We use the before described models for our agent-based simulation. Each simulation step simulates a time interval Δt, after which the state of all the cars is captured. This new state is only determined by the previous and there are no intermediate states in between, so every car is first locally updated before the new state is committed globally.

The implementation of the **??** and the **??** [**?**] is done in the class `DriverModel`.

To run simulations, one can instantiate the `Simulation` class with all its necessary parameters. For this we use `main.py`. This file handles creating multiple simulations, creating the data and saving it to a JSON file for use in the visualizer (`Metrics`), which it also calls afterwards.

For every simulation, an instance of the class `Simulation` is created. This class gets a list of car types and other necessary parameters like the road length and the amount of lanes, and handles creating a `Road` object. This `Road` will insert `Car` objects into the road, by randomly selecting a lane and checking if there is a sufficient distance to the leading vehicle, so cars are only inserted if it can be done safely. Once cars are at the end of the road, they are deleted. During a simulation step, every `Car` object is updated locally, then the new state is made visible globally.

A `Car` object can find the surrounding cars and update its velocity and position using the `DriverModel` class. Every time the state is updated, all cars have a probability p_{fail} of failing, which is how we model random perturbations. For our visualization, the position, velocity and other information about each `Car` object is saved each time they are updated.

The `Metrics` class handles the creation of all the graphs and plots which will be presented in this paper.

For further information, consult the *documentation* in the project repository.