Final Project Tips

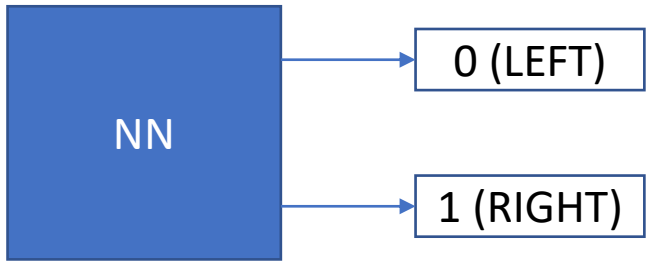
Observation

- Gray scale
- Resize
- Frame stack
- …

Discretize Action Space

- The actions in the environment: (motor, steering)
- If the continuous action space is too difficult to train, you can let your model learn some predefined discrete actions

Action id	(motor, steering)
0	(1.0, -1.0)
1	(1.0, 1.0)



Init Mode: random

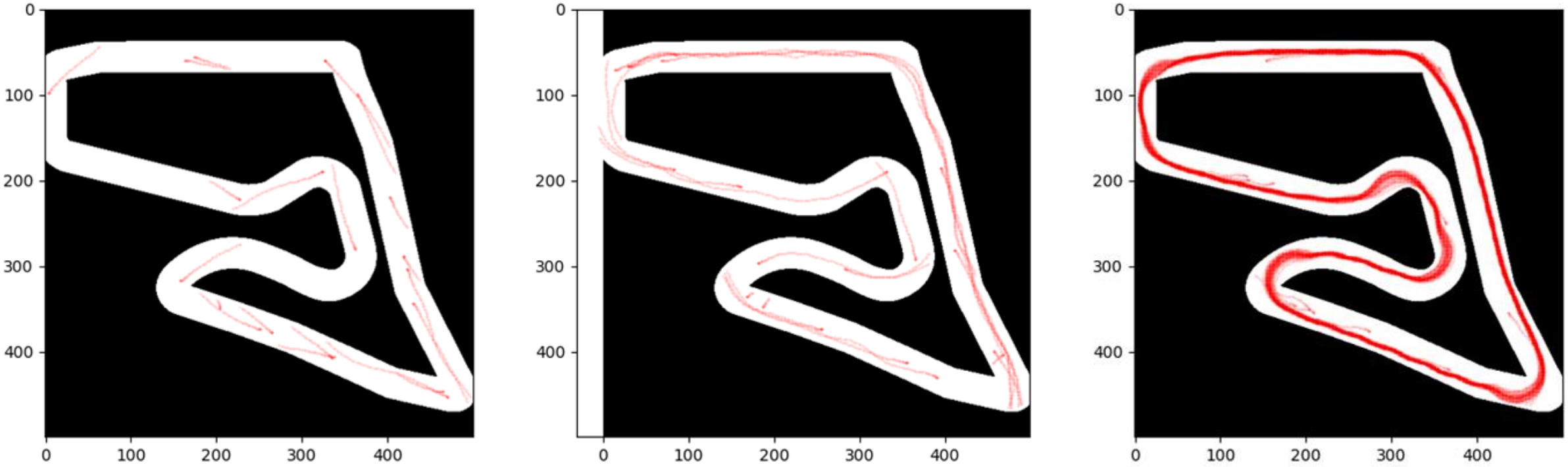
- class RaceEnv

```python
def reset(self, *args, **kwargs: dict):
    if kwargs.get('options'):
        kwargs['options']['mode'] = 'random'
    else:
        kwargs['options'] = {'mode': 'random'}
    self.cur_step = 0
    obs, *others = self.env.reset(*args, **kwargs)
    obs = self.observation_postprocess(obs)
    return obs, *others
```

- kwargs['options']['mode']= 'random'
  - randomly start from some different points

Init Mode: random

- kwargs['options']['mode']= random



拿地圖： models/scenes/austria_competition/maps/maps.npz

Reward Shaping

- Default reward: difference of progress
- You can use other information from the env to define reward
  - e.g. wall_collision, acceleration, velocity, progress, obstacle, …

Settings of Scenarios

- In directory: scenarios
  - austria_competition_collisionStop.yml
  - circle_cw_competition_collisionStop.yml
  - austria_competition.yml
  - circle_cw.yml

```yaml
world:
  name: austria_competition
agents:
  - id: A
    vehicle:
      name: racecar_competition
      actuators: [ motor_competition, steering_competition ]
      sensors: [ camera_competition ]
    task:
      task_name: maximize_progress_collision_time_reduce
      params: {
        laps: 99999999999,
        time_limit: 100.0, # <---
        terminate_on_collision: False, # <---
        collision_reward: 0.0,
        progress_reward: 1.0,
        frame_reward: 0.0,
      }
```

scenarios/austria_competition.yml

Settings of Scenarios: task

- The task we use is defined in racecar_gym/tasks/progress_based.py
  - Class: MaximizeProgressTaskCollisionInfluenceTimeLimit
    - Calculate the reward
    - Check the terminate condition

```yaml
world:
  name: austria_competition
agents:
  - id: A
    vehicle:
      name: racecar_competition
      actuators: [ motor_competition, steering_competition ]
      sensors: [ camera_competition ]
    task:
      task_name: maximize_progress_collision_time_reduce
      params: {
        laps: 99999999999,
        time_limit: 100.0, # <---
        terminate_on_collision: False, # <---
        collision_reward: 0.0,
        progress_reward: 1.0,
        frame_reward: 0.0,
      }
```

scenarios/austria_competition.yml

Settings of Scenarios: reward

- You can modify three types of reward
- The total reward is calculated in the task

```yaml
world:
  name: austria_competition
agents:
  - id: A
    vehicle:
      name: racecar_competition
      actuators: [ motor_competition, steering_competition ]
      sensors: [ camera_competition ]
    task:
      task_name: maximize_progress_collision_time_reduce
      params: {
        laps: 99999999999,
        time_limit: 100.0, # <---
        terminate_on_collision: False, # <---
        collision_reward: 0.0,
        progress_reward: 1.0,
        frame_reward: 0.0,
      }
```

scenarios/austria_competition.yml

```python
def reward(self, agent_id, state, action) -> float:
    agent_state = state[agent_id]
    progress = agent_state['lap'] + agent_state['progress']
    if self._last_stored_progress is None:
        self._last_stored_progress = progress
    delta = abs(progress - self._last_stored_progress)
    if delta > .5:  # the agent is crossing the starting line in the wrong direction
        delta = (1 - progress) + self._last_stored_progress
    reward = self._frame_reward
    if self._check_collision(agent_state):
        self.n_collision += 1
        reward += self._collision_reward
    reward += delta * self._progress_reward
    self._last_stored_progress = progress
    return reward
```

racecar_gym/tasks/progress_based.py

Settings of Scenarios: terminate condition

- Collision or time limit exceeded
- The scenarios: austria_competition and circle_cw CANNOT be used with reset_when_collision=False, or the car will stuck
  - Use austria_competition_collisionStop and circle_cw_competition_collisionStop with reset_when_collision=False

```python
def done(self, agent_id, state) -> bool:
    agent_state = state[agent_id]
    if self._terminate_on_collision and self._check_collision(agent_state):
        return True
    # Collision reduce the time limit
    total_penalty = sum(agent_state['collision_penalties'])
    lap_done = agent_state['lap'] > self._laps
    time_done = (self._time_limit - total_penalty) < agent_state['time']
    return lap_done or time_done
```

racecar_gym/tasks/progress_based.py