Human-Feedback 및 행동 복제를 활용한 심층강화학습 알고리즘의 학습 성능 개선 연구
중간보고서

팀명: 강아지도학습

201924504 심영찬
201624420 김동건
202025221 오현식

목차

1. 과제 배경 및 목적
   1.1. 과제 배경
   1.2. 과제 목표
2. 요구사항에 대한 수정사항
   2.1. 기존 요구사항
   2.2. 수정된 요구사항
3. 설계상세화 및 변경내역
   3.1. 개발 환경
   3.2. 전체 구성도
4. 갱신된 과제 추진 계획
5. 구성원별 진척도
6. 보고 시점까지의 과제 수행 내용 및 중간 결과

1. 과제 배경 및 목적

1.1. 과제 배경

최근 자율주행 자동차, 스마트 팩토리, 이족보행 로봇 등 다양한 분야에서 인공지능의 중요성이 점점 더 커지고 있다. [1] 그중에서도 강화학습 알고리즘은 가장 널리 활용되는 대표적인 방법의 하나이다. 강화학습이란, 인공지능 모델이 특정 알고리즘을 통해 주어진 과제를 해결하는 방법을 스스로 학습해 나가는 과정을 의미한다.

지금까지 다양한 고성능 심층강화학습 알고리즘들이 개발되어 활용되고 있지만 여전히 여러 단점이 존재한다. 그중 가장 치명적인 단점은 강화학습 기반의 인공지능 모델이 온전히 제 기능을 할 수 있는 만큼 학습되기까지 매우 많은 시간이 소모된다는 점이다. [2]

강화학습은 지도학습과는 달리 정답을 주지 않은 상태에서 시행착오를 통해 학습이 이루어진다. 초기 단계에서 모델은 다양한 예외 상황을 마주치며 실패를 경험하고, 그 경험을 바탕으로 점차 더 “올바른 선택”을 할 수 있게 된다. 이러한 시행착오적 접근은 환경과의 상호작용을 통해 스스로 학습해 나가는 강화학습의 특징이다.

그런데 딥러닝 기반의 모델들은 기본적으로 방대한 데이터와 반복 학습을 해야 하므로 학습에 오랜 시간이 소요되며, 강화학습 역시 마찬가지로 학습 시간이 길다는 공통적인 한계를 갖는다. 이처럼 심층 강화학습의 두 방식 모두 시간 소모가 크다는 사전 조사를 통해, 학습 시간을 단축할 방법의 필요성을 확인할 수 있었다.

이에 따라, 본 과제에서는 강화학습중인 모델에게도 지도학습처럼 “올바른 선택”을 함께 제시하는 것으로 이 문제를 해결하려고 한다. 학습중인 모델이 특정 상황을 마주쳤을 때 사용자가 실시간으로 피드백을 제공하거나, 사전에 정의된 “올바른 선택”을 제공함으로써 모델이 단축된 시간으로 학습하여 더 향상된 성능을 보이도록 한다.

1.2. 과제 목표

본 과제의 목표는 스스로 학습해 나가는 강화학습 기반 인공지능 모델이 사용자로부터 피드백을 받아 학습 시간을 단축할 수 있도록, 기존보다 효율적인 방식으로 알고리즘을 개선하고 이를 연구하는 것이다. 사용자의 실시간 피드백을 긍정/부정으로 구분하고 이를 통해 인공지능이 다양한 상황에서 보다 빠르게 “올바른 선택”을 학습할 수 있도록 유도함으로써, 전체 학습 효율을 높이는 것을 지향한다.

기존 학습 방법과 비교해 짧은 시간과 적은 자원을 사용하면서 성능은 유지하는 모델을 생성하는 사용자 실시간 피드백, 행동 복제를 적용한 학습 알고리즘을 개발한다.

구현된 새로운 알고리즘을 유니티 환경에 안정적으로 이식하고, ML-Agents를 gym으로 랩핑하여[3] 새로운 시뮬레이션 환경에서도 원활하게 동작하도록 한다.

사용자가 시뮬레이션 환경에서 모델에게 실시간으로 피드백을 쉽게 제공할 수 있도록 한다. 또한, 사용자 친화적인 인터페이스를 통해 필요한 모델 정보를 안정적이고 효과적으로 전달한다.

2. 요구사항에 대한 수정사항

2.1. 기존 요구사항

2.1.1. LLM과 행동복제를 기반으로 학습하는 심층강화학습 알고리즘 개발

(1) DQN을 활용한 새로운 알고리즘 구현

기존의 DQN 알고리즘을 개선하여 강화학습 중인 모델이 실시간 피드백을 제공받는 새로운 알고리즘을 구현한다. 실시간으로 제공되는 피드백은 전처리를 거쳐 모델에게 주어지며, 이는 모델의 보상 체계에 영향을 끼쳐 더 뛰어난 학습을 가능하게 한다.

(2) 사용자의 피드백을 변환할 수 있는 LLM 모델 구현

사용자가 음성/자판으로 제공하는 피드백들에 대해 pretrained-BERT모델을 개선하여 긍정/부정 감성분석을 구분해 낼 수 있는 모델을 구현한다.

2.1.2. 시뮬레이터 개발

(1) 유니티엔진을 이용하여 시뮬레이션 환경 개발

Unity엔진의 오픈 소스 강화학습 패키지인 ML-Agents를 이용하여 학습을 진행할 수 있도록 유니티에서 기본적으로 지원하는 3D 렌더링, 물리 시스템을 사용하여 실제 환경과 유사한 시뮬레이션을 개발한다.

(2) ML-Agents를 통해 시뮬레이션과 학습 알고리즘 연결

ML-Agents를 통해 학습 알고리즘을 유니티 시뮬레이션과 연결하여 학습을 진행한다.

(3) UI를 통한 사용자의 시뮬레이터 환경 조정

UI를 통해 사용자가 파라미터를 조정 및 모델 변경시, 변경사항을 적용하여 시뮬레이션을 자동으로 빌드하고 시뮬레이션 할 수 있게한다.

2.1.3. 웹 개발

(1) 시뮬레이터 화면 출력

구동중인 시뮬레이터 화면을 실시간 스트리밍으로 사용자에게 제공한다.

(2) 환경 선택 및 하이퍼파라미터 설정

실험 환경 및 학습 조건을 사용자 입력을 통해 동적으로 조정할 수 있도록 한다.

(3) 모델 저장 및 불러오기

학습 모델의 진행 상황을 저장하고, 필요한 시점에 불러올 수 있는 기능을 제공한다.

(4) 학습 곡선 출력 및 비교

학습 결과를 시각적으로 출력하고, 다양한 조건에서의 학습 성능을 그래프로 비교할 수 있도록 한다.

2.1.4. 비기능적 요구사항

(1) 직관적인 UI를 통해 사용자가 쉽게 사용할 수 있어야 한다.
(2) 실시간 피드백이 중요하므로, 모델이 구동되는 시뮬레이터는 원활하게 실행되어야 한다.
(3) 모든 학습 데이터를 보여주기 보다는 모델 개발에 주요한 데이터 위주로 사용자에게 제공한다.

2.2. 수정된 요구사항

2.2.1. LLM과 행동복제를 기반으로 학습하는 심층강화학습 알고리즘 개발

(1) 다른 알고리즘을 사용한 실험들과 정량적인 결과 비교가 가능해야 한다.
(2) 시뮬레이터 등 다른 환경으로 이식할 시 기존 실험과 동일한 결과를 도출할 수 있어야 한다.
(3) 웹에서 시각적으로 표현할 수 있도록 서버에 실험의 결과를 전달할 수 있어야 한다.

2.2.2. 시뮬레이터 개발

(1) 다중 에이전트를 지원하여 학습 속도를 가속 할 수 있게 한다.
(2) 시스템 경량화를 위해 매 학습 마다 유니티 에디터로 빌드하는 과정을 없애고, 이미 생성된 빌드를 이용한다.
(3) 과적합을 방지하기 위하여 학습마다 무작위로 선정된 환경을 제공한다.
(4) Gym으로 래핑하는 대신 호환성을 위하여 커스텀 래퍼를 이용하여 Gymnasium으로 래핑한다.

2.2.3. 웹 개발

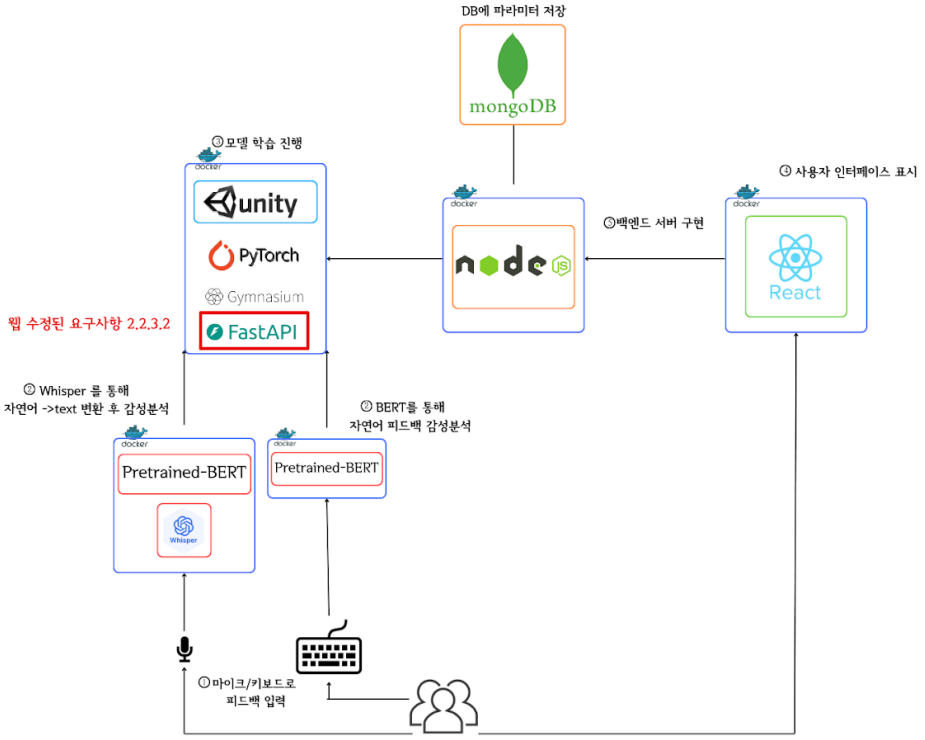
(1) 시뮬레이션 환경이 선택된 알고리즘 유형과 호환되지 않을 경우, 지원되지 않는 알고리즘으로 인한 잘못된 학습을 방지하기 위해 사용자에게 경고 메시지를 표시한다.
(2) FastAPI를 사용하여 클라이언트의 요청을 처리함으로써 강화 학습 환경 제어를 더욱 간편하게 하고, 해당 요청에 따라 Unity 시뮬레이터에 적절한 명령을 전송한다.
(3) 사용자가 학습 진행 상황을 쉽게 파악하고 일정 관리를 할 수 있도록, 강화 학습의 예상 완료 시간을 계산하여 화면에 표시한다.
(4) 서버 자원의 불필요한 소모를 줄이기 위해, 사용자가 웹에 접속할 때 Unity 시뮬레이션 프로세스를 자동으로 시작한다.

3. 설계상세화 및 변경내역

3.1. 개발환경

1. 강화학습 모델: pyTorch
2. LLM 모델: Pretrained-BERT
3. 시뮬레이터: unity(ML Agents), Gymnasium
4. 웹: React, node.js
5. 개발 및 배포 환경 구성 : docker
6. 데이터베이스: Mongo DB

3.2. 전체 구성도



1. 사용자가 마이크 또는 키보드로 피드백을 입력한다.
2. 자연어로 제공된 피드백이 whisper ai와 pretrained-bert LLM모델을 거쳐 긍정/부정 피드백으로 변환된다.
3. 제공된 피드백은 pyTorch를 기반으로 구현된 강화학습 모델에게 입력된다.
4. Unity 빌드에서 구동된 모델은 React 기술을 활용한 웹으로 사용자에게 제공된다.
5. React로 구현된 웹은 node.js와 docker기술을 활용해서 서버로부터 제공된다.

4. 갱신된 과제 추진 계획

개발구분	세부항목	5	6	7	8	9
기획	주제 선정 및 고도화					
	사전 조사					
인공지능 모델 개발	human-feedback 구현					
	pretrained-bert 감성분석 fine-tuining					
시뮬레이터 설계	시뮬레이터 환경 구현					
	사용자 상호작용 시스템 구현					
	구현된 알고리즘 gym-wrapping					
애플리케이션 개발	웹-시뮬레이션 시스템 연동					
	프론트엔드 개발					
	백엔드 개발					
배포 및 수정	보완사항 수정 및 배포					

5. 구성원별 진척도

5.1. 구성원 별 역할

이름	담당
심영찬	pyTorch활용 DQN개선 알고리즘 개발 pretrained-bert기반 피드백 입력 LLM 모델 개발
김동건	unity 기반 시뮬레이터 개발, Unity - server, 학습 알고리즘간 연결
오현식	React 기반 FE 개발, node.js, docker 기반 BE 개발

공통	관련 논문 분석, 보고서 작성

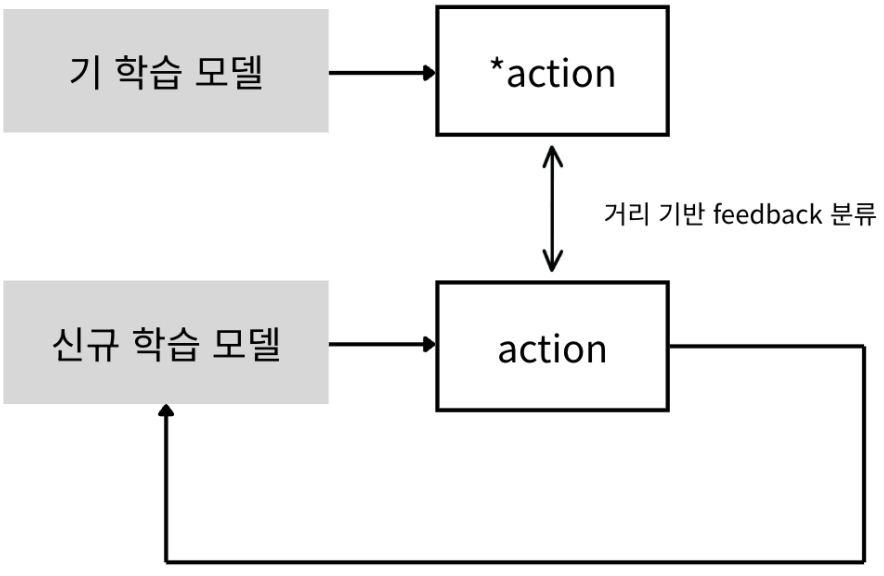
5.2. 구성원 별 진척도

이름	진척도
심영찬	human-feedback을 대체할 인공지능의 학습구조를 구현 pyTorch활용 DQN개선 알고리즘 개발
김동건	DQN으로 학습 가능한 에이전트 구현 Gymnasium래핑을 위한 커스텀 래퍼 클래스 구현 컨테이너화된 리눅스 환경에서 환경 시뮬레이션, stable-baselines3에서 제공하는 알고리즘을 통해 학습 진행
오현식	유니티 시뮬레이터 화면 출력 기능 구현 학습 제어 (시작/중지/재개/종료) 기능 구현 실험 설정 (실험 환경 선택, 알고리즘 선택) 기능 일부 구현

6. 보고시점까지의 과제 수행 내용 및 중간 결과

6.1. LLM과 행동복제를 기반으로 학습하는 심층강화학습 알고리즘 개발

6.1.1. human-feedback을 대체할 인공지능의 학습구조를 구현 완료



feedback종류에 따라 memory buffer에 저장하는 개수 조절

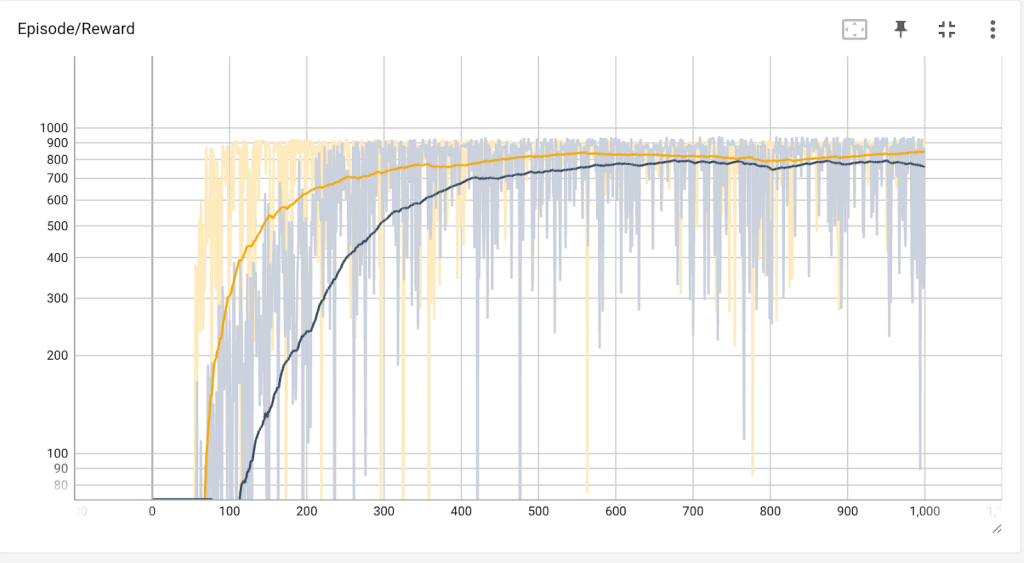
- 이미 학습된 모델과 함께 모델을 새로 학습시킴
- 실시간으로 각 state마다 두 모델의 action 값을 비교

- 정규화된 점수를 기반으로 action에 대해 피드백 flag를 부여
- flag에 따라 메모리 버퍼에 들어가는 샘플의 개수를 조절

6.1.2. pyTorch활용 DQN개선 알고리즘 개발

- memory buffer에 삽입되는 트랜지션의 개수를 조절하여 모델의 학습 성능을 향상시킴
- 초반 학습(50 episode)은 pure DQN알고리즘을 사용하여 학습
- 이후 부터는 human-feedback을 대체한 학습 구조를 바탕으로 각 모델의 행동 사이 거리 기반의 피드백을 활용하여 학습
- 학습의 진행 단계에 따라 memory buffer에 삽입되는 sample의 개수를 동적으로 결정

6.2. 실험 결과 비교 분석



- 노란색: DQN 개선 알고리즘, 검은색: 기존 dqn 알고리즘
- 동일 조건 하에서 실험
- 50에피소드까지는 순수 DQN알고리즘을 이용하고
- 이후 1000에피소드까지는 개선 알고리즘을 이용해서 학습
- 아래 표는 원시 데이터가 아닌 EMA스무딩0.99를 적용한 결과임

점수 기준	순수 DQN 필요 episode	개선 알고리즘 필요 episode	성능 향상 (×배)	성능 향상 (%)	step 감소율(%)
600점	352	**188**	약 1.87배	46.6%	46.6%감소
700점	410	**250**	약 1.64배	39.0%	39.0%감소
800점	없음	**455**	-	-	-

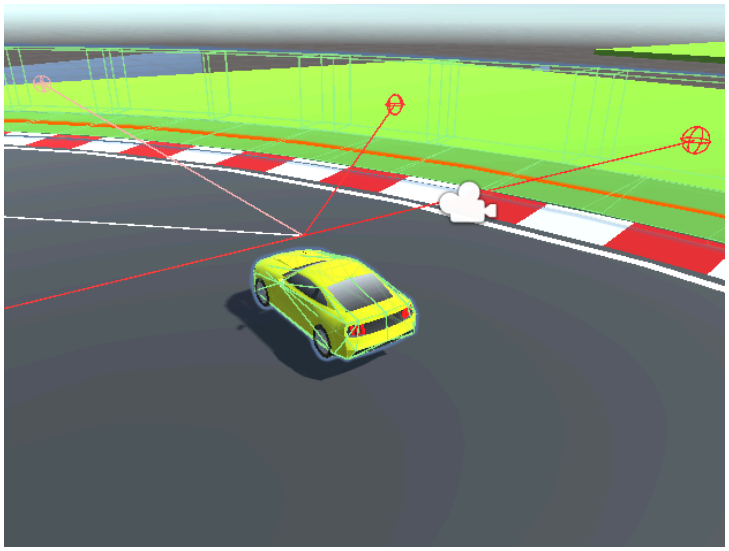
6.3. 개선 알고리즘 결과 분석

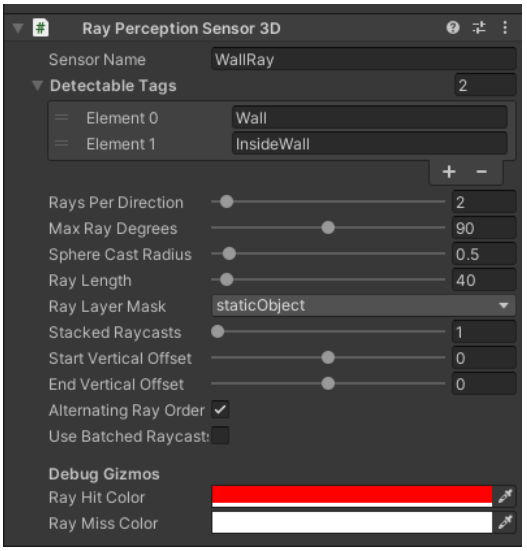
- 매 step마다 action간 거리 계산 후 feedback 분류 (step과 feedback수 동일)

지표 이름	의미	학습 진행 중 관찰 내용
Feedback / Average_Distance	학습이 진행될 동안 두 모델의 action 간 평균 거리	시간이 지날수록 두 모델의 action 거리가 **점차 가까워짐**
Feedback / Negative_Count	negative로 분류된 샘플 수	신규 모델의 부정적 행동 **횟수가 점차 감소**
Feedback / Negative_Ratio	negative로 분류된 샘플 비율	부정적 행동 **비율이 점차 감소**
Feedback / Netural_Count	neutral로 분류된 샘플 수	중립적 행동 **비율이 일정하게 유지**
Feedback / Positive_Count	positive로 분류된 샘플 수	긍정적 행동 **횟수가 점차 증가**
Feedback / Positive_Ratio	positive로 분류된 샘플 비율	긍정적 행동 **비율이 점차 증가**

6.4. 시뮬레이터 개발

6.4.1. DQN학습 및 시뮬레이션 수행을 위한 에이전트 구현






```
public override void CollectObservations(VectorSensor sensor)
{
    carController.SetSpeed();
    sensor.AddObservation(carController.CarSpeed);
    sensor.AddObservation(carController.CurrentSteering);
}

public override void OnActionReceived(ActionBuffers actionBuffers)
{
    //Debug.Log("Action: " + actionBuffers.ContinuousActions[0]);
    if(isCollidingWithWall)
    {
        //SetReward(-0.5f);
        EndEpisode();
    }
    if (isPassChecker)
    {
        isPassChecker = false;
        AddReward( increment 0.5f);
        curentReward += 0.5f;
        carController.SetRewardUI(curentReward);

    }
```

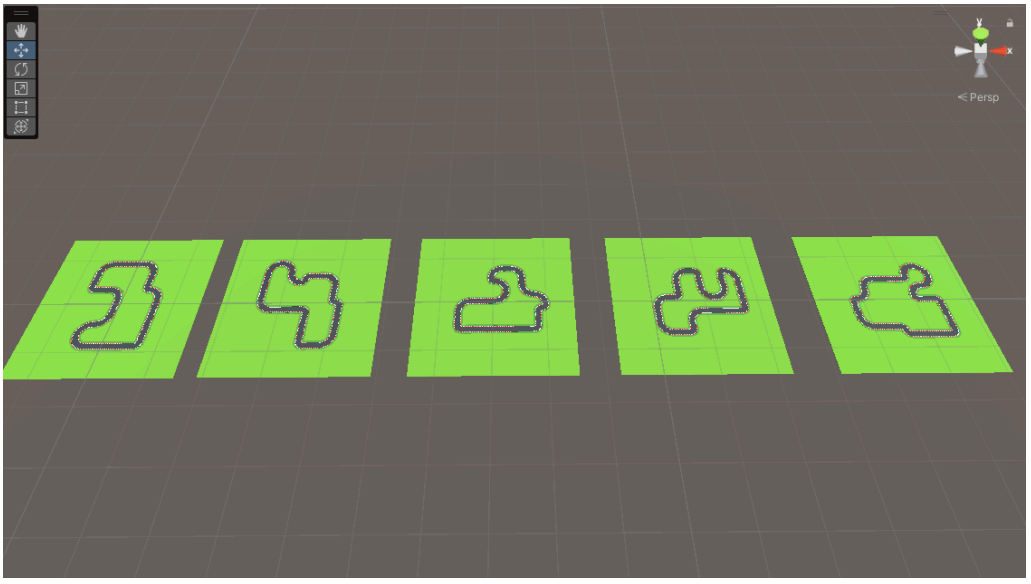
```
    if (isWrongPassChecker)
    {
        isWrongPassChecker = false;
        AddReward( increment -0.1f);
        curentReward += -0.1f;
        carController.SetRewardUI(curentReward);
    }
    var actionIndex = actionBuffers.DiscreteActions[0];
    //Debug.Log($"Action: {actionIndex}");
    carController.ControlCar(actionIndex);


}
```

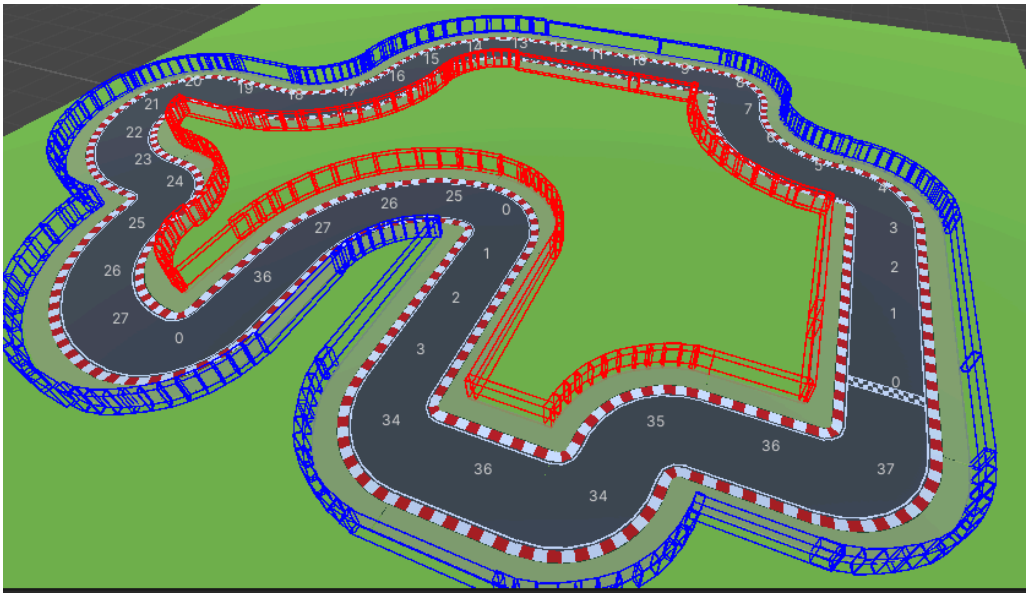
```
private Dictionary<int, (float steering, float gas, float brake)> actionMap =
    new Dictionary<int, (float, float, float)>
    {
        {0, (-1f, 1f, 1f)}, {1, (0f, 1f, 1f)}, {2, (1f, 1f, 1f)},
        {3, (-1f, 1f, 0f)},   {4, (0f, 1f, 0f)},   {5, (1f, 1f, 0f)},
        {6, (-1f, 0f, 1f)}, {7, (0f, 0f, 1f)}, {8, (1f, 0f, 1f)},
        {9, (-1f, 0f, 0f)},   {10, (0f, 0f, 0f)},  {11, (1f, 0f, 0f)}
    };
```

- ML-Agents의 agent클래스를 이용하여 학습에 사용될 차량 에이전트를 구현하였다. 차량의 물리 시스템은 Unity에서 제공하는 wheel Collider를 이용하였다.

- Ray를 이용하여 에이전트가 주행환경을 관찰하게 하였다. Ray는 트랙 모서리에 구현된 보이지 않는 벽의 collider를 탐지하여 현재 트랙의 형태를 관찰한다. 벽 collider는 wall, insideWall 태그로 구분되어 차량이 진행 방향을 잃지 않도록 하였다.
- DQN기반의 알고리즘에 대응하기 위해 이산 행동을 정의하여 차량의 스티어링, 엑셀, 브레이크를 조정한다.

6.4.2. 다양한 형태의 트랙 제작





- 과적합을 방지하기 위해서 학습시 다양한 형태의 트랙을 무작위로 제공한다. 에피소드 시작시 에이전트는 무작위로 트랙에 배정되고, 배정된 트랙의 설정 정보를 받아 학습에 이용한다. 각 트랙의 난이도, 길이가 다르므로 트랙별로 보상 값을 정규화하여 학습의 연결성을 유지한다.
- gizmo gui를 이용해 트랙 프리팹 배치 -> 트랙 순서 시각화 과정을 자동으로 이루어 지게끔 구현하여 주행 환경을 제작할때의 편의성을 높혔다.
- gizmo gui를 이용해 보이지 않는 벽을 표시하여 주행 환경 제작시 편의성을 높혔다.

6.4.3. 학습 환경 컨테이너 도커 파일 작성

```
FROM python:3.10.12-slim

# FastApi 설치
RUN pip install fastapi uvicorn[standard]

# 필수 패키지 설치
RUN apt-get update && \
    apt-get install -y --no-install-recommends \
        libgl1-mesa-glx \
        wget git libglib2.0-0 \
        xvfb x11vnc fluxbox\
    && rm -rf /var/lib/apt/lists/*

# 유니티 빌드 패키지 설치
RUN apt-get update && apt-get install -y \
    libx11-6 libx11-dev libsm6 libxrender1 libxext6 libxi6 libxcursor1 libxrandr2 libxinerama1 libxss1 libxtst6 libnss3 \
    libasound2 libatk1.0-0 libgtk-3-0 libxcomposite1 libxdamage1 libxfixes3 libxft2

# Python 패키지 설치
RUN pip install --upgrade pip
RUN pip install \
    #mlagents==1.1.0 \
    mlagents-envs==1.1.0 \
    #stable-baselines3==1.8.0 \
    gymnasium==1.1.1
    #shimmy>=2.0

COPY Unity/app /app
# 유니티 빌드 파일 마운트용 디렉토리
RUN mkdir /build
RUN mkdir /workspace




COPY Unity/entrypoint.sh /entrypoint.sh
RUN chmod +x /entrypoint.sh

EXPOSE 8080 5900 8000 3001 80

ENTRYPOINT ["/entrypoint.sh"]
```

- 해당 컨테이너는 학습에 필요한 유니티 빌드, ML-Agents, Gymnasium, fast api등을 실행시켜야 하므로 필요한 패키지들을 구성하였다. 학습 환경 송출을 위한 가상 모니터 패키지도 추가하였다.

6.4.4. 환경 래핑을 위한 커스텀 래퍼 클래스 작성

```python
class MLAgentsGymWrapper(gym.Env):
    def __init__(self, unity_env_path, worker_id=1):
        #self.sidechannel = CustomCommandChannel()
        self.unity_env = UnityEnvironment(
            file_name=unity_env_path,
            no_graphics=False,
            #side_channels=[self.sidechannel],
            worker_id=worker_id
        )
        self.unity_env.reset()
        self.behavior_names = list(self.unity_env.behavior_specs)
        print(self.behavior_names)
        self.behavior_name = self.behavior_names[0]
        print(self.behavior_name)
        behavior_spec = self.unity_env.behavior_specs[self.behavior_name]

        # 관찰 공간 설정 (기존 로직 유지)
        obs_shapes = []
        total_obs_size = 0
        for obs_spec in behavior_spec.observation_specs:
            obs_shapes.append(obs_spec.shape)
            total_obs_size += np.prod(obs_spec.shape)

            # gymnasium.spaces 사용
        self.observation_space = spaces.Box(
            low=-np.inf, high=np.inf,
            shape=(total_obs_size,), dtype=np.float32
        )

         # 행동 공간 설정
        if behavior_spec.action_spec.continuous_size > 0:
            self.action_space = spaces.Box(
                low=-1.0, high=1.0,
                shape=(behavior_spec.action_spec.continuous_size,),
                dtype=np.float32
            )
        else:
            # 이산 행동 공간 처리
            if behavior_spec.action_spec.discrete_size > 0:
                self.action_space = spaces.Discrete(
```

-

- 기존의 ML-Agents의 래퍼는 Gym래핑만 지원한다. Gymnasium에 래핑하기 위하여 관찰값, 행동값들을 알맞게 변형 시키는 커스텀 래퍼 클래스를 작성하였다. 해당 래퍼와 빌드를 이용하여 DQN알고리즘 학습이 가능함을 확인 하였다.

6.5. 웹 개발

6.5.1. 유니티 시뮬레이터 화면 출력 기능 구현

- Unity Render Streaming 패키지를 활용하여 Unity 시뮬레이터의 실시간 화면을 웹 브라우저에 출력할 수 있도록 구현하였다. React 프론트엔드에 맞게 WebRTC 시그널링 및 스트리밍 관련 코드를 수정하여 연동을 완료하였다.

6.5.2. 학습 제어 (시작/중지/재개/종료) 기능 구현

- 프론트엔드
  - 사용자는 시뮬레이터 화면 상단에 배치된 시작, 일시정지, 계속하기, 종료 버튼을 통해 학습을 제어할 수 있다. 각 버튼 클릭 시 FastAPI 서버로 해당 제어 명령을 전송한다.
- 백엔드
  - FastAPI를 이용하여 학습 제어 API를 구성하였다. 각 요청은 추후 강화학습 제어 로직 및 Unity 시뮬레이터와 연동되어 실제 동작하도록 설계할 계획이며, 현재는 요청 수신 및 로깅만 처리되고 있다.

API 엔드포인트	**메서드**	**설명**	**구현 상태**
/unity/start	POST	학습 시작 및 시뮬레이션 화면 출력	요청 수신만 구현
/unity/pause		시뮬레이션 일시 정지	
/unity/resume		시뮬레이션 재개	
/unity/stop		시뮬레이션 종료	

6.5.3. 실험 설정 (실험 환경 선택, 알고리즘 선택) 기능 일부 구현

- 실험 환경과 알고리즘을 선택한 뒤, 해당 정보를 JSON 형식으로 FastAPI 서버에 POST 요청하는 기능을 구현하였다. 현재는 정보 전송까지만 가능하며, 추후 환경과 알고리즘의 호환성을 검사하고, 알고리즘 선택 시 하이퍼파라미터 설정 창이 동적으로 생성되도록 구현할 예정이다.

7. 참고 문헌

[1] 송주상,「알파고 승리 이끈 강화학습, 로봇·자율주행서 재조명」,『IT조선』, 2021년 2월 28일, https://it.chosun.com/news/articleView.html?idxno=2021022602881

[2] Yuxi Li,「Deep Reinforcement Learning: An Overview」, https://arxiv.org/abs/1701.07274

[3] alex-mccarthy-unity,「Unity ML-Agents Gym Wrapper」, ml-agents/docs/Python-Gym-API.md at develop · Unity-Technolo1\9 . gies/ml-agents