JAVA ASSIGNMENT

Smart Traffic Signal Optimization

Scenario: You are part of a team working on an initiative to optimize traffic signal management in a busy city to reduce congestion and improve traffic flow efficiency using smart technologies.

Tasks:

1. **Data Collection and Modeling:**
   - Define the data structure to collect real-time traffic data from sensors (e.g., vehicle counts, speeds) at various intersections across the city.

```
public class TrafficData {

   private String intersectionId; // Unique identifier for the intersection

   private long timestamp; // Time of data collection

   private int vehicleCount; // Number of vehicles detected

   private double averageSpeed; // Average speed of vehicles

   private int pedestrianCount; // Number of pedestrians detected



   // Constructor

   public TrafficData(String intersectionId, long timestamp, int vehicleCount, double averageSpeed, int pedestrianCount) {

      this.intersectionId = intersectionId;

      this.timestamp = timestamp;

      this.vehicleCount = vehicleCount;

      this.averageSpeed = averageSpeed;

      this.pedestrianCount = pedestrianCount;

   }



   // Getters and Setters
```

```
    public String getIntersectionId() { return intersectionId; }

    public void setIntersectionId(String intersectionId) { this.intersectionId = intersectionId; }



    public long getTimestamp() { return timestamp; }

    public void setTimestamp(long timestamp) { this.timestamp = timestamp; }



    public int getVehicleCount() { return vehicleCount; }

    public void setVehicleCount(int vehicleCount) { this.vehicleCount = vehicleCount; }



    public double getAverageSpeed() { return averageSpeed; }

    public void setAverageSpeed(double averageSpeed) { this.averageSpeed = averageSpeed;
}



    public int getPedestrianCount() { return pedestrianCount; }

    public void setPedestrianCount(int pedestrianCount) { this.pedestrianCount =
pedestrianCount; }

}
```

2.Algorithm Design:

- Develop algorithms to analyze the collected data and optimize traffic signal timings dynamically based on current traffic conditions.
- Consider factors such as traffic density, vehicle queues, peak hours, and pedestrian crossings in your algorithm.

```
public class TrafficSignalOptimizer {



    // Define constants for traffic light timings and thresholds

    private static final int MAX_GREEN_TIME = 60; // Maximum green time in seconds

    private static final int MIN_GREEN_TIME = 30; // Minimum green time in seconds
    private static final int PEAK_HOUR_THRESHOLD = 50; // Vehicle count threshold for peak hours

    // Method to calculate green light duration based on traffic data
    public int calculateGreenTime(TrafficData data) {
        int greenTime;

        if (data.getVehicleCount() > PEAK_HOUR_THRESHOLD) {
            greenTime = Math.min(MAX_GREEN_TIME, MIN_GREEN_TIME + (data.getVehicleCount() / 10)); // Simple formula for green time
        } else {
            greenTime = MIN_GREEN_TIME;
        }

        return greenTime;
    }

    // Method to adjust signal timings based on current data
    public void adjustSignalTiming(TrafficData data) {
        int greenTime = calculateGreenTime(data);
        System.out.println("Adjusting green time to: " + greenTime + " seconds");
        // Implementation for adjusting the traffic signal would go here
    }
}
```

3.Implementation:

- Implement a Java application that integrates with traffic sensors and controls traffic signals at selected intersections.

- Ensure the application can adjust signal timings in real-time to respond to changing traffic patterns and optimize flow.

```
import java.util.Timer;
import java.util.TimerTask;


public class TrafficSignalControl {


    private TrafficSignalOptimizer optimizer = new TrafficSignalOptimizer();
    private Timer timer = new Timer();


    public void startTrafficControl() {
        timer.scheduleAtFixedRate(new TimerTask() {
            @Override
            public void run() {
                // Simulate data collection from sensors
                TrafficData data = collectTrafficData();
                optimizer.adjustSignalTiming(data);
            }
        }, 0, 5000); // Update every 5 seconds
    }


    private TrafficData collectTrafficData() {
        // Simulate data collection from traffic sensors
        // In a real implementation, this would interface with sensor APIs
        return new TrafficData("Intersection1", System.currentTimeMillis(), (int) (Math.random() * 100), 30 + Math.random() * 10, (int) (Math.random() * 20));
    }


    public static void main(String[] args) {

        TrafficSignalControl control = new TrafficSignalControl();

        control.startTrafficControl();

    }

}
```

4.Visualization and Reporting:

- Develop visualizations to monitor traffic conditions and signal timings in real-time.
- Generate reports on traffic flow improvements, average wait times, and overall congestion reduction achieved.

Code:

```
package com.example.trafficsignals;

import javafx.animation.KeyFrame;

import javafx.animation.Timeline;

import javafx.application.Application;

import javafx.scene.Scene;

import javafx.scene.layout.StackPane;

import javafx.scene.paint.Color;

import javafx.scene.shape.Circle;

import javafx.scene.layout.VBox;

import javafx.stage.Stage;

import javafx.util.Duration;


import java.io.IOException;
```

```
public class HelloApplication extends Application {
    @Override
    public void start(Stage primaryStage) {
        // Create the traffic light circles
        Circle redLight = new Circle(50, Color.RED);
        Circle yellowLight = new Circle(50, Color.GRAY);
        Circle greenLight = new Circle(50, Color.GRAY);

        // Arrange the circles in a vertical layout
        VBox root = new VBox(10);
        root.getChildren().addAll(redLight, yellowLight, greenLight);

        // Create the scene and set the stage
        Scene scene = new Scene(root, 200, 600);
        primaryStage.setTitle("Traffic Signal Animation");
        primaryStage.setScene(scene);
        primaryStage.show();

        // Create a timeline for the animation
        Timeline timeline = new Timeline(
                new KeyFrame(Duration.seconds(0), e -> {
                    redLight.setFill(Color.RED);
```

```
            yellowLight.setFill(Color.GRAY);
            greenLight.setFill(Color.GRAY);
        }),
        new KeyFrame(Duration.seconds(3), e -> {
            redLight.setFill(Color.GRAY);
            yellowLight.setFill(Color.YELLOW);
            greenLight.setFill(Color.GRAY);
        }),
        new KeyFrame(Duration.seconds(6), e -> {
            redLight.setFill(Color.GRAY);
            yellowLight.setFill(Color.GRAY);
            greenLight.setFill(Color.GREEN);
        }),
        new KeyFrame(Duration.seconds(9), e -> {
            redLight.setFill(Color.RED);
            yellowLight.setFill(Color.GRAY);
            greenLight.setFill(Color.GRAY);
        })
    );

    // Set the cycle count to indefinite to keep the animation running
    timeline.setCycleCount(Timeline.INDEFINITE);
    timeline.play();
}
public static void main(String[] args) {
        launch(args);

    }

}
```

Recording
2024-07-27 092121.m

5.User Interaction:

- Design a user interface for traffic managers to monitor and manually adjust signal timings if needed.
- Provide a dashboard for city officials to view performance metrics and historical data.

User Interface Design:

Traffic Manager Interface:

- **Real-Time Dashboard**: Displays current traffic data and signal timings.
- **Manual Override Controls**: Allows traffic managers to manually adjust signal timings if needed.

City Official Dashboard:

- **Performance Metrics**: Displays graphs and statistics on traffic flow, average wait times, and congestion levels.
- **Historical Data**: Allows officials to review historical traffic data and trends

```
import javafx.application.Application;

import javafx.scene.Scene;

import javafx.scene.control.Button;

import javafx.scene.control.Label;

import javafx.scene.layout.VBox;

import javafx.stage.Stage;



public class TrafficManagerUI extends Application {
```

```java
    @Override
    public void start(Stage stage) {
        stage.setTitle("Traffic Manager Interface");

        Label statusLabel = new Label("Current Signal Status: Normal");
        Button overrideButton = new Button("Manual Override");

        overrideButton.setOnAction(e -> {
            // Handle manual override logic
            statusLabel.setText("Signal Status: Manually Adjusted");
        });

        VBox vbox = new VBox(10, statusLabel, overrideButton);
        Scene scene = new Scene(vbox, 300, 200);

        stage.setScene(scene);
        stage.show();
    }

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

Deliverables Summary

- **Data Flow Diagram**: Illustrates the flow of data from sensors, through processing, to signal adjustment.
- **Pseudocode and Implementation**: Detailed pseudocode and Java code for data processing and signal adjustment algorithms.
- **Documentation**: Explains design decisions, assumptions, and potential improvements.
- **User Interface**: Intuitive interfaces for traffic managers and city officials.
- **Testing**: Includes test cases for various scenarios to ensure system effectiveness.