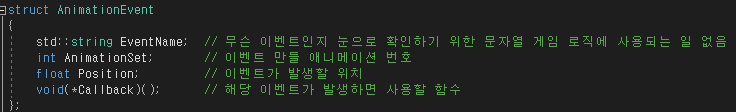
**2018180020 박재우(클라이언트) 27주차 기록**

**Animation Event**

지금의 공격은 공격 함수가 불리면 애니메이션이 재생되고 불린 순간 공격 판정이 이루어진다. 그래서 공격이 실재로 플레이어에게 닿기 전에 피격되어 hp가 깎인다. 공격을 판정하는 타이밍을 조금 늦춰 무기를 휘둘렀을 때 hp가 깎일 수 있도록 수정하도록 한다.

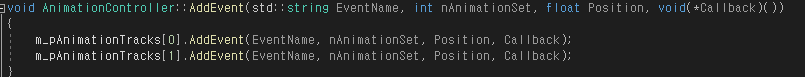
그러기 위해서 애니메이션이 움직일 때 적절한 타이밍에 공격 함수를 실행시켜야 한다. 애니메이션에 대한 정보를 나타내는 Animation Set 자체는 로드한 메쉬에 포함되어 있고 이 데이터는 같은 메쉬를 사용하는 모든 객체가 공유하기 때문에 AnimationSet에서 함수를 실행하면 어느 한 객체가 설정한 함수를 호출하게 될 때 같은 메쉬를 사용하는 모든 객체가 그 함수를 실행하게 될 것이다. 그러면 각 객체가 독립적으로 가지고 있는 Animation Controller나 Animation Track에 해당 기능을 만드는 것이 좋을 것이다. 현재는 Animation Controller 한 개가 Animation Track 3개를 가지고 있는 구조이다. 이 3개의 Track중 실제로 애니메이션을 재생하는 Track은 0번과 1번 두개이고 서로의 애니메이션이 달라도 실행되면 설정한 이벤트가 발생해야 할 것이다. 예를 들어 공격과 걷는 애니메이션을 블랜딩할 때 우연히 발자국 사운드 이벤트와 공격 판정 이벤트가 동시에 일어나면 발자국 소리도 나면서 공격 판정도 일어나야 한다. 그러니 이벤트를 발생시키는 위치는 각자의 애니메이션을 따로 재생하는 Animation Track이 적절 할 듯하다.

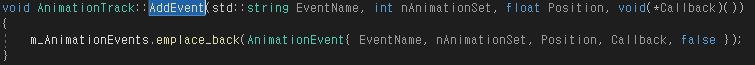
애니메이션 이벤트를 발생시키기 위해선 Track의 m\_fPosition값을 이용해야 한다. m\_fPosition이 현재 재생중인 애니메이션의 재생 위치이기 때문에 이 값을 비교해 Event를 발생시키게 한다. UpdatePosition 함수가 m\_fPosition 값을 매 프레임마다 바꿔주는데 이곳에서 Position값을 비교해 함수를 호출하게 만들 수 있지만 지금 만들려고 하는 기능은 공격뿐 아니라 애니메이션에 맞춰 소리를 재생한다 거나 하는 다른 많은 곳에서 사용될 수 있는 기능이다. 또한 애니메이션은 종류가 많고 또 모델마다도 애니메이션이 다 다르지만 사용하는 Animation Track은 모두 같은 class를 이용하기 때문에 모든 경우의 수를 UpdatePositon 함수에 구현하는 것은 매우 힘들고 코드도 길어지며 난해해질 우려가 있다. 따라서 Event 자체를 하나의 객체로 관리해 UpdatePosition에서는 Event의 Position 값만 비교하도록 한다.



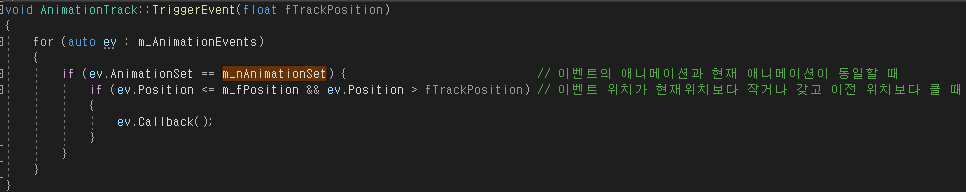
AnimationEvent 구조체이다. 이벤트의 이름은 눈으로 무슨 이벤트가 추가된 것인지 확인해 원활한 개발을 하기 위함이다.

AnimationSet은 이 이벤트가 일어날 애니메이션 번호이다. AnimationTrack에는 해당 모델이 갖고 있는 애니메이션의 배열을 가지고 있고 재생중인 애니메이션은 상황마다 다르기 때문에 AnimationEvent 자체가 자신이 이벤트를 일으키는 애니메이션의 정보를 가지고 있도록 했다. Position은 이벤트가 발생할 재생 위치이다. Callback은 이벤트가 발생했을 때 호출할 함수의 포인터이다. 이벤트는 여러 개가 만들어질 수 있고 이벤트마다 실행할 함수가 다를 것이기 때문에 함수 포인터를 이용해 콜백함수로 구현한다.

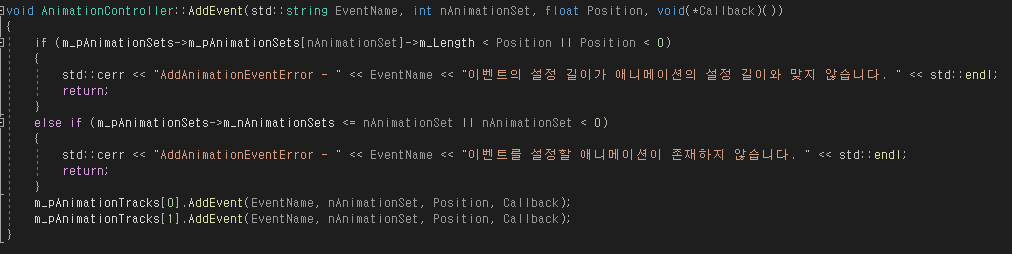




Animation Controller에서 AddEvent를 이용해 AnimationTrack의 AddEvent를 호출한다. AnimationTrack의 AddEvent는 m\_AnimationEvents라는 vector 컨테이너에 추가해준다.



Animation Track에 TriggerEvent 함수를 추가했다. 인자로 받아오는 fTrackPosition은 현재 애니메이션에서 프레임 시간을 더하기 이전의 위치이다. fTrackPosition과 m\_fPosition 사이에 이벤트의 Position이 위치한다면 이는 이번 프레임에 이 이벤트 위치를 넘겼다는 뜻이다. 이 때 등록해둔 콜백함수를 호출하면 원하는 타이밍에 원하는 기능을 작동시킬 수 있고 루프당 딱 한번만 호출될 것이다.



이벤트를 추가하는 곳에 몇 가지 예외를 설정했다. Position이 음수이거나 AnimationSet의 m\_Length보다 크다면 정상적인 애니메이션 재생 범위가 아닌 것이므로 로그를 출력하고 함수를 빠져나온다. 또 인자로 받아오는 nAnimtionSet이 음수이거나 AniamtionSets의 m\_nAnimationSets(애니메이션 개수) 보다 크거나 같으면 해당하는 애니메이션이 존재하지 않으므로 로그를 출력하고 함수를 빠져나오게 했다.

AddEvent로 이벤트를 추가하려 했는데 콜백함수를 넣어주려고 하니 함수 포인터의 형식이 달라 인자로 넘겨줄 수 없었다. 함수 포인터를 지정해 주려고 해도 이 함수를 부르는 클래스는 너무 다양하기 때문에 하나로 지정할 수 없었다.

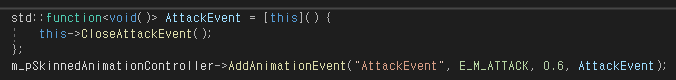
이 문제를 해결하기 위해 functional 라이브러리를 사용하기로 했다.





구조체의 콜백 함수 부분을 위와 같이 변경하였고 AddEvent도 똑같이 변경했다. 함수 이름은 더 직관적일 수 있도록 AddEvent에서 AddAnimationEvent로 변경하였다.

이벤트는 Goblin의 생성자에서 추가해 주었다.



CloseAttackEvent 함수는 여전히 Goblin::\* 이기 때문에 std::function<void()> 형태로 바꿔주기 위해 람다를 이용해 std::function<void()> 형태가 되도록 만들었다.



테스트해본 결과 위의 gif처럼 고블린이 왼쪽 손으로 공격했을 때 피격판정이 되는 것을 확인할 수 있다.