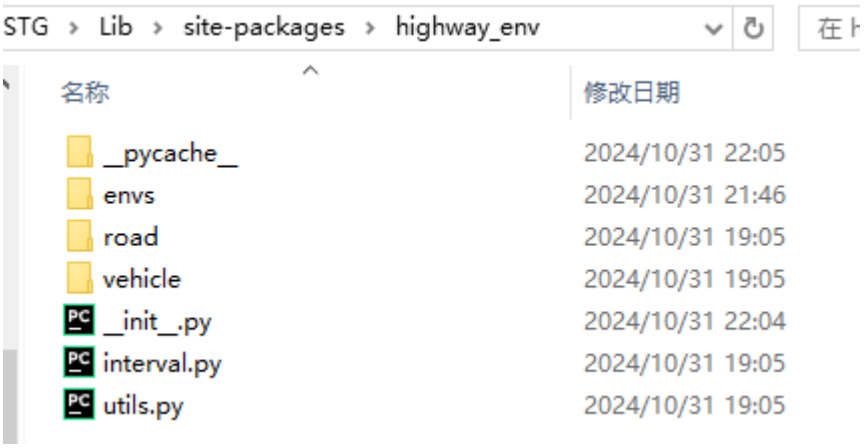
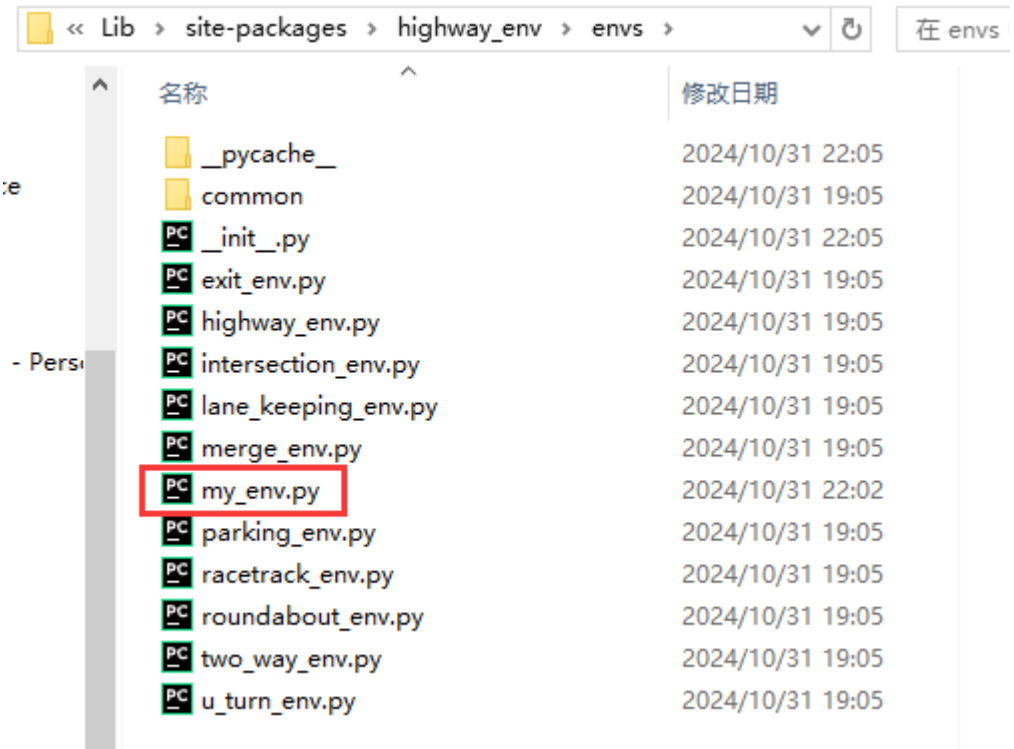
Create My Env

1. Install “highway_env” for python
2. Locate packages

```
(STG) F:\UCLA\209AS\final project>pip show highway_env
Name: highway-env
Version: 1.10.1
Summary: An environment for simulated highway driving tasks.
Home-page:
Author:
Author-email: Edouard Leurent <eleurent@gmail.com>
License: MIT License
Location: f:\anaconda3\envs\stg\lib\site-packages
Requires: farama-notifications, gymnasium, matplotlib, numpy, pandas, pygame, scipy
Required-by:
```



3. Create a new python file in ‘highway_env/envs’



4. Define ‘MyEnv’ class (must inherit from AbstractEnv), can be easily defined by copying other env class in other python files.

```python
class MyEnv(AbstractEnv):
    ACTIONS: dict[int, str] = {0: "SLOWER", 1: "IDLE", 2: "FASTER"}
    ACTIONS_INDEXES = {v: k for k, v in ACTIONS.items()}
```

5. In file ‘highway_env/envs/_init_.py’, add code:

```python
# "Below is our environment."
from highway_env.envs.my_env import MyEnv
```

6. In file ‘highway_env/_init_.py’, add code in ‘def _register_highway_envs()’

```python
def _register_highway_envs():
    """Import the envs module so that envs register themselves."""

    from highway_env.envs.common.abstract import MultiAgentWrapper

    # my_env.py
    register(
        id='my_env-v0',
        entry_point='highway_env.envs:MyEnv',
    )

    # exit_env.py
    register(
        id="exit-v0",
        entry_point="highway_env.envs.exit_env:ExitEnv",
    )
```

7. Use MyEnv Now!!

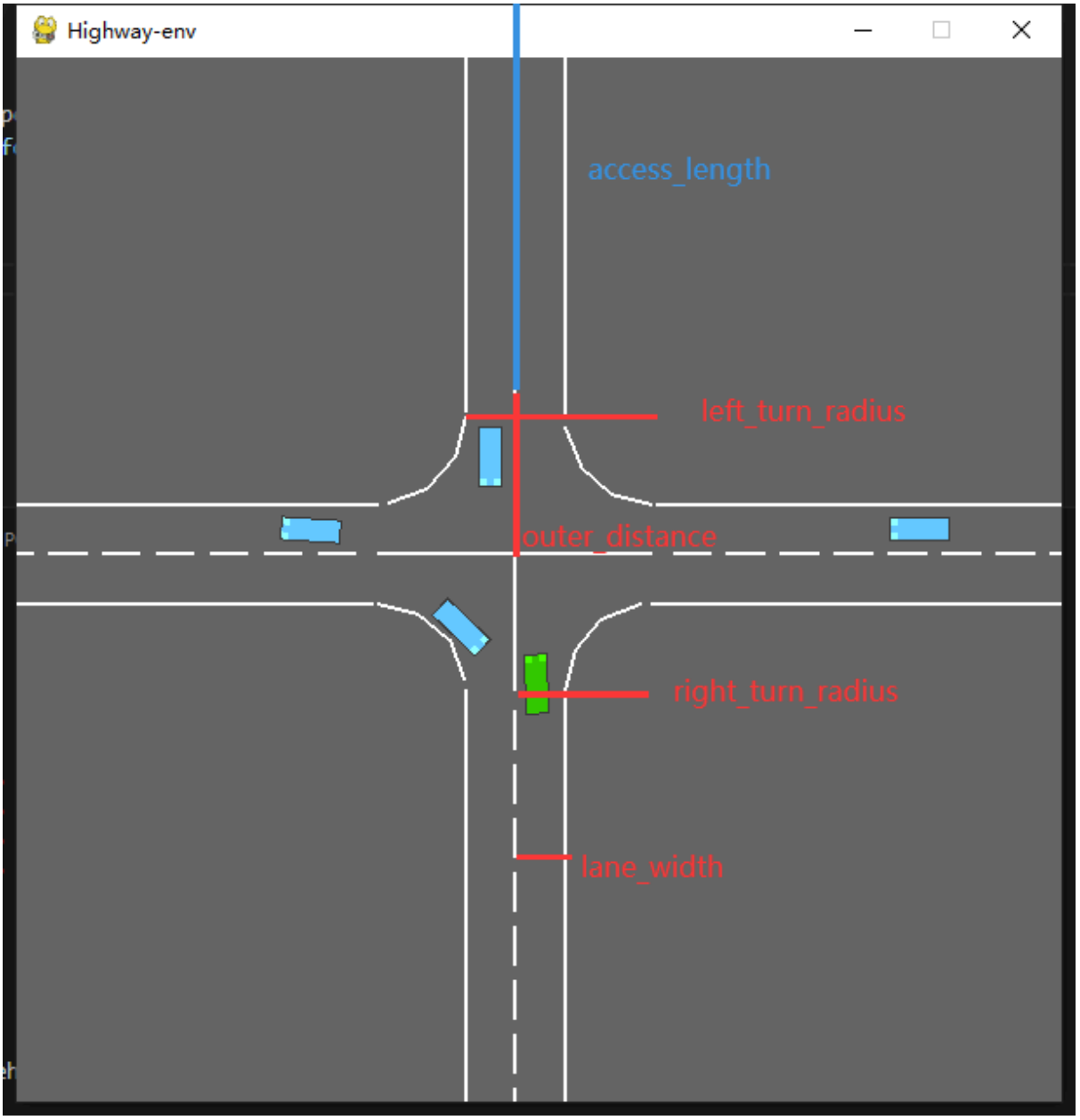
```python
import gymnasium
import pprint
import highway_env
from matplotlib import pyplot as plt

env = gymnasium.make('my_env-v0', render_mode='rgb_array')
env.reset()
pprint.pprint(env.unwrapped.config)
for _ in range(30):
    action = env.unwrapped.action_type.actions_indexes["IDLE"]
    obs, reward, done, truncated, info = env.step(action)
    env.render()

plt.imshow(env.render())
plt.show()
```

Modify My Env(Intersection)

1. length parameter define



2. make road

code may need to be modified is label by “TODO” in comment

```
142     def _make_road(self) -> None:
143
144         lane_width = AbstractLane.DEFAULT_WIDTH     # DEFAULT_WIDTH = 4
145
146         right_turn_radius = num_lanes * lane_width + 5      # radius of the left lane
147
148         left_turn_radius = right_turn_radius + lane_width   # radius of the left lane
149
150         outer_distance = right_turn_radius + lane_width / 2
151         access_length = 50 + 50
152
```

As for the first part “Incoming type”, we need to define a new start point & end point name based on corner and lane index.
Moreover, consider different line types according to the lane index.
Same as the other parts ……

```
        # Incoming lanes
        start = rotation @ np.array(
            [lane_offset + lane_width / 2, access_length + outer_distance]
        )
        end = rotation @ np.array([lane_offset + lane_width / 2, outer_distance])

        # Judge line type
        if num_lanes == 1:
            line_type = [c, c]
        elif lane_index == 0:
            line_type = [c, s]
        elif lane_index < num_lanes - 1:
            line_type = [s, s]
        else:
            line_type = [s, c]

        net.add_lane(
            # TODO: maybe need to add str(lane_index) to start&end point name
            "o" + str(corner) + "_" + str(lane_index),
            "ir" + str(corner) + "_" + str(lane_index),
            StraightLane(
                # TODO: judge to determine line_types, not just left-s, right-c
                start, end, line_types=line_type, priority=priority, speed_limit=10
            ),
        )
```

3. _make_vehicle
consider lane index in road navigation

```
335         # Controlled vehicles
336         self.controlled_vehicles = []
337         for ego_id in range(0, self.config["controlled_vehicles"]):
338             lane = self.np_random.integers(1, num_lanes)
339             ego_lane = self.road.network.get_lane(
340                 (f"o{ego_id % 4}_{lane}", f"ir{ego_id % 4}_{lane}", 0)
341             )
342
343             # destination = self.config["destination"] or "o" + str(self.np_random.i
344             destination = self.config["destination"] + f"_{lane}"
345
```

4. _spawn_vehicle
consider lane index in road navigation

```
382
383         route = self.np_random.choice(range(4), size=2, replace=False)
384         route[1] = (route[0] + 2) % 4 if go_straight else route[1]
385         lane = self.np_random.choice(range(num_lanes), size=2, replace=False)
386
387         vehicle_type = utils.class_from_path(self.config["other_vehicles_type"])
388         vehicle = vehicle_type.make_on_lane(
389             self.road,
390             (f"o{route[0]}_{lane[0]}", f"ir{route[0]}_{lane[0]}", 0),
391             longitudinal=(
392                 longitudinal + 5 + self.np_random.normal() * position_deviation
393             ),
394             speed=8 + self.np_random.normal() * speed_deviation,
395         )
```

Result: