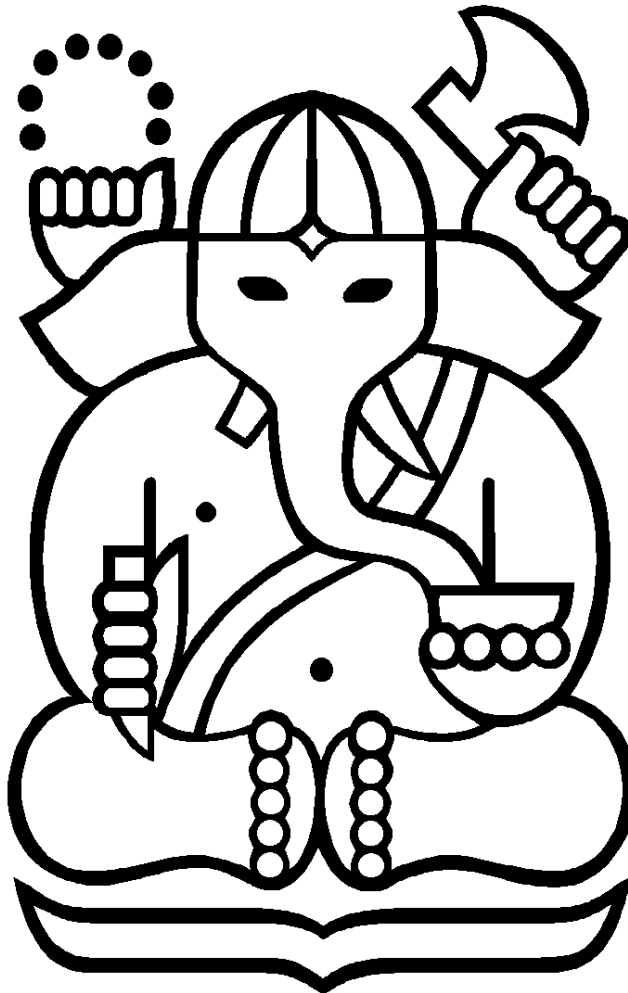


Laporan Tugas Kecil 3

Penyelesaian Puzzle Rush Hour dengan Algoritma Pathfinding
IF2211 - Strategi Algoritma



Oleh:

Orvin Andika Ikhsan Abhista - 13523017

Fajar Kurniawan - 13523027

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
2024 / 2025

Daftar Isi

Daftar Isi.....	2
Bab 1.....	3
Penjelasan Algoritma.....	3
1.1. Algoritma Uniform Cost Search.....	3
1.2. Algoritma Greedy Best First Search.....	3
1.3. Algoritma A*	3
1.4. Algoritma Beam Search.....	3
Bab 2.....	4
Analisis Algoritma.....	4
Bab 3.....	5
Source Code Program.....	5
3.1. Main.java.....	5
3.2. Papan.java.....	5
3.3. Piece.java.....	25
3.4. State.java.....	28
3.5. UCS.java.....	39
3.6. GBFS.java.....	40
3.7. AStarSearch.java.....	41
3.8. BeamSearch.java.....	42
Bab 4.....	45
Tangkapan Layar.....	45
4.1. Test Case 1.....	45
4.2. Test Case 2.....	51
4.3. Test Case 3.....	57
4.4. Test Case 4.....	60
Bab 5.....	62
Analisis Percobaan.....	62
Bab 6.....	63
Bonus.....	63
6.1. Algoritma Pathfinding Tambahan.....	63
6.2. Heuristik Tambahan.....	63
Lampiran.....	64

Bab 1

Penjelasan Algoritma

1.1. Algoritma Uniform Cost Search

Uniform Cost Search adalah algoritma yang fokus mengeksplorasi node dengan biaya total terkecil dari titik awal, tanpa menggunakan informasi tentang seberapa jauh node tersebut dari tujuan. Uniform Cost Search bekerja dengan mengeksplorasi node dengan biaya terkecil lebih dahulu. Algoritma ini menjamin akan menemukan jalur dengan biaya minimum jika jalur tersebut ada. Algoritma ini cocok untuk graf dengan langkah tidak sama dengan biaya. Algoritma ini biasanya akan menggunakan struktur data Priority Queue.

1.2. Algoritma Greedy Best First Search

Greedy Best-First Search adalah algoritma yang menggunakan heuristik untuk menentukan simpul mana yang akan dikunjungi setelahnya. Algoritma ini akan memilih simpul yang diperkirakan memiliki jarak paling dekat dengan tujuan, tidak peduli biaya yang sudah dikeluarkan dalam pencarian. Heuristik yang tidak terlalu bagus bisa menyesatkan algoritma ini sehingga akan terjebak pada maksimum lokal dan gagal menemukan solusi walaupun sebenarnya solusi itu ada. Oleh karena itu, algoritma ini disebut sebagai algoritma yang tidak lengkap.

1.3. Algoritma A*

Algoritma A* adalah algoritma yang menggabungkan pendekatan algoritma UCS dan GBFS. Algoritma A* menentukan simpul mana yang akan dikunjungi selanjutnya dengan mempertimbangkan jumlah total biaya yang sudah diambil dan estimasi jarak simpul sekarang ke simpul tujuan. Dengan ini, algoritma A* akan secara efisien mencari solusi yang optimal dengan mempertimbangkan baik biaya yang sudah dilakukan maupun potensi langkah di masa depan. Pendekatan ini membuat A* biasanya bisa menemukan solusi lebih cepat dan optimal dibanding UCS dan GBFS.

1.4. Algoritma Beam Search

Beam Search adalah algoritma yang membatasi jumlah state atau kandidat yang dieksplorasi pada setiap langkahnya. Algoritma ini bekerja dengan mempertahankan hanya sejumlah tetap state terbaik, yang disebut dengan *beam width*, berdasarkan biaya aktual dari awal hingga state tersebut dan estimasi heuristik biaya dari state ke tujuan. Dengan cara ini, Beam Search menjaga efisiensi dan menghindari eksplorasi seluruh ruang pencarian yang sangat besar, walaupun hal ini dapat mengorbankan jaminan menemukan solusi optimal. Algoritma ini sangat berguna pada masalah dengan ruang pencarian yang besar dan ketika waktu atau sumber daya komputasi terbatas.

Bab 2

Analisis Algoritma

Dalam algoritma pencarian, $g(n)$ adalah biaya yang sudah dikeluarkan dari simpul awal ke simpul sekarang. Dalam penyelesaian puzzle Rush Hour, $g(n)$ adalah jumlah langkah yang sudah dilakukan. Karena tiap langkah pada penyelesaian puzzle Rush Hour dianggap memiliki biaya sama, $g(n)$ dapat dihitung dengan jumlah langkah yang diambil. Sementara itu, $f(n)$ adalah total biaya, baik yang sudah dikeluarkan maupun yang berpotensi akan dikeluarkan. Karena merupakan total biaya, $f(n)$ berbeda-beda untuk tiap algoritma. Untuk algoritma Uniform Cost Search, karena hanya mempertimbangkan biaya nyata, nilai $f(n)$ akan sama dengan $g(n)$. Pada algoritma Greedy Best-Fit Search, biaya yang sudah dikeluarkan diabaikan sehingga nilai $f(n)$ akan sama dengan $h(n)$ atau nilai heuristik. Di algoritma A* dan Beam Search, algoritma mempertimbangkan biaya yang sudah dikeluarkan dan estimasi ke simpul tujuan sehingga nilai $f(n)$ akan sama dengan $g(n)+h(n)$.

Heuristik dianggap admissible jika tidak pernah melebihi biaya sebenarnya dari simpul sekarang ke simpul tujuan. Heuristik yang digunakan pada algoritma A* adalah jumlah mobil yang menghalangi mobil utama dan pintu keluar. Heuristik ini admissible karena agar mobil utama bisa keluar, semua mobil yang menghalangi harus dipindahkan, jadi jumlah mobil yang menghalangi pasti lebih sedikit atau sama dengan jumlah langkah sebenarnya yang diperlukan untuk mencapai simpul tujuan. Bahkan dalam beberapa kasus, untuk memindahkan satu mobil yang menghalangi bisa membutuhkan lebih dari 1 langkah, misalnya ketika harus menggeser mobil lain terlebih dahulu. Oleh karena itu, heuristik ini admissible karena selalu *underestimate* biaya sebenarnya.

Pada penyelesaian puzzle Rush Hour, setiap langkah memiliki biaya yang sama. UCS akan memilih node dengan biaya kumulatif terendah, yang berarti jumlah langkah paling sedikit. Hal ini sama dengan BFS yang menjelajahi semua simpul dengan kedalaman sama terlebih dahulu. Dengan demikian, urutan node yang dibangkitkan dan jalur solusi UCS akan sama dengan BFS.

Secara teoritis, algoritma A* akan lebih efisien dibandingkan algoritma UCS. A* memanfaatkan heuristik untuk mempersempit ruang pencarian. A* akan memprioritaskan jalur-jalur yang lebih menjanjikan sehingga akan mengurangi jumlah simpul yang dieksplorasi dibandingkan UCS yang akan tetap menjelajahi semua kemungkinan berdasarkan jumlah langkah saja. Efisiensi ini akan semakin meningkat seiring dengan meningkatnya kesulitan puzzle.

Algoritma Greedy Best-Fit Search tidak menjamin solusi optimal. GBFS hanya mempertimbangkan heuristik tanpa mempertimbangkan biaya yang sudah dikeluarkan. Akibatnya, algoritma ini bisa memilih jalur yang tampak dekat dengan tujuan tetapi sebenarnya membutuhkan lebih banyak langkah atau bahkan tidak menemukan solusi jika terjebak pada jalur buntu yang tampak menjanjikan. Oleh karena itu, algoritma ini tidak dapat menjamin solusi optimal.

Bab 3

Source Code Program

3.1. Main.java

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Map;

public class Main {
    public static void main(String[] args) {
        BufferedReader reader = new BufferedReader(new
InputStreamReader(System.in));

        while (true) {
            System.err.println("Masukkan nama file (atau
ketik 'exit' untuk keluar):");
            String fileName;
            try {
                fileName = reader.readLine();
                if (fileName == null ||
fileName.trim().equalsIgnoreCase("exit")) {
                    System.err.println("Program
dihentikan.");
                    break;
                }
            } catch (IOException e) {
                System.err.println("Error reading input: " +
e.getMessage());
                return;
            }

            Papan papan = new Papan();
            try {
                papan.readFromFile(fileName);
            } catch (RuntimeException e) {
                System.err.println("Gagal membaca file atau
data tidak valid: " + e.getMessage());
            }
        }
    }
}
```

```

        continue;
    }

    Map<Character, Piece> pieces =
papan.getMapPiece();
    pieces.put('P', papan.getPrimaryPiece());
    papan.printPapan();

    System.err.println("Pilih algoritma
pencarian:");
    System.err.println("1. UCS");
    System.err.println("2. GBFS");
    System.err.println("3. A*");
    System.err.println("4. Beam Search");
    System.err.println("5. Keluar");

    int choice;
    try {
        String input = reader.readLine();
        if (input == null ||
input.trim().equals("5")) {
            System.err.println("Program
dihentikan.");
            break;
        }
        choice = Integer.parseInt(input);
    } catch (IOException | NumberFormatException e)
{
        System.err.println("Input pilihan tidak
valid: " + e.getMessage());
        continue;
    }

    try {
        State initialState = new State(pieces,
papan, 0, null, "State Awal");
        switch (choice) {
            case 1:
                System.err.println("Menggunakan
Algoritma UCS");
                UCS.uniformCostSearch(initialState);
                break;

```

```

        case 2:
            System.err.println("Menggunakan
Algoritma GBFS");
            // GBFS.solve(initialState)
            break;
        case 3:
            System.err.println("Menggunakan
Algoritma A*");
            AStarSearch.solve(initialState);
            break;
        case 4:
            System.err.println("Menggunakan
Algoritma Beam Search");
            BeamSearch.solve(initialState);
            break;
        default:
            System.err.println("Pilihan tidak
valid.");
    }
} catch (Exception e) {
    System.err.println("Terjadi kesalahan saat
menjalankan algoritma: " + e.getMessage());
}
}
}
}

```

3.2. Papan.java

```

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

```

```

/**
!!!!TODO
Kasih validasi kalau format penulisan di file salah
*/

public class Papan {
    private char[][] papan;
    private int baris;
    private int kolom;
    private int jumlahNon;
    private char charUtama;
    private int keluarX;
    private int keluarY;
    private List<Piece> listNonPiece = new ArrayList<>();
    private Piece primaryPiece;

    public Papan(int baris, int kolom) {
        this.baris = baris;
        this.kolom = kolom;
        this.charUtama = 'P';
        this.papan = new char[baris][kolom];
        this.jumlahNon = 0;

        for (int i = 0; i < baris; i++) {
            for (int j = 0; j < kolom; j++) {
                papan[i][j] = ' ';
            }
        }
    }

    // Deep copy constructor
    public Papan(Papan papan) {
        this.baris = papan.baris;
        this.kolom = papan.kolom;
        this.charUtama = papan.charUtama;
        this.jumlahNon = papan.jumlahNon;
        this.papan = new char[baris][kolom];
        for (int i = 0; i < baris; i++) {
            System.arraycopy(papan.papan[i], 0,
this.papan[i], 0, kolom);
        }
    }
}

```



```

        this.keluarX = papan.keluarX;
        this.keluarY = papan.keluarY;
        this.listNonPiece = new
ArrayList<>(papan.listNonPiece);
        this.primaryPiece = papan.primaryPiece;
    }

    public Papan() {
        this.baris = 0;
        this.kolom = 0;
        this.charUtama = 'P';
        this.papan = new char[baris][kolom];
        this.jumlahNon = 0;
    }

    public void setChar(int baris, int kolom, char c) {
        if (baris >= 0 && baris < this.baris && kolom >= 0
&& kolom < this.kolom) {
            papan[baris][kolom] = c;
        }
    }

    public char getChar(int baris, int kolom) {
        if (baris >= 0 && baris < this.baris && kolom >= 0
&& kolom < this.kolom) {
            return papan[baris][kolom];
        }
        return ' ';
    }

    public void setCharUtama(char c) {
        this.charUtama = c;
    }

    public char getCharUtama() {
        return this.charUtama;
    }

    public void setJumlahNon(int jumlahNon) {
        this.jumlahNon = jumlahNon;
    }

    public int getJumlahNon() {
        return this.jumlahNon;
    }

    public void setPapan(char[][] papan) {
        this.papan = papan;
    }

```

```

    }
    public char[][] getPapan() {
        return this.papan;
    }
    public int getBaris() {
        return this.baris;
    }
    public int getKolom() {
        return this.kolom;
    }
    public void printPapan() {
        if (keluarY == -1){
            for (int i = 0; i < kolom; i++) {
                if (i != keluarX) {
                    System.out.print(" ");
                } else {
                    System.out.print("K");
                }
            }
            System.out.println();
            for (int i = 0; i < baris; i++) {
                for (int j = 0; j < kolom; j++) {
                    System.out.print(papan[i][j] + " ");
                }
                System.out.println();
            }
        } else if (keluarX == -1) {
            for (int i = 0; i < baris; i++) {
                if (i != keluarY) {
                    System.out.print(" ");
                    for (int j = 0; j < kolom; j++) {
                        System.out.print(papan[i][j] + " ");
                    }
                } else {
                    System.out.print("K ");
                    for (int j = 0; j < kolom; j++) {
                        System.out.print(papan[i][j] + " ");
                    }
                }
                System.out.println();
            }
        } else if (keluarX == this.kolom) {

```

```

        for (int i = 0; i < baris; i++) {
            for (int j = 0; j < kolom; j++) {
                System.out.print(papan[i][j] + " ");
            }
            if (i == keluarY) {
                System.out.print("K");
            }
            System.out.println();
        }
    } else {
        for (int i = 0; i < baris+1; i++) {
            if (i == keluarY) {
                for (int j = 0; j < kolom; j++) {
                    if (j == keluarX) {
                        System.out.print("K ");
                    } else {
                        System.out.print("  ");
                    }
                }
                System.out.println();
            } else {
                for (int j = 0; j < kolom; j++) {
                    System.out.print(papan[i][j] + " ");
                }
                System.out.println();
            }
        }
    }
}

public void clearPapan() {
    for (int i = 0; i < baris; i++) {
        for (int j = 0; j < kolom; j++) {
            papan[i][j] = ' ';
        }
    }
}

public void readFromFile(String fileName) {
    try (BufferedReader br = new BufferedReader(new
FileReader(fileName))) {
        String line;
        int i = 0;

```

```

        // Read the first line to get the dimensions
        line = br.readLine();
        if (line != null) {
            String[] dimensions = line.split(" ");
            this.baris =
Integer.parseInt(dimensions[0]);
            System.out.println("baris: " + baris);
            this.kolom =
Integer.parseInt(dimensions[1]);
            System.out.println("kolom: " + kolom);
            this.papan = new char[baris][kolom];
        }
        char[][] tempPapan = new char[baris+1][kolom+1];
        //fill the tempPapan with ' '
        for (int x = 0; x < baris + 1; x++) {
            for (int y = 0; y < kolom + 1; y++) {
                tempPapan[x][y] = ' ';
            }
        }
        // Read the next one line to read into jumlahNon
attribute
        line = br.readLine();
        if (line != null) {
            this.jumlahNon = Integer.parseInt(line);
        }
        // Read the rest of the lines to fill the board
        while ((line = br.readLine()) != null && i <
baris + 1) {
            for (int j = 0; j < kolom + 1; j++) {
                if (j < line.length()) {
                    tempPapan[i][j] = line.charAt(j);
                    if (line.charAt(j) == 'K') {
                        this.keluarX = j;
                        this.keluarY = i;
                        if (keluarY == 0 && keluarX <
this.kolom) {
                            this.keluarY = -1;
                        } else if (keluarX == 0 &&
keluarY < this.baris) {
                            this.keluarX = -1;
                        }
                    }
                }
            }
        }
    }

```

```

    }

    }

    i++;

}

List<Character> items = new ArrayList<>();
for (int k = 0; k < baris+1; k++) {
    for (int j = 0; j < tempPapan[k].length;
j++) {

        char c = tempPapan[k][j];
        if (c != ' ' && c != 'K') {
            items.add(c);
        }
    }
}

// Print all the items
// for (int k = 0; k < items.size(); k++) {
//     System.out.print(items.get(k) + " ");
// }

for (int k = 0; k < baris; k++) {
    for (int j = 0; j < kolom; j++) {
        this.papan[k][j] = items.get(k * kolom +
j);
    }
}

boolean[][] isChecked = new
boolean[baris][kolom];

for (int k = 0; k < baris; k++) {
    for (int j = 0; j < kolom; j++) {
        if (!isChecked[k][j]) {
            isChecked[k][j] = true;
            char c = papan[k][j];
            if (c == '.') continue;
            if (j+1 < this.kolom &&
papan[k][j+1] == c) {
                int count = 1;
                while (j+count < this.kolom &&
papan[k][j+count] == c) {
                    isChecked[k][j+count] =
true;

                    count++;
                }
            }
        }
    }
}

```

```

        Piece piece = new Piece(c,
count, Piece.Orientasi.HORIZONTAL, j, k);
        if (c == charUtama) {
            piece.setPrimary(true);
            this.primaryPiece = piece;
            continue;
        }
        listNonPiece.add(piece);
    } else if (k + 1 < baris &&
papan[k+1][j] == c) {
        int count = 1;
        while (k + count < baris &&
papan[k+count][j] == c) {
            isChecked[k+count][j] =
true;

            count++;
        }
        Piece piece = new Piece(c,
count, Piece.Orientasi.VERTICAL, j, k);
        if (c == charUtama) {
            piece.setPrimary(true);
            this.primaryPiece = piece;
            continue;
        }
        listNonPiece.add(piece);
    }
}

}

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

}

public boolean isFull() {
    for (int i = 0; i < baris; i++) {
        for (int j = 0; j < kolom; j++) {
            if (papan[i][j] == ' ') {
                return false;
            }
        }
    }
}

```

```

    }
    return true;
}

public Papan copy() {
    Papan newPapan = new Papan(this.baris, this.kolom);
    newPapan.setCharUtama(this.charUtama);
    newPapan.setJumlahNon(this.jumlahNon);
    newPapan.setKeluarX(this.keluarX);
    newPapan.setKeluarY(this.keluarY);

    for (int i = 0; i < baris; i++) {
        for (int j = 0; j < kolom; j++) {
            newPapan.setChar(i, j, this.papan[i][j]);
        }
    }

    return newPapan;
}

public boolean isEmpty(int baris, int kolom) {
    if (baris >= 0 && baris < this.baris && kolom >= 0
    && kolom < this.kolom) {
        return papan[baris][kolom] == '.';
    }
    return false;
}

public void saveToFile(String fileName) {
    try (BufferedWriter bw = new BufferedWriter(new
    FileWriter(fileName))) {
        for (int i = 0; i < baris; i++) {
            for (int j = 0; j < kolom; j++) {
                bw.write(papan[i][j]);
            }
            bw.newLine();
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}

```

```

public int getKeluarX() {
    return keluarX;
}
public int getKeluarY() {
    return keluarY;
}
public void setKeluarX(int keluarX) {
    this.keluarX = keluarX;
}
public void setKeluarY(int keluarY) {
    this.keluarY = keluarY;
}
public void addPiece(Piece Piece) {
    listNonPiece.add(Piece);
}
public List<Piece> getListNonPiece() {
    return listNonPiece;
}
public void setListNonPiece(List<Piece> listNonPiece) {
    this.listNonPiece = listNonPiece;
}
public void clearListNonPiece() {
    listNonPiece.clear();
}

public void removePiece(int index) {
    if (index >= 0 && index < listNonPiece.size()) {
        listNonPiece.remove(index);
    }
}
public void removePiece(char hurufPiece) {
    listNonPiece.removeIf(Piece -> Piece.getHurufPiece()
== hurufPiece);
}
public void printAllPiece() {
    for (Piece piece : listNonPiece) {
        piece.printPiece();
    }
}
public void movePiece(char hurufPiece, int jarak) {
    boolean found = false;
    for (Piece piece : listNonPiece) {

```



```

        if (piece.getHurufPiece() == hurufPiece) {
            found = true;
            if (piece.getOrientasi() ==
Piece.Orientasi.HORIZONTAL) {
                if (piece.getX() + jarak >= 0 &&
piece.getX() + jarak < this.kolom) {
                    piece.moveX(jarak);
                    // Clear old position
                    for (int i = 0; i <
piece.getUkuran(); i++) {
                        papan[piece.getY()][piece.getX()
- jarak + i] = '.';
                    }
                    // Set new position
                    for (int i = 0; i <
piece.getUkuran(); i++) {
                        papan[piece.getY()][piece.getX()
+ i] = piece.getHurufPiece();
                    }
                }
            } else {
                if (piece.getY() + jarak >= 0 &&
piece.getY() + jarak < this.baris) {
                    piece.moveY(jarak);
                    // Clear old position
                    for (int i = 0; i <
piece.getUkuran(); i++) {
                        papan[piece.getY() - jarak +
i][piece.getX()] = '.';
                    }
                    // Set new position
                    for (int i = 0; i <
piece.getUkuran(); i++) {
                        papan[piece.getY() +
i][piece.getX()] = piece.getHurufPiece();
                    }
                }
            }
            break;
        }
    }
    if (!found) {

```

```

        throw new IllegalArgumentException("Piece with
character " + hurufPiece + " not found.");
    }
}

public boolean canPrimaryExit() {
    boolean canExit = true;
    if (primaryPiece.getOrientasi() ==
Piece.Orientasi.HORIZONTAL) {
        int x = primaryPiece.getX();
        if (keluarX == -1) {
            x--;
            while (x > -1) {
                if (papan[primaryPiece.getY()][x] ==
'.'.') {

                    x--;
                } else {
                    canExit = false;
                    break;
                }
            }
        } else {
            x = x + primaryPiece.getUkuran();
            while (x < this.kolom) {
                if (papan[primaryPiece.getY()][x] ==
'.'.') {

                    x++;
                } else {
                    canExit = false;
                    break;
                }
            }
        }
    } else {
        int y = primaryPiece.getY();
        if (keluarY == -1) {
            y--;
            while (y > -1) {
                if (papan[y][primaryPiece.getX()] ==
'.'.') {

                    y--;

```

```

        } else {
            canExit = false;
            break;
        }
    }
} else {
    y = y + primaryPiece.getUkuran();
    while (y < this.baris) {
        if (papan[y][primaryPiece.getX()] ==
'.') {
            y++;
        } else {
            canExit = false;
            break;
        }
    }
}
return canExit;
}

public Piece getPrimaryPiece() {
    return primaryPiece;
}

public int movePieceToRightFarthest(char hurufPiece) {
    int moveDistance = 0;
    boolean found = false;
    for (Piece piece : listNonPiece) {
        if (piece.getHurufPiece() == hurufPiece) {
            int oldX = piece.getX();
            found = true;
            if (piece.getOrientasi() ==
Piece.Orientasi.HORIZONTAL) {
                int x = piece.getX() +
piece.getUkuran();
                while (x < this.kolom &&
papan[piece.getY()][x] == '.') {
                    x++;
                }
                piece.setX(x - piece.getUkuran());
                moveDistance = piece.getX() - oldX;
                // Clear old position
                for (int i = 0; i < piece.getUkuran());
            }
        }
    }
}

```

```

i++) {
            papan[piece.getY()][oldX + i] = '.';
        }
        // Set new position
        for (int i = 0; i < piece.getUkuran();
i++) {
            papan[piece.getY()][piece.getX() +
i] = piece.getHurufPiece();
        }
        } else {
            throw new
UnsupportedOperationException("Piece is not horizontal");
        }
    }
    }
    if (!found) {
        throw new IllegalArgumentException("Piece with
character " + hurufPiece + " not found.");
    }
    return moveDistance;
}

public int movePieceToLeftFarthest(char hurufPiece) {
    int moveDistance = 0;
    boolean found = false;
    for (Piece piece : listNonPiece) {
        if (piece.getHurufPiece() == hurufPiece) {
            int oldX = piece.getX();
            found = true;
            if (piece.getOrientasi() ==
Piece.Orientasi.HORIZONTAL) {
                int x = piece.getX() - 1;
                while (x >= 0 && papan[piece.getY()][x]
== '.') {
                    x--;
                }
                piece.setX(x + 1);
                moveDistance = piece.getX() - oldX;
                // Clear old position
                for (int i = 0; i < piece.getUkuran();
i++) {
                    papan[piece.getY()][oldX + i] = '.';
                }
            }
        }
    }
}

```

```

        // Set new position
        for (int i = 0; i < piece.getUkuran();
i++) {
            papan[piece.getY()][piece.getX() +
i] = piece.getHurufPiece();
        }
    } else {
        throw new
UnsupportedOperationException("Piece is not horizontal");
    }
}

}

if (!found) {
    throw new IllegalArgumentException("Piece with
character " + hurufPiece + " not found.");
}

return moveDistance;
}

public int movePieceToUpFarthest(char hurufPiece) {
    int moveDistance = 0;
    boolean found = false;
    for (Piece piece : listNonPiece) {
        if (piece.getHurufPiece() == hurufPiece) {
            int oldY = piece.getY();
            found = true;
            if (piece.getOrientasi() ==
Piece.Orientasi.VERTICAL) {
                int y = piece.getY() - 1;
                while (y >= 0 && papan[y][piece.getX()]
== '.') {
                    y--;
                }
                piece.setY(y + 1);
                moveDistance = piece.getY() - oldY;
                // Clear old position
                for (int i = 0; i < piece.getUkuran();
i++) {
                    papan[oldY + i][piece.getX()] = '.';
                }
                // Set new position
                for (int i = 0; i < piece.getUkuran();
i++) {

```

```

                papan[piece.getY() +
i][piece.getX()] = piece.getHurufPiece();
            }
        } else {
            throw new
UnsupportedOperationException("Piece is not vertical");
        }
    }
}
if (!found) {
    throw new IllegalArgumentException("Piece with
character " + hurufPiece + " not found.");
}
return moveDistance;
}

public int movePieceToDownFarthest(char hurufPiece) {
    int moveDistance = 0;
    boolean found = false;
    for (Piece piece : listNonPiece) {
        if (piece.getHurufPiece() == hurufPiece) {
            int oldY = piece.getY();
            found = true;
            if (piece.getOrientasi() ==
Piece.Orientasi.VERTICAL) {
                int y = piece.getY() +
piece.getUkuran();
                while (y < this.baris &&
papan[y][piece.getX()] == '.') {
                    y++;
                }
                piece.setY(y - piece.getUkuran());
                moveDistance = piece.getY() - oldY;
                // Clear old position
                for (int i = 0; i < piece.getUkuran();
i++) {
                    papan[oldY + i][piece.getX()] = '.';
                }
                // Set new position
                for (int i = 0; i < piece.getUkuran();
i++) {
                    papan[piece.getY() +
i][piece.getX()] = piece.getHurufPiece();

```

```

        }
    } else {
        throw new
UnsupportedOperationException("Piece is not vertical");
    }
}

if (!found) {
    throw new IllegalArgumentException("Piece with
character " + hurufPiece + " not found.");
}
return moveDistance;
}

public void movePieceRight(char hurufPiece, int jarak) {
    boolean found = false;
    for (Piece piece : listNonPiece) {
        if (piece.getHurufPiece() == hurufPiece) {
            found = true;
            if (piece.getOrientasi() ==
Piece.Orientasi.HORIZONTAL) {
                if (piece.getX() + jarak < 0) {
                    throw new
IllegalArgumentException("Piece cannot move outside the
board");
                }
                piece.moveX(jarak);
                // Clear old position
                for (int i = 0; i < piece.getUkuran();
i++) {
                    papan[piece.getY()][piece.getX() -
jarak + i] = '.';
                }
                // Set new position
                for (int i = 0; i < piece.getUkuran();
i++) {
                    papan[piece.getY()][piece.getX() +
i] = piece.getHurufPiece();
                }
            } else {
                throw new
UnsupportedOperationException("Piece is not horizontal");
            }
        }
    }
}

```

```

    }
}

if (!found) {
    throw new IllegalArgumentException("Piece with
character " + hurufPiece + " not found.");
}

}

public void movePieceLeft(char hurufPiece, int jarak) {
    boolean found = false;
    for (Piece piece : listNonPiece) {
        if (piece.getHurufPiece() == hurufPiece) {
            found = true;
            if (piece.getOrientasi() ==
Piece.Orientasi.HORIZONTAL) {
                if (piece.getX() + jarak < 0) {
                    throw new
IllegalArgumentException("Piece cannot move outside the
board");
                }
                piece.moveX(-jarak);
                // Clear old position
                for (int i = 0; i < piece.getUkuran();
i++) {
                    papan[piece.getY()][piece.getX() +
jarak + i] = '.';
                }
                // Set new position
                for (int i = 0; i < piece.getUkuran();
i++) {
                    papan[piece.getY()][piece.getX() +
i] = piece.getHurufPiece();
                }
            } else {
                throw new
UnsupportedOperationException("Piece is not horizontal");
            }
        }
    }

    if (!found) {
        throw new IllegalArgumentException("Piece with
character " + hurufPiece + " not found.");
    }
}

```



```

    }

    public void movePieceUp(char hurufPiece, int jarak) {
        boolean found = false;
        for (Piece piece : listNonPiece) {
            if (piece.getHurufPiece() == hurufPiece) {
                found = true;
                if (piece.getOrientasi() ==
Piece.Orientasi.VERTICAL) {
                    if (piece.getY() + jarak < 0) {
                        throw new
IllegalArgumentException("Piece cannot move outside the
board");
                    }
                    piece.moveY(-jarak);
                    // Clear old position
                    for (int i = 0; i < piece.getUkuran();
i++) {
                        papan[piece.getY() + jarak +
i][piece.getX()] = '.';
                    }
                    // Set new position
                    for (int i = 0; i < piece.getUkuran();
i++) {
                        papan[piece.getY() +
i][piece.getX()] = piece.getHurufPiece();
                    }
                } else {
                    throw new
UnsupportedOperationException("Piece is not vertical");
                }
            }
        }
        if (!found) {
            throw new IllegalArgumentException("Piece with
character " + hurufPiece + " not found.");
        }
    }

    public void movePieceDown(char hurufPiece, int jarak) {
        boolean found = false;
        for (Piece piece : listNonPiece) {
            if (piece.getHurufPiece() == hurufPiece) {
                found = true;

```

```

        if (piece.getOrientasi() ==
Piece.Orientasi.VERTICAL) {
            if (piece.getY() + jarak < 0) {
                throw new
IllegalArgumentException("Piece cannot move outside the
board");
            }
            piece.moveY(jarak);
            // Clear old position
            for (int i = 0; i < piece.getUkuran();
i++) {
                papan[piece.getY() - jarak +
i][piece.getX()] = '.';
            }
            // Set new position
            for (int i = 0; i < piece.getUkuran();
i++) {
                papan[piece.getY() +
i][piece.getX()] = piece.getHurufPiece();
            }
        } else {
            throw new
UnsupportedOperationException("Piece is not vertical");
        }
    }
}
if (!found) {
    throw new IllegalArgumentException("Piece with
character " + hurufPiece + " not found.");
}
}
public String serializePapan() {
    StringBuilder sb = new StringBuilder();
    for (int i = 0; i < baris; i++) {
        for (int j = 0; j < kolom; j++) {
            sb.append(papan[i][j]);
        }
    }
    return sb.toString();
}
public int countObstacleInFront() {
    int count = 0;

```

```

        char obs = '.';
        if (primaryPiece.getOrientasi() ==
Piece.Orientasi.HORIZONTAL) {
            if (keluarX == -1) {
                for (int i = primaryPiece.getX() - 1; i >=
0; i--) {
                    if (papan[primaryPiece.getY()][i] !=
obs) {
                        count++;
                    }
                }
            } else {
                for (int i = primaryPiece.getX() +
primaryPiece.getUkuran(); i < this.kolom; i++) {
                    if (papan[primaryPiece.getY()][i] !=
obs) {
                        count++;
                    }
                }
            }
        } else {
            if (keluarY == -1) {
                for (int i = primaryPiece.getY() - 1; i >=
0; i--) {
                    if (papan[i][primaryPiece.getX()] !=
obs) {
                        count++;
                    }
                }
            } else {
                for (int i = primaryPiece.getY() +
primaryPiece.getUkuran(); i < this.baris; i++) {
                    if (papan[i][primaryPiece.getX()] !=
obs) {
                        count++;
                    }
                }
            }
        }
        return count;
    }

    public Map<Character, Piece> getMapPiece() {

```

```

        Map<Character, Piece> pieces = new HashMap<>();
        for (Piece piece : listNonPiece) {
            pieces.put(piece.getHurufPiece(), piece);
        }
        pieces.put('P', primaryPiece);
        return pieces;
    }
}

```

3.3. Piece.java

```

public class Piece {
    private char hurufPiece;
    private int ukuran;
    enum Orientasi {
        HORIZONTAL,
        VERTIKAL
    }
    private Orientasi orientasi;
    private int x; // x dan y dimulai dari ujung kiri atas
    private int y;
    private boolean isPrimary;

    public Piece(char hurufPiece, int ukuran, Orientasi
orientasi, int x, int y) {
        this.hurufPiece = hurufPiece;
        this.ukuran = ukuran;
        this.orientasi = orientasi;
        this.x = x;
        this.y = y;
        this.isPrimary = false;
    }

    public char getHurufPiece() {
        return hurufPiece;
    }

    public int getUkuran() {

```

```

        return ukuran;
    }

    public Orientasi getOrientasi() {
        return orientasi;
    }

    public int getX() {
        return x;
    }

    public int getY() {
        return y;
    }

    public boolean isPrimary() {
        return isPrimary;
    }

    public void setPrimary(boolean primary) {
        isPrimary = primary;
    }

    public void setHurufPiece(char hurufPiece) {
        this.hurufPiece = hurufPiece;
    }

    public void setUkuran(int ukuran) {
        this.ukuran = ukuran;
    }

    public void setOrientasi(Orientasi orientasi) {
        this.orientasi = orientasi;
    }

    public void setX(int x) {
        this.x = x;
    }

    public void setY(int y) {
        this.y = y;
    }

    public void moveX(int deltaX) {
        if (orientasi == Orientasi.HORIZONTAL) {
            this.x += deltaX;
        } else {
            // Throw an exception
            throw new UnsupportedOperationException("Cannot
move X for vertical piece");
        }
    }

    public void moveY(int deltaY) {

```

```

        if (orientasi == Orientasi.VERTICAL) {
            this.y += deltaY;
        } else {
            // Throw an exception
            throw new UnsupportedOperationException("Cannot
move Y for horizontal piece");
        }
    }

    public void printPiece() {
        if (orientasi == Orientasi.HORIZONTAL) {
            for (int i = 0; i < ukuran; i++) {
                System.out.print(hurufPiece);
            }
            System.out.println();
        } else {
            for (int i = 0; i < ukuran; i++) {
                System.out.println(hurufPiece);
            }
        }
    }

    public Piece copy() {
        Piece newPiece = new Piece(this.hurufPiece,
this.ukuran, this.orientasi, this.x, this.y);
        newPiece.setPrimary(this.isPrimary);
        return newPiece;
    }
}

```

3.4. State.java

```

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.util.*;

public class State implements Comparable<State> {
    Map<Character, Piece> pieces; // Map id kendaraan ke

```

```

objek Vehicle
    Papan papan;
    int cost;
    State parent; // untuk melacak jalur solusi
    String move;
    int heuristic; // untuk A* search

    public State(Map<Character, Piece> pieces, Papan papan,
int cost, State parent, String move) {
        this.pieces = pieces;
        this.papan = papan;
        this.cost = cost;
        this.parent = parent;
        this.move = move;
        this.heuristic = computeHeuristic();
    }
    public int getCost() {
        return cost;
    }
    public Papan getPapan() {
        return papan;
    }
    public int getHeuristic() {
        return heuristic;
    }
    public String getMove() {
        return move;
    }
    public State getParent() {
        return parent;
    }
    public void setParent(State parent) {
        this.parent = parent;
    }
    private int computeHeuristic() {
        Piece xCar = pieces.get('P');
        if (xCar == null) return 0;

        int count = 0;

        if (xCar.getOrientasi() ==
Piece.Orientasi.HORIZONTAL) {

```

```

        int row = xCar.getY();

        if (papan.getKeluarX() == -1) {
            // Exit di kiri
            int leftCol = xCar.getX();
            for (int c = leftCol - 1; c >= 0; c--) {
                if (papan.getPapan()[row][c] != '.')
count++;
            }
        } else {
            // Exit di kanan
            int rightEnd = xCar.getX() +
xCar.getUkuran() - 1;
            for (int c = rightEnd + 1; c <
papan.getPapan()[0].length; c++) {
                if (papan.getPapan()[row][c] != '.')
count++;
            }
        }

        } else {
            int col = xCar.getX();

            if (papan.getKeluarY() == -1) {
                // Exit di atas
                int topRow = xCar.getY();
                for (int r = topRow - 1; r >= 0; r--) {
                    if (papan.getPapan()[r][col] != '.')
count++;
                }
            } else {
                // Exit di bawah
                int bottomEnd = xCar.getY() +
xCar.getUkuran() - 1;
                for (int r = bottomEnd + 1; r <
papan.getPapan().length; r++) {
                    if (papan.getPapan()[r][col] != '.')
count++;
                }
            }
        }
    }
}

```



```

        return count;
    }

    // Membandingkan state berdasarkan cost (UCS)
    @Override
    public int compareTo(State other) {
        return Integer.compare(this.cost, other.cost);
    }

    // Cek apakah mobil merah sudah di exit
    public boolean isGoal() {
        Piece redCar = pieces.get('P');
        if (redCar == null) return false;

        if (redCar.getOrientasi() ==
Piece.Orientasi.HORIZONTAL) {
            int leftCol = redCar.getX();
            int rightEndCol = redCar.getX() +
redCar.getUkuran() - 1;
            int row = redCar.getY();

            if (papan.getKeluarX() == -1) {
                // Exit di kiri, cek ujung kiri mobil sudah
di -1
                return papan.getKeluarY() == row && leftCol
== papan.getKeluarX();
            } else {
                // Exit di kanan atau kolom lain
                return papan.getKeluarY() == row &&
rightEndCol == papan.getKeluarX();
            }
        } else {
            int topRow = redCar.getY();
            int bottomEndRow = redCar.getY() +
redCar.getUkuran() - 1;
            int col = redCar.getX();

            if (papan.getKeluarY() == -1) {
                // Exit di atas, cek ujung atas mobil sudah

```

```

    di -1
        return papan.getKeluarX() == col && topRow
== papan.getKeluarY();
    } else {
        // Exit di bawah atau baris lain
        return papan.getKeluarX() == col &&
bottomEndRow == papan.getKeluarY();
    }
}

}

public List<State> getNextStates() {
    List<State> nextStates = new ArrayList<>();

    for (Piece p : pieces.values()) {
        // Untuk dua arah: maju (+1) dan mundur (-1)
        for (int direction : new int[]{1, -1}) {
            int steps = 1;
            while (true) {
                if (!canMove(p, direction, steps)) break;

                State newState = moveVehicle(p, direction *
steps);
                if (newState != null)
nextStates.add(newState);
                steps++;
            }
        }
    }

    return nextStates;
}

private boolean canMove(Piece p, int direction, int
steps) {
    if (p.getOrientasi() == Piece.Orientasi.HORIZONTAL) {
        int row = p.getY();
        if (direction == -1) {
            for (int i = 1; i <= steps; i++) {
                int col = p.getX() - i;
                if (col < 0) {
                    // cek exit kiri

```

```

        if (col == -1) {
            if (i == steps) {
                return papan.getKeluarY() == row
&& papan.getKeluarX() == -1;
            } else return false;
        }
        return false;
    }
    if (papan.getPapan()[row][col] != '.')
return false;
    }
    } else {
        int rightEnd = p.getX() + p.getUkuran() - 1;
        for (int i = 1; i <= steps; i++) {
            int col = rightEnd + i;
            if (col >= papan.getPapan()[0].length) {
                if (col == papan.getPapan()[0].length &&
i == steps) {
                    return papan.getKeluarY() == row &&
papan.getKeluarX() == col;
                } else return false;
            }
            if (col < papan.getPapan()[0].length &&
papan.getPapan()[row][col] != '.') return false;
        }
    }
    } else {
        int col = p.getX();
        if (direction == -1) {
            for (int i = 1; i <= steps; i++) {
                int row = p.getY() - i;
                if (row < 0) {
                    if (row == -1 && i == steps) {
                        return papan.getKeluarX() == col &&
papan.getKeluarY() == -1;
                    } else return false;
                }
                if (papan.getPapan()[row][col] != '.')
return false;
            }
        } else {
            int bottomEnd = p.getY() + p.getUkuran() - 1;

```

```

        for (int i = 1; i <= steps; i++) {
            int row = bottomEnd + i;
            if (row >= papan.getPapan().length) {
                if (row == papan.getPapan().length && i
== steps) {
                    return papan.getKeluarX() == col &&
papan.getKeluarY() == row;
                } else return false;
            }
            if (row < papan.getPapan().length &&
papan.getPapan()[row][col] != '.') return false;
        }
    }
    return true;
}

```

```

    private State moveVehicle(Piece p, int move) {
        // Copy kendaraan dan map kendaraan
        Map<Character, Piece> newPiece = new
HashMap<>();
        for (Map.Entry<Character, Piece> entry :
pieces.entrySet()) {
            newPiece.put(entry.getKey(),
entry.getValue().copy());
        }

        Piece movedPiece =
newPiece.get(p.getHurufPiece());

        // Update posisi kendaraan yang digerakkan
        if (movedPiece.getOrientasi() ==
Piece.Orientasi.HORIZONTAL) {
            int x = movedPiece.getX();
            movedPiece.setX(x + move);
        } else {
            int y = movedPiece.getY();
            movedPiece.setY(y + move);
        }

        // Buat board baru kosong
    }
}

```

```

        char[][] newBoard = new
char[papan.getPapan().length][papan.getPapan()[0].length];
        for (int i = 0; i < newBoard.length; i++) {
            Arrays.fill(newBoard[i], '.');
        }

        // Update posisi kendaraan ke board baru
        for (Piece pie : newPiece.values()) {
            int r = pie.getY();
            int c = pie.getX();
            for (int i = 0; i < pie.getUkuran(); i++) {
                if (pie.getOrientasi() ==
Piece.Orientasi.HORIZONTAL) {
                    int cc = c + i;
                    if (r >= 0 && r < newBoard.length &&
cc >= 0 && cc < newBoard[0].length) {
                        newBoard[r][cc] =
pie.getHurufPiece();
                    }
                } else {
                    int rr = r + i;
                    if (rr >= 0 && rr < newBoard.length
&& c >= 0 && c < newBoard[0].length) {
                        newBoard[rr][c] =
pie.getHurufPiece();
                    }
                }
            }
        }

        // Hitung cost baru (misal cost 1 per gerakan)
        int newCost = this.cost + 1;
        Papan newPapan = papan.copy();
        newPapan.setPapan(newBoard);
        if (movedPiece.getOrientasi() ==
Piece.Orientasi.HORIZONTAL) {
            return new State(newPiece, newPapan,
newCost, this, "Gerak " + p.getHurufPiece() + " ke " + (move
> 0 ? "kanan" : "kiri"));
        }
        return new State(newPiece, newPapan, newCost,

```

```

this, "Gerak " + p.getHurufPiece() + " ke " + (move > 0 ?
    "bawah" : "atas"));
    }

    // Untuk menyimpan state di Set, kita butuh equals dan
    hashCode (berdasarkan posisi kendaraan)
    // @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (!(o instanceof State)) return false;
        State other = (State) o;

        if (pieces.size() != other.pieces.size()) return
false;

        for (char id : pieces.keySet()) {
            Piece v1 = pieces.get(id);
            Piece v2 = other.pieces.get(id);
            if (v2 == null) return false;
            if (v1.getX() != v2.getX() || v1.getY() !=
v2.getY()) return false;
        }
        return true;
    }

    @Override
    public int hashCode() {
        int result = 17;
        for (Piece v : pieces.values()) {
            result = 31 * result + v.getHurufPiece();
            result = 31 * result + v.getY();
            result = 31 * result + v.getX();
        }
        return result;
    }

    @Override
    public String toString() {
        int keluarX = this.getPapan().getKeluarX();
        int keluarY = this.getPapan().getKeluarY();

        char[][] papan = this.getPapan().getPapan();

```

```

        int rows = papan.length;
        int cols = papan[0].length;

        int minRow = Math.min(0, keluarY);
        int maxRow = Math.max(rows - 1, keluarY);
        int minCol = Math.min(0, keluarX);
        int maxCol = Math.max(cols - 1, keluarX);

        StringBuilder sb = new StringBuilder();

        for (int y = minRow; y <= maxRow; y++) {
            for (int x = minCol; x <= maxCol; x++) {
                if (x == keluarX && y == keluarY) {
                    sb.append("K");
                } else {
                    if (y >= 0 && y < rows && x >= 0 && x <
cols) {
                        sb.append(papan[y][x]);
                    } else {
                        if ((x== keluarX && y !=
keluarY)|| (y== keluarY && x != keluarX)) {
                            sb.append(" ");
                        } else {
                            sb.append(".");
                        }
                    }
                }
            }
            sb.append("\n");
        }

        return sb.toString();
    }

    public void saveSolutionToFile(int nodeCount, long
executionTime) {
        BufferedReader reader = new BufferedReader(new
InputStreamReader(System.in));

        System.out.println("Masukkan nama file untuk
menyimpan solusi (tambahkan ekstensi .txt):");

        String filename;

```

```

        try {
            filename = reader.readLine();
            if (filename == null ||
filename.trim().isEmpty()) {
                throw new IllegalArgumentException("Nama
file tidak boleh kosong.");
            }
            if (!filename.endsWith(".txt")) {
                throw new IllegalArgumentException("Nama
file harus diakhiri dengan .txt");
            }
        } catch (IOException e) {
            throw new RuntimeException("Gagal membaca input
nama file: " + e.getMessage(), e);
        }

        List<State> path = new ArrayList<>();
        State current = this;
        while (current != null) {
            path.add(current);
            current = current.getParent();
        }
        Collections.reverse(path);

        try (PrintWriter writer = new PrintWriter(filename))
        {
            for (State state : path) {
                writer.println("Move: " + state.getMove());
                writer.println(state);
            }
            writer.println("Visited nodes: " + nodeCount);
            writer.println("Execution time: " +
executionTime + " ms");
        } catch (IOException e) {
            throw new RuntimeException("Gagal menyimpan
file: " + e.getMessage(), e);
        }
    }

    public void saveNoSolutionToFile(int nodeCount, long
executionTime) {
        BufferedReader reader = new BufferedReader(new
InputStreamReader(System.in));
    }

```



```

        System.out.println("Masukkan nama file untuk
menyimpan hasil (tambahkan ekstensi .txt):");
        String filename;
        try {
            filename = reader.readLine();
            if (filename == null ||
filename.trim().isEmpty()) {
                throw new IllegalArgumentException("Nama
file tidak boleh kosong.");
            }
            if (!filename.endsWith(".txt")) {
                throw new IllegalArgumentException("Nama
file harus diakhiri dengan .txt");
            }
        } catch (IOException e) {
            throw new RuntimeException("Gagal membaca input
nama file: " + e.getMessage(), e);
        }

        try (PrintWriter writer = new PrintWriter(filename))
        {
            writer.println("Tidak ada solusi ditemukan.");
            writer.println("Visited nodes: " + nodeCount);
            writer.println("Execution time: " +
executionTime + " ms");
        } catch (IOException e) {
            throw new RuntimeException("Gagal menyimpan
file: " + e.getMessage(), e);
        }
    }

    public void printSolution() {
        if (parent != null) {
            parent.printSolution();
        }
        System.out.println(move);
        papan.printPapan();
    }

    public List<State> getPath() {
        List<State> path = new ArrayList<>();

```

```

        State current = this;
        while (current != null) {
            path.add(current);
            current = current.parent;
        }
        Collections.reverse(path);
        return path;
    }
}

```

3.5. UCS.java

```

import java.util.*;

public class UCS {

    public static void uniformCostSearch(State initialState)
    {
        long startTime = System.currentTimeMillis();
        int visitCount = 0;

        // Priority queue berdasarkan cost ascending
        PriorityQueue<State> frontier = new
PriorityQueue<>(Comparator.comparingInt(s -> s.cost));
        Map<State, Integer> stateCostMap = new HashMap<>();
        // untuk menyimpan cost terkecil ke suatu state
        Set<State> explored = new HashSet<>();

        frontier.add(initialState);
        stateCostMap.put(initialState, initialState.cost);

        while (!frontier.isEmpty()) {

            State current = frontier.poll();
            visitCount++;

            if (explored.contains(current)) continue;
            explored.add(current);

```

```

        if (current.isGoal()) {
            long endTime = System.currentTimeMillis();
            System.out.println("Waktu eksekusi: " +
(endTime - startTime) + " ms");
            System.out.println("Node dikunjungi: " +
visitCount);
            current.printSolution();
            current.saveSolutionToFile(visitCount,
endTime - startTime);
            return;
        }

        for (State next : current.getNextStates()) {
            int newCost = next.cost;

            if (!explored.contains(next) &&
                (!stateCostMap.containsKey(next) ||
newCost < stateCostMap.get(next))) {

                frontier.add(next);
                stateCostMap.put(next, newCost);
            }
        }

        System.out.println("Tidak ditemukan solusi");
        long endTime = System.currentTimeMillis();
        initialState.saveNoSolutionToFile(visitCount,
endTime - startTime);
    }
}

```

3.6. GBFS.java



3.7. AStarSearch.java

```
import java.util.*;

public class AStarSearch {
    // Comparator untuk A*:  $f(n) = g(n) + h(n)$ 
    private static final Comparator<State> aStarComparator =
(s1, s2) -> {
        int f1 = s1.getCost() + s1.getHeuristic();
        int f2 = s2.getCost() + s2.getHeuristic();
        return Integer.compare(f1, f2);
    };

    public static void solve(State startState) {
        long startTime = System.currentTimeMillis();
        int visitCount = 0;
        PriorityQueue<State> openSet = new
PriorityQueue<>(aStarComparator);
        Map<State, Integer> bestFScore = new HashMap<>();

        openSet.add(startState);
        bestFScore.put(startState, startState.getCost() +
startState.getHeuristic());

        while (!openSet.isEmpty()) {
            State current = openSet.poll();
            visitCount++;

            if (current.isGoal()) {
                long endTime = System.currentTimeMillis();
                System.out.println("Waktu eksekusi: " +
(endTime - startTime) + " ms");
                System.out.println("Node dikunjungi: " +
visitCount);

                current.printSolution();
                current.saveSolutionToFile(visitCount,
endTime - startTime);
                return;
            }

            for (State next : current.getNextStates()) {
```

```

        int g = next.getCost();
        int f = g + next.getHeuristic();

        if (!bestFScore.containsKey(next) || f <
bestFScore.get(next)) {
            bestFScore.put(next, f);
            openSet.add(next);
        }
    }
}

System.out.println("Tidak ada solusi ditemukan.");
long endTime = System.currentTimeMillis();
startState.saveNoSolutionToFile(visitCount,
endTime-startTime);
}
}

```

3.8. BeamSearch.java

```

import java.util.*;

public class BeamSearch {
    private static int beamWidth = 100;

    public static void solve(State initialState) {
        long startTime = System.currentTimeMillis();
        Map<String, Integer> visitedFScore = new
HashMap<>(); // key: board, value: f = cost + heuristic
        int nodeCount = 0;
        List<State> currentBeam = new ArrayList<>();
        currentBeam.add(initialState);

        int iterations = 0;
        int maxIterations = 10000;

        while (!currentBeam.isEmpty() && iterations <
maxIterations) {

```

```

        iterations++;
        List<State> nextBeam = new ArrayList<>();

        for (State current : currentBeam) {
            if (current.isGoal()) {
                long endTime =
System.currentTimeMillis();
                System.out.println("Waktu eksekusi: " +
(endTime - startTime) + " ms");
                System.err.println("Node dikunjungi: " +
nodeCount);

                current.printSolution();
                current.saveSolutionToFile(nodeCount,
endTime - startTime);
                return;
            }

            String currentKey =
getBoardKey(current.getPapan().getPapan());
            int currentF = current.getCost() +
current.getHeuristic();
            visitedFScore.put(currentKey, currentF);

            List<State> successors =
current.getNextStates();
            for (State successor : successors) {
                String succKey =
getBoardKey(successor.getPapan().getPapan());
                int succF = successor.getCost() +
successor.getHeuristic();

                if (!visitedFScore.containsKey(succKey)
|| succF < visitedFScore.get(succKey)) {
                    visitedFScore.put(succKey, succF);
                    nextBeam.add(successor);
                    nodeCount++;
                }
            }
        }

        if (nextBeam.isEmpty()) break;

```

```

        // Sort by f = cost + heuristic
        nextBeam.sort(Comparator.comparingInt(s ->
s.getCost() + s.getHeuristic()));

        currentBeam = nextBeam.size() <= beamWidth ?
            nextBeam :
            nextBeam.subList(0, beamWidth);
    }

    System.out.println("Tidak ada solusi ditemukan dalam
" + iterations + " iterasi.");
    long endTime = System.currentTimeMillis();
    initialState.saveNoSolutionToFile(nodeCount, endTime
- startTime);
}

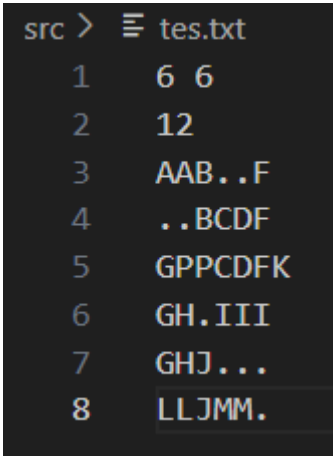
private static String getBoardKey(char[][] board) {
    StringBuilder key = new StringBuilder();
    for (char[] row : board) {
        key.append(row);
    }
    return key.toString();
}
}

```

Bab 4

Tangkapan Layar

4.1. Test Case 1

Input

Output UCS

Move: State Awal

```
A A B . . F
. . B C D F
G P P C D F K
G H . I I I
G H J . . .
L L J M M .
```

Move: Gerak C ke atas

```
A A B C . F
. . B C D F
G P P . D F K
G H . I I I
G H J . . .
L L J M M .
```

Move: Gerak I ke kiri

```
A A B C . F
. . B C D F
G P P . D F K
G H I I I .
G H J . . .
L L J M M .
```

Move: Gerak F ke bawah

```
A A B C . .
. . B C D .
G P P . D . K
G H I I I F
G H J . . F
L L J M M F
```

Move: Gerak D ke atas

```
A A B C D .  
. . B C D .  
G P P . . . K  
G H I I I F  
G H J . . F  
L L J M M F
```

Move: Gerak P ke kanan

```
A A B C D .  
. . B C D .  
G . . . . P K  
G H I I I F  
G H J . . F  
L L J M M F
```

Visited nodes: 210

Execution time: 41 ms

Output GBFS (Heuristic ketiga)

```
1 Solusi ditemukan:
2 Papan awal:
3 A A B . . F
4 . . B C D F
5 G P P C D F K
6 G H . I I I
7 G H J . . .
8 L L J M M .
9
10
11 Move: Move D UP, 1 blocks
12 A A B . D F
13 . . B C D F
14 G P P C . F K
15 G H . I I I
16 G H J . . .
17 L L J M M .
18
19
20 Move: Move C UP, 1 blocks
21 A A B C D F
22 . . B C D F
23 G P P . . F K
24 G H . I I I
25 G H J . . .
26 L L J M M .
27
28
29 Move: Move P RIGHT, 1 blocks
30 A A B C D F
31 . . B C D F
32 G . P P . F K
33 G H . I I I
34 G H J . . .
35 L L J M M .
36
```

```

38 Move: Move H UP, 1 blocks
39 A A B C D F
40 . . B C D F
41 G H P P . F K
42 G H . I I I
43 G . J . . .
44 L L J M M .
45
46
47 Move: Move I LEFT, 1 blocks
48 A A B C D F
49 . . B C D F
50 G H P P . F K
51 G H I I I .
52 G . J . . .
53 L L J M M .
54
55
56 Move: Move F DOWN, 3 blocks
57 A A B C D .
58 . . B C D .
59 G H P P . . K
60 G H I I I F
61 G . J . . F
62 L L J M M F
63
64
65 Visited nodes: 59
66 Execution time: 69 ms

```

Output A* (heuristik 1)

Move: State Awal

```
A A B . . F
. . B C D F
G P P C D F K
G H . I I I
G H J . . .
L L J M M .
```

Move: Gerak C ke atas

```
A A B C . F
. . B C D F
G P P . D F K
G H . I I I
G H J . . .
L L J M M .
```

Move: Gerak D ke atas

```
A A B C D F
. . B C D F
G P P . . F K
G H . I I I
G H J . . .
L L J M M .
```

Move: Gerak I ke kiri

```
A A B C D F
. . B C D F
G P P . . F K
G H I I I .
G H J . . .
L L J M M .
```

Move: Gerak F ke bawah

A A B C D .

. . B C D .

G P P . . . K

G H I I I F

G H J . . F

L L J M M F

Move: Gerak P ke kanan

A A B C D .

. . B C D .

G P K

G H I I I F

G H J . . F

L L J M M F

Visited nodes: 79

Execution time: 13 ms

Output Beam Search (heuristik 1)

Move: State Awal

```
A A B . . F
. . B C D F
G P P C D F K
G H . I I I
G H J . . .
L L J M M .
```

Move: Gerak C ke atas

```
A A B C . F
. . B C D F
G P P . D F K
G H . I I I
G H J . . .
L L J M M .
```

Move: Gerak D ke atas

```
A A B C D F
. . B C D F
G P P . . F K
G H . I I I
G H J . . .
L L J M M .
```

Move: Gerak I ke kiri

```
A A B C D F
. . B C D F
G P P . . F K
G H I I I .
G H J . . .
L L J M M .
```

Move: Gerak F ke bawah

A A B C D .

. . B C D .

G P P . . . K

G H I I I F

G H J . . F

L L J M M F

Move: Gerak P ke kanan

A A B C D .

. . B C D .

G P K

G H I I I F

G H J . . F

L L J M M F

Visited nodes: 289

Execution time: 22 ms

4.2. Test Case 2

Input

6 6

12

AAB..F

..BCDF

KGPPCDF

GH.III

GHJ...

LLJMM.

Output UCS

Move: State Awal

	A	A	B	.	.	F
	.	.	B	C	D	F
K	G	P	P	C	D	F
	G	H	.	I	I	I
	G	H	J	.	.	.
	L	L	J	M	M	.

Move: Gerak J ke atas

	A	A	B	.	.	F
	.	.	B	C	D	F
K	G	P	P	C	D	F
	G	H	J	I	I	I
	G	H	J	.	.	.
	L	L	.	M	M	.

Move: Gerak L ke kanan

	A	A	B	.	.	F
	.	.	B	C	D	F
K	G	P	P	C	D	F
	G	H	J	I	I	I
	G	H	J	.	.	.
	.	L	L	M	M	.

Move: Gerak G ke bawah

	A	A	B	.	.	F
	.	.	B	C	D	F
K	.	P	P	C	D	F
	G	H	J	I	I	I
	G	H	J	.	.	.
	G	L	L	M	M	.

Move: Gerak P ke kiri

```
A A B . . F
. . B C D F
K P . . C D F
G H J I I I
G H J . . .
G L L M M .
```

Visited nodes: 127

Execution time: 37 ms

Output GBFS (Heuristic kedua)

1 Solusi ditemukan:

2 Papan awal:

3 A A B . . F
4 . . B C D F
5 K G P P C D F
6 G H . I I I
7 G H J . . .
8 L L J M M .

9

10

11 Move: Move C UP, 1 blocks

12 A A B C . F
13 . . B C D F
14 K G P P . D F
15 G H . I I I
16 G H J . . .
17 L L J M M .

18

19

20 Move: Move M RIGHT, 1 blocks

21 A A B C . F
22 . . B C D F
23 K G P P . D F
24 G H . I I I
25 G H J . . .
26 L L J . M M

27

28

29 Move: Move I LEFT, 1 blocks

30 A A B C . F
31 . . B C D F
32 K G P P . D F
33 G H I I I .
34 G H J . . .
35 L L J . M M

36

37

```

74  ✓ Move: Move J UP, 1 blocks
75      A A B C D F
76      . . B C D F
77  ✓ K G . . P P F
78      G H J I I I
79      G H J . . .
80      L L . M M .
81
82
83  ✓ Move: Move M RIGHT, 1 blocks
84      A A B C D F
85      . . B C D F
86  ✓ K G . . P P F
87      G H J I I I
88      G H J . . .
89      L L . . M M
90
91
92  ✓ Move: Move H UP, 1 blocks
93      A A B C D F
94      . . B C D F
95  ✓ K G H . P P F
96      G H J I I I
97      G . J . . .
98      L L . . M M
99
100
101  ✓ Move: Move M LEFT, 2 blocks
102      A A B C D F
103      . . B C D F
104  ✓ K G H . P P F
105      G H J I I I
106      G . J . . .
107      L L M M . .
108

```

```

110  ✓ Move: Move J UP, 1 blocks
111      A A B C D F
112      . . B C D F
113  ✓ K G H J P P F
114      G H J I I I
115      G . . . . .
116      L L M M . .
117
118
119  ✓ Move: Move M RIGHT, 1 blocks
120      A A B C D F
121      . . B C D F
122  ✓ K G H J P P F
123      G H J I I I
124      G . . . . .
125      L L . M M .
126
127
128  ✓ Move: Move L RIGHT, 1 blocks
129      A A B C D F
130      . . B C D F
131  ✓ K G H J P P F
132      G H J I I I
133      G . . . . .
134      . L L M M .
135
136
137  ✓ Move: Move G UP, 1 blocks
138      A A B C D F
139      G . B C D F
140  ✓ K G H J P P F
141      G H J I I I
142      . . . . . .
143      . L L M M .
144

```

```

146 Move: Move M RIGHT, 1 blocks
147   A A B C D F
148   G . B C D F
149 K G H J P P F
150   G H J I I I
151   . . . . .
152   . L L . M M
153
154
155 Move: Move L RIGHT, 1 blocks
156   A A B C D F
157   G . B C D F
158 K G H J P P F
159   G H J I I I
160   . . . . .
161   . . L L M M
162
163
164 Move: Move J DOWN, 1 blocks
165   A A B C D F
166   G . B C D F
167 K G H . P P F
168   G H J I I I
169   . . J . . .
170   . . L L M M
171
172
173 Move: Move H DOWN, 2 blocks
174   A A B C D F
175   G . B C D F
176 K G . . P P F
177   G . J I I I
178   . H J . . .
179   . H L L M M
180

```

```
173 Move: Move H DOWN, 2 blocks
174 | A A B C D F
175 | G . B C D F
176 K G . . P P F
177 | G . J I I I
178 | . H J . . .
179 | . H L L M M
180
181
182 Move: Move G DOWN, 2 blocks
183 | A A B C D F
184 | . . B C D F
185 K . . . P P F
186 | G . J I I I
187 | G H J . . .
188 | G H L L M M
189
190
191 Visited nodes: 777
192 Execution time: 139 ms
```

Output A* (heuristik 1)

Move: State Awal

	A	A	B	.	.	F
	.	.	B	C	D	F
K	G	P	P	C	D	F
	G	H	.	I	I	I
	G	H	J	.	.	.
	L	L	J	M	M	.

Move: Gerak J ke atas

	A	A	B	.	.	F
	.	.	B	C	D	F
K	G	P	P	C	D	F
	G	H	J	I	I	I
	G	H	J	.	.	.
	L	L	.	M	M	.

Move: Gerak L ke kanan

	A	A	B	.	.	F
	.	.	B	C	D	F
K	G	P	P	C	D	F
	G	H	J	I	I	I
	G	H	J	.	.	.
	.	L	L	M	M	.

Move: Gerak G ke bawah

	A	A	B	.	.	F
	.	.	B	C	D	F
K	.	P	P	C	D	F
	G	H	J	I	I	I
	G	H	J	.	.	.
	G	L	L	M	M	.

Move: Gerak P ke kiri

	A	A	B	.	.	F
	.	.	B	C	D	F
K	P	.	.	C	D	F
	G	H	J	I	I	I
	G	H	J	.	.	.
	G	L	L	M	M	.

Visited nodes: 64

Execution time: 2 ms

Output Beam Search (heuristik 1)

Move: State Awal

	A	A	B	.	.	F
	.	.	B	C	D	F
K	G	P	P	C	D	F
	G	H	.	I	I	I
	G	H	J	.	.	.
	L	L	J	M	M	.

Move: Gerak J ke atas

	A	A	B	.	.	F
	.	.	B	C	D	F
K	G	P	P	C	D	F
	G	H	J	I	I	I
	G	H	J	.	.	.
	L	L	.	M	M	.

Move: Gerak L ke kanan

	A	A	B	.	.	F
	.	.	B	C	D	F
K	G	P	P	C	D	F
	G	H	J	I	I	I
	G	H	J	.	.	.
	.	L	L	M	M	.

Move: Gerak G ke bawah

	A	A	B	.	.	F
	.	.	B	C	D	F
K	.	P	P	C	D	F
	G	H	J	I	I	I
	G	H	J	.	.	.
	G	L	L	M	M	.

```
Move: Gerak P ke kiri
| A A B . . F
| . . B C D F
K P . . C D F
| G H J I I I
| G H J . . .
| G L L M M .

Visited nodes: 152
Execution time: 7 ms
```

4.3. Test Case 3

Input	
<pre>src > ≡ tes3.txt 1 6 6 2 12 3 K 4 A A B B . F 5 . . . C D F 6 G . P C D F 7 G H P I I I 8 G H J . . . 9 L L J M M .</pre>	
Output UCS	

Move: State Awal

```
| K
AABB.F
...CDF
G.PCDF
GHPIII
GHJ...
LLJMM.
```

Move: Gerak B ke kanan

```
| K
AA.BBF
...CDF
G.PCDF
GHPIII
GHJ...
LLJMM.
```

Move: Gerak P ke atas

```
| K
AAPBBF
...CDF
G..CDF
GH.III
GHJ...
LLJMM.
```

Visited nodes: 18

Execution time: 13 ms

Output GBFS (Heuristic pertama)

```
1 Solusi ditemukan:
2 Papan awal:
3 | K
4 A A B B . F
5 . . . C D F
6 G . P C D F
7 G H P I I I
8 G H J . . .
9 L L J M M .
10
11
12 Move: Move B RIGHT, 1 blocks
13 | K
14 A A . B B F
15 . . . C D F
16 G . P C D F
17 G H P I I I
18 G H J . . .
19 L L J M M .
20
21
22 Visited nodes: 2
23 Execution time: 4 ms
```

Output A* (heuristik 1)

Move: State Awal

```
| K
AABB.F
...CDF
G.PCDF
GHPIII
GHJ...
LLJMM.
```

Move: Gerak B ke kanan

```
| K
AA.BBF
...CDF
G.PCDF
GHPIII
GHJ...
LLJMM.
```

Move: Gerak P ke atas

```
| K
AAPBBF
...CDF
G..CDF
GH.III
GHJ...
LLJMM.
```

Visited nodes: 8

Execution time: 7 ms

Output Beam Search (heuristik 1)

Move: State Awal

| K

AABB.F

...CDF

G.PCDF

GHPIII

GHJ...

LLJMM.

Move: Gerak B ke kanan

| K

AA.BBF

...CDF

G.PCDF

GHPIII

GHJ...

LLJMM.

Move: Gerak P ke atas

| K

AAPBBF

...CDF

G..CDF

GH.III

GHJ...

LLJMM.

Visited nodes: 68

Execution time: 12 ms

4.4. Test Case 4

Input


```
src > ≡ tes4.txt
1    6 6
2    12
3    AAB.B.F
4    ...CDF
5    G.PCDF
6    GHPIII
7    GH....
8    LLJJMM
9    | K
```

Output UCS

```
Tidak ada solusi ditemukan.
Visited nodes: 588
Execution time: 58 ms
```

Output GBFS (Heuristic ketiga)

```
1  Tidak ada solusi ditemukan.
2  Visited nodes: 1935
3  Execution time: 331 ms
4
```

Output A* (heuristik 1)

```
Tidak ada solusi ditemukan.
Visited nodes: 588
Execution time: 37 ms
```

Output Beam Search (heuristik 1)

```
Tidak ada solusi ditemukan.
Visited nodes: 7718
Execution time: 86 ms
```


Bab 5

Analisis Percobaan

Dalam percobaan ini, empat algoritma pathfinding yaitu Uniform Cost Search (UCS), Greedy Best-First Search (GBFS), A*, dan Beam Search digunakan untuk menyelesaikan puzzle *Rush Hour*. Setiap algoritma memiliki pendekatan dan karakteristik yang berbeda dalam pencarian solusi, sehingga mempengaruhi waktu eksekusi, jumlah node yang dikunjungi, dan keoptimalan solusi.

Uniform Cost Search menjamin solusi optimal karena selalu memilih node dengan total biaya terkecil dari simpul awal hingga simpul saat ini. Namun, karena tidak menggunakan informasi heuristik untuk memperkirakan jarak ke tujuan, UCS sering kali mengeksplorasi banyak node yang tidak relevan. Hal ini menyebabkan kompleksitas waktu dan ruangnya menjadi tinggi, terutama pada konfigurasi puzzle yang kompleks. Dalam beberapa percobaan, UCS mengunjungi jumlah node yang jauh lebih banyak dibandingkan algoritma lain untuk mencapai solusi yang sama. Kompleksitas waktunya dalam kasus terburuk adalah $O(b^d)$ di mana b adalah branching factor dan d adalah kedalaman solusi.

Greedy Best-First Search hanya menggunakan fungsi heuristik ($h(n)$) untuk memandu pencarian ke arah tujuan secepat mungkin. Meskipun sangat cepat dalam menemukan solusi pada kasus tertentu, GBFS tidak mempertimbangkan biaya langkah sebelumnya, sehingga tidak menjamin solusi optimal. Dalam eksperimen, GBFS sering menemukan solusi lebih cepat daripada UCS, namun jalurnya bisa lebih panjang atau tidak efisien karena terjebak dalam local optimum. Kompleksitas waktunya dalam kasus terburuk adalah $O(b^d)$ di mana b adalah branching factor dan d adalah kedalaman solusi, tetapi dalam praktiknya lebih cepat dari UCS karena tiap simpul lebih mengarah ke tujuan.

A* merupakan algoritma yang menggabungkan karakteristik UCS dan GBFS dengan menggunakan fungsi evaluasi $f(n) = g(n) + h(n)$. A* memberikan keseimbangan antara eksplorasi dan eksploitasi, sehingga tidak hanya cepat, tetapi juga menjamin solusi optimal jika heuristik yang digunakan admissible. Dalam percobaan, A* menunjukkan performa terbaik secara keseluruhan dengan jumlah node yang lebih sedikit dari UCS dan solusi yang lebih baik dari GBFS. Kompleksitas waktunya dalam kasus terburuk adalah $O(b^d)$ di mana b adalah branching factor dan d adalah kedalaman solusi, tetapi dengan heuristik yang baik akan jauh lebih efisien.

Beam Search merupakan pendekatan heuristik yang membatasi jumlah node terbaik (beam width) pada setiap level pencarian. Dengan hanya mempertahankan k node paling menjanjikan, Beam Search jauh lebih hemat memori dan waktu, namun tidak menjamin solusi ditemukan atau optimal. Dalam percobaan, Beam Search sangat efisien dalam kasus sederhana, tetapi ketika tidak ada solusi, simpul yang dikunjungi sangat banyak. Kompleksitas waktunya adalah $O(kd)$, dengan k adalah beam width dan d adalah kedalaman pencarian.

Bab 6

Bonus

6.1. Algoritma Pathfinding Tambahan

Algoritma tambahan yang diimplementasikan adalah algoritma Beam Search. Beam Search adalah algoritma yang membatasi jumlah state atau kandidat yang dieksplorasi pada setiap langkahnya. Algoritma ini bekerja dengan mempertahankan hanya sejumlah tetap state terbaik, yang disebut dengan *beam width*, berdasarkan biaya aktual dari awal hingga state tersebut dan estimasi heuristik biaya dari state ke tujuan. Dengan cara ini, Beam Search menjaga efisiensi dan menghindari eksplorasi seluruh ruang pencarian yang sangat besar, walaupun hal ini dapat mengorbankan jaminan menemukan solusi optimal. Algoritma ini sangat berguna pada masalah dengan ruang pencarian yang besar dan ketika waktu atau sumber daya komputasi terbatas.

6.2. Heuristik Tambahan

Dua alternatif heuristik ditambahkan ke program agar terdapat tiga opsi heuristik yang dapat dipilih oleh pengguna. Heuristik tambahan yang pertama adalah menghitung derajat kebebasan pergerakan pieces yang ada. Semakin jauh suatu piece dapat bergerak, maka semakin rendah jarak heuristiknya yang berarti semakin tinggi prioritasnya dalam iterasi simpul pencarian. Heuristik yang kedua adalah berdasarkan tingkat terhalangnya sebuah *piece*. Jika sebuah piece bisa bergerak bebas dan tidak ada yang menghalangi maka jaraknya nol. Jika sebuah piece dihalangi oleh sebuah *piece* maka nilai jaraknya satu, jika *piece* tersebut juga dihalangi oleh sebuah *piece* lain atau tembok maka *piece* di awal tadi nilai jaraknya menjadi dua. Begitu seterusnya secara rekursif.

Lampiran

Pranala Repository: [Fajar2k5/Tucil3_13523017_13523027](#)

Poin	Ya	Tidak
1. Program berhasil dikompilasi tanpa kesalahan	V	
2. Program berhasil dijalankan	V	
3. Solusi yang diberikan program benar dan mematuhi aturan permainan	V	
4. Program dapat membaca masukan berkas .txt dan menyimpan solusi berupa print board tahap per tahap dalam berkas .txt	V	
5. [Bonus] Implementasi algoritma pathfinding alternatif	V	
6. [Bonus] Implementasi 2 atau lebih heuristik alternatif	V	
7. [Bonus] Program memiliki GUI		V
8. Program dan laporan dibuat (kelompok) sendiri	V	