Laporan Tugas Kecil 2 IF2211 Strategi Algoritma

Sem II tahun 2024/2025

Kompresi Gambar Dengan Metode Quadtree

Disusun oleh :

Fityatul Haq Rosyidi - 13523116

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

2025

Daftar Isi

BAB I : Deskripsi Tugas

Gambar 2. RushHour Puzzle

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

Rush Hour adalah sebuah permainan puzzle logika berbasis grid yang menantang pemain untuk menggeser kendaraan di dalam sebuah kotak (biasanya berukuran 6x6) agar mobil utama (biasanya berwarna merah) dapat keluar dari kemacetan melalui pintu keluar di sisi papan. Setiap kendaraan hanya bisa bergerak lurus ke depan atau ke belakang sesuai dengan orientasinya (horizontal atau vertikal), dan tidak dapat berputar. Tujuan utama dari permainan ini adalah memindahkan mobil merah ke pintu keluar dengan jumlah langkah seminimal mungkin.

Komponen penting dari permainan Rush Hour terdiri dari:

1. **Papan** – *Papan* merupakan tempat permainan dimainkan.
   Papan terdiri atas *cell*, yaitu sebuah *singular point* dari papan. Sebuah *piece* akan menempati *cell-cell* pada papan. Ketika permainan dimulai, semua *piece* telah diletakkan di dalam papan dengan konfigurasi tertentu berupa lokasi piece dan *orientasi*, antara *horizontal* atau *vertikal*.

Hanya *primary piece* yang dapat digerakkan keluar papan melewati *pintu keluar*. *Piece* yang bukan *primary piece* tidak dapat digerakkan keluar papan. Papan memiliki satu *pintu keluar* yang pasti berada di *dinding papan* dan sejajar dengan orientasi *primary piece*.

2. **Piece –** *Piece* adalah sebuah kendaraan di dalam papan. Setiap *piece* memiliki *posisi*, *ukuran*, dan *orientasi*. *Orientasi* sebuah *piece* hanya dapat berupa horizontal atau vertikal–tidak mungkin diagonal. *Piece* dapat memiliki beragam *ukuran*, yaitu jumlah *cell* yang ditempati oleh *piece*. Secara standar, variasi *ukuran* sebuah *piece* adalah *2-piece* (menempati 2 *cell*) atau *3-piece* (menempati 3 *cell*). Suatu *piece* tidak dapat digerakkan melewati/menembus *piece* yang lain.

3. **Primary Piece –** *Primary piece* adalah kendaraan utama yang harus dikeluarkan dari *papan* (biasanya berwarna merah). Hanya boleh terdapat satu primary piece.

4. **Pintu Keluar –** *Pintu keluar* adalah tempat *primary piece* dapat digerakkan keluar untuk menyelesaikan permainan

5. **Gerakan** — *Gerakan* yang dimaksudkan adalah pergeseran *piece* di dalam permainan. *Piece* hanya dapat bergerak/bergeser lurus sesuai orientasinya (atas-bawah jika vertikal dan kiri-kanan jika horizontal). Suatu *piece* tidak dapat digerakkan melewati/menembus *piece* yang lain.

BAB II : Rancangan Algoritma

Materi pokok pada tugas kecil 3 ini adalah pathfinding, jadi algoritma-algoritma yang digunakan tentunya adalah algoritma-algoritma pathfinding yang sudah dikenal luas dalam dunia Computer Science. Algoritma tersebut ialah A* Algorithm, UCS Algorithm, dan Greedy Best-First Search Algorithm. Algoritma PathFinding ini digunakan untuk mencari jalan terpendek menuju state solusi (goal node).

Struktur data board direpresentasikan sebagai grid matriks berukuran m kali n. Kendaraan direpresentasikan sebagai char yang berada di grid. Kendaraan Utama yang menjadi objektif game ini dilambangkan dengan karakter “P” dan grid kosong berisi karakter titik. Pintu keluar direpresentasikan dengan suatu koordinat. State Solusi adalah State Board dimana koordinat Door berisi Primary Vehicle yaitu P. Tujuan dari game adalah mencapai State Solusi. Berikut adalah Langkah-langkah untuk masing-masing Algoritma :

A* Algorithm

1. Dibentuk PrioQueue yang bertugas menyimpan state
2. State pertama adalah state yang dibaca dari file konfigurasi, masukkan ke queue
3. Ambil elemen pertama queue (dequeue)
4. Enumerasi state-state yang mungkin dari state yang barusan di dequeue
5. Untuk setiap state, dihitung cost nya dari penjumlahan g(n) dan h(n)
6. g(n) adalah jarak dari state root ke state saat ini
7. h(n) adalah fungsi heuristic yang menilai kelayakan suatu state
8. Masukaan state dengan cost tertinggi ke Queue, buang yang lainnya
9. Ulangi dari langkah ke-4, Lakukan hingga mencapai state solusi.

UCS Algorithm

1. Dibentuk PrioQueue yang bertugas menyimpan state
2. State pertama adalah state yang dibaca dari file konfigurasi, masukkan ke queue
3. Ambil elemen pertama queue (dequeue)
4. Enumerasi state-state yang mungkin dari state yang barusan di dequeue
5. Untuk setiap state, dihitung cost nya yaitu g(n)
6. g(n) adalah jarak dari state root ke state saat ini
7. Masukkan state-state tersebut ke PrioQueue dan diurut berdasarkan Cost Terbesar
8. Ulangi dari langkah ke-4, Lakukan hingga state solusi ditemukan.

Greedy Best-First Search Algorithm

1. Dibentuk PrioQueue yang bertugas menyimpan state
2. State pertama adalah state yang dibaca dari file konfigurasi, masukkan ke queue
3. Ambil elemen pertama queue (dequeue)
4. Enumerasi state-state yang mungkin dari state yang barusan di dequeue
5. Untuk setiap state, dihitung cost nya yaitu h(n)
6. h(n) adalah fungsi heuristic yang menilai kelayakan suatu state
7. Masukaan state dengan cost tertinggi ke Queue, buang yang lainnya
8. Ulangi dari langkah ke-4, Lakukan hingga state solusi ditemukan.

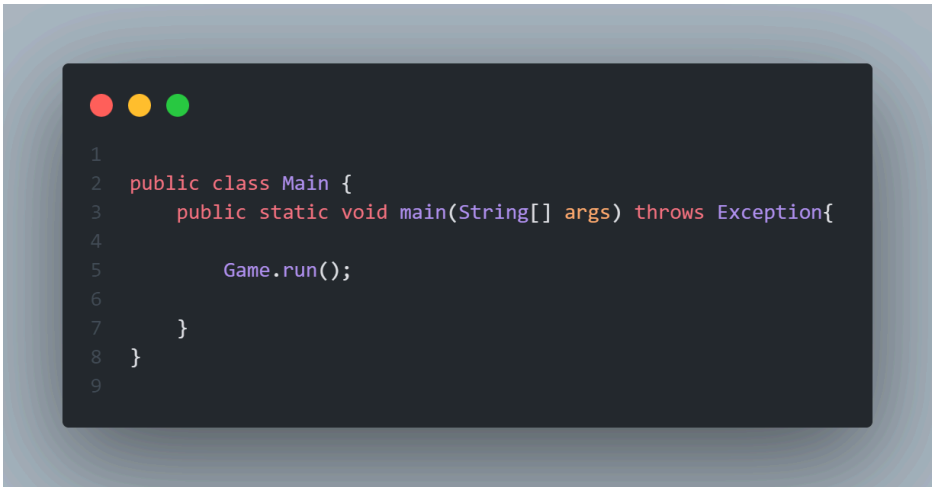
Secara umum, ketiga Algoritma diatas mirip. Perbedaannya hanya terdapat di perhitungan cost. Untuk A*, cost dihitung dengan menjumlahkan g(n) dan h(n), sedangkan UCS cukup g(n) saja, dan Greedy BFS cukup h(n) saja. Yang menjadi pembeda lain adalah UCS meng-extend semua state pada kedalaman saat ini (mirip BFS), sedangkan dua algoritma lainnya hanya meng-extend satu state dengan cost terbaik.

BAB III : Analisis Algoritma

Ditinjau dari 3 Algoritma yang digunakan, Algoritma terbaik adalah A* karena menggunakan heuristic dan fungsi cost. Greedy BFS tidak menjamin menghasilkan solusi yang optimal karena sifat Greedy nya. UCS pada persoalan ini mirip dengan algoritma BFS karena nilai g(n) yang dihasilkan sama untuk setiap node pada suatu level. Secara Teoritis, Algoritma A* lebih efisien daripada UCS karena A* juga menggunakan fungsi heuristic.

BAB IV : Implementasi

Snapshot Code

Nama File	Snapshot
main.java	

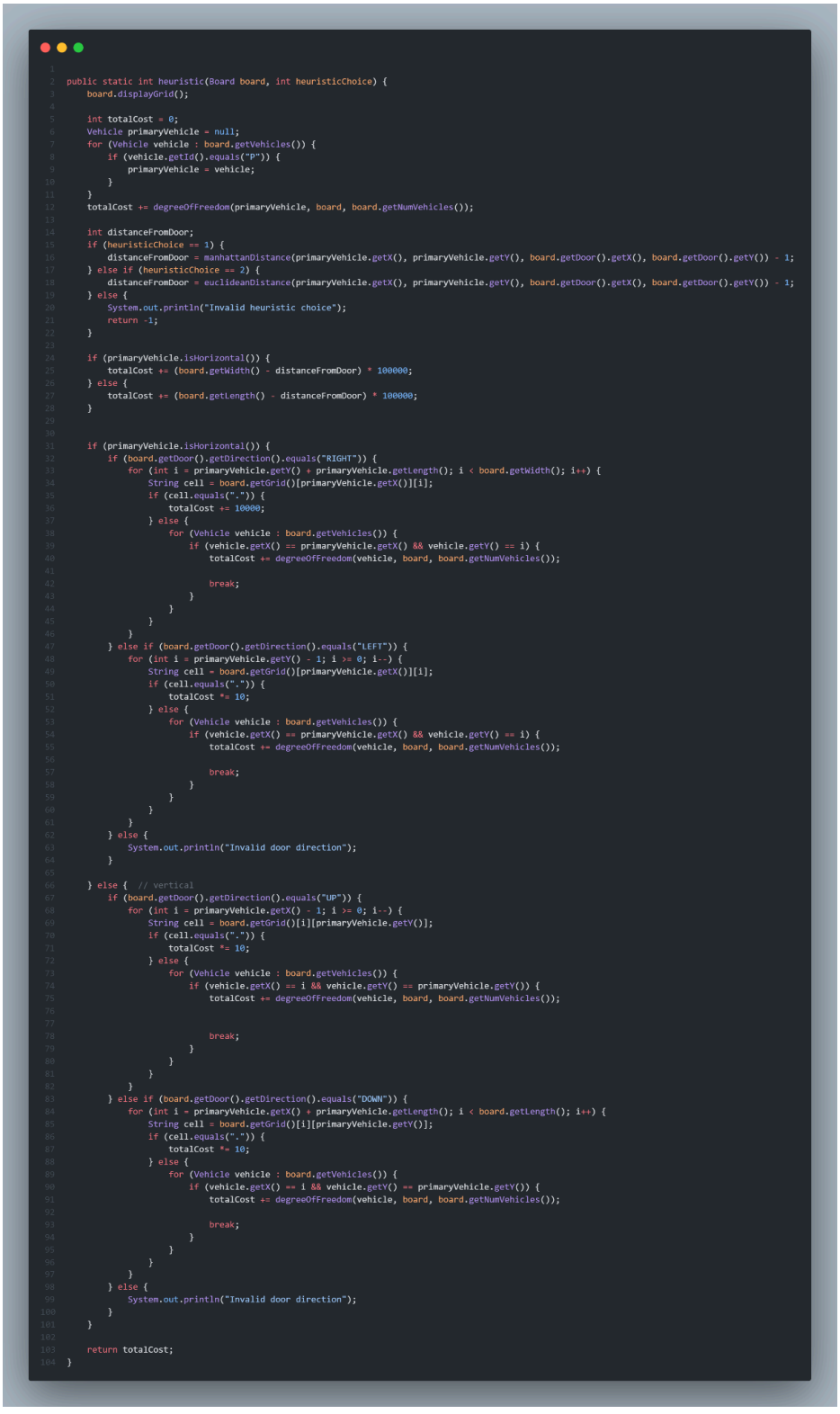
```
1
2  public class Main {
3      public static void main(String[] args) throws Exception{
4
5          Game.run();
6
7      }
8  }
9
```

Game.java

```java
import java.util.Scanner;
import java.io.FileNotFoundException;
import java.io.IOException;


public class Game {
    public static void run() throws Exception {
        while (true) {

            try {

                System.out.println("Welcome to the RushHour Solver!");
                System.out.println("Input path to the file containing the RushHour puzzle (e.g. p1.txt):");
                Scanner scanner = new Scanner(System.in);
                String filePath = scanner.nextLine();

                Board board = Loader.loadBoardFromFile("../test/" + filePath);
                Door door = Loader.loadDoorFromFile("../test/" + filePath);
                board.setDoor(door);

                System.out.println("Input Algorithm:");
                System.out.println("1. A star");
                System.out.println("2. UCS");
                System.out.println("3. Greedy BFS");
                int choice = scanner.nextInt();

                System.out.println("Input Heuristic:");
                System.out.println("1. Manhattan Distance");
                System.out.println("2. Euclidean Distance");
                int heuristicChoice = scanner.nextInt();

                switch (choice) {
                    case 1:
                        Solver.solveWithAStar(board, heuristicChoice);
                        break;
                    case 2:
                        Solver.solveWithUCS(board);
                        break;
                    case 3:
                        Solver.solveWithGreedyBFS(board, heuristicChoice);
                        break;
                    default:
                        System.out.println("Invalid choice. Please select a valid algorithm.");
                }

                System.out.println("Do you want to solve another puzzle? (0.no/ any.yes)");
                int continueChoice = scanner.nextInt();
                if (continueChoice == 0) {
                    System.out.println("Thank you for playing!");
                    scanner.close();
                    break;
                }

            } catch (IllegalArgumentException e) {
                System.out.println("Invalid input: " + e.getMessage());
                System.out.println("Please try again.");
            } catch (IndexOutOfBoundsException e) {
                System.out.println("Invalid index: " + e.getMessage());
                System.out.println("Please check the input file.");
            } catch (ClassCastException e) {
                System.out.println("Invalid class cast: " + e.getMessage());
                System.out.println("Please check the input file.");
            } catch (FileNotFoundException e) {
                System.out.println("File not found: " + e.getMessage());
                System.out.println("Please check the file path.");
            } catch (IOException e) {
                System.out.println("IO exception: " + e.getMessage());
                System.out.println("Please check the input file.");
            } catch (NullPointerException e) {
                System.out.println("Null pointer exception : " + e.getMessage());
                System.out.println("Please check the input file.");
            } catch (Exception e) {
                System.out.println("exception: " + e);
                System.out.println("An error occurred: " + e.getMessage());
                System.out.println("Please try again.");
            }

        }


    }
}

```

Solver.java

```java
public class Solver {
    public static int degreeOfFreedom(Vehicle vehicle, Board board, int count) {
        if (count == 0) {
            return 0;
        }

        int degree = 0;
        int currentX = vehicle.getX();
        int currentY = vehicle.getY();
        Vehicle obsVehicle1 = null;
        Vehicle obsVehicle2 = null;
        if (vehicle.isHorizontal()) {
            for (int y = currentY - 1; y >= 0; y--) {
                String cell = board.getGrid()[currentX][y];
                if (cell.equals(".")) {
                    degree++;
                } else {
                    obsVehicle1 = Finder.findVehicleWithId(board.getVehicles(), cell);
                    if (obsVehicle1 != null) {
                        break;
                    }
                }
            }
            for (int y = currentY + vehicle.getLength(); y < board.getWidth(); y++) {
                String cell = board.getGrid()[currentX][y];
                if (cell.equals(".")) {
                    degree++;
                } else {
                    obsVehicle2 = Finder.findVehicleWithId(board.getVehicles(), cell);
                    if (obsVehicle2 != null) {
                        break;
                    }
                }
            }
        } else {
            for (int x = currentX - 1; x >= 0; x++) {
                String cell = board.getGrid()[x][currentY];
                if (cell.equals(".")) {
                    degree++;
                } else {
                    obsVehicle1 = Finder.findVehicleWithId(board.getVehicles(), cell);
                    if (obsVehicle1 != null) {
                        break;
                    }
                }
            }
            for (int x = currentY + vehicle.getLength(); x < board.getLength(); x++) {
                String cell = board.getGrid()[x][currentY];
                if (cell.equals(".")) {
                    degree++;
                } else {
                    obsVehicle2 = Finder.findVehicleWithId(board.getVehicles(), cell);
                    if (obsVehicle2 != null) {
                        break;
                    }
                }
            }
        }
        degree *= count;
        if (obsVehicle1 != null) {
            degree += degreeOfFreedom(obsVehicle1, board, count - 1);
        }
        if (obsVehicle2 != null) {
            degree += degreeOfFreedom(obsVehicle2, board, count - 1);
        }

        return degree;
    }


```



```java
public static int heuristic(Board board, int heuristicChoice) {
    board.displayGrid();

    int totalCost = 0;
    Vehicle primaryVehicle = null;
    for (Vehicle vehicle : board.getVehicles()) {
        if (vehicle.getId().equals("P")) {
            primaryVehicle = vehicle;
        }
    }
    totalCost += degreeOfFreedom(primaryVehicle, board, board.getNumVehicles());

    int distanceFromDoor;
    if (heuristicChoice == 1) {
        distanceFromDoor = manhattanDistance(primaryVehicle.getX(), primaryVehicle.getY(), board.getDoor().getX(), board.getDoor().getY()) - 1;
    } else if (heuristicChoice == 2) {
        distanceFromDoor = euclideanDistance(primaryVehicle.getX(), primaryVehicle.getY(), board.getDoor().getX(), board.getDoor().getY()) - 1;
    } else {
        System.out.println("Invalid heuristic choice");
        return -1;
    }

    if (primaryVehicle.isHorizontal()) {
        totalCost += (board.getWidth() - distanceFromDoor) * 100000;
    } else {
        totalCost += (board.getLength() - distanceFromDoor) * 100000;
    }


    if (primaryVehicle.isHorizontal()) {
        if (board.getDoor().getDirection().equals("RIGHT")) {
            for (int i = primaryVehicle.getY() + primaryVehicle.getLength(); i < board.getWidth(); i++) {
                String cell = board.getGrid()[primaryVehicle.getX()][i];
                if (cell.equals(".")) {
                    totalCost += 10000;
                } else {
                    for (Vehicle vehicle : board.getVehicles()) {
                        if (vehicle.getX() == primaryVehicle.getX() && vehicle.getY() == i) {
                            totalCost += degreeOfFreedom(vehicle, board, board.getNumVehicles());

                            break;
                        }
                    }
                }
            }
        } else if (board.getDoor().getDirection().equals("LEFT")) {
            for (int i = primaryVehicle.getY() - 1; i >= 0; i--) {
                String cell = board.getGrid()[primaryVehicle.getX()][i];
                if (cell.equals(".")) {
                    totalCost *= 10;
                } else {
                    for (Vehicle vehicle : board.getVehicles()) {
                        if (vehicle.getX() == primaryVehicle.getX() && vehicle.getY() == i) {
                            totalCost += degreeOfFreedom(vehicle, board, board.getNumVehicles());

                            break;
                        }
                    }
                }
            }
        } else {
            System.out.println("Invalid door direction");
        }

    } else {  // vertical
        if (board.getDoor().getDirection().equals("UP")) {
            for (int i = primaryVehicle.getX() - 1; i >= 0; i--) {
                String cell = board.getGrid()[i][primaryVehicle.getY()];
                if (cell.equals(".")) {
                    totalCost *= 10;
                } else {
                    for (Vehicle vehicle : board.getVehicles()) {
                        if (vehicle.getX() == i && vehicle.getY() == primaryVehicle.getY()) {
                            totalCost += degreeOfFreedom(vehicle, board, board.getNumVehicles());


                            break;
                        }
                    }
                }
            }
        } else if (board.getDoor().getDirection().equals("DOWN")) {
            for (int i = primaryVehicle.getX() + primaryVehicle.getLength(); i < board.getLength(); i++) {
                String cell = board.getGrid()[i][primaryVehicle.getY()];
                if (cell.equals(".")) {
                    totalCost *= 10;
                } else {
                    for (Vehicle vehicle : board.getVehicles()) {
                        if (vehicle.getX() == i && vehicle.getY() == primaryVehicle.getY()) {
                            totalCost += degreeOfFreedom(vehicle, board, board.getNumVehicles());

                            break;
                        }
                    }
                }
            }
        } else {
            System.out.println("Invalid door direction");
        }
    }

    return totalCost;
}
```

```java

public static void solveWithAStar(Board board, int heuristicChoice) {
    System.out.println("Solving with A* algorithm...");

    int h = heuristic(board, heuristicChoice);
    int nodeCount = 0;
    int cost = 0;

    ArrayList<Vehicle> visited = new ArrayList<>();
    PriorityQueue<State> pq = new PriorityQueue<>(Comparator.reverseOrder());
    PriorityQueue<State> tempPQ = new PriorityQueue<>(Comparator.reverseOrder());
    pq.add(new State(board, 0 + h));
    while (!pq.isEmpty()) {
        nodeCount++;
        cost ++;

        if (nodeCount > 100) {
            System.out.println("Depth limit reached. No solution found.");
            return;
        }

        State currentState = pq.poll();
        Board currentBoard = currentState.getBoard();
        System.out.println("Step " + (nodeCount - 1) + ":");

        currentBoard.displayGrid();

        if (currentBoard.solved()) {
            System.out.println("Solution found!");
            System.out.println("Node Visited: " + nodeCount);
            return;
        }


        // Generate new states by moving the vehicle
        List<Board> newStates = currentBoard.getPossibleState();
        for (Board newState : newStates) {
            int newH = heuristic(newState, heuristicChoice);
            tempPQ.add(new State(newState, cost + 1 + newH));
        }
        while(!tempPQ.isEmpty()) {
            State tempState = tempPQ.poll();
            if (!visited.contains(tempState.getBoard().getMovedVehicle())) {
                visited.add(tempState.getBoard().getMovedVehicle());
                pq.add(tempState);
                break;
            }
        }
        tempPQ.clear();
    }
}

```

```java

public static void solveWithUCS(Board board) {
    System.out.println("Solving with UCS algorithm...");

    int g = 0;
    int nodeCount = 0;

    ArrayList<Vehicle> visited = new ArrayList<>();
    PriorityQueue<State> pq = new PriorityQueue<>(Comparator.reverseOrder());
    pq.add(new State(board, g));
    while (!pq.isEmpty()) {
        nodeCount++;

        State currentState = pq.poll();
        Board currentBoard = currentState.getBoard();
        int cost = currentState.getCost();
        if (cost > 100) {
            System.out.println("Depth limit reached. No solution found.");
            return;
        }
        System.out.println("Step " + (nodeCount - 1) + ":");
        currentBoard.displayGrid();

        if (currentBoard.solved()) {
            System.out.println("Solution found!");
            return;
        }

        if (visited.contains(currentBoard.getMovedVehicle())) {
            continue;
        } else {
            visited.add(currentBoard.getMovedVehicle());
        }

        // Generate new states by moving the vehicle
        List<Board> newStates = currentBoard.getPossibleState();
        for (Board newState : newStates) {
            pq.add(new State(newState, cost + 1));
        }
    }
}

public static void solveWithGreedyBFS(Board board, int heuristicChoice) {
    System.out.println("Solving with Greedy Best-First Search algorithm...");

    int h = heuristic(board, heuristicChoice);
    int depth = 0;
    int nodeCount = 0;

    ArrayList<Vehicle> visited = new ArrayList<>();
    PriorityQueue<State> pq = new PriorityQueue<>(Comparator.reverseOrder());
    PriorityQueue<State> tempPQ = new PriorityQueue<>(Comparator.reverseOrder());
    pq.add(new State(board, h));
    while (!pq.isEmpty(
```

```java

    public static void solveWithGreedyBFS(Board board, int heuristicChoice) {
        System.out.println("Solving with Greedy Best-First Search algorithm...");

        int h = heuristic(board, heuristicChoice);
        int depth = 0;
        int nodeCount = 0;

        ArrayList<Vehicle> visited = new ArrayList<>();
        PriorityQueue<State> pq = new PriorityQueue<>(Comparator.reverseOrder());
        PriorityQueue<State> tempPQ = new PriorityQueue<>(Comparator.reverseOrder());
        pq.add(new State(board, h));
        while (!pq.isEmpty()) {
            nodeCount++;

            if (depth > 100) {
                System.out.println("Depth limit reached. No solution found.");
                return;
            }

            State currentState = pq.poll();
            Board currentBoard = currentState.getBoard();
            System.out.println("Step " + (depth - 1) + ":");
            currentBoard.displayGrid();

            if (currentBoard.solved()) {
                System.out.println("Solution found!");
                return;
            }


            depth++;

            // Generate new states by moving the vehicle
            List<Board> newStates = currentBoard.getPossibleState();
            for (Board newState : newStates) {
                int newH = heuristic(newState, heuristicChoice);
                tempPQ.add(new State(newState, newH));
            }
            while(!tempPQ.isEmpty()) {
                State tempState = tempPQ.poll();
                if (!visited.contains(tempState.getBoard().getMovedVehicle())) {
                    visited.add(tempState.getBoard().getMovedVehicle());
                    pq.add(tempState);
                    break;
                }
            }
            tempPQ.clear();
        }
    }

    public static int manhattanDistance(int x1, int y1, int x2, int y2) {
        return Math.abs(x1 - x2) + Math.abs(y1 - y2);
    }

    public static int euclideanDistance(int x1, int y1, int x2, int y2) {
        return (int) Math.sqrt(Math.pow(x1 - x2, 2) + Math.pow(y1 - y2, 2));
    }


}


```

Board.java

```java
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Objects;

public class Board {
    private int length;
    private int width;
    private ArrayList<Vehicle> vehicles;
    private int numVehicles;
    private String[][] grid;
    private Door door;
    private Vehicle movedVehicle;

    public Board(int length, int width, int numVehicles, ArrayList<Vehicle> vehicles, Door door, Vehicle movedVehicle) {
        this.length = length;
        this.width = width;
        this.vehicles = vehicles;
        this.numVehicles = numVehicles;
        this.grid = new String[length][width];
        this.door = door;
        this.movedVehicle = movedVehicle;

        updateGrid();
    }

    // getters
    public int getLength() {
        return length;
    }

    public int getWidth() {
        return width;
    }

    public ArrayList<Vehicle> getVehicles() {
        return vehicles;
    }

    public int getNumVehicles() {
        return numVehicles;
    }

    public String[][] getGrid() {
        return grid;
    }

    public Door getDoor() {
        return door;
    }

    public Vehicle getMovedVehicle() {
        return movedVehicle;
    }


    // setters
    public void setLength(int length) {
        this.length = length;
    }

    public void setWidth(int width) {
        this.width = width;
    }

    public void setDoor(Door door) {
        this.door = door;
    }

    public void setMovedVehicle(Vehicle movedVehicle) {
        this.movedVehicle = movedVehicle;
    }

```

```java

// other services
public boolean solved() {
    return this.grid[door.getX()][door.getY()].equals("P");
}


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

    Board other = (Board) obj;

    // Bandingkan ukuran grid dulu (optional tapi efisien)
    if (this.length != other.length || this.width != other.width) return false;

    // Bandingkan isi grid menggunakan Arrays.deepEquals
    return Arrays.deepEquals(this.grid, other.grid);
}

@Override
public int hashCode() {
    // Gunakan Arrays.deepHashCode untuk array 2D
    int result = Objects.hash(length, width);
    result = 31 * result + Arrays.deepHashCode(grid);
    return result;
}

public void addVehicle(Vehicle vehicle) {
    vehicles.add(vehicle);
    numVehicles++;
    updateGrid();
}

public void updateGrid() {
    for (int i = 0; i < length; i++) {
        for (int j = 0; j < width; j++) {
            this.grid[i][j] = ".";
        }
    }
    for (Vehicle vehicle : vehicles) {
        int x = vehicle.getX();
        int y = vehicle.getY();
        String orientation = vehicle.getOrientation();
        int length = vehicle.getLength();

        if (orientation.equals("H")) {
            for (int j = 0; j < length; j++) {
                this.grid[x][y + j] = vehicle.getId();
            }
        } else if (orientation.equals("V")) {
            for (int i = 0; i < length; i++) {
                this.grid[x + i][y] = vehicle.getId();
            }
        }
    }
}
```

```java

    public void displayGrid() {
        System.out.println("-".repeat(length * 2 + 2));
        for (int i = 0; i < length; i++) {
            System.out.print("|");
            for (int j = 0; j < width; j++) {
                String cell = grid[i][j];
                if (movedVehicle != null && cell.equals(movedVehicle.getId())) {
                    System.out.print("\u001B[33m" + cell + "\u001B[0m" + " ");
                } else {
                    System.out.print(cell + " ");
                }
            }
            System.out.println("|");
        }
        System.out.println("-".repeat(length * 2 + 2));
    }

    public List<Board> getPossibleState() {
        List<Board> states = new ArrayList<>();
        for (Vehicle vehicle : vehicles) {
            int x = vehicle.getX();
            int y = vehicle.getY();
            String orientation = vehicle.getOrientation();
            int vehicleLength = vehicle.getLength();

            // Try moving the vehicle left or up
            if (orientation.equals("H")) {
                if (y - 1 >= 0 && grid[x][y - 1] == ".") {
                    Vehicle newVehicle = new Vehicle(vehicle.getId(), x, y - 1, orientation, vehicleLength);
                    states.add(createNewBoardState(vehicle, newVehicle));
                }
                if (y + vehicleLength < width && grid[x][y + vehicleLength] == ".") {
                    Vehicle newVehicle = new Vehicle(vehicle.getId(), x, y + 1, orientation, vehicleLength);
                    states.add(createNewBoardState(vehicle, newVehicle));
                }
            } else if (orientation.equals("V")) {
                if (x - 1 >= 0 && grid[x - 1][y] == ".") {
                    Vehicle newVehicle = new Vehicle(vehicle.getId(), x - 1, y, orientation, vehicleLength);
                    states.add(createNewBoardState(vehicle, newVehicle));
                }
                if (x + vehicleLength < length && grid[x + vehicleLength][y] == ".") {
                    Vehicle newVehicle = new Vehicle(vehicle.getId(), x + 1, y, orientation, vehicleLength);
                    states.add(createNewBoardState(vehicle, newVehicle));
                }
            }
        }
        return states;
    }

    private Board createNewBoardState(Vehicle oldVehicle, Vehicle newVehicle) {
        ArrayList<Vehicle> newVehicles = new ArrayList<>(vehicles);
        newVehicles.remove(oldVehicle);
        newVehicles.add(newVehicle);
        return new Board(length, width, newVehicles.size(), newVehicles, door, newVehicle);
    }



}

```

Loader.java	

```java
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;

public class Loader {
    public static Board loadBoardFromFile(String filePath) throws IOException {
        BufferedReader reader = new BufferedReader(new FileReader(filePath));

        // Membaca ukuran papan
        String[] dimensions = reader.readLine().split(" ");
        int length = Integer.parseInt(dimensions[0]);
        int width = Integer.parseInt(dimensions[1]);

        // Membaca jumlah kendaraan (tidak perlu digunakan langsung)
        int numVehicles = Integer.parseInt(reader.readLine());

        String[][] grid = new String[length][width];
        ArrayList<Vehicle> vehicles = new ArrayList<>();
        Door door = null;

        // Read grid configuration
        for (int i = 0; i < length; i++) {
            String line = reader.readLine();
            if (line == null) {
                throw new IOException("Unexpected end of file while reading grid.");
            }
            int colIndex = 0;
            for (int j = 0; j < width; j++) {
                String c = String.valueOf(line.charAt(j));
                if (c.equals("K")) {
                    if (i == 0) {
                        i--;
                        break;
                    } else {
                        continue;
                    }
                }
                if (!c.equals(" ")) { // Skip spaces
                    grid[i][colIndex] = c;
                    colIndex++;
                }
            }
        }
```

```
        // Menambahkan kendaraan ke papan berdasarkan konfigurasi
        for (char id = 'A'; id <= 'Z'; id++) {
            int xStart = -1, yStart = -1, xEnd = -1, yEnd = -1;

            // Mencari posisi kendaraan
            for (int i = 0; i < length; i++) {
                for (int j = 0; j < width; j++) {
                    if (grid[i][j].equals(String.valueOf(id))) {
                        if (xStart == -1) {
                            xStart = i;
                            yStart = j;
                        }
                        xEnd = i;
                        yEnd = j;
                    }
                }
            }

            // Jika kendaraan ditemukan, tambahkan ke papan
            if (xStart != -1) {
                int vehicleLength = Math.max(xEnd - xStart + 1, yEnd - yStart + 1);
                String orientation = (xStart == xEnd) ? "H" : "V";
                vehicles.add(new Vehicle(String.valueOf(id), xStart, yStart, orientation, vehicleLength));
            }
        }

        reader.close();
        return new Board(length, width, numVehicles, vehicles, door, null) ;
    }

    public static Door loadDoorFromFile(String filePath) throws IOException {
        BufferedReader reader = new BufferedReader(new FileReader(filePath));
        // Membaca ukuran papan
        String[] dimensions = reader.readLine().split(" ");
        int length = Integer.parseInt(dimensions[0]);
        int width = Integer.parseInt(dimensions[1]);

        // Membaca ukuran papan
        reader.readLine();

        Door door = null;
        int row = 0;

        // Membaca hingga akhir file
        String line;
        while ((line = reader.readLine()) != null) {
            for (int col = 0; col < line.length(); col++) {
                String c = String.valueOf(line.charAt(col));
                if (c.equals("K")) {
                    if (row == 0) {
                        door = new Door("UP", 0, col);
                    } else if (row == length) {
                        door = new Door("DOWN", length - 1, col);
                    } else if (col == 0) {
                        door = new Door("LEFT", row, 0);
                    } else if (col == width) {
                        door = new Door("RIGHT", row, width - 1);
                    }
                    break; // Tidak perlu melanjutkan pencarian setelah pintu ditemukan
                }
            }
            if (door != null) {
                break;
            }
            row++;
        }

        reader.close();

        if (door == null) {
            throw new IOException("No door (K) found in the configuration.");
        }

        return door;
    }
}
```

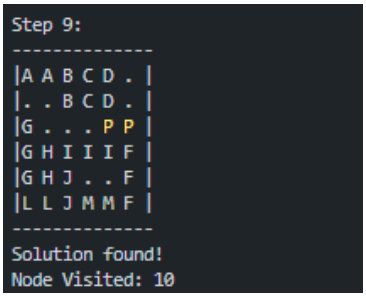
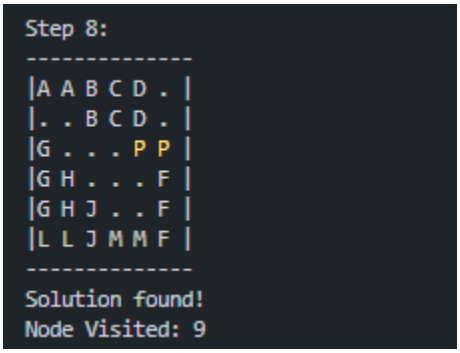
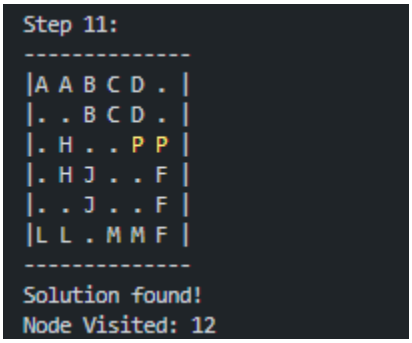
Vehicle.java	

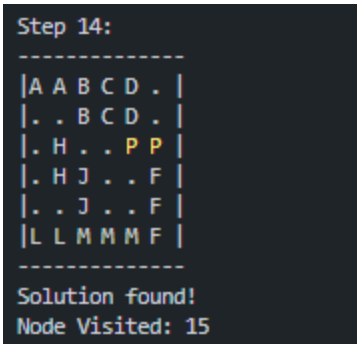
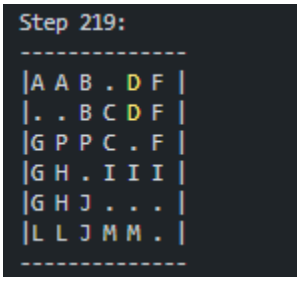
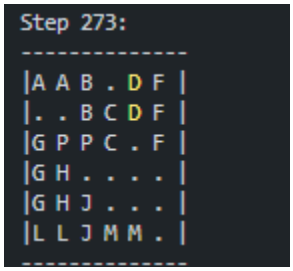
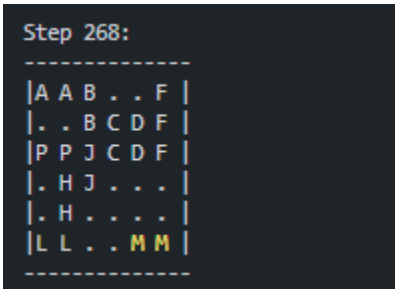
```java
public class Vehicle {
    private String id;
    private int x;
    private int y;
    private String orientation;
    private int length;

    public Vehicle(String id, int x, int y, String orientation, int length) {
        this.id = id;
        this.x = x;
        this.y = y;
        this.orientation = orientation;
        this.length = length;
    }

    // getters
    public String getId() {
        return id;
    }

    public int getX() {
        return x;
    }

    public int getY() {
        return y;
    }

    public String getOrientation() {
        return orientation;
    }

    public int getLength() {
        return length;
    }

    // setters
    public void setId(String id) {
        this.id = id;
    }

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

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

    public void setOrientation(String orientation) {
        this.orientation = orientation;
    }

    public void setLength(int length) {
        this.length = length;
    }
```

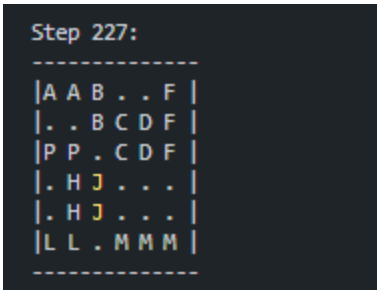
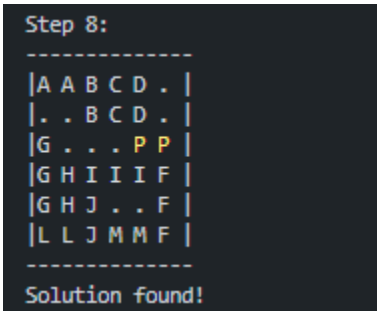
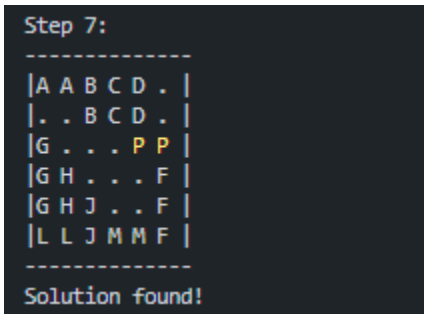
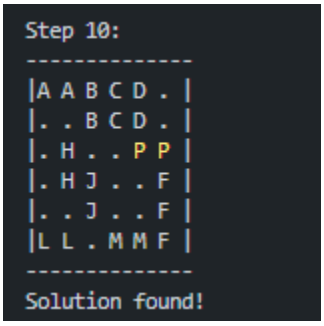
```java

    // other services
    public String toString() {
        return "Vehicle{" +
                "id='" + id + '\'' +
                ", x=" + x +
                ", y=" + y +
                ", orientation='" + orientation + '\'' +
                ", length=" + length +
                '}';
    }

    public boolean equals(Object o) {
        if (this == o) return true;
        if (!(o instanceof Vehicle)) return false;

        Vehicle vehicle = (Vehicle) o;

        if (x != vehicle.x) return false;
        if (y != vehicle.y) return false;
        if (length != vehicle.length) return false;
        if (id != vehicle.id) return false;
        return orientation.equals(vehicle.orientation);
    }

    public int hashCode() {
        int result = id.hashCode();
        result = 31 * result + x;
        result = 31 * result + y;
        result = 31 * result + orientation.hashCode();
        result = 31 * result + length;
        return result;
    }

    public Vehicle clone() {
        return new Vehicle(id, x, y, orientation, length);
    }

    public boolean isHorizontal() {
        return orientation.equals("H");
    }

    public boolean isVertical() {
        return orientation.equals("V");
    }

    public void giveColor() {
        String yellow = "\u001B[33m"; // ANSI kode warna kuning
        String reset = "\u001B[0m";   // ANSI kode reset warna
        this.id = yellow + id + reset; // Modifikasi id dengan warna kuning
    }

}
```

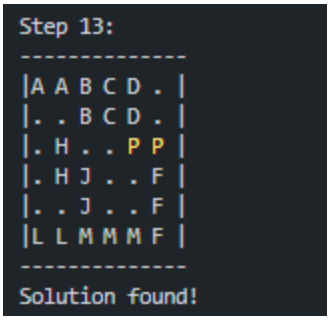
BAB V : Testing

Note : isi file konfigurasi dapat dilihat di repo github (di dalam folder test)

No	Deskripsi Kasus	Output Terminal
1	Konfigurasi 1 - A*	
2	Konfigurasi 2 - A*	
3	Konfigurasi 3 - A*	

4	Konfigurasi 4 - A*	
5	Konfigurasi 1 - UCS	 (unsolved)
6	Konfigurasi 2 - UCS	 (unsolved)
7	Konfigurasi 3 - UCS	 (unsolved)

8	Konfigurasi 4 - UCS	 (unsolved)
9	Konfigurasi 1 - Greedy BFS	
10	Konfigurasi 2 - Greedy BFS	
11	Konfigurasi 3 - Greedy BFS	

12	Konfigurasi 4 - Greedy BFS	

BAB VI : Lampiran

No	Poin	Ya	Tidak
1	1. Program berhasil dikompilasi tanpa kesalahan	✔	
2	1. Program berhasil dijalankan	✔	
3	3. Solusi yang diberikan program benar dan mematuhi aturan permainan	✔	
4	4. Program dapat membaca masukan berkas .txt dan menyimpan solusi berupa print board tahap per tahap dalam berkas .txt		✔
5	4. [Bonus] Implementasi algoritma pathfinding alternatif		✔
6	4. [Bonus] Implementasi 2 atau lebih heuristik alternatif	✔	
7	4. [Bonus] Program memiliki GUI		✔
8	4. Program dan laporan dibuat (kelompok) sendiri	✔	

~ Pranala Github : FityatulhaqRosyidi/Tucil3_13523116: Tugas Kecil 3 Strategi Algoritma - Penyelesaian Puzzle Rush Hour Menggunakan Algoritma Pathfinding

Referensi

A* Search Algorithm | GeeksforGeeks