Template Creation

Complete the installation procedure described in README.md

In this exercise, we are going to explore the Geoscenario-Orchestrator exceptions and success criteria. The goal of this exercise is to get familiar with rule creation and the type of errors that Geoscenario-Orchestrator can raise.

Given the following precedence table:

$$\text{STOP\_AT} \succ \text{TRACK\_SPEED}$$

And the following features:

"goal_point.x", "goal_point.y",

"goal_point_frenet.s", "goal_point_frenet.d"

"lane_config.max_velocity", "lane_config.left_bound", "lane_config.right_bound",
"lane_config._left_relationship", "lane_config._right_lane", "lane_config._right_relationship",
"lane_config.stopline_pos"

"lane_config._left_lane.max_velocity", "lane_config._left_lane.left_bound",
"lane_config._left_lane.right_bound", "lane_config._left_lane._left_relationship",
"lane_config._left_lane._right_lane", "lane_config._left_lane._right_relationship",
"lane_config._left_lane.stopline_pos"

"lane_config._right_lane.max_velocity", "lane_config._right_lane.left_bound",
"lane_config._right_lane.right_bound", "lane_config._right_lane._left_relationship",
"lane_config._right_lane._right_lane", "lane_config._right_lane._right_relationship",
"lane_config._right_lane.stopline_pos"

"simulation.sim_time", "simulation.tick"

"vehicle_state.x", "vehicle_state.x_vel", "vehicle_state.x_acc", " vehicle_state.y"
" vehicle_state.y_vel", " vehicle_state.y_acc", "vehicle_state.z", "vehicle_state.z_vel",
"vehicle_state.z_acc", "vehicle_state.s" "vehicle_state.s_vel", "vehicle_state.s_acc", "vehicle_state.d",
"d_vel", "vehicle_state.d_acc", "vehicle_state.yaw", " vehicle_state.steer"

In a json file named **ManeuverTemplates.json** at the root of the github folder. Define, for each of the following violations, *GoalOvershot* and *ScenarioTimeout*, a set of rules (i.e., an array) that one should apply when such violation is raised.

A rule has the following syntax

```
{
        "constraints": [
                {
                        "feature_name": STRING
                        "value":        BOOL | INT | FLOAT | STRING,
                        "option":       undefined | "CARDINALITY" | "GREATER_EQUAL_THAN" |
                                        "LOWER_EQUAL_THAN" | "IS_ARRAY | "NEGATE"
                }
        ],
        "consequent": {
                "MANEUVER": { "value": true },
                "parameters.parameter_name": {
                        "value":    BOOL | INT | FLOAT | STRING,
                        "option":   undefined | "INTERPRETED" | "MINIMUM",
                        "scaling":  undefined | FLOAT
                }
        }
}
```

For the constraints, feature_name is the string of your choice in the feature list of page 1. Value can be true, 6, "my_string", etc. For the constraint's options, the behaviour is:

undefined: Is the property set to be exactly the specified value?

CARDINALITY: Is the length of the array at the specified property match the specified value?

GREATER_EQUAL_THAN: Is the property greater equal than the specified value?

LOWER_EQUAL_THAN: Is the property smaller equal than the specified value?

IS_ARRAY: Is the specified property an array? The answer to the test must match the specified value.

NEGATE: Is the property not strictly equal to the specified value?

For the consequent, MANEUVER is one of the maneuvers specified in the precedence table.

"STOP_AT" maneuver needs to be parametrized with parameters.location.s, parameters.location.d

"TRACK_SPEED" maneuver needs to be parametrized with parameters.speed.

Scaling will multiply the specified value by the specified float (i.e., feature := value * scaling)

For the consequent options, the behaviour is:

undefined: The output feature parameters.parameter_name must be assigned with the specified value.

INTERPRETED: The output feature parameters.parameter_name must be assigned with the value of the input feature, i.e. the string of your choice in the feature list of page 1.

MINIMUM: The output feature parameters.parameter_name must be assigned with the minimum value of the set referenced by the input feature, i.e., the string of your choice in the feature list of page 1.

An example of a rule can be:

```
{
        "constraints": [
                {
                        "feature_name": "vehicle_state.x",
                        "value":        0,
                        "option":       "GREATER_EQUAL_THAN"
                },
                {
                        "feature_name": "vehicle_state.x",
                        "value":        10,
                        "option":       "NEGATE"
                },

        ],
        "consequent": {
                "TRACK_SPEED": { "value": true },
                "parameters.speed": {
                        "value":    "vehicle_state.x",
                        "options": "INTERPRETED",
                        "scaling":   0.2
                }
        }
}
```

This rule translates to

```
IF      vehicle_state.x >= 0
AND     vehicle_state.x ≠ 10
THEN    TRACK_SPEED := true;
        parameters.speed := valueOf("vehicle_state.x") * 0.2 km/h
```

For each of the following questions, rename ManeuverTemplates.json into q#_ManeuverTemplates.json to preserve your work. Note that only ManeuverTemplates.json will be considered when running the command **npm start**.

Q1. Raise the "Illogical Template" Exception

Find a set of template rules such that the Geoscenario-Orchestrator claims after a **single iteration** that a template is illogical.

If you succeed, when the orchestrator will terminate, it shall output something like:

```
Error: [Q1 Achieved] Illogical Template:
```

Q2. Raise the "Search Window Expansion" Exception

Find a set of template rules such that the Geoscenario-Orchestrator claims after **more than one iteration** that it needs to update one of the template rules by increasing the search window. For now, it is an exception since I have not yet migrated that piece of the algorithm.

If you succeed, when the orchestrator will terminate, it shall output something like:

```
Error: [Q2 Achieved] Search Window Expansion:
```

Q3. Successfully Solve the Lane Following Scenario

Find a set of template rules such that the Geoscenario-Orchestrator claims that all requirements are satisfied during **training** and **testing**.

If you succeed, when the orchestrator will terminate it shall output something like:

(index)	Success Rate	Triggered Violations
/src/scenarios/basic/lane_following_test.osm	'100.00% (1 out of 1)'	''

```
[Q3 Achieved] Sastify every requirements! Ready to deploy!
```

Q4. Coincidentally Solve the Lane Following Scenario

Find a set of template rules such that the Geoscenario-Orchestrator claims that all requirements are satisfied during **training** but for which **some of the requirements are unmet during testing**.

If you succeed, when the orchestrator will terminate it shall output something like:

(index)	Success Rate	Triggered Violations
/src/scenarios/basic/lane_following_test.osm	'0.00% (0 out of 1)'	'(1) ScenarioTimeout'

```
[Q4 Achieved] Satisfy training scenarios but failed to generalize to testing scenarios
```