GreenFlow: Decentralized Traffic Signal Optimization

Traffic Simulator Project Documentation

Overview

This project is a modular, extensible traffic simulation platform designed for testing multi-agent control strategies. It simulates intersections with discrete time steps, supporting custom agents for observation, control, and coordination.

Architecture

Core Components (`traffic_simulator.py`)

- `Lane`: Represents a single lane of traffic.
    - **Queue**: Stores arrival times of vehicles.
    - **Metrics**: Calculates current wait time (average of currently queued vehicles) and tracks total cleared vehicles.
- `Intersection`: Manages four approaches (N, E, S, W).
    - **Phases**: `NS_GREEN` and `EW_GREEN`.
    - **Logic**: Handles phase switching (timer-based or manual) and vehicle departure (probabilistic clearance).
    - **State**: Reports queue lengths, current phase, and average wait times.
- `TrafficSimulator`: The main engine.
    - **Time**: Discrete steps (1 step = 1 second).
    - **Traffic Generation**: Poisson process for vehicle arrivals.
    - **Execution**: Iterates through all intersections and steps them forward.

Agents

- `ObserverAgent` (`observer_agent.py`):
    - Runs in a background thread.
    - Polls the simulator state at each step.
    - Computes derived metrics (rolling average wait time, queue sums).
    - Publishes state updates via a callback (JSON format).
- `ControllerAgent` (`controller_agent.py`):
    - Subscribes to Observer updates.
    - Implements rule-based control logic:
        - **Switch**: If opposing queue is significantly larger.
        - **Extend**: If current queue is long (unless advised otherwise).
    - Sends actions (`SWITCH`) to the simulator.
- `CoordinatorAgent` (`coordinator_agent.py`):
    - Monitors the global network state.
    - Detects spillback (downstream congestion).
    - Sends advisories (e.g., `avoid_extend`) to upstream Controller Agents to prevent gridlock.

Dashboard

- **Web Dashboard (Flask)**:
    - **Backend (`server.py`)**: Runs the simulator loop and exposes API endpoints (`/api/state`, `/api/control`).
    - **Frontend**: HTML/CSS/JS interface for real-time visualization of queues, phases, and wait time history.

Key Algorithms

Wait Time Calculation

The simulator calculates the **Average Wait Time of Currently Queued Vehicles**.

- For each vehicle `v` in the queue: `wait_time = current_time - v.arrival_time`.
- `Lane Average` = `sum(wait_times) / len(queue)`.
- `Intersection Average` = Weighted average of all lanes.
- The `ObserverAgent` further smooths this value using a 30-second rolling average window to reduce volatility.

Control Logic

The `ControllerAgent` uses a simple heuristic:

1. **Min Green**: Enforce minimum green duration (5s).
2. **Max Green**: Enforce maximum green duration (25s).
3. **Switch Rule**: If `opposing_queue > current_queue + 5`, switch phase.
4. **Extend Rule**: If `current_queue > 10`, keep green (unless `avoid_extend` advisory is active).

How to Run

Prerequisites

- Python 3.x
- Flask (`pip install flask`)

Running the Dashboard

1. Navigate to the project directory:

   ```
   cd /home/shrinesh/antigravity_test/traffic_agent
   ```

2. Start the server:

   ```
   python3 server.py
   ```

3. Open your browser to `http://127.0.0.1:5000`.

Running Demos

- **Basic Simulation:** `python3 demo.py`
- **Observer Demo:** `python3 demo_observer.py`
- **Controller Demo:** `python3 demo_controller.py`
- **Coordinator Demo:** `python3 demo_coordinator.py`