jupyter sumo_test Last Checkpoint: 2 minutes ago (autosaved)   Logout

File Edit View Insert Cell Kernel Help   Trusted   Python 3 (ipykernel)

Run   Code

In [1]:
```python
import os
import sys
from pathlib import Path
if 'SUMO_HOME' in os.environ:
    tools = os.path.join(os.environ['SUMO_HOME'], 'tools')
    sys.path.append(tools)
else:
    sys.exit("Please declare the environment variable 'SUMO_HOME'")
import traci
import sumolib
from ray.rllib.env.multi_agent_env import MultiAgentEnv
from gym.envs.registration import EnvSpec
import numpy as np
import pandas as pd

# from environment.traffic_signal import TrafficSignal
```

```
/home/valaryan/.local/lib/python3.6/site-packages/tensorflow/python/framework/dtypes.py:526: FutureWarning: Passin
g (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as
(type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
/home/valaryan/.local/lib/python3.6/site-packages/tensorflow/python/framework/dtypes.py:527: FutureWarning: Passin
g (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as
(type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
/home/valaryan/.local/lib/python3.6/site-packages/tensorflow/python/framework/dtypes.py:528: FutureWarning: Passin
g (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as
(type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
/home/valaryan/.local/lib/python3.6/site-packages/tensorflow/python/framework/dtypes.py:529: FutureWarning: Passin
g (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as
(type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
/home/valaryan/.local/lib/python3.6/site-packages/tensorflow/python/framework/dtypes.py:530: FutureWarning: Passin
g (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as
(type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
/home/valaryan/.local/lib/python3.6/site-packages/tensorflow/python/framework/dtypes.py:535: FutureWarning: Passin
g (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as
(type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
```

In [ ]:
```python

```

In [2]:
```python
class TrafficSignal:
    """
    This class represents a Traffic Signal of an intersection
    It is responsible for retrieving information and changing the traffic phase using Traci API
    """

    def __init__(self, env, ts_id, delta_time, yellow_time, min_green, max_green):
        self.id = ts_id
        self.env = env
        self.delta_time = delta_time
        self.yellow_time = yellow_time
        self.min_green = min_green
        self.max_green = max_green
        self.green_phase = 0
        self.is_yellow = False
        self.time_since_last_phase_change = 0
        self.next_action_time = 0
        self.last_measure = 0.0
        self.last_reward = None
        self.phases = traci.trafficlight.getCompleteRedYellowGreenDefinition(self.id)[0].phases
        self.num_green_phases = len(self.phases) // 2  # Number of green phases == number of phases (green+yell
        self.lanes = list(dict.fromkeys(traci.trafficlight.getControlledLanes(self.id)))  # Remove duplicates a
        self.out_lanes = [link[0][1] for link in traci.trafficlight.getControlledLinks(self.id) if link]
        self.out_lanes = list(set(self.out_lanes))

        """
        Default observation space is a vector R^(#greenPhases + 2 * #lanes)
        s = [current phase one-hot encoded, density for each lane, queue for each lane]
        You can change this by modifing self.observation_space and the method _compute_observations()

        Action space is which green phase is going to be open for the next delta_time seconds
        """
        self.observation_space = spaces.Box(low=np.zeros(self.num_green_phases + 2*len(self.lanes)), high=np.or
        self.discrete_observation_space = spaces.Tuple((
            spaces.Discrete(self.num_green_phases),                       # Green Phase
            #spaces.Discrete(self.max_green//self.delta_time),            # Elapsed time of phase
            *(spaces.Discrete(10) for _ in range(2*len(self.lanes)))      # Density and stopped-density for eac
        ))
        self.action_space = spaces.Discrete(self.num_green_phases)

        programs = traci.trafficlight.getAllProgramLogics(self.id)
        logic = programs[0]
        logic.type = 0
        logic.phases = self.phases
        traci.trafficlight.setProgramLogic(self.id, logic)

    @property
    @property
    def phase(self):
        return traci.trafficlight.getPhase(self.id)

    @property
    def time_to_act(self):
        return self.next_action_time == self.env.sim_step

    def update(self):
        self.time_since_last_phase_change += 1
        if self.is_yellow and self.time_since_last_phase_change == self.yellow_time:
            traci.trafficlight.setPhase(self.id, int(self.green_phase))
            self.is_yellow = False

    def set_next_phase(self, new_phase):
        """
        Sets what will be the next green phase and sets yellow phase if the next phase is different than the cu

        :param new_phase: (int) Number between [0..num_green_phases]
        """
        new_phase *= 2
        if self.phase == new_phase or self.time_since_last_phase_change < self.min_green + self.yellow_time:
            self.green_phase = self.phase
            traci.trafficlight.setPhase(self.id, self.green_phase)
            self.next_action_time = self.env.sim_step + self.delta_time
        else:
            self.green_phase = new_phase
            traci.trafficlight.setPhase(self.id, self.phase + 1)  # turns yellow
            self.next_action_time = self.env.sim_step + self.delta_time + self.yellow_time
            self.is_yellow = True
            self.time_since_last_phase_change = 0

    def compute_observation(self):
        phase_id = [1 if self.phase//2 == i else 0 for i in range(self.num_green_phases)]  # one-hot encoding
        #elapsed = self.traffic_signals[ts].time_on_phase / self.max_green
        density = self.get_lanes_density()
        queue = self.get_lanes_queue()
        observation = np.array(phase_id + density + queue)
        return observation

    def compute_reward(self):
        self.last_reward = self._waiting_time_reward()
        return self.last_reward

    def _pressure_reward(self):
        return -self.get_pressure()

    def _queue_average_reward(self):
        new_average = np.mean(self.get_stopped_vehicles_num())
        reward = self.last_measure - new_average
        self.last_measure = new_average
        return reward

    def _queue_reward(self):
        return - (sum(self.get_stopped_vehicles_num()))**2

    def _waiting_time_reward(self):
        ts_wait = sum(self.get_waiting_time_per_lane()) / 100.0
        reward = self.last_measure - ts_wait
        self.last_measure = ts_wait
        return reward

    def _waiting_time_reward2(self):
        ts_wait = sum(self.get_waiting_time())
        self.last_measure = ts_wait
        if ts_wait == 0:
            reward = 1.0
        else:
            reward = 1.0/ts_wait
        return reward

    def _waiting_time_reward3(self):
        ts_wait = sum(self.get_waiting_time())
        reward = -ts_wait
        self.last_measure = ts_wait
        return reward

    def get_waiting_time_per_lane(self):
        wait_time_per_lane = []
        for lane in self.lanes:
            veh_list = traci.lane.getLastStepVehicleIDs(lane)
            wait_time = 0.0
            for veh in veh_list:
                veh_lane = traci.vehicle.getLaneID(veh)
                acc = traci.vehicle.getAccumulatedWaitingTime(veh)
                if veh not in self.env.vehicles:
                    self.env.vehicles[veh] = {veh_lane: acc}
                else:
                    self.env.vehicles[veh][veh_lane] = acc - sum([self.env.vehicles[veh][lane] for lane in self
                wait_time += self.env.vehicles[veh][veh_lane]
            wait_time_per_lane.append(wait_time)
        return wait_time_per_lane

    def get_pressure(self):
        return abs(sum(traci.lane.getLastStepVehicleNumber(lane) for lane in self.lanes) - sum(traci.lane.getLa

    def get_out_lanes_density(self):
        vehicle_size_min_gap = 7.5  # 5(vehSize) + 2.5(minGap)
        return [min(1, traci.lane.getLastStepVehicleNumber(lane) / (traci.lane.getLength(lane) / vehicle_size_m

    def get_lanes_density(self):
        vehicle_size_min_gap = 7.5  # 5(vehSize) + 2.5(minGap)
        return [min(1, traci.lane.getLastStepVehicleNumber(lane) / (traci.lane.getLength(lane) / vehicle_size_m

    def get_lanes_queue(self):
        vehicle_size_min_gap = 7.5  # 5(vehSize) + 2.5(minGap)
        return [min(1, traci.lane.getLastStepHaltingNumber(lane) / (traci.lane.getLength(lane) / vehicle_size_m

    def get_total_queued(self):
        return sum([traci.lane.getLastStepHaltingNumber(lane) for lane in self.lanes])

    def _get_veh_list(self):
        veh_list = []
        for lane in self.lanes:
            veh_list += traci.lane.getLastStepVehicleIDs(lane)
        return veh_list

```

In [3]:
```
# Multiagent
# env = SumoEnvironment(net_file='nets/4x4-Lucas/4x4.net.xml',
#                       route_file='nets/4x4-Lucas/4x4c1c2c1c2.rou.xml',
#                       use_gui=True,
#                       num_seconds=80000,
#                       max_depart_delay=0)
```

In [4]:
```
out_csv_name=None
use_gui=False
num_seconds=20000
max_depart_delay=100000
time_to_teleport=-1
delta_time=5
yellow_time=2
min_green=5
max_green=50
single_agent=False


net_file='nets/4x4-Lucas/4x4.net.xml'
route_file='nets/4x4-Lucas/4x4c1c2c1c2.rou.xml'
use_gui=True
num_seconds=80000
max_depart_delay=0
delta_time=5


_net = net_file
_route = route_file
use_gui = use_gui
if use_gui:
    _sumo_binary = sumolib.checkBinary('sumo-gui')
else:
    _sumo_binary = sumolib.checkBinary('sumo')

sim_max_time = num_seconds
delta_time = delta_time  # seconds on sumo at each step
max_depart_delay = max_depart_delay  # Max wait time to insert a vehicle
time_to_teleport = time_to_teleport
min_green = min_green
max_green = max_green
yellow_time = yellow_time

# start only to retrieve information

traci.start([sumolib.checkBinary('sumo'), '-n', _net])
ts_ids = traci.trafficlight.getIDList()
# traffic_signals = {ts: TrafficSignal(
#     ts, delta_time, yellow_time, min_green, max_green) for ts in ts_ids}
```

```
Retrying in 1 seconds
```

In [5]:
```
ts_ids
```

Out[5]:
```
('0',
 '1',
 '10',
 '11',
 '12',
 '13',
 '14',
 '15',
 '2',
 '3',
 '4',
 '5',
 '6',
 '7',
 '8',
 '9')
```

In [6]:
```
traci.trafficlight.getRedYellowGreenState(ts_ids[0])
# traci.trafficlight.getCompleteRedYellowGreenDefinition(ts_id[0])
# traci.trafficlight.getAllProgramLogics
# traci.trafficlight.getCompleteRedYellowGreenDefinition(ts_ids[0])
programs = traci.trafficlight.getAllProgramLogics(ts_ids[0])
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-6-c6c8f9cd9852> in <module>
      3 # traci.trafficlight.getAllProgramLogics
      4 # traci.trafficlight.getCompleteRedYellowGreenDefinition(ts_ids[0])
----> 5 programs = traci.trafficlight.getAllProgramLogics(ts_ids[0])

~/.local/lib/python3.6/site-packages/traci/_trafficlight.py in getAllProgramLogics(self, tlsID)
    179         Each Logic encodes a traffic light program for the given tlsID.
    180         """
--> 181         return self._getUniversal(tc.TL_COMPLETE_DEFINITION_RYG, tlsID)
    182 
    183     getCompleteRedYellowGreenDefinition = getAllProgramLogics

~/.local/lib/python3.6/site-packages/traci/domain.py in _getUniversal(self, varID, objectID, format, *values)
    170         if self._deprecatedFor:
    171             warnings.warn("The domain %s is deprecated, use %s instead." % (self._name, self._deprecatedFo
r))
--> 172         return _parse(self._retValFunc, varID, self._getCmd(varID, objectID, format, *values))
    173 
    174     def _getCmd(self, varID, objID, format="", *values):

~/.local/lib/python3.6/site-packages/traci/domain.py in _parse(valueFunc, varID, data)
     42     varType = data.read("!B")[0]
     43     if varID in valueFunc:
---> 44         return valueFunc[varID](data)
     45     if varType in (tc.POSITION_2D, tc.POSITION_LON_LAT):
     46         return data.read("!dd")

~/.local/lib/python3.6/site-packages/traci/_trafficlight.py in _readLogics(result)
     78         phases = []
     79         for __ in range(numPhases):
---> 80             result.readCompound(6)
     81             duration = result.readTypedDouble()
     82             state = result.readTypedString()

~/.local/lib/python3.6/site-packages/traci/storage.py in readCompound(self, expectedSize)
     92         t, s = self.read("!Bi")
     93         assert(t == tc.TYPE_COMPOUND)
---> 94         assert(expectedSize is None or s == expectedSize)
     95         return s
     96 

AssertionError: 
```

In [ ]:

```python
# class SumoEnvironment(MultiAgentEnv):
#     def __init__(self, net_file, route_file, out_csv_name=None, use_gui=False, num_seconds=20000, max_depart_
#                  time_to_teleport=-1, delta_time=5, yellow_time=2, min_green=5, max_green=50, single_agent=Fa
#         self._net = net_file
#         self._route = route_file
#         self.use_gui = use_gui
#         if self.use_gui:
#             self._sumo_binary = sumolib.checkBinary('sumo-gui')
#         else:
#             self._sumo_binary = sumolib.checkBinary('sumo')

#         self.sim_max_time = num_seconds
#         self.delta_time = delta_time  # seconds on sumo at each step
#         self.max_depart_delay = max_depart_delay  # Max wait time to insert a vehicle
#         self.time_to_teleport = time_to_teleport
#         self.min_green = min_green
#         self.max_green = max_green
#         self.yellow_time = yellow_time

#         # start only to retrieve information
#         print("Starting Traci")
#         traci.start([sumolib.checkBinary('sumo'), '-n', self._net])
#         print("Traci Started")

#         self.single_agent = single_agent

#         self.ts_ids = traci.trafficlight.getIDList()

#         print("Got traffic Signal Ids")
#         print(self.ts_ids)

#         print("Initiating Traffic Signal Class")
#         self.traffic_signals = {ts: TrafficSignal(
#             self, ts, self.delta_time, self.yellow_time, self.min_green, self.max_green) for ts in self.ts_ids
#         print("Traffic Signal Class Successfully Initiated")

#         self.vehicles = dict()

#         self.reward_range = (-float('inf'), float('inf'))

#         self.metadata = {}

#         self.spec = EnvSpec('SUMORL-v0')

#         self.run = 0
#         self.metrics = []
#         self.out_csv_name = out_csv_name

#         traci.close()
```

In [ ]:

```python
# time_to_teleport=-1
# yellow_time=2
# min_green=5
# max_green=50
# single_agent=False

# net_file='nets/4x4-Lucas/4x4.net.xml'
# route_file='nets/4x4-Lucas/4x4c1c2c1c2.rou.xml'
# use_gui=True
# num_seconds=80000
# max_depart_delay=0
# delta_time=5
# out_csv_name=None
# use_gui=False

# env = SumoEnvironment(net_file, route_file, out_csv_name, use_gui, num_seconds, max_depart_delay,
#                   time_to_teleport, delta_time, yellow_time, min_green, max_green, single_agent)
```

In [ ]:

In [ ]:

In [ ]: