| **비단길(Silk Road)** | |
| --- | --- |
| **17주차** | **2024.04.14 (일) ~ 2024.04.20 (토)** |

**■ 다음 주 세부 계획**

**▶ [권순원]**

* 패킷 클래스 제작

**▶ [박준영]**

* 맵 오브젝트 구현
* unity 에디터 상 맵과 동일하게 불러오기

**▶ [엄장헌]**

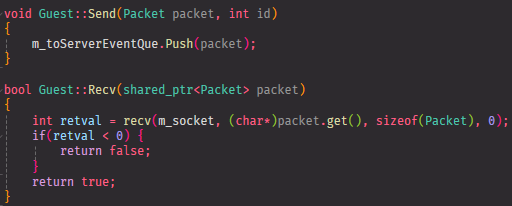
* 물리엔진에서 충돌시 회전량과 마찰력 다시 구현
* 플레이어 컨트롤 및 애니메이션 연동하기

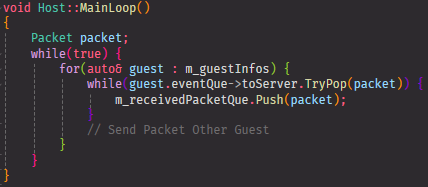
**■ 작업 일지**

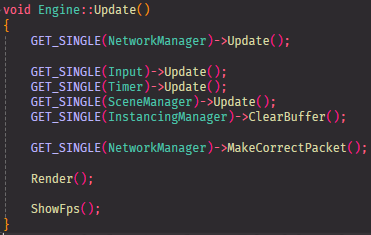
**▶ [권순원]**

지난 주 제작한 Receiver 함수와 반대 역할을 하는 Sender 함수도 필요하다 생각해 해당 함수도 제작했다.

이에 맞춰 클라이언트에서 호출할 Send, Recv 함수와 Host의 메인루플 함수도 변경해주었다.

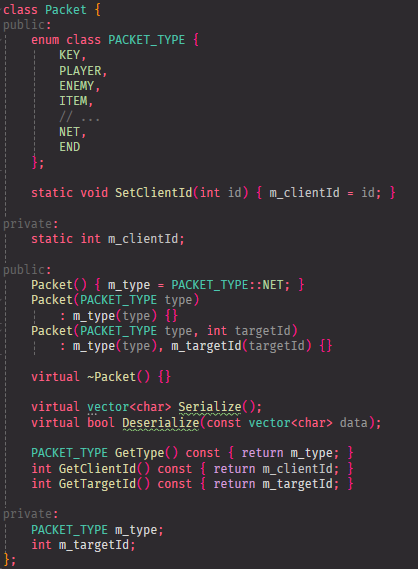
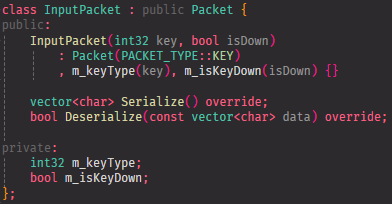
****

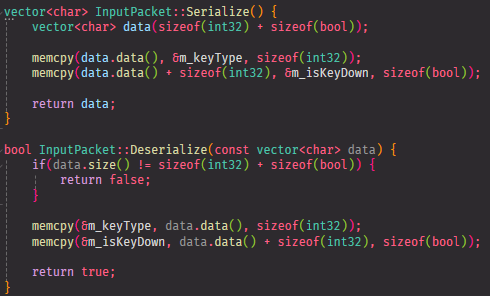
****

아직 테스트 단계에서 혼자 테스트 하기 편하게 2인으로 하고 있어 다른 게스트에게 보내는 코드는 작성하지 않았다.  
 

이제 Engine의 Update에서 NetworkManager의 Update를 호출하면 패킷에 타겟에 해당하는 객체의 정보를 다른 업데이트 전에 수행하게 되고, 정보가 저장된다.  
 그리고 MakeCorrectPacket 함수를 통해 Host의 객체 정보를 바탕으로 모든 플레이어의 데이터를 1초에 한 번씩 보정해준다.  
 현재 MakeCorrectPacket 함수는 두 가지 방법으로 각각 구현해보는 중인데 1. Update와 마찬가지로 모든 객체를 NetworkManager에서 받아와 위치 정보와 체력 등 중요한 정보를 각각 패킷으로 만들어 내보내는 방법과 2. 객체에게 네트워크 컴포넌트를 생성해 해당 컴포넌트를 가지고 있는 객체가 자체적으로 패킷을 만들어 보내주는 방식이다.

지금까지는 많은 종류의 패킷이 오고갈 일이 없기 때문에 구조체로 선언한 Packet으로 충분하다고 생각했지만 본격적인 패킷 종류를 제작하면서 구조체로는 부족하다는 느낌을 받았고 별도의 Packet 클래스를 제작했다.  
 패킷 클래스는 기본적으로 종류에 따라 구분되고 패킷을 보낸 클라이언트를 구분할 수 있는 아이디를 가지고 동작하며 클래스로 구현하는 김에 직렬화도 적용해보려고 함수를 제작했다.

****

****

맨 처음 만든 패킷은 입력을 전달하는 패킷이다.  
 vector<char>에 memcpy를 통해 직렬화 하는 방식을 선택했다. 이렇게 하면 send, recv 함수를 호출할 때 별다른 casting 없이 .data()를 메개변수로 넘겨주어 통신을 할 수 있기 때문에 더 간결하게 작성할 수 있었기 때문에 이런 방식으로 하였다.  
 패킷을 받아 역직렬화를 하는 클라이언트에서는 사이즈가 맞지 않으면 역직렬화를 하지 않고 false를 반환해 예외를 처리할 수 있도록 했다.

**▶ [박준영]**

실제 콘텐츠 구현의 일환으로 맵 배치 및 맵 오브젝트 구현을 진행중이다.

유니티 상에서 배치한 맵을 JSON으로 내보내고, 불러오는 과정은 예전에 구현해두었고 당장 플레이 가능한 맵으로서 여러 디테일들을 수정중이다.

첫번째로, 지난번에 Objects(unity) to Terrain(d3d) 변환을 거치며 제법 괜찮은 비주얼의 터레인 지형을 완성 할 수 있었는데, 우리게임의 맵은 1. height맵을 통해 생성한 터레인, 2. 돌맹이/나무/자동차 등 개별적인 오브젝트로 존재하는 맵 오브젝트, 3.스타팅 포인트나 적 스폰 포인트 등의 트리거

우리의 제작 과정상 2, 3번 요소들 또한 유니티 에디터에서 배치를 완료하여 그대로 읽어오는 것이기 때문에 서로간의 좌표가 통일되어야만 한다.

이를 고려하여 Height맵 생성 과정에서 유니티 에디터상에서의 지형의 최대높이, 최소높이를 디버그로 출력하여 이를 바탕으로 둘을 동기화 하고 있었는데, 실제로 지정된 위치에 오브젝트를 배치해보니 두 프로젝트간 높이값이 조금씩 차이가 있는 것으로 보인다. 이부분은 주요 변곡점에 ray를 쏴보며 테스트 중이다.

또한 주요 맵 오브젝트 또한 구현중이며, 각 맵에 호환되는 길찾기 노드 배열, F등의 키를 통한 상호작용 가능 객체 및 구역, 장식용 floating 객체(아이템 객체가 상속) 등을 구현중이다.

또한 현재 구현되고있는 인칭별 카메라에 맞춰 player 클래스에 대한 게임 규칙적 적용이 필요할 것 같아. (아이템등으로 변화 가능성이 있는 이동속도, 공격력, 공격속도, 점프력 등의 수치 제어) 관련된 함수들을 파편적으로 구현해 보았다.

사실 대부분의 시간을 맵 배치 및 각 오브젝트에 대한 콜라이더 적용등에 활용했는데 이렇게 배치한 맵이 실제 적용해보니 처음에 언급한 높이차등으로 인해 이상하게 표시되어 상당히 곤란한 상황이다.

어떻게 해도 해결책을 찾지 못하면 최후의 방법으로 맵에 최초 진입시 잠깐동안 중력을 적용해 오브젝트들의 높이값을 지형에 맞춰 확정시키는 방법도 고려중이긴 하지만 최후의 방법으로 남겨놓고 싶다.

**▶ [엄장헌]**

콜라이더의 세부 위치 조정을 위해 offset을 만들어 적용했다.

OBB의 충돌에서 너무 많은 연산을 한다. Sphere끼리의 연산보다 약 15~20배의 연산을 하기 때문에 프레임 차이가 크게 발생했다.

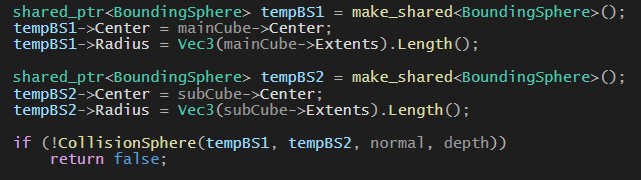
100개의 Sphere 충돌: 550 프레임

100개의 OBB 충돌 :270 프레임

이런 프레임 차이를 줄이기 위해 OBB충돌 연산 이전에 OBB를 감싸는 Sphere를 만들고 사전에 충돌이 가능한지 검사하였다.

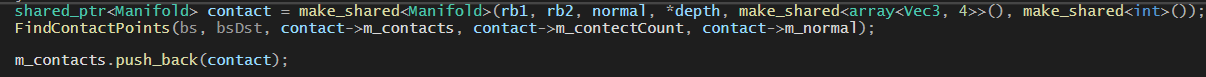
100개의 OBB 사전 검사후 충돌: 380프레임

위의 100개는 movable객체들이고 static의 비중이 높아질수록 더 빨라진다.



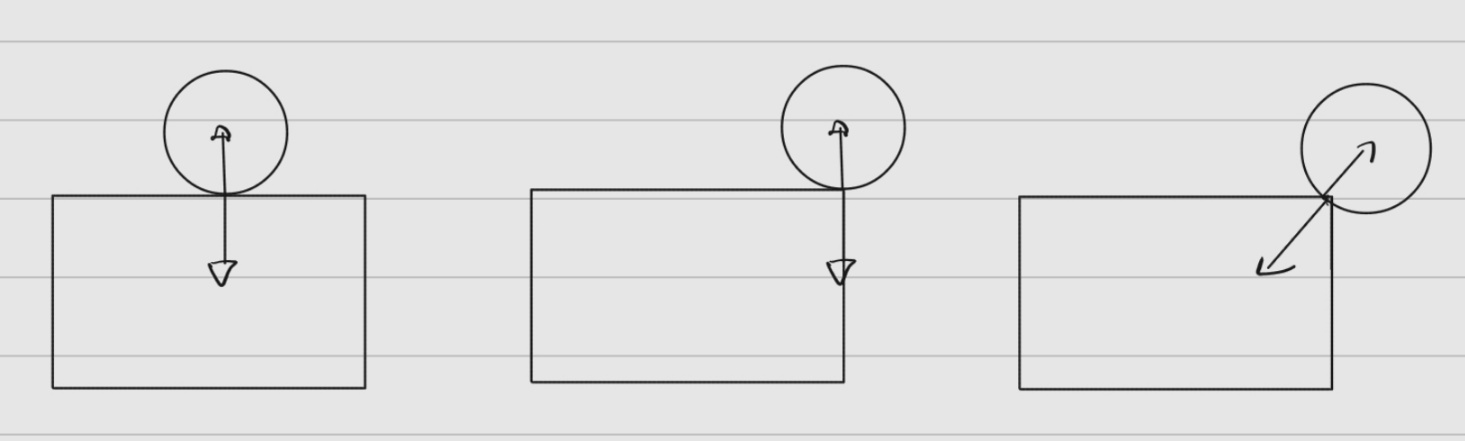
그리고 물리엔진 내에서 불필요하게 불리는 함수들과 테스트 코드들을 전부 정리하고 주석 작업을 진행했다. 이전에 옥트리에서 처리하던 충돌 함수들은 다른 파일로 분리시켰다.

여러개의 점에서 동시에 충돌이 발생하는경우의 문제를 해결하기 위해 Manifold를 만들어 충돌정보를 전부 담아둔다. 이후 모든 충돌을 검사하고 resolution단계를 동시에 진행했다.

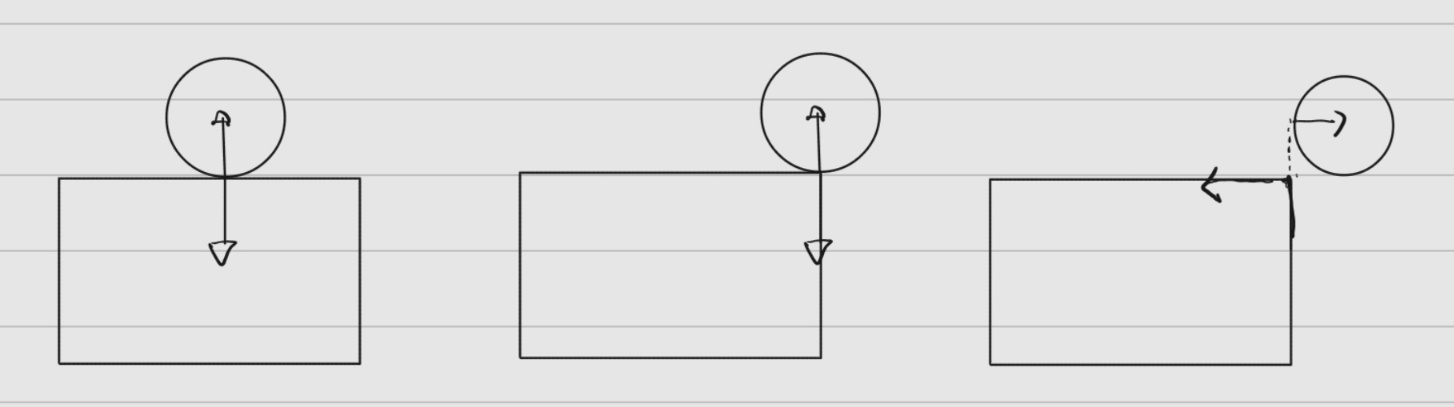


이 과정에서 문제를 하나 발견했다.

구와 육면체의 충돌에서 정확한 충돌점을 찾지 못했다는 것이다.



다음과 같이 육면체의 면 안에서 구의 중점과 가장 가까운 점을 찾고 그 점에서 구의 중점까지의 방향벡터, 겹쳐진 크기를 구해야되는데 회전하는 OBB특성상 구하기가 까다로웠다. 현재는 아래와 같이 육면체 면의 법선벡터로만의 축만 찾는중이다.



이를 해결하기 위해 OBB의 면위의 점중 구와 가장 가까운 점을 찾아야한다.

하지만 OBB는 회전이 가능하며 이러한 상황에서는 모든면에 수선의 발을 내려가며 찾아야하고, 수선의 발이 면 밖에 있을때 변으로 내적 시켜야하는등 꽤나 복잡한 절차가 필요하다. 따라서 구를 OBB의 로컬 공간으로 역변환 하는 방법을 사용했다.

OBB의 회전정보와 위치정보를 토대로 월드변환행렬의 역행렬을 구했고, 구의 중점을 변환해주었다. 그러면 축에 정렬된 육면체가 0,0,0의 중점을 갖는 공간으로 이동된다. 여기서는 단순히 clamp를 이용해 가장 가까운 점을 찾을 수 있고, 다시 월드좌표계로 변환하여 충돌축을 얻을 수 있다. 이 충돌축에 내적을 하여 겹친 크기도 알 수 있게된다. 다음과 같이 고치면서 구와 OBB사이에선 SAT를 사용할 필요가 없어져서 성능개선의 효과도 있었다.

다음으로는 OBB와 OBB의 충돌에서 닿는 지점을 찾아야한다. 가정으로는 최소 1개에서 최대 4개의 점이 발생한다. 처음 생각한 충돌지점을 찾는 방법은 SAT과정에서 얻어진 최소 침투축을 이용해 Ray Triangle 테스트를 모든 점과 면에서 진행하는 것이다. 다만 이는 시간이 너무 많이 걸릴 것 같았다. 다음으로 생각한 것이 SAT과정에서 최소침투축을 만드는 꼭짓점과 이 침투축으로의 depth가 거의 차이가 없는 점들을 모두 찾는 방법이다. 이 방법은 어차피 진행하는 SAT과정에서 조금의 코드를 추가해 만들 수 있을 것 같다. 일단은 다음주가 시험주간인 관계로 이후 과정은 나중에 할 것 같다.