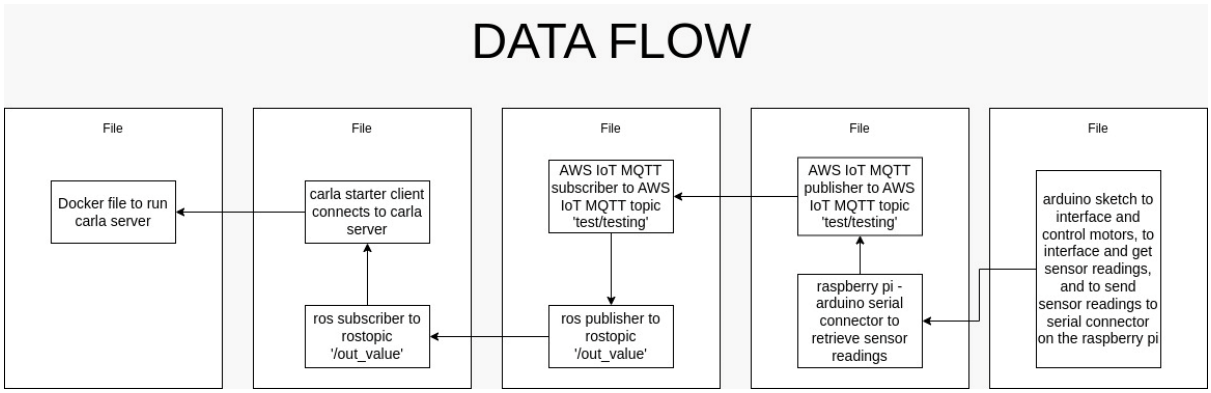
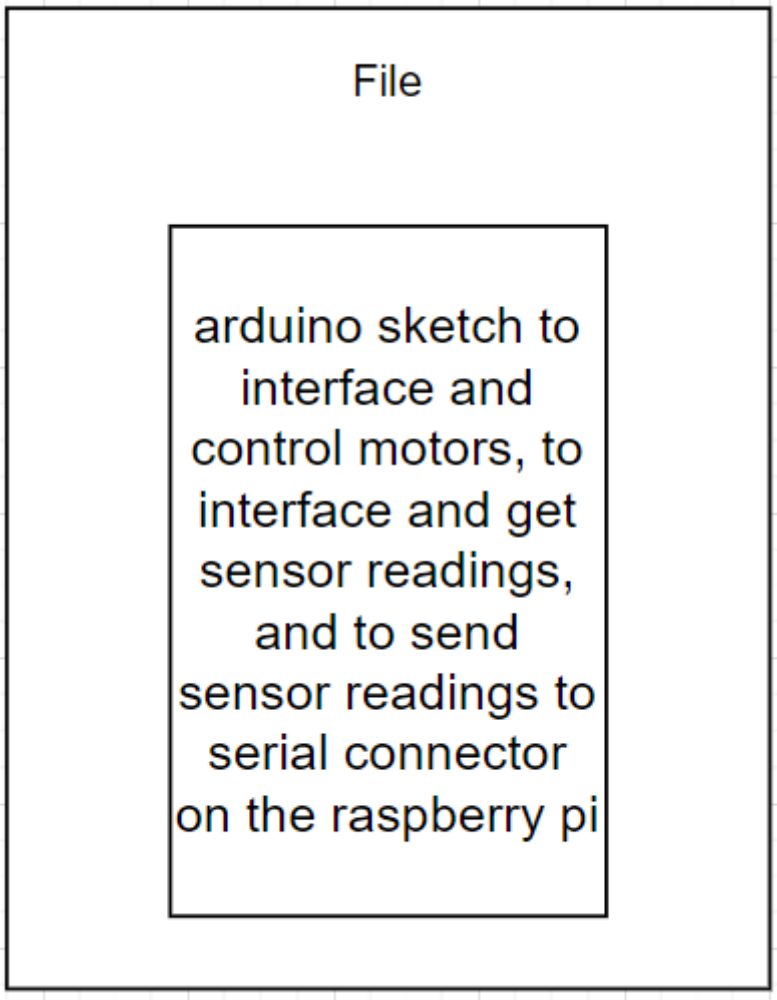


1.



(file)

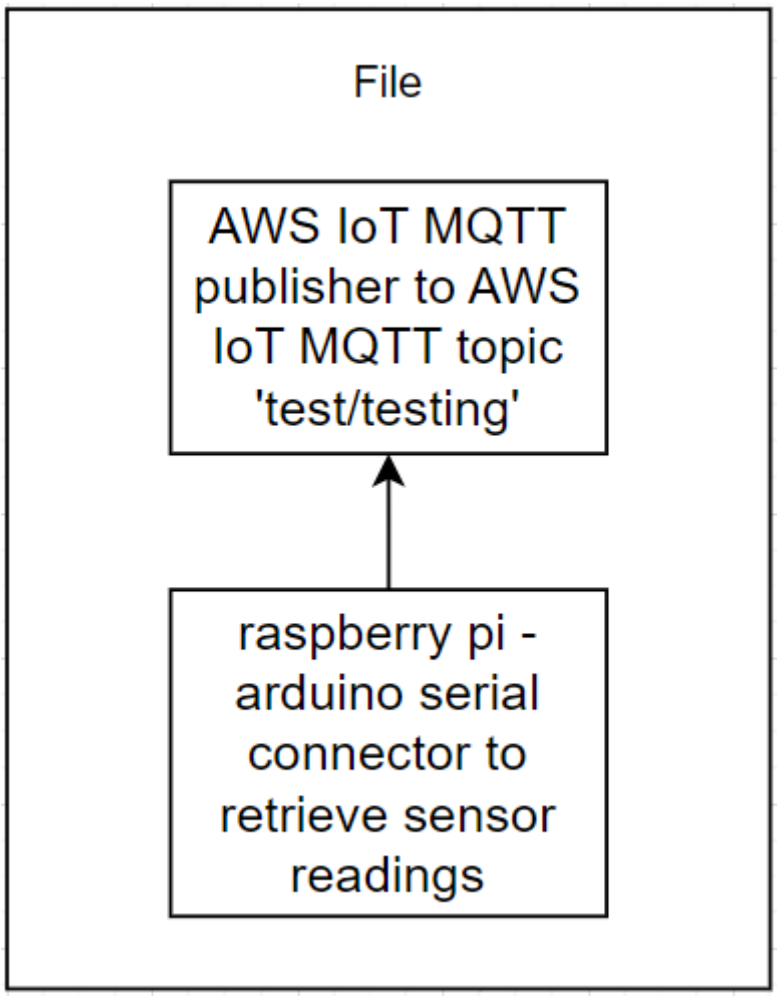
```
#include <AFMotor.h> // Motor Controller Library


AF_DCMotor motor1(1);
AF_DCMotor motor2(2);
```

```
AF_DCMotor motor3(3);
AF_DCMotor motor4(4);

#define echoPin 16 // attach pin D2 Arduino to pin Echo of HC-SR04
#define trigPin 17 //attach pin D3 Arduino to pin Trig of HC-SR04


int distance;
int min_distance;
int x= 0;
long t1;
long t2;
long duration; // variable for the duration of sound wave travel
int distanceU; // variable for the distance measurement


int ultrasonic()
{
  // Clears the trigPin condition
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  // Sets the trigPin HIGH (ACTIVE) for 10 microseconds
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  // Reads the echoPin, returns the sound wave travel time in microseconds
  duration = pulseIn(echoPin, HIGH);
  // Calculating the distance
  distanceU = duration * 0.034 / 2; // Speed of sound wave divided by 2 (go
and back)
  // Displays the distance on the Serial Monitor
  //Serial.print("Distance: ");
  //Serial.print(distanceU);
  //Serial.println(" cm");

  return distanceU;

}



void moveForward() {
  motor1.setSpeed(200);
  motor2.setSpeed(200);
  motor3.setSpeed(200);
  motor4.setSpeed(200);
  motor1.run(FORWARD);
  motor2.run(FORWARD);
  motor3.run(FORWARD);
```

```
  motor4.run(FORWARD);
}
void moveBackward() {
  motor1.setSpeed(200);
  motor2.setSpeed(200);
  motor3.setSpeed(200);
  motor4.setSpeed(200);
  motor1.run(BACKWARD);
  motor2.run(BACKWARD);
  motor3.run(BACKWARD);
  motor4.run(BACKWARD);
}

void Stop() {
  motor1.run(RELEASE);
  motor2.run(RELEASE);
  motor3.run(RELEASE);
  motor4.run(RELEASE);
}

void setup() {
  Serial.begin(9600);
  min_distance = 5;

  // Ultrasonic setup
  pinMode(trigPin, OUTPUT); // Sets the trigPin as an OUTPUT
  pinMode(echoPin, INPUT); // Sets the echoPin as an INPUT
  Serial.begin(9600); // // Serial Communication is starting with 9600 of
baudrate speed
  //Serial.println("Ultrasonic Sensor HC-SR04 Test"); // print some text in
Serial Monitor
  //Serial.println("with Arduino UNO R3");

}

void loop() {
//t1 = millis();
distance = ultrasonic();


  if (distance <= min_distance)
  {
    Stop();

    Serial.println(distance);
    //Serial.write('\n');
    //Serial.write("stop");
    //Serial.write('\n');
    //delay ();
```

```
  }
  else
  {
    moveForward();
    Serial.println(distance);
    //Serial.write('\n');
    //Serial.write("moving");
    //Serial.write('\n');
    //delay (3000);

  }
delay(1500);
//t2 = millis();


}
```

2.

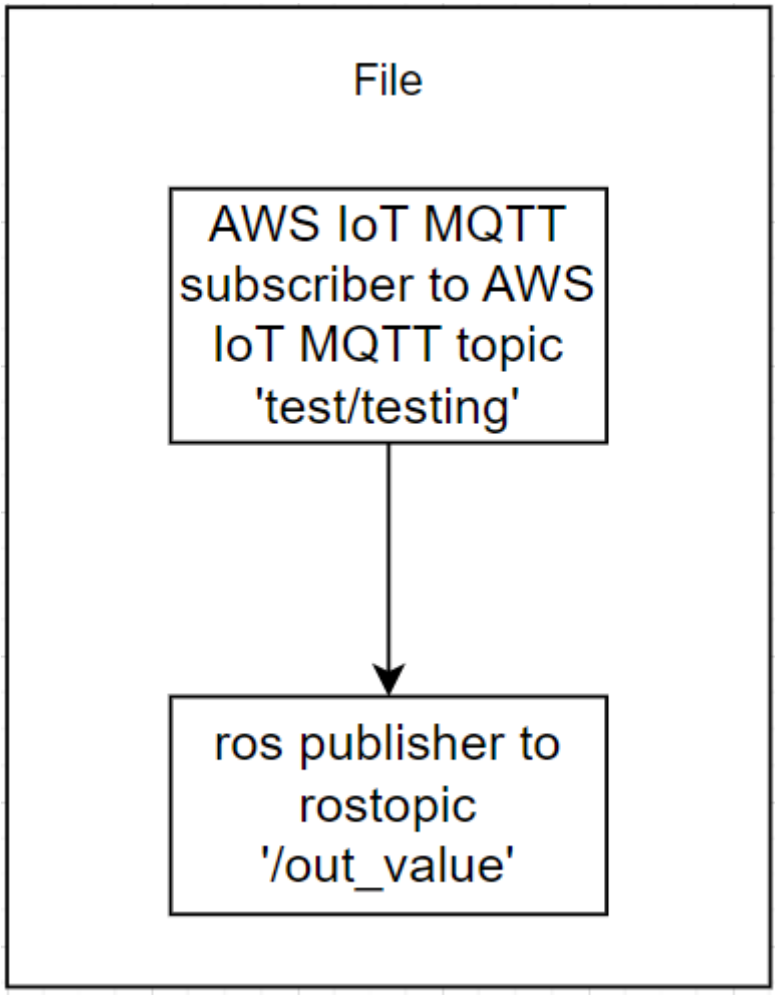


(file)

```
#!/usr/bin/env python3
import serial
import json
import time
import AWSIoTPythonSDK.MQTTLib as AWSIoTPyMQTT


# Define ENDPOINT, CLIENT_ID, PATH_TO_CERTIFICATE, PATH_TO_PRIVATE_KEY,
PATH_TO_AMAZON_ROOT_CA_1, MESSAGE, TOPIC, and RANGE
ENDPOINT = "avsnsxi1w2nv8-ats.iot.me-central-1.amazonaws.com"
CLIENT_ID = "iot_thing"
PATH_TO_CERTIFICATE =
"d42194e6a78b821823451cc7b1d29896291e00dffd00576f6197250175f33b2b-
certificate.pem.crt"
```

```
PATH_TO_PRIVATE_KEY =
"d42194e6a78b821823451cc7b1d29896291e00dffd00576f6197250175f33b2b-
private.pem.key"
PATH_TO_AMAZON_ROOT_CA_1 = "AmazonRootCA1.pem"
TOPIC = "test/testing"

myAWSIoTMQTTClient = AWSIoTPyMQTT.AWSIoTMQTTClient(CLIENT_ID)
myAWSIoTMQTTClient.configureEndpoint(ENDPOINT, 8883)
myAWSIoTMQTTClient.configureCredentials(PATH_TO_AMAZON_ROOT_CA_1,
PATH_TO_PRIVATE_KEY, PATH_TO_CERTIFICATE)
myAWSIoTMQTTClient.configureMQTTOperationTimeout(1000)
myAWSIoTMQTTClient.connect()

ser = serial.Serial('/dev/ttyACM0', 9600, timeout=2)
ser.reset_input_buffer()


def sr():
    reading = ser.readline().decode()
    reading = reading.strip()
    distance = int(reading) if len(reading) > 0 else 0
    return distance


def publish_to_cloud(msg):
    myAWSIoTMQTTClient.publish(TOPIC, json.dumps(msg), 1)


if __name__ == '__main__':
    counter = 0
    while True:
        distance = sr()
        publish_to_cloud(distance)
        #time.sleep(1)
        print(distance)
```

3.



(file – part1)

```
#!/usr/bin/env python3.7

import time
import sys
sys.path.append("/home/m/catkin_ws/src/pro/scripts/PythonAPI/iot")
import rospy
from std_msgs.msg import Float32
from subscribe import mqttc


class Echo(object):
    def __init__(self):
        rospy.init_node('echoer')
        self.pub = rospy.Publisher('/out_value', Float32, latch=True,
queue_size=10)
        rospy.Subscriber('/out_value', Float32)
        self.distance = 0.2
```

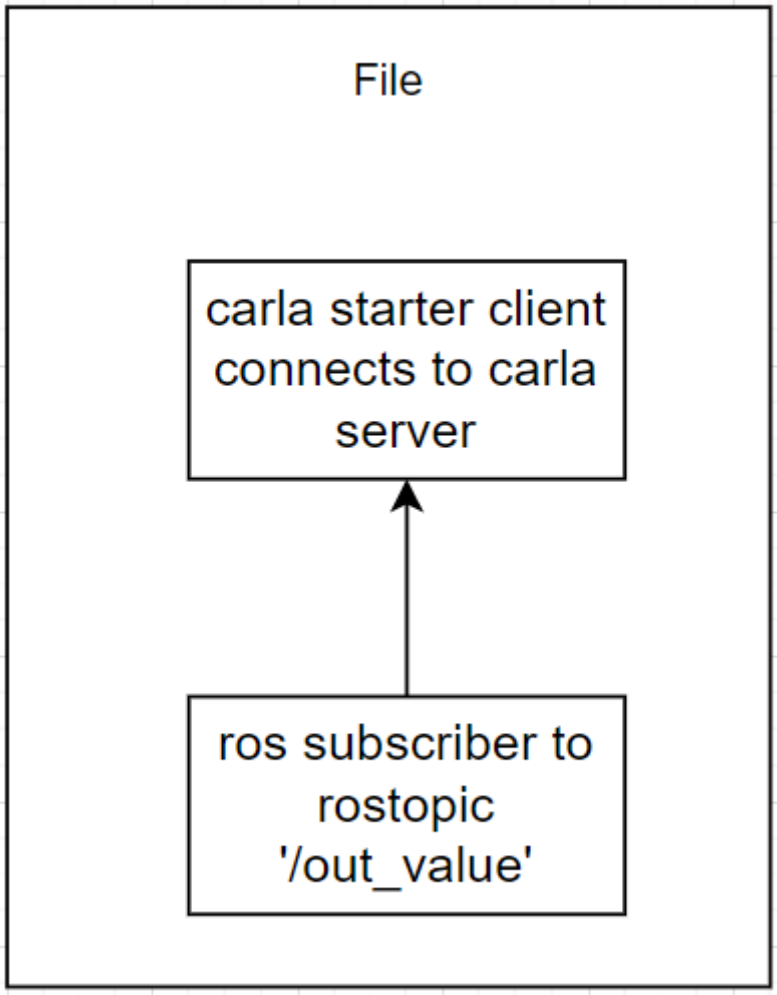
```
    def on_message(self, client, userdata, msg):  # Func for receiving msgs
        # print("topic: " + msg.topic)
        # print("payload: " + str(msg.payload))
        self.distance = int(msg.payload.decode("utf-8"))
        print(msg.payload, self.distance)

    def run(self):
        mqttc.loop_start()
        while not rospy.is_shutdown():
            self.pub.publish(self.distance)


if __name__ == '__main__':
    echo_obj = Echo()
    mqttc.on_message = echo_obj.on_message  # assign on_message func
    time.sleep(0.1)
    echo_obj.run()
```

(file – part2)

```
#!/usr/bin/env python3.7
import ssl
import paho.mqtt.client as paho


awshost = "avsnsxi1w2nv8-ats.iot.me-central-1.amazonaws.com"  # Endpoint
awsport = 8883  # Port no.
clientId = "iot_thing"  # Thing_Name
thingName = "iot_thing"  # Thing_Name
caPath = "AmazonRootCA1.pem"  # Root_CA_Certificate_Name
certPath = "d42194e6a78b821823451cc7b1d29896291e00dffd00576f6197250175f33b2b-
certificate.pem.crt"  # <Thing_Name>.cert.pem
keyPath = "d42194e6a78b821823451cc7b1d29896291e00dffd00576f6197250175f33b2b-
private.pem.key"  # <Thing_Name>.private.key
TOPIC = "test/testing"


def on_connect(client, userdata, flags, rc):  # func for making connection
    print("Connection returned result: " + str(rc))
    # Subscribing in on_connect() means that if we lose the connection and
    # reconnect then subscriptions will be renewed.
    client.subscribe(TOPIC, 1)  # Subscribe to all topics


# def on_log(client, userdata, level, msg):
#    print(msg.topic+" "+str(msg.payload))

mqttc = paho.Client()  # mqttc object
```

```
mqttc.on_connect = on_connect  # assign on_connect func
# mqttc.on_log = on_log

mqttc.tls_set(caPath, certfile=certPath, keyfile=keyPath,
cert_reqs=ssl.CERT_REQUIRED, tls_version=ssl.PROTOCOL_TLSv1_2, ciphers=None)
 # pass parameters
mqttc.connect(awshost, awsport, keepalive=60)  # connect to aws server
#mqttc.loop_start()
```

4.



(file)

```
#!/usr/bin/env python3.7

# Copyright (c) 2018 Intel Labs.
# authors: German Ros (german.ros@intel.com)
#
# This work is licensed under the terms of the MIT license.
```

```
# For a copy, see <https://opensource.org/licenses/MIT>.

"""Example of automatic vehicle control from client side."""

from __future__ import print_function

import argparse
import collections
import datetime
import glob
import logging
import math
import os
import numpy.random as random
import re
import sys
import weakref
import rospy
from std_msgs.msg import Float32

try:
    import pygame
    from pygame.locals import KMOD_CTRL
    from pygame.locals import K_ESCAPE
    from pygame.locals import K_q
except ImportError:
    raise RuntimeError('cannot import pygame, make sure pygame package is
installed')

try:
    import numpy as np
except ImportError:
    raise RuntimeError(
        'cannot import numpy, make sure numpy package is installed')

#
==============================================================================
# -- Find CARLA module ---------------------------------------------------
--
#
==============================================================================
try:
    sys.path.append(glob.glob('../carla/dist/carla-*%d.%d-%s.egg' % (
        sys.version_info.major,
        sys.version_info.minor,
        'win-amd64' if os.name == 'nt' else 'linux-x86_64'))[0])
except IndexError:
    pass
```

```
#
==============================================================================
# -- Add PythonAPI for release mode --------------------------------------------
--
#
==============================================================================
try:

sys.path.append(os.path.dirname(os.path.dirname(os.path.abspath(__file__))) +
'/carla')
except IndexError:
    pass

import carla
from carla import ColorConverter as cc

from agents.navigation.behavior_agent import BehaviorAgent  # pylint:
disable=import-error
from agents.navigation.basic_agent import BasicAgent  # pylint:
disable=import-error


#
==============================================================================
# -- Global functions ----------------------------------------------------------
--
#
==============================================================================


def find_weather_presets():
    """Method to find weather presets"""
    rgx = re.compile('.+?(?:(?<=[a-z])(?=[A-Z])|(?<=[A-Z])(?=[A-Z][a-z])|$)')
    def name(x): return ' '.join(m.group(0) for m in rgx.finditer(x))
    presets = [x for x in dir(carla.WeatherParameters) if re.match('[A-Z].+',
x)]
    return [(getattr(carla.WeatherParameters, x), name(x)) for x in presets]


def get_actor_display_name(actor, truncate=250):
    """Method to get actor display name"""
    name = ' '.join(actor.type_id.replace('_', '.').title().split('.')[1:])
    return (name[:truncate - 1] + u'\u2026') if len(name) > truncate else name


#
==============================================================================
# -- World ---------------------------------------------------------------
```

```
#
==============================================================================

class World(object):
    """ Class representing the surrounding environment """

    def __init__(self, client, carla_world, hud, args):
        """Constructor method"""
        self._args = args
        self.world = carla_world
        try:
            #self.map = self.world.get_map()
            self.map = client.load_world('Town01_Opt')
        except RuntimeError as error:
            print('RuntimeError: {}'.format(error))
            print('  The server could not send the OpenDRIVE (.xodr) file:')
            print('  Make sure it exists, has the same name of your town, and
is correct.')
            sys.exit(1)
        self.hud = hud
        self.player = None
        self.collision_sensor = None
        self.lane_invasion_sensor = None
        self.gnss_sensor = None
        self.camera_manager = None
        self._weather_presets = find_weather_presets()
        self._weather_index = 0
        self._actor_filter = args.filter
        self.restart(args)
        self.world.on_tick(hud.on_world_tick)
        self.recording_enabled = False
        self.recording_start = 0

    def restart(self, args):
        """Restart the world"""
        # Keep same camera config if the camera manager exists.
        cam_index = self.camera_manager.index if self.camera_manager is not
None else 0
        cam_pos_id = self.camera_manager.transform_index if
self.camera_manager is not None else 0

        # Get a random blueprint.
        blueprint =
self.world.get_blueprint_library().filter('charger_2020')[0]
        blueprint.set_attribute('role_name', 'hero')
        if blueprint.has_attribute('color'):
            color = blueprint.get_attribute('color').recommended_values[0]
            blueprint.set_attribute('color', color)
```

```
        # Spawn the player.
        if self.player is not None:
            spawn_point = self.player.get_transform()
            spawn_point.location.z += 2.0
            spawn_point.rotation.roll = 0.0
            spawn_point.rotation.pitch = 0.0
            self.destroy()
            self.player = self.world.try_spawn_actor(blueprint, spawn_point)
            self.modify_vehicle_physics(self.player)
        while self.player is None:
            spawn_point = carla.Transform()
            spawn_point.location.x = 10.868797
            spawn_point.location.y = 2.461965
            spawn_point.location.z = 0.3
            spawn_point.rotation.pitch = 0
            spawn_point.rotation.yaw = 0
            spawn_point.rotation.roll = 0

            self.player = self.world.try_spawn_actor(blueprint, spawn_point)
            self.modify_vehicle_physics(self.player)

        if self._args.sync:
            self.world.tick()
        else:
            self.world.wait_for_tick()

        # Set up the sensors.
        self.collision_sensor = CollisionSensor(self.player, self.hud)
        self.lane_invasion_sensor = LaneInvasionSensor(self.player, self.hud)
        self.gnss_sensor = GnssSensor(self.player)
        self.camera_manager = CameraManager(self.player, self.hud)
        self.camera_manager.transform_index = cam_pos_id
        self.camera_manager.set_sensor(cam_index, notify=False)
        actor_type = get_actor_display_name(self.player)
        self.hud.notification(actor_type)

    def next_weather(self, reverse=False):
        """Get next weather setting"""
        self._weather_index += -1 if reverse else 1
        self._weather_index %= len(self._weather_presets)
        preset = self._weather_presets[self._weather_index]
        self.hud.notification('Weather: %s' % preset[1])
        self.player.get_world().set_weather(preset[0])

    def modify_vehicle_physics(self, actor):
        #If actor is not a vehicle, we cannot use the physics control
        try:
```

```
            physics_control = actor.get_physics_control()
            physics_control.use_sweep_wheel_collision = True
            actor.apply_physics_control(physics_control)
        except Exception:
            pass

    def tick(self, clock):
        """Method for every tick"""
        self.hud.tick(self, clock)

    def render(self, display):
        """Render world"""
        self.camera_manager.render(display)
        self.hud.render(display)

    def destroy_sensors(self):
        """Destroy sensors"""
        self.camera_manager.sensor.destroy()
        self.camera_manager.sensor = None
        self.camera_manager.index = None

    def destroy(self):
        """Destroys all actors"""
        actors = [
            self.camera_manager.sensor,
            self.collision_sensor.sensor,
            self.lane_invasion_sensor.sensor,
            self.gnss_sensor.sensor,
            self.player]
        for actor in actors:
            if actor is not None:
                actor.destroy()


# ==============================================================================
# -- KeyboardControl -----------------------------------------------------------
# ==============================================================================


class KeyboardControl(object):
    def __init__(self, world):
        world.hud.notification("Press 'H' or '?' for help.", seconds=4.0)

    def parse_events(self):
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
```

```
                return True
            if event.type == pygame.KEYUP:
                if self._is_quit_shortcut(event.key):
                    return True

    @staticmethod
    def _is_quit_shortcut(key):
        """Shortcut for quitting"""
        return (key == K_ESCAPE) or (key == K_q and pygame.key.get_mods() &
KMOD_CTRL)

#
==============================================================================
# -- HUD -----------------------------------------------------------------
--
#
==============================================================================


class HUD(object):
    """Class for HUD text"""

    def __init__(self, width, height):
        """Constructor method"""
        self.dim = (width, height)
        font = pygame.font.Font(pygame.font.get_default_font(), 20)
        font_name = 'courier' if os.name == 'nt' else 'mono'
        fonts = [x for x in pygame.font.get_fonts() if font_name in x]
        default_font = 'ubuntumono'
        mono = default_font if default_font in fonts else fonts[0]
        mono = pygame.font.match_font(mono)
        self._font_mono = pygame.font.Font(mono, 12 if os.name == 'nt' else
14)
        self._notifications = FadingText(font, (width, 40), (0, height - 40))
        self.help = HelpText(pygame.font.Font(mono, 24), width, height)
        self.server_fps = 0
        self.frame = 0
        self.simulation_time = 0
        self._show_info = True
        self._info_text = []
        self._server_clock = pygame.time.Clock()

    def on_world_tick(self, timestamp):
        """Gets informations from the world at every tick"""
        self._server_clock.tick()
        self.server_fps = self._server_clock.get_fps()
        self.frame = timestamp.frame_count
        self.simulation_time = timestamp.elapsed_seconds
```

```
    def tick(self, world, clock):
        """HUD method for every tick"""
        self._notifications.tick(world, clock)
        if not self._show_info:
            return
        transform = world.player.get_transform()
        vel = world.player.get_velocity()

        control = world.player.get_control()
        heading = 'N' if abs(transform.rotation.yaw) < 89.5 else ''
        heading += 'S' if abs(transform.rotation.yaw) > 90.5 else ''
        heading += 'E' if 179.5 > transform.rotation.yaw > 0.5 else ''
        heading += 'W' if -0.5 > transform.rotation.yaw > -179.5 else ''
        colhist = world.collision_sensor.get_collision_history()
        collision = [colhist[x + self.frame - 200] for x in range(0, 200)]
        max_col = max(1.0, max(collision))
        collision = [x / max_col for x in collision]
        vehicles = world.world.get_actors().filter('vehicle.*')

        self._info_text = [
            'Server:  % 16.0f FPS' % self.server_fps,
            'Client:  % 16.0f FPS' % clock.get_fps(),
            '',
            'Vehicle: % 20s' % get_actor_display_name(world.player,
truncate=20),
            #'Map:     % 20s' % world.map.name.split('/')[-1],
            'Simulation time: % 12s' %
datetime.timedelta(seconds=int(self.simulation_time)),
            '',
            'Speed:   % 15.0f km/h' % (3.6 * math.sqrt(vel.x**2 + vel.y**2 +
vel.z**2)),
            u'Heading:% 16.0f\N{DEGREE SIGN} % 2s' % (transform.rotation.yaw,
heading),
            'Location:% 20s' % ('(% 5.1f, % 5.1f)' % (transform.location.x,
transform.location.y)),
            'GNSS:% 24s' % ('(% 2.6f, % 3.6f)' % (world.gnss_sensor.lat,
world.gnss_sensor.lon)),
            'Height:  % 18.0f m' % transform.location.z,
            '']
        if isinstance(control, carla.VehicleControl):
            self._info_text += [
                ('Throttle:', control.throttle, 0.0, 1.0),
                ('Steer:', control.steer, -1.0, 1.0),
                ('Brake:', control.brake, 0.0, 1.0),
                ('Reverse:', control.reverse),
                ('Hand brake:', control.hand_brake),
                ('Manual:', control.manual_gear_shift),
```

```
                'Gear:        %s' % {-1: 'R', 0: 'N'}.get(control.gear,
control.gear)]
        elif isinstance(control, carla.WalkerControl):
            self._info_text += [
                ('Speed:', control.speed, 0.0, 5.556),
                ('Jump:', control.jump)]
        self._info_text += [
            '',
            'Collision:',
            collision,
            '',
            'Number of vehicles: % 8d' % len(vehicles)]

        if len(vehicles) > 1:
            self._info_text += ['Nearby vehicles:']

        def dist(l):
            return math.sqrt((l.x - transform.location.x)**2 + (l.y -
transform.location.y)
                             ** 2 + (l.z - transform.location.z)**2)
        vehicles = [(dist(x.get_location()), x) for x in vehicles if x.id !=
world.player.id]

        for dist, vehicle in sorted(vehicles):
            if dist > 200.0:
                break
            vehicle_type = get_actor_display_name(vehicle, truncate=22)
            self._info_text.append('% 4dm %s' % (dist, vehicle_type))

    def toggle_info(self):
        """Toggle info on or off"""
        self._show_info = not self._show_info

    def notification(self, text, seconds=2.0):
        """Notification text"""
        self._notifications.set_text(text, seconds=seconds)

    def error(self, text):
        """Error text"""
        self._notifications.set_text('Error: %s' % text, (255, 0, 0))

    def render(self, display):
        """Render for HUD class"""
        if self._show_info:
            info_surface = pygame.Surface((220, self.dim[1]))
            info_surface.set_alpha(100)
            display.blit(info_surface, (0, 0))
            v_offset = 4
```

```
            bar_h_offset = 100
            bar_width = 106
            for item in self._info_text:
                if v_offset + 18 > self.dim[1]:
                    break
                if isinstance(item, list):
                    if len(item) > 1:
                        points = [(x + 8, v_offset + 8 + (1 - y) * 30) for x,
y in enumerate(item)]
                        pygame.draw.lines(display, (255, 136, 0), False,
points, 2)
                    item = None
                    v_offset += 18
                elif isinstance(item, tuple):
                    if isinstance(item[1], bool):
                        rect = pygame.Rect((bar_h_offset, v_offset + 8), (6,
6))
                        pygame.draw.rect(display, (255, 255, 255), rect, 0 if
item[1] else 1)
                    else:
                        rect_border = pygame.Rect((bar_h_offset, v_offset +
8), (bar_width, 6))
                        pygame.draw.rect(display, (255, 255, 255),
rect_border, 1)
                        fig = (item[1] - item[2]) / (item[3] - item[2])
                        if item[2] < 0.0:
                            rect = pygame.Rect(
                                (bar_h_offset + fig * (bar_width - 6),
v_offset + 8), (6, 6))
                        else:
                            rect = pygame.Rect((bar_h_offset, v_offset + 8),
(fig * bar_width, 6))
                        pygame.draw.rect(display, (255, 255, 255), rect)
                    item = item[0]
                if item:  # At this point has to be a str.
                    surface = self._font_mono.render(item, True, (255, 255,
255))
                    display.blit(surface, (8, v_offset))
                v_offset += 18
        self._notifications.render(display)
        self.help.render(display)

#
==============================================================================
# -- FadingText ----------------------------------------------------------
--
#
==============================================================================
```

```
class FadingText(object):
    """ Class for fading text """

    def __init__(self, font, dim, pos):
        """Constructor method"""
        self.font = font
        self.dim = dim
        self.pos = pos
        self.seconds_left = 0
        self.surface = pygame.Surface(self.dim)

    def set_text(self, text, color=(255, 255, 255), seconds=2.0):
        """Set fading text"""
        text_texture = self.font.render(text, True, color)
        self.surface = pygame.Surface(self.dim)
        self.seconds_left = seconds
        self.surface.fill((0, 0, 0, 0))
        self.surface.blit(text_texture, (10, 11))

    def tick(self, _, clock):
        """Fading text method for every tick"""
        delta_seconds = 1e-3 * clock.get_time()
        self.seconds_left = max(0.0, self.seconds_left - delta_seconds)
        self.surface.set_alpha(500.0 * self.seconds_left)

    def render(self, display):
        """Render fading text method"""
        display.blit(self.surface, self.pos)

#
==============================================================================
# -- HelpText ------------------------------------------------------------
--
#
==============================================================================


class HelpText(object):
    """ Helper class for text render"""

    def __init__(self, font, width, height):
        """Constructor method"""
        lines = __doc__.split('\n')
        self.font = font
        self.dim = (680, len(lines) * 22 + 12)
        self.pos = (0.5 * width - 0.5 * self.dim[0], 0.5 * height - 0.5 *
self.dim[1])
```

```
        self.seconds_left = 0
        self.surface = pygame.Surface(self.dim)
        self.surface.fill((0, 0, 0, 0))
        for i, line in enumerate(lines):
            text_texture = self.font.render(line, True, (255, 255, 255))
            self.surface.blit(text_texture, (22, i * 22))
            self._render = False
        self.surface.set_alpha(220)

    def toggle(self):
        """Toggle on or off the render help"""
        self._render = not self._render

    def render(self, display):
        """Render help text method"""
        if self._render:
            display.blit(self.surface, self.pos)

#
==============================================================================
# -- CollisionSensor -----------------------------------------------------
--
#
==============================================================================


class CollisionSensor(object):
    """ Class for collision sensors"""

    def __init__(self, parent_actor, hud):
        """Constructor method"""
        self.sensor = None
        self.history = []
        self._parent = parent_actor
        self.hud = hud
        world = self._parent.get_world()
        blueprint =
world.get_blueprint_library().find('sensor.other.collision')
        self.sensor = world.spawn_actor(blueprint, carla.Transform(),
attach_to=self._parent)
        # We need to pass the lambda a weak reference to
        # self to avoid circular reference.
        weak_self = weakref.ref(self)
        self.sensor.listen(lambda event:
CollisionSensor._on_collision(weak_self, event))

    def get_collision_history(self):
        """Gets the history of collisions"""
        history = collections.defaultdict(int)
```

```
        for frame, intensity in self.history:
            history[frame] += intensity
        return history

    @staticmethod
    def _on_collision(weak_self, event):
        """On collision method"""
        self = weak_self()
        if not self:
            return
        actor_type = get_actor_display_name(event.other_actor)
        self.hud.notification('Collision with %r' % actor_type)
        impulse = event.normal_impulse
        intensity = math.sqrt(impulse.x ** 2 + impulse.y ** 2 + impulse.z **
2)
        self.history.append((event.frame, intensity))
        if len(self.history) > 4000:
            self.history.pop(0)

#
==============================================================================
# -- LaneInvasionSensor ------------------------------------------------------
--
#
==============================================================================


class LaneInvasionSensor(object):
    """Class for lane invasion sensors"""

    def __init__(self, parent_actor, hud):
        """Constructor method"""
        self.sensor = None
        self._parent = parent_actor
        self.hud = hud
        world = self._parent.get_world()
        bp = world.get_blueprint_library().find('sensor.other.lane_invasion')
        self.sensor = world.spawn_actor(bp, carla.Transform(),
attach_to=self._parent)
        # We need to pass the lambda a weak reference to self to avoid
circular
        # reference.
        weak_self = weakref.ref(self)
        self.sensor.listen(lambda event:
LaneInvasionSensor._on_invasion(weak_self, event))

    @staticmethod
    def _on_invasion(weak_self, event):
        """On invasion method"""
```

```
        self = weak_self()
        if not self:
            return
        lane_types = set(x.type for x in event.crossed_lane_markings)
        text = ['%r' % str(x).split()[-1] for x in lane_types]
        self.hud.notification('Crossed line %s' % ' and '.join(text))

#
==============================================================================
# -- GnssSensor ----------------------------------------------------------
#
==============================================================================


class GnssSensor(object):
    """ Class for GNSS sensors"""

    def __init__(self, parent_actor):
        """Constructor method"""
        self.sensor = None
        self._parent = parent_actor
        self.lat = 0.0
        self.lon = 0.0
        world = self._parent.get_world()
        blueprint = world.get_blueprint_library().find('sensor.other.gnss')
        self.sensor = world.spawn_actor(blueprint,
carla.Transform(carla.Location(x=1.0, z=2.8)),
                                        attach_to=self._parent)
        # We need to pass the lambda a weak reference to
        # self to avoid circular reference.
        weak_self = weakref.ref(self)
        self.sensor.listen(lambda event: GnssSensor._on_gnss_event(weak_self,
event))

    @staticmethod
    def _on_gnss_event(weak_self, event):
        """GNSS method"""
        self = weak_self()
        if not self:
            return
        self.lat = event.latitude
        self.lon = event.longitude

#
==============================================================================
# -- CameraManager -------------------------------------------------------
--
#
==============================================================================
```

```
class CameraManager(object):
    """ Class for camera management"""

    def __init__(self, parent_actor, hud):
        """Constructor method"""
        self.sensor = None
        self.surface = None
        self._parent = parent_actor
        self.hud = hud
        self.recording = False
        bound_y = 0.5 + self._parent.bounding_box.extent.y
        attachment = carla.AttachmentType
        self._camera_transforms = [
            (carla.Transform(
                carla.Location(x=-5.5, z=2.5), carla.Rotation(pitch=8.0)),
attachment.SpringArm),
            (carla.Transform(
                carla.Location(x=1.6, z=1.7)), attachment.Rigid),
            (carla.Transform(
                carla.Location(x=5.5, y=1.5, z=1.5)), attachment.SpringArm),
            (carla.Transform(
                carla.Location(x=-8.0, z=6.0), carla.Rotation(pitch=6.0)),
attachment.SpringArm),
            (carla.Transform(
                carla.Location(x=-1, y=-bound_y, z=0.5)), attachment.Rigid)]
        self.transform_index = 1
        self.sensors = [
            ['sensor.camera.rgb', cc.Raw, 'Camera RGB'],
            ['sensor.camera.depth', cc.Raw, 'Camera Depth (Raw)'],
            ['sensor.camera.depth', cc.Depth, 'Camera Depth (Gray Scale)'],
            ['sensor.camera.depth', cc.LogarithmicDepth, 'Camera Depth
(Logarithmic Gray Scale)'],
            ['sensor.camera.semantic_segmentation', cc.Raw, 'Camera Semantic
Segmentation (Raw)'],
            ['sensor.camera.semantic_segmentation', cc.CityScapesPalette,
             'Camera Semantic Segmentation (CityScapes Palette)'],
            ['sensor.lidar.ray_cast', None, 'Lidar (Ray-Cast)']]
        world = self._parent.get_world()
        bp_library = world.get_blueprint_library()
        for item in self.sensors:
            blp = bp_library.find(item[0])
            if item[0].startswith('sensor.camera'):
                blp.set_attribute('image_size_x', str(hud.dim[0]))
                blp.set_attribute('image_size_y', str(hud.dim[1]))
            elif item[0].startswith('sensor.lidar'):
                blp.set_attribute('range', '50')
            item.append(blp)
```

```
        self.index = None

    def toggle_camera(self):
        """Activate a camera"""
        self.transform_index = (self.transform_index + 1) %
len(self._camera_transforms)
        self.set_sensor(self.index, notify=False, force_respawn=True)

    def set_sensor(self, index, notify=True, force_respawn=False):
        """Set a sensor"""
        index = index % len(self.sensors)
        needs_respawn = True if self.index is None else (
            force_respawn or (self.sensors[index][0] !=
self.sensors[self.index][0]))
        if needs_respawn:
            if self.sensor is not None:
                self.sensor.destroy()
                self.surface = None
            self.sensor = self._parent.get_world().spawn_actor(
                self.sensors[index][-1],
                self._camera_transforms[self.transform_index][0],
                attach_to=self._parent,

attachment_type=self._camera_transforms[self.transform_index][1])

            # We need to pass the lambda a weak reference to
            # self to avoid circular reference.
            weak_self = weakref.ref(self)
            self.sensor.listen(lambda image:
CameraManager._parse_image(weak_self, image))
        if notify:
            self.hud.notification(self.sensors[index][2])
        self.index = index

    def next_sensor(self):
        """Get the next sensor"""
        self.set_sensor(self.index + 1)

    def toggle_recording(self):
        """Toggle recording on or off"""
        self.recording = not self.recording
        self.hud.notification('Recording %s' % ('On' if self.recording else
'Off'))

    def render(self, display):
        """Render method"""
        if self.surface is not None:
            display.blit(self.surface, (0, 0))
```

```
    @staticmethod
    def _parse_image(weak_self, image):
        self = weak_self()
        if not self:
            return
        if self.sensors[self.index][0].startswith('sensor.lidar'):
            points = np.frombuffer(image.raw_data, dtype=np.dtype('f4'))
            points = np.reshape(points, (int(points.shape[0] / 4), 4))
            lidar_data = np.array(points[:, :2])
            lidar_data *= min(self.hud.dim) / 100.0
            lidar_data += (0.5 * self.hud.dim[0], 0.5 * self.hud.dim[1])
            lidar_data = np.fabs(lidar_data)  # pylint: disable=assignment-
from-no-return
            lidar_data = lidar_data.astype(np.int32)
            lidar_data = np.reshape(lidar_data, (-1, 2))
            lidar_img_size = (self.hud.dim[0], self.hud.dim[1], 3)
            lidar_img = np.zeros(lidar_img_size)
            lidar_img[tuple(lidar_data.T)] = (255, 255, 255)
            self.surface = pygame.surfarray.make_surface(lidar_img)
        else:
            image.convert(self.sensors[self.index][1])
            array = np.frombuffer(image.raw_data, dtype=np.dtype("uint8"))
            array = np.reshape(array, (image.height, image.width, 4))
            array = array[:, :, :3]
            array = array[:, :, ::-1]
            self.surface = pygame.surfarray.make_surface(array.swapaxes(0, 1))
        if self.recording:
            image.save_to_disk('_out/%08d' % image.frame)

#
==============================================================================
# -- Game Loop ---------------------------------------------------------
#
==============================================================================


global xscr
xscr = 0


def sub_callback(msg):
    global xscr
    xscr = msg.data
    #print(xscr)


def run_step():
    global xscr
```

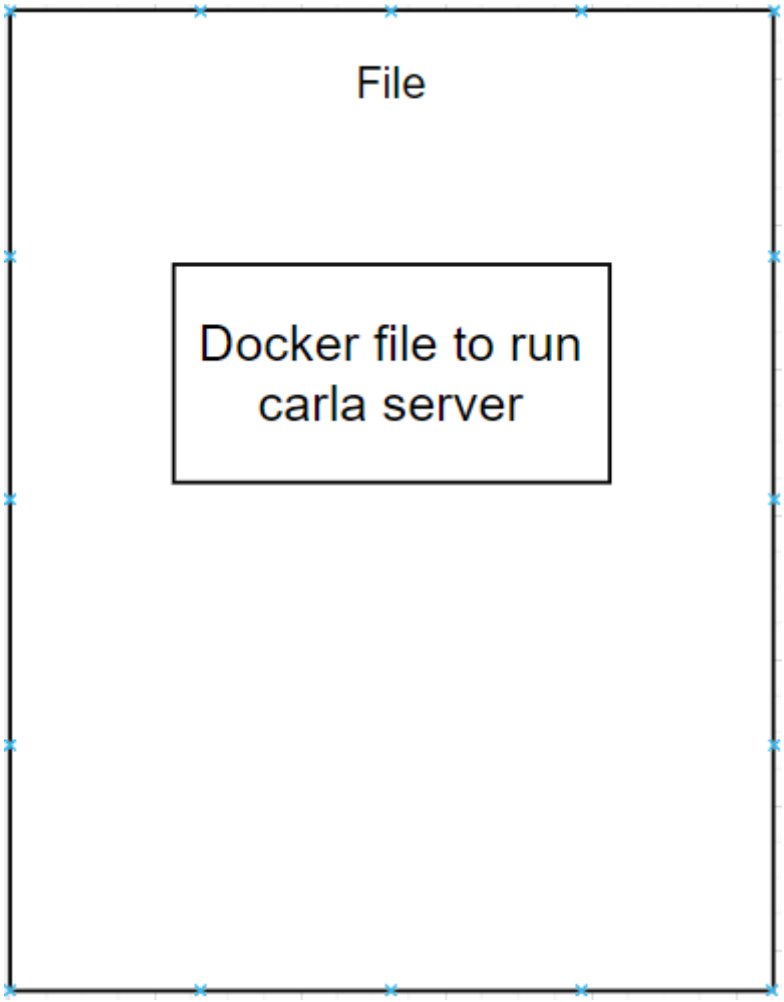
```
    control = carla.VehicleControl()
    control.steer = 0.0
    print(xscr)
    control.throttle = xscr
    control.brake = 0.0
    control.hand_brake = False
    control.manual_gear_shift = False
    control.reverse = False
    return control


def game_loop(args):
    """
    Main loop of the simulation. It handles updating all the HUD information,
    ticking the agent and, if needed, the world.
    """

    pygame.init()
    pygame.font.init()
    world = None

    try:
        if args.seed:
            random.seed(args.seed)

        client = carla.Client(args.host, args.port)
        client.set_timeout(4.0)

        traffic_manager = client.get_trafficmanager()
        sim_world = client.get_world()

        if args.sync:
            settings = sim_world.get_settings()
            settings.synchronous_mode = True
            settings.fixed_delta_seconds = 0.05
            sim_world.apply_settings(settings)

            traffic_manager.set_synchronous_mode(True)

        display = pygame.display.set_mode(
            (args.width, args.height),
            pygame.HWSURFACE | pygame.DOUBLEBUF)

        hud = HUD(args.width, args.height)
        world = World(client, client.get_world(), hud, args)
        controller = KeyboardControl(world)
        if args.agent == "Basic":
            agent = BasicAgent(world.player)
        else:
```

```
            agent = BehaviorAgent(world.player, behavior=args.behavior)

        clock = pygame.time.Clock()

        while True:
            clock.tick()
            if args.sync:
                world.world.tick()
            else:
                world.world.wait_for_tick()
            if controller.parse_events():
                return

            world.tick(clock)
            world.render(display)
            pygame.display.flip()

            # if agent.done():
            #     if args.loop:
            #
agent.set_destination(random.choice(spawn_points).location)
            #         world.hud.notification("The target has been reached,
searching for another target", seconds=4.0)
            #         print("The target has been reached, searching for
another target")
            #     else:
            #         print("The target has been reached, stopping the
simulation")
            #         break

            control = run_step()  #agent.run_step()
            world.player.apply_control(control)

    finally:

        if world is not None:
            settings = world.world.get_settings()
            settings.synchronous_mode = False
            settings.fixed_delta_seconds = None
            world.world.apply_settings(settings)
            traffic_manager.set_synchronous_mode(True)

            world.destroy()

        pygame.quit()


#
==============================================================================
```

```
# -- main() ---------------------------------------------------------------
#
# ==============================================================================


def main():
    """Main method"""

    argparser = argparse.ArgumentParser(
        description='CARLA Automatic Control Client')
    argparser.add_argument(
        '-v', '--verbose',
        action='store_true',
        dest='debug',
        help='Print debug information')
    argparser.add_argument(
        '--host',
        metavar='H',
        default='127.0.0.1',
        help='IP of the host server (default: 127.0.0.1)')
    argparser.add_argument(
        '-p', '--port',
        metavar='P',
        default=2000,
        type=int,
        help='TCP port to listen to (default: 2000)')
    argparser.add_argument(
        '--res',
        metavar='WIDTHxHEIGHT',
        default='1280x720',
        help='Window resolution (default: 1280x720)')
    argparser.add_argument(
        '--sync',
        action='store_true',
        help='Synchronous mode execution')
    argparser.add_argument(
        '--filter',
        metavar='PATTERN',
        default='vehicle.*',
        help='Actor filter (default: "vehicle.*")')
    argparser.add_argument(
        '-l', '--loop',
        action='store_true',
        dest='loop',
        help='Sets a new random destination upon reaching the previous one
(default: False)')
    argparser.add_argument(
        "-a", "--agent", type=str,
        choices=["Behavior", "Basic"],
```

```
        help="select which agent to run",
        default="Behavior")
    argparser.add_argument(
        '-b', '--behavior', type=str,
        choices=["cautious", "normal", "aggressive"],
        help='Choose one of the possible agent behaviors (default: normal) ',
        default='normal')
    argparser.add_argument(
        '-s', '--seed',
        help='Set seed for repeating executions (default: None)',
        default=None,
        type=int)

    args = argparser.parse_args()

    args.width, args.height = [int(x) for x in args.res.split('x')]

    log_level = logging.DEBUG if args.debug else logging.INFO
    logging.basicConfig(format='%(levelname)s: %(message)s', level=log_level)

    logging.info('listening to server %s:%s', args.host, args.port)

    print(__doc__)

    try:
        game_loop(args)

    except KeyboardInterrupt:
        print('\nCancelled by user. Bye!')


if __name__ == '__main__':
    rospy.init_node('listener', anonymous=True)
    rospy.Subscriber('/out_value', Float32, sub_callback)
    main()
```

5.




```
#!/bin/sh

sudo docker run -d -p 2000-2002:2000-2002 --gpus all -e
NVIDIA_VISIBLE_DEVICES=0 carlasim/carla:0.9.13 /bin/bash CarlaUE4.sh -
RenderOffScreen -quality-level=Low
```