강화학습을 통한
인공 생명체(나비) 구현

서강대학교 일반대학원
미디어 아트앤테크놀로지 전공
강명진

<목 차>

Ⅰ. 서론

1.1 연구 배경

정말로 살아있는 생명체처럼 인공지능 생명체를 구현할 수 있을까. 그렇다면 생명체가 보여주는 행동보다 그 행동을 하는 이유를 주목해야 한다는 관점에서 이 프로젝트를 고안하였다. 동화 '해와 바람'에서는 해와 바람이 지나가는 나그네의 겉옷을 누가 빨리 벗길 수 있는지 내기를 한다. 바람이 아무리 센 바람을 불어도 옷깃을 여미기만 했던 나그네는 결국 따뜻한 햇살에 겉옷을 벗어버리고 만다. 이처럼 인공지능 생명체에게 어떤 행동을 학습 시킨다는 것은 단순하게 인간이 인공지능 생명체에게 어떤 행동을 알려주는 것이 아닌, 인공지능 생명체로 하여금 어떤 행동을 유발하는 욕구를 느끼게 하고, 인공지능 생명체가 스스로 적절한 행동을 선택할 수 있는 과정이 되어야할 것이다. 예기치 못한 상황에서 인공지능 생명체가 스스로 적절한 행동을 고르게 된다면 우리는 인공지능 생명체를 정말로 살아있는 생명체라 부를 수 있을 것이다.

스키너는 훗날 '스키너 상자' 라고 알려지는 실험 장치를 발명해 독창적인 행동 실험을 했다. 바로 상자 안에 쥐를 넣고 쥐가 그 안에 설치된 막대를 누를 때마다 먹이가 제공되도록 만든 장치였다. 시간이 지나면서 쥐가 먹이를 얻기 위해 의도적으로 막대를 누르는 것을 확인할 수 있었다. 막대를 누르게 만드는 직접적인 자극은 없었고 쥐가 우연히 막대를 누르다가 먹이를 얻는 법을 학습하게 된 것이다. 스키너에 따르면 쥐가 막대를 누르는 행동을 행동의 긍정적인 결과(먹이)에 의해 '강화' 된 것이었다. 이 행동심리학에서 영감을 받은 강화학습(Reinforcement Learning)은 인공지능 머신러닝(Machine Learning)의 한 유형으로 어떤 환경 내에 정의된 에이전트가 현재의 상태를 인식하여, 선택 가능한 행동들 중 보상(Reward)을 최대화하는 행동 혹은 행동 순서를 선택하는 방법이다.[1)]

본 프로젝트에서 인공지능 생명체의 대상으로 '나비' 를 낙점하였다. 문학이나 미술에서 나비는 생명 혹은 영혼을 상징하고, 종교에서는 부활의 상징으로 보기도 하며, 문화에서는 인내심, 변화, 희망 등의 삶의 다양한 모습을 상징한다. 특히 프랑스의 박물학자 마르셀 롤랑은 "나비가 인간의 고통스러운 삶에 위안을 준다" 하였는데, 이는 보잘것없는 애벌레에서 그 어떤 생명체보다 화려하게 변신하는 나비의 특성에 많은 이들이 사로잡히기 때문일 것이다.[2)] 이처럼 인간이 나비에게서

1) 교양인 "[스키너의 행동심리학] 책 소개"

‘남루한 현실에서 초월하는 아름다운 영혼’을 떠올린다는 점에서 우리는 나비의 ‘초월’ 이라는 상징성에 주목하려 하며, 코로나 바이러스의 창궐로 어디에도 가기 어려워진 이 시기에 어떤 한계에도 얽매이지 않고 훨훨 날아다니는 나비를 대상화 하여 대리 만족을 꾀하려 한다.

1.2 연구 내용

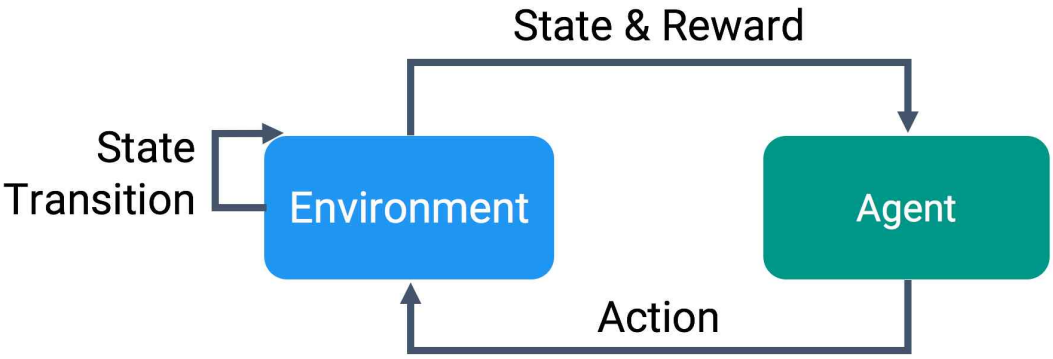
금번 프로젝트에서는 강화학습을 통해 정말로 살아있는 생명체, 나비를 구현하려 한다. Floating Island(유니티 ML-Agents : Hummingbirds 환경)에 상주하는 나비는 사용자가 없을 때는 꽃에 다가가 꿀을 먹거나 이리저리 날아다니게 된다. 만약 사용자가 실제 공간에 등장하면 사용자에게 접근하여 사용자의 손을 따라가게 된다. 강화학습을 위해서는 서버 PC(Server PC)를 원격으로 이용하게 되며, 서버 PC에서 사용자는 여러 나비 중에 원하는 나비를 고르고, 나비가 사용자의 손을 따라올 수 있게 길들인다. 이때 나비는 꽃에 다가가 꿀을 먹거나 사용자의 손을 따라가는 행동을 해냈을 때 보상을 받는 것만으로도 해당 행동들을 스스로 습득할 수 있게 된다.

Ⅱ. 본론

2.1 강화학습

강화학습(Reinforcement Learning)은 환경과의 상호작용을 통해 보상을 최대화하는 행동이나 행동 순서 등을 선택하여 반복 학습하는 방법이다. 특정한 환경에 정의된 에이전트(Agent)는 학습을 수행하거나 의사결정을 내려야하는 주체로, 현재 상태(State)와 환경(Environment)를 인식하며 행동(Action)을 실행하고, 에이전트의 행동으로 인해 환경은 또 다른 상태로 변경된다. 에이전트가 올바른 행동을 수행할 시 에이전트에게는 보상(Reward)를 제공한다. 행동은 환경이 변경된 상태에서 또 다른 행동을 수행하게 되며 강화학습은 에이전트의 보상을 최대화하는 행동 혹은 행동 순서를 선택하게 만드는 방법이다.[3)]

2) 최민성, “문화콘텐츠 주제학 시론 나비 상징을 중심으로”, 인문콘텐츠학회
3) 인공지능신문 “강화학습은 무엇이며, 어떻게 설정하고 해결해야 하는가?”



<그림-1>

2.2 ML-Agents

강화학습을 구성하는 요소는 크게 강화학습 알고리즘과 해당 알고리즘을 테스트할 환경이라고 할 수 있다. 기존에 강화학습을 연구하거나 이용할 때에는 강화학습 알고리즘을 구성하는 것도 문제였지만 강화학습 알고리즘을 실행할 환경을 구성하는 것 또한 매우 어려운 문제였다. 물론 유니티와 같은 툴을 이용하면 환경을 개발하는 것이 비교적 수월하지만 유니티 내부에서 강화학습 알고리즘, 특히 딥러닝(Deep Learning)을 이용한 강화학습 알고리즘을 구현하는 것 또한 매우 어렵다. 이런 어려움을 해결하기 위해 개발된 것이 유니티 ML-Agents(Machine Learning Agents)이다. 유니티 ML-Agents는 머신러닝 기술, 특히 강화학습을 유니티에서 간편하게 사용할 수 있도록 만들어진 개발 도구이다.[4)]

유니티 ML-Agents의 내장 알고리즘을 이용하는 과정은 다음과 같다. 먼저 유니티 ML-Agents를 이용해 환경을 제작하고 빌드한다. 그리고 해당 환경의 에이전트를 유니티 ML-Agents에서 기본적으로 제공하는 강화학습 알고리즘을 이용해 학습시킨다. 학습이 완료되고 나면 학습된 딥러닝 모델이 *.nn라는 확장자를 가진 파일로 저장된다. 이 파일을 유니티 환경의 에이전트에게 적용하면 유니티 내부에서 학습된 에이전트의 성능을 테스트해 볼 수도 있고 학습된 에이전트가 적용된 환경을 빌드해서 사용할 수도 있다. 학습된 에이전트가 적용된 환경을 빌드해서 사용하는 경우 빌드 된 환경을 실행하면 에이전트가 환경 내에서 학습된 결과에 따라 행동하게 된다.[5)]

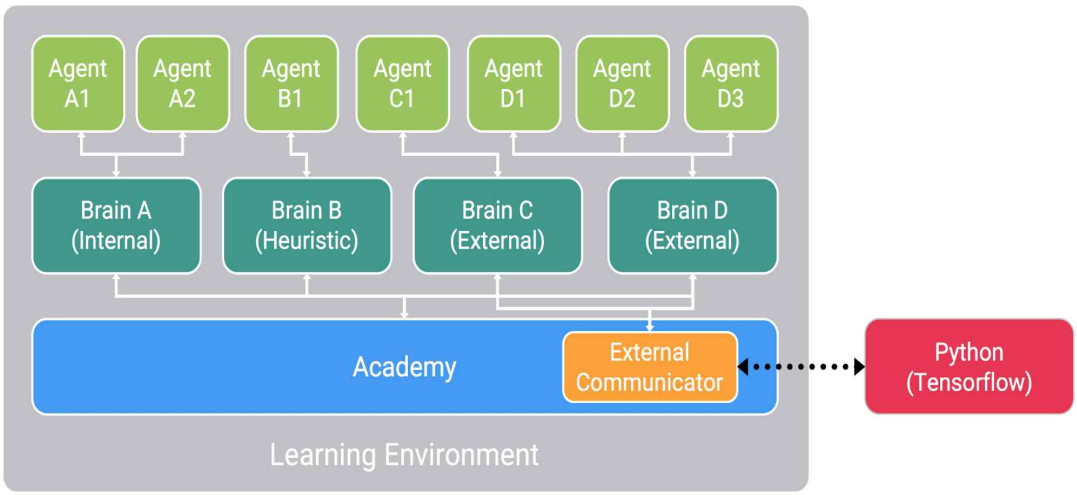
에이전트(Agent) : 환경(Environment)에서 어떠한 행위를 실제로 행하는 행위자

4) 홍성찬, "강화학습을 이용한 격투게임 인공지능 설계 및 구현", 학위논문 사항
5) 위키북스 "텐서플로와 유니티 ML-Agents로 배우는 강화학습"

(Actor)이다. 각 에이전트는 자신의 상태 및 환경에서 발생하는 정보를 토대로 행동(Action)이 결정되었을 때 환경에서 실재로 행동을 하는 주체이다. 에이전트가 실행한 행동이 목표에 부합하면 보상(Reward)를 받고, 목표에 부합하지 않거나 실패할 경우 처벌을 받게 된다.

브레인(Brain) : 각 브레인은 환경에서 발생하는 특정 상태, 행동 공간을 정의하고 연결된 에이전트가 어떤 행동을 취할지 결정하는 정책(Policy) 결정권자이다. 하나의 브레인에 여러 개의 에이전트를 연결할 수 있지만 기본 설정으로 연결된 에이전트들은 서로 정보를 교환, 공유하지 않는다. 하나의 브레인을 통해 각각 독립된 여러 에이전트들을 학습 시킬 수 있으며 이를 통해 서로 다른 학습 결과를 보여주는 에이전트가 발생하게 된다.

아카데미(Academy) : 아카데미는 유니티의 현재 씬(Scene)에 존재하는 모든 브레인을 자식으로 포함하고 있으며 외부 텐서플로우(Tensorflow)의 메시지를 전달 받아 자식 브레인에게 전달하고 에이전트와 브레인으로부터 발생한 정보를 텐서플로우에게 전달하는 역할을 한다. 씬에는 아카데미가 포함되어 있으며 각 에이전트의 훈련을 위한 씬의 범위를 정의하는 기능들이 존재한다.



<그림-2>

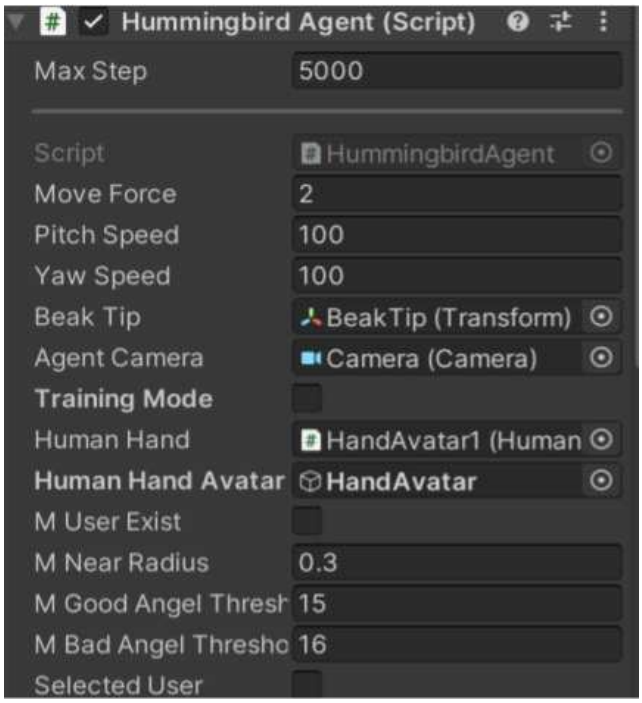
2.3 추진 내용

본 프로젝트에서 에이전트가 되는 나비는 고유의 상태(State) 및 관찰 값(Observation)을 가지고 있고 환경(Environment)에서 고유의 행동(Action)을 하며 환경 내부에서 일어나는 이벤트에 따라 고유의 보상(Reward)를 받는다. 나비의 행

동은 브레인에 의해 결정되며, 브레인은 특정 상태와 행동 공간(Action Space)를 정의하고, 나비가 어떤 행동을 취할지 결정한다.

허밍버드에이전트(HummingbirdAgent) 클래스는 에이전트 클래스를 상속 받고 에이전트 클래스는 MonoBehavior 클래스를 상속 받는다. 허밍버드에이전트 클래스의 기본적인 동작은 MonoBehavior 클래스를 따라가며, 에이전트에게 필요한 기능은 에이전트 클래스에서 가져온다. 허밍버드에이전트 클래스에서는 에이전트에 필요한 기능만을 정의한다.

고유의 상태 및 관측 값을 가지고 있고, 환경 내에서 어떤 행동을 하며 환경내부에서 일어나는 이벤트에 따라 보상을 받는 부분에 해당된다. Max Step속성은 에이전트가 한 에피소드(학습단위)내에서 무작위로 액션(이동)을 시도해보는 최대 횟수를 의미한다. 이 횟수동안 액션을 취했지만 아무런 보상이 없다면 더 이상 학습의 의미가 없기에 에피소드를 종료하고 다시 시작한다. MaxStep의 수치는 환경의 복잡도에 따라서 가장 적절한 값을 찾아야 한다.



<그림-3>

허밍버드에이전트 클래스가 활성화되면 가장 먼저 호출되는 것이 Initialize() 함수인데, 이 함수는 스크립트 인스턴스(Instance)가 로딩될 때 호출되는 Awake() 함수에서 시작된다. 아카데미 클래스 내에 정의된 Awake() 함수는 InitializeEnvironment() 함수를 호출하며 이 함수는 학습 및 환경 설정, 아카데미를

초기화하는 역할을 한다.

```
public override void Initialize()
{
    rigidbody = this.gameObject.GetComponent<Rigidbody>();
    flowerArea = this.gameObject.GetComponentInParent<FlowerArea>();

    reciveIndex = this.gameObject.GetComponent<ReciveIndex>();

    // if not training mode, no max steps, play forever
    if (!trainingMode)
    {
        MaxStep = 0;
    }
}
```

<그림-4>

개발자가 정한 MaxStep 값에 따라 에피소드가 종료되기도 하지만 올바른 결과 혹은 잘못된 결과에 도달하게 되면 에피소드가 종료되고 새로운 에피소드를 시작한다. OnEpisodeBegin() 메소드(Method)는 에피소드가 시작될 때마다 호출되는 메소드로 에이전트를 초기화하고 환경을 재설정한다. 이때 다양한 환경에서 에이전트를 학습시킬 수 있도록 초기화 및 재설정 과정에서 각 변수를 랜덤하게 설정한다.

```
public override void OnEpisodeBegin() //정책망에 weight데이터는 random vector다. action -> observation
{
    if (trainingMode)
    {
        // numberOfGoodSteps = 0;
        //numberOfBadSteps = 0;

        // reset flowers and one agent only
        flowerArea.ResetFlowers();
        if (flowerArea.transform.name != "FloatingIsland")
        {
            float random = Random.Range(0.0f, 1.0f);

            Debug.Log("Episode start");

            if (random < 0.5f)
            {
                //Debug.Log("In this time Flower");
                mUserExist = false;
                humanHandAvatar.SetActive(false);
            }
            else
            {
                //Debug.Log("In this time UserExist");
                mUserExist = true;
                if (humanHandAvatar.transform.position.y < 0)
                {
```

<그림-5>

CollectObservation() 함수는 에이전트의 관찰 값을 수집하는 기능을 한다.이 함수에서 수집한 관찰 값은 다음 VectorAction 값을 받을 때 Input 값으로 사용되어 강화학습시 기초자료로 활용된다. CollectObservation() 함수에 의해 수집된 관찰 값은 브레인에 전달되며, 강화학습을 위한 충분한 자료를 전달할 수 있도록 변수를

설계해야 효율적인 모델을 만들 수 있다. 우선 관찰 값으로 사용자의 유무를 체크하는 mUserExist 값을 저장하고, 나비와 목표(사용자의 손/Floating Island의 꽃) 사이의 계산한 각도를 저장한다. 또 나비의 회전축을 저장하고, 나비와 사용자의 손 사이의 거리를 저장한다. 나비가 사용자의 손을 따라가게 하고, 나비가 꽃에 다가가서 꿀을 먹게 하기 위해 총 19개의 관찰 값을 추출하며 AddObservation() 함수를 통해 관찰 값을 브레인으로 전달한다.

```
public override void CollectObservations(VectorSensor sensor)
{ // 여기서 수집한 observation vector는 다음 action값을 수집할 때 다음 input값으로 선택된다.
    //Debug.Log(" UserExist  " + mUserExist);
    // Debug.Log("In CollectObservations ");
    sensor.AddObservation(mUserExist);//1

    if (humanHandAvatar != null && humanHandAvatar.activeSelf == true)
    {

        Vector3 toHand = humanHandAvatar.transform.position - this.gameObject.transform.position;
        Vector3 targetDir = (toHand).normalized;
        float dot = Vector3.Dot(transform.forward, targetDir);
        float AngleBetweenDegree = Mathf.Acos(dot) * Mathf.Rad2Deg;
        float angle = 1 + (-1 + 1) * ((AngleBetweenDegree - 0) / (180 - 0));
        if (float.IsNaN(angle))
        {
            sensor.AddObservation(new float[1]); //1
        }
        else
        {
            sensor.AddObservation(angle);

        }
        sensor.AddObservation(this.gameObject.transform.localRotation.normalized); //4
        sensor.AddObservation(toHand.normalized); // 3
```

<그림-6>

OnActionReceived() 함수는 매 스텝(Step)마다 정책(Policy)으로부터 결정된 행동에 대해 에이전트가 수행해야 할 일들을 정의하고 있다. 허밍버드에이전트 클래스에서 행동을 결정하는 OnActionReceived() 함수는 아카데미 클래스에 정의된 FixedUpdate() 함수가 호출될 때 함께 실행되며, 인자로 전달되는 vectorAction은 브레인이 에이전트로 전달하는 값이다. 브레인은 SpaceSize을 통해 몇 개의 변수가 에이전트로 전달될지 정의한다. 브레인에서 전달받은 vectorAction으로 에이전트가 (X, Y, Z) 축으로 얼마나 이동할지(VectorAction[0], VectorAction[1], VectorAction[2]) 혹은 (X, Y) 축으로 얼마나 회전할지(VectorAction[3], VectorAction[4]) 결정하게 된다.

```
public override void OnActionReceived(float[] vectorAction)
{
    if (isFrozen) return;
    Vector3 move = new Vector3(vectorAction[0], vectorAction[1], vectorAction[2]);
    // add force
    rigidbody.AddForce(move * moveForce);
    (필드) Rigidbody HummingbirdAgent.rigidbody
    Vector3 rotationVector = transform.rotation.eulerAngles;
    // calculate pitch and yaw
    float pitchChange = vectorAction[3];
    float yawchange = vectorAction[4];
    // smooth rotation
    smoothPitchChange = Mathf.MoveTowards(
        smoothPitchChange, pitchChange, 2f * Time.fixedDeltaTime);
    smoothYawChange = Mathf.MoveTowards(
        smoothYawChange, yawchange, 2f * Time.fixedDeltaTime);
    // new pitch and new
    // clamp pitch to avoid flipping
    float pitch = rotationVector.x + smoothPitchChange * Time.fixedDeltaTime * pitchSpeed;
    if (pitch > 180f) pitch -= 360f;
    pitch = Mathf.Clamp(pitch, -MaxPitchAngle, +MaxPitchAngle);

    float yaw = rotationVector.y + smoothYawChange * Time.fixedDeltaTime * yawSpeed;

    // apply rotation
    transform.rotation = Quaternion.Euler(pitch, yaw, 0f);

}
```

<그림-7>

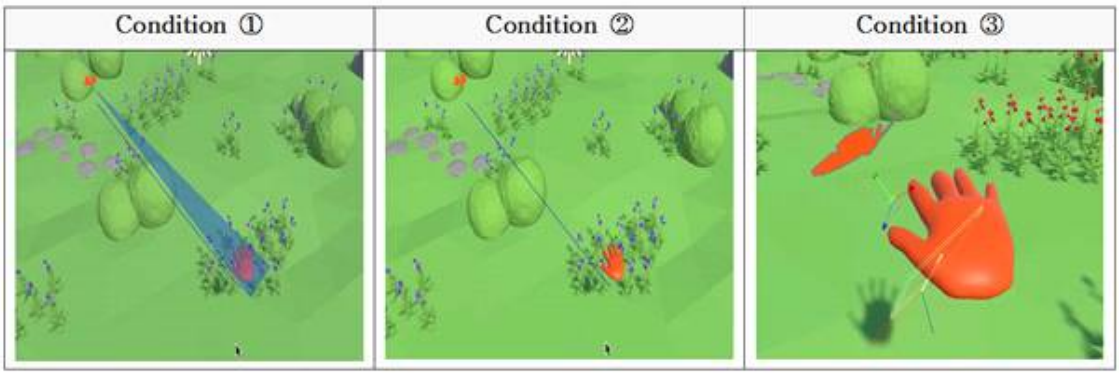
Decisions Requester Class는 에이전트가 어떻게 행동해야 할지 정책에 결정을 요청하는 컴포넌트다. 정책은 에이전트가 주변 환경정보를 수집하고 관찰한 정보를 토대로 학습된 것을 의미한다. 이 컴포넌트는 지속적으로 결정을 내려 받아 행동해야 하는 학습 패턴의 경우 사용된다. “관찰 - 결정 - 행동 -보상주기”의 한 반복을 시작한다. 브레인은 에이전트의 CollectObservations() 메소드를 호출하고 결정을 내리고 OnActionReceived()메소드 를 호출하여 이를 리턴(Return)한다. 브레인은 다른 반복을 시작하기 전에 에이전트가 다음 결정을 요청할 때까지 기다린다.

<그림-8>

사용자가 등장할 경우 사용자의 손을 따라가기 위한 첫 번째 조건은 나비가 사용자의 손을 향하고 있는 것이다. 나비의 시야각을 기준으로 사용자의 손이 15도 이하 내에 존재할 경우 좋은 행동이라 판단하여 보상을 주고, 사용자의 손이 20도 이

상 벗어나 있을 경우 나쁜 행동이라 판단하여 처벌을 준다.

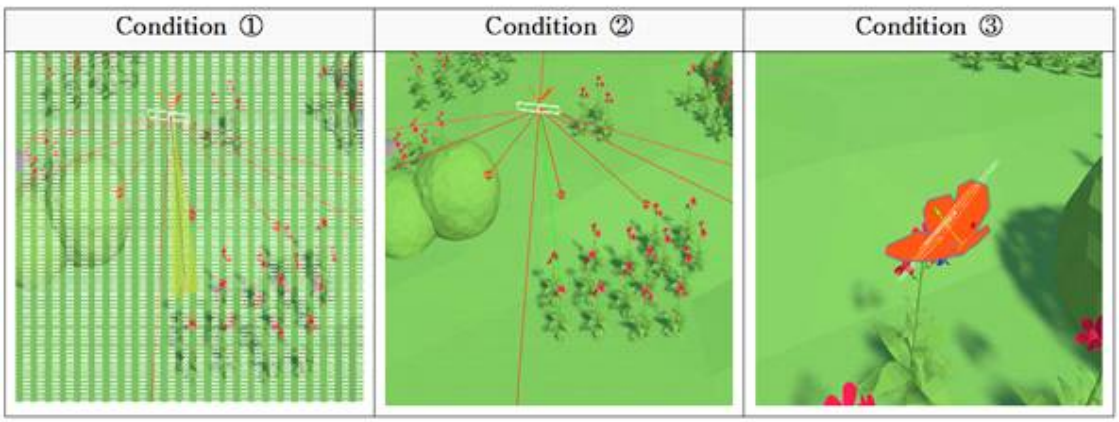
두 번째 조건은 나비가 사용자의 손을 향해 가고 있는 것으로, 사용자와 나비 간의 거리가 짧을수록 좋은 행동이라 판단, 보상을 주었다. 세 번째 조건은 나비가 사용자의 손에 머무르게 하는 것으로, 나비가 사용자의 손에 남아있으면 보상을 주었고, 손에서 벗어나면 처벌을 주었다.



<표-1>

사용자가 없을 경우 나비가 꽃에 다가가서 꿀을 먹는 첫 번째 조건은 나비가 꽃을 향하고 있는 것이다. 나비의 시야각을 기준으로 꽃이 15도 이하 내에 존재할 경우 좋은 행동이라 판단하여 보상을 주고, 꽃이 20도 이상 벗어나 있을 경우 나쁜 행동이라 판단하여 처벌을 준다.

두 번째 조건은 나비가 꽃을 향해 가고 있는 것으로, 꽃과 나비 간의 거리가 짧을수록 좋은 행동이라 판단, 보상을 주었다. 세 번째 조건은 나비가 꽃에 머무르게 하는 것으로, 나비가 꽃에 남아있으면 보상을 주었고, 꽃에서 벗어나면 처벌을 주었다.



<표-2>

Ⅲ. 결론

본 프로젝트에서는 행동보다 그 행동을 하는 이유에 주목하여 어떤 행동을 하게끔 욕구적인 관점인 보상과 처벌을 주어 인공 생명체를 구현하려 했다. Floating Island에 상주하는 나비는 사용자가 없을 때는 꽃에 다가가 꿀을 먹거나 이리저리 날아다니게 된다. 만약 사용자가 가상 공간 내에 등장하면 사용자에게 접근하여 사용자의 손을 따라가게 된다. 강화학습에는 어떤 것들을 학습시킬지에 대한 목적이 필요하며, 에이전트가 스스로 이를 선택하는 것이 아닌, 개발자에 의도에 따라 이루어지기 때문에 사람이 직접 코드를 작성한 대로 에이전트가 제어되는 휴리스틱(Heuristic) 모드와 크게 달라 보이지 않았다. 인공지능 생명체에게 어떤 행동을 학습 시킨다는 것은 단순하게 인간이 인공지능 생명체에게 어떤 것들을 학습시킬지에 대한 목적을 정해주는 것이 아닌, 인공지능 생명체로 하여금 어떤 행동을 유발하는 욕구를 느끼게 하고, 인공지능 생명체가 스스로 적절한 목적을 선택하고 그에 따른 행동을 선택할 수 있어야 할 것이다.

Reference

최민성, “문화콘텐츠 주제학 시론 나비 상징을 중심으로”, 인문콘텐츠학회
인공지능신문 “강화학습은 무엇이며, 어떻게 설정하고 해결해야 하는가?”
교양인 “[스키너의 행동심리학] 책 소개”
홍성찬, “강화학습을 이용한 격투게임 인공지능 설계 및 구현”, 학위논문 사항
위키북스 “텐서플로와 유니티 ML-Agents로 배우는 강화학습”