

Traffic Simulator

Aditya Singh

January 17, 2022

1 Environment Generation

This Traffic Simulator¹ aims to simulate real time traffic scenarios which can aid in the training of various autonomous vehicles. The environment is 2-dimensional and relies on the readily available pygame library in Python3, making it easy to use and not demanding on the user's system unlike some heavier simulation softwares like CARLA.

The user has to create his environments by following the given steps :

1. Simulate the road network
2. Simulate infinitely generated auto vehicles
3. Simulate single generated auto and ego vehicles

1.1 Simulate the road network

To first create our simulation, our user first has to create an object from the Simulation class. This object will hold all the necessary data of the roads, vehicles, traffic signs, etc.

```
1 sim = Simulation()
```

¹GitHub Repo : <https://github.com/aditya3434/TrafficSimulator>

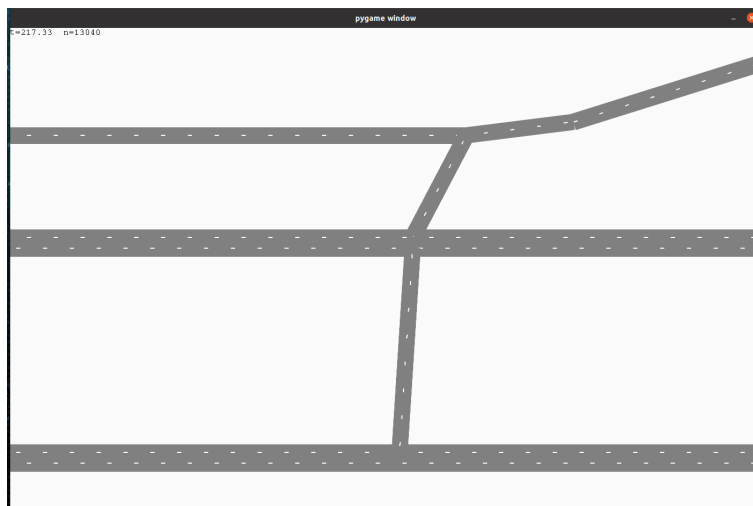
After which, the user will use the `create_roads()` function to enter an array of roads. Each road is defined by 2 (x, y) coordinates that signify the start and the end of the road respectively.

```
1  sim.create_roads([
2      ((300, 98), (0, 98)),
3      ((0, 102), (300, 102)),
4      ((180, 60), (0, 60)),
5      ((220, 55), (180, 60)),
6      ((300, 30), (220, 55)),
7      ((180, 60), (160, 98)),
8      ((0, 178), (300, 178)),
9      ((300, 182), (0, 182)),
10     ((160, 102), (155, 180)),
11     ((160, 98), (0, 98))
12 ])
```

To run this simulation, we create a window by instantiating an object of the `Window` class by giving the simulation object as an argument. After which the user can use the various methods of said class to create offsets, resize, run window, etc. We will discuss about the `run()` function in detail later.

```
1  sim = create_sim()
2  win = Window(sim)
3  win.offset = (-150, -110)
4  win.run(steps_per_update=5)
```

Running this code will give us the following output.



1.2 Simulate infinitely generated auto vehicles

As we all know, road traffic is always random and hundreds of vehicles are always entering and exiting different roads. Hence, to simulate such traffic, the user can use the `create_gen()` function to create a vehicle generator.

```
1  sim.create_gen({
2      'vehicle_rate': 10,
3      'vehicles': [
4          [1, {"path": [4, 3, 5, 9]}],
5          [1, {"path": [4, 3, 2]}],
6          [1, {"path": [1]}],
7          [1, {"path": [6]}],
8          [1, {"path": [7]}]
9      ]
10 })
```

The generator will keep spawning vehicles listed in the array at a speed dependant on the `vehicle_rate`. Each vehicles is depicted in the format `[1, ... "path": [A]]` where A is the array of roads that form the path the vehicle is gonna take. For example, if the 3rd vehicle spawns, it'll drive along road 1. Whereas if the 2nd vehicle spawns, it'll go along road 4, then 3 and then till the end of road 2. These vehicles are "auto" meaning that they follow predetermined paths and can't take acceleration and steer values as their input.

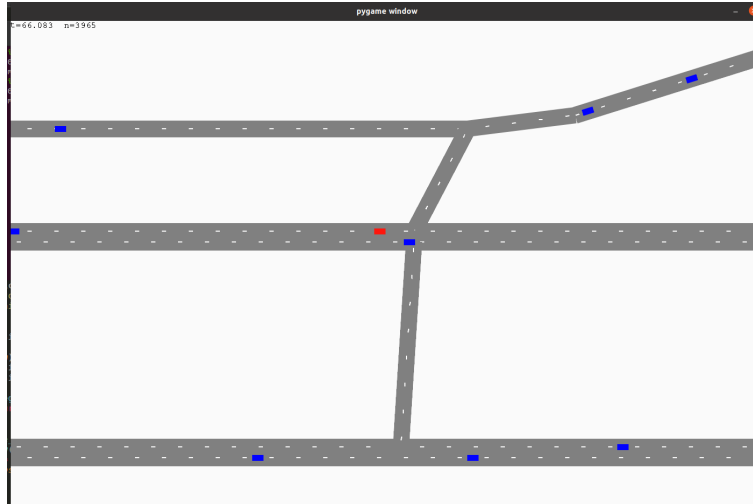
1.3 Simulate single generated auto and ego vehicles

While spawning common vehicles infinitely seems alright, there might be a scenario where only one car is present on the road. Or in other cases, we might want to put emphasis on an action/ego vehicle whose maneuver we want to customize and analyze. For these vehicles, we use the `create_gen()` function to create a single vehicle generator, which only spawns 1 vehicle.

```
1  ego_vehicle = sim.create-single-gen({
2      'auto' : False,
3      'vehicle': [1, {"acc": 10, "x": 0, "y": 98, "model": "...
4                      "Kinematic", "L": 2.5, "c_a": 2.0, "c_r": 0.01}]
5  })
```

The single vehicle spawned can be either "auto" or "ego". These vehicles can take many attributes as input like acceleration, steer, driving type, etc. to give

the car freedom of movement and can maneuver in the way the user wants. The figure below shows the vehicles (Auto : Blue, Ego : Red) running along the road network.



2 Play testing and Training Models

Now that we know how to simulate the environment, we observe how we can use this environment to play test certain scenarios as well as train different machine learning models. This would include defining rules in our environments, writing the step function, passing different arguments in the `run()` function, and many other steps that we will study.

2.1 Play testing scenarios

When we run the traffic simulator, we can see a visual representation of how the ego and the auto vehicles are driving on the road. However, we can observe that the vehicles have no interaction with each other and the environment. The user might want to declare certain rules that when broken, ends the simulation (e.g. collisions, off-roading, etc). To introduce these types of rules, the user can introduce certain play testing functions which they can pass as an argument to the `run()` function of the Window class.

For instance, if we want the simulation to end when ego vehicles collide, we can create the following function.

```
1 def function(sim):
2     ego1 = sim.action_vehicles[0]
3     ego2 = sim.action_vehicles[1]
4
5     # Check if the ego vehicles collide with each other or with ...
6     # any auto vehicles
7     if sim.ego_collide(ego1, ego2) or sim.auto_collide(ego1) or ...
8         sim.auto_collide(ego2):
9         return True
10
11 # Create simulation
12 sim = create_sim()
13
14 # Create window to display sim
15 win = Window(sim)
16 win.offset = (-150, -110)
17
18 # Run the simulation in the window with the test function
19 win.run(function, steps_per_update=5)
```

If the function returns True at any point, the simulation ends. This function can be modified to check for any other simulation ending conditions like off-roading and scenario timeouts.

2.2 Training ML models

The test function designed above can also be used as a step function to train ML models. In this case, we have to pass the step function, the episode number, the collected score and the model in our `run()` function.

Given below is the implementation of a DQN model to train the going straight maneuver.

```
1 # Discrete choices for our vehicle
2 def choose_action(choice):
3     if choice == 0:
4         return [0, 0]
5     elif choice == 1:
6         return [0.3, 0]
7     else:
8         return [0.8, 0]
9
10 # Agent step function
```

```

11 def step(sim, epi, loss, model, steps_per_update):
12
13     # Initializing reward and done state
14     reward = 0
15     done = False
16
17     # Ego vehicle which we are training
18     ego = sim.action_vehicles[0]
19
20     # X-coordinate of ego vehicle
21     x = ego.get_state()[1][0]
22
23     # Velocity of ego vehicle
24     vel = ego.get_state()[0]
25
26     # Current state
27     state = [x, vel]
28     state = np.reshape(state, (1,2))
29
30     # Model gives choice
31     choice = model.act(state)
32     action = choose_action(choice)
33     ego.set_state(action)
34
35     # Agent takes choice and updates environment
36     sim.run(steps_per_update)
37
38     # Analyzing new state
39     x = ego.get_state()[1][0]
40     vel = ego.get_state()[0]
41     next_state = [x, vel]
42     next_state = np.reshape(next_state, (1,2))
43
44     # Calculating step reward
45     reward -= (3-0.01*x)
46
47     # Model learning
48     model.remember(state, choice, reward, next_state, done)
49     model.replay(done, epi, loss)
50
51     # If ego vehicle reaches the end of the road, sim ends
52     if x > 299:
53         reward += 1000
54         done = True
55
56     return model, reward, done
57
58 # Keeping track of scores
59 loss = []
60
61 # DQN model for our autonomous vehicle
62 straight_model = md.DQN(3,2,'straight_model')
63 x = []
64
65 # No. of training episodes
66 episodes = 500
67

```

```

68 # Training Loop
69 for i in range(epochs):
70     # Create simulation
71     sim = create_sim()
72
73     # Create window to display simulation
74     win = Window(sim)
75     win.offset = (-150, -110)
76
77     # Get total score an updated model after simulation ends
78     score, straight_model = win.run(step, i, loss, ...
                                     straight_model, steps_per_update=4)
79
80     # Append total score to list
81     loss.append(score)
82     print("Episode : ", i+1, " --> Score : ", score)
83     x.append(i+1)
84
85 plt.plot(x, loss)
86 plt.savefig('train.png')

```