getWidth(); j++) {
                    save.write(awal.getBoard()[i][j]);
                }
                save.write("\n");
            }
        } else if (awal.getExitLoc()[0]==awal.getHeight()){
            for (int i = 0; i < awal.getHeight(); i++) {
                for (int j = 0; j < awal.getWidth(); j++) {
                    save.write(awal.getBoard()[i][j]);
                }
                save.write("\n");
            }
        }
    }
}

```

```

    }
    for (int i=0;i<awal.getWidth();i++){
        if (awal.getExitLoc()[1]==i){
            save.write("K");
        } else {
            save.write(" ");
        }
    }
    save.write("\n");
} else if (awal.getExitLoc()[1] == -1) {
    for (int i = 0; i < awal.getHeight(); i++) {
        if (i == awal.getExitLoc()[0]) {
            save.write("K");
        } else {
            save.write(" ");
        }
        for (int j = 0; j < awal.getWidth(); j++) {
            save.write(awal.getBoard()[i][j]);
        }
        save.write("\n");
    }
} else if (awal.getExitLoc()[1] == awal.getWidth()) {
    for (int i = 0; i < awal.getHeight(); i++) {
        for (int j = 0; j < awal.getWidth(); j++) {
            save.write(awal.getBoard()[i][j]);
        }
        if (i == awal.getExitLoc()[0]) {
            save.write("K");
        } else {
            save.write(" ");
        }
        save.write("\n");
    }
}

// print langkah2 ke file
if (output.isEmpty()) {
    save.write("Tidak ditemukan solusi... :(");
    save.close();
    return;
}

if (output.get(0).getIdPiece()=='-'){
    save.write("Papan sudah dalam kondisi menang.");
    save.close();
    return;
}

```

```

int counter = 1;
for (Simpul s : output) {
    save.write("\n");
    Piece piece = s.getBoard().getPiece(s.getIdPiece());

    String direction = null;
    if (piece.getOrientation() == 'h') {
        if (s.getMoveValue() > 0) {
            direction = "kanan";
        } else if (s.getMoveValue() < 0) {
            direction = "kiri";
        }
    } else {
        if (s.getMoveValue() > 0) {
            direction = "bawah";
        } else if (s.getMoveValue() < 0) {
            direction = "atas";
        }
    }

    save.write("Gerakan " + counter++ + ": " + s.getIdPiece() +
" sebanyak " + Math.abs(s.getMoveValue()) + " petak ke " + direction);
    save.write("\n");
    if (s.getBoard().getExitLoc()[0]==-1){
        for (int i=0;i<s.getBoard().getWidth();i++){
            if (s.getBoard().getExitLoc()[1]==i){
                save.write("K");
            } else {
                save.write(" ");
            }
        }
        save.write("\n");
        for (int i = 0; i < s.getBoard().getHeight(); i++) {
            for (int j = 0; j < s.getBoard().getWidth(); j++) {
                save.write(s.getBoard().getBoard()[i][j]);
            }
            save.write("\n");
        }
    } else if
(s.getBoard().getExitLoc()[0]==s.getBoard().getHeight()){
        for (int i = 0; i < s.getBoard().getHeight(); i++) {
            for (int j = 0; j < s.getBoard().getWidth(); j++) {
                save.write(s.getBoard().getBoard()[i][j]);
            }
            save.write("\n");
        }
    }
}

```



```

        for (int i=0;i<s.getBoard().getWidth();i++){
            if (s.getBoard().getExitLoc()[1]==i){
                save.write("K");
            } else {
                save.write(" ");
            }
        }
        save.write("\n");
    } else if (s.getBoard().getExitLoc()[1] == -1) {
        for (int i = 0; i < s.getBoard().getHeight(); i++) {
            if (i == s.getBoard().getExitLoc()[0]) {
                save.write("K");
            } else {
                save.write(" ");
            }
            for (int j = 0; j < s.getBoard().getWidth(); j++) {
                save.write(s.getBoard().getBoard()[i][j]);
            }
            save.write("\n");
        }
    } else if (s.getBoard().getExitLoc()[1] ==
s.getBoard().getWidth()) {
        for (int i = 0; i < s.getBoard().getHeight(); i++) {
            for (int j = 0; j < s.getBoard().getWidth(); j++) {
                save.write(s.getBoard().getBoard()[i][j]);
            }
            if (i == s.getBoard().getExitLoc()[0]) {
                save.write("K");
            } else {
                save.write(" ");
            }
            save.write("\n");
        }
    }
    }
    save.close();
} catch (IOException e){
    System.out.println("Terjadi kesalahan saat menulis file.");
    e.printStackTrace();
}
}

public static void printOutput(Board awal, List<Simpul> output) {
    System.out.println();
    System.out.println("Papan awal:");
    awal.displayBoard();
}

```

```

        if (output.isEmpty()) {
            System.out.println("Tidak ditemukan solusi... :(");
            return;
        }

        if (output.get(0).getIdPiece() == '-') {
            System.out.println("Papan sudah dalam kondisi menang.");
            return;
        }

        int counter = 1;
        for (Simpul s : output) {
            Board board = s.getBoard();
            Piece piece = s.getBoard().getPiece(s.getIdPiece());

            String direction = null;
            if (piece.getOrientation() == 'h') {
                if (s.getMoveValue() > 0) {
                    direction = "kanan";
                } else if (s.getMoveValue() < 0) {
                    direction = "kiri";
                }
            } else {
                if (s.getMoveValue() > 0) {
                    direction = "bawah";
                } else if (s.getMoveValue() < 0) {
                    direction = "atas";
                }
            }

            System.out.println("Gerakan " + counter++ + ": " +
                s.getIdPiece() + " sebanyak " + Math.abs(s.getMoveValue()) + " petak ke " +
                direction);
            board.displayBoardForOutput(s.getIdPiece());
        }
    }
}

```

Piece.java

Piece.java adalah file dengan kelas Piece. Kelas Piece adalah representasi dari balok permainan Rush Hour. Setiap balok mempunyai id dan lokasi yang berguna untuk mengidentifikasi balok.

```
package lib;
```

```
import java.util.Objects;
```

```
import java.util.Arrays;
```

```

import java.util.List;
import java.util.ArrayList;

public class Piece {
    private char id;
    private int[][] location; // atribut lokasi piece.
    private int width;
    private int height;

    // constructor
    public Piece(char id, int[][] loc, int w, int h){
        this.id = id;
        this.location = loc;
        this.width = w;
        this.height = h;
    }

    // copy constructor
    public Piece(Piece piece){
        this.id = piece.id;
        this.location = new
int[piece.location.length][piece.location[0].length];
        for (int i = 0; i < piece.location.length; i++) {
            this.location[i] = piece.location[i].clone();
        }
        this.width = piece.width;
        this.height = piece.height;
    }

    // getter setter
    public char getId(){return id;}
    public int[][] getLocation(){return location;}
    public int getWidth(){return width;}
    public int getHeight(){return height;}
    pub

Piece.java
    Piece.java

```

4.7 Piece.java

Piece.java adalah *file* dengan kelas Piece. Kelas Piece adalah representasi dari balok permainan Rush Hour. Setiap balok mempunyai id dan lokasi yang berguna untuk mengidentifikasi balok.

```

package lib;

import java.util.Objects;
import java.util.Arrays;
import java.util.List;
import java.util.ArrayList;

public class Piece {
    private char id;
    private int[][] location; // atribut lokasi piece.
    private int width;
    private int height;

    // constructor
    public Piece(char id, int[][] loc, int w, int h){
        this.id = id;
        this.location = loc;
        this.width = w;
        this.height = h;
    }

    // copy constructor
    public Piece(Piece piece){
        this.id = piece.id;
        this.location = new
int[piece.location.length][piece.location[0].length];
        for (int i = 0; i < piece.location.length; i++) {
            this.location[i] = piece.location[i].clone();
        }
        this.width = piece.width;
        this.height = piece.height;
    }

    // getter setter
    public char getId(){return id;}
    public int[][] getLocation(){return location;}
    public int getWidth(){return width;}
    public int getHeight(){return height;}
    public int getLength(){return width*height;}

    public void setId(char id){this.id=id;}
    public void setLocation(int[][] loc){location=loc;}
    public void setWidth(int w){width=w;}
    public void setHeight(int h){height=h;}

    public void printPiece(){
        System.out.println("Piece ID: "+id);
    }
}

```

```

        System.out.println("Piece Location: ");
        for (int i = 0; i < location.length; i++) {
            System.out.print("(" + location[i][0] + ", " + location[i][1] + ") ");
        }
        System.out.println();
        System.out.println("Piece Width: " + width);
        System.out.println("Piece Height: " + height);
        System.out.println("Piece Orientation: " + getOrientation());
    }

    public boolean isPrimaryPiece(){
        return id == 'P';
    }

    public char getOrientation(){
        if (width == 1) {
            return 'v';
        }
        return 'h';
    }

    public void movePiece(int change) {
        if (getOrientation() == 'h') {
            for (int i = 0; i < width; i++) {
                location[i][1] += change;
            }
        } else {
            for (int i = 0; i < height; i++) {
                location[i][0] += change;
            }
        }
    }

    public List<Piece> possiblePieceMoves(List<Integer> possibleMoves) {
        List<Piece> possiblePieces = new ArrayList<>();
        for (int move : possibleMoves) {
            Piece newPiece = new Piece(this);
            newPiece.movePiece(move);
            possiblePieces.add(newPiece);
        }
        return possiblePieces;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
    }

```

```

        Piece other = (Piece) o;
        return id == other.id &&
            width == other.width &&
            height == other.height &&
            Arrays.deepEquals(location, other.location);
    }

    @Override
    public int hashCode() {
        int result = Objects.hash(id, width, height);
        result = 31 * result + Arrays.deepHashCode(location);
        return result;
    }
}

```

4.8 Simpul.java

Simpul.java adalah *file* dengan kelas Simpul. Kelas ini berguna untuk mencatat hasil dari gerakan yang dilakukan oleh algoritma. Kelas Simpul menyimpan objek Board dan id piece untuk mengidentifikasi gerakan yang dilakukan.

```

package lib;

public class Simpul {
    Board board;
    char idPiece;
    int moveValue;
    int cost;

    // constructor
    public Simpul(Board board, char idPiece, int moveValue, int cost) {
        this.board = board;
        this.idPiece = idPiece;
        this.moveValue = moveValue;
        this.cost = cost;
    }

    // getters
    public Board getBoard() {
        return board;
    }
    public char getIdPiece() {
        return idPiece;
    }
    public int getMoveValue() {

```

```
        return moveValue;
    }
    public int getCost() {
        return cost;
    }

    // setters
    public void setBoard(Board board) {
        this.board = board;
    }
    public void setIdPiece(char idPiece) {
        this.idPiece = idPiece;
    }
    public void setMoveValue(int moveValue) {
        this.moveValue = moveValue;
    }
    public void setCost(int cost) {
        this.cost = cost;
    }
}
```

BAB V

MASUKAN DAN LUARAN PROGRAM

5.1 Program Utama

```
PS C:\Clone Github\Tucil3_13523026_13523031> javac -d bin -sourcepath src src/Main.java
PS C:\Clone Github\Tucil3_13523026_13523031> java -cp bin Main

RUSH HOUR SOLVER

Dibuat oleh: 13523026 - Bertha Soliany Frandi dan 13523031 - Rafen Max Alessandro

Masukkan nama file (ex.txt) atau 'exit' untuk keluar:
█
```

Tampilan terminal setelah program dijalankan

```
Masukkan nama file (ex.txt) atau 'exit' untuk keluar:
input.txt
A:3 B:3
Total piece selain primary piece: 2

Berikut adalah board yang dimasukkan:
AAP
..P
BB.
 K

Sekarang, pilih algoritma penyelesaian yang akan digunakan:
1. Algoritma UCS
2. Algoritma Greedy Best First Search
3. Algoritma A*
4. Keluar
Pilihan:
2

Pilihan kedua dipilih: menggunakan algoritma Greedy Best First Search
Pilih heuristik yang digunakan:
1. Manhattan Distance ke pintu keluar
2. Jumlah piece yang menghalangi
3. Keduanya
4. Kembali ke menu utama
Pilihan:
█
```

Program menerima masukan nama file dan meminta masukan algoritma serta heuristik

[illegible]

Masukan pilihan algoritma valid, program memberikan solusi

Validasi masukan nama file tidak tersedia dalam folder test

```
Masukkan nama file (ex.txt) atau 'exit' untuk keluar:
```

```
Kosong wak
```

```
Masukkan nama file (ex.txt) atau 'exit' untuk keluar:
```

Validasi masukan nama file kosong

```
Sekarang, pilih algoritma penyelesaian yang ingin kamu gunakan:
```

1. Algoritma UCS
2. Algoritma Greedy Best First Search
3. Algoritma A*
4. Keluar

```
Pilihan:
```

```
a
```

```
Input harus berupa angka.
```

Validasi masukan pilihan algoritma tidak valid, bukan berupa angka

```
1. Algoritma UCS
```

```
2. Algoritma Greedy Best First Search
```

```
3. Algoritma A*
```

```
4. Keluar
```

```
Pilihan:
```

```
5
```

```
Pilihan tidak valid, pastikan pilihan antara 1-4
```

Validasi masukan pilihan algoritma tidak valid, bukan berupa angka di antara 1 hingga 4

```
Pilihan kedua dipilih: menggunakan algoritma Greedy Best First Search
Pilih heuristik yang digunakan:
1. Manhattan Distance ke pintu keluar
2. Jumlah piece yang menghalangi
3. Keduanya
4. Kembali ke menu utama
Pilihan:
8
Pilihan tidak valid, pastikan pilihan antara 1-4
```

Validasi masukan pilihan heuristik tidak valid, bukan berupa angka di antara 1 hingga 4

```
test > ≡ input.txt
1 3
2 1
3 AAP
4 ..P
5 ...
6 K
```

```
Masukkan nama file (ex.txt) atau 'exit' untuk keluar:
input.txt
Gagal membaca file test\input.txt
java.io.IOException: Format ukuran papan tidak valid.
    at lib.InputOutput.readFile(InputOutput.java:21)
    at Main.main(Main.java:48)
```

Validasi masukan input tidak valid: dimensi papan tidak dalam bentuk panjang x lebar

```
test > ≡ input.txt
1 a 3
2 1
3 AAP
4 ..P
5 ...
6 K
```

```
test > ≡ input.txt
1 3 1.2
2 1
3 AAP
4 ..P
5 ...
6 K
```

```

Masukkan nama file (ex.txt) atau 'exit' untuk keluar:
input.txt
Gagal membaca file test\input.txt
java.io.IOException: Format salah: A dan B harus berupa bilangan bulat.
    at lib.InputOutput.readFile(InputOutput.java:29)
    at Main.main(Main.java:48)

```

Validasi masukan input tidak valid: dimensi papan tidak berupa bilangan bulat

```

test > ≡ input.txt
1 3 -5
2 1
3 AAP
4 ..P
5 ...
6 | K

```

```

Masukkan nama file (ex.txt) atau 'exit' untuk keluar:
input.txt
A:3 B:-5
Gagal membaca file test\input.txt
java.io.IOException: Format salah: A dan B harus berupa bilangan bulat positif.
    at lib.InputOutput.readFile(InputOutput.java:38)
    at Main.main(Main.java:48)

```

Validasi masukan input tidak valid: dimensi papan tidak berupa bilangan bulat positif

```

test > ≡ input.txt
1 3 3
2 AAP
3 ..P
4 ...
5 | K|

```

```

Masukkan nama file (ex.txt) atau 'exit' untuk keluar:
input.txt
A:3 B:3
Gagal membaca file test\input.txt
java.io.IOException: Format salah: tidak terdapat jumlah piece non-primary di baris kedua.
    at lib.InputOutput.readFile(InputOutput.java:45)
    at Main.main(Main.java:48)

```

Validasi masukan input tidak valid: tidak terdapat jumlah *piece* non-*primary*

test >	≡	input.txt
1		2 3
2		1
3		AAP
4		..P
5		...
6		K

```
Masukkan nama file (ex.txt) atau 'exit' untuk keluar:
input.txt
A:2 B:3
Total piece selain primary piece: 1

Gagal membaca file test\input.txt
java.io.IOException: Tinggi papan: 3 tidak sesuai dengan nilai A: 2
    at lib.InputOutput.readFile(InputOutput.java:76)
    at Main.main(Main.java:48)
```

Validasi masukan input tidak valid: nilai A tidak sesuai

test >	≡	input.txt
1		3 2
2		1
3		AAP
4		..P
5		...
6		K

```
Masukkan nama file (ex.txt) atau 'exit' untuk keluar:
input.txt
A:3 B:2
Total piece selain primary piece: 1

Gagal membaca file test\input.txt
java.io.IOException: Lebar papan 3 tidak sesuai dengan nilai B: 2
    at lib.InputOutput.readFile(InputOutput.java:101)
    at Main.main(Main.java:48)
```

Validasi masukan input tidak valid: nilai B tidak sesuai

```
test > ≡ input.txt
1 | 3 3
2 | 1
3 | AAP
4 | ..P
5 | ...
```

```
Masukkan nama file (ex.txt) atau 'exit' untuk keluar:
input.txt
A:3 B:3
Total piece selain primary piece: 1

Gagal membaca file test\input.txt
java.io.IOException: Pintu keluar (K) tidak ditemukan atau tidak valid.
    at lib.InputOutput.readFile(InputOutput.java:122)
    at Main.main(Main.java:48)
```

Validasi masukan input tidak valid: tidak terdapat pintu keluar K

```
test > ≡ input.txt
1 | 3 3
2 | 1
3 | AAP
4 | ..P
5 | K...
```

```
Masukkan nama file (ex.txt) atau 'exit' untuk keluar:
input.txt
A:3 B:3
Total piece selain primary piece: 1

Berikut adalah board yang dimasukkan:
AAP
..P
K...

Gagal membaca file test\input.txt
java.io.IOException: Konfigurasi board tidak solvable, primary piece (P) tidak akan dapat keluar dari pintu keluar (K) >:(
Silakan jalankan program lagi dengan konfigurasi board yang solvable!
```

Validasi masukan input tidak valid: *board* tidak dapat diselesaikan

test > ≡ input.txt

1	3 3
2	1
3	AAP
4	..P
5	...
6	K

```
Masukkan nama file (ex.txt) atau 'exit' untuk keluar:
input.txt
A:3 B:3
Total piece selain primary piece: 1

Berikut adalah board yang dimasukkan:
AAP
..P
...
K

Gagal membaca file test\input.txt
java.io.IOException: Konfigurasi board tidak solvable, primary piece (P) tidak akan dapat keluar dari pintu keluar (K) >:(
Silakan jalankan program lagi dengan konfigurasi board yang solvable!
```

Validasi masukan input tidak valid: *board* tidak dapat diselesaikan

test > ≡ input.txt

1	4 3
2	2
3	AAP
4	..P
5	..B
6	..B
7	K

```
Masukkan nama file (ex.txt) atau 'exit' untuk keluar:
input.txt
A:4 B:3
Total piece selain primary piece: 2

Berikut adalah board yang dimasukkan:
AAP
..P
..B
..B
K

Gagal membaca file test\input.txt
java.io.IOException: Konfigurasi board tidak solvable, terdapat piece yang menghalangi primary piece (P) untuk bergerak ke pintu keluar (K) >:(
Silakan jalankan program lagi dengan konfigurasi board yang solvable!
```

Validasi masukan input tidak valid: *board* tidak dapat diselesaikan

```
test > ≡ input.txt
1 | 3 3
2 | 1
3 | AAP
4 | .AP
5 | ...
6 | K
```

```
Masukkan nama file (ex.txt) atau 'exit' untuk keluar:
input.txt
A:3 B:3
Total piece selain primary piece: 1

Gagal membaca file test\input.txt
java.io.IOException: Piece A dengan ukuran 2 x 2 tidak valid.
    at lib.InputOutput.readFile(InputOutput.java:166)
    at Main.program(Main.java:52)
    at Main.main(Main.java:21)
```

Validasi masukan input tidak valid: terdapat *piece* dengan bentuk tidak valid

```
test > ≡ input.txt
1 | 3 3
2 | 1
3 | AAP
4 | ..P
5 | BB.
6 | K
```

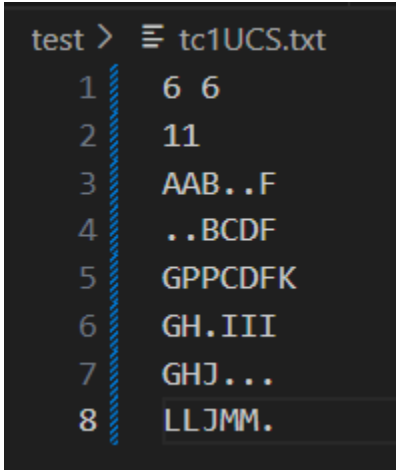

```
Masukkan nama file (ex.txt) atau 'exit' untuk keluar:
input.txt
A:3 B:3
Total piece selain primary piece: 1

Gagal membaca file test\input.txt
java.io.IOException: Jumlah piece 3 tidak sesuai dengan file input.
    at lib.InputOutput.readFile(InputOutput.java:177)
    at Main.program(Main.java:52)
    at Main.main(Main.java:21)
```

Validasi masukan input tidak valid: jumlah banyaknya *piece* tidak sesuai

5.3 Algoritma UCS

5.3.1 Test Case 1

File	
Masukan	

Luaran

```
      _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _  
      ( _ - < / _ \ / / / / ( _ - < / /  
      / _ _ \ _ _ / \ _ _ , / _ _ / /
```

Waktu penyelesaian: 43 ns

Total simpul kondisi board yang ditelusuri: 1636

Kondisi awal papan:

AAB..F

..BCDF

GPPCDFK

GH.III

GHJ...

LLJMM.

Gerakan 4: D sebanyak 1 petak ke atas

AABCD.

..BCD.

GPP...K

GHIIIF

GHJ..F

LLJMMF

Gerakan 5: P sebanyak 3 petak ke kanan

AABCD.

..BCD.

G...PPK

GHIIIF

GHJ..F

LLJMMF

Solusi ditemukan! ^^

5.3.2 Test Case 2

File	<pre>test > ≡ tc2UCS.txt 1 6 6 2 10 3 AEE.H. 4 AFG.HI 5 AFGPPIK 6 ...J.I 7 ..DJLL 8 ..DMM. 9 </pre>
Masukan	<pre>RUSH HOUR SOLVER Dibuat oleh: 13523026 - Bertha Soliany Frandi dan 13523031 - Rafen Max Alessandro Masukkan nama file (ex.txt) atau 'exit' untuk keluar: tc2UCS.txt A:6 B:6 Total piece selain primary piece: 10 Berikut adalah board yang dimasukkan: AEE.H. AFG.HI AFGPPIK ...J.I ..DJLL ..DMM. Sekarang, pilih algoritma penyelesaian yang akan digunakan: 1. Algoritma UCS 2. Algoritma Greedy Best First Search 3. Algoritma A* 4. Keluar Pilihan: 1 Pilihan pertama dipilih: menggunakan algoritma UCS</pre>

Luaran

```
      _ _ _ _ _ _ _ _ _ _  
      ( _ - < / _ \ / / / / ( _ - < / /  
      / _ _ \ _ _ / \ _ _ , / _ _ / /
```

Waktu penyelesaian: 205 ns

Total simpul kondisi board yang ditelusuri: 53112

Kondisi awal papan:

AEE.H.

AFG.HI

AFGPPIK

...J.I

..DJLL

..DMM.

Gerakan 8: I sebanyak 2 petak ke bawah

EEGJH.

..GJH.

APP...K

A....I

AFDLLI

.FDMMI

Gerakan 9: P sebanyak 3 petak ke kanan

EEGJH.

..GJH.

A...PPK

A....I

AFDLLI

.FDMMI

Solusi ditemukan! ^^

5.3.3 Test Case 3

File	<pre>test > ≡ tc3UCS.txt 1 6 6 2 11 3 PP.B.CK 4 DD.BEC 5 FGGBEC 6 FHH.E. 7 ..JIIY 8 ..JMMY 9 </pre>
Masukan	<pre>RUSH HOUR SOLVER Dibuat oleh: 13523026 - Bertha Soliany Frandi dan 13523031 - Rafen Max Alessandro Masukkan nama file (ex.txt) atau 'exit' untuk keluar: tc3UCS.txt A:6 B:6 Total piece selain primary piece: 11 Berikut adalah board yang dimasukkan: PP.B.CK DD.BEC FGGBEC FHH.E. ..JIIY ..JMMY Sekarang, pilih algoritma penyelesaian yang akan digunakan: 1. Algoritma UCS 2. Algoritma Greedy Best First Search 3. Algoritma A* 4. Keluar Pilihan: 1 Pilihan pertama dipilih: menggunakan algoritma UCS</pre>

Luaran

$\frac{\overline{f} \cdot \overline{g}}{(f - g) \wedge (f + g)}$

Waktu penyelesaian: 72 ns

Total simpul kondisi board yang ditelusuri: 1507

Kondisi awal papan:

PP.B.CK

DD.BEC

FGGBEC

FHH.E.

..JIIY

..JMMY

Gerakan 2: C sebanyak 1 petak ke bawah

PP...K

DD.BEC

FGGBEC

FHHBEC

..JIIY

..JMMY

Gerakan 3: P sebanyak 4 petak ke kanan

... PPK

DD.BEC

FGGBEC

FHHBEC

..JIIY

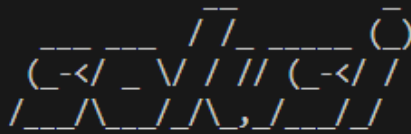
..JMMY

Solusi ditemukan! ^^

5.3.4 Test Case 4

File	
Masukan	

Luaran



Waktu penyelesaian: 125 ns

Total simpul kondisi board yang ditelusuri: 24532

Kondisi awal papan:

AEE.H.

AFG.HI

AFG**PP**IK

BBBJ.I

.DJLL

CCDMM.

Gerakan 50: I sebanyak 3 petak ke bawah

EEGJH.

A.GJH.

A..**PP**.**K**

AFBBBI

.FDLLI

CCDMMI

Gerakan 51: P sebanyak 1 petak ke kanan

EEGJH.

A.GJH.

A..**PP****K**

AFBBBI

.FDLLI

CCDMMI

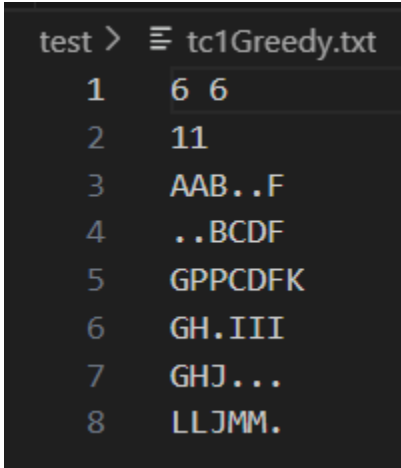
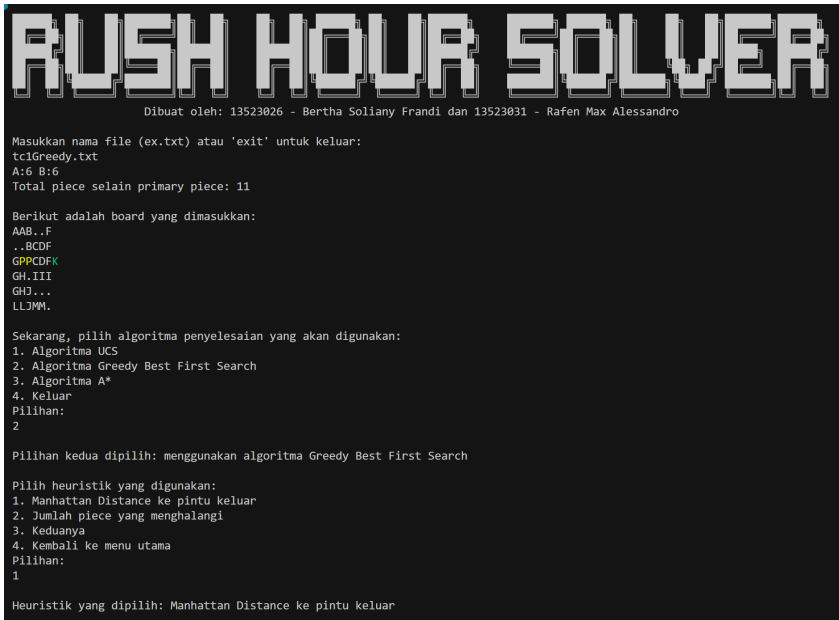
Solusi ditemukan! ^^

5.3.5 Test Case 5

File	<pre>test > ≡ tc5UCS.txt 1 5 6 2 6 3 F....A 4 F....A 5 FPP..AK 6 H.CCCC 7 HDDDEE</pre>
Masukan	<pre>RUSH HOUR SOLVER Dibuat oleh: 13523026 - Bertha Soliany Frandi dan 13523031 - Rafen Max Alessandro Masukkan nama file (ex.txt) atau 'exit' untuk keluar: tc5UCS.txt A:5 B:6 Total piece selain primary piece: 6 Berikut adalah board yang dimasukkan: F....A F....A FPP..AK H.CCCC HDDDEE Sekarang, pilih algoritma penyelesaian yang akan digunakan: 1. Algoritma UCS 2. Algoritma Greedy Best First Search 3. Algoritma A* 4. Keluar Pilihan: 1 Pilihan pertama dipilih: menggunakan algoritma UCS</pre>
Luaran	<pre> _ _ _ _ _ / / _ _ _ _ _ () (- < / _ \ / / / (- < / / / _ _ \ _ / \ , / _ _ / / Waktu penyelesaian: 16 ns Total simul kondisi board yang ditelusuri: 30 Tidak ditemukan solusi... :(</pre>

5.4 Algoritma *Greedy Best First Search*

5.4.1 Test Case 1

File	 <pre>test > ≡ tc1Greedy.txt 1 6 6 2 11 3 AAB..F 4 ..BCDF 5 GPPCDFK 6 GH.III 7 GHJ... 8 LLJMM.</pre>
Masukan	 <pre>RUSH HOUR SOLVER Dibuat oleh: 13523026 - Bertha Soliany Frandi dan 13523031 - Rafen Max Alessandro Masukkan nama file (ex.txt) atau 'exit' untuk keluar: tc1Greedy.txt A:6 B:6 Total piece selain primary piece: 11 Berikut adalah board yang dimasukkan: AAB..F ..BCDF GPPCDFK GH.III GHJ... LLJMM. Sekarang, pilih algoritma penyelesaian yang akan digunakan: 1. Algoritma UCS 2. Algoritma Greedy Best First Search 3. Algoritma A* 4. Keluar Pilihan: 2 Pilihan kedua dipilih: menggunakan algoritma Greedy Best First Search Pilih heuristik yang digunakan: 1. Manhattan Distance ke pintu keluar 2. Jumlah piece yang menghalangi 3. Keduanya 4. Kembali ke menu utama Pilihan: 1 Heuristik yang dipilih: Manhattan Distance ke pintu keluar</pre>

Luaran

```
      _ _ _ _ _  
      ( _ < / _ \ / / / ( _ < / /  
      / _ _ \ _ / \ , / _ / /
```

Waktu penyelesaian: 35 ns

Total simpul kondisi board yang ditelusuri: 268

Kondisi awal papan:

AAB..F

..BCDF

GPPCDFK

GH.III

GHJ...

LLJMM.

Gerakan 12: F sebanyak 1 petak ke bawah

AA.CD.

GHBCD.

GHBP.PK

GIII.F

..J..F

LLJMMF

Gerakan 13: P sebanyak 1 petak ke kanan

AA.CD.

GHBCD.

GHBP.PK

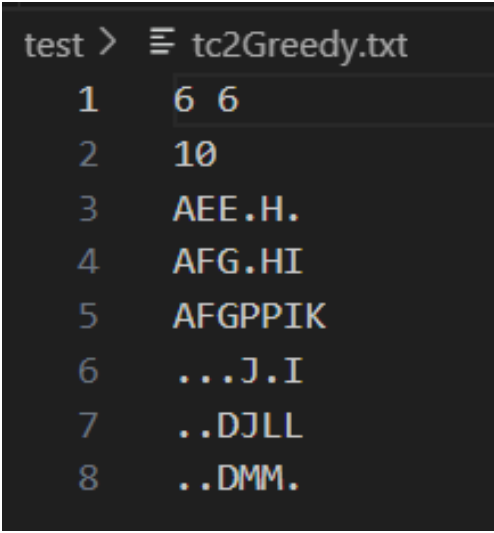
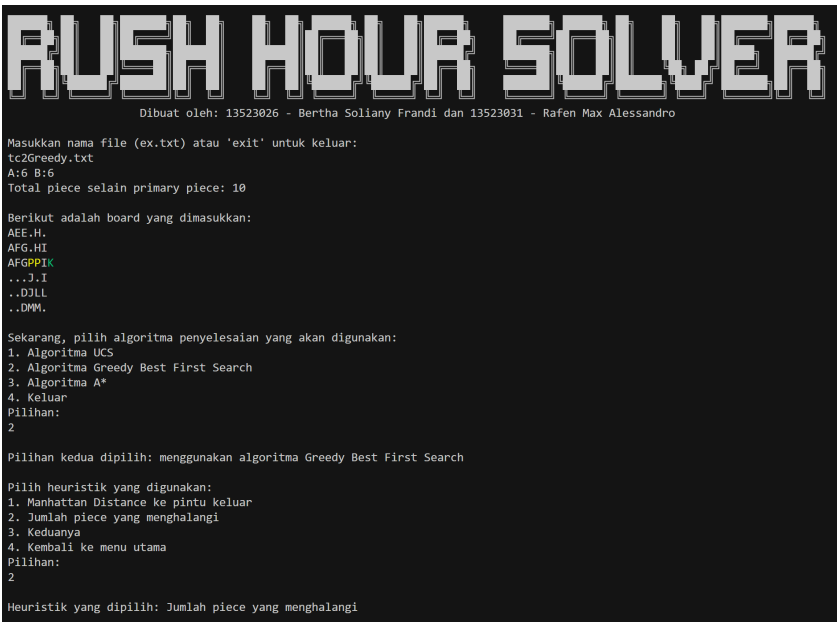
GIII.F

..J..F

LLJMMF

Solusi ditemukan! ^^

5.4.2 Test Case 2

File	 <pre>test > ≡ tc2Greedy.txt 1 6 6 2 10 3 AEE.H. 4 AFG.HI 5 AFGPPIK 6 ...J.I 7 ..DJLL 8 ..DMM.</pre>
Masukan	 <pre>RUSH HOUR SOLVER Dibuat oleh: 13523026 - Bertha Soliany Frandi dan 13523031 - Rafen Max Alessandro Masukkan nama file (ex.txt) atau 'exit' untuk keluar: tc2Greedy.txt A:6 B:6 Total piece selain primary piece: 10 Berikut adalah board yang dimasukkan: AEE.H. AFG.HI AFGPPIK ...J.I ..DJLL ..DMM. Sekarang, pilih algoritma penyelesaian yang akan digunakan: 1. Algoritma UCS 2. Algoritma Greedy Best First Search 3. Algoritma A* 4. Keluar Pilihan: 2 Pilihan kedua dipilih: menggunakan algoritma Greedy Best First Search Pilih heuristik yang digunakan: 1. Manhattan Distance ke pintu keluar 2. Jumlah piece yang menghalangi 3. Keduanya 4. Kembali ke menu utama Pilihan: 2 Heuristik yang dipilih: Jumlah piece yang menghalangi</pre>

Luaran

```

      ____  ____  / /  ____  ( )
      ( _- < /  _ \ / /  ( _- < /
      / ____ \ ____ / \ ____ /
Waktu penyelesaian: 315 ns
Total simpul kondisi board yang ditelusuri: 49438

```

Kondisi awal papan:

AEE.H.

AFG.HI

AFGPPIK

...J.I

..DJLL

. . DMM .

Gerakan 44: I sebanyak 3 petak ke bawah

.EEJH.

A..JH.

AFGPP.K

AFG..I

LLD..I

MMD..I

Gerakan 45: P sebanyak 1 petak ke kanan

.EEJH.

A..JH.

AFG.PPK

AFG..I

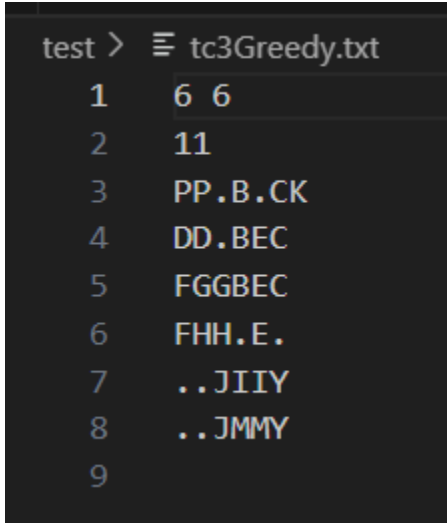
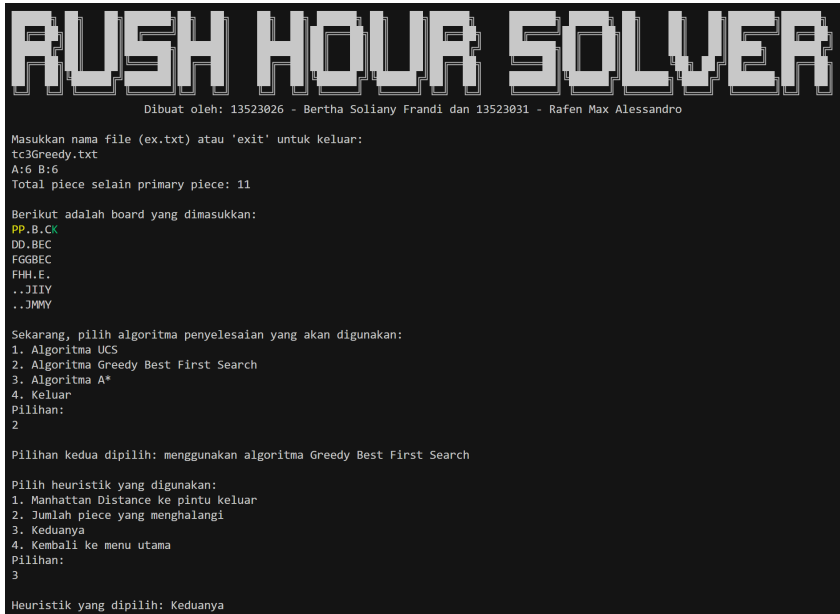
LLD..I

MMD..I

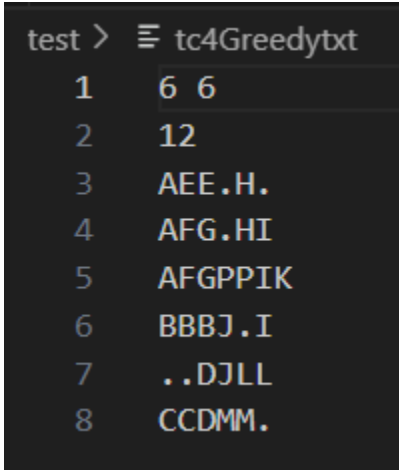
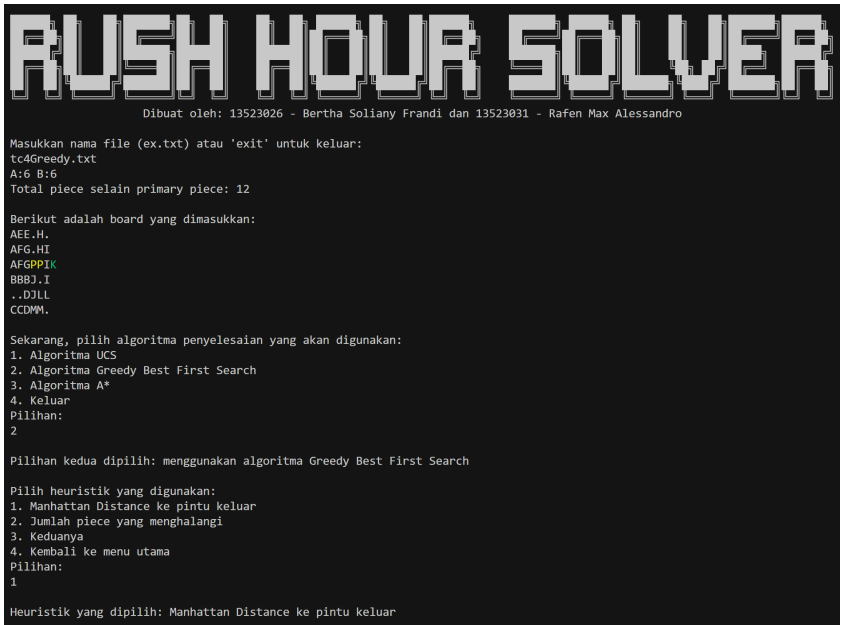
Solusi ditemukan! ^^

Berhasil menyimpan output ke dalam file test\solution_tc2Greedy.txt

5.4.3 Test Case 3

File	 <pre>test > ≡ tc3Greedy.txt 1 6 6 2 11 3 PP.B.CK 4 DD.BEC 5 FGGBEC 6 FHH.E. 7 ..JIIY 8 ..JMMY 9</pre>
Masukan	 <pre>RUSH HOUR SOLVER Dibuat oleh: 13523026 - Bertha Soliany Frandi dan 13523031 - Rafen Max Alessandro Masukkan nama file (ex.txt) atau 'exit' untuk keluar: tc3Greedy.txt A:6 B:6 Total piece selain primary piece: 11 Berikut adalah board yang dimasukkan: PP.B.CK DD.BEC FGGBEC FHH.E. ..JIIY ..JMMY Sekarang, pilih algoritma penyelesaian yang akan digunakan: 1. Algoritma UCS 2. Algoritma Greedy Best First Search 3. Algoritma A* 4. Keluar Pilihan: 2 Pilihan kedua dipilih: menggunakan algoritma Greedy Best First Search Pilih heuristik yang digunakan: 1. Manhattan Distance ke pintu keluar 2. Jumlah piece yang menghalangi 3. Keduanya 4. Kembali ke menu utama Pilihan: 3 Heuristik yang dipilih: Keduanya</pre>

5.4.4 Test Case 4

File	
Masukan	

Luaran

```
      _ _ _ _ _ _ _ _ _ _  
      (-</ _ \ / / / / (-</ /  
      / _ \ / _ \ / _ \ , / _ \ /
```

Waktu penyelesaian: 92 ns

Total simpul kondisi board yang ditelusuri: 11303

Kondisi awal papan:

AEE.H.

AFG.HI

AFGPPIK

BBBJ.I

..DJLL

CCDMM.

Gerakan 107: I sebanyak 1 petak ke bawah

.EEJH.

.FGJH.

AFGPP.K

A.BBBI

A.DLLI

CCDMMI

Gerakan 108: P sebanyak 1 petak ke kanan

.EEJH.

.FGJH.

AFG.PPK

A.BBBI

A.DLLI

CCDMMI

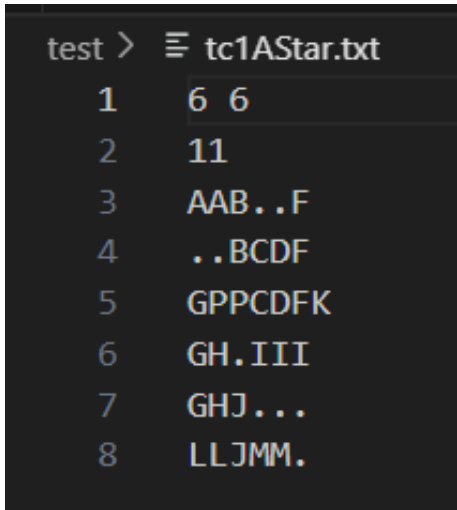
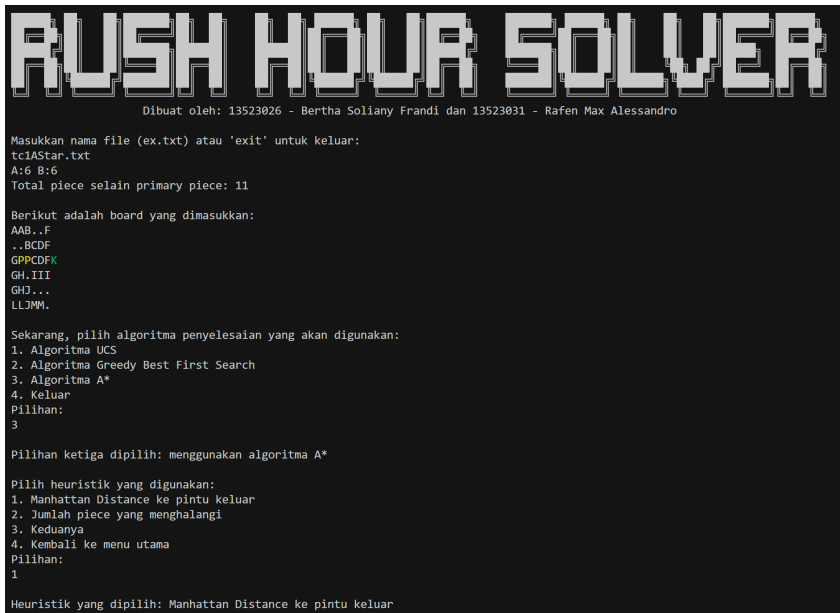
Solusi ditemukan! ^^

5.4.5 Test Case 5

<div>File</div>	<div> <pre> test > ≡ tc5Greedy.txt 1 5 6 2 6 3 F...A 4 F...A 5 FPP..AK 6 H.CCCC 7 HDDDEE </pre> </div>
<div>Masukan</div>	<div> <h1>RUSH HOUR SOLVER</h1> <p>Dibuat oleh: 13523026 - Bertha Soliany Frandi dan 13523031 - Rafen Max Alessandro</p> <p>Masukkan nama file (ex.txt) atau 'exit' untuk keluar: tc5Greedy.txt A:5 B:6 Total piece selain primary piece: 6</p> <p>Berikut adalah board yang dimasukkan:</p> <pre> F...A F...A FPP..AK H.CCCC HDDDEE </pre> <p>Sekarang, pilih algoritma penyelesaian yang akan digunakan:</p> <ol style="list-style-type: none"> 1. Algoritma UCS 2. Algoritma Greedy Best First Search 3. Algoritma A* 4. Keluar <p>Pilihan: 2</p> <p>Pilihan kedua dipilih: menggunakan algoritma Greedy Best First Search</p> <p>Pilih heuristik yang digunakan:</p> <ol style="list-style-type: none"> 1. Manhattan Distance ke pintu keluar 2. Jumlah piece yang menghalangi 3. Keduanya 4. Kembali ke menu utama <p>Pilihan: 2</p> <p>Heuristik yang dipilih: Jumlah piece yang menghalangi</p> </div>
<div>Luaran</div>	<div>  <p>Waktu penyelesaian: 8 ns Total simpul kondisi board yang ditelusuri: 30</p> <p>Tidak ditemukan solusi... :(Berhasil menyimpan output ke dalam file test\solution_tc5Greedy.txt</p> </div>

5.5 Algoritma A*

5.5.1 Test Case 1

File	 <pre>test > ≡ tc1AStar.txt 1 6 6 2 11 3 AAB..F 4 ..BCDF 5 GPPCDFK 6 GH.III 7 GHJ... 8 LLJMM.</pre>
Masukan	 <pre>RUSH HOUR SOLVER Dibuat oleh: 13523026 - Bertha Soliany Frandi dan 13523031 - Rafen Max Alessandro Masukkan nama file (ex.txt) atau 'exit' untuk keluar: tc1AStar.txt A:6 B:6 Total piece selain primary piece: 11 Berikut adalah board yang dimasukkan: AAB..F ..BCDF GPPCDFK GH.III GHJ... LLJMM. Sekarang, pilih algoritma penyelesaian yang akan digunakan: 1. Algoritma UCS 2. Algoritma Greedy Best First Search 3. Algoritma A* 4. Keluar Pilihan: 3 Pilihan ketiga dipilih: menggunakan algoritma A* Pilih heuristik yang digunakan: 1. Manhattan Distance ke pintu keluar 2. Jumlah piece yang menghalangi 3. Keduanya 4. Kembali ke menu utama Pilihan: 1 Heuristik yang dipilih: Manhattan Distance ke pintu keluar</pre>

Masukan

RUSH HOUR SOLVER

Dibuat oleh: 13523026 - Bertha Soliany Frandi dan 13523031 - Rafen Max Alessandro

Masukkan nama file (ex.txt) atau 'exit' untuk keluar:

tc2AStar.txt

A:6 B:6

Total piece selain primary piece: 10

Berikut adalah board yang dimasukkan:

AEE.H.

AFG.HI

AFGPPIK

...J.I

..DJLL

..DMM.

Sekarang, pilih algoritma penyelesaian yang akan digunakan:

1. Algoritma UCS

2. Algoritma Greedy Best First Search

3. Algoritma A*

4. Keluar

Pilihan:

3

Pilihan ketiga dipilih: menggunakan algoritma A*

Pilih heuristik yang digunakan:

1. Manhattan Distance ke pintu keluar

2. Jumlah piece yang menghalangi

3. Keduanya

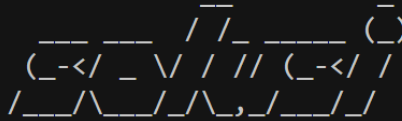
4. Kembali ke menu utama

Pilihan:

2

Heuristik yang dipilih: Jumlah piece yang menghalangi

Luaran



Waktu penyelesaian: 548 ns

Total simpul kondisi board yang ditelusuri: 153371

Kondisi awal papan:

AEE.H.

AFG.HI

AFGPPIK

...J.I

..DJLL

..DMM.

Gerakan 13: I sebanyak 3 petak ke bawah

EEGJH.

..GJH.

A..PP.K

AF...I

AFDILLI

..DMMI

Gerakan 14: P sebanyak 1 petak ke kanan

EEGJH.

..GJH.

A..PPK

AF...I

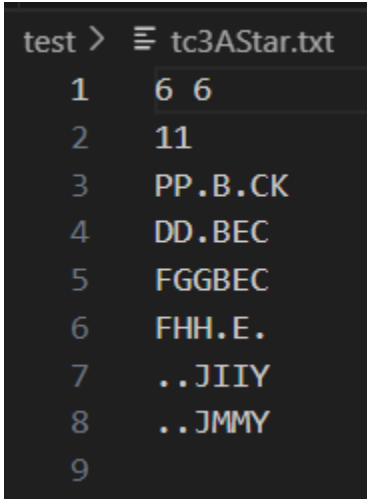
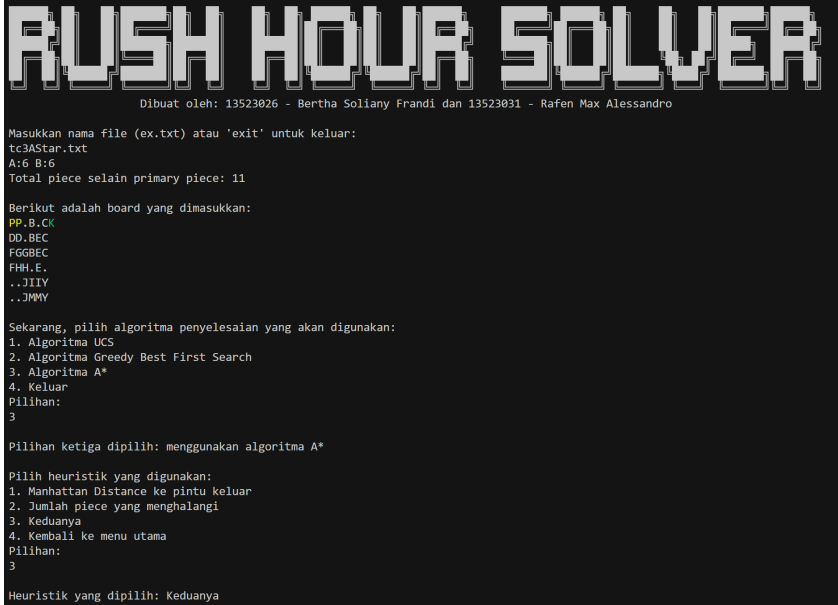
AFDILLI

..DMMI

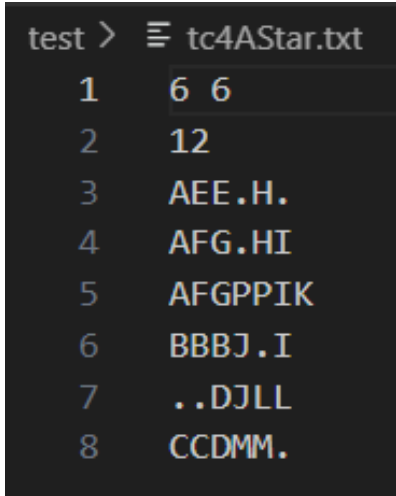
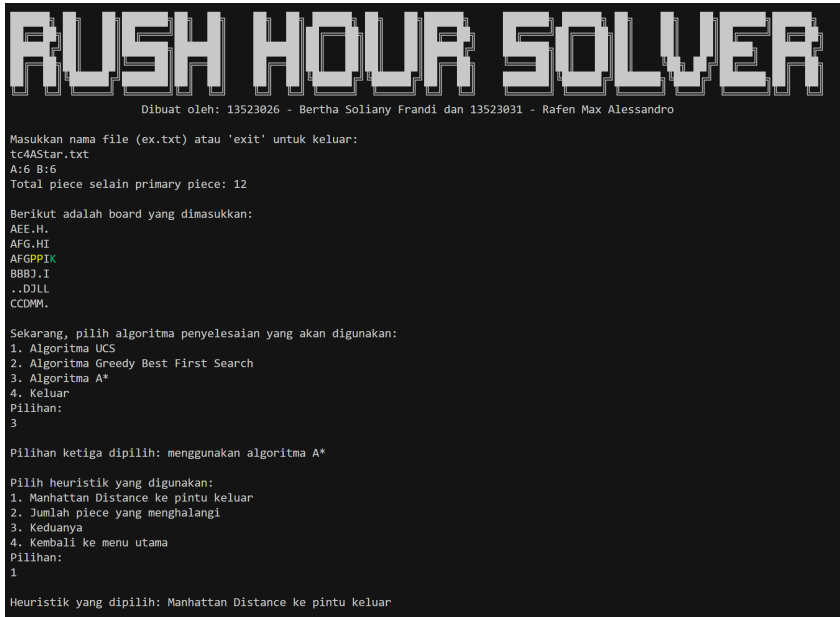
Solusi ditemukan! ^^

Berhasil menyimpan output ke dalam file test\solution_tc2AStar.txt

5.5.3 Test Case 3

File	
Masukan	

5.5.4 Test Case 4

File	 <pre>test > ≡ tc4AStar.txt 1 6 6 2 12 3 AEE.H. 4 AFG.HI 5 AFGPPIK 6 BBBJ.I 7 ..DJLL 8 CCDMM.</pre>
Masukan	 <pre>RUSH HOUR SOLVER Dibuat oleh: 13523026 - Bertha Soliany Frandi dan 13523031 - Rafen Max Alessandro Masukkan nama file (ex.txt) atau 'exit' untuk keluar: tc4AStar.txt A:6 B:6 Total piece selain primary piece: 12 Berikut adalah board yang dimasukkan: AEE.H. AFG.HI AFGPPIK BBBJ.I ..DJLL CCDMM. Sekarang, pilih algoritma penyelesaian yang akan digunakan: 1. Algoritma UCS 2. Algoritma Greedy Best First Search 3. Algoritma A* 4. Keluar Pilihan: 3 Pilihan ketiga dipilih: menggunakan algoritma A* Pilih heuristik yang digunakan: 1. Manhattan Distance ke pintu keluar 2. Jumlah piece yang menghalangi 3. Keduanya 4. Kembali ke menu utama Pilihan: 1 Heuristik yang dipilih: Manhattan Distance ke pintu keluar</pre>

BAB VI

ANALISIS PERCOBAAN

Agar dapat digunakan sebagai hasil analisis yang objektif, ketiga algoritma diuji menggunakan *test case* yang sama. Begitu pula dengan heuristik yang digunakan oleh *Greedy Best First Search* dan *A** dipastikan sama untuk *test case* yang sama. Dengan demikian, lama waktu penyelesaian, jumlah simpul yang ditelusuri, serta banyaknya langkah dapat dijadikan pembanding untuk menganalisis tingkat efisiensi dan optimalisasi dari ketiga algoritma dibandingkan satu dengan yang lain. Ketiga kriteria hasil dari ketiga algoritma ditabulasikan ke dalam tabel berikut.

Tabel perbandingan hasil kriteria ketiga algoritma untuk setiap *test case*

<i>Test case</i>	UCS			<i>Greedy Best First Search</i>			<i>A*</i>		
	waktu	simpul	langkah	waktu	simpul	langkah	waktu	simpul	langkah
1	43	1636	5	27	268	13	53	1163	6
2	205	53112	9	315	49438	45	548	153371	14
3	72	1507	3	19	37	3	38	462	3
4	125	24532	51	120	11303	108	194	22860	51
5	16	30	-	8	30	-	10	30	-

Berdasarkan hasil tabulasi data, dapat diambil kesimpulan hasil akhir sebagai berikut:

1. UCS memberikan jumlah langkah paling minimal untuk dua dari empat *test case* yang memberikan hasil. Namun, waktu penyelesaian dan jumlah simpul yang ditelusuri memiliki nilai paling maksimal dibandingkan kedua algoritma lainnya;
2. *Greedy Best First Search* memberikan waktu penyelesaian dan jumlah simpul yang ditelusuri paling minimal. Namun, jumlah langkah dari jalur yang dibentuk jauh lebih banyak dibandingkan kedua algoritma lainnya, dengan kasus terparah dinyatakan pada *test case* kedua dengan jumlah langkah hingga sembilan kali lebih panjang;
3. Walaupun *A** tidak memiliki kriteria yang bernilai minimal, dapat dilihat bahwa jumlah langkah dari jalur yang dibentuk tidak jauh berbeda dari UCS yang memberikan jumlah

langkah paling minimal, dengan perbedaan hanya mencapai lima langkah. Selain itu, waktu penyelesaian dan jumlah simpul yang ditelusuri umumnya lebih sedikit dibandingkan algoritma UCS, sehingga dapat dinyatakan bahwa algoritma A* memberikan hasil yang paling efisien dan optimal.

Hasil analisis selaras dengan penjelasan mengenai ketiga algoritma yang telah dijelaskan sebelumnya. *Greedy Best First Search* memberikan waktu eksekusi terpendek karena algoritma ini lebih berfokus pada penyelesaian cepat berdasarkan evaluasi simpul kondisi *board* yang lebih dekat dengan target berdasarkan heuristik. Algoritma A* turut memberikan waktu eksekusi lebih cepat dibandingkan UCS karena pemanfaatan heuristik, sedangkan UCS dipastikan akan memberikan waktu eksekusi paling lama di antara ketiga algoritma karena algoritma ini melakukan evaluasi terhadap setiap simpul kondisi *board* yang dibangkitkan selama belum mencapai *goal node*. Seperti yang telah dijelaskan sebelumnya, pada kasus penyelesaian permainan Rush Hour, karena *cost* dinyatakan sebagai jumlah langkah yang dilakukan, maka UCS akan menelusuri simpul layaknya BFS yang memaksimalkan jumlah simpul yang ditelusuri.

Selain itu, dampak dari algoritma *Greedy Best First Search* yang hanya mengandalkan fungsi heuristik sebagai penentuan *cost* terlihat dari jumlah langkah yang lebih besar secara signifikan, khususnya ketika dibandingkan dengan algoritma UCS dan A*. Algoritma UCS dapat memberikan jalur hasil yang relatif optimal karena sifatnya yang melakukan evaluasi terhadap setiap simpul sebagai pembangun jalur, dan algoritma A* juga memberikan jalur hasil yang relatif optimal karena mempertimbangkan *cost* terhadap *start node* dalam mengevaluasi simpul, sehingga simpul yang telah memberikan *cost* lebih kecil akan lebih diutamakan. Konsep ini digunakan oleh masing-masing algoritma UCS dan A* untuk memastikan bahwa solusi yang relatif optimal dapat ditentukan.

Pendekatan algoritma UCS yang memperluas ruang pemeriksaan tanpa mempertimbangkan perkiraan jarak menuju *goal node* mengakibatkan UCS dapat menjadi sangat tidak efisien, khususnya dalam kasus graf dengan faktor *branching* / percabangan besar dan *depth* / kedalaman jauh bagi *goal node*. Kompleksitas waktu dari algoritma UCS dapat mencapai $O(b^d)$ untuk b rata-rata faktor *branching* / percabangan dari setiap simpul kondisi *board* dan d

kedalaman *goal node*. Hal ini berbeda dengan algoritma *Greedy Best First Search* yang mempertimbangkan fungsi heuristik dalam mengevaluasi simpul. Penggunaan fungsi heuristik membantu kedua algoritma untuk tidak mengevaluasi simpul kondisi *board* yang tidak perlu, sehingga kompleksitas waktu hanya bergantung kepada seberapa *admissible* fungsi heuristik yang digunakan. Jika fungsi heuristik tidak berhasil mengarahkan pemilihan simpul kondisi *board* secara efektif menuju tujuan, maka jumlah simpul yang ditelusuri dapat meningkat secara tidak perlu. Heuristik yang baik, efektif, dan *admissible* mengakibatkan algoritma *Greedy Best First Search* dan A* untuk menghasilkan jalur dengan *cost* terendah, yang umumnya lebih baik dibandingkan kompleksitas waktu $O(b^d)$.

Dengan demikian, ketiga algoritma menjadi pendekatan yang baik sesuai dengan kebutuhan dari program. Jika dibutuhkan jalur yang dipastikan optimal tanpa memerhatikan waktu eksekusi, dapat digunakan algoritma UCS. Sedangkan jika lebih dipentingkan waktu eksekusi tercepat, algoritma *Greedy Best First Search* menjadi pilihan terbaik untuk digunakan, walaupun tidak dapat dipastikan akan memberikan solusi yang optimal (bahkan terkadang jauh dari optimal). Lalu, jika program mengutamakan efisiensi waktu dan hasil yang optimal, maka algoritma A* dapat digunakan sebagai pendekatan dalam pembentukan program.

BAB VII

KESIMPULAN

Melalui tugas kecil 3 Strategi Algoritma, kelompok diminta untuk membentuk program yang dapat menentukan jalur penyelesaian terpendek atau paling optimal untuk permainan Rush Hour, permainan *puzzle* logika berbasis papan dengan tujuan memindahkan *primary piece* ke posisi keluar dari papan yang telah ditentukan dengan menggerakkan *piece-piece* lain yang menghalangi. Implementasi program dibentuk menggunakan bahasa pemrograman Java dan mengimplementasikan tiga bentuk pendekatan algoritma *path-finding*, yaitu UCS, *Greedy Best First Search*, dan A*. Ketiga algoritma menyatakan pendekatan yang berbeda dalam menentukan jalur paling optimal dari kondisi awal papan permainan menuju kondisi menang, yang menunjukkan perbedaan tingkat efisiensi dan optimalisasi dalam memberikan hasil. Analisis dari kinerja program menunjukkan bahwa algoritma A* memberikan hasil yang cenderung optimal. Algoritma ini dapat menemukan jalur terpendek dalam waktu yang relatif lebih singkat dan banyak penelusuran simpul minimal dibandingkan kedua algoritma lainnya.

BAB VIII

LAMPIRAN

1. Referensi

GeeksforGeeks. (2024). *Uniform Cost Search (UCS) in AI*. Retrieved May 18, 2025 from <https://www.geeksforgeeks.org/uniform-cost-search-ucs-in-ai/>.

GeeksforGeeks. (2024). *Greedy Best First Search Algorithm*. Retrieved May 19, 2025, from <https://www.geeksforgeeks.org/greedy-best-first-search-algorithm/>.

GeeksforGeeks. (2024). *A* Search Algorithm*. Retrieved May 19, 2025 from <https://www.geeksforgeeks.org/a-search-algorithm/>.

Informatika.stei.itb.ac.id. (2025) Penentuan Rute (*Route/Path Planning*) Bagian 1: BFS, DFS, UCS, Greedy Best First Search. Diakses pada 20 Mei 2025, dari [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/21-Route-Planning-\(2025\)-Bagian1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/21-Route-Planning-(2025)-Bagian1.pdf)

Informatika.stei.itb.ac.id. (2025) Penentuan Rute (*Route/Path Planning*) Bagian 2: Algoritma A*. Diakses pada 20 Mei 2025, dari <https://informatika.stei>.

2. Pranala repository GitHub

https://github.com/BerthaSoliany/Tucil3_13523026_13523031

3. Tabel Checklist

Tabel *checklist*

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 dan menyimpan solusi berupa print board tahap per tahap dalam berkas .txt	✓	
5. [Bonus] Implementasi algoritma pathfinding alternatif		✓
6. [Bonus] Implementasi 2 atau lebih heuristik alternatif	✓	

7. [Bonus] Program memiliki GUI		✓
8. Program dan laporan dibuat (kelompok) sendiri	✓	