Custom Reward Function 만들기

Custom Reward Function 만들기

- **지금까지 살펴본 보상 함수의 변수들을 활용하여 직접 보상 함수를 설계할 수 있습니다.**
  - 보상 함수는 다음과 같은 형식을 따릅니다.

```
def reward_function(params) :
    reward = ...
    return float(reward)
```

Custom Reward Function 만들기

- 함수의 인자로 전달되는 params을 통해 앞서 살펴본 변수들을 함수 내부에서 사용할 수 있습니다.

```
def reward_function(params) :
    speed = params['speed']

    reward = ...

    return float(reward)
```

Custom Reward Function 만들기

- **모든 변수를 다 사용할 필요는 없습니다.**
  - 함수에서 사용하고자 하는 변수들만 사용하기 편리하게 저장하여 사용할 수 있습니다.

Custom Reward Function 만들기

- **보상 함수를 설계하는 기본적인 아이디어는 '어떤 상태가 되면 보상을 주겠다' 혹은 '어떤 상태가 되면 패널티를 주겠다'가 될 수 있습니다.**
  - DeepRacer는 주어진 환경(트랙)에서 학습하는 동안 특정 상태에서 행동을 선택합니다.
  - 그리고 그 행동은 상태를 변화시킴과 동시에 보상을 받게 됩니다. 따라서 보상 함수가 적용되는 시기는 한 번의 경험이 끝나는 시점이라고 볼 수 있습니다.

Custom Reward Function 만들기
- 빠르게 달릴수록 보상을 받게 하자

- DeepRacer의 속도가 빠르기를 원한다면 빠르게 달릴 때 보상을 크게 주고 느리게 달릴 때 보상을 적게 주는 보상 함수를 설계할 수 있습니다.

```
def reward_function(params) :
  speed = params['speed']

  if speed > 2 :
    reward = 10
  else :
    reward = 1

  return float(reward)
```

Custom Reward Function 만들기
- 빠르게 달릴수록 보상을 받게 하자

- **보상을 얼만큼을 주어야 하는지는 정해져 있지 않습니다.**
  - 특정 조건에서 보상의 차이를 만들 때, 보상의 차이가 매우 크다면 더 빠르게 큰 보상을 받는 행동을 찾아갈 수 있습니다.
  - 하지만 천천히 안정적으로 충분한 탐험을 통해 최적화를 하고 싶다면 차이를 작게 만들 수 있습니다.

Custom Reward Function 만들기
- 빠르게 달릴수록 보상을 받게 하자

- 위와 같이 보상함수를 설계한다면 학습하는 동안 빠른 속도로 움직일 수 있는 정책은 만들 수 있지만, 트랙 이탈에 대한 보상이 없기 때문에 트랙을 계속해서 이탈하게 됩니다.

Custom Reward Function 만들기
- 트랙을 이탈하지 않으면 보상을 받게 하자

- 앞서 보상함수 예시를 통해 트랙을 이탈하지 않을 수 있는 2가지 방법을 살펴봤습니다.
- 첫번째는 모든 바퀴가 트랙을 벗어나지 않았을 때 보상을 주는 방법입니다.
  - 이 방법은 all_wheels_on_track변수를 사용합니다.

```
def reward_function(params) :
  speed = params['speed']
  all_wheels_on_track = params['all_wheels_on_track']

  if all_wheels_on_track :
    if speed > 2 :
      reward = 10
    else :
      reward = 1
  else :
    reward = 0.01

  return float(reward)
```

Custom Reward Function 만들기 - 트랙을 이탈하지 않으면 보상을 받게 하자

- 단순히 속도가 빠를 때 보상을 주는 것이 아니라 모든 바퀴가 트랙 안에 있으면서 속도가 빠르면 보상이 주어지도록 설계되었습니다.

Custom Reward Function 만들기
- 트랙을 이탈하지 않으면 보상을 받게 하자

- 두번째 방법은 트랙의 중앙을 계산하고 트랙의 중앙에 가까울수록 보상을 높게 주는 방법입니다.
  - 이 방법은 track_width와 distance_from_center변수를 사용합니다.

```
def reward_function(params) :
  speed = params['speed']
  track_width = params['track_width']
  distance_from_center = params['distance_from_center']

  if distance_from_center <= track_width * 0.5 :
    if speed > 2 :
      reward = 10
    else :
      reward = 1
  else :
    reward = 1

  return float(reward)
```

Custom Reward Function 만들기
- 트랙을 이탈하지 않으면 보상을 받게 하자

- **트랙의 중앙과 DeepRacer 중앙까지의 거리가 트랙 너비의 50% 이하일 때 보상을 높게 주도록 설계되었습니다.**
  - 트랙 너비의 50%보다 크면 트랙을 이탈했다는 의미입니다.
  - 만약 트랙의 중앙에 좀 더 가깝게 주행하기를 원한다면 범위를 좁힐 수 있습니다.

Custom Reward Function 만들기
- 트랙을 이탈하지 않으면 보상을 받게 하자

- 트랙의 중앙과 DeepRacer 중앙까지의 거리가 트랙 너비의 10% 이하일 때 가장 높은 보상을 받도록 설계되었습니다.

```
def reward_function(params) :
  speed = params['speed']
  track_width = params['track_width']
  distance_from_center = params['distance_from_center']

  if distance_from_center <= track_width * 0.1 :
    if speed > 2 :
      reward = 10
    else :
      reward = 1
  elif distancd_from_center <= track_width * 0.5 :
    reward = 1
  else :
    reward = 0.01

  return float(reward)
```

Custom Reward Function 만들기 - 트랙의 중앙선 각도와 DeepRacer의 방향이 비슷할수록 보상을 받게 하자

- **이번에는 waypoints 변수를 활용하여 현재 트랙 중앙선의 각도를 계산하고, DeepRacer와 방향과 차이가 적을수록 보상을 크게 받게 하는 보상함수를 설계해보겠습니다.**
  - 이 방법은 직선 구간에서 불필요한 각도 조절을 최소화할 수 있고, 회전 구간에서 과도하게 각도를 조절하여 트랙을 이탈하는 것을 어느 정도 방지할 수 있습니다.

Custom Reward Function 만들기 - 트랙의 중앙선 각도와 DeepRacer의 방향이 비슷할수록 보상을 받게 하자

- 트랙의 waypoint들의 위치를 담은 waypoints 변수와 DeepRacer와 가까운 waypoint의 인덱스인 closest_waypoints, DeepRacer의 진행 방향을 알 수 있는 heading 변수를 사용합니다.

```
def reward_function(params) :
  waypoints = params['waypoints']
  closest_waypoints = params['closest_waypoints']
  heading = params['heading']

  reward = ...

  return float(reward)
```

Custom Reward Function 만들기 - 트랙의 중앙선 각도와 DeepRacer의 방향이 비슷할수록 보상을 받게 하자

- DeepRacer와 가장 가까운 두 개의 waypoint를 활용하여 현재 DeepRacer가 주행하고 있는 위치에서의 트랙 각도를 계산하고자 합니다.
  - 우선 가까운 waypoint를 각각 다른 변수에 저장하겠습니다.

```
def reward_function(params) :
  waypoints = params['waypoints']
  closest_waypoints = params['closest_waypoints']
  heading = params['heading']

  next_point = waypoints[closest_waypoints[1]]
  prev_point = waypoints[closest_waypoints[0]]

  reward = ...

  return float(reward)
```

Custom Reward Function 만들기 - 트랙의 중앙선 각도와 DeepRacer의 방향이 비슷할수록 보상을 받게 하자

- 두 waypoint의 좌표를 활용하여 각도를 계산합니다.
  - math라이브러리의 arctan함수를 사용하였습니다.

```python
import math
def reward_function(params) :
  waypoints = params['waypoints']
  closest_waypoints = params['closest_waypoints']
  heading = params['heading']

  next_point = waypoints[closest_waypoints[1]]
  prev_point = waypoints[closest_waypoints[0]]

  track_direction = math.atan2(next_point[1] - prev_point[1], next_point[0] - prev_point[0])

  reward = ...

  return float(reward)
```

Custom Reward Function 만들기 - 트랙의 중앙선 각도와 DeepRacer의 방향이 비슷할수록 보상을 받게 하자

- heading변수는 degree로 각도를 표현하기 때문에 각도를 변환해줍니다.

```
import math
def reward_function(params) :
  waypoints = params['waypoints']
  closest_waypoints = params['closest_waypoints']
  heading = params['heading']

  next_point = waypoints[closest_waypoints[1]]
  prev_point = waypoints[closest_waypoints[0]]

  track_direction = math.atan2(next_point[1] - prev_point[1], next_point[0] - prev_point[0])
  track_direction = math.degrees(track_direction)

  reward = ...

  return float(reward)
```

Custom Reward Function 만들기 - 트랙의 중앙선 각도와 DeepRacer의 방향이 비슷할수록 보상을 받게 하자

- 계산한 트랙의 각도와 DeepRacer의 진행방향과의 각도 차이를 계산합니다.
  - 180도가 넘어가면 작은 쪽의 각도를 각도의 차이로 저장하기 위해 360도에서 뺀 값을 저장해줍니다.

```python
import math
def reward_function(params) :
  waypoints = params['waypoints']
  closest_waypoints = params['closest_waypoints']
  heading = params['heading']

  next_point = waypoints[closest_waypoints[1]]
  prev_point = waypoints[closest_waypoints[0]]

  track_direction = math.atan2(next_point[1] - prev_point[1], next_point[0] - prev_point[0])
  track_direction = math.degrees(track_direction)

  direction_diff = abs(track_direction - heading)
  if direction_diff > 180:
      direction_diff = 360 - direction_diff

  reward = ...

  return float(reward)
```

Custom Reward Function 만들기 - 트랙의 중앙선 각도와 DeepRacer의 방향이 비슷할수록 보상을 받게 하자

- 마지막으로 계산한 값들을 활용하여 보상을 부여하는 코드를 작성합니다.

```
import math
def reward_function(params) :
  waypoints = params['waypoints']
  closest_waypoints = params['closest_waypoints']
  heading = params['heading']

  next_point = waypoints[closest_waypoints[1]]
  prev_point = waypoints[closest_waypoints[0]]

  track_direction = math.atan2(next_point[1] - prev_point[1], next_point[0] - prev_point[0])
  track_direction = math.degrees(track_direction)

  direction_diff = abs(track_direction - heading)
  if direction_diff > 180:
      direction_diff = 360 - direction_diff

 if direction_diff <= 10 :
    reward = 10
 else :
    reward = 1

  return float(reward)
```

Custom Reward Function 만들기 - 트랙의 중앙선 각도와 DeepRacer의 방향이 비슷할수록 보상을 받게 하자

- **위 코드는 트랙의 각도와 DeepRacer의 진행 방향의 차이가 10도 이내일 때 높은 보상을 주고 있습니다.**
  - 기준 각도를 변경할 수 있고, 기준 각도를 벗어나면 패널티를 받도록 변경할 수도 있습니다.

Custom Reward Function 만들기 - 트랙의 중앙선 각도와 DeepRacer의 방향이 비슷할수록 보상을 받게 하자

- 각도의 차이가 10도보다 작거나 같으면 초기 보상인 1을 그대로 받지만, 10도보다 크면 0.5로 감소하는 패널티를 받습니다.

```python
import math
def reward_function(params) :
  waypoints = params['waypoints']
  closest_waypoints = params['closest_waypoints']
  heading = params['heading']

  next_point = waypoints[closest_waypoints[1]]
  prev_point = waypoints[closest_waypoints[0]]

  track_direction = math.atan2(next_point[1] - prev_point[1], next_point[0] - prev_point[0])
  track_direction = math.degrees(track_direction)

  direction_diff = abs(track_direction - heading)
  if direction_diff > 180:
      direction_diff = 360 - direction_diff

 reward = 1
 if direction_diff > 10 :
    reward *= 0.5

  return float(reward)
```

Custom Reward Function 만들기

- 이외에도 변수를 다양하게 종합하여 활용하면 좀 더 정교하고 성능을 개선할 수 있는 보상함수를 설계할 수 있습니다.