SUMO Simulation Module Requirements Analysis

Module One: Urban Traffic Network Modeling and Traffic Rule Configuration

1.1 Import Basic Road Network

Requirement Description: Build a city-level SUMO road network as the foundation for traffic simulation.

Main Input: Original map data, i.e., OSM map data .osm file

Main Output: SUMO standard road network .net.xml file

Constraints: Manually check and fix network errors, disconnected sections, or incorrect attributes caused by OSM import; supplement missing nodes if necessary.

Development Implementation:

- Use SUMO's built-in tool netconvert
- Or manually edit or use the netedit tool to fine-tune the generated network

Interaction Requirement: None

Development Phase: MVP Phase

Estimated Development Time: 3-5 PD

1.2 Road and Intersection ID/Name Standardization

Requirement Description: Define unique and readable id and name for each road (edge) and intersection (junction).

Main Input: SUMO road network .net.xml file

Main Output: Updated SUMO road network file

Constraints: IDs must be unique throughout the entire network file and comply with naming conventions.

Development Implementation: netconvert automatically generates IDs when converting OSM data. netedit or scripts can be used to modify/standardize IDs and add name attributes.

Interaction Description: Backend and AI modules use these unique IDs to precisely specify the objects of operation.

- **Interaction Fields:** edgeID (string), junctionID (string)

Development Phase: MVP Phase

Estimated Development Time: 1-2 PD

1.3 Road Attribute Configuration

Requirement Description: Support setting physical and traffic attributes for each road, such as direction, number of lanes, maximum speed, vehicle type restrictions, etc., to meet traffic regulation

requirements.

Main Input: SUMO road network .net.xml file

Main Output: Updated SUMO road network file, containing detailed road attributes in <edge> and <lane> elements.

Constraints: Road attribute settings must comply with actual physical limitations and traffic rules.

Development Implementation: Use netedit for manual editing, or use scripts to modify the .net.xml file.

Interaction Description: Road attributes can be queried via TraCI.

- **Interaction Fields:** edgeID (string), laneID (string), traveltime (int), density (float), and so on
- **TraCI Interface:** Get fields

Development Phase: MVP Phase

Estimated Development Time: 2-3 PD

1.4 Configure Traffic Priority

Requirement Description: Configure traffic logic in the road network, such as priority roads, auxiliary roads, and roundabouts, to meet traffic regulation requirements.

Main Input: SUMO road network .net.xml file

Main Output: Updated SUMO road network file.

Constraints: Traffic rules must comply with local traffic regulations.

Development Implementation:

- Set junction attributes, such as priority, using netedit.
- Define attributes for the connections within these intersections.

Interaction Requirement: None

Development Phase: MVP Phase

Estimated Development Time: 1-2 PD

Module Two: Traffic Flow Configuration and Event Modeling

2.1 Vehicle Type Definition

Requirement Description: Define various types of vehicles used in the simulation, including emergency vehicles, and their physical and behavioral characteristics, such as length, maximum speed, acceleration, driving model parameters, permissions, etc.

Input: Vehicle type parameter list (emergency vehicles have permission to ignore traffic lights).

Output: <vType> elements in .rou.xml.

Core Constraints: Vehicle parameters should conform to actual vehicle characteristics.

Development Implementation: Manually write XML <vType> elements in the configuration file, or generate them via Python scripts.

Interaction Description: Backend can query the types and dynamic attributes of vehicles in the simulation.

- **Interaction Fields:** typeID (string), accel (float), length (int), maxSpeed (int), decel (float), sigma (float) and so on
- **TraCI Interface:** Get fields

Development Phase: MVP Phase

Estimated Development Time: 1-2 PD

2.2 Static Traffic Flow Configuration

Requirement Description: Support configuring static, predefined traffic flow, i.e., specifying the departure time, origin, destination, and route for individual vehicles, such as emergency vehicles and test vehicles.

Input: Vehicle ID, departure time, origin/destination edge ID.

Output: <vehicle> and <route> elements in .rou.xml.

Constraints: Origin and destination edges must exist in the road network; avoid overlapping conflicts; departure time must be non-negative.

Development Implementation: Manually write XML in the configuration file, or generate via Python scripts.

Interaction Description: Backend and AI modules can query data for specific vehicles and routes.

- **Interaction Fields:** vehID (string), routeID (string), depart (float) and so on
- **TraCI Interface:** Get fields

Development Phase: MVP Phase

Estimated Development Time: 1-2 PD

2.3 Setting Large-Scale Random Traffic Flow for Different Time Periods

Requirement Description: Support large-scale random traffic flow that can change dynamically over time, to simulate vehicle travel behavior and traffic density in a city during different periods.

Input: Vehicle type distribution, time period definitions, traffic flow parameters corresponding to each time period, random seed, etc.

Output: Vehicle flow or trip definitions in SUMO route files.

Constraints: Generation of random traffic flow should be based on reasonable travel distribution assumptions; flow should not be too large to prevent system lag.

Development Implementation: Use Python scripts combined with SUMO's Python tools to generate trips or flows.

Interaction Requirement: None

Estimated Development Phase: MVP (implement basic random flow), Feature Improvement (implement complete time-varying density)

Estimated Development Time: 2-3 PD (MVP basic random flow) + 2-3 PD (complete time-varying and refinement)

2.4 Special Event Modeling, Triggering, and Ending

2.4.1 Special Event Modeling

Requirement Description: Define and configure models for various types of special traffic events supported in SUMO simulation.

Input:

- List of special event types
- Specific configuration parameter definitions required for each event type:
  - Two-vehicle collision/Vehicle breakdown: Vehicle ID, Lane ID, Target set speed (configure a fixed value)
  - Lane closure: One or more Lane IDs, Prohibited vehicle types (defaults to ["all"], i.e., all vehicles)
  - Road closure: One or more Road IDs, Prohibited vehicle types
  - Emergency Vehicle: Vehicle ID, Start Time, End Time, and Travel Route

Output:

- A set of Python scripts/modules containing encapsulated functions or class methods for each special event type.
- A configuration parameter document for special events, clearly indicating which parameters have default values and which must be provided by the backend.

Constraints: Event models should reflect the traffic impact of corresponding real-world events as accurately as possible. Encapsulated interfaces should be clear, easy to use, and have well-defined parameters.

Development Implementation: Write independent Python functions for each supported special event type; these functions should include logic for both applying event effects and reverting event effects (recovery).

Interaction Description: None

Development Phase: Feature Improvement

Estimated Development Time: 3-5 PD (depends on the number and complexity of supported event types, modeling each event type takes approx. 0.5-1 PD)

2.4.2 Special Event Triggering

Requirement Description: Support dynamically triggering modeled special traffic events during SUMO simulation runtime via backend commands.

Input: Command from backend (JSON), including special event type and target road ID/lane ID, vehicle ID.

Output:

- The status of the corresponding road segment/vehicle in the SUMO simulation environment changes dynamically according to the event model.

- Return a JSON message to the backend indicating whether the trigger was successful, including a status code

Constraints: Self-check logic must accurately reflect whether the event has actually occurred in SUMO. Invalid event types or parameters should be handled appropriately, returning a failure.

Development Implementation: The TraCI control script of the SUMO module needs to provide an interface. Upon receiving a request, it calls the corresponding modeling function from 2.4.1 based on the event type. After calling the modeling function, execute self-check logic, then return the boolean result to the backend.

Interaction Description: The backend triggers special events by calling a specific TraCI encapsulated interface provided by the SUMO module (e.g., an API endpoint provided by a Python script). The backend sends a request JSON containing the event type and parameters and waits synchronously for the execution result returned by the SUMO module.

- **Interaction Fields:**
  - Input: Request JSON containing event type and parameters, including special event type and target road ID/lane ID, vehicle ID.
  - Output: JSON message including a status code indicating success/failure, and an error message

Development Phase: Feature Improvement (developed synchronously with 2.4.1, but interface exposure and joint debugging are here)

Estimated Development Time: 2-3 PD (mainly for interface implementation, self-check logic, and joint debugging with the backend)

2.4.3 Special Event Ending

Requirement Description: Support dynamically ending triggered special traffic events during SUMO simulation runtime via backend commands, restore the affected road/vehicle status to the normal state before the event, and remove the broken-down vehicle

Input: Command from backend (JSON), including special event type and target road ID/lane ID, vehicle ID.

Output:

- Restoration of the status of the corresponding road segment/vehicle in the SUMO simulation environment.
- The broken-down vehicle is removed from the simulation.
- Return a JSON message to the backend indicating whether the ending was successful, including a status code and an error message.

Constraints: Recovery logic must accurately restore the SUMO state to a reasonable state. Self-check logic must accurately reflect whether the state has been restored.

Development Implementation: The backend ends special events by calling a specific TraCI encapsulated interface provided by the SUMO module. Upon receiving a request, it calls the corresponding event recovery function from 2.4.1 based on the event type. After calling the function, execute self-check logic, then return the boolean result to the backend.

Interaction Description: When the backend system determines that a special event should end, it calls a specific TraCI encapsulated interface provided by the SUMO module to end the event. The backend sends a request containing the event type and parameters required for recovery and waits synchronously for the execution result returned by the SUMO module.

- **Interaction Fields:**
  - **Input:** Request JSON containing event type and parameters, including special event type and target road ID/lane ID, vehicle ID.
  - **Output:** JSON message including a status code

Development Phase: Feature Improvement (immediately following 2.4.2)

Estimated Development Time: 2-3 PD (mainly for recovery logic, interface implementation, self-check logic, and joint debugging with the backend)

2.5 Reproducibility Control

Requirement Description: Ensure the reproducibility of the traffic flow simulation process. By configuring a fixed random seed, multiple simulation runs can produce the same results, facilitating debugging, testing, and comparative analysis.

Input: A fixed integer as a random seed.

Output: Reproducible simulation results.

Constraints: All processes involving randomness should be affected by this seed.

Development Implementation: Set a fixed random seed in relevant Python scripts and specify the seed parameter when starting SUMO.

Interaction Description: The backend system needs to ensure the consistency of the SUMO simulation environment when conducting algorithm evaluation or scenario replay. The SUMO module should provide a way to set the random seed.

- **TraCI Interface (Configuration):** SUMO start-up parameter seed
- **Interaction Fields:** seed_value

Estimated Development Phase: MVP

Estimated Development Time: 0.5 PD (mainly for configuration and validation)

2.6 Real-time Segment Status

Requirement Description: Enable the backend system to query and obtain real-time traffic data for specified roads in the SUMO simulation.

Input: Road ID, list of required data names (vehicle_ids_list, vehicle_count, average_speed, occupancy, and so on).

Output: Real-time status JSON message for the lane, including all specified data fields:

- edge_id, vehicle_ids_list, vehicle_count, average_speed, occupancy, and so on.

Constraints:

- Data query response should be as real-time as possible to meet the immediacy requirements for monitoring and decision-making.

- Interface design should be stable, efficient, and capable of handling data requests for multiple road segments.
- Returned data units and definitions should be clear and consistent.

Development Implementation: The TraCI control script of the SUMO module needs to provide one or more interfaces. Upon receiving a request, use TraCI functions to obtain data for each road ID in the request. Organize the queried data into the specified JSON format for each road ID and return it to the backend.

Interaction Description: The backend system requests real-time traffic data for specified road segments by calling a specific TraCI encapsulated interface provided by the SUMO module. The request includes the specified road ID and a list of required data names. It then receives the specified road segment data packaged in JSON format.

Development Phase: MVP

Estimated Development Time: 1.5 - 2.5 PD

Module Three: Traffic Light Control and Interface Integration

3.1 Traffic Light Identification and Network Association

Requirement Description: Add traffic lights to intersections, ensuring each light has a unique id and name.

Input: SUMO road network .net.xml file.

Output: Each traffic light in the file has a unique id and name.

Constraints: Traffic light ID must correspond to the ID of the traffic light intersection already defined in the network.

Development Implementation: Automatically created by netconvert, or added/edited using netedit.

Backend Interaction Description: Backend queries a specific traffic light by its ID.

- **Interaction Fields:** tlsID
- **TraCI Interface:** Get fields

Estimated Development Phase: MVP

Estimated Development Time: 0.5 PD (completed in conjunction with road network generation)

3.2 Traffic Light Logic Configuration

Requirement Description: Modify the basic signal control logic of automatically generated traffic lights to meet realistic traffic rules.

Input: Traffic light ID, list of phase definitions, phase duration, phase switching sequence.

Output: Configuration file defines the logic for each traffic light.

Constraints: Phase definitions must ensure traffic safety and avoid conflicts. Total cycle duration should be within a reasonable range.

Development Implementation: Modify the XML file containing traffic light definitions using Python scripts.

Backend Interaction Description: Backend can query the current phase and basic timing plan of a specified traffic light.

- TraCI interface implements data acquisition for specified traffic lights.

Development Phase: MVP (at least covering key intersections)

Estimated Development Time: 2-3 PD (depends on the number of intersections and logic complexity)

3.3 Support Manual Intervention and AI Traffic Light Control

Requirement Description: Support administrators in modifying signal status in real-time via backend commands.

Input: Target traffic light ID, backend control command.

Output: The status of the corresponding traffic light in the SUMO simulation changes in real-time according to the command; JSON message includes a status code indicating success/failure, and an error message.

Development Implementation: Write Python scripts using the TraCI library to receive external commands and call the corresponding TraCI interface to modify the signal status. Perform a self-check to determine if the modification was successful.

Backend Interaction Description: Backend modifies the status or phase duration of a specified traffic light.

- **TraCI Interface:** Control specified traffic light
- **Interaction Fields:** tlsID (string), duration (int), stateString (string), JSON

Development Phase: MVP/Feature Improvement

Estimated Development Time: 2-3 PD

3.4 Emergency Vehicle Priority Passage

Requirement Description: Implement priority passage for emergency vehicles. The system must be able to dynamically track the position, speed, and route of emergency vehicles, and allow manual intervention to set traffic lights in conflicting directions on their route to red. After the emergency vehicle passes an intersection, the traffic light at that intersection should revert to its normal control logic.

Input:

- Emergency vehicle information: Vehicle ID
- Signal control command: Intersection ID, target traffic light ID, backend control command

Output:

- Emergency vehicle ID, list of intersection IDs, list of traffic light IDs, list of traffic light statuses, real-time position
- Priority passage behavior of emergency vehicles in the simulation environment.
- Dynamic changes to intersection traffic lights in the simulation environment

Development Implementation: Use TraCI to get emergency vehicle information and continuously query its real-time position and speed. Based on the obtained emergency vehicle route, analyze all the signal-controlled intersections it will pass through.

Interaction Description:

- The backend continuously queries the emergency vehicle's status via TraCI and sends it to the frontend to mark the vehicle's position on the map, while simultaneously displaying information about all intersections and their signal statuses along the route in a control panel.
- After monitoring the emergency vehicle on the map, the frontend user manually modifies the traffic light status through the frontend interface. This utilizes the mechanism from requirement 3.3 (Support manual intervention in traffic light logic).
- **Backend to Frontend transmission:** JSON message, including the output vehicle ID, list of intersection and traffic light IDs, and the status of traffic lights.

Development Phase: Feature Improvement

Estimated Development Time: 3-5 PD

3.5 Collect Control Log Data

Requirement Description: Must be able to generate and provide historical control status and operation log data for traffic lights.

Input: Traffic light ID, time, current phase, duration, content of the applied control command.

Output: Structured log data stream or batch data packets for the backend system to receive and store.

Constraints: Log format should be easy to parse and query.

Development Implementation: In Python scripts that control traffic lights via TraCI or detect changes in traffic light status, add logic to capture and format the above log fields. After each traffic light status change or external control, relevant log information will be collected.

Interaction Description: The SUMO module pushes status changes to the backend for recording in real-time via TraCI, i.e., sends a JSON object containing log data to the backend API.

- **Interaction Fields:** JSON message including traffic light ID, signal status and duration, success status, error message.

Development Phase: Integration and Optimization

Estimated Development Time: 1-2 PD

Module Four: Pedestrian and Pedestrian-Vehicle Interaction Modeling

4.1 Sidewalk Modeling

Requirement Description: Define sidewalk networks or areas/paths allowing pedestrian passage in the SUMO road network.

Input: Sidewalk information.

Output: Updated SUMO road network file, containing sidewalk elements or lanes allowing pedestrian passage.

Core Constraints: Pedestrian path network needs to have connectivity.

Interaction Requirement: None

Development Implementation:

- netedit for manual addition/modification.
- When importing .osm files, **-osm.sidewalk** can be used to import sidewalks for all roads.

Development Phase: Feature Improvement

Estimated Development Time: 1-2 PD

4.2 Crosswalk Modeling

Requirement Description: Set up pedestrian crossings at intersections and associate them with vehicle right-of-way logic (e.g., vehicles yielding to pedestrians).

Input: Intersections already defined in the network file, pedestrian crossing ID and position, and connection relationships.

Output: Updated network file where intersections include pedestrian crossing definitions.

Constraints: Crosswalk settings should comply with actual traffic design.

Development Implementation:

- Use netedit to add/edit pedestrian crossings.
- Crossings may be defined explicitly in plain XML input when describing connections (plain.con.xml) using the XML element `crossings`. They can also be placed with netedit.

Interaction Requirement: None

Development Phase: Feature Improvement

Estimated Development Time: 1 PD

4.3 Pedestrian Type Definition

Requirement Description: Define pedestrian types and their behavioral parameters, and configure pedestrian travel plans.

Input: Pedestrian type parameters, pedestrian travel plans.

Output: Updated file containing pedestrian definitions.

Development Implementation: Define pedestrians in the configuration file.

Interaction Requirement: None

Development Phase: Feature Improvement

Estimated Development Time: 1-2 PD

4.4 Traffic Light Coordination and Safety Control

Requirement Description: Configure pedestrian traffic lights for intersections and coordinate their phases with vehicle traffic light phases to ensure pedestrian crossing safety.

Input: Intersections with defined crosswalks, pedestrian traffic light ID/name, pedestrian traffic light phases, crossing/clearance time, coordination plan with vehicle traffic lights.

Output: Traffic light logic definition includes corresponding pedestrian traffic light data.

Constraints: Coordination between pedestrian and vehicle traffic lights must ensure safety.

Development Implementation: Modify the XML file containing traffic light definitions using Python scripts.

Interaction Requirement: None

Development Phase: Integration and Optimization

Estimated Development Time: 1-2 PD

4.5 Pedestrian Flow Configuration

Requirement Description: Support setting different pedestrian flow densities or travel demands for different time periods to simulate pedestrian activity in real-world scenarios.

Input: Time period, number of pedestrians, pedestrian origin and destination distribution.

Output: Configuration file contains definitions for pedestrians and pedestrian flows with corresponding numbers and departure times.

Constraints: Pedestrian flow settings should align with the scenario settings.

Development Implementation: Use Python scripts combined with SUMO's Python tools to generate trips or flows.

Interaction Requirement: None

Development Phase: Integration and Optimization

Estimated Development Time: 1 PD