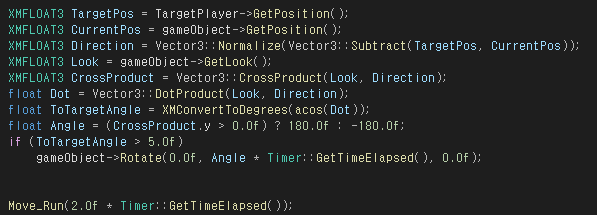
**2018180020 박재우(클라이언트) 18주차 기록**

**FSM으로 몬스터 움직이기**

지난주에 만든 FSM으로 실제로 몬스터를 움직여 보도록 한다.

TrackState일 때 몬스터는 플레이어를 따라가야 한다.

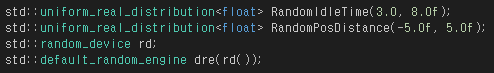


TargetPlayer는 몬스터가 따라갈 대상으로 CheckDistanceFromPlayer 함수에서 정해지도록 했다. 몬스터의 현재 위치와 대상 플레이어의 위치를 이용해 방향 벡터를 구하고 외적과 내적 계산으로 각도를 구하고 어느 방향으로 회전할 지 계산했다. 몬스터는 1초에 180도 회전하도록 하기 위해 Angle을 180단위로 맞췄다. ToTargetAngle은 타겟으로 몇 도를 회전시켜야 하는지를 계산해 놓은 변수로 이 값이 5도보다 클 때만 회전하도록 했다. 그러지 않으면 몬스터가 회전할 때 딱 각도가 0이 되도록 회전하지 않기 때문에 플레이어를 바라보며 따라올 때 좌우로 떨리게 된다. 그래서 회전 각이 5도를 넘을 때만 회전하도록 했다.

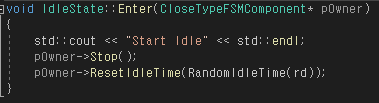
이제 플레이어를 보며 따라올 수 있게 됐다. 다음은 일정 거리가 되었을 때 공격을 하도록 한다. 공격하는 기능은 플레이어의 공격과 똑같이 공격하면 공격 모션이 재생되는 같은 기능이라고 볼 수 있다. 따라서 몬스터에도 Attack컴포넌트를 추가해준다. 하지만 Attack컴포넌트는 플레이어에 맞춰져 있고 update함수에도 플레이어가 입력한 키에 맞게 동작한다. 키입력에 반응하는 오브젝트는 플레이어 밖에 없기 때문에 몬스터가 키입력에 대한 동작을 하지 않도록 해야 한다. 방법은 간단하게 dynamic\_cast를 이용해서 해결했다. update함수에서 Attack을 부르는 곳 이전에 dynamic\_cast로 Player\*로 형변환을 시도한다. 만약 Play 클래스가 아니라면 NULL이 반환될 것이기 때문에 if문으로 NULL이 아닐 때만 키입력이 되도록 하였다.

WanderState를 작동되게 하려한다. 배회하는 도중에는 딱히 목적지는 없고 여기저기 앞으로 움직였다가 멈췄다가 반복한다. 그러기 위해 IdleState를 하나 더 만든다.

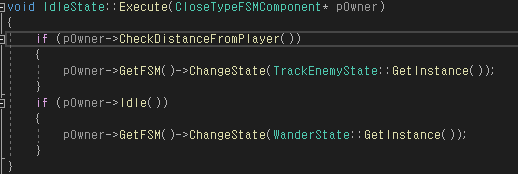
IdleState는 시작할 때 Idle애니메이션이 재생되도록 한다. 그 후 랜덤한 실수를 생성해 그만큼 제자리에서 머문다.



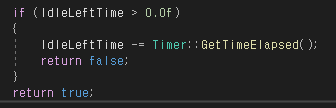
랜덤 값은 위의 값들을 이용했고 다음과 같이 ResetIdelTime의 인자로 사용한다.



새롭게 정한 IdleTime동안 아무행동도 하지 않는다. Execute에서 IdleTime을 체크하고 플레이어의 위치를 체크해 다음 State로 변경될 수 있도록 했다.

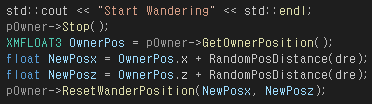


Idle 함수는 다음과 같이 만들었다.



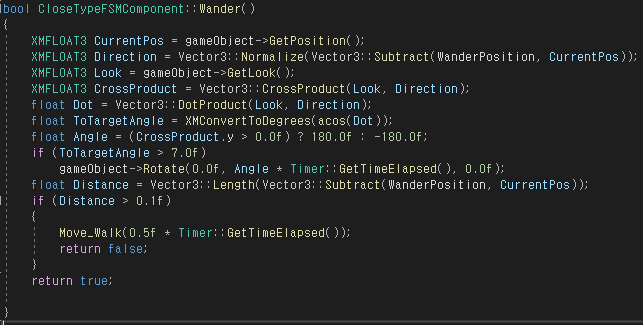
IdleTime을 경과시간만큼 빼주며 false를 반환한다. 이 때 IdleTime이 0 이하가 된다면 true를 반환하게 되고 Execute함수에서 Idle함수의 반환 값에 의해 WanderState로 변환되게 될 것이다.

WanderStae 또한 IdleState와 비슷하다.



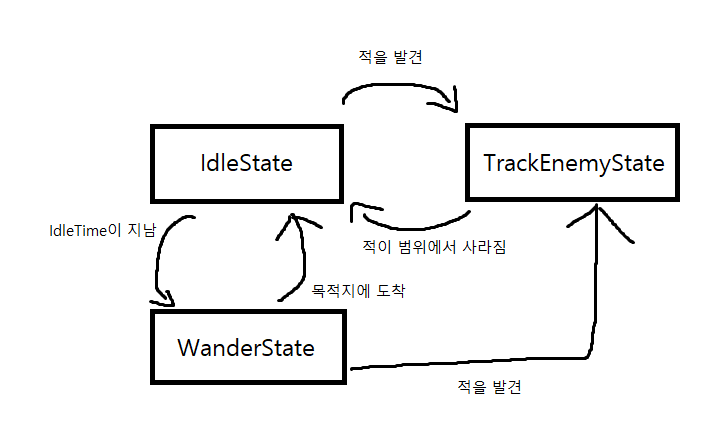
시작할 때 Enter함수에서 랜덤한 위치를 생성한다. ResetWanderPosition 함수에서는 받아온 X, Y 좌표값을 이용해 MainScene의 Terrain을 가져와 높이를 구해 Y좌표를 생성한다. X와 Z는 월드상의 임의의 위치로 하면 이동거리가 매우 길어지게 된다. 따라서 Owner의 현재 위치를 받아 그 곳으로부터 랜덤한 값만큼 떨어진 곳으로 정했다. 그러면 멀리 이동하지 않고 그 주위를 배회할 것이다.

그 후 Idle함수와 비슷하게 Execute함수에서 if조건문 안에서 Wander함수를 호출한다.



Wander함수도 bool값을 반환해주는데 Wander의 true 반환 조건은 해당 위치에 도달했을 경우에 true를 반환하고 그 외에는 false를 반환하며 해당위치를 향해 움직인다. 목적지를 향해 움직이는 로직은 TrackState와 동일하며 목적지가 플레이어의 위치에서 새로 생성한 위치로 바뀌었다. 도착했다는 것의 판단은 목적지와의 거리를 이용했다. Move함수는 일정 거리를 움직이는 함수이다. 이 함수로는 이동을 하면 절대로 목적지와 정확히 같은 위치로 갈 수 없다. 정확히 같은 위치로 갈 확률은 매우 적고 반드시 오차가 생기기 마련이다. 이 상황에서 위치를 비교하게 되면 계속해서 그 위치로 움직이려고 할 것이기 때문에 근처에 가기만 해도 도착했다고 판단하도록 거리를 구해서 값을 비교했다. 지금은 거리가 0.1 이하이면 도착했다고 판단했다.

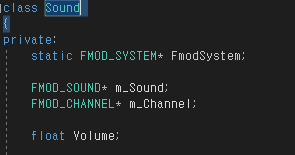
이렇게 FSM이 어느정도 완성되었다. 다음은 지금까지 만든 FSM의 요약이다.



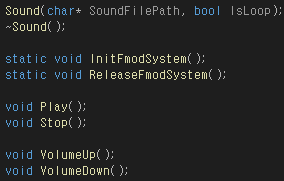
**사운드 재생**

간단하게 배경음악 정도만 재생되도록 만든다. 청각을 빼앗는 기능이 있는 우리 게임에서 사운드는 꽤 중요한 역할을 한다. 먼저 사운드가 재생되도록 한 후 여유가 된다면 난청 효과까지 만들어보도록 한다.

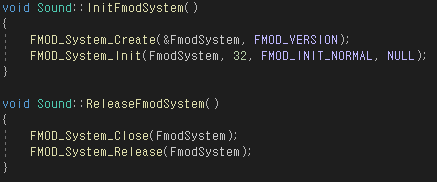
사운드 재생을 위한 Sound 클래스를 만든다.



Sound 클래스의 멤버변수들을 먼저 선언해준다. FMOD 사운드와 채널을 가지고 볼륨 값을 가진다. Fmod로 사운드를 재생하기 위해서는 FMOD\_SYSTEM이 필요하다. 사실 이 System은 Sound 클래스가 가질 필요는 없다. 딱 한번만 초기화 해두기만 하면 이 System을 이용해 사운드를 재생할 수 있다. 때문에 클래스 외부에서, 예를 들어 게임이 시작될 때 딱 한번 초기화 해주고 끝날 때 해제해줘도 상관없다. 그럼에도 static으로 클래스 내부에 만들어준 이유는 앞서 말했듯이 System을 통해 사운드를 재생하기 때문에 사운드를 재생할 때 마다 접근할 수 있어야 한다. 그래서 접근하기 편하게 클래스 내부에 static으로 만들었다.

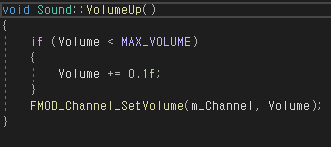


멤버 함수들이다. 생성자는 재생할 사운드 파일의 위치와 반복 재생 여부에 대한 값을 받아온다. FMOD\_System\_CreateSound 함수를 이용해 파일을 m\_Sound에 받아온다. 이때 IsLoop값에 따라 재생 모드를 다르게 해준다. 소멸자에선 FMOD\_Sound\_Release함수로 사운드를 지워준다.



시스템 초기화 함수와 해제 함수이다. 둘 다 static으로 선언 되어있으며 프로그램 최초에 InitFmodSystem()을 호출하고 프로그램을 종료하기 직전에 ReleaseFmodSystem()을 호출한다. 초기화 할 때 최대 채널 개수를 지정해주는데 사운드가 몇 개나 들어갈 지 몰라서 32개로 설정해 두었다.

Play와 Stop은 FMOD\_System\_PlaySound 함수와 FMOD\_Channel\_Stop 함수를 호출하는 것이 끝이다. 시작할 때 System을 이용해 사운드를 재생하고 멈추는 것은 채널만 조작하면 된다. 채널을 조작해 일시 정지 후 다시 재생도 가능하다.



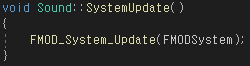
볼륨을 높이는 함수다. 볼륨도 채널을 조작해서 변경이 가능하다. 볼륨이 최댓값 보다 작으면 0.1씩 더하고 더한 볼륨 값을 FMOD\_Channel\_SetVolume 함수에 넘겨준다. 볼륨을 낮추는 함수도 똑같이 작동하고 값을 빼서 볼륨을 낮춘다.

이렇게 Sound클래스의 기본적인 기능은 완성을 했고 재생을 해보도록 한다. Init함수를 FrameWork의 Oncreate함수에서 호출하고 Release함수는 OnDestroy에서 호출하도록 했다.



BuildObject가 끝나는 시점에서 해당 코드를 실행했다. 사운드가 아주 잘 재생된다. 처음 소리가 너무 커서 Volume을 기본 0.3으로 맞추고 시작하도록 수정했다.

찾아보니 System을 주기적으로 업데이트를 해줘야 하는 모양이다. 그래서 Update함수도 추가로 만들어줬다.

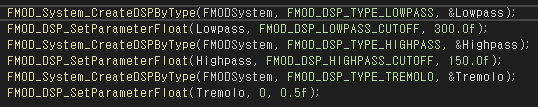


이 함수 역시 static이고 FrameWork의 FrameAdvance 함수에서 호출하여 매 플레임마다 호출되도록 했다. 시스템 변수명을 FmodSystem보다는 FMODSystem으로 대문자를 사용하는 것이 적절할 것 같아 변수명을 변경했다.

**난청 효과**

난청 효과를 위한 기능도 FMOD에서 제공한다. 저주파 필터인 Lawpass 필터와 떨림 효과를 주는 Tremolo 모듈이 있다. 이 외에도 Echo 효과나 Highpass등 여러 효과를 줄 수 있다. 나는 Lowpass와 Highpass 그리고 Tremolo를 이용해 난청효과를 표현했다.

Lowpass 일정 주파수 밑으로만 들리도록 잘라주는 역할이고 Highpass는 일정 주파수 위로만 들리도록 잘라주는 역할이다. 둘을 잘 조합하면 특정 영역의 주파수만 들리도록 할 수 있다. 너무 낮은 주파수만 들리면 음악의 베이스같이 매우 낮은 음만 둥둥 들리게 된다. 이것도 좋긴 하지만 음악이 아예 들리지 않게 하는 것은 게임의 재미를 떨어뜨린다고 생각한다. 그래서 Highpass와 Lowpass로 최저/최대 주파수를 조절해서 먹먹한 소리를 표현하고자 했다. 여기서도 충분히 난청 같지만 Tremolo를 이용해 소리의 떨림을 주었다.



생성자에서 각 효과들을 만들고 수치를 정해줬다. 이 효과들을 DSP라고 부른다. Lowpass의 값과 Highpass의 값은 주파수를 의미한다. 300과 150 사이의 주파수만 들리도록 값을 설정해 줬다. Tremolo값은 0.5로 매우 낮게 설정해주었다. 빠르게 떨리지 않고 매우 느리게 떨리는 효과를 주어 뭔가 어지러운 듯한 느낌도 줄 수 있었다.