IF2211 Strategi Algoritma

Penyelesaian Puzzle Rush Hour Menggunakan Algoritma Pathfinding

Laporan Tugas Kecil

Disusun untuk memenuhi tugas besar mata kuliah IF2211 Strategi Algoritma pada Semester II Tahun Akademik 2024/2025

Oleh

Dita Maheswari 13523125

Amira Izani 13523143

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

BANDUNG

2025

DAFTAR ISI

BAB 1 Analisis Algoritma Pathfinding dalam Pencarian Solusi Permainan Rush Hour

A. Greedy Best First Search (GBFS)

Greedy Best First Search adalah algoritma *pathfinding* yang digunakan untuk menemukan solusi puzzle Rush Hour secara efisien dengan mengandalkan fungsi heuristik sebagai dasar pengambilan keputusan. Dalam konteks Rush Hour, setiap node dalam pencarian merepresentasikan satu konfigurasi papan, yaitu posisi semua mobil pada grid. Tujuan dari pencarian adalah mencapai konfigurasi di mana mobil merah berhasil keluar dari papan melalui celah di sisi kanan.

Algoritma GBFS menentukan langkah berikutnya berdasarkan nilai heuristik $h(n)$, yang mengukur seberapa dekat konfigurasi saat ini ke kondisi solusi. Salah satu heuristik yang umum digunakan dalam Rush Hour adalah jumlah kendaraan yang menghalangi jalur mobil merah ke pintu keluar, atau lebih kompleks lagi, jumlah langkah minimum yang dibutuhkan untuk menggeser kendaraan penghalang tersebut. GBFS akan selalu memilih node yang memiliki nilai heuristik paling kecil (paling menjanjikan), dan mengabaikan total biaya langkah dari posisi awal ke node tersebut $g(n)$.

Berbeda dengan algoritma seperti A*, GBFS menggunakan fungsi evaluasi:

$$f(n) = h(n)$$

tanpa mempertimbangkan akumulasi biaya sebelumnya. Oleh karena itu, GBFS dapat dengan cepat menemukan solusi awal jika heuristiknya cukup akurat dalam mengarahkan pencarian ke arah tujuan. Namun, karena tidak mempertimbangkan efisiensi jalur (jumlah langkah), GBFS tidak menjamin solusi optimal dan bisa saja memilih jalur yang tampak menjanjikan secara heuristik tetapi lebih panjang secara total langkah.

Struktur utama yang digunakan adalah *priority queue*, di mana setiap konfigurasi dimasukkan berdasarkan nilai heuristiknya. Untuk mencegah eksplorasi konfigurasi yang sama berulang kali, digunakan juga visited set untuk menyimpan state yang sudah pernah diproses.

Secara umum, penggunaan GBFS pada puzzle Rush Hour sangat cocok jika tujuannya adalah mendapatkan solusi secara cepat, misalnya untuk eksplorasi awal atau dalam sistem yang mementingkan kecepatan respons. Namun, jika yang dibutuhkan adalah solusi paling efisien (minimum langkah), maka algoritma seperti Uniform Cost Search (UCS) atau A* akan lebih sesuai. Pemilihan heuristik yang baik sangat penting dalam menentukan efektivitas GBFS dalam kasus ini.

B. Uniform Cost Search (UCS)

UCS adalah algoritma *pathfinding* yang mencari jalur dengan biaya paling rendah dari suatu titik awal ke tujuan. UCS merupakan varian dari algoritma Breadth First Search (BFS), tetapi dilengkapi dengan pertimbangan biaya langkah. Dalam UCS, node dikembangkan berdasarkan total cost $g(n)$ dari awal hingga node tersebut, bukan hanya berdasarkan kedalaman atau urutan penciptaannya. Dalam konteks puzzle Rush Hour, UCS bekerja dengan memperlakukan setiap perpindahan kendaraan (baik maju atau mundur satu langkah) sebagai satu unit biaya. Dengan asumsi bahwa semua gerakan memiliki biaya yang sama(misalnya, 1 untuk satu langkah gerakan), UCS akan memilih jalur yang menghasilkan jumlah total langkah paling sedikit hingga primary piece keluar dari papan.

Struktur utama UCS adalah *priority queue* yang diurutkan berdasarkan $f(n) = g(n)$ dan node dengan nilai $g(n)$ terkecil akan dieksekusi pertama, dengan $f(n) = g(n)$ adalah fungsi evaluasi UCS dan $g(n)$ adalah total biaya dari state awal ke node tujuan . Setiap kali node di-*expand*, UCS memilih node dengan biaya kumulatif terkecil. Proses ini berlanjut hingga ditemukan node tujuan (goal state), yaitu kondisi dimana primary piece mencapai pintu keluar.

Apakah UCS sama dengan BFS pada penyelesaian puzzle Rush Hour? Jawabannya adalah tergantung pada definisi biaya gerakan. Jika semua gerakan memiliki biaya yang sama (uniform cost), maka UCS akan menghasilkan urutan jelajah node yang sama seperti BFS karena pemilihan node berdasarkan jumlah langkah yang paling sedikit (kedalaman). Kemudian, path yang dihasilkan UCS akan sama dengan BFS, yaitu solusi dengan jumlah langkah minimum dari root ke goal. Hal ini menyebabkan algoritma UCS dan BFS pada penyelesaian puzzle Rush Hour adalah identik secara fungsional. Jika biaya tiap gerakan berbeda (misal bergantung pada jenis piece), maka UCS akan berbeda dari BFS karena akan mempertimabngkan total cost. UCS juga bisa menemukan jalur optimal dalam hal biaya, meskipun itu tidak selalu jalur dengan langkah paling sedikit.

C. A*

Algoritma A* adalah algoritma *pathfinding* yang mengkombinasikan dua komponen utama dalam proses evaluasi, yaitu biaya sebenarnya dari titik awal ke suatu state ($g(n)$), dan estimasi biaya dari state tersebut ke tujuan yang disebut $h(n)$. Nilai evaluasi total didefinisikan sebagai $f(n) = g(n) + h(n)$, dimana $f(n)$ merepresentasikan estimasi total biaya dari awal hingga mencapai tujuan melalui node tersebut. Berbeda dengan algoritma GBFS yang hanya mengandalkan heuristik $h(n)$, A* mempertimbangkan baik riwayat perjalanan maupun estimasi sisa perjalanan, sehingga menjadikannya lebih akurat dan optimal dalam pencarian solusi. A* bersifat lengkap dan optimal asalkan fungsi heuristik yang digunakan adalah *admissible*.

Heuristik yang digunakan dalam A* untuk puzzle Rush Hour bersifat *admissible*, sesuai dengan definisi pada salindia kuliah, yaitu sebuah heuristik disebut *admissible* apabila untuk semua node n, nilai heuristiknya tidak pernah melebihi biaya sebenarnya paling murah dari n ke tujuan ($h(n) \leq h^*(n)$). Dalam hal ini, heuristik berupa jumlah kendaraan penghalang tidak pernah melebih-lebihkan jumlah langkah yang sebenarnya dibutuhkan karena untuk mencapai tujuan, semua piece memang perlu digeser dan bahkan bisa membutuhkan lebih banyak langkah. Karena heuristik ini tidak melampaui estimasi minimum yang diperlukan, maka dapat disimpulkan bahwa heuristik tersebut *admissible*. Hal ini memastikan bahwa A* akan menemukan solusi optimal untuk puzzle Rush Hour, selama ruang pencariannya ditelusuri sepenuhnya.

BAB 2 Penjelasan Algoritma Pathfinding dalam Pencarian Solusi Permainan Rush Hour

A. Heuristik

Kami menerapkan 2 heuristik untuk algoritma Greedy Best First Search dan A*. Penjelasan dua heuristik sebagai berikut:

Heuristik 1 : Jarak ke Pintu Keluar (distanceToExit)

Tujuan dari heuristik ini adalah untuk menghitung jarak langsung dari bagian terluar PrimaryPiece ke posisi PintuKeluar, dalam arah orientasinya.
Alur dari heuristik ini adalah sebagai berikut:

1. Ambil posisi ujung (endPosition) dari PrimaryPiece.
2. Bandingkan baris atau kolom dari ujung tersebut dengan posisi PintuKeluar, tergantung pada orientasi PrimaryPiece:
    a. Jika orientasinya horizontal, maka yang dibandingkan adalah kolom.
    b. Jika orientasinya vertikal, maka yang dibandingkan adalah baris.
3. Jika ujung PrimaryPiece sejajar dengan pintu keluar (baris atau kolom sama), maka jarak dihitung sebagai:
$$Math.abs(exit - endPosition)$$
4. Jika tidak sejajar (misal, PrimaryPiece horizontal tapi tidak di baris yang sama dengan pintu), maka kembalikan Integer.MAX_VALUE karena tidak mungkin mencapai pintu dalam kondisi ini.

Heuristik 2 : Jumlah Piece Penghalang (countBlockingPieces)

Tujuan dari heuristik ini adalah untuk menghitung berapa banyak Piece lain yang menghalangi jalur langsung ke PrimaryPiece ke PintuKeluar.
Alur dari heuristik ini adalah sebagai berikut:

1. Ambil orientasi dan posisi ujung dari PrimaryPiece.
2. Lihat arah menuju PintuKeluar, kemudian iterasi seluruh cell di jalur tersebut:
    a. Jika horizontal: telusuri kolom-kolom di baris yang sama.
    b. Jika vertikal: telusuri baris-baris di kolom yang sama.
3. Untuk setiap cell yang dilalui:
    a. Jika ada piece lain di sana (papan.getPiece(row, col) != null), maka counter ditambah.

B. Greedy-Best First Search

Berikut adalah alur algoritma GBFS dari program kami:

1. Mulai dari state awal (diberikan sebagai heuristic di konstruktor).
2. Selama openSet tidak kosong:

    a. Ambil state dengan nilai heuristik terkecil dari openSet.
    b. Jika itu adalah tujuan (goal), kembalikan langkah-langkahnya.
    c. Tandai state ini sebagai dikunjungi (closedSet).
    d. Hasilkan semua successor (state tetangga).
    e. Jika successor belum dikunjungi dan belum ada di openSet, tambahkan ke openSet
3. Jika openSet habis tanpa menemukan solusi, kembalikan list kosong.

C. Uniform Cost Search

Berikut adalah alur algoritma UCS dari program kami:

1. Mulai dari state awal, yang diberikan melalui konstruktor sebagai objek heuristic.
2. Selama openSet tidak kosong:
    a. Ambil state dengan biaya terkecil (g(n)) dari openSet.
    b. Jika state tersebut adalah tujuan (isGoal() bernilai true), kembalikan daftar langkah (getMoveHistory()) menuju goal.
    c. Tambahkan state tersebut ke closedSet sebagai sudah dikunjungi.
    d. Hasilkan semua state tetangga dengan memanggil getSuccessors().
    e. Untuk setiap successor:
        - Jika belum ada di closedSet dan belum ada di openSet, tambahkan ke openSet.
        - Jika openSet kosong tanpa menemukan solusi, kembalikan list kosong (new ArrayList<>()).

D. A*

Berikut adalah alur algoritma A* dari program kami:

1. Mulai dari state awal, yaitu objek heuristic yang diberikan melalui konstruktor.
2. Selama openSet (antrian prioritas) tidak kosong, lakukan langkah berikut:
    a. Ambil state current dengan nilai f(n) (yaitu g(n) + h(n)) terkecil dari openSet.
    b. Jika current adalah state tujuan (current.isGoal()), maka:
        - Cetak jumlah iterasi (explored)
        - Kembalikan daftar langkah (current.getMoveHistory()) yang mengarah ke solusi.
    c. Tandai current sebagai sudah dikunjungi dengan menambahkannya ke closedSet.
    d. Hasilkan semua successor dari current (current.getSuccessors()).
    e. Untuk setiap successor:
        - Jika successor sudah dikunjungi (closedSet), abaikan.
        - Jika belum ada di openSet, tambahkan ke openSet.

- Jika openSet habis dan tidak ada solusi ditemukan, cetak pesan dan kembalikan list kosong.

BAB 3 Source Code

Papan.java

```java
public class Papan {
    private int width;
    private int height;
    private Piece[][] pieces;

    public Papan(int width, int height) {
        this.width = width;
        this.height = height;
        this.pieces = new Piece[height][width]; // row x col
    }

    public int getWidth() {
        return width;
    }

    public int getHeight() {
        return height;
    }

    public Piece[][] getPieces() {
        return pieces;
    }

    public Piece getPiece(int row, int col) {
        if (row < 0 || row >= height || col < 0 || col >= width) {
            throw new IllegalArgumentException("Koordinat di luar batas papan.");
        }
        return pieces[row][col];
    }

    public void addPiece(Piece piece) {
        // Validasi: Pastikan posisi piece valid dan belum ditempati
        for (Position pos : piece.getPositions()) {
            int row = pos.getRow();
            int col = pos.getCol();
            if (row < 0 || row >= height || col < 0 || col >= width) {
                throw new IllegalArgumentException("Piece keluar dari batas papan.");
            }
            if (pieces[row][col] != null) {
                throw new IllegalArgumentException("Cell pada papan sudah terisi.");
            }
        }

        // Tempatkan piece ke papan sesuai posisi-posisinya
```

```
        for (Position pos : piece.getPositions()) {
            int row = pos.getRow();
            int col = pos.getCol();
            pieces[row][col] = piece;
        }
    }

    public void setPiece(int row, int col, Piece piece) {
        pieces[row][col] = piece;
    }

    public void removePiece(Piece piece) {
        // Hapus piece dari papan
        for (Position pos : piece.getPositions()) {
            int row = pos.getRow();
            int col = pos.getCol();
            pieces[row][col] = null;
        }
    }

    public void clear() {
        for (int row = 0; row < height; row++) {
            for (int col = 0; col < width; col++) {
                pieces[row][col] = null;
            }
        }
    }

    // Fungsi untuk mencetak papan
    public void printPapan() {
        for (int row = 0; row < height; row++) {
            for (int col = 0; col < width; col++) {
                Piece piece = pieces[row][col];
                if (piece != null) {
                    System.out.print(piece.getId() + " ");
                } else {
                    System.out.print(". ");
                }
            }
            System.out.println();
        }
    }
}
```

Piece.java

```
import java.util.*;
```

```java
public class Piece {
    private List<Position> positions;
    private Orientation orientation;
    private String id; // ID piece

    public Piece(String id) {
        if (!id.matches("[A-Z]")) {
            throw new IllegalArgumentException("ID harus huruf A-Z.");
        }
        // Validasi 'P' dan 'K' hanya jika bukan kelas turunan
        if ((id.equals("P") || id.equals("K")) && getClass() == Piece.class) {
            throw new IllegalArgumentException("ID 'P' dan 'K' khusus digunakan untuk
primary piece dan pintu keluar.");
        }
        this.id = id;
        this.positions = new ArrayList<>();
    }

    // cctor
    public Piece(Piece piece) {
        this.id = piece.id;
        this.orientation = piece.orientation;
        this.positions = new ArrayList<>();
        for (Position position : piece.positions) {
            this.positions.add(new Position(position));
        }
    }

    public void addPosition(int row, int col) {
        if(positions.size() >= 3) {
            throw new IllegalStateException("Piece tidak boleh memiliki lebih dari 3
posisi.");
        }
        positions.add(new Position(row, col));
    }

    public void determineOrientation() {
        if (positions.size() < 2 || positions.size() > 3) {
            throw new IllegalArgumentException("Lebar piece harus 2 atau 3 posisi.");
        }

        int firstRow = positions.get(0).getRow();
        boolean allSameRow = true;

        for (int i = 1; i < positions.size(); i++) {
            if (positions.get(i).getRow() != firstRow) {
                allSameRow = false;
                break;
            }
        }
        orientation = allSameRow ? Orientation.HORIZONTAL : Orientation.VERTICAL;
```

```
        // sort position
        if(orientation == Orientation.HORIZONTAL) {
            positions.sort((p1, p2) -> Integer.compare(p1.getCol(), p2.getCol()));
        }
        else {
            positions.sort((p1, p2) -> Integer.compare(p1.getRow(), p2.getRow()));
        }
    }
    public void move(Direction direction) {
        for (Position position : positions) {
            position.gerakan(direction);
        }
    }
    public String getId() {
        return id;
    }
    public List<Position> getPositions() {
        return positions;
    }
    public void setPositions(List<Position> positions) {
        this.positions = new ArrayList<>(positions);
    }
    public Orientation getOrientation() {
        return orientation;
    }
    public Position getStartPosition() {
        return positions.get(0);
    }
    public Position getEndPosition() {
        return positions.get(positions.size() - 1);
    }
    public int getSize() {
        return positions.size();
    }

    @Override
    public boolean equals(Object obj) {
        // bandingin apakah dua piece itu sama
        if (this == obj) return true;
        if (obj == null || getClass() != obj.getClass()) return false;
        Piece piece = (Piece) obj;
        if(!id.equals(piece.id)) return false;
        if(!positions.equals(piece.positions)) return false;
        if(orientation != piece.orientation) return false;

        for (int i = 0; i < positions.size(); i++) {
            if (!positions.get(i).equals(piece.positions.get(i))) {
                return false;
            }
        }
```

```
        return true;
    }
    @Override
    public int hashCode() {
        // menghasilkan kode hash unik dari piece
        return Objects.hash(id, positions, orientation);
    }
}
```

PrimaryPiece.java

```
public class PrimaryPiece extends Piece {

    public PrimaryPiece() {
        super("P"); // Primary piece selalu diberi ID 'P'
    }

    @Override
    public void determineOrientation() {
        super.determineOrientation();
        // Memastikan primary piece memiliki dimensi yang valid
        // if (getSize() != 2) {
        // throw new IllegalArgumentException("Primary piece harus memiliki ukuran 2.");
        // }
    }

    public boolean canExitAt(PintuKeluar exit) {
        // memeriksa apakah orientasi primary piece cocok dengan pintu keluar
        if (getOrientation() == Orientation.HORIZONTAL && !exit.isHorizontal()) {
            return false;
        }
        if (getOrientation() == Orientation.VERTICAL && exit.isHorizontal()) {
            return false;
        }

        // Memeriksa apakah primary piece berada di posisi pintu keluar
        for (Position pos : getPositions()) {

            // Horizontal: keluar kiri/kanan
            if (getOrientation() == Orientation.HORIZONTAL) {
                // Kiri
                if (exit.getCol() == -1 && pos.getCol() == 0 && pos.getRow() ==
exit.getRow()) {
                    return true;
                }
                // Kanan
                if (exit.getCol() == exit.getPosition().getCol() && pos.getCol() ==
exit.getCol() - 1
                        && pos.getRow() == exit.getRow()) {
```

```
                    return true;
                }
            } else {
                // Vertical: keluar atas/bawah
                // Atas
                if (exit.getRow() == -1 && pos.getRow() == 0 && pos.getCol() ==
exit.getCol()) {
                    return true;
                }
                // Bawah
                if (exit.getRow() == exit.getPosition().getRow() && pos.getRow() ==
exit.getRow() - 1
                        && pos.getCol() == exit.getCol()) {
                    return true;
                }
            }
        }
        return false;
    }
}
```

Gerakan.java

```
public class Gerakan {
    private String pieceID;
    private Direction direction;

    public Gerakan(String pieceID, Direction direction) {
        this.pieceID = pieceID;
        this.direction = direction;
    }

    public String getPieceID() {
        return pieceID;
    }

    public Direction getDirection() {
        return direction;
    }

    // validasi apakah gerakan sesuai dengan orientasi piece
    public boolean isValidForPiece(Piece piece) {
        if (piece == null) {
            return false;
        }

        Orientation orientation = piece.getOrientation();

        // piece horizontal hanya bisa bergerak kiri-kanan
```

```
        if (orientation == Orientation.HORIZONTAL) {
            return direction == Direction.LEFT || direction == Direction.RIGHT;
        }
        // piece vertical hanya bisa bergerak atas-bawah
        else if (orientation == Orientation.VERTICAL) {
            return direction == Direction.UP || direction == Direction.DOWN;
        }

        return false;
    }

    @Override
    public String toString() {
        String dirString = "";
        switch (direction) {
            case UP:
                dirString = "atas";
                break;
            case DOWN:
                dirString = "bawah";
                break;
            case LEFT:
                dirString = "kiri";
                break;
            case RIGHT:
                dirString = "kanan";
                break;
        }
        return pieceID + "-" + dirString;
    }
}
```

Position.java

```
// ngetrack posisi piece di board dengan koordinat baris dan kolom.

import java.util.Objects;

public class Position {
    private int row;
    private int col;


    public Position(int row, int col) {
        this.row = row;
        this.col = col;
    }
    public Position(Position other) {
```

```
        this.row = other.row;
        this.col = other.col;
    }
    public int getRow() {
        return row;
    }
    public int getCol() {
        return col;
    }
    public void gerakan(Direction direction) {
        switch (direction) {
            case UP:
                row--;
                break;
            case DOWN:
                row++;
                break;
            case LEFT:
                col--;
                break;
            case RIGHT:
                col++;
                break;
        }
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj) return true;
        if (obj == null || getClass() != obj.getClass()) return false;
        Position position = (Position) obj;
        return row == position.row && col == position.col;
    }

    @Override
    public int hashCode() {
        return Objects.hash(row, col);
    }

}
```

PintuKeluar.java

```
public class PintuKeluar {
    private Position position;
    private boolean isHorizontal;

    public PintuKeluar(int row, int col, boolean isHorizontal) {
        this.position = new Position(row, col);
```

```
        this.isHorizontal = isHorizontal;
    }

    public Position getPosition() {
        return position;
    }

    public int getRow() {
        return position.getRow();
    }

    public int getCol() {
        return position.getCol();
    }

    public boolean isHorizontal() {
        return isHorizontal;
    }

    // memeriksa apakah pintu keluar cocok dengan orientasi piece
    public boolean matchOrientation(Orientation orientation) {
        return (isHorizontal && orientation == Orientation.HORIZONTAL) ||
               (!isHorizontal && orientation == Orientation.VERTICAL);
    }
}
```

Orientation.java

```
public enum Orientation {
    HORIZONTAL,
    VERTICAL
}
```

Direction.java

```
public enum Direction {
    UP,
    DOWN,
    LEFT,
    RIGHT;
}
```

GreedyBestFirstSearch.java

```java
import java.util.*;

public class GreedyBestFirstSearch {
    public enum HeuristicType {
        DISTANCE_TO_EXIT,
        BLOCKING_PIECES
    }

    private HeuristicType heuristicType;
    private Heuristic heuristic;

    public GreedyBestFirstSearch(HeuristicType heuristicType, Heuristic heuristic) {
        this.heuristic = heuristic;
        this.heuristicType = heuristicType;
    }

    /**
     * Runs the GBFS algorithm to find a solution
     * @return List of moves that lead to the solution, or empty list if no solution found
     */

    public List<Gerakan> solve() {
        PriorityQueue<Heuristic> openSet = new PriorityQueue<>(
            (a, b) -> Integer.compare(h(a), h(b)) // compare heuristic values
        );

        // closed set to track of visited states
        Set<Heuristic> closedSet = new HashSet<>();
        // add the initial state to the open set
        openSet.add(heuristic);

        // keep tracj of the number of iterations
        int explored = 0;
        while (!openSet.isEmpty()) {
            // get the state with the lowest heuristic value
            Heuristic current = openSet.poll();
            explored++;

            // check if the current state is the goal state
            if (current.isGoal()) {
                System.out.println("Found solution in " + explored + " iterations.");
                return current.getMoveHistory(); // return the moves that lead to the
solution
            }
            // add the current state to the closed set
```

```
            closedSet.add(current);

            // generate successors
            List<Heuristic> successors = current.getSuccessors();
            for (Heuristic successor : successors) {
                if (closedSet.contains(successor)) {
                    continue;
                }
                if (!openSet.contains(successor)) {
                    openSet.add(successor);
                }
            }
        }
        System.out.println("No solution found.");
        return new ArrayList<>(); // return empty list if no solution found
    }

    /**
     * Heuristic function to evaluate the state
     * @param state the state to evaluate
     * @return the heuristic value
     */

    private int h(Heuristic state) {
        switch (heuristicType) {
            case DISTANCE_TO_EXIT:
                return state.distanceToExit();
            case BLOCKING_PIECES:
                return state.countBlockingPieces();
            default:
                throw new IllegalArgumentException("Unknown heuristic type: " +
heuristicType);
        }
    }
    /**
     * f(n) function for the heuristic
     * @param state the state to evaluate
     * @return the heuristic value
     */
    private int f(Heuristic state) {
        return h(state);
    }

    // g(n) for counting the number of moves
    private int g(Heuristic state) {
        return state.getMoveCount();
    }
}
```

UniformCostSearch.java

```java
import java.util.*;

public class UniformCostSearch {

    private Heuristic heuristic;

    public UniformCostSearch(Heuristic heuristic) {
        this.heuristic = heuristic;
    }

    /**
     * Runs the UCS algorithm to find a solution
     * @return List of moves that lead to the solution, or empty list if no solution found
     */

    public List<Gerakan> solve() {
        PriorityQueue<Heuristic> openSet = new PriorityQueue<>(
            (a, b) -> Integer.compare(g(a), g(b)) // compare cost only
        );

        // closed set to track of visited states
        Set<Heuristic> closedSet = new HashSet<>();
        // add the initial state to the open set
        openSet.add(heuristic);

        // keep tracj of the number of iterations
        int explored = 0;
        while (!openSet.isEmpty()) {
            // get the state with the lowest heuristic value
            Heuristic current = openSet.poll();
            explored++;

            // check if the current state is the goal state
            if (current.isGoal()) {
                System.out.println("Found solution in " + explored + " iterations.");
                return current.getMoveHistory(); // return the moves that lead to the
solution
            }
            // add the current state to the closed set
            closedSet.add(current);

            // generate successors
            List<Heuristic> successors = current.getSuccessors();
            for (Heuristic successor : successors) {
                if (closedSet.contains(successor)) {
                    continue;
                }
```

```
                if (!openSet.contains(successor)) {
                    openSet.add(successor);
                }
            }
        }
        System.out.println("No solution found.");
        return new ArrayList<>(); // return empty list if no solution found
    }

    /**
     * f(n) function for the heuristic
     * @param state the state to evaluate
     * @return the heuristic value
     * f(n) = g(n)
     */
    private int f(Heuristic state) {
        return g(state);
    }

    // g(n) for count cost
    private int g(Heuristic state) {
        return state.getMoveCount();
    }
}
```

AStar.java

```
import java.util.*;

public class AStar {
    public enum HeuristicType {
        DISTANCE_TO_EXIT,
        BLOCKING_PIECES
    }

    private HeuristicType heuristicType;
    private Heuristic heuristic;

    public AStar(HeuristicType heuristicType, Heuristic heuristic) {
        this.heuristic = heuristic;
        this.heuristicType = heuristicType;
    }

    /**
     * Runs the A* algorithm to find a solution
     * @return List of moves that lead to the solution, or empty list if no solution found
     */

    public List<Gerakan> solve() {
```

```
        PriorityQueue<Heuristic> openSet = new PriorityQueue<>(
            (a, b) -> Integer.compare(f(a), f(b)) // compare g(n) + h(n)
        );

        // closed set to track of visited states
        Set<Heuristic> closedSet = new HashSet<>();
        // add the initial state to the open set
        openSet.add(heuristic);

        // keep tracj of the number of iterations
        int explored = 0;
        while (!openSet.isEmpty()) {
            // get the state with the lowest heuristic value
            Heuristic current = openSet.poll();
            explored++;

            // check if the current state is the goal state
            if (current.isGoal()) {
                System.out.println("Found solution in " + explored + " iterations.");
                return current.getMoveHistory(); // return the moves that lead to the
solution
            }
            // add the current state to the closed set
            closedSet.add(current);

            // generate successors
            List<Heuristic> successors = current.getSuccessors();
            for (Heuristic successor : successors) {
                if (closedSet.contains(successor)) {
                    continue;
                }
                if (!openSet.contains(successor)) {
                    openSet.add(successor);
                }
            }
        }
        System.out.println("No solution found.");
        return new ArrayList<>(); // return empty list if no solution found
    }

    /**
     * Heuristic function to evaluate the state
     * @param state the state to evaluate
     * @return the heuristic value
     */

    private int h(Heuristic state) {
        switch (heuristicType) {
            case DISTANCE_TO_EXIT:
                return state.distanceToExit();
            case BLOCKING_PIECES:
```

```
                return state.countBlockingPieces();
            default:
                throw new IllegalArgumentException("Unknown heuristic type: " +
heuristicType);
        }
    }

     /**
     * g(n) - Path cost function
     * Represents the cost of the path from the initial state to the current state
     * For Rush Hour, this is simply the number of moves made so far
     */
    private int g(Heuristic state) {
        return state.getMoveCount();
    }

    /**
     * f(n) function for the heuristic
     * @param state the state to evaluate
     * @return the heuristic value
     * g(n) + h(n)
     * g(n) is the cost to reach the current state
     * h(n) is the estimated cost to reach the goal state
     */
    private int f(Heuristic state) {
        return g(state) + h(state);
    }
}
```

Heuristic.java

```java
import java.util.*;

public class Heuristic {
    private Map<String, Piece> pieces;
    private List<Gerakan> moveHistory;
    private PrimaryPiece primaryPiece;
    private PintuKeluar pintuKeluar;
    private Papan papan;

    public Heuristic(Map<String, Piece> pieces, PintuKeluar pintuKeluar, Papan papan) {
        this.pieces = pieces;
        this.moveHistory = new ArrayList<>();
        this.pintuKeluar = pintuKeluar;
        this.papan = papan;

        // deep copy pieces
        for (String pieceId : pieces.keySet()) {
            Piece originalPiece = pieces.get(pieceId);
            if (originalPiece instanceof PrimaryPiece) {
                this.primaryPiece = new PrimaryPiece();

                for (Position position : originalPiece.getPositions()) {
                    this.primaryPiece.addPosition(position.getRow(), position.getCol());
                }
                this.primaryPiece.determineOrientation();
                this.pieces.put(pieceId, this.primaryPiece);
            } else {
                Piece newPiece = new Piece(originalPiece);
                this.pieces.put(pieceId, newPiece);
            }
        }
        this.papan.clear();
        for (Piece piece : this.pieces.values()) {
            this.papan.addPiece(piece);
        }
    }

    private Heuristic(Heuristic parentState, Gerakan move) {
        this.papan = new Papan(parentState.papan.getWidth(), parentState.papan.getHeight());
        this.pieces = new HashMap<>();
        this.pintuKeluar = parentState.pintuKeluar;

        // copy all pieces
        for (String pieceId : parentState.pieces.keySet()) {
            Piece originalPiece = parentState.pieces.get(pieceId);

            if (originalPiece instanceof PrimaryPiece) {
                this.primaryPiece = new PrimaryPiece();
```

```
                for (Position position : originalPiece.getPositions()) {
                    this.primaryPiece.addPosition(position.getRow(), position.getCol());
                }
                this.primaryPiece.determineOrientation();
                this.pieces.put(pieceId, this.primaryPiece);
            } else {
                Piece newPiece = new Piece(originalPiece);
                this.pieces.put(pieceId, newPiece);
            }
        }

        // **Clear papan supaya posisi lama hilang**
        this.papan.clear();

        // copy move history and add new move
        this.moveHistory = new ArrayList<>(parentState.moveHistory);
        this.moveHistory.add(move);

        // apply the move to this state
        applyMove(move);

        // places the pieces on the board
        for (Piece piece : this.pieces.values()) {
            try {
                this.papan.addPiece(piece);
            } catch (IllegalArgumentException e) {
                throw new IllegalStateException("Invalid move: " + e.getMessage());
            }
        }
    }

    private void applyMove(Gerakan move) {
        String pieceId = move.getPieceID();
        Direction direction = move.getDirection();
        Piece piece = pieces.get(pieceId);

        if (piece != null) {
            piece.move(direction);
        } else {
            throw new IllegalArgumentException("Piece with ID " + pieceId + " not found.");
        }
    }

    // make successors
    public List<Heuristic> getSuccessors() {
        List<Heuristic> successors = new ArrayList<>();

        for (String pieceId : pieces.keySet()) {
            Piece piece = pieces.get(pieceId);
```

```
            for (Direction direction : Direction.values()) {
                Gerakan move = new Gerakan(pieceId, direction);

                // skip invalid moves
                if (!move.isValidForPiece(piece)) {
                    continue;
                }
                // check if the move is valid
                if (isValidMove(piece, direction)) {
                    // create a new state with the move applied
                    Heuristic successor = new Heuristic(this, move);
                    successors.add(successor);
                }
            }
        }
        return successors;
    }

    public void addPrimaryPiece(Piece piece) {
        // Validasi posisi untuk primary piece
        if (piece.getId() == "P") {
            for (Position pos : piece.getPositions()) {
                int row = pos.getRow();
                int col = pos.getCol();

                // Primary piece hanya boleh keluar jika posisinya menuju pintu keluar
                if (row < 0 || row >= papan.getHeight() || col < 0 || col >=
papan.getWidth()) {
                    if (!primaryPiece.canExitAt(pintuKeluar)) {
                        throw new IllegalArgumentException("Primary piece keluar dari batas
papan tanpa pintu keluar.");
                    }
                } else if (papan.getPiece(row, col) != null) {
                    throw new IllegalArgumentException("Cell pada papan sudah terisi.");
                }
            }
        } else {
            // Validasi posisi untuk piece biasa
            for (Position pos : piece.getPositions()) {
                int row = pos.getRow();
                int col = pos.getCol();
                if (row < 0 || row >= papan.getHeight() || col < 0 || col >=
papan.getWidth()) {
                    throw new IllegalArgumentException("Piece keluar dari batas papan.");
                }
                if (papan.getPiece(row, col) != null) {
                    throw new IllegalArgumentException("Cell pada papan sudah terisi.");
                }
            }
        }
```

```
        // Tempatkan piece ke papan sesuai posisi-posisinya
        for (Position pos : piece.getPositions()) {
            int row = pos.getRow();
            int col = pos.getCol();
            if (row >= 0 && row < papan.getHeight() && col >= 0 && col < papan.getWidth()) {
                papan.setPiece(row, col, piece);
            }
        }
    }

    private boolean isValidMove(Piece piece, Direction direction) {
        Piece tempPiece = new Piece(piece);
        tempPiece.move(direction);

        for (Position position : tempPiece.getPositions()) {
            int row = position.getRow();
            int col = position.getCol();

            // Jika posisi berada di luar papan
            if (row < 0 || row >= papan.getHeight() || col < 0 || col >= papan.getWidth()) {
                // Cek jika ini adalah primary piece dan keluar melalui pintu keluar
                if (piece.getId().equals("P") && pintuKeluar != null) {
                    PrimaryPiece tempPrimary = new PrimaryPiece();
                    for (Position pos : tempPiece.getPositions()) {
                        tempPrimary.addPosition(pos.getRow(), pos.getCol());
                    }
                    tempPrimary.determineOrientation();
                }
                return false;
            }

            // cek tabrakan dengan piece lain
            Piece otherPiece = papan.getPiece(row, col);
            if (otherPiece != null && !otherPiece.getId().equals(piece.getId())) {
                return false;
            }
        }
        return true;
    }

    // check if the primary piece is at the exit
    public boolean isGoal() {
        return primaryPiece.canExitAt(pintuKeluar);
    }

    // heuristic 1 : distance to exit
    public int distanceToExit() {
        Position exitPosition = pintuKeluar.getPosition();

        if (primaryPiece.getOrientation() == Orientation.HORIZONTAL) {
            Position endPosition = primaryPiece.getEndPosition();
```

```
            if (exitPosition.getRow() == endPosition.getRow()) {
                return Math.abs(exitPosition.getCol() - endPosition.getCol());
            } else {
                return Integer.MAX_VALUE;
            }
        } else {
            Position endPosition = primaryPiece.getEndPosition();
            if (exitPosition.getCol() == endPosition.getCol()) {
                return Math.abs(exitPosition.getRow() - endPosition.getRow());
            } else {
                return Integer.MAX_VALUE;
            }
        }
    }

    // heuristic 2 : count the number of blocking pieces between the primary piece
    // and the exit door
    public int countBlockingPieces() {
        int count = 0;
        Position exitPosition = pintuKeluar.getPosition();

        if (primaryPiece.getOrientation() == Orientation.HORIZONTAL) {
            Position endPosition = primaryPiece.getEndPosition();
            int row = endPosition.getRow();

            int startCol = endPosition.getCol() + 1;
            int endCol = exitPosition.getCol();

            for (int col = startCol; col <= endCol; col++) {
                if (col < papan.getWidth() && papan.getPiece(row, col) != null) {
                    count++;
                }
            }
        } else {
            Position endPosition = primaryPiece.getEndPosition();
            int col = endPosition.getCol();

            int startRow = endPosition.getRow() + 1;
            int endRow = exitPosition.getRow();

            for (int row = startRow; row <= endRow; row++) {
                if (row < papan.getHeight() && papan.getPiece(row, col) != null) {
                    count++;
                }
            }
        }
        return count;
    }

    public Map<String, Piece> getPieces() {
        return pieces;
```

```
    }

    public List<Gerakan> getMoveHistory() {
        return moveHistory;
    }

    public PrimaryPiece getPrimaryPiece() {
        return primaryPiece;
    }

    public Papan getPapan() {
        return papan;
    }

    public PintuKeluar getPintuKeluar() {
        return pintuKeluar;
    }

    // count the number of moves made
    public int getMoveCount() {
        return moveHistory.size();
    }

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

        Heuristic other = (Heuristic) obj;

        // compare pieces positions
        if (this.pieces.size() != other.pieces.size())
            return false;

        for (String pieceId : this.pieces.keySet()) {
            Piece thisPiece = this.pieces.get(pieceId);
            Piece otherPiece = other.pieces.get(pieceId);

            if (otherPiece == null || !thisPiece.equals(otherPiece))
                return false;
        }
        return true;
    }

    @Override
    public int hashCode() {
        int hash = 7; // initial hash value
        for (String pieceId : pieces.keySet()) {
            Piece piece = pieces.get(pieceId);
```

```
            hash = 31 * hash + (piece != null ? piece.hashCode() : 0); // combine hash codes
        }
        return hash;
    }

    // print the current board
    public void currentBoard() {
        papan.printPapan();
    }

}
```

Main.java

```
import java.io.IOException;
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        try {
            // Get the input file name
            System.out.print("Enter the puzzle configuration file name: ");
            String fileName = scanner.nextLine();

            // Read the puzzle configuration
            InputHandler inputHandler = new InputHandler();
            inputHandler.readConfigFromFile(fileName);

            // Get the map of pieces, primary piece, exit door, and the board
            Map<String, Piece> pieces = inputHandler.getPieces();
            PrimaryPiece primaryPiece = inputHandler.getPrimaryPiece();
            PintuKeluar pintuKeluar = inputHandler.getPintuKeluar();
            Papan papan = inputHandler.getPapan();

            // Choose the pathfinding algorithm
            System.out.println("\nSelect pathfinding algorithm:");
            System.out.println("1. Uniform Cost Search (UCS)");
            System.out.println("2. A* Search");
            System.out.println("3. Greedy Best-First Search (GBFS)");
            System.out.print("Enter your choice (1-3): ");
            int algorithmChoice = Integer.parseInt(scanner.nextLine());

            // Choose the heuristic function for A* and GBFS
            AStar.HeuristicType heuristicType = AStar.HeuristicType.DISTANCE_TO_EXIT; //
default
            GreedyBestFirstSearch.HeuristicType gbfsHeuristicType =
GreedyBestFirstSearch.HeuristicType.DISTANCE_TO_EXIT; // default
```

```
            if (algorithmChoice == 2 || algorithmChoice == 3) {
                System.out.println("\nSelect heuristic function:");
                System.out.println("1. Distance to Exit");
                System.out.println("2. Blocking Pieces");
                System.out.print("Enter your choice (1-2): ");
                int heuristicChoice = Integer.parseInt(scanner.nextLine());

                if (heuristicChoice == 1) {
                    heuristicType = AStar.HeuristicType.DISTANCE_TO_EXIT;
                } else if (heuristicChoice == 2) {
                    heuristicType = AStar.HeuristicType.BLOCKING_PIECES;
                } else {
                    System.out.println("Invalid choice. Using Distance to Exit as
default.");
                }
            }

            // Create the initial state heuristic
            Heuristic initialState = new Heuristic(pieces, pintuKeluar, papan);

            // Solve the puzzle using the selected algorithm
            List<Gerakan> solution = null;
            long startTime = System.currentTimeMillis();

            switch (algorithmChoice) {
                case 1:
                    System.out.println("\nSolving with Uniform Cost Search (UCS)...");
                    UniformCostSearch ucs = new UniformCostSearch(initialState);
                    solution = ucs.solve();
                    break;
                case 2:
                    System.out.println("\nSolving with A* Search...");
                    AStar aStar = new AStar(heuristicType, initialState);
                    solution = aStar.solve();
                    break;
                case 3:
                    System.out.println("\nSolving with Greedy Best-First Search (GBFS)...");
                    GreedyBestFirstSearch gbfs = new
GreedyBestFirstSearch(gbfsHeuristicType, initialState);
                    solution = gbfs.solve();
                    break;
                default:
                    System.out.println("Invalid choice. Using Uniform Cost Search as
default.");
                    UniformCostSearch defaultUcs = new UniformCostSearch(initialState);
                    solution = defaultUcs.solve();
            }

            long endTime = System.currentTimeMillis();
            long executionTime = endTime - startTime;
```

```
            // Print execution statistics
            System.out.println("\nExecution time: " + executionTime + " ms");

            // Create output handler
            OutputHandler outputHandler = new OutputHandler(solution, pieces, pintuKeluar,
papan);

            // Ask user if they want to save to file
            System.out.println("\nDo you want to save the solution to a file? (y/n)");
            String saveChoice = scanner.nextLine().trim().toLowerCase();

            if (saveChoice.equals("y") || saveChoice.equals("yes")) {
                System.out.print("Enter the output file name: ");
                String outputFileName = scanner.nextLine();
                outputHandler.saveSolutionToFile(outputFileName);
            }

            // Print the solution to console
            outputHandler.printSolution();

        } catch (IOException e) {
            System.out.println("Error reading/writing file: " + e.getMessage());
        } catch (Exception e) {
            System.out.println("An error occurred: " + e.getMessage());
            e.printStackTrace();
        } finally {
            scanner.close();
        }
    }
}
```

InputHandler.java

```
import java.io.*;
import java.util.*;

public class InputHandler {
    private int width;
    private int height;
    private int numPieces;
    private Map<String, Piece> pieces;
    private PrimaryPiece primaryPiece;
    private PintuKeluar pintuKeluar;
    private Papan papan;

    public InputHandler() {
        pieces = new HashMap<>();
```

```
    }

    /**
     * Reads the puzzle configuration from a file
     *
     * @param fileName the name of the file to read
     * @throws IOException if there's an error reading the file
     */
    public void readConfigFromFile(String fileName) throws IOException {
        BufferedReader reader = new BufferedReader(new FileReader(fileName));

        // Read dimensions
        String[] dimensions = reader.readLine().split(" ");
        height = Integer.parseInt(dimensions[0]);
        width = Integer.parseInt(dimensions[1]);

        // Read number of non-primary pieces (unused in this example, but read anyway)
        numPieces = Integer.parseInt(reader.readLine());

        // Read all remaining lines (including possible extra border lines)
        ArrayList<String> allLines = new ArrayList<>();
        String line;
        while ((line = reader.readLine()) != null) {
            if (!line.trim().isEmpty()) {
                allLines.add(line);
            }
        }
        reader.close();

        // Cari posisi K (pintu keluar)
        // Bisa di luar papan (di baris sebelum papan, sesudah papan, atau di luar
        // kolom)
        boolean foundK = false;
        int exitRow = -1, exitCol = -1;
        boolean isHorizontal = false;

        // Misal papan dimulai di baris tertentu dalam allLines:
        // Karena kita baca height baris papan, coba cari window baris mana yang jadi
        // papan:
        // Asumsi: papan ada di tengah allLines, bisa coba cari window height baris yang
        // cocok (atau asumsi langsung di tengah)

        // Sederhana: cari semua K di allLines, cari posisi dengan memperhitungkan papan
        // di tengah:
        for (int i = 0; i < allLines.size(); i++) {
            String currLine = allLines.get(i);
            for (int j = 0; j < currLine.length(); j++) {
                if (currLine.charAt(j) == 'K') {
                    foundK = true;
                    exitRow = i;
                    exitCol = j;
```

```
                    break;
                }
            }
            if (foundK)
                break;
        }

        if (!foundK) {
            throw new IllegalArgumentException("Pintu keluar (K) tidak ditemukan di
papan.");
        }

        // Sekarang kita asumsikan papan berada di allLines dari baris tertentu
        // Kita cari window baris di allLines yang ukurannya height dengan asumsi papan
        // ada di tengah (atau coba cocokkan)
        // Supaya mudah, coba papan dimulai di baris: startIndex = exitRow - height/2
        // (asumsi pintu keluar di luar board)
        int startIndex = -1;

        // Coba cari startIndex agar allLines.subList(startIndex, startIndex + height)
        // panjangnya height dan tiap baris >= width
        for (int possibleStart = 0; possibleStart <= allLines.size() - height;
possibleStart++) {
            boolean valid = true;
            for (int k = 0; k < height; k++) {
                if (allLines.get(possibleStart + k).length() < width) {
                    valid = false;
                    break;
                }
            }
            if (valid) {
                startIndex = possibleStart;
                break;
            }
        }

        if (startIndex == -1) {
            throw new IllegalArgumentException("Tidak ditemukan baris papan yang valid di
file input.");
        }

        // Inisialisasi papan 2D
        char[][] board = new char[height][width];

        boolean pintuKiri = false;
        int pintuKiriRow = -1;

        if (exitRow >= startIndex && exitRow < startIndex + height && exitCol == 0) {
            pintuKiri = true;
            pintuKiriRow = exitRow - startIndex;
            for (int i = 0; i < height; i++) {
```

```
                String rowLine = allLines.get(startIndex + i);
                if (i == pintuKiriRow) {
                    // Baris yang ada K di kolom 0 tidak boleh diawali spasi
                    if (rowLine.charAt(0) == ' ') {
                        throw new IllegalArgumentException("Baris pintu kiri (K) tidak boleh
diawali spasi.");
                    }
                } else {
                    // Baris lain harus diawali spasi
                    if (rowLine.length() <= width || rowLine.charAt(0) != ' ') {
                        throw new IllegalArgumentException(
                                "Baris ke-" + (i + 1) + " harus diawali spasi jika pintu
keluar di kiri.");
                    }
                }
            }
        }

        for (int i = 0; i < height; i++) {
            String rowLine = allLines.get(startIndex + i);
            if (pintuKiri) {
                if (i == pintuKiriRow && rowLine.charAt(0) == 'K') {
                    // Baris pintu: abaikan K, ambil substring setelah K
                    rowLine = rowLine.substring(1);
                } else if (i != pintuKiriRow && rowLine.length() > width) {
                    // Baris lain: abaikan spasi pertama
                    rowLine = rowLine.substring(1);
                }
            }
            for (int j = 0; j < width; j++) {
                board[i][j] = rowLine.charAt(j);
            }
        }

        // Tentukan apakah pintu keluar horizontal atau vertical
        // Jika exitRow < startIndex → pintu keluar di atas papan → horizontal
        // Jika exitRow >= startIndex + height → pintu keluar di bawah papan →
        // horizontal
        // Jika exitRow di dalam range papan → pintu keluar di samping → vertical
        if (exitRow < startIndex || exitRow >= startIndex + height) {
            isHorizontal = false;
            // Karena pintu di luar baris papan, exitRow relatif ke papan = -1 (atas) atau
            // height (bawah)
            if (exitRow < startIndex) {
                exitRow = -1; // atas
            } else {
                exitRow = height; // bawah
            }
        } else {
            isHorizontal = true;
            // pintu keluar di samping kiri (-1) atau kanan (width)
```

```
            if (exitCol == 0) {
                exitCol = -1;
            } else if (exitCol >= width) {
                exitCol = width;
            }
            // exitRow sudah disesuaikan karena di dalam papan
            exitRow = exitRow - startIndex;
        }

        pintuKeluar = new PintuKeluar(exitRow, exitCol, isHorizontal);

        // Simpan dan proses papan
        papan = new Papan(width, height);
        processBoardConfiguration(board);

        // Validasi jumlah piece (N) sesuai input, N tidak menghitung primary piece
        int nonPrimaryCount = 0;
        for (String id : pieces.keySet()) {
            if (!id.equals("P")) {
                nonPrimaryCount++;
            }
        }
        if (nonPrimaryCount != numPieces) {
            throw new IllegalArgumentException("Jumlah piece pada papan tidak sesuai dengan
input.");
        }

        // Validasi: primary piece dan pintu keluar harus sesuai aturan orientasi dan
        // baris/kolom
        if (primaryPiece != null && pintuKeluar != null) {
            if (primaryPiece.getOrientation() == Orientation.HORIZONTAL) {
                if (!pintuKeluar.isHorizontal()) {
                    throw new IllegalArgumentException("Primary piece horizontal, pintu
keluar harus horizontal.");
                }
                boolean sameRow = false;
                for (Position pos : primaryPiece.getPositions()) {
                    if (pos.getRow() == pintuKeluar.getRow()) {
                        sameRow = true;
                        break;
                    }
                }
                if (!sameRow) {
                    throw new IllegalArgumentException(
                            "Primary piece horizontal, pintu keluar harus di baris yang
sama.");
                }
            } else if (primaryPiece.getOrientation() == Orientation.VERTICAL) {
                if (pintuKeluar.isHorizontal()) {
                    throw new IllegalArgumentException("Primary piece vertikal, pintu keluar
harus vertikal.");
```

```
                }
                boolean sameCol = false;
                for (Position pos : primaryPiece.getPositions()) {
                    if (pos.getCol() == pintuKeluar.getCol()) {
                        sameCol = true;
                        break;
                    }
                }
                if (!sameCol) {
                    throw new IllegalArgumentException(
                            "Primary piece vertikal, pintu keluar harus di kolom yang
sama.");
                }
            }
        }

        // Validasi primary piece dan pintu keluar
        if (primaryPiece != null && pintuKeluar != null) {
            if (primaryPiece.getOrientation() == Orientation.HORIZONTAL) {
                if (!pintuKeluar.isHorizontal()) {
                    throw new IllegalArgumentException("Posisi pintu keluar tidak sesuai.");
                }
                boolean sameRow = false;
                for (Position pos : primaryPiece.getPositions()) {
                    if (pos.getRow() == pintuKeluar.getRow()) {
                        sameRow = true;
                        break;
                    }
                }
                if (!sameRow) {
                    throw new IllegalArgumentException(
                            "Posisi pintu keluar tidak sesuai.");
                }
            } else if (primaryPiece.getOrientation() == Orientation.VERTICAL) {
                if (pintuKeluar.isHorizontal()) {
                    throw new IllegalArgumentException("Posisi pintu keluar tidak sesuai.");
                }
                boolean sameCol = false;
                for (Position pos : primaryPiece.getPositions()) {
                    if (pos.getCol() == pintuKeluar.getCol()) {
                        sameCol = true;
                        break;
                    }
                }
                if (!sameCol) {
                    throw new IllegalArgumentException(
                            "Posisi pintu keluar tidak sesuai.");
                }
            }
        }
```

```
        checkMultipleK(allLines);
        checkBarisKolomValid(allLines, startIndex, height, width, pintuKiri, pintuKiriRow);

    }

    /**
     * Processes the board configuration to identify all pieces
     *
     * @param board the board configuration as a 2D array of characters
     */
    private void processBoardConfiguration(char[][] board) {
        processPieces(board);

        for (Piece piece : pieces.values()) {
            papan.addPiece(piece);
        }
    }

    /**
     * Identifies all pieces in the board configuration
     *
     * @param board the board configuration
     */
    private void processPieces(char[][] board) {
        boolean[][] processed = new boolean[height][width];
        HashSet<String> idSet = new HashSet<>();

        for (int i = 0; i < height; i++) {
            for (int j = 0; j < width; j++) {

                if (processed[i][j] || board[i][j] == '.' || board[i][j] == 'K') {
                    continue;
                }

                String pieceId = String.valueOf(board[i][j]);
                if (idSet.contains(pieceId)) {
                    throw new IllegalArgumentException("Terdapat id piece yang sama: " +
pieceId);
                }
                idSet.add(pieceId);

                Piece piece;
                if (pieceId.equals("P")) {
                    primaryPiece = new PrimaryPiece();
                    piece = primaryPiece;
                } else {
                    piece = new Piece(pieceId);
                }

                identifyPieceCells(board, processed, piece, i, j, pieceId);
                piece.determineOrientation();
```

```
                pieces.put(pieceId, piece);
            }
        }
    }

    private void identifyPieceCells(char[][] board, boolean[][] processed, Piece piece, int
startRow, int startCol,
            String pieceId) {
        piece.addPosition(startRow, startCol);
        processed[startRow][startCol] = true;

        int[][] directions = { { 0, 1 }, { 1, 0 }, { 0, -1 }, { -1, 0 } };
        for (int[] dir : directions) {
            int newRow = startRow + dir[0];
            int newCol = startCol + dir[1];

            if (isValidCell(newRow, newCol) && !processed[newRow][newCol] &&
                    board[newRow][newCol] == pieceId.charAt(0)) {
                identifyPieceCells(board, processed, piece, newRow, newCol, pieceId);
            }
        }
    }

    private boolean isValidCell(int row, int col) {
        return row >= 0 && row < height && col >= 0 && col < width;
    }

    // Fungsi untuk mengecek jika ada lebih dari satu K di input
    private void checkMultipleK(ArrayList<String> allLines) {
        int kCount = 0;
        for (String line : allLines) {
            for (int i = 0; i < line.length(); i++) {
                if (line.charAt(i) == 'K') {
                    kCount++;
                }
            }
        }
        if (kCount > 1) {
            throw new IllegalArgumentException("Terdapat lebih dari satu pintu keluar (K)
pada input.");
        }
    }



    // Fungsi untuk validasi kelebihan baris dan kolom sebelum parsing papan
    private void checkBarisKolomValid(ArrayList<String> allLines, int startIndex, int
height, int width,
            boolean pintuKiri, int pintuKiriRow) {
        // Validasi jumlah baris window papan
        if (startIndex < 0 || startIndex + height > allLines.size()) {
```

```
            throw new IllegalArgumentException("Window papan di file input tidak valid.");
        }
        // Validasi jumlah baris
        int barisPapan = 0;
        for (int i = 0; i < height; i++) {
            String rowLine = allLines.get(startIndex + i);
            int effectiveLength = rowLine.length();

            // Handle pintu kiri
            if (pintuKiri) {
                if (i == pintuKiriRow) {
                    if (rowLine.length() == 0 || rowLine.charAt(0) != 'K') {
                        throw new IllegalArgumentException("Baris pintu kiri harus diawali
'K'.");
                    }
                    // Abaikan K di kolom 0
                    effectiveLength = rowLine.length() - 1;
                } else {
                    if (rowLine.length() == 0 || rowLine.charAt(0) != ' ') {
                        throw new IllegalArgumentException("Baris non-pintu kiri harus
diawali spasi.");
                    }
                    // Abaikan spasi pertama
                    effectiveLength = rowLine.length() - 1;
                }
            } else {
                // Handle pintu kanan: baris yang kelebihan 1 kolom dan kolom terakhir
adalah
                // 'K'
                if (rowLine.length() > width && rowLine.charAt(rowLine.length() - 1) == 'K')
{
                    effectiveLength = rowLine.length() - 1;
                }
                // Handle pintu atas/bawah: baris normal, tidak perlu pengurangan
                // Handle K di kolom 0 (bukan pintu kiri)
                else if (rowLine.length() > 0 && rowLine.charAt(0) == 'K') {
                    effectiveLength = rowLine.length() - 1;
                }
            }

            // Sekarang effectiveLength adalah jumlah kolom papan sebenarnya (tanpa K/spasi)
            if (effectiveLength < width) {
                throw new IllegalArgumentException("Baris ke-" + (i + 1) + " kurang dari
lebar papan.");
            }
            if (effectiveLength > width) {
                throw new IllegalArgumentException("Baris ke-" + (i + 1) + " melebihi lebar
papan.");
            }
            barisPapan++;
        }
```

```
        // Validasi jumlah baris papan
        if (barisPapan != height) {
            throw new IllegalArgumentException("Jumlah baris papan tidak sesuai dengan
height pada input. Ditemukan: "
                    + barisPapan + ", seharusnya: " + height);
        }
    }

    public Map<String, Piece> getPieces() {
        return pieces;
    }

    public PrimaryPiece getPrimaryPiece() {
        return primaryPiece;
    }

    public PintuKeluar getPintuKeluar() {
        return pintuKeluar;
    }

    public Papan getPapan() {
        return papan;
    }

}
```

OutputHandler.java

```
import java.io.*;
import java.util.*;

public class OutputHandler {
    private List<Gerakan> moveHistory;
    private Map<String, Piece> initialPieces;
    private PintuKeluar pintuKeluar;
    private Papan initialPapan;

    public OutputHandler(List<Gerakan> moveHistory, Map<String, Piece> initialPieces,
            PintuKeluar pintuKeluar, Papan initialPapan) {
        this.moveHistory = moveHistory;
        this.initialPieces = initialPieces;
        this.pintuKeluar = pintuKeluar;
        this.initialPapan = initialPapan;
    }

    /**
```

```
     * Prints the solution sequence to the console
     */
    public void printSolution() {
        if (moveHistory.isEmpty()) {
            System.out.println("No solution found.");
            return;
        }

        System.out.println("Papan Awal");
        printBoard(initialPapan, pintuKeluar, null);
        System.out.println();

        // Create a deep copy of the initial state
        Map<String, Piece> currentPieces = deepCopyPieces(initialPieces);
        Papan currentPapan = new Papan(initialPapan.getWidth(), initialPapan.getHeight());

        // Place pieces on the board
        for (Piece piece : currentPieces.values()) {
            currentPapan.addPiece(piece);
        }

        boolean primaryPieceExited = false;

        // Execute each move and print the board state
        for (int i = 0; i < moveHistory.size(); i++) {
            Gerakan move = moveHistory.get(i);
            System.out.println("Gerakan " + (i + 1) + ": " + move);

            // Apply the move
            String pieceId = move.getPieceID();
            Piece piece = currentPieces.get(pieceId);

            // Create a new board for this state
            currentPapan = new Papan(initialPapan.getWidth(), initialPapan.getHeight());

            // Apply the move to the piece
            piece.move(move.getDirection());

            // Check if primary piece has exited
            if (pieceId.equals("P") && ((PrimaryPiece)piece).canExitAt(pintuKeluar)) {
                primaryPieceExited = true;
                // If primary piece exited, remove it from the current pieces
                currentPapan.removePiece(piece);
                currentPieces.remove("P");
                System.out.println("Primary piece successfully exited!");
            }

            // Place all pieces on the new board
            for (Piece p : currentPieces.values()) {
                if (!(p instanceof PrimaryPiece) || !primaryPieceExited) {
                    currentPapan.addPiece(p);
```

```
                }
            }

            // Print the board state
            printBoard(currentPapan, pintuKeluar, pieceId);
            System.out.println();
        }

        System.out.println("Solution found with " + moveHistory.size() + " moves.");
    }

    /**
     * Creates a deep copy of the pieces map
     *
     * @param pieces the original pieces map
     * @return a deep copy of the pieces map
     */
    private Map<String, Piece> deepCopyPieces(Map<String, Piece> pieces) {
        Map<String, Piece> copy = new HashMap<>();
        for (String pieceId : pieces.keySet()) {
            Piece originalPiece = pieces.get(pieceId);

            if (originalPiece instanceof PrimaryPiece) {
                PrimaryPiece primaryCopy = new PrimaryPiece();
                for (Position pos : originalPiece.getPositions()) {
                    primaryCopy.addPosition(pos.getRow(), pos.getCol());
                }
                primaryCopy.determineOrientation();
                copy.put(pieceId, primaryCopy);
            } else {
                Piece pieceCopy = new Piece(originalPiece);
                copy.put(pieceId, pieceCopy);
            }
        }
        return copy;
    }

    /**
     * Saves the solution sequence to a text file
     *
     * @param filename the name of the file to save to
     * @throws IOException if there's an error writing to the file
     */
    public void saveSolutionToFile(String filename) throws IOException {
        if (moveHistory.isEmpty()) {
            try (PrintWriter writer = new PrintWriter(new FileWriter(filename))) {
                writer.println("No solution found.");
            }
            return;
        }
```

```
        try (PrintWriter writer = new PrintWriter(new FileWriter(filename))) {
            writer.println("Papan Awal");
            saveBoardToFile(writer, initialPapan);
            writer.println();

            // Create a deep copy of the initial state
            Map<String, Piece> currentPieces = deepCopyPieces(initialPieces);
            Papan currentPapan = new Papan(initialPapan.getWidth(),
initialPapan.getHeight());

            // Place pieces on the board
            for (Piece piece : currentPieces.values()) {
                currentPapan.addPiece(piece);
            }

            // Execute each move and save the board state
            for (int i = 0; i < moveHistory.size(); i++) {
                Gerakan move = moveHistory.get(i);
                writer.println("Gerakan " + (i + 1) + ": " + move);

                // Apply the move
                String pieceId = move.getPieceID();
                Piece piece = currentPieces.get(pieceId);

                // Create a new board for this state
                currentPapan = new Papan(initialPapan.getWidth(), initialPapan.getHeight());

                // Apply the move to the piece
                piece.move(move.getDirection());

                // Place all pieces on the new board
                for (Piece p : currentPieces.values()) {
                    currentPapan.addPiece(p);
                }

                // Save the board state
                saveBoardToFile(writer, currentPapan);
                writer.println();
            }

            writer.println("Solution found with " + moveHistory.size() + " moves.");
            System.out.println("Solution saved to file: " + filename);
        }
    }

    /**
     * Saves the board state to a file
     *
     * @param writer the PrintWriter to write to
     * @param papan  the board to save
     */
```

```java
    private void saveBoardToFile(PrintWriter writer, Papan papan) {
        int height = papan.getHeight();
        int width = papan.getWidth();

        for (int row = 0; row < height; row++) {
            StringBuilder line = new StringBuilder();
            for (int col = 0; col < width; col++) {
                Piece piece = papan.getPiece(row, col);

                if (piece != null) {
                    line.append(piece.getId());
                } else {
                    // Check if this is the exit door position
                    if (pintuKeluar.getRow() == row && pintuKeluar.getCol() == col) {
                        line.append("K");
                    } else {
                        line.append(".");
                    }
                }
            }
            writer.println(line.toString());
        }
    }

    /**
     * Prints the board state with colored output for the primary piece, exit door,
     * and moved piece
     * s
     *
     * @param papan the board to print
     */
    private void printBoard(Papan papan, PintuKeluar pintuKeluar, String movedPieceId) {
        // ANSI color codes
        final String RESET = "\u001B[0m";
        final String PRIMARY_COLOR = "\u001B[32m"; // Green for primary piece
        final String EXIT_COLOR = "\u001B[35m"; // Purple for exit door
        final String MOVED_COLOR = "\u001B[31m"; // Red for moved piece

        int height = papan.getHeight();
        int width = papan.getWidth();

        boolean isHorizontalExit = pintuKeluar.isHorizontal();
        int exitRow = pintuKeluar.getRow();
        int exitCol = pintuKeluar.getCol();

        // Print top exit if it exists (exitRow == -1)
        if (exitRow == -1) {
            // Print top border with exit
            for (int col = 0; col < width; col++) {
                if (col == exitCol) {
                    System.out.print(EXIT_COLOR + "K" + RESET);
```

```
                } else {
                    System.out.print(" ");
                }
            }
            System.out.println();
        }

        // Print the board rows
        for (int row = 0; row < height; row++) {
            // Print left exit if it exists (exitCol == -1 and current row matches exitRow)
            if (exitCol == -1 && row == exitRow) {
                System.out.print(EXIT_COLOR + "K" + RESET);
            } else if (exitCol == -1) {
                System.out.print(" ");
            }

            // Print board content
            for (int col = 0; col < width; col++) {
                Piece piece = papan.getPiece(row, col);

                if (piece != null) {
                    String pieceId = piece.getId();

                    if (pieceId.equals("P")) {
                        System.out.print(PRIMARY_COLOR + pieceId + RESET);
                    } else if (movedPieceId != null && pieceId.equals(movedPieceId)) {
                        System.out.print(MOVED_COLOR + pieceId + RESET);
                    } else {
                        System.out.print(pieceId);
                    }
                } else {
                    System.out.print(".");
                }
            }

            // Print right exit if it exists (exitCol == width and current row matches
            // exitRow)
            if (exitCol == width && row == exitRow) {
                System.out.print(EXIT_COLOR + "K" + RESET);
            }

            System.out.println();
        }

        // Print bottom exit if it exists (exitRow == height)
        if (exitRow == height) {
            // Print bottom border with exit
            for (int col = 0; col < width; col++) {
                if (col == exitCol) {
                    System.out.print(EXIT_COLOR + "K" + RESET);
                } else {
```

```
                    System.out.print(" ");
                }
            }
            System.out.println();
        }
    }
}
```

BAB 4 Pengujian

1. Uniform Cost Search

Link hasil keseluruhan algoritma UCS : Hasil testing UCS

test > 1.txt

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

```
AAB..F
..BCDF
GPPCDFK
GH.III
GHJ...
LLJMM.

Select pathfinding algorithm:
1. Uniform Cost Search (UCS)
2. A* Search
3. Greedy Best-First Search (GBFS)
Enter your choice (1-3): 1
```

```
WARNING: [DEBUG] Tabrakan dengan piece lain: F
Found solution in 574 iterations.

Execution time: 3683 ms

Do you want to save the solution to a file? (y/n)
y
Enter the output file name: hasil1.txt
Solution saved to file: hasil1.txt
Papan Awal
AAB..F
..BCDF
GPPCDFK
GH.III
GHJ...
LLJMM.

Gerakan 1: I-kiri
AAB..F
..BCDF
GPPCDFK
GHIII.
GHJ...
LLJMM.
```

```
Gerakan 8: P-kanan
AABCD.
..BCD.
G..PP.K
GHIIIF
GHJ..F
LLJMMF

Gerakan 9: P-kanan
Primary piece successfully exited!
AABCD.
..BCD.
G.....K
GHIIIF
GHJ..F
LLJMMF

Solution found with 9 moves.
```

test > 2.txt

```
6 6
10
JJ...I
LL...I
FEGGGD
KFEBPPD
CEB..D
C.B.AA
```

```
INFO: Piece A ditemukan di posisi: 54
 JJ...I
 LL...I
 FEGGGD
KFEBPPD
 CEB..D
 C.B.AA

Select pathfinding algorithm:
1. Uniform Cost Search (UCS)
2. A* Search
3. Greedy Best-First Search (GBFS)
Enter your choice (1-3): 1
```

```
Found solution in 2629 iterations.

Execution time: 2884 ms

Do you want to save the solution to a file? (y/n)
y
Enter the output file name: hasil2.txt
Solution saved to file: hasil2.txt
Papan Awal
 JJ...I
 LL...I
 FEGGGD
KFEBPPD
 CEB..D
 C.B.AA

Gerakan 1: L-kanan
 JJ...I
 .LL..I
 FEGGGD
KFEBPPD
 CEB..D
 C.B.AA
```

<table>
<tr>
<td></td>
<td>

```
  Gerakan 17: P-kiri
   .EBJJI
   FEBLLI
   FEBGGG
  K.PP..D
   C....D
   C..AAD

  Gerakan 18: P-kiri
  Primary piece successfully exited!
   .EBJJI
   FEBLLI
   FEBGGG
  K.....D
   C....D
   C..AAD

  Solution found with 18 moves.
```

</td>
</tr>
<tr>
<td>

test > 3.txt

```
1   6 6
2   6
3     K
4   .....A
5   .....A
6   BBBC.A
7   DDPC..
8   E.PC..
9   EFFF..
10
```

</td>
<td>

```
INFO: Piece F ditemukan di posisi: 51
  K
.....A
.....A
BBBC.A
DDPC..
E.PC..
EFFF..

Select pathfinding algorithm:
1. Uniform Cost Search (UCS)
2. A* Search
3. Greedy Best-First Search (GBFS)
Enter your choice (1-3): 1
```

</td>
</tr>
</table>

```
Found solution in 262 iterations.

Execution time: 319 ms

Do you want to save the solution to a file? (y/n)
y
Enter the output file name: hasil3.txt
Solution saved to file: hasil3.txt
Papan Awal
  K
.....A
.....A
BBBC.A
DDPC..
E.PC..
EFFF..

Gerakan 1: F-kanan
  K
.....A
.....A
BBBC.A
DDPC..
E.PC..
E.FFF.

Gerakan 2: F-kanan
  K
.....A
.....A
BBBC.A
DDPC..
E.PC..
E..FFF
```

```
Gerakan 48: P-atas
  K
E.....
E.P...
..PBBB
DD.C.A
...C.A
FFFC.A

Gerakan 49: P-atas
Primary piece successfully exited!
  K
E.....
E.....
...BBB
DD.C.A
...C.A
FFFC.A

Solution found with 49 moves.
```

(error handling, sengaja error)

```
test > 10.txt
1   6 6
2   5
3   .AAAA.
4   B..PP.K
5   BCC...
6   E.....
7   E.....
8   DDD...
9
```

```
VICTUS 16@LAPTOP-A8M3C9QR MINGW64 ~/OneDrive/Documents/Tucil3_13523125_13523143/src (testing)
$ java Main
Enter the puzzle configuration file name: ../test/10.txt
May 21, 2025 12:24:11 AM InputHandler readConfigFromFile
INFO: Pintu keluar di kanan papan.
May 21, 2025 12:24:11 AM InputHandler readConfigFromFile
INFO: Pintu keluar ditemukan di: 1
May 21, 2025 12:24:11 AM InputHandler readConfigFromFile
INFO: Pintu keluar ditemukan di: 6
May 21, 2025 12:24:11 AM InputHandler readConfigFromFile
INFO: Papan dimensi: 6 x 6
An error occurred: Piece tidak boleh memiliki lebih dari 3 posisi.
java.lang.IllegalStateException: Piece tidak boleh memiliki lebih dari 3 posisi.
        at Piece.addPosition(Piece.java:32)
        at InputHandler.identifyPieceCells(InputHandler.java:308)
        at InputHandler.identifyPieceCells(InputHandler.java:318)
        at InputHandler.identifyPieceCells(InputHandler.java:318)
        at InputHandler.identifyPieceCells(InputHandler.java:318)
        at InputHandler.processPieces(InputHandler.java:298)
        at InputHandler.processBoardConfiguration(InputHandler.java:261)
        at InputHandler.readConfigFromFile(InputHandler.java:168)
        at Main.main(Main.java:17)
```

2. Greedy Best First Search

Link hasil keseluruhan algoritma GBFS : Hasil testing GBFS

```
test > 4.txt
1   6 6
2   13
3   .AAAB.
4   CDDDBE
5   CFFPHE
6   IJJPHL
7   IMNNNL
8   .MOOO.
9     K
```

```
INFO: Piece O ditemukan di posisi: 52
.AAAB.
CDDDBE
CFFPHE
IJJPHL
IMNNNL
.MOOO.
   K

Select pathfinding algorithm:
1. Uniform Cost Search (UCS)
2. A* Search
3. Greedy Best-First Search (GBFS)
Enter your choice (1-3): 3

Select heuristic function:
1. Distance to Exit
2. Blocking Pieces
Enter your choice (1-2): 1
```

```
WARNING: [DEBUG] Gerakan keluar dari papan tidak valid untuk piece selain primary.
Found solution in 349 iterations.

Execution time: 3887 ms

Do you want to save the solution to a file? (y/n)
y
Enter the output file name: ../hasil4.txt
Solution saved to file: ../hasil4.txt
Papan Awal
.AAAB.
CDDDBE
CFFPHE
IJJPHL
IMNNNL
.MOOO.
   K

Gerakan 1: I-bawah
.AAAB.
CDDDBE
CFFPHE
.JJPHL
IMNNNL
IMOOO.
   K

Gerakan 2: E-atas
.AAABE
CDDDBE
CFFPH.
.JJPHL
IMNNNL
IMOOO.
   K
```

```
Gerakan 61: O-kiri
CAAABE
CDDDBE
IM..FF
IM.PJJ
NNNPHL
OOO.HL
   K

Gerakan 62: P-bawah
Primary piece successfully exited!
CAAABE
CDDDBE
IM..FF
IM..JJ
NNN.HL
OOO.HL
   K

Solution found with 62 moves.
```

test > 5.txt

```
6 6
5
......
......
K.PPE..
.DDE.A
.C.E.A
.CBB.A
```

```
INFO: Piece B ditemukan di posisi: 52
 ......
 ......
K.PPE..
 .DDE.A
 .C.E.A
 .CBB.A

Select pathfinding algorithm:
1. Uniform Cost Search (UCS)
2. A* Search
3. Greedy Best-First Search (GBFS)
Enter your choice (1-3): 3

Select heuristic function:
1. Distance to Exit
2. Blocking Pieces
Enter your choice (1-2): 2
```

```
Do you want to save the solution to a file? (y/n)
y
Enter the output file name: ../hasil5.txt
Solution saved to file: ../hasil5.txt
Papan Awal
 ......
 ......
K.PPE..
 .DDE.A
 .C.E.A
 .CBB.A

Gerakan 1: P-kiri
Primary piece successfully exited!
 ......
 ......
K...E..
 .DDE.A
 .C.E.A
 .CBB.A

Solution found with 1 moves.
```

test > 6.txt

```
6 6
9
.AAA..
BBCD..
EECD..
KFPPH..
FGGH..
FIII..
```

```
INFO: Piece I ditemukan di posisi: 51
 .AAA..
 BBCD..
 EECD..
KFPPH..
 FGGH..
 FIII..

Select pathfinding algorithm:
1. Uniform Cost Search (UCS)
2. A* Search
3. Greedy Best-First Search (GBFS)
Enter your choice (1-3): 3

Select heuristic function:
1. Distance to Exit
2. Blocking Pieces
Enter your choice (1-2): 2
```

```
WARNING: [DEBUG] Tabrakan dengan piece lain: H
Found solution in 1532 iterations.

Execution time: 5587 ms

Do you want to save the solution to a file? (y/n)
y
Enter the output file name: ../hasil6.txt
Solution saved to file: ../hasil6.txt
Papan Awal
 .AAA..
 BBCD..
 EECD..
KFPPH..
 FGGH..
 FIII..

Gerakan 1: A-kiri
 AAA...
 BBCD..
 EECD..
KFPPH..
 FGGH..
 FIII..

Gerakan 2: I-kanan
 AAA...
 BBCD..
 EECD..
KFPPH..
 FGGH..
 F.III.
```

```
Gerakan 98: P-kiri
 FAAA..
 F.CDBB
 F.CDEE
K.PP...
 GG.H..
 IIIH..

Gerakan 99: P-kiri
Primary piece successfully exited!
 FAAA..
 F.CDBB
 F.CDEE
K......
 GG.H..
 IIIH..

Solution found with 99 moves.
```

(error handling, sengaja error)

```
test > 11.txt
1   6 6
2   6
3   AAA..B
4   CC...B
5   ..DP.B
6   E.DP..K
7   E..FF.
8   E.....
9
```

```
VICTUS 16@LAPTOP-A8M3C9QR MINGW64 ~/OneDrive/Documents/Tucil3_13523125_13523143/bin (testing)
$ java Main
Enter the puzzle configuration file name: ../test/11.txt
May 21, 2025 12:50:47 AM InputHandler readConfigFromFile
INFO: Pintu keluar di kanan papan.
May 21, 2025 12:50:47 AM InputHandler readConfigFromFile
INFO: Pintu keluar ditemukan di: 3
May 21, 2025 12:50:47 AM InputHandler readConfigFromFile
INFO: Pintu keluar ditemukan di: 6
May 21, 2025 12:50:47 AM InputHandler readConfigFromFile
INFO: Papan dimensi: 6 x 6
May 21, 2025 12:50:47 AM InputHandler processPieces
INFO: Piece A ditemukan di posisi: 00
May 21, 2025 12:50:47 AM InputHandler processPieces
INFO: Piece B ditemukan di posisi: 05
May 21, 2025 12:50:47 AM InputHandler processPieces
INFO: Piece C ditemukan di posisi: 10
May 21, 2025 12:50:47 AM InputHandler processPieces
INFO: Piece D ditemukan di posisi: 22
May 21, 2025 12:50:47 AM InputHandler processPieces
INFO: Piece P ditemukan di posisi: 23
May 21, 2025 12:50:47 AM InputHandler processPieces
INFO: Piece E ditemukan di posisi: 30
May 21, 2025 12:50:47 AM InputHandler processPieces
INFO: Piece F ditemukan di posisi: 43
An error occurred: Primary piece vertikal, pintu keluar harus vertikal.
java.lang.IllegalArgumentException: Primary piece vertikal, pintu keluar harus vertikal.
        at InputHandler.readConfigFromFile(InputHandler.java:201)
        at Main.main(Main.java:17)
```

3. A*

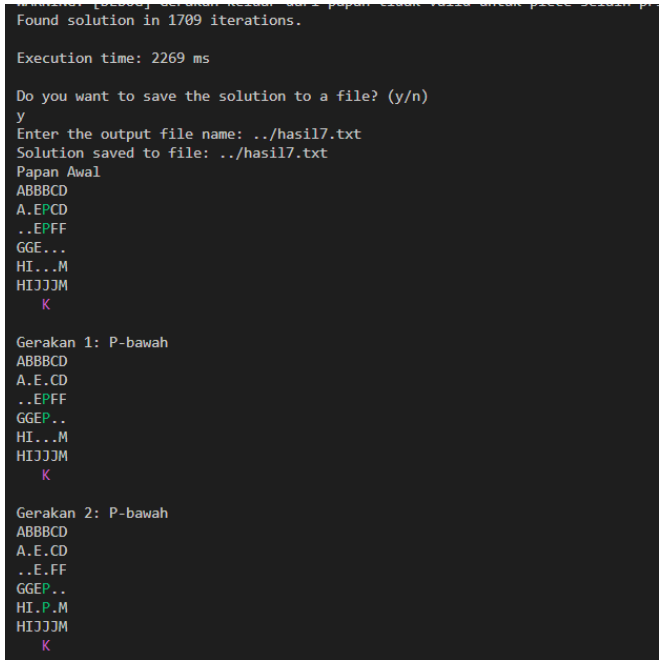
Link hasil keseluruhan testing algoritma A* : Hasil testing A*

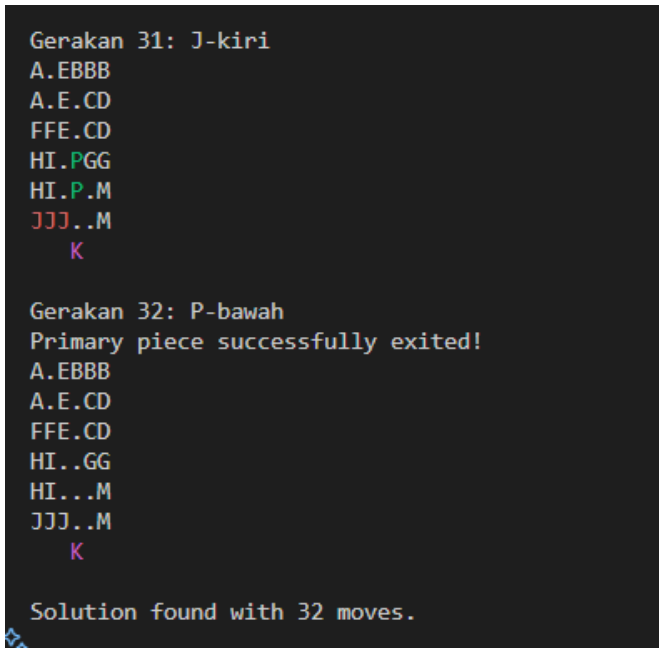
```
test > 7.txt
1   6 6
2   11
3   ABBBCD
4   A.EPCD
5   ..EPFF
6   GGE...
7   HI...M
8   HIJJJM
9      K
10
```

```
INFO: Piece J ditemukan di posisi: 52
ABBBCD
A.EPCD
..EPFF
GGE...
HI...M
HIJJJM
   K

Select pathfinding algorithm:
1. Uniform Cost Search (UCS)
2. A* Search
3. Greedy Best-First Search (GBFS)
Enter your choice (1-3): 2

Select heuristic function:
1. Distance to Exit
2. Blocking Pieces
Enter your choice (1-2): 1
```


Found solution in 1709 iterations.
Execution time: 2269 ms
Do you want to save the solution to a file? (y/n)
y
Enter the output file name: ../hasil7.txt
Solution saved to file: ../hasil7.txt
Papan Awal
ABBBCD
A.EPCD
..EPFF
GGE...
HI...M
HIJJJM
K
Gerakan 1: P-bawah
ABBBCD
A.E.CD
..EPFF
GGEP..
HI...M
HIJJJM
K
Gerakan 2: P-bawah
ABBBCD
A.E.CD
..E.FF
GGEP..
HI.P.M
HIJJJM
K


Gerakan 31: J-kiri
A.EBBB
A.E.CD
FFE.CD
HI.PGG
HI.P.M
JJJ..M
K
Gerakan 32: P-bawah
Primary piece successfully exited!
A.EBBB
A.E.CD
FFE.CD
HI..GG
HI...M
JJJ..M
K
Solution found with 32 moves.

test > 8.txt

```
6 6
5
......
......
.PPA..K
.BBA.C
.D.A.C
.DEE.C

```

```
INFO: Piece E ditemukan di posisi: 52
......
......
.PPA..K
.BBA.C
.D.A.C
.DEE.C

Select pathfinding algorithm:
1. Uniform Cost Search (UCS)
2. A* Search
3. Greedy Best-First Search (GBFS)
Enter your choice (1-3): 2

Select heuristic function:
1. Distance to Exit
2. Blocking Pieces
Enter your choice (1-2): 2
```

```
Found solution in 799 iterations.

Execution time: 680 ms

Do you want to save the solution to a file? (y/n)
y
Enter the output file name: ../hasil8.txt
Solution saved to file: ../hasil8.txt
Papan Awal
......
......
.PPA..K
.BBA.C
.D.A.C
.DEE.C

Gerakan 1: A-atas
......
...A..
.PPA..K
.BBA.C
.D...C
.DEE.C

Gerakan 2: C-atas
......
...A..
.PPA.CK
.BBA.C
.D...C
.DEE..
```

```
Gerakan 32: P-kanan
.D....
.D....
...PP.K
.BBA.C
...A.C
.EEA.C

Gerakan 33: P-kanan
Primary piece successfully exited!
.D....
.D....
......K
.BBA.C
...A.C
.EEA.C

Solution found with 33 moves.
```

test > 9.txt

```
6 6
7
  K
...ABB
...A.C
DDDA.C
E.PFFC
E.P...
GGG...

```

```
INFO: Piece G ditemukan di posisi: 50
  K
...ABB
...A.C
DDDA.C
E.PFFC
E.P...
GGG...

Select pathfinding algorithm:
1. Uniform Cost Search (UCS)
2. A* Search
3. Greedy Best-First Search (GBFS)
Enter your choice (1-3): 2

Select heuristic function:
1. Distance to Exit
2. Blocking Pieces
Enter your choice (1-2): 1
```

```
Found solution in 829 iterations.

Execution time: 895 ms

Do you want to save the solution to a file? (y/n)
y
Enter the output file name: ../hasil9.txt
Solution saved to file: ../hasil9.txt
Papan Awal
  K
...ABB
...A.C
DDDA.C
E.PFFC
E.P...
GGG...

Gerakan 1: G-kanan
  K
...ABB
...A.C
DDDA.C
E.PFFC
E.P...
.GGG..

Gerakan 2: G-kanan
  K
...ABB
...A.C
DDDA.C
E.PFFC
E.P...
..GGG.
```

```
Gerakan 55: B-kanan
  K
E..BB.
E.P...
..PDDD
FF.A.C
...A.C
GGGA.C

Gerakan 56: P-atas
Primary piece successfully exited!
  K
E..BB.
E.....
...DDD
FF.A.C
...A.C
GGGA.C

Solution found with 56 moves.
```

(error handling, sengaja error)

test > 12.txt

```
1  6 6
2  5
3  BBB..K
4  C.AA..
5  C....P
6  C.D..P
7  E.D..P
8  E.D...
9
```

```
VICTUS 16@LAPTOP-A8M3C9QR MINGW64 ~/OneDrive/Documents/Tucil3_13523125_13523143/bin (testing)
$ java Main
Enter the puzzle configuration file name: ../test/12.txt
May 21, 2025 1:41:11 AM InputHandler readConfigFromFile
INFO: Pintu keluar ditemukan di: 0
May 21, 2025 1:41:11 AM InputHandler readConfigFromFile
INFO: Pintu keluar ditemukan di: 5
May 21, 2025 1:41:11 AM InputHandler readConfigFromFile
INFO: Papan dimensi: 6 x 6
May 21, 2025 1:41:11 AM InputHandler processPieces
INFO: Piece B ditemukan di posisi: 00
May 21, 2025 1:41:11 AM InputHandler processPieces
INFO: Piece C ditemukan di posisi: 10
May 21, 2025 1:41:11 AM InputHandler processPieces
INFO: Piece A ditemukan di posisi: 12
May 21, 2025 1:41:11 AM InputHandler processPieces
INFO: Piece P ditemukan di posisi: 25
May 21, 2025 1:41:11 AM InputHandler processPieces
INFO: Piece D ditemukan di posisi: 32
May 21, 2025 1:41:11 AM InputHandler processPieces
INFO: Piece E ditemukan di posisi: 40
An error occurred: Primary piece vertikal, pintu keluar harus vertikal.
java.lang.IllegalArgumentException: Primary piece vertikal, pintu keluar harus vertikal.
        at InputHandler.readConfigFromFile(InputHandler.java:220)
        at Main.main(Main.java:17)
```

BAB 5 Analisis Percobaan Algoritma Pathfinding

1. Analisis Percobaan Algoritma Greedy Best First Search (GBFS)

Dalam percobaan ini, GBFS selalu menunjukkan waktu eksekusi tercepat dan jumlah eksplorasi state terendah dibandingkan algoritma lainnya. Waktu eksekusi cepat karena GBFS hanya fokus pada node dengan nilai heuristik terbaik tanpa mempertimbangkan biaya jalur yang sudah ditempuh.

Walaupun waktu eksekusi cepat, GBFS tidak menghasilkan solusi yang optimal karena hanya mempertimbangkan nilai heuristik tanpa memperhatikan biaya perjalanan g(n). Hal ini membuat algoritma ini menghasilkan solusi yang lebih panjang dari solusi yang optimal. Pengujian dua jenis heuristik menunjukkan jumlah iterasi yang sama, dan waktu eksekusi yang relatif mirip.

2. Analisis Algoritma Uniform Cost Search (UCS)

Percobaan ini menunjukkan bahwa UCS selalu menghasilkan solusi optimal, namun dengan eksplorasi state paling banyak dan waktu eksekusi yang lebih lama dibandingkan GBFS. Jumlah eksplorasi yang besar ini disebabkan karena UCS mencari semua kemungkinan jalur dengan biaya sama atau lebih rendah tanpa penggunaan heuristik.

Algoritma ini hanya mengevaluasi state berdasarkan jumlah g(n) tanpa menggunakan heuristik sehingga efektif untuk menemukan solusi optimal namun kurang efisien dalam waktu eksekusi dan iterasi. Kelebihan UCS adalah pada struktur priority queue yang memastikan pemeriksaan menyeluruh pada semua kemungkinan jalur dengan biaya rendah terlebih dahulu.

3. Analisis Algoritma A*

Percobaan ini menunjukkan bahwa A* selalu menghasilkan solusi optimal seperti UCS, namun dengan jumlah eksplorasi state yang lebih sedikit dan waktu eksekusi yang lebih efisien. Efisiensi ini diperoleh karena A* menggunakan kombinasi antara biaya perjalanan (g(n)) dan penggunaan heuristik (h(n)) dalam fungsi penilaian f(n) sehingga mampu menyeimbangkan antara jalur biaya rendah dan arah menuju solusi.

A* dengan dua jenis heuristik yang diuji memperlihatkan perbedaan yang signifikan. Heuristik 1 (Jarak ke Pintu Keluar) cenderung mengeksplorasi lebih sedikit state namun memiliki waktu eksekusi lebih lama, sedangkan Heuristik 2 (Jumlah Piece Penghalang) menunjukkan waktu eksekusi lebih baik meskipun dengan jumlah eksplorasi yang sedikit lebih besar. Hal ini disebabkan karena Heuristik 1 bersifat umum dan hanya mempertimbangkan jarak, tidak memperhitungkan apakah jalan tersebut bisa dilalui atau terhalang sedangkan Heuristik 2

memberikan estimasi yang lebih spesifik karena langsung memperhitungkan hambatan ke arah jalur keluar.

4. Perbandingan Ketiga Algoritma

Berdasarkan hasil analisis, secara teori, pengembangan algoritma sesuai. GBFS memiliki waktu eksekusi tercepat dan eksplorasi state terendah namun menghasilkan solusi tidak optimal. UCS selalu menghasilkan solusi optimal tetapi membutuhkan eksplorasi terbanyak dengan waktu eksekusi lebih lama dari GBFS. A* menghasilkan eksplorasi state yang lebih sedikit dibanding UCS, solusi yang optimal, dan waktu eksekusi yang berbeda tergantung heuristik yang digunakan. Perbedaan performa ini dipengaruhi langsung oleh perbedaan fokus evaluasi, GBFS hanya menggunakan h(n), UCS hanya menggunakan g(n), sedangkan A* menggunakan f(n) = g(n) + h(n).

Lampiran

Link github : https://github.com/DitaMaheswari05/Tucil3_13523125_13523143.git

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	✔	

DAFTAR PUSTAKA

IF2211 Strategi Algoritma - Semester II Tahun 2023/2024
https://www.geeksforgeeks.org/greedy-best-first-search-in-ai/
http://etheses.uin-malang.ac.id/59496/6/18650122.pdf
https://www.trivusi.web.id/2023/01/algoritma-a-star.html