| **비단길(Silk Road)** | |
| --- | --- |
| **6주차** | **2024.01.28 (일) ~ 2024.02.03 (토)** |

**■ 주간 회의록**

**▶ 패킷**

* 서버와 호스트, 각 게스트들이 주고 받을 패킷의 형식이 제대로 정해지지 않아 네트워크 프레임워크를 제작하는데 어려움이 있음
* 하지만 아직 패킷 내용 결정에 큰 영향을 미치는 물리엔진이 완성되지 않음
* 간단하게 플레이어들의 위치만 주고 받는 서버로 일단 만들고 패킷 내용이 결정되고 이후 하나씩 수정해 나아가자는 의견이 있었고 모두 동의함

**■ 다음 주 세부 계획**

**▶ [권순원]**

* 주간 회의 내용에 맞게 네트워크 완성
* 네트워크 완성 시킨 후 2월부터 하기로 계획되어 있던 조작 관련 구현

**▶ [박준영]**

* unity에서 구현한 맵 정보 불러오기

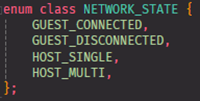
**▶ [엄장헌]**

* 옥트리 갱신 및 객체 삭제시의 노드 삭제
* 육면체 콜라이더 추가후 충돌 테스트

**■ 작업 일지**

**▶ [권순원]**

이전에 작업하던 Server class의 이름을 Network로 수정했고 제작을 완료했다.  
일단 네트워크 상태를 4가지로 정의했다. 플레이어가 싱글플레이를 하는 경우, 호스트 플레이어가 멀티플레이를 생성한 경우, 그리고 게스트 플레이어가 호스트 서버에 접속중인/접속이 끊긴 상태로 분류했다.



기본적으로 게임을 처음 시작할 때 상태는 HOST\_SINGLE이다. 해당 플레이어가 상호작용을 통해 멀티플레이 방을 생성하면 HOST\_MULTI 상태로 변경되고 서버입장코드가 발급되고 서버의 역할을 하게된다. 두 가지GUEST\_ 상태는 말 그대로 게스트로 호스트 서버에 접속한 상태이다. \_CONNECTED 상태는 연결이 정상적으로 유지되며 게임을 하고 있는 상태이다. 이 상태에 있는 클래스는 지속해서 서버와 패킷을 주고 받으며 게임이 진행된다. 서버입장코드를 입력하고 접속을 시도하는 중이나, 연결 중에 문제가 생겨 연결이 끊기는 경우(호스트의 인터넷이나 게임에 문제가 생겨 연결이 끊기는 경우도 포함해서) \_DISCONNECTED 상태가 되며 다시 접속을 몇 번 시도하다가 실패하면 HOST\_SINGLE 상태로 변경되게 할 것이다.

Network 클래스는 코드 전반에 걸쳐 활용될 여지가 있다고 생각했다. 때문에 프로젝트 다른 클래스에서 사용된 싱글톤 패턴을 그대로 가져왔다. GET\_SINGLE() 매크로를 통해 프로젝트 어디서든 활용이 가능하게 만들었다.

클래스에 쓰레드 함수들이 존재하고 클래스 내에서 공유될 수 있는 변수들은 두 가지 방법으로 동기화를 계획했다. 첫 번째는 이전부터 사용한 LockQueue이다. LockQueue는 네트워크 클래스의 쓰레드 뿐만 아니라 메인 쓰레드와의 동기화에도 사용할 것이다. 두 번째는 atomic<T>이다. Atomic이 사용될 변수들은 작동 상태를 저장할 bool 변수나 위에 정의한 상태를 나타내는 enum class 등이다. 복잡한 기능을 하거나 쓰기/읽기 많지는 않지만 여러 쓰레드에서 같이 쓰일 여지가 있는 변수들은 atomic으로 사용하는게 좋을 것이라고 생각했다. 하지만 여지가 있는 것이지 지금까지 내가 생각한 논리 상에서는 동시에 사용될 일이 없기 때문에 만약 이로 인해 문제가 생기거나 성능에 저하 등이 발생한다면 이 부분또한 mutex등으로 변경할 것이다.

함수들은 이번주 주간회의까지도 패킷 정의 때문에 갈팡질팡하며 썻다 지웠다를 반복하고 있었다. 아직 물리엔진이 아직 어떤 방식을 사용할지 여러가지 테스트와 시도중에 있기 때문에 플레이어들과 서버 사이에 주고받을 데이터가 정확히 정해지지 않았기 때문이다. 때문에 이번 주간 회의에서 얘기를 해보았는데 일단 간단하게 서로의 위치 정보를 보내 갱신하는 방식으로 제작을 하고 물리엔진이 완성되는대로 하나씩 변경해보자는 의견이 나왔다.

완성되는 시기까지 기다려서 작업을 하는 것보다는 끊기는 모습이더라도 위치라도 변경되는 기능을 구현하고 하나씩 수정해 나아가는 것이 작업에 있어 더욱 유리할 것이라고 모두 생각, 그렇게 구현하기로 결정했다.

위 사항을 기반으로 작업에 들어갔다.

현재 서버의 주요 쓰레드를 실행 정지 시키는 기능을 완성하였다.

그리고 멀티 플레이를 생성하면 게스트의 접속을 받을 수 있는 listen socket을 유지시키기 위한 대기 쓰레드를 실행하는 기능을 완성했다.

번외로 C++20 으로 작업하면서 bind함수가 전혀 다른 함수인 std::bind로 동작하여 문제가 발생했다. 현재까지 프로젝트에서 using namespace std를 선언해 사용해서 생긴 문제였다. 이제와 모든 코드를 수정할 수는 없기 때문에 ::bind로 네임스페이스가 std가 아니라고 명시적으로 표기해 해결했다. 혼자 작업할 때는 using namespace를 사용하지 않았기 때문에 이런 문제가 없었기 때문에 조금 헤맸다.

**▶ [박준영]**

간단한 버튼 기능을 구현하였다.  
테스트용으로 아직 텍스쳐 파일을 통해 눈에 보이는 버튼은 아니며, 정해진 영역 내의 클릭을 감지하고, 버튼 객체에 저장된 함수 포인터를 통해 지정된 함수를 호출하는 방식으로 작동한다.

중간발표때는 실제 버튼 표현과 버튼에 마우스 클릭, 클릭중, 클릭을 중지 했을 때를 나누는 기능도 구현할 예정이다. (버튼을 누르고 있으면 꾹 눌린 모습으로 바뀐 뒤 릴리즈 시 해당 버튼의 기능이 동작하도록)

또한 게임 내 업그레이드 및 서버 접속 등에 사용될 팝업 창을 구현하기 위해 스택구조로 되어있는 팝업 매니저를 만들었다.

이후는 인게임 객체들 표시 및 네트워크 기능 구현이 우선적으로 되어야 한다고 생각하여 유니티를 통해 오브젝트들을 export하는 방법을 계획하고 스크립트를 짜보았다.

기본적인 지형 같은 경우 물리엔진을 처리하고 있는 장헌학생이 현재 다양한 수정을 거치는 터라 차치하고 우선 그 외의 맵 내 오브젝트들을 배치 후 불러오는 구조를 짜고있다.

현재 각 맵별로 필요한 맵 내 오브젝트라 하면 몬스터 생성지점, 아이템 상자 생성 지점, 플레이어 스폰 포인트, 보스 소환 게이트 등 후보가 될 지점의 위치 정보가 필요한 오브젝트 들이다.

각 맵마다 이것들을 미리 유니티에서 배치해두고 이 오브젝트들을 하나의 오브젝트 아래로 두어 상위 오브젝트에 스크립트를 적용 후 실행시키기만 하면 산하 오브젝트들의 모든 위치, 크기, 회전 정보와 오브젝트 타입, 기타 특수 속성(예: 어떤 종류의 상자가 나오는지, 이 지점에선 어떤 몬스터가 몇마리 나오는지)를 담는 JSON파일을 내보내는 것을 구현중이다. JSON을 많이 써보지 않아서 다시 공부를 하긴 했지만 메쉬 내보내기 보단 쉽게 구현이 가능할 것 같다.

**▶ [엄장헌]**

scene에 그리기 위해 오브젝트를 추가하는 기존 함수에서 콜라이더 컴포넌트가 있는지 확인하고 있다면 충돌가능한 게임 객체들을 모아두는 Vector를 따로 추가했다.

옥트리와 노드 클래스를 만들어서 광역탐색을 위한 기반마련

각 노드는 부모 노드와 8개의 자식 노드를 가진다.

객체를 트리에 삽입하는 함수에서는 root 노드부터 내려가며 현재 노드에 완전히 포함되는지 확인을 하며, 현재 노드에 자식이 없다면 생성하는 식으로 진행,

조건에 만족하는 위치를 찾으면 해당 위치에 저장한다.

충돌검사를 할때에는 자신의 노드 위치에서의 부모 자식 노드와 충돌검사를 진행한다.

대략적으로 위의 정도는 구현이 되었다. 다만 아직 구현 안된 부분이 객체가 이동하거나 삭제될때마다 자식 노드들을 확인하여 비었다면 해당 노드를 지우는것과 이동하는 프레임마다 지속적으로 트리를 갱신을 해줘야 할텐데, 움직이는 객체가 고정된 객체보다 훨씬 많아지면 갱신에대한 오버헤드가 크지않을까 걱정된다. 물론 우리 게임의 경우 꽤나 많은 자연물과 장애물이 있고 기획단계에서 맵을 넓게 설정했기때문에 효율이 있을거라고는 생각하지만, 테스트는 필요해보인다.

addforce를 이용해 테스트를 하던 중 중력, 마찰, 충돌, 이동 등 하나에 객체에 들어가는 계산이 많아지는데 addforce의 계산의 순서 문제인지 제대로 가속이 들어가지 않는 부분이 확인되었다. 이것저것 확인하며 물리엔진에 대한 영상을 구경하다가 물리량을 계산하는 부분에서 verlet 알고리즘에 대한 영상을 보고 2시간정도 공부하며 테스트해봤다.. 속도 대신에 위치의 변화를 중점으로 계산하는 방식이였는데, 구현을 하던 도중 입자 시뮬레이션에 최적화된 방식이라는 것을 알게 되었다.. 나중에 파티클에 써먹을 수 있으면 한번 도전해 봐야겠다.