Unity ML-Agent Toolkit을 이용한 인공지능 개발

김홍준

1. 프로젝트 목표

   Unity에서 지원하는 ML-Agent Toolkit을 이용하여 Unity로 구현된 간단한 게임을 플레이하는 인공지능을 만드는 것을 목표로 한다.

2. 프로젝트 진행 과정

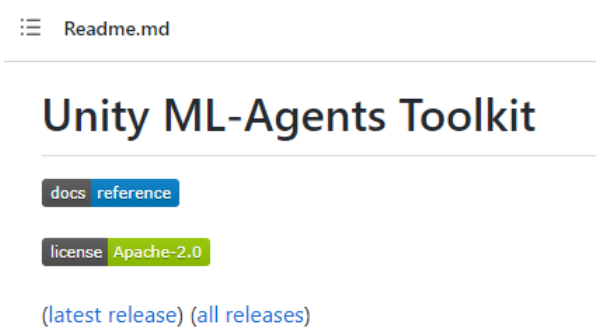
   먼저 mlagents 사용을 위해 Anaconda로 개인 환경 구축과 mlagents 사용법을 Unity에서 제공하는 mlagents 샘플 프로젝트를 통해 익힌 후 Unity로 만든 간단한 게임 프로젝트를 가져와 프로젝트를 Agent 학습에 좋게 수정하고 mlagents를 사용하여 모델을 학습하여 실제로 게임에 적용을 할 것이다.

3. 프로젝트 과정 및 결과

A. 참고 사이트 및 프로젝트 사전 준비

프로젝트의 진행에 앞서 ML-Agent Toolkit의 사용법을 익히고자 Unity에서 공식으로 제공하는 오픈소스[①]를 활용하였다.

2022년 12월 기준 해당 오픈소스에서 제일 최근버전인 Release 20을 채택하여 해당 버전의 파일을 받아 프로젝트를 진행하였다.



Releases & Documentation

Our latest, stable release is `Release 20`. Click here to get started with the latest release of ML-Agents.

Version	Release Date	Source	Documentation	Download	Python Package	Unity Package
Release 20	November 21, 2022	source	docs	download	0.30.0	2.3.0
main (unstable)	--	source	docs	download	--	--
Verified Package 1.0.8	May 26, 2021	source	docs	download	0.16.1	1.0.8

① https://github.com/Unity-Technologies/ml-agents 이 사이트를 참고하였음

그리고 인공지능을 사용할 게임은 해외 샘플 게임 사이트[2]에서 Flappy-Plane-Game이라는 게임을 가져와서 사용하였다. 참고로 해당 게임은 점프 하나로 나무 사이를 지나가며 점수를 얻는 방식에 게임이다.

그 후에는 개인 환경 구성을 진행하였으며 오픈소스 환경 구성 설명[3]에서는 아래처럼 설치를 진행하라고 한다.

Consequently, to install and use the ML-Agents Toolkit you will need to:

- Install Unity (2021.3 or later)
- Install Python (3.8.13 or higher)
- Clone this repository (Optional)
  - **Note:** If you do not clone the repository, then you will not be able to access the example environments and training configurations or the com.unity.ml-agents.extensions package. Additionally, the Getting Started Guide assumes that you have cloned the repository.
- Install the com.unity.ml-agents Unity package
- Install the com.unity.ml-agents.extensions Unity package (Optional)
- Install the mlagents Python package

추가로 윈도우 한정으로 Pytorch 설치를 명시하고 있다.

(Windows) Installing PyTorch

On Windows, you'll have to install the PyTorch package separately prior to installing ML-Agents. Activate your virtual environment and run from the command line:

pip3 install torch~=1.7.1 -f https://download.pytorch.org/whl/torch_stable.html

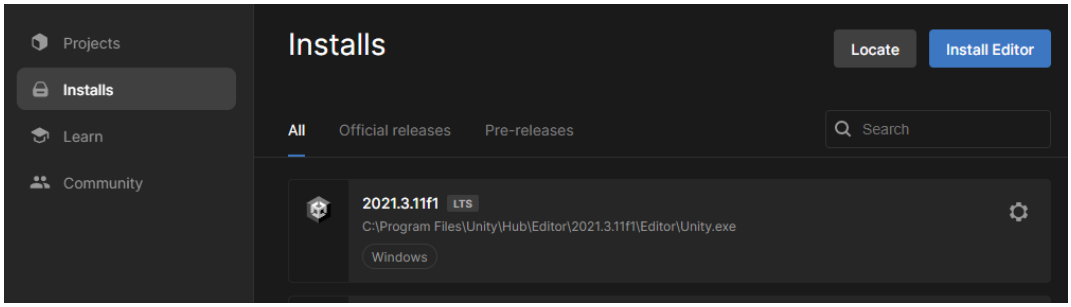
---

[2] https://code-projects.org/flappy-plane-game-in-unity-engine-with-source-code/ 해당 사이트에서 파일을 받음

[3] https://github.com/Unity-Technologies/ml-agents/blob/develop/docs/Installation.md 해당 사이트에서 참고

B. 프로젝트 환경 구성

프로젝트 실습 환경은 사전에 명시된 “Install Unity (2021.3 or later)” 버전인 Unity 2021.3.11f1 버전의 에디터 및 개인 환경 터미널 구축이 가능한 Anaconda를 사용하였다.



Anaconda에 설치를 진행할 패키지는 명시되어 있던 것과 같이

“Install Python (3.8.13 or higher)
(Windows) Installing PyTorch
Install the mlagents Python package” 3가지 이다.

따라서 2022년 12월 기준 Anaconda가 지원하는 python 3.8 버전 중 최신 버전인 3.8.15로 설정하고 UnityMLAgent라는 이름으로 환경을 만들었다.

Anaconda Prompt (anaconda3) - conda deactivate - conda create -n UnityMLAgent python=3.8.15

```
(base) C:₩Users₩was56>conda create -n UnityMLAgent python=3.8.15
```

그리고 UnityMLAgent 환경에서 Pytorch 1.7.1버전 및 mlagents 0.30.0버전도 설치하였다.

Anaconda Prompt (anaconda3) - conda deactivate

```
(base) C:₩Users₩was56>conda activate UnityMLAgent

(UnityMLAgent) C:₩Users₩was56>pip3 install torch~=1.7.1 -f https://download.pytorch.org/whl/torch_stable.html
Looking in links: https://download.pytorch.org/whl/torch_stable.html
Collecting torch~=1.7.1
  Downloading https://download.pytorch.org/whl/cu110/torch-1.7.1%2Bcu110-cp38-cp38-win_amd64.whl (2050.2 MB)
     ---------------------------------------- 2.1/2.1 GB 382.1 kB/s eta 0:00:00
Collecting typing-extensions
  Using cached typing_extensions-4.4.0-py3-none-any.whl (26 kB)
Collecting numpy
  Downloading numpy-1.23.5-cp38-cp38-win_amd64.whl (14.7 MB)
     ---------------------------------------- 14.7/14.7 MB 7.7 MB/s eta 0:00:00
Installing collected packages: typing-extensions, numpy, torch
Successfully installed numpy-1.23.5 torch-1.7.1+cu110 typing-extensions-4.4.0

(UnityMLAgent) C:₩Users₩was56>
```

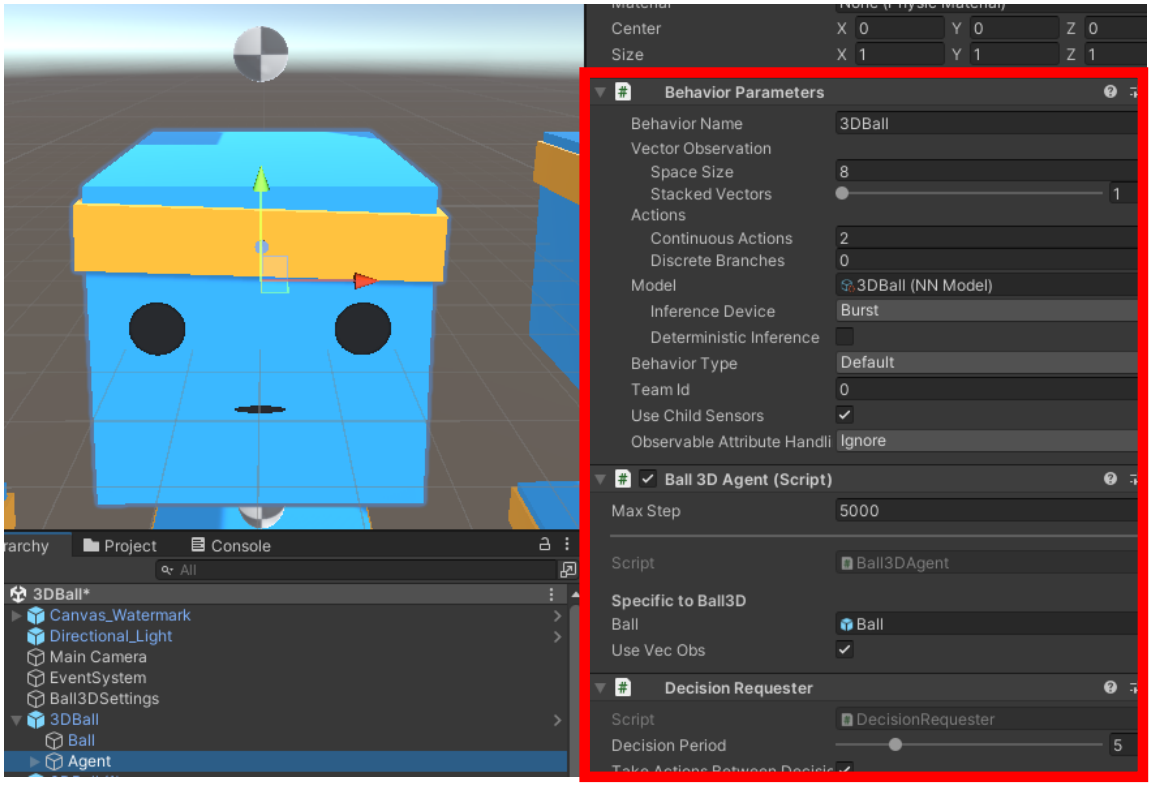
Anaconda Prompt (anaconda3) - conda deactivate

```
(UnityMLAgent) C:₩Users₩was56>python -m pip install mlagents==0.30.0
```

C. 예제 프로젝트 분석 (3DBall Project를 중점으로)

참고로 프로젝트 분석은 Unity에서 제공하는 영상[④]과 같은 주제로 프로젝트를 만들어보는 블로그[⑤]를 참고하였다.

3D Ball Project는 파란색 Agent 위에 있는 공이 떨어지지 않도록 자신을 회전하는 인공지능 프로젝트이다.



해당 프로젝트에서 Agent에 적용된 스크립트가 중요한데 깊게 보아야 할 부분은 Ball3DAgent Script이고 Behavior Parameter는 Script에서 클래스를 MonoBehavior가 아닌 Agent 클래스로 상속받으면 자동으로 생기는 Hyper 파라미터 설정 창이다.

④ https://www.youtube.com/watch?v=GWd4g1qCDxo&t=557s

⑤ https://unity3dstudy.com/2020/08/22/MummyBasic-for-ML-Agents-02/

Behavior Parameter에는 Behavior Name, Vector Observation, Actions, Model이 주로 사용할 설정이며 Behavior Name은 mlagents를 이용할 때 쓸 식별자이며 Model은 mlagents를 사용하면 결과로 반환되는 모델을 넣으면 적용되는 파라미터이다.

여기서 중요한 건 Vector Observation, Actions이며 Vector Observation은 인공지능의 판단을 근거로 하는 관측치를 저장할 공간을 설정할 수 있는 파라미터이고 Action은 결과로 나올 값들을 설정할 수 있는 파라미터이다.

또한 Vector Observation에 Space size는 사용할 관측치의 개수를 의미하며 일반적으로 Script에서 추가한 관측치의 개수만큼 기입한다. Stacked Vector는 과거 시점의 데이터를 파라미터의 값만큼 남겨두어 다음 판단에 사용할 관측치 공간의 개수를 나타내는 파라미터이다.

Actions에는 Continuous Action과 Discrete Branches이 있으며 Continuous Action은 연속적인 값을 결과로 필요로 하는 개수이고 Discrete Branches는 이산적인 값(0, 1, 2 등)을 결과로 필요로 하는 개수를 명시한다. 또한 Discrete Branches의 개수만큼 Branch [개수] size 항목이 나오는데 이 항목은 해당 Branch에서 사용할 이산적인 값의 개수를 명시한다.

해당 설명은 사전에 설명한 Unity에서 제공하는 영상[6]을 참고하였다.

Ball3DAgent Script는 Agent 클래스를 상속받기에 스크립트에 override할 4가지 핵심 메소드들이 있다고 한다.[7] 그 메소드들은 아래와 같다.

1. public override void Initialize()
   초기화 작업을 위해 한번 호출되는 메소드
2. public override void CollectObservations(VectorSensor sensor)
   관측된 환경 정보를 정책 결정을 위한 브레인에 전달하는 메소드
3. public override void OnActionReceived(ActionBuffers actionBuffers)
   브레인으로부터 전달받은 행동을 실행하는 메소드
4. public override void OnEpisodeBegin()
   에피소드가 시작할 때마다 호출되는 메소드

---

[6] https://www.youtube.com/watch?v=GWd4g1qCDxo&t=557s

[7] https://unity3dstudy.com/2020/08/22/MummyBasic-for-ML-Agents-02/

Ball3DAgent Script에서 먼저 객체가 개인적으로 사용한 메소드들은

```
Frequently called  2 usages
public void SetResetParameters() { SetBall(); }

Frequently called  1 usage
public void SetBall()
{
    //Set the attributes of the ball by fetching the information from the academy
    m_BallRb.mass = m_ResetParams.GetWithDefault( key: "mass",  defaultValue: 1.0f);
    var scale :float  = m_ResetParams.GetWithDefault( key: "scale",  defaultValue: 1.0f);
    ball.transform.localScale = new Vector3( x: scale,  y: scale,  z: scale);
}
```

다시 사용할 객체들을 초기화 시켜주는 메소드와 Ball의 질량과 크기 및 m_ResetParams로부터 받은 값으로 설정하는 메소드이다.

Initialize() 메소드는

```
0+1 usages
public override void Initialize()
{
    m_BallRb = ball.GetComponent<Rigidbody>();
    m_ResetParams = Academy.Instance.EnvironmentParameters;
    SetResetParameters();
}
```

agent위에 있는 공의 RigidBody와 에피소드를 다시 진행할 때 사용할 환경 변수를 캐시에 저장을 하고 SetResetParameters()에 의해 공과 agent의 값이 초기화 된다.

OnEpisodeBegin() 메소드는

```
Frequently called  0+3 usages
public override void OnEpisodeBegin()
{
    gameObject.transform.rotation = new Quaternion( x: 0f,  y: 0f,  z: 0f,  w: 0f);
    gameObject.transform.Rotate( axis: new Vector3( x: 1,  y: 0,  z: 0),  angle: Random.Range(-10f, 10f));
    gameObject.transform.Rotate( axis: new Vector3( x: 0,  y: 0,  z: 1),  angle: Random.Range(-10f, 10f));
    m_BallRb.velocity = new Vector3( x: 0f,  y: 0f,  z: 0f);
    ball.transform.position = new Vector3( x: Random.Range(-1.5f, 1.5f),  y: 4f,  z: Random.Range(-1.5f, 1.5f))
        + gameObject.transform.position;
    //Reset the parameters when the Agent is reset.
    SetResetParameters();
}
```

에피소드가 시작할 때 Agent의 초기 회전 각도와 공의 위치를 랜덤으로 준다.

CollectObservations(VectorSensor sensor) 메소드는

```
Frequently called  0+2 usages
public override void CollectObservations(VectorSensor sensor)
{
    if (useVecObs)
    {
        sensor.AddObservation(gameObject.transform.rotation.z);
        sensor.AddObservation(gameObject.transform.rotation.x);
        sensor.AddObservation(ball.transform.position - gameObject.transform.position);
        sensor.AddObservation(m_BallRb.velocity);
    }
}
```

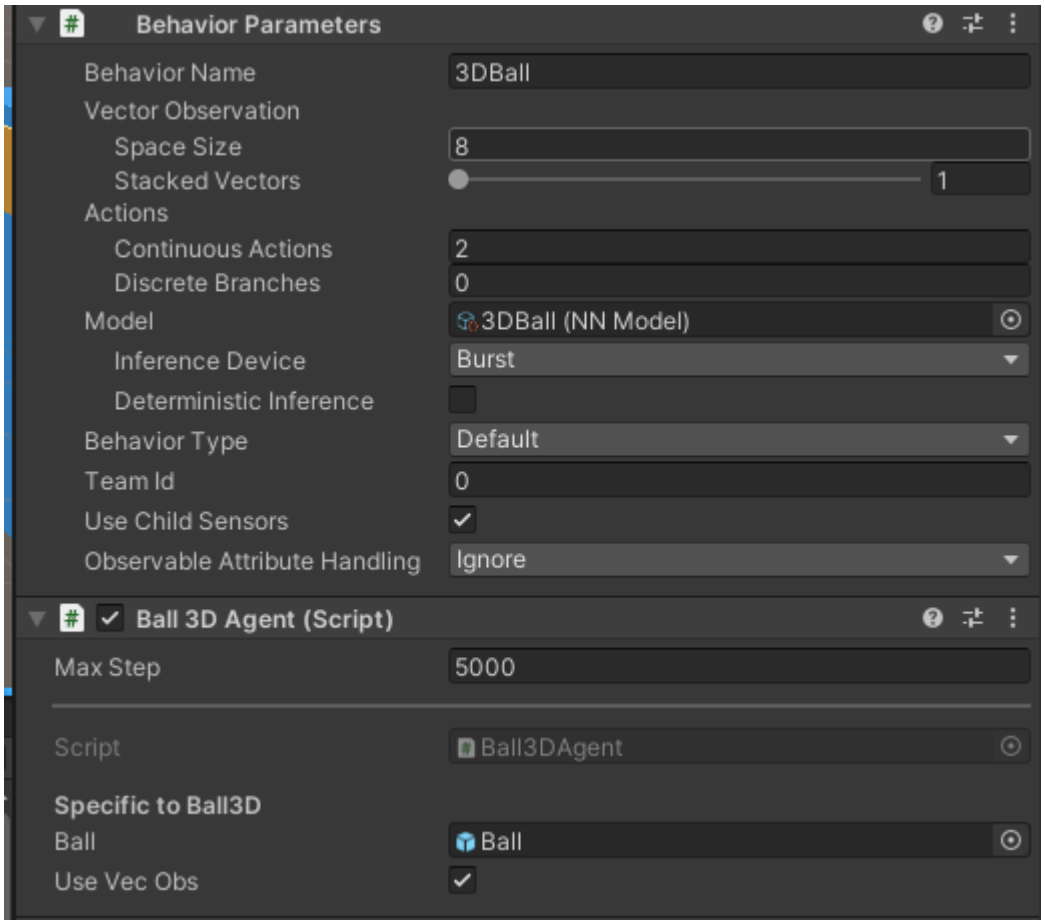
Agent의 rotation x, z, 상자가 보는 공의 상대 위치 그리고 공의 속도를 관측치로 둔다.

OnActionReceived(ActionBuffers actionBuffers) 메소드는

```
Frequently called  0+2 usages
public override void OnActionReceived(ActionBuffers actionBuffers)
{
    var actionZ = 2f * Mathf.Clamp(actionBuffers.ContinuousActions[0], -1f, 1f);
    var actionX = 2f * Mathf.Clamp(actionBuffers.ContinuousActions[1], -1f, 1f);

    if ((gameObject.transform.rotation.z < 0.25f && actionZ > 0f) ||
        (gameObject.transform.rotation.z > -0.25f && actionZ < 0f))
    {
        gameObject.transform.Rotate(new Vector3(0, 0, 1), actionZ);
    }

    if ((gameObject.transform.rotation.x < 0.25f && actionX > 0f) ||
        (gameObject.transform.rotation.x > -0.25f && actionX < 0f))
    {
        gameObject.transform.Rotate(new Vector3(1, 0, 0), actionX);
    }
    if ((ball.transform.position.y - gameObject.transform.position.y) < -2f ||
        Mathf.Abs(ball.transform.position.x - gameObject.transform.position.x) > 3f ||
        Mathf.Abs(ball.transform.position.z - gameObject.transform.position.z) > 3f)
    {
        SetReward(-1f);
        EndEpisode();
    }
    else
    {
        SetReward(0.1f);
    }
}
```

Agent가 축 X, Z에 대해 회전할 값을 연속적인 값으로 받아 회전하고 공이 Agent에게서 떨어지지 않으면 (Reward) 보상치를 0.1씩 추가하며 떨어지면 보상치를 -1만큼 받고 에피소드를 끝낸다.

위 스크립트에 따라 Behavior Parameter 값은 아래와 같이 설정된다.



1. Vector Observation은 CollectObservations(VectorSensor sensor) 메소드에서 Agent의 Rotation X, Z값 (2개) Agent에 대한 Ball의 상대 위치(x, y, z 3개), Ball의 속도(x, y, z 3개) 총 8개이므로 8로 지정해준다.
2. Actions는 OnActionReceived(ActionBuffers actionBuffers) 메소드에서 사용한 actionBuffers.ContinuousActions의 개수에 따라 2로 지정해준다.

해당 설정이 끝나면 모델 학습을 진행할 수 있다.

모델 학습은 개인 환경으로 구축한 Anaconda의 UnityMLAgent 환경으로 진행하며 전에 명시했던 블로그의 다른 포스트[8]를 참고하였다.

먼저 예제 프로젝트가 있는 곳으로 터미널을 이동한 후

```
Anaconda Prompt (anaconda3)
(UnityMLAgent) C:₩Users₩was56>cd C:₩Users₩was56₩Desktop₩ml-agents-latest_release
```

"mlagents-learn config/ppo/3DBall.yaml –run-id=3DBall"(블로그 발췌[9]) 명령어로 학습 준비를 한다.

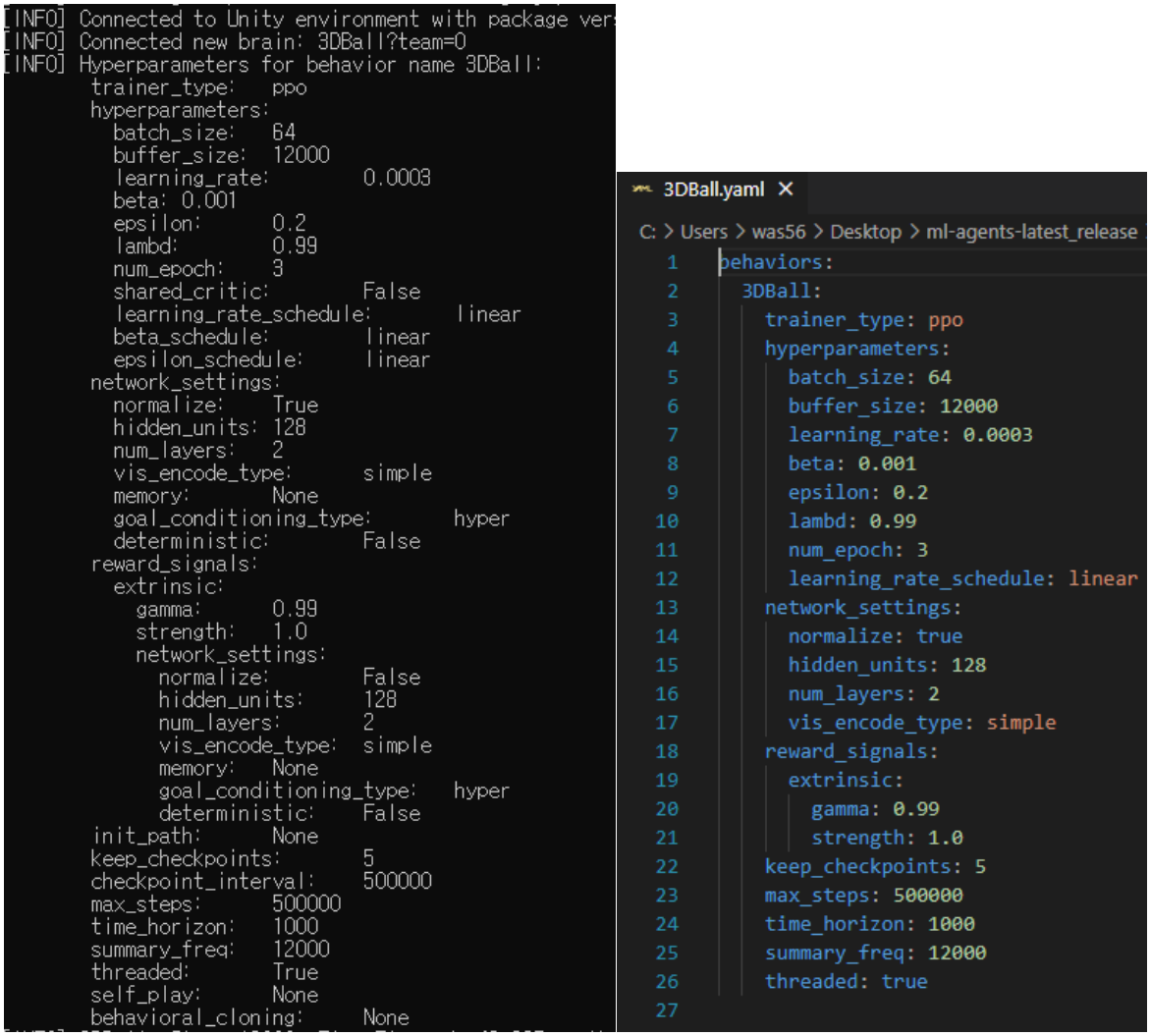
```
e>mlagents-learn config/ppo/3DBall.yaml --run-id=3DBall
```

그러면 아래처럼 5004번 포트에 대기를 하며 유니티 에디터와 자동으로 연결되고 유니티 에디터에서 3DBall 씬을 실행하면 작동한다.

```
(UnityMLAgent) C:₩Users₩was56₩Desktop₩ml-agents-latest_release>mlagents-learn config/ppo/3DBall.
C:₩Users₩was56₩anaconda3₩envs₩UnityMLAgent₩lib₩site-packages₩torch₩cuda₩__init__.py:52: UserWarn
und no NVIDIA driver on your system. Please check that you have an NVIDIA GPU and installed a dr
com/Download/index.aspx (Triggered internally at  ..₩c10₩cuda₩CUDAFunctions.cpp:100.)
  return torch._C._cuda_getDeviceCount() > 0

 Version information:
  ml-agents: 0.30.0,
  ml-agents-envs: 0.30.0,
  Communicator API: 1.5.0,
  PyTorch: 1.7.1+cu110
C:₩Users₩was56₩anaconda3₩envs₩UnityMLAgent₩lib₩site-packages₩torch₩cuda₩__init__.py:52: UserWarn
und no NVIDIA driver on your system. Please check that you have an NVIDIA GPU and installed a dr
com/Download/index.aspx (Triggered internally at  ..₩c10₩cuda₩CUDAFunctions.cpp:100.)
  return torch._C._cuda_getDeviceCount() > 0
[INFO] Listening on port 5004. Start training by pressing the Play button in the Unity Editor.
```

[8] https://unity3dstudy.com/2020/08/26/MummyBasic-for-ML-Agents-03/

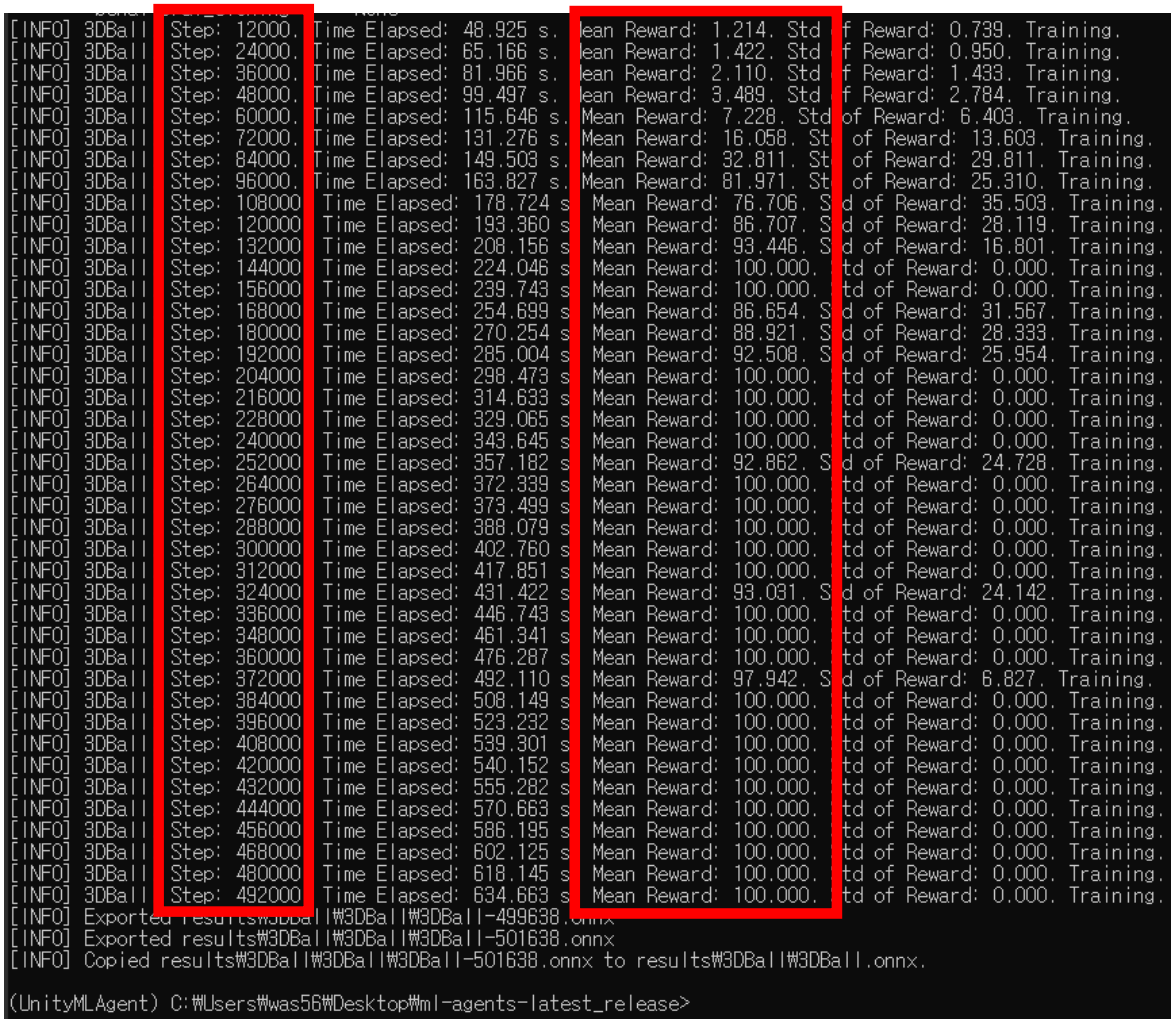
[9] https://unity3dstudy.com/2020/08/26/MummyBasic-for-ML-Agents-03/



유니티 에디터의 실행과 터미널의 연결이 성공하면 위처럼 3DBall.yaml에 설정해 놓은 파라미터들이 명시되며 명시되어 있는 파라미터 설정대로 진행한다.

따라서 learning_rate인 학습률 0.0003, hidden_unit인 히든유닛의 개수 128개, num_layers인 레이어의 개수 2개, max_steps인 반복 최대 횟수 500000번, summary_freq인 요약 명시를 12000에 1번으로 실행된다.

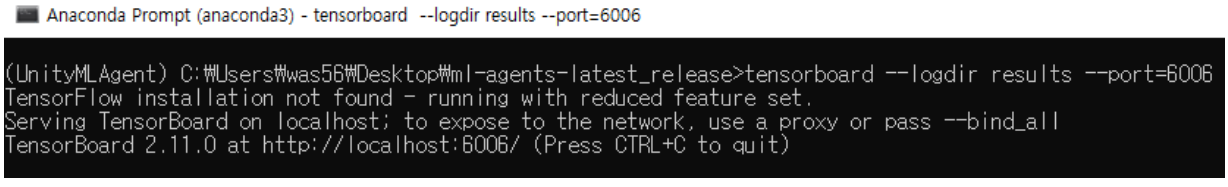
학습하면 아래와 같은 결과가 나오며



Step 수에 따라 점점 보상치가 증가하여 최종적으로 100에 수렴하는 것을 볼 수 있다.

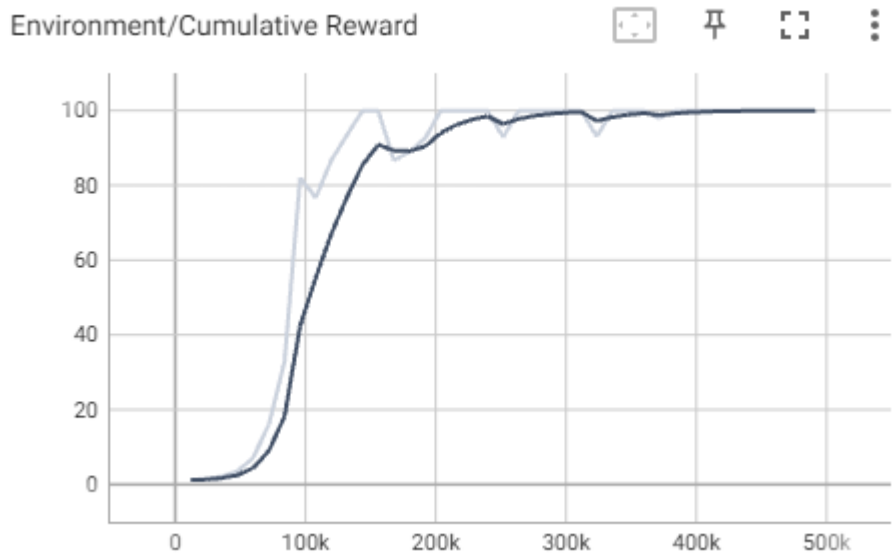
해당 결과들을 TensorBoard를 통해 그래프로 볼 수 있으며

“tensorboard --logdir results --port=6006”(mlagents docs 발췌[10])를 실행하면 아래와 같이 localhost:6006에서 볼 수 있다고 명시한다.

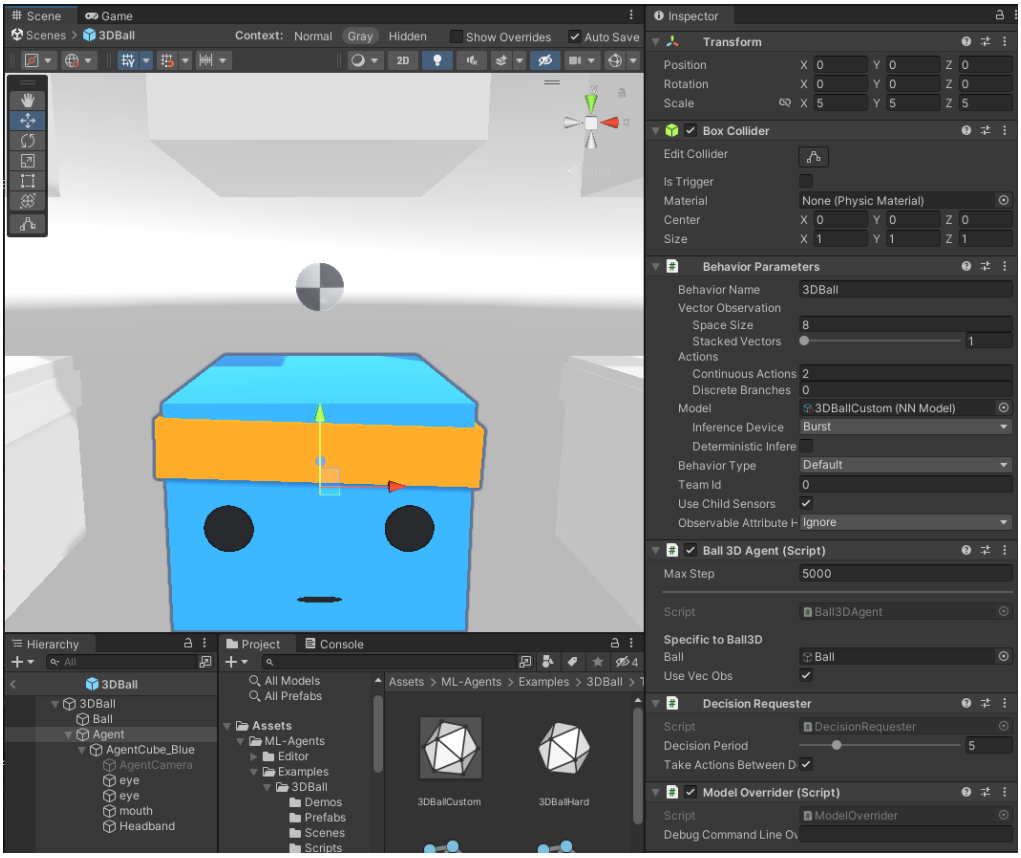


[10] https://github.com/Unity-Technologies/ml-agents/blob/develop/docs/Getting-Started.md

따라서 해당 모델의 결과 값의 그래프는 아래와 같이 나왔다.

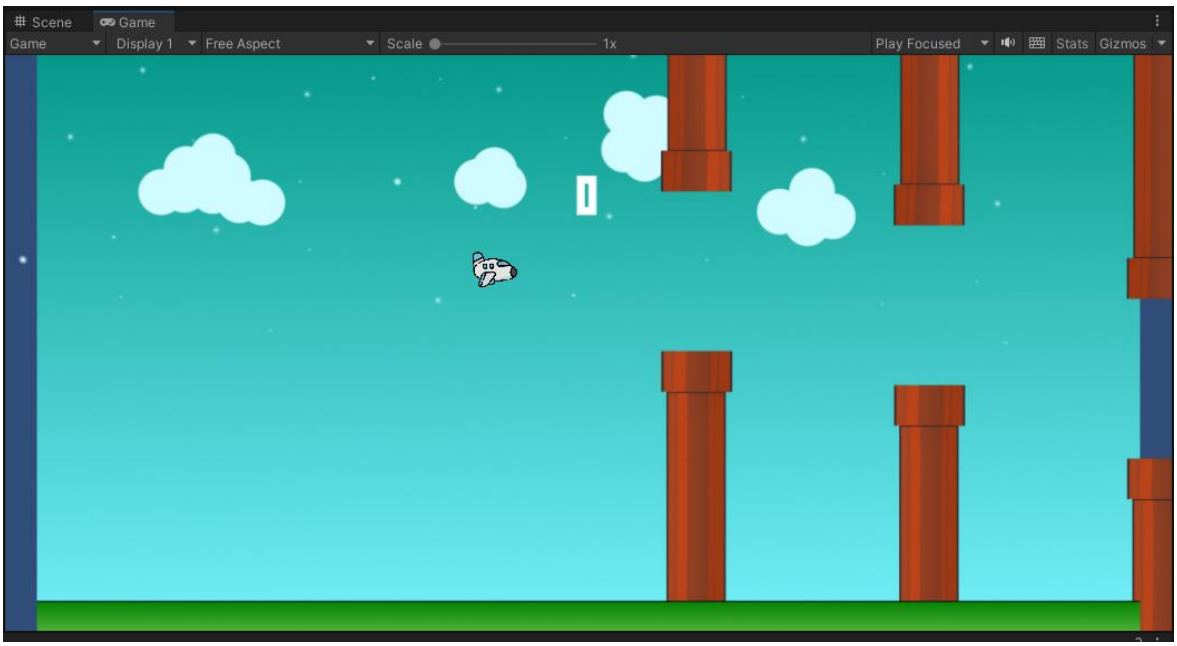


그리고 해당 학습으로 나온 모델을 아래와 같이 3DBallCustom 모델을 Behavior Parameter에 적용시키면 정상적으로 잘 작동하는 것을 볼 수 있었다.

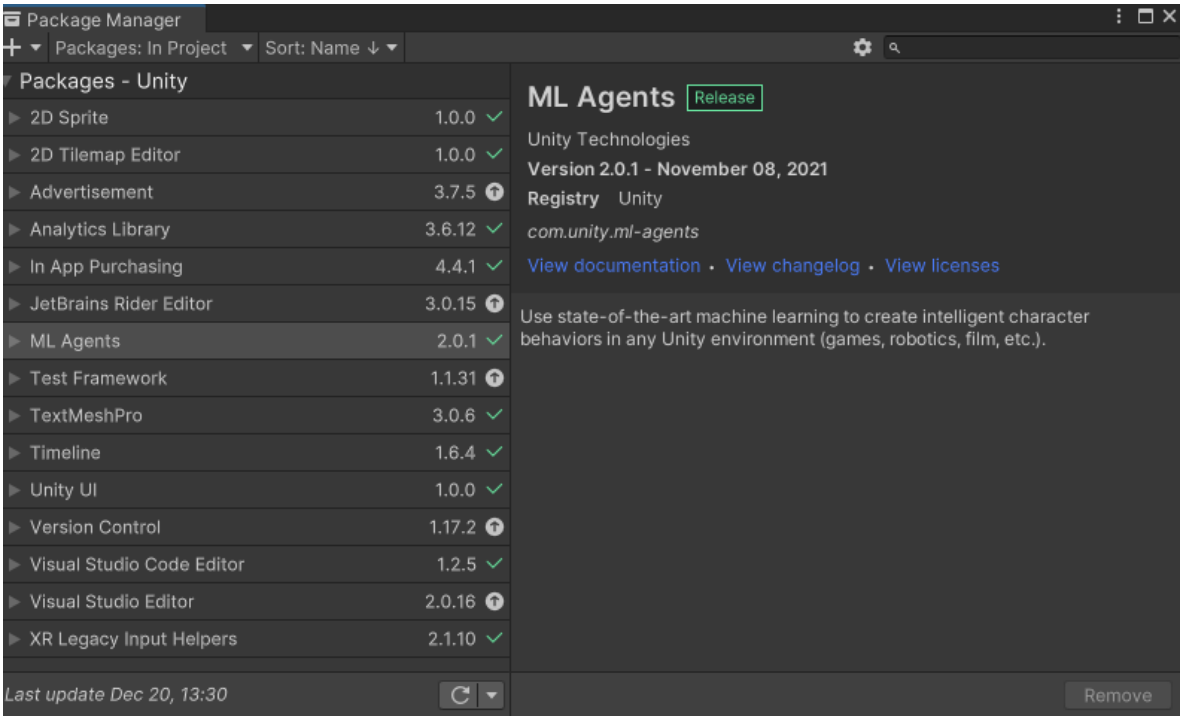


D. 간단한 게임에 적용하기 (사전 준비)

Flappy-Plane Game은 아래와 같이 나무 사이를 지나가 점수를 얻는 방식에 게임이다.



해당 프로젝트에 mlagents를 적용시키기 위해 ML Agent 패키지를 설치하였다.



해당 게임은 위로 무한정 올라가면 나무에 닿지 않아 편하게 최고 점수를 달성할 수 있는 버그가 존재하여 위에 바닥을 배치하여 화면 위로 나가지 못하게 했다.

AIAgent 스크립트를 새로 작성하여 아래와 같이 작성하였고 Initialize() 메소드에는

```
public class AIAgent : Agent
{
    public Rigidbody2D playerRb;  Unchanged
    public GameManager manager;  Unchanged
    private TapController playerInput;

    public bool getScore = false;  Unchanged
    public bool dead = false;  Unchanged

    public override void Initialize()
    {
        playerRb = GetComponent<Rigidbody2D>();
        playerInput = GetComponent<TapController>();
        manager = GameObject.Find("Main Camera").GetComponent<GameManager>();
    }
```

Player객체의 RigidBody와 PlayerInput을 받는 Player객체의 TapController, 게임을 관리하는 GameManager를 캐시에 저장하게끔 하였다.

OnEpisodeBegin() 메소드에는

```
public override void OnEpisodeBegin()
{
    dead = false;
    manager.StartGame();
}
```

게임을 바로 시작할 수 있도록 manager.StartGame()을 실행시키도록 했다.

Agent가 관측할 관측치로 CollectObservations(VectorSensor sensor)에는

```
public override void CollectObservations(VectorSensor sensor)
{
    sensor.AddObservation(transform.position.x);
    sensor.AddObservation(transform.position.y);
    sensor.AddObservation(playerRb.velocity.y);
}
```

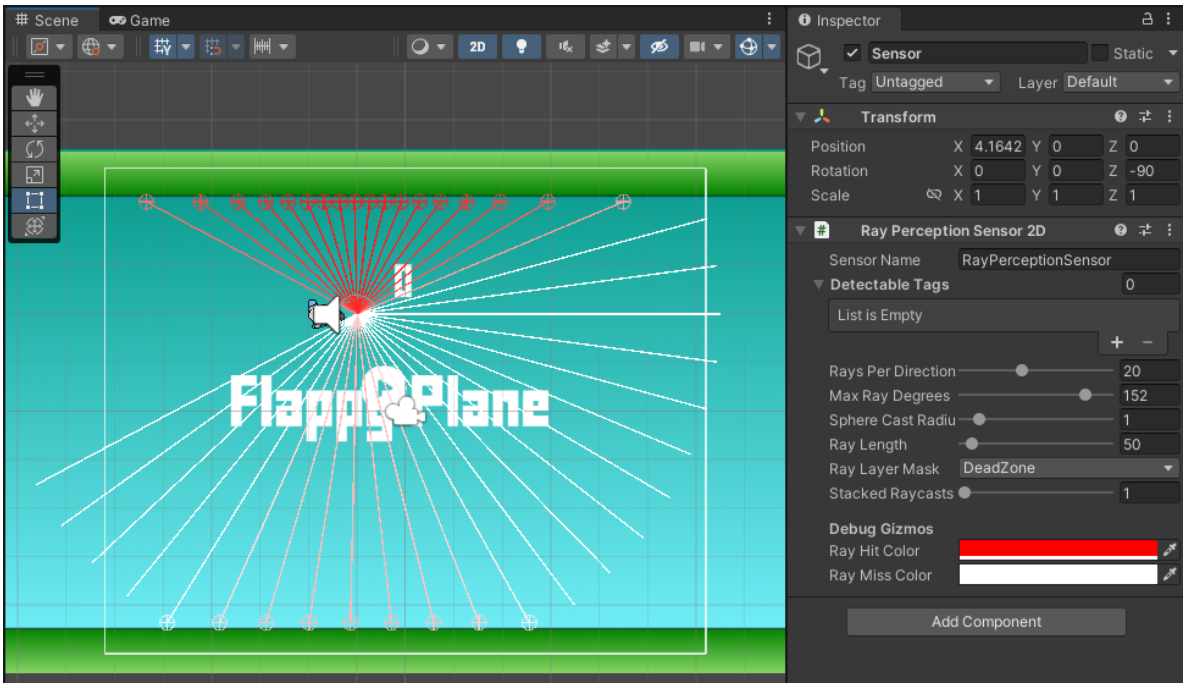
Player 객체의 위치, 속도(y값에 대해서만)를 관측하게 하였다.

OnActionReceived(ActionBuffers actionBuffers) 메소드에는

```
public override void OnActionReceived(ActionBuffers actionBuffers)
{
    var jump int = actionBuffers.DiscreteActions[0];

    switch (jump)
    {
        case 0:
            playerInput.pressedJump = true;
            SetReward(-0.05f);
            break;
        case 1:
            break;
    }

    if(getScore)
    {
        SetReward(3.0f);
        getScore = false;
    }

    if(!dead) SetReward(0.5f);
    else
    {
        manager.ConfirmedGameOver();
        SetReward(-10.0f);
        EndEpisode();
    }

}
```

점프를 최소화 하기 위해 미미한 수준의 패널티를 주었고 계속 살아 있으면 0.5씩 보상값, 스코어를 얻으면 3 보상값, 죽으면 -10 보상값을 받게끔 지정하였다.

추가로 Agent가 장애물을 인식하기 위해서 닿으면 죽는 패널에 DeadZone 레이어를 추가하고 Ray Perception 2D라는 센서를 통해 장애물을 인식할 수 있게 끔 추가하였다.



해당 센서는 Ray Layer Mask를 통해 장애물만 인지하고 Rays Per Direction과 Max Ray Degrees의 값을 조정하여 볼 수 있는 범위와 밀집도를 조정하였다.

그리고 Agent가 Player 객체 및 게임을 조종할 수 있게끔 하기 위해

Player 객체가 나무 사이를 지나서 점수를 얻으면 agent.getScore로 알려주고 죽으면 agent.dead를 통해 Agent에게 알리도록 하였다.

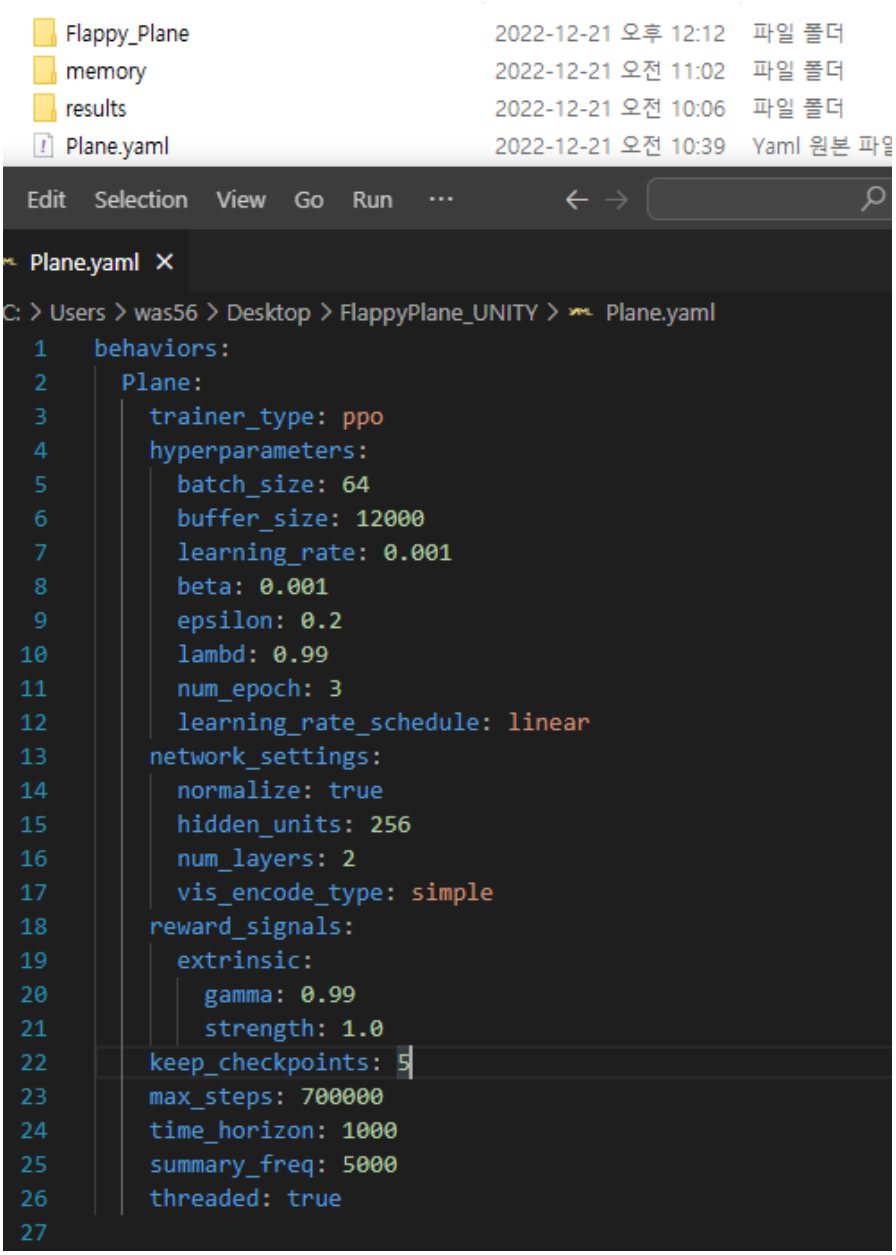
```
void OnTriggerEnter2D(Collider2D col){

    if (col.gameObject.tag == "ScoreZone") {

        OnPlayerScored ();//event GameManagera gönderilir.
        scoreAudio.Play();
        agent.getScore = true;
    }

    if (col.gameObject.tag == "DeadZone")
    {
        agent.dead = true;
        rigidbody.simulated = false;
        OnPlayerDied ();//event Gamemanagera gönderilir.
        dieAudio.Play();
    }
```

그리고 게임을 Agent가 받을 수 있도록 bool 형태의 변수로 받아 조작할 수 있도록 하였다.

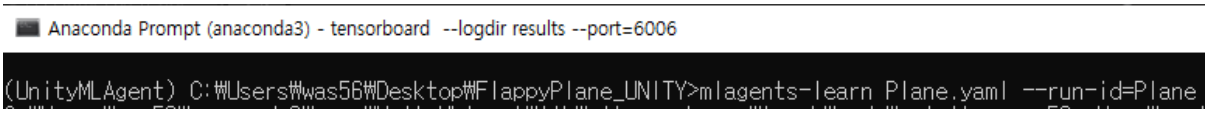
```
void Update(){
    if (game.GameOver) {
        rigidbody.simulated = false;
        return;
    }

    if(pressedJump /*Input.GetMouseButtonDown(0)*/)
    {
        tapAudio.Play ();
        transform.rotation = forwardrotation;
        rigidbody.velocity = (Vector2)Vector3.zero;
        rigidbody.AddForce (Vector2.up * tapForce, ForceMode2D.Force);

        pressedJump = false;
    }
```

E. 간단한 게임에 적용하기 (모델 실습)

프로젝트에 미리 Plane.yaml이라는 설정을 먼저 만들어 두었고 아래 설정과 같이 진행하였다. (파라미터 값은 실습 하면서 바뀔 예정)



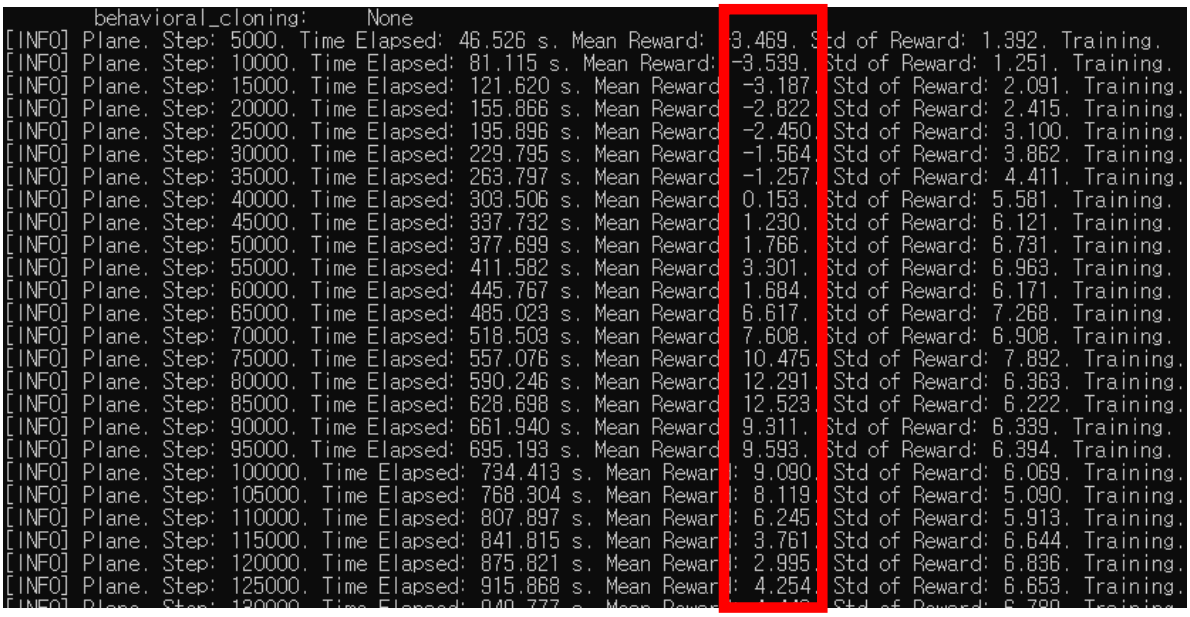
그리고 해당 디렉토리에서 mlagents로 학습을 하였다.



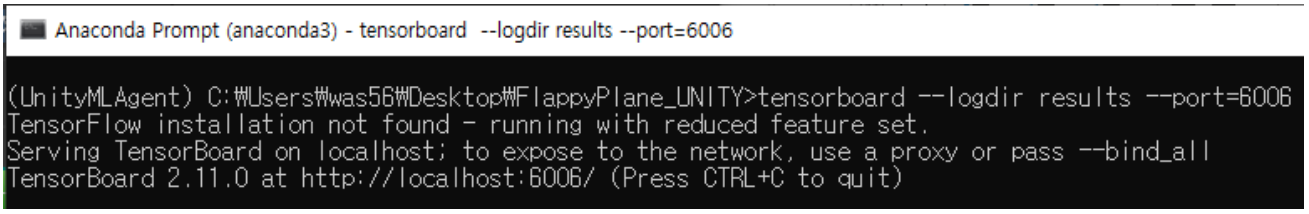
i. 첫 번째 시도

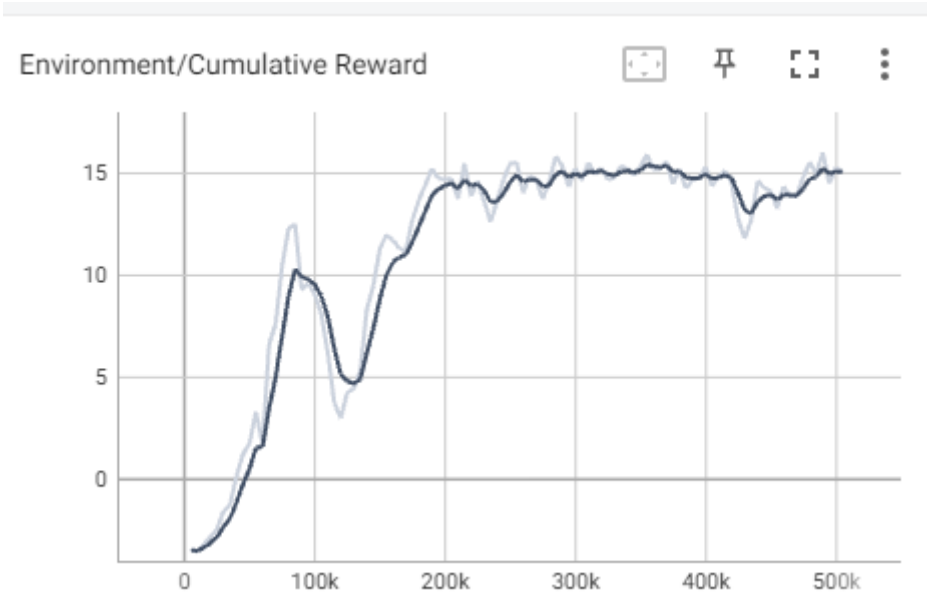
해당 파라미터에서 learning rate 를 0.1 로 두고 실행하였다.

그 결과 15 에서 16 사이의 임계값을 넘지 못하면서 학습하였다.



해당 결과를 tensorBoard 를 통해 그래프로 그리면





해당 그래프가 나온 것을 볼 수 있으며 모델 적용 결과 0 점에서 2 점 사이만을 갈 수 있었다.

ii. 두 번째 시도

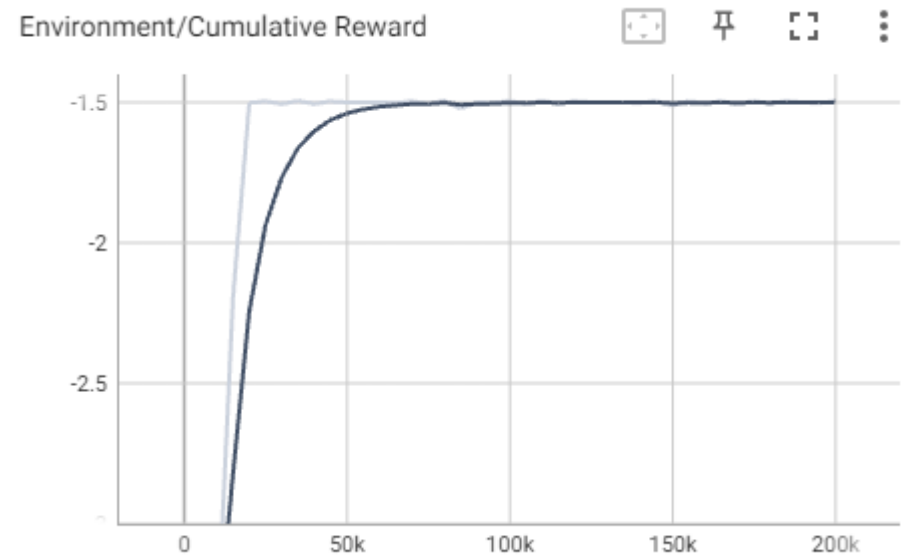
```
[INFO] Hyperparameters for behavior name Plane:
        trainer_type:   ppo
        hyperparameters:
          batch_size:   64
          buffer_size:  12000
          learning_rate:        0.01
          beta: 0.001
          epsilon:      0.2
          lambd:        0.99
          num_epoch:    3
          shared_critic:        False
          learning_rate_schedule:       linear
          beta_schedule:        linear
          epsilon_schedule:     linear
        network_settings:
          normalize:    True
          hidden_units: 256
          num_layers:   2
          vis_encode_type:      simple
          memory:       None
          goal_conditioning_type:       hyper
          deterministic:        False
        reward_signals:
          extrinsic:
            gamma:      0.99
            strength:   1.0
            network_settings:
              normalize:        False
              hidden_units:     128
              num_layers:       2
              vis_encode_type:  simple
              memory:   None
              goal_conditioning_type:   hyper
              deterministic:    False
        init_path:      None
        keep_checkpoints:       5
        checkpoint_interval:    500000
        max_steps:      5000000
        time_horizon:   1000
        summary_freq:   5000
        threaded:       True
        self_play:      None
        behavioral_cloning:     None
```

첫 번째 시도와 파라미터는 똑같지만 다른 결과가 나올 수 있다는 가능성을 고려하여 다시 한번 해보았다.

그 결과 이번에는 약 -1.5 로 고정되며 점프를 하지 않고 떨어지기 만을 반복하였다.

```
[INFO] Resuming training from step 145013.
[INFO] Plane. Step: 150000. Time Elapsed: 47.155 s. Mean Reward: -1.511. Std of Reward: 0.655. Training.
[INFO] Plane. Step: 155000. Time Elapsed: 85.483 s. Mean Reward: -1.496. Std of Reward: 0.633. Training.
[INFO] Plane. Step: 160000. Time Elapsed: 132.395 s. Mean Reward: -1.504. Std of Reward: 0.632. Training.
[INFO] Plane. Step: 165000. Time Elapsed: 171.212 s. Mean Reward: -1.496. Std of Reward: 0.633. Training.
[INFO] Plane. Step: 170000. Time Elapsed: 219.374 s. Mean Reward: -1.505. Std of Reward: 0.631. Training.
[INFO] Plane. Step: 175000. Time Elapsed: 259.435 s. Mean Reward: -1.496. Std of Reward: 0.633. Training.
[INFO] Plane. Step: 180000. Time Elapsed: 299.230 s. Mean Reward: -1.504. Std of Reward: 0.632. Training.
[INFO] Plane. Step: 185000. Time Elapsed: 359.568 s. Mean Reward: -1.496. Std of Reward: 0.633. Training.
[INFO] Plane. Step: 190000. Time Elapsed: 399.239 s. Mean Reward: -1.500. Std of Reward: 0.633. Training.
[INFO] Plane. Step: 195000. Time Elapsed: 448.321 s. Mean Reward: -1.502. Std of Reward: 0.631. Training.
[INFO] Plane. Step: 200000. Time Elapsed: 488.190 s. Mean Reward: -1.496. Std of Reward: 0.633. Training.
[INFO] Learning was interrupted. Please wait while the graph is generated.
[INFO] Exported results₩Plane₩Plane₩Plane-200303.onnx
[INFO] Copied results₩Plane₩Plane₩Plane-200303.onnx to results₩Plane₩Plane.onnx.
```

해당 결과를 그래프로 표현하면 아래와 같다.



해당 결과들을 통해 learning rate 가 너무 커서 최선의 모델을 찾지 못하고 지나치는 것이라는 결론이 나왔다.

iii. 세 번째 시도

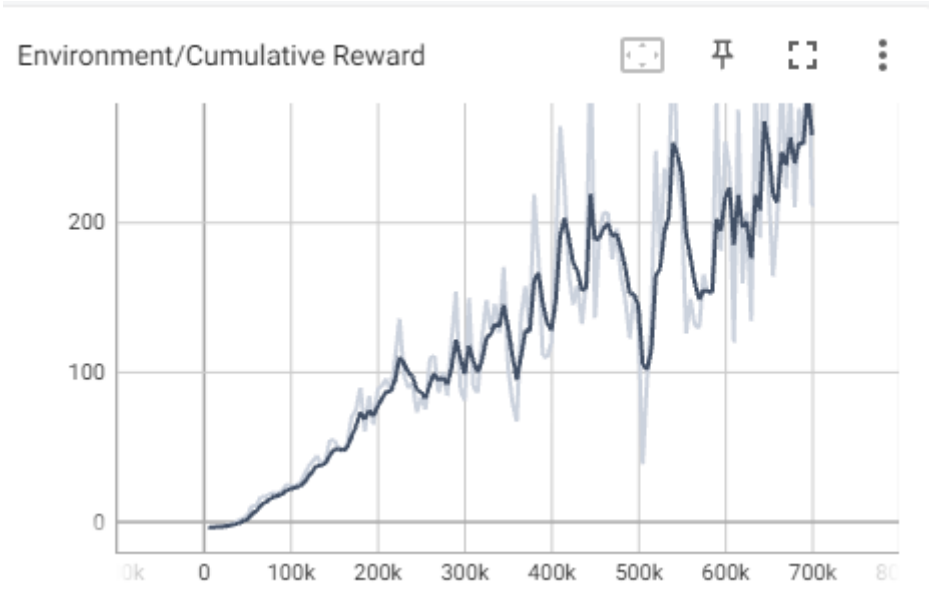
```
trainer_type:    ppo
hyperparameters:
  batch_size:    64
  buffer_size:   12000
  learning_rate:         0.001
  beta: 0.001
  epsilon:       0.2
  lambd:         0.99
  num_epoch:     3
  shared_critic:         False
  learning_rate_schedule:        linear
  beta_schedule:         linear
  epsilon_schedule:      linear
network_settings:
  normalize:     True
  hidden_units: 256
  num_layers:    2
  vis_encode_type:       simple
  memory:        None
  goal_conditioning_type:        hyper
  deterministic:         False
reward_signals:
  extrinsic:
    gamma:       0.99
    strength:    1.0
    network_settings:
      normalize:         False
      hidden_units:      128
      num_layers:        2
      vis_encode_type:   simple
      memory:    None
      goal_conditioning_type:    hyper
      deterministic:     False
init_path:       None
keep_checkpoints:        5
checkpoint_interval:     500000
max_steps:       5000000
time_horizon:    1000
summary_freq:    5000
threaded:        True
self_play:       None
behavioral_cloning:      None
```

첫 번째와 두 번째 시도에서 learning rate가 너무 큰 것인가 염려하여 0.001로 지정을 하고 모델을 학습시켜 보았다.

그 결과 보상값이 점점 올라가며 학습하면 학습할수록 유의미한 결과를 보여주었다.

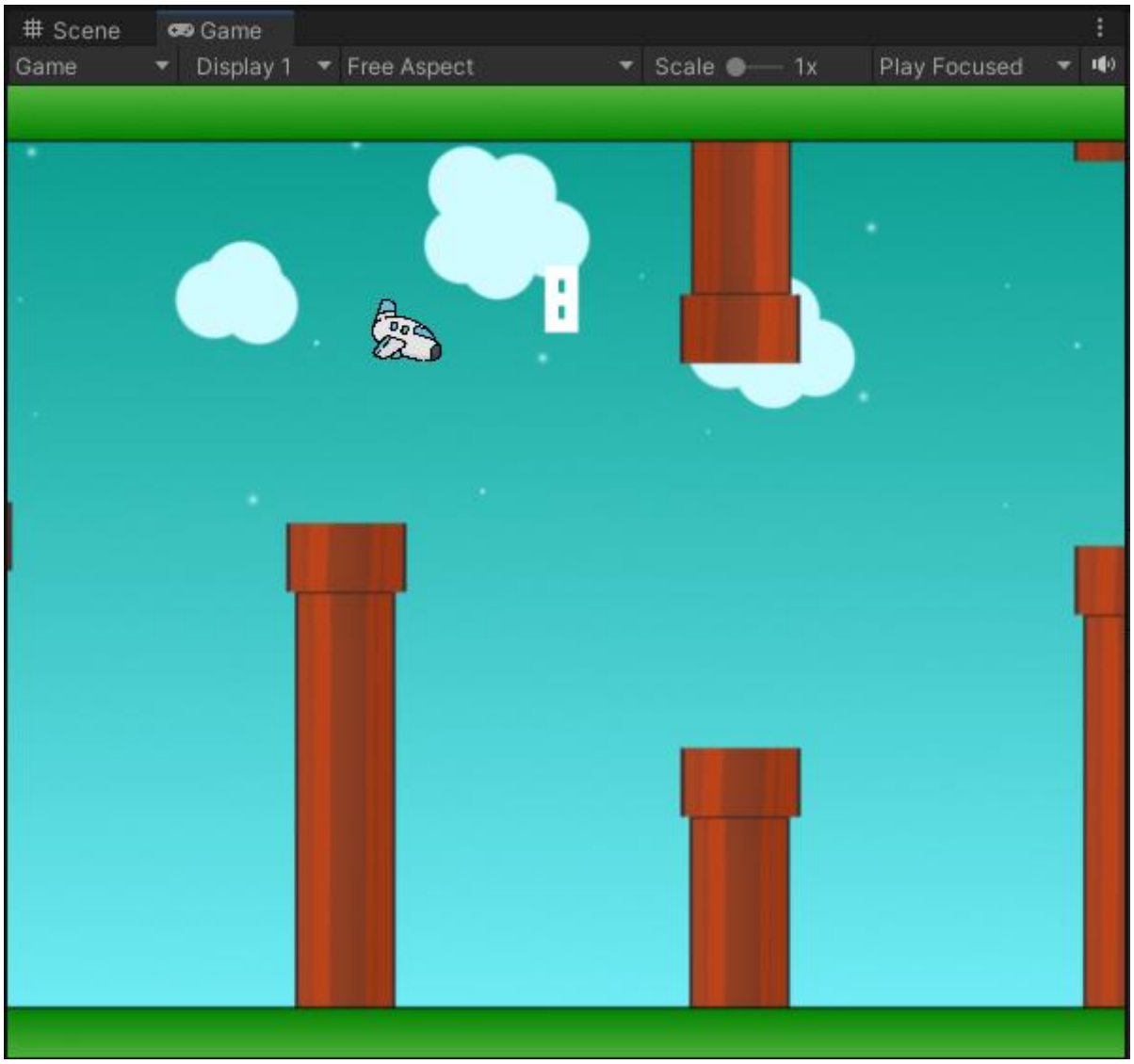
```
[INFO] Resuming from results₩Plane₩Plane.
[INFO] Resuming training from step 100121.
[INFO] Plane. Step: 105000. Time Elapsed: 42.452 s. Mean Reward: 24.321. Std of Reward: 22.660. Training.
[INFO] Plane. Step: 110000. Time Elapsed: 74.685 s. Mean Reward: 25.413. Std of Reward: 17.778. Training.
[INFO] Plane. Step: 115000. Time Elapsed: 114.421 s. Mean Reward: 30.459. Std of Reward: 25.434. Training.
[INFO] Plane. Step: 120000. Time Elapsed: 147.664 s. Mean Reward: 36.821. Std of Reward: 26.673. Training.
[INFO] Plane. Step: 125000. Time Elapsed: 191.448 s. Mean Reward: 40.561. Std of Reward: 33.530. Training.
[INFO] Plane. Step: 130000. Time Elapsed: 220.747 s. Mean Reward: 44.163. Std of Reward: 37.178. Training.
[INFO] Plane. Step: 135000. Time Elapsed: 254.823 s. Mean Reward: 37.490. Std of Reward: 28.358. Training.
[INFO] Plane. Step: 140000. Time Elapsed: 295.354 s. Mean Reward: 41.924. Std of Reward: 25.077. Training.
[INFO] Plane. Step: 145000. Time Elapsed: 327.010 s. Mean Reward: 54.842. Std of Reward: 46.135. Training.
[INFO] Plane. Step: 150000. Time Elapsed: 366.375 s. Mean Reward: 54.587. Std of Reward: 43.639. Training.
[INFO] Plane. Step: 155000. Time Elapsed: 399.529 s. Mean Reward: 49.107. Std of Reward: 30.215. Training.
[INFO] Plane. Step: 160000. Time Elapsed: 432.757 s. Mean Reward: 46.872. Std of Reward: 34.214. Training.
[INFO] Plane. Step: 165000. Time Elapsed: 473.834 s. Mean Reward: 52.612. Std of Reward: 50.098. Training.
[INFO] Plane. Step: 170000. Time Elapsed: 505.671 s. Mean Reward: 70.532. Std of Reward: 53.742. Training.
[INFO] Plane. Step: 175000. Time Elapsed: 545.252 s. Mean Reward: 74.983. Std of Reward: 66.054. Training.
[INFO] Plane. Step: 180000. Time Elapsed: 577.301 s. Mean Reward: 89.720. Std of Reward: 63.816. Training.
[INFO] Plane. Step: 185000. Time Elapsed: 622.684 s. Mean Reward: 60.314. Std of Reward: 50.279. Training.
[INFO] Plane. Step: 190000. Time Elapsed: 649.280 s. Mean Reward: 84.385. Std of Reward: 69.633. Training.
[INFO] Plane. Step: 195000. Time Elapsed: 684.355 s. Mean Reward: 65.088. Std of Reward: 56.040. Training.
[INFO] Plane. Step: 200000. Time Elapsed: 723.151 s. Mean Reward: 88.740. Std of Reward: 68.594. Training.
[INFO] Learning was interrupted. Please wait while the graph is generated.
[INFO] Exported results₩Plane₩Plane₩Plane-200322.onnx
[INFO] Copied results₩Plane₩Plane₩Plane-200322.onnx to results₩Plane₩Plan.onnx.
[ERROR] SubprocessEnvManager had workers that didn't signal shutdown
[INFO] Plane. Step: 630000. Time Elapsed: 42.574 s. Mean Reward: 134.423. Std of Reward: 137.203. Training.
[INFO] Plane. Step: 635000. Time Elapsed: 74.796 s. Mean Reward: 291.778. Std of Reward: 327.892. Training.
[INFO] Plane. Step: 640000. Time Elapsed: 111.236 s. Mean Reward: 190.091. Std of Reward: 165.673. Training.
[INFO] Plane. Step: 645000. Time Elapsed: 142.562 s. Mean Reward: 369.333. Std of Reward: 226.855. Training.
[INFO] Plane. Step: 650000. Time Elapsed: 173.872 s. Mean Reward: 222.654. Std of Reward: 172.370. Training.
[INFO] Plane. Step: 655000. Time Elapsed: 213.829 s. Mean Reward: 164.077. Std of Reward: 145.762. Training.
[INFO] Plane. Step: 660000. Time Elapsed: 247.363 s. Mean Reward: 204.667. Std of Reward: 182.002. Training.
[INFO] Plane. Step: 665000. Time Elapsed: 284.223 s. Mean Reward: 305.125. Std of Reward: 212.752. Training.
[INFO] Plane. Step: 670000. Time Elapsed: 318.159 s. Mean Reward: 223.000. Std of Reward: 219.218. Training.
[INFO] Plane. Step: 675000. Time Elapsed: 365.077 s. Mean Reward: 288.643. Std of Reward: 212.182. Training.
[INFO] Plane. Step: 680000. Time Elapsed: 391.970 s. Mean Reward: 210.462. Std of Reward: 129.945. Training.
[INFO] Plane. Step: 685000. Time Elapsed: 425.952 s. Mean Reward: 276.389. Std of Reward: 189.352. Training.
[INFO] Plane. Step: 690000. Time Elapsed: 459.962 s. Mean Reward: 251.500. Std of Reward: 280.428. Training.
[INFO] Plane. Step: 695000. Time Elapsed: 494.769 s. Mean Reward: 343.571. Std of Reward: 211.561. Training.
[INFO] Plane. Step: 700000. Time Elapsed: 537.754 s. Mean Reward: 211.167. Std of Reward: 219.827. Training.
```

해당 결과를 그래프로 보여주면 아래와 같다.



세 번째 시도가 제일 학습을 잘한 모델이며 바로 프로젝트에 적용해보았다.

그 결과 Agent 는 잘 작동하여서 0 점일 때는 많지만 반대로 10 점 이상 따는 경우도 많은 모델이 되었다.



4. 결과에 대한 고찰

   해당 모델은 전체적으로 잘 작동하지만 보상값의 표준오차가 큰 나머지 일부 케이스에 대해서만 과적합하다고 볼 수 있다. 따라서 이 표준오차를 줄이기 위해서는 해당 모델을 좀 더 느슨하게 (혹은 learning rate 가 낮도록) 디자인해야 할 것이다.

5. 미래에 쓰일 수 있는 일

   Mlagents 로 만든 AI 를 이용하여 적 AI 를 만들고 경쟁하도록 할 수 있을 것이며 플레이어에 따른 체감 난이도는 Agent 의 목표 보상값을 지정한다면 난이도에 맞는 적 AI 를 만들 것을 생각된다. 이 덕분에 AI 의 난이도 조절을 mlagents 를 이용하여 비교적 쉽게 구현할 수 있을 것이며 행동 트리, 미니맥스 알고리즘처럼 비교적 구현을 덜 하여도 될 것이다.

6. 참고 사이트 및 코드

Unity 게임 프로젝트 다운 사이트
https://code-projects.org/flappy-plane-game-in-unity-engine-with-source-code/

Unity MLAgent 사용법, 설치 및 개인 환경 구축

https://github.com/Unity-Technologies/ml-agents/blob/develop/docs/Getting-Started.md

https://github.com/Unity-Technologies/ml-agents/blob/develop/docs/Installation.md

https://github.com/Unity-Technologies/ml-agents/tree/develop/docs

Unity mlagents 제공 영상

https://www.youtube.com/watch?v=GWd4g1qCDxo&t=557s

mlagents 를 이용한 프로젝트 블로그

https://unity3dstudy.com/2020/08/26/MummyBasic-for-ML-Agents-03/

https://unity3dstudy.com/2020/08/22/MummyBasic-for-ML-Agents-02/