

Unity를 활용한 강화학습 툴킷

Reinforcement Learning Toolkit with Unity

지도교수 : 강동완

제출일 : 2020년 10월 30일

조장 : 14109347 송한솔

조원 : 14109353 유호균

14109377 조영익



정보통신대학 컴퓨터공학과
서울과학기술대학교

목 차

1. 캡스톤디자인 소개	3
1.1 캡스톤디자인 배경	
1.2 캡스톤디자인 목표	
1.3 인공지능 최근 동향 및 관심	
2. 시스템 아키텍처 개요	4
2.1 개발 환경	
2.2 시스템 아키텍처 구조 및 설명	
2.3 설계 및 제작	
2.4 시험 및 평가	
2.5 조원 편성	
2.6 추진 계획	
3. 캡스톤디자인 구현과정 및 피드백	7
3.1 캡스톤디자인 구현과정 서론	
3.2 캡스톤디자인 초기 구현과정	
3.3 캡스톤디자인 피드백 및 보완내용	
4. 캡스톤디자인 결과 효과	11
5. 사용자 매뉴얼	13
6. 소스코드	22
7. 참고 자료, URL, 문헌	39

캡스톤디자인 보고서 제출

윤리서약서

본인은 서울과학기술대학교 컴퓨터공학과 학생으로 캡스톤디자인 작성과정에서 다음과 같은 윤리의 기본원칙을 준수할 것을 서약합니다.

첫째, 지도교수의 지도를 받아 정직하고 엄정한 캡스톤디자인을 수행하여 학위 졸업작품에 필요한 프로젝트를 작성함.

둘째, 캡스톤디자인 구현시 위조, 변조, 표절 등 학문적 진실성을 훼손하는 어떤 연구 부정행위도 하지 않았음.

개인정보 수집·이용 및 활용 동의서

서울과학기술대학교 컴퓨터공학과 캡스톤디자인 구현과 결과보고서 제출을 위해 아래와 같이 개인정보 수집·이용과 관련한 관계법령에 따라 고지하오니 동의하여 주시기 바랍니다.

1. 개인정보 수집·이용·목적: 캡스톤디자인 구현과 결과보고서 수합, 관리 및 보관
2. 개인정보 수집·이용·항목 : 학번, 성명, 연락처
3. 개인정보 보유 및 이용 기간: 캡스톤디자인 보고서 제출 후 4년
4. 상기 개인정보 수집 및 활용 동의서에 관한 사항에 동의하십니까?

☒ 동의함

☐ 동의하지 않음

5. 개인정보 「민감정보 처리」 규정에 따라 수집하고 활용하고자 합니다.

상기 개인정보 수집 및 활용 동의서에 관한 사항에 동의하십니까?

☒ 동의함

☐ 동의하지 않음

■ 제공된 개인정보는 상기 제시된 목적으로만 수집 및 활용됩니다.

■ 제출된 캡스톤디자인의 내용은 보호되며 사유의 목적으로 이용하지 않고 작성자에게 모든 권한과 책임이 있습니다.

■ 개인정보 보호법에 따라 개인정보를 수집 및 이용에 관하여 거부할 수 있으며, 동의 거부시 캡스톤디자인 결과 등에 불이익이 있을 수 있습니다.

캡스톤디자인 구현과 결과보고서 제출을 위해 위 윤리서약 준수와 개인정보 활용에 아래의 조원(팀)은 동의합니다.

색인번호	구분	성명	서명(성명을 필기체로넣으세요)
202011_0903	조장	송한솔	송한솔
지도교수 강동완	조원	유호균	유호균
		조영익	조영익

2020. 10. 30.

서울과학기술대학교 컴퓨터공학장 귀하

1. 캡스톤 디자인 소개

1.1 캡스톤디자인 배경

최근까지 많은 강화 학습 사례들이 사람들에게 소개가 되고 있지만 실질적으로 강화 학습에 대해서 사람들은 어떤 방식과 원리로 작동을 하는지, 어떤 효과를 보여줄 수 있는지 직접 확인할 수 있는 방법은 제한되어 있다. 강화 학습의 복잡한 이론은 이해하기 어려울 뿐만 아니라 인공지능에 대해서 관심이 있는 사람들도 쉽게 접근하기가 어렵다. 그래서 이번 캡스톤디자인을 통해서 강화학습의 실제 작동하는 코드의 내용을 자세히는 모르더라도 간단한 프로그램을 통해서 사람들이 실제로 강화학습을 통해서 에이전트가 어떻게 동작하고 개발되어 지는지 시각적으로 보여주고 내부적인 코드와 하이퍼 파라미터를 조절하여 간접적으로나마 강화학습을 경험해볼 수 있는 학습용 강화학습 툴킷을 만드는 것이다.

1.2 캡스톤디자인 목표

자동차가 학습하여 주차하는 프로그램을 구현하기 위해 2D/3D의 비디오 게임 개발환경을 지원하는 게임 엔진인 Unity를 이용하여, 주차환경을 구현 하고, 그 속에서 인공지능이 스스로 주차하기에 최적의 경로를 찾아가게끔 학습하는 코드를 작성하여 프로그램을 구현한다. 시뮬레이션 동작을 통해서 학습하는 과정을 사용자가 직접 보고, 학습된 결과물인 인공지능 모델을 다시 실제 모델에 적용해봄으로써 직접 체험을 볼 수 있는 알고리즘과 기능들을 구현한다. 기본적으로 설정되어있는 환경과 보상값 설정 코드 외에 사용자가 직접 환경과 보상값 설정 코드를 수정하여서 학습해 볼 수 있도록 한다.

1.3 인공지능 최근 동향 및 관심

지난 몇 년 간 인공지능에 대한 사람들의 관심이 높아지고 실생활에도 인공지능을 적용한 많은 유용한 시스템과 제품들이 나오고 있다. 세계 정상 수준의 프로선수와 바둑을 해서 이기는 AI 알파고, 자동차 테슬라 회사의 무인주행시스템, 인공지능을 사용하여 사람들의 안면인식을 하는 사이버 보안 시스템 등 실제로도 현재 우리의 삶의 많은 유용한 도움을 주고 사람들 또한 관심을 가지고 있다. 인공지능이라는 분야가 예전부터 있었던 학문이지만 현대에 들어와서 더욱 더 많은 개발이 이루어져 있고 각종 대기업들도 인공지능 분야에 뛰어들고 있다. 컴퓨터관련 학문을 공부를 한 사람이라면 누구나 관심을 가질만한 핫 이슈이고 관련 직종을 수행할 수 있는 인력은 계속 필요하고 산업은 계속해서 발전해 나갈 것이기 때문에, 앞으로의 인공지능은 더욱 더 큰 발전을 이룰 것이다. 지금도 로봇을 이용해서도 집안의 가전기기나 휴대용 전자기기에서도 IOT 기술과 접목이 되고 있다. 많은 과학자들이 인공지능 분야에 집중하고 있으며 앞으로는 대부분의 업무를 기계가 수행한다고 전망하고 있는 것처럼 컴퓨터 공학과를 다니는 학생이라면 인공지능 학문에 대해서 관심을 가지고 공부해야 한다고 생각한다. 그래서 이번 프로젝트 주제를 인공지능 분야로 초점을 맞춰서 설계를 하였다.

2. 시스템 아키텍처 개요

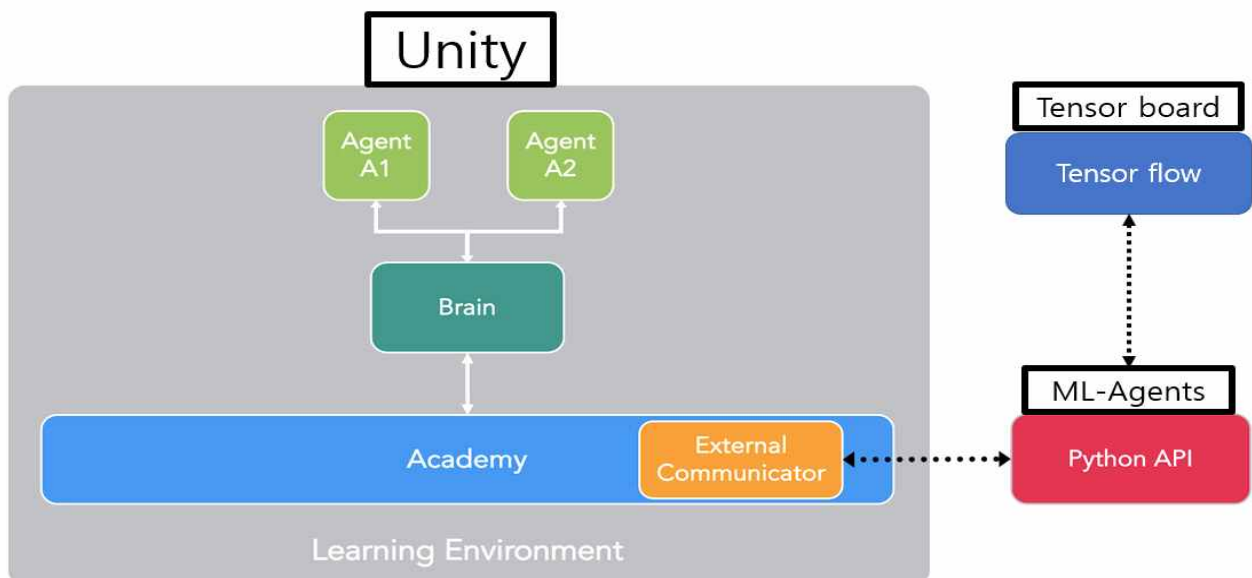
2.1 개발 환경

기본적으로 Unity 게임 엔진과 Unity에서 개발 환경을 제공하는 시뮬레이션 플러그인 ML-Agents 을 사용하였다. 학습 과정 동안에 강화학습의 수학적 계산과 환경 처리를 돕기 위해서 Python과 Tensorflow도 추가 설치하여 환경을 구성해서 진행하였다. 아래에는 각 환경 별 버전이 나와 있다.

- Unity version 2020.1.4f1
- Unity ML-Agents 0.19.0
- Python 3.7.6
- Tensorflow 1.7.1
- conda 5.1.0

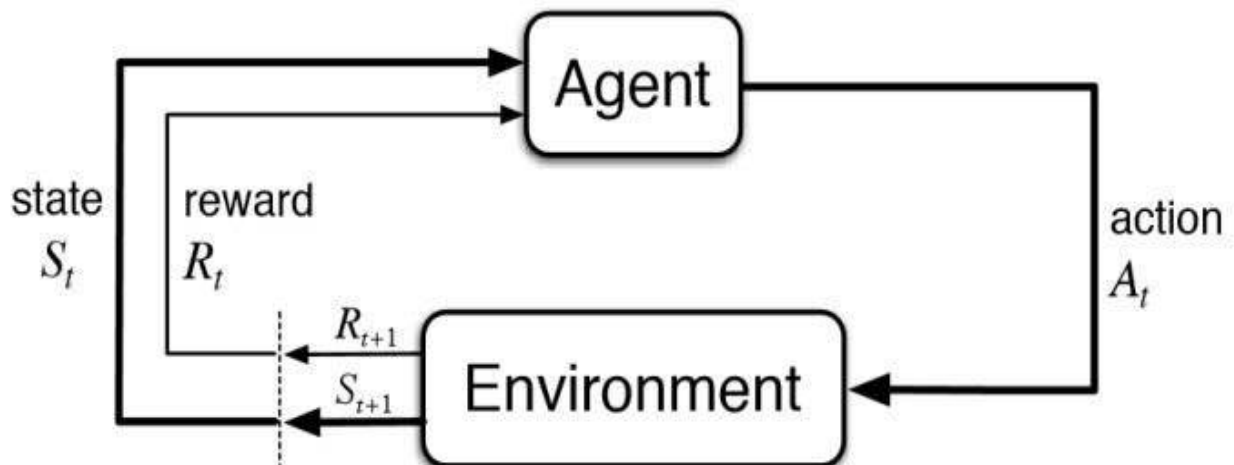
2.2 시스템 아키텍처 구조 및 설명

1) 전체 시스템 구조



먼저 강화 학습하기 이전에 Unity에서 Model 학습 환경과 Agent의 행동 알고리즘, 보상을 구현한다. 이후 Unity와 ML-Agent Plug-In을 서로 연결을 하고 ML-agent 내부에 설치되어 있는 Tensorflow, Python을 이용해서 내부적으로 수학적 계산을 하면서 학습을 시작한다. Brain은 강화학습이 진행 될 때에 올바른 보상을 얻기 위해 Agent가 센서를 통해서 관측값을 입력값으로 넣고 적합한 행동을 수행하기 위한 출력값들을 반복해서 학습을 해나간다. Academy는 전체적으로 Brain과 Agent를 통제하고 미리 설정된 Hyper parameter를 적용하여 학습을 조절한다.

2) Unity 내부적 구조



인공지능 강화학습 이론에 따라서, Unity를 통해서 학습할 Agent와 Environment를 구성한 후에 각각의 Agent의 행동 곧 state에 따라서 reward를 부여해서 올바르게 학습할 수 있게 유도하는 구조이다. 기본적으로 PPO algorithm을 통해서 각각의 Agent는 최대의 reward를 얻기 위한 행동을 하며 최종적으로는 최대의 reward를 얻기 위한 행동을 하는 모델을 산출한다.

2.3 설계 및 제작

	구 분	연관성	내 용
설계 구성요소	범위결정	O	소프트웨어 주제/동기/개발방향/구현기능 개요 작성
	요구분석	O	요구사항 도출 및 설계
	설 계	O	소프트웨어 구조 설계 및 점검 작성

2.4 시험 및 평가

	구 분	연관성	내 용
설계 구성요소	구 현	O	기능 구현, 인공지능 학습, 시뮬레이션 등
	시험평가	O	사용자설명서 작성

2.5 조원 편성

구분	학번	이름	역할 분담
조장	14109347	송한솔	팀장, 게임 환경 및 AI 구성
조원	14109353	유호균	게임 UI 구성 주차 환경 구성
	14109377	조영익	자동차 주행 구성 및 보상체계 구성

2.6 추진 계획

연구기간(월) 세부계획	4	5	6	7	8	9	10	11	비고
개발 일정 세분화 및 연구									
주차 환경 구현									
주차 환경에 AI 학습 적용									
피드백 및 추가 기능 구현									
테스트 및 버그 수정									

3. 캡스톤디자인 구현과정 및 피드백

3.1 캡스톤디자인 구현과정 서론

이 프로젝트의 구현 초기에는 강화학습을 테스트하는 주차 게임이라는 컨셉에 초점을 맞추어 개발을 진행하였고, 초기 구현과정에서는 최대한 주차 환경을 구성하고 그에 맞추어 강화학습을 진행할 예정이었다. 그러나 개발과정에서 ML-agent의 재밌는 점을 더 활용하고 개발 시간도 단축하고자 강화학습을 활용한 주차 게임에서 강화학습 학습용 툴킷으로 방향을 틀게 되었다. 본 단락에서는 초기 구현 과정에서부터 그 이후의 피드백과 변화 과정을 순서대로 서술했다,

3.2 캡스톤디자인 초기 구현과정

1) 주차 환경

초기 개발 단계에서는 계획에 맞추어 단순히 자동차가 고정된 위치의 드럼통 목표 지점에 갈 수 있는 정도로 학습할 것이라 생각하고 개발을 진행하였다. 개발되는 과정에서 단순하게 목표물이 움직이는 것은 쉽게 구현이 가능했지만, 부자연스러운 움직임이 많았다. 그래서 자동차의 자연스러운 움직임을 보여주기 위해서 Unity에서 제공하는 자동차 실제 움직임을 제공하는 Wheel Colliders를 사용하였다.



fig 3.1 개발 초기 단계의 주차 환경

2) 환경 간소화

이후 개발 단계에서는 강화학습 출력 값으로 motor, steer, brake를 주어서 학습을 진행해 나갔고 랜덤한 Agent와 목표를 학습하기 이전에 고정된 위치에서 조금 더 학습에 초점을 맞추

기 위하여 최대한 학습할 환경을 간소화하고 한눈에 보일 수 있도록 설계하였다. 그 과정에서 현실과 동일한 주차 환경 구성을 위해서 건물과 같은 오브젝트들을 추가하였다.



fig 3.2 이후 개발 단계에서의 주차 환경

3) 학습 및 보상

초기에 환경구성을 한 후 간단한 보상 값으로 학습을 시작했다. 초기에는 단순히 방향이 일치하는 정도와 거리에 반비례하는 보상, 그리고 장애물에 부딪히면 음의 보상을 크게 주는 식으로 보상을 주었다. 그 결과로 목표물인 드럼통까지로 가는데 큰 문제가 없이 빠르게 올바른 목표물로 학습이 잘 되었다. 각 환경에 따라 Agent와 목표물을 구현하여 다수의 학습 환경을 가지는 멀티에이전트를 통해 학습 속도를 빠르게 해보는 시도를 하였고, 또한 주차라는 목적에 맞게 목표물을 드럼통에서 특정 구역을 표시하는 식으로 학습을 진행했다.

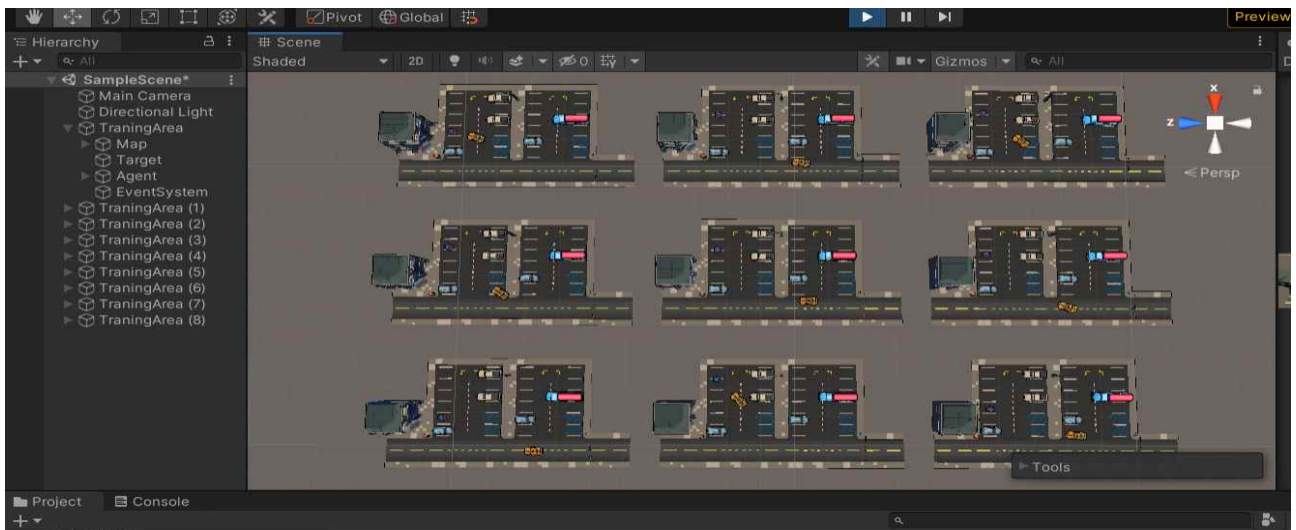


fig 3.3 멀티 에이전트로 학습 진행

3.3 캡스톤디자인 피드백 및 보완내용

1) 주차 환경 개선

기존 주차환경이 단조롭고 툴킷으로서 학습 과정을 보여주기에는 크기가 작다는 피드백을 반영하여, Unity에서 새로운 에셋을 구매하여 주차 환경을 더욱 깔끔하게 구성하였으며, 환경 크기도 같이 확장해서 학습 과정을 더욱 자세히 볼 수 있게 하였다.

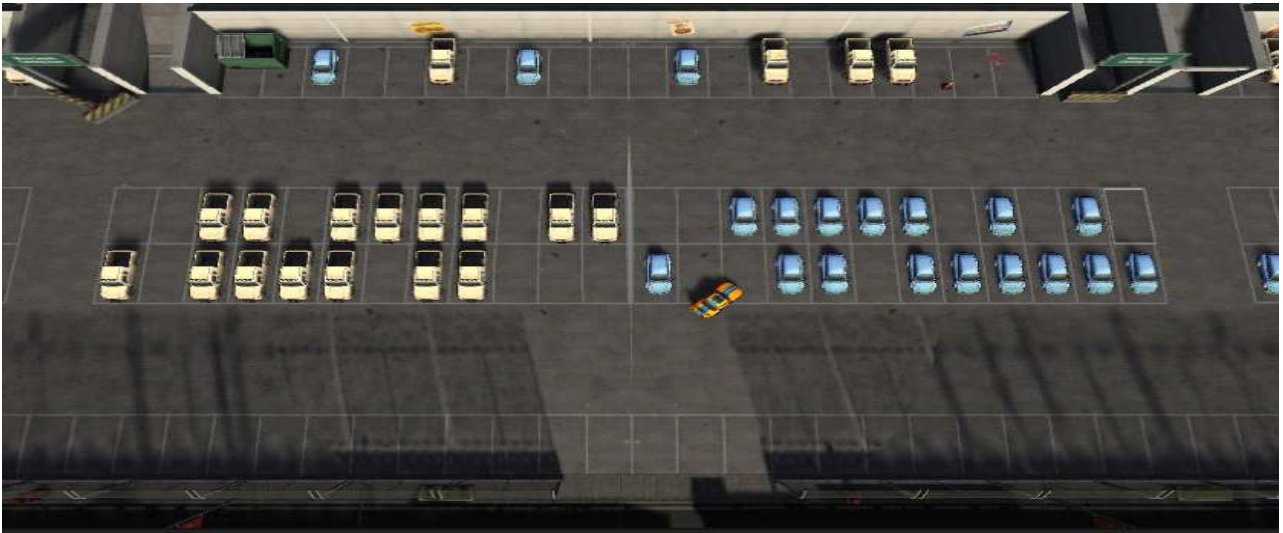


fig 3.4 주차 환경 확장과 시각적 효과 개선

2) 학습 관련 개선

자연스러운 주차 학습을 보여주기 위해 주어진 네모 공간에 얼마나 정확히 주차했느냐에 따른 보상이 필요했고 주차 각도를 통해서 reward를 가중해서 주는 방식을 선택해서 정확한 각도로 자동차를 주차할수록 보상을 더 증가시키고, 장애물과 부딪혔을 때 보상을 더 가중해서 주차할 때 되도록 장애물이 부딪히지 않는 쪽으로 하였다. 또한 넓어진 환경에 맞추어 자동차에 센서를 추가하여 자동차가 그 환경 속에서 목표물을 잘 탐지할 수 있도록 하였다.

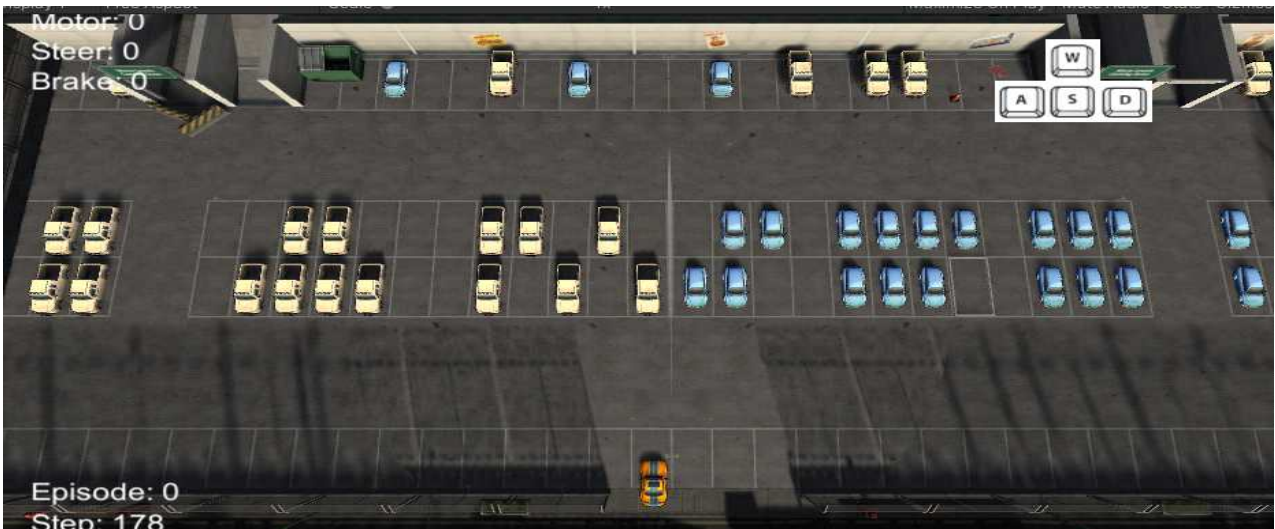


fig 3.5 UI 개선

3) UI 개선

기존에 자동차가 어떻게 움직이는지 정도로만 학습 과정을 보여주는 것 보다는, 실제 학습하는 과정에서 출력 값을 통해 실제 사람이 key board를 눌러서 움직이는 것처럼 보이게 구현을 하였다. 수치적인 부분뿐만이 아니라 실제로 key board로 컴퓨터가 학습을 하는 것을 더욱 시각적으로 보여줄 것 같다는 피드백을 반영해서 Image Button UI를 통해 학습 과정을 더욱 손쉽게 볼 수 있도록 했다. 또한 으로 위에서 내려다보는 환경시점과 일반 아케이드 게임처럼 차를 따라가는 시점을 추가해서 전체적으로 생동감 있게 학습 과정을 체험 할 수 있게 하였다.

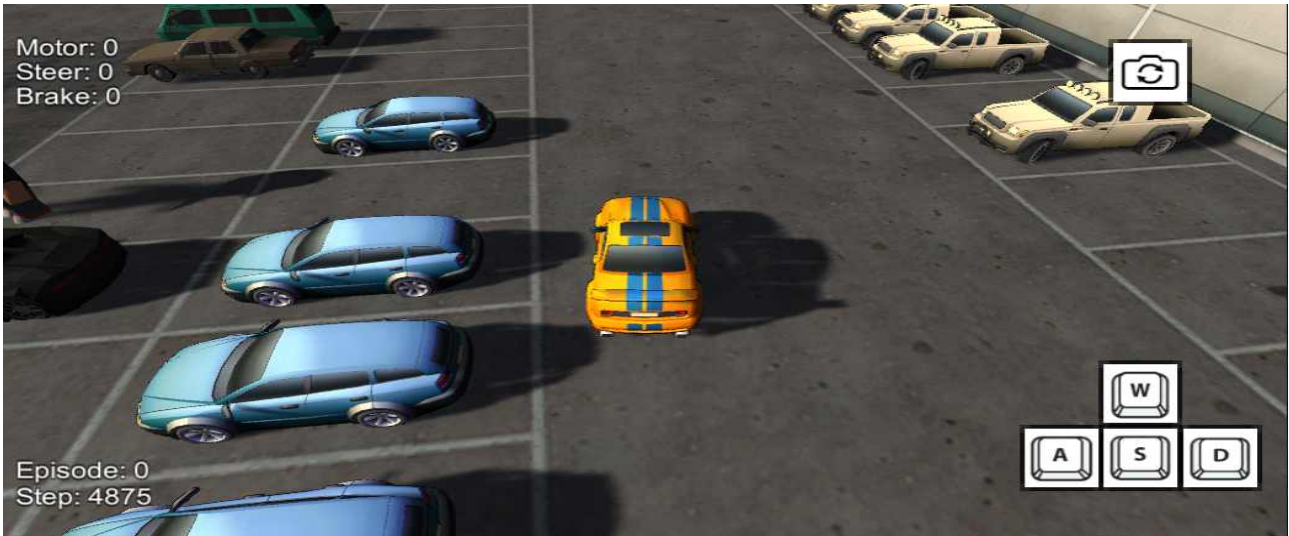


fig 3.6 시점 추가

4. 결과 및 효과

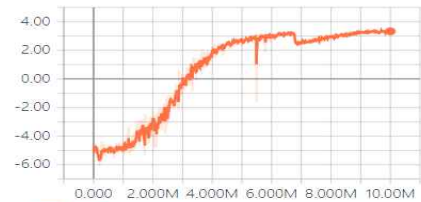
위 프로젝트를 실행해 봄으로 코딩에 대해서 어렵게 생각하고 재미를 못 느끼고 있는 학생들에게 이슈로 떠오르고 있는 인공지능 분야에 대해서 재미를 느끼게 할 수 있다. 또한 내부의 코드나 학습값, 여러 가지 하이퍼 파라미터를 조정해 보면서 이론적으로만 배웠던 인공지능 분야를 실제적으로도 적용해보면서 이해를 넓힐 수가 있다. Unity도 많은 게임 개발자들에게 사용이 되고 취업을 하는데 있어서도 이점을 주는 만큼 Untiy에 대한 이해도도 높이면서 다른 게임 분야 개발까지 해나갈 수 있다. 현재 프로젝트는 자율주차를 학습시키는 것을 목표로 진행을 했었는데 심화적인 단계로 테슬라 회사에서 개발하고 있는 자율주행자동차처럼 실제로 IOT 사물인터넷에 인공지능 알고리즘을 적용시켜 차후 다른 프로젝트를 개발할 수 있다.



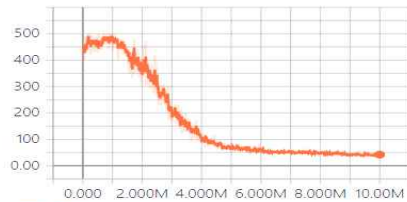
🔍 Filter tags (regular expressions supported)

Environment

Environment/Cumulative Reward

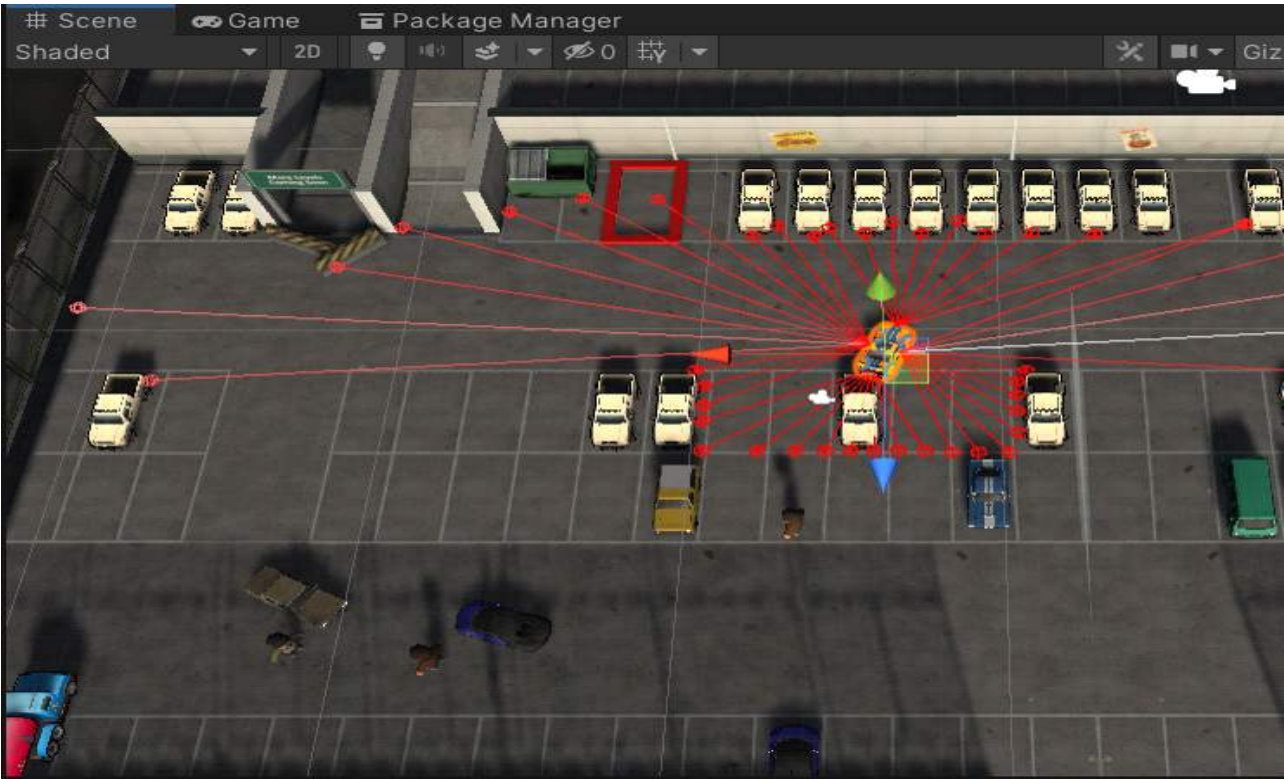
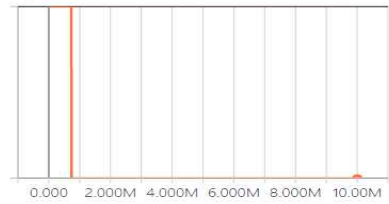


Environment/Episode Length



Is Training

Is Training



5. 사용자 매뉴얼

목차

프로젝트 구성요소

2p

실행

9p

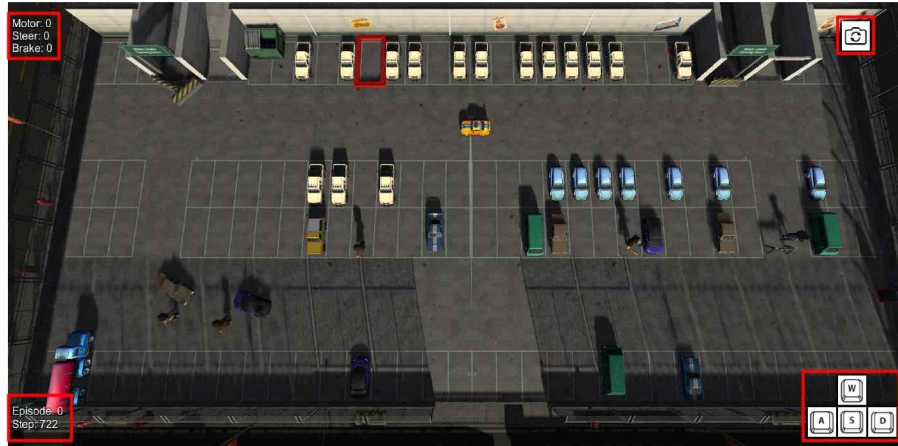
학습 설정

15p

프로젝트 구성요소

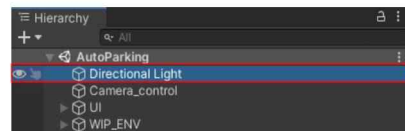
○ Agent에 작용하는 힘

○ 시점 전환 버튼

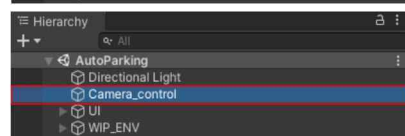


○ 이 환경의 Episode와 Step 수

○ 입력받은 Key



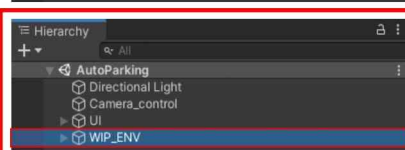
○ 광원 Object



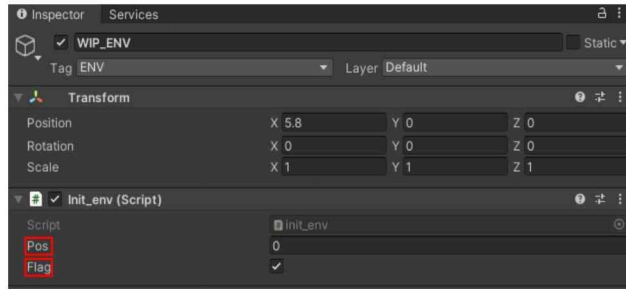
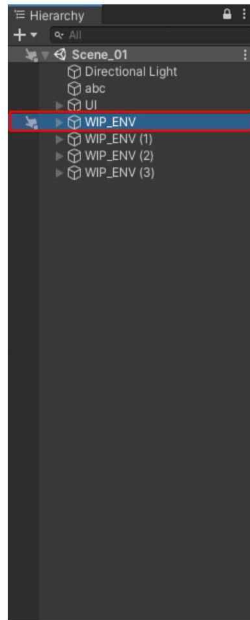
○ 카메라 Script 접근 object



○ UI 관련 ※5p



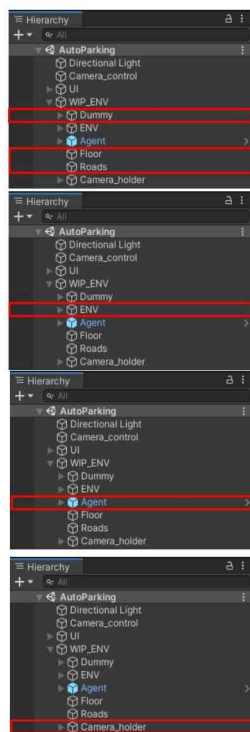
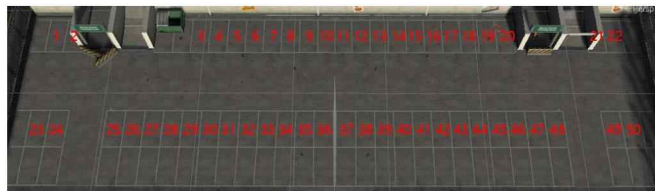
○ 학습 환경 관련



○ Pos: 고정시킬 목표지점의 위치, Flag가 활성화 중이면 무시됨 (1~50)

○ Flag: Episode마다 목표지점의 재배치 유무

▷ Pos값에 따른 목표지점 위치



○ 단순 주차장 구성 요소

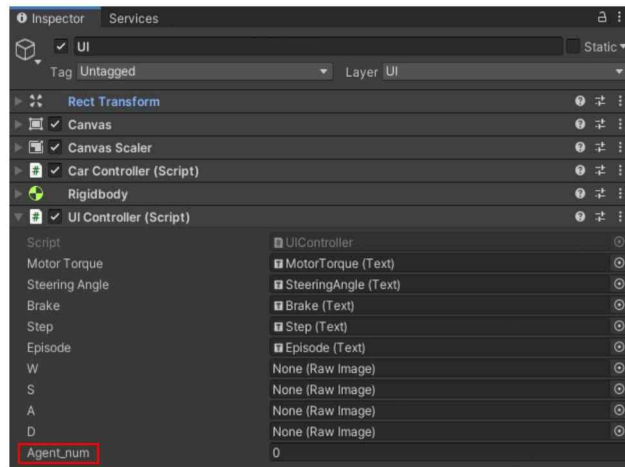
○ 충돌, 주차 성공 등 판정 관련 요소

Obs_ : 충돌/ OUT/ 성공 판정을 갖는 Object 모음
Block : 충돌판정
Parking_SPOT: 주차 공간에 주차된 차/목표지점
OUT : OUT판정
Fence : 주차장 외벽 Object

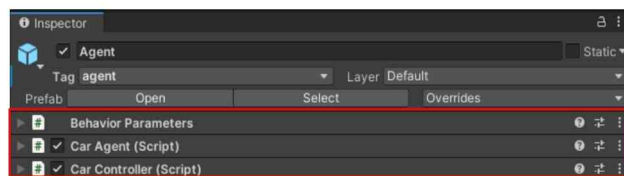
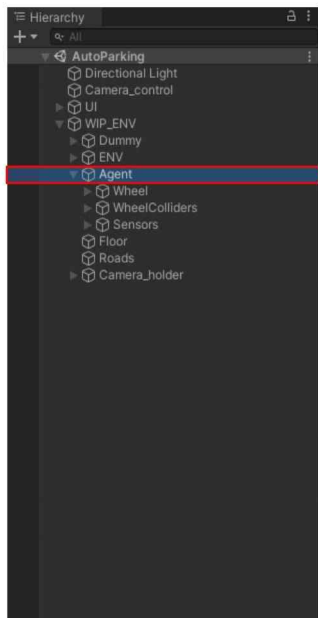
○ Agent 구성요소 ※6p

○ 해당 환경의 카메라 모음.

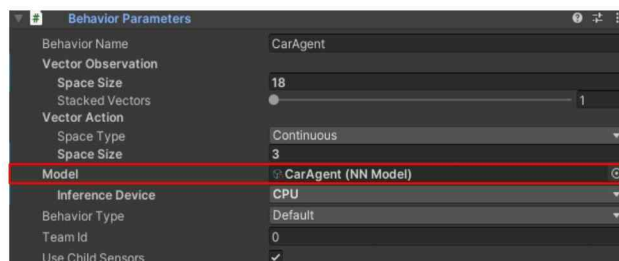
Agent Camera : Agent를 따라다니는 카메라
Main Camera : 해당 환경 Top뷰 시점의 카메라



- Agent_num: 관찰할 환경 번호.
카메라 시점과 UI가 번호에 해당하는 것으로 변경



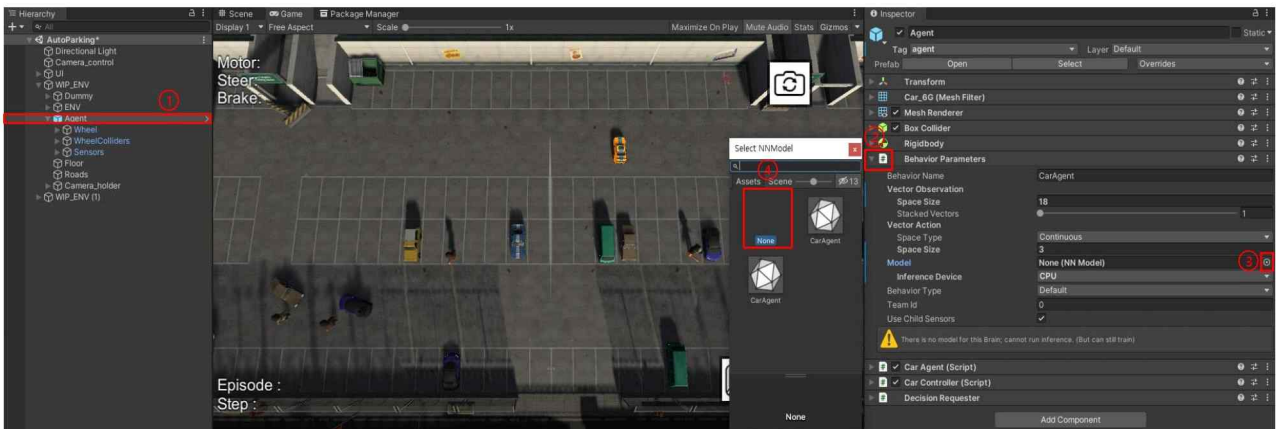
- Car Agent : 보상값 설정 Script
- Behavior Parameters : 학습된 모델 설정
- Car Controller : 자동차 움직임에 관한 Script



실행

직접 Play	10p
학습시켜보기	11p
학습된 모델 Play	13p
학습결과 확인	14p

직접 Play




- ① Hierarchy 뷰의 WIP_ENV 하위 오브젝트인 Agent 선택
- ② Inspector 뷰의 Behavior Parameters 선택
- ③ Model의 오른쪽에 있는 단추모양 버튼 클릭
- ④ None 선택 후 창 닫기

학습시켜보기

- ① ML-agents 환경 활성화 (conda activate)
- ② 프로젝트가 들어있는 디렉토리로 경로 이동
- ③ "식별자명"에 원하는 이름을 입력하고, 실행

```
(ml-agents) >mlagents-learn --train --run-id="식별자명" CarAgent.yaml
```

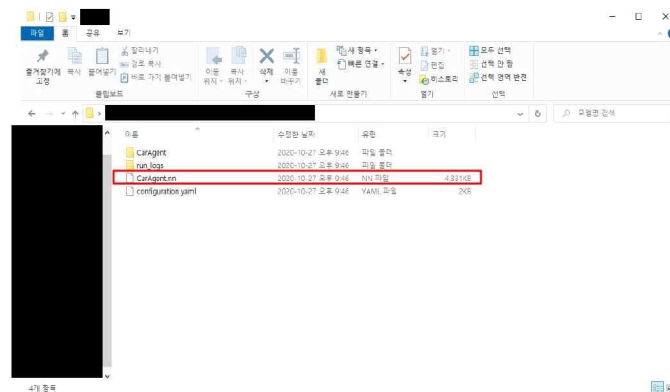


```
at.py:98: disable_resource_variables (from tensorflow.python.ops.variable_scope) is deprecated and will be removed in a future version.
Instructions for updating:
non-resource variables are not supported in the long term

Version information:
ml-agents: 0.19.0,
ml-agents-envs: 0.19.0,
Communicator API: 1.0.0,
TensorFlow: 2.3.1
2020-10-27 21:46:13 WARNING [learn.py:256] The --train option has been deprecated. Train mode is now the default. Use --inference to run i
n inference mode.
2020-10-27 21:46:13 INFO [learn.py:271] run_seed set to 4050
2020-10-27 21:46:14 INFO [tensorflow/stream_executor/platform/default/dso_loader.cc:48] Successfully opened dynamic library cudart64_1
01.dll
WARNING: tensorflow: From C:\Users\ghrb\AppData\Local\Continuum\anaconda3\envs\ml-agents\lib\site-packages\tensorflow\python\compat\v2_comp
at.py:98: disable_resource_variables (from tensorflow.python.ops.variable_scope) is deprecated and will be removed in a future version.
Instructions for updating:
non-resource variables are not supported in the long term
2020-10-27 21:46:16 INFO [environment.py:198] Listening on port 5004. Start training by pressing the Play button in the Unity Editor.
```

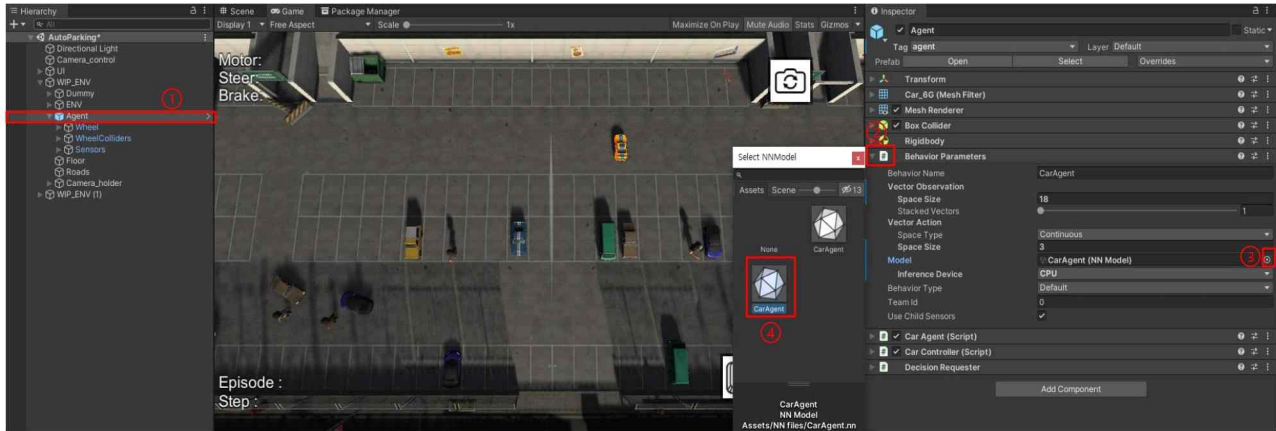
- ④ 유니티 Editor에서 게임 실행 ▶

- ⑤ 학습이 끝날때 까지 기다린다.
- ⑥ 학습이 끝나면, "프로젝트 디렉토리" \Results 경로로 이동



- ⑦ CarAgent.nn파일을 Assets\WNN files로 파일 이동

학습된 모델 Play



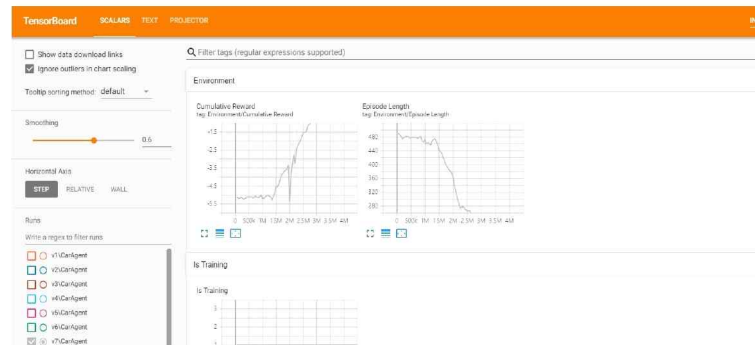
- ① Hierarchy 뷰의 WIP_ENV 하위 오브젝트인 Agent 선택
- ② Inspector 뷰의 Behavior Parameters 선택
- ③ Model의 오른쪽에 있는 단추모양 버튼 클릭
- ④ 학습한 모델 선택 후 창 닫기

학습결과 확인

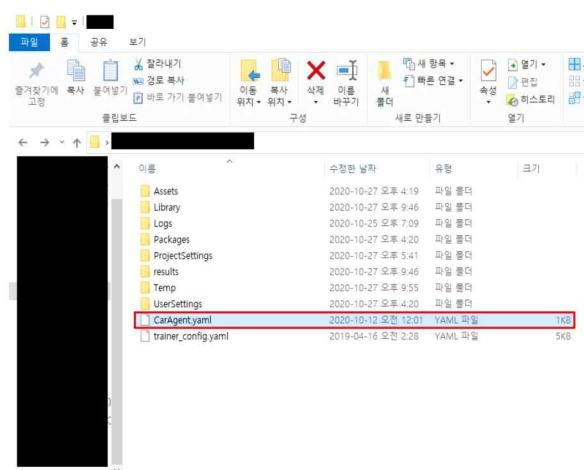
- ① ML-agents 환경 활성화 (conda activate)
- ② 프로젝트가 들어있는 디렉토리로 경로 이동
- ③ tensorboard --logdir results --port 6006

```
(ml-agents) $ tensorboard --logdir results --port 6006
2020-10-28 18:31:59.42651: I tensorflow/stream_executor/platform/default/dso_loader.cc:40] Successfully opened dynamic library cudart64_101.dll
Serving TensorBoard on localhost; to expose to the network, use a proxy or pass --bind_all
TensorBoard 2.3.0 at http://localhost:6006/ (Press CTRL+C to quit)
```

- ③ 웹 브라우저에서 <http://localhost:6006/>로 이동



학습 설정



○ 프로젝트 폴더에 있는 CarAgent.yaml 파일을 연다

```
behaviors:
  default:
    trainer_type: ppo
    hyperparameters:
      batch_size: 512
      buffer_size: 5120
      learning_rate_schedule: linear
      learning_rate: 3.0e-4
    network_settings:
      hidden_units: 512
      normalize: false
      num_layers: 5
      vis_encode_type: simple
    memory:
      memory_size: 512
      sequence_length: 512
    max_steps: 10.0e6
    time_horizon: 64
    summary_freq: 10000
    reward_signals:
      extrinsic:
        strength: 1.0
        gamma: 0.99

CarAgent:
  trainer_type: ppo
  hyperparameters:
    batch_size: 512
    buffer_size: 5120
  network_settings:
    hidden_units: 512
    num_layers: 5
    max_steps: 30.0e6
    time_horizon: 128
```

```

behaviors:
  default:
    trainer_type: ppo
    hyperparameters:
      batch_size: 512
      buffer_size: 5120
      learning_rate_schedule: linear
      learning_rate: 3.0e-4
    network_settings:
      hidden_units: 512
      normalize: false
      num_layers: 5
      vis_encode_type: simple
      memory:
        memory_size: 512
        sequence_length: 512
      max_steps: 10.0e6
      time_horizon: 64
      summary_freq: 10000
      reward_signals:
        extrinsic:
          strength: 1.0
          gamma: 0.99

CarAgent:
  trainer_type: ppo
  hyperparameters:
    batch_size: 512
    buffer_size: 5120
  network_settings:
    hidden_units: 512
    num_layers: 5
    max_steps: 30.0e6
    time_horizon: 128

```

trainer_type: ppo = ppo 알고리즘을 사용

batch_size : 경사하강법의 iteration 동안의 경험 수.
iteration : 한 epoch를 나누어서 실행하는 횟수

buffer_size : Policy를 업데이트 하기 전에 수집 할 경험 수

learning_rate_schedule: 학습률이 시간에 따라 어떻게 변하는지 결정. (linear / constant)

hidden_units : 신경망의 은닉층 (hidden layer)의 유닛(노드) 수

normalize : vector observation 입력에 정규화를 적용하는지 유무

num_layers : 신경망의 은닉층의 개수

vis_encode_type : visual observation을 위한 endoeer type

strength : 유니티에서 만든 환경의 보상설정값에 부여할 배율

gamma : Discount factor

time_horizon : 경험 버퍼(experience replay buffer)에 추가하기 전에 agent가 수행할 step수

※ 더 자세한 설정법은 공식문서 참조

<https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Training-Configuration-File.md>

6. 소스코드

CarAgent.cs

```
using System.Collections;
using System.Collections.Generic;
using System.Net.NetworkInformation;
using UnityEngine;
using Unity.MLAgents;
using Unity.MLAgents.Sensors;
using System.Diagnostics;
using System;
using Debug = UnityEngine.Debug;
using Random = UnityEngine.Random;
```

```
[RequireComponent(typeof(CarController))]
[RequireComponent(typeof(Rigidbody))]
```

```
/*
```

```
Initialize() - 매 학습마다 시작 전에 불러오는 함수
OnEpisodeBegin() - 매 Episode마다 실행되는 함수
CollectObservation() - Agent가 환경에서의 값들을 수집하는 함수
OnActionReceived() - Agent가 수집된 정보를 계산하여, 최대 보상을 얻기 위해 도출해 내는
결과값 함수
Heuristic() - Agent에 사용자의 추정치를 입력하는 함수
OnCollisionEnter() - 충돌판정을 식별하고 보상을 주는 함수
OnTriggerEnter() - 최초로 주차공간에 충돌했을 때 보상 주는 함수
OnTriggerStay() - 주차공간 안에 들어가 목표하는 허용각에 따라 보상 주는 함수
```

```
*/
```

```
public class CarAgent : Agent
{
    public bool onetime = true;
    private float[] _lastActions;
    private Rigidbody car_rigidbody;
    private CarController car_controller;

    public GameObject obs_car;
```

```

public GameObject goal;
public Vector3 Center_of_Agent;

// 자동차 WheelCol관련 변수
public Transform frontDriver, frontPass;
public Transform backDriver, backPass;

float a_angle = 20f;    // (허용각)
int crashcount = 1;    // (장애물 충돌 횟수)
private init_env init_env;

public override void Initialize()
{
    car_rigidbody = GetComponent<Rigidbody>();
    car_controller = GetComponent<CarController>();
    init_env = transform.parent.GetComponent<init_env>();
}

public override void OnEpisodeBegin()
{
    init_env.clear_Parkinglot();
    this.car_rigidbody.velocity = Vector3.zero;
    this.car_rigidbody.angularVelocity = Vector3.zero;
    this.car_controller.CurrentSteeringAngle = 0f;
    this.car_controller.CurrentAcceleration = 0f;
    this.car_controller.CurrentBrakeTorque = 0f;
    this.transform.rotation = Quaternion.Euler(0, Random.Range(90,270), 0);
    this.transform.position = transform.parent.position + new
Vector3(Random.Range(-20f, 20f), 0f, 9f);
    crashcount = 1;
    init_env.Init_Parkinglot();
}

public override void CollectObservations(VectorSensor sensor)
{
    // Agent가 Sensor로 관측값을 받아오는 부분, 총 6개의 센서 3 dimension vector로
    // 구성되어 있으며,
    // Space Size = 3 * 6 = 18

    Vector3 dirToTarget = (goal.transform.position - transform.position).normalized;
    sensor.AddObservation(transform.position.normalized);

```



```

        sensor.AddObservation(this.transform.InverseTransformPoint(goal.transform.position));

sensor.AddObservation(this.transform.InverseTransformVector(car_rigidbody.velocity.normalized)
);

        sensor.AddObservation(this.transform.InverseTransformDirection(dirToTarget));
        sensor.AddObservation(transform.forward);
        sensor.AddObservation(transform.right);
        float velocityAlignment = Vector3.Dot(dirToTarget, car_rigidbody.velocity);
        AddReward(0.001f * velocityAlignment);
    }

    public override void OnActionReceived(float[] vectorAction)
    {
        // 이전 행동값 저장, Accel, Steer, Brake 3개의 값을 인공지능망 결과값으로 출력
        _lastActions = vectorAction;
        car_controller.CurrentSteeringAngle = vectorAction[0];
        car_controller.CurrentAcceleration = vectorAction[1];
        car_controller.CurrentBrakeTorque = vectorAction[2];

        //매초마다 음수의 보상값을 주어서 목표에 빨리 도달하기 위한 보상값
        AddReward(-0.001f);
    }

    public override void Heuristic(float[] actionsOut)
    {
        // 추정치로 입력을 받아서 Agent 행동 결정
        actionsOut[0] = Input.GetAxis("Horizontal");
        actionsOut[1] = Input.GetAxis("Vertical");
    }

    void OnCollisionEnter(Collision collision)
    {
        if (collision.gameObject.CompareTag("Obstacle"))
        {
            AddReward(-0.007f * crashcount);
            crashcount++;
            Debug.Log("obstacle");
        }
        if (collision.gameObject.CompareTag("OUT"))
        {
            Debug.Log("OUT");
        }
    }

```

```

        AddReward(-1f);
        EndEpisode();
    }
}

void OnCollisionExit(Collision collision)
{
    if (collision.gameObject.CompareTag("Obstacle"))
    {
        AddReward(-0.01f);
        Debug.Log("Out of Contact");
    }
}

public void OnTriggerEnter(Collider other)
{
    // 목표하는 지점에 도달하기 위해서 goal_line에 도착하면 보상을 주는 코드
    if (other.transform.CompareTag("goal_line"))
    {
        SetReward(0.001f * ((2000 + init_env.get_angle()) / (20 +
init_env.get_angle())));
        if(init_env.get_angle()<=2*a_angle&& onetime == true )
        {
            Debug.Log("Nice");
            SetReward(0.2f);
            onetime = false;
        }
    }

    // 목표하는 지점에 도달한 후, 차와 주차공간의 각도를 계산해서 보상을 주는 코드
    if (other.transform.CompareTag("goal"))
    {
        Vector3 swap = new Vector3(0, 0, 0); // 임시로 쓰는 변수
        gameobject.position은 x,y,z값을 직접적으로 수정할수 없어서 사용
        Center_of_Agent = frontDriver.position + backPass.position;
        Center_of_Agent = Center_of_Agent * 0.5f; // 차의 중심좌표
        Center_of_Agent.y = 0;
        swap = goal.transform.GetChild(4).transform.position;
        swap.y = 0f;
    }
}

```

```

        SetReward(0.01f * (1000 + init_env.get_angle()) / (10 + init_env.get_angle()));
    }

    if (other.transform.CompareTag("OUT"))
    {
        SetReward(-1.0f);
        EndEpisode();
    }
}

public void OnTriggerStay(Collider other)
{
    // 목표 지점에 들어간 후 정해진 각도에 들어올때까지 판정을 한 뒤에
    // 보상을 주고 에피소드를 끝내는 코드
    if (other.transform.CompareTag("goal"))
    {
        Debug.Log("판정중");
        Vector3 swap = new Vector3(0, 0, 0); // 임시로 쓰는 변수
        gameobject.position은 x,y,z값을 직접적으로 수정할수 없어서 사용
        Center_of_Agent = frontDriver.position + backPass.position;
        Center_of_Agent = Center_of_Agent * 0.5f; // 차의 중심좌표
        Center_of_Agent.y = 0;
        swap = goal.transform.GetChild(4).transform.position;
        swap.y = 0f;

        if (Vector3.Distance(Center_of_Agent, swap) <= 0.5f)
        {
            SetReward(0.03f * (1000 + init_env.get_angle()) / (10 +
init_env.get_angle()));
            EndEpisode();
        }
    }
}
}
}

```

CarController.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CarController : MonoBehaviour
{

    public List<AxleInfo> axleInfos;
    public float maxMotorTorque;
    public float maxSteeringAngle;
    public float maxBrakeTorque;

    // 현재 Accel 값 설정
    public float CurrentAcceleration
    {
        get => m_currentAcceleration;
        set => m_currentAcceleration = Mathf.Clamp(value, -1f, 1f);
    }

    // 현재 Brake 값 설정
    public float CurrentBrakeTorque
    {
        get => m_currentBrakeTorque;
        set
        {
            m_currentBrakeTorque = value <= 0.8f ? 0f : value;
        }
    }

    // 현재 Steer 값 설정
    public float CurrentSteeringAngle
    {
        get => m_currentSteeringAngle;
        set => m_currentSteeringAngle = Mathf.Clamp(value, -1f, 1f);
    }

    private float m_currentSteeringAngle = 0f;
    private float m_currentAcceleration = 0f;
    private float m_currentBrakeTorque = 0f;
```

```

// Input값을 Steer, Accel, Brake 순으로 가져옴
public double[] CurrentInputs
{
    get { return new double[] { m_currentSteeringAngle, m_currentAcceleration,
m_currentBrakeTorque }; }
}

// 변경되는 Steer, Accel, Brake 값 객체에 반영
public void FixedUpdate()
{
    float motor = maxMotorTorque * m_currentAcceleration;
    float steering = maxSteeringAngle * m_currentSteeringAngle;
    float brake = maxBrakeTorque * m_currentBrakeTorque;

    foreach (AxleInfo axleInfo in axleInfos)
    {
        if (axleInfo.steering)
        {
            axleInfo.leftWheel.steerAngle = steering;
            axleInfo.rightWheel.steerAngle = steering;
        }
        if (axleInfo.motor)
        {
            axleInfo.leftWheel.motorTorque = motor;
            axleInfo.rightWheel.motorTorque = motor;
        }

        if (axleInfo.brake)
        {
            axleInfo.leftWheel.brakeTorque = brake;
            axleInfo.rightWheel.brakeTorque = brake;
        }
    }
}

// 차량 전단부, 후단부로 나눠서 Wheel, motor 등 조절
[System.Serializable]
public class AxleInfo
{

```

```

    public WheelCollider leftWheel;
    public WheelCollider rightWheel;
    public bool motor;
    public bool steering;
    public bool brake;
}

```

init_env.cs

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

```

```

/*
 *
 * Init_Parkinglot()      -      에피소드가 시작할때 호출되어, 주차된 차/ 목표지점을 초기화해
주는 함수
 *      set_target()      -      목표지점을 기능적으로 "목표" 기능을 하게 도와주는 함수
 *      geta_angle()      -      각을 0~90범위로 변환
 *      GetAngle()        -      각도를 구하는 함수. (원점에서부터 각 점까지의 직선, 그 사이
각)
 *      cal_angle()        -      차의 주차 각도를 재는 함수
 * clear_Parkinglot()      -      에피소드가 시작할 때 제일먼저 호출되어 주차된 차/목표지점이
있다면 제거하는 함수
 *
 *
 */

```

```

public class init_env : MonoBehaviour
{

    private GameObject obs_car;
    private List<GameObject> slot = new List<GameObject>();
    GameObject child = null;
    private CarAgent car_agent;

    [SerializeField]
    public float Pos;
    [SerializeField]
    bool flag;
}

```

```

// Start is called before the first frame update
void Start()
{
    car_agent = this.transform.GetChild(2).GetComponent<CarAgent>();
}

public void Init_Parkinglot()
{
    // 임시변수
    float z_val = 0;
    float x_val = 0; // 차가 생성되는 시작점 좌표가
    들어가는 변수.
    List<float> stored_x = new List<float>(); // 생성되는 차의 좌표를 저장할 변
    수
    List<float> stored_z = new List<float>();
    int num = 0; // random값을 저장하는 변수

    #region Parking First Line Code
    for (int i = 0; i < 2; i++)
    {
        num = Random.Range(1, 40);
        if (num < 25)
        {
            obs_car = Instantiate(Resources.Load("Car_18C"), new Vector3(0, 0, 0),
Quaternion.Euler(0, 0, 0)) as GameObject;
            obs_car.transform.parent = this.transform.GetChild(1).GetChild(0).GetChild(1);
            obs_car.transform.localPosition = new Vector3(32.3f + x_val, -0.8f, 2.7f);
            obs_car.tag = "Obstacle";
            slot.Add(obs_car);
            stored_x.Add(obs_car.transform.localPosition.x);
            stored_z.Add(obs_car.transform.localPosition.z);
        }
        x_val = x_val - (float)2.01;
    }
    x_val = 0;

    x_val = 0;
    for (int i = 0; i < 18; i++)
    {

```

```

        num = Random.Range(1, 40);
        if (num < 25)
        {
            obs_car = Instantiate(Resources.Load("Car_18C"), new Vector3(0, 0, 0),
Quaternion.Euler(0, 0, 0)) as GameObject;
            obs_car.transform.parent = this.transform.GetChild(1).GetChild(0).GetChild(1);
            obs_car.transform.localPosition = new Vector3(16.1f + x_val, -0.8f, 2.7f);
            obs_car.tag = "Obstacle";
            slot.Add(obs_car);
            stored_x.Add(obs_car.transform.localPosition.x);
            stored_z.Add(obs_car.transform.localPosition.z);
        }
        x_val = x_val - (float)2.01;
    }

    x_val = 0;
    for (int i = 0; i < 2; i++)
    {
        num = Random.Range(1, 40);
        if (num < 25)
        {
            obs_car = Instantiate(Resources.Load("Car_14I"), new Vector3(0, 0, 0),
Quaternion.Euler(0, 0, 0)) as GameObject;
            obs_car.transform.parent = this.transform.GetChild(1).GetChild(0).GetChild(1);
            obs_car.transform.localPosition = new Vector3(-28.35f + x_val, -0.8f, 2.7f);
            obs_car.tag = "Obstacle";
            slot.Add(obs_car);
            stored_x.Add(obs_car.transform.localPosition.x);
            stored_z.Add(obs_car.transform.localPosition.z);
        }
        x_val = x_val - (float)2.01;
    }
    #endregion

    #region Parking Second Line Code

    for (int j = 1; j < 2; j++)
    {
        x_val = 0;

        for (int i = 0; i < 2; i++)

```



```

    {
        num = Random.Range(1, 40);
        if (num < 25)
        {
            obs_car = Instantiate(Resources.Load("Car_18C"), new Vector3(0, 0, 0),
Quaternion.Euler(0, 0, 0)) as GameObject;
            obs_car.transform.parent =
this.transform.GetChild(1).GetChild(0).GetChild(1);
            obs_car.transform.localPosition = new Vector3(30.47f + x_val, -0.8f,
13.7f + z_val);

            obs_car.tag = "Obstacle";
            slot.Add(obs_car);
            stored_x.Add(obs_car.transform.localPosition.x);
            stored_z.Add(obs_car.transform.localPosition.z);
        }
        x_val = x_val + (float)2.01;
    }

    x_val = 0;

    for (int i = 4; i < 12; i++)
    {
        num = Random.Range(1, 40);
        if (num < 25)
        {
            obs_car = Instantiate(Resources.Load("Car_18C"), new Vector3(0, 0, 0),
Quaternion.Euler(0, 0, 0)) as GameObject;
            obs_car.transform.parent =
this.transform.GetChild(1).GetChild(0).GetChild(1);
            obs_car.transform.localPosition = new Vector3(2.18f + x_val, -0.8f, 13.7f
+ z_val);

            obs_car.tag = "Obstacle";
            slot.Add(obs_car);
            stored_x.Add(obs_car.transform.localPosition.x);
            stored_z.Add(obs_car.transform.localPosition.z);
        }
        x_val = x_val + (float)2.01;
    }

    x_val = 0;

```

```

for (int i = 0; i < 9; i++)
{
    num = Random.Range(1, 40);
    if (num < 25)
    {
        obs_car = Instantiate(Resources.Load("Car_14I"), new Vector3(0, 0, 0),
Quaternion.Euler(0, 0, 0)) as GameObject;
        obs_car.transform.parent =
this.transform.GetChild(1).GetChild(0).GetChild(1);
        obs_car.transform.localPosition = new Vector3(-22.38f + x_val, -0.8f,
13.7f + z_val);

        obs_car.tag = "Obstacle";
        slot.Add(obs_car);
        stored_x.Add(obs_car.transform.localPosition.x);
        stored_z.Add(obs_car.transform.localPosition.z);
    }
    x_val = x_val + (float)2.01;
}

x_val = 0;

for (int i = 0; i < 2; i++)
{
    num = Random.Range(1, 40);
    if (num < 25)
    {
        obs_car = Instantiate(Resources.Load("Car_14I"), new Vector3(0, 0, 0),
Quaternion.Euler(0, 0, 0)) as GameObject;
        obs_car.transform.parent =
this.transform.GetChild(1).GetChild(0).GetChild(1);
        obs_car.transform.localPosition = new Vector3(-30.47f + x_val, -0.8f,
13.7f + z_val);

        obs_car.tag = "Obstacle";
        slot.Add(obs_car);
        stored_x.Add(obs_car.transform.localPosition.x);
        stored_z.Add(obs_car.transform.localPosition.z);
    }
    x_val = x_val + (float)2.01;
}

```

```

        z_val -= 4f;
    }

#endregion Parking Seconde Line Code

#region Flag Setting Code
if (flag)
{
    num = Random.Range(0, slot.Count);
    Destroy(slot[num]);
    car_agent.goal = Instantiate(Resources.Load("Target_Area"), new Vector3(0, 0, 0),
Quaternion.Euler(0, 0, 0)) as GameObject;
    car_agent.goal.transform.parent =
this.transform.GetChild(1).GetChild(0).GetChild(1);
    car_agent.goal.transform.localPosition = new Vector3(stored_x[num], -0.5f,
stored_z[num]);
    switch (stored_z[num])
    {
        case 2.7f:
            car_agent.goal.transform.rotation = Quaternion.Euler(0, 0, 0);
            break;
        case 13.7f:
            car_agent.goal.transform.rotation = Quaternion.Euler(0, 180, 0);
            break;

    }
    slot.Add(car_agent.goal);
    set_target(car_agent.goal);
}
else
{
    if (Pos <= 1 | Pos > 50)
    {
        Debug.Log("Input 1~50!");

        if (UnityEditor.EditorApplication.isPlaying)
        {
            UnityEditor.EditorApplication.isPlaying = false;
        }
    }
}

```

```

        else
        {
            Application.Quit();
        }

    }
    else if (Pos <= 2)
    {
        x_val = 32.3f + (-2.01f * (Pos - 1));
        z_val = 2.7f;
    }
    else if (Pos <= 20)
    {
        x_val = 16.1f + (-2.01f * (Pos - 3));
        z_val = 2.7f;
    }
    else if (Pos <= 22)
    {
        x_val = -28.35f + (-2.01f * (Pos - 21));
        z_val = 2.7f;
    }
    else if (Pos <= 24)
    {
        x_val = 32.48f + (-2.01f * (Pos - 23));
        z_val = 13.7f;
    }
    else if (Pos <= 36)
    {
        x_val = 24.29f + (-2.01f * (Pos - 25));
        z_val = 13.7f;
    }
    else if (Pos <= 48)
    {
        x_val = -0.27f + (-2.01f * (Pos - 37));
        z_val = 13.7f;
    }
    else if (Pos <= 50)
    {
        x_val = -28.46f + (-2.01f * (Pos - 49));
        z_val = 13.7f;
    }
}

```

```

        car_agent.goal = Instantiate(Resources.Load("Target_Area"), new Vector3(0, 0, 0),
Quaternion.Euler(0, 0, 0)) as GameObject;
        car_agent.goal.transform.parent =
this.transform.GetChild(1).GetChild(0).GetChild(1);
        car_agent.goal.transform.localPosition = new Vector3(x_val, -0.8f, z_val);

switch (z_val)
{
    case 2.7f:
        car_agent.goal.transform.rotation = Quaternion.Euler(0, 0, 0);
        break;
    case 13.7f:
        car_agent.goal.transform.rotation = Quaternion.Euler(0, 180, 0);
        break;

}

foreach (var k in slot)
{
    if (k.transform.position == car_agent.goal.transform.position)
    {
        Destroy(k);
    }
}
slot.Add(car_agent.goal);
set_target(car_agent.goal);
}
#endregion
}

public void clear_Parkinglot()
{
    foreach (var k in slot)
    {
        Destroy(k);
    }
    slot.Clear();
}

```

```

public void set_target(GameObject target)
{
    child = car_agent.goal.transform.GetChild(4).gameObject; // 자식 object 연결
    child.tag = "goal";

    for (int i = 0; i < 4; i = i + 2)
    {
        child = car_agent.goal.transform.GetChild(i).gameObject;
        child.tag = "goal_line";
    }
}

public float cal_angle()
{
    Vector3 car_angle = new Vector3(0, 0, 0); // 임시변수
    Vector3 goal_angle = new Vector3(0, 0, 0); // 임시변수
    car_angle = car_agent.frontDriver.position - car_agent.backDriver.position;
    car_angle.y = 0f;
    goal_angle = car_agent.goal.transform.GetChild(0).transform.position -
car_agent.goal.transform.GetChild(2).transform.position;
    goal_angle.y = 0f;
    return GetAngle(car_angle, goal_angle);
}

public static float GetAngle(Vector3 point1, Vector3 point2) // 각도 구하기
(원점에서부터 각 점까지의 직선, 그 사이각)
{
    Vector3 k = point1;
    Vector3 l = point2;

    return (Mathf.Atan2(k.x, k.z) * Mathf.Rad2Deg - Mathf.Atan2(l.x, l.z) *
Mathf.Rad2Deg);
}

public float geta_angle() // 각을 0~90범위로 변환
{
    float k = cal_angle();

    if (k >= -90 && k < 0) // -90<k>0

```

```

    {
        return -k;
    }
    else if (k >= 0 && k < 90)    // 0<k<90
    {
        return k;
    }
    else if (k >= 90 && k < 180)    // 90 < k < 180
    {
        return (180 - k);
    }
    else if (k > -180 && k < -90)    // -180 < k < -90
    {
        return (-k + 180);
    }
    else
    {
        return 0;
    }
}
}

```

CameraController.cs

...

CameraViewChanger.cs

...

UIController.cs

...

카메라, UI 관련 코드는 프로젝트 내 첨부

7. 참고자료, URL, 문헌

[1] Unity Learn

<https://learn.unity.com/>

[2] Youtube, AI Learns to Park - Deep Reinforcement Learning

https://www.youtube.com/watch?v=VMp6pq6_QjI

[3] Unity Manual

<https://docs.unity3d.com/Manual/index.html>

[4] Github ML-agents release Document

<https://github.com/Unity-Technologies/ml-agents>

[5] Proximal Policy Optimization Algorithms

<https://arxiv.org/pdf/1707.06347.pdf>

[6] Soft Actor-Critic Algorithms

<https://spinningup.openai.com/en/latest/algorithms/sac.html>

[7] “텐서플로와 유니티 ML-Agents로 배우는 강화학습”, 리디북스, pp.181-237