| **비단길(Silk Road)** | |
| --- | --- |
| **20주차** | **2024.05.5 (일) ~ 2024.05.11 (토)** |

**■ 다음 주 세부 계획**

**▶ [권순원]**

* 다른 객체들(몬스터, 총알) 움직임 동기화
* 네트워크 예외처리
* 하드코딩된 recv 과정 등 함수화

**▶ [박준영]**

* 플레이어 추가 스킬 구현

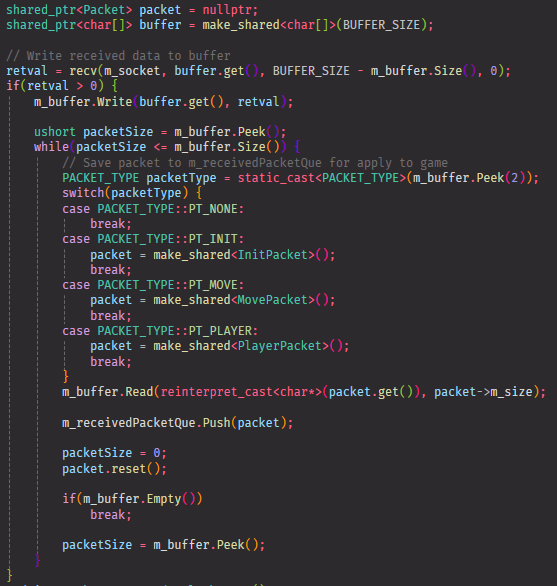
**▶ [엄장헌]**

* 이번 테스트 과정에서 발견된 물리엔진 버그들 수정
* 물리엔진 충돌 검사 성능 개선

**■ 작업 일지**

**▶ [권순원]**

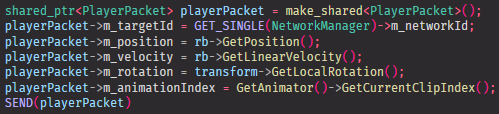
**패킷 송수신 작업 완료**

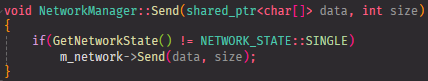
****

지난 주에 구조를 다시 잡아 작업한 네트워크 부분은 제대로 동작하도록 완성했다. Recv 함수의 구조는 buffer 선언 방식이나 Peek()함수를 인덱스로 접근할 수 있도록 하는 등의 약간의 변화를 제외하고는 그대로 사용했다.

recv로 전달된 데이터를 모두 버퍼에 옮긴 다음, 그 크기를 확인, 패킷 타입을 확인해 포인터를 선언해 실제 게임에 적용하기 Update에서 불리는 큐에 넣는다.

Send를 하는 부분은 이전과 달라졌다. 패킷의 크기는 종류마다 다르기 때문에 기존처럼 Sender함수로 Packet을 넘기는 대신 Packet 클래스와 사이즈를 넘겨주었다.

****

****

Send 함수에 char배열을 매개변수로 주어 Packet 클래스로 넘겨주었을 때 Packet 크기보다 클때 쓰레기값이 들어가는 버그를 해결해주었다.

이제 게임의 오브젝트에 따라 패킷을 계속 만들어주고, 다른 플레이어의 게임 종료, 튕김 등등 예외처리만 해주면 네트워크는 잘 마무리할 수 있을것 같다.

**플레이어 동기화(PlayerPacket & NetworkPlayer)**

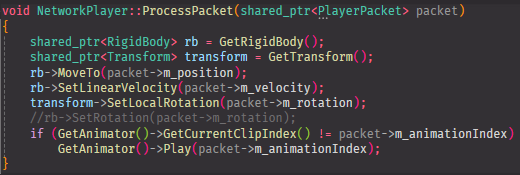
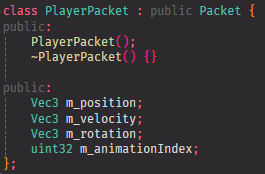
가장 중요한 플레이어의 동기화를 준영이와 같이 작업했다. 플레이어는 키 입력을 같이 처리해야 하기 때문에 기존의 Player 클래스 내에서 분기를 하거나 추가 처리를 하는 것이 아닌, NetworkPlayer 클래스를 별도로 만들어서 동작하도록 했다.

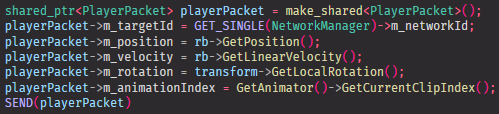
PlayerPacket은 위치와 속도, 회전, 애니메이션 인덱스를 가지고 있고 이를 네트워크 플레이어의 ProcessPacket() 함수를 호출 함으로써 받아온 패킷으로 움직임을 동기화할 수 있게 하였다.

추후 다른 객체들의 네트워크 처리도 비슷하게 ProcessPacket이란 함수를 통해 진행하게 될 것 같다.

패킷을 발생시키는 부분은 기존 Player 클래스에서 다른 모든 정보가 업데이트 된 후인 LateUpdate()에서 만들어 Send하게 하였다.

이렇게해서 각 클라이언트는 자신이 조종하는 플레이어와 다른 플레이어의 움직임에 물리적인 처리를 하고 위치와 애니메이션만 동기화해주는 방식으로 네트워크 구성을 하였다.

****

**▶ [박준영]**

지난주에 만든 Active On/Off 기능 사용 시 인스턴싱을 통한 렌더부분에서 문제가 발생하는 것을 확인하여 Active 상태 전환 대신 사용하지 않는 게임오브젝트는 보이지 않는 위치로 옮기고, SetStatic을 On하여 물리적 상호작용을 막는 방법을 선택하였다.

또한 총알에 적 객체 충돌시 실제적인 데미지 부여가 일어나며 이에 따라 적 객체의 HP가 0이하가 되었을때 사망 애니메이션 재생 + 물리적 상태를 overlap 상태로 변경하여 물리적 상호작용을 막고 사망 애니메이션이 끝나면 객체를 삭제하도록 하였다.

임시로 테스트중이던 네트워크 연결을 통한 플레이어의 이동을 NetworkPlayer 클래스를 통해 전담하도록 하는 작업을 순원이와 함께 작업하였다.

현재 순원이가 클라이언트간 연결과 패킷 송수신 프레임워크를 만들어 놨기 때문에 새롭게 PLAYER패킷을 만드는 작업과 해당 패킷을 수신하여 작동할 NetworkPlayer 클래스만 정의하여 작동시킬 수 있었다.

각 유저들은 다른 플레이어가 보낸 PLAYER 패킷을 수신받게 되는데, 이 때 해당 패킷에는 플레이어의 Position, LinearVelocity, Rotation, 현재 재생중인 애니메이션 클립의 인덱스를 받게된다.

NetworkPlayer는 해당 정보를 받아 다른 플레이어의 화면에서 해당 플레이어가 움직이는 것을 보여주는 역할을 한다.

기존에는 단순히 패킷을 받을 때 마다 Position을 덮어쓰는 식으로 작동을 했다. 그래서 실제로 플레이어가 움직이는 상황에도 다른 플레이어의 화면에서는 패킷을 받을때만 그 위치 정보가 변경되기 때문에, 게임의 FPS가 120이 나올 경우 다른 플레이어는 최소 1프레임 동안 가만히 있고 이후 이동하는 방식으로 업데이트 되기 때문에 다른 플레이어의 이동이 버벅이거나 부드럽지 못한 문제가 있었다.

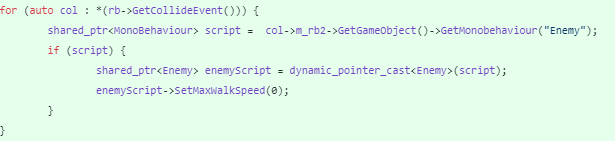
그래서 패킷 내용물에 선형적 속도 또한 추가하여 패킷을 받지 못한 프레임에선 마지막으로 받은 패킷에서의 선형 속도에 델타타임을 곱해 이동을 수행하도록 하였다.

이렇게 되면 패킷을 받지 못한 프레임 도중에 다른 플레이어가 이동 방향등을 수정하면 약간의 오차가 발생할 수 있으나, 실제 적용한 결과 매우 부드럽고 자연스럽게 작동하는것을 확인할 수 있었다.

추가적으로 중간 발표에 대비하여 데모 시연 시 이동할 플레이어의 동선에 맞춰 맵 오브젝트들을 더 배치 하였다. 화면 중앙부근에 밟고 이동가능한 객체들을 다양하게 구성하여 우리 게임의 물리적 상호작용을 보여줄 수 있게 하고, 동선 중간에는 처치 가능한 적 객체, 공격 타입을 폭탄으로 바꾸는 아이템 객체, 폭탄을 통해 폭발시켜 날아가는 것을 시연할 수 있는 충돌 가능 객체들을 배치할 예정이다.

**▶ [엄장헌]**

기존의 충돌 정보를 받아올 때 대상의 RigidBody만 가져온다. 따라서 물리적인 이벤트만 넘겨줄 수 있었는데, 대상에게 데미지를 주거나, 이벤트를 발동시킨다던가 하려면 그 컴포넌트를 직접 사용가능해야 한다. 따라서 상대 리지드 바디를 통해 내가 원하는 컴포넌트의 shared\_ptr값을 받아오는 함수를 만들었다.



언리얼에서 cast to 함수를 이용하는 것처럼 받아온 shared\_ptr<Monobehaviour>을 원하는 컴포넌트로 캐스팅해줘야한다. 이후 대상의 퍼블릭 함수에 접근할 수 있게된다.

RigidBody 속성으로 Overlap을 추가했다. 충돌 정보는 원하지만 block되는 것은 원하지 않는 경우가 있다. 예를 들어 아이템을 먹을 때나 이벤트 발생용 트리거를 만들 때 필요하다.

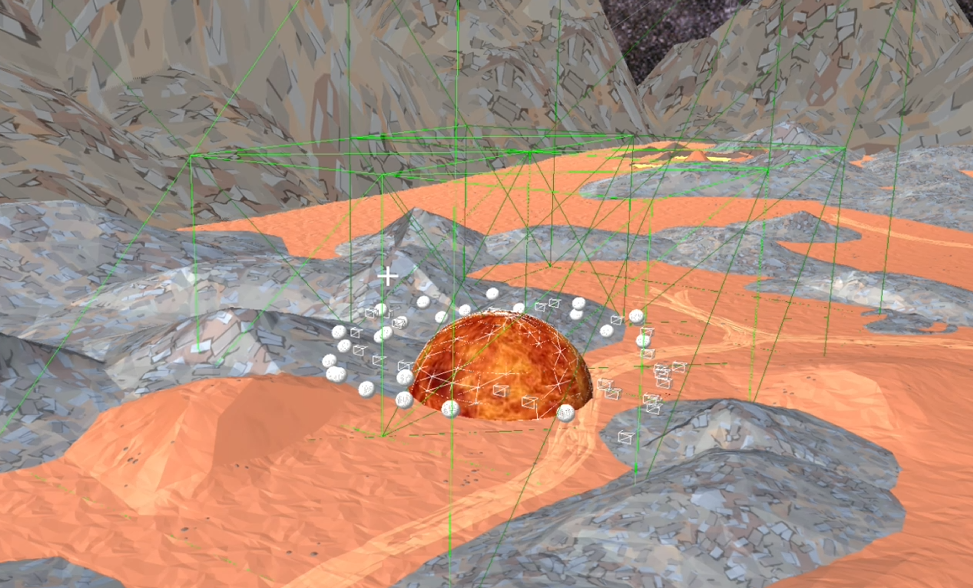
RigidBody로 너무 높은 delta\_time이 넘어올 때가 있다. 게임이 focus\_out되거나 로딩을 할 때이다. 이때 물리처리에서 문제가 생긴다. 따라서 너무 큰 delta\_time, 지금의 프로젝트에서는 0.5초 이상의 시간은 무시하게 변경했다.

Scene을 설정할 때 하나하나 설정하는 코드가 굉장히 길어진다. 따라서 게임 오브젝트를 리소스 코드에 몰아서 만들고 간단하게 불러올 수 있도록 프리펩화 하였다. 아직 모든 객체에 적용된 것은 아니고 일부에 적용했다.

원래 40~50라인을 차지하던 부분을 재사용이 쉽게 변경하였다.



폭발에 임시로 텍스쳐를 입혔다. 차후 파티클 효과나 다른 이펙트를 만들 수 있다면 수정할 것 같다. 또한 폭발이 최대 범위로 커졌다가 갑자기 사라지는 부분이 어색했다. 시간에 따른 크기변화를 sin함수를 이용해 자연스럽게 커졌다 작아지게 만들었다. 또한 폭발 크기, 폭발 속도, 폭발력 등의 세부적인 조절이 가능하게 기능을 추가했다.



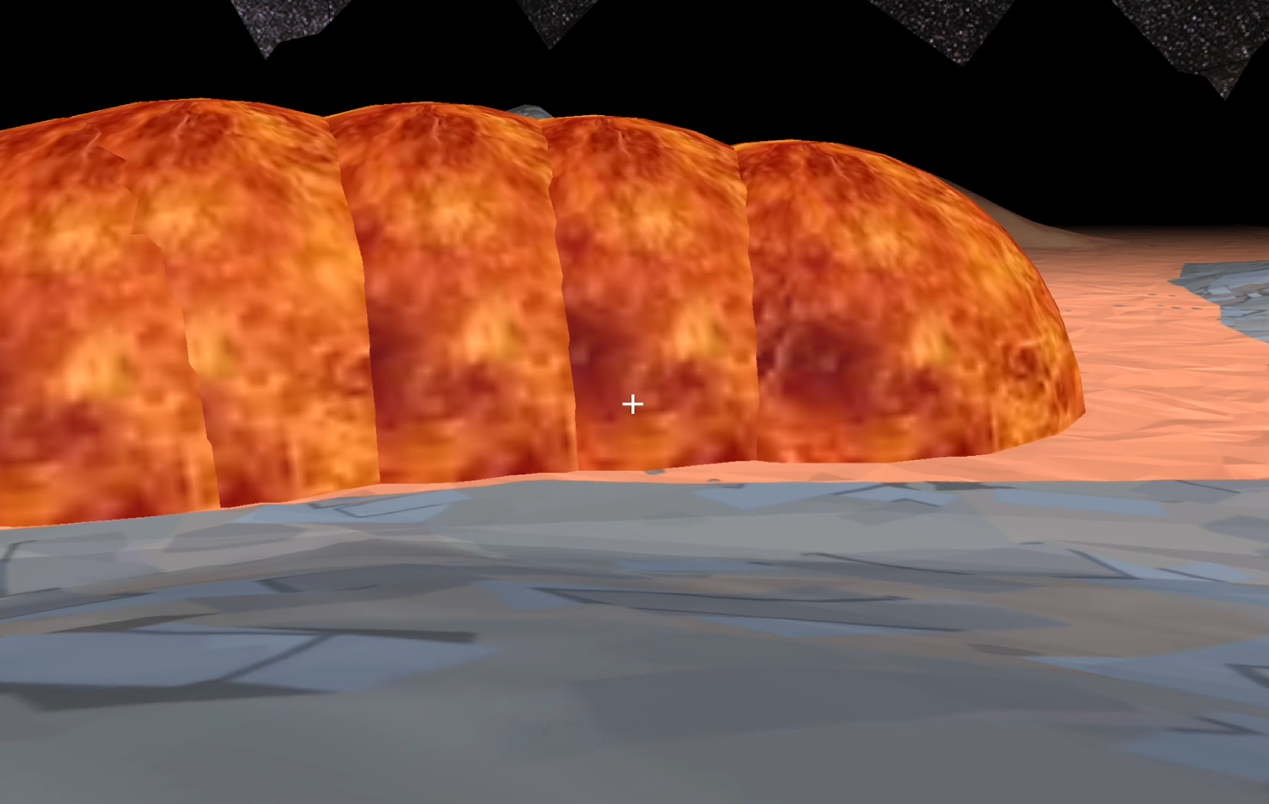
씬매니저에 묶여있던 씬정보들을 분리했다. 또한 로딩 씬을 따로 만들었고 유저가 키를 눌렀을 때 로딩이 시작되어 메인 씬으로 전환된다.



총알에 폭발 기능을 내장했다. 총알에 타입을 enum으로 정의하고 fire함수에 총알 타입을 넘김으로 작동되게 하였다.



모든 총알의 타입을 폭발로 선택하고 사격했을 때의 모습이다.



임시적으로 총구가 플레이어의 조준을 따라가기 위해 플레이어의 pitch회전 제한을 풀었다. 차후 필요하다면 상하체 분리 애니메이션으로 수정할 예정이다.