IQRA UNIVERSITY IU

FACULTY OF ENGINEERING SCIENCES AND TECHNOLOGY

Department: **Computer Science** Program: **BS(CS)**

Data Structures and Algorithm

Announced date: 12-05-2025 **Due Date: 15-06-2025** **Total Marks = 10**

Complex Computing Problem (CCP)		
Mapped CLO	**SDG**	**Complex Problem Solving Mapped**
CLO3	4 & 9	WP1 (Depth of knowledge required) WP2 (Range of conflicting requirements required) WP3 (Depth of analysis required) WP4 (Familiarity of Issues)

Efficient Traffic Management System Simulation Using Data Structures

Problem Statement

Urban traffic congestion poses a significant challenge to transportation efficiency. As cities grow, the complexity of managing vehicle flow across intersections increases. This project aims to simulate and optimize vehicle flow at intersections using core data structures, supporting features such as vehicle priority handling, adjustable signal timings, and real-time congestion analysis.

Objectives

- Simulate urban traffic using data structures like queues, priority queues, and graphs.

- Handle up to 10,000 vehicles and 50 intersections efficiently.

- Integrate real-time adjustable traffic light control.

- Visualize traffic using JavaFX GUI.

- Ensure support for emergency vehicles with priority handling.

Tools & Technologies Used

- Programming Language: Java
- GUI Framework: JavaFX
- IDE: VS Code
- Deployment: Github

System Design

The simulator consists of core components:

- Intersection: Represents a junction with traffic control.

- Vehicle: Encapsulates attributes like ID, arrival time, and priority.

- TrafficSimulator: Manages the entire flow of the simulation.

- TrafficSimulatorGUI: Visual representation using JavaFX.

- Timelines & Scheduling: Controls signal switching and vehicle movement.

The system is designed to simulate both normal and emergency traffic, dynamically adjusting signal timing based on congestion.

Data Structures Used

- **Queue:** For normal vehicles at each intersection.
- **PriorityQueue:** For emergency vehicles with higher priority.
- **LinkedList:** For flexible storage and iteration of vehicle data.
- **Graph:** Represents the city road network for future path-finding and scalability.

Implementation Summary

Intersection.java: Manages vehicles in normal and emergency lanes.

Vehicle.java: Contains vehicle type, ID, and time-related data.

TrafficSimulatorGUI.java: JavaFX-based GUI managing intersections and simulation.

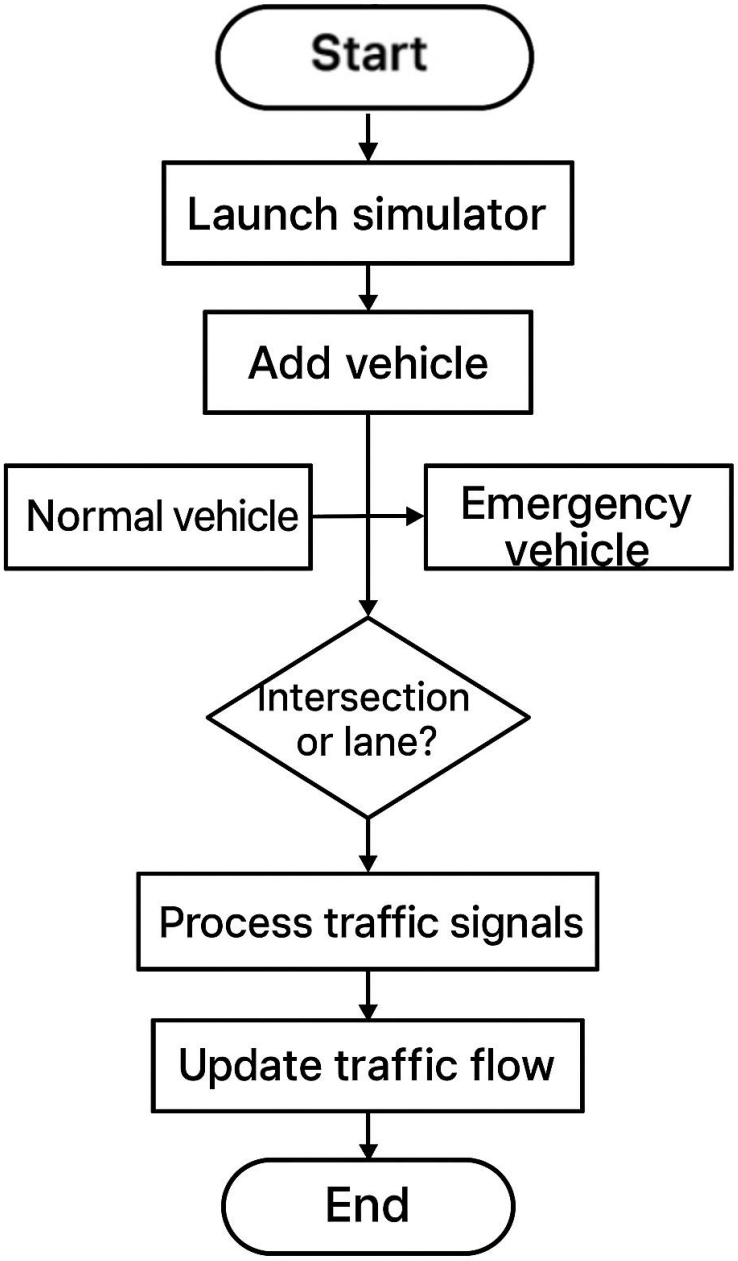
TrafficSimulator.java: Controls simulation logic, timing, and GUI updates.

Testing & Results

- Add Intersection - Successfully added intersections in simulation.
- Add Vehicles - Vehicle queuing (normal/emergency) works correctly.

- Signal Timing - Green light switches and vehicle movement tested.
- Emergency Handling - Emergency vehicles processed with priority.
- GUI - Real-time vehicle flow and signal update tested visually.

Flowchart



Code Snippets

Main Java:

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.stage.Stage;

public class Main extends Application {
    @Override
    public void start(Stage primaryStage) {
        TrafficSimulatorGUI simulator = new TrafficSimulatorGUI();
        Scene scene = new Scene(simulator.getRoot(), 1200, 800);

        primaryStage.setTitle("Traffic Management System Simulator");
        primaryStage.setScene(scene);
        primaryStage.show();

        simulator.startSimulation();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

Point 2D Java:

```
public class Point2D {
    private final double x, y;

    public Point2D(double x, double y) {
        this.x = x;
        this.y = y;
    }

    public double getX() {
        return x;
    }

    public double getY() {
        return y;
    }
}
```

Road Network Java:

```
import java.util.*;


public class RoadNetwork {
    private final Map<String, Intersection> intersections = new
HashMap<>();
    private final Map<String, Map<String, Integer>> adjacentIntersections
= new HashMap<>();

    public void addIntersection(String id) {
        intersections.put(id, new Intersection(id));
        adjacentIntersections.put(id, new HashMap<>());
    }

    public void addRoad(String from, String to, int travelTime) {
        adjacentIntersections.get(from).put(to, travelTime);
        adjacentIntersections.get(to).put(from, travelTime);
    }

    public Intersection getIntersection(String id) {
        return intersections.get(id);
    }

    public Map<String, Map<String, Integer>> getAdjacentIntersections() {
        return adjacentIntersections;
    }
}
```

Vehicle Java:

```


import javafx.scene.shape.Rectangle;

public class Vehicle implements Comparable<Vehicle> {
    private final String id;
    private final boolean emergency;
    private final long arrivalTime;
    private final String source;
    private final String destination;
    private Rectangle visual; // 🔷 this is for GUI representation
```

```
    public Vehicle(String id, boolean emergency, String source, String
destination) {
        this.id = id;
        this.emergency = emergency;
        this.arrivalTime = System.currentTimeMillis();
        this.source = source;
        this.destination = destination;
    }

    private boolean inTransit = false;

    public boolean isInTransit() {
        return inTransit;
    }

    public void setInTransit(boolean inTransit) {
        this.inTransit = inTransit;
    }

    public String getId() {
        return id;
    }

    public boolean isEmergency() {
        return emergency;
    }

    public String getSource() {
        return source;
    }

    public String getDestination() {
        return destination;
    }

    public Rectangle getVisual() {
        return visual;
    }

    public void setVisual(Rectangle visual) {
        this.visual = visual;
    }
```

```
    @Override
    public int compareTo(Vehicle other) {
        if (this.emergency && !other.emergency)
            return -1;
        if (!this.emergency && other.emergency)
            return 1;
        return Long.compare(this.arrivalTime, other.arrivalTime);
    }
}
```

Intersection Node Java:

```
import javafx.scene.layout.StackPane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;
import javafx.scene.text.Text;

public class IntersectionNode {
    private final StackPane visual = new StackPane();
    private final Circle circle = new Circle(30);
    private final Text idText = new Text();
    private final Text countText = new Text();

    public IntersectionNode(String id, double x, double y) {
        circle.setStroke(Color.BLACK);
        circle.setFill(Color.LIGHTGRAY);

        idText.setText(id);
        idText.setFill(Color.BLACK);

        countText.setFill(Color.WHITE);
        countText.setTranslateY(20);

        visual.getChildren().addAll(circle, idText, countText);
        visual.setLayoutX(x);
        visual.setLayoutY(y);
    }

    public void update(int vehicleCount, boolean isGreen, int
emergencyCount) {
        circle.setFill(isGreen ? Color.GREEN : Color.RED);
```

```
        countText.setText("N:" + vehicleCount + " E:" + emergencyCount);
    }

    public StackPane getVisual() {
        return visual;
    }
}
```

Intersection Java:

```
// Intersection.java


import java.util.*;


public class Intersection {
    private final String id;
    private final Queue<Vehicle> normalLane = new LinkedList<>();
    private final PriorityQueue<Vehicle> emergencyLane = new
PriorityQueue<>();
    private int greenDuration = 30;
    private boolean isGreenLight = false;
    private boolean forceGreen = false; // ✅ New flag

    public Intersection(String id) {
        this.id = id;
    }

    public void addVehicle(Vehicle vehicle) {
        if (vehicle.isEmergency()) {
            emergencyLane.add(vehicle);
        } else {
            normalLane.add(vehicle);
        }
    }

    public Queue<Vehicle> getNormalLane() {
        return normalLane;
    }

    public PriorityQueue<Vehicle> getEmergencyLane() {
        return emergencyLane;
```

```
    }

    public String getQueueStatus() {
        return "Normal: " + normalLane.size() + ", Emergency: " +
emergencyLane.size();
    }

    public int getQueueSize() {
        return normalLane.size() + emergencyLane.size();
    }

    public int getEmergencyQueueSize() {
        return emergencyLane.size();
    }

    public boolean isGreenLight() {
        return isGreenLight;
    }

    // ✅ Add this method to allow external classes to force green light
    public void setForceGreenForEmergency(boolean forceGreen) {
        this.forceGreen = forceGreen;
    }

    // ✅ Modify logic to prioritize emergency override
    public void updateLightTiming(int duration) {
        this.greenDuration = duration;

        if (forceGreen) {
            isGreenLight = true;
            forceGreen = false; // ✅ Reset after forcing once
        } else {
            isGreenLight = (System.currentTimeMillis() / 1000) % (duration
* 2) < duration;
        }
    }

    public String getClearanceOrder() {
        List<String> order = new ArrayList<>();
        for (Vehicle v : emergencyLane)
            order.add(v.getId());
        for (Vehicle v : normalLane)
            order.add(v.getId());
```

```
        return String.join(" -> ", order);
    }
}
```

Traffic Simulator GUI Java:

```
// TrafficSimulatorGUI.java

import javafx.scene.layout.*;
import javafx.scene.control.*;
import javafx.scene.paint.Color;
import javafx.scene.shape.*;
import javafx.scene.text.Text;
import javafx.animation.*;
import javafx.util.Duration;
import java.util.*;

public class TrafficSimulatorGUI {
    private final BorderPane root = new BorderPane();
    private final RoadNetwork roadNetwork = new RoadNetwork();
    private final Map<String, IntersectionNode> intersectionNodes = new
HashMap<>();
    private final Pane simulationPane = new Pane();
    private final TextArea logArea = new TextArea();
    private final Timeline simulationTimeline;
    private int simulationTime = 0;

    private final List<Vehicle> vehicles = new ArrayList<>();
    private final Map<String, Point2D> positions = Map.of(
            "A", new Point2D(200, 200),
            "B", new Point2D(600, 200),
            "C", new Point2D(600, 600),
            "D", new Point2D(200, 600));

    public TrafficSimulatorGUI() {
        initializeRoadNetwork();
        setupUI();

        simulationTimeline = new Timeline(
                new KeyFrame(Duration.seconds(1), e -> {
                    simulationTime++;
                    updateSimulation();
                }));
```

```
        simulationTimeline.setCycleCount(Timeline.INDEFINITE);
    }

    private void initializeRoadNetwork() {
        String[] intersections = { "A", "B", "C", "D" };
        for (String id : intersections) {
            roadNetwork.addIntersection(id);
        }

        roadNetwork.addRoad("A", "B", 2);
        roadNetwork.addRoad("B", "C", 3);
        roadNetwork.addRoad("A", "D", 5);
        roadNetwork.addRoad("D", "C", 4);

        addVehicle(new Vehicle("V1", false, "A", "C"));
        addVehicle(new Vehicle("V2", true, "A", "B"));
        addVehicle(new Vehicle("V3", false, "A", "D"));
        addVehicle(new Vehicle("V4", false, "A", "C"));
    }

    private void setupUI() {
        simulationPane.setPrefSize(900, 700);
        drawRoadNetwork();

        VBox controlPanel = new VBox(10);
        Button startButton = new Button("Start Simulation");
        Button pauseButton = new Button("Pause");
        Button resetButton = new Button("Reset");

        startButton.setOnAction(e -> startSimulation());
        pauseButton.setOnAction(e -> simulationTimeline.pause());
        resetButton.setOnAction(e -> resetSimulation());

        logArea.setEditable(false);
        logArea.setWrapText(true);
        logArea.setPrefHeight(150);

        controlPanel.getChildren().addAll(startButton, pauseButton,
resetButton, new Label("Simulation Log:"), logArea);
        controlPanel.setPadding(new javafx.geometry.Insets(10));

        root.setCenter(simulationPane);
```

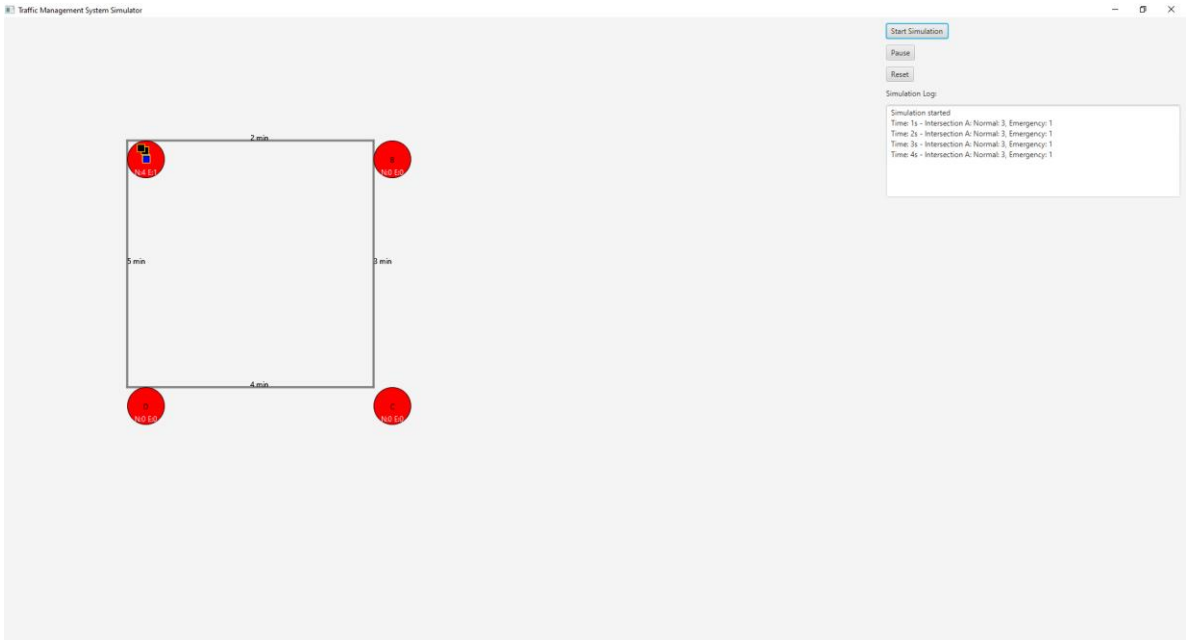
```
        root.setRight(controlPanel);
    }

    private void drawRoadNetwork() {
        simulationPane.getChildren().clear();
        intersectionNodes.clear();

        for (String from :
roadNetwork.getAdjacentIntersections().keySet()) {
            for (String to :
roadNetwork.getAdjacentIntersections().get(from).keySet()) {
                Point2D start = positions.get(from);
                Point2D end = positions.get(to);

                Line road = new Line(start.getX(), start.getY(),
end.getX(), end.getY());
                road.setStroke(Color.GRAY);
                road.setStrokeWidth(3);

                Text label = new Text(
                        (start.getX() + end.getX()) / 2,
                        (start.getY() + end.getY()) / 2,
                        roadNetwork.getAdjacentIntersections().get(from).g
et(to) + " min");

                simulationPane.getChildren().addAll(road, label);
            }
        }

        for (String id : positions.keySet()) {
            Point2D pos = positions.get(id);
            IntersectionNode node = new IntersectionNode(id, pos.getX(),
pos.getY());
            intersectionNodes.put(id, node);
            simulationPane.getChildren().add(node.getVisual());
        }

        for (Vehicle v : vehicles) {
            simulationPane.getChildren().add(v.getVisual());
        }
    }

    private void updateSimulation() {
```

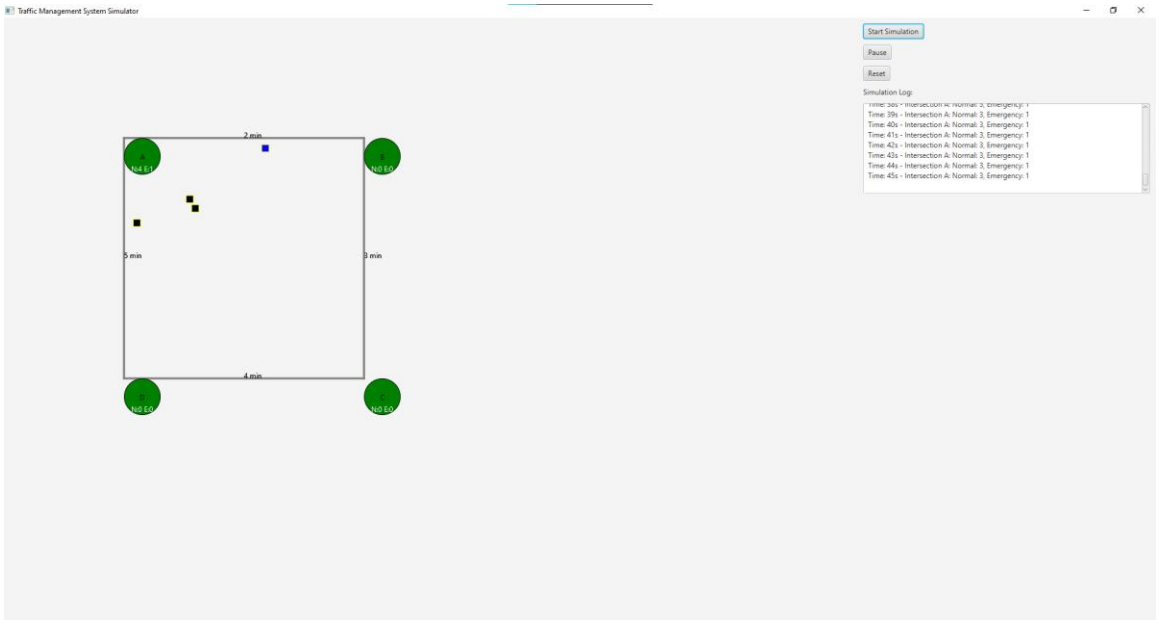
```
        for (String id : intersectionNodes.keySet()) {
            Intersection intersection = roadNetwork.getIntersection(id);
            int queueSize = intersection.getQueueSize();
            int greenDuration = Math.min(60, 30 + queueSize * 3);
            intersection.updateLightTiming(greenDuration);

            intersectionNodes.get(id).update(
                    intersection.getQueueSize(),
                    intersection.isGreenLight(),
                    intersection.getEmergencyQueueSize());
        }

        moveVehicles();

        logArea.appendText("Time: " + simulationTime + "s - ");
        logArea.appendText("Intersection A: " +
                roadNetwork.getIntersection("A").getQueueStatus() + "\n");
    }

    private void moveVehicles() {
        for (Vehicle v : vehicles) {
            Rectangle rect = v.getVisual();
            double x = rect.getLayoutX();
            double y = rect.getLayoutY();
            Point2D source = positions.get(v.getSource());
            Point2D dest = positions.get(v.getDestination());

            double dx = dest.getX() - x;
            double dy = dest.getY() - y;
            double distance = Math.sqrt(dx * dx + dy * dy);

            Intersection sourceIntersection =
roadNetwork.getIntersection(v.getSource());
            boolean canMove = v.isEmergency() ||
sourceIntersection.isGreenLight();

            if (distance > 5 && canMove) {
                double step = 5;
                double nx = x + (dx / distance) * step;
                double ny = y + (dy / distance) * step;
                rect.setLayoutX(nx);
                rect.setLayoutY(ny);
            } else if (distance <= 5 && v.isEmergency()) {
```

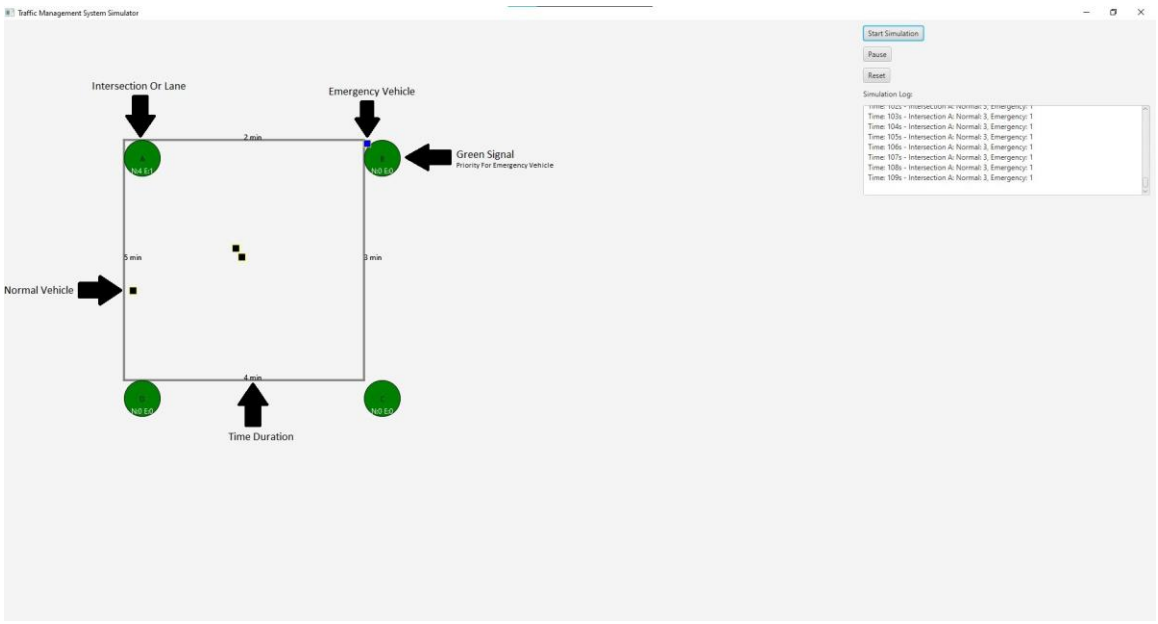
```
                Intersection destIntersection =
roadNetwork.getIntersection(v.getDestination());
                destIntersection.setForceGreenForEmergency(true); // 💡
Force green
            }
        }
    }

    private void addVehicle(Vehicle vehicle) {
        vehicles.add(vehicle);
        roadNetwork.getIntersection(vehicle.getSource()).addVehicle(vehicl
e);

        Point2D pos = positions.get(vehicle.getSource());
        Rectangle visual = new Rectangle(12, 12, vehicle.isEmergency() ?
Color.BLUE : Color.BLACK);
        visual.setStroke(Color.YELLOW);
        Tooltip.install(visual, new Tooltip(vehicle.getId()));

        visual.setLayoutX(pos.getX() + Math.random() * 30);
        visual.setLayoutY(pos.getY() + Math.random() * 30);

        vehicle.setVisual(visual);
        simulationPane.getChildren().add(visual);
    }

    private void resetSimulation() {
        simulationTimeline.stop();
        simulationTime = 0;
        vehicles.clear();
        simulationPane.getChildren().clear();
        initializeRoadNetwork();
        drawRoadNetwork();
        logArea.clear();
    }

    public void startSimulation() {
        logArea.clear();
        logArea.appendText("Simulation started\n");
        simulationTimeline.play();
    }

    public BorderPane getRoot() {
```

```
            return root;
        }
}
```

Output







Github Link:

https://github.com/AneeqUllahKhan/Java-Traffic-Simulation

Key Learnings

This project reinforced practical understanding of data structures in real-time systems, highlighted challenges in concurrency and simulation timing, and demonstrated how structured programming can address complex real-world problems like urban traffic congestion.

Conclusion

The Traffic Management Simulator effectively models traffic behavior at intersections using key data structures and algorithms. It enables real-time simulation with GUI support and vehicle priority, demonstrating both technical knowledge and practical implementation skills.