

LAPORAN TUGAS KECIL 3
IF2211 STRATEGI ALGORITMA
Penyelesaian Puzzle Rush Hour Menggunakan Algoritma
Pathfinding



Disusun oleh:

Joel Hotlan Haris Siahaan 13523025

Julius Arthur 13523030

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

2025

DAFTAR ISI

DAFTAR ISI.....	1
BAB I.....	2
LANDASAN TEORI.....	2
Algoritma Uniform Cost Search.....	2
Algoritma Greedy Best First.....	3
Algoritma A*.....	4
BAB II.....	6
ANALISIS DAN IMPLEMENTASI ALGORITMA.....	6
Algoritma Uniform Cost Search.....	6
Algoritma Greedy Best First.....	7
Algoritma A*.....	7
BAB III.....	10
SOURCE PROGRAM.....	10
BAGIAN IV.....	26
HASIL PENGUJIAN.....	26
3.1 Input file: papan1.txt.....	26
3.2 Input file: papan2.txt.....	26
3.3 Input file: papan3.txt.....	35
3.4 Input file: speed.jpg.....	36
BAGIAN V.....	39
IMPLEMENTASI BONUS.....	39
LINK REPOSITORY.....	42
TABEL CHECKLIST.....	42

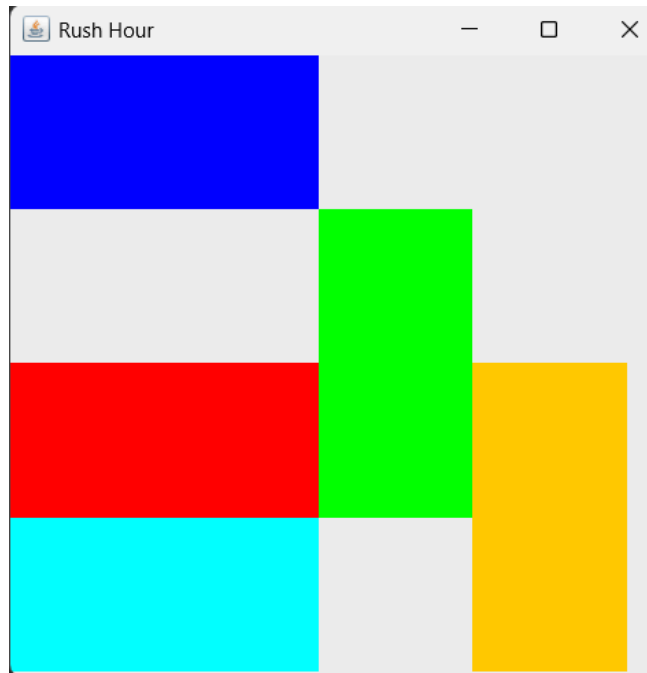
BAB I

LANDASAN TEORI

Algoritma Uniform Cost Search

Uniform Cost Search / UCS adalah sebuah algoritma pencarian tanpa mengetahui domainnya (uninformed search). Algoritma ini tidak memperhatikan keadaan pencarian dan ruang pencarian. Algoritma ini mirip dengan dengan BFS dalam mencari solusi termurah. Namun, BFS lebih menekankan pencarian dengan kedalaman terdekat yang tidak menjamin solusi termurah. Sedangkan, UCS akan mencari solusi termurah dari berbagai kemungkinan yang ada.

Biaya/ cost merupakan harga yang dibutuhkan untuk mencapai sebuah keadaan dari kondisi awal. UCS akan selalu melakukan pencarian terhadap jalur/ path termurah yang tersedia. Jika terdapat beberapa cara mencapai sebuah kondisi, UCS hanya akan menelusuri jalur termurah dari berbagai jalur tersebut. Akibatnya, UCS akan menghasilkan jalur dengan cost termurah. Perhatikan bahwa UCS tidak menjamin solusi akan memiliki kedalaman terkecil, tapi menjamin solusi termurah.



Misal *piece* utama adalah *piece* merah dan gambar di atas adalah kondisi awal permainan. Perhatikan bahwa terdapat banyak kondisi yang dapat diraih dari kondisi ini. Setiap kondisi dapat diraih dengan menggerakkan satu *piece*. Pada algoritma ini, terdapat sebuah *Priority Queue*, yang menyimpan keadaan/*state* yang diraih serta ongkos mencapai keadaan tersebut.

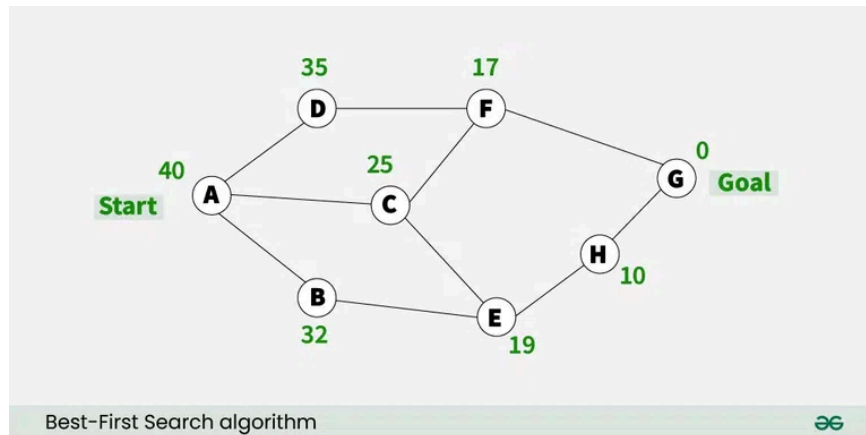
Piece hijau dapat digerakkan ke dua arah, atas dan bawah. Untuk arah atas, *piece* ini dapat digerakkan 1 satuan ke atas dan 2 satuan ke atas. Sedangkan pada arah bawah, *piece* ini dapat digerakkan 1 satuan ke bawah dan 2 satuan ke bawah. Keempat keadaan tersebut memiliki ongkos/ *cost* yang sama, misal 1. Setiap keadaan yang dapat diraih kemudian dimasukkan ke *priority queue*, sesuai dengan ongkosnya. Hal yang sama diulangi untuk setiap *piece* pada papan, untuk mencapai seluruh kemungkinan keadaan yang dapat diraih.

Pencarian dilanjutkan dengan menelusuri keadaan terdepan pada *priority queue*. Pencarian dihentikan saat keadaan yang diproses adalah *goal*, yaitu *piece* utama sudah mencapai pintu keluar. Solusi ini merupakan langkah termurah untuk mencapai tujuan.

https://www.cs.ucdavis.edu/~vemuri/classes/ecs170/blindsearches_files/blind_searches.htm#:~:text=Blind%20Searches-,Introduction,no%20information%20about%20its%20domain.

Algoritma Greedy Best First

Greedy Best First Search (GBFS) adalah algoritma pencarian berbasis heuristik yang memilih langkah yang tampak paling menjanjikan menuju solusi berdasarkan estimasi terbaik. Dalam prosesnya, GBFS selalu memilih node dengan nilai heuristik terkecil tanpa mempertimbangkan jalur sebelumnya. Tidak seperti algoritma pencarian lain seperti Uniform Cost Search (UCS) yang mempertimbangkan biaya dari awal hingga tujuan, GBFS hanya berfokus pada perkiraan jarak ke solusi, yang membuatnya dapat menemukan solusi awal dengan cepat tetapi tidak selalu menjamin hasil optimal.

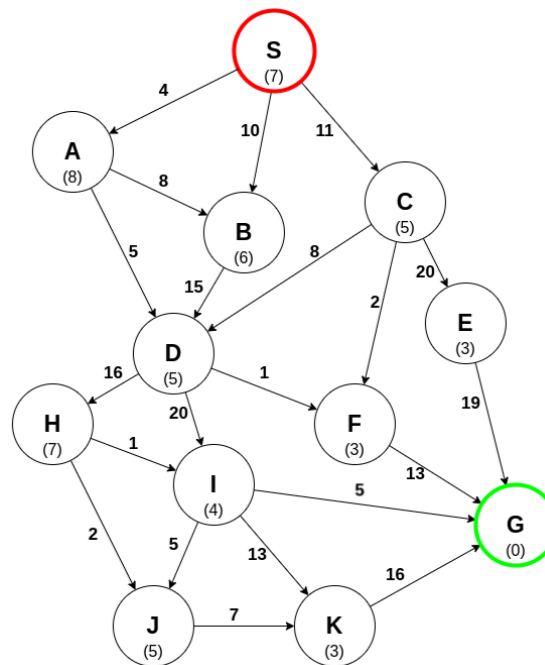


(sumber: <https://www.geeksforgeeks.org/greedy-best-first-search-algorithm/>)

Pada kasus seperti gambar di atas, fungsi heuristik menentukan seberapa baik suatu node untuk mencapai target. Hasil pencarian dengan algoritma GBFS akan memberikan hasil rute A - C - F - G dengan total heuristik $40 + 25 + 17 + 0 = 82$. Rute tersebut dipilih berdasarkan optimal lokal sehingga tidak terjamin sebagai solusi paling optimal.

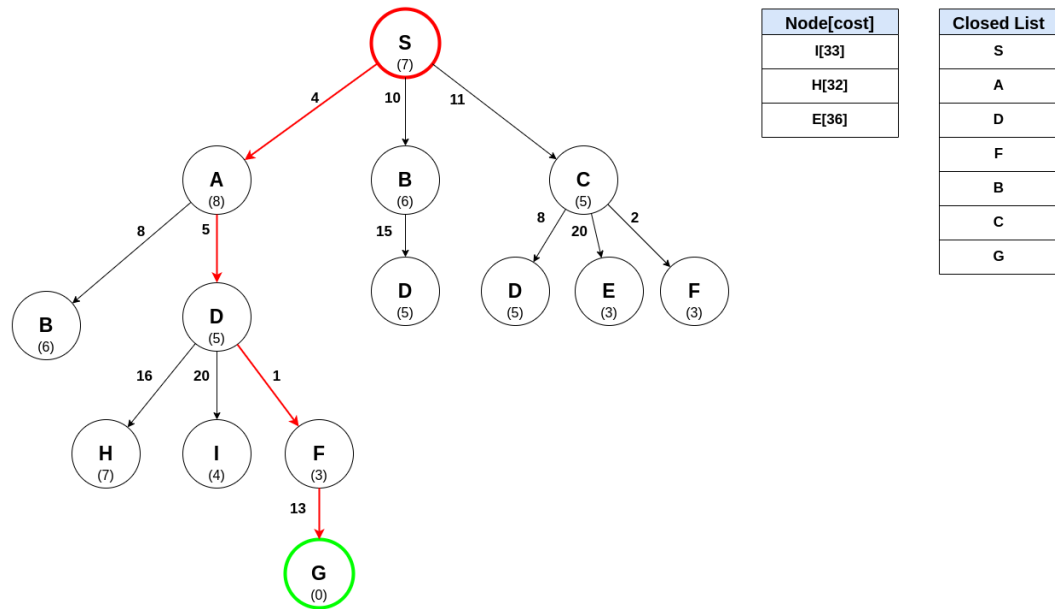
Algoritma A*

A* (A-Star) adalah algoritma pencarian yang mirip dengan algoritma Dijkstra tetapi dengan tambahan pemanfaatan heuristik. Pencarian ini menggabungkan dua pendekatan utama: pencarian berdasarkan biaya total dan pencarian berbasis heuristik. A* bekerja dengan mengevaluasi setiap kemungkinan langkah menggunakan fungsi $f(n) = g(n) + h(n)$, di mana $g(n)$ adalah biaya perjalanan dari titik awal ke posisi saat ini, dan $h(n)$ adalah estimasi biaya menuju tujuan. Dengan mempertimbangkan kedua faktor ini, A* tidak hanya mencari jalur terpendek tetapi juga memastikan eksplorasi yang lebih cerdas dan efisien dibandingkan algoritma lain seperti Uniform Cost Search (UCS) atau Greedy Best First Search (GBFS).



(sumber: <https://www.codecademy.com/resources/docs/ai/search-algorithms/a-star-search>)

Dengan pendekatan A* akan dievaluasi setiap langkah menggunakan fungsi $f(n)$. Kemudian algoritma ini menggunakan priority queue untuk selalu memilih node dengan nilai $f(n)$ terkecil sehingga didapatkan hasil seperti di bawah ini.



(sumber: <https://www.codecademy.com/resources/docs/ai/search-algorithms/a-star-search>)

BAB II

ANALISIS DAN IMPLEMENTASI ALGORITMA

Algoritma Uniform Cost Search

UCS merupakan suatu algoritma *blind search* yang memperhatikan ongkos pada setiap pencarian. Maka $g(n)$ adalah total ongkos yang dibutuhkan dari kondisi awal hingga kondisi ke- n . Sedangkan $f(n)$ adalah total ongkos yang dibutuhkan dari kondisi awal hingga kondisi ke- n dengan kondisi ke- n adalah tujuan.

Pada UCS yang digunakan pada gim Rush Hour, ongkos yang dibutuhkan untuk mencapai *node* anak selalu sama, yaitu 1. Artinya, nilai $g(n)$ pada setiap langkah UCS selalu sama dengan kedalaman pohon. Oleh karena itu, jalur/ *node* yang dibangkitkan oleh algoritma UCS sama dengan jalur yang dibangkitkan oleh algoritma BFS. Dengan nilai ongkos yang seragam, dapat dikatakan bahwa algoritma UCS sama dengan algoritma BFS.

Berikut adalah langkah - langkah algoritma UCS pada gim Rush Hour:

1. Program menerima papan permainan awal.
2. Program membuat sebuah *priority queue* yang dapat menyimpan kondisi/ *state* permainan. Antrian ini diurutkan berdasarkan ongkos yang diperlukan mencapai kondisi tersebut dari kondisi papan awal. Antrian dimulai dengan memasukkan kondisi awal papan.
3. Program memeriksa kondisi papan paling depan pada antrian. Lalu, program mulai memproses setiap *piece* yang ada pada papan. Pada tiap *piece*, program akan mencoba menggeser *piece* tersebut ke seluruh arah yang mungkin, dengan jarak pergeseran sebanyak jumlah kotak yang mungkin. Program akan membuat sebuah *state* untuk tiap gerakan valid dan memasukkannya ke dalam antrian, jika kondisi papan tersebut belum pernah dicapai sebelumnya. Untuk setiap gerakan, ongkos hanya bertambah sebesar 1.
4. Ulangi proses nomor 3 hingga kondisi terdepan pada antrian merupakan kondisi tujuan, yakni *piece* utama sudah mencapai pintu keluar atau antrian menjadi kosong. Jika mencapai tujuan, maka program menemukan solusi. Sedangkan jika antrian kosong, tidak terdapat solusi yang mungkin dari kondisi awal papan.

Pada kondisi terburuk, UCS akan menelusuri setiap langkah yang mungkin pada sebuah permainan. Jika b adalah *branching factor* (jumlah gerakan yang mungkin pada suatu saat), maka pada kondisi terburuk terdapat b^k kondisi yang perlu diperiksa. Maka, kompleksitas ruang algoritma UCS adalah $O(b^k)$.

Algoritma ini menggunakan *priority queue* untuk menyimpan dan mengeluarkan *node*/ kondisi yang diproses. Diketahui bahwa kompleksitas waktu *insert* pada *priority queue* adalah $O(\log n)$. Jika UCS memeriksa seluruh kemungkinan permainan, maka *priority queue* akan berkembang hingga b^k . Oleh karena itu, kompleksitas waktu algoritma UCS adalah $O(b^k \log n)$.

Algoritma Greedy Best First

Dalam penyelesaian permainan Rush Hour, algoritma Greedy Best First Search (GBFS) digunakan untuk menemukan jalur tercepat menuju solusi berdasarkan estimasi heuristik. GBFS tidak mempertimbangkan jalur sebelumnya atau biaya total langkah yang telah diambil, tetapi hanya berfokus pada pilihan yang tampak paling dekat dengan tujuan.

Berikut adalah langkah-langkah algoritma yang diimplementasikan:

1. Program menerima kondisi awal permainan Rush Hour dalam bentuk grid, yang merepresentasikan posisi piece.
2. Program membuat priority queue untuk mengurutkan keadaan berdasarkan nilai heuristik, di mana node dengan nilai $h(n)$ terkecil akan diproses lebih dahulu.
3. Program membuat set visited yang digunakan untuk melacak keadaan yang sudah pernah dieksplorasi guna menghindari siklus berulang.
4. Program membuat list gerakan untuk menyimpan daftar langkah piece menuju solusi.
5. Keadaan awal permainan dimasukkan ke dalam priority queue dengan nilai $h(n)$ sesuai estimasi ke tujuan.
6. Program mengecek jika piece utama mencapai pintu keluar, algoritma berhenti dan mengembalikan daftar gerakan sebagai solusi.
7. Setiap piece yang bisa bergerak diperiksa, dan setiap kemungkinan langkah dibuat sebagai state baru.
8. Memasukkan Keadaan Baru ke Priority Queue: Semua state yang belum pernah dikunjungi ditambahkan ke priority queue, diurutkan berdasarkan estimasi heuristik.
9. Mengulang Hingga Solusi Ditemukan: Program akan terus memproses node dengan nilai heuristik terkecil hingga priority queue kosong atau solusi ditemukan.

Dalam permainan Rush Hour, GBFS bekerja dengan hanya mempertimbangkan $h(n)$ sebagai estimasi jarak euclidean menuju pintu keluar ditambah dengan jumlah blok yang menghalangi jalur ke pintu keluar, tanpa memperhitungkan biaya total langkah yang telah diambil. Pendekatan ini membuat algoritma lebih cepat dalam menemukan solusi awal dibandingkan Uniform Cost Search (UCS) atau A^* , tetapi memiliki kelemahan yaitu dapat terjebak dalam local minimal. Algoritma dapat memilih langkah yang tampak terbaik saat itu tetapi sebenarnya kurang optimal secara keseluruhan.

Algoritma A^*

A^* adalah pengembangan dari UCS dan GBFS. Algoritma ini memadukan biaya sebenarnya dengan perkiraan heuristik, sehingga mampu mencari jalur optimal dengan efisiensi tinggi. A^* menggunakan fungsi evaluasi berikut:

$$f(n) = g(n) + h(n)$$

Dengan:

- $g(n)$ adalah biaya dari awal sampai node saat ini.
- $h(n)$ adalah perkiraan biaya dari node saat ini ke tujuan (heuristik).
- $f(n)$ adalah total biaya estimasi untuk mencapai tujuan.

Berikut adalah langkah-langkah algoritma A* yang diimplementasikan untuk menyelesaikan permainan Rush Hour:

1. Program menerima papan permainan awal yang terdiri dari kendaraan dan rintangan dalam grid.
2. Program membuat priority queue yang diurutkan berdasarkan nilai $f(n)$, yaitu hasil penjumlahan $g(n)$ (jumlah langkah dari awal) dan $h(n)$ (perkiraan menuju solusi).
3. Program membuat map gScore untuk menyimpan nilai $g(n)$ dari setiap keadaan permainan.
4. Program memasukkan keadaan awal ke dalam priority queue dengan nilai $f(n)$ awal.
5. Selama priority queue tidak kosong, keadaan dengan nilai $f(n)$ paling rendah akan diambil dan diproses.
6. Jika kendaraan utama sudah mencapai pintu keluar, program berhenti dan mengembalikan daftar langkah sebagai solusi.
7. Jika belum, program memperbarui gScore dan menambahkan semua kemungkinan pergerakan kendaraan ke priority queue, masing-masing dengan nilai $f(n)$ yang diperbarui.
8. Program mengulangi langkah 5 hingga 7 sampai solusi ditemukan atau priority queue kosong.

Secara teoritis, algoritma A* lebih efisien dibandingkan Uniform Cost Search (UCS) dalam menyelesaikan permainan Rush Hour karena A* menggabungkan biaya aktual dengan perkiraan heuristik untuk menemukan solusi. Algoritma A* tidak hanya memastikan solusi dengan jumlah langkah paling sedikit seperti UCS, tetapi juga menghindari eksplorasi yang tidak perlu. Selama heuristik yang digunakan admissible, A* tetap optimal dan cenderung lebih cepat dibandingkan UCS.

Dalam program Rush Hour, heuristik yang digunakan terdiri dari jumlah blok yang menghalangi jalur keluar dan Jarak Euclidean dari piece utama ke pintu keluar.

- Pada program ini jumlah blok yang menghalangi adalah heuristik yang tidak bersifat admissible, karena tidak mempertimbangkan jika ada piece lain yang perlu dipindahkan sebelum bisa menggeser penghalang piece utama.
- Pada program ini jarak Euclidean bersifat admissible, karena dalam permainan Rush Hour, piece bergerak secara horizontal dan vertikal saja dengan pintu keluar selalu sesuai orientasi arah piece utama sehingga jarak dalam konteks ini tidak akan melebihi-lebihkan estimasi biaya sebenarnya.

BAB III

SOURCE PROGRAM

Program utama ditulis dalam bahasa Java dengan memakai *packages* di antaranya:

1. Java.util
2. Java.io
3. Java.awt
4. Javax.swing

Program ini terdiri atas 14 *files* yang bersesuaian dengan kelas nya dengan satu *file* sebagai Main driver dari programnya. *Files* utama pada program ini adalah sebagai berikut

- Main.java
- UCS.java
- GreedyBestFirstSearch.java
- AStarSearch.java

Kelas Main

```
package Tucil3_13523025_13523030.src;

import java.util.List;
import java.util.Scanner;

public class Main {
    Run | Debug
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println(x:"Selamat datang di permainan Rush Hour!");
        System.out.printf(format:"Silakan masukkan nama file papan (e.g: papan1): ");

        String fileName = scanner.nextLine();
        String filePath = "../data/" + fileName + ".txt";
        System.out.println(filePath);

        Papan tes = Load.Load_Papan(filePath);
        Papan copy = new Papan(tes);

        System.out.println(x:"Strategi pencarian yang tersedia:");
        System.out.println(x:"1. Uniform Cost Search");
        System.out.println(x:"2. Greedy Best First Search");
        System.out.println(x:"3. A* Search");
        System.out.printf(format:"Silakan pilih strategi pencarian (1/2/3): ");

        int pilihan = scanner.nextInt();
        System.out.println();
    }
}
```

```

        if (pilihan < 1 || pilihan > 3) {
            System.out.println(x:"Pilihan tidak valid. Silakan pilih 1, 2, atau 3.");
            return;
        }

        if (pilihan == 1) {
            UCS ucs = new UCS(tes);
            List<Gerakan> steps = ucs.solve();
            GUI gui = new GUI(copy,steps);
        } else if (pilihan == 2) {
            GreedyBestFirstSearch greedy = new GreedyBestFirstSearch(tes);
            List<Gerakan> steps = greedy.solve();
            GUI gui = new GUI(tes,steps);
        } else if (pilihan == 3) {
            AStarSearch astar = new AStarSearch(tes);
            List<Gerakan> steps = astar.solve();
            GUI gui = new GUI(tes,steps);
        }
    }
}

```

Main.java

Kelas UCS

```

package Tucil3_13523025_13523030_src;

import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.PriorityQueue;

public class UCS {
    private Papan initBoard;
    public UCS(Papan initBoard) {
        this.initBoard = initBoard;
    }

    public List<Gerakan> solve(){
        System.out.println(x:"Starting UCS class ");
        PriorityQueue<State> pq = new PriorityQueue<State>();
        ArrayList<Gerakan> step = new ArrayList<>();
        HashSet<String> visited = new HashSet<>();

        Papan papanCopy = new Papan(initBoard);

        State current_state = new State(papanCopy, step, skor:0);
        pq.add(current_state);
    }
}

```

```

while (!pq.isEmpty()) {
    current_state = pq.poll();

    papanCopy = new Papan(current_state.getPapan());

    if (visited.contains(papanCopy.hashString())){
        continue;
    }
    else{
        visited.add(papanCopy.hashString());
    }

    if (isAtExit2(papanCopy, papanCopy.getPrimaryPiece(), papanCopy.getExitRow(), papanCopy.getExitCol())){
        System.out.println(x:"JAWABAN:");
        initBoard.displayBoard();
        System.out.println(x:"");

        for (Gerakan g : current_state.getGerakan()){
            initBoard.movePiece(g);
            initBoard.displayBoard();
            System.out.println(x:"");
        }
        System.out.println(x:"Exit from loop");
        System.out.printf(format:"Baris: %d ", papanCopy.getPrimaryPiece().getBaris());
        System.out.printf(format:"Kolom: %d\n", papanCopy.getPrimaryPiece().getKolom());
        return current_state.getGerakan();
    }

    ArrayList<Piece> current_pieces = new ArrayList<>();
    for (Piece temp : papanCopy.getPieces()){
        current_pieces.add(new Piece(temp));
    }

    for (Piece p : current_pieces){
        if (p.isHorizontal()){
            for (int steps = 1; steps <= papanCopy.getKolom(); steps++) {

```

```

                Gerakan move = new Gerakan(new Piece(p), arah:"kanan", steps);
                Papan tempBoard = new Papan(current_state.getPapan());

                if (tempBoard.movePiece(move)) {

                    ArrayList<Gerakan> newSteps = new ArrayList<>(current_state.getGerakan());
                    newSteps.add(move);

                    State newState = new State(tempBoard, newSteps, current_state.getSkor() + 1);
                    if (!visited.contains(tempBoard.hashString())) {
                        pq.add(newState);
                    }
                } else {
                    break;
                }
            }
        }

        for (int steps = 1; steps <= papanCopy.getKolom(); steps++) {
            Gerakan move = new Gerakan(new Piece(p), arah:"kiri", steps);
            Papan tempBoard = new Papan(current_state.getPapan());

            if (tempBoard.movePiece(move)) {
                ArrayList<Gerakan> newSteps = new ArrayList<>(current_state.getGerakan());
                newSteps.add(move);

                State newState = new State(tempBoard, newSteps, current_state.getSkor() + 1);

```

```

        if (!visited.contains(tempBoard.hashString())) {
            pq.add(newState);
        }
    } else {
        break;
    }
}

}else{
    for (int steps = 1; steps <= papanCopy.getBaris(); steps++) {
        Gerakan move = new Gerakan(new Piece(p), arah:"atas", steps);
        Papan tempBoard = new Papan(current_state.getPapan());

        if (tempBoard.movePiece(move)) {
            ArrayList<Gerakan> newSteps = new ArrayList<>(current_state.getGerakan());
            newSteps.add(move);

            State newState = new State(tempBoard, newSteps, current_state.getSkor() + 1);
            if (!visited.contains(tempBoard.hashString())) {
                pq.add(newState);
            }
        } else {
            break;
        }
    }

    for (int steps = 1; steps <= papanCopy.getBaris(); steps++) {
        Gerakan move = new Gerakan(new Piece(p), arah:"bawah", steps);
        Papan tempBoard = new Papan(current_state.getPapan());

        if (tempBoard.movePiece(move)) {
            ArrayList<Gerakan> newSteps = new ArrayList<>(current_state.getGerakan());
            newSteps.add(move);

            State newState = new State(tempBoard, newSteps, current_state.getSkor() + 1);

```

```

            if (!visited.contains(tempBoard.hashString())) {
                pq.add(newState);
            }
        } else {
            break;
        }
    }
}

}

System.out.println(x:"no solution");
return null;
}

```

```

private boolean isAtExit2(Papan papan, Piece primary, int exitRow, int exitCol) {
    int startRow = primary.getBaris();
    int startCol = primary.getKolom();
    int length = primary.getPanjang();

    if (primary.isHorizontal()) {
        // Check if primary piece covers exit column on the same row
        return (startRow == exitRow && startCol <= exitCol && startCol + length - 1 >= exitCol);
    } else {
        // Check if primary piece covers exit row on the same column
        return (startCol == exitCol && startRow <= exitRow && startRow + length - 1 >= exitRow);
    }
}
}

```

UCS.java

Kelas GreedyBestFirstSearch

```

package Tucil3_13523025_13523030.src;

import java.util.*;

public class GreedyBestFirstSearch {
    private Papan initialState;
    private HashMap<String, Boolean> visited;
    private PriorityQueue<Node> priorityQueue;
    private int expandedNodes;
    private Heuristic heuristic;
    private int maxIterations = 100000;

    public GreedyBestFirstSearch(Papan initialState) {
        this.initialState = initialState;
        this.visited = new HashMap<>();
        this.expandedNodes = 0;
        this.heuristic = new BlockingDistanceHeuristic();
        this.priorityQueue = new PriorityQueue<>(Comparator.comparingInt(Node::getHeuristic));

        System.out.println(x:"Greedy Best First Search");
        System.out.println(x:"Papan awal: ");
        initialState.displayBoard();

        if (!initialState.isPapanValid()) {
            System.out.println(x:"Papan tidak valid. Silakan periksa input.");
            return;
        }
    }
}

```

```

public List<Gerakan> solve() {
    Node startNode = new Node(initialState, parent:null, action:null, heuristic.calculate(initialState));
    priorityQueue.add(startNode);

    visited.put(initialState.hashString(), value:true);

    while (!priorityQueue.isEmpty() && expandedNodes < maxIterations) {
        Node currentNode = priorityQueue.poll();
        expandedNodes++;

        if (isSolved(currentNode.getState())) {
            List<Gerakan> solution = reconstructPath(currentNode);
            return solution;
        }

        List<Node> successors = generateSuccessors(currentNode);

        for (Node successor : successors) {
            String stateHash = successor.getState().hashString();

            if (!visited.containsKey(stateHash)) {
                visited.put(stateHash, value:true);
                priorityQueue.add(successor);
            }
        }
    }

    if (expandedNodes >= maxIterations) {
        System.out.println("Reached maximum iterations (" + maxIterations + "). Search terminated.");
        return null;
    } else {
        System.out.println("Solusi tidak ditemukan.");
        return null;
    }
}

```

```

private List<Node> generateSuccessors(Node node) {
    List<Node> successors = new ArrayList<>();
    Papan currentState = node.getState();
    int boardSize = currentState.getBaris();

    for (Piece piece : currentState.getPieces()) {
        char pieceName = piece.getNama();
        int row = piece.getBaris();
        int col = piece.getKolom();
        int length = piece.getPanjang();
        boolean isHorizontal = piece.isHorizontal();
        char[][] board = currentState.getPapan();

        if (isHorizontal) {
            int steps = 1;
            if (col - steps >= 0 && board[row][col - steps] == '.') {
                Papan newState = new Papan(currentState);

                Piece newPiece = newState.getPieceByName(pieceName);
                if (newPiece == null) {
                    System.out.println("Error: Piece " + pieceName + " tidak ditemukan di state.");
                    continue;
                }
            }
        }
    }
}

```



```

Gerakan move = new Gerakan(newPiece, arah:"kiri", steps);
boolean isValid = newState.movePiece(move);

if (isValid) {
    String stateHash = newState.hashString();
    if (!visited.containsKey(stateHash)) {
        int heuristicValue = heuristic.calculate(newState);
        Node successor = new Node(newState, node, move, heuristicValue);
        successors.add(successor);
    }
}

int rightEnd = col + length - 1;
steps = 1;
if (rightEnd + steps < currentState.getKolom() &&
    (board[row][rightEnd + steps] == '.' ||
    (piece.isUtama() && row == currentState.getExitRow() && rightEnd + steps == currentState.getExitCol())) {

    Papan newState = new Papan(currentState);

    Piece newPiece = newState.getPieceByName(pieceName);
    if (newPiece == null) {
        System.out.println("Error: Piece " + pieceName + " tidak ditemukan di state.");
        continue;
    }

    Gerakan move = new Gerakan(newPiece, arah:"kanan", steps);
    boolean isValid = newState.movePiece(move);

    if (isValid) {
        String stateHash = newState.hashString();
        if (!visited.containsKey(stateHash)) {
            int heuristicValue = heuristic.calculate(newState);
            Node successor = new Node(newState, node, move, heuristicValue);
            successors.add(successor);
        }
    }
}

```

```

}
} else {

    int steps = 1;
    if (row - steps >= 0 && board[row - steps][col] == '.') {
        Papan newState = new Papan(currentState);

        Piece newPiece = newState.getPieceByName(pieceName);
        if (newPiece == null) {
            System.out.println("Error: Piece " + pieceName + " tidak ditemukan di state.");
            continue;
        }

        Gerakan move = new Gerakan(newPiece, arah:"atas", steps);
        boolean isValid = newState.movePiece(move);

        if (isValid) {
            String stateHash = newState.hashString();
            if (!visited.containsKey(stateHash)) {
                int heuristicValue = heuristic.calculate(newState);
                Node successor = new Node(newState, node, move, heuristicValue);
                successors.add(successor);
            }
        }
    }

    int bottomEnd = row + length - 1;
    steps = 1;
    if (bottomEnd + steps < currentState.getBaris() &&
        (board[bottomEnd + steps][col] == '.' ||
        (piece.isUtama() && bottomEnd + steps == currentState.getExitRow() && col == currentState.getExitCol())) {

        Papan newState = new Papan(currentState);
    }
}
}

```

```

        Piece newPiece = newState.getPieceByName(pieceName);
        if (newPiece == null) {
            System.out.println("Error: Piece " + pieceName + " tidak ditemukan di state.");
            continue;
        }

        Gerakan move = new Gerakan(newPiece, arah:"bawah", steps);
        boolean isValid = newState.movePiece(move);

        if (isValid) {
            String stateHash = newState.hashString();
            if (!visited.containsKey(stateHash)) {
                int heuristicValue = heuristic.calculate(newState);
                Node successor = new Node(newState, node, move, heuristicValue);
                successors.add(successor);
            }
        }
    }
}

return successors;
}

private boolean isSolved(Papan state) {
    Piece primaryPiece = state.getPrimaryPiece();
    if (primaryPiece == null) {
        return false;
    }

    int exitRow = state.getExitRow();
    int exitCol = state.getExitCol();

    if (primaryPiece.isHorizontal()) {
        int pieceRow = primaryPiece.getBaris();
        int pieceCol = primaryPiece.getKolom();

```

```

        if (pieceCol == exitCol && pieceRow == exitRow) {
            return true;
        }

        if (pieceCol < exitCol) {
            int pieceRightCol = pieceCol + primaryPiece.getPanjang() - 1;

            if (pieceRightCol >= exitCol) {
                return pieceRow == exitRow;
            }
        } else {
            int pieceLeftCol = pieceCol;

            if (pieceLeftCol <= exitCol) {
                return pieceRow == exitRow;
            }
        }
    }
}

```

```

    else {
        int pieceCol = primaryPiece.getKolom();
        int pieceRow = primaryPiece.getBaris();

        if (pieceCol == exitCol && pieceRow == exitRow) {
            return true;
        }

        if (pieceRow < exitRow) {
            int pieceTopRow = pieceRow;

            if (pieceTopRow >= exitRow) {
                return pieceCol == exitCol;
            }
        } else {
            int pieceBottomRow = pieceRow + primaryPiece.getPanjang() - 1;

            if (pieceBottomRow <= exitRow) {
                return pieceCol == exitCol;
            }
        }
    }
    return false;
}

private List<Gerakan> reconstructPath(Node node) {
    List<Gerakan> path = new ArrayList<>();

    while (node.getParent() != null) {
        path.add(index:0, node.getAction());
        node = node.getParent();
    }

    return path;
}

```

```

private void printSolution(List<Gerakan> solution, Papan initialState) {
    if (solution.isEmpty()) {
        System.out.println(x:"Puzzle sudah diselesaikan!");
        return;
    }
    System.out.println(x:"\n=== SOLUSI ===");

    Papan currentState = new Papan(initialState); // Create a deep copy to manipulate

    for (int i = 0; i < solution.size(); i++) {
        Gerakan originalMove = solution.get(i);
        Piece originalPiece = originalMove.getPiece();
        String direction = originalMove.getArah();
        int steps = originalMove.getJumlahKotak();

        Piece currentPiece = currentState.getPieceByName(originalPiece.getNama());
    }
}

```

```

        if (currentPiece == null) {
            System.out.println("Error: Piece " + originalPiece.getNama() + " not found in current state.");
            continue;
        }

        Gerakan currentMove = new Gerakan(currentPiece, direction, steps);
        System.out.println("Gerakan " + (i+1) + ": " + currentPiece.getNama() + "-" + direction);
        boolean moveSuccess = currentState.movePiece(currentMove);

        if (!moveSuccess) {
            System.out.println("Warning: Failed to apply move " + (i+1) + " in solution print.");
        }

        currentState.displayBoard();
        System.out.println();
    }
}

```

GreedyBestFirstSearch.java

Kelas AStarSearch

```

package Tucil3_13523025_13523030.src;

import java.util.*;

public class AStarSearch {
    private Papan initialState;
    private HashMap<String, Boolean> visited;
    private HashMap<String, Integer> gScore; // path cost ke setiap state
    private PriorityQueue<Node> openSet;
    private int expandedNodes;
    private Heuristic heuristic;
    private int maxIterations = 100000;

    public AStarSearch(Papan initialState) {
        this.initialState = initialState;
        this.visited = new HashMap<>();
        this.gScore = new HashMap<>();
        this.expandedNodes = 0;
        this.heuristic = new BlockingDistanceHeuristic();

        this.openSet = new PriorityQueue<>(Comparator.comparingInt((node ->
            node.getPathCost() + node.getHeuristic())));

        System.out.println(x: "A* Search");
        System.out.println(x: "Papan awal: ");
        initialState.displayBoard();
    }
}

```

```

    if (!initialState.isPapanValid()) {
        System.out.println(x:"Papan tidak valid. Silakan periksa input.");
        return;
    }
}

public List<Gerakan> solve() {
    int initialHeuristic = heuristic.calculate(initialState);

    Node startNode = new Node(initialState, parent:null, action:null, initialHeuristic);
    startNode.setPathCost(pathCost:0);

    String startStateHash = initialState.hashString();
    gScore.put(startStateHash, value:0);

    openSet.add(startNode);

    while (!openSet.isEmpty() && expandedNodes < maxIterations) {
        Node currentNode = openSet.poll();
        String currentStateHash = currentNode.getState().hashString();

        expandedNodes++;

        // Mengecek node yang sudah dikunjungi, mungkin menemukan jalur yang lebih baik
        if (visited.containsKey(currentStateHash) &&
            gScore.get(currentStateHash) <= currentNode.getPathCost()) {
            continue;
        }

        visited.put(currentStateHash, value:true);

        if (isSolved(currentNode.getState())) {
            List<Gerakan> solution = reconstructPath(currentNode);
            return solution;
        }

        List<Node> successors = generateSuccessors(currentNode);

        for (Node successor : successors) {
            String stateHash = successor.getState().hashString();
            int tentativeGScore = currentNode.getPathCost() + 1; // Each move costs 1

```

```

            if (gScore.containsKey(stateHash) && tentativeGScore >= gScore.get(stateHash)) {
                continue;
            }

            gScore.put(stateHash, tentativeGScore);
            successor.setPathCost(tentativeGScore);

            // Ditambahkan ke openSet meskipun sudah dikunjungi, mungkin menemukan jalur lebih baik
            openSet.add(successor);
        }
    }
}

```

```

    if (expandedNodes >= maxIterations) {
        System.out.println("Reached maximum iterations (" + maxIterations + "). Search terminated.");
        return null;
    } else {
        System.out.println(x: "Solusi tidak ditemukan");
        return null;
    }
}

```

```

private List<Node> generateSuccessors(Node node) {
    List<Node> successors = new ArrayList<>();
    Papan currentState = node.getState();

    for (Piece piece : currentState.getPieces()) {
        char pieceName = piece.getNama();
        int row = piece.getBaris();
        int col = piece.getKolom();
        int length = piece.getPanjang();
        boolean isHorizontal = piece.isHorizontal();
        char[][] board = currentState.getPapan();

        if (isHorizontal) {
            int steps = 1;
            if (col - steps >= 0 && board[row][col - steps] == '.') {
                Papan newState = new Papan(currentState);
                Piece newPiece = newState.getPieceByName(pieceName);

                if (newPiece == null) {
                    System.out.println("Error: Piece " + pieceName + " tidak ditemukan di state.");
                    continue;
                }

                Gerakan move = new Gerakan(newPiece, arah: "kiri", steps);
                boolean isValid = newState.movePiece(move);

                if (isValid) {
                    String stateHash = newState.hashString();
                    int heuristicValue = heuristic.calculate(newState);
                    Node successor = new Node(newState, node, move, heuristicValue);
                    successors.add(successor);
                }
            }

```

```

            int rightEnd = col + length - 1;
            steps = 1;
            if (rightEnd + steps < currentState.getKolom() &&
                (board[row][rightEnd + steps] == '.' ||
                 (piece.isUtama() && row == currentState.getExitRow() && rightEnd + steps == currentState.getExitCol())) {
                Papan newState = new Papan(currentState);
                Piece newPiece = newState.getPieceByName(pieceName);

                if (newPiece == null) {
                    System.out.println("Error: Piece " + pieceName + " tidak ditemukan di state.");
                    continue;
                }

                Gerakan move = new Gerakan(newPiece, arah: "kanan", steps);
                boolean isValid = newState.movePiece(move);
            }

```

```

        if (isValid) {
            String stateHash = newState.hashString();
            int heuristicValue = heuristic.calculate(newState);
            Node successor = new Node(newState, node, move, heuristicValue);
            successors.add(successor);
        }
    } else {

        int steps = 1;
        if (row - steps >= 0 && board[row - steps][col] == '.') {
            Papan newState = new Papan(currentState);
            Piece newPiece = newState.getPieceByName(pieceName);

            if (newPiece == null) {
                System.out.println("Error: Piece " + pieceName + " tidak ditemukan di state.");
                continue;
            }

            Gerakan move = new Gerakan(newPiece, arah:"atas", steps);
            boolean isValid = newState.movePiece(move);

            if (isValid) {
                String stateHash = newState.hashString();
                int heuristicValue = heuristic.calculate(newState);
                Node successor = new Node(newState, node, move, heuristicValue);
                successors.add(successor);
            }
        }

        int bottomEnd = row + length - 1;
        steps = 1;
        if (bottomEnd + steps < currentState.getBaris() &&
            (board[bottomEnd + steps][col] == '.' ||
             (piece.isUtama() && bottomEnd + steps == currentState.getExitRow() && col == currentState.getExitCol()))) {

            Papan newState = new Papan(currentState);
            Piece newPiece = newState.getPieceByName(pieceName);

            if (newPiece == null) {
                System.out.println("Error: Piece " + pieceName + " tidak ditemukan di state.");
            }
        }
    }
}

```

```

        continue;
    }

    Gerakan move = new Gerakan(newPiece, arah:"bawah", steps);
    boolean isValid = newState.movePiece(move);

    if (isValid) {
        String stateHash = newState.hashString();
        int heuristicValue = heuristic.calculate(newState);
        Node successor = new Node(newState, node, move, heuristicValue);
        successors.add(successor);
    }
}

return successors;
}

private boolean isSolved(Papan state) {
    Piece primaryPiece = state.getPrimaryPiece();
    if (primaryPiece == null) {
        return false;
    }

    int exitRow = state.getExitRow();
    int exitCol = state.getExitCol();

    if (primaryPiece.isHorizontal()) {
        int pieceRow = primaryPiece.getBaris();
        int pieceCol = primaryPiece.getKolom();

        if (pieceCol == exitCol && pieceRow == exitRow) {
            return true;
        }

        if (pieceCol < exitCol) {
            int pieceRightCol = pieceCol + primaryPiece.getPanjang() - 1;

            if (pieceRightCol >= exitCol) {
                return pieceRow == exitRow;
            }
        }
    }
}

```



```

    } else {
        int pieceLeftCol = pieceCol;

        if (pieceLeftCol <= exitCol) {
            return pieceRow == exitRow;
        }
    }

}

else {
    int pieceCol = primaryPiece.getKolom();
    int pieceRow = primaryPiece.getBaris();

    if (pieceCol == exitCol && pieceRow == exitRow) {
        return true;
    }

    if (pieceRow < exitRow) {
        int pieceTopRow = pieceRow;

        if (pieceTopRow >= exitRow) {
            return pieceCol == exitCol;
        }
    } else {
        int pieceBottomRow = pieceRow + primaryPiece.getPanjang() - 1;

        if (pieceBottomRow <= exitRow) {
            return pieceCol == exitCol;
        }
    }
}

return false;
}

private List<Gerakan> reconstructPath(Node node) {
    List<Gerakan> path = new ArrayList<>();

    while (node.getParent() != null) {
        path.add(index:0, node.getAction());
        node = node.getParent();
    }
}

```

```

        return path;
    }

    private void printSolution(List<Gerakan> solution, Papan initialState) {
        if (solution.isEmpty()) {
            System.out.println(x: "Puzzle sudah diselesaikan!");
            return;
        }

        System.out.println(x: "\n=== SOLUSI ===");

        Papan currentState = new Papan(initialState);

        for (int i = 0; i < solution.size(); i++) {
            Gerakan originalMove = solution.get(i);
            Piece originalPiece = originalMove.getPiece();
            String direction = originalMove.getArah();
            int steps = originalMove.getJumlahKotak();

            Piece currentPiece = currentState.getPieceByName(originalPiece.getNama());
            if (currentPiece == null) {
                System.out.println("Error: Piece " + originalPiece.getNama() + " not found in current state.");
                continue;
            }

            Gerakan currentMove = new Gerakan(currentPiece, direction, steps);

            System.out.println("\nGerakan " + (i+1) + ": " + currentPiece.getNama() + "-" + direction);

            boolean moveSuccess = currentState.movePiece(currentMove);
            if (!moveSuccess) {
                System.out.println("Warning: Failed to apply move " + (i+1) + " in solution print.");
            }

            currentState.displayBoard();
        }
    }
}

```

AStarSearch.java

BAGIAN IV

HASIL PENGUJIAN

3.1 Input file: papan1.txt

4 4
5
K
AABC
E.BC
E.DD

Hasil :

```
Selamat datang di permainan Rush Hour!  
Silakan masukkan nama file papan (e.g: papan1): papan1  
../data/papan1.txt  
=====  
Papan Puzzle  
Ukuran papan: 4 x 4  
Jumlah piece: 5  
Index 2 out of bounds for length 2  
Gagal memuat papan dari file. Pastikan file ada dan format benar.
```

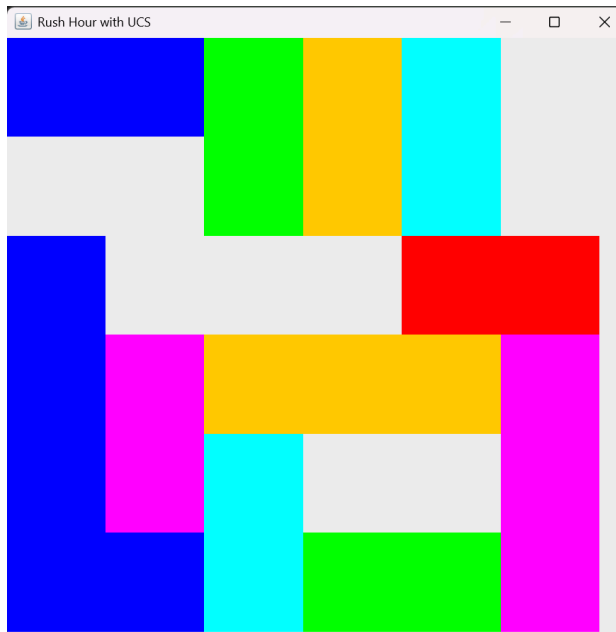
Jumlah baris pada *input file* tidak sesuai dengan konfigurasi (4).

3.2 Input file: papan2.txt

6 6
11
AAB..F
..BCDF
GPPCDFK
GH.III
GHJ...
LLJMM.

Hasil :

Algoritma Uniform Cost Search :



```
Selamat datang di permainan Rush Hour!  
Silakan masukkan nama file papan (e.g: papan1): papan2  
../data/papan2.txt
```

```
=====
```

```
Papan Puzzle  
Ukuran papan: 6 x 6  
Jumlah piece: 11  
Pintu Keluar: 2, 5  
Primary Piece: P
```

```
Papan:
```

```
AAB..F
```

```
..BCDF
```

```
GPPCDF
```

```
GH.III
```

```
GHJ...
```

```
LLJMM.
```

```
=====
```

```
Strategi pencarian yang tersedia:
```

1. Uniform Cost Search
2. Greedy Best First Search
3. A* Search

```
Silakan pilih strategi pencarian (1/2/3): 1
```

```
Starting UCS class
```

```
JAWABAN:
```

```
AAB..F
```

```
..BCDF
```

```
GPPCDF
```

```
GH.III
```

```
GHJ...
```

```
LLJMM.
```

```
Gerakan 1: D-atas
```

```
AAB.DF
```

```
..BCDF
```

```
GPPC.F
```

```
GH.III
```

```
GHJ...
```

```
LLJMM.
```

```
Gerakan 2: I-kiri
```

```
AAB.DF
```

```
..BCDF
```

```
GPPC.F
```

```
GHIII.
```

```
GHJ...
```

```
LLJMM.
```

Gerakan 3: C-atas

AABCDF

..BCDF

GPP..F

GHI..

GHJ...

LLJMM.

Gerakan 4: F-bawah

AABCD.

..BCD.

GPP...

GHIIF

GHJ..F

LLJMMF

Gerakan 5: P-kanan

AABCD.

..BCD.

G...PP

GHIIF

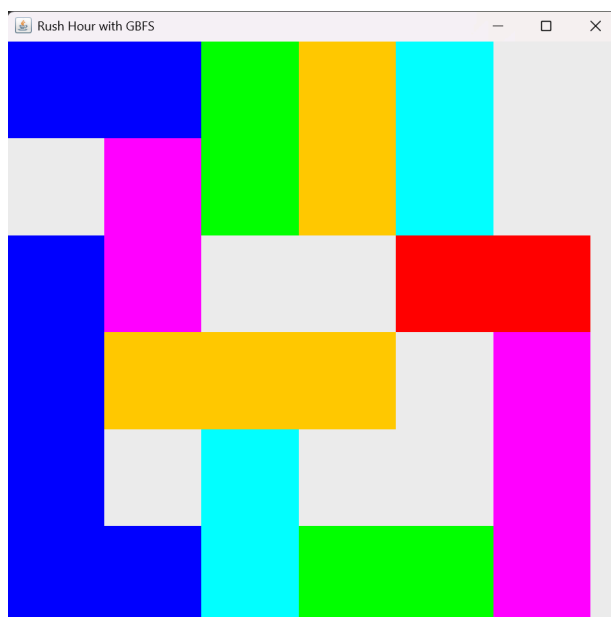
GHJ..F

LLJMMF

Banyak node diperiksa: 753

Waktu eksekusi: 252 ms

Algoritma Greedy Best First :



```
Selamat datang di permainan Rush Hour!
Silakan masukkan nama file papan (e.g: papan1): papan2
../data/papan2.txt
=====
Papan Puzzle
Ukuran papan: 6 x 6
Jumlah piece: 11
Pintu Keluar: 2, 5
Primary Piece: P

Papan:
AAB..F
..BCDF
GPPCDF
GH.III
GHJ...
LLJMM.
=====

Strategi pencarian yang tersedia:
1. Uniform Cost Search
2. Greedy Best First Search
3. A* Search
Silakan pilih strategi pencarian (1/2/3): 2

Greedy Best First Search
Papan awal:
AAB..F
..BCDF
GPPCDF
GH.III
GHJ...
LLJMM.
```

=== SOLUSI ===

Gerakan 1: C-atas

AABC.F

..BCDF

GPP.DF

GH.III

GHJ...

LLJMM.

Gerakan 2: D-atas

AABCDF

..BCDF

GPP..F

GH.III

GHJ...

LLJMM.

Gerakan 3: P-kanan

AABCDF

..BCDF

G..PP.F

GH.III

GHJ...

LLJMM.

Gerakan 4: P-kanan

AABCDF

..BCDF

G..PPF

GH.III

GHJ...

LLJMM.

Gerakan 5: H-atas

AABCDF

..BCDF

GH.PPF

GH.III

G..J...

LLJMM.

Gerakan 6: I-kiri

AABCDF

..BCDF

GH.PPF

GHIII.

G..J...

LLJMM.

Gerakan 7: F-bawah

AABCD.
..BCDF
GH.PPF
GHIIF
G.J...
LLJMM.

Gerakan 8: H-atas

AABCD.
.HBCDF
GH.PPF
G.IIIF
G.J...
LLJMM.

Gerakan 9: F-bawah

AABCD.
.HBCD.
GH.PPF
G.IIIF
G.J..F
LLJMM.

Gerakan 10: F-bawah

AABCD.
.HBCD.
GH.PP.
G.IIIF
G.J..F
LLJMMF

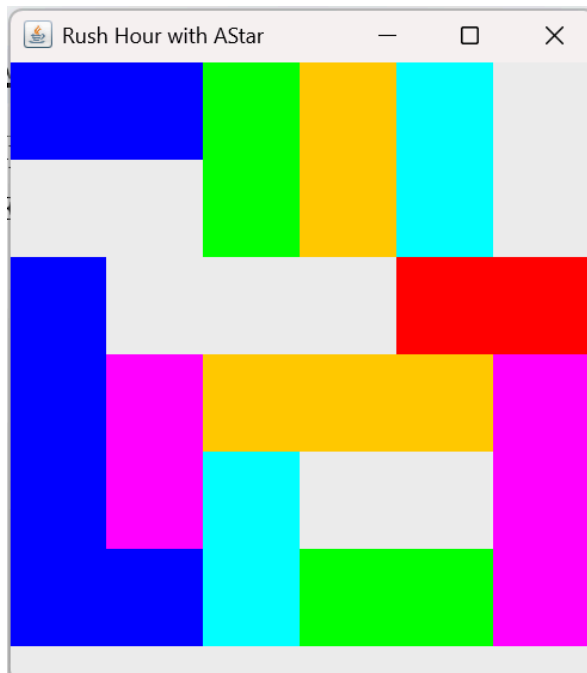
Gerakan 11: P-kanan

AABCD.
.HBCD.
GH..PP
G.IIIF
G.J..F
LLJMMF

Jumlah node yang diperiksa: 70

Waktu eksekusi: 620 ms

Algoritma A* :



```
Silakan masukkan nama file papan (e.g: papan1): papan2
../data/papan2.txt
=====
Papan Puzzle
Ukuran papan: 6 x 6
Jumlah piece: 11
Pintu Keluar: 2, 5
Primary Piece: P

Papan:
AAB..F
..BCDF
GPPCDF
GH.III
GHJ...
LLJMM.
=====

Strategi pencarian yang tersedia:
1. Uniform Cost Search
2. Greedy Best First Search
3. A* Search
Silakan pilih strategi pencarian (1/2/3): 3
```

=== SOLUSI ===

Gerakan 1: C-atas

AABC.F

..BCDF

GPP.DF

GH.III

GHJ...

LLJMM.

Gerakan 2: D-atas

AABCDF

..BCDF

GPP..F

GH.III

GHJ...

LLJMM.

Gerakan 3: P-kanan

AABCDF

..BCDF

G..PP.F

GH.III

GHJ...

LLJMM.

Gerakan 4: P-kanan

AABCDF

..BCDF

G..PPF

GH.III

GHJ...

LLJMM.

Gerakan 5: I-kiri

AABCDF

..BCDF

G..PPF

GHIII.

GHJ...

LLJMM.

Gerakan 6: F-bawah

AABCD.

..BCDF

G..PPF

GHIIF

GHJ...

LLJMM.

```

Gerakan 7: F-bawah
AABCD.
..BCD.
G..PPF
GHIIIF
GHJ..F
LLJMM.

Gerakan 8: F-bawah
AABCD.
..BCD.
G..PP.
GHIIIF
GHJ..F
LLJMMF

Gerakan 9: P-kanan
AABCD.
..BCD.
G...PP
GHIIIF
GHJ..F
LLJMMF
Jumlah node yang diperiksa: 204
Waktu eksekusi: 1038 ms

```

3.3 Input file: papan3.txt

```

6 6
11
K
AAB..F
..BCDF
GPPCDF
GH.III
GHJ...
LLJMM.

```

Hasil:

```

Selamat datang di permainan Rush Hour!
Silakan masukkan nama file papan (e.g: papan1): papan3
../data/papan3.txt
=====
Papan Puzzle
Ukuran papan: 6 x 6
Jumlah piece: 11
Pintu Keluar: 0, 0
Primary Piece: P

Papan:
AAB..F
..BCDF
GPPCDF
GH.III
GHJ...
LLJMM.
=====
Papan tidak valid. Silakan periksa file input.

```

Orientasi *piece* utama dan pintu keluar tidak sesuai.

3.4 Input file: speed.jpg

```

6 6
11
AAB...
..BCC.
PPB.D.K
II.FD.
HHHFE.
...FE.

```

Hasil :

Algoritma Uniform Cost Search:

```
Selamat datang di permainan Rush Hour!
Silakan masukkan nama file papan (e.g: papan1): papan4
../data/papan4.txt
=====
Papan Puzzle
Ukuran papan: 6 x 6
Jumlah piece: 11
Pintu Keluar: 2, 5
Primary Piece: P

Papan:
AAB...
..BCC.
PPB.D.
II.FD.
HHHFE.
...FE.
=====

Strategi pencarian yang tersedia:
1. Uniform Cost Search
2. Greedy Best First Search
3. A* Search
Silakan pilih strategi pencarian (1/2/3): 1

Starting UCS class
Solusi tidak ditemukan.
Banyak node diperiksa: 240
Tidak ada solusi ditemukan untuk input yang diberikan.
Waktu eksekusi: 213 ms
```

Algoritma Greedy Best First :

```

Selamat datang di permainan Rush Hour!
Silakan masukkan nama file papan (e.g: papan1): papan4
../data/papan4.txt
=====
Papan Puzzle
Ukuran papan: 6 x 6
Jumlah piece: 11
Pintu Keluar: 2, 5
Primary Piece: P

Papan:
AAB...
..BCC.
PPB.D.
II.FD.
HHHFE.
...FE.
=====

Strategi pencarian yang tersedia:
1. Uniform Cost Search
2. Greedy Best First Search
3. A* Search
Silakan pilih strategi pencarian (1/2/3): 2

Greedy Best First Search
Papan awal:
AAB...
..BCC.
PPB.D.
II.FD.
HHHFE.
...FE.
Solusi tidak ditemukan.

```

Algoritma A* Search:

```
Selamat datang di permainan Rush Hour!
Silakan masukkan nama file papan (e.g: papan1): papan4
../data/papan4.txt
=====
Papan Puzzle
Ukuran papan: 6 x 6
Jumlah piece: 11
Pintu Keluar: 2, 5
Primary Piece: P

Papan:
AAB...
..BCC.
PPB.D.
II.FD.
HHHFE.
...FE.
=====

Strategi pencarian yang tersedia:
1. Uniform Cost Search
2. Greedy Best First Search
3. A* Search
Silakan pilih strategi pencarian (1/2/3): 3

A* Search
Papan awal:
AAB...
..BCC.
PPB.D.
II.FD.
HHHFE.
...FE.
Solusi tidak ditemukan
Tidak ada solusi ditemukan untuk input yang diberikan.
```

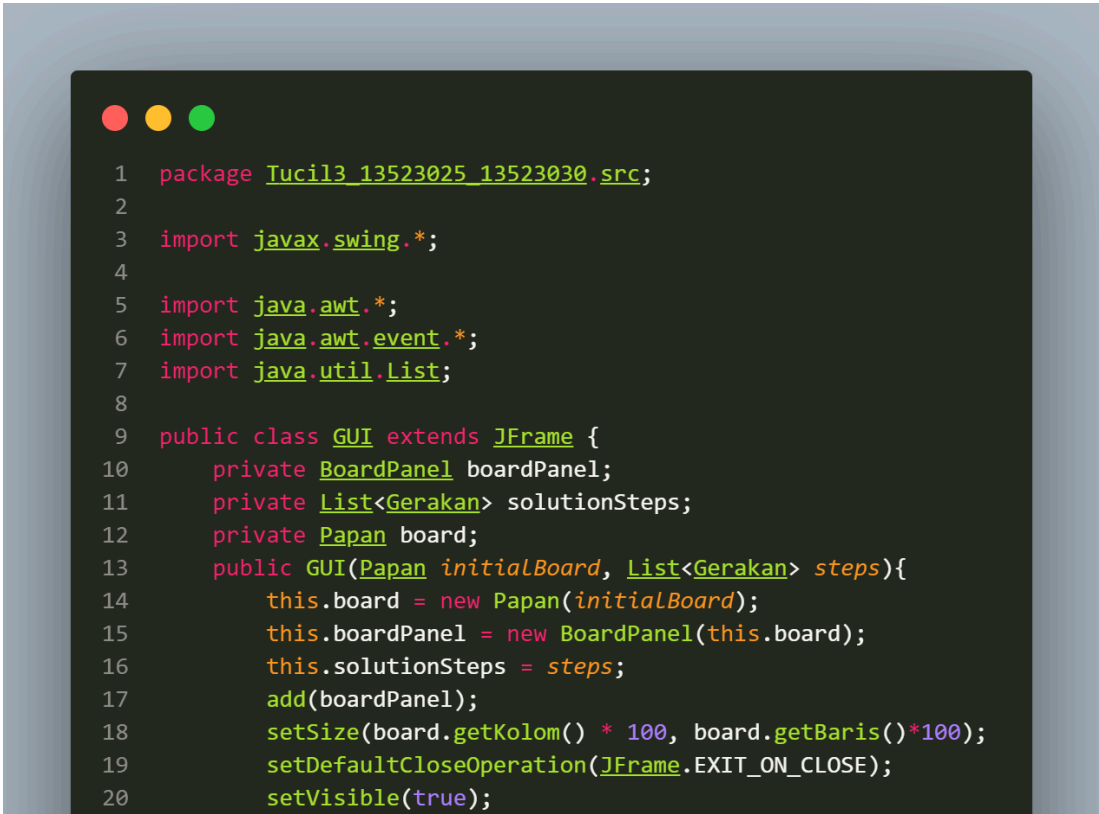
BAGIAN V

IMPLEMENTASI BONUS

Program Rush Hour Solver memiliki fitur GUI menggunakan AWT, yang memungkinkan pengguna melihat bagaimana algoritma pencarian seperti UCS, Greedy Best First Search, dan A* bekerja dalam menyelesaikan permainan. Dengan AWT, program dapat menampilkan papan permainan secara visual dan memperbarui posisi kendaraan berdasarkan daftar gerakan yang dihasilkan oleh algoritma.

Setelah algoritma menemukan solusi, setiap langkah ditampilkan dalam GUI untuk memperlihatkan bagaimana kendaraan dipindahkan hingga mobil utama mencapai pintu keluar. Fitur ini membantu pengguna memahami cara kerja algoritma dengan lebih intuitif dibandingkan hanya melihat output teks. Program dapat menyajikan simulasi pergerakan kendaraan secara langsung. Dengan visualisasi ini, pengguna dapat melihat bagaimana algoritma mencari jalur optimal untuk menyelesaikan permainan Rush Hour.

Berikut ini adalah *snapshot code* yang berkaitan dengan proses implementasi bonus GUI pada program:



```
1 package Tucil3_13523025_13523030.src;
2
3 import javax.swing.*;
4
5 import java.awt.*;
6 import java.awt.event.*;
7 import java.util.List;
8
9 public class GUI extends JFrame {
10     private BoardPanel boardPanel;
11     private List<Gerakan> solutionSteps;
12     private Papan board;
13     public GUI(Papan initialBoard, List<Gerakan> steps){
14         this.board = new Papan(initialBoard);
15         this.boardPanel = new BoardPanel(this.board);
16         this.solutionSteps = steps;
17         add(boardPanel);
18         setSize(board.getKolom() * 100, board.getBaris()*100);
19         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
20         setVisible(true);
```

```
22         setTitle("Rush Hour");
23         runSolution();
24     }
25
26     private void runSolution(){
27         Timer timer = new Timer(3000, new ActionListener() {
28             private int index = 0;
29
30             public void actionPerformed(ActionEvent e) {
31                 if (index < solutionSteps.size()) {
32                     Gerakan move = solutionSteps.get(index);
33                     board.movePiece(move);
34                     boardPanel.repaint();
35                     index++;
36                 } else {
37                     ((Timer)e.getSource()).stop();
38                 }
39             }
40         });
41         timer.start();
42     }
43 }
44
45
46
```

LINK REPOSITORY

https://github.com/jhotlann/Tucil3_13523025_13523030

TABEL CHECKLIST

No	Poin	Ya	Tidak
1	Program berhasil dikompilasi tanpa kesalahan	✓	
2	Program berhasil dijalankan	✓	
3	Solusi yang diberikan program benar dan mematuhi aturan permainan	✓	
4	Program dapat membaca masukan berkas .txt 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	✓	
9	Program dan laporan dibuat (kelompok) sendiri	✓	