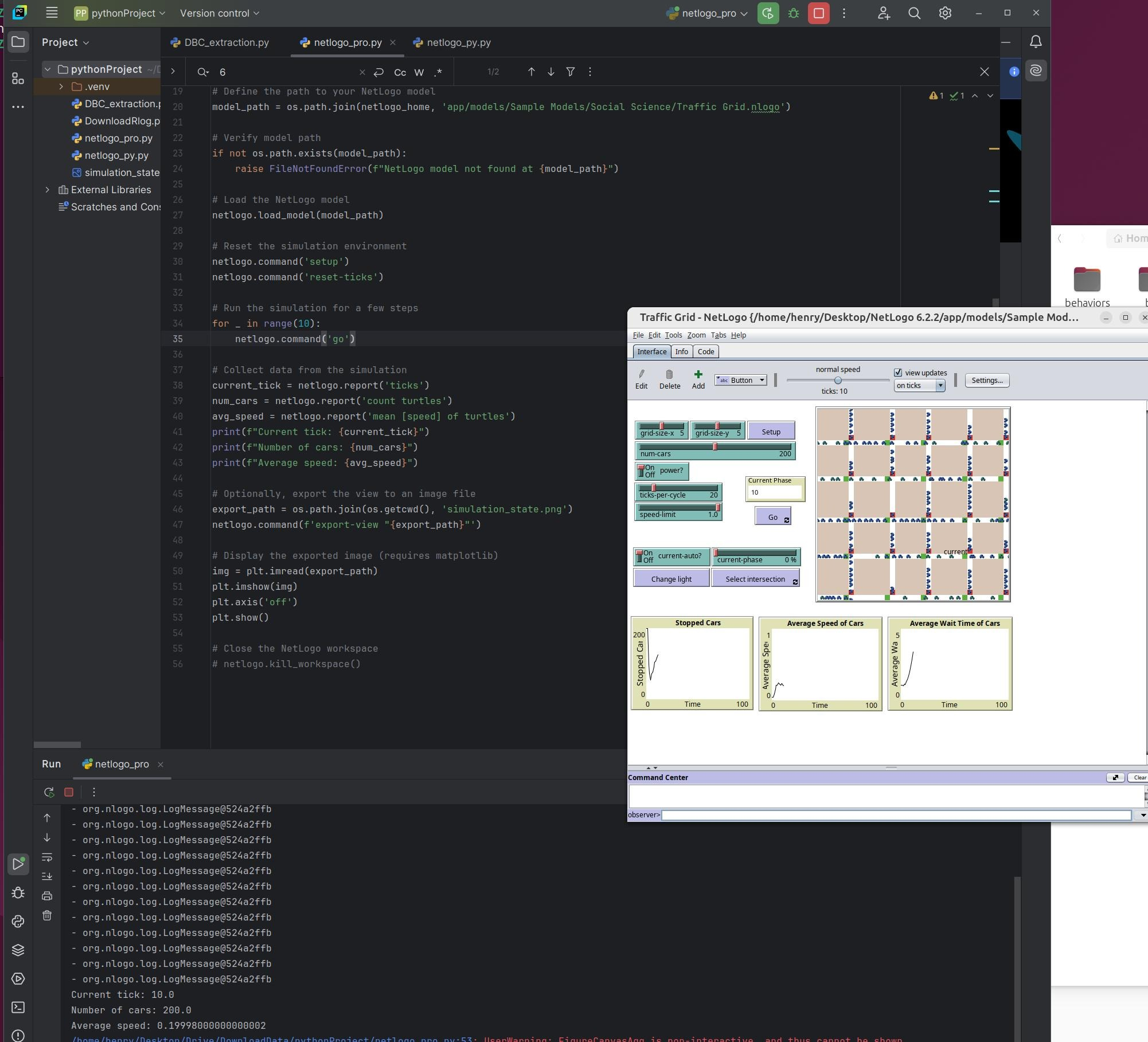
It has the python API, which means that we can do basic functions for RL training, including the initialization (reset) and tick the clock of the simulation (step).



To leverage all the exsiting advanced RL algorithms, we just need open-ai GYM style environments. To define an environment, we just need reset() function and the step() function, which can both be done through python api.

The true limitation is: i) the underlying dynamics of the model, and ii) the available actions for us to take to change the environment, and the iii) observations the model has. Basically the agent is limited to the available actions it can take, and the observation it can have as well. Those limitations are specific to the netlogo model we use.

The available actions in this Traffic Grid simulator

# Define the coordinates of the intersection you want to control

intersection\_x = 0

intersection\_y = 0

# Make the intersection at (intersection\_x, intersection\_y) the current light

netlogo.command(f'make-current patch {intersection\_x} {intersection\_y}')

# Switch the traffic light at the current intersection

netlogo.command('change-current')

The Gym-style netlogo env example:  
  
import gym

from gym import spaces

import numpy as np

import pynetlogo

import os

import matplotlib

matplotlib.use('Agg') # Use non-interactive backend

import matplotlib.pyplot as plt

import logging

logging.basicConfig(level=logging.INFO)

logger = logging.getLogger(\_\_name\_\_)

class NetLogoTrafficGridEnv(gym.Env):

"""Custom Environment for NetLogo Traffic Grid Model"""

metadata = {'render.modes': ['human', 'rgb\_array']}

def \_\_init\_\_(self):

super(NetLogoTrafficGridEnv, self).\_\_init\_\_()

# Initialize NetLogoLink

self.netlogo\_home = '/home/henry/Desktop/NetLogo 6.2.2/'

self.netlogo\_version = '6.2.2'

# Verify NetLogo home path

if not os.path.exists(self.netlogo\_home):

raise FileNotFoundError(f"NetLogo home directory not found at {self.netlogo\_home}")

# Initialize the NetLogo link

self.netlogo = pynetlogo.NetLogoLink(gui=False, netlogo\_home=self.netlogo\_home)

# Define the path to your NetLogo model

self.model\_path = os.path.join(self.netlogo\_home, 'app/models/Sample Models/Social Science/Traffic Grid.nlogo')

# Verify model path

if not os.path.exists(self.model\_path):

raise FileNotFoundError(f"NetLogo model not found at {self.model\_path}")

# Load the NetLogo model

self.netlogo.load\_model(self.model\_path)

# Set the random seed for reproducibility

random\_seed = 42

self.netlogo.command(f'random-seed {random\_seed}')

np.random.seed(random\_seed)

# Get the grid size

self.grid\_size\_x = int(self.netlogo.report('grid-size-x'))

self.grid\_size\_y = int(self.netlogo.report('grid-size-y'))

self.num\_intersections = self.grid\_size\_x \* self.grid\_size\_y

# Define action and observation space

self.action\_space = spaces.MultiBinary(self.num\_intersections)

low = np.zeros((self.num\_intersections, 3), dtype=int)

high = np.ones((self.num\_intersections, 3), dtype=int)

high[:, 1:] = 10 # Assuming max 10 cars per direction

self.observation\_space = spaces.Box(low=low, high=high, dtype=np.int32)

self.current\_tick = 0

self.max\_ticks = 1000 # Maximum simulation steps per episode

# Custom setup

self.\_setup\_simulation()

def \_setup\_simulation(self):

try:

self.netlogo.command('setup')

except Exception as e:

logger.error(f"Error during setup: {str(e)}")

raise

def reset(self):

self.\_setup\_simulation()

obs = self.\_get\_observation()

self.current\_tick = 0

return obs

def step(self, action):

# Execute the action for each intersection

for i in range(self.num\_intersections):

if action[i] == 1:

self.\_switch\_light(i)

# Advance the simulation by one tick

try:

self.netlogo.command('go')

except Exception as e:

logger.error(f"Error during 'go' command: {str(e)}")

raise

self.current\_tick += 1

# Get the new observation

obs = self.\_get\_observation()

# Compute reward (negative of total waiting time)

try:

total\_wait\_time = int(self.netlogo.report('sum [wait-time] of turtles'))

reward = -total\_wait\_time

except Exception as e:

logger.error(f"Error computing reward: {str(e)}")

reward = 0

# Check if the episode is done

done = self.current\_tick >= self.max\_ticks

# Additional info

info = {}

try:

info['num\_cars\_stopped'] = int(self.netlogo.report('num-cars-stopped'))

info['avg\_speed'] = float(self.netlogo.report('mean [speed] of turtles'))

except Exception as e:

logger.error(f"Error getting additional info: {str(e)}")

return obs, reward, done, info

def render(self, mode='human'):

# Export the view to an image file

export\_path = os.path.join(os.getcwd(), 'simulation\_state.png')

try:

self.netlogo.command(f'export-view "{export\_path}"')

except Exception as e:

logger.error(f"Error exporting view: {str(e)}")

return None

# Read the exported image

img = plt.imread(export\_path)

if mode == 'rgb\_array':

return img

elif mode == 'human':

plt.imshow(img)

plt.axis('off')

plt.savefig('rendered\_state.png')

logger.info("Rendered state saved as 'rendered\_state.png'")

else:

raise ValueError(f"Unsupported render mode: {mode}")

def close(self):

self.netlogo.kill\_workspace()

def \_get\_observation(self):

obs = []

for y in range(self.grid\_size\_y):

for x in range(self.grid\_size\_x):

try:

green\_light\_up = int(self.netlogo.report(f'[green-light-up?] of one-of intersections with [my-row = {y} and my-column = {x}]'))

cars\_waiting\_ns = int(self.netlogo.report(f'count turtles-on patches with [pxcor = {x} \* grid-x-inc - floor(grid-x-inc - 1) and pycor = {y} \* grid-y-inc and pcolor = red]'))

cars\_waiting\_ew = int(self.netlogo.report(f'count turtles-on patches with [pycor = {y} \* grid-y-inc and pxcor = {x} \* grid-x-inc - floor(grid-x-inc - 1) and pcolor = red]'))

obs.append([green\_light\_up, cars\_waiting\_ns, cars\_waiting\_ew])

except Exception as e:

logger.error(f"Error getting observation for intersection ({x}, {y}): {str(e)}")

obs.append([0, 0, 0]) # Default values in case of error

return np.array(obs, dtype=np.int32)

def \_switch\_light(self, intersection\_id):

y = intersection\_id // self.grid\_size\_x

x = intersection\_id % self.grid\_size\_x

try:

self.netlogo.command(f'ask one-of intersections with [my-row = {y} and my-column = {x}] [ set green-light-up? not green-light-up? set-signal-colors ]')

except Exception as e:

logger.error(f"Error switching light at intersection ({x}, {y}): {str(e)}")

def test\_environment():

# Create an instance of the environment

env = NetLogoTrafficGridEnv()

# Reset the environment and get the initial observation

obs = env.reset()

logger.info("Initial Observation:")

logger.info(obs)

logger.info(f"Observation shape: {obs.shape}")

# Run a few steps with random actions

for i in range(5):

logger.info(f"\nStep {i + 1}")

# Generate a random action

action = env.action\_space.sample()

logger.info(f"Action: {action}")

# Take a step in the environment

obs, reward, done, info = env.step(action)

logger.info(f"Observation: {obs}")

logger.info(f"Reward: {reward}")

logger.info(f"Done: {done}")

logger.info(f"Info: {info}")

# Render the final state

env.render(mode='human')

# Close the environment

env.close()

if \_\_name\_\_ == "\_\_main\_\_":

test\_environment()