1) Python 코드(reward function)

```
  import math

MODE = "shortCut"  # 모드를 shortcut으로 설정
MAX_SIGHT = 1.0 # 최대 시야


def dist(point1, point2): # 두 point간 거리 구하기
    return ((point1[0] - point2[0]) ** 2 + (point1[1] - point2[1]) **
2) ** 0.5


# thanks to https://stackoverflow.com/questions/20924085/python-
conversion-between-coordinates
def rect(r, theta): #극좌표를 직각좌표로 변환
    """
    theta in degrees
    returns tuple; (float, float); (x,y)
    """

    x = r * math.cos(math.radians(theta))
    y = r * math.sin(math.radians(theta))
    return x, y


# thanks to https://stackoverflow.com/questions/20924085/python-
conversion-between-coordinates
def polar(x, y): #직각 좌표를 극좌표 변환
    """
    returns r, theta(degrees)
    """

    r = (x ** 2 + y ** 2) ** .5
    theta = math.degrees(math.atan2(y, x))
    return r, theta
```

```
def angle_mod_360(angle): # 각도를 [-180,180]범위 내로 조정
    """
    Maps an angle to the interval -180, +180.
    Examples:
    angle_mod_360(362) == 2
    angle_mod_360(270) == -90
    :param angle: angle in degree
    :return: angle in degree. Between -180 and +180
    """

    n = math.floor(angle / 360.0) # 정수반환

    angle_between_0_and_360 = angle - n * 360.0

    if angle_between_0_and_360 <= 180.0:
        return angle_between_0_and_360
    else:
        return angle_between_0_and_360 - 360


def get_waypoints_ordered_in_driving_direction(params):
    if MODE == "shortCut":
        waypoints = get_shortcut_waypoints()#shortcut mode이면 아래
불러온 shortcut waypoint를 따른다
    else:
        waypoints = params['waypoints']#아니면 기본 waypoint 사용

    # waypoints are always provided in counter clock wise order
    if params['is_reversed']:  # driving clock wise. #시계방향 주행시
waypoint list reverse 시킨다.
        return list(reversed(params['waypoints']))
    else:  # 반시계방향 주행
        return waypoints


def up_sample(waypoints, factor): #주어진 waypoint 사이waypoint
추가
    """
    Adds extra waypoints in between provided waypoints
```

```
    :param waypoints:
    :param factor: integer. E.g. 3 means that the resulting list has 3 times as many points.
    :return:
    """
    p = waypoints
    n = len(p)

    return [[i / factor * p[(j + 1) % n][0] + (1 - i / factor) * p[j][0],
             i / factor * p[(j + 1) % n][1] + (1 - i / factor) * p[j][1]] for j in range(n) for i in range(factor)]


def get_target_point(params): # 목표점 설정
    # 최대 0.9만큼 보게함.

    waypoints = up_sample(get_waypoints_ordered_in_driving_direction(params), 20)

    car = [params['x'], params['y']]# 차의 위치

    distances = [dist(p, car) for p in waypoints] #waypoints와 현재 차의 거리를 담은 list
    min_dist = min(distances) #가장 가까운 waypoint와 현재 차 간의 거리
    i_closest = distances.index(min_dist) #최소 거리의 인덱스

    n = len(waypoints)

    waypoints_starting_with_closest = [waypoints[(i + i_closest) % n] for i in range(n)] #차의 현재 위치를 기준으로 가까운 순서대로 waypoint list 재 정렬

    if MODE == "shortCut": #shortcut mode시 sight 축소
        sight = MAX_SIGHT * 0.5
    else:
        sight = MAX_SIGHT

    r = params['track_width'] * sight #차의 시야 범위에 사용할 반지름
```

정의

```
    is_inside = [dist(p, car) < r for p in
waypoints_starting_with_closest]#반지름 내의 waypoints
    i_first_outside = is_inside.index(False)#반지름 밖의 첫번째
waypoint

    if i_first_outside < 0:  #  r이 너무 큰 값일 경우 i_first_outside 가
존재x
        return waypoints[i_closest] # 가장 가까운 waypoint 출력

    return waypoints_starting_with_closest[i_first_outside]


def get_target_steering_degree(params): # target 각도 구하기
    tx, ty = get_target_point(params) # 목표점의 좌표
    car_x = params['x'] #현재 차의 좌표
    car_y = params['y']
    dx = tx - car_x # 두 점의 차이
    dy = ty - car_y
    heading = params['heading'] #x축에 대한 차의 진행 각도

    _, target_angle = polar(dx, dy)

    steering_angle = target_angle - heading

    return angle_mod_360(steering_angle) # final unit: degree


# reinvent2019: get_shortcut_waypoints
def get_shortcut_waypoints():
    return [
        [0.63069109, 2.80611932],
        [0.63367125, 2.69079621],
        [0.6467188, 2.57569291],
        [0.66972231, 2.46183988],
        [0.70251506, 2.35022569],
        [0.74487589, 2.24177514],
        [0.79652923, 2.1373277],
```

```
[0.85714459, 2.03761659],
[0.92633571, 1.94324872],
[1.00365975, 1.85468625],
[1.08861721, 1.77223051],
[1.1806537, 1.69600972],
[1.27916562, 1.6259728],
[1.38351227, 1.56189156],
[1.4930358, 1.50337335],
[1.60708637, 1.44988322],
[1.72504192, 1.40077175],
[1.84630443, 1.35530161],
[1.97025603, 1.31266612],
[2.09617545, 1.2719922],
[2.2231517, 1.23232374],
[2.35576976, 1.190681],
[2.48814156, 1.14836059],
[2.62010372, 1.1049143],
[2.75155515, 1.06006668],
[2.88245693, 1.01371411],
[3.01283929, 0.96594252],
[3.14278403, 0.91697795],
[3.27239538, 0.86710616],
[3.40178871, 0.81664194],
[3.52534292, 0.76798582],
[3.64882806, 0.72098717],
[3.77225054, 0.67657779],
[3.89565744, 0.63576533],
[4.01912628, 0.59935302],
[4.14273194, 0.56802807],
[4.26652265, 0.54240254],
[4.390508, 0.52303129],
[4.5146562, 0.51041311],
[4.63889641, 0.50498126],
[4.76312271, 0.50708933],
[4.88719857, 0.51699788],
[5.01096146, 0.53486531],
[5.13422789, 0.56074489],
[5.25679889, 0.59458804],
[5.3784661, 0.63625275],
```

```
[5.49901791, 0.68551547],
[5.61824556, 0.74208505],
[5.73594876, 0.8056172],
[5.85194051, 0.87572882],
[5.96605088, 0.95201138],
[6.0781297, 1.03404317],
[6.18804798, 1.12140005],
[6.29569809, 1.21366463],
[6.40099292, 1.31043383],
[6.50386394, 1.41132475],
[6.60425847, 1.51597899],
[6.70213638, 1.62406543],
[6.79746619, 1.73528168],
[6.89022097, 1.84935431],
[6.980374, 1.96603801],
[7.06789434, 2.08511385],
[7.15274241, 2.20638682],
[7.23486553, 2.32968268],
[7.31419351, 2.4548443],
[7.39063436, 2.58172754],
[7.46407004, 2.71019664],
[7.5343525, 2.84011927],
[7.60129994, 2.9713612],
[7.6646937, 3.10378064],
[7.72427579, 3.2372224],
[7.77974745, 3.37151184],
[7.83076905, 3.50644891],
[7.87696157, 3.64180249],
[7.91791, 3.77730515],
[7.9531689, 3.91264892],
[7.98227014, 4.0474821],
[8.00473301, 4.18140776],
[8.02007634, 4.31398396],
[8.02783246, 4.44472614],
[8.02756242, 4.57311169],
[8.01887186, 4.69858671],
[8.00142678, 4.82057488],
[7.97496831, 4.93848799],
[7.9393258, 5.05173771],
```

```
[7.89442733, 5.15974804],
[7.84030706, 5.26196766],
[7.77710908, 5.35788145],
[7.7050874, 5.44702069],
[7.62460227, 5.52897101],
[7.536113, 5.60337802],
[7.44016781, 5.66995014],
[7.33739134, 5.72845868],
[7.22847048, 5.77873548],
[7.11413941, 5.82066844],
[6.99516423, 5.85419554],
[6.87232807, 5.8792982],
[6.74641682, 5.89599457],
[6.61820582, 5.90433369],
[6.48844757, 5.90439106],
[6.35786039, 5.89626615],
[6.22711782, 5.88008231],
[6.09683868, 5.85598899],
[5.96757762, 5.82416624],
[5.83981608, 5.78483111],
[5.71395379, 5.73824529],
[5.59030113, 5.68472331],
[5.46907261, 5.62464036],
[5.35038212, 5.55843893],
[5.23424046, 5.48663315],
[5.12055594, 5.40981015],
[5.00913994, 5.32862503],
[4.89972666, 5.24377382],
[4.79201344, 5.15593375],
[4.6857631, 5.06560545],
[4.5807927, 4.97315917],
[4.48081878, 4.88243357],
[4.37959748, 4.79420328],
[4.27684354, 4.70906838],
[4.17224941, 4.62769566],
[4.06553767, 4.55071821],
[3.95644174, 4.47877854],
[3.84474817, 4.41244104],
[3.73031118, 4.35215634],
```

```
        [3.61306059, 4.29823493],
        [3.49300409, 4.25082828],
        [3.37022468, 4.20991786],
        [3.24487511, 4.17530971],
        [3.1171707, 4.14663361],
        [2.98738247, 4.12334422],
        [2.85585139, 4.10466958],
        [2.72282052, 4.09005797],
        [2.58852158, 4.07896026],
        [2.45315288, 4.07089104],
        [2.31675601, 4.06579649],
        [2.18425079, 4.05787239],
        [2.05356546, 4.04652377],
        [1.9251617, 4.03105575],
        [1.79950825, 4.01084184],
        [1.67721911, 3.98511327],
        [1.55873527, 3.9534865],
        [1.44466638, 3.91541352],
        [1.33564671, 3.8704324],
        [1.23222981, 3.81830074],
        [1.13519251, 3.7586213],
        [1.04519953, 3.69128277],
        [0.96285535, 3.61637202],
        [0.88870748, 3.5341512],
        [0.82324005, 3.44504611],
        [0.76686864, 3.34963169],
        [0.71993756, 3.24861421],
        [0.68271928, 3.1428114],
        [0.65541543, 3.03313123],
        [0.63815905, 2.92055024],
        [0.63069109, 2.80611932]
    ]


def score_steer_to_point_ahead(params):

    # track 범위 나누기
    marker_1 = 0.1 * params['track_width']
    marker_2 = 0.2 * params['track_width']
```

```
    marker_3 = 0.3 * params['track_width']
    marker_4 = 0.4 * params['track_width']

    # parameter 불러오기
    distance_from_center = params['distance_from_center']
    speed = params['speed']
    all_wheels_one_track = params['all_wheels_on_track']
    best_stearing_angle = get_target_steering_degree(params) # ideal
value
    steering_angle = params['steering_angle'] # current value

    SPEED_THRESHOLD = 1.0
    # 기본 STEERING_THRESHOLD 가 20으로 설정
    ABS_STEERING_THRESHOLD = 20

    error = (steering_angle - best_stearing_angle) / 60.0  # 현재
각도와 이상적인 각도 값의 차이를 통해 error 구하기. 60도 이상 차이가
나면 큰 error로 간주하여 60도를 최대값으로 본다

    score = 1.0 - abs(error)

    if score <= 0:
        reward = 1e-3
    else:  # score > 0
        if distance_from_center <= marker_1:
            reward = 1
            ABS_STEERING_THRESHOLD = 5
            SPEED_THRESHOLD = 2.0
        elif distance_from_center <= marker_2:
            reward = 0.7
            ABS_STEERING_THRESHOLD = 10
            SPEED_THRESHOLD = 1.0
        elif distance_from_center <= marker_3:
            reward = 0.5
            ABS_STEERING_THRESHOLD = 20
            SPEED_THRESHOLD = 0.5
        elif distance_from_center <= marker_4:
            reward = 0.1
            ABS_STEERING_THRESHOLD = 30
```

```
            SPEED_THRESHOLD = 0.5
        else:
            reward = 1e-3
            ABS_STEERING_THRESHOLD = 40
            SPEED_THRESHOLD = 0.1

    # reward에 따른 score 갱신
    score *= reward

    # Penalize reward if the car is steering too much
    if abs(steering_angle) > ABS_STEERING_THRESHOLD:  # 차가
너무 기울어져있으면
        score *= 0.5

    if not all_wheels_one_track:  # Penalize if the car goes off track
        score *= 1e-3
    elif speed < SPEED_THRESHOLD:  # Penalize if the car goes too
slow
        score *= 0.5
    else:  # High reward if the car stays on track and goes fast
        score *= 1.0

    # give a reward at each time of the completing a lap
    if params['progress'] == 100: # 완주율
        score += 10000

    return max(score, 0.01)  # optimizer is rumored to struggle with
negative numbers and numbers too close to zero



def reward_function(params):
    return float(score_steer_to_point_ahead(params))
```

2) 코드 설명

- reward 방식: score와 reward 두 가지 변수를 곱하여 최종 보상을 결정한다. 결정된 reward는 상수 0.01과 max함수를 통과하여 reward가 아주 작더라도 0.01점의 보상을 보장한다. 이는 차량이 생존에 대한 기본적 보상을 받을 수 있게 하기 위함이다.
- score 변수: 주행하는 차량의 speed와 angle에 대한 보상이다. 보상을 나누는 기준은 각 상황에 따른 threshold값이다. 속도에 대한 threshold 값으로 SPEED_THRESHOLD와 (default = 1.0) 각도에 대한 threshold 값으로 ABS_STEERING_THRESHOLD (default = 20) 전역변수를 사용한다. 이 두 변수는 차량이 중앙선으로 부터 떨어진 거리에 따라 매 순간 결정된다.
- get_shortcut_waypoints 함수: 실제 track의 waypoint가 아닌 2019track의 경우 보다 최적의 경로를 설정하고 이 경로의 waypoint를 새로 지정한 값을 출력하는 함수다. 이는 ShortCut Mode에서 불러올 수 있다.
- waypoints_starting_with_closest: waypoints의 절대적 위치가 아닌 차량의 위치에 대해 상대적인 way points를 계산하여 현재 차량의 위치와 가까운 순으로 재구성한 list변수.
- angle 결정: waypoint에 따른 미분치를 target_angle로 설정하고 차량의 heading과의 차이를 극좌표로 계산 후 degree로 변환하여 streering_angle로 설정한다. streering_angle과 best_streering_angle의 차이가 60도일 때를 worst로 보고 각의 차이가 60도로 나누었을 때
  값이 1 이상이면 최소의 reward 값을 준다. 그 외의 상황에서는 중앙선과의 거리차이를 기준으로 reward 값을 받는다.
  ShortCut Mode: shortcut Mode 선언 시 불러온 최단거리 point를 따라가게 설정하고 max_sight에 0.5배하여 sight를 정의 후 track_width에 sight를 곱해 반지름 크기를 설정한다.
- up_sample 함수: 원래 waypoints(또는 get_shortcut_waypoints에서 불러온 waypoints)들이 나타내는 경로를 더욱 자잘하게 쪼개어 up sampling하는 함수다.
- reward 변수: 주행하는 차량의 위치를 중앙선까지 떨어진 거리를 기준으로 4 단계로 나누어 보상을 준다.

- 모든 angle의 단위는 degree이며, 극좌표로 변환하거나 연산을 위해 radian단위를 사용하기 위한 함수는 따로 구현했다.

3) 조원 간 참여율

이름	참여 내용	참여율
윤수연	experience 설계	100%
노민종	experience 후 graph 등 결과 분석	100%
윤지수	log analysis 및 개선점 제안	100%
장동훈	reward function 작성 및 experience	100%