

# DeepRacer 심화

동양미래대학교

# Agent, State, Action

# Agent, State, Action

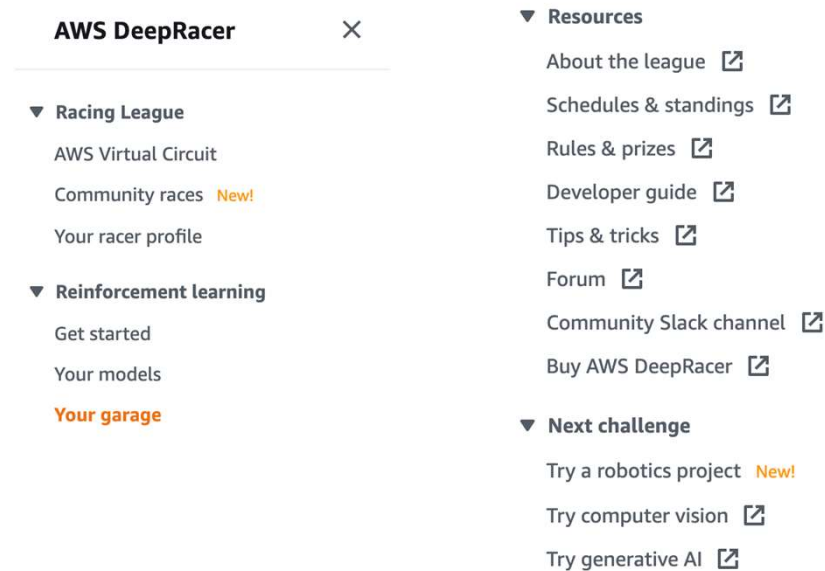
---

- DeepRacer의 agent는 훈련에서 사용되는 가상 레이싱 자동차를 의미합니다.
  - 더 자세히는 neural network를 포함하고 있어서 state에 대한 action을 결정할 수 있습니다.



# Agent, State, Action

- AWS DeepRacer Console에서 Agent 를 생성하는 과정을 자세히 살펴보겠습니다.
  - 1. AWS DeepRacer Console에 로그인합니다.
  - 2. 왼쪽 탭에서 **Your garage**를 선택합니다.



# Agent, State, Action

---

- DeepRacer는 강화학습을 통해 자율 주행 모델을 만들고자 하기 때문에 학습을 하는 주체인 Agent는 운행이 가능한 자동차의 형태로 정의됩니다.
  - DeepRacer의 Agent를 만드는 곳은 실제 자동차를 세워두는 차고를 생각하게 합니다.
  - 차고 안에서 나만의 Agent, 즉 커스텀 자동차를 만들어볼 수 있습니다.

# Agent, State, Action

---

- 3. Build new vehicle을 선택합니다.

AWS DeepRacer > Your garage

## Garage Info

Create model

Build new vehicle

Configure your vehicle with one or more sensors, choose a neural network topology, and customize the action space to meet your racing criteria. Customize your vehicle's appearance for personalized visualization in training.

# Agent, State, Action - Step 1 : Personalization

- 4. Step 1 : Personalization 에서는 자동차의 이름과 외형, 태그를 결정하게 됩니다.
  - 원하는 이름과 외형, 색을 고르고 **Next** 버튼을 클릭하세요.

The screenshot shows a 'Personalization' window with the following sections:

- Customize vehicle appearance**: Subtitle 'Color your vehicle to make it stand out in the crowd'. It includes a text input field for 'Name your vehicle' with the placeholder 'IndyRacer1000' and a note: 'The vehicle name must have 1-32 characters. Valid characters: A-Z, a-z, 0-9, \_ (underscore) - (hyphen)'.
- Vehicle shell (3)**: Subtitle 'Choose a vehicle model type to customize'. It displays three vehicle models in a row: a black and blue car, a yellow and red car, and a white and grey car. The first model is selected with a blue border.
- Vehicle model**: Subtitle 'DeepRacer'. It shows a large 3D model of the selected black and blue car.
- Trim colors**: A row of six color swatches (red, black, grey, white, blue, yellow).
- Tags**: A section with a right-pointing arrow and the text 'Tags'.
- Navigation**: 'Cancel' and 'Next' buttons at the bottom right.

# Agent, State, Action - Step 2 : Mod specifications

- 5. Step 2 : Mod specifications 에서는 차량의 외부 데이터 수집을 위해 센서를 세팅합니다.
  - 카메라의 개수와 LIDAR 센서의 유무를 결정하게 됩니다.

### Mod your own vehicle

#### Mod specifications

Configure your vehicle with one or more sensors, choose a neural network topology, and customize the action space to meet your racing criteria. Customize your vehicle's appearance for personalized visualization in training.

#### Sensor modification

Swap sensors to improve your DeepRacer's racing performance

☒ Camera

Single-lens 120-degree field of view camera capturing at 15fps. The images are converted into greyscale before being fed to the neural network.

► Benefits of the front-facing camera

☐ Stereo camera


Composed of two single-lens cameras, stereo camera can generate depth information of the objects in front of the agent and thus be used to detect and avoid obstacles on the track. The cameras capture images with the same resolution and frequency. Images from both cameras are converted into grey scale, stacked and then fed into the neural network.

► Benefits of the stereo camera

#### Add-on sensors

☐ LIDAR sensor

LIDAR is a light detection and ranging sensor. It scans its environment and provides inputs to the model to determine when to overtake another vehicle and beat it to the finish line. It provides continuous visibility of its surroundings and can see in all directions and always know its distances from objects or other vehicles on the track.



CAMERA

Cancel Previous Next



# Agent, State, Action - Step 2 : Mod specifications

---

- Single Camera(카메라 1대)

- 전방 120도 공간의 정보를 수집할 수 있는 카메라 입니다. 훈련 또는 추론을 위한 neural network에 전달되기 전에 grayscale로 변환됩니다.
- 카메라를 2대 사용하는 것과 비교했을 때 neural network에 전달되는 입력값이 적으므로 네트워크가 비교적 간단하며 수렴 속도가 빠릅니다.
- Race type이 time trial인 경우 single camera 센서만으로도 비교적 주행을 잘 하는 모델을 만들 수 있지만, 환경이 복잡해지거나 장애물 피하기와 같은 주행 환경에서는 부적합할 수 있습니다.

# Agent, State, Action - Step 2 : Mod specifications

---

- Stereo Camera(카메라 2대)

- 사람이 거리를 탐지하는 감각기관이 없음에도 불구하고 원근감을 구분할 수 있는 이유는 눈이 두 개이기 때문입니다.  
두 눈이 하나의 물체를 볼 때 만들어내는 시야각을 활용해 거리감을 인지하게 됩니다.
- Stereo camera도 이런 원리와 비슷합니다. 두개의 카메라로부터 정보가 수집되기 때문에 single camera와 비교했을 때 물체와의 거리와 같은 이미지의 깊이에 대한 정보를 추가적으로 수집할 수 있습니다.
- 다만, 훈련과 추론을 위한 neural network로 전달되는 입력값이 더 많아지므로 비교적 수렴하는 속도가 느릴 수 있습니다.

# Agent, State, Action - Step 2 : Mod specifications

---

- LIDAR 센서

- LIDAR 센서는 빛을 감지하고 거리를 측정하는 센서입니다. 카메라는 초당 정해진 프레임 수 만큼만 불연속적으로 정보를 수집하는 데에 비해, LIDAR는 연속적인 정보 수집이 가능하며 360도로 정보를 수집합니다.
- 물체와의 거리를 카메라에 비해 더 많은 정보를 통해 인지하기 때문에 장애물 피하기와 복잡한 환경 (트랙)에서의 주행에 적합합니다.

# Agent, State, Action

---

- DeepRacer에서 state는 environment의 현재 상태를 의미합니다.
  - 선택한 센서를 통해 state가 agent가 인식할 수 있는 형태로 전달됩니다.



# Agent, State, Action

---

- DeepRacer 학습에 사용되는 모델은 강화학습과 딥러닝을 접목한 알고리즘을 사용하고 있는데, 학습에서 사용하는 neural network에 입력될 값이 센서로부터 수집된 데이터입니다.
  - 훈련 후에 평가 혹은 실제 트랙에서 주행할 때는 학습된 모델(neural network)을 기반으로 하여 추론이 진행되는데, 추론 엔진에 입력될 값도 센서로부터 수집된 데이터입니다.
  - 따라서 활용할 센서에 따라 neural network의 입력의 개수가 달라집니다.

# Agent, State, Action - Step 3 : Action space

- 6. Step 3 : Action space 에서는 차량의 action space를 정할 수 있습니다.

**Action space** [Info](#)

**Action space** refers to the available set or range of actions an agent can pick from in response to each situation, or **state**, in simulation or the physical world.

Choose your action space type [Info](#)

☐ **Discrete**  
A discrete action space represents all of the agent's possible actions for each state in a **finite set**.

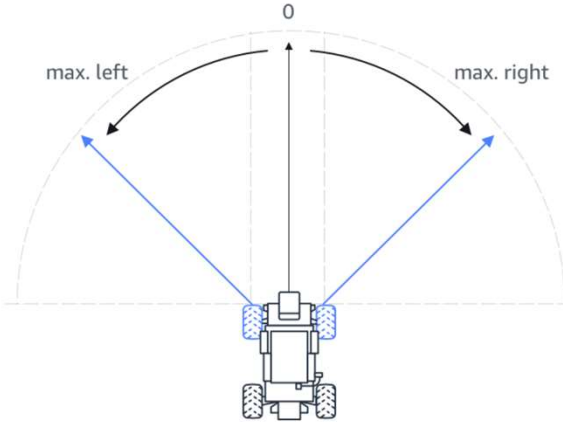
☒ **Continuous**  
A continuous action space allows the agent to select an action from a **range of values** for each state.

**Right steering angle range**  
  
degrees  
Values are between -30 and 0.

**Left steering angle range**  
  
degrees  
Values are between 0 and 30.

**Minimum speed**  
  
m/s  
Values are between 0.5 and 4.

**Maximum speed**  
  
m/s  
Values are between 0.5 and 4.



The diagram illustrates the action space for a car. It shows a car at the bottom center with a semi-circular field of view above it. A vertical line represents the straight-ahead steering position, labeled '0'. Two curved arrows indicate the range of steering angles: 'max. left' on the left and 'max. right' on the right. Two blue arrows point from the car towards the 'max. left' and 'max. right' positions, representing the range of steering actions. The entire field of view is enclosed in a dashed semi-circle.

[Cancel](#) [Previous](#) [Done](#)

# Agent, State, Action

---

- 강화학습 모델이 훈련되는 과정에서 agent는 주어진 환경에서 행동을 취하고 그 행동은 환경에 다시 반영되는 과정이 반복됩니다.
  - Action space는 agent가 할 수 있는 행동의 집합을 의미합니다.

# Agent, State, Action

- 예를 들어 벽돌깨기 게임하는 상황을 가정하면, agent는 공을 튕겨내는 막대 혹은 막대를 움직이는 가상의 사용자가 됩니다.
- 그리고 agent가 취할 수 있는 action은 오른쪽으로 이동, 왼쪽으로 이동으로 나눌 수 있습니다.
- Action을 좀 더 세분화하자면, 막대를 움직이게 될 방향(왼쪽, 오른쪽)과 속도의 조합이 action이 될 수 있습니다.





# Agent, State, Action

- DeepRacer의 action도 이와 비슷합니다.
  - 자동차가 어떤 방향으로, 얼마큼의 속도로 주행할지를 결정해야 합니다.
  - 방향이 왼쪽, 오른쪽, 직진으로 세 종류가 있고, 속도는 느림과 빠름 두 종류가 있다면 자동차가 결정할 수 있는 행동은  $3 \times 2 = 6$ 가지가 됩니다.
  - 이 6가지의 행동의 집합을 action space라고 합니다.



## Agent, State, Action - Step 3 : Action space

---

- AWS DeepRacer Console에서 action space를 정의할 때 가장 먼저 골라야 할 것은 Discrete과 Continuous입니다.

Choose your action space type [Info](#)

☐ Discrete

A discrete action space represents all of the agent's possible actions for each state in a **finite set**.

☒ Continuous

A continuous action space allows the agent to select an action from a **range of values** for each state.

# Agent, State, Action - Step 3 : Action space (Discrete)

- Discrete은 action space를 이산적으로 정의하게 됩니다.
  - Discrete을 선택하면 최대 회전 각도와 각도 세분성, 최대 속력과 속도 세분성을 선택하게 됩니다.

Choose your action space type [Info](#)



**Discrete**

A discrete action space represents all of the agent's possible actions for each state in a **finite set**.



**Continuous**

A continuous action space allows the agent to select an action from a **range of values** for each state.

Maximum steering angle

30

degrees

Max values are between 1 and 30.

Steering angle granularity

5

Maximum speed

1

m/s

Select values between 0.1 and 4.

Speed granularity

2

# Agent, State, Action - Step 3 : Action space (Discrete)

---

- 최대 회전 각도

- 정면을 기준으로 좌우로 몇도까지 회전을 허용할지 선택합니다.
- 선택 가능한 범위는 1도부터 30도까지 입니다.

# Agent, State, Action - Step 3 : Action space (Discrete)

---

- 회전 세분성

- 좌우 최대 회전 각도 안에서 몇 개로 각도를 세분화할지 선택합니다.
- 정면 선택 보장을 위해 홀수 개수로만 선택할 수 있습니다.

Steering angle granularity

5 ▲

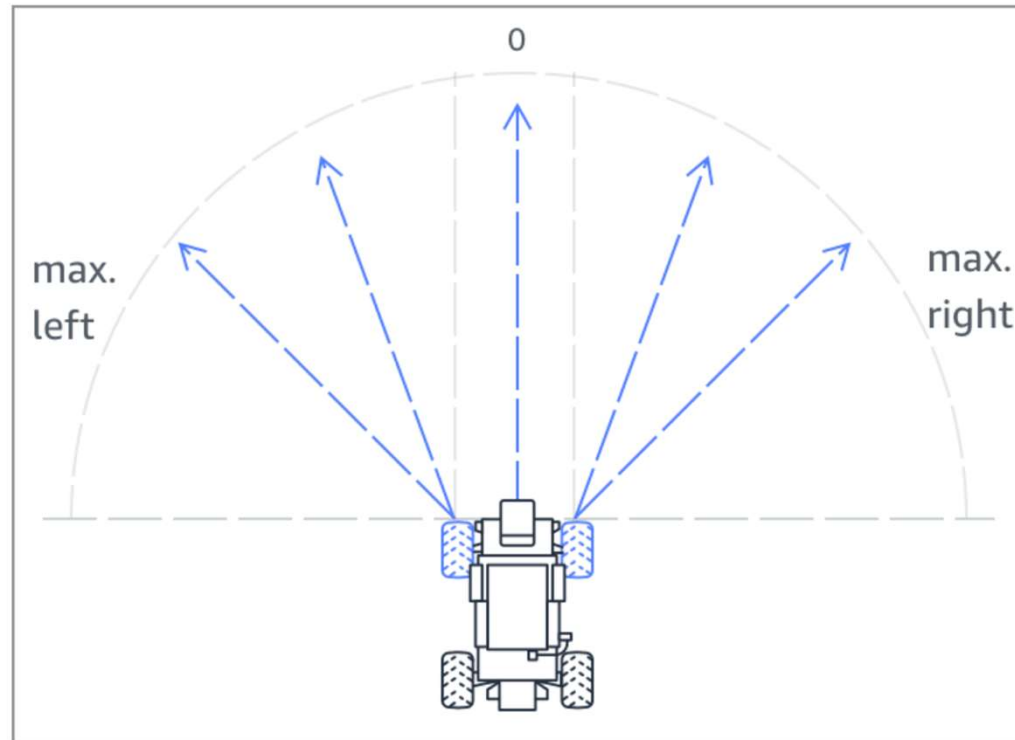
3

5

7

## Agent, State, Action - Step 3 : Action space (Discrete)

- 예를 들어 최대 회전 각도를 30으로 선택하고 세분성을 5로 선택한다면 -30~30도의 값을 5개로 균등하게 나누어 선택할 수 있는 각도를 만들게 됩니다.



## Agent, State, Action - Step 3 : Action space (Discrete)

---

- 최대 속도

- 자동차가 움직이는 최대 속도를 선택합니다. 선택 가능한 범위는 0.1m/s부터 4m/s 입니다.

# Agent, State, Action - Step 3 : Action space (Discrete)

- 속력 세분성

- 최대 속력 안에서 몇 개로 속력을 세분화할지 선택합니다.

Speed granularity

2	▲
1	
2	
3	

ACTION LIST



## Agent, State, Action - Step 3 : Action space (Discrete)

---

- 예를 들어 최대 속력을 1m/s로 선택하고 세분성을 2로 선택한다면 선택 가능한 속력은 0.5m/s와 1m/s가 됩니다.

# Agent, State, Action - Step 3 : Action space (Discrete)

- 4가지에 대한 선택을 마치면 action list가 표시됩니다.

Action list

Action number	Steering angle	Speed
0	-30 degrees	0.5 m/s
1	-30 degrees	1 m/s
2	-15 degrees	0.5 m/s
3	-15 degrees	1 m/s
4	0 degrees	0.5 m/s
5	0 degrees	1 m/s
6	15 degrees	0.5 m/s
7	15 degrees	1 m/s
8	30 degrees	0.5 m/s
9	30 degrees	1 m/s

## Agent, State, Action - Step 3 : Action space (Discrete)

---

- 앞선 선택한 항목들을 통해 action list를 살펴보면, 최대 회전 각도는 30도이고 5개로 세분화를 하기로 선택했기 때문에 선택 가능한 각도는 -30, -15, 0, 15, 30도가 있습니다.
  - 최대 속력은 1m/s 이고 2개로 세분화를 하기로 선택했기 때문에 선택 가능한 속력은 0.5, 1이 됩니다.
  - 따라서 각도와 속력을 조합해서 선택 가능한 action은 총 10개 가 됩니다.

# Agent, State, Action - Step 3 : Action space (Continuous)

- 다음으로는 Continuous Action Space를 만들어보겠습니다.
  - Discrete과 다르게 continuous에서는 action space를 연속적으로, 즉 범위로 정의하게 됩니다.
  - Continuous를 선택하면 각도의 범위와 속력의 범위를 선택하게 됩니다.

Choose your action space type [Info](#)

☐ Discrete

A discrete action space represents all of the agent's possible actions for each state in a **finite set**.

☒ Continuous

A continuous action space allows the agent to select an action from a **range of values** for each state.

Right steering angle range

-30

degrees

Values are between -30 and 0.

Left steering angle range

30

degrees

Values are between 0 and 30.

Minimum speed

0.5

m/s

Values are between 0.5 and 4.

Maximum speed

1

m/s

Values are between 0.5 and 4.

# Agent, State, Action - Step 3 : Action space (Continuous)

---

- **우측 최대 각도**

- 우측 최대 회전각을 선택합니다. 선택 가능한 범위는 0도부터 30도까지 입니다.

- **좌측 최대 각도**

- 좌측 최대 회전각을 선택합니다. 선택 가능한 범위는 0도부터 30도까지 입니다.

- **최소 속력**

- 최소 속력을 선택합니다. 선택 가능한 범위는 0.5m/s부터 4m/s까지 입니다.

- **최대 속력**

- 최대 속력을 선택합니다. 선택 가능한 범위는 0.5m/s부터 4m/s까지 입니다.

## Agent, State, Action - Step 3 : Action space (Continuous)

---

- 선택 가능한 각도와 선택 가능한 속력이 이산적으로 정해지고 이를 조합하여 한정적인 action space를 만들 수 있는 discrete과 달리 continuous에서는 설정한 범위 내에서 어떤 값이든 선택이 가능하기 때문에 agent가 취할 수 있는 action이 무수히 많습니다.

# Agent, State, Action

---

- 연속적인 action space를 정의하면 discrete과 비교했을 때 움직임이 부드럽다는 장점이 있습니다.
  - 또한, reward function에서 각도와 속도에 대한 보상을 정의하는 것이 가능해지기 때문에 좀 더 잘 주행하는 모델을 만들 수 있습니다.

# Agent, State, Action

---

- 하지만 설정한 범위 안에서 optimal한 각도와 속력을 뽑아내는 과정이 시간이 오래 걸릴 수 있습니다.
  - 이산적으로 action을 설정하면 한정적인 action을 평가하고 주어진 환경에서 최선의 선택을 고르기 때문에 비교적 훈련 과정이 짧습니다.



# Agent, State, Action

- 7. 만든 agent 확인하기
  - Agent를 다 만들고 나면 agent card가 생성됩니다.

**agent1**Mod vehicle

Sensor(s)  
Camera

---


Neural network topology  
3 Layer CNN

---

Action space  
Type: Discrete  
Speed: { 0.4, 0.8 } m/s  
Steering angle: { -30, -15, 0, 15, 30 } °

---

Shell type  
DeepRacer



# Agent, State, Action

---

- 앞서 설정한대로 센서, action space, agent의 외형이 설정된 것을 확인할 수 있습니다.
  - 설정하지 않았지만 확인할 수 있는 항목은 **Neural network topology**입니다.
  - **Neural network topology**는 agent가 훈련하고 주행하는 동안 state를 받아 action을 결정하는 데에 활용될 neural network의 층(layer)수와 모델입니다.
  - AWS DeepRacer Console에서는 agent를 생성할 때 디폴트 값으로 3개의 층을 가진 CNN으로 설정됩니다.

# Environment

# Environment

- DeepRacer에서 environment는 agent가 이동할 수 있고 위치와 상태의 표시가 가능한 트랙을 의미합니다.
- 또한 모델을 만들 때 정의한 race type도 environment 정의에 포함됩니다.



# Environment - Track

---

- 1. Track 선택하기

- 모델을 만드는 과정에서 트랙을 선택할 수 있습니다.
- 선택한 트랙이 온라인 시뮬레이션 훈련 환경에서 가상 트랙으로 구성되고, agent가 해당 트랙을 탐험하며 학습하게 됩니다.

# Environment - Track

- 선택할 수 있는 트랙은 35가지가 있습니다.
- 처음 학습에는 복잡하고 어려운 트랙보다 간단하고 쉬운 트랙을 선택하는 것을 권장합니다.

## Environment simulation [Info](#)

Simulated environment emulates a track to train your model.

Choose a track

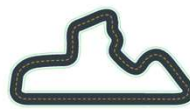
### ☒ Baja Highway

At 65.18m, The Baja Highway extends upon the shorter turnpike with pro level difficulty and even more opportunities for generating speed. A second massive straightaway mid course is similarly capped by a hairpin and increasing radius turn, along with an unforgiving technical section full of wrought with tight cutbacks and hairpins.



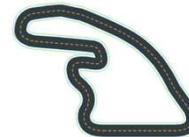
### ☐ Baja Turnpike

At 40.79m, the Baja Turnpike is a short angular track that mixes full throttle straightaways with hard decelerating turns. Signature features include a long high speed sprint over the finish line end capped by a harsh hairpin and tight 90 degree turn. While you won't find any sand traps on this course there are no shortage of places to get stuck.



### ☐ Kuei Raceway

Kuei Raceway is a fast short track (46.15m) with a friendly high speed arch and two straightaways interspersed between 2 hairpins, a chicane, and a technical cutback hairpin. It is named after 3rd place DeepRacer League finalists Kuei of NCTU CGI Taiwan.



# Environment - Race type

---

- 2. Race type 선택하기

- 모델을 만드는 과정에서 race type을 선택할 수 있습니다.
- 선택한 type에 따라서 온라인 시뮬레이션 훈련 환경이 각각 다르게 구성됩니다.

# Environment - Race type : Time trial

- Time trial은 1 번에서 선택한 트랙으로만 구성됩니다.

## Race type

Choose a race type

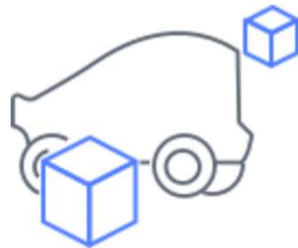
☒ Time trial

The agent races against the clock on a well-marked track without stationary obstacles or moving competitors.



☐ Object avoidance

The vehicle races on a two-lane track with a fixed number of stationary obstacles placed along the track.



☐ Head-to-head racing

The vehicle races against other moving vehicles on a two-lane track.

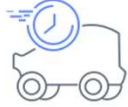


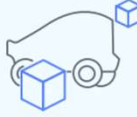


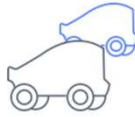
# Environment - Race type : Object avoidance

- Object avoidance는 트랙과 트랙 위에 장애물이 포함됩니다.

Choose a race type

☐ Time trial  
The agent races against the clock on a well-marked track without stationary obstacles or moving competitors.  


☒ Object avoidance  
The vehicle races on a two-lane track with a fixed number of stationary obstacles placed along the track.  


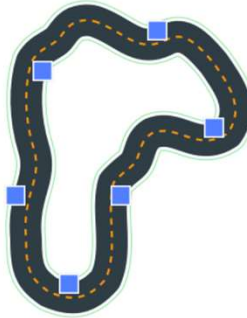
☐ Head-to-head racing  
The vehicle races against other moving vehicles on a two-lane track.  


Select your criteria for how DeepRacer learns to avoid objects

☒ Fixed location [Info](#)  
Objects are evenly distributed in fixed locations across two lanes along the track. The objects' positions do not change from episode to episode. Training tends to converge faster when obstacles are in fixed locations. But such models might overfit and may not generalize well to real-world tracks.

☐ Random location [Info](#)  
Objects are randomly distributed across two lanes along the track at the beginning of each episode. Training usually takes longer to converge for models to avoid obstacles in random locations that change from episode to episode. However, the trained models are more robust and generalize better to real-world tracks.


The depicted track illustrates 6 objects evenly distributed at fixed locations along the track with 3 of them on the left lane and 3 others on the right lane.



Number of objects on a track  
Choose the number of objects on a track. Increasing objects on tracks might take longer to converge, but more robust.

3 ▼

Object Type [Info](#)

  
Box

## Environment - Race type : Object avoidance

---

- 장애물을 고정된 위치에 놓을 것인지 혹은 랜덤한 위치에 놓을 것인지 선택할 수 있습니다.
- 고정된 위치에 장애물을 배치하는 경우에는 모델이 비교적 빠르게 수렴합니다.
  - 학습하는 동안 장애물의 위치가 변경되지 않기 때문에 state에 대한 최선의 action을 선택하는 것이 수월하기 때문입니다.
  - 반대로 랜덤한 위치에 장애물을 배치하는 경우에는 모델이 수렴하는 속도가 비교적 느립니다.
  - 학습하는 동안 장애물의 위치가 계속 변경되기 때문에 최선의 action을 결정하는 것이 어렵기 때문입니다.

## Environment - Race type : Object avoidance

---

- 하지만 고정된 위치에 장애물을 배치하고 학습을 진행하면 모델이 고정된 장애물 위치에 오버피팅될 가능성이 높습니다.
  - 반대로 랜덤한 위치에 장애물을 배치하고 학습을 진행하면 실제 트랙에서 장애물을 피할 수 있는 가능성이 높아집니다.

# Environment - Race type : Object avoidance

- 장애물의 개수도 선택할 수 있습니다.


beginning of each episode Training

1
2
3
4
5
6

as

2 ▲

Object Type [Info](#)

 Box

## Environment - Race type : Object avoidance

---

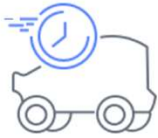
- 만약 장애물이 6개이면서 고정된 위치를 선택했다면 안쪽 레인에 3개, 바깥쪽 레인에 3개가 배치됩니다. 박스 사이의 간격 또한 균등하게 배치 됩니다.
- 실제 트랙에서 장애물 피하기 타입으로 학습한 모델을 테스트 할때는 트랙의 어두운 바닥 색과 구분될 수 있는 색깔의 직육면체 혹은 정육면체 상자를 사용하는 것이 적합합니다.

# Environment - Race type : Head-to-Head racing

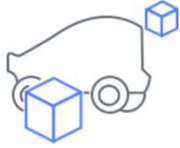
- Head-to-Head racing은 트랙과 트랙을 주행하는 다른 DeepRacer가 포함됩니다.

Choose a race type


☐ Time trial  
The agent races against the clock on a well-marked track without stationary obstacles or moving competitors.



☐ Object avoidance  
The vehicle races on a two-lane track with a fixed number of stationary obstacles placed along the track.



☒ Head-to-head racing  
The vehicle races against other moving vehicles on a two-lane track.



Choose the number of bot vehicle(s)  
Bot vehicle(s) follow predefined paths with set speeds.

1 ▼

Set the speed  
Choose a consistent speed for bot vehicles. No variation for turns or straightways

Speed (m/s)


0.5

Select values between 0.1 and 4.

☐ Enable lane changes  
Choose a minimum and a maximum time interval for each bot to randomly change lanes.

Lane changes occur randomly between 1-5 seconds

1 2 3 4 5



# Environment - Race type : Head-to-Head racing

---

- Bot vehicle의 개수와 bot vehicle의 속도를 선택할 수 있습니다.

Choose the number of bot vehicle(s)

Bot vehicles(s) follow predefined paths with set speeds.

1	▲
1	
2	
3	
4	

Select values between 0.1 and 4

es. No variation fo

## Environment - Race type : Head-to-Head racing

---

- 처음에는 2대 이하의 bot을 선택하고 agent의 최대 속도보다 작은 값으로 속도를 설정하는 것을 권장합니다.

### Set the speed

Choose a consistent speed for bot vehicles. No variation for turns or straightways

Speed (m/s)

Select values between 0.1 and 4.



## Environment - Race type : Head-to-Head racing

---

- Bot vehicle의 레인 변경 가능 여부와 시간 간격을 선택할 수 있습니다.

☒ Enable lane changes

Choose a minimum and a maximum time interval for each bot to randomly change lanes.

Lane changes occur randomly between 2-3 seconds



## Environment - Race type : Head-to-Head racing

---

- 레인 변경을 체크하지 않으면 학습하는 동안 bot들은 자신의 레인을 벗어나지 않고 일정한 속도로 주행합니다.
  - 레인 변경을 체크하면 시간 간격을 선택할 수 있게 됩니다.
  - 2와 3을 선택하면 2-3초 마다 bot들은 레인을 변경하게 됩니다.
- 처음 head-to-head racing 타입으로 학습할 때는 레인 변경에 체크하지 않고 학습을 진행하는 것을 권장합니다.

# Hyperparameter

# Hyperparameter

---

- AWS DeepRacer Console에서 모델을 만드는 과정에서 hyperparameter를 선택할 수 있습니다.
  - Hyperparameter는 모델을 훈련하는 동안 내부에서 최적화 되는 값들이 아닌 좋은 모델을 만들기 위해서 외부에서 설정하는 값들입니다.
  - 안타깝게도 hyperparameter는 경험적일 수 밖에 없습니다.
  - 조건에 의해 정할 수 있는 최적의 값이 존재할 수 없기 때문에 여러 번의 경험과 시행착오로 만들고자 하는 모델의 필요한 hyperparameter를 찾을 수 있습니다.

# Hyperparameter

---

- AWS DeepRacer Console에서 설정할 수 있는 hyperparameter들을 알아보기 전에 이해에 필요한 용어를 먼저 살펴보겠습니다.

# Hyperparameter - 관련 용어

---

- Data point

- Data point는 하나의 경험을 의미하며,  $(s, a, r, s')$ 와 같이 표현할 수 있습니다.
- $s$ 는 DeepRacer에 부착된 카메라 센서로 감지된 현재 상태(state)를 의미합니다.  $a$ 는 해당 state( $s$ )에서 차량의 움직임(action)을 의미합니다.
- $r$ 은 차량의 움직임( $a$ )에 대한 보상(reward)을 의미합니다.  $s'$ 는 차량의 움직임( $a$ )으로 변화된 다음 상태(state)를 의미합니다.
- 이 4가지의 값들을 하나의 경험이라고 할 수 있으며, 이를 data point라고 정의합니다.

# Hyperparameter - 관련 용어

---

- Episode

- Episode는 DeepRacer가 트랙의 출발점에서 주행을 시작해서 트랙을 완주하거나 벗어나기까지의 경험들의 나열을 의미하며, data point의 시퀀스로 표현됩니다.
- 각각의 episode는 다른 개수의 data point, 즉 다른 길이로 표현될 수 있습니다.

# Hyperparameter - 관련 용어

---

- Experience buffer

- 정해진 횟수의 episode를 통해 수집한 data point의 집합입니다.
- 즉, 훈련하는 동안 수집한 **경험 전체**를 의미합니다.



# Hyperparameter - 관련 용어

---

- Batch

- Experience buffer 전체를 훈련에 사용하지 않고 일부의 data point를 추출하여 훈련을 진행합니다.
- Batch는 훈련에서 사용되는 experience buffer(전체 데이터)의 일부를 의미하며 policy network를 한번 업데이트할 때 사용하는 단위입니다.

# Hyperparameter - 관련 용어

---

- Training data

- Training data는 훈련에 사용된 batch들의 집합입니다.
- 훈련을 위해 experience buffer에서 batch 단위로 data point를 추출할 때 랜덤하게 추출하기 때문에 training data와 experience buffer가 늘 같지는 않습니다.

# Hyperparameter

---

- 다음은 hyperparameter에 대해 자세히 살펴보겠습니다.

# Hyperparameter - Gradient descent batch size

---

- Policy network를 한번 업데이트에 사용할 batch의 크기입니다.
  - Batch size를 크게 선택할수록 더 많은 경험을 사용하기 때문에 안정적으로 훈련할 수 있지만, 최적의 policy에 수렴하는데 걸리는 시간이 길어집니다.
  - Batch size가 정해지면 experience buffer에서 랜덤하게 batch를 추출합니다.
- Gradient descent batch size는 32, 64, 128, 256, 512 중에 고를 수 있으며, 디폴트 값은 64입니다.

# Hyperparameter - Number of epochs

---

- Training data를 한번 사용하여 훈련하는 것이 1 epoch입니다.
  - 즉, training data를 몇 번 반복하여 훈련할지를 결정합니다.
  - Epoch의 개수를 크게 하면 안정적으로 업데이트할 수 있지만, 훈련의 시간이 길어집니다.
  - Batch size가 작다면 epoch의 개수를 크게 설정할 필요는 없습니다.
- Number of epochs는 3과 10사이의 정수로 설정할 수 있으며, 디폴트 값은 3입니다.

# Hyperparameter - Learning rate

---

- Policy network를 업데이트할 때 gradient descent 혹은 gradient ascent 방법을 사용합니다.
  - 현재 위치에서의 기울기 정보를 활용하여 최적화하는 방법입니다.
  - Learning rate는 기울기 정보를 얼마나 활용할지에 대한 값입니다.
  - Learning rate가 너무 크면 기울기 정보를 많이 반영하기 때문에 업데이트의 정도도 커집니다. 따라서 수렴하지 못할 수 있습니다.
  - 반대로 learning rate가 너무 작으면 기울기 정보를 적게 반영하고 업데이트의 정도도 작아집니다. 업데이트가 느리면 수렴하는 속도가 매우 느릴 수 있습니다.

# Hyperparameter - Learning rate

---

- Learning rate는 0.00000001부터 0.001 사이의 실수값으로 설정할 수 있으며, 디폴트 값은 0.0003 입니다.

# Hyperparameter - Entropy

---

- 현재 action을 결정할 때 사용하는 policy의 불확실성을 의미합니다.
  - Entropy가 크다면 policy에 대한 확신도가 적다는 의미이므로 agent의 탐험 가능성이 높아집니다.
  - Entropy가 작으면 policy에 대한 확신도가 크다는 의미이므로 agent는 탐험보다 지금까지의 경험을 더 많이 활용하게 됩니다.
  - 탐험과 활용은 강화학습에서 중요한 요소이지만, trade-off 관계에 있습니다.
- Entropy는 0과 1 사이의 실수값으로 설정할 수 있으며, 디폴트 값은 0.01입니다.



# Hyperparamete - Discount factor

---

- Discount factor는 미래에 받을 보상의 기여 정도를 의미합니다.
  - Discount factor가 크다면 더 먼 미래의 보상까지 agent가 현재에서 고려하게 되므로 훈련 속도가 느려집니다.
- Discount factor는 0과 1사이의 실수값으로 설정할 수 있지만 0.99, 0.999, 0.9999 중에 선택하여 설정하는 것을 권장합니다.
  - 디폴트 값은 0.999입니다.

# Hyperparamete - Loss type

---

- Loss type은 policy network를 업데이트하는데 필요한 loss function 유형입니다.
  - 딥러닝이 적용된 강화학습에서는 Q-learning을 통해 최적의 행동 가치 함수(Q-value)를 미리 구하고 최적의 행동 가치 함수를 갖게 하는 policy를 neural network를 통해 찾아가는 과정을 진행합니다.
  - 이 과정에서 최적의 행동 가치함수를 '정답값', 현재 업데이트하고 있는 policy와 입력된 데이터(state)를 통해 구한 행동 가치 함수를 '예측값'으로 보고 딥러닝을 진행합니다.

# Hyperparamete - Loss type

---

- 선택 가능한 loss type은 Huber loss와 Mean squared error loss가 있습니다.
  - Huber loss는 MSE와 MAE의 조합이므로 이상치에 덜 민감합니다.
  - 따라서 업데이트 규모가 클 경우 Mean squared error loss보다 Huber loss가 적합합니다.  
디폴트 값은 Huber loss 입니다.

# Hyperparameter - Number of experience episodes between each policy-updating iteration

---

- Episode의 횟수를 의미합니다.
  - 즉, agent가 트랙의 시작점에서 트랙을 완주하거나 트랙을 벗어나기까지의 과정을 몇번 반복할지를 설정합니다.
  - 단순한 환경에서의 학습에서는 많은 episode의 개수가 필요하지 않을 수 있습니다.
  - 하지만 복잡한 환경에서는 policy를 최적화하는 과정에서 local optima에 빠질 수 있으므로 더 많은 episode가 필요합니다.
  - Episode가 많아지면 훈련은 안정적이어지지만, 훈련 속도는 느려집니다.

## Hyperparamete - Number of experience episodes between each policy-updating iteration

---

- Number of experience의 값은 5와 100 사이의 정수로 설정할 수 있으며 10, 20, 40을 권장합니다. 디폴트 값은 20입니다.

# Hyperparamete - SAC alpha ( $\alpha$ ) value

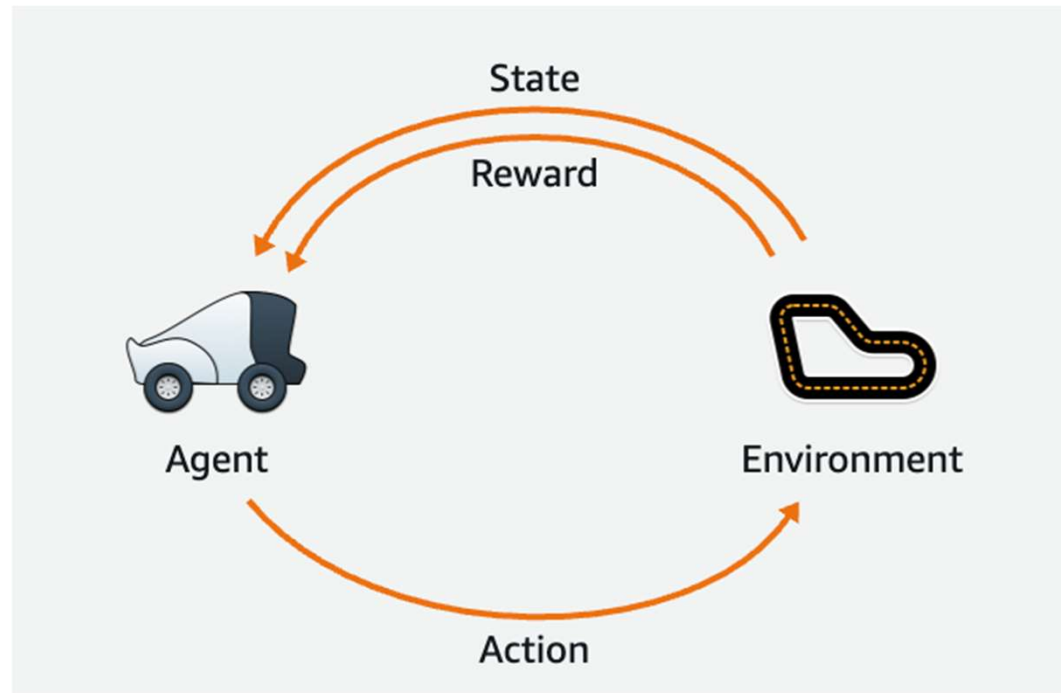
---

- SAC 알고리즘을 선택했을 때만 설정하는 hyperparameter입니다.
  - SAC alpha 값은 가치 함수를 최적화 하는데만 집중하지 않고 agent가 탐험할 수 있는 정도를 의미합니다.
  - Alpha 값이 클수록 탐색을 더 많이 장려하게 됩니다.
- SAC alpha value는 0과 1 사이의 실수값으로 설정할 수 있으며, 디폴트 값은 0.2 입니다.

# Reward Function

# Reward Fuction

- DeepRacer의 reward는 agent의 action에 대한 점수입니다.
  - 점수를 주는 방법은 모델을 만드는 과정에서 reward function을 통해 상세하게 정의할 수 있습니다.





# Reward Fuction

- Python 언어를 사용하여 reward function을 작성할 수 있습니다.
  - 처음 모델을 만들 때는 코드를 직접 작성하는게 어려울 수 있으니 예시 함수들을 사용하는 것을 권장합니다.
  - [Reward function examples](#) 버튼을 클릭하면 4개의 예시 보상함수를 확인할 수 있습니다.

## Reward function [Info](#)

The reward function describes immediate feedback (as a score for reward or penalty) when the vehicle takes an action to move from a given position on the track to a new position. Its purpose is to encourage the vehicle to make moves along the track to reach its destination quickly. The model training process will attempt to find a policy which maximizes the average total reward the vehicle experiences. [Learn more](#) about the reward function and the reward input parameters you can use in your function.

Code editor

Reward function examples

Reset

Validate

# Reward Fuction - Follow the center line

- Time trial - follow the center line

- Race type을 time trial로 선택했을 때 사용할 수 있는 보상함수입니다.
- 트랙의 중앙선을 잘 따라갈수록 보상을 받도록 설계되어 있습니다.

## ▼ Time trial - follow the center line (Default)

This example determines how far away the agent is from the center line and gives higher reward if it is closer to the center of the track. It will incentivize the agent to closely follow the center line.

[Use code](#)

```
1 def reward_function(params):
2     '''
3     Example of rewarding the agent to follow center line
4     '''
5
6     # Read input parameters
7     track_width = params['track_width']
8     distance_from_center = params['distance_from_center']
9
10    # Calculate 3 markers that are at varying distances away from the center line
11    marker_1 = 0.1 * track_width
12    marker_2 = 0.25 * track_width
13    marker_3 = 0.5 * track_width
14
15    # Give higher reward if the car is closer to center line and vice versa
16    if distance_from_center <= marker_1:
17        reward = 1.0
18    elif distance_from_center <= marker_2:
19        reward = 0.5
20    elif distance_from_center <= marker_3:
21        reward = 0.1
22    else:
23        reward = 1e-3 # likely crashed/ close to off track
24
25    return float(reward)
```

# Reward Fuction - Follow the center line

---

- 코드를 자세하게 살펴보겠습니다.
- 사용할 변수는 **track\_width** 와 **distance\_from\_center** 입니다.
  - **track\_width** 는 트랙의 너비를 의미하고 **distance\_from\_center** 는 트랙의 중앙과 agent 사이의 거리를 의미합니다.

```
6 # Read input parameters
7 track_width = params['track_width']
8 distance_from_center = params['distance_from_center']
```

# Reward Fuction - Follow the center line

- 트랙의 너비를 기준으로 트랙을 3단계로 구분합니다. **marker\_1** 은 트랙의 10%, **marker\_2** 는 트랙의 25%, **marker\_3** 는 트랙의 50% 입니다.

```
10 # Calculate 3 markers that are at varying distances away from the center line
11 marker_1 = 0.1 * track_width
12 marker_2 = 0.25 * track_width
13 marker_3 = 0.5 * track_width

15 # Give higher reward if the car is closer to center line and vice versa
16 if distance_from_center <= marker_1:
17     reward = 1.0
18 elif distance_from_center <= marker_2:
19     reward = 0.5
20 elif distance_from_center <= marker_3:
21     reward = 0.1
22 else:
23     reward = 1e-3 # likely crashed/ close to off track
```

## Reward Function - Follow the center line

---

- Agent와 트랙의 중앙과의 거리가 트랙의 너비의 10% 이하라면 reward를 1을 받습니다.
- Agent와 트랙의 중앙과의 거리가 트랙의 너비의 10% 보다 크고 25% 이하라면 reward를 0.5를 받습니다.
- Agent와 트랙의 중앙과의 거리가 트랙의 너비의 25% 보다 크고 50% 이하라면 reward를 0.1를 받습니다.
- Agent와 트랙의 중앙과의 거리가 트랙의 너비의 50% 보다 크다면 트랙을 벗어난 것과 같으므로 reward를 0에 가까운 값으로 받습니다.

# Reward Function - Stay inside the two borders

- Time trial - stay inside the two borders

- Race type을 time trial로 선택했을 때 사용할 수 있는 보상함수입니다.
- 트랙의 흰색 경계선 안에 있을 때 보상을 받도록 설계되어 있습니다

## ▼ Time trial - stay inside the two borders

This example simply gives high rewards if the agent stays inside the borders and lets the agent figure out what is the best path to finish a lap. It is easy to program and understand, but will be likely to take longer time to converge.

[Use code](#)

```
1 def reward_function(params):
2     '''
3     Example of rewarding the agent to stay inside the two borders of the track
4     '''
5
6     # Read input parameters
7     all_wheels_on_track = params['all_wheels_on_track']
8     distance_from_center = params['distance_from_center']
9     track_width = params['track_width']
10
11     # Give a very low reward by default
12     reward = 1e-3
13
14     # Give a high reward if no wheels go off the track and
15     # the agent is somewhere in between the track borders
16     if all_wheels_on_track and (0.5*track_width - distance_from_center) >= 0.05:
17         reward = 1.0
18
19     # Always return a float value
20     return float(reward)
```

# Reward Fuction - Stay inside the two borders

---

- 코드를 자세하게 살펴보겠습니다.
- 사용할 변수는 **all\_wheels\_on\_track, distance\_from center, track\_width** 입니다.
  - 새로 추가된 변수는 **all\_wheels\_on\_track** 이며 4개의 바퀴가 모두 트랙 위에 있을 때 **True**, 한 개의 바퀴라도 트랙을 벗어나면 **False** 입니다.

```
6 # Read input parameters
7 all_wheels_on_track = params['all_wheels_on_track']
8 distance_from_center = params['distance_from_center']
9 track_width = params['track_width']
```

# Reward Fuction - Stay inside the two borders

---

- 초기 reward는 아주 작은 값으로 설정합니다.
  - 4개의 바퀴가 모두 트랙 위에 있으면서 트랙의 50%가 agent와 트랙 중앙과의 거리보다 0.05 이상 일 때(차량이 트랙을 벗어나지 않았을 때), reward를 1로 받습니다.

```
11 # Give a very low reward by default
12 reward = 1e-3
13
14 # Give a high reward if no wheels go off the track and
15 # the agent is somewhere in between the track borders
16 if all_wheels_on_track and (0.5*track_width - distance_from_center) >= 0.05:
17     reward = 1.0
```



# Reward Function - Prevent zig-zag

- Time trial - prevent zig-zag
- Race type을 time trial로 선택했을 때 사용할 수 있는 보상함수입니다.
  - 기본적으로 **Follow the center line** 함수와 같이 중앙선을 잘 따라갈수록 보상을 많이 받도록 하고, 각도를 많이 전환하면 보상이 줄어드는 코드가 추가되었습니다.

## ▼ Time trial - prevent zig-zag

This example incentivizes the agent to follow the center line but penalizes with lower reward if it steers too much, which will help prevent zig-zag behavior. The agent will learn to drive smoothly in the simulator and likely display the same behavior when deployed in the physical vehicle.

[Use code](#)

```
1 def reward_function(params):
2     """
3     Example of penalize steering, which helps mitigate zig-zag behaviors
4     """
5
6     # Read input parameters
7     distance_from_center = params['distance_from_center']
8     track_width = params['track_width']
9     abs_steering = abs(params['steering_angle']) # Only need the absolute steering angle
10
11     # Calculate 3 marks that are farther and father away from the center line
12     marker_1 = 0.1 * track_width
13     marker_2 = 0.25 * track_width
14     marker_3 = 0.5 * track_width
15
16     # Give higher reward if the car is closer to center line and vice versa
17     if distance_from_center <= marker_1:
18         reward = 1.0
19     elif distance_from_center <= marker_2:
20         reward = 0.5
21     elif distance_from_center <= marker_3:
22         reward = 0.1
23     else:
24         reward = 1e-3 # likely crashed/ close to off track
25
26     # Steering penalty threshold, change the number based on your action space setting
27     ABS_STEERING_THRESHOLD = 15
28
29     # Penalize reward if the car is steering too much
30     if abs_steering > ABS_STEERING_THRESHOLD:
31         reward *= 0.8
32     return float(reward)
```

# Reward Fuction - Prevent zig-zag

---

- 코드를 자세하게 살펴보겠습니다.
- 사용할 변수는 **steering\_angle, distance\_from center, track\_width** 입니다.
  - 새로 추가된 변수는 **steering\_angle** 이며 회전각을 받아 절댓값을 씌워 **abs\_steering** 변수에 저장합니다.

```
6 # Read input parameters
7 distance_from_center = params['distance_from_center']
8 track_width = params['track_width']
9 abs_steering = abs(params['steering_angle']) # Only need the absolute steering angle
```

# Reward Fuction - Prevent zig-zag

---

- Follow the center line 에서 작성된 코드와 같은 코드입니다.
  - Agent가 트랙 중앙선에 가까울수록 큰 보상을 받고, 멀어질수록 작은 보상을 받게 됩니다.

```
11 # Calculate 3 marks that are farther and father away from the center line
12 marker_1 = 0.1 * track_width
13 marker_2 = 0.25 * track_width
14 marker_3 = 0.5 * track_width
15
16 # Give higher reward if the car is closer to center line and vice versa
17 if distance_from_center <= marker_1:
18     reward = 1.0
19 elif distance_from_center <= marker_2:
20     reward = 0.5
21 elif distance_from_center <= marker_3:
22     reward = 0.1
23 else:
24     reward = 1e-3 # likely crashed/ close to off track
```

# Reward Function - Prevent zig-zag

---

- 회전각의 임계값을 설정하고 설정한 값보다 각도가 커질 때마다 보상의 20%를 줄입니다.
  - 패널티를 받는 것과 유사하기 때문에 회전각을 작게하는 action을 선택할 가능성이 높아집니다.
  - 임계값 설정은 agent의 action space에 따라 조정이 필요합니다.

```
26 # Steering penalty threshold, change the number based on your action space setting
27 ABS_STEERING_THRESHOLD = 15
28
29 # Penalize reward if the car is steering too much
30 if abs_steering > ABS_STEERING_THRESHOLD:
31     reward *= 0.8
```

# Reward Fuction - Stay on one lane and not crashing

---

- Object avoidance and head-to-head - stay on one lane and not crashing
  - Race type을 object avoidance 혹은 head-to-head racing으로 선택했을 때 사용할 수 있는 보상함수입니다.
  - 기본적으로 Stay inside two boaders 함수와 같이 흰색 경계선 안에 있을 때 보상을 받도록 하고, 장애물에 너무 가까워지면 보상이 줄어드는 코드가 추가되었습니다.

# Reward Fuction - Stay on one lane and not crashing

---

- 코드를 자세하게 살펴보겠습니다.

```
7 all_wheels_on_track = params['all_wheels_on_track']
8 distance_from_center = params['distance_from_center']
9 track_width = params['track_width']
10 objects_location = params['objects_location']
11 agent_x = params['x']
12 agent_y = params['y']
13 _, next_object_index = params['closest_objects']
14 objects_left_of_center = params['objects_left_of_center']
15 is_left_of_center = params['is_left_of_center']
```

# Reward Fuction - Stay on one lane and not crashing

---

- 사용할 변수는 `all_wheels_on_track`, `distance_from_center`, `track_width`, `objects_location`, `x,y`, `closest_objects`, `objects_left_of_center`, `is_left_of_center` 입니다.
  - `objects_location`은 장애물 혹은 bot vehicle의 위치(x,y)의 리스트 이고 `x,y`는 agent의 위치입니다.
  - `closest_objects`는 agent로부터 가장 가까운 물체의 객체 자체를 반환합니다.
  - 첫번째 인덱스는 agent 뒤에 있는 물체 중 가장 가까운 물체, 두번째 인덱스는 agent 앞에 있는 물체 중 가장 가까운 물체입니다.
  - `objects_left_of_center`는 물체가 중앙선을 기준으로 왼쪽에 있다면 `True` 오른쪽에 있다면 `False`입니다.
  - `is_left_of_center` 는 agent가 중앙선을 기준으로 왼쪽에 있다면 `True`, 오른쪽에 있다면 `False`입니다.

## Reward Fuction - Stay on one lane and not crashing

---

- Stay inside two borders 에서 작성된 코드와 같은 코드입니다. Agent가 흰색 경계선을 벗어나지 않고 주행할 때 보상을 받게 됩니다. 주행에 대한 보상은 **reward\_lane**에 저장합니다.

```
16 # Initialize reward with a small number but not zero
17 # because zero means off-track or crashed
18 reward = 1e-3
19 # Reward if the agent stays inside the two borders of the track
20 if all_wheels_on_track and (0.5 * track_width - distance_from_center) >= 0.05:
21     reward_lane = 1.0
22 else:
23     reward_lane = 1e-3
```



# Reward Fuction - Stay on one lane and not crashing

---

- 장애물에 대한 보상은 `reward_avoid`에 저장하며 초기값은 1입니다.
- Agent와 장애물 사이의 거리를 측정하기 위해 필요한 계산을 구현합니다.
  - Agent의 앞에 있는 물체 중에 가장 가까운 물체가 `next_object_index`에 저장되어 있고 이 객체로 `objects_location`을 인덱싱하면, agent의 앞에 있는 물체 중에 가장 가까운 물체의 위치 값 x, y를 반환하게 됩니다.
  - Agent의 위치 값인 `agent_x, agent_y` 를 활용하여 agent와 물체 사이의 거리를 계산합니다. 거리 계산은 유클리드 거리 계산 방법을 활용합니다.

```
24 # Penalize if the agent is too close to the next object
25 reward_avoid = 1.0
26 # Distance to the next object
27 next_object_loc = objects_location[next_object_index]
28 distance_closest_object = math.sqrt((agent_x - next_object_loc[0])**2 + (agent_y - next_object_loc[1])**2)
```

# Reward Fuction

---

- Agent와 물체가 같은 레인에 있는지 확인합니다.
  - 물체와 agent가 트랙 중앙선을 기준으로 둘다 왼쪽에 있거나, 둘다 오른쪽에 있는 경우 `is_same_lane`은 `True`를 갖게 됩니다.

```
29 # Decide if the agent and the next object is on the same lane
30 is_same_lane = objects_left_of_center[next_object_index] == is_left_of_center
```

# Reward Fuction

- Agent와 물체가 같은 레인에 있다면 **distance\_closest\_object**를 통해 패널티를 줍니다. **distance\_closest\_object**가 0.5보다 크거나 같고 0.8보다 작으면 reward를 50% 감소시키고, 0.3보다 크거나 같고 0.5보다 작으면 reward를 80% 감소시킵니다. 그보다 더 가까워지면 충돌 상황으로 가정하고 아주 작은 값으로 reward를 변경합니다.

```
31 ▾ if is_same_lane:
32 ▾     if 0.5 <= distance_closest_object < 0.8:
33 ▾         reward_avoid *= 0.5
34 ▾     elif 0.3 <= distance_closest_object < 0.5:
35 ▾         reward_avoid *= 0.2
36 ▾     elif distance_closest_object < 0.3:
37 ▾         reward_avoid = 1e-3 # Likely crashed
```

# Reward Fuction

---

- 이렇게 설계한 두 가지의 reward인 **reward\_lane**와 **reward\_avoid**의 값을 적절한 비율로 조합하여 최종 reward를 저장합니다.

```
38 # Calculate reward by putting different weights on
39 # the two aspects above
40 reward += 1.0 * reward_lane + 4.0 * reward_avoid
41 return reward
```

# Reward Parameter

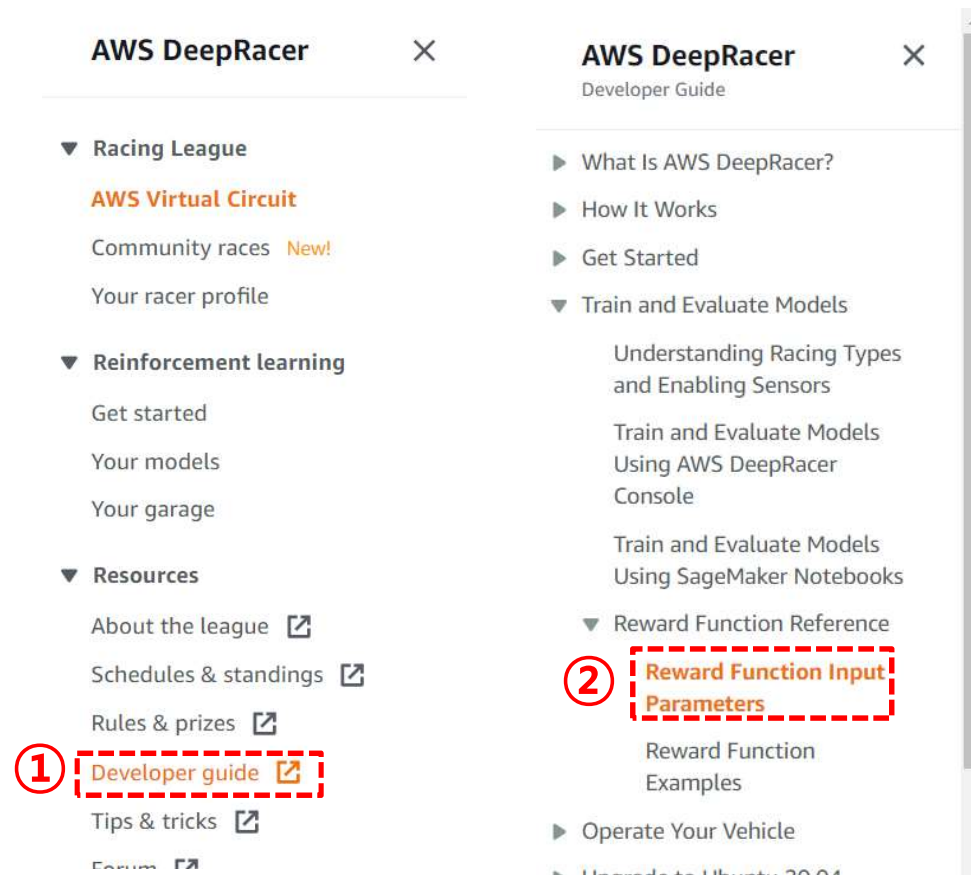
# Reward Parameter

- Reward Parameter 에 대한 자세한 내용은 아래 링크에 있습니다.

<https://docs.aws.amazon.com/deepracer/latest/developerguide/deepracer-reward-function-input.html>

## 링크 경로 추적 방법

- ① AWS DeepRacer 콘솔 메뉴에서  
Resources  
-> Developer guide  
클릭
- ② Developer guide 메뉴에서  
Train and Evaluate Models  
-> Reward Function Reference  
-> Reward Function Input Parameters  
클릭



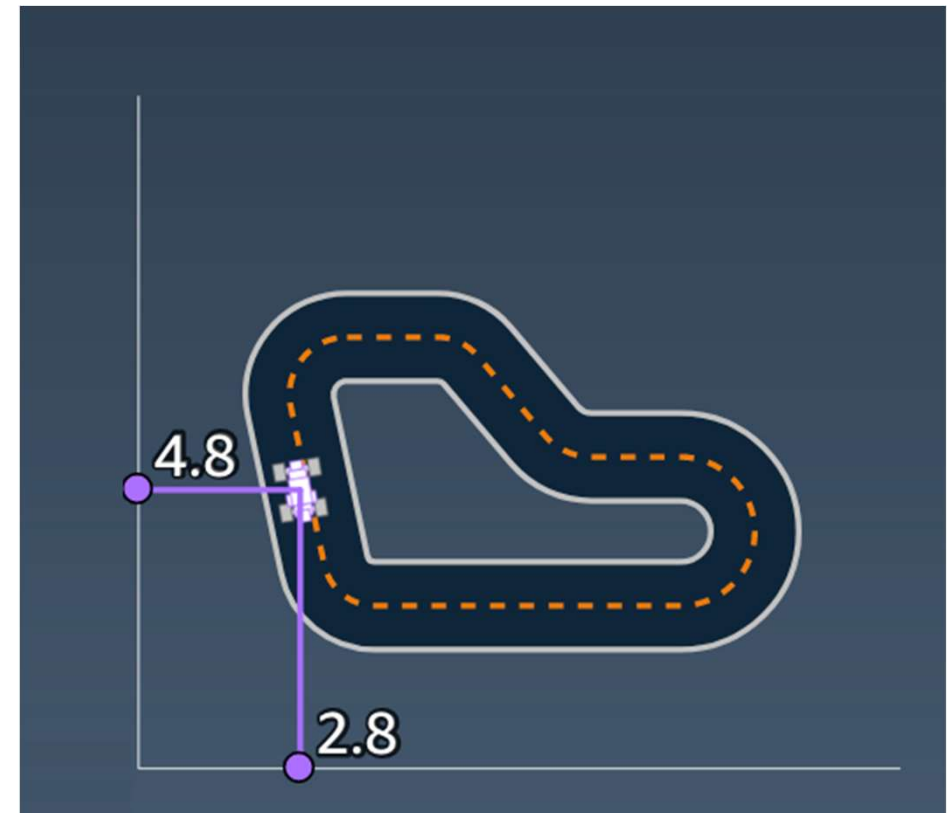
# Reward Parameter

---

- 보상 함수에 사용할 수 있는 parameter들에 대해서 알아보겠습니다. 크게 agent에 대한 변수, track에 대한 변수, object(장애물)에 대한 변수로 나누어 살펴보겠습니다.

# Reward Parameter - Agent

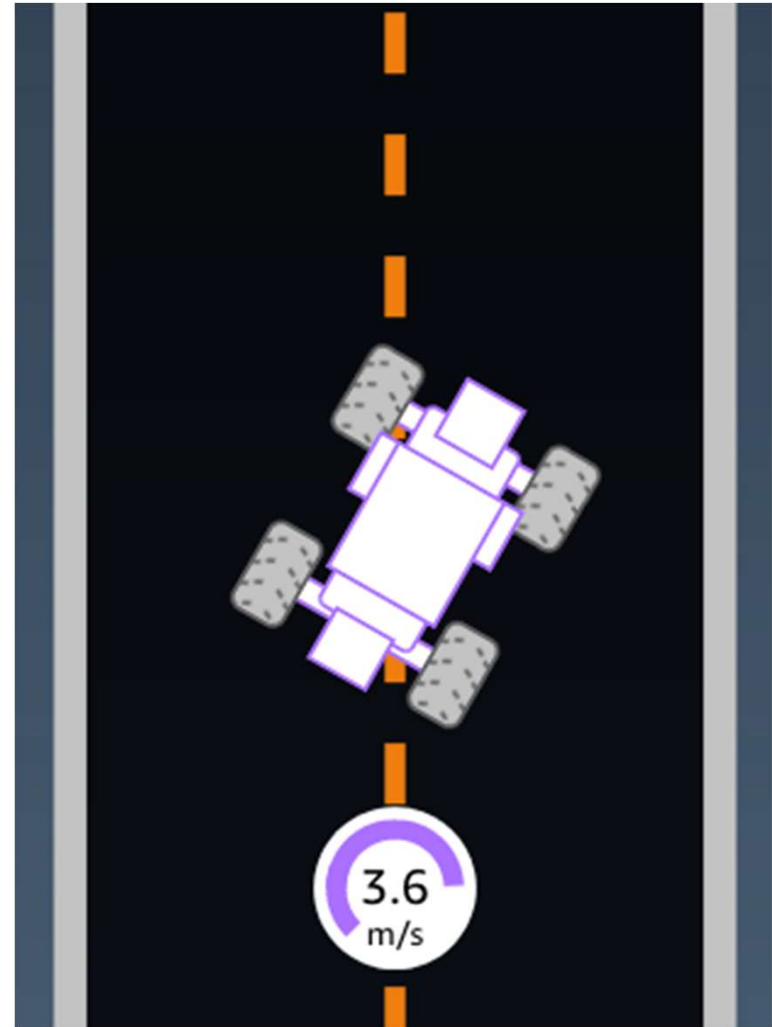
- $x, y$
- Type: float
- 트랙이 포함된 시뮬레이션 환경에서 x축과 y축에 따른 agent의 중앙 위치(m)를 나타냅니다. 원점은 왼쪽 하단 모서리입니다.





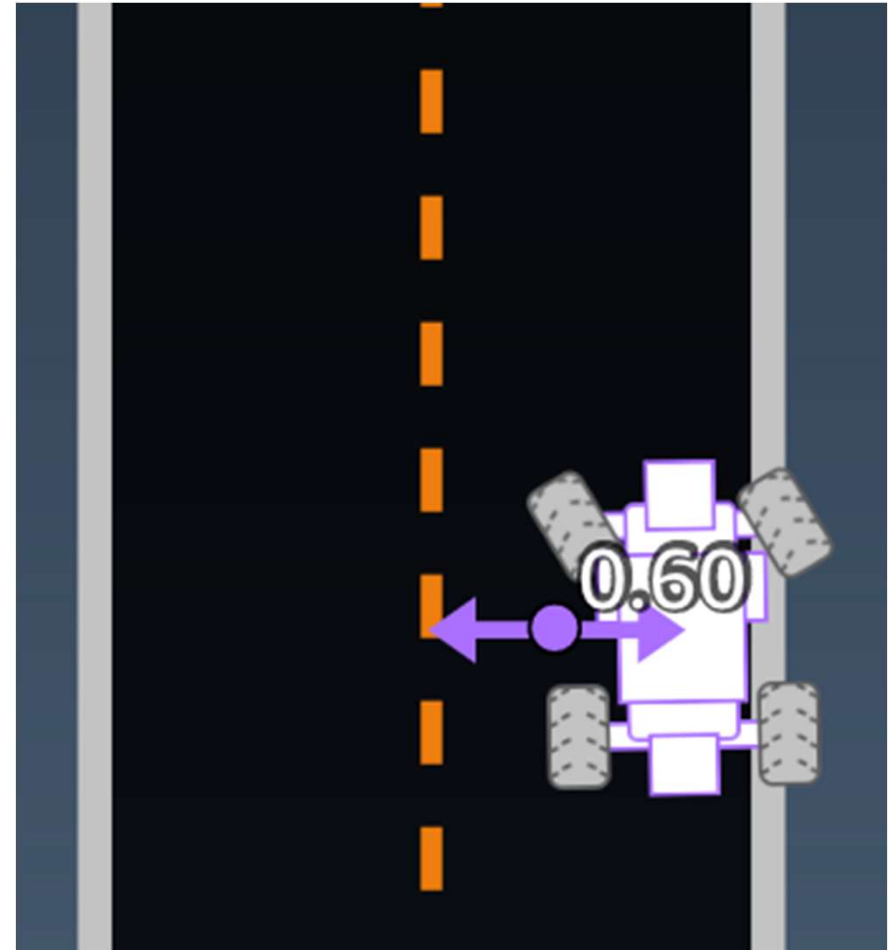
# Reward Parameter - Agent

- **speed**
- Type: float
- 범위: 0.0~5.0
- Agent의 속도(m/s)입니다.



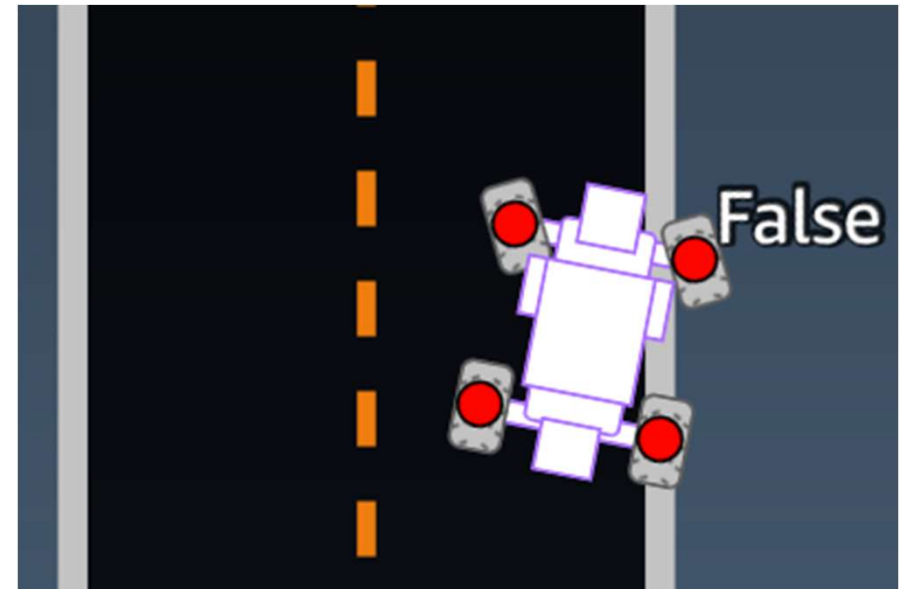
# Reward Parameter - Agent

- **distance\_from\_center**
- Type: float
- 범위:  $0 \sim \text{track\_width}/2$
- Agent의 중앙과 트랙의 중앙 사이의 거리 (m)를 나타냅니다.
  - Agent의 한 쪽 바퀴가 트랙의 흰색 경계선을 벗어났을 때 최댓값을 가집니다.
  - 이 때의 거리는 약 트랙의 너비/2 입니다.



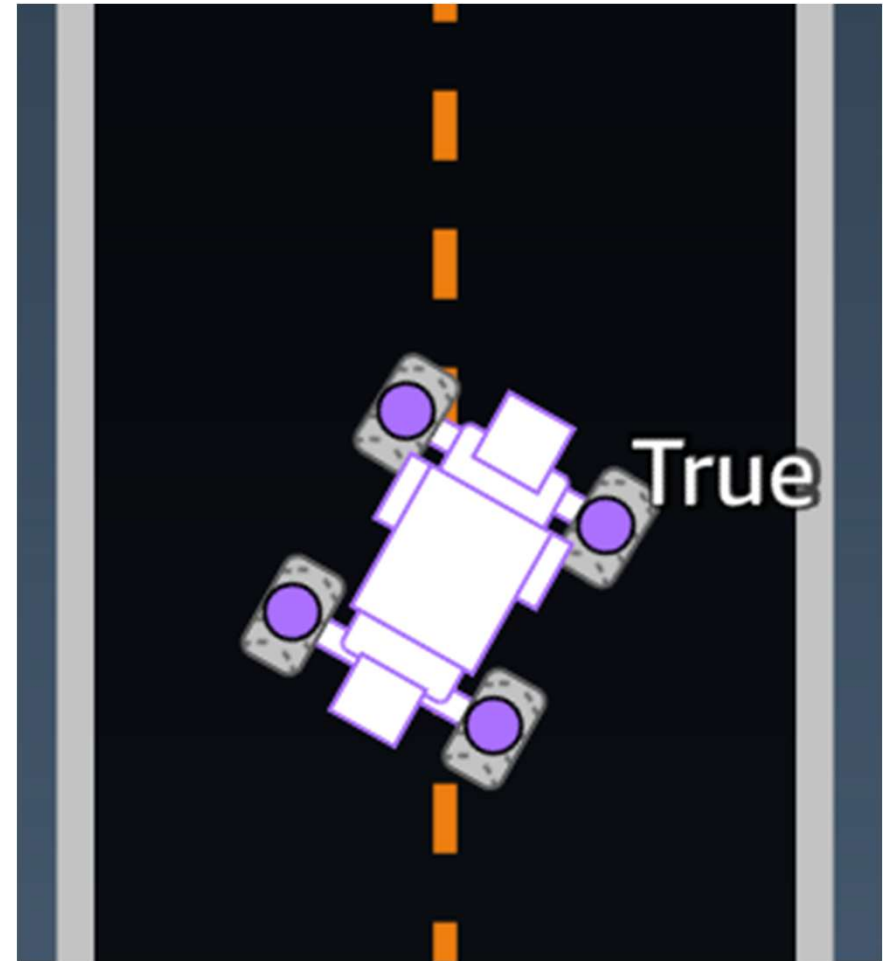
# Reward Parameter - Agent

- **all\_wheels\_on\_track**
- Type: Boolean
- Agent의 트랙 이탈 여부를 확인하는 Boolean 값입니다.
  - 바퀴가 하나라도 트랙의 흰색 경계선을 벗어나면 False입니다.



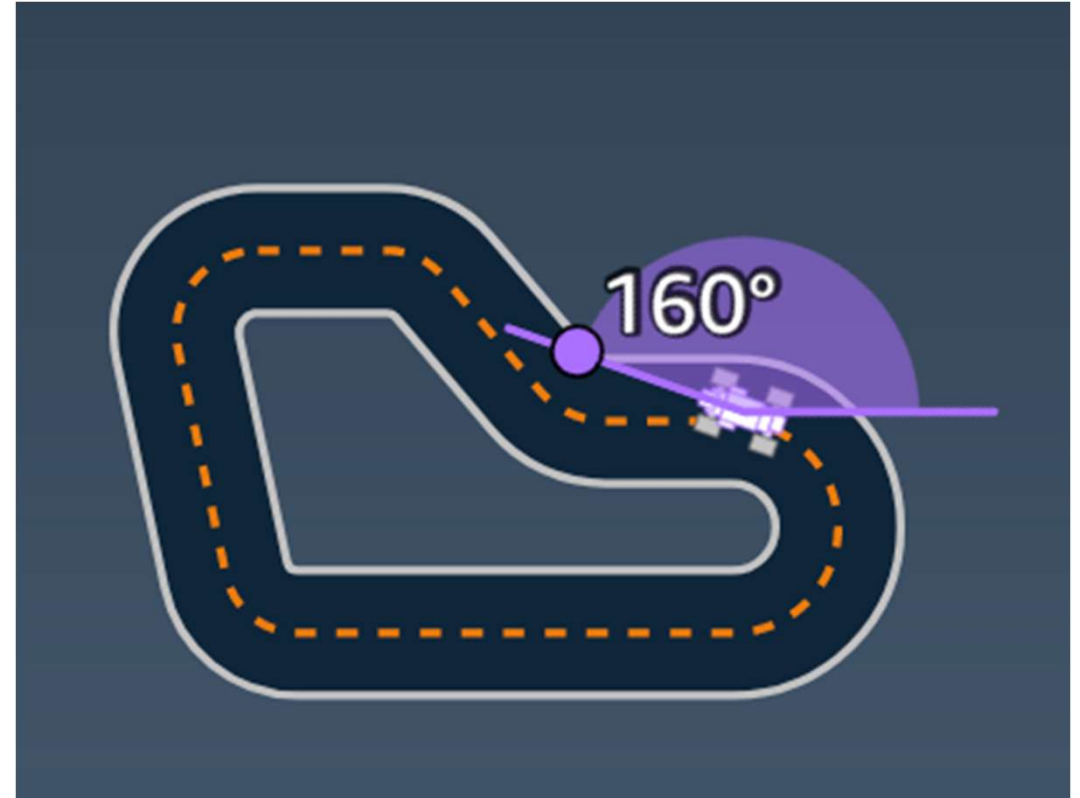
# Reward Parameter - Agent

- 바퀴가 모두 트랙의 경계선 안에 있으면 true입니다.



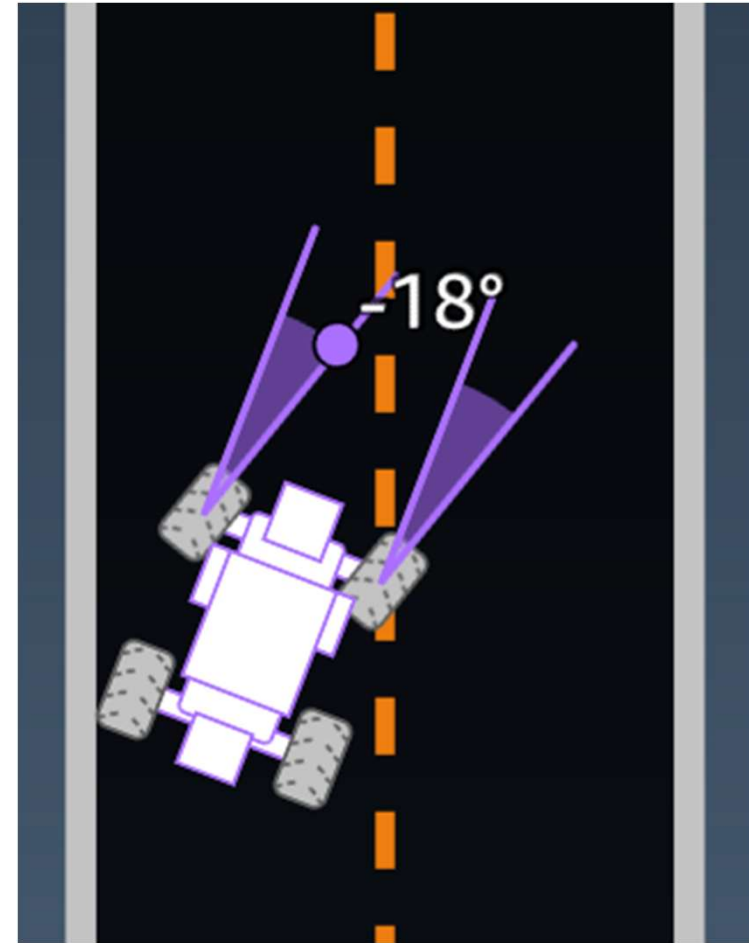
# Reward Parameter - Agent

- **heading**
- Type: float
- 범위: -180~180
- 트랙의 x축에 대한 agent의 진행 방향 (각도)를 나타냅니다.



# Reward Parameter - Agent

- **steering\_angle**
- Type: float
- 범위: -30~30
- Agent의 진행 방향(heading)에 대한 앞바퀴의 각도입니다.
  - 앞바퀴가 진행 방향보다 오른쪽으로 꺾여 있으면 음의 값을, 왼쪽으로 꺾여 있으면 양의 값을 가집니다.



# Reward Parameter - Agent

---

- **steps**
- Type: int
- 완료한 step의 개수를 나타냅니다.
  - Step은 agent가 state 정보를 입력 받고 현재의 정책에 따라 취할 하나의 행동을 결정하는 일련의 과정 한 번을 의미하며, 하나의 경험과 같은 의미입니다.

# Reward Parameter - Agent

---

- **progress**
- Type: float
- 범위: 0~100
- 트랙을 얼마나 주행했는지를 퍼센트로 나타냅니다.
  - 100이라면 트랙을 완주했음을 의미합니다.



# Reward Parameter - Agent

---

- **is\_crashed**
- Type: Boolean
- Agent와 다른 object와의 충돌 여부로, 충돌했다면 True 충돌하지 않았다면 False를 갖습니다.

# Reward Parameter - Agent

---

- **is\_left\_of\_center**
- Type: Boolean
- Agent가 트랙 중앙에서 왼쪽에 있으면 True, 오른쪽에 있으면 False를 갖는 변수입니다.

# Reward Parameter - Agent

---

- **is\_offtrack**
- Type: Boolean
- Agent가 종료 시점에 트랙을 벗어났으면 True, 벗어나지 않았으면 False를 갖는 변수입니다.

# Reward Parameter - Agent

---

- **is\_reversed**
- Type: Boolean
- Agent의 바퀴가 시계 방향으로 주행하면 True, 시계 반대 방향으로 주행하면 False를 나타내는 변수입니다.

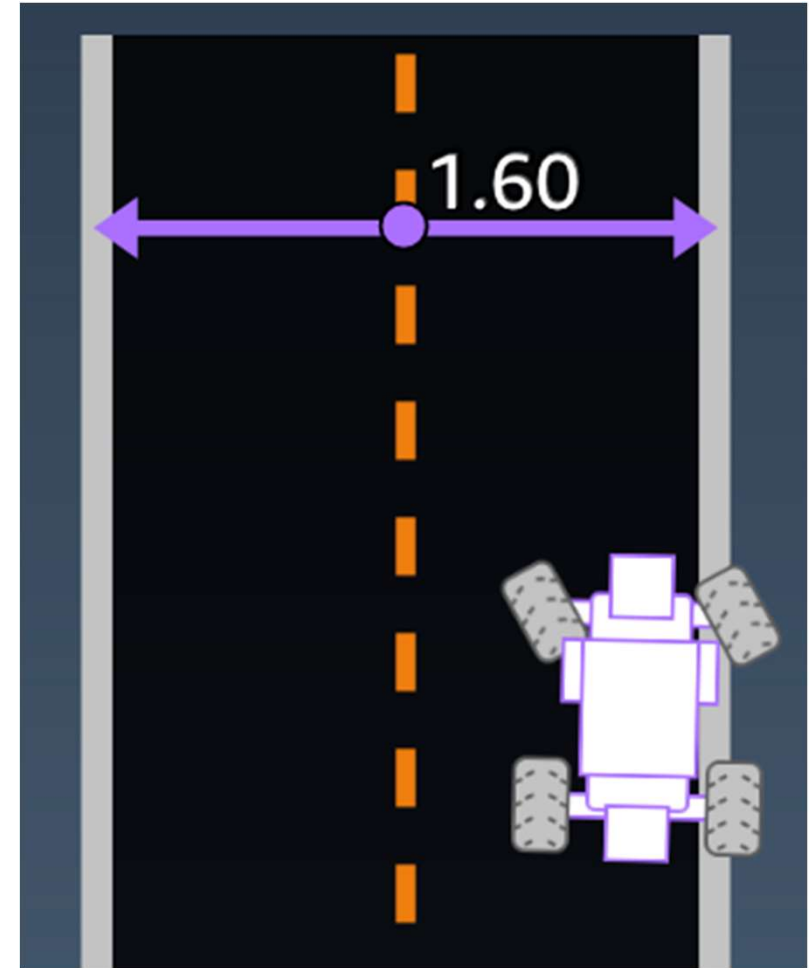
# Reward Parameter - Track

---

- **track\_length**
- Type: float
- 트랙의 길이(m)입니다.
  - 트랙의 시작점으로부터 중앙선을 따라 다시 시작점으로 돌아가기까지의 거리를 의미합니다.

# Reward Parameter - Track

- **track\_width**
- Type: float
- 트랙의 너비(m)입니다.
  - 트랙의 흰색 경계선 사이 거리를 의미합니다.



# Reward Parameter - Track

---

- **waypoints**
- Type: `[[float, float], [float, float], ...]`
- 트랙 중앙을 따라 정해진 waypoint의 위치를 순서대로 나열한 리스트입니다.
  - 순환 트랙의 경우 시작과 끝 지점의 waypoint는 같습니다.
  - 트랙에 따라 waypoint의 개수는 다릅니다.

# Reward Parameter - Track

---

- **closest\_waypoints**
- Type: [int, int]
- Agent의 현재 위치에서 가장 가깝게 위치한 두 waypoint의 index를 나타냅니다.
  - Agent와의 거리는 유클리드 거리 계산법으로 구합니다.
- 첫 번째 값은 agent의 뒤에서 가장 가까운 waypoint를 나타내고, 두 번째 값은 agent의 앞에서 가장 가까운 waypoint를 나타냅니다.



# Reward Parameter - Track

- 가까운 waypoint를 찾을 때의 기준은 agent의 앞바퀴 위치입니다.
  - 다음 그림에서 **closest\_waypoints**는 [16, 17]입니다.



# Reward Parameter - Object

---

- **closest\_objects**
- Type: [int, int]
- Agent의 현재 위치에서 가장 가깝게 위치한 두 object의 index를 나타냅니다.
- 첫 번째 값은 agent의 뒤에서 가장 가까운 object를 나타내고, 두 번째 값은 agent의 앞에서 가장 가까운 object를 나타냅니다.
  - Object가 하나만 있는 경우 두 값은 모두 0이 됩니다.

# Reward Parameter - Object

---

- **objects\_distance**
- Type: [float, ...]
- 트랙의 시작점으로부터 Object들의 거리를 나타냅니다.
  - i번째 요소는 i번째 object의 시작점으로부터의 거리를 의미합니다.

# Reward Parameter - Object

---

- **objects\_heading**
- Type: [float, ...]
- Object들의 heading을 나타냅니다.
  - i번째 요소는 i번째 object의 heading을 의미합니다.
  - 움직이지 않는 object는 heading이 0이 됩니다.

# Reward Parameter - Object

---

- **objects\_left\_of\_center**
- Type: [Boolean, ...]
- Object들이 중앙선을 기준으로 왼쪽에 있는지 오른쪽에 있는지를 나타냅니다.
  - 왼쪽에 있다면 True, 오른쪽에 있다면 False입니다.

# Reward Parameter - Object

---

- **objects\_location**
- Type: `[[float, float], [float, float], ...]`
- Object들의 위치값( $x, y$ )을 나타냅니다.

# Reward Parameter - Object

---

- **objects\_speed**
- Type: [float, ...]
- Object들의 speed를 나타냅니다.
  - 움직이지 않는 object는 speed가 0이 됩니다.

# 과제

---

## 과제!!

**1. 알고리즘, 행동공간, 하이퍼파라미터, Reward Function 등을 (기본 값이 아닌) 본인이 원하는 대로 튜닝하여 훈련시키기**

\* 제약조건

- Agent 의 센서 : Camera 1개
- Race type : Time Trial
- 훈련시간 : 90분

**2. 위에서 훈련 된 모델을 오프라인 차량에 업로드하기**