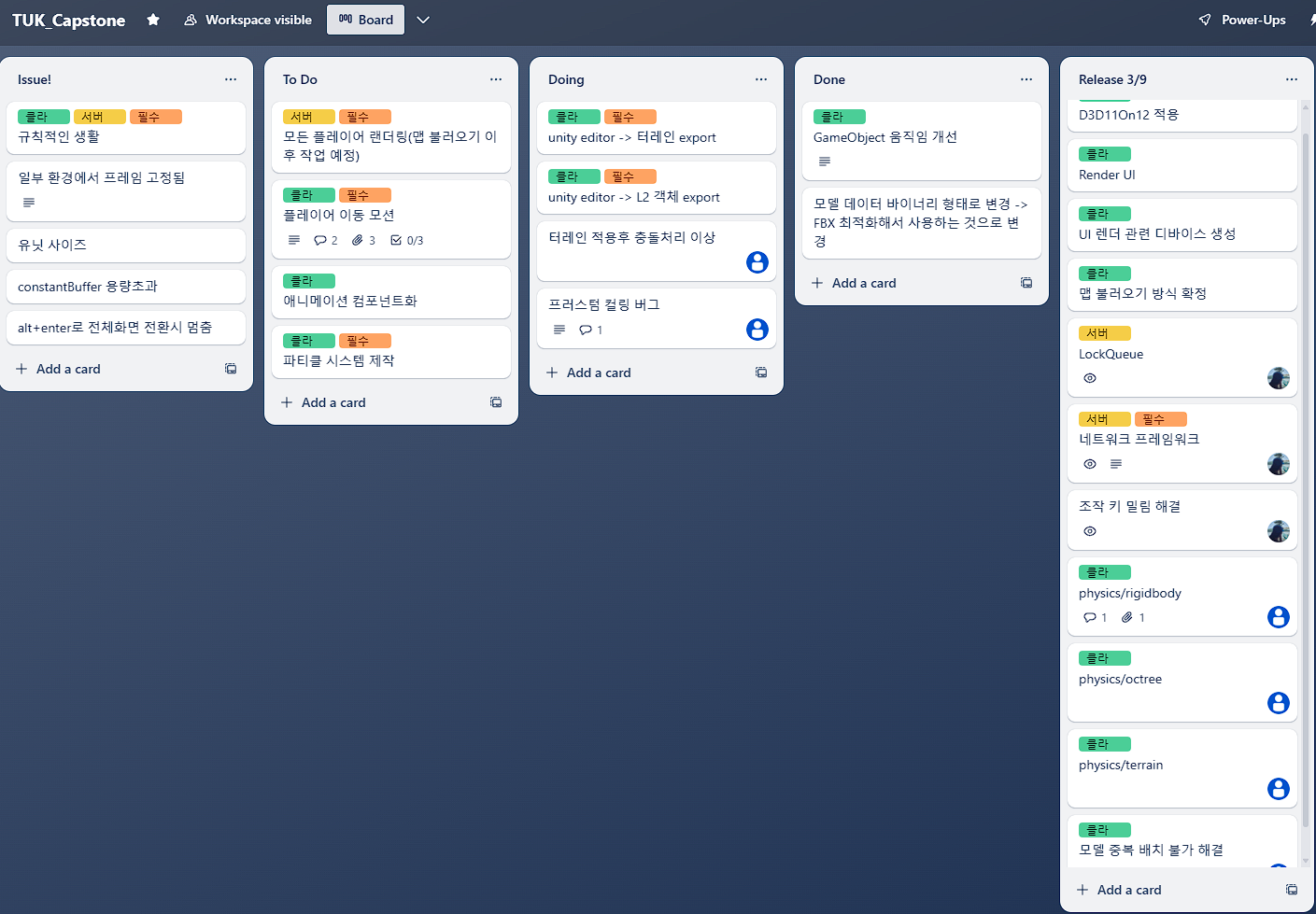
| **비단길(Silk Road)** | |
| --- | --- |
| **11주차** | **2024.03.03 (일) ~ 2024.03.09 (토)** |

**■ 주간 회의**

**▶ 방학동안 작업물 최종 병합**

* 방학 기간 중 작업했던 부분들을 병합함
* 각자 파트를 잘 나누어 작업해서 그런지 문제 없이 쉽게 병합 완료 했음

**▶ 기타 작업하면서 발견한 문제점 공유(트렐로 이용)**

****

**▶ 3, 4월 목표**

* 싱글 플레이 가능한 스테이지 하나 구현(콘텐츠 구현 위주)
* 현재 트렐로 이슈에 있는 문제들 해결

**■ 3월 개인 목표**

**▶ [권순원]**

* 서버 작업 때문에 작업 못한 파티클 시스템 작업 시작
* 맵 불러오기가 완료되면 멀티플레이 가능하도록 플레이어들 랜더링 작업

**▶ [박준영]**

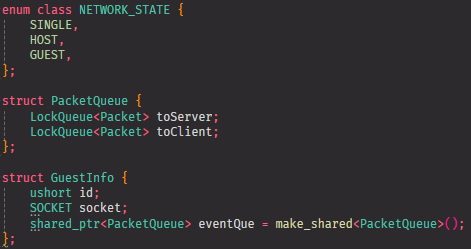
* unity editor 상에서 배치한 맵 2 트랙으로 export (터레인, 기타 오브젝트(리지드바디 별도)
* ai 컴포넌트 구현

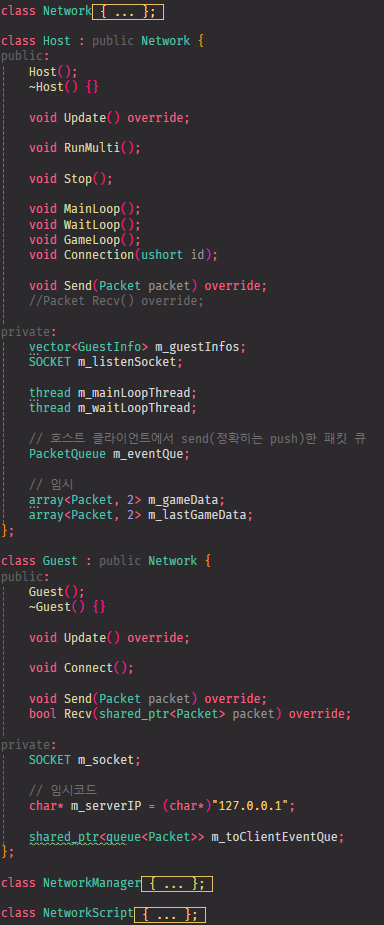
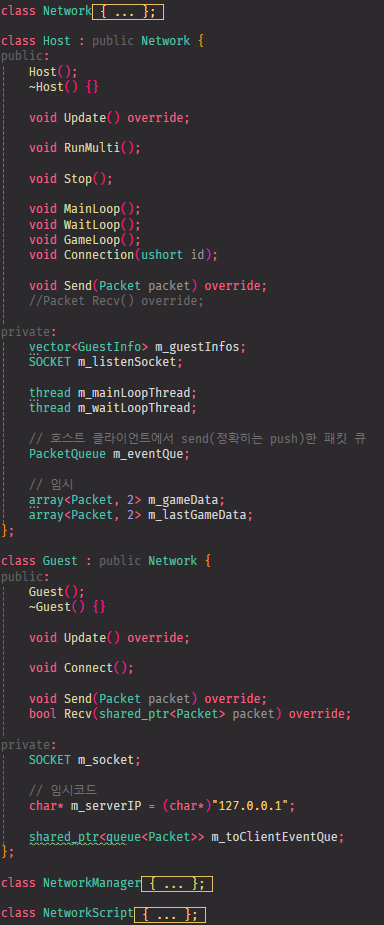
**▶ [엄장헌]**

* 애니메이션 데이터만 추출 후 컴포넌트화
* 서로 다른 모델에서 원하는 애니메이션만 골라서 AddComponent가 가능하게 만들기
* 이를 전반적으로 관리하기 위한 애니메이션 컨트롤러 추가

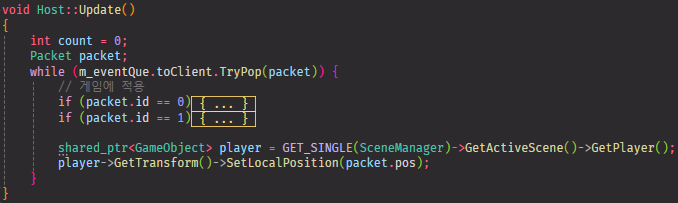
**■ 작업 일지 및 방학 리뷰**

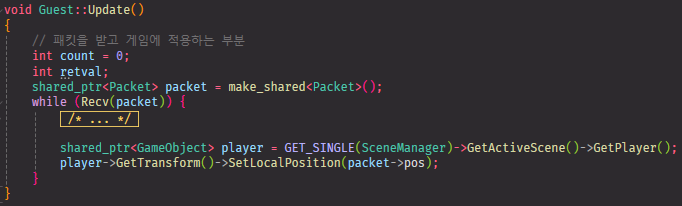
**▶ [권순원]**

지난 주에 계획했던 방향으로 네트워크 코드 개선을 완료했다.

클라이언트에서 네트워크에 관련된 작업은 모두 NetworkManager를 GET\_SINGLE로 호출해서 사용하게 되고 관련된 키 입력은 NetworkScript에서 처리하도록 만들었다.

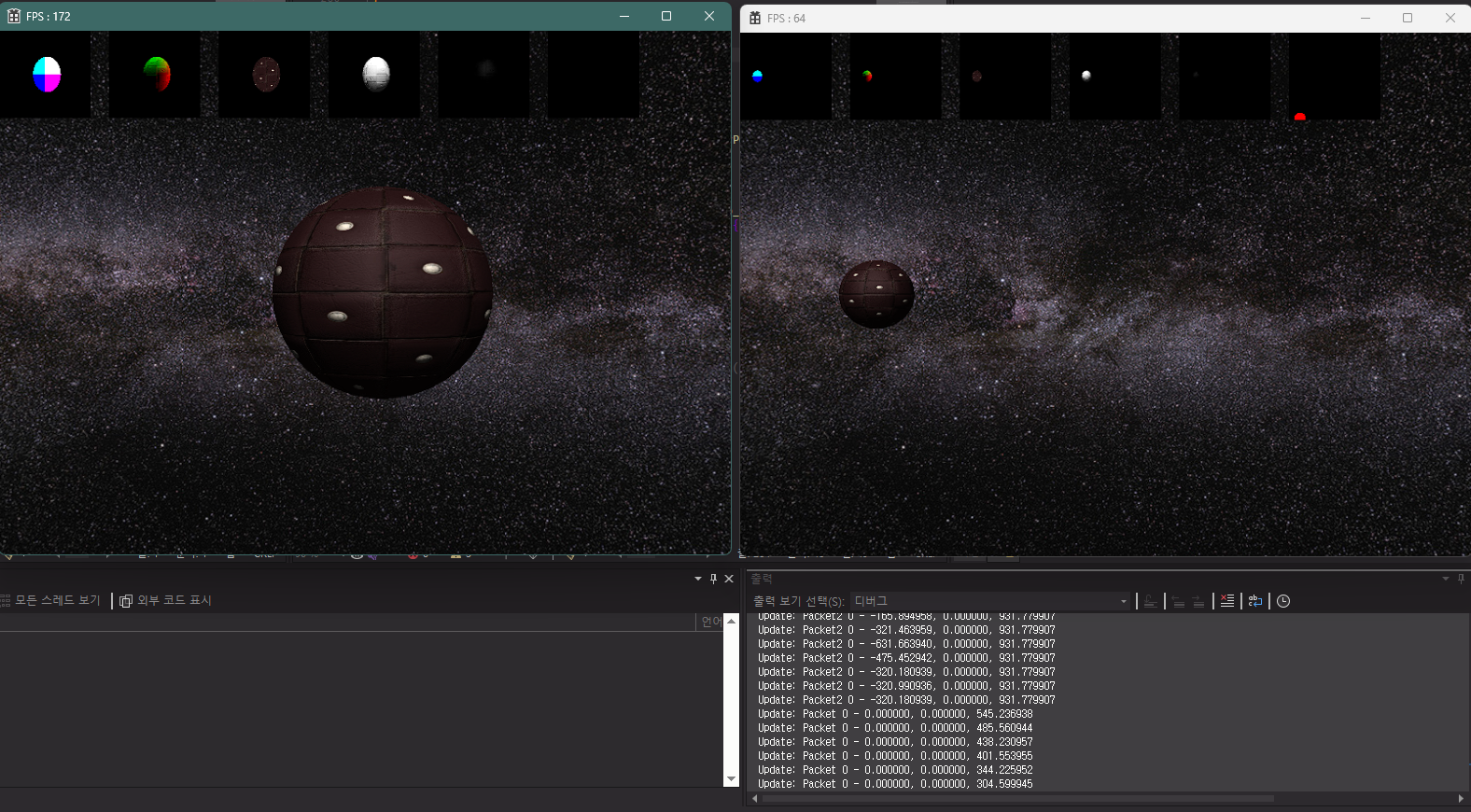
Host, Guest의 Update()는 서버가 처리한 후 보내준 데이터를 클라이언트에 적용하는 역할을 하는 함수로 만들었다.



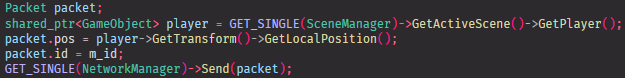


자신이 가지고 있는 EventQueue에 패킷이 존재하면, 큐가 빌때까지 패킷을 꺼내 적용하는 방식으로 제작했다. 현재는 씬에 플레이어에게만 데이터를 적용하게 만들었다. 추후 Packet을 수정하게 되면 이 부분도 패킷에 적힌 오브젝트에 적용하는 방식으로 교체할 예정이다.

Host와 Guest의 Update()를 구분한 것은 Host는 자신의 클라이언트에서 쓰레드로 서버가 동작하는 것이기 때문에 따로 Send, Recv를 사용하지 않고 EvnetQue를 직접 이용해 데이터를 주고 받을 수 있도록 만들었다.

****

현재는 단순하게 위치정보만 주고 받기 때문에 별다른 적용없이 디버그 콘솔에 위치를 출력하여 테스트했다.

****

데이터를 서버로 보내는 부분은, 임시로 씬에 플레이어 shared\_ptr로 만들어 접근해 정보를 데이터를 넘기도록 제작했다. 이 부분은 씬에서 원하는 객체를 찾는 함수를 제작해 변경해야하는 부분이다.

테스트하는 과정에서 한 가지 문제점을 발견했는데 호스트 클라이언트의 프레임과 게스트 클라이언트의 프레임이 차이가 발생하는 문제가 있었다. 예상으로는 서버와 패킷을 주고 받는게 너무 빈번하게 발생해서 그런거 같다는 생각을 했다. 이 부분은 본격적으로 캐릭터를 적용하고 맵을 만들면 패킷 구조를 개선하고, 1초에 서버를 처리하는 횟수를 조절해 가면서 해결 할 수 있을 것 같다.

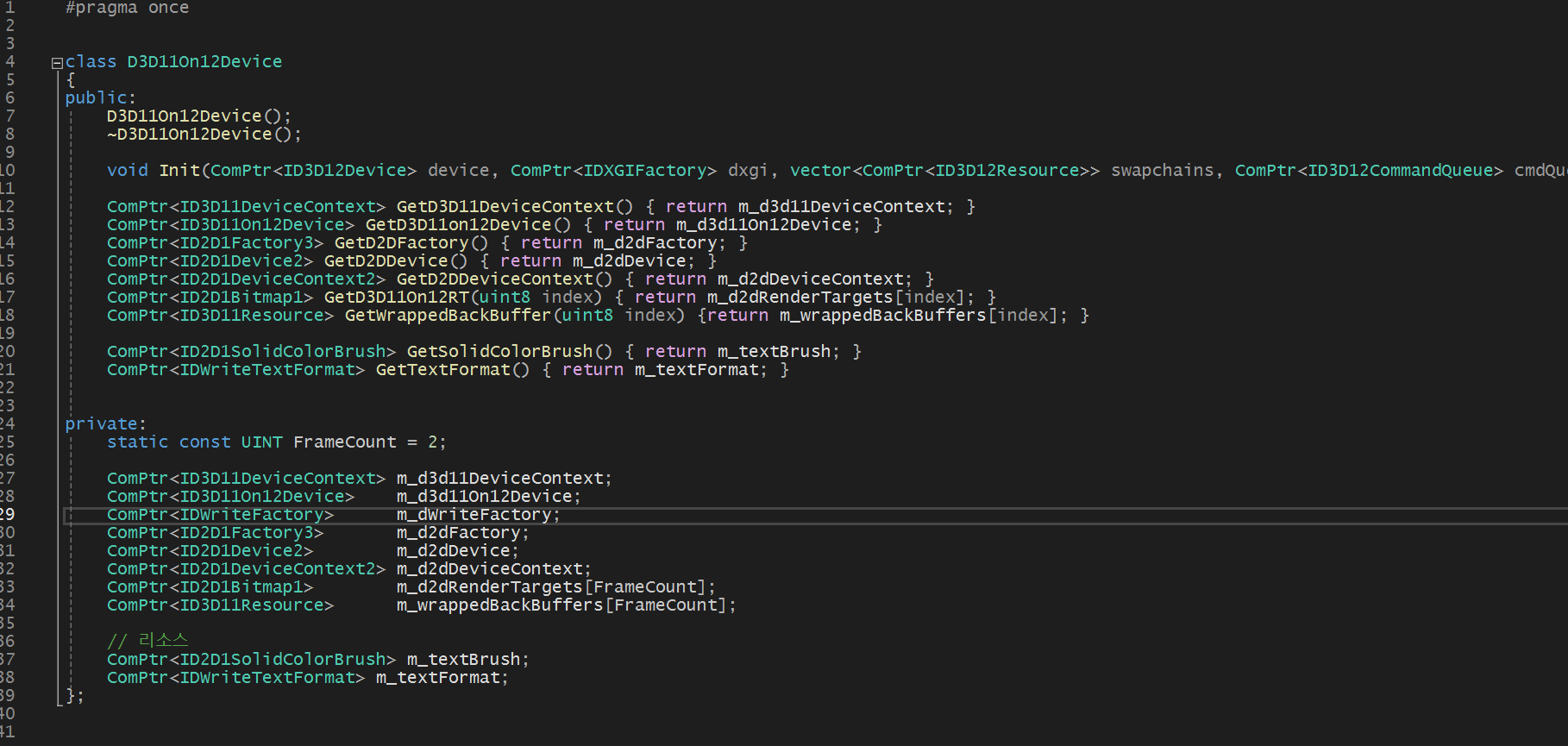
방학 기간동안 생각보다 많은 작업을 하지 못했다.

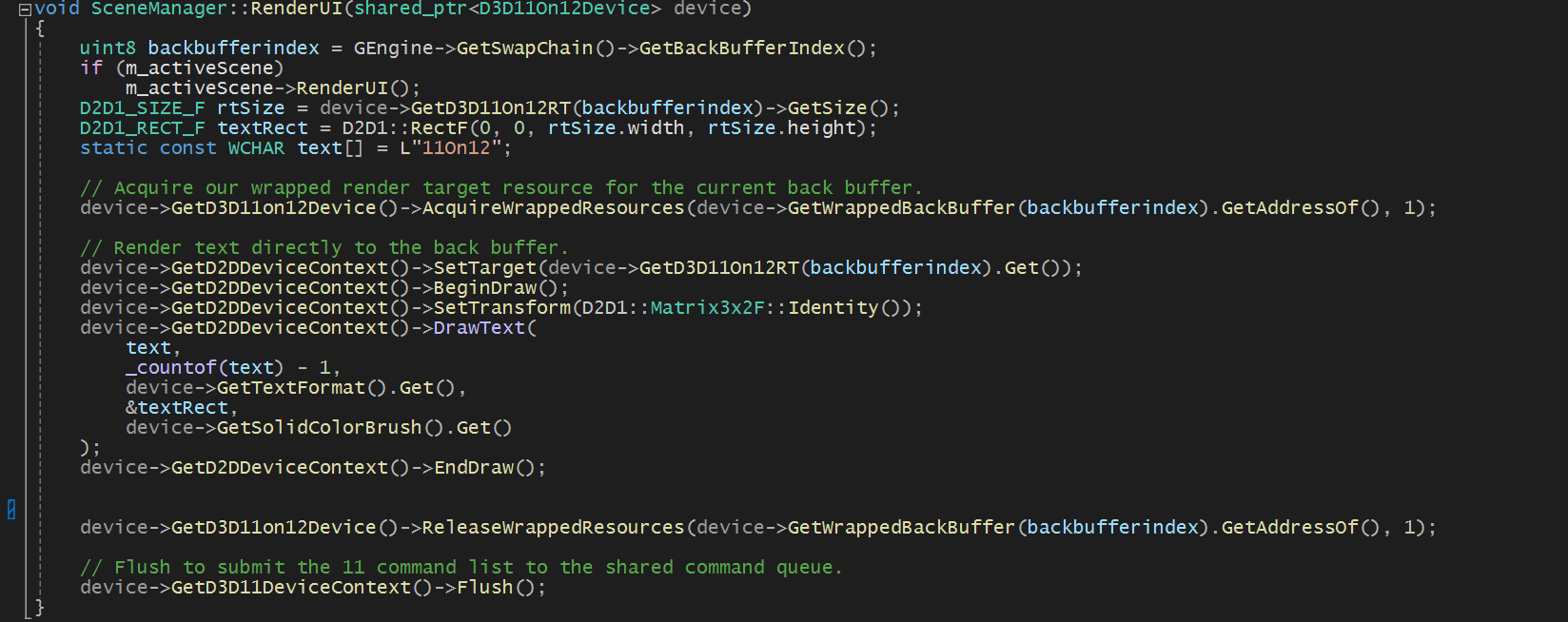
서버 부분을 내가 담당하겠다고 했을때 지난 학기 수강한 네트워크게임프로그래밍 정도의 작업을 생각하고 쉽게 끝내고 다른 작업을 할 수 있을 것이라고 생각했는데, 역시 생각대로 잘 되는 일은 없었다.

넷겜프와 다르게 모든 팀원이 네트워크에 관련된 부분만 집중적으로 작업을 할 수 없기 때문에 추후에 팀원들이 네트워크 기능을 사용하기 쉽고, 내가 아닌 다른 사람이 작업을 해야하는 부분이 있을 때를 고려하려고 하다보니 더욱 어려웠던 것 같다.

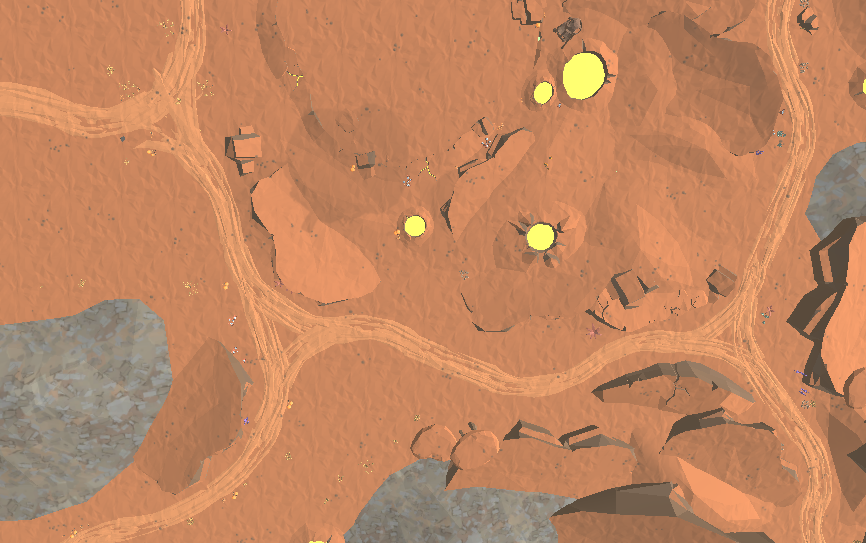
3월부터는 근로도 하게되었고, 강의도 들으면서 과제가 있기 방학 때보다 조금 더 부지런히 공부하고 작업할 수 있도록 노력해야겠다.

**▶ [박준영]**

****

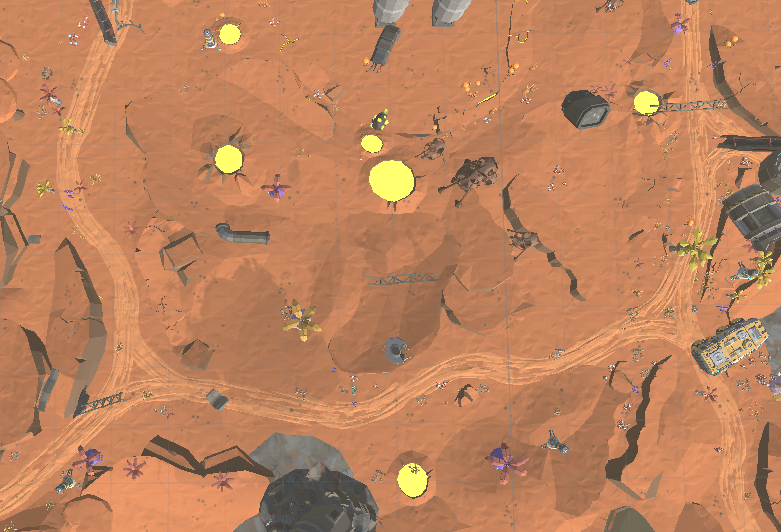
****

UI렌더를 위해 directx12 Sample 및 여러 자료를 참고하여 최종 결과물 렌더타겟에 direct2d, directdraw를 이용해 폰트, 도형, 이미지를 출력하는 기능을 수행하였다. 처음 해보는 작업이라 디바이스 셋팅 등에서 고생하였으나 현재 잘 작동중이며 API기능 사용 자체는 Win32API를 연상시킬 정도로 간단하여 UI 객체 생산성 자체는 문제가 없을 것 같다. 추가로 이 기능을 활용하여 Scene 내에 포함된 싱글톤 객체인 Canvas에 Render UI 기능을 호출하여 씬에 속한 UI 오브젝트들을 Draw하는 기능을 제작했으며, 현재는 적 타격시 표시되는 데미지, 플레이어의 체력바 표시 등 UI 오브젝트가 다른 게임오브젝트와 데이터를 공유하여 Update할 수 있는 시스템을 개발중이다.

****

**+**

****

****

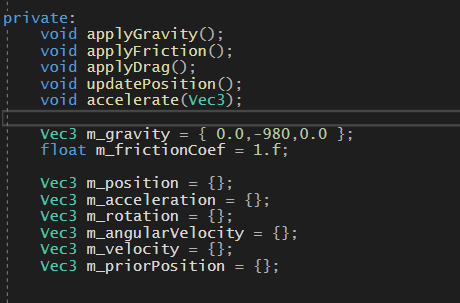
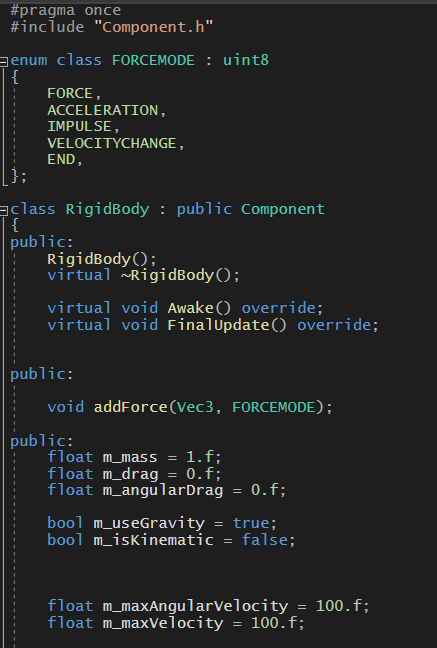
울퉁불퉁한 바위나 산지 등의 터레인과 건물, 폐차량등의 오브젝트가 혼재된 우리 게임 맵 구성에 맞게 unity editor에서 제작한 맵을 L1(터레인) L2(기타 오브젝트들)로 나누어 export하는 기능을 개발하였다. 추가로 L2단계에서는 각 오브젝트들이 별도의 Cube 형태의 Collider 박스를 갖고 있게 하여 물리 연산 및 적 개체 AI가 길을 찾는 등의 동작에 사용할 수 있도록 하였다.

AI 개발또한 착수하였는데 위 첫번째 사진인 터레인 의 각 표면 지점들에 노드를 생성하여 a\*를 통한 길찾기에 활용할 예정이다.  
맵 크기가 큰 편이다 보니 지형과 오브젝트들을 조화롭게 배치하는데 상당힌 시간이 소요되는 중이지만 완성 목표가 스테이지 3개인 만큼 각 스테이지를 꼼꼼히 제작하여 완성도를 높여보고 싶다.

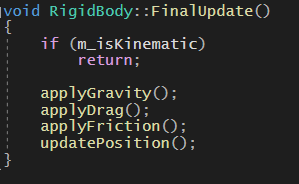
점점 게임 플레이 윤곽이 보이는 만큼 본격적인 콘텐츠 개발에 진입시 더욱 즐겁게 작업을 할 수 있을 것 같아 기대된다.

개발 착수 이후 사회복무로 인해 기간에 비해 작업이 더디게 된 점이 아쉽다. 이제 전역도 하였고 개강도 한 복학생으로서 중간발표때까지 만족스러운 게임플레이를 완성하기 위해 더욱 노력하여 작업해야겠다.

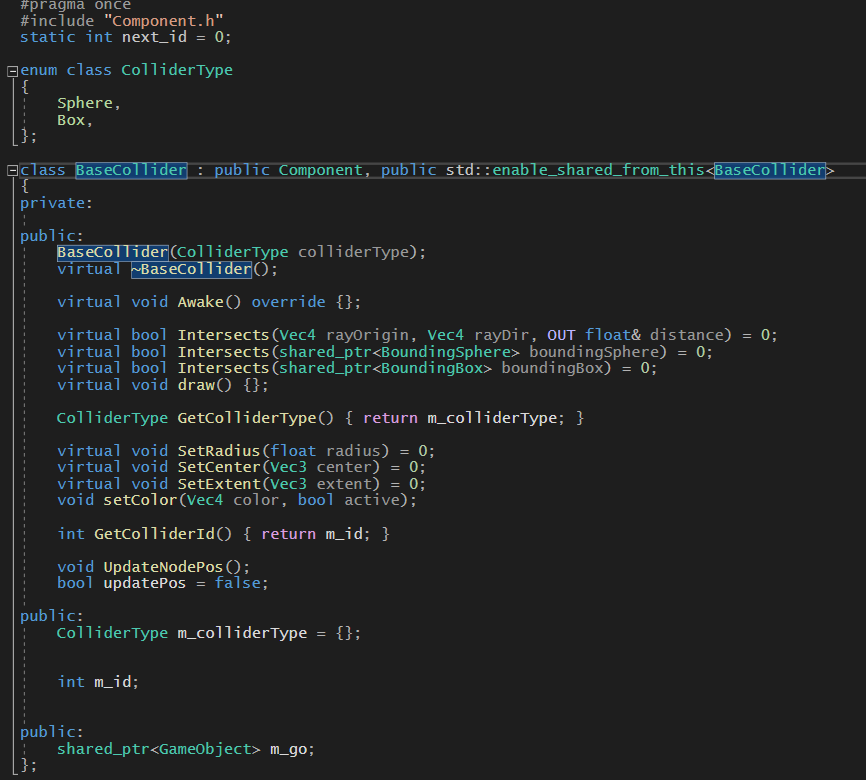
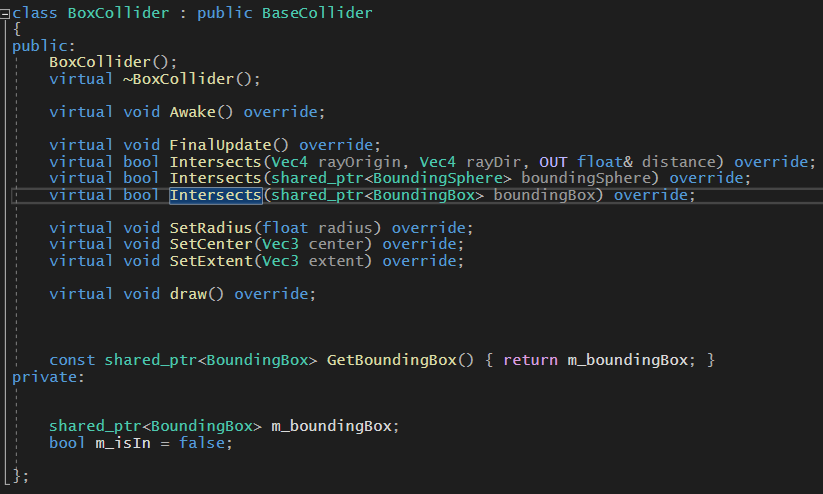
**▶ [엄장헌]**



물리를 적용하기위해 게임오브젝트에 추가할 수 있는 컴포넌트인 RigidBody를 가장 우선적으로 만들었다. 중력이나 물리적인 영향을 받을지를 정하는 플래그들과, 질량, 속도, 회전량 등 물리적 속성들을 가지고 있다. 현재 공기저항, 마찰, 힘, 중력, 가속등의 물리적 요소들을 작성하였다.



게임 파이프라인에서 랜더링 직전에 물리적 작용을 반영후 그리게 된다.



ㅁ

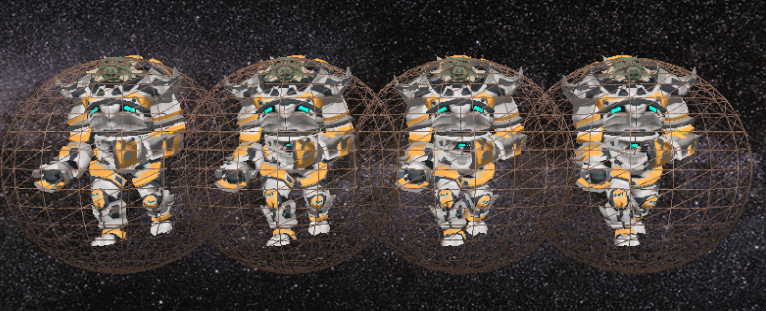
그 다음으로는 물리충돌의 기반이 될 콜라이더를 만들었다.

RigidBody를 갖는 객체는 반드시 이 콜라이더를 가져야한다.

그리고 이는 scene에서 m\_collidableGameObjects로 관리된다.

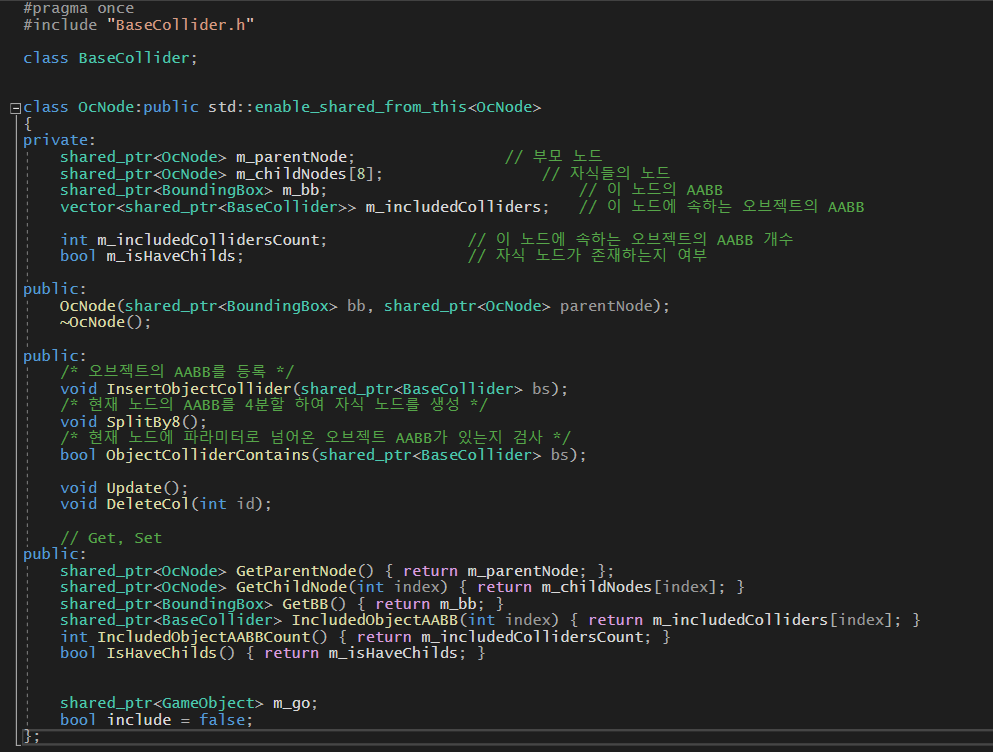
충돌범위를 시각화하기 위해서 해당 콜라이더의 모양과 크기를 갖는 게임 오브젝트를 소유하고 있다.

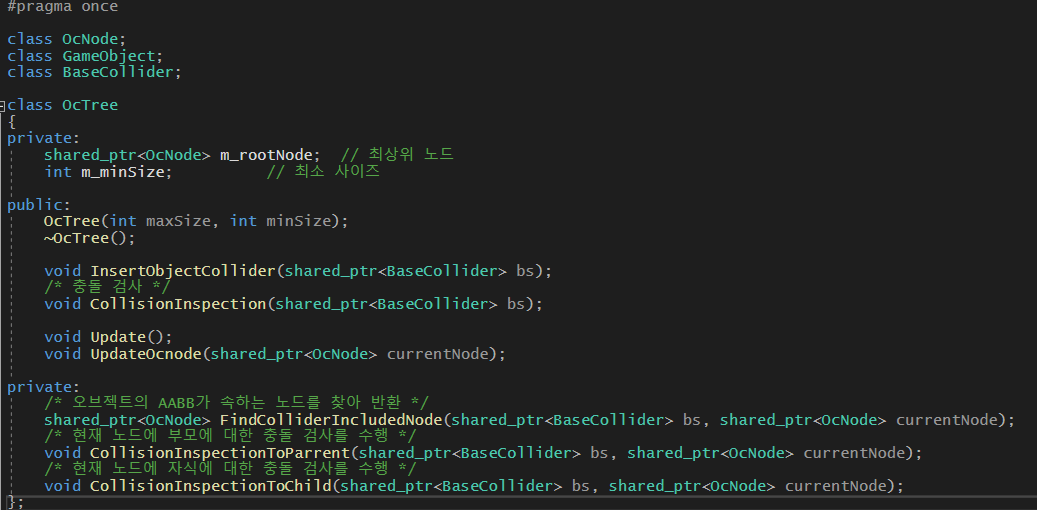
현재는 baseCollider를 상속받아 sphere, box 형태의 콜라이더를 만들었다.



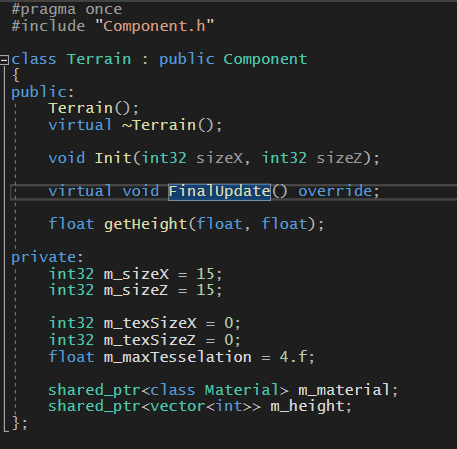
directXCollision을 이용해서 intersect와 contain여부를 판별하였고

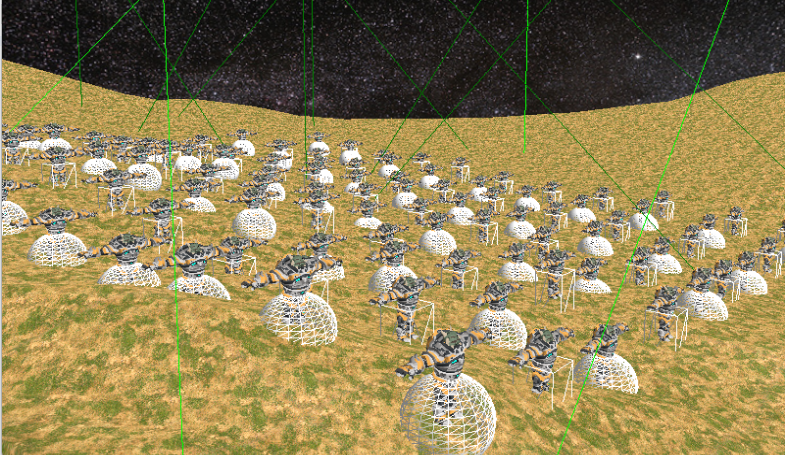
디버깅용으로 닿을시 붉은색으로 콜라이더의 색이 변화한다.





옥노드와 옥트리이다. 우리의 게임은 파티클, 적, 장애물, 총알, 아이템 등등 서로 충돌되는 객체가 굉장히 많고 넓을 것으로 예상된다. 이에 따라 n^2의 충돌 탐색은 비효율적이라고 느꼈고, 옥트리를 이용한 공간분할을 진행하였다. 처음에는 눈에 보이지도 않고 directX에서 제공하는 함수의 이해도 낮아서 많이 힘들었다. 이후 각 노드의 공간을 시각화 하기 위해서 콜라이더 때 했던 방법으로 각 노드에 육면체 메쉬를 넣었다. 그리고 강체가 이동함에 따라 노드 갱신을 해주고 옥노드를 활성, 비활성이 되게 만들었다.



중력을 적용하였으나 땅이 없었다. 터레인을 하나의 컴포넌트로 만들어 게임 객체에 add할 수 있게 만들었으며, 높이맵을 이용해서 쉐이더에서 높이를 지정해 그려지게 된다. 또한 테셀레이션을 활용해 너무 각진 모양의 땅이 아닌 좀 더 부드러운 지형을 만들었다. 그리고 터레인은 높이맵을 읽어 벡터로 저장해두었다. 터레인의 사이즈와 scale의 영향을받아 펼쳐지게 되는데, 각 rigid\_body의 위치를 기반으로 터레인에서 높이 정보를 선형보간하여 얻을 수 있다.

단체로 공중에서 떨어지면 각자 터레인 높이에 맞게 서있게 된다.

또한 초록색 선으로 보이는 각 공간 안에서의 객체끼리만 충돌검사를 한다.

이 경우 한 공간에 너무 많은 객체가 들어가지 않게 크기를 조절할 필요가 있어보인다.

이번 방학에는 1~2월의 계획이었던 물리엔진의 기본적인 틀과 터레인 랜더링과 충돌, 물리관련 루프를 구현하였다. 코드를 작성하다보니까 잘 동작하지 않아 서로 참조하면서 커플링도 생기고 코드도 조금 더럽게 작성된 느낌이든다. 3월에는 이러한 점을 조금 개선해보고 3월 계획이었던 애니메이션을 적용할 예정이다.

애니메이션의 경우는 모든 모델이 같은 애니메이션 데이터를 갖고 있는것은 비효율 적이라고 생각되어 따로 컴포넌트처럼 만들 예정이다. 서로 다른 모델이 하나의 애니메이션 정보를 공유하여 사용가능하게 하며 이를 위해 컨트롤러도 만들 계획이다.