SYNOPSIS OF EXPLORATORY PROJECT (EP)

ON

<u>Traffic Signal Simulation System</u>

Traffic Signal Simulation and Vehicle Flow Management

submitted in partial fulfilment of the requirements for the award of degree of

BACHELOR OF ENGINEERING

In

COMPUTER SCIENCE AND ENGINEERING

Submitted by:

Sneha (2210990856)
Harsh (2210991612)
Jinny Kapur (2210990462)
Khushi (2210991796)

Supervised By:

Roshni Mali
Lead Trainer
BridgeLabz

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CHITKARA UNIVERSITY INSTITUTE OF ENGINEERING AND TECHNOLOGY
CHITKARA UNIVERSITY, PUNJAB, INDIA

CONTENTS

1. Abstract

Traffic congestion has become one of the most critical challenges faced by urban environments today. The rapid growth in population, the increasing number of vehicles, and the lack of adaptive traffic control systems have severely impacted traffic efficiency, fuel consumption, and travel time. Traditional traffic signal systems rely primarily on static timing cycles rather than dynamic vehicle flow, resulting in poor handling of peak-hour traffic, emergency clearances, or irregular congestion patterns. Simulation models play a pivotal role in studying traffic behaviour and evaluating better strategies before implementing expensive real-world infrastructure changes.

This project introduces a **Traffic Signal and Vehicle Flow Simulation System using Java Multithreading**, aimed at providing a virtual yet realistic representation of a signal-controlled intersection. The system simulates two major operations functioning simultaneously: a multi-phase signal controller that switches cyclically between red, yellow, and green states, and a vehicle flow system that generates vehicles randomly and processes them based on the current signal phase. The simulation incorporates concurrency control mechanisms to prevent race conditions and ensures synchronized access to shared variables such as the current signal status and total queued vehicles.

The system logs every signal change and vehicle count to a structured CSV file, enabling researchers and system designers to analyse traffic patterns and response trends. The logged dataset can be used later to test performance, generate visualizations, and train machine learning algorithms for predictive traffic management. The proposed model not only demonstrates the practical application of Java multithreading but also establishes a foundation for future enhancements, including density-based adaptive switching, emergency vehicle priority clearance, and integration with IoT-enabled smart traffic systems. Ultimately, this simulation offers a cost-effective and controlled framework to experiment with traffic control strategies and evaluate their impact on urban mobility.

2. Introduction

Urban traffic systems are evolving rapidly, yet congestion continues to be a persistent problem in most cities. The imbalance between road capacity and the exponential rise in the number of vehicles has led to severe inefficiencies, especially during peak travel hours. Traffic signals play an essential role in regulating vehicle flow at intersections, but most systems still operate on preprogrammed, fixed-duration cycles that do not respond to dynamic traffic demands. As a result, intersections often become bottlenecks where vehicle queues increase despite the signal timings being mathematically correct on paper.

To solve real-world congestion, researchers and engineers must first understand the behavioural dynamics of traffic under different conditions before deploying hardware or infrastructure upgrades. **Traffic simulation** acts as an indispensable tool that enables experimentation and optimization without affecting real-world road users. By simulating vehicles, queues, signal phases, and response patterns, analysts can determine how signal timings influence waiting time, queue length, and traffic throughput.

The current project proposes a simulation of a **Java-based traffic signal and vehicle flow management system** that integrates multithreading to mimic real-world concurrency. While one thread regulates cyclic signal phases, another generates vehicles at randomized intervals, and both operate concurrently while sharing the same state information. Vehicles are allowed to exit only during the green phase, mimicking real-world traffic stop-and-go behaviour. A logging system captures signal states and queue counts at every transition, providing a detailed timeline of system performance.

The project reinforces core computer science concepts including concurrency, synchronization, thread communication, file management, shared resources, timing operations, and object-oriented design. By bridging theoretical operating system principles with a relatable real-world problem, the simulation serves as an educational yet practical initiative that can be extended into advanced intelligent traffic management solutions.

3. Methodology

The methodology follows an incremental and modular development approach designed to clearly separate system responsibilities. The simulation is structured into the following major stages and functional blocks:

3.1 Requirement Analysis & Conceptual Design

- Identified the core requirement: simulate a traffic signal with realistic vehicle flow and queueing.
- Defined main entities: Signal, Vehicle, TrafficService, LogService, and FileService.
- Established rules for realistic simulation: vehicle queue increases during Red/Yellow and decreases during Green.

3.2 Multithreading Model Design

- Two independent endless threads were planned:
    - **Signal Controller Thread**: cycles through Red → Green → Yellow with realistic durations.
    - **Vehicle Flow Thread**: generates random vehicle arrivals and processes exits only during Green.
- Race conditions were anticipated due to shared resource usage (vehicle count and current signal).

3.3 Synchronization Strategy

- The Signal class exposes synchronized methods for setting/getting state.
- AtomicInteger ensures thread-safe vehicle count updates.
- Controlled timing using precise sleep durations.

3.4 Logging and Data Collection

- LogService logs timestamp, signal state, and queue count to log.csv.
- This enables future throughput analysis and pattern plotting.

3.5 Testing Scenarios

- Verified simulation under multiple durations, high vehicle density, and rapid vehicle arrival rates.

- Ensured signals log correctly even if vehicle thread is running aggressively.

4. Tools and Technologies

4.1 Java (JDK 8+):

Java is the primary programming language used in this simulation because of its strong support for multithreading and platform-independent execution. The language provides built-in libraries and APIs for thread handling, time control, and concurrency operations, making it ideal for implementing real-time simulation of signal changes and vehicle flow.

4.2 Java Multithreading:

The simulation relies on multithreading to run two continuous processes simultaneously — the signal controller thread and the vehicle generator thread. Multithreading ensures that both processes operate in parallel without blocking each other, representing real-world traffic behaviour where vehicle arrival and signal switching occur independently.

4.3 Synchronization and Atomic Operations:

To prevent race conditions between threads accessing shared data (such as vehicle count and current signal), synchronized methods and AtomicInteger are used. Synchronization ensures that only one thread can modify shared state at a time, while atomic operations guarantee safe increment and decrement of vehicle numbers during high concurrency.

4.4 FileWriter and CSV-Based Logging:

A persistent logging system is implemented using Java's FileWriter, which stores data in a CSV format. Each change in signal state, along with the number of waiting vehicles, is recorded with a timestamp. This log file supports later analysis, visualization, dataset creation, and performance review of the simulation.

4.5 Object-Oriented Programming (OOP) Architecture:

The project follows an OOP-based modular structure where Signal, Vehicle, TrafficService, LogService, and FileService act as independent components. This design enhances code readability, debugging efficiency, scalability, and future upgrades — such as adding density-based signalling or priority routing for emergency vehicles.

4.6 Integrated Development Environment (IDE):

The simulation was developed and tested using IDEs such as IntelliJ IDEA / Eclipse / VS Code. These tools simplify debugging, provide real-time console monitoring, and allow proper organization of source files, all of which are crucial when validating thread timings and concurrency behaviour.

5. Project Plan

Phase	Duration	Tasks
Phase 1 — Requirement Analysis and System Planning	**Week 1**	Finalizing requirements; identifying traffic behaviour model; defining signal cycle duration and vehicle handling rules.
Phase 2 — System Design and Thread Architecture	**Week 2**	Designing classes (Signal, Vehicle, Service modules); structuring multithreading model; planning logging format; preparing basic flow diagrams.
Phase 3 — Core Implementation (Signal Cycle + Vehicle Flow Threads)	**Week 3**	Implementing signal cycle (Red–Green–Yellow), vehicle generator logic, synchronized shared state access, vehicle count management, and initial testing.
Phase 4 — Logging, Optimization & Testing	**Week 4**	Implementing CSV logging, fixing concurrency issues, stress-testing timing scenarios, validating real-time queue behaviour, and optimizing simulation stability.
Phase 5 — Documentation & Final Review	**Final Days**	Preparing project report, refining console output, cleaning code, preparing presentation/synopsis for evaluation.