

LAPORAN TUGAS KECIL 3
IF2211 STRATEGI ALGORITMA



Disusun oleh

Muhammad Ghifary Komara Putra 13523066
Sabilul Huda 13523072

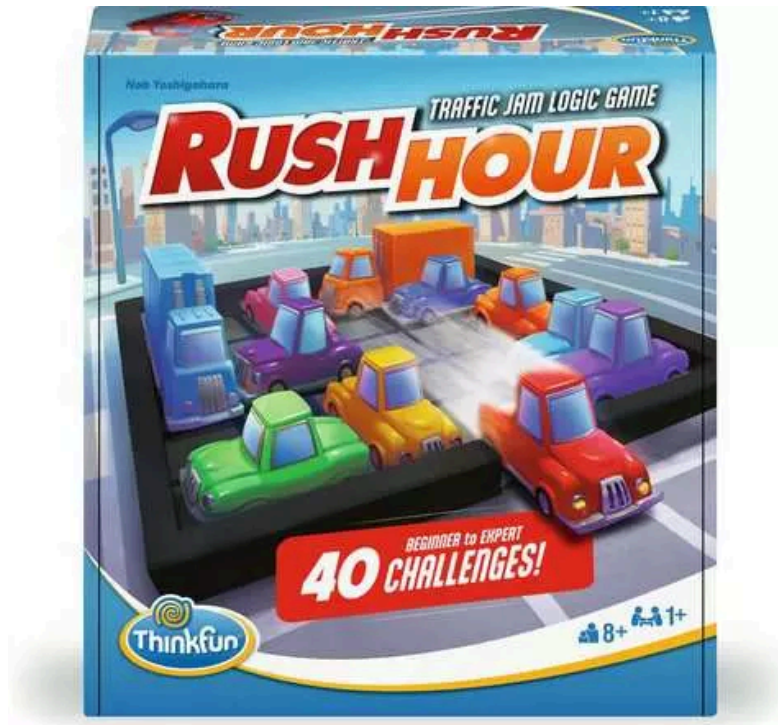
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2025

DAFTAR ISI

BAB I	2
DESKRIPSI TUGAS	2
BAB II	7
LANDASAN TEORI	7
2.1 Algoritma UCS	7
2.2 Algoritma Greedy Best First Search	7
2.3 Algoritma A*	7
BAB III	8
PENJELASAN LANGKAH-LANGKAH	8
3.1. Algoritma UCS	8
3.2 Algoritma Best First Search	8
3.3 Algoritma A*	8
BAB IV	9
ANALISIS ALGORITMA	9
4.1. Algoritma UCS	9
4.2 Algoritma Best First Search	9
4.3 Algoritma A*	9
BAB V	10
SOURCE CODE PROGRAM	10
5.1 BoardState.java	11
5.2 Coordinate.java	11
5.3 Cost.java	11
5.4 InputFile.java	11
5.5 Main.java	11
5.6 Orientation.java	11
5.7 Output.java	11
5.8 Piece.java	11
5.9 Solver.java	11
5.10 Tree.java	11
5.11 Validation.java	11
BAB VI	12
PENGUJIAN	12
BAB VII	13
KESIMPULAN DAN SARAN	13
5.1 Kesimpulan	13
5.2 Saran	13
5.3 Refleksi	13
LAMPIRAN	14
DAFTAR PUSTAKA	15

BAB I

DESKRIPSI TUGAS



Gambar 1. Rush Hour Puzzle

(Sumber: <https://www.thinkfun.com/en-US/products/educational-games/rush-hour-76582>)

Rush Hour adalah sebuah permainan puzzle logika berbasis grid yang menantang pemain untuk menggeser kendaraan di dalam sebuah kotak (biasanya berukuran 6x6) agar mobil utama (biasanya 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 *singular point* dari papan. Sebuah *piece* akan menempati *cell-cell* pada papan. Ketika permainan dimulai, semua *piece* telah diletakkan di dalam papan dengan konfigurasi tertentu berupa lokasi *piece* dan *orientasi*, antara *horizontal* atau *vertikal*.

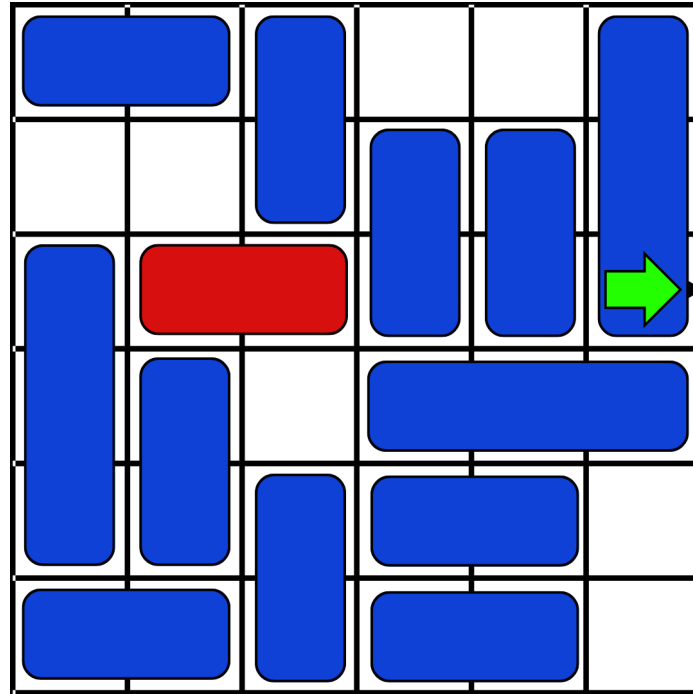
Hanya **primary piece** yang dapat digerakkan **keluar papan melewati pintu keluar**. *Piece*

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*.
4. **Pintu Keluar** – *Pintu keluar* adalah tempat *primary piece* dapat digerakkan keluar untuk menyelesaikan permainan
5. **Gerakan** — *Gerakan* yang dimaksudkan 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.

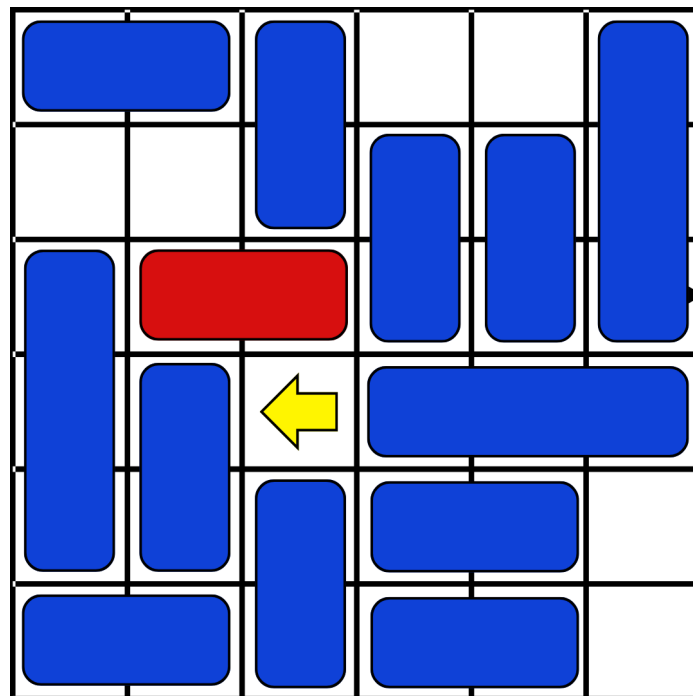
Ilustrasi kasus :

Diberikan sebuah *papan* berukuran 6 x 6 dengan 12 *piece* kendaraan dengan 1 *piece* merupakan *primary piece*. *Piece* ditempatkan pada *papan* dengan posisi dan orientasi sebagai berikut.

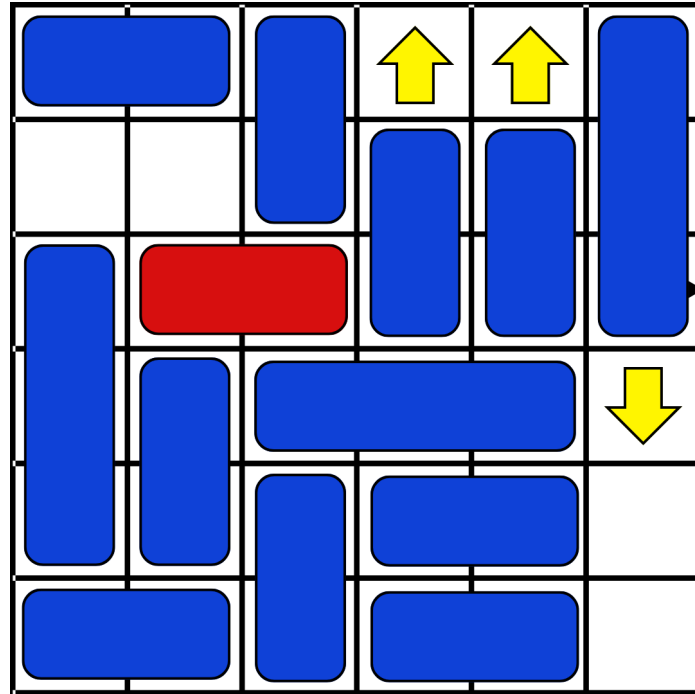


Gambar 2. Awal Permainan Game Rush Hour

Pemain dapat menggeser-geser *piece* (termasuk *primary piece*) untuk membentuk jalan lurus antara *primary piece* dan *pintu keluar*.

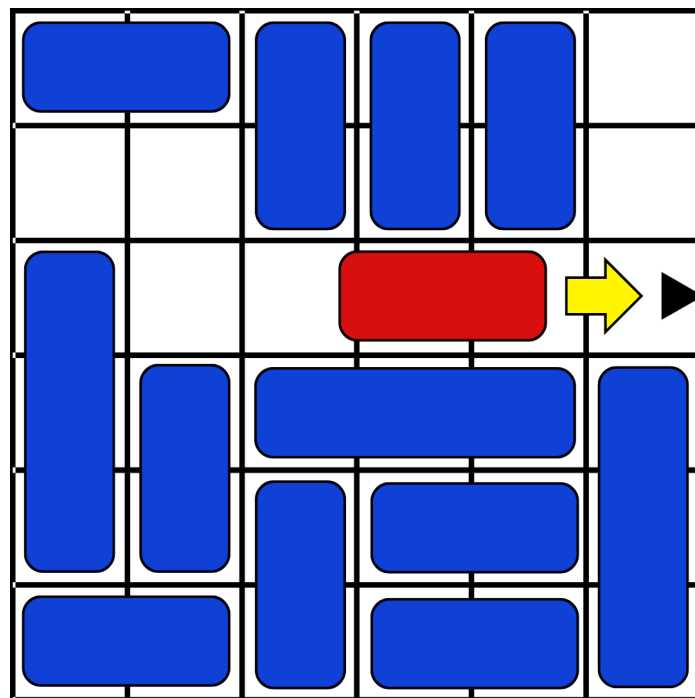


Gambar 3. Gerakan Pertama Game Rush Hour




Gambar 4. Gerakan Kedua Game Rush Hour

Puzzle berikut dinyatakan telah selesai apabila *primary piece* dapat digeser keluar papan melalui *pintu keluar*.



Gambar 5. Pemain Menyelesaikan Permainan

Agar lebih jelas, silahkan amati video cara bermain berikut:

 The New Rush Hour by ThinkFun!

Anda juga dapat melihat gif berikut untuk melihat contoh permainan [Rush Hour Solution](#).

BAB II

LANDASAN TEORI

2.1 Algoritma UCS

Uniform Cost Search (UCS) adalah salah satu algoritma pencarian graf berbasis biaya yang termasuk dalam kategori uninformed search (pencarian tanpa informasi heuristik). UCS bekerja dengan memilih node yang memiliki biaya total dari simpul awal (start node) paling kecil. Artinya, prioritas eksplorasi diberikan pada jalur yang sejauh ini memiliki biaya terendah, tanpa mempertimbangkan seberapa dekat simpul tersebut ke tujuan.

UCS sangat mirip dengan algoritma Dijkstra dan menjamin menemukan solusi optimal jika semua biaya lintasan tidak negatif. Algoritma ini menggunakan priority queue untuk menyimpan node-node yang akan dieksplorasi, diurutkan berdasarkan total biaya kumulatif dari awal. UCS akan berhenti ketika node tujuan di-expand untuk pertama kalinya, karena pada saat itu, ia sudah merupakan lintasan dengan biaya minimum.

2.2 Algoritma Greedy Best First Search

Greedy Best First Search adalah algoritma pencarian yang termasuk dalam informed search, yang artinya algoritma ini menggunakan informasi tambahan berupa fungsi heuristik untuk memperkirakan jarak dari suatu node ke tujuan. Pada setiap langkah, algoritma ini memilih node yang memiliki nilai heuristik terkecil, yaitu node yang tampak paling dekat dengan tujuan menurut estimasi.

Ciri utama dari Greedy BFS adalah hanya mempertimbangkan nilai heuristik $h(n)$ dan mengabaikan biaya dari simpul awal ke node saat ini. Hal ini membuat algoritma ini lebih cepat dalam beberapa kasus, namun tidak menjamin solusi optimal, karena bisa jadi memilih jalur yang tampaknya dekat tetapi sebenarnya lebih mahal secara total.

2.3 Algoritma A*

Algoritma A* (dibaca: A star) adalah algoritma pencarian heuristik yang menggabungkan keunggulan dari UCS dan Greedy BFS. A* menggunakan fungsi evaluasi:

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

di mana:

$g(n)$ adalah biaya dari simpul awal ke simpul n ,

$h(n)$ adalah estimasi biaya dari n ke tujuan (heuristik).

Dengan mempertimbangkan kedua komponen ini, A* dapat mengevaluasi node berdasarkan total estimasi biaya dari awal hingga ke tujuan melalui simpul n . Jika fungsi heuristik $h(n)$ adil dan konsisten (admissible and consistent), maka A* menjamin solusi yang optimal dan efisien. A* sangat banyak digunakan dalam berbagai aplikasi seperti pencarian rute (pathfinding), perencanaan robotik, dan permainan, karena kombinasi kecepatan dan jaminan optimalitasnya.

BAB III PENJELASAN ALGORITMA

4.1. Algoritma UCS

Solusi dengan pendekatan algoritma Uniform Cost Search (UCS) yang telah dikembangkan adalah sebagai berikut:

1. Buatlah sebuah node untuk kondisi papan pada awal permainan, hitung nilai $f(n) = g(n)$ untuk kondisi papan yang bersangkutan
2. Masukkan node tersebut ke dalam *priority queue*. *Priority queue* ini akan mengutamakan node dengan nilai $f(n)$ terkecil
3. Kunjungi node paling awal dalam *priority queue*, lalu hapus dari *priority queue*, tambahkan dalam suatu himpunan node yang telah dikunjungi
4. Tinjau apakah node tersebut merupakan solusi akhir. Jika iya, pencarian dihentikan
5. Jika tidak, bangkitkan seluruh anak dari node tersebut, yaitu kumpulan node yang berisi kondisi papan hasil pergerakan yang mungkin dari node tersebut dan belum dikunjungi sebelumnya. Untuk setiap node anak, hitung nilai $f(n)$, masukkan ke dalam *priority queue*
6. Ulangi langkah 3-5 hingga *priority queue* kosong atau solusi telah ditemukan
7. Jika *priority queue* kosong, tidak ditemukan solusi
8. Jika solusi ditemukan, petakan kembali jalur dari node akhir ke node awal, hilangkan node yang tidak diperlukan dalam tree
9. Cetak rute keseluruhan

4.2 Algoritma *Best First Search*

Solusi dengan pendekatan algoritma *Best First Search* yang telah dikembangkan adalah sebagai berikut:

1. Buatlah sebuah node untuk kondisi papan pada awal permainan, hitung nilai $f(n) = h(n)$ untuk kondisi papan yang bersangkutan
2. Masukkan node tersebut ke dalam *priority queue*. *Priority queue* ini akan mengutamakan node dengan nilai $f(n)$ terkecil
3. Kunjungi node paling awal dalam *priority queue*, lalu hapus dari *priority queue*, tambahkan dalam suatu himpunan node yang telah dikunjungi
4. Tinjau apakah node tersebut merupakan solusi akhir. Jika iya, pencarian dihentikan
5. Jika tidak, bangkitkan seluruh anak dari node tersebut, yaitu kumpulan node yang berisi kondisi papan hasil pergerakan yang mungkin dari node tersebut dan belum dikunjungi sebelumnya. Untuk setiap node anak, hitung nilai $f(n)$, masukkan ke dalam *priority queue*
6. Ulangi langkah 3-5 hingga *priority queue* kosong atau solusi telah ditemukan
7. Jika *priority queue* kosong, tidak ditemukan solusi
8. Jika solusi ditemukan, petakan kembali jalur dari node akhir ke node awal, hilangkan node yang tidak diperlukan dalam tree
9. Cetak rute keseluruhan

4.3 Algoritma A*

Solusi dengan pendekatan algoritma A* yang telah dikembangkan adalah sebagai berikut:

1. Buatlah sebuah node untuk kondisi papan pada awal permainan, hitung nilai $f(n) = g(n) + h(n)$ untuk kondisi papan yang bersangkutan
2. Masukkan node tersebut ke dalam *priority queue*. *Priority queue* ini akan mengutamakan node dengan nilai $f(n)$ terkecil
3. Kunjungi node paling awal dalam *priority queue*, lalu hapus dari *priority queue*, tambahkan dalam suatu himpunan node yang telah dikunjungi
4. Tinjau apakah node tersebut merupakan solusi akhir. Jika iya, pencarian dihentikan
5. Jika tidak, bangkitkan seluruh anak dari node tersebut, yaitu kumpulan node yang berisi kondisi papan hasil pergerakan yang mungkin dari node tersebut dan belum dikunjungi sebelumnya. Untuk setiap node anak, hitung nilai $f(n)$, masukkan ke dalam *priority queue*
6. Ulangi langkah 3-5 hingga *priority queue* kosong atau solusi telah ditemukan
7. Jika *priority queue* kosong, tidak ditemukan solusi
8. Jika solusi ditemukan, petakan kembali jalur dari node akhir ke node awal, hilangkan node yang tidak diperlukan dalam tree
9. Cetak rute keseluruhan

4.4 Analisis tambahan

Beberapa istilah yang relevan dalam pencarian rute dengan algoritma UCS, *Best First Search*, dan A* adalah $g(n)$, $h(n)$, dan $f(n)$. $g(n)$ merupakan suatu nilai yang menyatakan *cost* dari node awal hingga node saat ini (n). Di sisi lain, $h(n)$ merupakan suatu nilai heuristik yang menyatakan estimasi *cost* dari node saat ini (n) ke node tujuan/akhir. Lalu, $f(n)$ merupakan fungsi evaluasi yang digunakan untuk menghitung nilai yang dimiliki oleh suatu node. Nilai $f(n)$ merupakan penentu node manakah yang akan dikunjungi berikutnya dalam pencarian. Dalam program yang dikembangkan, nilai $f(n)$ yang lebih kecil akan dikunjungi terlebih dahulu. Nilai $f(n)$ untuk algoritma UCS, *Best First Search*, dan A* berturut-turut adalah $g(n)$, $h(n)$, dan $g(n) + h(n)$.

Dalam program yang dikembangkan, fungsi $h(n)$ yang dikembangkan adalah banyaknya *piece* yang menghalangi *primary piece* dalam perjalanannya menuju petak *exit*. Perhatikan bahwa heuristik ini bersifat *admissible*, yaitu bahwa *cost* hasil estimasi tidak akan pernah melebihi *cost* sebenarnya (estimasi bersifat optimis). Dalam kasus terburuk, estimasi $h(n)$ akan lebih kecil satu satuan ketimbang *cost* sebenarnya (yang dalam konteks ini adalah banyak langkah yang diperlukan agar *primary piece* keluar dari papan). Hal ini terjadi ketika untuk setiap *piece* yang menghalangi *primary piece*, *piece* tersebut dapat langsung digerakkan sedemikian hingga dapat membuka jalur bagi *primary piece*. Untuk kasus umum, estimasi ini bersifat optimis karena akan ada langkah tambahan untuk menggerakkan *piece-piece* lain yang menghalangi *piece* yang menghalangi *primary piece* tersebut.

Pada program ini, $g(n)$ didefinisikan sebagai

$$g(n) = G(1) - G(n)$$

Dengan $G(n)$ adalah fungsi pembantu yang didefinisikan sebagai jumlah *piece* yang berbeda orientasi dengan *piece* x yang berada di depan atau di belakang *piece* x , untuk setiap *piece* x anggota *piece* yang ada pada *puzzle*. Algoritma UCS berbeda dengan BFS. Pada algoritma BFS, node yang sudah dibangkitkan akan masuk ke queue normal dan dilayani secara berurutan. Sedangkan pada UCS, node yang sudah dikunjungi, akan masuk ke *priority queue* yang diurutkan berdasarkan nilai $g(n)$, yang dalam program ini $g(n)$ dengan nilai lebih tinggi akan diproses terlebih dahulu.

Sebagai algoritma yang berbeda, algoritma UCS, *Best First Search*, dan A^* memiliki keunggulan dan performa masing-masing. Secara teoritis, dalam penyelesaian *puzzle* Rush Hour, algoritma A^* dengan fungsi $h(n)$ yang *admissible* akan lebih efisien ketimbang algoritma UCS. Hal ini terjadi karena keberadaan estimasi melalui $h(n)$ akan mengakibatkan pencarian memprioritaskan langkah-langkah yang “menjanjikan” terlebih dahulu (lebih dekat ke *node* tujuan berdasarkan nilai $h(n)$ yang lebih kecil). Proses pencarian ini tentu akan jauh lebih efektif ketimbang hanya memperhitungkan *cost* dari kondisi awal ke kondisi papan saat ini, juga hanya akan mengekskan dan mengunjungi *node-node* yang diharapkan lebih diperlukan saja. Lalu, karena $h(n)$ bersifat *admissible*, pencarian tidak akan salah mendeteksi rute efektif menjadi rute dengan *cost* tinggi, membuat algoritma A^* mampu menemukan solusi optimal.

Selain itu, terdapat properti unik yang dimiliki oleh algoritma *best first search*, yaitu bahwa algoritma ini tidak menjamin solusi optimal untuk penyelesaian *puzzle* Rush Hour. Artinya, solusi yang ditemukan tidak dijamin merupakan solusi dengan jumlah langkah terpendek untuk mengeluarkan *primary piece* dari papan permainan. Ingat kembali bahwa nilai $h(n)$ menyatakan estimasi, bukan banyak langkah sebenarnya dari kondisi saat ini ke kondisi akhir. Suatu langkah dengan $h(n)$ yang cukup kecil tidak menjamin bahwa $h(n)$ yang mengikutinya akan kecil pula dan berlaku sebaliknya. Dalam arti lain, algoritma ini dapat terjebak dalam rute-rute yang bersifat minimum lokal, yang tentunya tidak selalu sama dengan minimum global.

BAB IV

SOURCE CODE PROGRAM

Program dibangun menggunakan bahasa java. Struktur direktori program sebagai berikut:

```
├── bin
│   ├── BoardState.class
│   ├── Coordinate.class
│   ├── Cost.class
│   ├── InputFile.class
│   ├── Main.class
│   ├── Orientation.class
│   ├── Output.class
│   ├── Piece.class
│   ├── Solver.class
│   ├── Tree.class
│   └── Validation.class
├── doc
│   └── Tucil3_13523066_13523072.pdf
├── Makefile
├── README.md
├── src
│   ├── BoardState.java
│   ├── Coordinate.java
│   ├── Cost.java
│   ├── InputFile.java
│   ├── Main.java
│   ├── Orientation.java
│   ├── Output.java
│   ├── Piece.java
│   ├── Solver.java
│   ├── Tree.java
│   └── Validation.java
└── test
    └── input.txt
```

Berikut adalah *source code* program pada tiap file:

5.1 BoardState.java

```
import java.util.*;

public class BoardState {
    public static int rows, cols;
    public static Map<Character, Piece> pieces = new HashMap<>();
    public static int exitRow, exitCol;
    public static Piece primaryPiece = new Piece('P', Orientation.HORIZONTAL,
0, null);
    private char[][] board;
    private Map<Character, Coordinate> piecesLocation = new HashMap<>();

    public BoardState(int rows, int cols) {
        BoardState.rows = rows;
        BoardState.cols = cols;
    }

    public void resetStatic(){
        pieces = new HashMap<>();
        primaryPiece = new Piece('P', Orientation.HORIZONTAL, 0, null);
    }

    public Map<Character, Coordinate> getPiecesLocation(){return
this.piecesLocation;}
    public void setPiecesLocation(Map<Character, Coordinate>
piecesLocation){this.piecesLocation = piecesLocation;}

    public void setExitPoint(Coordinate c) {
        BoardState.exitRow = c.r;
        BoardState.exitCol = c.c;
    }

    public BoardState cloneBoardState(){
        BoardState newState = new BoardState(rows, cols);
        newState.board = new char[rows][];
        for(int i=0;i<rows;i++){
            newState.board[i] = board[i].clone();
        }

        for(Map.Entry<Character, Coordinate> entry :
this.piecesLocation.entrySet()){
            Character id = entry.getKey();
            Coordinate coor = entry.getValue();
            newState.piecesLocation.put(id, new Coordinate(coor.r, coor.c));
        }

        return newState;
    }
}
```

```

public boolean addCell(char id, int r, int c) {
    if (id == 'K' && (id < 'A' || id > 'Z' || id != '.')) {
        return false;
    }
    if (id == '.') return true;
    if (id == 'P') {
        if (pieces.containsKey(id)) {
            if (primaryPiece.getUpLeft().r == r){
                primaryPiece.setOrientation(Orientation.HORIZONTAL);
            } else if (primaryPiece.getUpLeft().c == c) {
                primaryPiece.setOrientation(Orientation.VERTICAL);
            } else {
                return false;
            }
            pieces.get(id).addLength(1);
        } else {
            pieces.put(id, primaryPiece);
            primaryPiece.setUpLeft(new Coordinate(r, c));
            pieces.get(id).addLength(1);
        }

        return true;
    }
    if (pieces.containsKey(id)) {
        pieces.get(id).addLength(1);
        Coordinate p = pieces.get(id).getUpLeft();
        if (r == p.r){
            pieces.get(id).setOrientation(Orientation.HORIZONTAL);
        } else if (c == p.c) {
            pieces.get(id).setOrientation(Orientation.VERTICAL);
        } else {
            return false;
        }
    } else {
        Piece p = new Piece(id, Orientation.HORIZONTAL, 1, new
Coordinate(r, c));
        pieces.put(id, p);
        return true;
    }
    return true;
}

public void setBoard(char[][] board) {
    this.board = new char[rows][];
    for (int i = 0; i < rows; i++) {
        this.board[i] = board[i].clone();
    }
}

```

```

    }

    public char[][] getBoard() {
        return board;
    }

    public void addPiece(char id, int r, int c){
        // menambahkan piece ke papan diberikan id dan lokasi kiri-atas nya
        if((r== -1 && c== -1)){
            Coordinate currentCoordinate = new Coordinate(r, c);
            piecesLocation.put(id, currentCoordinate);
            return;
        }
        if(board[r][c] != '.'){
            return;
        }
        Piece currentPiece = pieces.get(id);
        if(currentPiece.getOrientation() == Orientation.HORIZONTAL){
            for(int i=0;i<currentPiece.getLength();i++){
                board[r][c+i] = id;
            }
        } else{
            for(int i=0;i<currentPiece.getLength();i++){
                board[r+i][c] = id;
            }
        }
        Coordinate currentCoordinate = new Coordinate(r, c);
        piecesLocation.put(id, currentCoordinate);
    }

    public void deletePiece(char id){
        // menghapus piece dari papan diberikan id nya
        if(!piecesLocation.containsKey(id)){
            return;
        }
        Coordinate currentCoordinate = piecesLocation.get(id);
        Piece currentPiece = pieces.get(id);
        int r = currentCoordinate.r;
        int c = currentCoordinate.c;
        if(currentPiece.getOrientation() == Orientation.HORIZONTAL){
            for(int i=0;i<currentPiece.getLength();i++){
                board[r][c+i] = '.';
            }
        } else{
            for(int i=0;i<currentPiece.getLength();i++){
                board[r+i][c] = '.';
            }
        }
    }

```

```

        piecesLocation.remove(id);
    }

    private void addMove(Map<Character, ArrayList<Coordinate>> res, char id,
        Coordinate coor){
        res.computeIfAbsent(id, unused -> new ArrayList<>()).add(coor);
    }

    public Map<Character, ArrayList<Coordinate>> generateLegalMoves(){
        // coordinate of (-1, -1) means a primary piece is going outside the
        board (hence the puzzle is solved)
        Map<Character, ArrayList<Coordinate>> res = new HashMap<>();

        // cek setiap piece di dalam papan
        for(Map.Entry<Character, Coordinate> entry :
piecesLocation.entrySet()){
            char currId = entry.getKey();
            Piece currPiece = pieces.get(currId);
            Coordinate currCoordinate = piecesLocation.get(currId);
            int r = currCoordinate.r;
            int c = currCoordinate.c;
            int i;
            if(currPiece.getOrientation() == Orientation.HORIZONTAL){
                // cek legal move sebelah kiri
                for(i=0;c-1-i>=0;i++){
                    char checkId = board[r][c-1-i];
                    if(checkId!='.'){break;}
                    Coordinate checkCoordinate = new Coordinate(r, c-1-i);
                    addMove(res, currId, checkCoordinate);
                }
                if(currId=='P' && c-1-i == exitCol){
                    addMove(res, currId, new Coordinate(-1, -1));
                }
                // cek legal move sebelah kanan
                for(i=0;c+currPiece.getLength()+i<cols;i++){
                    char checkId = board[r][c+currPiece.getLength()+i];
                    if(checkId!='.'){break;}
                    Coordinate checkCoordinate = new Coordinate(r, c+i+1);
                    addMove(res, currId, checkCoordinate);
                }
                if(currId=='P' && c+currPiece.getLength()+i == exitCol){
                    addMove(res, currId, new Coordinate(-1, -1));
                }
            } else{
                // cek legal move sebelah atas
                for(i=0;r-1-i>=0;i++){
                    char checkId = board[r-1-i][c];
                    if(checkId!='.'){break;}

```



```

        Coordinate checkCoordinate = new Coordinate(r-1-i, c);
        addMove(res, currId, checkCoordinate);
    }
    if(currId=='P' && r-1-i == exitRow){
        addMove(res, currId, new Coordinate(-1, -1));
    }
    // cek legal move sebelah bawah
    for(i=0;r+currPiece.getLength()+i<rows;i++){
        char checkId = board[r+currPiece.getLength()+i][c];
        if(checkId!='.'){break;}
        Coordinate checkCoordinate = new Coordinate(r+i+1, c);
        addMove(res, currId, checkCoordinate);
    }
    if(currId=='P' && r+currPiece.getLength()+i == exitRow){
        addMove(res, currId, new Coordinate(-1, -1));
    }
    }
};

return res;
}

public void printAllPieces() {
    for (Map.Entry<Character, Piece> entry : pieces.entrySet()) {
        entry.getValue().printPiece();
    }
}

public void printGameState() {
    System.out.println("\nBoard:");
    for (int i = 0; i < board.length; i++) {
        for (int j = 0; j < board[i].length; j++) {
            System.out.print(board[i][j] + " ");
        }
        System.out.println();
    }
}

@Override
public boolean equals(Object obj) {
    if (this == obj) return true;
    if (!(obj instanceof BoardState)) return false;
    BoardState o = (BoardState) obj;

    for(int i=0;i<rows;i++){
        for(int j=0;j<cols;j++){
            if(this.board[i][j] != o.board[i][j]){
                return false;
            }
        }
    }
    return true;
}

```

```

        }
    }
    return true;
}

@Override
public int hashCode() {
    return Arrays.deepHashCode(this.board);
}
}

```

5.2 Coordinate.java

```

public class Coordinate {
    public int r;
    public int c;
    public Coordinate(int r, int c) {
        this.r = r;
        this.c = c;
    }
    public Coordinate() {
        this.r = 0;
        this.c = 0;
    }
    public Coordinate(Coordinate c) {
        this.r = c.r;
        this.c = c.c;
    }
}

```

5.3 Cost.java

```

import java.util.Map;

public class Cost {
    public static int algorithm; // 1. UCS, 2.GBFS, 3.A*
    public static BoardState stateBefore;

    public Cost(int algorithm, BoardState stateBefore){
        Cost.algorithm = algorithm;
        Cost.stateBefore = stateBefore;
    }

    public static int f(BoardState state){
        if(algorithm==1){
            return g(state);
        } else if(algorithm==2){

```

```

        return h(state);
    } else if(algorithm==3){
        return g(state)+h(state);
    }
    return 0;
}

public static int g(BoardState stateAfter){
    /**
     * definisi g(n)
     *  $g(n) = G(n-1) - G(n)$ 
     * dengan G(n) adalah fungsi pembantu
     * G(n) = jumlah mobil di depan atau di belakang mobil x yang berbeda
    orientasi dengan mobil x, x adalah seluruh mobil yang ada
     */
    BoardState stateBefore = Cost.stateBefore;
    Map<Character, Piece> pieces = BoardState.pieces;
    int GnBefore = 0;
    int GnAfter = 0;
    Map<Character, Coordinate> piecesLocationBefore =
stateBefore.getPiecesLocation();
    Map<Character, Coordinate> piecesLocationAfter =
stateAfter.getPiecesLocation();
    for (Map.Entry<Character, Piece> entry : pieces.entrySet()){
        char id1 = entry.getKey();
        Piece piece1 = entry.getValue();
        // mencari GnBefore. cek piecesLocationBefore apakah ada piece
yang berbeda orientasi dengan piece.id, hitung jumlahnya
        for (Map.Entry<Character, Coordinate> entryBefore :
piecesLocationBefore.entrySet()){
            char id2 = entryBefore.getKey();
            Piece piece2 = pieces.get(id2);
            if (piece1.getOrientation() != piece2.getOrientation()){
                Coordinate coor1 = entryBefore.getValue();
                Coordinate coor2 = piecesLocationBefore.get(id1);
                int len2 = piece2.getLength();
                if (piece2.getOrientation() == Orientation.HORIZONTAL){
                    int coor1y = coor1.c;
                    for (int i = 0; i < len2; i++){
                        int coor2y = coor2.c + i;
                        if (coor1y == coor2y){
                            GnBefore++;
                        }
                    }
                } else {
                    int coor1x = coor1.r;
                    for (int i = 0; i < len2; i++){
                        int coor2x = coor2.r + i;

```

```

        if (coor1x == coor2x){
            GnBefore++;
        }
    }
}

// mencari GnAfter. cek piecesLocationAfter apakah ada piece yang
berbeda orientasi dengan piece.id, hitung jumlahnya
for (Map.Entry<Character, Coordinate> entryAfter :
piecesLocationAfter.entrySet()){
    char id2 = entryAfter.getKey();
    Piece piece2 = pieces.get(id2);
    if (piece1.getOrientation() != piece2.getOrientation()){
        Coordinate coor1 = entryAfter.getValue();
        Coordinate coor2 = piecesLocationAfter.get(id1);
        int len2 = piece2.getLength();
        if (piece2.getOrientation() == Orientation.HORIZONTAL){
            int coor1y = coor1.c;
            for (int i = 0; i < len2; i++){
                int coor2y = coor2.c + i;
                if (coor1y == coor2y){
                    GnAfter++;
                }
            }
        } else {
            int coor1x = coor1.r;
            for (int i = 0; i < len2; i++){
                int coor2x = coor2.r + i;
                if (coor1x == coor2x){
                    GnAfter++;
                }
            }
        }
    }
}

int gn = GnBefore - GnAfter;

return gn;
}

public static int h(BoardState state){
    int res = 0;
    Piece primPiece = BoardState.pieces.get('P');
    Coordinate primCoor = new Coordinate();
    if (state.getPiecesLocation().containsKey('P')){

```

```

        primCoor = state.getPiecesLocation().get('P');
    } else{
        return 0;
    }
    if(primCoor.r==-1 && primCoor.c==-1){
        return 0;
    }
    if(primPiece.getOrientation() == Orientation.HORIZONTAL){
        if(primCoor.c < BoardState.exitCol){ // exit di sebelah kanan
            for(int
i=0;primCoor.c+primPiece.getLength()+i<BoardState.cols;i++){
                char checkId =
state.getBoard()[primCoor.r][primCoor.c+primPiece.getLength()+i];
                if(checkId!='.'){
                    res++;
                }
            }
        } else if(primCoor.c > BoardState.exitCol){ // exit di sebelah kiri
            for(int i=0;primCoor.c-1-i>=0;i++){
                char checkId =
state.getBoard()[primCoor.r][primCoor.c-1-i];
                if(checkId!='.'){
                    res++;
                }
            }
        }
    } else{
        if(primCoor.r < BoardState.exitRow){ // exit di sebelah atas
            for(int i=0;primCoor.r-1-i>=0;i++){
                char checkId =
state.getBoard()[primCoor.r-1-i][primCoor.c];
                if(checkId!='.'){
                    res++;
                }
            }
        } else if(primCoor.r > BoardState.exitRow){ // exit di sebelah
bawah
            for(int
i=0;primCoor.r+primPiece.getLength()+i<BoardState.rows;i++){
                char checkId =
state.getBoard()[primCoor.r+primPiece.getLength()+i][primCoor.c];
                if(checkId!='.'){
                    res++;
                }
            }
        }
    }
}

```

```

        return res;
    }
}

```

5.4 InputFile.java

```

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.HashSet;
import java.util.List;
import java.util.Map;
import java.util.Set;

public class InputFile {
    private final String filePath;
    private final BoardState boardState;

    public InputFile(String filePath) {
        this.filePath = filePath;
        this.boardState = new BoardState(0, 0);
        loadBoardFromFile();
    }

    public BoardState getBoardState() {
        return boardState;
    }

    private void addPiecesLocation() {
        Map<Character, Coordinate> piecesLocation = new HashMap<>();
        for (Map.Entry<Character, Piece> entry : BoardState.pieces.entrySet())
        {
            char id = entry.getKey();
            Piece piece = entry.getValue();
            Coordinate coordinate = piece.getUpLeft();
            piecesLocation.put(id, coordinate);
        }
        boardState.setPiecesLocation(piecesLocation);
    }

    private void loadBoardFromFile() {
        try (BufferedReader reader = new BufferedReader(new
FileReader(filePath))) {
            int rows = 0, cols = 0;
            int expectedPieces;
            List<Character> foundPieceIds = new ArrayList<>();

```

```

        // Parse header lines
        String line = reader.readLine();
        int[] dims = parseDimensions(line);
        rows = dims[0];
        cols = dims[1];

        line = reader.readLine();
        expectedPieces = parseExpectedPieceCount(line);

        // Prepare board matrix
        char[][] matrix = new char[rows][cols];

        boolean exitFound = parseTopExit(line = reader.readLine(), matrix,
foundPieceIds);

        // Read remaining rows
        exitFound |= parseBoardRows(reader, matrix, foundPieceIds, rows,
cols, exitFound);

        validateMatrixSize(matrix, rows, cols);
        validateUniquePieces(foundPieceIds, expectedPieces);

        // If exit still not found, check last line
        if (!exitFound) {
            line = reader.readLine();
            if (line != null) {
                parseBottomExit(line, rows - 1);
            } else {
                throw new IllegalArgumentException("Exit point not found");
            }
        }

        // Finalize board state
        BoardState.rows = rows;
        BoardState.cols = cols;
        boardState.setBoard(matrix);
        populateBoardState(matrix, rows, cols);
        addPiecesLocation();

    } catch (IOException e) {
        e.printStackTrace();
    }
}

private int[] parseDimensions(String header) {
    String[] parts = header.split(" ");
    int r = Integer.parseInt(parts[0]);

    int c = Integer.parseInt(parts[1]);

```

```

        return new int[]{r, c};
    }

    private int parseExpectedPieceCount(String line) {
        String[] parts = line.split(" ");
        return Integer.parseInt(parts[0]);
    }

    private boolean parseTopExit(String line, char[][] matrix, List<Character>
pieceIds) {
        char[] cells = line.toCharArray();

        for (int j = 0; j < cells.length; j++) {
            if (cells[j] == 'K') {
                boardState.setExitPoint(new Coordinate(-1, (j + 1) / 2));
                return true;
            }
        }
        // Also process row 0 for pieces
        extractRowData(cells, matrix, 0, pieceIds);
        return false;
    }

    private boolean parseBoardRows(BufferedReader reader, char[][] matrix,
List<Character> pieceIds, int totalRows, int totalCols, boolean
exitAlreadyFound) throws IOException {
        boolean exitFound = exitAlreadyFound;
        int up = totalRows;
        int down = 1;
        if (exitAlreadyFound) {
            down = 0;
        }
        for (int i = down; i < up; i++) {
            String line = reader.readLine();

            if (line == null) break;

            char[] cells = line.toCharArray();
            // Check side exits
            if (!exitFound) {
                if (cells[0] == 'K') {
                    boardState.setExitPoint(new Coordinate(i, -1));
                    exitFound = true;
                } else if (cells[cells.length - 1] == 'K') {
                    boardState.setExitPoint(new Coordinate(i, totalCols));
                    exitFound = true;
                }
            }
        }
    }
}

```



```

        extractRowData(cells, matrix, i, pieceIds);
    }
    return exitFound;
}

private void extractRowData(char[] cells, char[][] matrix, int rowIndex,
List<Character> pieceIds) {
    int colIndex = 0;
    for (char cell : cells) {
        if (cell != ' ' && cell != 'K') {
            if (colIndex >= matrix[rowIndex].length) {
                colIndex = matrix[rowIndex].length - 1;
            }
            matrix[rowIndex][colIndex] = cell;
            if (!pieceIds.contains(cell)) {
                pieceIds.add(cell);
            }
            colIndex++;
        }
    }
}

private void parseBottomExit(String line, int bottomRow) {
    char[] cells = line.toCharArray();

    for (int j = 0; j < cells.length; j++) {
        if (cells[j] == 'K') {
            boardState.setExitPoint(new Coordinate(bottomRow+1, (j + 1) /
2));
            break;
        }
    }
}

private void validateMatrixSize(char[][] matrix, int rows, int cols) {
    if (matrix.length != rows || matrix[0].length != cols) {
        throw new IllegalArgumentException("Invalid matrix size");
    }
}

private void validateUniquePieces(List<Character> pieceIds, int
expectedCount) {
    Set<Character> unique = new HashSet<>(pieceIds);
    // exclude '.' and 'P'
    unique.remove('.');
    unique.remove('P');

```

```

        if (unique.size() != expectedCount) {

```

```

        throw new IllegalArgumentException(
            "Invalid number of pieces, expected: " + expectedCount + ",
found: " + unique.size()
            + " elements: " + unique);
    }
}

private void populateBoardState(char[][] matrix, int rows, int cols) {
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            char cell = matrix[i][j];
            boolean valid = boardState.addCell(cell, i, j);
            if (!valid) {
                System.out.println("Invalid cell: " + cell + " at (" + i +
", " + j + ")");
            }
        }
    }
}
}

```

5.5 Main.java

```

import java.util.Scanner;
import java.util.*;

public class Main {
    public static void main(String[] args) {
        System.out.println("Selamat datang di program Rush Hour Puzzle
Solver!");

        Scanner scanner = new Scanner(System.in);
        while(true){
            String filePath;
            boolean isValidatingFile = true;
            BoardState boardState = new BoardState(1,1);
            boardState.resetStatic();
            while(isValidatingFile){
                System.out.print("Nama File Input: ");
                filePath = scanner.nextLine();
                if (filePath.equals("exit")) {
                    System.out.println("Keluar dari program...");
                    return;
                }
                if (filePath.endsWith(".txt")) {
                    try {

```

```

        InputFile inputFile = new InputFile("test/" +
filePath);

        boardState = inputFile.getBoardState();
        Validation val = new Validation(boardState);
        val.validation();
    } catch (Exception e) {
        System.out.println("^Ditemukan kesalahan pada file: " +
e.getMessage());
        continue;
    }
    isValidatingFile = false;
} else {
    System.out.println("File tidak valid. Silakan masukan
kembali file *.txt");
}
}
System.out.println("-----ALGORITMA PENCARIAN-----");
System.out.println("1. Uniform Cost Search (UCS)");
System.out.println("2. Greedy Best First Search");
System.out.println("3. A*");
System.out.println("-----");

int algorithmCode = 0;
boolean isValidatingCode = true;
while(isValidatingCode){
    System.out.print("Algoritma Pencarian (1-3): ");
    String input = scanner.nextLine();
    try{
        algorithmCode = Integer.parseInt(input);
        if(algorithmCode<1 || algorithmCode>3){
            System.out.println("Harap masukkan angka 1-3");
            continue;
        }
    } catch(NumberFormatException e){
        System.out.println("Harap masukkan angka 1-3");
        continue;
    }
    isValidatingCode = false;
}

// Pencarian Solusi
// Root node, kondisi awal
Tree tree = new Tree(boardState);
Cost cost = new Cost(algorithmCode, tree.getState());
Solver solver = new Solver();

System.out.println("Mencari solusi...");
long startTimeSolve = System.currentTimeMillis();

```

```

Tree goal = solver.solve(tree);
long endTimeSolve = System.currentTimeMillis();
long solveTime = endTimeSolve - startTimeSolve;

// output
Tree path = Solver.generatePath(goal);
Output output = new Output(path);

System.out.println("-----RUTE MENUJU SOLUSI-----");
output.printBoard();
System.out.println();
System.out.println("-----");
System.out.println("Waktu pencarian (ms) : " + solveTime);
System.out.println("Banyak node dikunjungi: " + solver.count);
System.out.println("-----");

System.out.print("Keluar dari program? (y): ");
String confirm = scanner.nextLine();
if(confirm.equals("y") || confirm.equals("Y")){
    System.out.println("Sampai jumpa kembali!");
    System.exit(0);
}
}
}
}

```

5.6 Orientation.java

```

public enum Orientation {
    HORIZONTAL, VERTICAL
}

```

5.7 Output.java

```

import java.util.*;

public class Output {
    private Tree root;
    private char exitPosition; // 'u' 'd' 'l' 'r'
    public static final String RESET = "\u001B[0m";
    public static final String RED = "\u001B[31m";
    public static final String GREEN = "\u001B[32m";
    public static final String NAVY = "\u001B[34m";
    public static final String YELLOW_BG = "\u001B[43m";

```

```

    public Output(Tree root){
        this.root = root;

```

```

        this.exitPosition = setExitPosition();
    }

    private char setExitPosition(){
        int row = BoardState.exitRow;
        int col = BoardState.exitCol;
        if (row == -1){
            return 'u';
        } else if (row == BoardState.rows){
            return 'd';
        } else if (col == -1){
            return 'l';
        } else if (col == BoardState.cols){
            return 'r';
        }
        return ' ';
    }

    public void printBoard(){
        if (root == null){
            System.out.println("Tidak ditemukan solusi!");
            return;
        }
        if (root.getParent() == null){
            printRootNode(root);
        } else {
            System.out.println("Papan Awal");
        }
        printBoardHelper(root.getChildren().get(0), 1);
    }

    private void printBoardHelper(Tree node, int i){
        if (node == null){
            return;
        }

        Coordinate primaryPieceCurrent =
node.getState().getPiecesLocation().get('P');
        if (primaryPieceCurrent.r == -1 && primaryPieceCurrent.c == -1){
            node.getState().getPiecesLocation().remove('P');

            System.out.println();
            System.out.println("Gerakan " + i + ": Primary piece bergerak
keluar dari papan!");

```

```

        char[][] boardString =
addExitPointInBoard(node.getState().getBoard(), new
Coordinate(BoardState.exitRow, BoardState.exitCol));
        for (int j = 0; j < boardString.length; j++){
            for (int k = 0; k < boardString[0].length; k++){
                System.out.print(boardString[j][k]);
            }
            System.out.println();
        }
        return;
    }
    System.out.println("Gerakan " + i + ": " + node.getIdMoved() + "-" +
node.getMoveType());
    String[][] boardString = boardCharToString(node);
    for (int j = 0; j < boardString.length; j++){
        for (int k = 0; k < boardString[0].length; k++){
            System.out.print(boardString[j][k]);
        }
        System.out.println();
    }
    printBoardHelper(node.getChildren().get(0), i+1);
}

private String[][] boardCharToString(Tree node){
    Coordinate exitPoint = new Coordinate(BoardState.exitRow,
BoardState.exitCol);
    char[][] board = addExitPointInBoard(node.getState().getBoard(),
exitPoint);
    String[][] boardString = new String[board.length][board[0].length];
    for (int i = 0; i < board.length; i++){
        for (int j = 0; j < board[0].length; j++){
            if (board[i][j] == 'K'){
                boardString[i][j] = GREEN + String.valueOf(board[i][j]) +
RESET;
            } else if (board[i][j] == 'P'){
                boardString[i][j] = RED + String.valueOf(board[i][j]) +
RESET;
            } else {
                boardString[i][j] = String.valueOf(board[i][j]);
            }
        }
    }
    char idMoved = node.getIdMoved();
    Coordinate piece = node.getState().getPiecesLocation().get(idMoved);
    int length = BoardState.pieces.get(idMoved).getLength();
    char moveType = node.getMoveType();
    int steps = node.getSteps();

```

```

        if (exitPosition == '1'){

```

```

        piece.c = piece.c + 1;
    }
    if (exitPosition == 'u'){
        piece.r = piece.r + 1;
    }
    if (moveType == 'u'){
        int start = piece.r;
        int end = piece.r + steps + length - 1;
        for (int i = start; i <= end; i++){
            boardString[i][piece.c] = NAVY + YELLOW_BG +
boardString[i][piece.c] + RESET;
        }
    } else if (moveType == 'd'){
        int start = piece.r - steps;
        int end = piece.r + length - 1;
        for (int i = start; i <= end; i++){
            boardString[i][piece.c] = NAVY + YELLOW_BG +
boardString[i][piece.c] + RESET;
        }
    } else if (moveType == 'l'){
        int start = piece.c;
        int end = piece.c + steps + length - 1;
        for (int i = start; i <= end; i++){
            boardString[piece.r][i] = NAVY + YELLOW_BG +
boardString[piece.r][i] + RESET;
        }
    } else if (moveType == 'r'){
        System.out.println("piece.c: " + piece.c);
        int start = piece.c - steps;
        int end = piece.c + length - 1;
        for (int i = start; i <= end; i++){
            boardString[piece.r][i] = NAVY + YELLOW_BG +
boardString[piece.r][i] + RESET;
        }
    }
    System.out.println();
    return boardString;
}

private void printRootNode(Tree node){
    if (node.getParent() == null){
        System.out.println("Papan Awal");
        char[][] board = node.getState().getBoard();
        board = addExitPointInBoard(board, new
Coordinate(BoardState.exitRow, BoardState.exitCol));
        for (int i = 0; i < board.length; i++){
            for (int j = 0; j < board[0].length; j++){

```

```

        System.out.print(board[i][j]);

    }
    System.out.println();
}
}

private char[][] addExitPointInBoard(char[][] board, Coordinate exitPoint){
    int row = exitPoint.r;
    int col = exitPoint.c;
    if (row == -1){
        char[][] newBoard = new char[BoardState.rows+1][BoardState.cols];
        for (int i = 0; i < BoardState.cols; i++){
            newBoard[0][i] = ' ';
        }
        newBoard[0][col] = 'K';
        for (int i = 1; i < BoardState.rows+1; i++){
            newBoard[i] = board[i-1];
        }
        return newBoard;
    } else if (col == -1){
        char[][] newBoard = new char[BoardState.rows][BoardState.cols+1];
        for (int i = 0; i < BoardState.rows; i++){
            newBoard[i][0] = ' ';
        }
        newBoard[row][0] = 'K';
        for (int i = 0; i < BoardState.rows; i++){
            for (int j = 1; j < BoardState.cols+1; j++){
                newBoard[i][j] = board[i][j-1];
            }
        }
        return newBoard;
    } else if (row == BoardState.rows){
        char[][] newBoard = new char[BoardState.rows+1][BoardState.cols];
        for (int i = 0; i < BoardState.cols; i++){
            newBoard[BoardState.rows][i] = ' ';
        }
        newBoard[BoardState.rows][col] = 'K';
        for (int i = 0; i < BoardState.rows; i++){
            newBoard[i] = board[i];
        }
        return newBoard;
    } else if (col == BoardState.cols){
        char[][] newBoard = new char[BoardState.rows][BoardState.cols+1];
        for (int i = 0; i < BoardState.rows; i++){

            newBoard[i][BoardState.cols] = ' ';
        }
    }
}

```



```

        newBoard[row][BoardState.cols] = 'K';

        for (int i = 0; i < BoardState.rows; i++){
            for (int j = 0; j < BoardState.cols; j++){
                newBoard[i][j] = board[i][j];
            }
        }
        return newBoard;
    }
    return board;
}
}

```

5.8 Piece.java

```

public class Piece {
    public char id;
    public Orientation orientation;
    public int length;
    public Coordinate upLeft;

    public Piece(char id, Orientation orientation, int length, Coordinate
upLeft) {
        this.id = id;
        this.orientation = orientation;
        this.length = length;
        this.upLeft = upLeft;
    }

    public char getId() {
        return id;
    }
    public Orientation getOrientation() {
        return orientation;
    }
    public int getLength() {
        return length;
    }
    public Coordinate getUpLeft() {
        return upLeft;
    }
    public void setId(char id) {
        this.id = id;
    }
    public void setOrientation(Orientation orientation) {
        this.orientation = orientation;
    }
}

```

```

    }
    public void setLength(int length) {

```

```

        this.length = length;
    }

    public void setUpLeft(Coordinate upLeft) {
        this.upLeft = upLeft;
    }
    public void addLength(int length) {
        this.length += length;
    }
    public void printPiece() {
        System.out.println("Piece ID: " + id);
        System.out.println("Orientation: " + orientation);
        System.out.println("Length: " + length);
        System.out.println("UpLeft Coordinate: (" + upLeft.r + ", " + upLeft.c
+ ")");
    }
}

```

5.9 Solver.java

```

import java.util.*;

public class Solver {
    public int algorithm; // 1. UCS, 2.GBFS, 3.A*
    public int count;

    public Tree solve(Tree t){
        PriorityQueue<Tree> pq = new PriorityQueue<>();
        Set<BoardState> visited = new HashSet<>();

        pq.add(t);
        count = 0;
        while(pq.isEmpty()==false){
            count++;
            Tree currentNode = pq.poll();
            if(currentNode.isGoal()){
                return currentNode;
            }

            visited.add(currentNode.getState());
            currentNode.generateChildren();
            for(Tree childNode : currentNode.getChildren()){
                if(visited.contains(childNode.getState())==false){
                    pq.add(childNode);
                }
            }
        }
    }
}

```

```

        return null;
    }

    public static Tree generatePath(Tree goalNode) {
        if (goalNode == null) {
            return null;
        }
        ArrayList<Tree> path = new ArrayList<>();
        Tree current = goalNode;
        while(current != null){
            path.add(current);
            current=current.getParent();
        }
        Collections.reverse(path);

        for(int i=0;i<path.size()-1;i++){
            ArrayList<Tree> child = new ArrayList<>();
            child.add(path.get(i+1));
            path.get(i).setChildren(child);
        }
        ArrayList<Tree> nochild = new ArrayList<>();
        path.get(path.size()-1).setChildren(nochild);
        return path.get(0);
    }
}

```

5.10 Tree.java

```

import java.util.*;

public class Tree implements Comparable<Tree>{
    private BoardState state;
    private ArrayList<Tree> children;
    private char moveType; // u r d l
    private int fn;
    private Tree parent;
    private int steps;
    private char idMoved;

    Tree(BoardState state){
        this.state = state;
        this.children = new ArrayList<Tree>();
        this.moveType = ' ';
        this.fn = Cost.f(state);
        this.parent = null;
    }
}

```

```

}

```

```

//getter
public BoardState getState(){return this.state;}
public ArrayList<Tree> getChildren(){return this.children;}
public Tree getParent(){return this.parent;}

public int getFn(){return this.fn;}
public char getMoveType(){return this.moveType;}

// setter
public void updateChildren(BoardState state, int fn, char mt, int steps,
char idMoved){
    Tree child = new Tree(state);
    child.fn = fn;
    child.moveType = mt;
    child.parent = this;
    child.steps = steps;
    child.idMoved = idMoved;
    this.children.add(child);
}

public void setChildren(ArrayList<Tree> children){
    this.children = children;
}

@Override
public int compareTo(Tree other){
    return Integer.compare(this.fn, other.fn);
}

public boolean isGoal(){
    return (state.getPiecesLocation().get('P').r == -1 &&
state.getPiecesLocation().get('P').c == -1);
}

public void generateChildren(){
    Map<Character, ArrayList<Coordinate>> legalMoves =
state.generateLegalMoves();
    for(Map.Entry<Character, ArrayList<Coordinate>> entry :
legalMoves.entrySet()){
        char currId = entry.getKey();
        ArrayList<Coordinate> currCoordinates = entry.getValue();

        for(int i=0;i<currCoordinates.size();i++){
            int r = currCoordinates.get(i).r;
            int c = currCoordinates.get(i).c;
            BoardState newState = state.cloneBoardState();

            // cek tipe gerakan
            char mt = ' ';

```

```

        int oldR = state.getPiecesLocation().get(currId).r;
        int oldC = state.getPiecesLocation().get(currId).c;
        if(r>oldR){mt = 'd';}
        else if(r<oldR){mt = 'u';}

        else if(c<oldC){mt = 'l';}
        else if(c>oldC){mt = 'r';}

        // movement
        newState.deletePiece(currId);
        newState.addPiece(currId, r, c);

        int fn = Cost.f(newState);

        int steps = Math.abs(oldR - r) + Math.abs(oldC - c);
        if(r== -1 && c== -1){
            if(BoardState.pieces.get('P').getOrientation() ==
Orientation.HORIZONTAL){
                steps = Math.abs(oldC - c);
            } else{
                steps = Math.abs(oldR - r);
            }
        }
        updateChildren(newState, fn, mt, steps, currId);
    }
}

// getter idMoved
public char getIdMoved(){
    return this.idMoved;
}

// setter idMoved
public void setIdMoved(char idMoved){
    this.idMoved = idMoved;
}

// getter steps
public int getSteps(){
    return this.steps;
}

// setter steps
public void setSteps(int steps){
    this.steps = steps;
}
}

```

5.11 Validation.java

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

```

public class Validation {
    private BoardState boardState;
    private Piece primaryPiece;

    public Validation(BoardState boardState) {
        this.boardState = boardState;
        this.primaryPiece = BoardState.primaryPiece;
    }
    public void validation(){
        validation1();
        validation2();
        validation3();
        validation4();
    }

    /**
     * Validasi 1: Pastikan exit point berada satu baris/kolom dengan primary
    piece.
     * Jika primary piece horizontal → exitCol harus satu baris dengan piece.
     * Jika vertical → exitRow harus satu kolom dengan piece.
     */
    private boolean validation1() {
        if (primaryPiece.getOrientation() == Orientation.HORIZONTAL) {
            if (primaryPiece.getUpLeft().r != BoardState.exitRow) {
                throw new IllegalStateException("Exit point is not in the same
    row as the primary piece.");
            }
        } else {
            if (primaryPiece.getUpLeft().c != BoardState.exitCol) {
                throw new IllegalStateException("Exit point is not in the same
    column as the primary piece.");
            }
        }
        return true;
    }

    /**
     * Validasi 2: Cek apakah jalur menuju exit point kosong.
     * Jika horizontal, periksa semua sel di kanan piece sampai exit.
     * Jika vertical, periksa semua sel di bawah piece sampai exit.
     */
    private boolean validation2(){
        boolean res = true;
        if (primaryPiece.getOrientation() == Orientation.HORIZONTAL) {
            for (int i = primaryPiece.getUpLeft().c + 1; i <
    boardState.cols; i++) {

```

```

        for (int j = primaryPiece.getUpLeft().r; j <
primaryPiece.getUpLeft().r + primaryPiece.getLength(); j++) {
            if (boardState.getBoard()[j][i] != '.') {
                res = false;
                break;
            }
        }
    } else {
        for (int i = primaryPiece.getUpLeft().r + 1; i <
boardState.rows; i++) {
            if (boardState.getBoard()[i][primaryPiece.getUpLeft().c] !=
'.') {
                for (int j = primaryPiece.getUpLeft().c; j <
primaryPiece.getUpLeft().c + primaryPiece.getLength(); j++) {
                    if (boardState.getBoard()[i][j] != '.') {
                        res = false;
                        break;
                    }
                }
            }
        }
    }
}
if (res) {
    throw new NullPointerException("There is a block (opposite
orientation with piece) in the way of primary piece");
}
return true;
}

```

```

/**
 * Validasi 3: Pastikan tidak ada block dengan orientasi sama menghalangi
dari ujung primary ke exit.
 */

```

```

private boolean validation3() {
    if (BoardState.primaryPiece.orientation == Orientation.HORIZONTAL){
        int ex = BoardState.exitCol;
        int pry = BoardState.primaryPiece.upLeft.c;
        int prx = BoardState.primaryPiece.upLeft.r;
        int len = BoardState.primaryPiece.length;

        System.out.println("orientation: " +
BoardState.primaryPiece.orientation);
        if (ex < pry){
            char[][] matrix = boardState.getBoard();

```

```

            for (int i = ex+2; i < pry; i++){
                if (matrix[prx][i] == matrix[prx][i-1]){

```

```

        System.out.println("a");
        throw new NullPointerException("There is a block (same
orientation with piece) in the way of primary piece to exit point: " +
matrix[i][prx]);
    }
}
} else {

    char[][] matrix = boardState.getBoard();
    for (int i = pry+len+1; i < ex; i++){
        if (matrix[prx][i] == matrix[prx][i-1]){
            System.out.println("b");
            throw new NullPointerException("There is a block (same
orientation with piece) in the way of primary piece to exit point: " +
matrix[i][prx]);
        }
    }
}
} else {
    int ex = BoardState.exitRow;
    int pry = BoardState.primaryPiece.upLeft.c;
    int prx = BoardState.primaryPiece.upLeft.r;
    int len = BoardState.primaryPiece.length;

    if (ex < prx){
        char[][] matrix = boardState.getBoard();
        for (int i = ex+2; i < pry; i++){
            if (matrix[i][pry] == matrix[i-1][pry]){
                System.out.println("c");
                throw new NullPointerException("There is a block (same
orientation with piece) in the way of primary piece to exit point: " +
matrix[i][pry]);
            }
        }
    } else {
        char[][] matrix = boardState.getBoard();
        for (int i = pry+len+1; i < ex; i++){
            if (matrix[i][pry] == matrix[i-1][pry]){
                System.out.println("d");
                throw new NullPointerException("There is a block (same
orientation with piece) in the way of primary piece to exit point: " +
matrix[i][pry]);
            }
        }
    }
}

return true;
}

```



```

    /**
     * Validasi 4: misal koordinat exit point adalah (i,0) atau (0,j) maka h =
     max(i, row - i) atau max(col - j, j)
     * akan di cek seluruh piece yang berbeda orientasi dengan primary piece,
     apakah ada yang lebih dari h, jika iya maka board tidak valid
     */
    private boolean validation4() {

        boolean found = false;
        Orientation orientation = BoardState.primaryPiece.getOrientation();
        int h;
        if (BoardState.primaryPiece.getOrientation() ==
Orientation.HORIZONTAL) {
            h = Math.max(BoardState.exitRow, BoardState.rows -
BoardState.exitRow);
        } else {
            h = Math.max(BoardState.exitCol, BoardState.cols -
BoardState.exitCol);
        }

        Map<Character, Piece> pieces = BoardState.pieces;
        for (Map.Entry<Character, Piece> entry : pieces.entrySet()) {
            Piece piece = entry.getValue();
            if (piece.getOrientation() != orientation) {
                if (piece.getLength() > h) {
                    System.out.println("h : " + h);
                    int rowPiece = piece.getUpLeft().r;
                    int colPiece = piece.getUpLeft().c;
                    int rowExit = BoardState.exitRow;
                    int colExit = BoardState.exitCol;
                    int rowPrimary = BoardState.primaryPiece.getUpLeft().r;
                    int colPrimary = BoardState.primaryPiece.getUpLeft().c;
                    boolean isBetween = (
                        (rowPiece >= Math.min(rowExit, rowPrimary) && rowPiece
<= Math.max(rowExit, rowPrimary)) ||
                        (colPiece >= Math.min(colExit, colPrimary) && colPiece
<= Math.max(colExit, colPrimary))
                    );
                    if (isBetween) {
                        found = true;
                    }
                    break;
                }
            }
        }
    }
}

```

```

    if (found) {
        System.out.println("exitRow: " + BoardState.exitRow);
    }
}

```

```

        throw new NullPointerException("There is a block (different
orientation with piece) that is too long to exit point");
    }
    return true;
}
}

```

5.11 Output.java (tambahan)

```

public void writeOutputTxt(String filename, long searchTime, int nodeCount, int
algorithmCode){
    try{
        FileWriter writer = new FileWriter("test/" +
filename.replace(".txt", "-solution-" + algorithmCode + ".txt"));
        writer.write("-----RUTE MENUJU SOLUSI-----\n");
        if (root == null){
            writer.write("Tidak ditemukan solusi!\n");
            writer.write("-----\n");
            writer.write("Waktu pencarian (ms) : " + searchTime + "\n");
            writer.write("Banyak node dikunjungi: " + nodeCount + "\n");
            writer.write("-----");
            writer.close();
            System.out.println("Solusi berhasil disimpan dalam folder
test.");
            return;
        }
        writer.write("Papan Awal\n");
        char[][] board = root.getState().getBoard();
        board = addExitPointInBoard(board, new
Coordinate(BoardState.exitRow, BoardState.exitCol));
        for (int i = 0; i < board.length; i++){
            for (int j = 0; j < board[0].length; j++){
                writer.write(board[i][j]);
            }
            writer.write("\n");
        }
        printBoardHelperTxt(writer, root.getChildren().get(0), 1);
        writer.write("-----\n");
        writer.write("Waktu pencarian (ms) : " + searchTime + "\n");
        writer.write("Banyak node dikunjungi: " + nodeCount + "\n");
        writer.write("-----");
        writer.close();
        System.out.println("Solusi berhasil disimpan dalam folder test.");
        writer.close();
    }
}

```

```

        } catch(IOException e){
            System.out.println("Terjadi error saat penulisan file");
        }
    }

    private void printBoardHelperTxt(FileWriter writer, Tree node, int i)
    throws IOException{
        if (node == null){
            return;
        }

        Coordinate primaryPieceCurrent =
        node.getState().getPiecesLocation().get('P');
        if (primaryPieceCurrent.r == -1 && primaryPieceCurrent.c == -1){
            node.getState().getPiecesLocation().remove('P');

            writer.write("\n");
            writer.write("Gerakan " + i + ": Primary piece bergerak keluar dari
papan!\n");

            char[][] boardString =
            addExitPointInBoard(node.getState().getBoard(), new
            Coordinate(BoardState.exitRow, BoardState.exitCol));
            for (int j = 0; j < boardString.length; j++){
                for (int k = 0; k < boardString[0].length; k++){
                    writer.write(boardString[j][k]);
                }
                writer.write("\n");
            }
            return;
        }
        writer.write("\nGerakan " + i + ": " + node.getIdMoved() + "-" +
        node.getMoveType());
        writer.write("\n");
        char[][] boardString = addExitPointInBoard(node.getState().getBoard(),
        new Coordinate(BoardState.exitRow, BoardState.exitCol));
        for (int j = 0; j < boardString.length; j++){
            for (int k = 0; k < boardString[0].length; k++){
                writer.write(boardString[j][k]);
            }
            writer.write("\n");
        }
        printBoardHelperTxt(writer, node.getChildren().get(0), i+1);
    }
}

```

BAB V PENGUJIAN

Pengujian dilakukan sebagai untuk mengecek apakah algoritma yang sudah dikembangkan mampu bekerja sebagaimana seharusnya, sesuai dengan kasus uji yang diberikan. Dalam pengujian ini, terdapat sebanyak 12 kasus uji, dengan 4 buah kasus uji untuk masing-masing algoritma. Hasil pengujian terdapat dalam tabel di bawah ini.

No.	Input	Output
1.	UCS 6 6 11 AAB..F ..BCDF GPPCDFK GH.III GHJ... LLJMM.	Nama File Input: test1.txt -----ALGORITMA PENCARIAN----- 1. Uniform Cost Search (UCS) 2. Greedy Best First Search 3. A* ----- Algoritma Pencarian (1-3): 1 Mencari solusi... -----RUTE MENUJU SOLUSI----- Papan Awal AAB..F ..BCDF GPPCDFK GH.III GHJ... LLJMM. Gerakan 1: J-u AAB..F ..BCDF GPPCDFK GHJIII GHJ... LL.MM. Gerakan 2: L-r piece.c: 1 AAB..F ..BCDF GPPCDFK GHJIII GHJ... .LLMM. Gerakan 3: C-u

	AABC.F ..BCDF GPP.DFK GHJII GHJ... .LLMM. Gerakan 4: P-r piece.c: 2 AABC.F ..BCDF G.PPDFK GHJII GHJ... .LLMM. Gerakan 5: D-u AABCD F ..BCDF G.PP.FK GHJII GHJ... .LLMM. Gerakan 6: P-r piece.c: 3 AABCD F ..BCDF G..PPFK GHJII GHJ... .LLMM. Gerakan 7: B-d AA.CDF ..BCDF G.BPPFK GHJII GHJ... .LLMM. Gerakan 8: A-r piece.c: 1 .AACDF ..BCDF G.BPPFK
--	---

		<p>GHJIII GHJ... .LLMM. Gerakan 9: L-l</p> <p>.AACDF ..BCDF G.BPPFK GHJIII GHJ... LL.MM. Gerakan 10: J-d</p> <p>.AACDF ..BCDF G.BPPFK GH.III GHJ... LLJMM. Gerakan 11: I-l</p> <p>.AACDF ..BCDF G.BPPFK GH.III. GHJ... LLJMM. Gerakan 12: F-d</p> <p>.AACD. ..BCD. G.BPP.K GH.IIF GHJ..F LLJMMF Gerakan 13: P-r piece.c: 4</p> <p>.AACD. ..BCD. G.B.PPK GH.IIF GHJ..F LLJMMF Gerakan 14: C-d</p>
--	--	--

		.AA.D. ..BCD. G.BCPPK GHIIIF GHJ..F LLJMMF Gerakan 15: A-r piece.c: 2 ..AAD. ..BCD. G.BCPPK GHIIIF GHJ..F LLJMMF Gerakan 16: G-u ..AAD. G.BCD. G.BCPPK GHIIIF .HJ..F LLJMMF Gerakan 17: G-u G.AAD. G.BCD. G.BCPPK .HIIIF .HJ..F LLJMMF Gerakan 18: H-u GHAAD. GHBCD. G.BCPPK ..IIIF ..J..F LLJMMF Gerakan 19: Primary piece bergerak keluar dari papan! GHAAD. GHBCD. G.BC..K ..IIIF
--	--	--

		<p>..J..F LLJMMF</p> <p>-----</p> <p>Waktu pencarian (ms) : 135 Banyak node dikunjungi: 1168</p> <p>-----</p>
2.	<p>UCS 6 6 5 ..BBBC ..A..C PPA..CKDDEE</p>	<p>Nama File Input: test2.txt ----ALGORITMA PENCARIAN----- 1. Uniform Cost Search (UCS) 2. Greedy Best First Search 3. A* ----- Algoritma Pencarian (1-3): 1 Mencari solusi... -----RUTE MENUJU SOLUSI----- Papan Awal ..BBBC ..A..C PPA..CKDDEE Gerakan 1: A-d ..BBBCC PPA..CK ..A... ..DDEE Gerakan 2: C-d ..BBB.C PPA..CK ..A..C ..DDEE Gerakan 3: B-r piece.c: 3 ...BBBC PPA..CK</p>

		<p>..A..C ..DDEE Gerakan 4: A-u</p> <p>...BBB ..A..C PPA..CK C ..DDEE Gerakan 5: A-u</p> <p>..ABBB ..A..C PP..CK C ..DDEE Gerakan 6: P-r piece.c: 3</p> <p>..ABBB ..A..C ...PPCK C ..DDEE Gerakan 7: A-d</p> <p>...BBB ..A..C ..APPCK C ..DDEE Gerakan 8: B-l</p> <p>..BBB. ..A..C ..APPCK C ..DDEE Gerakan 9: D-l</p>
--	--	--

		<p>..BBB. ..A..C ..APPCK C DD..EE Gerakan 10: A-d</p> <p>..BBB. C ...PPCK C DDA.EE ..A... Gerakan 11: E-l</p> <p>..BBB. C ...PPCK C DDAEE. ..A... Gerakan 12: C-d</p> <p>..BBB. PPCK C DDAEEC ..A... Gerakan 13: B-r piece.c: 3</p> <p>...BBB PPCK C DDAEEC ..A... Gerakan 14: P-l</p> <p>...BBB PP..CK C DDAEEC</p>
--	--	---

		<p>..A... Gerakan 15: P-l</p> <p>...BBB PP...CKC DDAEEC ..A... Gerakan 16: C-d</p> <p>...BBB PP...KC DDAEEC ..A..C</p> <p>Gerakan 17: Primary piece bergerak keluar dari papan! ...BBBKC DDAEEC ..A..C</p> <p>----- Waktu pencarian (ms) : 138 Banyak node dikunjungi: 1425 -----</p>
3.	<p>UCS 5 5 6 K DAAA. D..P. BC.P. BCEEE .CFF.</p>	<p>Nama File Input: test3.txt -----ALGORITMA PENCARIAN----- 1. Uniform Cost Search (UCS) 2. Greedy Best First Search 3. A*</p> <p>----- Algoritma Pencarian (1-3): 1 Mencari solusi... -----RUTE MENUJU SOLUSI----- Papan Awal K DAAA. D..P. BC.P.</p>

		<p>BCEEE .CFF. Gerakan 1: B-d</p> <p>K DAAA. D..P. .C.P. BCEEE BCFF. Gerakan 2: C-u</p> <p>K DAAA. DC.P. .C.P. BCEEE B.FF. Gerakan 3: F-l</p> <p>K DAAA. DC.P. .C.P. BCEEE BFF.. Gerakan 4: A-r piece.c: 2</p> <p>K D.AAA DC.P. .C.P. BCEEE BFF.. Gerakan 5: D-d</p> <p>K ..AAA DC.P. DC.P. BCEEE BFF.. Gerakan 6: A-l</p> <p>K</p>
--	--	--

		AAA.. DC.P. DC.P. BCEEE BFF.. Gerakan 7: Primary piece bergerak keluar dari papan! K AAA.. DC.. DC.. BCEEE BFF.. ----- Waktu pencarian (ms) : 5 Banyak node dikunjungi: 25 -----
4.	UCS 5 6 8 .HHHHH .PEDDG .PECCG FFEB.G .AAB.. K	Nama File Input: test4.txt -----ALGORITMA PENCARIAN----- 1. Uniform Cost Search (UCS) 2. Greedy Best First Search 3. A* ----- Algoritma Pencarian (1-3): 1 Mencari solusi... -----RUTE MENUJU SOLUSI----- Tidak ditemukan solusi! ----- Waktu pencarian (ms) : 32 Banyak node dikunjungi: 101 -----
5.	GBFS 6 6 11 AAB..F ..BCDF GPPCDFK GH.III GHJ... LLJMM.	Selamat datang di program Rush Hour Puzzle Solver! Nama File Input: test1.txt -----ALGORITMA PENCARIAN----- 1. Uniform Cost Search (UCS) 2. Greedy Best First Search 3. A* ----- Algoritma Pencarian (1-3): 2 Mencari solusi... -----RUTE MENUJU SOLUSI----- Papan Awal

		AAB..F ..BCDF GPPCDFK GH.III GHJ... LLJMM. Gerakan 1: C-u AABC.F ..BCDF GPP.DFK GH.III GHJ... LLJMM. Gerakan 2: D-u AABCDF ..BCDF GPP..FK GH.III GHJ... LLJMM. Gerakan 3: M-r piece.c: 4 AABCDF ..BCDF GPP..FK GH.III GHJ... LLJ.MM Gerakan 4: I-l AABCDF ..BCDF GPP..FK GH.III. GHJ... LLJ.MM Gerakan 5: M-l AABCDF ..BCDF GPP..FK GH.III. GHJ...
--	--	--

		LLJMM. Gerakan 6: F-d AABCD. ..BCD. GPP...K GHIIF GHJ..F LLJMMF Gerakan 7: Primary piece bergerak keluar dari papan! AABCD. ..BCD. G....K GHIIF GHJ..F LLJMMF ----- Waktu pencarian (ms) : 8 Banyak node dikunjungi: 26 -----
6.	GBFS 6 6 5 ..BBBC ..A..C PPA..CKDDEE	Nama File Input: test2.txt ----ALGORITMA PENCARIAN----- 1. Uniform Cost Search (UCS) 2. Greedy Best First Search 3. A* ----- Algoritma Pencarian (1-3): 2 Mencari solusi... -----RUTE MENUJU SOLUSI----- Papan Awal ..BBBC ..A..C PPA..CKDDEE Gerakan 1: D-l ..BBBC ..A..C PPA..CK DD..EE

		<p>.....</p> <p>Gerakan 2: A-d</p> <p>..BBBC</p> <p>.....C</p> <p>PP...CK</p> <p>..A...</p> <p>DDA.EE</p> <p>.....</p> <p>Gerakan 3: P-r</p> <p>piece.c: 1</p> <p>..BBBC</p> <p>.....C</p> <p>..PP..CK</p> <p>..A...</p> <p>DDA.EE</p> <p>.....</p> <p>Gerakan 4: E-l</p> <p>..BBBC</p> <p>.....C</p> <p>..PP..CK</p> <p>..A...</p> <p>DDAEE.</p> <p>.....</p> <p>Gerakan 5: C-d</p> <p>..BBB.</p> <p>.....</p> <p>..PP...K</p> <p>..A..C</p> <p>DDAEEC</p> <p>.....C</p> <p>Gerakan 6: P-l</p> <p>..BBB.</p> <p>.....</p> <p>PP...K</p> <p>..A..C</p> <p>DDAEEC</p> <p>.....C</p> <p>Gerakan 7: Primary piece bergerak keluar dari papan!</p> <p>..BBB.</p> <p>.....</p>
--	--	---

		<p>.....K ..A..C DDAEEC C</p> <p>-----</p> <p>Waktu pencarian (ms) : 1 Banyak node dikunjungi: 13</p> <p>-----</p>
7.	<p>GBFS 5 5 6 K DAAA. D..P. BC.P. BCEEE .CFF.</p>	<p>Nama File Input: test3.txt ----ALGORITMA PENCARIAN----- 1. Uniform Cost Search (UCS) 2. Greedy Best First Search 3. A*</p> <p>-----</p> <p>Algoritma Pencarian (1-3): 2 Mencari solusi...</p> <p>-----RUTE MENUJU SOLUSI-----</p> <p>Papan Awal K DAAA. D..P. BC.P. BCEEE .CFF. Gerakan 1: F-r piece.c: 3</p> <p>K DAAA. D..P. BC.P. BCEEE .C.FF Gerakan 2: C-u</p> <p>K DAAA. DC.P. BC.P. BCEEE ...FF Gerakan 3: F-l</p> <p>K</p>

		DAAA. DC.P. BC.P. BCEEE .FF.. Gerakan 4: F-l K DAAA. DC.P. BC.P. BCEEE FF.. Gerakan 5: A-r piece.c: 2 K D.AAA DC.P. BC.P. BCEEE FF.. Gerakan 6: C-u K DCAAA DC.P. BC.P. B.EEE FF.. Gerakan 7: F-r piece.c: 1 K DCAAA DC.P. BC.P. B.EEE .FF.. Gerakan 8: B-d K DCAAA DC.P. .C.P. B.EEE
--	--	---

		<p>BFF.. Gerakan 9: D-d</p> <p>K .CAAA DC.P. DC.P. B.EEE BFF.. Gerakan 10: C-d</p> <p>K ..AAA DC.P. DC.P. BCEEE BFF.. Gerakan 11: A-l</p> <p>K AAA.. DC.P. DC.P. BCEEE BFF.. Gerakan 12: Primary piece bergerak keluar dari papan! K AAA.. DC... DC... BCEEE BFF.. ----- Waktu pencarian (ms) : 3 Banyak node dikunjungi: 37 -----</p>
8.	<p>GBFS 5 6 8 .HHHHH .PEDDG .PECCG</p>	<p>Nama File Input: test4.txt -----ALGORITMA PENCARIAN----- 1. Uniform Cost Search (UCS) 2. Greedy Best First Search 3. A* -----</p>

	FFEB.G .AAB.. K	<p>Algoritma Pencarian (1-3): 2</p> <p>Mencari solusi...</p> <p>-----RUTE MENUJU SOLUSI-----</p> <p>Tidak ditemukan solusi!</p> <p>-----</p> <p>Waktu pencarian (ms) : 4</p> <p>Banyak node dikunjungi: 55</p> <p>-----</p>
9.	A* 6 6 11 AAB..F ..BCDF GPPCDFK GH.III GHJ... LLJMM.	<p>Selamat datang di program Rush Hour Puzzle Solver!</p> <p>Nama File Input: test1.txt</p> <p>----ALGORITMA PENCARIAN----</p> <p>1. Uniform Cost Search (UCS)</p> <p>2. Greedy Best First Search</p> <p>3. A*</p> <p>-----</p> <p>Algoritma Pencarian (1-3): 3</p> <p>Mencari solusi...</p> <p>-----RUTE MENUJU SOLUSI-----</p> <p>Papan Awal</p> <p>AAB..F ..BCDF GPPCDFK GH.III GHJ... LLJMM.</p> <p>Gerakan 1: C-u</p> <p>AABC.F ..BCDF GPP.DFK GH.III GHJ... LLJMM.</p> <p>Gerakan 2: P-r piece.c: 2</p> <p>AABC.F ..BCDF G.PPDFK GH.III GHJ... LLJMM.</p> <p>Gerakan 3: D-u</p>

		AABCD F ..BCDF G.PP.FK GH.III GHJ... LLJMM. Gerakan 4: J-u AABCD F ..BCDF G.PP.FK GHJIII GHJ... LL.MM. Gerakan 5: L-r piece.c: 1 AABCD F ..BCDF G.PP.FK GHJIII GHJ... .LLMM. Gerakan 6: P-r piece.c: 3 AABCD F ..BCDF G..PPFK GHJIII GHJ... .LLMM. Gerakan 7: B-d AA.CDF ..BCDF G.BPPFK GHJIII GHJ... .LLMM. Gerakan 8: A-r piece.c: 1 .AACDF ..BCDF G.BPPFK
--	--	--

		<p>GHJII GHJ... .LLMM. Gerakan 9: L-l</p> <p>.AACDF ..BCDF G.BPPFK GHJII GHJ... LL.MM. Gerakan 10: J-d</p> <p>.AACDF ..BCDF G.BPPFK GH.III GHJ... LLJMM. Gerakan 11: I-l</p> <p>.AACDF ..BCDF G.BPPFK GH.III. GHJ... LLJMM. Gerakan 12: F-d</p> <p>.AACD. ..BCD. G.BPP.K GH.IIF GHJ..F LLJMMF Gerakan 13: P-r piece.c: 4</p> <p>.AACD. ..BCD. G.B.PPK GH.IIF GHJ..F LLJMMF Gerakan 14: C-d</p>
--	--	--

		.AA.D. ..BCD. G.BCPPK GHIIIF GHJ..F LLJMMF Gerakan 15: A-r piece.c: 2 ..AAD. ..BCD. G.BCPPK GHIIIF GHJ..F LLJMMF Gerakan 16: G-u ..AAD. G.BCD. G.BCPPK GHIIIF .HJ..F LLJMMF Gerakan 17: G-u G.AAD. G.BCD. G.BCPPK .HIIIF .HJ..F LLJMMF Gerakan 18: H-u GHAAD. GHBCD. G.BCPPK ..IIIF ..J..F LLJMMF Gerakan 19: Primary piece bergerak keluar dari papan! GHAAD. GHBCD. G.BC..K ..IIIF
--	--	---

		...J..F LLJMMF ----- Waktu pencarian (ms) : 146 Banyak node dikunjungi: 497 -----
10.	A* 6 6 5 ..BBBC ..A..C PPA..CK DDEE	Selamat datang di program Rush Hour Puzzle Solver! Nama File Input: test2.txt -----ALGORITMA PENCARIAN----- 1. Uniform Cost Search (UCS) 2. Greedy Best First Search 3. A* ----- Algoritma Pencarian (1-3): 3 Mencari solusi... -----RUTE MENUJU SOLUSI----- Papan Awal ..BBBC ..A..C PPA..CK DDEE Gerakan 1: A-d ..BBBCC PPA..CK ..A... ..DDEE Gerakan 2: C-d ..BBB.C PPA..CK ..A..C ..DDEE Gerakan 3: B-r piece.c: 3 ...BBBC

		PPA..CK ..A..C ..DDEE Gerakan 4: A-u ..ABBB ..A..C PP..CKC ..DDEE Gerakan 5: P-r piece.c: 3 ..ABBB ..A..C ...PPCKC ..DDEE Gerakan 6: A-d ...BBBC ..APPCK ..A..C ..DDEE Gerakan 7: B-l ..BBB.C ..APPCK ..A..C ..DDEE Gerakan 8: C-u ..BBBCC ..APPCK ..A... ..DDEE Gerakan 9: D-l
--	--	---

		<p> ..BBBC C ..APPCK ..A... DD..EE Gerakan 10: A-d </p> <p> ..BBBC C ...PPCK DDA.EE ..A... Gerakan 11: E-l </p> <p> ..BBBC C ...PPCK DDAEE. ..A... Gerakan 12: C-d </p> <p> ..BBB. PP.K C DDAEEC ..A..C Gerakan 13: P-l </p> <p> ..BBB. PP...K C DDAEEC ..A..C Gerakan 14: Primary piece bergerak keluar dari papan! ..BBB. K C </p>
--	--	--

		DDAEEC ..A..C ----- Waktu pencarian (ms) : 80 Banyak node dikunjungi: 541 -----
11.	A* 5 5 6 K DAAA. D..P. BC.P. BCEEE .CFF.	Nama File Input: test3.txt ----ALGORITMA PENCARIAN---- 1. Uniform Cost Search (UCS) 2. Greedy Best First Search 3. A* ----- Algoritma Pencarian (1-3): 3 Mencari solusi... -----RUTE MENUJU SOLUSI----- Papan Awal K DAAA. D..P. BC.P. BCEEE .CFF. Gerakan 1: B-d K DAAA. D..P. .C.P. BCEEE BCFF. Gerakan 2: C-u K DAAA. DC.P. .C.P. BCEEE B.FF. Gerakan 3: F-l K DAAA. DC.P. .C.P.

		<p>BCEEE BFF.. Gerakan 4: A-r piece.c: 2</p> <p>K D.AAA DC.P. .C.P. BCEEE BFF.. Gerakan 5: D-d</p> <p>K ..AAA DC.P. DC.P. BCEEE BFF.. Gerakan 6: A-l</p> <p>K AAA.. DC.P. DC.P. BCEEE BFF.. Gerakan 7: Primary piece bergerak keluar dari papan! K AAA.. DC... DC... BCEEE BFF.. ----- Waktu pencarian (ms) : 5 Banyak node dikunjungi: 20 -----</p>
12.	<p>A* 5 6 8 .HHHHH</p>	<p>Selamat datang di program Rush Hour Puzzle Solver! Nama File Input: test4.txt -----ALGORITMA PENCARIAN----- 1. Uniform Cost Search (UCS)</p>

	.PEDDG .PECCG FFEB.G .AAB.. K	2. Greedy Best First Search 3. A* ----- Algoritma Pencarian (1-3): 3 Mencari solusi... -----RUTE MENUJU SOLUSI----- Tidak ditemukan solusi! ----- Waktu pencarian (ms) : 51 Banyak node dikunjungi: 101 -----
--	---	---

BAB VI

ANALISIS ALGORITMA

4.1. Algoritma UCS

Secara teoritis, kompleksitas waktu dari algoritma UCS adalah $O(b^m)$, dengan b merupakan *branching factor* (banyaknya percabangan maksimal yang mungkin dari suatu *node*) dan m merupakan kedalaman maksimum dari pohon yang terbentuk (banyak langkah menuju solusi). Namun, perlu diperhatikan bahwa kompleksitas waktu tersebut hanya memperhitungkan algoritma UCS saja tanpa memperhitungkan proses komputasi lainnya. Sebagai contoh, jika memperhitungkan kompleksitas waktu pemasukan kondisi papan ke dalam *priority queue* diperhitungkan, yaitu $O(\log n)$ pada setiap pemrosesan *priority queue*, kompleksitas waktu pada skenario terburuk akan menjadi $O(b^d * \log(b^d))$, yang berarti algoritma akan membangkitkan b^d node dan setiap node akan melakukan pemrosesan *priority queue* dengan kompleksitas waktu $O(\log(b^d))$.

4.2 Algoritma Best First Search

Secara teoritis, kompleksitas waktu dari algoritma *Best First Search* adalah $O(b^m)$, dengan b merupakan *branching factor* (banyaknya percabangan maksimal yang mungkin dari suatu *node*) dan m merupakan kedalaman maksimum dari pohon yang terbentuk (banyak langkah menuju solusi). Namun, perlu diperhatikan bahwa kompleksitas waktu tersebut hanya memperhitungkan algoritma UCS saja tanpa memperhitungkan proses komputasi lainnya. Sebagai contoh, jika memperhitungkan kompleksitas waktu pemasukan kondisi papan ke dalam *priority queue* diperhitungkan, yaitu $O(\log n)$ pada setiap pemrosesan *priority queue*, kompleksitas waktu pada skenario terburuk akan menjadi $O(b^d * \log(b^d))$, yang berarti algoritma akan membangkitkan b^d node dan setiap node akan melakukan pemrosesan *priority queue* dengan kompleksitas waktu $O(\log(b^d))$.

4.3 Algoritma A*

Secara teoritis, kompleksitas waktu dari algoritma A* adalah $O(b^m)$, dengan b merupakan *branching factor* (banyaknya percabangan maksimal yang mungkin dari suatu *node*) dan m merupakan kedalaman maksimum dari pohon yang terbentuk (banyak langkah menuju solusi). Namun, perlu diperhatikan bahwa kompleksitas waktu tersebut hanya memperhitungkan algoritma A* saja tanpa memperhitungkan proses komputasi lainnya. Sebagai contoh, jika memperhitungkan kompleksitas waktu pemasukan kondisi papan ke dalam *priority queue* diperhitungkan, yaitu $O(\log n)$ pada setiap pemrosesan *priority queue*, kompleksitas waktu pada skenario terburuk akan menjadi $O(b^d * \log(b^d))$, yang berarti algoritma akan membangkitkan b^d node dan setiap node akan melakukan pemrosesan *priority queue* dengan kompleksitas waktu $O(\log(b^d))$.

BAB VII

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Dari tugas ini, kami berhasil menerapkan algoritma UCS (*Uniform Cost Search*), Greedy Best First Search, dan A* dalam proses *searching solution* pada permainan *Rush Hour*. Kami semakin paham bagaimana cara menggunakan algoritma-algoritma tersebut dalam penyelesaian masalah komputasi.

5.2 Saran

Berikut merupakan beberapa saran pengembangan dari program yang telah dikembangkan:

1. Mengembangkan alternatif heuristik yang jauh lebih kompleks
2. Mengembangkan program untuk dapat mengatasi *piece* dengan lebar lebih dari 1 satuan, banyak *exit*, dan variasi permainan lainnya
3. Mengembangkan pencarian solusi dengan alternatif algoritma lain dan membandingkan hasilnya dengan tiga algoritma yang telah dikembangkan

5.3 Refleksi

Setelah menyelesaikan tugas ini, ada beberapa hal yang dapat kami jadikan sebagai bahan refleksi untuk kedepannya. Hal pertama adalah sisihkan cukup waktu untuk menentukan konsep dasar dan struktur data yang ingin digunakan secara matang supaya tidak terjadi perubahan di tengah pengerjaan. Kedua, buat metode yang modular supaya dapat digunakan kembali di fungsi lain sehingga kode bisa lebih bersih dan mudah dibaca. Pembagian kerja tugas juga sudah diusahakan seimbang satu sama lain. Komunikasi dalam pengerjaan tugas ini juga sudah cukup baik dan kami dapat bekerja sama dengan baik satu sama lain.

LAMPIRAN

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

Tautan repository : https://github.com/bill2247/Tucil3_13523066_13523072

DAFTAR PUSTAKA

Oracle. 2025. The Java™ Tutorials diakses pada tanggal 20 Mei 2025 pada tautan <https://docs.oracle.com/javase/tutorial/>

GeeksforGeeks. 2025. Uniform Cost Search (UCS) Algorithm diakses pada tanggal 20 Mei 2025 pada tautan <https://www.geeksforgeeks.org/uniform-cost-search-dijkstra-for-large-graphs/>

GeeksforGeeks. 2025. Greedy Best First Search Algorithm diakses pada tanggal 20 Mei 2025 pada tautan <https://www.geeksforgeeks.org/greedy-best-first-search-algorithm/>

Red Blob Games. 2025. A* Pathfinding for Beginners diakses pada tanggal 20 Mei 2025 pada tautan <https://www.redblobgames.com/pathfinding/a-star/introduction.html>

ThinkFun. 2025. Rush Hour Puzzle Overview diakses pada tanggal 20 Mei 2025 pada tautan <https://www.thinkfun.com/products/rush-hour/>

Wikipedia. 2025. Rush Hour (puzzle) diakses pada tanggal 20 Mei 2025 pada tautan [https://en.wikipedia.org/wiki/Rush_Hour_\(puzzle\)](https://en.wikipedia.org/wiki/Rush_Hour_(puzzle))

Munir, Rinaldi. 2025. Penentuan Rute (Part 1) diakses pada tanggal 20 Mei 2025 pada tautan [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)

Munir, Rinaldi. 2025. Penentuan Rute (Part 2) diakses pada tanggal 20 Mei 2025 pada tautan [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/22-Route-Planning-\(2025\)-Bagian2.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/22-Route-Planning-(2025)-Bagian2.pdf)