

File: ./software/data\_analysis/analysis.py

```
1 import argparse
2 from classes.parsing import Parser
3 import json
4
5 parameters_file = "analysis_parameters.json"
6
7 argparser = argparse.ArgumentParser(description=__doc__)
8 argparser.add_argument(
9     '--save_location', '-s',
10     required=True,
11     help='Where to save the analysis')
12 args = argparser.parse_args()
13
14 def main():
15     try:
16         parameters = read_parameters()
17         parser = Parser(parameters, args.save_location)
18         try:
19             parser.parse()
20         except Exception as e:
21             print("An exception occurred parsing the recording files: ", e)
22         try:
23             parser.save()
24         except Exception as e:
25             print("An exception occurred saving the analysed data: ", e)
26     finally:
27         print("Finished analysing.")
28
29 def read_parameters():
30     try:
31         with open("analysis_parameters.json", "r") as file:
32             parameters = json.load(file)
33     except Exception as e:
34         print("Something wrong happened trying to get data from {}".format(parameters_file), e)
35
36
37 if __name__ == '__main__':
38     main()
```

File: ./software/data\_analysis/classes/parsing.py

```

1 from classes.data_analysers import CollisionDataAnalyser, VehicleLightDataAnalyser, RouteDataAnalyser, LaneMarkingDataAnalyser, SpeedingDataAnalyser, RoadTrafficDataAnalyser
2 import xml.etree.ElementTree as ET
3 from xml.dom.minidom import parseString
4
5 # Creates a DataAnalyser given the analysis type.
6 class DataAnalyserFactory:
7     @staticmethod
8     def create_analyser(analysis_type, scenarios, participants):
9         if analysis_type == 'collision_data':
10             return CollisionDataAnalyser(scenarios, participants)
11         elif analysis_type == 'lane_marking_violation_data':
12             return LaneMarkingDataAnalyser(scenarios, participants)
13         elif analysis_type == 'vehicle_light_misuse_data':
14             return VehicleLightDataAnalyser(scenarios, participants)
15         elif analysis_type == 'route_data':
16             return RouteDataAnalyser(scenarios, participants)
17         elif analysis_type == 'speeding_data':
18             return SpeedingDataAnalyser(scenarios, participants)
19         elif analysis_type == 'road_traffic_violation_data':
20             return RoadTrafficDataAnalyser(scenarios, participants)
21         else:
22             raise ValueError('Invalid analyser type')
23
24 # Used to read and process recorded data according to the specification provided
25 # in analysis parameters.json
26 class Parser():
27
28     def __init__(self, parameters, save_location):
29         self.participants = parameters["participants"]
30         self.scenarios = parameters["scenarios"]
31         self.metrics = parameters["metrics"]
32         self.save_location = '../data/analysed_data/data/{}'.format(save_location)
33
34     try:
35         self analysers = []
36         for metric in self.metrics:
37             analyser = DataAnalyserFactory.create_analyser(metric, self.scenarios, self.participants)
38             self analysers.append(analyser)
39     except Exception as e:
40         print("Could not initiate all the analysers correctly", e)
41
42 # Make all the DataAnalysers do their analyses
43 def parse(self):
44     for analyser in self analysers:
45         analyser.run()
46
47 # Save the analysed data
48 def save(self):
49     try:
50         for analyser in self analysers:
51             analyser.save_points(self.save_location + "/points")
52         self.save_participants()
53         self.save_scenarios()
54     except Exception as e:
55         print("Something wrong happened trying to save the analysed data:", e)
56
57 # Write relevant data to the participants.xml file
58 def save_participants(self):
59     try:
60         root = ET.Element("Participants")
61         for participant in self.participants:
62             participant_element = ET.SubElement(root, "Participant")
63             ET.SubElement(participant_element, "Name").text = str(participant)
64             analysis = ET.SubElement(participant_element, "Analysis")
65             # For each participant write what each of the analysers calculated
66             total_amount_of_points = 0
67             for analyser in self analysers:
68                 ET.SubElement(analysis, analyser.save_file_xml_element_name_average).text = str('%2f' % (analyser.participant_score[participant] / len(self.metrics)))
69                 if analyser.category != "route_data":
70                     total_amount_of_points += analyser.participant_score[participant]
71             ET.SubElement(participant_element, "TotalNumberOfPenaltyPoints").text = str('%2f' % total_amount_of_points)
72         xml_str = ET.tostring(root, 'utf-8', method='xml')
73         xml_formatted = parseString(xml_str).toprettyxml(indent=" ")
74         with open("{}participants.xml".format(self.save_location, participant), "w+") as file_writer:
75             file_writer.write(xml_formatted)
76     except IOError as e:
77         print("An error occurred trying to write analytics to participants.xml file:", e)
78     except Exception as e:
79         print("Something wrong happened trying to save participants analytics:", e)
80
81 # Write relevant data to the scenarios.xml file
82 def save_scenarios(self):
83     try:
84         root = ET.Element("Scenarios")
85         for scenario in self.scenarios:
86             scenario_element = ET.SubElement(root, "Scenario")
87             ET.SubElement(scenario_element, "Name").text = str(scenario)
88             analysis = ET.SubElement(scenario_element, "Analysis")
89             # For each scenario write what each of the analysers calculated
90             for analyser in self analysers:
91                 ET.SubElement(analysis, analyser.save_file_xml_element_name_average).text = str('%2f' % (analyser.scenario_score[scenario] / len(self.participants)))
92             # Write to file
93             xml_str = ET.tostring(root, 'utf-8', method='xml')
94             xml_formatted = parseString(xml_str).toprettyxml(indent=" ")
95             with open("{}scenarios.xml".format(self.save_location, scenario), "w+") as file_writer:
96                 file_writer.write(xml_formatted)
97     except IOError as e:
98         print("An error occurred trying to write analytics to scenarios.xml file:", e)
99     except Exception as e:
100         print("Something wrong happened trying to save scenario analytics:", e)
101
102
103

```

File: ./software/data\_analysis/classes/data\_analysers.py

```

1 import os
2 import xml.etree.ElementTree as ET
3 from xml.dom.minidom import parseString
4
5 # DataAnalysers are used to parse data from files according to their type and then save it to analysed data files
6 class DataAnalyser:
7     def __init__(self, scenarios, participants):
8         self.path_to_recordings = '../data/recordings/'
9         self.scenarios = scenarios
10        self.participants = participants
11
12        # A list of pairs: (participant, score on all scenarios)
13        self.participant_score = {participant: 0.0 for participant in participants}
14
15        # A list of pairs: (scenario, score of all participants)
16        self.scenario_score = {scenario: 0.0 for scenario in scenarios}
17
18        # A list of pairs: (scenario, interesting points of all participants)
19        self.scenario_points = {scenario: [] for scenario in scenarios}
20
21        # This method goes through the recording files and processes files that are relevant

```

```

22 def analyse_files(self, xml_category):
23     try:
24         # First it looks for a directory for each participant according to the participants specified
25         # in the analysis.parameters.json
26         for participant in self.participants:
27             participant_directories = os.listdir(self.path_to_recordings)
28             if participant in participant_directories:
29                 item_path = os.path.join(self.path_to_recordings, participant)
30                 if os.path.isdir(item_path):
31                     # Then it looks for a directory for each scenario specified in the analysis.parameters.json
32                     for scenario in self.scenarios:
33                         scenario_directories = os.listdir(item_path)
34                         if scenario in scenario_directories:
35                             path = os.path.join(item_path, scenario, "{}.xml".format(self.category))
36                             # And then it looks for a file according to the analysers category
37                             # If one is found, it is processed
38                             if os.path.isfile(path):
39                                 try:
40                                     self.process_file(path, scenario, participant, xml_category)
41                                 except Exception as e:
42                                     print("An exception occurred trying to process file {}: ".format(path), e)
43         except Exception as e:
44             print("An exception occurred trying to gather data from recording files:", e)
45
46     # Used to process a single file
47 def process_file(self, path_to_file, scenario_name, participant_name, xml_category):
48     tree = ET.parse(path_to_file)
49     root = tree.getroot()
50     if self.category == "route_data":
51         penalty_points_total = root.find("ProportionOfRouteCompleted").text
52         location_data = []
53         for instance in root.findall("{}Instance".format(xml_category)):
54             if instance.find("WasReached").text == "0":
55                 x = instance.find("Location").find("X").text
56                 y = instance.find("Location").find("Y").text
57                 location_data.append((x, y))
58     else:
59         penalty_points_total = root.find("PenaltyPointsTotal").text
60         location_data = []
61         for instance in root.findall("{}Instance/Location".format(xml_category)):
62             x = instance.find("X").text
63             y = instance.find("Y").text
64             location_data.append((x, y))
65
66     self.scenario_points[scenario_name].extend(location_data)
67     self.scenario_score[scenario_name] += float(penalty_points_total)
68     self.participant_score[participant_name] += float(penalty_points_total)
69
70     # Used to write all gathered point to the points directory.
71 def save_points(self, path):
72     try:
73         for scenario, points_list in self.scenario_points.items():
74             root = ET.Element("Points")
75             for x, y in points_list:
76                 point = ET.SubElement(root, "Point")
77                 ET.SubElement(point, "X").text = str(x)
78                 ET.SubElement(point, "Y").text = str(y)
79
80             xml_str = ET.tostring(root, 'utf-8', method='xml')
81             xml_formatted = parseString(xml_str).toprettyxml(indent=" ")
82
83             with open("{}({}){}.xml".format(path, scenario, self.category), "w+") as file_writer:
84                 file_writer.write(xml_formatted)
85     except IOError as e:
86         print("An error occurred trying to save points of {}: ".format(self.category), e)
87     except Exception as e:
88         print("Something wrong happened trying to save participants analytics:", e)
89
90     # Run the data analyser
91 def run(self):
92     self.process_data()
93
94     # Abstract method
95 def process_data(self):
96     raise NotImplementedError("The method being called is abstract")
97
98
99
100 # Different data analysers that look for different metrics
101 class CollisionDataAnalyser(DataAnalyser):
102     def __init__(self, scenarios, participants):
103         self.category = "collision_data"
104         self.save_file_xml_element_name_average = "CollisionPenaltyPointsAverage"
105         super().__init__(scenarios, participants)
106
107     def process_data(self):
108         self.analyse_files("CollisionInstances")
109
110
111 class VehicleLightDataAnalyser(DataAnalyser):
112     def __init__(self, scenarios, participants):
113         self.category = "vehicle_light_misuse_data"
114         self.save_file_xml_element_name_average = "VehicleLightMisusePenaltyPointsAverage"
115         super().__init__(scenarios, participants)
116
117     def process_data(self):
118         self.analyse_files("VehicleLightsMisuseInstances")
119
120
121 class RouteDataAnalyser(DataAnalyser):
122     def __init__(self, scenarios, participants):
123         self.category = "route_data"
124         self.save_file_xml_element_name_average = "ProportionOfRouteCompletedAverage"
125         super().__init__(scenarios, participants)
126
127     def process_data(self):
128         self.analyse_files("Waypoints")
129
130
131 class LaneMarkingDataAnalyser(DataAnalyser):
132     def __init__(self, scenarios, participants):
133         self.category = "lane_marking_violation_data"
134         self.save_file_xml_element_name_average = "LaneMarkingViolationPenaltyPointsAverage"
135         super().__init__(scenarios, participants)
136
137     def process_data(self):
138         self.analyse_files("LaneMarkingViolationInstances")
139
140
141 class SpeedingDataAnalyser(DataAnalyser):
142     def __init__(self, scenarios, participants):
143         self.category = "speeding_data"
144         self.save_file_xml_element_name_average = "SpeedingPenaltyPointsAverage"
145         super().__init__(scenarios, participants)
146
147     def process_data(self):
148         self.analyse_files("SpeedingInstances")
149
150
151 class RoadTrafficDataAnalyser(DataAnalyser):
152     def __init__(self, scenarios, participants):

```

```

152     self.category = "road_traffic_violation_data"
153     self.save_file_xml_element_name_average = "RoadTrafficViolationPenaltyPointsAverage"
154     super().__init__(scenarios, participants)
155
156     def process_data(self):
157         self.analyse_files("RoadTrafficViolationInstances")

```

File: ./software/data\_analysis/score\_calculator.py

```

1 import argparse
2 import json
3 import xml.etree.ElementTree as ET
4
5 # The gamma value
6 GAMMA = 0.7
7
8 metrics = [
9     "collision_data",
10    "lane_marking_violation_data",
11    "vehicle_light_misuse_data",
12    "route_data",
13    "speeding_data",
14    "road_traffic_violation_data"
15 ]
16
17 argparser = argparse.ArgumentParser(description=__doc__)
18 argparser.add_argument(
19     '--scenario', '-s',
20     help='The simulation to consider')
21 argparser.add_argument(
22     '--name', '-n',
23     help='The name of the participant')
24 args = argparser.parse_args()
25
26 # Used to print the final score of the participant to the screen
27 def main():
28     try:
29         details = read_details()
30         try:
31             score, ideal_score = calculate_the_score(details)
32             print("The score for {} in {} is: {}".format(args.name, args.scenario) + str(score))
33             print("The ideal score is: {} and above.".format(ideal_score))
34         except Exception as e:
35             print("An exception occurred calculating the score:", e)
36     except Exception as e:
37         print("Something went wrong...", e)
38
39 # Function that calculates the final score according to the formula specified in the final report Chapter 7
40 def calculate_the_score(details):
41     score = 0
42     d = details["difficulty"]
43     alpha = details["intensity"]
44     v_avg = details["average_speed"]
45     s = details["distance"]
46     c, t, penalty_points = do_route_analysis(args.scenario, args.name)
47     time_for_additional_stops = sum([entry["time"] for entry in details["stops"]])
48
49     # The final formula in action
50     t_o = (s / v_avg) * (1 + alpha) + time_for_additional_stops
51     score = (c * (t_o / t) * d) - GAMMA * penalty_points
52     return float("%.3f" % score), d
53
54 # Sum up all the penalty points
55 def do_route_analysis(scenario, name):
56     penalty_points = 0
57     try:
58         for metric in metrics:
59             tree = ET.parse("../data/recordings/{}.{}/{}.xml".format(name, scenario, metric))
60             root = tree.getroot()
61             if metric == "route_data":
62                 c = float(root.find("ProportionOfRouteCompleted").text)
63                 t = float(root.find("SimulationTime").text[:-1])
64             else:
65                 penalty_points += float(root.find("PenaltyPointsTotal").text)
66     except Exception as e:
67         print("An exception occurred trying to gather data from recording files:", e)
68
69     return c, t, penalty_points
70
71 # Read the details of the simulation
72 def read_details():
73     try:
74         with open("../data/simulation_details/{}.json".format(args.scenario), "r") as file:
75             details = json.load(file)
76         return details
77     except Exception as e:
78         print("Something wrong happened trying to get data from {}.json".format(args.scenario), e)
79
80 if __name__ == '__main__':
81     if args.name and args.scenario:
82         main()
83     else:
84         "Please make sure that you pass the required arguments -s (scenario name) and -n (participant's name)"

```

File: ./software/data\_analysis/README.md

```

1  # Data Analysis
2
3  ## Introduction
4  This directory is dedicated to processing the recorded simulation data and presenting it in a meaningful way. In addition, it contains tools to evaluate the perfor
5  ## Content
6  The directory contains the following:
7  - analysis_parameters.json: A JSON file that holds information about the participants, scenarios, and metrics to be analyzed.
8  - analysis.py: A Python script that performs the data analysis based on the parameters defined in the analysis_parameters.json file.
9  - run_analysis.sh: A shell script to run the analysis using the analysis.py script.
10 - visualisation: Subdirectory containing scripts that for graphical representation.
11 - classes: Additional classes to help the analysis of recorded data files.
12 - score_calculator.py: a script to calculate the score of the participant in the given scenario
13 - scenario_analyser.py: a script to analyse the scenarios generating files that help to evaluate the performance of a participant
14
15 ## analysis parameters.json
16 Imagine that analysis_parameters.json file has the following structure:
17 ...
18 {
19   "participants": [
20     "ParticipantA",
21     "ParticipantB"
22   ],
23   "scenarios": [
24     "scenario1"
25   ],
26   "metrics": [
27     "collision_data",
28     "road_traffic_violation_data"
29   ]
30 }
31 ...
32
33 This file states that we want to analyze the collisions and road traffic violation data from scenario1 driven by Participants A and B.
34
35 **The list of all possible metrics:**
36 ...
37 "collision_data",
38 "lane marking_violation_data",
39 "vehicle_light_misuse_data",
40 "route_data",
41 "speeding_data",
42 "road_traffic_violation_data"
43 ...
44
45
46 ## Running the Analysis
47 To run the analysis, simply execute the following command in the terminal:
48 ...
49 bash run_analysis.sh name_of_analysis
50 ...
51
52
53 Analysed data will be saved in the ./analysed_data/data/name_of_analysis directory.
54
55 Replace "name_of_analysis" with a desired name for the analysis instance.
56
57 ## Results
58 The analysis tool will process the recording according to the parameters specified in the analysis_parameters.json file and produce the following files:
59 - participants.xml: Contains information about each of the specified participants and their average scores.
60 - scenarios.xml: Contains information about each of the specified scenarios and the average scores of participants.
61 - n .xml files in the points subdirectory: Each file contains a list of x and y coordinates, representing interesting patterns in the data.
62
63 For instance, the file scenario1_collision_data.xml will contain all the collision points for Participants A and B driving scenario1.
64
65 ## Visual Representation
66 For visual representation of the data, please refer to the subdirectory [visualisation](./visualisation/).
67
68 ## Scenario analyser
69 Before evaluating the score of a participant, we need to run the scenario analyser that will gather data from the simulated world and will create a file that will
70
71 The `scenario_analyser.py` script takes two arguments: -p indicating the path file and -s indicating the scenario file. It creates a file in the [simulation_detail
72
73 ## Score evaluation
74 The score evaluation is performed using the `score_calculator.py` script. The script takes arguments -n indicating the name of the participant and -s argument indi

```

#### File: ./software/data\_analysis/run\_analysis.sh

```

1  #!/bin/bash
2
3  name=$1
4
5
6  if [ -z "$name" ]; then
7      echo "If you want to run data analysis, the name for the analysis has to be specified."
8      echo "Example use: ./run_analysis.sh name"
9      exit 1
10 fi
11
12 cd ../../data/analysed_data/data/
13
14 if [ -d "$name" ]; then
15     echo "Directory $name already exists. Please choose another name. Aborting."
16     exit 1
17 fi
18
19 mkdir $name
20 mkdir ../$name/points
21
22 cd ../../../../software/data_analysis
23 python analysis.py -s $name

```

#### File: ./software/data\_analysis/visualisation/README.md

```

1 ## Information about drawing
2
3 This directory contains subdirectories for different kind of visual analysis methods.
4
5 Please refer to the subdirectories for more details.

```

#### File: ./software/data\_analysis/visualisation/diagram\_drawing/draw\_diagram.py

```

1 import xml.etree.ElementTree as ET
2 import argparse
3 import seaborn as sns
4 import matplotlib.pyplot as plt
5 import sys
6
7 def draw_bar_chart(args):
8     # Parse XML file
9     try:
10         tree = ET.parse(args.xml_file)
11     except ET.ParseError as e:
12         print(f"Error parsing XML file: {e}")
13         sys.exit(1)
14     root = tree.getroot()
15     names = []
16     scores = []
17     for participant in root:
18         name = participant.find("Name").text
19         try:
20             score = float(participant.find("Analysis").find(args.category).text)
21         except ValueError as e:
22             print(f"Error parsing score value: {e}")
23             sys.exit(1)
24         names.append(name)
25         scores.append(score)
26
27     # Plot bar chart
28     sns.set_palette("Blues", color_codes=True)
29     sns.set_style("darkgrid")
30     sns.set(font_scale=1.3)
31
32     sns.set_context("notebook")
33     sns.barplot(x=names, y=scores)
34     plt.xlabel(args.xlabel)
35     plt.ylabel("Penalty Points")
36     plt.title(args.title)
37     if args.save:
38         try:
39             plt.savefig(args.save)
40         except Exception as e:
41             print(f"Error saving the diagram: {e}")
42             sys.exit(1)
43     plt.show()
44
45 def main():
46     # Parse command line arguments
47     parser = argparse.ArgumentParser()
48     parser.add_argument("--xml_file", type=str, help="Path to the XML file")
49     parser.add_argument("--category", type=str, help="Category of the data")
50     parser.add_argument("--save", type=str, help="Path to save the diagram")
51     parser.add_argument("--title", type=str, help="Title of the diagram")
52     parser.add_argument("--xlabel", type=str, help="xlabel text")
53     args = parser.parse_args()
54
55     # Draw bar chart
56     draw_bar_chart(args)
57
58 if __name__ == "__main__":
59     main()

```

File: `./software/data_analysis/visualisation/diagram_drawing/README.md`

```

1 ## Information about diagram drawing
2
3 To draw the diagram look at the draw_diagram_example.sh (./draw_diagram_example.sh).
4
5 **The list of all possible categories:**
6
7 ...
8 CollisionPenaltyPointsAverage
9 LaneMarkingViolationPenaltyPointsAverage
10 VehicleLightMisusePenaltyPointsAverage
11 ProportionOfRouteCompletedAverage
12 SpeedingPenaltyPointsAverage
13 RoadTrafficViolationPenaltyPointsAverage
14 ...
15 - save_filename - specifies the name of the file that the diagram should be saved to. This file is an image.
16 - title - specifies what should be written at the top of the diagram
17 - xlabel - x axis label
18 - **data_filename** - points to the analysed data directory and then to either participants.xml or scenarios.xml. Please note, that at this moment, the diagrams (

```

File: `./software/data_analysis/visualisation/diagram_drawing/draw_diagram_example.sh`

```

1 #!/bin/bash
2
3 ##### ONLY THIS BIT IS MEANT TO BE EDITED #####
4 category="LaneMarkingViolationPenaltyPointsAverage"
5 title="Lane Marking Violation Penalty Points Average in Scenario 1"
6 xlabel="Participants"
7 save_filename="example_participants"
8 data_filename="example_analysis/participants.xml"
9 #####
10
11 data_path="../../data/analysed_data/data/$data_filename"
12 save_path="../../data/analysed_data/diagrams/$save_filename"
13
14 python draw_diagram.py --xml_file "$data_path" --category "$category" --save "$save_path" --title "$title" --xlabel "$xlabel"

```

File: `./software/data_analysis/visualisation/map_drawing/draw_map.sh`

```

1 #!/bin/bash
2
3 # map specified which commonroad format file to use to depict the map layout
4 map=$1
5
6 # first_file is path to an .xml file containing the locations to mark on the map
7 first_file=$2
8
9 # second file contains locations the same way the first_file does
10 second_file=$3
11
12 # For this, necessary python libraries need to be installed
13 # More information on this can be found in the python_libraries directory
14
15 # Check if any of the arguments were provided
16 if [[ -z "$map" ]] || -z "$first_file" ]]; then
17     # If either map or first_file is missing, exit the script and echo a message
18     echo "Error: missing arguments. Please specify map and first_file with points."
19     exit 1
20 fi
21
22 # Check if map or first file is missing
23 if [[ -f "$map" ]] || -f "$first_file" ]]; then
24     # If either map or first_file is missing, exit the script and echo a message
25     echo "Error: map file or first_file does not exist. Please check the paths provided."
26     exit 1
27 fi
28
29 # Launch crdesigner with appropriate arguments and disable any output to the terminal
30 echo "Drawing the map and marking the points."
31 if [ -n "$second_file" ]; then
32     crdesigner -pl "$map" -i "$first_file" -i "$second_file" -p2 "$map" -p2 "$second_file" -p2 "$second_file"
33 else
34     crdesigner -pl "$map" -i "$first_file" -i "$second_file" -p2 "$map" -p2 "$second_file" -p2 "$second_file"
35 fi
36 echo "Finished drawing the map and points."
37
38 # Remove files directory and its contents
39 rm -rf "files"

```

#### File: ./software/data\_analysis/visualisation/map\_drawing/README.md

```

1 ## Data visualisation on the map
2
3 This directory contains scripts that visualise the specific points on the map using the modified crdesigner module which can be found [here](../python_libraries/crdesigner/).
4
5 Please note that I am not the author of the crdesigner module. It is a slightly modified version allowing for additional files to be specified when launching it re
6
7 To use the map drawing tool, please refer to the 'draw_map_example.sh' script.
8
9 This script takes a CommonRoad type XML map description and a list of points from the analysed data. An example of how files containing points look like can be fo
10
11 The 'draw_map_example.sh' script also allows to provide a second file containing points.
12
13 Points from the first file are marked in blue color and points from the second file are marked in red color. This can be useful to analyse where human drivers and

```

#### File: ./software/data\_analysis/visualisation/map\_drawing/draw\_map\_example.sh

```

1 #!/bin/bash
2
3 first_file="example_analysis/points/scenario2_collision_data.xml"
4 second_file="example_analysis/points/scenario1_collision_data.xml"
5 map="Town07_commonRoad.xml"
6
7 # Second file can also be passed into the draw_map script to draw two depict two different datasets on the map
8 bash draw_map.sh $map $first_file $second_file

```

#### File: ./software/data\_analysis/scenario\_analyser.py

```

1 import glob
2 import os
3 import sys
4 import time
5 import random
6 import logging
7 import argparse
8 import xml.etree.ElementTree as ET
9 import numpy
10 import json
11 sys.path.append("..")
12 from carla_scripts.config import SAMPLING_RESOLUTION
13 from agentS.navigation.global_route_planner import GlobalRoutePlanner
14
15 try:
16     sys.path.append(glob.glob("../carla/dist/carla-*%d.%d-%s.egg" % (
17         sys.version_info.major,
18         sys.version_info.minor,
19         'win-amd64' if os.name == 'nt' else 'linux-x86_64'))[0])
20 except IndexError:
21     pass
22
23 import carla
24
25
26 argparser = argparse.ArgumentParser(description=__doc__)
27 argparser.add_argument(
28     '--scenario', '-s',
29     help='The scenario to consider')
30 argparser.add_argument(
31     '--path', '-p',
32     help='The path to consider')
33 args = argparser.parse_args()
34
35
36 # From location makes a route
37 def get_route(map, locations):
38     waypoints = []
39     grp = GlobalRoutePlanner(map, SAMPLING_RESOLUTION)
40
41     for i in range(len(locations)-1):
42         waypoints.extend(grp.trace_route(locations[i], locations[i+1]))
43     return waypoints
44
45 # Read the path file
46 def read_path_file():
47     file_path = "../data/paths/{}.json".format(args.path)
48     try:
49         with open(file_path) as file:
50             data = json.load(file)
51             locations = []
52             for loc in data["path_checkpoints"]:

```

```

53         locations.append(carla.Location(x=float(loc["x"]), y=float(loc["y"]), z=float(loc["z"])))
54         return locations, data["town"]
55     except Exception as e:
56         print("Could not read the file {}".format(args.path))
57
58 # Retrieve road elements from the OpenDRIVE file
59 def get_road_elements(map):
60     path_to_file = "%s/../data/maps/opendrive_format/{}.xodr".format(map)
61     tree = ET.parse(path_to_file)
62     root = tree.getroot()
63
64     # Get all road elements from the XML
65     roads = []
66     for road in root.findall("./road"):
67         roads.append(road)
68     return roads
69
70 # Get total distance of all waypoints
71 def get_total_distance(waypoints):
72     total_distance = 0
73     for i in range(len(waypoints)-1):
74         distance = waypoints[i][0].transform.location.distance(waypoints[i+1][0].transform.location)
75     total_distance = distance
76     return total_distance
77
78 # Calculate the intensity and retrieve the difficulty
79 def analyse_scenario(total_number_of_spawn_points):
80     file_path = "%s/../data/scenario_generation_data/generated_scenarios/{}.json".format(args.scenario)
81     try:
82         with open(file_path) as file:
83             data = json.load(file)
84             total_vehicle_amount = int(data["number_of_vehicles"]) + int(data["number_of_two_wheel_vehicles"])
85             # Assume that at most 80% of the spawn points are viable since if we were to put cars into all of them, it would result in the biggest gridlock in the
86             possible_spawn_point_amount = int(0.8 * total_number_of_spawn_points)
87             intensity = total_vehicle_amount / possible_spawn_point_amount
88             if intensity > 1: intensity = 1
89
90             return int(data["difficulty"]), intensity
91     except Exception as e:
92         print("Could not read the file {}".format(file_path))
93
94 # Calculate the average speed of all waypoints and also return how many road elements do not have the speed specified at all
95 def get_average_speed(waypoints, roads):
96     road_speed, how_many_elements_have_no_max_speed_specified = get_road_speed_dictionary(roads)
97     average_speed = 0
98     speed_limits = []
99
100    for waypoint in waypoints:
101        id = waypoint[0].road_id
102        if waypoint[0].road_id is not None and id in road_speed:
103            speed_limits.append(road_speed[waypoint[0].road_id])
104
105    if len(speed_limits) > 0:
106        average_speed = sum(speed_limits) / len(speed_limits)
107
108    return average_speed, how_many_elements_have_no_max_speed_specified
109
110 # Create a dictionary of road_id ---> max allowed speed
111 def get_road_speed_dictionary(roads):
112     speed_dict = {}
113     i = 0
114     for road in roads:
115         id = int(road.get('id'))
116         type = road.find('type')
117         speed_elem = None
118
119         if type is not None:
120             speed_elem = type.find('speed')
121
122         if speed_elem is None:
123             # The standard speed --> 50km/h converted to m/s
124             i += 1
125             speed_dict[id] = float(50 / 3.6)
126         else:
127             unit = speed_elem.get('unit')
128             value = float(speed_elem.get('max'))
129             if unit == 'mph':
130                 speed_dict[id] = value * 1.60934 / 3.6
131             else:
132                 speed_dict[id] = value / 3.6
133
134     return speed_dict, i
135
136 # For now count how many junctions there are and for each assign a time value
137 def get_stops(waypoints):
138     stops = []
139     junctions = 0
140     last_waypoint = "road"
141     for waypoint in waypoints:
142         if waypoint[0].is_junction:
143             if last_waypoint == "road":
144                 junctions += 1
145                 last_waypoint = "junction"
146             else:
147                 if last_waypoint == "junction":
148                     last_waypoint = "road"
149
150     for i in range(junctions):
151         stops.append(
152             {
153                 'name': "Junction",
154                 'time': 12
155             }
156         )
157     return stops
158
159 # Save scenario to file
160 def write_to_file(obj):
161     with open("%s/../data/simulation_details/{}.json".format(args.scenario), "w+") as f:
162         json.dump(obj, f, indent = 4)
163
164 # Used to analyse the path and the scenario to retrieve important data needed to evaluate the performance of the participants
165 def main():
166     try:
167         client = carla.Client("localhost", 2000)
168         client.set_timeout(10)
169         locations, map_name = read_path_file()
170
171         # Change the map to get the map element
172         world = client.load_world(map_name)
173         map = world.get_map()
174
175         waypoints = get_route(map, locations)
176         roads = get_road_elements(map_name)
177
178         total_distance = get_total_distance(waypoints)
179         difficulty, intensity = analyse_scenario(len(map.get_spawn_points()))
180         average_speed, how_many_elements_have_no_max_speed_specified = get_average_speed(waypoints, roads)
181
182         stops = get_stops(waypoints)

```



```

183
184     final_obj = {
185         "path": args.path,
186         "scenario": args.scenario,
187         "difficulty": float('%%.3f' % difficulty),
188         "intensity": float('%%.3f' % intensity),
189         "distance": float('%%.3f' % total_distance),
190         "average_speed": float('%%.3f' % average_speed),
191         "stops": stops
192     }
193
194     write_to_file(final_obj)
195
196     # Uncomment to see how (not) will the OpenDRIVE maps in carla are specified
197     # print("")
198     # print("In {} only {}% of all road elements have speed values specified".format(map_name, ('%.2f' % (100 - how_many_elements_have_no_max_speed_specified
199     # print(""))
200     print("Successfully analysed and save data to data/scenario_details/{}.json file".format(args.scenario))
201 except Exception as e:
202     print("Something has gone wrong.", e)
203
204 if __name__ == '__main__':
205     main()

```

#### File: ./software/carla\_scripts/replay.sh

```

1 #!/bin/bash
2
3 file="$1_scenario$2_recording.log"
4 python3 replayer.py -f $file

```

#### File: ./software/carla\_scripts/path\_generator.py

```

1 import argparse
2 import json
3 import logging
4 from scenarios.path_builder import PathBuilder
5
6 parser = argparse.ArgumentParser(description="Generate path parameters for a given scenario")
7 parser.add_argument('-s', '--scenario', default='example', help='Name of scenario')
8 parser.add_argument('-f', '--filename', default='example', help='Filename')
9 args = parser.parse_args()
10
11 logger = logging.getLogger(__name__)
12 logging.basicConfig(level=logging.INFO, filename="../../data/paths/logs/logs.log", filemode="w", format="%(asctime)s - %(name)s - %(levelname)s - %(message)s")
13
14 # Get a file indicating scenario parameters
15 def load_scenario():
16     logger.info("Loading scenario {} data".format(args.scenario))
17     file_path = "../../data/scenario_generation_data/generated_scenarios/{}.json".format(args.scenario)
18     try:
19         with open(file_path) as file:
20             scenario_data = json.load(file)
21             logger.info("Successfully read scenario {}.json data".format(args.scenario))
22             return scenario_data
23     except FileNotFoundError as e:
24         logger.exception("Scenario {} file could not be found.".format(args.scenario), e)
25     except json.JSONDecodeError as e:
26         logger.exception("The file {}.json could not be decoded as JSON".format(args.scenario), e)
27
28 # The main script
29 def main(args):
30     logger.info("Begin path generation")
31     try:
32         scenario_data = load_scenario()
33         path = PathBuilder.generate_details(scenario_data, args.filename)
34         if path: print("Generated and saved.")
35     except Exception as e:
36         logger.fatal("Fatal error. Shutting down the generation.")
37
38 if __name__ == '__main__':
39     main(args)

```

#### File: ./software/carla\_scripts/run.py

```

1 import argparse
2 import json
3 import os
4 import shutil
5 import logging
6 import random
7 from simulation_manager import SimulationManager
8 import subprocess
9
10 parser = argparse.ArgumentParser(description="Run all the scenarios for the participant")
11 parser.add_argument('-n', '--name', default='example', help='Name of the participant')
12 parser.add_argument(
13     '--ai',
14     action='store_true',
15     help='Is the AI driving?'
16 )
17 args = parser.parse_args()
18
19 logger = logging.getLogger(__name__)
20 logging.basicConfig(level=logging.INFO, filename="../../data/recordings/{}/session_logs.log".format(args.name), filemode="a", format="%(asctime)s - %(name)s - %(levelname)s - %(message)s")
21
22 scenario_list_file = "scenario_list.json"
23
24 # Read the scenario json file and extract all names of the scenarios to run
25 def get_scenarios():
26     try:
27         with open(scenario_list_file) as file:
28             scenarios = json.load(file)
29             logger.info("Successfully extracted scenarios from file {}".format(scenario_list_file))
30             return scenarios
31     except FileNotFoundError as e:
32         logger.error("Scenario list file could not be found.", exc_info=True)
33         raise e
34     except json.JSONDecodeError as e:
35         logger.error("The file {} could not be decoded as JSON".format(scenario_list_file), exc_info=True)
36         raise e
37
38 # Create a directory in the participant's results directory to store results about the simulation of a particular scenario
39 def create_directory_for_scenario(scenario_name):
40     path = "../../data/recordings/{}/{}".format(args.name, scenario_name)
41     if os.path.exists(path):
42         shutil.rmtree(path)
43     try:
44         os.makedirs(path)
45     except OSError:
46         logger.error("Could not create a directory for scenario {} for participant {}".format(scenario_name, args.name))

```

```

46         raise Exception
47     logger.info("Successfully created a directory for scenario {}".format(scenario_name))
48
49 def try_to_get_name_for_new_scenario():
50     path = "../data/scenario_generation_data/generated_scenarios/"
51     i = 0
52     # Try to find a name for a new scenario file for a 1000000 times
53     while i < 1000000:
54         name = "scenario{}".format(i)
55         path = os.path.join(path, name)
56         if not os.path.exists("{}{}.json".format(path)):
57             return name
58         i += 1
59     raise Exception("Could not find an unoccupied name for the new scenario. Most likely names scenario[1-1000000] are taken")
60
61 def check_scenario(scenario):
62     if scenario["name"] is not None:
63         return scenario["name"]
64     else:
65         logger.info("The scenario name was null. Predicting a new scenario.")
66         # Generate a new scenario and give it a name and return it
67         try:
68             name = try_to_get_name_for_new_scenario()
69             difficulty = random.randint(1, 1000)
70             logger.info("Scenario generator will now try to create a new scenario named {} of difficulty {}".format(name, difficulty))
71             # Launch the first person driving mode
72             except Exception as e:
73                 logger.error("The scenario file was not specified and also a new one could not be created", exc_info=True)
74                 raise e
75
76         try:
77             logger.info("Launching create_scenario.py script.")
78             # python3: can't open file '../software/generating_scenarios/create_scenario.py'
79             process = subprocess.run(["python3", "../software/generating_scenarios/create_scenario.py", "-d{}".format(difficulty), "-f{}".format(name)], stdout=
80             # Uncomment to print to the terminal
81             # print(process.stdout.decode(), process.stderr.decode())
82             logger.info("Successfully generated a new scenario {}".format(name))
83             return name
84             except Exception as e:
85                 logger.error("Could not open the driver.py script", exc_info=True)
86                 raise e
87
88     current_simulation_manager = None
89
90 # Launch a simulation manager for the scenario
91 def launch_scenario(scenario):
92     name = check_scenario(scenario)
93     create_directory_for_scenario(name)
94     global current_simulation_manager
95     current_simulation_manager = SimulationManager(name, scenario["details"], args.name, scenario["vehicle"], scenario["randomization_seed"], args.ai)
96     current_simulation_manager.begin_simulation()
97
98
99 # The main script
100 def main(args):
101     logger.info("THE SIMULATIONS BEGIN NOW")
102     failed_scenarios = []
103     try:
104         data = get_scenarios()
105         for scenario in data["scenarios"]:
106             try:
107                 logger.info("Starting to work on scenario {}".format(scenario["name"]))
108                 launch_scenario(scenario)
109             except Exception as e:
110                 failed_scenarios.append(scenario["name"])
111                 logger.error("Failed to simulate the scenario {}".format(scenario), exc_info=True)
112
113     # The end of all the simulations
114     logger.info("Successfully completed {} out of {} simulations".format((len(data["scenarios"])-len(failed_scenarios)), len(data["scenarios"])))
115     if len(failed_scenarios) > 0:
116         logger.warning("The failed scenarios are:")
117         for f_s in failed_scenarios:
118             logger.warning(f_s)
119     except KeyboardInterrupt:
120         logger.warning("The simulation was quit manually.")
121     except Exception as e:
122         logger.error("Something went horribly wrong.", exc_info=True)
123     finally:
124         global current_simulation_manager
125         if current_simulation_manager is not None:
126             current_simulation_manager.close_scenario()
127         logger.info("Shutting down the simulation.")
128
129 if __name__ == '__main__':
130     main(args)

```

File: ./software/carla\_scripts/replayer.py

```

1  #!/usr/bin/env python
2
3  # Copyright (c) 2019 Computer Vision Center (CVC) at the Universitat Autònoma de
4  # Barcelona (UAB).
5  #
6  # This work is licensed under the terms of the MIT license.
7  # For a copy, see <https://opensource.org/licenses/MIT>.
8
9  import glob
10 import os
11 import sys
12
13 import carla
14
15 import argparse
16
17 try:
18     sys.path.append(glob.glob('../carla/dist/carla-*%d.%d-%s.egg' % (
19         sys.version_info.major,
20         sys.version_info.minor,
21         'win-amd64' if os.name == 'nt' else 'linux-x86_64'))[0])
22 except IndexError:
23     pass
24
25 # Used to replay the specified recording
26 # Provide the necessary args
27 def main():
28
29     argparser = argparse.ArgumentParser(
30         description=__doc__)
31     argparser.add_argument(
32         '--host',
33         metavar='H',
34         default='127.0.0.1',
35         help='IP of the host server (default: 127.0.0.1)')
36     argparser.add_argument(
37         '-p', '--port',
38         metavar='P',
39         default=2000,
40         type=int,
41         help='TCP port to listen to (default: 2000)')
42     argparser.add_argument(
43         '-s', '--start',
44         metavar='S',
45         default=0.0,
46         type=float,
47         help='starting time (default: 0.0)')
48     argparser.add_argument(
49         '-d', '--duration',
50         metavar='D',
51         default=0.0,
52         type=float,
53         help='duration (default: 0.0)')
54     argparser.add_argument(
55         '-f', '--recorder-filename',
56         metavar='F',
57         default="test.log",
58         help='recorder filename (test1.log)')
59     argparser.add_argument(
60         '-c', '--camera',
61         metavar='C',
62         default=0,
63         type=int,
64         help='camera follows an actor (ex: 82)')
65     argparser.add_argument(
66         '-x', '--time-factor',
67         metavar='X',
68         default=1.0,
69         type=float,
70         help='time factor (default 1.0)')
71     argparser.add_argument(
72         '-i', '--ignore-hero',
73         action='store_true',
74         help='ignore hero vehicles')
75     argparser.add_argument(
76         '--spawn-sensors',
77         action='store_true',
78         help='spawn sensors in the replayed world')
79     args = argparser.parse_args()
80
81     try:
82         print("Beginning to replay the recording at {}".format(args.recorder_filename))
83         client = carla.Client(args.host, args.port)
84         client.set_timeout(120.0)
85
86         # Set the time factor for the replayer
87         client.set_replayer_time_factor(args.time_factor)
88
89         # Set to ignore the hero vehicles or not
90         client.set_replayer_ignore_hero(args.ignore_hero)
91
92         # Replay the session
93         client.replay_file(args.recorder_filename, args.start, args.duration, args.camera, args.spawn_sensors)
94
95     except KeyboardInterrupt:
96         pass
97     except Exception as e:
98         print("Something wrong happened trying to replay the recording:", e)
99     finally:
100         print("Stopped replaying.")
101
102
103 if __name__ == '__main__':
104     main()

```

File: ./software/carla\_scripts/config.py

```

1 # General variables
2
3 DISTANCE_FROM_WAYPOINT = 0.7
4 FINISH_DISTANCE_FROM_WAYPOINT = 3.0
5 DISTANCE_FROM_WAYPOINT_CHANGING_LANE = 1.0
6 SAMPLING_RESOLUTION = 0.5
7 SAMPLING_RESOLUTION_AI = 5
8 FPS = 60
9 RANDOMIZATION_SEED = 187349
10
11 # Penalty related variables
12
13 NO_BEAMS_NO_FOG_LIGHTS = 50
14 NO_BEAMS_NO_FOG_LIGHTS_TEXT = "Dark and foggy - no lights at all"
15
16 NO_BEAMS = 30
17 NO_BEAMS_TEXT = "Dark - no low beams"
18
19 NO_FOG_LIGHTS = 10
20 NO_FOG_LIGHTS_TEXT = "Dark and foggy - no fog lights"
21
22 HIT_PEDESTRIAN_TEXT = "Hit pedestrian"
23 HIT_PEDESTRIAN_PENALTY = 600
24 HIT_PEDESTRIAN_PENALTY_SPEEDING = 1200
25
26 HIT_VEHICLE_TEXT = "Hit vehicle"
27 HIT_VEHICLE_PENALTY = 250
28 HIT_VEHICLE_PENALTY_SPEEDING = 500
29
30 HIT_BICYCLE_TEXT = "Hit bicycle"
31 HIT_BICYCLE_PENALTY = 400
32 HIT_BICYCLE_PENALTY_SPEEDING = 800
33
34 HIT_ROAD_OBJECT_TEXT = "Hit road object"
35 HIT_ROAD_OBJECT_PENALTY = 150
36 HIT_ROAD_OBJECT_PENALTY_SPEEDING = 300
37
38 RED_LIGHT_TEXT = "Red light"
39 RED_LIGHT_PENALTY = 50
40 RED_LIGHT_PENALTY_SPEEDING = 100
41
42 SOLID_TEXT = "Solid lane marking"
43 SOLID_PENALTY = 20
44 SOLID_PENALTY_SPEEDING = 60
45
46 DOUBLE_SOLID_TEXT = "Double solid lane marking"
47 DOUBLE_SOLID_PENALTY = 40
48 DOUBLE_SOLID_PENALTY_SPEEDING = 100
49
50 BROKEN_NO_TURN_INDICATOR_TEXT = "Broken line wrong or no turn indicator"
51 BROKEN_NO_TURN_INDICATOR_PENALTY = 10
52 BROKEN_NO_TURN_INDICATOR_PENALTY_SPEEDING = 30
53
54 LIGHT_SPEEDING_PENALTY = 1
55 LIGHT_SPEEDING_TEXT = "Speeding under 20km/h"
56
57 HEAVY_SPEEDING_PENALTY = 3
58 HEAVY_SPEEDING_TEXT = "Speeding over 20km/h"
59
60 # If speeding exceeds this amount, it is considered heavy speeding (50 allowed ---> 71+ is considered heavy speeding)
61 HEAVY_SPEEDING_THRESHOLD = 20

```

File: ./software/carla\_scripts/classes/route.py

```

1 from config import FINISH_DISTANCE_FROM_WAYPOINT, DISTANCE_FROM_WAYPOINT, DISTANCE_FROM_WAYPOINT_CHANGING_LANE
2 from classes.helpers import distanceBetweenTwoLocations
3
4 class Route():
5
6     def __init__(self, waypoints):
7         self.routepoints = []
8         for waypoint in waypoints:
9             self.routepoints.append(waypoint, 0)
10        # Last routepoint is out finish point
11        self.finish_lane_waypoints = self.get_finish_lane_waypoints(waypoints[len(waypoints) - 1][0])
12
13
14    def get_finish_lane_waypoints(self, waypoint):
15        return [*self.getAllLeftWaypointsRecursively(waypoint, [], 0), *self.getAllRightWaypointsRecursively(waypoint, [], 0), waypoint]
16
17    # Return percent of route completed
18    def get_route_completion(self):
19        length = len(self.routepoints)
20        completed = sum(x[1] for x in self.routepoints)
21        return '%.2f' % (completed / length)
22
23    # Get shortest distance comparing each waypoint in waypoints
24    # with the provided location
25    def get_shortest_distance(self, location):
26        shortest_distance = None
27        for routepoint in self.routepoints:
28            waypoint_location = routepoint[0][0].transform.location
29            distance = distanceBetweenTwoLocations(location, waypoint_location)
30            if shortest_distance is None or distance < shortest_distance:
31                shortest_distance = distance
32        return shortest_distance
33
34    # Return true if the last location on route was reached
35    def is_finished(self, player_location):
36        for waypoint in self.finish_lane_waypoints:
37            distance = distanceBetweenTwoLocations(player_location, waypoint.transform.location)
38            if distance <= FINISH_DISTANCE_FROM_WAYPOINT:
39                return True
40        return False
41
42    # Marks the routepoints (x, 1) if the routepoint's distance is <= DISTANCE_FROM_WAYPOINT
43    def recalculate(self, location):
44        for i in range(len(self.routepoints)):
45            routepoint = self.routepoints[i]
46            waypoint_location = routepoint[0][0].transform.location
47            distance = distanceBetweenTwoLocations(location, waypoint_location)
48            if distance <= DISTANCE_FROM_WAYPOINT:
49                waypoint = routepoint[0]
50                self.routepoints.pop(i)
51                self.routepoints.append((waypoint, 1))
52
53        # print("Completed: " + str(self.get_completage()))
54
55    ### ADVANCED STUFF
56
57    # Marks the routepoints (x, 1) if the routepoint's distance is <= DISTANCE_FROM_WAYPOINT
58    def advanced_recalculate(self, location, is_changing_lane):
59        for i in range(len(self.routepoints)):
60            routepoint = self.routepoints[i]
61            waypoint = routepoint[0]
62            left_waypoints = self.getLeftValidWaypointsRecursively(waypoint[0], [], 0)

```

```

63         right_waypoints = self.getRightValidWaypointsRecursively(waypoint[0], [], 0)
64         waypoints = [*left_waypoints, *right_waypoints, waypoint[0]]
65         for w in waypoints:
66             distance = distanceBetweenTwoLocations(location, w.transform.location)
67             if is_changing_lane:
68                 if distance <= DISTANCE_FROM_WAYPOINT:
69                     self.routepoints.pop(i)
70                     self.routepoints.append((waypoint, 1))
71             else:
72                 if distance <= DISTANCE_FROM_WAYPOINT_CHANGING_LANE:
73                     self.routepoints.pop(i)
74                     self.routepoints.append((waypoint, 1))
75         # print("Completed: " + str(self.get_completage()))
76
77     def get_all_waypoints(self):
78         waypoints = []
79         for routepoint in self.routepoints:
80             waypoint = routepoint[0][0]
81             waypoints.append(waypoint)
82             for w in self.getLeftValidWaypointsRecursively(waypoint, [], 0):
83                 waypoints.append(w)
84             for w in self.getRightValidWaypointsRecursively(waypoint, [], 0):
85                 waypoints.append(w)
86         return waypoints
87
88     # Get all lanes on the current road (to the left of the waypoint) that are valid drive for the vehicle driving this route.
89     def getLeftValidWaypointsRecursively(self, waypoint, list, recursive_calls):
90         if waypoint == None or recursive_calls > 8:
91             return list
92         lane_change = waypoint.lane_change
93         if str(lane_change) == "Left" or str(lane_change) == "Both":
94             left_lane_waypoint = waypoint.get_left_lane()
95             if left_lane_waypoint == None:
96                 return list
97             list.append(left_lane_waypoint)
98             recursive_calls += 1
99             return self.getLeftValidWaypointsRecursively(left_lane_waypoint, list, recursive_calls)
100         else:
101             return list
102
103     # Get all road lanes on the map to the left of the current lane. Primarily used when marking the finish line.
104     def getAllLeftWaypointsRecursively(self, waypoint, list, recursive_calls):
105         left_lane_waypoint = waypoint.get_left_lane()
106         if left_lane_waypoint is not None:
107             list.append(left_lane_waypoint)
108             recursive_calls += 1
109             return self.getLeftValidWaypointsRecursively(left_lane_waypoint, list, recursive_calls)
110         else:
111             return list
112
113     # Get all lanes on the current road (to the right of the waypoint) that are valid drive for the vehicle driving this route.
114     def getRightValidWaypointsRecursively(self, waypoint, list, recursive_calls):
115         if waypoint == None or recursive_calls > 8:
116             return list
117         lane_change = waypoint.lane_change
118         if str(lane_change) == "Right" or str(lane_change) == "Both":
119             right_lane_waypoint = waypoint.get_right_lane()
120             if right_lane_waypoint == None:
121                 return list
122             list.append(right_lane_waypoint)
123             recursive_calls += 1
124             return self.getRightValidWaypointsRecursively(right_lane_waypoint, list, recursive_calls)
125         else:
126             return list
127
128     # Get all road lanes on the map to the right of the current lane. Primarily used when marking the finish line.
129     def getAllRightWaypointsRecursively(self, waypoint, list, recursive_calls):
130         right_lane_waypoint = waypoint.get_right_lane()
131         if right_lane_waypoint is not None:
132             list.append(right_lane_waypoint)
133             recursive_calls += 1
134             return self.getRightValidWaypointsRecursively(right_lane_waypoint, list, recursive_calls)
135         else:
136             return list

```

File: ./software/carla\_scripts/classes/helpers.py

```

1  import numpy
2
3  # Get distance between start and finish locations
4  # start and finish are both of type carla.Location
5  def distanceBetweenTwoLocations(start, finish):
6      first = numpy.array([start.x, start.y, start.z])
7      second = numpy.array([finish.x, finish.y, finish.z])
8      return numpy.linalg.norm(first-second)
9
10 # The following list is used to match the vehicle light state and figure out what lights are on
11 # The first element of the tuple is the "code" that the simulation returns about the state of car's lights
12 # The second element is a tuple of five True/False values representing the states of the following
13 # (LeftBlinker, RightBlinker, LowBeam, HighBeam, FogLights)
14 pairs = [
15     ("NONE", (False, False, False, False, False)),
16     # Left
17     ("LeftBlinker", (True, False, False, False, False)),
18     ("40", (True, False, False, False, False)),
19     ("96", (True, False, False, False, False)),
20     ("104", (True, False, False, False, False)),
21     # Left Position
22     ("33", (True, False, False, False, False)),
23     ("41", (True, False, False, False, False)),
24     ("97", (True, False, False, False, False)),
25     ("105", (True, False, False, False, False)),
26     # Left Low
27     ("99", (True, False, True, False, False)),
28     ("107", (True, False, True, False, False)),
29     ("35", (True, False, True, False, False)),
30     ("43", (True, False, True, False, False)),
31     # Left Fog
32     ("227", (True, False, True, False, True)),
33     ("235", (True, False, True, False, True)),
34     ("163", (True, False, True, False, True)),
35     ("171", (True, False, True, False, True)),
36     # Right
37     ("RightBlinker", (False, True, False, False, False)),
38     ("24", (False, True, False, False, False)),
39     ("80", (False, True, False, False, False)),
40     ("88", (False, True, False, False, False)),
41     # Right Position
42     ("17", (False, True, False, False, False)),
43     ("25", (False, True, False, False, False)),
44     ("81", (False, True, False, False, False)),
45     ("89", (False, True, False, False, False)),
46     # Right Low
47     ("19", (False, True, True, False, False)),
48     ("27", (False, True, True, False, False)),
49     ("83", (False, True, True, False, False)),
50     ("91", (False, True, True, False, False)),
51     # Right Fog
52     ("211", (False, True, True, False, True)),
53     ("219", (False, True, True, False, True)),
54     ("147", (False, True, True, False, True)),
55     ("155", (False, True, True, False, True)),
56     # Low
57     ("3", (False, False, True, False, False)),
58     ("11", (False, False, True, False, False)),
59     ("67", (False, False, True, False, False)),
60     ("75", (False, False, True, False, False)),
61     # Fog
62     ("131", (False, False, True, False, True)),
63     ("139", (False, False, True, False, True)),
64     ("195", (False, False, True, False, True)),
65     ("203", (False, False, True, False, True)),
66     # High
67     ("HighBeam", (False, False, False, True, False)),
68     ("12", (False, False, False, True, False)),
69     ("68", (False, False, False, True, False)),
70     ("76", (False, False, False, True, False)),
71     # Left High
72     ("36", (True, False, False, True, False)),
73     ("44", (True, False, False, True, False)),
74     ("100", (True, False, False, True, False)),
75     ("108", (True, False, False, True, False)),
76     # Right High
77     ("20", (False, True, False, True, False)),
78     ("28", (False, True, False, True, False)),
79     ("84", (False, True, False, True, False)),
80     ("92", (False, True, False, True, False)),
81     # Low High
82     ("7", (False, False, True, True, False)),
83     ("71", (False, False, True, True, False)),
84     ("79", (False, False, True, True, False)),
85     ("15", (False, False, True, True, False)),
86     # Fog High
87     ("135", (False, False, False, True, True)),
88     ("143", (False, False, False, True, True)),
89     ("199", (False, False, False, True, True)),
90     ("207", (False, False, False, True, True)),
91     # Left High Low
92     ("39", (True, False, True, True, False)),
93     ("47", (True, False, True, True, False)),
94     ("103", (True, False, True, True, False)),
95     ("111", (True, False, True, True, False)),
96     # Left High Fog
97     ("167", (True, False, True, True, True)),
98     ("175", (True, False, True, True, True)),
99     ("231", (True, False, True, True, True)),
100    ("239", (True, False, True, True, True)),
101    # Right High Low
102    ("23", (False, True, True, True, False)),
103    ("31", (False, True, True, True, False)),
104    ("87", (False, True, True, True, False)),
105    ("95", (False, True, True, True, False)),
106    # Right High Fog
107    ("151", (False, True, True, True, True)),
108    ("159", (False, True, True, True, True)),
109    ("215", (False, True, True, True, True)),
110    ("223", (False, True, True, True, True))
111 ]
112
113 # The default value to return in case the match among the pairs was not found (NO LIGHTS ARE ON)
114 default_match = (False, False, False, False, False)

```

File: ./software/carla\_scripts/recording/lane\_monitor.py

```

1 import xml.etree.ElementTree as ET
2 from xml.dom.minidom import parseString
3 from config import SOLID_PENALTY_SPEEDING, SOLID_TEXT, SOLID_PENALTY, DOUBLE_SOLID_PENALTY_SPEEDING, DOUBLE_SOLID_TEXT, DOUBLE_SOLID_PENALTY, BROKEN_NO_TURN_INDICA
4
5 import logging
6 logger = logging.getLogger(__name__)
7 class LaneMonitor():
8
9     def __init__(self, save_file):
10         self.save_file = save_file
11         self.buffer = []
12         self.sum = 0
13
14     def add_to_buffer(self, lane_type, timestep, player, is_speeding, turn_indicator_text):
15         location = player.get_location()
16         steering_angle = player.get_control().steer
17         points_message = self.points_given_and_message(lane_type, is_speeding, steering_angle, turn_indicator_text)
18         # Add only violations and leave-out 0 penalty-point entries
19         if points_message and points_message[0] != 0:
20             self.sum += points_message[0]
21             self.buffer.append([
22                 'points': points_message[0],
23                 'message': points_message[1],
24                 'time': str('%3f' % (timestep / 1000)) + "s",
25                 'location':
26                     'x': '%.6f' % location.x,
27                     'y': '%.6f' % location.y,
28                     'z': '%.6f' % location.z
29             ])
30
31     def write_to_file(self):
32         length = len(self.buffer)
33         root = ET.Element("LaneMarkingViolationData")
34         ET.SubElement(root, "PenaltyPointsTotal").text = str(self.sum)
35         ET.SubElement(root, "NumberOfLaneMarkingViolationInstances").text = str(length)
36
37         entries = ET.SubElement(root, "LaneMarkingViolationInstances")
38         for entry in self.buffer:
39             entry_xml = ET.SubElement(entries, "Instance")
40             ET.SubElement(entry_xml, "PenaltyPoints").text = str(entry['points'])
41             ET.SubElement(entry_xml, "Description").text = entry['message']
42             ET.SubElement(entry_xml, "Time").text = str(entry['time'])
43             location = ET.SubElement(entry_xml, "Location")
44             ET.SubElement(location, "X").text = str(entry['location']['x'])
45             ET.SubElement(location, "Y").text = str(entry['location']['y'])
46             ET.SubElement(location, "Z").text = str(entry['location']['z'])
47
48         xml_str = ET.tostring(root, 'utf-8', method='xml')
49         xml_formatted = parseString(xml_str).toprettyxml(indent=" ")
50
51         try:
52             path = "{}lane_marking_violation_data.xml".format(self.save_file)
53             with open(path, 'w') as file_writer:
54                 file_writer.write(xml_formatted)
55             logger.info("Lane monitor successfully saved recordings to the file {}".format(path))
56         except Exception as e:
57             logger.error("Lane monitor did not save data to the file {}".format(path), e)
58
59     def points_given_and_message(self, lane_type, speeding, steering_angle, turn_indicator_text):
60         if lane_type == "Solid":
61             if speeding:
62                 return(SOLID_PENALTY_SPEEDING, SOLID_TEXT + " speeding")
63             else:
64                 return(SOLID_PENALTY, SOLID_TEXT)
65         elif lane_type == "SolidSolid":
66             if speeding:
67                 return(DOUBLE_SOLID_PENALTY_SPEEDING, DOUBLE_SOLID_TEXT + " speeding")
68             else:
69                 return(DOUBLE_SOLID_PENALTY, DOUBLE_SOLID_TEXT)
70         elif lane_type == "Broken":
71             if turn_indicator_text == "Left" and steering_angle <= 0:
72                 # Turning left legally
73                 pass
74             elif turn_indicator_text == "Right" and steering_angle >= 0:
75                 # Turning right legally
76                 pass
77             else:
78                 if speeding:
79                     return(BROKEN_NO_TURN_INDICATOR_PENALTY_SPEEDING, BROKEN_NO_TURN_INDICATOR_TEXT + " speeding")
80                 else:
81                     return(BROKEN_NO_TURN_INDICATOR_PENALTY, BROKEN_NO_TURN_INDICATOR_TEXT)
82         else:
83             return(0, 'None')
84

```

File: ./software/carla\_scripts/recording/speeding\_monitor.py

```

1 import xml.etree.ElementTree as ET
2 from xml.dom.minidom import parseString
3 from config import LIGHT_SPEEDING_PENALTY, HEAVY_SPEEDING_PENALTY, LIGHT_SPEEDING_TEXT, HEAVY_SPEEDING_TEXT, HEAVY_SPEEDING_THRESHOLD
4
5 import logging
6 logger = logging.getLogger(__name__)
7 class SpeedingMonitor():
8
9     def __init__(self, save_file):
10         self.save_file = save_file
11         self.buffer = []
12         self.sum = 0
13
14     def add_to_buffer(self, allowed_speed, player_speed, timestep, location):
15         points_message = self.points_given_and_message(allowed_speed, player_speed)
16         self.sum += points_message[0]
17         self.buffer.append({
18             'points': points_message[0],
19             'message': points_message[1],
20             'allowed_speed': allowed_speed,
21             'player_speed': player_speed,
22             'time': str('%3f' % (timestep / 1000)) + "s",
23             'location': {
24                 'x': '%.6f' % location.x,
25                 'y': '%.6f' % location.y,
26                 'z': '%.6f' % location.z
27             }
28         })
29
30     def write_to_file(self):
31         length = len(self.buffer)
32         root = ET.Element("SpeedingData")
33         ET.SubElement(root, "PenaltyPointsTotal").text = str(self.sum)
34         ET.SubElement(root, "NumberOfSpeedingInstances").text = str(length)
35
36         entries = ET.SubElement(root, "SpeedingInstances")
37         for entry in self.buffer:
38             entry_xml = ET.SubElement(entries, "Instances")
39             ET.SubElement(entry_xml, "PenaltyPoints").text = str(entry['points'])
40             ET.SubElement(entry_xml, "Description").text = entry['message']
41             ET.SubElement(entry_xml, "AllowedSpeed").text = str(entry['allowed_speed'])
42             ET.SubElement(entry_xml, "PlayerSpeed").text = str(entry['player_speed'])
43             ET.SubElement(entry_xml, "Time").text = str(entry['time'])
44             location = ET.SubElement(entry_xml, "Location")
45             ET.SubElement(location, "X").text = str(entry['location']['x'])
46             ET.SubElement(location, "Y").text = str(entry['location']['y'])
47             ET.SubElement(location, "Z").text = str(entry['location']['z'])
48
49         xml_str = ET.tostring(root, 'utf-8', method='xml')
50         xml_formatted = parseString(xml_str).toprettyxml(indent=" ")
51
52         try:
53             path = "{}speeding_data.xml".format(self.save_file)
54             with open(path, 'w') as file_writer:
55                 file_writer.write(xml_formatted)
56             logger.info("Speeding monitor successfully saved recordings to the file {}".format(path))
57         except Exception as e:
58             logger.error("Speeding monitor did not save data to the file {}".format(path), e)
59
60     def points_given_and_message(self, allowed_speed, player_speed):
61         absolute_value = player_speed - allowed_speed
62         if absolute_value <= HEAVY_SPEEDING_THRESHOLD:
63             return (LIGHT_SPEEDING_PENALTY, LIGHT_SPEEDING_TEXT)
64         else:
65             return (HEAVY_SPEEDING_PENALTY, HEAVY_SPEEDING_TEXT)

```

File: ./software/carla\_scripts/recording/recorder.py

```

1 import glob
2 import os
3 import sys
4
5 try:
6     sys.path.append(glob.glob('../carla/dist/carla-*%d.%d-%s.egg' % (
7         sys.version_info.major,
8         sys.version_info.minor,
9         'win-amd64' if os.name == 'nt' else 'linux-x86_64'))[0])
10 except IndexError:
11     pass
12
13 import carla
14
15 import argparse
16 import random
17 import time
18 import logging
19
20 import logging
21 logger = logging.getLogger(__name__)
22
23 class Recorder():
24
25     def __init__(self, participant_name, scenario_name):
26         self.client = carla.Client('127.0.0.1', 2000)
27         self.participant_name = participant_name
28         self.scenario_name = scenario_name
29
30     def start_recording(self):
31         try:
32             path = '{}{}_recording.log'.format(self.participant_name, self.scenario_name)
33             self.client.start_recorder(path, True)
34             logger.info("Successfully started recording the simulation to {}".format(path))
35         except Exception as e:
36             logger.error("An exception occurred trying to start recording", e)
37
38     def stop_recording(self):
39         try:
40             self.client.stop_recorder()
41             logger.info("Stopped the recorder. The recording log file can be found in CarlaUE4/Saved/")
42         except Exception as e:
43             logger.error("Could not stop the recorder", e)

```

File: ./software/carla\_scripts/recording/traffic\_monitor.py



```

1 import xml.etree.ElementTree as ET
2 from xml.dom.minidom import parseString
3 from config import RED_LIGHT_PENALTY, RED_LIGHT_PENALTY_SPEEDING, RED_LIGHT_TEXT
4
5 import logging
6 logger = logging.getLogger(__name__)
7 class TrafficMonitor():
8
9     def __init__(self, save_file):
10         self.save_file = save_file
11         self.buffer = []
12         self.sum = 0
13
14     def add_to_buffer(self, speeding, timestep, location, type):
15         points_message = self.points_given_and_message(type, speeding)
16         self.sum += points_message[0]
17         self.buffer.append({
18             'points': points_message[0],
19             'message': points_message[1],
20             'time': str('%3f' % (timestep / 1000)) + "s",
21             'location': {
22                 'x': '%.6f' % location.x,
23                 'y': '%.6f' % location.y,
24                 'z': '%.6f' % location.z
25             }
26         })
27
28     def write_to_file(self):
29         length = len(self.buffer)
30         root = ET.Element("RoadTrafficViolationData")
31         ET.SubElement(root, "PenaltyPointsTotal").text = str(self.sum)
32         ET.SubElement(root, "NumberOfRoadTrafficViolationInstances").text = str(length)
33
34         entries = ET.SubElement(root, "RoadTrafficViolationInstances")
35         for entry in self.buffer:
36             entry_xml = ET.SubElement(entries, "Instance")
37             ET.SubElement(entry_xml, "PenaltyPoints").text = str(entry['points'])
38             ET.SubElement(entry_xml, "Description").text = entry['message']
39             ET.SubElement(entry_xml, "Time").text = str(entry['time'])
40             location = ET.SubElement(entry_xml, "Location")
41             ET.SubElement(location, "X").text = str(entry['location']['x'])
42             ET.SubElement(location, "Y").text = str(entry['location']['y'])
43             ET.SubElement(location, "Z").text = str(entry['location']['z'])
44
45         xml_str = ET.tostring(root, 'utf-8', method='xml')
46         xml_formatted = parseString(xml_str).toprettyxml(indent=" ")
47
48         try:
49             path = "{}road_traffic_violation_data.xml".format(self.save_file)
50             with open(path, 'w+') as file_writer:
51                 file_writer.write(xml_formatted)
52             logger.info("Traffic monitor successfully saved recordings to the file {}".format(path))
53         except Exception as e:
54             logger.error("Traffic monitor did not save data to the file {}".format(path), e)
55
56     def points_given_and_message(self, type, is_speeding):
57         if type == "RedLight":
58             if is_speeding:
59                 return(RED_LIGHT_PENALTY_SPEEDING, RED_LIGHT_TEXT + " speeding")
60             else:
61                 return(RED_LIGHT_PENALTY, RED_LIGHT_TEXT)
62         else:
63             return(0, "Unknown")

```

File: ./software/carla\_scripts/recording/vehicle\_light\_monitor.py

```

1 import xml.etree.ElementTree as ET
2 from xml.dom.minidom import parseString
3 from classes.helpers import pairs, default_match
4 from config import FPS, NO_BEAMS_NO_FOG_LIGHTS, NO_BEAMS, NO_FOG_LIGHTS, NO_BEAMS_NO_FOG_LIGHTS_TEXT, NO_BEAMS_TEXT, NO_FOG_LIGHTS_TEXT
5
6 import logging
7 logger = logging.getLogger(__name__)
8 class VehicleLightMonitor():
9
10     def __init__(self, save_file):
11         self.save_file = save_file
12         self.buffer = []
13         self.sum = 0
14         self.just_wrote = True
15
16         self.left_turn_indicator = False
17         self.right_turn_indicator = False
18         self.fog_lights = False
19         self.low_beam = False
20         self.high_beam = False
21
22     def set_lights(self, code, weather, time, location):
23         lights = self.match_lights(code)
24         self.left_turn_indicator = lights[0]
25         self.right_turn_indicator = lights[1]
26         self.low_beam = lights[2]
27         self.high_beam = lights[3]
28         self.fog_lights = lights[4]
29         # Write every 10s
30         value = (time/1000) % 10
31         if not self.just_wrote and (value > 0 and value <= 0.2):
32             data = self.check_lights(weather)
33             if(data):
34                 self.add_to_buffer(data[0], data[1], data[2], data[3], time, location)
35                 self.just_wrote = True
36             elif value > 0.2:
37                 self.just_wrote = False
38             else:
39                 pass
40
41     def check_lights(self, weather):
42         sun_altitude_angle = weather.sun_altitude_angle
43         fog_density = weather.fog_density
44         if(sun_altitude_angle < 30):
45             if(fog_density > 50):
46                 if(not self.fog_lights):
47                     if(not self.low_beam):
48                         return (NO_BEAMS_NO_FOG_LIGHTS, NO_BEAMS_NO_FOG_LIGHTS_TEXT, sun_altitude_angle, fog_density)
49                     else:
50                         return (NO_FOG_LIGHTS, NO_FOG_LIGHTS_TEXT, sun_altitude_angle, fog_density)
51                 else:
52                     if(not self.low_beam):
53                         return (NO_BEAMS, NO_BEAMS_TEXT, sun_altitude_angle, fog_density)
54                     else:
55                         if(not self.low_beam):
56                             return (NO_BEAMS, NO_BEAMS_TEXT, sun_altitude_angle, fog_density)
57             else:
58                 if(fog_density > 50):
59                     if(not self.fog_lights):
60                         if(not self.low_beam):
61                             return (NO_BEAMS_NO_FOG_LIGHTS, "Foggy - no lights at all", sun_altitude_angle, fog_density)
62                         else:
63                             return (NO_FOG_LIGHTS, "Foggy - no fog lights", sun_altitude_angle, fog_density)
64                     else:
65                         if(not self.low_beam):
66                             return (NO_BEAMS, "Foggy - no low beams", sun_altitude_angle, fog_density)
67                     else:
68                         return None
69
70     def turning_left(self):
71         return self.left_turn_indicator
72
73     def turning_right(self):
74         return self.right_turn_indicator
75
76     def add_to_buffer(self, points_given, type, sun_altitude_angle, fog_density, timestep, location):
77         self.sum += points_given
78         self.buffer.append([
79             'points': points_given,
80             'message': type,
81             'sun_altitude_angle': sun_altitude_angle,
82             'fog_density': fog_density,
83             'time': str('%3f' % (timestep / 1000)) + "s",
84             'location': [
85                 'x': '%.6f' % location.x,
86                 'y': '%.6f' % location.y,
87                 'z': '%.6f' % location.z
88             ]
89         ])
90
91     def write_to_file(self):
92         length = len(self.buffer)
93         root = ET.Element("VehicleLightMisuseData")
94         ET.SubElement(root, "PenaltyPointsTotal").text = str(self.sum)
95         ET.SubElement(root, "NumberOfVehicleLightMisuseInstances").text = str(length)
96
97         entries = ET.SubElement(root, "VehicleLightMisuseInstances")
98         for entry in self.buffer:
99             entry_xml = ET.SubElement(entries, "Instance")
100             ET.SubElement(entry_xml, "PenaltyPoints").text = str(entry['points'])
101             ET.SubElement(entry_xml, "Description").text = entry['message']
102             ET.SubElement(entry_xml, "SunAltitudeAngle").text = str(entry['sun_altitude_angle'])
103             ET.SubElement(entry_xml, "FogDensity").text = str(entry['fog_density'])
104             ET.SubElement(entry_xml, "Time").text = str(entry['time'])
105             location = ET.SubElement(entry_xml, "Location")
106             ET.SubElement(location, "X").text = str(entry['location']['x'])
107             ET.SubElement(location, "Y").text = str(entry['location']['y'])
108             ET.SubElement(location, "Z").text = str(entry['location']['z'])
109
110         xml_str = ET.tostring(root, 'utf-8', method='xml')
111         xml_formatted = parseString(xml_str).toprettyxml(indent=" ")
112
113         try:
114             path = "{}vehicle_light_misuse_data.xml".format(self.save_file)
115             with open(path, "w+") as file_writer:
116                 file_writer.write(xml_formatted)
117             logger.info("Vehicle light misuse monitor successfully saved recordings to the file {}".format(path))
118         except Exception as e:
119             logger.error("Vehicle light misuse monitor did not save data to the file {}".format(path), e)
120
121
122     def match_lights(self, code):
123         for pair in pairs:
124             if pair[0] == str(code):
125                 return pair[1]
126         return default_match
127
128

```

File: ./software/carla\_scripts/recording/route\_monitor.py

```
1 import carla
2 import xml.etree.ElementTree as ET
3 from xml.dom.minidom import parseString
4 from classes.route import Route
5 from agents.navigation.global_route_planner import GlobalRoutePlanner
6 from config import SAMPLING_RESOLUTION
7 import json
8
9 import logging
10 logger = logging.getLogger(__name__)
11
12 class RouteMonitor():
13     def __init__(self, path_locations, map, save_file):
14         self.route = self.get_route(path_locations, map)
15         self.save_file = save_file
16
17     # Takes a location as an argument and checks if it belongs to the route
18     def is_on_path(self, location):
19         pass
20
21     def update(self, location, is_changing_lane):
22         self.route.advanced_recalculate(location, is_changing_lane)
23
24     def finish_reached(self, player_location):
25         if self.route.is_finished(player_location):
26             return True
27
28     def write_to_file(self, finish_reached, simulation_time):
29         root = ET.Element("RouteData")
30         ET.SubElement(root, "SimulationTime").text = str('%3f' % (simulation_time / 1000)) + "s"
31         ET.SubElement(root, "ProportionOfRouteCompleted").text = str(self.route.get_route_completion())
32         ET.SubElement(root, "FinishReached").text = str(finish_reached)
33         ET.SubElement(root, "NumberOfWaypoints").text = str(len(self.route.routepoints))
34         waypoints = ET.SubElement(root, "Waypoints")
35         for routepoint in self.route.routepoints:
36             location = routepoint[0][0].transform.location
37             entry_xml = ET.SubElement(waypoints, "Instance")
38             ET.SubElement(entry_xml, "WasReached").text = str(routepoint[1])
39             location_elem = ET.SubElement(entry_xml, "Location")
40             ET.SubElement(location_elem, "X").text = str('%6f' % location.x)
41             ET.SubElement(location_elem, "Y").text = str('%6f' % location.y)
42             ET.SubElement(location_elem, "Z").text = str('%6f' % location.z)
43
44         xml_str = ET.tostring(root, 'utf-8', method='xml')
45         xml_formatted = parseString(xml_str).toprettyxml(indent="  ")
46         try:
47             path = "{}route_data.xml".format(self.save_file)
48             with open(path, 'w+') as file_writer:
49                 file_writer.write(xml_formatted)
50             logger.info("Route monitor successfully saved recordings to the file {}".format(path))
51         except Exception as e:
52             logger.error("Route monitor did not save data to the file {}".format(path), e)
53
54     def get_route(self, path_locations, map):
55         waypoints = []
56         grp = GlobalRoutePlanner(map, SAMPLING_RESOLUTION)
57         locations = self.transform_to_carla_locations(path_locations)
58         for i in range(len(locations)-1):
59             waypoints.extend(grp.trace_route(locations[i], locations[i+1]))
60         return Route(waypoints)
61
62     def transform_to_carla_locations(self, path_locations):
63         locations = []
64         for loc in path_locations:
65             locations.append(carla.Location(x=float(loc["x"]), y=float(loc["y"]), z=float(loc["z"])))
66         return locations
67
68     # WORKS WHERE THERE ARE ONLY TWO LOCATIONS PROVIDED
69     #
70     #
71     #
72     # def get_route(self, pathName, map):
73     #     grp = GlobalRoutePlanner(map, SAMPLING_RESOLUTION)
74     #     locations = self.read_path_file(pathName)
75     #     # Define a route
76     #     waypoints = grp.trace_route(locations[0], locations[1])
77     #     return Route(waypoints)
78
79     # def read_path_file(self, pathName):
80     #     start = None
81     #     finish = None
82     #     with open("./paths/{}.txt".format(pathName), "r") as file:
83     #         i = 0
84     #         for line in file.readlines():
85     #             values = [float(j) for j in line.split(',') if j.strip()]
86     #             if i == 0:
87     #                 start = carla.Location(x=values[0], y=values[1], z=values[2])
88     #             else:
89     #                 finish = carla.Location(x=values[0], y=values[1], z=values[2])
90     #             i+=1
91     #     return (start, finish)
```

File: ./software/carla\_scripts/recording/collision\_monitor.py

```

1 import xml.etree.ElementTree as ET
2 from xml.dom.minidom import parseString
3 from config import HIT_PEDESTRIAN_PENALTY_SPEEDING, HIT_PEDESTRIAN_TEXT, HIT_PEDESTRIAN_PENALTY, HIT_VEHICLE_PENALTY_SPEEDING, HIT_VEHICLE_TEXT, HIT_VEHICLE_PENALTY
4 from config import HIT_BICYCLE_TEXT, HIT_BICYCLE_PENALTY, HIT_ROAD_OBJECT_PENALTY_SPEEDING, HIT_ROAD_OBJECT_TEXT, HIT_ROAD_OBJECT_PENALTY
5
6 import logging
7 logger = logging.getLogger(__name__)
8 class CollisionMonitor():
9
10     def __init__(self, save_file):
11         self.save_file = save_file
12         self.buffer = []
13         self.sum = 0
14
15     def add_to_buffer(self, intensity, other_actor, location, timestep, is_speeding):
16         points_message = self.points_given_and_message(other_actor.type_id, intensity, is_speeding)
17         self.sum += points_message[0]
18         self.buffer.append({
19             'points': points_message[0],
20             'message': points_message[1],
21             'intensity': intensity,
22             'time': str('%3f' % (timestep / 1000)) + "s",
23             'location': {
24                 'x': '%.6f' % location.x,
25                 'y': '%.6f' % location.y,
26                 'z': '%.6f' % location.z
27             }
28         })
29
30     def write_to_file(self):
31         length = len(self.buffer)
32         root = ET.Element("CollisionData")
33         ET.SubElement(root, "PenaltyPointsTotal").text = str(self.sum)
34         ET.SubElement(root, "NumberOfCollisionInstances").text = str(length)
35
36         speeding_entries = ET.SubElement(root, "CollisionInstances")
37         for entry in self.buffer:
38             entry_xml = ET.SubElement(speeding_entries, "Instance")
39             ET.SubElement(entry_xml, "PenaltyPoints").text = str(entry['points'])
40             ET.SubElement(entry_xml, "Description").text = str(entry['message'])
41             ET.SubElement(entry_xml, "CollisionIntensity").text = str(entry['intensity'])
42             ET.SubElement(entry_xml, "Time").text = str(entry['time'])
43             location = ET.SubElement(entry_xml, "Location")
44             ET.SubElement(location, "X").text = str(entry['location']['x'])
45             ET.SubElement(location, "Y").text = str(entry['location']['y'])
46             ET.SubElement(location, "Z").text = str(entry['location']['z'])
47
48         xml_str = ET.tostring(root, 'utf-8', method='xml')
49         xml_formatted = parseString(xml_str).toprettyxml(indent=" ")
50         try:
51             path = "{}collision_data.xml".format(self.save_file)
52             with open(path, 'w+') as file_writer:
53                 file_writer.write(xml_formatted)
54             logger.info("Collision monitor successfully saved recordings to the file {}".format(path))
55         except Exception as e:
56             logger.error("Collision monitor did not save data to the file {}".format(path), e)
57
58     def points_given_and_message(self, victim_type, intensity, speeding):
59         # Intensity to be used
60         if victim_type.startswith("walker"):
61             if speeding:
62                 return HIT_PEDESTRIAN_PENALTY_SPEEDING, HIT_PEDESTRIAN_TEXT + " speeding"
63             else:
64                 return (HIT_PEDESTRIAN_PENALTY, HIT_PEDESTRIAN_TEXT)
65         elif victim_type.startswith("vehicle"):
66             if speeding:
67                 return HIT_VEHICLE_PENALTY_SPEEDING, HIT_VEHICLE_TEXT + " speeding"
68             else:
69                 return (HIT_VEHICLE_PENALTY, HIT_VEHICLE_TEXT)
70         elif victim_type.startswith("bicycle"):
71             if speeding:
72                 return HIT_BICYCLE_PENALTY_SPEEDING, HIT_BICYCLE_TEXT + " speeding"
73             else:
74                 return (HIT_BICYCLE_PENALTY, HIT_BICYCLE_TEXT)
75         else:
76             if speeding:
77                 return HIT_ROAD_OBJECT_PENALTY_SPEEDING, HIT_ROAD_OBJECT_TEXT + " speeding"
78             else:
79                 return (HIT_ROAD_OBJECT_PENALTY, HIT_ROAD_OBJECT_TEXT)
80
81     ### UNUSED
82
83     def get_actor_display_name(self, actor, truncate=250):
84         name = ' '.join(actor.type_id.replace('_', '.').title().split('.')[1:])
85         return (name[:truncate - 1] + u'\u2026') if len(name) > truncate else name

```

File: ./software/carla\_scripts/recording/manager.py

```

1 import carla
2 import weakref
3 import math
4 import time
5 import sys
6 from recording.route_monitor import RouteMonitor
7 from recording.lane_monitor import LaneMonitor
8 from recording.collision_monitor import CollisionMonitor
9 from recording.traffic_monitor import TrafficMonitor
10 from recording.speeding_monitor import SpeedingMonitor
11 from recording.vehicle_light_monitor import VehicleLightMonitor
12 from recording.recorder import Recorder
13
14 import logging
15 logger = logging.getLogger(__name__)
16 class Manager():
17
18     def __init__(self, path_locations, world, player, participant_name, scenario_name):
19         self.path_locations = path_locations
20         self.world = world
21         self.player = player
22         self.participant_name = participant_name
23         self.scenario_name = scenario_name
24         self.map = self.world.get_map()
25         self.time = 0
26
27         self.speeding = False
28         self.junction_visited_in_last_timestep = False
29
30         self.collision_sensor = CollisionSensor(self.player, self.world, self)
31         self.lane_invasion_sensor = LaneInvasionSensor(self.player, self.world, self)
32         self.save_file = '../data/recordings/{}|{}/'.format(participant_name, scenario_name)
33         self.route_monitor = RouteMonitor(self.path_locations, self.map, self.save_file)
34         self.lane_monitor = LaneMonitor(self.save_file)
35         self.collision_monitor = CollisionMonitor(self.save_file)
36         self.speeding_monitor = SpeedingMonitor(self.save_file)
37         self.traffic_monitor = TrafficMonitor(self.save_file)
38         self.vehicle_light_monitor = VehicleLightMonitor(self.save_file)

```

```

39
40     self.recorder = Recorder(participant_name, scenario_name)
41     self.recorder.start_recording()
42
43     self.time_of_last_collision = None
44     self.actor_of_last_collision = None
45     logger.info("Manager was successfully initialised")
46
47 # Called at each timestep. Checks for violations and records to respective monitors
48 def record(self, simulation_time):
49     # logger.info("Manager Recording at {}".format(str('%%.3f' % (simulation_time / 1000))+ "s"))
50     try:
51         player_location = self.player.get_location()
52         self.time = simulation_time
53         # Check if turn indicators are on
54         weather = self.world.get_weather()
55         self.set_lights(weather, self.time, player_location)
56         # Check if turn indicators are on
57         is_changing_lane = False
58         if self.vehicle_light_monitor.turning_left() or self.vehicle_light_monitor.turning_right():
59             is_changing_lane = True
60
61         self.route_monitor.update(player_location, is_changing_lane)
62         speeding = self.check_speeding()
63         violating_traffic = self.check_traffic_violations()
64         if speeding:
65             self.speeding_monitor.add_to_buffer(speeding[1], speeding[0], self.time, player_location)
66         if violating_traffic:
67             self.traffic_monitor.add_to_buffer(self.speeding, self.time, player_location, "redlight")
68         # logger.info("Observations were added to the buffers successfully.")
69         if self.route_monitor.finish_reached(player_location):
70             logger.info("Finish was reached. Terminating the simulation.")
71             self.shut_down(True)
72             return True
73     except Exception as e:
74         logger.error("Something went wrong adding observations to monitor buffers.", e)
75
76 # Called when finish is reached or simulation is quit manually
77 def shut_down(self, finish_reached):
78     logger.info("Shutting The manager down. Saving data from monitors to files.")
79     try:
80         self.route_monitor.write_to_file(finish_reached, self.time)
81         self.lane_monitor.write_to_file()
82         self.collision_monitor.write_to_file()
83         self.traffic_monitor.write_to_file()
84         self.speeding_monitor.write_to_file()
85         self.vehicle_light_monitor.write_to_file()
86         self.recorder.stop_recording()
87     except Exception as e:
88         logger.error("Something went wrong saving data from monitors to files.", e)
89
90     try:
91         logger.info("Destroying sensors.")
92         self.delete_sensors()
93     except Exception as e:
94         logger.error("Something went wrong destroying sensors owned by the Manager.", e)
95
96 # After the simulation is over, delete sensors
97 def delete_sensors(self):
98     self.collision_sensor.sensor.stop()
99     self.collision_sensor.sensor.destroy()
100     self.lane_invasion_sensor.sensor.stop()
101     self.lane_invasion_sensor.sensor.destroy()
102
103 # Record the collision to the collision monitor buffer
104 def record_collision(self, intensity, other_actor, player_location):
105     # If there was a collision with the same actor in the past 2sec, disregards this (collision sensor sometimes counts a collision multiple times)
106     if self.actor_of_last_collision and self.actor_of_last_collision == other_actor.id and (self.time - self.time_of_last_collision)/1000 < 2:
107         pass
108     else:
109         self.time_of_last_collision = self.time
110         self.actor_of_last_collision = other_actor.id
111         self.collision_monitor.add_to_buffer[intensity, other_actor, player_location, self.time, self.speeding]
112
113 # Record the lane marking violations to the lane monitor buffer
114 def record_lane_invasion(self, text):
115     if self.vehicle_light_monitor.turning_left():
116         light_indicator_text = "Left"
117     elif self.vehicle_light_monitor.turning_right():
118         light_indicator_text = "Right"
119     else:
120         light_indicator_text = "None"
121     self.lane_monitor.add_to_buffer(text, self.time, self.player, self.speeding, light_indicator_text)
122
123 # Check if the vehicle speeding
124 def check_speeding(self):
125     current_speed = self.player.get_velocity().x
126     # Convert from m/ to km/h and round to 2 places after comma
127     current_speed = float('%%.2f' % (abs(current_speed)*3.6))
128     speed_limit = self.player.get_speed_limit()
129     if current_speed and speed_limit and current_speed > speed_limit:
130         self.speeding = True
131         return (current_speed, speed_limit)
132     else:
133         self.speeding = False
134         return None
135
136 # Check if any traffic violations occurred (for now checks only for red lights)
137 def check_traffic_violations(self):
138     waypoint = self.map.get_waypoint(self.player.get_location())
139     if waypoint.is_junction:
140         if not self.junction_visited_in_last_timestep and str(self.player.get_traffic_light_state()) == "Red":
141             self.junction_visited_in_last_timestep = True
142             return True
143         else:
144             return False
145     else:
146         self.junction_visited_in_last_timestep = False
147         return False
148
149 # Update vehicle light monitor state (update which lights are on)
150 def set_lights(self, weather, time, player_location):
151     light_state = self.player.get_light_state()
152     self.vehicle_light_monitor.set_lights(light_state, weather, time, player_location)
153
154 # The foundations of this sensor were borrowed from CARLA developers
155 class LaneInvasionSensor(object):
156     def __init__(self, parent_actor, world, manager):
157         self.sensor = None
158         self.manager = manager
159         bp = world.get_blueprint_library().find('sensor.other.lane_invasion')
160         self.sensor = world.spawn_actor(bp, carla.Transform(), attach_to_parent_actor)
161         # We need to pass the lambda a weak reference to self to avoid circular
162         # reference.
163         weak_self = weakref.ref(self)
164         self.sensor.listen(lambda event: LaneInvasionSensor._on_invasion(weak_self, event))
165
166     @staticmethod
167     def _on_invasion(weak_self, event):
168         self = weak_self()
169         if not self:

```

```

169         return
170     lane_types = set(x.type for x in event.crossed_lane_markings)
171     text = ['%r' % str(x).split()[-1] for x in lane_types]
172     self.manager.record_lane_invasion(text[0].replace("'", ""))
173
174
175 # The foundations of this sensor were borrowed from CARLA developers
176 class CollisionSensor(object):
177     def __init__(self, parent_actor, world, manager):
178         self.sensor = None
179         self.player = parent_actor
180         self.manager = manager
181         bp = world.get_blueprint_library().find('sensor.other.collision')
182         self.sensor = world.spawn_actor(bp, carla.Transform()), attach to self.player)
183         # We need to pass the lambda a weak reference to self to avoid circular
184         # reference.
185         weak_self = weakref.ref(self)
186         self.sensor.listen(lambda event: CollisionSensor._on_collision(weak_self, event))
187
188     @staticmethod
189     def _on_collision(weak_self, event):
190         self = weak_self()
191         if not self:
192             return
193         impulse = event.normal_impulse
194         intensity = math.sqrt(impulse.x**2 + impulse.y**2 + impulse.z**2)
195         self.manager.record_collision(intensity, event.other_actor, self.player.get_location())
196

```

#### File: ./software/carla\_scripts/run.sh

```

1  #!/bin/bash
2
3  name=$1
4  ai=$2
5
6  if [ -z "$name" ]; then
7      echo "If you want to run simulations, you need to provide the name of the participant"
8      echo "Example use: ./run.sh participant1"
9      echo "The name is needed to associate each participant with their simulation results"
10     exit 1
11 fi
12
13 # First create a directory for the participant
14 cd ../../data/recordings/
15 mkdir -p $name
16
17 # Create a log file for the session inside
18 touch $name/session_logs.log
19 chmod 666 $name/session_logs.log
20
21 cd ../../software/carla_scripts
22
23 if [ -n "$ai" ]; then
24     if [ "$ai" = "ai" ]; then
25         echo "Starting the simulations for AI agent. For the logs, check the $name/session_logs.log file in the recordings directory."
26         python3 run.py -n "$name" --ai > /dev/null 2>&1
27     else
28         echo "If you want the AI implementation to run the scenarios write ai as the second argument of this script"
29         exit 1
30     fi
31 else
32     echo "Starting the simulations in manual control. For the logs, check the $name/session_logs.log file in the recordings directory."
33     python3 run.py -n "$name" > /dev/null 2>&1
34 fi
35 echo "Finished."

```

#### File: ./software/carla\_scripts/README.md

```

1  ## About the simulations
2
3  To run the simulations, your machine must meet the software requirements stated in the [README.md](../../README.md) file in the software directory.
4
5  Launch the UE4 engine running CARLA server following the guide in CARLA's [documentation](https://carla.readthedocs.io/en/latest/).
6
7  ...
8  bash run.sh [participant's name e.g. wilson]
9
10 This will run n scenarios sequentially, as stated in the [scenario_list.json](../scenario_list.json) file.
11
12 To let the CARLA Agent execute the scenarios, please run the command stated above but add 'ai' at the end.
13
14 Example:
15 ...
16 bash run.sh [participant's name e.g. carla_agent] ai
17 ...
18
19 ## Recordings
20
21 All simulations are recorded and monitored. The recorded data will be stored in:
22 'data/recordings/[participant's name]/'
23
24 The recording of the simulation itself will be stored in:
25
26 'CarlaUE4/Saved/'
27
28 ...
29 bash replay.sh [participant's name] [scenario name]
30
31
32 This will replay the simulation of scenario driven by the participant.

```

#### File: ./software/carla\_scripts/self\_driver.py

```

1  import os
2  import glob
3  import sys
4  import numpy as np
5
6  try:
7      sys.path.append(glob.glob('../carla/dist/carla-*%d.%d-%s.egg' % (
8          sys.version_info.major,
9          sys.version_info.minor,
10         'win-amd64' if os.name == 'nt' else 'linux-x86_64'))[0])
11 except IndexError:
12     pass
13 import carla
14
15 import math
16 import random

```

```

17 import time
18 import base64
19 import pygame
20 import pickle
21 from agents.navigation.global_route_planner import GlobalRoutePlanner
22 from config import SAMPLING_RESOLUTION_AI
23 from carla import ColorConverter as cc
24 from agents.navigation.behavior_agent import BehaviorAgent
25
26 import argparse
27 import collections
28 import datetime
29 import logging
30 import math
31 import random
32 import re
33 import weakref
34 from recording.manager import Manager
35 from config import FPS
36 from pygame import K_TAB
37 from pygame import K_ESCAPE
38 import argparse
39 import logging
40 from config import FPS
41
42
43 # =====
44 # -- Global functions -----
45 # =====
46
47
48 def find_weather_presets():
49     rgx = re.compile('.+?(?:(?<=[a-z])(?=[A-Z])|(?<[A-Z])(?=[A-Z][a-z])|$)')
50     name = lambda x: ' '.join(m.group(0) for m in rgx.finditer(x))
51     presets = [x for x in dir(carla.WeatherParameters) if re.match('[A-Z].+', x)]
52     return [(getattr(carla.WeatherParameters, x), name(x)) for x in presets]
53
54
55 def get_actor_display_name(actor, truncate=250):
56     name = ' '.join(actor.type_id.replace('_', '.').title().split('.')[1:])
57     return (name[:truncate - 1] + u'\u2026') if len(name) > truncate else name
58
59 def get_actor_blueprints(world, filter, generation):
60     bps = world.get_blueprint_library().filter(filter)
61     if generation.lower() == "all":
62         return bps
63
64     # If the filter returns only one bp, we assume that this one needed
65     # and therefore, we ignore the generation
66     if len(bps) == 1:
67         return bps
68
69     try:
70         int_generation = int(generation)
71         # Check if generation is in available generations
72         if int_generation in [1, 2]:
73             bps = [x for x in bps if int(x.get_attribute('generation')) == int_generation]
74             return bps
75         else:
76             print("Warning! Actor Generation is not valid. No actor will be spawned.")
77             return []
78     except:
79         print("Warning! Actor Generation is not valid. No actor will be spawned.")
80         return []
81
82
83 # =====
84
85 class World(object):
86     def __init__(self, carla_world, hud, args, scenario, client, path_details):
87         self.world = carla_world
88         self.sync = args.sync
89         self.actor_role_name = args.rolename
90         self.path_details = path_details
91         self.scenario = scenario
92         self.client = client
93         self.scenario_name = args.scenario
94         self.participant_name = args.name
95         try:
96             self.map = self.world.get_map()
97         except RuntimeError as error:
98             print('RuntimeError: {}'.format(error))
99             print('The server could not send the OpenDRIVE (.xodr) file:')
100             print('Make sure it exists, has the same name of your town, and is correct.')
101             sys.exit(1)
102         self.spawn_point = None
103         self.init_spawn_point(path_details["start_location"])
104         self.hud = hud
105         self.player = None
106         self.collision_sensor = None
107         self.lane_invasion_sensor = None
108         self.gnss_sensor = None
109         self.imu_sensor = None
110         self.radar_sensor = None
111         self.camera_manager = None
112         self._weather_presets = find_weather_presets()
113         self._weather_index = 0
114         self._actor_filter = args.filter
115         self._actor_generation = args.generation
116         self.gamma = args.gamma
117         self.restart(args)
118         self.recording_enabled = 0
119
120         # All about the simulation
121         self.simulation_clock = None
122         self.simulation_start_tick = 0
123         self.simulation_time = 0
124         self.manager = None
125
126         self.recording_start = 0
127         self.constant_velocity_enabled = False
128         self.show_vehicle_telemetry = False
129         self.doors_are_open = False
130         self.current_map_layer = 0
131         self.map_layer_names = [
132             carla.MapLayer.NONE,
133             carla.MapLayer.Buildings,
134             carla.MapLayer.Decals,
135             carla.MapLayer.Foliage,
136             carla.MapLayer.Ground,
137             carla.MapLayer.ParkedVehicles,
138             carla.MapLayer.Particles,
139             carla.MapLayer.Props,
140             carla.MapLayer.StreetLights,
141             carla.MapLayer.Walls,
142             carla.MapLayer.All
143         ]
144
145     def init_spawn_point(self, start_location):
146         try:

```

```

147         self.spawn_point = carla.Transform(carla.Location(x=start_location["x"], y=start_location["y"], z=start_location["z"]), carla.Rotation(pitch=0.0, yaw
148     except Exception as e:
149         logger.error("Could not initiate spawn location for the driver", exc_info=True)
150         raise e
151
152     def restart(self, args):
153         self.player_max_speed = 1.589
154         self.player_max_speed_fast = 3.713
155         # Keep same camera config if the camera manager exists.
156         cam_index = self.camera_manager.index if self.camera_manager is not None else 0
157         cam_pos_index = self.camera_manager.transform_index if self.camera_manager is not None else 0
158         # Get a random blueprint.
159         blueprint = random.choice(get_actor_blueprints(self.world, self._actor_filter, self._actor_generation))
160         blueprint.set_attribute('role_name', self.actor_role_name)
161         if blueprint.has_attribute('color'):
162             blueprint.set_attribute('color', '{}({},{})'.format(args.red, args.green, args.blue))
163         if blueprint.has_attribute('driver_id'):
164             driver_id = random.choice(blueprint.get_attribute('driver_id').recommended_values)
165             blueprint.set_attribute('driver_id', driver_id)
166         if blueprint.has_attribute('is_invincible'):
167             blueprint.set_attribute('is_invincible', 'true')
168         # set the max speed
169         if blueprint.has_attribute('speed'):
170             self.player_max_speed = float(blueprint.get_attribute('speed').recommended_values[1])
171             self.player_max_speed_fast = float(blueprint.get_attribute('speed').recommended_values[2])
172
173     # Spawn the player.
174     if self.player is not None:
175         spawn_point = self.player.get_transform()
176         spawn_point.location.z += 2.0
177         spawn_point.rotation.roll = 0.0
178         spawn_point.rotation.pitch = 0.0
179         self.destroy()
180         self.player = self.world.try_spawn_actor(blueprint, spawn_point)
181         self.show_vehicle_telemetry = False
182         self.modify_vehicle_physics(self.player)
183     while self.player is None:
184         if not self.map.get_spawn_points():
185             print("There are no spawn points available in your map/town.")
186             print("Please add some Vehicle Spawn Point to your UE4 scene.")
187             sys.exit(1)
188         if self.spawn_point is None:
189             spawn_points = self.map.get_spawn_points()
190             self.spawn_point = random.choice(spawn_points) if spawn_points else carla.Transform()
191         self.player = self.world.try_spawn_actor(blueprint, self.spawn_point)
192         self.show_vehicle_telemetry = False
193         self.modify_vehicle_physics(self.player)
194     # Set up the sensors.
195     self.collision_sensor = CollisionSensor(self.player, self.hud)
196     self.lane_invasion_sensor = LaneInvasionSensor(self.player, self.hud)
197     self.gnss_sensor = GnssSensor(self.player)
198     self.imu_sensor = IMUSensor(self.player)
199     self.camera_manager = CameraManager(self.player, self.hud, self._gamma)
200     self.camera_manager.transform_index = cam_pos_index
201     self.camera_manager.set_sensor(cam_index, notify=False)
202     actor_type = get_actor_display_name(self.player)
203     self.hud.notification(actor_type)
204
205     if self.sync:
206         self.world.tick()
207     else:
208         self.world.wait_for_tick()
209
210     def toggle_radar(self):
211         if self.radar_sensor is None:
212             self.radar_sensor = RadarSensor(self.player)
213         elif self.radar_sensor.sensor is not None:
214             self.radar_sensor.sensor.destroy()
215             self.radar_sensor = None
216
217     def modify_vehicle_physics(self, actor):
218         #If actor is not a vehicle, we cannot use the physics control
219         try:
220             physics_control = actor.get_physics_control()
221             physics_control.use_sweep_wheel_collision = True
222             actor.apply_physics_control(physics_control)
223         except Exception:
224             pass
225
226     def startScenario(self):
227         if self.simulation_clock is None:
228             logger.info("Starting the scenario")
229             self.scenario.start()
230             self.hud.notification("The simulation has started")
231         self.manager = Manager(self.path_details["path_checkpoints"], self.world, self.player, self.participant_name, self.scenario_name)
232         self.simulation_clock = pygame.time.Clock()
233         self.simulation_start_tick = pygame.time.get_ticks()
234
235     def finishScenario(self):
236         logger.warning("Finish was NOT REACHED. Terminating the simulation.")
237         self.manager.shut_down(False)
238         self.scenario.finish(self.client)
239
240     def tick(self):
241         if self.simulation_clock is not None:
242             self.simulation_clock.tick(60)
243             self.simulation_time += self.simulation_clock.get_time()
244             if self.manager.record(self.simulation_time):
245                 return True
246             self.hud.tick(self, self.simulation_time/1000)
247         else:
248             self.hud.tick(self, 0)
249
250     def render(self, display):
251         self.camera_manager.render(display)
252         self.hud.render(display)
253
254     def destroy_sensors(self):
255         self.camera_manager.sensor.destroy()
256         self.camera_manager.sensor = None
257         self.camera_manager.index = None
258
259     def destroy(self):
260         if self.radar_sensor is not None:
261             self.toggle_radar()
262         sensors = [
263             self.camera_manager.sensor,
264             self.collision_sensor.sensor,
265             self.lane_invasion_sensor.sensor,
266             self.gnss_sensor.sensor,
267             self.imu_sensor.sensor]
268         for sensor in sensors:
269             if sensor is not None:
270                 sensor.stop()
271                 sensor.destroy()
272         if self.player is not None:
273             self.player.destroy()
274
275 # =====
276 # -- HUD -----

```



```

277
278
279 class HUD(object):
280     def __init__(self, width, height):
281         self.dim = (width, height)
282         font = pygame.font.Font(pygame.font.get_default_font(), 20)
283         font_name = 'courier' if os.name == 'nt' else 'mono'
284         fonts = [x for x in pygame.font.get_fonts() if font_name in x]
285         default_font = 'ubuntumono'
286         mono = default_font if default_font in fonts else fonts[0]
287         mono = pygame.font.match_font(mono)
288         self.font_mono = pygame.font.Font(mono, 12 if os.name == 'nt' else 14)
289         self.notifications = FadingText(font, (width, 40), (0, height - 40))
290         self.server_fps = 0
291         self.frame = 0
292         self.simulation_time = 0
293         self._show_info = True
294         self._info_text = []
295         self._server_clock = pygame.time.Clock()
296
297     def on_world_tick(self, timestamp):
298         self._server_clock.tick()
299         self.server_fps = self._server_clock.get_fps()
300         self.frame = timestamp.frame
301         self.simulation_time = timestamp.elapsed_seconds
302
303     def tick(self, world, time):
304         self.notifications.tick(world, time)
305         if not self._show_info:
306             return
307         t = world.player.get_transform()
308         v = world.player.get_velocity()
309         c = world.player.get_control()
310         compass = world.imu_sensor.compass
311         heading = 'N' if compass > 270.5 or compass < 89.5 else ''
312         heading += 'S' if 90.5 < compass < 269.5 else ''
313         heading += 'E' if 0.5 < compass < 179.5 else ''
314         heading += 'W' if 180.5 < compass < 359.5 else ''
315         colhist = world.collision_sensor.get_collision_history()
316         collision = [colhist[x + self.frame - 200] for x in range(0, 200)]
317         max_col = max(1.0, max(collision))
318         collision = [x / max_col for x in collision]
319         vehicles = world.world.get_actors().filter('vehicle.*')
320         self._info_text = [
321             'Vehicle: % 20s' % get_actor_display_name(world.player, truncate=20),
322             'Route time: % 12s' % datetime.timedelta(seconds=int(time)),
323             '',
324             'Speed: % 15.0f km/h' % (3.6 * math.sqrt(v.x**2 + v.y**2 + v.z**2)),
325             ''
326         ]
327         if isinstance(c, carla.VehicleControl):
328             self._info_text += [
329                 ('Throttle:', c.throttle, 0.0, 1.0),
330                 ('Steer:', c.steer, -1.0, 1.0),
331                 ('Brake:', c.brake, 0.0, 1.0),
332                 ('Reverse:', c.reverse),
333                 ('Hand brake:', c.hand_brake),
334                 ('Manual:', c.manual_gear_shift),
335                 'Gear: %s' % [-1: 'R', 0: 'N'].get(c.gear, c.gear)]
336         self._info_text += [
337             '',
338             'Collision:',
339             collision,
340             ''
341         ]
342     def toggle_info(self):
343         self._show_info = not self._show_info
344
345     def notification(self, text, seconds=2.0):
346         self.notifications.set_text(text, seconds=seconds)
347
348     def error(self, text):
349         self.notifications.set_text('Error: %s' % text, (255, 0, 0))
350
351     def render(self, display):
352         if self._show_info:
353             info_surface = pygame.Surface((220, self.dim[1]))
354             info_surface.set_alpha(100)
355             display.blit(info_surface, (0, 0))
356             v_offset = 4
357             bar_h_offset = 100
358             bar_width = 106
359             for item in self._info_text:
360                 if v_offset + 18 > self.dim[1]:
361                     break
362                 if isinstance(item, list):
363                     if len(item) > 1:
364                         points = [(x + 8, v_offset + 8 + (1.0 - y) * 30) for x, y in enumerate(item)]
365                         pygame.draw.lines(display, (255, 136, 0), False, points, 2)
366                     item = None
367                     v_offset += 18
368                 elif isinstance(item, tuple):
369                     if isinstance(item[1], bool):
370                         rect = pygame.Rect((bar_h_offset, v_offset + 8), (6, 6))
371                         pygame.draw.rect(display, (255, 255, 255), rect, 0 if item[1] else 1)
372                     else:
373                         rect_border = pygame.Rect((bar_h_offset, v_offset + 8), (bar_width, 6))
374                         pygame.draw.rect(display, (255, 255, 255), rect_border, 1)
375                         f = (item[1] - item[2]) / (item[3] - item[2])
376                         if item[2] < 0.0:
377                             rect = pygame.Rect((bar_h_offset + f * (bar_width - 6), v_offset + 8), (6, 6))
378                         else:
379                             rect = pygame.Rect((bar_h_offset, v_offset + 8), (f * bar_width, 6))
380                         pygame.draw.rect(display, (255, 255, 255), rect)
381                     item = item[0]
382                 if item: # At this point has to be a str.
383                     surface = self.font_mono.render(item, True, (255, 255, 255))
384                     display.blit(surface, (8, v_offset))
385                     v_offset += 18
386             self.notifications.render(display)
387
388 # =====
389 # -- FadingText -----
390 # =====
391
392
393 class FadingText(object):
394     def __init__(self, font, dim, pos):
395         self.font = font
396         self.dim = dim
397         self.pos = pos
398         self.seconds_left = 0
399         self.surface = pygame.Surface(self.dim)
400
401     def set_text(self, text, color=(255, 255, 255), seconds=2.0):
402         text_texture = self.font.render(text, True, color)
403         self.surface = pygame.Surface(self.dim)
404         self.seconds_left = seconds
405         self.surface.fill((0, 0, 0, 0))
406         self.surface.blit(text_texture, (10, 11))

```

```

407
408     def tick(self, _, time):
409         delta_seconds = 1e-3 * time
410         self.seconds_left = max(0.0, self.seconds_left - delta_seconds)
411         self.surface.set_alpha(500.0 * self.seconds_left)
412
413     def render(self, display):
414         display.blit(self.surface, self.pos)
415
416
417 # =====
418 # -- CollisionSensor -----
419 # =====
420
421
422 class CollisionSensor(object):
423     def __init__(self, parent_actor, hud):
424         self.sensor = None
425         self.history = []
426         self._parent = parent_actor
427         self.hud = hud
428         world = self._parent.get_world()
429         bp = world.get_blueprint_library().find('sensor.other.collision')
430         self.sensor = world.spawn_actor(bp, carla.Transform(), attach_to=self._parent)
431         # We need to pass the lambda a weak reference to self to avoid circular
432         # reference.
433         weak_self = weakref.ref(self)
434         self.sensor.listen(lambda event: CollisionSensor._on_collision(weak_self, event))
435
436     def get_collision_history(self):
437         history = collections.defaultdict(int)
438         for frame, intensity in self.history:
439             history[frame] += intensity
440         return history
441
442     @staticmethod
443     def _on_collision(weak_self, event):
444         self = weak_self()
445         if not self:
446             return
447         actor_type = get_actor_display_name(event.other_actor)
448         self.hud.notification('Collision with %r' % actor_type)
449         impulse = event.normal_impulse
450         intensity = math.sqrt(impulse.x**2 + impulse.y**2 + impulse.z**2)
451         self.history.append((event.frame, intensity))
452         if len(self.history) > 4000:
453             self.history.pop(0)
454
455
456 # =====
457 # -- LaneInvasionSensor -----
458 # =====
459
460
461 class LaneInvasionSensor(object):
462     def __init__(self, parent_actor, hud):
463         self.sensor = None
464
465         # If the spawn object is not a vehicle, we cannot use the Lane Invasion Sensor
466         if parent_actor.type_id.startswith("vehicle."):
467             self._parent = parent_actor
468             self.hud = hud
469             world = self._parent.get_world()
470             bp = world.get_blueprint_library().find('sensor.other.lane_invasion')
471             self.sensor = world.spawn_actor(bp, carla.Transform(), attach_to=self._parent)
472             # We need to pass the lambda a weak reference to self to avoid circular
473             # reference.
474             weak_self = weakref.ref(self)
475             self.sensor.listen(lambda event: LaneInvasionSensor._on_invasion(weak_self, event))
476
477     @staticmethod
478     def _on_invasion(weak_self, event):
479         self = weak_self()
480         if not self:
481             return
482         lane_types = set(x.type for x in event.crossed_lane_markings)
483         text = ['%r' % str(x).split()[-1] for x in lane_types]
484         self.hud.notification('Crossed line %s' % ' and '.join(text))
485
486
487 # =====
488 # -- GnssSensor -----
489 # =====
490
491
492 class GnssSensor(object):
493     def __init__(self, parent_actor):
494         self.sensor = None
495         self._parent = parent_actor
496         self.lat = 0.0
497         self.lon = 0.0
498         world = self._parent.get_world()
499         bp = world.get_blueprint_library().find('sensor.other.gnss')
500         self.sensor = world.spawn_actor(bp, carla.Transform(carla.Location(x=1.0, z=2.8)), attach_to=self._parent)
501         # We need to pass the lambda a weak reference to self to avoid circular
502         # reference.
503         weak_self = weakref.ref(self)
504         self.sensor.listen(lambda event: GnssSensor._on_gnss_event(weak_self, event))
505
506     @staticmethod
507     def _on_gnss_event(weak_self, event):
508         self = weak_self()
509         if not self:
510             return
511         self.lat = event.latitude
512         self.lon = event.longitude
513
514
515 # =====
516 # -- IMUSensor -----
517 # =====
518
519
520 class IMUSensor(object):
521     def __init__(self, parent_actor):
522         self.sensor = None
523         self._parent = parent_actor
524         self.accelerometer = (0.0, 0.0, 0.0)
525         self.gyroscope = (0.0, 0.0, 0.0)
526         self.compass = 0.0
527         world = self._parent.get_world()
528         bp = world.get_blueprint_library().find('sensor.other.imu')
529         self.sensor = world.spawn_actor(
530             bp, carla.Transform(), attach_to=self._parent)
531         # We need to pass the lambda a weak reference to self to avoid circular
532         # reference.
533         weak_self = weakref.ref(self)
534         self.sensor.listen(
535             lambda sensor_data: IMUSensor._IMU_callback(weak_self, sensor_data))
536

```

```

537 @staticmethod
538 def _IMU_callback(weak_self, sensor_data):
539     self = weak_self()
540     if not self:
541         return
542     limits = (-99.9, 99.9)
543     self.accelerometer = (
544         max(limits[0], min(limits[1], sensor_data.accelerometer.x)),
545         max(limits[0], min(limits[1], sensor_data.accelerometer.y)),
546         max(limits[0], min(limits[1], sensor_data.accelerometer.z)))
547     self.gyroscope = (
548         max(limits[0], min(limits[1], math.degrees(sensor_data.gyroscope.x))),
549         max(limits[0], min(limits[1], math.degrees(sensor_data.gyroscope.y))),
550         max(limits[0], min(limits[1], math.degrees(sensor_data.gyroscope.z))))
551     self.compass = math.degrees(sensor_data.compass)
552
553
554 # =====
555 # -- RadarSensor -----
556 # =====
557
558
559 class RadarSensor(object):
560     def __init__(self, parent_actor):
561         self.sensor = None
562         self.parent = parent_actor
563         bound_x = 0.5 + self.parent.bounding_box.extent.x
564         bound_y = 0.5 + self.parent.bounding_box.extent.y
565         bound_z = 0.5 + self.parent.bounding_box.extent.z
566
567         self.velocity_range = 7.5 # m/s
568         world = self.parent.get_world()
569         self.debug = world.debug
570         bp = world.get_blueprint_library().find('sensor.other.radar')
571         bp.set_attribute('horizontal_fov', str(35))
572         bp.set_attribute('vertical_fov', str(20))
573         self.sensor = world.spawn_actor(
574             bp,
575             carla.Transform(
576                 carla.Location(x=bound_x + 0.05, z=bound_z+0.05),
577                 carla.Rotation(pitch=5)),
578             attach_to=self.parent)
579         # We need a weak reference to self to avoid circular reference.
580         weak_self = weakref.ref(self)
581         self.sensor.listen(
582             lambda radar_data: RadarSensor._Radar_callback(weak_self, radar_data))
583
584 @staticmethod
585 def _Radar_callback(weak_self, radar_data):
586     self = weak_self()
587     if not self:
588         return
589     # To get a numpy [vel, altitude, azimuth, depth],...[,,]:
590     # points = np.frombuffer(radar_data.raw_data, dtype=np.dtype('f4'))
591     # points = np.reshape(points, (len(radar_data), 4))
592
593     current_rot = radar_data.transform.rotation
594     for detect in radar_data:
595         azi = math.degrees(detect.azimuth)
596         alt = math.degrees(detect.altitude)
597         # The 0.25 adjusts a bit the distance so the dots can
598         # be properly seen
599         fw_vec = carla.Vector3D(x=detect.depth - 0.25)
600         carla.Transform(
601             carla.Location(),
602             carla.Rotation(
603                 pitch=current_rot.pitch + alt,
604                 yaw=current_rot.yaw + azi,
605                 roll=current_rot.roll)).transform(fw_vec)
606
607         def clamp(min_v, max_v, value):
608             return max(min_v, min(value, max_v))
609
610         norm_velocity = detect.velocity / self.velocity_range # range [-1, 1]
611         r = int(clamp(0.0, 1.0, 1.0 - norm_velocity) * 255.0)
612         g = int(clamp(0.0, 1.0, 1.0 - abs(norm_velocity)) * 255.0)
613         b = int(abs(clamp(- 1.0, 0.0, - 1.0 - norm_velocity)) * 255.0)
614         self.debug.draw_point(
615             radar_data.transform.location + fw_vec,
616             size=0.075,
617             life_time=0.06,
618             persistent_lines=False,
619             color=carla.Color(r, g, b))
620
621 # =====
622 # -- CameraManager -----
623 # =====
624
625
626 class CameraManager(object):
627     def __init__(self, parent_actor, hud, gamma_correction):
628         self.sensor = None
629         self.surface = None
630         self.parent = parent_actor
631         self.hud = hud
632         self.recording = False
633         Attachment = carla.AttachmentType
634
635         self.camera_transforms = [
636             # Third-person
637             (carla.Transform(carla.Location(x=-5.0, z=2.0), carla.Rotation(pitch=6.0)), Attachment.SpringArm),
638             # Watch back
639             (carla.Transform(carla.Location(x=4.0, z=2.0), carla.Rotation(yaw=0, pitch=6)), Attachment.SpringArm),
640             # Watch to the left
641             (carla.Transform(carla.Location(x=0, z=2.0, y=3), carla.Rotation(yaw=-90, pitch=0)), Attachment.Rigid),
642             # Watch to the right
643             (carla.Transform(carla.Location(x=0, z=2.0, y=-3), carla.Rotation(yaw=90, pitch=0)), Attachment.Rigid),
644             # First-person
645             (carla.Transform(carla.Location(y=-0.42, x=0.03, z=1.2), carla.Rotation(yaw=0, roll=0, pitch=-10)), Attachment.Rigid)]
646
647     self.transform_index = 1
648     self.sensors = [
649         ['sensor.camera.rgb', cc.Raw, 'Camera RGB', [],
650          ['sensor.camera.depth', cc.Raw, 'Camera Depth (Raw)', [],
651           ['sensor.camera.depth', cc.Depth, 'Camera Depth (Gray Scale)', [],
652            ['sensor.camera.depth', cc.LogarithmicDepth, 'Camera Depth (Logarithmic Gray Scale)', [],
653             ['sensor.camera.semantic_segmentation', cc.Raw, 'Camera Semantic Segmentation (Raw)', [],
654              ['sensor.camera.semantic_segmentation', cc.CityScapesPalette, 'Camera Semantic Segmentation (CityScapes Palette)', [],
655               ['sensor.camera.instance_segmentation', cc.CityScapesPalette, 'Camera Instance Segmentation (CityScapes Palette)', [],
656                ['sensor.camera.instance_segmentation', cc.Raw, 'Camera Instance Segmentation (Raw)', [],
657                 ['sensor.lidar.ray_cast', None, 'Lidar (Ray-Cast)', ('range': '50')],
658                  ['sensor.camera.dvs', cc.Raw, 'Dynamic Vision Sensor', [],
659                   ['sensor.camera.rgb', cc.Raw, 'Camera RGB Distorted',
660                    ['lens_circle_multiplier': '3.0',
661                     'lens_circle_falloff': '3.0',
662                     'chromatic_aberration_intensity': '0.5',
663                     'chromatic_aberration_offset': '0']],
664                    ['sensor.camera.optical_flow', cc.Raw, 'Optical Flow', []],
665                     ]
666                 ]
667             ]
668             ]
669             ]
670         ]
671     world = self.parent.get_world()

```

```

667 bp_library = world.get_blueprint_library()
668 for item in self.sensors:
669     bp = bp_library.find(item[0])
670     if item[0].startswith('sensor.camera'):
671         bp.set_attribute('image_size_x', str(hud.dim[0]))
672         bp.set_attribute('image_size_y', str(hud.dim[1]))
673         if bp.has_attribute('gamma'):
674             bp.set_attribute('gamma', str(gamma_correction))
675         for attr_name, attr_value in item[3].items():
676             bp.set_attribute(attr_name, attr_value)
677     elif item[0].startswith('sensor.lidar'):
678         self.lidar_range = 50
679
680         for attr_name, attr_value in item[3].items():
681             bp.set_attribute(attr_name, attr_value)
682             if attr_name == 'range':
683                 self.lidar_range = float(attr_value)
684
685     item.append(bp)
686 self.index = None
687
688 def toggle_camera(self):
689     self.transform_index = (self.transform_index + 1) % len(self._camera_transforms)
690     self.set_sensor(self.index, notify=False, force_respawn=True)
691
692 def set_sensor(self, index, notify=True, force_respawn=False):
693     index = index % len(self.sensors)
694     needs_respawn = True if self.index is None else \
695         (force_respawn or (self.sensors[index][2] != self.sensors[self.index][2]))
696     if needs_respawn:
697         if self.sensor is not None:
698             self.sensor.destroy()
699             self.surface = None
700         self.sensor = self.parent.get_world().spawn_actor(
701             self.sensors[index][1],
702             self._camera_transforms[self.transform_index][0],
703             attach_to=self.parent,
704             attachment_type=self._camera_transforms[self.transform_index][1])
705         # We need to pass the lambda a weak reference to self to avoid
706         # circular reference.
707         weak_self = weakref.ref(self)
708         self.sensor.listen(lambda image: CameraManager._parse_image(weak_self, image))
709     if notify:
710         self.hud.notification(self.sensors[index][2])
711     self.index = index
712
713 def next_sensor(self):
714     self.set_sensor(self.index + 1)
715
716 def toggle_recording(self):
717     self.recording = not self.recording
718     self.hud.notification('Recording %s' % ('On' if self.recording else 'Off'))
719
720 def render(self, display):
721     if self.surface is not None:
722         display.blit(self.surface, (0, 0))
723
724 @staticmethod
725 def _parse_image(weak_self, image):
726     self = weak_self()
727     if not self:
728         return
729     if self.sensors[self.index][0].startswith('sensor.lidar'):
730         points = np.frombuffer(image.raw_data, dtype=np.dtype('f4'))
731         points = np.reshape(points, (int(points.shape[0] / 4), 4))
732         lidar_data = np.array(points[:, :2])
733         lidar_data *= min(self.hud.dim) / (2.0 * self.lidar_range)
734         lidar_data += (0.5 * self.hud.dim[0], 0.5 * self.hud.dim[1])
735         lidar_data = np.fabs(lidar_data) # pylint: disable=E1111
736         lidar_data = lidar_data.astype(np.int32)
737         lidar_data = np.reshape(lidar_data, (-1, 2))
738         lidar_img_size = (self.hud.dim[0], self.hud.dim[1], 3)
739         lidar_img = np.zeros(lidar_img_size, dtype=np.uint8)
740         lidar_img[tuple(lidar_data.T)] = (255, 255, 255)
741         self.surface = pygame.surfarray.make_surface(lidar_img)
742     elif self.sensors[self.index][0].startswith('sensor.camera.dvs'):
743         # Example of converting the raw data from a carla.DVSEventArray
744         # sensor into a NumPy array and using it as an image
745         dvs_events = np.frombuffer(image.raw_data, dtype=np.dtype([
746             ('x', np.uint16), ('y', np.uint16), ('t', np.int64), ('pol', np.bool)]))
747         dvs_img = np.zeros((image.height, image.width, 3), dtype=np.uint8)
748         # Blue is positive, red is negative
749         dvs_img[dvs_events[:, 1]['x'], dvs_events[:, 1]['y'], dvs_events[:, 1]['pol'] * 2] = 255
750         self.surface = pygame.surfarray.make_surface(dvs_img.swapaxes(0, 1))
751     elif self.sensors[self.index][0].startswith('sensor.camera.optical_flow'):
752         image = image.get_color_coded_flow()
753         array = np.frombuffer(image.raw_data, dtype=np.dtype("uint8"))
754         array = np.reshape(array, (image.height, image.width, 4))
755         array = array[:, :, :3]
756         array = array[:, :, ::-1]
757         self.surface = pygame.surfarray.make_surface(array.swapaxes(0, 1))
758     else:
759         image.convert(self.sensors[self.index][1])
760         array = np.frombuffer(image.raw_data, dtype=np.dtype("uint8"))
761         array = np.reshape(array, (image.height, image.width, 4))
762         array = array[:, :, :3]
763         array = array[:, :, ::-1]
764         self.surface = pygame.surfarray.make_surface(array.swapaxes(0, 1))
765     if self.recording:
766         image.save_to_disk('_out/%08d' % image.frame)
767
768
769 def transform_to_carla_locations(path_locations):
770     locations = []
771     for loc in path_locations:
772         locations.append(carla.Location(x=float(loc["x"]), y=float(loc["y"]), z=float(loc["z"])))
773     return locations
774
775 def parse_keys(world):
776     for event in pygame.event.get():
777         if event.type == pygame.QUIT or (event.type == pygame.KEYUP and event.key == K_ESCAPE):
778             world.finishScenario()
779             return True
780         elif event.type == pygame.KEYUP:
781             if event.key == K_TAB:
782                 world.camera_manager.toggle_camera()
783
784
785 def game_loop(args, scenario, path_details):
786     pygame.init()
787     pygame.font.init()
788     world = None
789     original_settings = None
790     try:
791         client = carla.Client(args.host, args.port)
792         client.set_timeout(20.0)
793         sim_world = client.get_world()
794         if args.sync:
795             original_settings = sim_world.get_settings()
796             settings = sim_world.get_settings()

```

```

797         if not settings.synchronous_mode:
798             settings.synchronous_mode = True
799             settings.fixed_delta_seconds = 0.05
800         sim_world.apply_settings(settings)
801         traffic_manager = client.get_trafficmanager()
802         traffic_manager.set_synchronous_mode(True)
803
804         display = pygame.display.set_mode(
805             (args.width, args.height),
806             pygame.HWSURFACE | pygame.DOUBLEBUF)
807         display.fill((0,0,0))
808         pygame.display.flip()
809         scenario.setup(traffic_manager, sim_world)
810         scenario.spawn(client)
811         logger.info("The scenario was set. Spawning the hero vehicle and launching the simulation")
812         client.set_timeout(10.0)
813         hud = HUD(args.width, args.height)
814         try:
815             world = World(sim_world, hud, args, scenario, client, path_details)
816         except Exception as e:
817             logger.error("Could not initiate the simulation world correctly", exc_info=True)
818         if args.sync:
819             sim_world.tick()
820         else:
821             sim_world.wait_for_tick()
822
823         clock = pygame.time.Clock()
824         map = sim_world.get_map()
825         time.sleep(3)
826         try:
827             locations = transform_to_carla_locations(path_details["path_checkpoints"])
828             player = None
829             actors = sim_world.get_actors()
830             for actor in actors:
831                 if actor.attributes.get('role_name') == 'hero':
832                     player = actor
833
834             # traffic_manager.set_path(player, locations[1:])
835             traffic_manager.auto_lane_change(player, True)
836             traffic_manager.update_vehicle_lights(player, True)
837             agent = BehaviorAgent(player, behavior="aggressive")
838             #player.set_autopilot(True)
839             world.startScenario()
840
841         except Exception as e:
842             logger.error("Could not setup the hero vehicle correctly", exc_info=True)
843             raise e
844
845         current_destination_index = 1
846         agent.set_destination(locations[current_destination_index])
847         while True:
848             try:
849                 if agent.done():
850                     if current_destination_index == len(locations)-1:
851                         return
852                     else:
853                         current_destination_index += 1
854                         agent.set_destination(locations[current_destination_index])
855                 player.apply_control(agent.run_step())
856
857                 if args.sync:
858                     sim_world.tick()
859                 clock.tick_busy_loop(FPS)
860                 if world.tick():
861                     return
862                 if parse_keys(world):
863                     return
864                 world.render(display)
865                 pygame.display.flip()
866             except Exception as e:
867                 raise e
868         except Exception as e:
869             logger.error("Something went wrong trying to launch the self_driver.py", exc_info=True)
870         finally:
871             if (scenario.populated):
872                 scenario.finish(client)
873             if (world and world.manager):
874                 logger.info("Total time simulated: {}s".format(str(world.simulation_time/1000)))
875
876             if original_settings:
877                 sim_world.apply_settings(original_settings)
878
879             if world is not None:
880                 world.destroy()
881
882         pygame.quit()
883
884 argparser = argparse.ArgumentParser(
885     description='CARLA AI Client')
886 argparser.add_argument(
887     '-v', '--verbose',
888     action='store_true',
889     dest='debug',
890     help='print debug information')
891 argparser.add_argument(
892     '--host',
893     metavar='H',
894     default='127.0.0.1',
895     help='IP of the host server (default: 127.0.0.1)')
896 argparser.add_argument(
897     '-p', '--port',
898     metavar='P',
899     default=2000,
900     type=int,
901     help='TCP port to listen to (default: 2000)')
902 argparser.add_argument(
903     '-a', '--autopilot',
904     action='store_true',
905     help='enable autopilot')
906 argparser.add_argument(
907     '--res',
908     metavar='WIDTHxHEIGHT',
909     default='1280x720',
910     help='window resolution (default: 1280x720)')
911 argparser.add_argument(
912     '--filter', '-f',
913     metavar='PATTERN',
914     default='vehicle.*',
915     help='actor filter (default: "vehicle.*")')
916 argparser.add_argument(
917     '--generation',
918     metavar='G',
919     default='2',
920     help='restrict to certain actor generation (values: "1","2","All" - default: "2")')
921 argparser.add_argument(
922     '--rolename',
923     metavar='NAME',
924     default='hero',
925     help='actor role name (default: "hero")')
926 argparser.add_argument(

```

```

927     '--gamma',
928     default=2.2,
929     type=float,
930     help='Gamma correction of the camera (default: 2.2)')
931 argparser.add_argument(
932     '--sync',
933     action='store_true',
934     help='Activate synchronous mode execution')
935 argparser.add_argument(
936     '--red',
937     default="0",
938     help='Red part of the RGB color palette to paint the hero car')
939 argparser.add_argument(
940     '--green',
941     default="0",
942     help='Green part of the RGB color palette to paint the hero car')
943 argparser.add_argument(
944     '--blue',
945     default="0",
946     help='Blue part of the RGB color palette to paint the hero car')
947 argparser.add_argument(
948     '-n', '--name',
949     metavar='P',
950     default="example",
951     help='Participant name (name)')
952 argparser.add_argument(
953     '-s', '--scenario',
954     metavar='P',
955     default="example",
956     help='Scenario name (name)')
957 argparser.add_argument("s_b64")
958 argparser.add_argument("p_b64")
959 args = argparser.parse_args()
960
961 logger = logging.getLogger("self-driver-thread")
962 logging.basicConfig(level=logging.INFO, filename="../../data/recordings/{}/session_logs.log".format(args.name), filemode="a", format="%(asctime)s - %(name)s - %(
963
964 def main():
965     args.width, args.height = [int(x) for x in args.res.split('x')]
966
967     print(__doc__)
968
969     try:
970
971         scenario_b64 = args.s_b64
972         path_b64 = args.p_b64
973
974         scenario_bytes = base64.b64decode(scenario_b64.encode())
975         path_bytes = base64.b64decode(path_b64.encode())
976
977         scenario = pickle.loads(scenario_bytes)
978         path_details = pickle.loads(path_bytes)
979         game_loop(args, scenario, path_details)
980
981     except KeyboardInterrupt as e:
982         logger.warning('Keyboard interrupt while running self_driver.py script')
983     except Exception as e:
984         logger.error("Something went wrong.", exc_info=True)
985
986
987 if __name__ == '__main__':
988     main()
989

```

#### File: ./software/carla\_scripts/driver\_steeringwheel.py

```

1  #!/usr/bin/env python
2
3  # Copyright (c) 2019 Intel Labs
4  #
5  # This work is licensed under the terms of the MIT license.
6  # For a copy, see <https://opensource.org/licenses/MIT>.
7
8  # Allows controlling a vehicle with a keyboard. For a simpler and more
9  # documented example, please take a look at tutorial.py.
10
11  """
12  Welcome to CARLA manual control with steering wheel Logitech G29.
13
14  To drive start by pressing the brake pedal.
15  Change your wheel_config.ini according to your steering wheel.
16
17  To find out the values of your steering wheel use jstest-gtk in Ubuntu.
18
19  """
20
21 from __future__ import print_function
22
23
24 # =====
25 # -- find carla module -----
26 # =====
27
28
29 import glob
30 import os
31 import sys
32
33 try:
34     sys.path.append(glob.glob('../carla/dist/carla-*%d.%d-%s.egg' % (
35         sys.version_info.major,
36         sys.version_info.minor,
37         'win-amd64' if os.name == 'nt' else 'linux-x86_64'))[0])
38 except IndexError:
39     pass
40
41
42 # =====
43 # -- imports -----
44 # =====
45
46
47 import carla
48
49 from carla import ColorConverter as cc
50
51 import argparse
52 import collections
53 import datetime
54 import logging
55 import math
56 import random
57 import re
58 import weakref
59 from recording.manager import Manager

```

```

61 from config import FPS
62 import pickle
63
64 if sys.version_info >= (3, 0):
65
66     from configparser import ConfigParser
67
68 else:
69
70     from ConfigParser import RawConfigParser as ConfigParser
71
72 try:
73     import pygame
74     from pygame.locals import KMOD_CTRL
75     from pygame.locals import KMOD_SHIFT
76     from pygame.locals import K_0
77     from pygame.locals import K_9
78     from pygame.locals import K_BACKQUOTE
79     from pygame.locals import K_BACKSPACE
80     from pygame.locals import K_COMMA
81     from pygame.locals import K_DOWN
82     from pygame.locals import K_ESCAPE
83     from pygame.locals import K_F1
84     from pygame.locals import K_LEFT
85     from pygame.locals import K_PERIOD
86     from pygame.locals import K_RIGHT
87     from pygame.locals import K_SLASH
88     from pygame.locals import K_SPACE
89     from pygame.locals import K_TAB
90     from pygame.locals import K_UP
91     from pygame.locals import K_a
92     from pygame.locals import K_c
93     from pygame.locals import K_d
94     from pygame.locals import K_h
95     from pygame.locals import K_m
96     from pygame.locals import K_p
97     from pygame.locals import K_q
98     from pygame.locals import K_r
99     from pygame.locals import K_s
100    from pygame.locals import K_w
101 except ImportError:
102     raise RuntimeError('cannot import pygame, make sure pygame package is installed')
103
104 try:
105     import numpy as np
106 except ImportError:
107     raise RuntimeError('cannot import numpy, make sure numpy package is installed')
108
109
110 # =====
111 # -- Global functions -----
112 # =====
113
114 def find_weather_presets():
115     rgx = re.compile('.+?(?:(?<=[a-z])(?=[A-Z])|(?<=[A-Z])(?=[A-Z][a-z]))($)')
116     name = lambda x: ' '.join(m.group(0) for m in rgx.finditer(x))
117     presets = [x for x in dir(carla.WeatherParameters) if re.match('[A-Z].+', x)]
118     return [(getattr(carla.WeatherParameters, x), name(x)) for x in presets]
119
120
121 def get_actor_display_name(actor, truncate=250):
122     name = ' '.join(actor.type_id.replace('_', '.').title().split('.')[1:])
123     return (name[:truncate - 1] + u'\u2026') if len(name) > truncate else name
124
125
126 # =====
127 # -- World -----
128 # =====
129
130
131 class World(object):
132     def __init__(self, carla_world, hud, args, scenario, client, path_details):
133         self.world = carla_world
134         self.sync = args.sync
135         self.path_details = path_details
136         self.scenario = scenario
137         self.client = client
138         self.scenario_name = args.scenario
139         self.participant_name = args.name
140
141         self.spawn_point = None
142         self.init_spawn_point(path_details["start_location"])
143         self.hud = hud
144         self.player = None
145         self.collision_sensor = None
146         self.lane_invasion_sensor = None
147         self.gnss_sensor = None
148         self.camera_manager = None
149         self.weather_presets = find_weather_presets()
150         self.weather_index = 0
151         self.actor_filter = args.filter
152         self.gamma = args.gamma
153         self.restart(args)
154
155         # All about the simulation
156         self.simulation_clock = None
157         self.simulation_start_tick = 0
158         self.simulation_time = 0
159         self.manager = None
160
161         self.world.on_tick(hud.on_world_tick)
162
163     def init_spawn_point(self, start_location):
164         try:
165             self.spawn_point = carla.Transform(carla.Location(x=start_location["x"], y=start_location["y"], z=start_location["z"]), carla.Rotation(pitch=0.0, yaw=0.0))
166         except Exception as e:
167             logger.error("Could not initiate spawn location for the driver", e)
168
169     def restart(self, args):
170         # Keep same camera config if the camera manager exists.
171         cam_index = self.camera_manager.index if self.camera_manager is not None else 0
172         cam_pos_index = self.camera_manager.transform_index if self.camera_manager is not None else 0
173         # Get a random blueprint.
174         blueprint = random.choice(self.world.get_blueprint_library().filter(self.actor_filter))
175         blueprint.set_attribute('role_name', 'hero')
176         if blueprint.has_attribute('color'):
177             blueprint.set_attribute('color', '{}({},{})'.format(args.red, args.green, args.blue))
178         # Spawn the player.
179         if self.player is not None:
180             spawn_point = self.player.get_transform()
181             spawn_point.location.z += 2.0
182             spawn_point.rotation.roll = 0.0
183             spawn_point.rotation.pitch = 0.0
184             self.destroy()
185             self.player = self.world.try_spawn_actor(blueprint, spawn_point)
186         while self.player is None:
187             self.player = self.world.try_spawn_actor(blueprint, self.spawn_point)
188         # Set up the sensors.

```

```

190 self collision_sensor = CollisionSensor(self.player, self.hud)
191 self lane_invasion_sensor = LaneInvasionSensor(self.player, self.hud)
192 self gnss_sensor = GnssSensor(self.player)
193 self camera_manager = CameraManager(self.player, self.hud, args.gamma)
194 self camera_manager.transform_index = cam_pos_index
195 self camera_manager.set_sensor(cam_index, notify=False)
196 actor_type = get_actor_display_name(self.player)
197 self.hud.notification(actor_type)
198
199 if self.sync:
200     self.world.tick()
201 else:
202     self.world.wait_for_tick()
203
204 def next_weather(self, reverse=False):
205     self._weather_index += -1 if reverse else 1
206     self._weather_index %= len(self._weather_presets)
207     preset = self._weather_presets[self._weather_index]
208     self.hud.notification('Weather: %s' % preset[1])
209     self.player.get_world().set_weather(preset[0])
210
211 def startScenario(self):
212     if self.simulation_clock is None:
213         logger.info("Starting the scenario")
214         self.scenario.start()
215         self.hud.notification('The simulation has started')
216         self.manager = Manager(self.path_details["path_checkpoints"], self.world, self.player, self.participant_name, self.scenario_name)
217         self.collision_sensor = self.manager.collision_sensor
218         self.simulation_clock = pygame.time.Clock()
219         self.simulation_start_tick = pygame.time.get_ticks()
220
221 def finishScenario(self):
222     logger.warning("Finish was NOT REACHED. Terminating the simulation.")
223     self.manager.shut_down(False)
224     self.scenario.finish(self.client)
225
226 def tick(self):
227     if self.simulation_clock is not None:
228         self.simulation_clock.tick(60)
229         self.simulation_time += self.simulation_clock.get_time()
230         if self.manager.record(self.simulation_time):
231             return True
232         self.hud.tick(self, self.simulation_time/1000)
233     else:
234         self.hud.tick(self, 0)
235
236 def render(self, display):
237     self.camera_manager.render(display)
238     self.hud.render(display)
239
240 def destroy(self):
241     sensors =
242         self.camera_manager.sensor,
243         self.collision_sensor.sensor,
244         self.lane_invasion_sensor.sensor,
245         self.gnss_sensor.sensor)
246     for sensor in sensors:
247         if sensor is not None:
248             sensor.stop()
249             sensor.destroy()
250     if self.player is not None:
251         self.player.destroy()
252
253 # =====
254 # -- DualControl -----
255 # =====
256
257
258 class DualControl(object):
259     def __init__(self, world, start_in_autopilot):
260         self._autopilot_enabled = start_in_autopilot
261         if isinstance(world.player, carla.Vehicle):
262             self._control = carla.VehicleControl()
263             self._lights = carla.VehicleLightState.NONE
264             world.player.set_autopilot(self._autopilot_enabled)
265         elif isinstance(world.player, carla.Walker):
266             self._control = carla.WalkerControl()
267             self._autopilot_enabled = False
268             self._rotation = world.player.get_transform().rotation
269         else:
270             raise NotImplementedError("Actor type not supported")
271         self._steer_cache = 0.0
272         world.hud.notification("Press 'H' or '?' for help.", seconds=4.0)
273
274         # initialize steering wheel
275         pygame.joystick.init()
276         self._word = world
277         joystick_count = pygame.joystick.get_count()
278         if joystick_count > 1:
279             raise ValueError("Please Connect Just One Joystick")
280
281         self._joystick = pygame.joystick.Joystick(0)
282         self._joystick.init()
283
284         self._parser = ConfigParser()
285         self._parser.read('wheel_config.ini')
286         self._steer_idx = int(
287             self._parser.get('G29 Racing Wheel', 'steering_wheel'))
288         self._throttle_idx = int(
289             self._parser.get('G29 Racing Wheel', 'throttle'))
290         self._brake_idx = int(self._parser.get('G29 Racing Wheel', 'brake'))
291         self._reverse_idx = int(self._parser.get('G29 Racing Wheel', 'reverse'))
292         self._handbrake_idx = int(
293             self._parser.get('G29 Racing Wheel', 'handbrake'))
294
295     def parse_events(self, client, world, clock, sync_mode):
296         if isinstance(self._control, carla.VehicleControl):
297             current_lights = self._lights
298             for event in pygame.event.get():
299                 if event.type == pygame.QUIT:
300                     return True
301                 elif event.type == pygame.JOYBUTTONDOWN:
302                     if event.button == 1:
303                         world.hud.toggle_info()
304                     elif event.button == 19:
305                         world.camera_manager.toggle_camera()
306                     elif event.button == 19:
307                         world.camera_manager.toggle_camera_backward()
308                     elif event.button == self._reverse_idx:
309                         self._control.gear = 1 if self._control.reverse else -1
310                     elif event.button == 23:
311                         if not self._lights & carla.VehicleLightState.Position:
312                             world.hud.notification("Position lights")
313                             current_lights |= carla.VehicleLightState.Position
314                         else:
315                             world.hud.notification("Low beam lights")
316                             current_lights |= carla.VehicleLightState.LowBeam
317                         if self._lights & carla.VehicleLightState.LowBeam:
318                             world.hud.notification("Fog lights")
319                             current_lights |= carla.VehicleLightState.Fog

```



```

320         if self._lights & carla.VehicleLightState.Fog:
321             world.hud.notification("Lights off")
322             current_lights ^= carla.VehicleLightState.Position
323             current_lights ^= carla.VehicleLightState.LowBeam
324             current_lights ^= carla.VehicleLightState.Fog
325         elif event.button == 3 and not world.simulation_clock:
326             world.startScenario()
327         elif event.button == 3 and world.simulation_clock:
328             world.finishScenario()
329             return True
330         elif event.button == 5:
331             current_lights ^= carla.VehicleLightState.LeftBlinker
332         elif event.button == 4:
333             current_lights ^= carla.VehicleLightState.RightBlinker
334
335     elif event.type == pygame.KEYUP:
336         if self._is_quit_shortcut(event.key):
337             return True
338         elif event.key == K_BACKSPACE and not world.simulation_clock:
339             world.startScenario()
340         elif event.key == K_BACKSPACE and world.simulation_clock:
341             world.finishScenario()
342             return True
343         elif event.key == K_F1:
344             world.hud.toggle_info()
345         elif event.key == K_h or (event.key == K_SLASH and pygame.key.get_mods() & KMOD_SHIFT):
346             world.hud.help.toggle()
347         elif event.key == K_TAB:
348             world.camera_manager.toggle_camera()
349         elif event.key == K_c and pygame.key.get_mods() & KMOD_SHIFT:
350             world.next_weather(reverse=True)
351         elif event.key == K_c:
352             world.next_weather()
353         elif event.key == K_BACKQUOTE:
354             world.camera_manager.next_sensor()
355         elif event.key > K_0 and event.key <= K_9:
356             world.camera_manager.set_sensor(event.key - 1 - K_0)
357         elif event.key == K_r:
358             world.camera_manager.toggle_recording()
359         if isinstance(self._control, carla.VehicleControl):
360             if event.key == K_q:
361                 self._control.gear = 1 if self._control.reverse else -1
362             elif event.key == K_m:
363                 self._control.manual_gear_shift = not self._control.manual_gear_shift
364                 self._control.gear = world.player.get_control().gear
365                 world.hud.notification('%s Transmission' %
366                                     ('Manual' if self._control.manual_gear_shift else 'Automatic'))
367             elif self._control.manual_gear_shift and event.key == K_COMMA:
368                 self._control.gear = max(-1, self._control.gear - 1)
369             elif self._control.manual_gear_shift and event.key == K_PERIOD:
370                 self._control.gear = self._control.gear + 1
371             elif event.key == K_p:
372                 self._autopilot_enabled = not self._autopilot_enabled
373                 world.player.set_autopilot(self._autopilot_enabled)
374                 world.hud.notification('Autopilot %s' % ('On' if self._autopilot_enabled else 'Off'))
375
376     if not self._autopilot_enabled:
377         if isinstance(self._control, carla.VehicleControl):
378             self._parse_vehicle_keys(pygame.key.get_pressed(), clock.get_time())
379             self._parse_vehicle_wheel()
380             self._control.reverse = self._control.gear < 0
381
382             if self._control.brake:
383                 current_lights |= carla.VehicleLightState.Brake
384             else: # Remove the Brake flag
385                 current_lights ^= carla.VehicleLightState.Brake
386             if self._control.reverse:
387                 current_lights |= carla.VehicleLightState.Reverse
388             else: # Remove the Reverse flag
389                 current_lights ^= carla.VehicleLightState.Reverse
390             if current_lights != self._lights: # Change the light state only if necessary
391                 self._lights = current_lights
392                 world.player.set_light_state(carla.VehicleLightState(self._lights))
393
394         elif isinstance(self._control, carla.WalkerControl):
395             self._parse_walker_keys(pygame.key.get_pressed(), clock.get_time())
396             world.player.apply_control(self._control)
397
398     def _parse_vehicle_keys(self, keys, milliseconds):
399         self._control.throttle = 1.0 if keys[K_UP] or keys[K_w] else 0.0
400         steer_increment = 5e-4 * milliseconds
401         if keys[K_LEFT] or keys[K_a]:
402             self._steer_cache -= steer_increment
403         elif keys[K_RIGHT] or keys[K_d]:
404             self._steer_cache += steer_increment
405         else:
406             self._steer_cache = 0.0
407         self._steer_cache = min(0.7, max(-0.7, self._steer_cache))
408         self._control.steer = round(self._steer_cache, 1)
409         self._control.brake = 1.0 if keys[K_DOWN] or keys[K_s] else 0.0
410         self._control.hand_brake = keys[K_SPACE]
411
412     def _parse_vehicle_wheel(self):
413         numAxes = self._joystick.get_numaxes()
414         jsInputs = [float(self._joystick.get_axis(i)) for i in range(numAxes)]
415         # print (jsInputs)
416         jsButtons = [float(self._joystick.get_button(i)) for i in
417                     range(self._joystick.get_numbuttons())]
418
419         # Custom function to map range of inputs [1, -1] to outputs [0, 1] i.e 1 from inputs means nothing is pressed
420         # For the steering, it seems fine as it is
421         K1 = 1.0 # 0.55
422         steerCmd = K1 * math.tan(1.1 * jsInputs[self._steer_idx])
423
424         K2 = 1.6 # 1.6
425         throttleCmd = K2 + (2.05 * math.log10(
426             -0.7 * jsInputs[self._throttle_idx] + 1.4) - 1.2) / 0.92
427         if throttleCmd <= 0:
428             throttleCmd = 0
429         elif throttleCmd > 1:
430             throttleCmd = 1
431
432         brakeCmd = 1.6 + (2.05 * math.log10(
433             -0.7 * jsInputs[self._brake_idx] + 1.4) - 1.2) / 0.92
434
435         if brakeCmd <= 0:
436             brakeCmd = 0
437         elif brakeCmd > 1:
438             brakeCmd = 1
439
440         self._control.steer = steerCmd
441         self._control.brake = brakeCmd
442         self._control.throttle = throttleCmd
443
444         #toggle = jsButtons[self._reverse_idx]
445
446         self._control.hand_brake = bool(jsButtons[self._handbrake_idx])
447
448     def _parse_walker_keys(self, keys, milliseconds):
449         self._control.speed = 0.0

```

```

450     if keys[K_DOWN] or keys[K_s]:
451         self._control.speed = 0.0
452     if keys[K_LEFT] or keys[K_a]:
453         self._control.speed = -0.01
454         self._rotation.yaw -= 0.08 * milliseconds
455     if keys[K_RIGHT] or keys[K_d]:
456         self._control.speed = 0.01
457         self._rotation.yaw += 0.08 * milliseconds
458     if keys[K_UP] or keys[K_w]:
459         self._control.speed = 5.556 if pygame.key.get_mods() & KMOD_SHIFT else 2.778
460     self._control.jump = keys[K_SPACE]
461     self._rotation.yaw = round(self._rotation.yaw, 1)
462     self._control.direction = self._rotation.get_forward_vector()
463
464     @staticmethod
465     def _is_quit_shortcut(key):
466         return (key == K_ESCAPE) or (key == K_q and pygame.key.get_mods() & KMOD_CTRL)
467
468 # =====
469 # -- HUD -----
470 # =====
471
472
473
474 class HUD(object):
475     def __init__(self, width, height):
476         self.dim = (width, height)
477         font = pygame.font.Font(pygame.font.get_default_font(), 20)
478         font_name = 'courier' if os.name == 'nt' else 'mono'
479         fonts = [x for x in pygame.font.get_fonts() if font_name in x]
480         default_font = 'ubuntumono'
481         mono = default_font if default_font in fonts else fonts[0]
482         mono = pygame.font.match_font(mono)
483         self._font_mono = pygame.font.Font(mono, 12 if os.name == 'nt' else 14)
484         self._notifications = FadingText(font, (width, 40), (0, height - 40))
485         self.help = HelpText(pygame.font.Font(mono, 24), width, height)
486         self.server_fps = 0
487         self.frame = 0
488         self.simulation_time = 0
489         self._show_info = True
490         self._info_text = []
491         self._server_clock = pygame.time.Clock()
492
493     def on_world_tick(self, timestamp):
494         self._server_clock.tick()
495         self.server_fps = self._server_clock.get_fps()
496         self.frame = timestamp.frame
497         self.simulation_time = timestamp.elapsed_seconds
498
499     def tick(self, world, time):
500         self._notifications.tick(world, time)
501         if not self._show_info:
502             return
503         t = world.player.get_transform()
504         v = world.player.get_velocity()
505         c = world.player.get_control()
506         heading = 'N' if abs(t.rotation.yaw) < 89.5 else ''
507         heading += 'S' if abs(t.rotation.yaw) > 90.5 else ''
508         heading += 'E' if 179.5 > t.rotation.yaw > 0.5 else ''
509         heading += 'W' if -0.5 > t.rotation.yaw > -179.5 else ''
510         colhist = world.collision_sensor.get_collision_history()
511         collision = [colhist[x + self.frame - 200] for x in range(0, 200)]
512         max_col = max(1.0, max(collision))
513         collision = [x / max_col for x in collision]
514         vehicles = world.world.get_actors().filter('vehicle.*')
515         self._info_text = []
516         'Vehicle: % 20s' % get_actor_display_name(world.player, truncate=20),
517         'Route time: % 12s' % datetime.timedelta(seconds=int(time)),
518         '',
519         'Speed: % 15.0f km/h' % (3.6 * math.sqrt(v.x**2 + v.y**2 + v.z**2)),
520         ''
521         if isinstance(c, carla.VehicleControl):
522             self._info_text += [
523                 ('Throttle:', c.throttle, 0.0, 1.0),
524                 ('Steer:', c.steer, -1.0, 1.0),
525                 ('Brake:', c.brake, 0.0, 1.0),
526                 ('Reverse:', c.reverse),
527                 ('Hand brake:', c.hand_brake),
528                 ('Manual:', c.manual_gear_shift),
529                 'Gear: %s' % [-1: 'R', 0: 'N'].get(c.gear, c.gear)]
530         elif isinstance(c, carla.WalkerControl):
531             self._info_text += [
532                 ('Speed:', c.speed, 0.0, 5.556),
533                 ('Jump:', c.jump)]
534         self._info_text += [
535             '',
536             'Collision:',
537             collision,
538             '']
539
540     def toggle_info(self):
541         self._show_info = not self._show_info
542
543     def notification(self, text, seconds=2.0):
544         self._notifications.set_text(text, seconds=seconds)
545
546     def error(self, text):
547         self._notifications.set_text('Error: %s' % text, (255, 0, 0))
548
549     def render(self, display):
550         if self._show_info:
551             info_surface = pygame.Surface((220, self.dim[1]))
552             info_surface.set_alpha(100)
553             display.blit(info_surface, (0, 0))
554             v_offset = 4
555             bar_h_offset = 100
556             bar_width = 106
557             for item in self._info_text:
558                 if v_offset + 18 > self.dim[1]:
559                     break
560                 if isinstance(item, list):
561                     if len(item) > 1:
562                         points = [(x + 8, v_offset + 8 + (1.0 - y) * 30) for x, y in enumerate(item)]
563                         pygame.draw.lines(display, (255, 136, 0), False, points, 2)
564                     item = None
565                     v_offset += 18
566                 elif isinstance(item, tuple):
567                     if isinstance(item[1], bool):
568                         rect = pygame.Rect((bar_h_offset, v_offset + 8), (6, 6))
569                         pygame.draw.rect(display, (255, 255, 255), rect, 0 if item[1] else 1)
570                     else:
571                         rect_border = pygame.Rect((bar_h_offset, v_offset + 8), (bar_width, 6))
572                         pygame.draw.rect(display, (255, 255, 255), rect_border, 1)
573                         f = (item[1] - item[2]) / (item[3] - item[2])
574                         if item[2] < 0.0:
575                             rect = pygame.Rect((bar_h_offset + f * (bar_width - 6), v_offset + 8), (6, 6))
576                         else:
577                             rect = pygame.Rect((bar_h_offset, v_offset + 8), (f * bar_width, 6))
578                         pygame.draw.rect(display, (255, 255, 255), rect)
579                     item = item[0]

```

```

580         if item: # At this point has to be a str.
581             surface = self._font_mono.render(item, True, (255, 255, 255))
582             display.blit(surface, (8, v_offset))
583             v_offset += 18
584         self._notifications.render(display)
585         self._help.render(display)
586
587
588 # =====
589 # -- FadingText -----
590 # =====
591
592
593 class FadingText(object):
594     def __init__(self, font, dim, pos):
595         self._font = font
596         self._dim = dim
597         self._pos = pos
598         self._seconds_left = 0
599         self._surface = pygame.Surface(self._dim)
600
601     def set_text(self, text, color=(255, 255, 255), seconds=2.0):
602         text_texture = self._font.render(text, True, color)
603         self._surface = pygame.Surface(self._dim)
604         self._seconds_left = seconds
605         self._surface.fill((0, 0, 0, 0))
606         self._surface.blit(text_texture, (10, 11))
607
608     def tick(self, self, _, time):
609         delta_seconds = 1e-3 * time
610         self._seconds_left = max(0.0, self._seconds_left - delta_seconds)
611         self._surface.set_alpha(500.0 * self._seconds_left)
612
613     def render(self, self, display):
614         display.blit(self._surface, self._pos)
615
616
617 # =====
618 # -- HelpText -----
619 # =====
620
621
622 class HelpText(object):
623     def __init__(self, font, width, height):
624         lines = _doc_.split('\n')
625         self._font = font
626         self._dim = (680, len(lines) * 22 + 12)
627         self._pos = (0.5 * width - 0.5 * self._dim[0], 0.5 * height - 0.5 * self._dim[1])
628         self._seconds_left = 0
629         self._surface = pygame.Surface(self._dim)
630         self._surface.fill((0, 0, 0, 0))
631         for n, line in enumerate(lines):
632             text_texture = self._font.render(line, True, (255, 255, 255))
633             self._surface.blit(text_texture, (22, n * 22))
634             self._render = False
635         self._surface.set_alpha(220)
636
637     def toggle(self):
638         self._render = not self._render
639
640     def render(self, self, display):
641         if self._render:
642             display.blit(self._surface, self._pos)
643
644
645 # =====
646 # -- CollisionSensor -----
647 # =====
648
649
650 class CollisionSensor(object):
651     def __init__(self, parent_actor, hud):
652         self._sensor = None
653         self._history = []
654         self._parent = parent_actor
655         self._hud = hud
656         world = self._parent.get_world()
657         bp = world.get_blueprint_library().find('sensor.other.collision')
658         self._sensor = world.spawn_actor(bp, carla.Transform(), attach_to=self._parent)
659         # We need to pass the lambda a weak reference to self to avoid circular
660         # reference.
661         weak_self = weakref.ref(self)
662         self._sensor.listen(lambda event: CollisionSensor._on_collision(weak_self, event))
663
664     def get_collision_history(self):
665         history = collections.defaultdict(int)
666         for frame, intensity in self._history:
667             history[frame] += intensity
668         return history
669
670     @staticmethod
671     def _on_collision(weak_self, event):
672         self = weak_self()
673         if not self:
674             return
675         actor_type = get_actor_display_name(event.other_actor)
676         self._hud.notification('Collision with %r' % actor_type)
677         impulse = event.normal_impulse
678         intensity = math.sqrt(impulse.x**2 + impulse.y**2 + impulse.z**2)
679         self._history.append((event.frame, intensity))
680         if len(self._history) > 4000:
681             self._history.pop(0)
682
683
684 # =====
685 # -- LaneInvasionSensor -----
686 # =====
687
688
689 class LaneInvasionSensor(object):
690     def __init__(self, parent_actor, hud):
691         self._sensor = None
692         self._parent = parent_actor
693         self._hud = hud
694         world = self._parent.get_world()
695         bp = world.get_blueprint_library().find('sensor.other.lane_invasion')
696         self._sensor = world.spawn_actor(bp, carla.Transform(), attach_to=self._parent)
697         # We need to pass the lambda a weak reference to self to avoid circular
698         # reference.
699         weak_self = weakref.ref(self)
700         self._sensor.listen(lambda event: LaneInvasionSensor._on_invasion(weak_self, event))
701
702     @staticmethod
703     def _on_invasion(weak_self, event):
704         self = weak_self()
705         if not self:
706             return
707         lane_types = set(x.type for x in event.crossed_lane_markings)
708         text = '%r' % str(x).split()[-1] for x in lane_types
709         self._hud.notification('Crossed line %s' % ' and '.join(text))

```

```

710
711 # =====
712 # -- GnssSensor -----
713 # =====
714
715
716 class GnssSensor(object):
717     def __init__(self, parent_actor):
718         self.sensor = None
719         self.parent = parent_actor
720         self.lat = 0.0
721         self.lon = 0.0
722         world = self.parent.get_world()
723         bp = world.get_blueprint_library().find('sensor.other.gnss')
724         self.sensor = world.spawn_actor(bp, carla.Transform(carla.Location(x=1.0, z=2.8)), attach_to=self.parent)
725         # We need to pass the lambda a weak reference to self to avoid circular
726         # reference.
727         weak_self = weakref.ref(self)
728         self.sensor.listen(lambda event: GnssSensor._on_gnss_event(weak_self, event))
729
730     @staticmethod
731     def _on_gnss_event(weak_self, event):
732         self = weak_self()
733         if not self:
734             return
735         self.lat = event.latitude
736         self.lon = event.longitude
737
738
739 # =====
740 # -- CameraManager -----
741 # =====
742
743
744 class CameraManager(object):
745     def __init__(self, parent_actor, hud, gamma_correction):
746         self.sensor = None
747         self.surface = None
748         self.parent = parent_actor
749         self.hud = hud
750         self.recording = False
751         self.camera_transforms = [
752             carla.Transform(carla.Location(x=-5.5, z=2.8), carla.Rotation(pitch=-15)),
753             carla.Transform(carla.Location(x=1.6, z=1.7))]
754         self.transform_index = 1
755         self.sensors = [
756             ['sensor.camera.rgb', cc.Raw, 'Camera RGB'],
757             ['sensor.camera.depth', cc.Raw, 'Camera Depth (Raw)'],
758             ['sensor.camera.depth', cc.Depth, 'Camera Depth (Gray Scale)'],
759             ['sensor.camera.depth', cc.LogarithmicDepth, 'Camera Depth (Logarithmic Gray Scale)'],
760             ['sensor.camera.semantic_segmentation', cc.Raw, 'Camera Semantic Segmentation (Raw)'],
761             ['sensor.camera.semantic_segmentation', cc.CityScapesPalette,
762              'Camera Semantic Segmentation (CityScapes Palette)'],
763             ['sensor.lidar.ray_cast', None, 'Lidar (Ray-Cast)']]
764         world = self.parent.get_world()
765         bp_library = world.get_blueprint_library()
766         for item in self.sensors:
767             bp = bp_library.find(item[0])
768             if item[0].startswith('sensor.camera'):
769                 bp.set_attribute('image_size_x', str(hud.dim[0]))
770                 bp.set_attribute('image_size_y', str(hud.dim[1]))
771             elif item[0].startswith('sensor.lidar'):
772                 bp.set_attribute('range', '50')
773             item.append(bp)
774         self.index = None
775
776     def toggle_camera(self):
777         self.transform_index = (self.transform_index + 1) % len(self.camera_transforms)
778         self.sensor.set_transform(self.camera_transforms[self.transform_index])
779
780     def toggle_camera_backward(self):
781         self.transform_index = (self.transform_index - 1) % len(self.camera_transforms)
782         self.sensor.set_transform(self.camera_transforms[self.transform_index])
783
784     def set_sensor(self, index, notify=True):
785         index = index % len(self.sensors)
786         needs_respawn = True if self.index is None \
787             else self.sensors[index][0] != self.sensors[self.index][0]
788         if needs_respawn:
789             if self.sensor is not None:
790                 self.sensor.destroy()
791                 self.surface = None
792             self.sensor = self.parent.get_world().spawn_actor(
793                 self.sensors[index][1],
794                 self.camera_transforms[self.transform_index],
795                 attach_to=self.parent)
796             # We need to pass the lambda a weak reference to self to avoid
797             # circular reference.
798             weak_self = weakref.ref(self)
799             self.sensor.listen(lambda image: CameraManager._parse_image(weak_self, image))
800         if notify:
801             self.hud.notification(self.sensors[index][2])
802         self.index = index
803
804     def next_sensor(self):
805         self.set_sensor(self.index + 1)
806
807     def toggle_recording(self):
808         self.recording = not self.recording
809         self.hud.notification('Recording %s' % ('On' if self.recording else 'Off'))
810
811     def render(self, display):
812         if self.surface is not None:
813             display.blit(self.surface, (0, 0))
814
815     @staticmethod
816     def _parse_image(weak_self, image):
817         self = weak_self()
818         if not self:
819             return
820         if self.sensors[self.index][0].startswith('sensor.lidar'):
821             points = np.frombuffer(image.raw_data, dtype=np.dtype('f4'))
822             points = np.reshape(points, (int(points.shape[0] / 4), 4))
823             lidar_data = np.array(points[:, :2])
824             lidar_data *= min(self.hud.dim) / 100.0
825             lidar_data += 0.5 * self.hud.dim[0], 0.5 * self.hud.dim[1]
826             lidar_data = np.fabs(lidar_data) # pylint: disable=E1111
827             lidar_data = lidar_data.astype(np.int32)
828             lidar_data = np.reshape(lidar_data, (-1, 2))
829             lidar_img_size = (self.hud.dim[0], self.hud.dim[1], 3)
830             lidar_img = np.zeros(lidar_img_size)
831             lidar_img[tuple(lidar_data.T)] = (255, 255, 255)
832             self.surface = pygame.surfarray.make_surface(lidar_img)
833         else:
834             image.convert(self.sensors[self.index][1])
835             array = np.frombuffer(image.raw_data, dtype=np.dtype('uint8'))
836             array = np.reshape(array, (image.height, image.width, 4))
837             array = array[:, :, :3]
838             array = array[:, :, :~1]
839             self.surface = pygame.surfarray.make_surface(array.swapaxes(0, 1))

```

```

840         if self.recording:
841             image.save_to_disk('_out/%08d' % image.frame)
842
843
844 # =====
845 # -- game_loop() -----
846 # =====
847
848
849 def game_loop(args, scenario, path_details):
850     pygame.init()
851     pygame.font.init()
852     world = None
853     original_settings = None
854     try:
855         client = carla.Client(args.host, args.port)
856         client.set_timeout(20.0)
857         sim_world = client.get_world()
858         if args.sync:
859             original_settings = sim_world.get_settings()
860             settings = sim_world.get_settings()
861             if not settings.synchronous_mode:
862                 settings.synchronous_mode = True
863                 settings.fixed_delta_seconds = 0.05
864             sim_world.apply_settings(settings)
865             traffic_manager = client.get_trafficmanager()
866             traffic_manager.set_synchronous_mode(True)
867
868         display = pygame.display.set_mode(
869             (args.width, args.height),
870             pygame.HWSURFACE | pygame.DOUBLEBUF)
871
872         display.fill((0,0,0))
873         pygame.display.flip()
874         ### MY PART ###
875         scenario.setup(traffic_manager, sim_world)
876         scenario.spawn(client)
877         logger.info("The scenario was set. Spawning the hero vehicle and launching the simulation")
878         ### ### ###
879         hud = HUD(args.width, args.height)
880         try:
881             world = World(sim_world, hud, args, scenario, client, path_details)
882             controller = DualControl(world, args.autopilot)
883         except Exception as e:
884             logger.error(e)
885         if args.sync:
886             sim_world.tick()
887         else:
888             sim_world.wait_for_tick()
889
890         clock = pygame.time.Clock()
891         while True:
892             try:
893                 if args.sync:
894                     sim_world.tick()
895                     clock.tick_busy_loop(FPS)
896                     if controller.parse_events(client, world, clock, args.sync):
897                         return
898                     if world.tick():
899                         return
900                     world.render(display)
901                     pygame.display.flip()
902             except Exception as e:
903                 logger.error(e)
904         except Exception as e:
905             logger.error("Something went wrong trying to launch the driver.py", e)
906         finally:
907             ### MY PART ###
908             if scenario.populated:
909                 scenario.finish(client)
910             if world and world.manager:
911                 logger.info("Total time simulated: {}".format(str(world.simulation_time/1000)))
912             ### ### ###
913
914             if original_settings:
915                 sim_world.apply_settings(original_settings)
916
917             if world is not None:
918                 world.destroy()
919
920             pygame.quit()
921
922 # =====
923 # -- main() -----
924 # =====
925
926
927 argparser = argparse.ArgumentParser(
928     description='CARLA Manual Control Client')
929 argparser.add_argument(
930     '-v', '--verbose',
931     action='store_true',
932     dest='debug',
933     help='print debug information')
934 argparser.add_argument(
935     '--host',
936     metavar='H',
937     default='127.0.0.1',
938     help='IP of the host server (default: 127.0.0.1)')
939 argparser.add_argument(
940     '-p', '--port',
941     metavar='P',
942     default=2000,
943     type=int,
944     help='TCP port to listen to (default: 2000)')
945 argparser.add_argument(
946     '-a', '--autopilot',
947     action='store_true',
948     help='enable autopilot')
949 argparser.add_argument(
950     '--res',
951     metavar='WIDTHxHEIGHT',
952     default='1280x720',
953     help='window resolution (default: 1280x720)')
954 argparser.add_argument(
955     '--filter', '-f',
956     metavar='PATTERN',
957     default='vehicle.*',
958     help='actor filter (default: "vehicle.*")')
959 argparser.add_argument(
960     '--generation',
961     metavar='G',
962     default='2',
963     help='restrict to certain actor generation (values: "1","2","All" - default: "2")')
964 argparser.add_argument(
965     '--rolename',
966     metavar='NAME',
967     default='hero',
968     help='actor role name (default: "hero")')
969 argparser.add_argument(

```

```

970     '--gamma',
971     default=2.2,
972     type=float,
973     help='Gamma correction of the camera (default: 2.2)')
974 argparser.add_argument(
975     '--sync',
976     action='store_true',
977     help='Activate synchronous mode execution')
978 argparser.add_argument(
979     '--red',
980     default="0",
981     help='Red part of the RGB color palette to paint the hero car')
982 argparser.add_argument(
983     '--green',
984     default="0",
985     help='Green part of the RGB color palette to paint the hero car')
986 argparser.add_argument(
987     '--blue',
988     default="0",
989     help='Blue part of the RGB color palette to paint the hero car')
990 argparser.add_argument(
991     '-n' '--name',
992     metavar='F',
993     default="example",
994     help='Participant name (name)')
995 argparser.add_argument(
996     '-s', '--scenario',
997     metavar='F',
998     default="example",
999     help='Scenario name (name)')
1000 argparser.add_argument("s_b64")
1001 argparser.add_argument("p_b64")
1002 args = argparser.parse_args()
1003
1004 logger = logging.getLogger("driver-thread")
1005 logging.basicConfig(level=logging.INFO, filename="../../data/recordings/{}/session_logs.log".format(args.name), filemode="a", format="%(asctime)s - %(name)s - %
1006
1007 def main():
1008     args.width, args.height = [int(x) for x in args.res.split('x')]
1009
1010     print(__doc__)
1011
1012     try:
1013
1014         scenario_b64 = args.s_b64
1015         path_b64 = args.p_b64
1016
1017         scenario_bytes = base64.b64decode(scenario_b64.encode())
1018         path_bytes = base64.b64decode(path_b64.encode())
1019
1020         scenario = pickle.loads(scenario_bytes)
1021         path_details = pickle.loads(path_bytes)
1022         game_loop(args, scenario, path_details)
1023
1024     except KeyboardInterrupt as e:
1025         logger.warning('Keyboard interrupt while running driver.py script', e)
1026     except Exception as e:
1027         logger.error("Something went wrong.", e)
1028
1029
1030 if __name__ == '__main__':
1031     main()
1032

```

File: ./software/carla\_scripts/simulation\_manager.py

```

1  import json
2  import logging
3  import sys
4  from scenarios.scenario_factory import ScenarioFactory
5  from scenarios.path_builder import PathBuilder
6  import glob
7  import os
8  import time
9  import pickle
10 import subprocess
11 import base64
12 from threading import Thread
13
14 try:
15     sys.path.append(glob.glob('../carla/dist/carla-*%d.%d-%s.egg' % (
16         sys.version_info.major,
17         sys.version_info.minor,
18         'win-amd64' if os.name == 'nt' else 'linux-x86_64'))[0])
19 except IndexError:
20     pass
21
22 import carla
23
24
25 logger = logging.getLogger(__name__)
26
27 class SimulationManager():
28
29     def __init__(self, scenario_name, path_name, participant_name, vehicle_blueprint, randomization_seed, ai):
30         self.vehicle_blueprint = vehicle_blueprint
31         self.scenario_name = scenario_name
32         self.path_name = path_name
33         self.participant_name = participant_name
34         self.ai = ai
35         self.seed = randomization_seed
36         self.scenario = None
37
38     def begin_simulation(self):
39         try:
40             scenario_data = self.load_scenario()
41             path_details = self.determine_path_details(scenario_data)
42             self.change_map(path_details["town"])
43             self.change_weather(scenario_data)
44             self.scenario = ScenarioFactory.make_scenario(scenario_data, path_details, self.seed)
45             self.launch_driving_mode(self.scenario, path_details)
46
47         except Exception as e:
48             raise e
49
50     # Either generate the path or get it from file if such is defined
51     def determine_path_details(self, scenario_data):
52         if self.path_name:
53             details = self.read_path_details()
54         else:
55             logger.info("No predefined details to load. The details will be generated automatically.")
56             details = PathBuilder.generate_details(scenario_data)
57         return details
58

```

```

59 # Read path details to predefine the scenario if the path is specified
60 def read_path_details(self):
61     logger.info("Loading predefined details for the scenario")
62     file_path = ".../data/paths/{}.json".format(self.path_name)
63     try:
64         with open(file_path) as file:
65             details = json.load(file)
66             logger.info("Successfully read additional scenario details from {}.json data".format(self.path_name))
67             return details
68     except FileNotFoundError as e:
69         logger.error("Path {} file could not be found.".format(self.path_name), exc_info=True)
70         raise e
71     except json.JSONDecodeError as e:
72         logger.error("The file {}.json could not be decoded as JSON".format(self.path_name), exc_info=True)
73         raise e
74
75 # Get a file indicating scenario parameters
76 def load_scenario(self):
77     logger.info("Loading {} data".format(self.scenario_name))
78     file_path = ".../data/scenario_generation_data/generated_scenarios/{}.json".format(self.scenario_name)
79     try:
80         with open(file_path) as file:
81             scenario_data = json.load(file)
82             logger.info("Successfully read scenario {}.json data".format(self.scenario_name))
83             return scenario_data
84     except FileNotFoundError as e:
85         logger.error("Scenario {} file could not be found.".format(self.scenario_name), exc_info=True)
86         raise e
87     except json.JSONDecodeError as e:
88         logger.error("The file {}.json could not be decoded as JSON".format(self.scenario_name), exc_info=True)
89         raise e
90
91 # Used to kill all the actors in the simulation
92 def close_scenario(self):
93     client = carla.Client('127.0.0.1', 2000)
94     client.set_timeout(20.0)
95     if self.scenario is not None and self.scenario.populated:
96         logger.info("The scenario was running. Finishing it.")
97         self.scenario.finish(client)
98
99 # Change the weather in the simulation
100 def change_weather(self, scenario_data):
101     try:
102         logger.info("Changing the weather")
103         client = carla.Client('localhost', 2000)
104         world = client.get_world()
105         weather = carla.WeatherParameters(
106             cloudiness = scenario_data["cloudiness"],
107             precipitation = scenario_data["precipitation"],
108             precipitation_deposits = scenario_data["precipitation_deposits"],
109             wind_intensity = scenario_data["wind_intensity"],
110             sun_azimuth_angle = scenario_data["sun_azimuth_angle"],
111             sun_altitude_angle = scenario_data["sun_altitude_angle"],
112             fog_density = scenario_data["fog_density"],
113             fog_distance = scenario_data["fog_distance"],
114             wetness = scenario_data["wetness"],
115             fog_falloff = scenario_data["fog_falloff"],
116             scattering_intensity = scenario_data["scattering_intensity"],
117             mie_scattering_scale = scenario_data["mie_scattering_scale"],
118             rayleigh_scattering_scale = scenario_data["rayleigh_scattering_scale"],
119             # dust_storm = scenario_data["dust_storm"],
120         )
121     except Exception as e:
122         logger.error("Something wrong happened trying to change the weather. Aborting...", exc_info=True)
123         raise e
124
125 # Change the simulation map
126 def change_map(self, map):
127     try:
128         logger.info("Changing the map to {}".format(map))
129         client = carla.Client('localhost', 2000)
130         client.set_timeout(10.0)
131         world = client.load_world(map)
132         # Give 2 sec for the map to load
133         time.sleep(2)
134         logger.info("Finished changing the map")
135     except Exception as e:
136         logger.error("Something wrong happened trying to change the map. Aborting...", exc_info=True)
137         raise e
138
139 # Launch the first person driving mode
140 def launch_driving_mode(self, scenario, path_details):
141     scenario_bytes = pickle.dumps(scenario)
142     path_bytes = pickle.dumps(path_details)
143
144     s_b64 = base64.b64encode(scenario_bytes).decode()
145     p_b64 = base64.b64encode(path_bytes).decode()
146     try:
147         if self.ai:
148             logger.info("Launching self_driver.py script as a subprocess to let AI implementation drive the scenarios.")
149             process = subprocess.run(["python3", "self_driver.py", s_b64, p_b64, "-n{}".format(self.participant_name), "--sync", "-s{}".format(self.scenario_name)])
150         else:
151             logger.info("Launching driver.py script as a subprocess to drive the generated scenario.")
152             process = subprocess.run(["python3", "driver_keyboard.py", s_b64, p_b64, "-n{}".format(self.participant_name), "--sync", "-s{}".format(self.scenario_name)])
153             # process = subprocess.run(["python3", "driver_steeringwheel.py", s_b64, p_b64, "-n{}".format(self.participant_name), "--sync", "-s{}".format(self.scenario_name)])
154             # Uncomment to print statements coming from driver.py to the terminal
155             # print(process.stdout.decode(), process.stderr.decode())
156     except Exception as e:
157         logger.error("Could not launch driving script properly", exc_info=True)
158         raise e
159
160 # # MULTI-THREADING
161
162 # # Launch the first person driving mode
163 # def launch_driving_mode(self, scenario, path_details):
164 #     try:
165 #         flag = [False]
166 #
167 #         threads = [
168 #             Thread(target=simulate, args=[scenario, path_details, self.participant_name, self.scenario_name, self.vehicle_blueprint, self.ai, flag]),
169 #             Thread(target=draw_lanes, args=[path_details["path_checkpoints"], self.participant_name]),
170 #             Thread(target=draw_hero, args=[self.participant_name])
171 #         ]
172 #
173 #         for thread in threads:
174 #             thread.start()
175 #
176 #         while True:
177 #             time.sleep(1)
178 #             if flag[0]:
179 #                 for thread in threads:
180 #                     thread.join()
181 #                 break
182 #     except Exception as e:
183 #         logger.error("Something went wrong when starting the threads.", e)

```





```

1 from scenarios.scenario import Scenario
2 import sys
3 import os
4 import glob
5 import logging
6
7 try:
8     sys.path.append(glob.glob('../carla/dist/carla-*%d.%d-%s.egg' % (
9         sys.version_info.major,
10        sys.version_info.minor,
11        'win-amd64' if os.name == 'nt' else 'linux-x86_64'))[0])
12 except IndexError:
13     pass
14
15 import carla
16
17 logger = logging.getLogger(__name__)
18
19
20 class ScenarioFactory():
21
22     @staticmethod
23     def make_scenario(parameters, path_details, seed):
24         logger.info("Creating a scenario according to the generated path and scenario details")
25         return Scenario(parameters, path_details, seed)
26

```

File: ./software/carla\_scripts/scenarios/change\_weather.py

```

1 import glob
2 import os
3 import sys
4 import time
5 import random
6 import argparse
7
8 try:
9     sys.path.append(glob.glob('../carla/dist/carla-*%d.%d-%s.egg' % (
10        sys.version_info.major,
11        sys.version_info.minor,
12        'win-amd64' if os.name == 'nt' else 'linux-x86_64'))[0])
13 except IndexError:
14     pass
15
16 import carla
17
18 import logging
19 logger = logging.getLogger(__name__)
20 logging.basicConfig(level=logging.INFO, filename="log.log", filemode="a", format="%(asctime)s - %(name)s - %(levelname)s - %(message)s")
21
22 argparser = argparse.ArgumentParser(description=__doc__)
23 argparser.add_argument(
24     '-n', '--name',
25     default='scenario1',
26     help='Specify the name of the scenario')
27 argparser.add_argument(
28     '--log',
29     action='store_true',
30     help='Activate logging')
31 args = argparser.parse_args()
32
33 # Logging is used only when maps are changed during the simulations (logged to the participant's directory)
34 if args.log:
35     logging.basicConfig(level=logging.INFO, filename='../recording/recordings/{}/session_logs.log'.format(args.name), filemode="a", format="%(asctime)s - %(levelname)s - %(message)s")
36     logger = logging.getLogger("ChangeWeather")
37
38 # Log INFO message
39 def inform(message):
40     logger.info(message) if args.log else print(message)
41
42 def main():
43     try:
44         inform("Changing the weather for {}".format(args.name))
45         if args.name == "scenario1":
46             pass
47         else:
48             pass
49         inform("Finished changing the weather")
50     except KeyboardInterrupt:
51         message = "Manually exited the weather changing script."
52         logger.warn(message) if args.log else print(message)
53     except:
54         message = "Something wrong happened trying to change the weather. Aborting..."
55         logger.exception(message) if args.log else print(message)
56
57
58 if __name__ == '__main__':
59     main()

```

File: ./software/carla\_scripts/scenarios/change\_map.py

```

1 import glob
2 import os
3 import sys
4 import time
5 import random
6 import argparse
7
8 try:
9     sys.path.append(glob.glob('../carla/dist/carla-*%d.%d-%s.egg' % (
10         sys.version_info.major,
11         sys.version_info.minor,
12         'win-amd64' if os.name == 'nt' else 'linux-x86_64'))[0])
13 except IndexError:
14     pass
15
16 import carla
17
18 argparser = argparse.ArgumentParser(description=__doc__)
19 argparser.add_argument(
20     '-m', '--map',
21     default='Town04',
22     help='Specify the name of the map')
23 argparser.add_argument(
24     '-n', '--name',
25     default='ParticipantA',
26     help='Name of the participant')
27 args = argparser.parse_args()
28
29
30
31 def main():
32     try:
33         print("Changing the map to {}".format(args.map))
34         client = carla.Client('localhost', 2000)
35         world = client.load_world('{}'.format(args.map))
36         time.sleep(10)
37         print("Finished changing the map")
38     except KeyboardInterrupt:
39         message = "Manually exited the weather changing script."
40         print(message)
41     except Exception as e:
42         message = "Something wrong happened trying to change the map. Aborting..."
43         print(message)
44
45 if __name__ == '__main__':
46     main()

```

#### File: ./software/carla\_scripts/scenarios/scenario.py

```

1 import carla
2 import random
3 import numpy as np
4 import logging
5
6 logger = logging.getLogger(__name__)
7
8
9 class Scenario():
10
11     def __init__(self, parameters, path_details, seed):
12         self.scenario_parameters = parameters
13         self.path_details = path_details
14         # Used for reproducible randomization
15         self.seed = seed
16
17         self.vehicles_list = []
18         self.walkers_list = []
19         self.two_wheel_vehicles_list = []
20         self.all_id = []
21         self.all_actors = []
22
23         self.traffic_manager = None
24         self.synchronous_master = False
25         self.world = None
26         self.populated = False
27
28         self.walker_speed = None
29
30     def spawn(self, client):
31         logger.info("Spawning the scenario actors")
32         try:
33             blueprints_vehicles_all = get_actor_blueprints(self.world, "vehicle.*", "All")
34
35             blueprints_vehicles = get_safe_vehicle_blueprints(blueprints_vehicles_all)
36             blueprints_walkers = get_actor_blueprints(self.world, "walker.pedestrian.*", "2")
37             blueprints_two_wheel_vehicles = get_two_wheel_vehicle_blueprints(blueprints_vehicles_all)
38         except Exception as e:
39             logger.error("An error occurred trying to get the blueprints", exc_info=True)
40             raise e
41
42
43     try:
44         # Read all the parameters from the scenario object
45         parameters = self.scenario_parameters
46         number_of_pedestrians = int(parameters["number_of_pedestrians"])
47         number_of_vehicles = int(parameters["number_of_vehicles"])
48         number_of_two_wheel_vehicles = int(parameters["number_of_two_wheel_vehicles"])
49         proportion_of_speeding_vehicles = float(parameters["proportion_of_speeding_vehicles"])
50         proportion_of_vehicles_without_lights = float(parameters["proportion_of_vehicles_without_lights"])
51         proportion_of_light_ignoring_vehicles = float(parameters["proportion_of_light_ignoring_vehicles"])
52         light_ignoring_percent = float(parameters["light_ignoring_percent"])
53         proportion_of_sign_ignoring_vehicles = float(parameters["proportion_of_sign_ignoring_vehicles"])
54         sign_ignoring_percent = float(parameters["sign_ignoring_percent"])
55         proportion_of_vehicle_ignoring_vehicles = float(parameters["proportion_of_vehicle_ignoring_vehicles"])
56         vehicle_ignoring_percent = float(parameters["vehicle_ignoring_percent"])
57         proportion_of_walker_ignoring_vehicles = float(parameters["proportion_of_walker_ignoring_vehicles"])
58         walker_ignoring_percent = float(parameters["walker_ignoring_percent"])
59         proportion_of_keeping_right_vehicles = float(parameters["proportion_of_keeping_right_vehicles"])
60         keeping_right_percent = float(parameters["keeping_right_percent"])
61         proportion_of_lane_changing_vehicles = float(parameters["proportion_of_lane_changing_vehicles"])
62         lane_change_percent = float(parameters["lane_change_percent"])
63         proportion_of_misbehaving_pedestrians = float(parameters["proportion_of_misbehaving_pedestrians"])
64         proportion_of_running_pedestrians = float(parameters["proportion_of_running_pedestrians"])
65         proportion_of_road_crossing_pedestrians = float(parameters["proportion_of_road_crossing_pedestrians"])
66     except Exception as e:
67         logger.error("An error occurred trying to get retrieve scenario parameters", exc_info=True)
68         raise e
69
70
71 # ALL NECESSARY PARAMETERS WERE EXTRACTED
72
73 try:
74     SpawnActor = carla.command.SpawnActor
75     SetAutopilot = carla.command.SetAutopilot
76     FutureActor = carla.command.FutureActor
77

```

```

78     start = self.path_details["start_location"]
79     hero_spawn_location = carla.Location(x=float(start["x"]), y=float(start["y"]), z=float(start["z"]))
80
81     # Removes ones too close to the player's spawn location
82     spawn_points = self.world.get_map().get_spawn_points()
83     spawn_points = filter_spawn_points(spawn_points, hero_spawn_location, 5)
84     number_of_spawn_points = len(spawn_points)
85
86     total_number_of_vehicles = (number_of_vehicles + number_of_two_wheel_vehicles)
87     if total_number_of_vehicles <= number_of_spawn_points:
88         random.shuffle(spawn_points)
89
90     else:
91         logger.warning("There are too many vehicles for too few spawn points. Reducing the number of vehicles. There are {} spawn points for vehicles in t
92         difference = total_number_of_vehicles - number_of_spawn_points
93         if (number_of_vehicles - difference) >= 0:
94             number_of_vehicles = (number_of_vehicles - difference)
95         else:
96             # Over is going to be negative
97             over = (number_of_vehicles - difference)
98             number_of_two_wheel_vehicles = number_of_two_wheel_vehicles + over
99
100     total_number_of_vehicles = (number_of_vehicles + number_of_two_wheel_vehicles)
101     vehicle_spawn_points, two_wheel_vehicle_spawn_points = divide_list(spawn_points, number_of_vehicles, number_of_two_wheel_vehicles)
102
103     # Make all vehicles aim for the speed limit
104     self.traffic_manager.global_percentage_speed_difference(0)
105 except Exception as e:
106     logger.error("An error occurred trying to get the spawn points", exc_info=True)
107     raise e
108
109 try:
110     # -----
111     # Spawn vehicles
112     # -----
113     batch = []
114     for n, transform in enumerate(vehicle_spawn_points):
115         if n == number_of_vehicles:
116             break
117         blueprint = random.choice(blueprints_vehicles)
118         if blueprint.has_attribute('color'):
119             color = random.choice(blueprint.get_attribute('color').recommended_values)
120             blueprint.set_attribute('color', color)
121         if blueprint.has_attribute('driver_id'):
122             driver_id = random.choice(blueprint.get_attribute('driver_id').recommended_values)
123             blueprint.set_attribute('driver_id', driver_id)
124         blueprint.set_attribute('role_name', 'autopilot')
125         # Spawn the cars and set their autopilot and light state all together
126         batch.append(SpawnActor(blueprint, transform)
127                     .then(SetAutopilot(FutureActor, False, self.traffic_manager.get_port()))))
128
129     for response in client.apply_batch_sync(batch, self.synchronous_master):
130         if response.error:
131             logger.error(response.error)
132         else:
133             self.vehicles_list.append(response.actor_id)
134
135     logger.info("The {}({}) vehicles were successfully spawned".format(len(self.vehicles_list), number_of_vehicles))
136
137 except Exception as e:
138     logger.error("Error creating vehicles on the map.", exc_info=True)
139     raise e
140
141 try:
142     # -----
143     # Spawn two-wheel vehicles
144     # -----
145     batch = []
146     for n, transform in enumerate(two_wheel_vehicle_spawn_points):
147         if n == number_of_two_wheel_vehicles:
148             break
149         blueprint = random.choice(blueprints_two_wheel_vehicles)
150         if blueprint.has_attribute('color'):
151             color = random.choice(blueprint.get_attribute('color').recommended_values)
152             blueprint.set_attribute('color', color)
153         if blueprint.has_attribute('driver_id'):
154             driver_id = random.choice(blueprint.get_attribute('driver_id').recommended_values)
155             blueprint.set_attribute('driver_id', driver_id)
156         blueprint.set_attribute('role_name', 'autopilot')
157         # Spawn the cars and set their autopilot and light state all together
158         batch.append(SpawnActor(blueprint, transform)
159                     .then(SetAutopilot(FutureActor, False, self.traffic_manager.get_port()))))
160
161     for response in client.apply_batch_sync(batch, self.synchronous_master):
162         if response.error:
163             logger.error(response.error)
164         else:
165             self.two_wheel_vehicles_list.append(response.actor_id)
166
167     logger.info("The {}({}) two-wheel vehicles were successfully spawned".format(len(self.two_wheel_vehicles_list), number_of_two_wheel_vehicles))
168
169 except Exception as e:
170     logger.error("Error creating two-wheel vehicles on the map.", exc_info=True)
171     raise e
172
173 try:
174     logger.info("Setting the vehicle behavior according to the scenario parameters")
175     ### SETUP VEHICLE BEHAVIOUR
176
177     all_vehicles = self.vehicles_list + self.two_wheel_vehicles_list
178     amount_vehicles = len(all_vehicles)
179     all_vehicle_actors = self.world.get_actors(all_vehicles)
180
181     # Distance
182     for actor in all_vehicle_actors:
183         # Select a distance to keep between 1 and 5 metres randomly.
184         diff = random.randint(1, 5)
185         self.traffic_manager.auto_lane_change(actor, True)
186         self.traffic_manager.distance_to_leading_vehicle(actor, diff)
187
188     # Set not update vehicle lights for certain vehicles
189     amount_of = int(np.round(amount_vehicles * proportion_of_vehicles_without_lights))
190     random_vehicles = random.sample(all_vehicles, amount_of)
191     for v in random_vehicles:
192         actor = self.world.get_actor(v)
193         self.traffic_manager.update_vehicle_lights(actor, False)
194
195     # Update the lights for the rest
196     for v in all_vehicles:
197         if (v not in random_vehicles):
198             actor = self.world.get_actor(v)
199             self.traffic_manager.update_vehicle_lights(actor, True)
200
201     # Speeding
202     amount_of = int(np.round(amount_vehicles * proportion_of_speeding_vehicles))
203     random_vehicles = random.sample(all_vehicles, amount_of)
204     for v in random_vehicles:

```

```

208         actor = self.world.get_actor(v)
209         # Select a speeding increase randomly
210         diff = random.randint(-30, -1)
211         self.traffic_manager.vehicle_percentage_speed_difference(actor, diff)
212
213     # Light ignoring
214     amount_of = int(np.round(amount_vehicles * proportion_of_light_ignoring_vehicles))
215     random_vehicles = random.sample(all_vehicles, amount_of)
216     for v in random_vehicles:
217         actor = self.world.get_actor(v)
218         self.traffic_manager.ignore_lights_percentage(actor, light_ignoring_percent)
219
220     # Sign ignoring
221     amount_of = int(np.round(amount_vehicles * proportion_of_sign_ignoring_vehicles))
222     random_vehicles = random.sample(all_vehicles, amount_of)
223     for v in random_vehicles:
224         actor = self.world.get_actor(v)
225         self.traffic_manager.ignore_signs_percentage(actor, sign_ignoring_percent)
226
227     # Vehicle ignoring
228     amount_of = int(np.round(amount_vehicles * proportion_of_vehicle_ignoring_vehicles))
229     random_vehicles = random.sample(all_vehicles, amount_of)
230     for v in random_vehicles:
231         actor = self.world.get_actor(v)
232         self.traffic_manager.ignore_vehicles_percentage(actor, vehicle_ignoring_percent)
233
234     # Walker ignoring
235     amount_of = int(np.round(amount_vehicles * proportion_of_walker_ignoring_vehicles))
236     random_vehicles = random.sample(all_vehicles, amount_of)
237     for v in random_vehicles:
238         actor = self.world.get_actor(v)
239         self.traffic_manager.ignore_walkers_percentage(actor, walker_ignoring_percent)
240
241     # Keep right rule
242     amount_of = int(np.round(amount_vehicles * proportion_of_keeping_right_vehicles))
243     random_vehicles = random.sample(all_vehicles, amount_of)
244     for v in random_vehicles:
245         actor = self.world.get_actor(v)
246         self.traffic_manager.keep_right_rule_percentage(actor, keeping_right_percent)
247
248     # Lane changing left
249     amount_of = int(np.round(amount_vehicles * proportion_of_lane_changing_vehicles))
250     random_vehicles = random.sample(all_vehicles, amount_of)
251     for v in random_vehicles:
252         actor = self.world.get_actor(v)
253         self.traffic_manager.random_left_lanechange_percentage(actor, lane_change_percent)
254
255     # Lane changing right
256     amount_of = int(np.round(amount_vehicles * proportion_of_lane_changing_vehicles))
257     random_vehicles = random.sample(all_vehicles, amount_of)
258     for v in random_vehicles:
259         actor = self.world.get_actor(v)
260         self.traffic_manager.random_right_lanechange_percentage(actor, lane_change_percent)
261
262     logger.info("The behaviour of vehicles was successfully set")
263
264 except Exception as e:
265     logger.error("Error setting the behaviour of vehicles.", exc_info=True)
266     raise e
267
268
269 try:
270     # -----
271     # Spawn Walkers
272     # -----
273     # Random locations to spawn
274
275     number_of_pedestrians_initial = number_of_pedestrians
276
277     try:
278         spawn_points = []
279         for i in range(number_of_pedestrians):
280             spawn_point = carla.Transform()
281             loc = self.world.get_random_location_from_navigation()
282             if (loc != None):
283                 spawn_point.location = loc
284                 spawn_points.append(spawn_point)
285
286             if (len(spawn_points) >= number_of_pedestrians):
287                 logger.debug("There is a required amount of spawn points for pedestrians on the map.")
288             else:
289                 number_of_pedestrians = len(spawn_points)
290         except Exception as e:
291             logger.error(e)
292
293     # Spawn walker objects
294     walker_speed = []
295     walker_speed2 = []
296     logger.debug("Need to spawn {} pedestrians".format(number_of_pedestrians))
297     for i in range(number_of_pedestrians):
298
299         # Modify the blueprint
300         walker_bp = random.choice(blueprints_walkers)
301         # set as not invincible
302         if walker_bp.has_attribute('is_invincible'):
303             walker_bp.set_attribute('is_invincible', 'false')
304         # set the max speed
305         if walker_bp.has_attribute('speed'):
306             if (random.Random() > proportion_of_running_pedestrians):
307                 # walking
308                 walker_speed.append(walker_bp.get_attribute('speed').recommended_values[1])
309             else:
310                 # running
311                 walker_speed.append(walker_bp.get_attribute('speed').recommended_values[2])
312         else:
313             walker_speed.append(0.0)
314
315     logger.debug("The blueprint was set.")
316     try:
317         # Try to spawn the pedestrian 10000 times and
318         y = 0
319         not_spawned = True
320         while (not_spawned and y < 10000):
321             spawn_point = carla.Transform()
322             loc = self.world.get_random_location_from_navigation()
323             if (loc != None):
324                 spawn_point.location = loc
325                 batch = []
326                 batch.append(SpawnActor(walker_bp, spawn_point))
327                 results = client.apply_batch_sync(batch, True)
328                 if results[0].error:
329                     logger.debug(results[0].error)
330                     y += 1
331             else:
332                 not_spawned = False
333                 self.walkers_list.append({"id": results[0].actor_id})
334                 walker_speed2.append(walker_speed[i])
335                 break
336         else:
337             y += 1

```

```

338         except Exception as e:
339             logger.error(e)
340             logger.debug("There were {} respawns for this walker".format(y))
341
342
343     logger.info("Successfully spawned {} walkers.".format(len(self.walkers_list), number_of_pedestrians_initial))
344     self.walker_speed = walker_speed2
345
346     # Spawn the walker controller
347     batch = []
348     walker_controller_bp = self.world.get_blueprint_library().find('controller.ai.walker')
349     for i in range(len(self.walkers_list)):
350         batch.append(SpawnActor(walker_controller_bp, carla.Transform(), self.walkers_list[i]["id"]))
351     results = client.apply_batch_sync(batch, True)
352     for i in range(len(results)):
353         if results[i].error:
354             logger.error(results[i].error)
355         else:
356             self.walkers_list[i]["con"] = results[i].actor_id
357
358     # Put together the walkers and controllers id to get the objects from their id
359     for i in range(len(self.walkers_list)):
360         self.all_id.append(self.walkers_list[i]["con"])
361         self.all_id.append(self.walkers_list[i]["id"])
362     self.all_actors = self.world.get_actors(self.all_id)
363
364     except Exception as e:
365         logger.error("Error creating walkers on the map.", exc_info=True)
366         raise e
367
368     self.world.tick()
369     self.world.set_pedestrians_cross_factor(proportion_of_road_crossing_pedestrians)
370     self.populated = True
371
372 # Setup the traffic manager and other parameters before the simulation
373 def setup(self, traffic_manager, world):
374     logger.info("Setting up the scenario")
375     self.traffic_manager = traffic_manager
376     self.world = world
377     self.traffic_manager.set_global_distance_to_leading_vehicle(2.5)
378     self.traffic_manager.set_random_device_seed(self.seed)
379     self.world.set_pedestrians_seed(self.seed)
380     random.seed(self.seed)
381     self.traffic_manager.set_hybrid_physics_mode(True)
382     self.traffic_manager.set_hybrid_physics_radius(70.0)
383
384     settings = self.world.get_settings()
385     self.traffic_manager.set_synchronous_mode(True)
386     if not settings.synchronous_mode:
387         self.synchronous_master = True
388         settings.synchronous_mode = True
389         settings.fixed_delta_seconds = 0.05
390     else:
391         self.synchronous_master = False
392
393 # Unfreeze all actors and let them move as specified
394 def start(self):
395     logger.info("Unfreezing all actors in the simulation.")
396     port = self.traffic_manager.get_port()
397
398     # Unfreeze vehicles
399     for v in self.vehicles_list:
400         actor = self.world.get_actor(v)
401         actor.set_autopilot(True, port)
402
403     # Unfreeze two-wheel vehicles
404     for b in self.two_wheel_vehicles_list:
405         actor = self.world.get_actor(b)
406         actor.set_autopilot(True, port)
407
408     # Unfreeze walkers
409     for i in range(0, len(self.all_id), 2):
410         self.all_actors[i].start()
411         self.all_actors[i].go_to_location(self.world.get_random_location_from_navigation())
412         self.all_actors[i].set_max_speed(float(self.walker_speed*int(i/2)))
413
414 # Change settings back to default and destroy all actors
415 def finish(self, client):
416     logger.info("Finishing the scenario")
417     if self.synchronous_master:
418         settings = self.world.get_settings()
419         settings.synchronous_mode = False
420         settings.no_rendering_mode = False
421         settings.fixed_delta_seconds = None
422         self.world.apply_settings(settings)
423
424     logger.info('Destroying {} vehicles'.format(len(self.vehicles_list)))
425     client.apply_batch([carla.command.DestroyActor(x) for x in self.vehicles_list])
426
427     logger.info('Destroying {} two-wheel vehicles'.format(len(self.two_wheel_vehicles_list)))
428     client.apply_batch([carla.command.DestroyActor(x) for x in self.two_wheel_vehicles_list])
429     # Stop walker controllers (list is [controller, actor, controller, actor ...])
430     for i in range(0, len(self.all_id), 2):
431         self.all_actors[i].stop()
432     logger.info('Destroying {} walkers'.format(len(self.walkers_list)))
433     client.apply_batch([carla.command.DestroyActor(x) for x in self.all_id])
434
435     self.populated = False
436
437
438 def get_actor_blueprints(world, filter, generation):
439     bps = world.get_blueprint_library().filter(filter)
440     if generation.lower() == "all":
441         return bps
442
443     # If the filter returns only one bp, we assume that this one needed
444     # and therefore, we ignore the generation
445     if len(bps) == 1:
446         return bps
447
448     try:
449         int_generation = int(generation)
450         # Check if generation is in available generations
451         if int_generation in [1, 2]:
452             bps = [x for x in bps if int(x.get_attribute('generation')) == int_generation]
453             return bps
454         else:
455             logger.warning("Actor Generation is not valid. No actor will be spawned.")
456             return []
457     except:
458         logger.warning("Actor Generation is not valid. No actor will be spawned.")
459         return []
460
461 def get_safe_vehicle_blueprints(blueprints):
462     blueprints = [x for x in blueprints if int(x.get_attribute('number_of_wheels')) == 4]
463     blueprints = [x for x in blueprints if not x.id.endswith('microlino')]
464     blueprints = [x for x in blueprints if not x.id.endswith('carlaola')]
465     blueprints = [x for x in blueprints if not x.id.endswith('cbertruck')]
466     blueprints = [x for x in blueprints if not x.id.endswith('t2')]
467     blueprints = [x for x in blueprints if not x.id.endswith('sprinter')]
468     blueprints = [x for x in blueprints if not x.id.endswith('firetruck')]

```

```

468 blueprints = [x for x in blueprints if not x.id.endswith('ambulance')]
469 blueprints = sorted(blueprints, key=lambda bp: bp.id)
470 return blueprints
471
472 def get_two_wheel_vehicle_blueprints(blueprints):
473     return [x for x in blueprints if int(x.get_attribute('number_of_wheels')) == 2]
474
475 def distanceBetweenTwoLocations(start, finish):
476     first = np.array([start.x, start.y])
477     second = np.array([finish.x, finish.y])
478     return np.linalg.norm(first-second)
479
480 def filter_spawn_points(spawn_points, hero_spawn_location, min_distance):
481     try:
482         filtered_spawn_points = []
483         for point in spawn_points:
484             distance = distanceBetweenTwoLocations(point.location, hero_spawn_location)
485             # If the spawn location is further than two metres from the player's location, add to the filtered ones
486             if distance > min_distance:
487                 filtered_spawn_points.append(point)
488         return filtered_spawn_points
489     except Exception as e:
490         logger.error(e)
491
492 def divide_list(input_list, x, y):
493     first_list = input_list[:x]
494     second_list = input_list[x:y]
495     return first_list, second_list

```

File: ./software/carla\_scripts/scenarios/path\_builder.py

```

1  import logging
2  import xml.etree.ElementTree as ET
3  import os
4  import random
5  import json
6
7  logger = logging.getLogger(__name__)
8  # How many times to try and recreate the path if it keeps on failing or reaching a dead-end before
9  MAX_SEARCH_LENGTH = 10000
10 path_to_save_path = "../data/paths/"
11
12 class PathBuilder():
13
14     @staticmethod
15     # Used to find the path, the map and the start location automatically
16     def generate_details(scenario_data, save_filename = None):
17         try:
18             map, road_elements = PathBuilder.find_map_and_path(scenario_data)
19         except Exception as e:
20             logger.exception("Something went wrong generating the path", e)
21             raise Exception
22         try:
23             start_location = PathBuilder.find_start_location(road_elements[0])
24             path_details = PathBuilder.form_json(map, road_elements, start_location, save_filename)
25             logger.info(path_details)
26             if save_filename:
27                 logger.info("Path was saved to file {}.json in paths directory amongst other data".format(save_filename))
28             return path_details
29         except Exception as e:
30             logger.exception("Something went wrong combining the path object", e)
31             raise Exception
32
33     @staticmethod
34     # Used to return the newly generated path and everything in a json-like python structure
35     # And save it, if the save filename is specified
36     def form_json(map, road_elements, start_location, save_filename):
37         # Build a json-like file
38         # Save if needed and return
39         locations = []
40         for i in range(len(road_elements)):
41             if(i == len(road_elements)-1):
42                 locations.append(PathBuilder.get_road_location_object(road_elements[i], finish_road=True))
43             else:
44                 locations.append(PathBuilder.get_road_location_object(road_elements[i]))
45         attributes = {
46             "path_checkpoints": locations,
47             "start_location": start_location,
48             "town": map.replace(".", "xodr", "")
49         }
50         if save_filename:
51             with open("{}[{}].json".format(path_to_save_path, save_filename), "w+") as f:
52                 json.dump(attributes, f, indent=4)
53         return attributes
54
55     @staticmethod
56     # Used to find and return a start location among the given path_checkpoints
57     def find_start_location(road_element):
58         return PathBuilder.get_road_location_object(road_element, yaw=True)
59
60
61     @staticmethod
62     # Extract the location from the road object
63     def get_road_location_object(road, finish_road = None, yaw = None):
64         geometries = road.find("planView").findall("geometry")
65         geometry = geometries[len(geometries)-1] if finish_road else geometries[0]
66         x = float(geometry.attrib["x"])
67         y = float(geometry.attrib["y"])
68         rotation = float(geometry.attrib["hdg"])
69         # Determine z according to the height in that place
70         # yaw determine from the road_element
71         z = 0.4
72         start_location = {
73             "x": x,
74             "y": y,
75             "z": z,
76             "yaw": rotation
77         }
78         return start_location
79
80     @staticmethod
81     # Used to find the appropriate city
82     def find_map_and_path_parameters():
83         # Look for an appropriate map iterating through all files in maps/opendrive_format directory
84         logger.info("Looking for a map that could be used in the scenario and that satisfies the scenario parameters.")
85         map_filenames = os.listdir("../data/maps/opendrive_format")
86         already_considered_maps = []
87         for i in range(len(map_filenames)):
88             # Loop through all mapfiles randomly and if the match is found, return
89             unconsidered_maps = [file for file in map_filenames if file not in already_considered_maps and file.endswith(".xodr")]
90             random_map = random.sample(unconsidered_maps, 1)[0]
91             processed_map = PathBuilder.process_map(random_map, parameters)
92             if processed_map:
93                 return random_map, processed_map
94             else:
95                 logger.warning("The algorithm was unable to find a path given the scenario parameters in the map {}".format(random_map))

```

```

96         already_considered_maps.append(random_map)
97         if i == (len(map_filenames)-1):
98             logger.warning("The search was exhausted and no valid path was found for the given scenario and all the map files. RETURNING NONE.")
99         # If this place was reached, it is impossible to create a map
100         return None
101
102     @staticmethod
103     # Process the map file to check if it is appropriate for this scenario
104     def process_map(filename, parameters):
105         logger.info("Checking if {} is a valid map for this scenario".format(filename))
106         path_to_file = "../../data/maps/opendrive_format/{}".format(filename)
107         tree = ET.parse(path_to_file)
108         root = tree.getroot()
109
110         # Get all road and junction elements from the XML
111         roads = []
112         junctions = []
113         for road in root.findall("./road"):
114             roads.append(road)
115         for junction in root.findall("./junction"):
116             junctions.append(junction)
117
118         verified = PathBuilder.check_if_parameters_are_satisfied(parameters, roads, junctions, filename)
119         if not verified: return None
120
121         ## MAKE PATH
122         #print("Working on map: {}".format(filename))
123         logger.info("Working on map: {}".format(filename))
124         return PathBuilder.make_path(roads, junctions, parameters)
125
126     @staticmethod
127     # Checks if the scenario parameters are satisfied in the map
128     def check_if_parameters_are_satisfied(parameters, roads, junctions, filename):
129         # Check if there are enough junctions in the map
130         if len(junctions) >= parameters["number_of_junctions"]:
131             logger.info("Junction parameter is satisfied")
132         else:
133             logger.warning("There are not enough junctions in {} to accomodate the scenario.".format(filename))
134             return None
135
136         # Check if there is distance requirement is satisfied
137         total_road_length = 0.0
138         for road in roads:
139             total_road_length += float(road.attrib["length"])
140         if total_road_length >= float(parameters["distance_in_metres"]):
141             logger.info("Distance parameter is satisfied")
142         else:
143             logger.warning("There is not enough unique drivable road to accomodate the scenario {}".format(filename))
144             return None
145         return True
146
147     @staticmethod
148     # Used to run iteration and try to create a path from the given road and junction objects
149     def make_path(roads, junctions, parameters):
150         go = True
151         i = 0
152         path = None
153         # Try to create a path for a MAX_SEARCH_LENGTH times
154         while(go and i < MAX_SEARCH_LENGTH):
155             temp = None
156             try:
157                 temp = PathBuilder.get_path(roads, junctions, parameters["number_of_junctions"], parameters["distance_in_metres"])
158             except Exception as e:
159                 logger.warning("Something went wrong trying to create the path. Retrying.")
160             if temp:
161                 logger.info("The path was successfully generated")
162                 go = False
163                 path = temp
164             i += 1
165         if not path:
166             logger.warning("No success in {} tries to create a path".format(MAX_SEARCH_LENGTH))
167             # print("No success in {} tries...".format(i))
168             return path
169
170     @staticmethod
171     # Used to generate a random path over the map
172     def get_path(roads, junctions, max_number_of_junctions_allowed, min_distance):
173         path = []
174         chosen_road_queue = []
175         current_road = random.choice(roads)
176         path.append(current_road)
177         id, length, predecessor, successor = PathBuilder.analyse_road_element(current_road)
178         chosen_road_queue.append(id)
179         # print("Id of the first one: {}".format(id))
180         current_junctions = 0
181         current_length = length
182         while current_length < min_distance and current_junctions < max_number_of_junctions_allowed:
183             if len(chosen_road_queue) > 7 and chosen_road_queue[-8:].count(id) >= 3:
184                 # Check if the same road id appeared among the last 8 roads chosen
185                 # It means that the algorithm got stuck and now is moving back and forth
186                 return None
187             # print("Current length: {}".format(current_length))
188             # print("Already used roads: {}".format(chosen_road_queue))
189             if successor[0] == "junction":
190                 # print("Junction approached. Its id is: {}".format(successor[1]))
191                 current_junctions += 1
192                 junction = PathBuilder.get_the_right_junction(junctions, successor[1])
193                 all_exits = PathBuilder.analyse_junction_and_get_possible_exits(junction, id, chosen_road_queue)
194                 # If a dead-end was reached (no more exits)
195                 if len(all_exits) == 0:
196                     # Return a failure, because a dead-end was reached
197                     return None
198                 # print("All possible exits from this are: {}".format(all_exits))
199                 random_exit = random.choice(all_exits)
200                 new_road_to_take = PathBuilder.get_the_right_road(roads, random_exit)
201                 id, length, predecessor, successor = PathBuilder.analyse_road_element(new_road_to_take)
202                 # print("The chosen way is: {}".format(id))
203                 current_length += length
204                 chosen_road_queue.append(id)
205             else:
206                 new_road_to_take = PathBuilder.get_the_right_road(roads, successor[1])
207                 id, length, predecessor, successor = PathBuilder.analyse_road_element(new_road_to_take)
208                 # print("The chosen way is: {}".format(id))
209                 current_length += length
210                 chosen_road_queue.append(id)
211                 path.append(new_road_to_take)
212             # print("\n")
213
214         return path
215
216     @staticmethod
217     # Used to parse the xml road element and retrieve useful data
218     def analyse_road_element(road):
219         id = road.attrib["id"]
220         length = float(road.attrib["length"])
221         link = road.findall("link")[0]
222         predecessor = (link.find("predecessor").attrib["elementType"], link.find("predecessor").attrib["elementId"])
223         successor = (link.find("successor").attrib["elementType"], link.find("successor").attrib["elementId"])
224

```

```

226         return id, length, predecessor, successor
227
228     @staticmethod
229     # Used to
230     def analyse_junction_and_get_possible_exits(junction, incoming_road_id, already_used_ids):
231         # First try to get new roads to drive on
232         possible_next_roads = []
233         id = junction.attrib["id"]
234         for connection in junction.findall("connection"):
235             if connection.attrib["incomingRoad"] == incoming_road_id and connection.attrib["connectingRoad"] not in already_used_ids:
236                 possible_next_roads.append(connection.attrib["connectingRoad"])
237
238         # If there are no unique roads, allow picking of old ones
239         if len(possible_next_roads) == 0:
240             for connection in junction.findall("connection"):
241                 if connection.attrib["incomingRoad"] == incoming_road_id:
242                     possible_next_roads.append(connection.attrib["connectingRoad"])
243         # print("Possible next moves: {}".format(possible_next_roads))
244         return possible_next_roads
245
246     @staticmethod
247     # Used to find the right junction object from all junctions using its id
248     def get_the_right_junction(junctions, id):
249         for junction in junctions:
250             if junction.attrib["id"] == id:
251                 return junction
252         return None
253
254     @staticmethod
255     # Used to find the right road object from all road objects using its id
256     def get_the_right_road(roads, id):
257         for road in roads:
258             if road.attrib["id"] == id:
259                 return road
260         return None
261
262

```

File: ./software/carla\_scripts/clean.sh

```

1 #!/bin/bash
2 find . -name __pycache__ -type d -exec rm -rf {} +
3
4 find . -name ".*" -type f -delete
5
6 find . -name "*.sh" -exec chmod +x {} \;

```

File: ./software/carla\_scripts/wheel\_config.ini

```

1 [G29 Racing Wheel]
2 steering_wheel = 0
3 throttle = 2
4 brake = 3
5 reverse = 6
6 handbrake = 4

```

File: ./software/carla\_scripts/driver\_keyboard.py

```

1 #!/usr/bin/env python
2
3 # Copyright (c) 2019 Computer Vision Center (CVC) at the Universitat Autònoma de
4 # Barcelona (UAB).
5 #
6 # This work is licensed under the terms of the MIT license.
7 # For a copy, see <https://opensource.org/licenses/MIT>.
8
9 # Allows controlling a vehicle with a keyboard. For a simpler and more
10 # documented example, please take a look at tutorial.py.
11 """
12 Welcome to CARLA manual control.
13
14 Use ARROWS or WASD keys for control.
15
16     W            : throttle
17     S            : brake
18     A/D          : steer left/right
19     Q            : toggle reverse
20     Space        : hand-brake
21     P            : toggle autopilot
22     M            : toggle manual transmission
23     ,/.         : gear up/down
24
25     L            : toggle next light type
26     SHIFT + L    : toggle high beam
27     Z/X          : toggle right/left blinker
28     I            : toggle interior light
29
30     TAB         : change camera position
31
32     ENTER        : start route
33
34     F1           : toggle HUD
35     H/?         : toggle help
36     ESC         : quit
37 """
38
39 from __future__ import print_function
40
41
42
43
44 # =====
45 # -- find carla module -----
46 # =====
47
48
49 import glob
50 import os
51 import sys
52 import pickle
53 from scenarios.scenario import Scenario
54 import base64
55
56 try:
57     sys.path.append(glob.glob('../carla/dist/carla-*%d.%d-%s.egg' % (
58         sys.version_info.major,
59         sys.version_info.minor,
60         'win-amd64' if os.name == 'nt' else 'linux-x86_64'))[0])
61 except IndexError:
62     pass

```



```

63
64
65 # =====
66 # -- imports -----
67 # =====
68
69
70 import carla
71
72 from carla import ColorConverter as cc
73
74 import argparse
75 import collections
76 import datetime
77 import logging
78 import math
79 import random
80 import re
81 import weakref
82 from recording.manager import Manager
83 from config import FPS
84
85 try:
86     import pygame
87     from pygame import import KMOD_CTRL
88     from pygame import import KMOD_SHIFT
89     from pygame import import K_0
90     from pygame import import K_9
91     from pygame import import K_BACKQUOTE
92     from pygame import import K_BACKSPACE
93     from pygame import import K_DOWN
94     from pygame import import K_ESCAPE
95     from pygame import import K_COMMA
96     from pygame import import K_F1
97     from pygame import import K_LEFT
98     from pygame import import K_PERIOD
99     from pygame import import K_RIGHT
100    from pygame import import K_SLASH
101    from pygame import import K_SPACE
102    from pygame import import K_TAB
103    from pygame import import K_UP
104    from pygame import import K_a
105    from pygame import import K_b
106    from pygame import import K_c
107    from pygame import import K_d
108    from pygame import import K_g
109    from pygame import import K_h
110    from pygame import import K_i
111
112    from pygame import import K_l
113    from pygame import import K_m
114    from pygame import import K_n
115    from pygame import import K_o
116    from pygame import import K_p
117    from pygame import import K_q
118    from pygame import import K_r
119    from pygame import import K_s
120    from pygame import import K_t
121    from pygame import import K_v
122    from pygame import import K_w
123    from pygame import import K_x
124    from pygame import import K_z
125    from pygame import import K_MINUS
126    from pygame import import K_EQUALS
127 except ImportError:
128     raise RuntimeError('cannot import pygame, make sure pygame package is installed')
129
130 try:
131     import numpy as np
132 except ImportError:
133     raise RuntimeError('cannot import numpy, make sure numpy package is installed')
134
135
136 # =====
137 # -- Global functions -----
138 # =====
139
140
141 def find_weather_presets():
142     rgx = re.compile('.+?(?:(?<=[a-z])(?=[A-Z])|(?<[A-Z])(?=[A-Z][a-z])|$)')
143     name = lambda x: ' '.join(m.group(0) for m in rgx.finditer(x))
144     presets = [x for x in dir(carla.WeatherParameters) if re.match('[A-Z].+', x)]
145     return [(getattr(carla.WeatherParameters, x), name(x)) for x in presets]
146
147
148 def get_actor_display_name(actor, truncate=250):
149     name = ' '.join(actor.type_id.replace('_', '.').title().split('.')[1:])
150     return (name[:truncate - 1] + u'\u2026') if len(name) > truncate else name
151
152
153 def get_actor_blueprints(world, filter, generation):
154     bps = world.get_blueprint_library().filter(filter)
155     if generation.lower() == "all":
156         return bps
157
158     # If the filter returns only one bp, we assume that this one needed
159     # and therefore, we ignore the generation
160     if len(bps) == 1:
161         return bps
162
163     try:
164         int_generation = int(generation)
165         # Check if generation is in available generations
166         if int_generation in [1, 2]:
167             bps = [x for x in bps if int(x.get_attribute('generation')) == int_generation]
168             return bps
169         else:
170             print(" Warning! Actor Generation is not valid. No actor will be spawned.")
171             return []
172     except:
173         print(" Warning! Actor Generation is not valid. No actor will be spawned.")
174         return []
175
176 # =====
177 # -- World -----
178 # =====
179
180
181 class World(object):
182     def __init__(self, carla_world, hud, args, scenario, client, path_details):
183         self.world = carla_world
184         self.sync = args.sync
185         self.actor_role_name = args.role_name
186         self.path_details = path_details
187         self.scenario = scenario
188         self.client = client
189         self.scenario_name = args.scenario
190         self.participant_name = args.name
191         try:
192             self.map = self.world.get_map()

```

```

193 except RuntimeError as error:
194     print('RuntimeError: {}'.format(error))
195     print(' The server could not send the OpenDRIVE (.xodr) file:')
196     print(' Make sure it exists, has the same name of your town, and is correct.')
197     sys.exit(1)
198 self.spawn_point = None
199 self.init_spawn_point(path_details["start_location"])
200 self.hud = hud
201 self.player = None
202 self.collision_sensor = None
203 self.lane_invasion_sensor = None
204 self.gnss_sensor = None
205 self.imu_sensor = None
206 self.radar_sensor = None
207 self.camera_manager = None
208 self.weather_presets = find_weather_presets()
209 self.weather_index = 0
210 self.actor_filter = args.filter
211 self.actor_generation = args.generation
212 self.gamma = args.gamma
213 self.restart(args)
214 self.recording_enabled = 0
215
216 # All about the simulation
217 self.simulation_clock = None
218 self.simulation_start_tick = 0
219 self.simulation_time = 0
220 self.manager = None
221
222 self.recording_start = 0
223 self.constant_velocity_enabled = False
224 self.show_vehicle_telemetry = False
225 self.doors_are_open = False
226 self.current_map_layer = 0
227 self.map_layer_names = [
228     carla.MapLayer.NONE,
229     carla.MapLayer.Buildings,
230     carla.MapLayer.Decals,
231     carla.MapLayer.Foliage,
232     carla.MapLayer.Ground,
233     carla.MapLayer.ParkedVehicles,
234     carla.MapLayer.Particles,
235     carla.MapLayer.Props,
236     carla.MapLayer.StreetLights,
237     carla.MapLayer.Walls,
238     carla.MapLayer.All
239 ]
240
241 def init_spawn_point(self, start_location):
242     try:
243         self.spawn_point = carla.Transform(carla.Location(x=start_location["x"], y=start_location["y"], z=start_location["z"]), carla.Rotation(pitch=0.0, yaw=0.0))
244     except Exception as e:
245         logger.error("Could not initiate spawn location for the driver", e)
246
247 def restart(self, args):
248     self.player_max_speed = 1.589
249     self.player_max_speed_fast = 3.713
250     # Keep same camera config if the camera manager exists.
251     cam_index = self.camera_manager.index if self.camera_manager is not None else 0
252     cam_pos_index = self.camera_manager.transform_index if self.camera_manager is not None else 0
253     # Get a random blueprint.
254     blueprint = random.choice(get_actor_blueprints(self.world, self.actor_filter, self.actor_generation))
255     blueprint.set_attribute('role_name', self.actor_role_name)
256     if blueprint.has_attribute('color'):
257         blueprint.set_attribute('color', '{}({},{})'.format(args.red, args.green, args.blue))
258     if blueprint.has_attribute('driver_id'):
259         driver_id = random.choice(blueprint.get_attribute('driver_id').recommended_values)
260         blueprint.set_attribute('driver_id', driver_id)
261     if blueprint.has_attribute('is_invincible'):
262         blueprint.set_attribute('is_invincible', 'true')
263     # set the max speed
264     if blueprint.has_attribute('speed'):
265         self.player_max_speed = float(blueprint.get_attribute('speed').recommended_values[1])
266         self.player_max_speed_fast = float(blueprint.get_attribute('speed').recommended_values[2])
267
268 # Spawn the player.
269 if self.player is not None:
270     spawn_point = self.player.get_transform()
271     spawn_point.location.z += 2.0
272     spawn_point.rotation.roll = 0.0
273     spawn_point.rotation.pitch = 0.0
274     self.destroy()
275     self.player = self.world.try_spawn_actor(blueprint, spawn_point)
276     self.show_vehicle_telemetry = False
277     self.modify_vehicle_physics(self.player)
278 while self.player is not None:
279     if not self.map.get_spawn_points():
280         print('There are no spawn points available in your map/town.')
281         print('Please add some Vehicle Spawn Point to your UE4 scene.')
282         sys.exit(1)
283     if self.spawn_point is None:
284         spawn_points = self.map.get_spawn_points()
285         self.spawn_point = random.choice(spawn_points) if spawn_points else carla.Transform()
286         self.player = self.world.try_spawn_actor(blueprint, self.spawn_point)
287         self.show_vehicle_telemetry = False
288         self.modify_vehicle_physics(self.player)
289 # Set up the sensors.
290 self.collision_sensor = CollisionSensor(self.player, self.hud)
291 self.lane_invasion_sensor = LaneInvasionSensor(self.player, self.hud)
292 self.gnss_sensor = GnssSensor(self.player)
293 self.imu_sensor = IMUSensor(self.player)
294 self.camera_manager = CameraManager(self.player, self.hud, self.gamma)
295 self.camera_manager.transform_index = cam_pos_index
296 self.camera_manager.set_sensor(cam_index, notify=False)
297 actor_type = get_actor_display_name(self.player)
298 self.hud.notification(actor_type)
299
300 if self.sync:
301     self.world.tick()
302 else:
303     self.world.wait_for_tick()
304
305 def toggle_radar(self):
306     if self.radar_sensor is None:
307         self.radar_sensor = RadarSensor(self.player)
308     elif self.radar_sensor.sensor is not None:
309         self.radar_sensor.sensor.destroy()
310         self.radar_sensor = None
311
312 def modify_vehicle_physics(self, actor):
313     # If actor is not a vehicle, we cannot use the physics control
314     try:
315         physics_control = actor.get_physics_control()
316         physics_control.use_sweep_wheel_collision = True
317         actor.apply_physics_control(physics_control)
318     except Exception:
319         pass
320
321 def startScenario(self):
322     if self.simulation_clock is None:
323         logger.info("Starting the scenario")

```

[illegible]

```

453         currentLights = carla.VehicleLightState.Position
454         currentLights = carla.VehicleLightState.LowBeam
455         currentLights = carla.VehicleLightState.Fog
456     elif event.key == K_i:
457         currentLights = carla.VehicleLightState.Interior
458     elif event.key == K_2:
459         currentLights = carla.VehicleLightState.LeftBlinker
460     elif event.key == K_x:
461         currentLights = carla.VehicleLightState.RightBlinker
462
463 if not self._autopilot_enabled:
464     if isinstance(self._control, carla.VehicleControl):
465         self._parse_vehicle_keys(pygame.key.get_pressed(), clock.get_time())
466         self._control.reverse = self._control.gear < 0
467         # Set automatic control-related vehicle lights
468         if self._control.brake:
469             currentLights = carla.VehicleLightState.Brake
470         else: # Remove the Brake flag
471             currentLights = ~carla.VehicleLightState.Brake
472         if self._control.reverse:
473             currentLights = carla.VehicleLightState.Reverse
474         else: # Remove the Reverse flag
475             currentLights = ~carla.VehicleLightState.Reverse
476         if currentLights != self._lights: # Change the light state only if necessary
477             self._lights = currentLights
478             world.player.set_light_state(carla.VehicleLightState(self._lights))
479     elif isinstance(self._control, carla.WalkerControl):
480         self._parse_walker_keys(pygame.key.get_pressed(), clock.get_time(), world)
481     world.player.apply_control(self._control)
482
483 def _parse_vehicle_keys(self, keys, milliseconds):
484     if keys[K_UP] or keys[K_W]:
485         self._control.throttle = min(self._control.throttle + 0.01, 1.00)
486     else:
487         self._control.throttle = 0.0
488
489     if keys[K_DOWN] or keys[K_S]:
490         self._control.brake = min(self._control.brake + 0.2, 1)
491     else:
492         self._control.brake = 0
493
494     steer_increment = 5e-4 * milliseconds
495     if keys[K_LEFT] or keys[K_A]:
496         if self._steer_cache > 0:
497             self._steer_cache = 0
498         else:
499             self._steer_cache -= steer_increment
500     elif keys[K_RIGHT] or keys[K_D]:
501         if self._steer_cache < 0:
502             self._steer_cache = 0
503         else:
504             self._steer_cache += steer_increment
505     else:
506         self._steer_cache = 0.0
507     self._steer_cache = min(0.7, max(-0.7, self._steer_cache))
508     self._control.steer = round(self._steer_cache, 1)
509     self._control.hand_brake = keys[K_SPACE]
510
511 def _parse_walker_keys(self, keys, milliseconds, world):
512     self._control.speed = 0.0
513     if keys[K_DOWN] or keys[K_S]:
514         self._control.speed = 0.0
515     if keys[K_LEFT] or keys[K_A]:
516         self._control.speed = 0.01
517     if keys[K_RIGHT] or keys[K_D]:
518         self._control.speed = 0.01
519     if keys[K_UP] or keys[K_W]:
520         self._rotation.yaw += 0.08 * milliseconds
521     if keys[K_UP] or keys[K_W]:
522         self._control.speed = world.player_max_speed_fast if pygame.key.get_mods() & KMOD_SHIFT else world.player_max_speed
523     self._control.jump = keys[K_SPACE]
524     self._rotation.yaw = round(self._rotation.yaw, 1)
525     self._control.direction = self._rotation.get_forward_vector()
526
527 @staticmethod
528 def _is_quit_shortcut(key):
529     return (key == K_ESCAPE) or (key == K_q and pygame.key.get_mods() & KMOD_CTRL)
530
531 # =====
532 # -- HUD -----
533 # =====
534
535 class HUD(object):
536     def __init__(self, width, height):
537         self.dim = (width, height)
538         font = pygame.font.Font(pygame.font.get_default_font(), 20)
539         font_name = 'courier' if os.name == 'nt' else 'mono'
540         fonts = [x for x in pygame.font.get_fonts() if font_name in x]
541         default_font = 'ubuntumono'
542         mono = default_font if default_font in fonts else fonts[0]
543         mono = pygame.font.match_font(mono)
544         self._font_mono = pygame.font.Font(mono, 12 if os.name == 'nt' else 14)
545         self._notifications = FadingText(font, (width, 40), (0, height - 40))
546         self.help = HelpText(pygame.font.Font(mono, 16), width, height)
547         self.server_fps = 0
548         self.frame = 0
549         self.simulation_time = 0
550         self._show_info = True
551         self._info_text = []
552         self._server_clock = pygame.time.Clock()
553
554     def on_world_tick(self, timestamp):
555         self._server_clock.tick()
556         self.server_fps = self._server_clock.get_fps()
557         self.frame = timestamp.frame
558         self.simulation_time = timestamp.elapsed_seconds
559
560     def tick(self, world, time):
561         self._notifications.tick(world, time)
562         if not self._show_info:
563             return
564         t = world.player.get_transform()
565         v = world.player.get_velocity()
566         c = world.player.get_control()
567         compass = world.imu_sensor.compass
568         heading = 'N' if compass > 270.5 or compass < 89.5 else ''
569         heading += 'S' if 90.5 < compass < 269.5 else ''
570         heading += 'E' if 0.5 < compass < 179.5 else ''
571         heading += 'W' if 180.5 < compass < 359.5 else ''
572         colhist = world.collision_sensor.get_collision_history()
573         collision = [colhist[x + self.frame - 200] for x in range(0, 200)]
574         max_col = max(1.0, max(collision))
575         collision = [x / max_col for x in collision]
576         vehicles = world.world.get_actors().filter('vehicle.*')
577         self._info_text = [
578             'Vehicle: %20s' % get_actor_display_name(world.player, truncate=20),
579             'Route time: %20s' % datetime.timedelta(seconds=int(time)),
580             ''
581         ]

```

```

583         'Speed: % 15.0f km/h' % (3.6 * math.sqrt(v.x**2 + v.y**2 + v.z**2)),
584         ''
585     if isinstance(c, carla.VehicleControl):
586         self._info_text += [
587             ('Throttle:', c.throttle, 0.0, 1.0),
588             ('Steer:', c.steer, -1.0, 1.0),
589             ('Brake:', c.brake, 0.0, 1.0),
590             ('Reverse:', c.reverse),
591             ('Hand brake:', c.hand_brake),
592             ('Manual:', c.manual_gear_shift),
593             'Gear: %s' % ['R', 0: 'N'].get(c.gear, c.gear)]
594     self._info_text += [
595         '',
596         'Collision:',
597         collision,
598         '']
599
600     def toggle_info(self):
601         self._show_info = not self._show_info
602
603     def notification(self, text, seconds=2.0):
604         self._notifications.set_text(text, seconds=seconds)
605
606     def error(self, text):
607         self._notifications.set_text('Error: %s' % text, (255, 0, 0))
608
609     def render(self, display):
610         if self._show_info:
611             info_surface = pygame.Surface((220, self.dim[1]))
612             info_surface.set_alpha(100)
613             display.blit(info_surface, (0, 0))
614             v_offset = 4
615             bar_h_offset = 100
616             bar_width = 106
617             for item in self._info_text:
618                 if v_offset + 18 > self.dim[1]:
619                     break
620                 if isinstance(item, list):
621                     if len(item) > 1:
622                         points = [(x + 8, v_offset + 8 + (1.0 - y) * 30) for x, y in enumerate(item)]
623                         pygame.draw.lines(display, (255, 136, 0), False, points, 2)
624                     item = None
625                     v_offset += 18
626                 elif isinstance(item, tuple):
627                     if isinstance(item[1], bool):
628                         rect = pygame.Rect((bar_h_offset, v_offset + 8), (6, 6))
629                         pygame.draw.rect(display, (255, 255, 255), rect, 0 if item[1] else 1)
630                     else:
631                         rect_border = pygame.Rect((bar_h_offset, v_offset + 8), (bar_width, 6))
632                         pygame.draw.rect(display, (255, 255, 255), rect_border, 1)
633                         f = (item[1] - item[2]) / (item[3] - item[2])
634                         if item[2] < 0.0:
635                             rect = pygame.Rect((bar_h_offset + f * (bar_width - 6), v_offset + 8), (6, 6))
636                         else:
637                             rect = pygame.Rect((bar_h_offset, v_offset + 8), (f * bar_width, 6))
638                         pygame.draw.rect(display, (255, 255, 255), rect)
639                     item = item[0]
640                 if item: # At this point has to be a str.
641                     surface = self._font_mono.render(item, True, (255, 255, 255))
642                     display.blit(surface, (8, v_offset))
643                     v_offset += 18
644             self._notifications.render(display)
645             self.help.render(display)
646
647 # =====
648 # -- FadingText -----
649 # =====
650
651
652
653 class FadingText(object):
654     def __init__(self, font, dim, pos):
655         self.font = font
656         self.dim = dim
657         self.pos = pos
658         self.seconds_left = 0
659         self.surface = pygame.Surface(self.dim)
660
661     def set_text(self, text, color=(255, 255, 255), seconds=2.0):
662         text_texture = self.font.render(text, True, color)
663         self.surface = pygame.Surface(self.dim)
664         self.seconds_left = seconds
665         self.surface.fill((0, 0, 0))
666         self.surface.blit(text_texture, (10, 11))
667
668     def tick(self, _time):
669         delta_seconds = 1e-3 * _time
670         self.seconds_left = max(0.0, self.seconds_left - delta_seconds)
671         self.surface.set_alpha(500.0 * self.seconds_left)
672
673     def render(self, display):
674         display.blit(self.surface, self.pos)
675
676 # =====
677 # -- HelpText -----
678 # =====
679
680
681
682 class HelpText(object):
683     """Helper class to handle text output using pygame"""
684     def __init__(self, font, width, height):
685         lines = _doc_.split('\n')
686         self.font = font
687         self.line_space = 18
688         self.dim = (780, len(lines) * self.line_space + 12)
689         self.pos = (0.5 * width - 0.5 * self.dim[0], 0.5 * height - 0.5 * self.dim[1])
690         self.seconds_left = 0
691         self.surface = pygame.Surface(self.dim)
692         self.surface.fill((0, 0, 0))
693         for n, line in enumerate(lines):
694             text_texture = self.font.render(line, True, (255, 255, 255))
695             self.surface.blit(text_texture, (22, n * self.line_space))
696             self._render = False
697         self.surface.set_alpha(220)
698
699     def toggle(self):
700         self._render = not self._render
701
702     def render(self, display):
703         if self._render:
704             display.blit(self.surface, self.pos)
705
706 # =====
707 # -- CollisionSensor -----
708 # =====
709
710
711
712

```

```

713 class CollisionSensor(object):
714     def __init__(self, parent_actor, hud):
715         self.sensor = None
716         self.history = []
717         self.parent = parent_actor
718         self.hud = hud
719         world = self.parent.get_world()
720         bp = world.get_blueprint_library().find('sensor.other.collision')
721         self.sensor = world.spawn_actor(bp, carla.Transform(), attach_to=self.parent)
722         # We need to pass the lambda a weak reference to self to avoid circular
723         # reference.
724         weak_self = weakref.ref(self)
725         self.sensor.listen(lambda event: CollisionSensor._on_collision(weak_self, event))
726
727     def get_collision_history(self):
728         history = collections.defaultdict(int)
729         for frame, intensity in self.history:
730             history[frame] += intensity
731         return history
732
733     @staticmethod
734     def _on_collision(weak_self, event):
735         self = weak_self()
736         if not self:
737             return
738         actor_type = get_actor_display_name(event.other_actor)
739         self.hud.notification('Collision with %r' % actor_type)
740         impulse = event.normal_impulse
741         intensity = math.sqrt(impulse.x**2 + impulse.y**2 + impulse.z**2)
742         self.history.append((event.frame, intensity))
743         if len(self.history) > 4000:
744             self.history.pop(0)
745
746 # =====
747 # -- LaneInvasionSensor -----
748 # =====
749
750
751 class LaneInvasionSensor(object):
752     def __init__(self, parent_actor, hud):
753         self.sensor = None
754
755         # If the spawn object is not a vehicle, we cannot use the Lane Invasion Sensor
756         if parent_actor.type_id.startswith("vehicle."):
757             self.parent = parent_actor
758             self.hud = hud
759             world = self.parent.get_world()
760             bp = world.get_blueprint_library().find('sensor.other.lane_invasion')
761             self.sensor = world.spawn_actor(bp, carla.Transform(), attach_to=self.parent)
762             # We need to pass the lambda a weak reference to self to avoid circular
763             # reference.
764             weak_self = weakref.ref(self)
765             self.sensor.listen(lambda event: LaneInvasionSensor._on_invasion(weak_self, event))
766
767     @staticmethod
768     def _on_invasion(weak_self, event):
769         self = weak_self()
770         if not self:
771             return
772         lane_types = set(x.type for x in event.crossed_lane_markings)
773         text = ['%r' % str(x).split()[-1] for x in lane_types]
774         self.hud.notification('Crossed line %s' % ' and '.join(text))
775
776 # =====
777 # -- GnssSensor -----
778 # =====
779
780
781 class GnssSensor(object):
782     def __init__(self, parent_actor):
783         self.sensor = None
784         self.parent = parent_actor
785         self.lat = 0.0
786         self.lon = 0.0
787         world = self.parent.get_world()
788         bp = world.get_blueprint_library().find('sensor.other.gnss')
789         self.sensor = world.spawn_actor(bp, carla.Transform(carla.Location(x=1.0, z=2.8)), attach_to=self.parent)
790         # We need to pass the lambda a weak reference to self to avoid circular
791         # reference.
792         weak_self = weakref.ref(self)
793         self.sensor.listen(lambda event: GnssSensor._on_gnss_event(weak_self, event))
794
795     @staticmethod
796     def _on_gnss_event(weak_self, event):
797         self = weak_self()
798         if not self:
799             return
800         self.lat = event.latitude
801         self.lon = event.longitude
802
803 # =====
804 # -- IMUSensor -----
805 # =====
806
807
808 class IMUSensor(object):
809     def __init__(self, parent_actor):
810         self.sensor = None
811         self.parent = parent_actor
812         self.accelerometer = (0.0, 0.0, 0.0)
813         self.gyroscope = (0.0, 0.0, 0.0)
814         self.compass = 0.0
815         world = self.parent.get_world()
816         bp = world.get_blueprint_library().find('sensor.other.imu')
817         self.sensor = world.spawn_actor(
818             bp, carla.Transform(), attach_to=self.parent)
819         # We need to pass the lambda a weak reference to self to avoid circular
820         # reference.
821         weak_self = weakref.ref(self)
822         self.sensor.listen(
823             lambda sensor_data: IMUSensor._IMU_callback(weak_self, sensor_data))
824
825     @staticmethod
826     def _IMU_callback(weak_self, sensor_data):
827         self = weak_self()
828         if not self:
829             return
830         limits = (-99.9, 99.9)
831         self.accelerometer = (
832             max(limits[0], min(limits[1], sensor_data.accelerometer.x)),
833             max(limits[0], min(limits[1], sensor_data.accelerometer.y)),
834             max(limits[0], min(limits[1], sensor_data.accelerometer.z)))
835         self.gyroscope = (
836             max(limits[0], min(limits[1], math.degrees(sensor_data.gyroscope.x))),
837             max(limits[0], min(limits[1], math.degrees(sensor_data.gyroscope.y))),
838             max(limits[0], min(limits[1], math.degrees(sensor_data.gyroscope.z))))
839         self.compass = math.degrees(sensor_data.compass)

```

```

843
844
845 # =====
846 # -- RadarSensor -----
847 # =====
848
849
850 class RadarSensor(object):
851     def __init__(self, parent_actor):
852         self.sensor = None
853         self._parent = parent_actor
854         bound_x = 0.5 + self._parent.bounding_box.extent.x
855         bound_y = 0.5 + self._parent.bounding_box.extent.y
856         bound_z = 0.5 + self._parent.bounding_box.extent.z
857
858         self.velocity_range = 7.5 # m/s
859         world = self._parent.get_world()
860         self.debug = world.debug
861         bp = world.get_blueprint_library().find('sensor.other.radar')
862         bp.set_attribute('horizontal_fov', str(35))
863         bp.set_attribute('vertical_fov', str(20))
864         self.sensor = world.spawn_actor(
865             bp
866             carla.Transform(
867                 carla.Location(x=bound_x + 0.05, z=bound_z+0.05),
868                 carla.Rotation(pitch=5)),
869             attach_to=self._parent)
870         # We need a weak reference to self to avoid circular reference.
871         weak_self = weakref.ref(self)
872         self.sensor.listen(
873             lambda radar_data: RadarSensor._Radar_callback(weak_self, radar_data))
874
875     @staticmethod
876     def _Radar_callback(weak_self, radar_data):
877         self = weak_self()
878         if not self:
879             return
880         # To get a numpy [[vel, altitude, azimuth, depth],...[,...]]:
881         # points = np.frombuffer(radar_data.raw_data, dtype=np.dtype('f4'))
882         # points = np.reshape(points, (len(radar_data), 4))
883
884         current_rot = radar_data.transform.rotation
885         for detect in radar_data:
886             azi = math.degrees(detect.azimuth)
887             alt = math.degrees(detect.altitude)
888             # The 0.25 adjusts a bit the distance so the dots can
889             # be properly seen
890             fw_vec = carla.Vector3D(x=detect.depth - 0.25)
891             carla.Transform(
892                 carla.Location(),
893                 carla.Rotation(
894                     pitch=current_rot.pitch + alt,
895                     yaw=current_rot.yaw + azi,
896                     roll=current_rot.roll)).transform(fw_vec)
897
898             def clamp(min_v, max_v, value):
899                 return max(min_v, min(value, max_v))
900
901             norm_velocity = detect.velocity / self.velocity_range # range [-1, 1]
902             r = int(clamp(0.0, 1.0, 1.0 - norm_velocity) * 255.0)
903             g = int(clamp(0.0, 1.0, 1.0 - abs(norm_velocity)) * 255.0)
904             b = int(abs(clamp(- 1.0, 0.0, - 1.0 - norm_velocity)) * 255.0)
905             self.debug.draw_point(
906                 radar_data.transform.location + fw_vec,
907                 size=0.075,
908                 life_time=0.06,
909                 persistent_lines=False,
910                 color=carla.Color(r, g, b))
911
912 # =====
913 # -- CameraManager -----
914 # =====
915
916
917 class CameraManager(object):
918     def __init__(self, parent_actor, hud, gamma_correction):
919         self.sensor = None
920         self.surface = None
921         self._parent = parent_actor
922         self.hud = hud
923         self.recording = False
924         bound_x = 0.5 + self._parent.bounding_box.extent.x
925         bound_y = 0.5 + self._parent.bounding_box.extent.y
926         bound_z = 0.5 + self._parent.bounding_box.extent.z
927         Attachment = carla.AttachmentType
928
929         self._camera_transforms = [
930             # Third-person
931             (carla.Transform(carla.Location(x=-5.0, z=2.0), carla.Rotation(pitch=6.0)), Attachment.SpringArm),
932             # Watch back
933             (carla.Transform(carla.Location(x=4.0, z=2.0), carla.Rotation(yaw=0, pitch=6)), Attachment.SpringArm),
934             # Watch to the left
935             (carla.Transform(carla.Location(x=0, z=2.0, y=3), carla.Rotation(yaw=-90, pitch=0)), Attachment.Rigid),
936             # Watch to the right
937             (carla.Transform(carla.Location(x=0, z=2.0, y=-3), carla.Rotation(yaw=90, pitch=0)), Attachment.Rigid),
938             # First-person
939             (carla.Transform(carla.Location(y=-0.42, x=0.03, z=1.2), carla.Rotation(yaw=0, roll=0, pitch=-10)), Attachment.Rigid)]
940
941         self.transform_index = 1
942         self.sensors = [
943             ['sensor.camera.rgb', cc.Raw, 'Camera RGB', {}, {}],
944             ['sensor.camera.depth', cc.Raw, 'Camera Depth (Raw)', {}, {}],
945             ['sensor.camera.depth', cc.Depth, 'Camera Depth (Gray Scale)', {}, {}],
946             ['sensor.camera.depth', cc.LogarithmicDepth, 'Camera Depth (Logarithmic Gray Scale)', {}, {}],
947             ['sensor.camera.semantic_segmentation', cc.Raw, 'Camera Semantic Segmentation (Raw)', {}, {}],
948             ['sensor.camera.semantic_segmentation', cc.CityScapesPalette, 'Camera Semantic Segmentation (CityScapes Palette)', {}, {}],
949             ['sensor.camera.instance_segmentation', cc.CityScapesPalette, 'Camera Instance Segmentation (CityScapes Palette)', {}, {}],
950             ['sensor.camera.instance_segmentation', cc.Raw, 'Camera Instance Segmentation (Raw)', {}, {}],
951             ['sensor.lidar.ray_cast', None, 'Lidar (Ray-Cast)', {'range': '50'}],
952             ['sensor.camera.dvs', cc.Raw, 'Dynamic Vision Sensor', {}, {}],
953             ['sensor.camera.rgb', cc.Raw, 'Camera RGB Distorted',
954              {'lens_circle_multiplier': '3.0',
955               'lens_circle_falloff': '3.0',
956               'chromatic_aberration_intensity': '0.5',
957               'chromatic_aberration_offset': '0'}],
958             ['sensor.camera.optical_flow', cc.Raw, 'Optical Flow', {}, {}],
959         ]
960         world = self._parent.get_world()
961         bp_library = world.get_blueprint_library()
962         for item in self.sensors:
963             bp = bp_library.find(item[0])
964             if item[0].startswith('sensor.camera'):
965                 bp.set_attribute('image_size_x', str(hud.dim[0]))
966                 bp.set_attribute('image_size_y', str(hud.dim[1]))
967                 if bp.has_attribute('gamma'):
968                     bp.set_attribute('gamma', str(gamma_correction))
969                 for attr_name, attr_value in item[3].items():
970                     bp.set_attribute(attr_name, attr_value)
971             elif item[0].startswith('sensor.lidar'):
972                 self.lidar_range = 50

```

```

973         for attr_name, attr_value in item[3].items():
974             bp.set_attribute(attr_name, attr_value)
975             if attr_name == 'range':
976                 self.lidar_range = float(attr_value)
977
978     item.append(bp)
979     self.index = None
980
981
982 def toggle_camera(self):
983     self.transform_index = (self.transform_index + 1) % len(self._camera_transforms)
984     self.set_sensor(self.index, notify=False, force_respawn=True)
985
986 def set_sensor(self, index, notify=True, force_respawn=False):
987     index = index % len(self.sensors)
988     needs_respawn = True if self.index is None else \
989         (force_respawn or (self.sensors[index][2] != self.sensors[self.index][2]))
990     if needs_respawn:
991         if self.sensor is not None:
992             self.sensor.destroy()
993             self.surface = None
994         self.sensor = self._parent.get_world().spawn_actor(
995             self.sensors[index][1],
996             self._camera_transforms[self.transform_index][0],
997             attach_to=self._parent,
998             attachment_type=self._camera_transforms[self.transform_index][1])
999         # We need to pass the lambda a weak reference to self to avoid
1000         # circular reference.
1001         weak_self = weakref.ref(self)
1002         self.sensor.listen(lambda image: CameraManager._parse_image(weak_self, image))
1003     if notify:
1004         self.hud.notification(self.sensors[index][2])
1005     self.index = index
1006
1007 def next_sensor(self):
1008     self.set_sensor(self.index + 1)
1009
1010 def toggle_recording(self):
1011     self.recording = not self.recording
1012     self.hud.notification('Recording %s' % ('On' if self.recording else 'Off'))
1013
1014 def render(self, display):
1015     if self.surface is not None:
1016         display.blit(self.surface, (0, 0))
1017
1018 @staticmethod
1019 def _parse_image(weak_self, image):
1020     self = weak_self()
1021     if not self:
1022         return
1023     if self.sensors[self.index][0].startswith('sensor.lidar'):
1024         points = np.frombuffer(image.raw_data, dtype=np.dtype('f4'))
1025         points = np.reshape(points, (int(points.shape[0] / 4), 4))
1026         lidar_data = np.array(points[:, :2])
1027         lidar_data *= min(self.hud.dim) / (2.0 * self.lidar_range)
1028         lidar_data += (0.5 * self.hud.dim[0], 0.5 * self.hud.dim[1])
1029         lidar_data = np.fabs(lidar_data) # pylint: disable=E1111
1030         lidar_data = lidar_data.astype(np.int32)
1031         lidar_data = np.reshape(lidar_data, (-1, 2))
1032         lidar_img_size = (self.hud.dim[0], self.hud.dim[1], 3)
1033         lidar_img = np.zeros((lidar_img_size), dtype=np.uint8)
1034         lidar_img_tuple(lidar_data.T) = (255, 255, 255)
1035         self.surface = pygame.surfarray.make_surface(lidar_img)
1036     elif self.sensors[self.index][0].startswith('sensor.camera.dvs'):
1037         # Example of converting the raw data from a carla.DVSEventArray
1038         # sensor into a NumPy array and using it as an image
1039         dvs_events = np.frombuffer(image.raw_data, dtype=np.dtype([
1040             ('x', np.uint16), ('y', np.uint16), ('z', np.int64), ('pol', np.bool)]))
1041         dvs_img = np.zeros((image.height, image.width, 3), dtype=np.uint8)
1042         # Blue is positive, red is negative
1043         dvs_img[dvs_events[:, 'y'], dvs_events[:, 'x'], dvs_events[:, 'pol'] * 2] = 255
1044         self.surface = pygame.surfarray.make_surface(dvs_img.swapaxes(0, 1))
1045     elif self.sensors[self.index][0].startswith('sensor.camera.optical_flow'):
1046         image = image.get_color_coded_flow()
1047         array = np.frombuffer(image.raw_data, dtype=np.dtype('uint8'))
1048         array = np.reshape(array, (image.height, image.width, 4))
1049         array = array[:, :, :3]
1050         array = array[:, :, ::-1]
1051         self.surface = pygame.surfarray.make_surface(array.swapaxes(0, 1))
1052     else:
1053         image.convert(self.sensors[self.index][1])
1054         array = np.frombuffer(image.raw_data, dtype=np.dtype('uint8'))
1055         array = np.reshape(array, (image.height, image.width, 4))
1056         array = array[:, :, :3]
1057         array = array[:, :, ::-1]
1058         self.surface = pygame.surfarray.make_surface(array.swapaxes(0, 1))
1059     if self.recording:
1060         image.save_to_disk('_out/%08d' % image.frame)
1061
1062
1063 # =====
1064 -- game_loop() -----
1065 # =====
1066
1067 def game_loop(args, scenario, path_details):
1068     pygame.init()
1069     pygame.font.init()
1070     world = None
1071     original_settings = None
1072     try:
1073         client = carla.Client(args.host, args.port)
1074         client.set_timeout(20.0)
1075         sim_world = client.get_world()
1076         if args.sync:
1077             original_settings = sim_world.get_settings()
1078             settings = sim_world.get_settings()
1079             if not settings.synchronous_mode:
1080                 settings.synchronous_mode = True
1081                 settings.fixed_delta_seconds = 0.05
1082             sim_world.apply_settings(settings)
1083             traffic_manager = client.get_trafficmanager()
1084             traffic_manager.set_synchronous_mode(True)
1085
1086         if args.autopilot and not sim_world.get_settings().synchronous_mode:
1087             print("WARNING: You are currently in asynchronous mode and could "
1088                   "experience some issues with the traffic simulation")
1089
1090     display = pygame.display.set_mode(
1091         (args.width, args.height),
1092         pygame.HWSURFACE | pygame.DOUBLEBUF)
1093     display.fill((0, 0, 0))
1094     pygame.display.flip()
1095     ### MY PART ###
1096     scenario.setup(traffic_manager, sim_world)
1097     scenario.spawn(client)
1098     logger.info("The scenario was set. Spawning the hero vehicle and launching the simulation")
1099     ### ### ###
1100     client.set_timeout(10.0)
1101     hud = HUD(args.width, args.height)

```



```

1103 world = World(sim_world, hud, args, scenario, client, path_details)
1104 controller = KeyboardControl(world, args.autopilot)
1105 if args.sync:
1106     sim_world.tick()
1107 else:
1108     sim_world.wait_for_tick()
1109
1110 clock = pygame.time.Clock()
1111 while True:
1112     try:
1113         if args.sync:
1114             sim_world.tick()
1115             clock.tick_busy_loop(FPS)
1116             if controller.parse_events(client, world, clock, args.sync):
1117                 return
1118             if world.tick():
1119                 return
1120             world.render(display)
1121             pygame.display.flip()
1122         except Exception as e:
1123             logger.error(e)
1124     except Exception as e:
1125         logger.error("Something went wrong trying to launch the driver.py", e)
1126 finally:
1127     ### MY PART ###
1128     if (scenario.populated):
1129         scenario.finish(client)
1130     if (world and world.manager):
1131         logger.info('Total time simulated: {}'.format(str(world.simulation_time/1000)))
1132     ### ### ###
1133
1134     if original_settings:
1135         sim_world.apply_settings(original_settings)
1136
1137     if (world and world.recording_enabled):
1138         client.stop_recorder()
1139
1140     if world is not None:
1141         world.destroy()
1142
1143     pygame.quit()
1144
1145 # =====
1146 # -- main() -----
1147 # =====
1148
1149 argparser = argparse.ArgumentParser(
1150     description='CARLA Manual Control Client')
1151 argparser.add_argument(
1152     '-v', '--verbose',
1153     action='store_true',
1154     dest='debug',
1155     help='print debug information')
1156 argparser.add_argument(
1157     '--host',
1158     metavar='H',
1159     default='127.0.0.1',
1160     help='IP of the host server (default: 127.0.0.1)')
1161 argparser.add_argument(
1162     '-p', '--port',
1163     metavar='P',
1164     default=2000,
1165     type=int,
1166     help='TCP port to listen to (default: 2000)')
1167 argparser.add_argument(
1168     '-a', '--autopilot',
1169     action='store_true',
1170     help='enable autopilot')
1171 argparser.add_argument(
1172     '--res',
1173     metavar='WIDTHxHEIGHT',
1174     default='1280x720',
1175     help='window resolution (default: 1280x720)')
1176 argparser.add_argument(
1177     '--filter', '-f',
1178     metavar='PATTERN',
1179     default='vehicle.*',
1180     help='actor filter (default: "vehicle.*")')
1181 argparser.add_argument(
1182     '--generation',
1183     metavar='G',
1184     default='2',
1185     help='restrict to certain actor generation (values: "1","2","All" - default: "2")')
1186 argparser.add_argument(
1187     '--rolename',
1188     metavar='NAME',
1189     default='hero',
1190     help='actor role name (default: "hero")')
1191 argparser.add_argument(
1192     '--gamma',
1193     default=2.2,
1194     type=float,
1195     help='Gamma correction of the camera (default: 2.2)')
1196 argparser.add_argument(
1197     '--sync',
1198     action='store_true',
1199     help='Activate synchronous mode execution')
1200 argparser.add_argument(
1201     '--red',
1202     default="0",
1203     help='Red part of the RGB color palette to paint the hero car')
1204 argparser.add_argument(
1205     '--green',
1206     default="0",
1207     help='Green part of the RGB color palette to paint the hero car')
1208 argparser.add_argument(
1209     '--blue',
1210     default="0",
1211     help='Blue part of the RGB color palette to paint the hero car')
1212 argparser.add_argument(
1213     '-n', '--name',
1214     metavar='F',
1215     default="example",
1216     help='Participant name (name)')
1217 argparser.add_argument(
1218     '-s', '--scenario',
1219     metavar='F',
1220     default="example",
1221     help='Scenario name (name)')
1222 argparser.add_argument("--s_b64")
1223 argparser.add_argument("--p_b64")
1224 args = argparser.parse_args()
1225
1226 logger = logging.getLogger("driver-thread")
1227 logging.basicConfig(level=logging.INFO, filename="../../data/recordings/{}/session_logs.log".format(args.name), filemode="a", format="%(asctime)s - %(name)s - %
1228
1229 def main():
1230     args.width, args.height = [int(x) for x in args.res.split('x')]
1231
1232     print(__doc__)

```

```

1233
1234     try:
1235
1236         scenario_b64 = args.s_b64
1237         path_b64 = args.p_b64
1238
1239         scenario_bytes = base64.b64decode(scenario_b64.encode())
1240         path_bytes = base64.b64decode(path_b64.encode())
1241
1242         scenario = pickle.loads(scenario_bytes)
1243         path_details = pickle.loads(path_bytes)
1244         game_loop(args, scenario, path_details)
1245
1246     except KeyboardInterrupt as e:
1247         logger.warning('Keyboard interrupt while running driver.py script', e)
1248     except Exception as e:
1249         logger.error("Something went wrong.", e)
1250
1251
1252 if __name__ == '__main__':
1253
1254     main()

```

**File: ./software/carla\_scripts/helper\_scripts/print\_coordinates.py**

```

1 import glob
2 import os
3 import sys
4 import time
5 import random
6
7 from agents.navigation.behavior_agent import BehaviorAgent, BasicAgent
8
9 try:
10     sys.path.append(glob.glob('../carla/dist/carla-*%d.%d-%s.egg' % (
11         sys.version_info.major,
12         sys.version_info.minor,
13         'win-amd64' if os.name == 'nt' else 'linux-x86_64'))[0])
14 except IndexError:
15     pass
16
17 import carla
18
19 # Used to print coordinates of the spectator (the one flying in the simulator and observing the world) to the terminal
20 # Mainly used to assist in scenario designing to know exact coordinates of specific locations
21 def main():
22     client = carla.Client('localhost', 2000)
23     client.set_timeout(2.0)
24     world = client.get_world()
25     try:
26         print("Started printing spectator's coordinates.")
27         while True:
28             spectator = world.get_spectator().get_transform()
29             coordinates_string = "{}, {}, {} | x, y, z".format('%0.3f' % spectator.location.x, '%0.3f' % spectator.location.y, '%0.3f' % spectator.location.z)
30             print(coordinates_string)
31             time.sleep(1)
32     except KeyboardInterrupt:
33         print("Stopped printing coordinates.")
34     except:
35         print("Something wrong happened trying to print the spectator's coordinates.")
36
37
38
39 if __name__ == '__main__':
40     main()

```

**File: ./software/carla\_scripts/helper\_scripts/draw\_lanes\_terminal.py**

```

1 import glob
2 import os
3 import sys
4 import time
5 import random
6 import logging
7 import argparse
8 import numpy
9 import json
10 import base64
11 import pickle
12 sys.path.append("../")
13 from classes.route import Route
14 from config import *
15 from agents.navigation.global_route_planner import GlobalRoutePlanner
16
17 try:
18     sys.path.append(glob.glob('../carla/dist/carla-*%d.%d-%s.egg' % (
19         sys.version_info.major,
20         sys.version_info.minor,
21         'win-amd64' if os.name == 'nt' else 'linux-x86_64'))[0])
22 except IndexError:
23     pass
24
25 import carla
26
27 argparser = argparse.ArgumentParser(
28     description=__doc__)
29 argparser.add_argument(
30     '-p', '--path',
31     metavar='F',
32     default='path1',
33     help='Path filename (path1)')
34 args = argparser.parse_args()
35
36 # Reads file and extracts locations
37 def get_route(map):
38     waypoints = []
39     grp = GlobalRoutePlanner(map, SAMPLING_RESOLUTION)
40     locations = read_path_file()
41     for i in range(len(locations)-1):
42         waypoints.extend(grp.trace_route(locations[i], locations[i+1]))
43     return Route(waypoints)
44
45 def read_path_file():
46     file_path = "../data/paths/{}.json".format(args.path)
47     try:
48         with open(file_path) as file:
49             data = json.load(file)
50             locations = []
51             for loc in data["path_checkpoints"]:
52                 locations.append(carla.Location(x=float(loc["x"]), y=float(loc["y"]), z=float(loc["z"])))
53             return locations
54     except Exception as e:
55         print("Could not read the file {}".format(args.path))
56
57
58
59 # Used to draw the specified path on the current map
60 def main():
61     client = carla.Client("localhost", 2000)
62     client.set_timeout(10)
63     world = client.get_world()
64     map = world.get_map()
65
66     route = get_route(map)
67     try:
68         print("Started drawing {}.format(args.path))
69         finished = False
70         all_waypoints = route.get_all_waypoints()
71         start_color = carla.Color(r=0, g=0, b=255)
72         finish_color = carla.Color(r=255, g=0, b=0)
73         num_waypoints = len(all_waypoints)
74         while(not finished):
75             # Draw lanes that can be driven
76             for i, waypoint in enumerate(all_waypoints):
77                 color_progress = float(i) / (num_waypoints - 1)
78
79                 r = (finish_color.r - start_color.r) / (num_waypoints - 1)
80                 g = (finish_color.g - start_color.g) / (num_waypoints - 1)
81                 b = (finish_color.b - start_color.b) / (num_waypoints - 1)
82                 color = carla.Color()
83                 r = int(start_color.r + i * r),
84                 g = int(start_color.g + i * g),
85                 b = int(start_color.b + i * b)
86                 world.debug.draw_string(waypoint.transform.location, 'o', draw_shadow=False,color=color, life_time=10,persistent_lines=True)
87             time.sleep(9)
88     except KeyboardInterrupt:
89         print("Stopped drawing the path.")
90     except Exception as e:
91         print("Something wrong happened trying to draw the path. Aborting...", e)
92
93 if __name__ == '__main__':
94     main()

```

File: /software/carla\_scripts/helper\_scripts/save\_opendrive\_map.py

```

1 #!/usr/bin/env python
2
3 # Copyright (c) 2019 Computer Vision Center (CVC) at the Universitat Autònoma de
4 # Barcelona (UAB).
5 #
6 # This work is licensed under the terms of the MIT license.
7 # For a copy, see <https://opensource.org/licenses/MIT>.
8
9 import glob
10 import os
11 import sys
12
13 import carla
14
15 try:
16     sys.path.append(glob.glob('../carla/dist/carla-*%d.%d-%s.egg' % (
17         sys.version_info.major,
18         sys.version_info.minor,
19         'win-amd64' if os.name == 'nt' else 'linux-x86_64'))[0])
20 except IndexError:
21     pass
22
23 # Used to save the current map in the simulation to an .xml file
24 def main():
25
26     try:
27         client = carla.Client('127.0.0.1', 2000)
28         client.set_timeout(10)
29         world = client.get_world()
30         map = world.get_map()
31
32         name = map.name.rsplit('/', 1)[-1]
33         world.get_map().save_to_disk("maps/{}.xml".format(name))
34
35     except Exception as e:
36         print("Something wrong happened:", e)
37     finally:
38         print("Finished")
39
40
41 if __name__ == '__main__':
42     main()

```

**File: ./software/carla\_scripts/helper\_scripts/hero\_info.py**

```

1 import glob
2 import os
3 import sys
4 import time
5 import random
6
7 from agents.navigation.behavior_agent import BehaviorAgent, BasicAgent
8
9 try:
10     sys.path.append(glob.glob('../carla/dist/carla-*%d.%d-%s.egg' % (
11         sys.version_info.major,
12         sys.version_info.minor,
13         'win-amd64' if os.name == 'nt' else 'linux-x86_64'))[0])
14 except IndexError:
15     pass
16
17 import carla
18
19 # Used to print the bounding box of the "hero" vehicle
20 def main():
21     client = carla.Client("localhost", 2000)
22     client.set_timeout(10)
23     world = client.get_world()
24     try:
25         # Get the player
26         player = None
27         actors = world.get_actors()
28         for actor in actors:
29             if actor.attributes.get('role_name') == 'hero':
30                 player = actor
31
32         box = player.bounding_box.extent
33         print("The bounding box (x, y, z):")
34         print(box.x)
35         print(box.y)
36         print(str(box.z) + "\n")
37     except Exception as e:
38         print("Something wrong happened trying to get the bounding box of the actor named hero:", e)
39
40 if __name__ == '__main__':
41     main()

```

**File: ./software/carla\_scripts/helper\_scripts/draw\_points\_terminal.py**

```

1 import glob
2 import os
3 import sys
4 import time
5 import random
6 import logging
7 import argparse
8 import numpy
9 import json
10 import base64
11 import pickle
12 sys.path.append("..")
13 from classes.route import Route
14 from config import *
15 from agents.navigation.global_route_planner import GlobalRoutePlanner
16
17 try:
18     sys.path.append(glob.glob('../carla/dist/carla-*%d.%d-%s.egg' % (
19         sys.version_info.major,
20         sys.version_info.minor,
21         'win-amd64' if os.name == 'nt' else 'linux-x86_64'))[0])
22 except IndexError:
23     pass
24
25 import carla
26
27 argparser = argparse.ArgumentParser(
28     description=__doc__)
29 argparser.add_argument(
30     '-p', '--path',
31     metavar='F',
32     default="path1",
33     help='Path filename (path1)')
34 args = argparser.parse_args()
35
36 def read_path_file():
37     file_path = "../data/paths/{}.json".format(args.path)
38     try:
39         with open(file_path) as file:
40             data = json.load(file)
41             locations = []
42             for loc in data["path_checkpoints"]:
43                 locations.append(carla.Location(x=float(loc["x"]), y=float(loc["y"]), z=float(loc["z"])))
44             return locations
45     except Exception as e:
46         print("Could not read the file {}".format(args.path))
47
48
49 # Used to draw the specified path on the current map
50 def main():
51     client = carla.Client("localhost", 2000)
52     client.set_timeout(10)
53     world = client.get_world()
54     map = world.get_map()
55
56     locations = read_path_file()
57     try:
58         finished = False
59         while(not finished):
60
61             for l in locations[1:]:
62                 world.debug.draw_string(l, 'o000o', draw_shadow=False,color=carla.Color(r=0, g=0, b=255), life_time=10,persistent_lines=True)
63
64             world.debug.draw_string(locations[0], 'o000o', draw_shadow=False,color=carla.Color(r=255, g=0, b=0), life_time=10,persistent_lines=True)
65             time.sleep(5)
66     except KeyboardInterrupt:
67         print("Stopped drawing the path.")
68     except Exception as e:
69         print("Something wrong happened trying to draw the path. Aborting...", e)
70
71 if __name__ == '__main__':
72     main()

```

**File:** ./software/carla\_scripts/helper\_scripts/draw\_hero.py

```

1 import glob
2 import os
3 import sys
4 import time
5 import random
6 import logging
7 import argparse
8 import numpy
9 import json
10 import base64
11 import pickle
12 sys.path.append("../")
13 from classes.route import Route
14 from config import *
15 from agents.navigation.global_route_planner import GlobalRoutePlanner
16
17 try:
18     sys.path.append(glob.glob('../carla/dist/carla-*%d.%d-%s.egg' % (
19         sys.version_info.major,
20         sys.version_info.minor,
21         'win-amd64' if os.name == 'nt' else 'linux-x86_64'))[0])
22 except IndexError:
23     pass
24
25 import carla
26
27 argparser = argparse.ArgumentParser(
28     description=__doc__)
29 argparser.add_argument(
30     '-l', '--log',
31     help='Log path')
32 args = argparser.parse_args()
33
34 logger = None
35 if args.log:
36     logger = logging.getLogger("draw_hero-thread")
37     logging.basicConfig(level=logging.INFO, filename=args.log, filemode="a", format="%(asctime)s - %(name)s - %(levelname)s - %(message)s")
38
39 # Used to draw the specified path on the current map
40 def main():
41     client = carla.Client("localhost", 2000)
42     client.set_timeout(10)
43     world = client.get_world()
44
45     try:
46         message = "Begin marking the player's location in green"
47         if logger is not None: logger.info(message)
48         else: print(message)
49         # Every step
50         finished = False
51         while(not finished):
52             # Get the player
53             player = None
54             actors = world.get_actors()
55             for actor in actors:
56                 if actor.attributes.get('role_name') == 'hero':
57                     player = actor
58             if player:
59                 world.debug.draw_string(player.get_location(), 'O', draw_shadow=False,color=carla.Color(r=0, g=255, b=0), life_time=0.2,persistent_lines=True)
60             else:
61                 time.sleep(2)
62             time.sleep(0.1)
63     except KeyboardInterrupt:
64         if logger is not None: print("Stopped marking the player's location.")
65     except Exception as e:
66         message = "Something wrong happened trying to mark the player's location. Aborting..."
67         if logger is not None: logger.error(message, e)
68         else: print(message)
69     finally:
70         message = "Stopped drawing lanes."
71         if logger is not None: logger.info(message)
72         else: print(message)
73
74 if __name__ == '__main__':
75     main()

```

File: ./software/carla\_scripts/helper\_scripts/draw\_lanes.py

```

1  import glob
2  import os
3  import sys
4  import time
5  import random
6  import logging
7  import argparse
8  import numpy
9  import json
10 import base64
11 import pickle
12 sys.path.append("..")
13 from classes.route import Route
14 from config import *
15 from agents.navigation.global_route_planner import GlobalRoutePlanner
16
17 try:
18     sys.path.append(glob.glob('../carla/dist/carla-*%d.%d-%s.egg' % (
19         sys.version_info.major,
20         sys.version_info.minor,
21         'win-amd64' if os.name == 'nt' else 'linux-x86_64'))[0])
22 except IndexError:
23     pass
24
25 import carla
26
27 argparser = argparse.ArgumentParser(
28     description=__doc__)
29 argparser.add_argument(
30     '-p', '--path',
31     metavar='F',
32     default="path1",
33     help='Path filename (path1)')
34 argparser.add_argument(
35     '-l', '--log',
36     help='Log path')
37 argparser.add_argument("locations")
38 args = argparser.parse_args()
39
40 logger = None
41 if args.log:
42     logger = logging.getLogger("draw_lanes-thread")
43     logging.basicConfig(level=logging.INFO, filename=args.log, filemode="a", format="%(asctime)s - %(name)s - %(levelname)s - %(message)s")
44
45
46 def extract_locations(path_details):
47     locations = []
48     for loc in path_details:
49         locations.append(carla.Location(x=float(loc["x"]), y=float(loc["y"]), z=float(loc["z"])))
50     return locations
51
52 # Reads file and extracts locations
53 def get_route(map, details):
54     waypoints = []
55     grp = GlobalRoutePlanner(map, SAMPLING_RESOLUTION)
56     if details is not None: locations = extract_locations(details)
57     else: locations = read_path_file
58
59     for i in range(len(locations)-1):
60         waypoints.extend(grp.trace_route(locations[i], locations[i+1]))
61     return Route(waypoints)
62
63 def read_path_file():
64     file_path = "../data/paths/{}.json".format(args.path)
65     try:
66         with open(file_path) as file:
67             data = json.load(file)
68             locations = []
69             for loc in data["path_checkpoints"]:
70                 locations.append(carla.Location(x=float(loc["x"]), y=float(loc["y"]), z=float(loc["z"])))
71             return locations
72     except Exception as e:
73         print("Could not read the file {}".format(args.path))
74
75
76 # Used to draw the specified path on the current map
77 def main():
78     client = carla.Client("localhost", 2000)
79     client.set_timeout(10)
80     world = client.get_world()
81     map = world.get_map()
82
83     locations = None
84     if args.locations:
85         locs = args.locations
86         locs_decoded = base64.b64decode(locs.encode())
87         locations = pickle.loads(locs_decoded)
88     route = get_route(map, locations)
89
90     try:
91         if logger is not None: logger.info("Started drawing the path")
92         else: print("Started drawing {}".format(args.path))
93         # Every step
94         finished = False
95         while(not finished):
96             all_waypoints = route.get_all_waypoints()
97
98             # Draw lanes that can be driven
99             for waypoint in all_waypoints:
100                 world.debug.draw_string(waypoint.transform.location, 'o', draw_shadow=False,color=carla.Color(r=0, g=0, b=255), life_time=10,persistent_lines=True)
101
102             # Draw finish line
103             for waypoint in route.finish_lane_waypoints:
104                 world.debug.draw_string(waypoint.transform.location, 'o', draw_shadow=False,color=carla.Color(r=255, g=0, b=0), life_time=10,persistent_lines=True)
105             time.sleep(9)
106     except KeyboardInterrupt:
107         if logger is None: print("Stopped drawing the path.")
108     except Exception as e:
109         if logger is not None: logger.error("Something wrong happened trying to draw the path. Aborting...", e)
110         else: print("Something wrong happened trying to draw the path. Aborting...", e)
111     finally:
112         if logger is not None: logger.info("Stopped drawing lanes.")
113
114 if __name__ == '__main__':
115     main()

```

File: ./software/python\_libraries/requirements.txt

```

antlr4-python3-runtime==4.9.3
appnope==0.1.3
asgiref==3.5.2
asttokens==2.2.1
backcall==0.2.0
certifi==2022.12.7
charset-normalizer==3.0.1
click==8.1.3

```

commonroad-all==0.0.1  
commonroad-drivability-checker==2022.2.1  
commonroad-io==2022.3  
commonroad-route-planner==2022.3  
commonroad-scenario-designer==0.6.0  
commonroad-vehicle-models==3.0.2  
contourpy==1.0.7  
cyclery==0.11.0  
decorator==5.1.1  
Django==3.2  
docker==6.0.1  
enum34==1.1.10  
executing==1.2.0  
ffmpeg==1.4  
fonttools==4.38.0  
grpcio==1.51.1  
idna==3.4  
ipython==8.9.0  
iso3166==2.1.1  
jedi==0.18.2  
joblib==1.2.0  
kiwisolver==1.4.4  
libsumo==1.15.0  
lxml==4.9.2  
matplotlib==3.6.3  
matplotlib-inline==0.1.6  
mercantile==1.2.1  
networkx==3.0  
numpy==1.24.1  
omegaconf==2.3.0  
opendrive2lanelet==1.2.1  
ordered-set==4.1.0  
packaging==23.0  
parso==0.8.3  
pexpect==4.8.0  
pickleshare==0.7.5  
Pillow==9.4.0  
prompt-toolkit==3.0.36  
protobuf==3.20.1  
ptyprocess==0.7.0  
pure-eval==0.2.2  
Pygments==2.14.0  
pyparsing==3.0.9  
pyproj==3.4.1  
PyQt5==5.15.8  
PyQt5-Qt5==5.15.2  
PyQt5-sip==12.11.1  
python-dateutil==2.8.2  
pytz==2022.5  
PyYAML==6.0  
requests==2.28.2  
Rtree==1.0.1  
scikit-learn==1.2.0  
scipy==1.10.0  
Shapely==1.8.5  
six==1.16.0  
sqlparse==0.4.3  
stack-data==0.6.2  
sumocr==2023.1  
sumolib==1.15.0  
threadpoolctl==3.1.0  
tqdm==4.64.1  
traci==1.15.0  
traitlets==5.9.0  
urllib3==1.26.14  
utm==0.7.0  
wcwidth==0.2.6  
websocket-client==1.5.0

**File: ./software/python\_libraries/README.md**

```
1 ## Introduction
2 This directory contains the `crdesigner` python library and the requirements.txt file containing a list of python libraries needed to work with this framework.
3
4 `crdesigner` is a modified version of the `crdesigner` library (version 0.6.0). This version was the most recent release available at the time of the research.
5 More about crdesigner can be found [here](https://gitlab.lrz.de/tum-cps/commonroad-scenario-designer).
6
7 ## Modification
8 The original `commonroad-scenario-designer` library was modified because it could not read input map files correctly and could not draw a list of specific points c
9
10 ## Requirements
11 The `requirements.txt` file lists the dependencies required to use this project.
12 These dependencies can be installed by creating a virtual environment at the root of this project and running the command
13 ``pip install -r requirements.txt``
```

**File: ./software/README.md**



```

1  # Software directory
2
3  This directory is part of the CS Bachelor's thesis on the topic "Comparing human and AI behavior in simulated driving scenarios". It contains all the software impl
4
5  ## Technicalities
6
7  The software is working with [CARLA] (https://carla.org) driving simulator.
8
9  - CARLA Version 0.9.13
10 - CARLA API Version 0.9.13
11 - OS: Ubuntu 18.04 and 20.04
12 - Python 3.6.9
13
14 All the required python libraries are listed [here] (./python\_libraries/requirements.txt).
15
16 ## Structure
17
18 The following is the structure of this software bundle. More details can be found in the respective directories.
19
20 - [CARLA scripts] (./carla\_scripts/) - contains scripts for scenario processing and simulations.
21 - [Data analytics tools] (./data\_analysis/) - contains tools for parsing and analysing recorded data.
22 - [Python libraries] (./python\_libraries/) - required libraries for the software.
23 - [Scenario generation tool] (./generating\_scenarios/) - machine learning algorithm able to estimate the parameters of new scenarios.
24 - [Testing] (./testing/) - directory containing various tests of software components.

```

#### File: ./software/testing/README.md

```

1 The approach to testing...

```

#### File: ./software/generating\_scenarios/README.md

```

1  ## General information
2
3  This directory contains scripts needed to generate new driving scenarios.
4
5  The data used in learning is stored in the [learning_data] (../../data/scenario\_generation\_data/learning\_data/) directory.
6
7  Newly generated scenarios go to the [generated_scenarios] (../../data/scenario\_generation\_data/generated\_scenarios/) directory.
8
9  The `regressor` file contains the algorithm used to train the model and estimate new scenarios.
10
11 To create a new scenario, run the `create_scenario.py` file giving flags `-d` specifying the difficulty of the scenario, `-s` specifying the sun's altitude, `-w` s
12
13 Example: `python create_scenario.py -d 200 -w 20 -s 30 -f new_scenario`

```

#### File: ./software/generating\_scenarios/regressor.py

```

1  import pandas as pd
2  import numpy as np
3  from sklearn.multioutput import MultiOutputRegressor
4  from sklearn.ensemble import RandomForestRegressor
5  from sklearn.model_selection import train_test_split
6  import json
7
8  SCENARIO_GENERATION_DATA_PATH = "../../data/scenario_generation_data"
9
10 # This class is responsible for initialising the MultiOutputRegressor algorithm which
11 # is trained on the data contained in scenarios.csv file and is able to estimate new
12 # scenario parameters given difficulty
13 class Regressor():
14
15     def __init__(self, randomise):
16         # First read the .csv file containing scenario entries
17         scenarios_dataframe = pd.read_csv("{}learning_data/scenarios.csv".format(SCENARIO_GENERATION_DATA_PATH), delimiter = ",")
18         random_state = 10 if randomise else None
19         # Create a new dataset with only the difficulty as a feature and the rest of the attributes as targets
20         X = scenarios_dataframe[["difficulty", "wetness", "sun_altitude_angle"]]
21         y = scenarios_dataframe.drop(["difficulty", "wetness", "sun_altitude_angle"], axis=1)
22         self.attribute_names = y.columns
23         # Split the data into training and testing sets using the standard 80-20 approach
24         self.X_train, self.X_test, self.y_train, self.y_test = train_test_split(X, y, test_size=0.1, random_state=random_state)
25         # Train the multi-target regression model
26         self.model = MultiOutputRegressor(RandomForestRegressor(n_estimators=100, random_state=random_state))
27         self.model.fit(self.X_train.values, self.y_train)
28
29     # This method will create a new scenario given the difficulty and then save it to the file
30     def create_new_scenario(self, difficulty, wetness, sun, filename):
31         attributes = self.predict_scenario_attributes(difficulty, wetness, sun)
32         self.write_scenario_to_file(attributes, filename)
33
34     # Given the difficulty, predict what attributes a scenario could have
35     def predict_scenario_attributes(self, difficulty, wetness, sun):
36         arr = np.array([difficulty, wetness, sun]).reshape(1, -1)
37         attributes = self.model.predict(arr)[0]
38         rounding = pd.read_csv("{}learning_data/rounding.csv".format(SCENARIO_GENERATION_DATA_PATH))
39         maximums = pd.read_csv("{}learning_data/maximums.csv".format(SCENARIO_GENERATION_DATA_PATH))
40         minimums = pd.read_csv("{}learning_data/minimums.csv".format(SCENARIO_GENERATION_DATA_PATH))
41         rounded_attributes = []
42         # For each predicted value, check if it is not exceeding the maximum allowed value and is not lower than the mimum allowed value
43         # Also round the final value accordingly
44         list_of_attribute_names = self.attribute_names.tolist()
45         for i in range(len(attributes)):
46             value = self.format_attribute_value(value = float(attributes[i]), minimum = float(minimums.loc[0, list_of_attribute_names[i]]), maximum = maximums.loc[
47             rounded_attributes.append(value)
48
49         # Also add difficulty, wetness and sun altitude angle to the final scenario file
50         dictionary = dict(zip(self.attribute_names, rounded_attributes))
51         dictionary["difficulty"] = self.format_attribute_value(value = float(difficulty), minimum = float(minimums.loc[0, "difficulty"]), maximum = maximums.loc[0,
52         dictionary["wetness"] = self.format_attribute_value(value = float(wetness), minimum = float(minimums.loc[0, "wetness"]), maximum = maximums.loc[0, "wetness
53         dictionary["sun_altitude_angle"] = self.format_attribute_value(value = float(sun), minimum = float(minimums.loc[0, "sun_altitude_angle"]), maximum = maximu
54         return dictionary
55
56     def format_attribute_value(self, value, minimum, maximum, roundn):
57         if value < minimum:
58             value = minimum
59         if maximum == "inf":
60             pass
61         else:
62             if value > float(maximum):
63                 value = float(maximum)
64             value = int(np.round(value)) if roundn == 0 else np.round(value, roundn)
65         return value
66
67     # Create a .json file and save the attribute values there
68     def write_scenario_to_file(self, attributes, filename):
69         with open("{}generated_scenarios/{}.json".format(SCENARIO_GENERATION_DATA_PATH, filename), "w+") as f:
70             json.dump(attributes, f, indent=4)

```

**File: ./software/generating\_scenarios/create\_scenario.py**

```
1 from regressor import Regressor
2 import argparse
3
4 # The purpose of this script is to create new scenario.json file according to the difficulty, wetness and sun altitude angle
5 # and save it in the file specified in the arguments
6 def main():
7     parser = argparse.ArgumentParser(description='Create a new scenario based on difficulty.')
8     parser.add_argument('--difficulty', '-d', type=float, default='70', help='Difficulty value')
9     parser.add_argument('--sun', '-s', type=float, default='68', help='Sun altitude angle')
10    parser.add_argument('--wetness', '-w', type=float, default='10', help='Wetness value')
11    parser.add_argument('--filename', '-f', type=str, default='example', help='Filename to save the scenario')
12    parser.add_argument('--random', '-r', action='store_false', help='Everytime build and train the algorithm differently')
13    args = parser.parse_args()
14
15    try:
16        regressor = Regressor(args.random)
17        regressor.create_new_scenario(args.difficulty, args.wetness, args.sun, args.filename)
18        print("New scenario was created and save into {}.json file.".format(args.filename))
19    except Exception as e:
20        print("An error occurred trying to create a new scenario:", str(e))
21
22 if __name__ == '__main__':
23     main()
```

**File: ./README.md**

```
1 # Testing vehicle safety in simulated driving scenarios
2
3 This repository holds data and code related to the CS Bachelor's thesis.
4
5 - [Forms](./forms/)
6 - [Software bundle](./software)
7 - [Data storage](./data)
8
9 ---
10
11 I verify that I am the sole author of this project, except where explicitly stated to the contrary.
12
13 **Author:**
14
15 > Vakariss Paulavicius
16
17 **Date:**
18
19 > 2023-04-03
```

**File: ./gitignore**

```
# Created by https://www.toptal.com/developers/gitignore/api/windows,linux,macos,intellij,pycharm,visualstudiocode,python
# Edit at https://www.toptal.com/developers/gitignore?templates=windows,linux,macos,intellij,pycharm,visualstudiocode,python
```

```
### IntelliJ ###
# Covers JetBrains IDEs: IntelliJ, RubyMine, PhpStorm, AppCode, PyCharm, CLion, Android Studio, WebStorm and Rider
# Reference: https://intellij-support.jetbrains.com/hc/en-us/articles/206544839
```

```
# User-specific stuff
.idea/**/workspace.xml
.idea/**/tasks.xml
.idea/**/usage.statistics.xml
.idea/**/dictionaries
.idea/**/shelf
```

```
# AWS User-specific
.idea/**/aws.xml
```

```
# Generated files
.idea/**/contentModel.xml
```

```
# Sensitive or high-churn files
.idea/**/dataSources/
.idea/**/dataSources.ids
.idea/**/dataSources.local.xml
.idea/**/sqlDataSources.xml
.idea/**/dynamic.xml
.idea/**/uiDesigner.xml
.idea/**/dbnavigator.xml
```

```
# Gradle
.idea/**/gradle.xml
.idea/**/libraries
```

```
# Gradle and Maven with auto-import
# When using Gradle or Maven with auto-import, you should exclude module files,
# since they will be recreated, and may cause churn. Uncomment if using
# auto-import.
# .idea/artifacts
# .idea/compiler.xml
# .idea/jarRepositories.xml
# .idea/modules.xml
# .idea/*.iml
# .idea/modules
# *.iml
# *.ipr
```

```
# CMake
cmake-build-*/
```

```
# Mongo Explorer plugin
.idea/**/mongoSettings.xml
```

```
# File-based project format
*.iws
```

```
# IntelliJ
out/
```

```
# mpeltonen/sbt-idea plugin
.idea_modules/
```

```

# JIRA plugin
atlassian-ide-plugin.xml

# Cursive Clojure plugin
.idea/replstate.xml

# SonarLint plugin
.idea/sonarlint/

# Crashlytics plugin (for Android Studio and IntelliJ)
com_crashlytics_export_strings.xml
crashlytics.properties
crashlytics-build.properties
fabric.properties

# Editor-based Rest Client
.idea/httpRequests

# Android studio 3.1+ serialized cache file
.idea/caches/build_file_checksums.ser

### IntelliJ Patch ###
# Comment Reason: https://github.com/joelblau/gitignore.io/issues/186#issuecomment-215987721

# * .iml
# modules.xml
# .idea/misc.xml
# * .ipr

# Sonarlint plugin
# https://plugins.jetbrains.com/plugin/7973-sonarlint
.idea/**/sonarlint/

# SonarQube Plugin
# https://plugins.jetbrains.com/plugin/7238-sonarqube-community-plugin
.idea/**/sonarIssues.xml

# Markdown Navigator plugin
# https://plugins.jetbrains.com/plugin/7896-markdown-navigator-enhanced
.idea/**/markdown-navigator.xml
.idea/**/markdown-navigator-enh.xml
.idea/**/markdown-navigator/

# Cache file creation bug
# See https://youtrack.jetbrains.com/issue/JBR-2257
.idea/SCACHE_FILES

# CodeStream plugin
# https://plugins.jetbrains.com/plugin/12206-codestream
.idea/codestream.xml

# Azure Toolkit for IntelliJ plugin
# https://plugins.jetbrains.com/plugin/8053-azure-toolkit-for-intellij
.idea/**/azureSettings.xml

### Linux ###
*~

# temporary files which can be created if a process still has a handle open of a deleted file
.fuse_hidden*

# KDE directory preferences
.directory

# Linux trash folder which might appear on any partition or disk
.Trash-*

# .nfs files are created when an open file is removed but is still being accessed
.nfs*

### macOS ###
# General
.DS_Store
.AppleDouble
.LSOverride

# Icon must end with two \r
Icon

# Thumbnails
.*

# Files that might appear in the root of a volume
.DocumentRevisions-V100
.fsevents
.Spotlight-V100
.TemporaryItems
.Trashes
.VolumeIcon.icns
.com.apple.timemachine.donotpresent

# Directories potentially created on remote AFP share
.AppleDB
.AppleDesktop
.Network Trash Folder
.Temporary Items
.apdisk

### macOS Patch ###
# iCloud generated files
*.icloud

### PyCharm ###
# Covers JetBrains IDEs: IntelliJ, RubyMine, PhpStorm, AppCode, PyCharm, CLion, Android Studio, WebStorm and Rider
# Reference: https://intellij-support.jetbrains.com/hc/en-us/articles/206544839

# User-specific stuff

```

```

# AWS User-specific

# Generated files

# Sensitive or high-churn files

# Gradle

# Gradle and Maven with auto-import
# When using Gradle or Maven with auto-import, you should exclude module files,
# since they will be recreated, and may cause churn. Uncomment if using
# auto-import.
# .idea/artifacts
# .idea/compiler.xml
# .idea/jarRepositories.xml
# .idea/modules.xml
# .idea/*.iml
# .idea/modules
# * .iml
# * .ipr

# CMake

# Mongo Explorer plugin

# File-based project format

# IntelliJ

# mpeltonen/sbt-idea plugin

# JIRA plugin

# Cursive Clojure plugin

# SonarLint plugin

# Crashlytics plugin (for Android Studio and IntelliJ)

# Editor-based Rest Client

# Android studio 3.1+ serialized cache file

#### PyCharm Patch ####
# Comment Reason: https://github.com/joelblau/gitignore.io/issues/186#issuecomment-215987721

# * .iml
# modules.xml
# .idea/misc.xml
# * .ipr

# Sonarlint plugin
# https://plugins.jetbrains.com/plugin/7973-sonarlint

# SonarQube Plugin
# https://plugins.jetbrains.com/plugin/7238-sonarqube-community-plugin

# Markdown Navigator plugin
# https://plugins.jetbrains.com/plugin/7896-markdown-navigator-enhanced

# Cache file creation bug
# See https://youtrack.jetbrains.com/issue/JBR-2257

# CodeStream plugin
# https://plugins.jetbrains.com/plugin/12206-codestream

# Azure Toolkit for IntelliJ plugin
# https://plugins.jetbrains.com/plugin/8053-azure-toolkit-for-intellij

#### Python ####
# Byte-compiled / optimized / DLL files
__pycache__/
*.py[cod]
*$py.class

# C extensions
*.so

# Distribution / packaging
.Python
build/
develop-eggs/
dist/
downloads/
eggs/
.eggs/
lib/
lib64/
parts/
sdist/
var/
wheels/
share/python-wheels/
*.egg-info/
.installed.cfg
*.egg
MANIFEST

# PyInstaller
# Usually these files are written by a python script from a template
# before PyInstaller builds the exe, so as to inject date/other infos into it.
*.manifest
*.spec

# Installer logs
pip-log.txt
pip-delete-this-directory.txt

```

```

# Unit test / coverage reports
htmlcov/
.tox/
.nox/
.coverage
.coverage.*
.cache
nosetests.xml
coverage.xml
*.cover
*.py.cover
.hypothesis/
.pytest_cache/
cover/

# Translations
*.mo
*.pot

# Django stuff:
*.log
local_settings.py
db.sqlite3
db.sqlite3-journal

# Flask stuff:
instance/
.webassets-cache

# Scrappy stuff:
.scrappy

# Sphinx documentation
docs/_build/

# PyBuilder
.pybuilder/
target/

# Jupyter Notebook
.ipynb_checkpoints

# IPython
profile_default/
ipython_config.py

# pyenv
# For a library or package, you might want to ignore these files since the code is
# intended to run in multiple environments; otherwise, check them in:
# .python-version

# pipenv
# According to pypa/pipenv#598, it is recommended to include Pipfile.lock in version control.
# However, in case of collaboration, if having platform-specific dependencies or dependencies
# having no cross-platform support, pipenv may install dependencies that don't work, or not
# install all needed dependencies.
#Pipfile.lock

# poetry
# Similar to Pipfile.lock, it is generally recommended to include poetry.lock in version control.
# This is especially recommended for binary packages to ensure reproducibility, and is more
# commonly ignored for libraries.
# https://python-poetry.org/docs/basic-usage/#commit-your-poetrylock-file-to-version-control
#poetry.lock

# pdm
# Similar to Pipfile.lock, it is generally recommended to include pdm.lock in version control.
#pdm.lock
# pdm stores project-wide configurations in .pdm.toml, but it is recommended to not include it
# in version control.
# https://pdm.fming.dev/#use-with-ide
.pdm.toml

# PEP 582; used by e.g. github.com/David-OConnor/pyflow and github.com/pdm-project/pdm
__pypackages__/_

# Celery stuff
celerybeat-schedule
celerybeat.pid

# SageMath parsed files
*.sage.py

# Environments
.env
.venv
env/
venv/
ENV/
env.bak/
venv.bak/

# Spyder project settings
.spyderproject
.spyproject

# Rope project settings
.ropeproject

# mkdocs documentation
/site

# mypy
.mypy_cache/
.dmypy.json
dmypy.json

```

```

# Pyre type checker
.pyre/

# pytype static type analyzer
.pytype/

# Cython debug symbols
cython_debug/

# PyCharm
# JetBrains specific template is maintained in a separate JetBrains.gitignore that can
# be found at https://github.com/github/gitignore/blob/main/Global/JetBrains.gitignore
# and can be added to the global gitignore or merged into this file. For a more nuclear
# option (not recommended) you can uncomment the following to ignore the entire idea folder.
.idea/

### Python Patch ###
# Poetry local configuration file - https://python-poetry.org/docs/configuration/#local-configuration
poetry.toml

# ruff
.ruff_cache/

# LSP config files
pyrightconfig.json

### VisualStudioCode ###
.vscode/*
!.vscode/settings.json
!.vscode/tasks.json
!.vscode/launch.json
!.vscode/extensions.json
!.vscode/*.code-snippets

# Local History for Visual Studio Code
.history/

# Built Visual Studio Code Extensions
*.vsix

### VisualStudioCode Patch ###
# Ignore all local history of files
.history
.ionide

### Windows ###
# Windows thumbnail cache files
Thumbs.db
Thumbs.db:encryptable
ehthumbs.db
ehthumbs_vista.db

# Dump file
*.stackdump

# Folder config file
[Dd]esktop.ini

# Recycle Bin used on file shares
$RECYCLE.BIN/

# Windows Installer files
*.cab
*.msi
*.msix
*.msm
*.msp

# Windows shortcuts
*.lnk

# End of https://www.toptal.com/developers/gitignore/api/windows,linux,macos,intellij,pycharm,visualstudiocode,python

```

**File: ./code2pdf**

```

:directories:
- .git
- data
- forms
- software/python_libraries/crdesigner

:files:
- .DS_Store
- __init__.py
- analysis_parameters.json
- scenario_list.json

```