

**Efficient Traffic Management System Simulation Using Data Structure**

|  |  |
| --- | --- |
| **Course** | **Data Structure and Algorithms** |
| **Submitted to** | **Israr Ali** |
| **Submitted by** | **Elisha Joseph (68660)**  **Gajanand (68677)**  **Avinash Talreja (69535)** |
| **Slot** | **Tues- Thurs (2:30 – 3:45 PM) (E-704)** |

**1. Documentation**

**Project Proposal**

**Objectives:**Develop a smart, GUI-based traffic management system in Java that dynamically controls traffic lights at multiple intersections using real-time vehicle counts and integrates with a machine learning (ML) API to optimize green light duration. The system aims to improve traffic flow, support emergency and pedestrian scenarios, and provide a visual simulation for educational and research purposes.

**Scope:**

* Four-intersection simulation (A1, A2, B1, B2)
* Real-time vehicle count tracking per intersection
* Dynamic traffic light control (red, green, yellow)
* ML-powered green time prediction via REST API
* Emergency and pedestrian handling
* Day/Night theme toggle
* Visual vehicle and arrow animations
* User-friendly GUI and logging

**Methodology:**

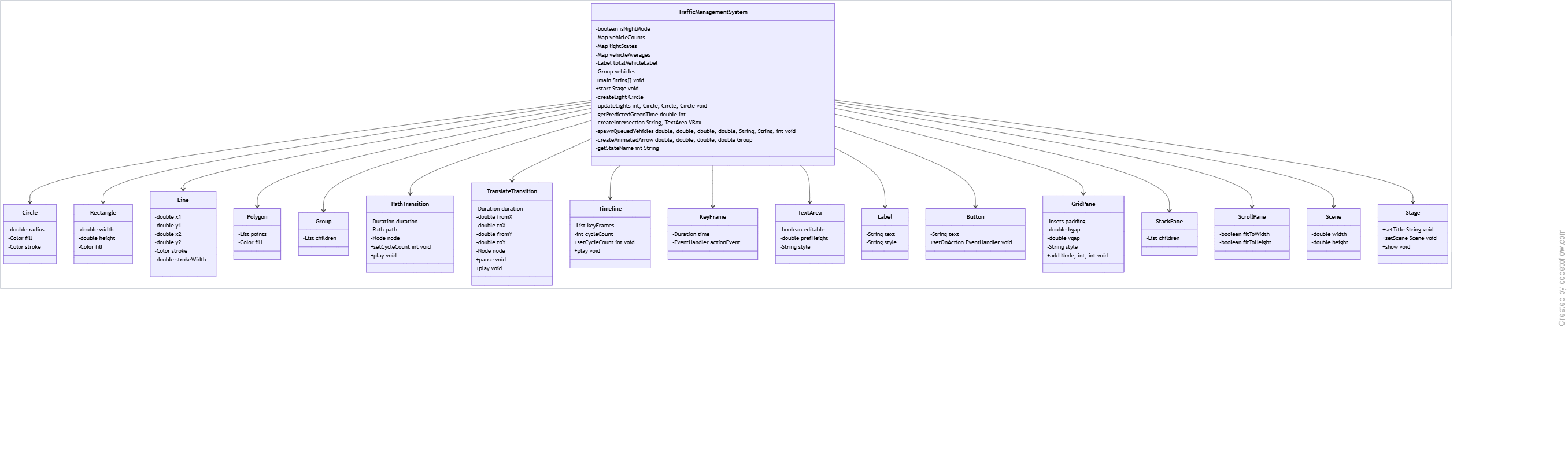
* JavaFX for GUI and animation
* Modular class design for intersections, vehicles, and lights
* REST communication with ML backend for adaptive control
* Timeline and event-driven programming for real-time simulation
* Comprehensive logging and user interaction controls

**Design Document**

**System Architecture:**

* **Main Application:** Entry point, initializes GUI and simulation.
* **Intersection Module:** Manages lights, vehicle counts, state transitions, and user controls.
* **Vehicle Module:** Animates vehicles based on light states and type (normal, emergency, pedestrian).
* **ML Integration:** Sends vehicle averages to a local ML API and parses the predicted green time.

**UML Diagrams:**

****

**2. Codebase**

* The codebase is a single JavaFX application class (TrafficManagementSystem) implementing all core features.

**Key Features:**

* Modular intersection creation (createIntersection)
* Real-time vehicle count simulation and display
* RESTful ML API integration for adaptive timing
* Emergency and pedestrian controls
* Animated vehicle and arrow movement
* Day/Night mode toggle
* Comprehensive in-app logging

**Code:**

package application;

import javafx.animation.\*;

import javafx.application.Application;

import javafx.geometry.Insets;

import javafx.geometry.Pos;

import javafx.scene.Scene;

import javafx.scene.control.\*;

import javafx.scene.layout.\*;

import javafx.scene.paint.Color;

import javafx.scene.shape.\*;

import javafx.stage.Stage;

import javafx.util.Duration;

import javafx.scene.Group;

import java.io.\*;

import java.net.HttpURLConnection;

import java.net.URL;

import java.util.HashMap;

import java.util.Map;

public class TrafficManagementSystem extends Application {

private boolean isNightMode = false;

private final Map<String, Integer> vehicleCounts = new HashMap<>();

private final Map<String, Integer> lightStates = new HashMap<>();

private final Map<String, Double> vehicleAverages = new HashMap<>();

private final Label totalVehicleLabel = new Label("Total Vehicles: 0");

private final Group vehicles = new Group();

public static void main(String[] args) {

launch(args);

}

private Circle createLight() {

Circle circle = new Circle(20);

circle.setFill(Color.DARKGRAY);

circle.setStroke(Color.BLACK);

return circle;

}

private void updateLights(int state, Circle red, Circle yellow, Circle green) {

red.setFill(Color.DARKGRAY);

yellow.setFill(Color.DARKGRAY);

green.setFill(Color.DARKGRAY);

switch (state) {

case 0 -> red.setFill(Color.RED);

case 1 -> green.setFill(Color.LIMEGREEN);

case 2 -> yellow.setFill(Color.YELLOW);

}

}

private int getPredictedGreenTime(double avg) {

try {

URL url = new URL("http://localhost:5000/predict");

HttpURLConnection conn = (HttpURLConnection) url.openConnection();

conn.setRequestMethod("POST");

conn.setRequestProperty("Content-Type", "application/json");

conn.setDoOutput(true);

String jsonInput = "{"average\_vehicle\_count":" + avg + "}";

try (OutputStream os = conn.getOutputStream()) {

byte[] input = jsonInput.getBytes("utf-8");

os.write(input, 0, input.length);

}

try (BufferedReader br = new BufferedReader(

new InputStreamReader(conn.getInputStream(), "utf-8"))) {

StringBuilder response = new StringBuilder();

String line;

while ((line = br.readLine()) != null) {

response.append(line.trim());

}

String responseText = response.toString();

int start = responseText.indexOf(":") + 1;

int end = responseText.indexOf("}");

String value = responseText.substring(start, end).trim();

return Integer.parseInt(value);

}

} catch (Exception e) {

System.err.println("ML API failed, using default green time. Error: " + e.getMessage());

return 6; // fallback

}

}

private VBox createIntersection(String name, TextArea logArea) {

Circle red = createLight();

Circle yellow = createLight();

Circle green = createLight();

Label timerLabel = new Label("Time: 6s");

Label vehicleLabel = new Label("Vehicles: 0");

Label pedestrianCountdown = new Label("");

timerLabel.setStyle("-fx-text-fill: white;");

vehicleLabel.setStyle("-fx-text-fill: white;");

pedestrianCountdown.setStyle("-fx-text-fill: cyan;");

Button overrideBtn = new Button("Next Light");

Button pedestrianBtn = new Button("Pedestrian");

Button emergencyBtn = new Button("Emergency");

VBox box = new VBox(10);

box.setAlignment(Pos.CENTER);

box.setPadding(new Insets(10));

box.setStyle("-fx-background-color: #222; -fx-border-color: yellow;");

Label nameLabel = new Label("Intersection " + name);

nameLabel.setStyle("-fx-text-fill: white;");

box.getChildren().addAll(nameLabel, red, yellow, green, timerLabel, pedestrianCountdown, vehicleLabel,

overrideBtn, pedestrianBtn, emergencyBtn);

int[] state = {0};

int[] countdown = {6};

int[] pedestrianTime = {0};

vehicleCounts.put(name, 0);

lightStates.put(name, 0);

vehicleAverages.put(name, 0.0);

updateLights(state[0], red, yellow, green);

Timeline lightCycle = new Timeline(new KeyFrame(Duration.seconds(1), e -> {

countdown[0]--;

int vCount = vehicleCounts.get(name);

timerLabel.setText("Time: " + countdown[0] + "s");

if (countdown[0] <= 0) {

double prevAvg = vehicleAverages.get(name);

double avg = (prevAvg + vCount) / 2;

vehicleAverages.put(name, avg);

state[0] = (state[0] + 1) % 3;

lightStates.put(name, state[0]);

updateLights(state[0], red, yellow, green);

if (state[0] == 1) {

countdown[0] = getPredictedGreenTime(avg);

} else {

countdown[0] = 6;

}

logArea.appendText(name + ": Switched to " + getStateName(state[0]) + "\n");

}

if (pedestrianTime[0] > 0) {

pedestrianTime[0]--;

pedestrianCountdown.setText("Walk: " + pedestrianTime[0] + "s");

pedestrianCountdown.setVisible(pedestrianTime[0] % 2 == 0);

if (pedestrianTime[0] == 0) pedestrianCountdown.setText("");

}

}));

lightCycle.setCycleCount(Animation.INDEFINITE);

lightCycle.play();

Timeline vehicleUpdater = new Timeline(new KeyFrame(Duration.seconds(2), e -> {

int newVehicles = vehicleCounts.get(name) + (int) (Math.random() \* 5);

vehicleCounts.put(name, newVehicles);

vehicleLabel.setText("Vehicles: " + newVehicles);

}));

vehicleUpdater.setCycleCount(Animation.INDEFINITE);

vehicleUpdater.play();

overrideBtn.setOnAction(e -> {

state[0] = (state[0] + 1) % 3;

lightStates.put(name, state[0]);

updateLights(state[0], red, yellow, green);

countdown[0] = 6;

});

pedestrianBtn.setOnAction(e -> {

state[0] = 0;

lightStates.put(name, state[0]);

updateLights(state[0], red, yellow, green);

pedestrianTime[0] = 5;

countdown[0] = 6;

});

emergencyBtn.setOnAction(e -> {

state[0] = 1;

lightStates.put(name, state[0]);

updateLights(state[0], red, yellow, green);

countdown[0] = 8;

});

return box;

}

private void spawnQueuedVehicles(double x1, double y1, double x2, double y2, String type, String lightName, int delay) {

Timeline spawner = new Timeline(new KeyFrame(Duration.seconds(delay), e -> {

Shape vehicle;

if (type.equals("emergency")) {

vehicle = new Rectangle(20, 10, Color.RED);

} else if (type.equals("pedestrian")) {

vehicle = new Circle(5, Color.YELLOW);

} else {

vehicle = new Rectangle(15, 8, Color.GRAY);

}

vehicle.setTranslateX(x1);

vehicle.setTranslateY(y1);

TranslateTransition move = new TranslateTransition(Duration.seconds(4), vehicle);

move.setFromX(0);

move.setToX(x2 - x1);

move.setFromY(0);

move.setToY(y2 - y1);

move.setCycleCount(Animation.INDEFINITE);

move.pause();

Timeline stateMonitor = new Timeline(new KeyFrame(Duration.seconds(1), ev -> {

int current = lightStates.getOrDefault(lightName, 0);

if (type.equals("emergency")) {

move.play();

} else if (current == 1) {

move.play();

} else {

move.pause();

}

}));

stateMonitor.setCycleCount(Animation.INDEFINITE);

stateMonitor.play();

vehicles.getChildren().add(vehicle);

}));

spawner.setCycleCount(Animation.INDEFINITE);

spawner.play();

}

private Group createAnimatedArrow(double x1, double y1, double x2, double y2) {

Line line = new Line(x1, y1, x2, y2);

line.setStroke(Color.YELLOW);

line.setStrokeWidth(3);

Polygon head = new Polygon(0, 0, -10, -5, -10, 5);

head.setFill(Color.YELLOW);

Group group = new Group(line, head);

PathTransition t = new PathTransition(Duration.seconds(2), line, head);

t.setCycleCount(Animation.INDEFINITE);

t.play();

head.setRotate(Math.toDegrees(Math.atan2(y2 - y1, x2 - x1)));

return group;

}

private String getStateName(int state) {

return switch (state) {

case 0 -> "RED";

case 1 -> "GREEN";

case 2 -> "YELLOW";

default -> "UNKNOWN";

};

}

@Override

public void start(Stage stage) {

TextArea logArea = new TextArea();

logArea.setEditable(false);

logArea.setPrefHeight(150);

logArea.setStyle("-fx-control-inner-background: black; -fx-text-fill: white;");

GridPane grid = new GridPane();

grid.setPadding(new Insets(40));

grid.setHgap(40);

grid.setVgap(40);

grid.setStyle("-fx-background-color: black;");

VBox a1 = createIntersection("A1", logArea);

VBox a2 = createIntersection("A2", logArea);

VBox b1 = createIntersection("B1", logArea);

VBox b2 = createIntersection("B2", logArea);

grid.add(a1, 0, 0);

grid.add(a2, 1, 0);

grid.add(b1, 0, 1);

grid.add(b2, 1, 1);

Group arrows = new Group(

createAnimatedArrow(250, 100, 400, 100),

createAnimatedArrow(420, 160, 420, 300),

createAnimatedArrow(400, 310, 250, 310),

createAnimatedArrow(230, 300, 230, 160)

);

spawnQueuedVehicles(250, 90, 400, 90, "normal", "A1", 4);

spawnQueuedVehicles(420, 150, 420, 300, "emergency", "A2", 4);

spawnQueuedVehicles(400, 310, 250, 310, "pedestrian", "B2", 4);

spawnQueuedVehicles(230, 300, 230, 160, "normal", "B1", 4);

StackPane layered = new StackPane(grid, arrows, vehicles);

layered.setPrefHeight(600);

Button toggleTheme = new Button("Toggle Day/Night Mode");

toggleTheme.setOnAction(e -> {

isNightMode = !isNightMode;

grid.setStyle("-fx-background-color: " + (isNightMode ? "#111" : "white") + ";");

});

totalVehicleLabel.setStyle("-fx-font-size: 16px; -fx-text-fill: orange;");

VBox root = new VBox(10, totalVehicleLabel, toggleTheme, layered, logArea);

root.setPadding(new Insets(10));

ScrollPane scroll = new ScrollPane(root);

scroll.setFitToWidth(true);

scroll.setFitToHeight(true);

Scene scene = new Scene(scroll, 950, 720);

Timeline dashboard = new Timeline(new KeyFrame(Duration.seconds(2), e -> {

int total = vehicleCounts.values().stream().mapToInt(Integer::intValue).sum();

totalVehicleLabel.setText("Total Vehicles: " + total);

}));

dashboard.setCycleCount(Animation.INDEFINITE);

dashboard.play();

stage.setTitle("Smart Traffic System - ML Powered");

stage.setScene(scene);

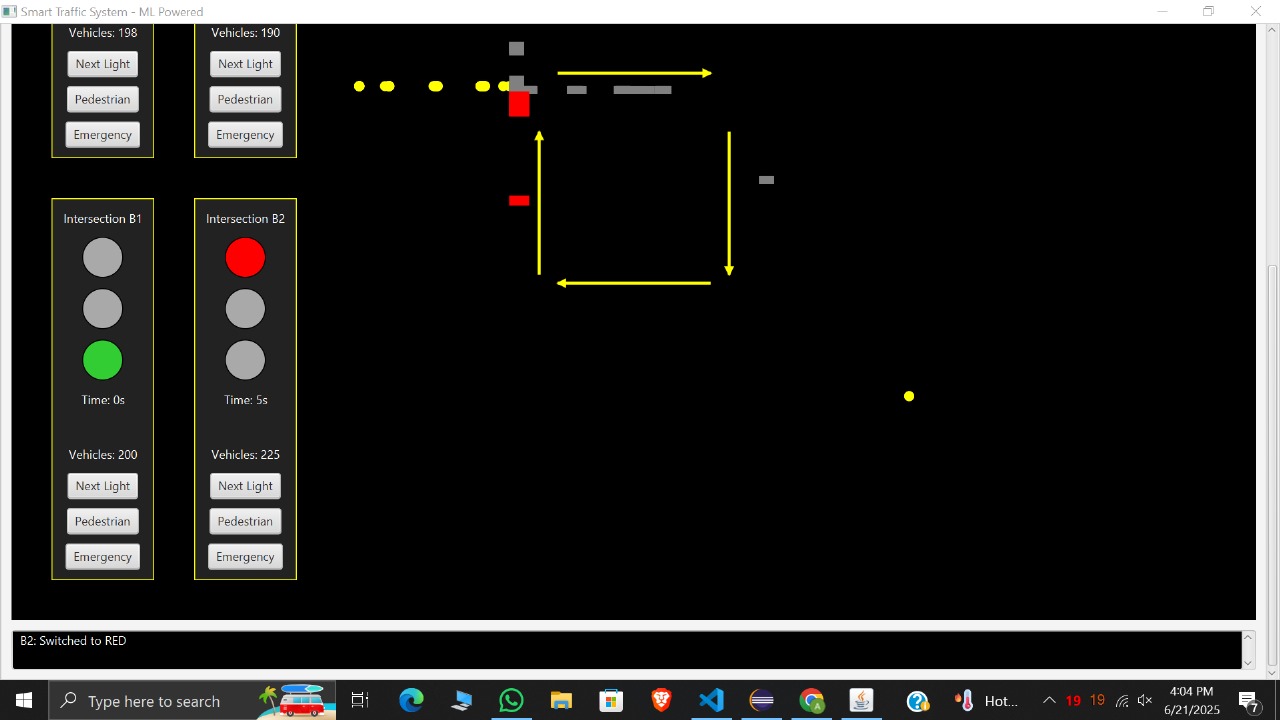
stage.show();

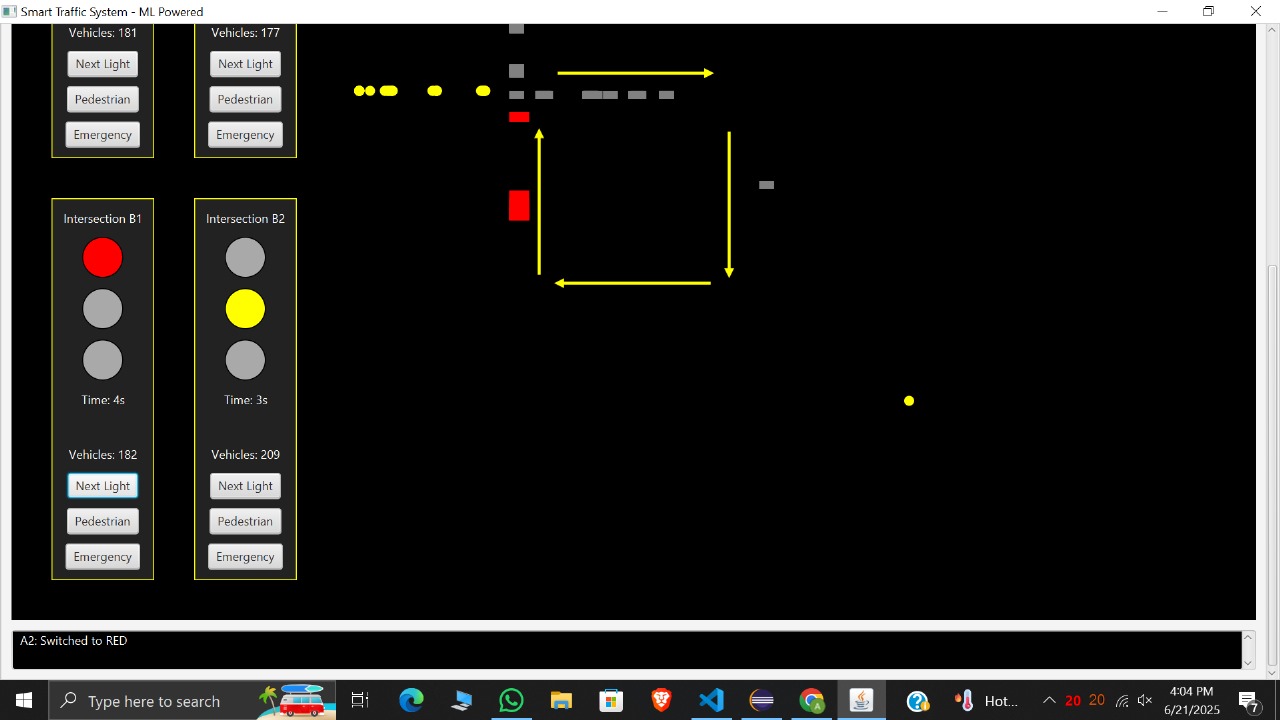
}

}

**Output:**

****





**3. Simulation Tool**

**GUI Overview:**

* Four intersections displayed in a grid.
* Each intersection has traffic lights, timers, vehicle counters, and control buttons.
* Animated vehicles and directional arrows visualize movement.
* Log area displays system events and state changes.
* Theme toggle for day/night visualization.

**Interactive Features:**

* Manual override for light cycle
* Pedestrian crossing request
* Emergency vehicle prioritization
* Real-time vehicle count updates and total dashboard1

**4. Test Cases and Results**

**Test Scenarios:**

**Normal Traffic:**

* Vehicles accumulate and green light duration adapts to average vehicle count via ML API.
* Vehicles move only on green; animation pauses otherwise.

**Emergency Scenario:**

* Emergency button forces green light and longer duration.
* Emergency vehicles (red rectangles) move regardless of light state.

**Pedestrian Scenario:**

* Pedestrian button triggers red light and walk timer.
* Pedestrian animation (yellow circles) moves during walk phase.

**Manual Override:**

* Next Light button cycles through states manually.

**Night/Day Mode:**

* Toggle changes background for visibility testing.

**Performance Results:**

* Light cycles adapt correctly to ML API predictions.
* Vehicle and pedestrian animations are synchronized with light states.
* Emergency vehicles bypass normal restrictions.
* System remains responsive under multiple simultaneous actions.

**5. Presentation**

**Key Points:**

**Challenges:**

* Integrating real-time ML API with GUI
* Ensuring smooth animation and state synchronization
* Handling concurrent user actions (emergency/pedestrian/override)

**Solutions:**

* Robust error handling for API failures (fallback to default green time)
* Modular timeline-based animation for vehicles and lights
* Clear separation of concerns in code structure

**Lessons Learned:**

* Importance of asynchronous operations in UI applications
* Value of user feedback (log area) for debugging and usability
* Scalability considerations for more intersections or complex scenarios

**6. User Guide**

**How to Use the System:**

**Start Application:**

* Launch the Java application; GUI opens with four intersections.

**Monitor Traffic:**

* Vehicle counts and light states update in real time.

**Control Lights:**

* Use "Next Light" to manually cycle through light states.
* Use "Pedestrian" to request a walk phase at any intersection.
* Use "Emergency" to prioritize emergency vehicles.

**Toggle Theme:**

* Click "Toggle Day/Night Mode" for visual preference.

**View Logs:**

* Check the log area for detailed system events and transitions.

**Observe Animations:**

* Watch vehicles and arrows to understand traffic flow and light impact.