| **비단길(Silk Road)** | |
| --- | --- |
| **14주차** | **2024.03.23 (일) ~ 2024.03.30 (토)** |

**■ 다음 주 세부 계획**

**▶ [권순원]**

* 패킷을 클라이언트에 적용하는 인터페이스 함수 제작
* 패킷 직렬화 / 패킷 핸들러 클래스 제작

**▶ [박준영]**

* 중간발표 스테이지 지형지물 제작 및 오브젝트들 박스콜라이더 수정
* ai 개발

**▶ [엄장헌]**

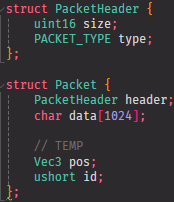
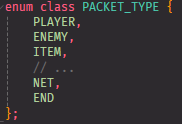
* 물리엔진 관련해서 속도와 마찰력 사이에 문제가 있는 점 수정
* 애니메이션 부품화 준비

**■ 작업 일지**

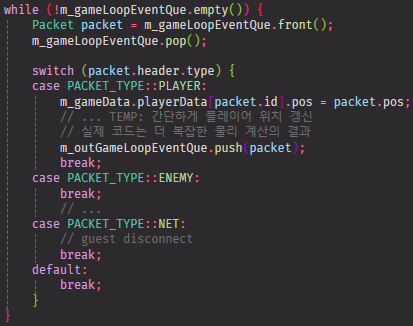
**▶ [권순원]**

이번 주는 원래 파티클 작업을 하기로 계획했지만 팀원들과 상의를 하고 나서 파티클 시스템 보다 서버 개선 작업이 더 우선이라고 판단되어 서버 작업을 위주로 진행했다.

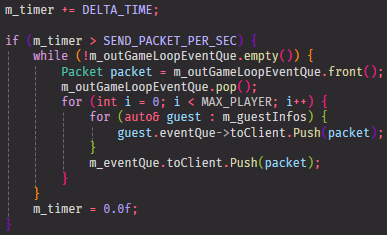
첫 번째로 Packet 구조를 개선 했다. 이전에 단순히 연결을 확인하기 위한 용도로 만들었던 Packet 구조를 추후에도 사용할 수 있도록 변경하였다.

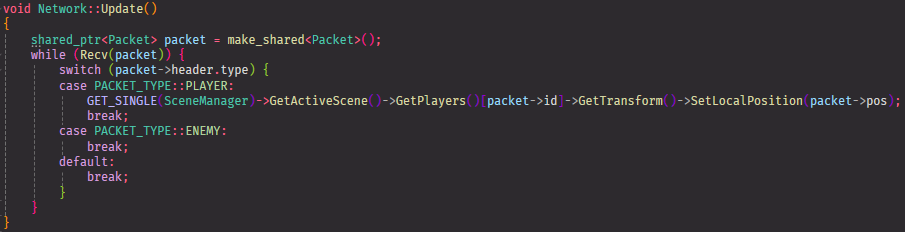


일단 패킷의 타입을 설정했다. 이는 서버나 클라이언트가 패킷을 전달 받았을 때 어떤 게임 객체에 적용하고 어떤 방법으로 패킷을 해석할지 알려줄 수 있는 타입으로 일단 PLAYER와 NPC를 통칭하는 ENEMY, ITEM, 초기설정이나 네트워크 설정에 관한 패킷을 전달할 때 사용할 NET 정도로 나누었다. 추후 인게임 컨텐츠가 더 추가됨에 따라 상황에 맞게 변경해서 사용할 것이다.  
 그리고 패킷의 전체 사이즈와 앞서 말한 타입을 담을 Header가 존재하고 이들을 활용하는 패킷이 있다. 패킷 구조체는 헤더와 char 타입 데이터를 보관하는데 패킷에 실을 데이터를 직렬화하여 바이트 형태로 주고 받기 위해 사용할 예정이다. id는 현재는 여전히 플레이어의 위치를 주고 받는 것이 목적이기에 간단하게 클라이언트마다 지정된 id를 저장하지만 추후에는 타겟으로 하는 오브젝트의 id를 저장할 예정이다.



패킷을 받아 해석하는 부분이다. m\_gameLoopEventQue는 서버가 각 클라이언트에게 받은 패킷을 한 곳에 담아 보관하기 위한 큐이다. 해당 큐는 오직 게임 로직을 처리하는 메인루프와 메인루프 중간에 실행되는 게임루프에만 존재, 즉 한 스레드에서만 사용될 자료구조이기 때문에 동기화 등의 처리가 필요없어 일반적인 큐로 생성하였다.  
 꺼내온 패킷은 타입을 확인해 switch문에서 각 패킷에 맞는 처리를 한다. 그 후 각 클라이언트들에게 알릴 내용이 있으면 m\_outGameLoopEventQue에 집어넣는다. 해당 자료구조 또한 위 큐처럼 메인루프 안에서만 사용될 예정이기에 일반적인 큐로 선언했다.  
 현재는 주석으로 처리를 해놓았지만 추후 물리엔진 작업과 인게임 작업이 어느정도 완료되면 계산 코드(함수)를 추가할 예정이다.

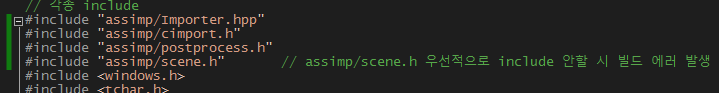
  
 게임루프가 끝나고 outQue가 비어있을 때까지 안에서 패킷을 꺼내 각 클라이언트에게 전송한다. 해당 작업은 전역으로 존재하는 DELTA\_TIME을 이용해 SEND\_PACKET\_PER\_SEC에 지정된 시간마다 패킷을 전송하게 된다. 현재는 1/60초로 설정되어 1초에 60번씩 통신을 시도한다.

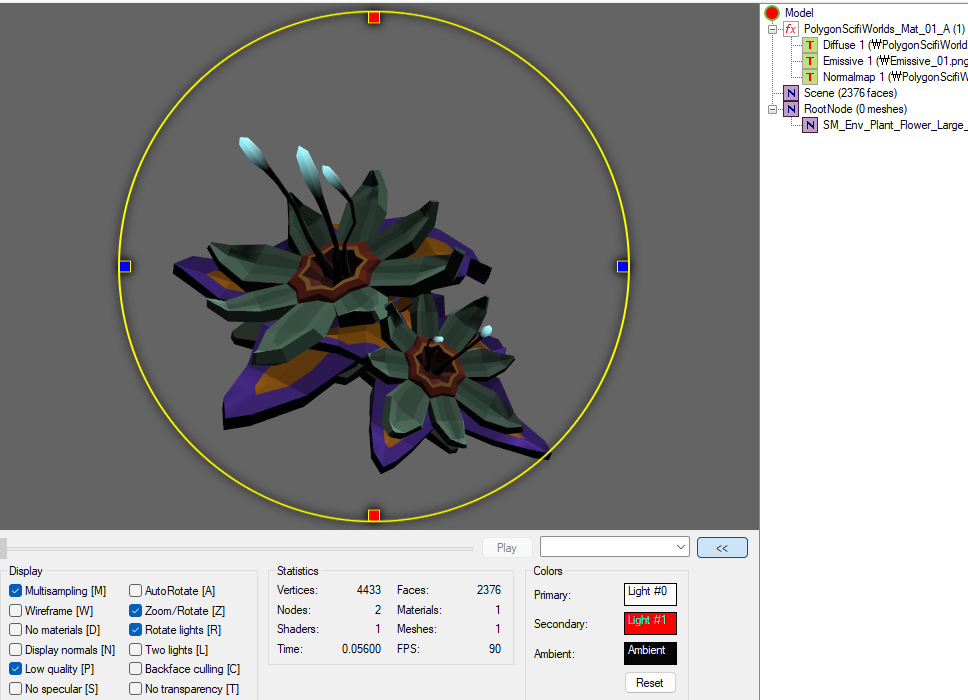


마지막으로 패킷구조가 바뀌었으니 클라이언트에서 업데이트되는 부분도 변경되었다. 서버에서와 비슷한 방식으로 타입을 확인하고 각 타입에 맞게 패킷의 내용을 게임에 적용하는 방식이다.

지금까지 작업을 서버에 관련되어서만 작업을 해놓았기 때문에 다음주는 변경된 서버, 패킷 기능을 기반으로 클라이언트에 적용하기 쉽도록 함수를 제작할 예정이다.

**▶ [박준영]**



  
현재 우리 프로젝트에서 fbxsdk를 통해 fbx모델을 로드하고 있는데 대부분 잘 작동하나 버텍스 및 인덱스를 불러오는 과정에서 메시 데이터에 따른 UV가 이상하게 불러와짐을 확인했다. 대체하는 방법을 찾아보던 도중 assimp라는 모델 에셋 임포터 오픈소스를 발견하여 이 라이브러리를 적용하여 일부 구조를 수정하였다. 현재 장헌이가 작업중인 애니메이션등 대부분 기능들이 fbx 로더와 연결되어 있는점에 의해 모든 모델 로드 과정을 assimp로 대체하는 것은 무리가 있어, 기존의 모델 로드 과정중 버텍스와 인덱스 구조체들을 불러오는 과정에 assimp를 통해 불러온 노드를 통해 이 데이터를 덮어씌우는 과정을 추가중이다. 우리가 익스포트한 fbx모델을 assimp를 통한 directx12 샘플 프로젝트에 적용했을 땐 정상적으로 텍스쳐가 표시되는 것을 보아 충분히 정상작동 할 것으로 보인다.

추가로 AI작업을 진행중이다. 아직 상태에 따라 애니메이션이 변동되는 것은 미적용했으나 게임오브젝트에 추가한 AI 컴포넌트를 통해 각 플레이어의 위치정보 등을 업데이트 호출 시 탐지하여 상태 변경에 이용한다.

일부 넘어갈 수 없는 커다란 구조체(건물, 차량)등은 해당 구역을 박스 콜라이더로 적용하고 a\*노드를 우회경로에 배치하여 플레이어 추적 시 플레이어가 건물 너머로 도망칠 시 추적가능하도록 구현중이다.

**▶ [엄장헌]**

애니메이션 인스턴싱 관련해서 굉장히 많은 고민을 하였다. 우선 같은 메쉬를 공유하면서 애니메이션은 따로 사용해야되는 경우 인스턴싱이 힘들다는 점이다.

처음에는 같은 메쉬를 쓰면서 각자 다른 애니메이션을 적용하기 위해서 compute shader를 어떻게 쓸지 고민하였다. 우리의 인스턴싱 매니저에 의해서 애니메이션이 있는 객체는 1. compute shader를 이용해서 UAV를 통해 bone의 관계를 사용하여 계산을 하고 버퍼에 저장해둔다. 2. 인스턴싱 객체들의 위치정보와 회전정보들을 이용해서 한번에 그린다-> 이 과정이다. 처음에는 VS에서 각 객체가 애니메이션 계산을 불러오면 되지 않을까 생각했으나, 정점마다 계산시 너무 큰 손해가 발생한다고 생각했다. 그 다음으로는 compute shader에 넘기는 structured buffer를 개조해서 보내는 것을 생각했다. 하지만 이 경우에 너무 많은 곳에서 structured buffer를 사용하고 이를 전부 변경하기에는 성공할지, 이게 맞는 방법인지 의심이 들어서 선뜻 진행하지 못했다. 이후 몇가지 실험을 통해 한가지 사실을 알았다.

우리의 모델은 로우 폴리곤이라는 점이, 각 객체의 메쉬정보를 그래픽카드로 보내는 시간보다 애니메이션 계산의 시간이 더 길다는 사실이다. 따라서 시간복잡도가 애니메이션 계산에 종속되어 인스턴싱의 장점이 사라진다는 점이었다.

인스턴싱 없이 랜더링할 경우, 화면상에 애니메이션 객체가 64개일때 프레임이 170~200프레임이었고, 우리의 게임에서 평균적으로 한 화면에 애니메이션 객체가 10개 이하라는점과 이때 프레임이250~300이라는 점에서 인스턴싱을 사용하지 않는것이 꽤 좋은 선택지라는 점이다.

일단 머테리얼을 클론하여 하나의 정보로 공유하여 사용하는 것이 아니라 효율이 떨어졌다… 하지만 지금의 내 실력으론 해결하기 힘들어 보이고 더 이상 시간을 쓰기에는 기한이 촉박했다. 또한 내 정신적 스트레스가 심해서 놓아버렸다. 너무 아쉬운 마음이 크지만 이것만 잡고 있을수는 없어서 우선 인스턴싱 없이 클론된 머테리얼을 쓰는 차선책을 사용했고, 목표였던 서로 다른 애니메이션 로드는 성공했다.