**2018180020 박재우(클라이언트) 17주차 기록**

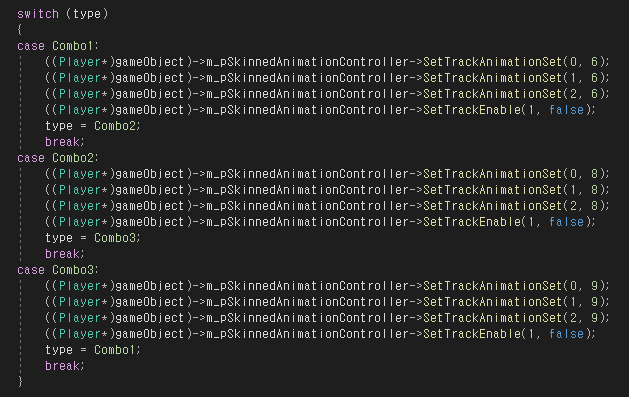
**걷기와 달리기**

왼쪽 Shift를 눌렀을 때 걷고 땠을 때 뛰도록 만든다.

걸을 때와 뛸 때의 속도는 다르기 때문에 각 상황에 맞게 속도를 조절해 줄 필요가 있다. Movement 컴포넌트는 윈도우 메세지를 받아오지 않는다. 따라서 WM\_KEYDOWN 또는 WM\_KEYUP을 이용해 필요 할 때 딱 한번 속도를 바꿔 줄 수 없다. 외부의 다른 함수에서 윈도우 메세지를 받아오는 함수가 있긴 하지만 속도를 조절하는 것 역시 움직임에 해당하기 때문에 컴포넌트 안에서 처리해 주고 싶다. 그렇게 되면 업데이트마다 속도를 설정하게 될 것이다. 함수로 호출하긴 하지만 단순한 대입 연산이기 때문에 성능상 큰 무리가 없을 것이라 생각된다. Shift를 눌렀을 때 속도를 적절히 바꿔주고 대쉬 할 때도 속도를 잠깐 높였다가 원래대로 낮춰 대쉬가 정상 작동하도록 했다.

**Attack Componenet 콤보기능**

2단 콤보는 만들었지만 우리 게임은 3단 콤보 공격이다.

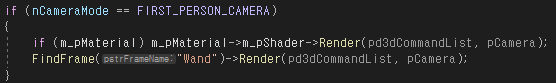


현재 몇 콤보인지 알기 위한 enum을 만들었고 해당 콤보에 맞는 애니메이션이 재생될 수 있도록 만들었다. 애니메이션을 재생시키고 타입을 무조건 다음 콤보로 바꿔준다.

일정 시간이 지나면 콤보가 끊기고 첫 콤보부터 시작해야 한다. update함수에서 AttackTimeLeft의 값을 0이 될 때까지 빼 주는 부분이 있는데 else문을 이용해 0이 되면 콤보를 다시 Combo1로 바꿔주었다.

공격이 끝나고 0.5초 안에 다시 공격하면 다음 콤보로 이어지는 기능이 완성되었다.

**1인칭 시점**

****

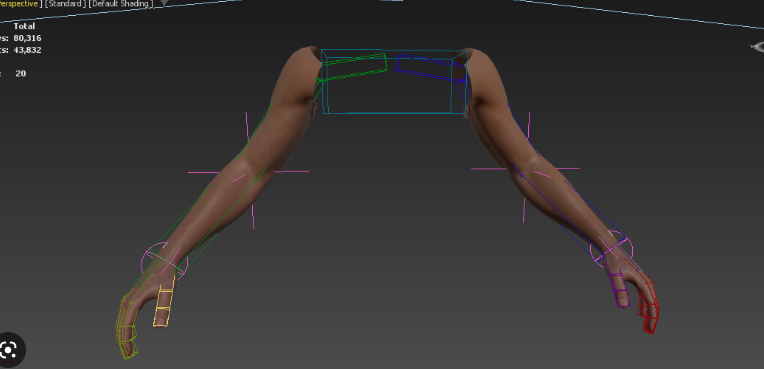
1인칭 시점을 구현하기 위해 카메라 모드가 1인칭일 때만 Wand라는 이름의 Frame만 렌더링 하도록 했다. Wand는 무기의 Mesh를 가지고 있고 가장 마지막 노드이기 때문에 무기만 렌더링 된다.



달리는 동안 무기가 흔들리는 모습이 보이고 공격을 할 때 휘두르는 모습도 볼 수 있다.



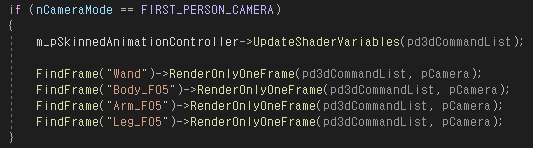
하지만 가만히 있을 때, Idle 모션이 재생될 때는 애니메이션 자체가 무기는 뒤쪽에 위치하도록 되어있어 시점을 바닥을 보도록 해야 조금 보이게 된다.



많은 게임의 1인칭 시점은 무기만 나오거나 또는 무기와 팔만 나오는 게임이 매우 많다. 그런 게임들의 경우는 위의 사진과 같이 대부분 1인칭용의 무기와 팔의 Mesh가 있다.

1인칭 시점에 대해 무기만 나오는 것이 좋은 지, 팔과 무기만 나오는 것이 좋은 지, 머리를 제외한 모든 부위가 나오는 것이 좋은 지 팀원들의 의견을 물어봤다. 몸통 전체가 나오는 것이 보기 좋은 것 같다고 해서 머리를 제외하고 렌더링 하기로 했다.

몸통이나, 팔 같은 부분은 일반 Render 함수로 렌더링을 하려고 해도 Sibling 노드까지 전부 렌더링 하게 되어있어 결국 전체가 렌더링 되게 된다. 그래서 Sibling 노드나 Child로 넘어가는 부분을 빼고 렌더링하는 RenderOnlyOneFrame 함수를 만들었다.



1인칭 시점일 때 필요한 부분만 렌더링 한다. 그 전에 Animation Controller의 Update Shader Variables 함수를 호출한다. 평소처럼 렌더링 하면 Render 함수에서 호출이 된다. 하지만 Animation Controller는 오브젝트의 최 상단에 위치하기 때문에 밑의 특정 노드를 렌더링 하려하면 그 노드엔 Animation Controller가 없기 때문에 애니메이션 데이터가 갱신되지 않는다. 따라서 미리 함수를 호출해준다.

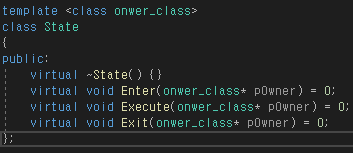


1인칭 화면이 완성되었다. 위 사진은 뛰어다니는 애니메이션이 재생되는 모습을 찍은 사진이다.

**Finite State Machine (FSM)**

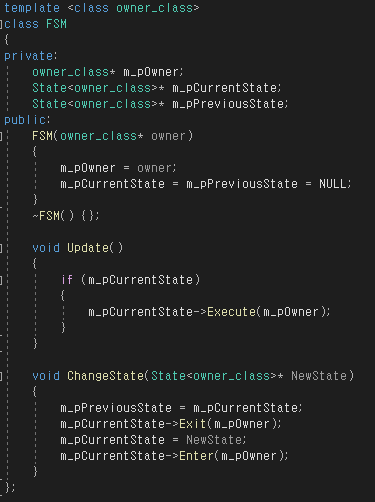
몬스터들을 움직일 상태기계를 만든다. 몬스터들은 종족별로 있지만 행동 패턴은 다르지 않고 몬스터가 원거리형 인지, 근거리형 인지, 돌진형 인지, 보스 인지에 따라서 행동 패턴이 달라진다. 따라서 상태기계는 4가지로 만든다.

우선 상태 기계 클래스를 만든다. 상태 기계는 상태라는 정보를 가지고 있어야 한다. 지금 생각하는 상태로는 적을 발견했을 때의 추적, 평소 상태의 배회 2가지 정도만 생각하고 있고 이 2가지를 만드는 것이 중간 발표까지의 목표이다. 상태가 2개뿐이라 현재 상태만 알고 있어도 좋지만 이후 만일을 위해 이전 상태의 정보도 저장하도록 한다. 상태도 클래스로 표현한다. 그러기 위한 State 클래스를 만들었다. State클래스는 3개의 함수가 있다. 처음 상태가 바뀔 때 불릴 Enter와 계속 실행될 Execute, 상태가 종료될 때의 Exit가 있다.



State 클래스 자체는 사용할 일이 없다. 상태마다 행동이 완전히 다르기 때문에 무조건 그 상태에 맞는 클래스가 이 State를 상속받아 사용해야 한다. 그래서 State는 순수 가상함수로 구성하여 추상 클래스로 만들었다. State의 Execute함수에서 각 상황에 맞는 행동을 할 것이다. 그 행동을 하기위해 행동을 할 주체가 필요해서 Owner가 될 객체의 포인터를 받아오도록 했다. Owner는 컴포넌트가 될 예정이고 컴포넌트 안에서 몇 가지 행동에 대한 함수를 만들고 그 함수를 State가 호출하는 방식으로 동작하게 할 것이다. 하지만 어떤 컴포넌트가 이 State를 소유할지 알 수 없어 template을 사용했다. State를 만들었으니 상태기계도 만들도록 한다. 앞서 설명했듯이 현재 상태와 이전 상태를 저장해야 한다. 그리고 현재 상태를 실행할 때 인자로 넘겨줄 Owner 역시 알고 있어야 한다.

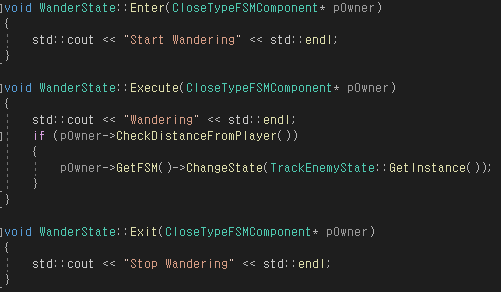
상태기계가 해줄 일은 상태 변환, 현재 상태 실행 두가지를 할 수 있으면 된다. 상태를 변환해 줄 때는 이전 상태를 현재 상태로 바꿔주고 현재 상태의 Exit 함수를 불러 상태를 종료해준다. 그 후 현재 상태를 다음 새로운 상태로 바꿔주고 바꾼 상태의 Enter함수를 불러주면 상태 변환이 끝난다. 상태 실행은 그냥 상태의 Execute 함수를 호출해 주기만 하면 되는 간단한 작업이다.



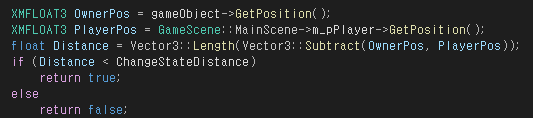
이렇게 해서 FSM클래스를 만들었고 이 FSM을 소유할 Owner역시 컴포넌트로 어떤 컴포넌트인지 알 수 없기 때문에 template을 사용했다. State는 게임이 진행되는 동안 매우 많이 변경될 것이다. 변경될 때 마다 State를 생성하고 제거하면 그만큼 부담이 될 것이기 때문에 싱글톤 패턴을 이용해 각 State의 인스턴스를 하나 생성해 하나의 인스턴스를 돌려가며 사용하는 방법으로 한다.

WanderState와 TrackEnemyState 2개의 State 클래스를 만들었다. WanderState는 평소의 상태로 주변을 배회한다. TrackEnemyState는 적을 발견했을 때 적을 쫓아가는 상태이다.

State의 순수 가상함수 Enter Execute Exit 함수를 정의한다. 지금은 간단하게 콘솔창에 텍스트만 띄우도록 한다.

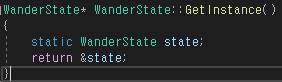


Execute함수에서는 현재 이 상태를 가진 몬스터와 플레이어와의 거리를 계산해 상태를 바꿀지를 판단한다.



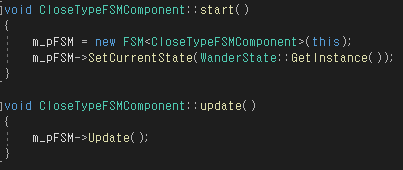
CheckDistacneFromPlayer 함수이다. 이 함수는 CloseTypeFSMComponent라는 컴포넌트 클래스의 함수이다. 이 컴포넌트를 소유한 오브젝트의 위치와 Scene에서 받아온 플레이어의 위치를 계산해 거리를 얻고 일정 거리에 따라 bool값을 반환해준다. 기준이 되는 값은 현재 20.0 이다.

이렇게 계산한 bool값으로 FSM의 상태를 변경해준다.

변경해줄 때 사용한 GetInstance 함수이다.

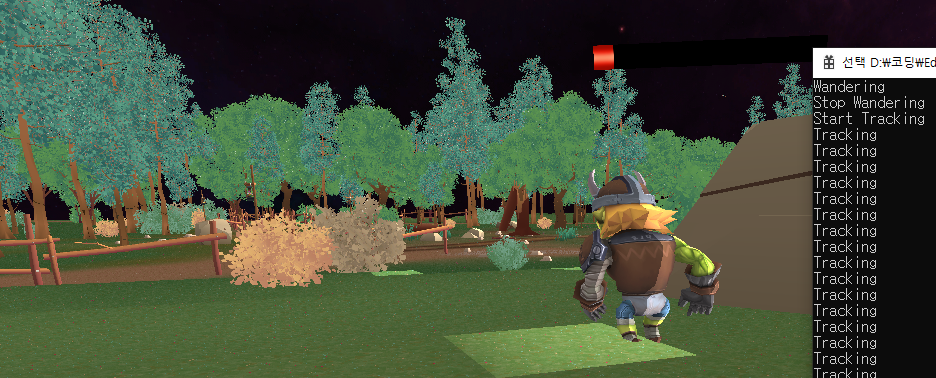
GetInstance 함수를 이용해 상태를 변경한다.

TrackEnemyState도 Execute함수에서의 조건이 반대인 것만 빼면 WanderState와 동일하다.



CloseTypeFSMComponent의 start 함수에서 FSM을 생성하고 현재의 상태를 Wander State로 설정한다. update함수에서는 FSM의 update 함수를 불러준다.

그럼 이제 Goblin에 CloseTypeFSMComponent를 추가하여 테스트를 해보자



처음엔 고블린과 거리가 가까워 WanderState에서 바로 TrackEnemyState로 바뀌는 모습을 확인할 수 있다. 바뀔 때도 Stop Wandering 후 Start Tracking을 하는 것까지 확인되었다.



충분히 멀어지면 Track을 중단하고 다시 Wander로 바뀐다. 다시 가까워지면 Wander에서 Track으로 바뀐다.

조건에 따라 상태가 변경되고 FSM의 update도 잘 작동하는 것을 확인했다. 다음주에 몬스터의 애니메이션을 상태에 따라 바꾸고 움직이는 것을 만들어 보겠다.