| **비단길(Silk Road)** | |
| --- | --- |
| **5주차** | **2024.01.21 (일) ~ 2023.01.27 (토)** |

**■ 주간 회의록ㅣ**

**■ 다음 주 세부 계획**

**▶ [권순원]**

* 서버 클래스 작성 완료
* 호스트, 게스트 연결 구현

**▶ [박준영]**

* 게임 월드 내 오브젝트 picking 이전에 작동하는 UI 객체 클릭 구현
* unity를 통해 레벨 디자인 후 맵 내 모든 객체 불러오기 시도

**▶ [엄장헌]**

* 삼차원 공간에서의 완전탄성충돌 공부후 적용
* 아직 n^2으로 충돌검사 중 -> 공간분할 재도전

**■ 작업 일지**

**▶ [권순원]**

일단 메인쓰레드와 서버쓰레드 간의 데이터 전달을 위한 버퍼 역할을 할 수 있는 변수를 만들려고 했는데, 생각해보니까 게임루프에 필요한 많은 정보들을 하나의 구조체나 클래스로 만들어서 넘기는 것은 어려울 것 같다고 생각했다. 그래서 이전에 임시로 만들어 놓은 패킷 구조처럼 앞부분에 헤더를 두어 타입에 따라 데이터를 확인하고 적용할 수 있도록 하는 방식을 생각해 보았다.

코드를 작성하기 전에 쓰레드와 쓰레드 동기화에 대한 부분이 헷갈려서 테스트 프로젝트에서 여러가지 실험을 하면서 다시 공부했다.

몇몇 키워드나 함수가 헷갈렸지만 다행이 대부분 기억하고 있어 금방 내가 원하는 방식대로 간단하게 쓰레드 동기화를 구현하고 다시 공부해볼 수 있었다.

쓰레드끼리 데이터를 주고받는 것은 테스트를 통해 떠올릴 수 있었다. 그런데 새로운 문제가 생긴 부분은 데이터가 간단하지 않고 많은 오브젝트에게 각각 데이터를 전달해야 한다는 부분이었다. 우리 게임에서는 오브젝트가 한두개가 있는 것이 아니다. 모든 오브젝트가 서버에 의존할 필요는 없지만 서버에서 정보를 받아 랜더링을 해야 하는 오브젝트가 적어도 수십개는 될 것이다. 그렇다면 여기서 각각의 오브젝트들에게 서버에서 받은 정보를 전달해줘야 하는데 어떤 식으로 처리를 해야 될지 고민이 생겼다.  
당장 생각나는 데이터를 각각 적용하는데 사용할 수 있는 것  
Object 클래스의 m\_id  
지난주에 패킷 임시로 지정한 것처럼 큐에 id와 변경되야 하는 데이터를 넣고 클라이언트에서 큐가 비어있지 않으면 데이터들을 꺼내 확인하는 방식으로?조금 복잡하긴 하지만, 이런 식으로 오브젝트의 ID를 구할 수 있다. 아마 내가 생각한 방식대로 ID를 통해 업데이트를 하게 된다면 이 과정을 간소화할 필요가 있을 것 같다.

위에 고민한 것들을 토대로 이전에 만들었던 서버 클래스의 형태를 다시 생각해보았다.

* 엔진 클래스가 서버 클래스를 하나 가지고 있음
* 서버는 쓰레드 함수를 가지고 실행/종료 시킬 수 있음
* 쓰레드 간 동기화에 사용할 동기화 변수는 서버 클래스가 가지고 관리
* 서버 클래스가 게스트와 통신까지 한 번에 관리(이전에 생각했던 PacketManager는 단순히 편의를 위한 랩핑 클래스 역할밖에 하지 못할 것 같았다.)
* 게스트 플레이어가 호스트에 접속할 때 쓰레드를 종료하게 하는 것도 서버 쓰레드에서 관리(이와 관련해서는 detached된 쓰레드를 종료하는 방법이 필요하다 생각을 했고 [[C++] C++에서 std::thread를 어떻게 종료시킬 수 있을까? (wookiist.dev)](https://wookiist.dev/70) 이런 내용을 찾아볼 수 있었다.

이번 주는 코드를 작성하는 과정이 거의 없었다. 하나를 만들면 새로운 고민거리가 생기고, 이를 해결하면 다시 문제가 생기는 등 서버를 제작하는데 있어 내 지식이 아직은 많이 부족하다고 느꼈다. 서버를 목표로 하진 않았기 때문에 쉽게 넘어갈 수 있을법한 간단한 부분도 계속 고민하게되고 찾아보게 되는 것 같다.

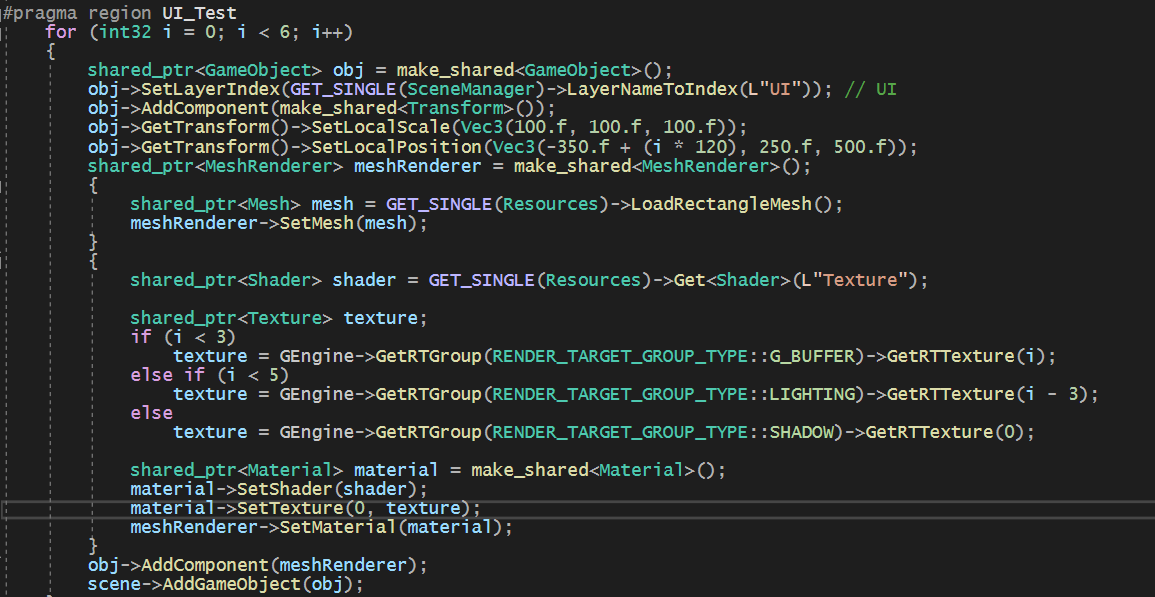
앞으로 남은 한 달의 방학기간 안에 서버 기반을 마련하고 내가 담당한 클라이언트 부분으로 넘어갈 수 있도록 노력이 많이 필요할 것 같다.

**▶ [박준영]**



현재 DWrite를 통한 텍스트 출력은 가능하며

D3D11On12는 상당한 병목을 일으킨다고 한다. 우리 프로젝트 정도 규모에서 유의미한 차이가 날지는 아직 실험해보지 못했으나 가급적 텍스쳐를 별도의 변환없이 그대로 스왑체인에 마지막으로 렌더 하여 네이티브로 개발하는것이 좋긴 할 것이다. 그러나 폰트 출력까지 네이티브로 구현하는건 상당한 시간이 걸릴 것으로 예상되어(특히 한글) D2D 기반으로 그대로 가야할 것 같다.

****



기존에 프레임워크 제작 당시 구성했던 UI출력 방식은 위와 같다.

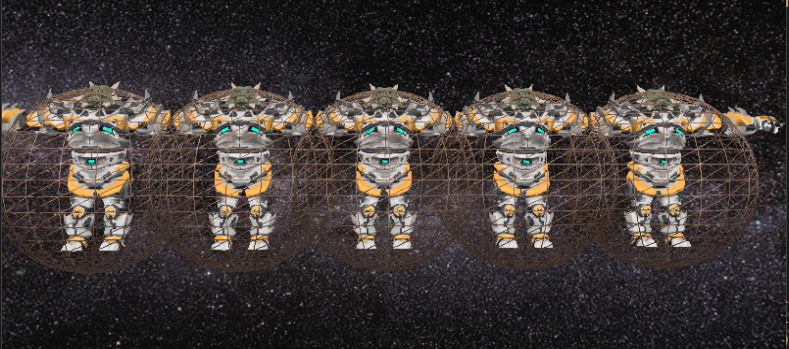
저 당시는 텍스트 출력에 대한 부분은 큰 고민을 안하였으나 RectangleMesh 객체를 통해 원하는 텍스쳐를 그대로 화면상에 마지막으로 render하는 것은 가능하다.

버튼, UI레이아웃, 백그라운드 창 등은 dds를 불러와서 위의 방식으로 구현해도 좋을 것 같다.

우선 현재 목표는 picking 기능을 손봐서 ui 객체를 클릭할 수 있게 까지 개발 하는 것이다. 원래 이번주까지 완료 되었어야 했는데 b형독감에 걸려버려 며칠간 제대로 작업이 힘들었다. 스택 구조의 팝업 창까지 이번에 구현해보고 싶었으나, 우선은 게임 내에서 등장할 모든 오브젝트들을 씬에 불러오는 것 우선적으로 진행해야 할 것 같다.

UI는 기반 자체는 어느정도 잡아 놨으니 남은 방학기간 동안은 남은 게임 플레이에 필수적인 컴포넌트들 개발과 맵 구성을 위주로 진행하고 UI는 중간발표 이전에 한번 더 다듬는게 좋을 것 같다. 이번 주말동안 버튼까지만 끝내고 다음주 근무시간 남을때 짬내서 유니티를 통한 맵 export 방식을 고민 해봐야 겠다.

**▶ [엄장헌]**

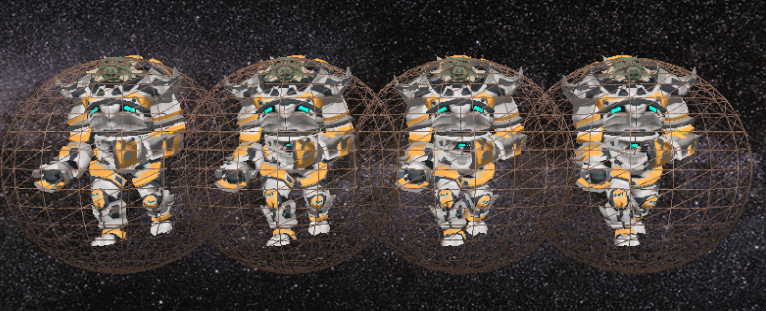


저번주에 만든 충돌 처리를 위한 collider를 테스트하기 위해 시각화 하였다.

콜라이더만을 위한 쉐이더를 만들고 wireFrame으로 세팅을 하였으며 우선은 sphereCollider에만 적용해보았다.

게임 객체에 collider 컴포넌트를 장착만 하면 알아서 해당 콜라이더가 그려진다. 테스트 용이며 끄고 킬 수 있다.

작업을 하던 도중 몇가지 문제가 있었다.



먼저 하나의 메테리얼을 여러 객체가 공유해서 사용한다. 이 때 애니메이션 데이터도 함께 공유하게 된다. 정지된 물체의 경우에는 공유, 인스턴싱 방식이 굉장히 메모리를 절약할 수 있지만 애니메이션의 경우에 모든 객체가 동일한 애니메이션을 나타냈다. 따라서 나중에 clone등의 함수를 만들어서 따로 복사해서 사용해야 할 것 같다.

두번째로는 우리의 프러스텀 컬링 코드에 문제가 있다는 것을 확인했다.

분명 카메라 안에 있음에도 처음에 프러스텀 밖에 있었다면 컬링 되는 현상을 발견했고 이는 차후 고쳐야 할 것 같다.

이전에 충돌을 위한 매니저를 따로 만들었는데, 오버헤드도 많고 따로 매니저를 만들었음에도 불구하고 active\_scene에 너무 결합된 느낌이 컸다.

따라서 충돌을 위한 코드를 scene\_manager로 옮겼다. 완전 탄성 충돌로 충돌 테스트를 진행하였다. 다만 충돌에 대한 물리 식이 선형 충돌에 국한되어 있어서 부자연스러운 느낌이 강하다. 다음주에 각도를 적용한 3차원상의 완전 탄성 충돌을 공부해봐야겠다.