

**FACULTY OF ENGINEERING SCIENCES AND TECHNOLOGY**

**Project proposal**

**Group members :**

* **Beeram kumar 69667**
* **Dev kumar 68708**
* **Goveend kumar 68504**

**Project Title:**

Traffic Management System Simulation

**Objective:**

To design and implement a JavaFX-based traffic simulation system that visualizes and optimizes traffic flow using object-oriented programming and data structures like graphs and priority queues. The simulation prioritizes emergency vehicles, simulates real-time vehicle movement, and models road networks to reflect realistic traffic scenarios.

**Scope:**

* Simulate movement of regular and emergency vehicles through a road network.
* Visualize roads, intersections, and vehicle animation in real-time using JavaFX.
* Apply **priority queues** to prioritize emergency vehicles.
* Model road networks using **graph data structures**.
* Use **object-oriented design** principles for modular, scalable code.

**Core Features:**

* **Real-Time Vehicle Flow Simulation**: Vehicles are animated across roads, with emergency vehicles moving first.
* **Graph-Based Road Network**: Road intersections and paths are managed using nodes and edges.
* **Vehicle Priority Handling**: Emergency vehicles (like ambulances) are given precedence using a max-priority queue.
* **Interactive JavaFX GUI**: Roads, vehicles, and nodes are displayed in a user-friendly pane.
* **Dijkstra’s Integration Potential**: The graph structure supports shortest path logic, allowing for future route optimization features.

**Technology Stack:**

* **Java 17+**
* **JavaFX (GUI & Animation)**
* **Core Java Collections (HashMap, PriorityQueue, List)**
* **OOP Principles (Encapsulation, Inheritance, Modularity)**

**System Architecture:**

**Class Overview:**

|  |  |
| --- | --- |
| **Class Name** | **Responsibility** |
| Vehicle | Stores vehicle ID and priority status (emergency or regular). |
| Graph | Represents the road network as a graph (nodes and weighted edges). |
| Edge | Inner class to represent weighted connections between nodes. |
| TrafficSimulationApp | Main class responsible for simulation flow, GUI layout, vehicle animation. |

**UML Diagram Summary:**

**Class Diagram:**

* TrafficSimulationApp uses Vehicle and Graph classes.
* Graph contains a Map<String, List<Edge>> representing nodes and connections.
* Vehicle contains attributes for ID and emergency flag.
* PriorityQueue<Vehicle> ensures emergency vehicle prioritization.

**Sequence Diagram (Simplified):**

1. App starts → road network and vehicles are initialized.
2. Timeline ticks → vehicle dequeued from priority queue.
3. GUI animates vehicle across predefined road.
4. Repeat until vehicle queue is empty.

**Simulation Tool Capabilities:**

* **Dynamic road layout** with node labeling.
* **Vehicle animation** with color-coding (blue for normal, red for emergency).
* **Console output** to show vehicle movements and queue state.
* **Scalable road addition** and future support for advanced algorithms like Dijkstra.

**Test Scenarios & Expected Outcomes:**

|  |  |  |
| --- | --- | --- |
| **Scenario** | **Input** | **Expected Outcome** |
| **Vehicle Priority Test** | Add emergency and normal vehicles | Emergency vehicles (red) appear first in animation |
| **Graph Construction Test** | Add nodes A, B, C, D and connect roads | Correct road layout displayed on JavaFX Pane |
| **Animation Flow Test** | Add 5 vehicles | Vehicles move one after another across defined lanes |
| **Visual Clarity Test** | Use colors and spacing | Nodes and labels clearly visible and organized |

**Conclusion:**

This project simulates a simplified but functional traffic management system using JavaFX, illustrating key data structures like graphs and priority queues while leveraging OOP design. It lays the groundwork for adding advanced features like congestion-aware routing and Dijkstra’s pathfinding in future iterations.

**CODE:**

**package application;**

**import javafx.animation.KeyFrame;**

**import javafx.animation.Timeline;**

**import javafx.application.Application;**

**import javafx.scene.Scene;**

**import javafx.scene.control.Label;**

**import javafx.scene.layout.\*;**

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

**import javafx.scene.shape.Circle;**

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

**import javafx.stage.Stage;**

**import javafx.util.Duration;**

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

**public class TrafficSimulationApp extends Application {**

**private Pane roadPane = new Pane();**

**private PriorityQueue<Vehicle> vehicleQueue = new PriorityQueue<>(**

**Comparator.comparing(Vehicle::isEmergency).reversed()**

**);**

**private Graph graph = new Graph();**

**private int vehicleCounter = 0;**

**private double laneY = 80;**

**@Override**

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

**Label title = new Label("Traffic Management System");**

**title.setStyle("-fx-font-size: 18px; -fx-padding: 10px;");**

**VBox root = new VBox(10);**

**roadPane.setPrefHeight(400);**

**roadPane.setStyle("-fx-background-color: #f0f0f0; -fx-border-color: gray;");**

**root.getChildren().addAll(title, roadPane);**

**Scene scene = new Scene(root, 900, 500);**

**primaryStage.setTitle("Traffic Simulation - DSA CCP Project");**

**primaryStage.setScene(scene);**

**primaryStage.show();**

**buildGraph();**

**generateVehicles();**

**drawRoadNetwork();**

**simulateTrafficFlow();**

**}**

**// Create test vehicles**

**private void generateVehicles() {**

**vehicleQueue.add(new Vehicle("V1", false));**

**vehicleQueue.add(new Vehicle("V2", true)); // emergency**

**vehicleQueue.add(new Vehicle("V3", false));**

**vehicleQueue.add(new Vehicle("V4", true)); // emergency**

**vehicleQueue.add(new Vehicle("V5", false));**

**}**

**// Basic city road network as a graph**

**private void buildGraph() {**

**graph.addNode("A");**

**graph.addNode("B");**

**graph.addNode("C");**

**graph.addNode("D");**

**graph.addEdge("A", "B", 2);**

**graph.addEdge("B", "C", 3);**

**graph.addEdge("A", "D", 5);**

**graph.addEdge("D", "C", 4);**

**}**

**// 🗺 Draw simple road network**

**private void drawRoadNetwork() {**

**// Lines representing roads**

**Line road1 = new Line(100, 100, 300, 100); // A->B**

**Line road2 = new Line(300, 100, 500, 100); // B->C**

**Line road3 = new Line(100, 100, 100, 300); // A->D**

**Line road4 = new Line(100, 300, 500, 100); // D->C**

**road1.setStrokeWidth(5);**

**road2.setStrokeWidth(5);**

**road3.setStrokeWidth(5);**

**road4.setStrokeWidth(5);**

**roadPane.getChildren().addAll(road1, road2, road3, road4);**

**// Nodes**

**addNodeLabel("A", 90, 90);**

**addNodeLabel("B", 290, 90);**

**addNodeLabel("C", 490, 90);**

**addNodeLabel("D", 90, 290);**

**}**

**private void addNodeLabel(String name, double x, double y) {**

**Label label = new Label(name);**

**label.setLayoutX(x);**

**label.setLayoutY(y);**

**label.setStyle("-fx-font-weight: bold; -fx-background-color: white; -fx-padding: 2px;");**

**roadPane.getChildren().add(label);**

**}**

**// Simulate vehicle movement**

**private void simulateTrafficFlow() {**

**Timeline timeline = new Timeline(new KeyFrame(Duration.seconds(2), event -> {**

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

**Vehicle vehicle = vehicleQueue.poll();**

**moveVehicle(vehicle);**

**}**

**}));**

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

**timeline.play();**

**}**

**// Animate vehicles across roads**

**private void moveVehicle(Vehicle vehicle) {**

**Circle car = new Circle(10);**

**car.setFill(vehicle.isEmergency() ? Color.RED : Color.BLUE);**

**car.setTranslateX(100);**

**car.setTranslateY(laneY + (vehicleCounter \* 30));**

**Label label = new Label(vehicle.getId());**

**label.setTranslateX(70);**

**label.setTranslateY(laneY + (vehicleCounter \* 30) - 5);**

**vehicleCounter++;**

**roadPane.getChildren().addAll(car, label);**

**Timeline move = new Timeline(new KeyFrame(Duration.millis(20), e -> {**

**car.setTranslateX(car.getTranslateX() + 2);**

**label.setTranslateX(label.getTranslateX() + 2);**

**}));**

**move.setCycleCount(200);**

**move.play();**

**}**

**// Vehicle Class**

**static class Vehicle {**

**private String id;**

**private boolean emergency;**

**public Vehicle(String id, boolean emergency) {**

**this.id = id;**

**this.emergency = emergency;**

**}**

**public String getId() {**

**return id;**

**}**

**public boolean isEmergency() {**

**return emergency;**

**}**

**}**

**// Simple Graph Class**

**static class Graph {**

**private Map<String, List<Edge>> adj = new HashMap<>();**

**public void addNode(String label) {**

**adj.putIfAbsent(label, new ArrayList<>());**

**}**

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

**adj.get(from).add(new Edge(to, weight));**

**}**

**static class Edge {**

**String to;**

**int weight;**

**Edge(String to, int weight) {**

**this.to = to;**

**this.weight = weight;**

**}**

**}**

**}**

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

**launch(args);**

**}**

**}  
OUTPUT:**

