

Traffic management system:

|  |
| --- |
| Submitted by:  Rukhsar Fatima (68481)  Farzand Ali (69702)  Fawad Ahmed(68671)  Course:  Data Structure (DS)  Submitted To:  Israr Ali  Date of submission:  30-June-2025 |

Project proposal:

Objective:

Design and implement a Traffic Management System using JavaFX that simulates and visualizes traffic scenarios, with a focus on improving vehicle flow and reducing congestion.

Scope:

* Simulate vehicle movements, traffic signals, and congestion levels.
* Create an interactive graphical interface for real-time visualization.
* Utilize Dijkstra's algorithm for pathfinding based on traffic congestion.

Utilization:

* Apply object-oriented programming concepts.
* Use JavaFX for building the graphical user interface.
* Leverage data structures such as graphs and queues to manage traffic flow efficiently.

Design document:

Uml diagram:

***1. Class Diagram:***

* **Core Components:** Includes classes such as Graph, Road, Node, Lane, TrafficLight, TrafficVisualizer, and Vehicle.
* **Relationships:** The Graph structure links roads and intersections, while lanes manage the flow of vehicles through queues.

***2. Sequence Diagram:***

* **Traffic Simulation Flow:** The simulation starts with lane updating vehicle queues → traffic light status changes → the system computes the shortest path → the GUI refreshes to reflect the updated traffic state.

Code base:

VEHICLE CLASS:

**package** application;

**public** **class** Vehicle {

**private** **final** String name;

**private** **final** **boolean** isEmergency;

**public** Vehicle(String name, **boolean** isEmergency) {

**this**.name = name;

**this**.isEmergency = isEmergency;

}

**public** String getName() {

**return** name;

}

**public** **boolean** isEmergency() {

**return** isEmergency;

}

}

LANE CLASS:

**package** application;

**import** java.util.LinkedList;

**import** java.util.Queue;

**public** **class** Lane {

**private** **final** Queue<Vehicle> vehicles = **new** LinkedList<>();

**public** **void** addVehicle(String name, **boolean** isEmergency) {

vehicles.add(**new** Vehicle(name, isEmergency));

}

**public** String processVehicle() {

**if** (!vehicles.isEmpty()) {

Vehicle vehicle = vehicles.poll();

**return** vehicle.getName();

}

**return** **null**;

}

**public** Queue<String> getAllVehicles() {

Queue<String> vehicleNames = **new** LinkedList<>();

**for** (Vehicle vehicle : vehicles) {

vehicleNames.add(vehicle.getName());

}

**return** vehicleNames;

}

**public** **int** getCongestionLevel() {

**return** vehicles.size();

}

}

GRAPH CLASS:

**package** application;

**import** java.util.\*;

**public** **class** Graph {

**private** **final** Map<String, List<Road>> adjacencyList = **new** HashMap<>();

**private** **final** List<String> intersections = **new** ArrayList<>();

**public** **void** addIntersection(String name) {

intersections.add(name);

adjacencyList.put(name, **new** ArrayList<>());

}

**public** **void** addRoad(String from, String to, **int** weight) {

adjacencyList.get(from).add(**new** Road(from, to, weight));

adjacencyList.get(to).add(**new** Road(to, from, weight)); // Bidirectional

}

**public** List<String> getIntersections() {

**return** intersections;

}

**public** List<Road> getRoads() {

List<Road> roads = **new** ArrayList<>();

**for** (List<Road> roadList : adjacencyList.values()) {

roads.addAll(roadList);

}

**return** roads;

}

**public** Map<String, Integer> findShortestPath(String source) {

Map<String, Integer> distances = **new** HashMap<>();

PriorityQueue<Node> pq = **new** PriorityQueue<>(Comparator.*comparingInt*(Node::getDistance));

Set<String> visited = **new** HashSet<>();

**for** (String intersection : intersections) {

distances.put(intersection, Integer.***MAX\_VALUE***);

}

distances.put(source, 0);

pq.add(**new** Node(source, 0));

**while** (!pq.isEmpty()) {

Node currentNode = pq.poll();

String current = currentNode.getName();

**if** (visited.contains(current)) **continue**;

visited.add(current);

**for** (Road road : adjacencyList.get(current)) {

String neighbor = road.getDestination();

**int** newDist = distances.get(current) + road.getWeight();

**if** (newDist < distances.get(neighbor)) {

distances.put(neighbor, newDist);

pq.add(**new** Node(neighbor, newDist));

}

}

}

**return** distances;

}

**public** **void** updateWeightsBasedOnCongestion(Map<String, Integer> congestionData) {

**for** (String intersection : congestionData.keySet()) {

**int** congestionLevel = congestionData.get(intersection);

**for** (Road road : adjacencyList.get(intersection)) {

road.setWeight(road.getWeight() + congestionLevel);

}

}

}

}

**class** Node {

**private** **final** String name;

**private** **final** **int** distance;

**public** Node(String name, **int** distance) {

**this**.name = name;

**this**.distance = distance;

}

**public** String getName() {

**return** name;

}

**public** **int** getDistance() {

**return** distance;

}

}

**class** Road {

**private** **final** String source;

**private** **final** String destination;

**private** **int** weight;

**public** Road(String source, String destination, **int** weight) {

**this**.source = source;

**this**.destination = destination;

**this**.weight = weight;

}

**public** String getSource() {

**return** source;

}

**public** String getDestination() {

**return** destination;

}

**public** **int** getWeight() {

**return** weight;

}

**public** **void** setWeight(**int** weight) {

**this**.weight = weight;

}

}

TRAFFIC LIGHT CLASS:

**public** **class** TrafficLight {

**private** **final** **int** greenDuration;

**private** **final** **int** redDuration;

**private** **int** timer = 0;

**public** TrafficLight(**int** greenDuration, **int** redDuration) {

**this**.greenDuration = greenDuration;

**this**.redDuration = redDuration;

}

**public** **void** displayTiming() {

timer = (timer + 1) % (greenDuration + redDuration);

String status = timer < greenDuration ? "GREEN" : "RED";

System.***out***.println("Traffic Light: " + status);

}

}

TRAFFIC VISUALIZER CLASS:

**import** javafx.animation.KeyFrame;

**import** javafx.animation.Timeline;

**import** javafx.scene.Scene;

**import** javafx.scene.image.Image;

**import** javafx.scene.image.ImageView;

**import** javafx.scene.layout.Pane;

**import** javafx.scene.paint.Color;

**import** javafx.scene.shape.Line;

**import** javafx.scene.text.Text;

**import** javafx.stage.Stage;

**import** javafx.util.Duration;

**import** java.util.HashMap;

**import** java.util.Map;

**public** **class** TrafficVisualizer {

**private** **final** Pane pane;

**private** **final** Graph graph;

**public** TrafficVisualizer(Stage primaryStage, Graph graph) {

**this**.pane = **new** Pane();

**this**.graph = graph;

Scene scene = **new** Scene(pane, 800, 600);

primaryStage.setTitle("Traffic Management System");

primaryStage.setScene(scene);

primaryStage.show();

}

**public** **void** startSimulation(Lane lane, TrafficLight light) {

drawGraph();

Timeline timeline = **new** Timeline(**new** KeyFrame(Duration.*seconds*(1), event -> {

light.displayTiming();

Map<String, Integer> congestionData = **new** HashMap<>();

congestionData.put("A", lane.getCongestionLevel());

graph.updateWeightsBasedOnCongestion(congestionData);

Map<String, Integer> shortestPaths = graph.findShortestPath("A");

System.***out***.println("Shortest paths from A: " + shortestPaths);

String vehicle = lane.processVehicle();

**if** (vehicle != **null**) {

System.***out***.println("Vehicle passed: " + vehicle);

}

updateVisualization(lane);

}));

timeline.setCycleCount(Timeline.***INDEFINITE***);

timeline.play();

}

**private** **void** drawGraph() {

**int** xOffset = 100;

**int** yOffset = 300;

**for** (String intersection : graph.getIntersections()) {

Text label = **new** Text(xOffset - 10, yOffset - 10, intersection);

pane.getChildren().add(label);

xOffset += 150;

}

**for** (Road road : graph.getRoads()) {

Line edge = **new** Line(

100 \* graph.getIntersections().indexOf(road.getSource()) + 100,

300,

100 \* graph.getIntersections().indexOf(road.getDestination()) + 100,

300

);

edge.setStroke(Color.***BLACK***);

edge.setStrokeWidth(2);

pane.getChildren().add(edge);

}

}

**private** **void** updateVisualization(Lane lane) {

pane.getChildren().removeIf(node -> node **instanceof** Text && ((Text) node).getText().startsWith("Congestion:"));

Text congestionText = **new** Text(10, 50, "Congestion: " + lane.getCongestionLevel());

congestionText.setFill(lane.getCongestionLevel() > 5 ? Color.***RED*** : Color.***GREEN***);

pane.getChildren().add(congestionText);

pane.getChildren().removeIf(node -> node **instanceof** ImageView);

**int** xOffset = 100;

}

}

MAIN CLASS:

**import** javafx.application.Application;

**import** javafx.stage.Stage;

**import** javafx.scene.Scene;

**import** javafx.scene.layout.BorderPane;

**import** javafx.application.Application;

**import** javafx.stage.Stage;

**public** **class** MainClass **extends** Application {

@Override

**public** **void** start(Stage primaryStage) {

Graph cityGraph = **new** Graph();

cityGraph.addIntersection("A");

cityGraph.addIntersection("B");

cityGraph.addIntersection("C");

cityGraph.addIntersection("D");

cityGraph.addRoad("A", "B", 2);

cityGraph.addRoad("B", "C", 3);

cityGraph.addRoad("A", "D", 5);

cityGraph.addRoad("D", "C", 4);

Lane laneA = **new** Lane();

TrafficLight lightA = **new** TrafficLight(10, 5);

laneA.addVehicle("Car1", **false**);

laneA.addVehicle("Ambulance1", **true**);

laneA.addVehicle("Car2", **false**);

TrafficVisualizer visualizer = **new** TrafficVisualizer(primaryStage, cityGraph);

visualizer.startSimulation(laneA, lightA);

}

**public** **static** **void** main(String[] args) {

*launch*(args);

}

}

SUMMARY:

* **Package application**: Contains the core components and classes for the traffic simulation system.
* **Main Class**: Sets up the simulation environment, including the road network, user interface, and simulation logic.
* **Graph Class**: Oversees the structure of the road network, managing intersections, road segments, and route calculations.
* **Lane Class**: Manages traffic flow by controlling vehicle lines and monitoring congestion levels.
* **TrafficLight Class**: Controls the operation and timing cycles of traffic signals.
* **TrafficVisualizer Class**: Provides graphical representation and visualization of the traffic simulation.
* **Vehicle Class**: Defines vehicle behavior and properties, distinguishing between emergency and standard vehicles.

Simulation Tool:

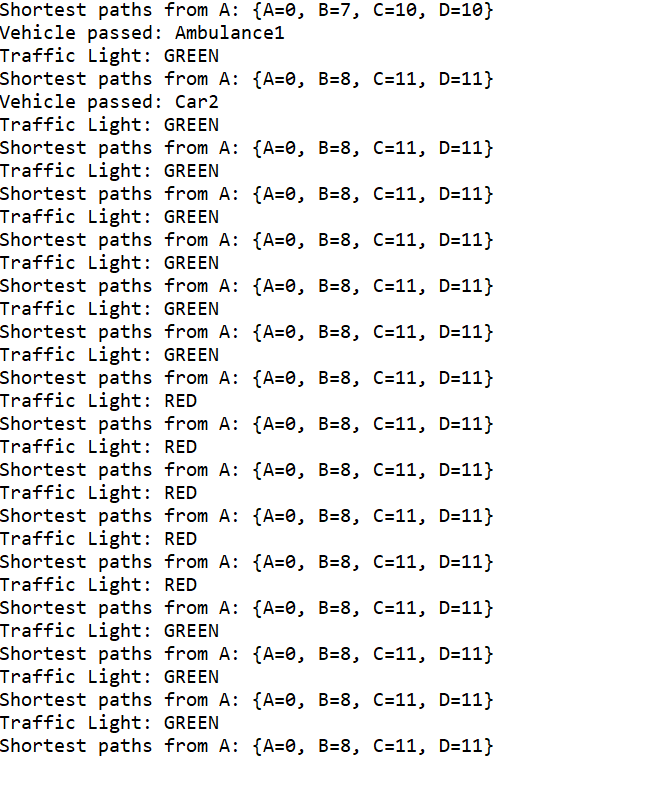
***Interactive GUI Capabilities:***

* Displays the road layout, intersections, and real-time traffic movement.
* Continuously updates to reflect changes in congestion and vehicle positions.
* Uses color codes to visually indicate varying levels of traffic congestion for easy interpretation.

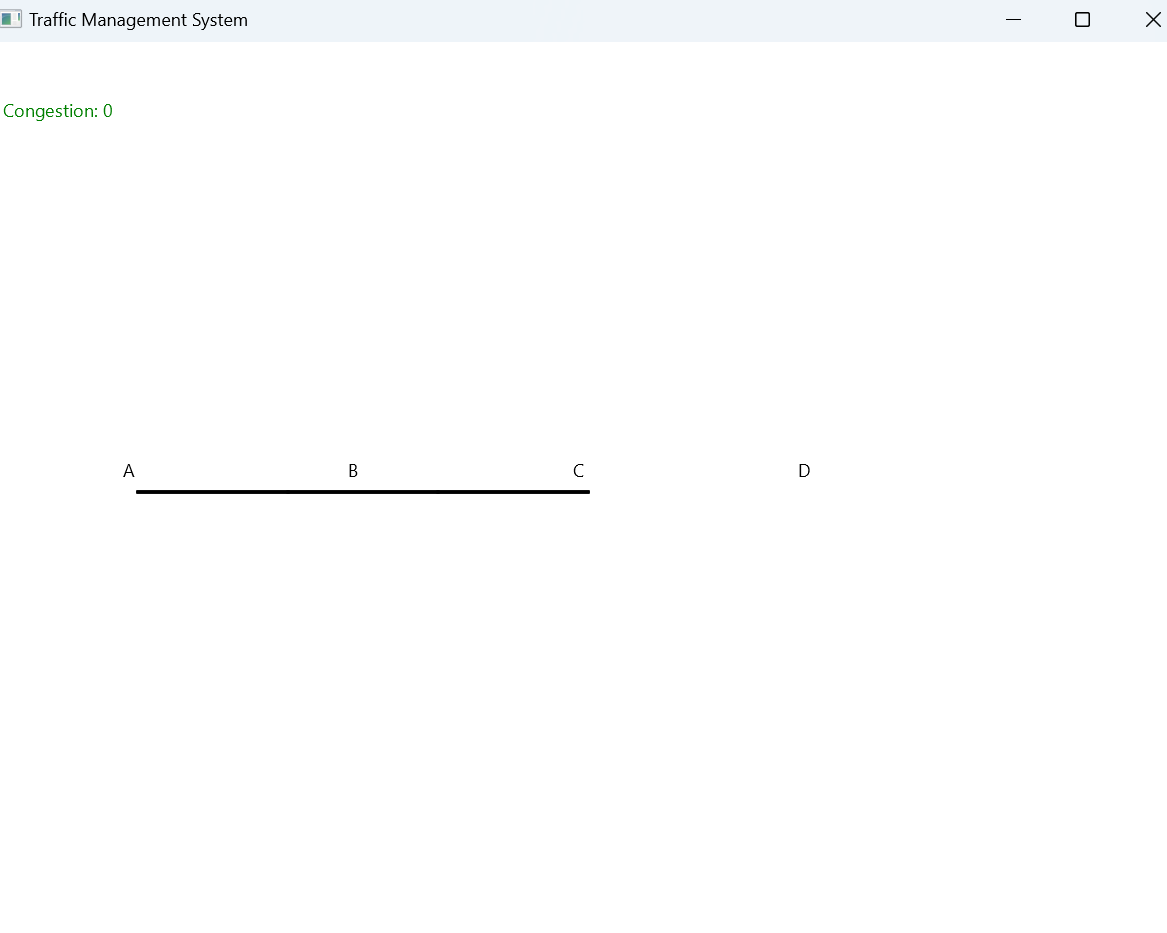
Test Scenarios and Outcomes

1. ***Traffic Congestion Test:***
   * **Input:** Insert 5 vehicles into a single lane.
   * **Expected Result:** Congestion level shown as "5"; vehicles move one after the other.
2. ***Route Optimization Test:***
   * **Input:** Starting point set at intersection "A".
   * **Expected Result:** System displays the accurate shortest routes to all other intersections.
3. ***Traffic Signal Timing Test:***
   * **Input:** Green light duration set to 10 seconds, red to 5 seconds.
   * **Expected Result:** Console reflects the correct switching of light states at specified intervals.

Output:







PresentatioN

***Challenges Encountered:***

* Managing congestion data effectively and adjusting road weights in real-time.
* Displaying live traffic flow changes in the graphical interface.

***Solutions Adopted:***

* Implemented Dijkstra’s algorithm to calculate shortest paths efficiently.
* Utilized JavaFX animation features to enable real-time updates in the GUI.

***Key Takeaways:***

* Learned the value of modular programming for easier expansion and maintenance.
* Gained practical insights into applying graph-based algorithms to solve traffic-related problems.

User Guide

***How to Begin:***

1. Launch the application by running the Main class.
2. Watch the GUI as vehicles move and traffic lights operate in the simulation.
3. Monitor the console for updates on congestion levels and computed shortest paths.

***System Testing Instructions:***

* Insert vehicles into a lane using the Lane.addVehicle() function.
* Adjust the traffic light durations in the TrafficLight class.
* Update road weights based on congestion metrics to influence routing.