IF2211 - Strategi Algoritma

Tugas Kecil 3

Pemecah Permainan Puzzle Rush Hour dengan Berbagai Algoritma Pathfinding

Disusun Oleh:

Nadhif Al Rozin	13523076
David Bakti Lodianto	13523083

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

2025

DAFTAR ISI

BAB 1
LANDASAN TEORI

1. Penjelasan Algoritma UCS

Algoritma UCS (Uniform Cost Search) adalah algoritma pencarian uninformed (tidak mengetahui jarak ke tujuan) yang memilih node yang akan dikunjungi dengan mencari cost / biaya kumulatif paling rendah dari titik awal. Dilakukan dengan menyiapkan suatu state (kondisi board) yang berisi isi board, posisi exit, biaya kumulatif saat state ini dan mobil mobil di dalamnya.

a. Langkah pertama adalah dengan menilai apa state itu sudah menjadi tujuan akhir, jika tidak dilanjutkan ke langkah ke 2.
b. Mencari semua kemungkinan gerakan yang dilakukan dan dimasukkan ke pool pilihan
c. Memasuki state dengan biaya kumulatif terkecil, ulangi dari langkah 1 hingga pool pilihan habis.
d. Jika habis, tidak ada solusi yang mungkin.

2. Penjelasan Algoritma Greedy Best First Search

Algoritma Greedy Best First Search adalah algoritma pencarian informed (mengetahui jarak ke tujuan) yang menggunakan fungsi heuristik untuk menimbang jarak ke tujuan dari state saat ini. Algoritma akan memilih node yang dikunjungi selanjutnya berdasarkan jarak terpendek menuju tujuan yang diperkirakan lewat heuristik. Sangat tergantung dengan seberapa akurat fungsi heuristik yang digunakan. Dilakukan dengan menyiapkan suatu state (kondisi board) yang berisi isi board, posisi exit, fungsi heuristik dan mobil mobil di dalamnya.

a. Langkah pertama adalah dengan menilai apa state itu sudah menjadi tujuan akhir, jika tidak dilanjutkan ke langkah ke 2.
b. Mencari semua kemungkinan gerakan yang dilakukan dan dimasukkan ke pool pilihan
c. Memasuki state dengan nilai heuristik terkecil, ulangi dari langkah 1 hingga pool pilihan habis.
d. Jika habis, tidak ada solusi yang mungkin.

3. **Penjelasan Algoritma A***

Algoritma A* adalah algoritma pencarian yang merupakan kombinasi dari UCS dan GBFS. Termasuk algoritma pencarian informed (mengetahui jarak ke tujuan) yang menggunakan fungsi heuristik untuk menimbang jarak ke tujuan dari state saat ini dan mempertimbangkan juga biaya kumulatif dari titik awal. Algoritma akan memilih node yang dikunjungi selanjutnya berdasarkan hasil terendah dari g(n) + h(n). Dengan g(n) adalah biaya kumulatif dan h(n) adalah fungsi heuristik. Dilakukan dengan menyiapkan suatu state (kondisi board) yang berisi isi board, posisi exit, fungsi heuristik, biaya kumulatif saat state ini dan mobil mobil di dalamnya.

a. Langkah pertama adalah dengan menilai apa state itu sudah menjadi tujuan akhir, jika tidak dilanjutkan ke langkah ke 2.
b. Mencari semua kemungkinan gerakan yang dilakukan dan dimasukkan ke pool pilihan
c. Memasuki state dengan nilai g(n) + h(n) terkecil, ulangi dari langkah 1 hingga pool pilihan habis.
d. Jika habis, tidak ada solusi yang mungkin.

4. **Penjelasan Algoritma Simulated Annealing**

Algoritma Simulated Annealing adalah algoritma pencarian yang menggunakan fungsi heuristik. Algoritma ini terinspirasi dari proses pendinginan logam (annealing), dimana logam dipanaskan dan didinginkan perlahan untuk mencari kondisi yang stabil seiring menurunnya suhu. Dilakukan dengan menyiapkan suatu state (kondisi board) yang berisi isi board, posisi exit, fungsi heuristik, mobil mobil di dalamnya.

a. “Memanaskan logam”, memberikan suhu awal
b. Langkah selanjutnya adalah dengan menilai apa state itu sudah menjadi tujuan akhir, jika tidak maka lanjutkan.
c. Mencari semua kemungkinan gerakan yang bisa dilakukan dan mengambil 1 secara random
d. Jika random lebih rendah maka random akan menjadi state saat ini

e. Jika heuristik saat ini lebih rendah, ada kemungkinan yang berskala dengan selisih dan temperatur saat ini. Yang membuat semakin kecil skala dan semakin tinggi temperatur semakin tinggi kemungkinan untuk memilih state yang “salah”
f. Pilihan apapun pada d) dan e) akan mengurangi temperatur.
g. Jika temperatur di atas limit tertentu lanjutkan ke langkah b)

BAB 2
ANALISIS TEORETIS ALGORITMA PATHFINDING

1. **Analisis algoritma UCS**
    a. f(n) pada algoritma UCS adalah g(n) dengan g(n) adalah jumlah step / mobil digerakkan (tidak membedakan jenis dan seberapa jauh)
    b. Pada kasus rush hour, algoritma UCS memiliki perilaku yang sama dengan algoritma BFS. Hal ini dikarenakan cost yang digunakan untuk menggerakkan suatu mobil, tidak peduli jenisnya apa, seberapa panjang mobil itu dan seberapa jauh mobil digerakkan tetaplah 1. Oleh karena itu, tiap "anak" state dari suatu state saat ini akan selalu memiliki cost lebih tinggi 1. Yang dapat dikatakan jika Priority Queue akan selalu mengambil state pada satu level / kedalaman terlebih dahulu, sebelum masuk ke kedalaman selanjutnya.
2. **Analisis algoritma Greedy Best First Search**
    a. f(n) pada algoritma Greedy best first search adalah h(n), dengan h(n) adalah fungsi heuristik yang dibuat untuk memperkirakan jarak state saat ini ke tujuan
    b. Algoritma GBFS secara teoritis tidak dapat menjamin solusi yang optimal. Karena solusi dengan langkah yang cukup jauh tetap dapat terlihat lebih rendah dari segi heuristik. Dan karena GBFS tidak mempertimbangkannya hasil yang didapat tidak dijamin optimal
3. **Analisis algoritma A***
    a. f(n) pada algoritma A* adalah g(n) + h(n), dengan g(n) adalah jumlah step / mobil digerakkan (tidak membedakan jenis dan seberapa jauh) dan h(n) adalah fungsi heuristik yang dibuat untuk memperkirakan jarak state saat ini ke tujuan.
    b. A* lebih efisien daripada UCS jika heuristik yang digunakan admissible dan informatif sehingga mengurangi jumlah node yang di ekspansi secara signifikan, jika heuristik buruk (tidak informatif) seperti semua h(n) = 0, maka akan berperilaku sama dengan UCS.
4. **Analisis algoritma Simulated Annealing**
    a. f(n) pada Simulated Annealing adalah h(n), dengan h(n) adalah fungsi heuristik yang dibuat untuk memperkirakan jarak state saat ini ke tujuan

b. Algoritma ini bersifat stokastik, artinya hasil bisa berubah ubah setiap kali eksekusi. Algoritma tidak menjamin solusi optimal eksak, namun cocok untuk ruang solusi yang besar dan tidak terstruktur

BAB 3

IMPLEMENTASI

Aplikasi ini diimplementasikan menggunakan bahasa pemrograman Java. Berikut kelas-kelas yang dibuat untuk menyelesaikan pemecahan puzzle.

1. Board

```java
// Board.java

package com.tucil3.backend.lib;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

public class Board {
    private final char[][] board;
    private final int width;
    private final int height;
    private final int totalCost;
    private Board parent;
    private Coor exit;
    private final List<Car> cars;

    public Board(char[][] board, Coor exit) {
        this.board = deepCopyBoard(board);
        this.width = board[0].length;
        this.height = board.length;
        this.exit = exit;
        this.cars = new ArrayList<>();
        // scan through horizontal cars
        for (int i = 0; i < height; i++) {
            for (int j = 0; j < width; j++) {
                if (board[i][j] == '.') {
                    continue;
                }
                char current = board[i][j];
                int startX = j;
```

```
                do {
                    j++;
                } while (j < height && board[i][j] == current);
                if (j - startX > 1) {
                     cars.add(new Car(current, new Coor(i, startX), new Coor(i,
j - 1)));
                }
                j--;
            }
        }

        for (int j = 0; j < width; j++) {
            for (int i = 0; i < height; i++) {
                if (board[i][j] == '.') {
                    continue;
                }
                char current = board[i][j];
                int startY = i;
                do {
                    i++;
                } while (i < height && board[i][j] == current);
                if (i - startY > 1) {
                    cars.add(new Car(current, new Coor(startY, j), new Coor(i -
1, j)));
                }
                i--;
            }
        }

        this.totalCost = 0;
        this.parent = null;
    }

    public Coor getExit() {
        return exit;
    }

    public void setParent(Board par) {
```

```
        this.parent = par;
    }


    public void setExit(Coor ex) {
        this.exit = ex;
    }

    public Board(char[][] board, int totalCost, Board parent, List<Car> cars) {
        this.board = deepCopyBoard(board);
        this.width = board[0].length;
        this.height = board.length;
        this.totalCost = totalCost;
        this.parent = parent;
        this.exit = parent.exit;
        this.cars = deepCopyCars(cars);
    }

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

    public int getWidth() {
        return width;
    }

    public int getHeight() {
        return height;
    }

    public List<Car> getCars() {
        return deepCopyCars(cars);
    }

    public int getCost() {
        return totalCost;
    }
```

```java
    public Board getParent() {
        return parent;
    }

    public boolean isGoal() {
        return board[exit.Y][exit.X] == 'P';
    }

    private char[][] deepCopyBoard(char[][] original) {
        char[][] copy = new char[original.length][];
        for (int i = 0; i < original.length; i++) {
            copy[i] = original[i].clone();
        }
        return copy;
    }

    private List<Car> deepCopyCars(List<Car> original) {
        List<Car> copy = new ArrayList<>();
        for (Car car : original) {
            copy.add(new Car(
                car.getSymbol(),
                car.getStart().copy(),
                car.getEnd().copy()
            ));
        }
        return copy;
    }

    public int heuristik() {
        Car primary = getCarBySymbol('P');

        if (primary == null) return Integer.MAX_VALUE;

        int blocking = 0;

        if (primary.getDirection() == 'X') {
            int y = primary.getStart().Y;
            int x = primary.getEnd().X + 1;
```

```
            while (x < width) {
                if (board[y][x] != '.') {
                    blocking++;
                }
                if (x == exit.X && y == exit.Y) break;
                x++;
            }

            return (exit.X - primary.getEnd().X) + blocking;

        } else { // direction == 'Y'
            int x = primary.getStart().X;
            int y = primary.getEnd().Y + 1;

            while (y < height) {
                if (board[y][x] != '.') {
                    blocking++;
                }
                if (x == exit.X && y == exit.Y) break;
                y++;
            }

            return (exit.Y - primary.getEnd().Y) + blocking;
        }
    }

    public int heuristik_2() {
        Car primary = getCarBySymbol('P');

        if (primary == null) return Integer.MAX_VALUE;

        int penalty = 0;

        if (primary.getDirection() == 'X') {
            int y = primary.getStart().Y;
            int x = primary.getEnd().X + 1;
```

```
            while (x < width) {
                char cell = board[y][x];
                if (cell != '.' && cell != primary.getSymbol()) {
                    // System.out.println(cell);
                    Car blocker = getCarBySymbol(cell);

                    if (canMove(blocker, 1) || canMove(blocker, -1)) {
                        penalty -= 1;
                    } else {
                        penalty -= 3;
                    }
                }

                if (x == exit.X && y == exit.Y) break;
                x++;
            }

        } else {
            int x = primary.getStart().X;
            int y = primary.getEnd().Y + 1;

            while (y < height) {
                char cell = board[y][x];
                if (cell != '.' && cell != primary.getSymbol()) {
                    Car blocker = getCarBySymbol(cell);

                    if (canMove(blocker, 1) || canMove(blocker, -1)) {
                        penalty -= 1;
                    } else {
                        penalty -= 3;
                    }
                }

                if (x == exit.X && y == exit.Y) break;
                y++;
            }
        }
```

```
        int distance = (primary.getDirection() == 'X') ?
                (exit.X - primary.getEnd().X) :
                (exit.Y - primary.getEnd().Y);

        return distance + Math.abs(penalty);
    }

    private Car getCarBySymbol(char symbol) {
        for (Car car : cars) {
            // System.out.println(car.getSymbol());
            if (car.getSymbol() == symbol) {
                return car;
            }
        }
        return null;
    }

    public int heuristik_3() {
        Car primary = getCarBySymbol('P');
        if (primary == null) return Integer.MAX_VALUE;

        Set<Character> level1Blockers = new HashSet<>();
        Set<Character> level2Blockers = new HashSet<>();

        int y = primary.getStart().Y;
        int x = primary.getEnd().X + 1;
        if (primary.getDirection() == 'Y') {
            x = primary.getStart().X;
            y = primary.getEnd().Y + 1;
        }

        while (true) {
            if (x >= width || y >= height) break;
            char cell = board[y][x];
            if (cell != '.' && cell != primary.getSymbol()) {
                level1Blockers.add(cell);

                Car blocker = getCarBySymbol(cell);
```

```
                if (blocker != null) {
                    List<Coor> path = getPath(blocker, 1);
                    path.addAll(getPath(blocker, -1));

                    for (Coor p : path) {
                        // p.debugCoor();
                        char c = board[p.Y][p.X];
                        // System.out.println(c);
                        if (c != '.' && c != blocker.getSymbol()) {
                            if (c == 'P') {return Integer.MAX_VALUE;}
                            level2Blockers.add(c);
                        }
                    }
                }
            }

            if (x == exit.X && y == exit.Y) break;

            if (primary.getDirection() == 'X') {
                x++;
            } else {
                y++;
            }
        }

        System.out.println("Level 1 blockers: " + level1Blockers);
        System.out.println("Level 2 blockers: " + level2Blockers);

        int distance = (primary.getDirection() == 'X') ?
            (exit.X - primary.getEnd().X) :
            (exit.Y - primary.getEnd().Y);

        // Example scoring: level 1 blockers weighted more heavily than level 2
blockers
        return distance + level1Blockers.size() * 3 + level2Blockers.size();
    }

    public List<Coor> getPath(Car car, int step) {
```

```
        List<Coor> path = new ArrayList<>();
        Coor cursor = step > 0 ? car.getEnd() : car.getStart();
        char direction = car.getDirection();

        while (true) {
            int x = cursor.X + (direction == 'X' ? step : 0);
            int y = cursor.Y + (direction == 'Y' ? step : 0);

            if (x < 0 || y < 0 || x >= width || y >= height) break;

            char cell = board[y][x];

            if (cell != '.' && cell != car.getSymbol()) {
                path.add(new Coor(y, x));
                break;
            }

            path.add(new Coor(y, x));
            cursor = new Coor(y, x);
        }

        return path;
    }

    public List<Board> generateChild() {
        List<Board> children = new ArrayList<>();
        int border = Math.max(width, height);

        for (Car car : cars) {
            // car.debugCar();
            int maxStep = border - car.length();
            // System.out.println(maxStep);

            for (int step = -maxStep; step <= maxStep; step++) {
                if (step == 0) continue; // Skip no movement

                if (!canMove(car, step)) continue;
```

```
                char[][] newBoard = this.getBoard();
                List<Car> newCars = this.getCars();

                Car movedCar = null;
                Car oldCar = null;

                for (int i = 0; i < newCars.size(); i++) {
                    if (newCars.get(i).getSymbol() == car.getSymbol()) {
                        oldCar = newCars.get(i);
                                        movedCar = new Car(oldCar.getSymbol(),
oldCar.getStart().copy(), oldCar.getEnd().copy());
                        movedCar.move(step);
                        newCars.set(i, movedCar);
                        break;
                    }
                }

                updateBoard(newBoard, oldCar, movedCar);

                   Board childBoard = new Board(newBoard, totalCost + 1, this,
newCars);
                children.add(childBoard);
            }
        }

        return children;
    }

    private void updateBoard(char[][] board, Car oldCar, Car newCar) {
        // Clear old car
        if (oldCar == null) return;
        if (oldCar.getDirection() == 'X') {
            int y = oldCar.getStart().Y;
            for (int x = oldCar.getStart().X; x <= oldCar.getEnd().X; x++) {
                board[y][x] = '.';
            }
        } else {
            int x = oldCar.getStart().X;
```

```
            for (int y = oldCar.getStart().Y; y <= oldCar.getEnd().Y; y++) {
                board[y][x] = '.';
            }
        }

        // Draw new car
        if (newCar.getDirection() == 'X') {
            int y = newCar.getStart().Y;
            for (int x = newCar.getStart().X; x <= newCar.getEnd().X; x++) {
                board[y][x] = newCar.getSymbol();
            }
        } else {
            int x = newCar.getStart().X;
            for (int y = newCar.getStart().Y; y <= newCar.getEnd().Y; y++) {
                // newCar.getStart().debugCoor();
                // newCar.getEnd().debugCoor();
                board[y][x] = newCar.getSymbol();
            }
        }
    }

    public boolean canMove(Car car, int step) {
        Coor start = car.getStart();
        Coor end = car.getEnd();
        char symbol = car.getSymbol();
        char direction = car.getDirection();

        if (direction == 'X') {
            int newStart = start.X + step;
            int newEnd = end.X + step;

            if (newStart < 0 || newEnd >= width) return false;

            int y = start.Y;
            int max = Math.max(newEnd, end.X);
            int min = Math.min(newStart, start.X);

            for (int i = min; i <= max; i++) {
```

```
                if (board[y][i] != '.' && board[y][i] != symbol) return false;
            }
            return true;

        } else if (direction == 'Y') {
            int newStart = start.Y + step;
            int newEnd = end.Y + step;

            if (newStart < 0 || newEnd >= height) return false;

            int x = start.X;
            int max = Math.max(newEnd, end.Y);
            int min = Math.min(newStart, start.Y);

            for (int i = min; i <= max; i++) {
                if (board[i][x] != '.' && board[i][x] != symbol) return false;
            }
            return true;
        }

        return false; // invalid direction
    }

    public boolean equals(Board b) {
        if (b == null) return false;

        for (int i = 0; i < height; i++) {
            if (!new String(this.board[i]).equals(new String(b.board[i]))) {
                return false;
            }
        }

        return this.totalCost <= b.totalCost;
    }

    public Board withoutPrimary() {
        char[][] newBoard = deepCopyBoard(board);
        List<Car> newCars = deepCopyCars(cars);
```

```
        Car pr = newCars.get(0);
        for (Car car : newCars) {
            if (car.getSymbol() == 'P') {
                pr = car;
            }
        }
        if (pr.getDirection() == 'X') {
            int y = pr.getStart().Y;
            for (int x = pr.getStart().X; x <= pr.getEnd().X; x++) {
                newBoard[y][x] = '.';
            }
        } else {
            int x = pr.getStart().X;
            for (int y = pr.getStart().Y; y <= pr.getEnd().Y; y++) {
                newBoard[y][x] = '.';
            }
        }
        newCars.remove(pr);
        return new Board(newBoard, totalCost, this, newCars);
    }

    public void debugBoard() {
        System.out.println("Board:");
        for (int i = 0; i < height; i++) {
            for (int j = 0; j < width; j++) {
                System.out.print(board[i][j] + " ");
            }
            System.out.println();
        }
        System.out.println("Cost: " + totalCost);
        System.out.println("Heuristik 1: " + heuristik());
        System.out.println("Heuristik 2: " + heuristik_2());
        System.out.println("Heuristik 3: " + heuristik_3());
        System.out.println("Exit: (" + exit.X + ", " + exit.Y + ")");
    }
}
```

2. Coordinate

```java
// Coor.java

package com.tucil3.backend.lib;
public class Coor {
    public int X;
    public int Y;

    public Coor(int Y, int X) {
        this.X = X;
        this.Y = Y;
    }

    public Coor copy() {
        return new Coor(Y, X);
    }

    public boolean equal(Coor c) {
        return this.X == c.X && this.Y == c.Y;
    }

    public void debugCoor() {
        System.out.println("Coord: (" + Y + ", " + X + ")");
    }
}
```

3. Car

```java
// Car.java

package com.tucil3.backend.lib;
public class Car {
    private final char symbol;
    private Coor start;
    private Coor end;
    private final char direction;
```

```
public Car(char symbol, Coor start, Coor end) {
    this.symbol = symbol;
    this.start = start;
    this.end = end;
    if (start.X == end.X) {
        direction = 'Y';
    } else {
        direction = 'X';
    }
}

public int length() {
    if (direction == 'Y') {
        return end.Y - start.Y;
    } else {
        return end.X - start.X;
    }
}

public Coor getStart() {
    return start;
}

public Coor getEnd() {
    return end;
}

public char getDirection() {
    return direction;
}

public char getSymbol() {
    return symbol;
}

public void move(int step) {
    if (direction == 'X') {
```

```
            start.X += step;
            end.X += step;
        } else {
            start.Y += step;
            end.Y += step;
        }
    }

    public void debugCar() {
        System.out.println("Car: " + symbol);
        start.debugCoor();
        end.debugCoor();
        System.out.println("Direction: " + direction);
    }
}
```

4. Solver

```
// Solver.java
package com.tucil3.backend.lib;
import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;
import java.util.PriorityQueue;
import java.util.Random;

public class Solver {
    private final Board initialBoard;
    private List<Board> visitedBoard;
    private final Random random = new Random();
    public int heuristik = 1;
    public int moveCount = 0;
    public int executionTime = 0;


    public Solver(Board initialBoard) {
        this.initialBoard = initialBoard;
```

```
        this.visitedBoard = new ArrayList<>();
    }

    public List<Board> solving(String method, int heur) {
        long startTime = System.nanoTime();

        if (!validSetup()) {
            System.out.println("Invalid setup.");
            return null;
        }
        heuristik = heur;

        if (method.equalsIgnoreCase("SA")) {
            List<Board> result = simulatedAnnealing();
            executionTime = (int)((System.nanoTime() - startTime) / 1_000_000);
            System.out.println("Time taken: " + executionTime + " ms");
            return result;
        }

        PriorityQueue<Board> q;
        if (method.equalsIgnoreCase("UCS")) {
            q = new PriorityQueue<>(Comparator.comparingInt(Board::getCost));
        } else if (method.equalsIgnoreCase("A*")) {
            switch (heuristik) {
                 case 1 -> q = new PriorityQueue<>(Comparator.comparingInt(b ->
b.getCost() + b.heuristik()));
                 case 2 -> q = new PriorityQueue<>(Comparator.comparingInt(b ->
b.getCost() + b.heuristik_2()));
                 case 3 -> q = new PriorityQueue<>(Comparator.comparingInt(b ->
b.getCost() + b.heuristik_3()));
                default -> {
                    System.out.println("Unknown heuristic for A*.");
                    return null;
                }
            }
        } else if (method.equalsIgnoreCase("GBFS")) {
            switch (heuristik) {
                                                case 1 -> q = new
```

```
PriorityQueue<>(Comparator.comparingInt(Board::heuristik));
                                                    case   2   ->   q   =   new
PriorityQueue<>(Comparator.comparingInt(Board::heuristik_2));
                                                    case   3   ->   q   =   new
PriorityQueue<>(Comparator.comparingInt(Board::heuristik_3));
                default -> {
                    System.out.println("Unknown heuristic for GBFS.");
                    return null;
                }
            }
        } else {
            System.out.println("Unknown method: " + method);
            return null;
        }

        q.add(initialBoard);

        while (!q.isEmpty()) {
            moveCount++;
            Board current = q.poll();

            if (isVisited(current)) continue;
            visitedBoard.add(current);

            // current.debugBoard();

            if (current.isGoal()) {
                System.out.println("Solution found!");
                System.out.println("Board visited: " + moveCount);

                      executionTime = (int)((System.nanoTime() - startTime) /
1_000_000);
                System.out.println("Time taken: " + executionTime + " ms");
                return buildPath(current);
            }

            for (Board child : current.generateChild()) {
                if (!isVisited(child)) {
```

```
                    q.add(child);
                }
            }
        }

        System.out.println("No solution found.");
        System.out.println("Board visited: " + moveCount);

        executionTime = (int)((System.nanoTime() - startTime) / 1_000_000);
        System.out.println("Time taken: " + executionTime + " ms");
        return null;
    }

    private boolean isVisited(Board curr) {
        for (Board b : visitedBoard) {
            if (b.equals(curr)) {
                return true;
            }
        }
        return false;
    }

    public boolean validSetup() {
        Coor ex = initialBoard.getExit();
        Coor pr = null;
        char direction = ' ';
        for (Car car: initialBoard.getCars()) {
//            car.debugCar();
            if (car.getSymbol() == 'P') {
                pr = car.getStart();
                direction = car.getDirection();
                break;
            }
        }
        if (pr == null) {
            return false;
        }
        if (direction == 'X') {
```

```
            if (!(ex.X == -1 || ex.X == initialBoard.getWidth())) {
                return false;
            }
            if (!(ex.Y == pr.Y)) {
                return false;
            }
            ex.X += (ex.X == -1) ? 1 : -1;
            initialBoard.setExit(ex);
            return true;
        } else {
            if (!(ex.Y == -1 || ex.Y == initialBoard.getWidth())) {
                return false;
            }
            if (!(ex.X == pr.X)) {
                return false;
            }
            ex.Y += (ex.Y == -1) ? 1 : -1;
            initialBoard.setExit(ex);
            return true;
        }
    }

    private List<Board> buildPath(Board current) {
        List<Board> path = new ArrayList<>();
        while (current != null) {
            path.add(0, current);
            current = current.getParent();
        }
        path.add(path.get(path.size() - 1).withoutPrimary());
        return path;
    }

    private List<Board> simulatedAnnealing() {
        moveCount = 0;

        Board current = initialBoard;
        double temperature = 1000.0;
        double coolingRate = 0.003;
```

```
        while (temperature > 1e-9) {
            moveCount++;
            // current.debugBoard();
            if (!isVisited(current)) {
                visitedBoard.add(current);
            }
            if (current.isGoal()) {
                System.out.println("Solution found!");
                System.out.println("Board visited: " + moveCount);
                return buildPath(current);
            }

            List<Board> children = current.generateChild();
            if (children.isEmpty()) break;

            Board next = null;
            while (!children.isEmpty()) {
                next = randomChild(children);
                children.remove(next);

                if (!isVisited(next) && next.getCost() <= 900) {
                    break; // valid candidate
                }

                next = null; // reset so we can check later
            }

            // If no valid child was found, backtrack
            if (next == null) {
                if (current.getParent() == null) {
                    return null;
                }
                current = current.getParent();
                continue;
            }
            int delta;
            switch (heuristik) {
```

```
                case 1 -> delta = current.heuristik() - next.heuristik();
                case 2 -> delta = current.heuristik() - next.heuristik_2();
                case 3 -> delta = current.heuristik() - next.heuristik_3();
                default -> {
                    System.out.println("Unknown heuristic for SA.");
                    return null;
                }
            }

            if (delta > 0) {
                current = next;
            }
            else {
                double probability = Math.exp(delta / temperature);
                if (random.nextDouble() < probability) {
                    current = next;
                }
            }

            temperature *= (1 - coolingRate); // cooling
        }

        System.out.println("No solution found.");
        System.out.println("Board visited: " + moveCount);
        return null;
    }

    private Board randomChild(List<Board> children) {
        if (children.isEmpty()) {return null;}
        return children.get(random.nextInt(children.size()));
    }
}
```

BAB 4

HASIL PENGUJIAN

Input yang digunakan berjumlah 4, dengan tingkat kesulitan yang meningkat. Berikut konfigurasi papan yang disimpan pada file .txt

1. 1.txt

```
6 6
8
AA...G
B..D.G
BPPD.GK
B..D..
C...EE
C.FFF.
```

2. 2.txt

```
6 6
12
.BEELL
.B.FJJ
PP.FGIK
ACCCGI
A.D.GH
A.D..H
```

3. 3.txt

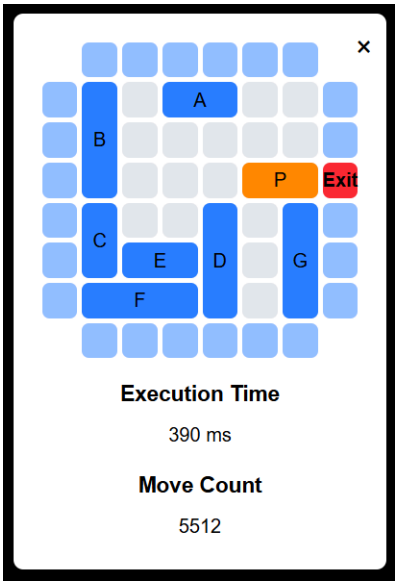
```
6 6
13
E.CGGG
E.CHHB
JJCIIB
FAAP..
FD.PLL
.DMMM.
   K
```

4. 4.txt

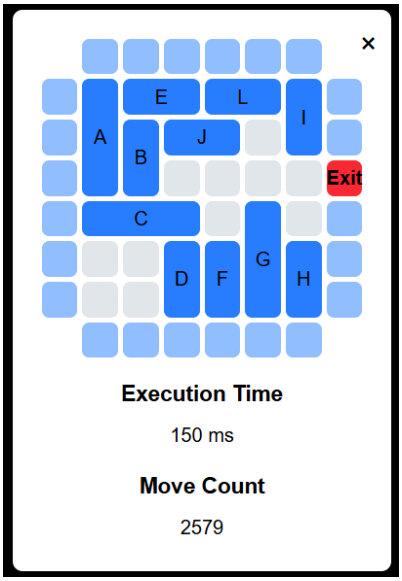
```
6 6
12
.GHHII
.G.FJJ
PP.FE.K
ABBBEL
A.C.EL
A.CDDL
```

1. Uniform Cost Search (UCS)

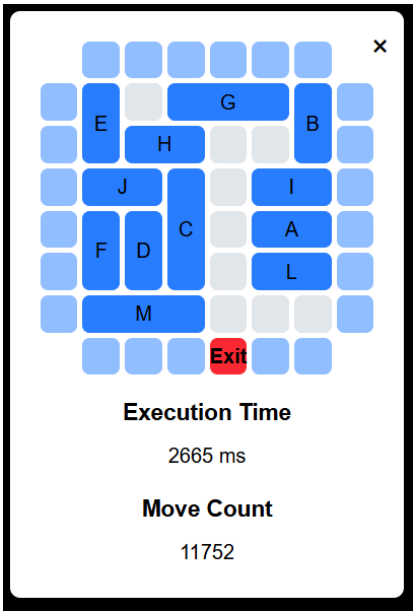
a. 1.txt



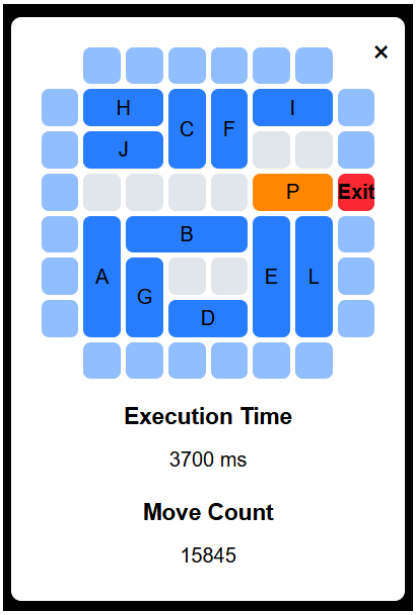
b. 2.txt



c. 3.txt

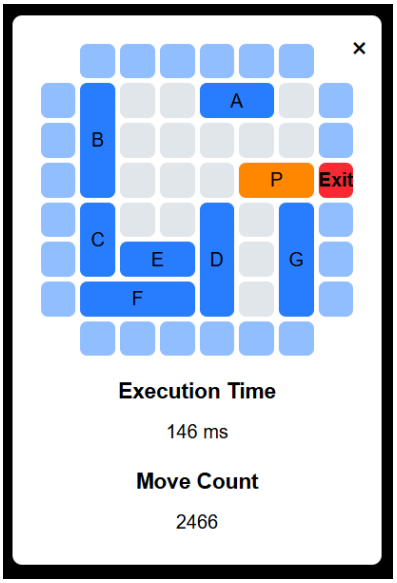


d. 4.txt

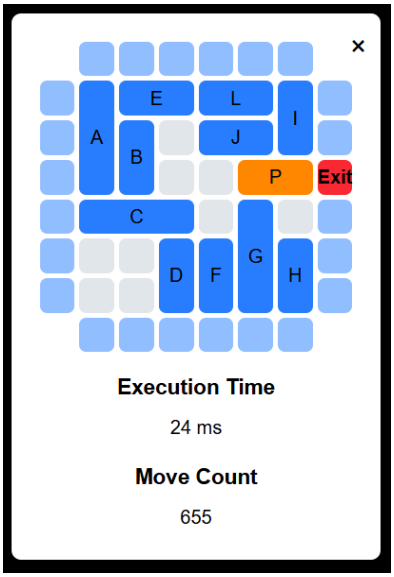


2. A*

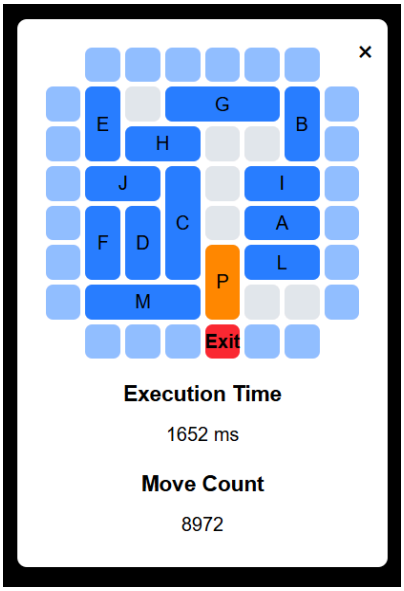
a. 1.txt



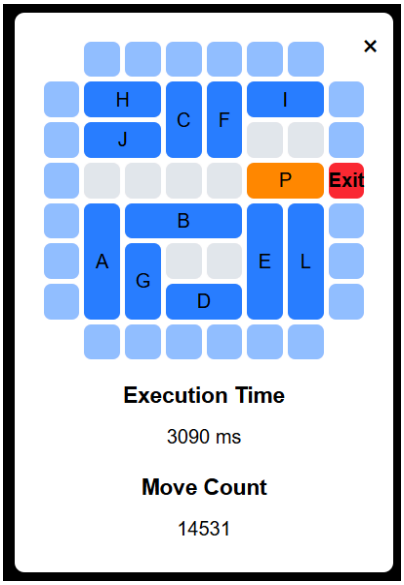
b. 2.txt



c. 3.txt



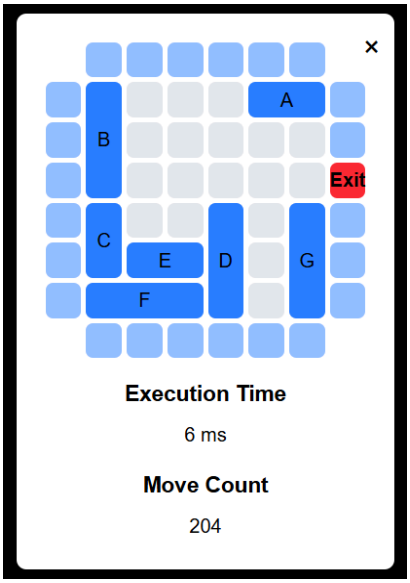
d. 4.txt



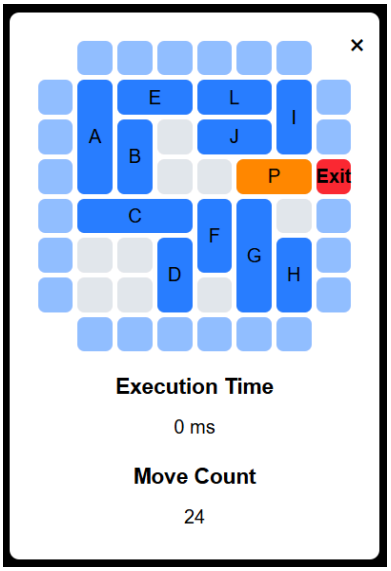
e. 1.txt heuristik 3

3. Greedy Best-First Search (GBFS)

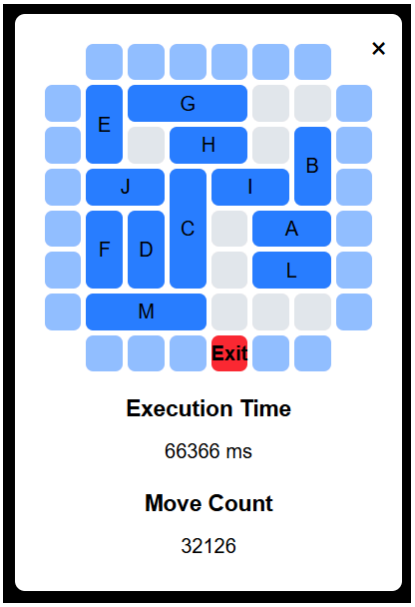
a. 1.txt



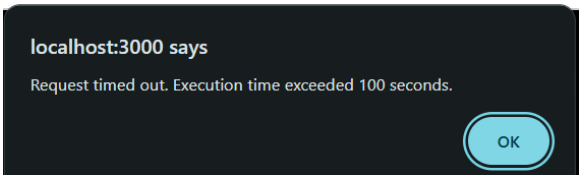
b. 2.txt



c. 3.txt

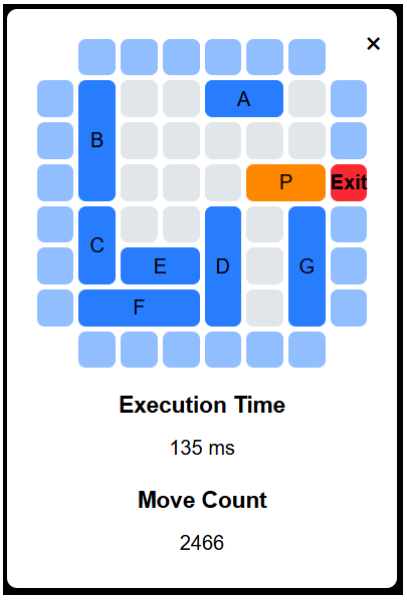


d. 4.txt

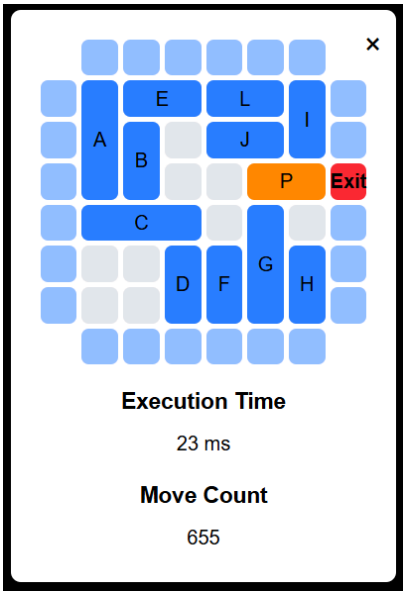


4. Simulated Annealing (SA) [BONUS]

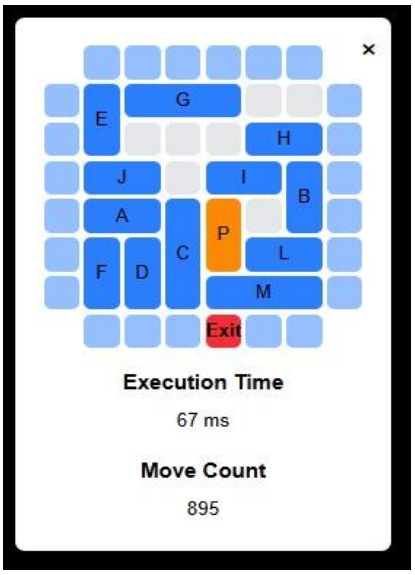
a. 1.txt



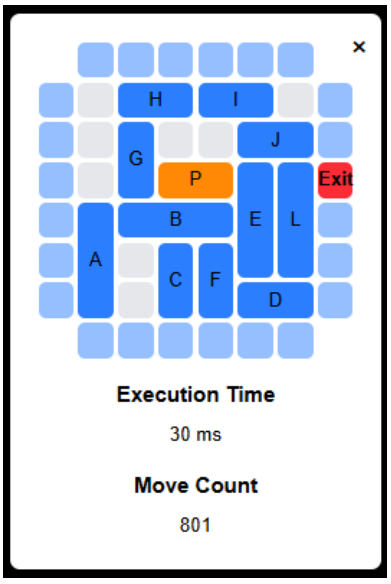
b. 2.txt



c. 3.txt



d. 4.txt



BAB 5
ANALISIS

Dari hasil percobaan antara 4 algoritma dapat dilihat, waktu yang didapat algoritma SA sangat cepat di nomor 3 dan 4, meskipun begitu dikarenakan SA yang probabilistik, hal ini hanya bisa dicapai setelah melakukan solve beberapa kali (selama pengujian ini 10-12 kali solving). Secara umum GBFS lebih cepat menyelesaikan puzzle untuk persoalan yang sederhana seperti nomer 1 dan 2. Meskipun begitu sangat lambat untuk soal yang kompleks. Algoritma UCS bisa mencari jawaban untuk semua soal namun dengan jumlah node dikunjungi yang jauh lebih banyak. Sedangkan A* lebih baik dari UCS di semua soal dan lebih konsisten untuk mencari solusi. Pada implementasi yang dibuat, heuristik yang dibuat dirasa masih kurang baik dalam menagivasi GBFS untuk menyelesaikan puzzle karena itu juga selisih A* dan UCS tidak terlalu jauh.

Secara teoritis dapat dilihat secara umum antara 3 algoritma (UCS, GBFS dan A*). Untuk mempermudah, papan akan dibatasi ke suatu persegi (s x s) dengan n buah mobil dengan masing masing panjang 2. Diawali dengan Priority Queue yang diisi dengan child pada generate Child yang pada worst case menghasilkan paling banyak n (jumlah mobil) * s (panjang sisi - 2). Dan kedalaman (d) yang bisa digunakan dalam tiap node tidak lebih dari n x s. Oleh karena itu jumlah node yang mungkin dibuat adalah kemungkinan dengan tiap kemungkinan melakukan copy board ($O(s^2)$), copy cars ($O(n)$) dan tiap pengecekan goal adalah $O(1)$.

Dalam algoritma UCS, tidak ada fungsi heuristik yang harus dijalankan, karena total cost / akumulasi biaya sudah dilakukan sebagai atribut suatu board. Oleh karena itu algoritma UCS memiliki kompleksitas : worst case $O((n \cdot s)^{n.s} \cdot s^2)$, best case $O(s^2)$ dan secara umum $O(\text{jumlah node} \cdot s^2)$. Jumlah node dapat didekati dengan hasil eksperimen untuk mencari rata rata.

Dalam algoritma GBFS, dibatasi penggunaan heuristik hanya untuk heuristik 1 (mencari jarak ke tujuan + menghitung jumlah mobil yang menghalangi). Algoritma heuristik tersebut memiliki kompleksitas $O(s)$. Oleh karena itu algoritma GBFS memiliki kompleksitas : worst case $O((n \cdot s)^{n.s} \cdot s^2)$, best case $O(s^2)$ dan secara umum $O(\text{jumlah node} \cdot s^2)$. Jumlah node dapat didekati dengan hasil eksperimen untuk mencari rata rata. Hasil yang sama juga berlaku

untuk algoritma A* hal ini karena kompleksitas algoritma yang digunakan hanya O(s). jika menggunakan heuristik lain yang lebih tinggi maka kompleksitas juga akan meningkat.

Dalam algoritma SA, dengan metode generate Child yang sama, namun ditambah metode random untuk memilih satu Child yang akan dibandingkan heuristiknya dengan state saat ini, jika lebih rendah (lebih dekat tujuan) Child maka akan langsung berganti ke child. Dengan suhu awal yang ditentukan 1000 dan cooling rate 0.3 % di tiap iterasi dalam kasus worst case (tidak ada jalur yang ditemukan) akan dilakukan 4613 node, threshold yang digunakan untuk menghentikan loop adalah 0.001. Oleh karena itu kompleksitasnya adalah $O(s^2)$.

BAB 6

SPESIFIKASI BONUS

1. **Simulated Annealing**

   Algoritma pathfinding Simulated Annealing adalah algoritma yang bersifat stokastik (berubah setiap kali eksekusi) dikarenakan adanya unsur probabilitas yang dimasukkan pada penghitungannya. Algoritma ini terinspirasi dari proses pendinginan logam (annealing), dimana logam dipanaskan dan didinginkan perlahan untuk mencari kondisi yang stabil seiring menurunnya suhu.

2. **Heuristik**

   Heuristik tambahan yang digunakan adalah heuristik 2 dan 3. Yang masing masing adalah heuristik 1 yang lebih dirincikan Heuristik 1 adalah jarak primary ke exit ditambah jumlah mobil yang menghalangi, heuristik 2 adalah jarak yang sama ditambah dengan pemberatan mobil yang membloking berdasarkan apakah bisa dipindah / bergerak. Heuristik 3 adalah jarak yang sama ditambah dengan penghitungan 2 level blocking, level 1 bernilai 3 kali lebih berat dibanding level ke dua.

3. **GUI**

   GUI diimplementasikan menggunakan aplikasi berbasis web, dengan menggunakan framework NextJS untuk frontend website dan Java Spring Boot untuk backend.

   a. React Components

      i. Square

         Component Square diimplementasikan untuk meng-handle drag-and-drop dari pieces, sebagai penerima.

```
// square.js

"use client"

import { useDrop } from "react-dnd";

export default function Square({
    style,
    children,
    setExit,
```

```
    row,
    col,
    onDrop,
    canPlacePiece,
    isEdge,
    disableDrag
}) {
    if (disableDrag) {
        return (
            <div className={`${style} z-0`}
                style={{
                    "--row": Number(row) + 1,
                    "--column": Number(col) + 1,
                    "--row-span": 1,
                    "--column-span": 1,
                }}
            >
                {children}
            </div>
        );
    }

    const onClick = (e) => {
        e.stopPropagation();
        e.preventDefault();
        if (isEdge) {
            setExit({ exitRow: row, exitCol: col });
        }
    };

    const [{ isOver, canDrop }, drop] = useDrop(() => ({
        accept: 'piece',
        drop: (item, monitor) => onDrop(item, row - 1, col - 1),
        canDrop: (item, monitor) => canPlacePiece(item, row - 1, col - 1),
        collect: (monitor) => ({
            isOver: !!monitor.isOver(),
            canDrop: !!monitor.canDrop(),
        }),
```

```
    }));

    const borderStyle = isOver && canDrop ?
        "border-2 border-green-500" : canDrop ?
        "border-2 border-blue-500" : "border-2 border-gray-300";

    return (
        <div className={`${style} ${borderStyle} z-0`}
            ref={drop}
            onClick={onClick}
            style={{
                "--row": Number(row) + 1,
                "--column": Number(col) + 1,
                "--row-span": 1,
                "--column-span": 1,
            }}
        >
        {children}
        </div>
    );
};
```

ii. Piece

Component Piece diimplementasikan sebagai item yang dapat di-drag.

```
// piece.js

"use client"

import { useDrag } from "react-dnd";
import * as motion from "motion/react-client"

export default function Piece({ id, row, col, spanRow, spanCol, disableDrag}) {
    const isPrimary = id === 'P';
    const bg_color = isPrimary ? "bg-orange-400" : "bg-blue-500";
    if (disableDrag) {
        return (
            <div
```

```
            className={`z-10 ${bg_color}
                dynamic-grid-item rounded-md flex
                items-center justify-center text-center`}
            style={{
                "--row": row,
                "--column": col,
                "--row-span": spanRow,
                "--column-span": spanCol,
            }}
        >
            {id}
        </div>
    );
}

const [{ isDragging }, drag] = useDrag(() => ({
    type: "piece",
    item: { id, row, col, spanRow, spanCol},
    collect: (monitor) => ({
        isDragging: !!monitor.isDragging(),
    }),
}));

return (
    <motion.div
        ref={drag}
        className={`z-10 ${bg_color}
            dynamic-grid-item rounded-md
            flex items-center justify-center text-center`}
        style={{
            "--row": row,
            "--column": col,
            "--row-span": spanRow,
            "--column-span": spanCol,
        }}
        whileHover={{ scale: 0.75, cursor: "grab" }}
        whileTap={{ scale: 0.75, cursor: "grabbing" }}
        whileDrag={{ scale: 0.75, rotate: 10, cursor: "grabbing" }}
```

```
        >
            {id}
        </motion.div>
    );
};
```

iii. Board

Component Board akan me-render Square dan Piece tergantung dari state yang ada.

```
// board.js

"use client"
import { use, useEffect, useRef } from "react";
import { DndProvider } from 'react-dnd'
import { HTML5Backend } from 'react-dnd-html5-backend'

import Square from "@/components/square";
import Piece from "@/components/piece";

export default function Board({
    rows,
    cols,
    pieces,
    setPieces,
    occupiedCells,
    setOccupiedCells,
    primaryPiece,
    exit,
    setExit,
    disableDrag = false,
}) {
    const sizeRef = useRef({rows, cols});
    const occupiedCellsRef = useRef(occupiedCells);

    useEffect(() => {
        if (disableDrag) return;
```

```
        setExit({ exitRow: exit.exitRow, exitCol: exit.exitCol });
        sizeRef.current = { rows: Number(rows) + 2, cols: Number(cols) + 2 };
    }, [rows, cols]);

    useEffect(() => {
        occupiedCellsRef.current = occupiedCells;
    }, [occupiedCells]);

    function renderPieces() {
        return pieces.map(piece => {
            const pieceRow = Number(piece.y) + 2;
            const pieceCol = Number(piece.x) + 2;
            const spanRow = piece.orientation === "horizontal" ?
                1 : piece.length;
            const spanCol = piece.orientation === "horizontal" ?
                piece.length : 1;
            const isPrimary = piece.id === primaryPiece;
            return (
                <Piece
                    key={piece.id}
                    row={pieceRow}
                    col={pieceCol}
                    spanRow={spanRow}
                    spanCol={spanCol}
                    id={piece.id}
                    isPrimary={isPrimary}
                    disableDrag={disableDrag}
                />
            );
        });
    };

    function renderCells() {
        const canPlacePiece = (piece, targetRow, targetCol) => {
            if (disableDrag) return false;
            const currentOccupied = occupiedCellsRef.current;
            const pieceLength = Math.max(piece.spanRow, piece.spanCol);
            const orientation = piece.spanCol > 1 ?
```

```
            "horizontal" : "vertical";
        const newRows = sizeRef.current.rows;
        const newCols = sizeRef.current.cols;
        if (
            targetRow < 0 ||
            targetCol < 0 ||
            targetRow > newRows - 1 ||
            targetCol > newCols - 1
        ) return false;
        if (orientation === "horizontal") {
            if (targetCol + pieceLength > newCols - 2) return false;
            if (targetRow >= newRows - 2) return false;
        } else {
            if (targetRow + pieceLength > newRows - 2) return false;
            if (targetCol >= newCols - 2) return false;
        }

        for (let i = 0; i < pieceLength; i++) {
            const checkX = orientation === "horizontal" ?
                targetCol + i : targetCol;
            const checkY = orientation === "vertical" ?
                targetRow + i : targetRow;
            if (currentOccupied.some(cell =>
                cell.x === checkX &&
                cell.y === checkY &&
                cell.pieceId !== piece.id
            )) {
                return false;
            }
        }
        return true;
    }


    function handleDrop(piece, targetRow, targetCol) {
        if (disableDrag) return;
        if (!canPlacePiece(piece, targetRow, targetCol)) return;
```

```
            setPieces(prevPieces =>
                prevPieces.map(p =>
                p.id === piece.id
                    ? { ...p, x: targetCol, y: targetRow }
                    : p
                )
            );

            setOccupiedCells(prevCells => {
                const filteredCells = prevCells.filter(
                    cell => cell.pieceId !== piece.id);

                const orientation = piece.spanCol > 1 ?
                    "horizontal" : "vertical";
                const length = Math.max(piece.spanRow, piece.spanCol);
                const newCells = Array.from({ length }, (_, i) => ({
                    x: orientation === "horizontal" ?
                        targetCol + i : targetCol,
                    y: orientation === "vertical" ?
                        targetRow + i : targetRow,
                    pieceId: piece.id,
                }));

                return [...filteredCells, ...newCells];
            });
        }

        const Squares = [];
        const newRows = sizeRef.current.rows;
        const newCols = sizeRef.current.cols;
        for (let index = 0; index < newRows * newCols; index++) {
            let className = "w-full dynamic-grid-item aspect-square rounded-md
                transition-all bg-blue-500 flex items-center justify-center
                text-center ";
            let content = "";
            const exitRow = exit.exitRow, exitCol = exit.exitCol;
            const row = Math.floor(index / newCols);
            const col = index % newCols;
```

```
const isCorner =
    (row === 0 && col === 0) ||
    (row === 0 && col === newCols - 1) ||
    (row === newRows - 1 && col === 0) ||
    (row === newRows - 1 && col === newCols - 1);

const isExit = index === newCols * exitRow + exitCol;

const isEdge =
    !isCorner && (
        row === 0 ||                        // top
        row === newRows - 1 ||              // bottom
        col === 0 ||                        // left
        col === newCols - 1                 // right
    );

if (isExit) {
    className += "bg-red-500 text-black font-bold";
    content = "Exit";
} else if (isCorner) {
    className += "bg-purple-400 text-white opacity-0";
} else if (isEdge) {
    className += "bg-blue-400 text-white opacity-50
        hover:bg-blue-600";
} else {
    className += "bg-gray-200 text-black";
}
Squares.push(
    <Square
        key={`row-${row}-col-${col}`}
        style={className}
        row={row}
        col={col}
        setExit={setExit}
        onDrop={handleDrop}
        canPlacePiece={canPlacePiece}
        isEdge={isEdge}
```

```
                    disableDrag={disableDrag}
                >
                    {content}
                </Square>
            );
        }
        return Squares;
    }

    return (
        <DndProvider backend={HTML5Backend}>
            <div
                key={primaryPiece}
                className={`
                    w-1/6
                    grid
                    [grid-template-columns:repeat(${sizeRef.current.cols},1fr)]
                    [grid-template-rows:repeat(${sizeRef.current.rows},1fr)]
                    gap-1
                `}
            >
                {renderPieces()}
                {renderCells()}
            </div>
        </DndProvider>
    )
}
```

iv. PieceSpawner

Component ini betujuan membuat Piece baru dan menaruhnya langsung pada Board.

```
// pieceSpawner.js

"use client"

export default function PieceSpawner({
```

```
    handleOnSubmit,
    orientation,
    setOrientation
}) {
    return (
        <form
            onSubmit={handleOnSubmit}
            className="my-2 flex flex-col gap-4 bg-white text-black p-4
                rounded-lg shadow-md"
        >
            <div className="flex flex-row gap-2 w-1/2">
                <label className="text-sm font-medium">Length</label>
                <input
                    type="number"
                    name="length"
                    min={2}
                    defaultValue={2}
                    required
                    className="border border-gray-300 rounded-lg p-2
                        focus:outline-none focus:ring-2 focus:ring-blue-500"
                />
            </div>
            <div className="flex flex-row gap-4 items-center">
                <span className="text-sm font-medium">Orientation:</span>
                <label className="flex items-center gap-1">
                    <input
                        type="radio"
                        name="orientation"
                        value="horizontal"
                        checked={orientation === "horizontal"}
                        onChange={() => setOrientation("horizontal")}
                    />
                    Horizontal
                </label>
                <label className="flex items-center gap-1">
                    <input
                        type="radio"
                        name="orientation"
```

```
                        value="vertical"
                        checked={orientation === "vertical"}
                        onChange={() => setOrientation("vertical")}
                    />
                    Vertical
                </label>
            </div>
            <button
                type="submit"
                className="bg-blue-500 text-white rounded-lg p-2
                    hover:bg-blue-600 transition duration-200"
            >
                Create Piece
            </button>
        </form>
    )
}
```

v. SizeForm

Component ini berguna untuk mengatur ukuran dari Board serta me-reset Board dari Piece.

```
// sizeForm.js

"use client"

export default function SizeForm({ setGridSize, setRefresh }) {
    const handleSubmit = (e) => {
        e.preventDefault();
        const rows = Number(e.target.rows.value);
        const cols = Number(e.target.cols.value);
        setGridSize({rows, cols});
        setRefresh(true);
    }

    return (
        <form
```

```
            onSubmit={handleSubmit}
            className="flex flex-col gap-4 bg-white text-black
                p-2 rounded-lg shadow-md"
        >
            <div className="flex flex-row gap-2 w-full">
                <label className="text-sm font-medium">Rows</label>
                <input
                    type="number"
                    name="rows"
                    defaultValue={3}
                    className="w-1/2 border border-gray-300 rounded-lg
                        p-2 focus:outline-none focus:ring-2
                        focus:ring-blue-500 bg-white text-black"
                />
                <label className="text-sm font-medium">Columns</label>
                <input
                    type="number"
                    name="cols"
                    defaultValue={3}
                    className="w-1/2 border border-gray-300 rounded-lg p-2
                        focus:outline-none focus:ring-2 focus:ring-blue-500
                        bg-white text-black"
                />
            </div>
            <button
                type="submit"
                className="bg-blue-500 text-white rounded-lg p-2
                    hover:bg-blue-600 transition duration-200"
            >
                Initialize Board
            </button>
        </form>
    )
}
```

vi. TextFileUploadForm

Component ini berguna untuk membaca file .txt dan merender hasil bacaannya ke Board.

```
// fileUpload.js

'use client';

import { useRef } from 'react';

export default function TextFileUploadForm(
    {
        setExit,
        setPieces,
        setOccupiedCells,
        setPrimaryPiece,
        setGridSize,
    }
) {
    const fileInputRef = useRef(null);

    const processTextContent = (text) => {
        try {
            const lines = text.split(/\r?\n/);
            console.log("Lines: ", lines);
            const gridSize = lines[0].split(' ').map(Number);
            console.log("Grid Size: ", gridSize);
            const rows = gridSize[0];
            const cols = gridSize[1];
            let exit = undefined;
            if (lines.length === rows + 3) {
                if (lines[2].includes('K')) {
                    exit = { exitRow: 0, exitCol: lines[2].indexOf('K') + 1};
                } else if (lines[lines.length - 1].includes('K')) {
                    exit = {
                        exitRow: rows + 1,
                        exitCol: lines[lines.length - 1].indexOf('K') + 1
                    };
                }
```

```
        }
        else if (lines.length < rows + 2) {
            throw new Error("Invalid number of lines in input file.");
        }

        const pieces = [];
        const occupiedCells = [];
        const pieceIdMap = new Map();
        let primaryPiece = undefined;
        const rowSkip = (lines[2].includes('K')) ? 3 : 2;
        for (let i = rowSkip; i < rows + rowSkip; i++) {
            if (!exit && lines[i].includes('K')) {
                exit = ({
                    exitRow: i - 1,
                    exitCol: (lines[i].indexOf('K') === 0) ?
                        0 : (lines[i].indexOf('K') + 1)});
            }
            for (let j = 0; j < lines[i].length; j++) {
                if (
                    lines[i].length !== cols &&
                    lines[i].length !== cols + 1
                ) {
                    throw new Error(
                        "Invalid number of columns in input file."
                    );
                }
                const cell = lines[i].charAt(j);
                if (
                    cell === '.' ||
                    cell === 'K' ||
                    cell === ' ' ||
                    cell === '\r'
                ) {
                    continue;
                }
                let currentJ = j;
                do {
                    j++;
```

```
                } while(
                    j < lines[i].length && lines[i].charAt(j) === cell
                );
                const length = j - currentJ;
                j--;
                if (length < 2) {
                    continue;
                }
                if (pieceIdMap.has(cell) ) {
                    throw new Error(`Duplicate piece ID found: ${cell}`);
                }
                pieceIdMap.set(cell, true);
                const piece = {
                    id: cell,
                    x: currentJ,
                    y: i - rowSkip,
                    orientation: 'horizontal',
                    length: length,
                };
                pieces.push(piece);
                if (cell === 'P') {
                    primaryPiece = true;
                }

                for (let k = 0; k < length; k++) {
                    occupiedCells.push({
                        x: currentJ + k,
                        y: i - rowSkip,
                        pieceId: cell,
                    });
                }
            }
        }
        const extraCol = exit?.exitCol === 0 ? 1 : 0;
        for (let j = 0; j < cols + extraCol; j++) {
            for (let i = rowSkip; i < rows + rowSkip; i++) {
                const cell = lines[i].charAt(j);
                if (
```

```
            cell === '.' ||
            cell === 'K' ||
            cell === ' ' ||
            cell === '\r'
        ) {
            continue;
        }
        let currentI = i;
        do {
            i++;
        } while(i < lines.length && lines[i].charAt(j) === cell);
        const length = i - currentI;
        i--;
        if (length < 2) {
            continue;
        }
        if (pieceIdMap.has(cell)) {
            throw new Error(`Duplicate piece ID found: ${cell}`);
        }
        pieceIdMap.set(cell, true);
        const piece = {
            id: cell,
            x: j,
            y: currentI - rowSkip,
            orientation: 'vertical',
            length: length,
        };
        if (cell === 'P') {
            primaryPiece = true;
        }
        pieces.push(piece);
        for (let k = 0; k < length; k++) {
            occupiedCells.push({
                x: j,
                y: currentI + k - rowSkip,
                pieceId: cell,
            });
        }
```

```
            }
        }
        const expectedPieces = Number(lines[1]);
        if (isNaN(expectedPieces)) {
            throw new Error("Invalid number of pieces in input file.");
        }
        if (pieces.length !== expectedPieces) {
            throw new Error(
                `Expected ${expectedPieces} pieces, but found
                 ${pieces.length}.`
            );
        }
        const mapIds = Array.from(pieceIdMap.keys()).filter(
             id => id !== 'P' && id !== 'K' && id !== ' '
        );
        mapIds.sort();
        const validIds = [];
        let charCode = 'A'.charCodeAt(0);
        while (validIds.length < mapIds.length) {
            if (String.fromCharCode(charCode) !== 'K') {
                validIds.push(String.fromCharCode(charCode));
            }
            charCode++;
        }
        if (JSON.stringify(mapIds) !== JSON.stringify(validIds)) {
            throw new Error(
                `Piece IDs must be consecutive letters starting from 'A'
                (skipping 'K'). Found: ${mapIds.join(', ')}`
            );
        }
        setGridSize({ rows: rows, cols: cols });
        if (primaryPiece) setPrimaryPiece(primaryPiece);
        if (exit) setExit(exit);
        setPieces(pieces);
        setOccupiedCells(occupiedCells);
    } catch (error) {
        console.log("Error processing text content:", error);
        alert("Error processing file: " + error.message);
```

```
    }
}

const handleSubmit = async (e) => {
    e.preventDefault();
    const file = fileInputRef.current.files[0];
    if (!file) {
        alert("Please select a file.");
        return;
    }
    const reader = new FileReader();
    reader.onload = (event) => {
        const text = event.target.result;
        processTextContent(text);
    };
    reader.readAsText(file);
};

return (
    <form
        onSubmit={handleSubmit}
        className="p-6 max-w-md mx-auto bg-white rounded-lg shadow
             space-y-6 my-2"
    >
        <div>
            <label
                htmlFor="file-upload"
                className="block text-sm font-medium text-gray-700 mb-2"
            >
                Load from a .txt file
            </label>
            <input
                id="file-upload"
                type="file"
                accept=".txt"
                ref={fileInputRef}
                className="block w-full text-sm text-gray-900 border
                    border-gray-300 rounded-lg cursor-pointer
```

```
                        bg-gray-50 focus:outline-none"
                />
            </div>
            <button
                type="submit"
                className="w-full bg-blue-600 text-white px-4 py-2
                    rounded-lg hover:bg-blue-700 transition-colors"
            >
                Submit
            </button>
        </form>
    );
}
```

vii. ResultBoard

Component ini berguna untuk menampilkan hasil akhir Board tiap-tiap gerakan.

```
// resultBoard.js

“use client”

import Board from "@/components/board";

export default function ResultBoard({
    board,
    exit,
}) {
    const grid = board.board;
    const rows = grid.length;
    const cols = grid[0].length;
    const cars = board.cars;
    const primaryPiece = 'P';

    const pieces = [];
    cars.forEach((car, _) => {
```

```
        const orient = car.direction === "X" ? "horizontal" : "vertical";
        const piece = {
            id: car.symbol,
            x: car.start.X,
            y: car.start.Y,
            orientation: orient,
            length: orient === "horizontal" ?
                car.end.X - car.start.X + 1 : car.end.Y - car.start.Y + 1,
        };
        pieces.push(piece);
    });

    return (
        <Board
            rows={rows + 2}
            cols={cols + 2}
            pieces={pieces}
            primaryPiece={primaryPiece}
            exit={exit}
            disableDrag={true}
        />
    );
}
```

viii. ResultModal

Component ini berguna untuk menampilkan hasil akhir dalam animasi.

```
// resultModal.js

"use client";

import React, { useEffect, useState } from "react";
import ResultBoard from "@/components/resultBoard";
import { useRouter } from "next/navigation";

export default function ResultModal({
    boards,
```

```
    exit,
    executionTime,
    moveCount,
}) {
    const router = useRouter();
    const [index, setIndex] = useState(0);

    useEffect(() => {
        const interval = setInterval(() => {
            setIndex((prev) => (prev + 1) % boards.length);
        }, 500);
        return () => clearInterval(interval);
    }, [boards.length]);

    return (
        <div className="fixed inset-0 bg-black bg-opacity-50
            flex items-center justify-center z-[1000]">
            <div className="bg-white text-black rounded-lg p-6
                shadow-lg min-w-[300px] relative">
                <button
                    onClick={(e) => {router.back();}}
                    className="absolute top-3 right-3 bg-transparent
                        border-none text-2xl cursor-pointer"
                    aria-label="Close"
                >
                    &times;
                </button>
                <ResultBoard
                    board={boards[index]}
                    exit={exit}
                />
                <div className="mt-4 text-center">
                    <h2 className="text-lg font-semibold mb-2">
                        Execution Time
                    </h2>
                    <p>{executionTime} ms</p>
                    <h2 className="text-lg font-semibold mt-4 mb-2">
                         Move Count
```

```
                    </h2>
                    <p>{moveCount}</p>
                </div>
            </div>
        </div>
    );
}
```

b. Main Page

```
// page.js

"use client"

import Board from "@/components/board";
import SizeForm from "@/components/sizeForm";
import { useEffect, useState } from "react";
import PieceSpawner from "@/components/pieceSpawner";
import axios from "axios";
import { useRouter } from "next/navigation";
import { useSearchParams } from "next/navigation";
import ResultModal from "@/components/resultModal";
import TextFileUploadForm from "@/components/fileUpload";
import AlgorithmSelector from "@/components/algoSelect";

export default function Home() {
  const searchParams = useSearchParams();
  const router = useRouter();
  const [grids, setGrids] = useState([[[]]]);
  const [executionTime, setExecutionTime] = useState(0);
  const [moveCount, setMoveCount] = useState(0);
  const showResult = searchParams.get("showResult");
  const [gridSize, setGridSize] = useState({rows: 3, cols: 3});
  const [refresh, setRefresh] = useState(false);
  const [primaryPiece, setPrimaryPiece] = useState();
  const [pieces, setPieces] = useState([]);
  const [occupiedCells, setOccupiedCells] = useState([]);
```

```
  const [orientation, setOrientation] = useState("horizontal");
  const [exit, setExit] = useState({
    exitRow: 1, exitCol: Number(gridSize.cols) + 1
  });
  const [loading, setLoading] = useState(false);

  const baseUrl = process.env.NEXT_PUBLIC_API_URL ||
    "http://localhost:8080/api";

  useEffect(() => {
    if (refresh) {
      setPieces([]);
      setOccupiedCells([]);
      setPrimaryPiece(undefined);
      setExit({exitRow: 1, exitCol: Number(gridSize.cols) + 1});
      setRefresh(false);
    }
  }, [refresh]);

  const handleAddPiece = (e) => {
    e.preventDefault();
    const length = Number(e.target.length.value);
    let idIndex = pieces.length - 1;
    if (idIndex >= 10) idIndex++; // Skip 'K'
    if (idIndex >= 15) idIndex++; // Skip 'P'
    if (pieces.length === 0 || primaryPiece === undefined) {
      setPrimaryPiece(true);
      idIndex = 15;
    }
    const id = String.fromCharCode(65 + idIndex);
    let newPiece = {
      id: id,
      length: length,
      orientation: orientation,
      x: 0,
      y: 0
    };
    let placed = false;
```

```
  for (let x = 0; x < gridSize.cols; x++) {
    for (let y = 0; y < gridSize.rows; y++) {
      if (
        (
          orientation === "horizontal" &&
          x + Number(length) > gridSize.cols
        ) ||
        (
          orientation === "vertical" &&
          y + Number(length) > gridSize.rows
        )
      ) {
        continue;
      }

      let overlap = false;
      for (let i = 0; i < length; i++) {
        let checkX = orientation === "horizontal" ? x + i : x;
        let checkY = orientation === "vertical" ? y + i : y;
        if (occupiedCells.some(
          cell => cell.x === checkX && cell.y === checkY
        )) {
          overlap = true;
          break;
        }
      }

      if (!overlap) {
        placed = true;
        newPiece.x = x;
        newPiece.y = y;
        break;
      }
    }
    if (placed) break;
  }
  if (placed && pieces.length < 25) {
    setPieces([...pieces, newPiece]);
```

```
      setOccupiedCells([
        ...occupiedCells,
        ...Array.from({ length }, (_, i) => ({
          x: newPiece.x + (orientation === "horizontal" ? i : 0),
          y: newPiece.y + (orientation === "vertical" ? i : 0),
          pieceId: newPiece.id,
        })),
      ]);
    }
    else if (pieces.length >= 25) {
      alert("Maximum number of pieces reached");
    } else {
      alert("Piece cannot be placed on the board");
    }
  };

  const submitBoard = async (selectedHeuristic) => {
    if (!primaryPiece) {
      alert("Board must contain a primary piece");
      return;
    }
    setLoading(true);
    const formattedExit = {
      exitRow: exit.exitRow - 1,
      exitCol: exit.exitCol - 1,
    };
    const boardState = {
      gridSize,
      primaryPiece,
      occupiedCells,
      exit: formattedExit,
      heuristics: selectedHeuristic,
    };
    const boardStateJson = JSON.stringify(boardState, null, 2);
    console.log(boardStateJson);
    try {
      const response = await axios.post(`${baseUrl}/board`, boardState, {
        timeout: 30000, // 30 seconds timeout
```

```
      });
      if (response.data && response.data.boards) {
        router.push("?showResult=true");
        console.log("Solution found!");
        console.log("Response data:", response.data.boards);
        setGrids(response.data.boards);
        setExecutionTime(response.data?.executionTime || 0);
        setMoveCount(response.data?.moveCount || 0);
      } else {
        alert("Solution not found!");
      }
    } catch (error) {
      if (error.code === "ECONNABORTED") {
        console.error("Request timed out");
      } else {
        console.error("Error solving puzzle:", error);
      }
    } finally {
      setLoading(false);
    }
  };
  return (
    <div className = "flex flex-row justify-between">
      <div className="flex-auto w-full h-screen justify-center px-10 py-10">
        <Board
          rows={gridSize.rows}
          cols={gridSize.cols}
          pieces={pieces}
          setPieces={setPieces}
          primaryPiece={primaryPiece}
          occupiedCells={occupiedCells}
          setOccupiedCells={setOccupiedCells}
          exit={exit}
          setExit={setExit}
        />
      </div>
      <div className="flex-col w-1/4 h-screen bg-gray-500 p-4">
        <SizeForm setGridSize={setGridSize} setRefresh={setRefresh}/>
```

```
        <PieceSpawner
          handleOnSubmit={handleAddPiece}
          orientation={orientation}
          setOrientation={setOrientation}
        />
        <TextFileUploadForm
          setGridSize={setGridSize}
          setPieces={setPieces}
          setOccupiedCells={setOccupiedCells}
          setPrimaryPiece={setPrimaryPiece}
          setExit={setExit}
        />
        <AlgorithmSelector
          submitBoard={submitBoard}
        />
      </div>
      <>
        {loading && (
          <div
            className="fixed inset-0 flex items-center justify-center
              bg-black bg-opacity-50 z-50"
            role="dialog"
            aria-modal="true"
            aria-label="Solving puzzle"
          >
            <div className="bg-white p-8 rounded-lg shadow-lg flex
              flex-col items-center">
              <span className="text-lg font-semibold mb-4">
                Solving, please wait...
              </span>
              <div className="w-12 h-12 border-4 border-gray-300
                border-t-red-500 rounded-full animate-spin">
              </div>
            </div>
          </div>
        )}

        {showResult && (
```

```
          <ResultModal
            boards={grids}
            exit={exit}
            executionTime={executionTime}
            moveCount={moveCount}
          />
        )}
      </>
    </div>
  );
}
```

4. Server

a. SolveController (API)

```
// SolveController.java
package com.tucil3.backend.controller;
import java.util.*;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;
import com.tucil3.backend.lib.Board;
import com.tucil3.backend.lib.Coor;
import com.tucil3.backend.lib.Solver;
@RestController
@CrossOrigin(origins = "*")
public class SolveController {
    public static class GridSize {
        public int rows;
```

```
        public int cols;
    }
    public static class OccupiedCells {
        public char pieceId;
        public int x;
        public int y;
    }
    public static class Exit {
        public int exitCol;
        public int exitRow;
    }
    public static class Heuristic {
        public int heur;
        public String pathfinding;
    }
    public static class RequestPayload {
        public GridSize gridSize;
        public List<OccupiedCells> occupiedCells;
        public Exit exit;
        public Heuristic heuristics;
    }
    public static class ResponsePayload {
        public List<Board> boards;
        public int executionTime;
        public int moveCount;
        public ResponsePayload(
            List<Board> boards,
            int executionTime,
            int moveCount
        ) {
            this.boards = boards;
```

```
            this.executionTime = executionTime;
            this.moveCount = moveCount;
        }
    }
    @PostMapping("/api/board")
    public ResponseEntity<?> solveBoard(@RequestBody RequestPayload payload) {
        int width = payload.gridSize.cols;
        int height = payload.gridSize.rows;
        char[][] grid = new char[height][width];
        for (int i = 0; i < height; i++) {
            Arrays.fill(grid[i], '.');
        }
        for (OccupiedCells cell : payload.occupiedCells) {
            grid[cell.y][cell.x] = cell.pieceId;
        }
        int exitX = payload.exit.exitCol;
        int exitY = payload.exit.exitRow;
        Coor exitCoor = new Coor(exitY, exitX);
        Solver solver = new Solver(board);
        List<Board> resultBoards = solver.solving("A*", 1);
        if (resultBoards == null) {
            return ResponseEntity.ok("Solution not found!");
        }
        return ResponseEntity.ok( new ResponsePayload(
            resultBoards,
            solver.executionTime,
            solver.moveCount
        ));
    }
}
```

b. Application (Main Class)

```java
// BackendApplication.java
package com.tucil3.backend;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication
public class BackendApplication {
    public static void main(String[] args) {
        SpringApplication.run(BackendApplication.class, args);
    }
}
```

BAB 7

LAMPIRAN

Repositori GitHub : https://github.com/Narrr21/Tucil3_13523076_13523083

Tabel Spesifikasi

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	☑	☐