| **비단길(Silk Road)** | |
| --- | --- |
| **9주차** | **2024.02.18 (일) ~ 2024.02.24 (토)** |

**■ 다음 주 세부 계획**

**▶ [권순원]**

* 이번주에 계속 진행하고 있던 네트워크 부분 계속해서 테스트
* 다음 주 회의에서 얘기할 개강 후 작업 계획 세우기

**▶ [박준영]**

* 길찾기 알고리즘 구현
* fbx 경량화

**▶ [엄장헌]**

* 터레인과의 충돌
* 옥트리에서 노드의 갱신 버그 수정

**■ 작업 일지**

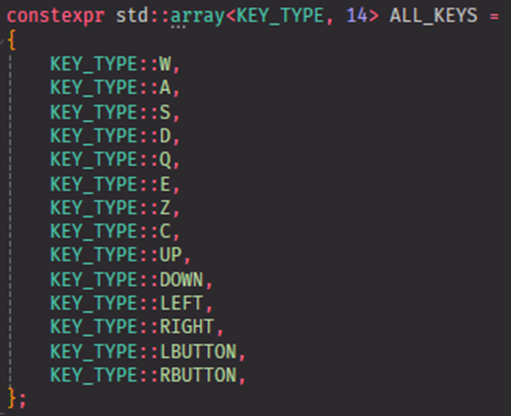
**▶ [권순원]**

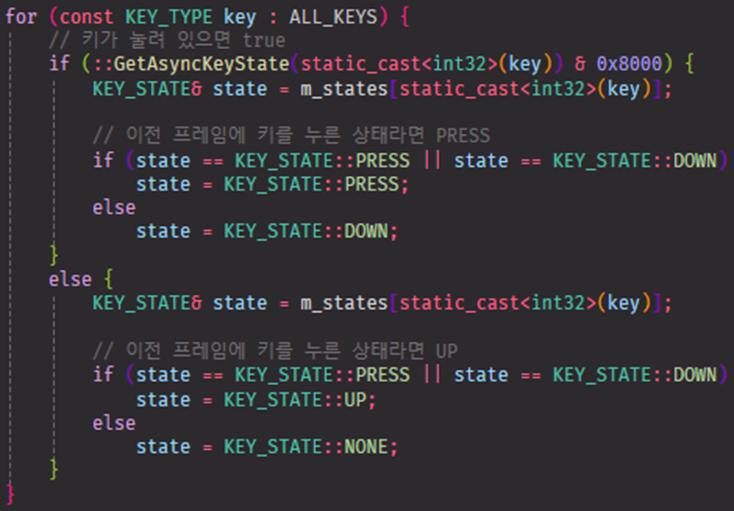
우리가 이전 방학 동안 만든 프레임워크의 키 입력(Input) 부분에서 문제가 하나 있었는데 프레임에 따라서 키 입력이 밀리는 현상이었다.

현재 코드에서 키 입력 상태를 가져오는 방법으로 GetKeyboardState 함수가 쓰였다. 이 함수는 현재 키보드의 모든 입력 상태를 가져오는 함수이다. 때문에 입력에 사용되지 않는 키보드 입력까지 같이 처리해서 문제가 생기는 것이라고 일단 생각하고 수정작업을 해보았다.

사용하지 않는 모든 키 입력도 검사하고 상태를 확인하는 것이 문제라고 생각하였으니 GetKeyboardState를 대체할 수 있는 함수를 찾아보았다.

GetKeyState와 GetAsyncKeyState 두 개의 대체 함수를 찾을 수 있었다. 일단 두 함수는 모두 매게변수로 상태를 알기 원하는 키를 전달하고 SHORT 형식으로 값을 반환해준다. 두 함수의 차이는 Async가 붙은 함수는 말 그대로 비동기적으로 처리해 준다는 뜻이다. 때문에 현재 키의 상태를 즉시 반환해 더 정확한 타이밍에 키 입력을 처리할 수 있다고 한다. 대신 비동기 함수는 caps lock 등의 부가적인 정보를 반환 받을 수 있다는 점이 있다.

일단 게임에서는 사용자 입력에 대한 빠른 반응이 더 중요하다고 생각했고 이미 기존 코드에 KEY\_STATE 를 통해 입력 상태를 구분하는 코드를 작성했었다. 그리고 caps lock 여부와 같은 부가적인 정보는 우리 게임에는 필요가 없다고 생각하여 일단 비동기 함수를 이용해 기존 코드를 수정하였다.

일단 게임 내에서 사용하는 키들만 반복문을 통해 접근하기 쉽도록 ALL\_KEYS 라는 배열을 하나 생성하였다.  
이런 식으로 사용하여 기존의 모든 키를 순회하는 방식 대신 진짜 사용하는 키만 처리하도록 변경했다.

상태를 변경하는 기존 코드는 누고 for문과 키 상태를 가져오는 부분을 수정했다.  
해당 코드를 수정한 뒤에 테스트를 해보았을 때 문제가 발생하지 않았다. 추가로 오브젝트를 많이 생성하여 밀림이 심한 상태를 만들어 놓고 테스트를 해보았는데 밀림이 발생하지 않았다.

**▶ [박준영]**

맵데이터 내보내기/불러오기는 완료했으며 맵데이터의 각 오브젝트의 메시 id(현재 임의의 값)를 통해 오브젝트에 알맞은 메시를 적용시키고 위치, 크기, 회전 정보에 따라 scene에 배치하는 것까지 완료되었다. 아직 맵 구현 세부사항 (바닥을 터레인 or 오브젝트 배치)이 완료되지 않아 시각적으로 구현되진 않았다

팀 전체적으로 구현할 부분이 완료될 때 까지 개인작업이 붕 뜬 느낌이 있어 조금 앞서지만 몬스터 인공지능 구현 관련된 기반을 개발중이다.

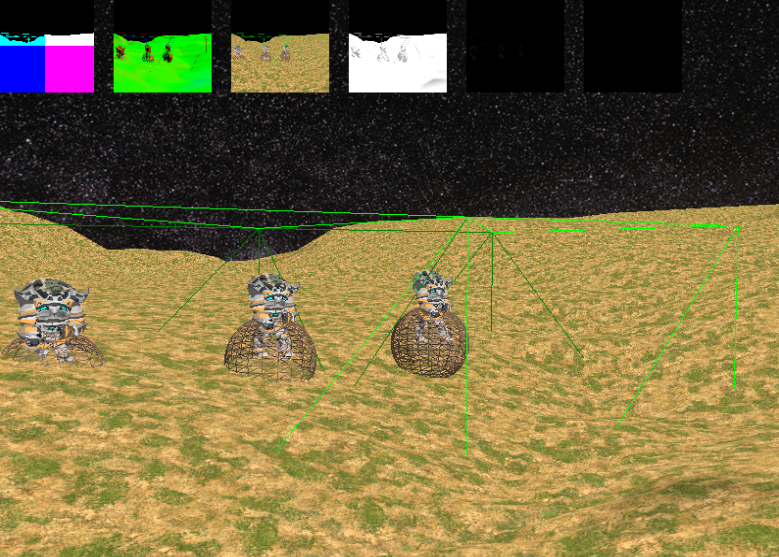
길찾기 알고리즘이 리소스 개발(맵 배치)에 관련되며 가장 핵심적이라고 판단되어 우선적으로 개발중이다.

기본적으로 일반 근거리 지상 몬스터는 스폰될 경우 가장 가까운 플레이어를 타겟으로 하여 chasing상태가 되어 플레이어를 향해 이동한다. 현재 고려중인 예외 상황은 두가지 이다. 하나는 추적도중 플레이어와 몬스터 사이의 직선 경로 내에 장애물(건물, 벽)이 존재하는 경우, 또 하나는 플레이어가 몬스터의 인지 범위(지름 20m) 바깥으로 나가는 경우다. 전자의 경우는 unity에디터 상에서 맵 제작 시 지면에 이동경로 노드들을 구성하여 A\* 알고리즘을 통해 이를 우회하는 방법으로 구현중이다. (연산 속도에 따라 다른 알고리즘을 적용해야할 수 있다..) 후자의 경우 몹의 스폰지점으로 돌아가는 방법과 chasing 상태가 끝난 지점에 서서 범위 내에 다시 플레이어가 들어올 때 까지 기다리는 방법이다.

현재는 두번째 방법을 사용할 예정이며 적용 했을 때 움직임이 너무 부자연스럽지 않다면 그대로 확정할 것이다.

현재 순원이는 네트워크, 장헌이는 물리 관련 구현에 힘쓰는 중이니 오랜 숙제였던 fbx파일 경량화는 내가 병행해서 진행하는게 좋을 것 같다.

**▶ [엄장헌]**



옥트리로 공간 분할을 성공하였다.

이전의 문제는 콜라이더의 위치와 크기를 Awake 단계에서 진행하였다.

하지만 처음 옥트리를 만드는 것은 sceneManager에서 게임 객체를 자료구조에 넣는 순간 만들기 때문에 모든 콜라이더의 위치가 (0,0,0) 으로 고정되어 있었다.

순서를 조금 바꾼 뒤 해결되었다고 생각했다

위치를 바꾸면서 실험 중, 내 코드는 분명 옥트리의 각 노드 공간에 contain 된 collider만 저장했는데 자세히 보니까 Intersect된 collider까지 저장하고 있었다.

if (currentNode->GetBB()->Contains(\*(sphereCollider->GetBoundingSphere())))

이 부분이다.

한참 버그를 찾다가 내가 Contains 함수에 대한 이해가 부족했나 라는 생각으로 DirectXCollision내부로 들어가 코드를 하나하나 읽어봤다.

return (XMVector3EqualInt(InsideAll, XMVectorTrueInt())) ? CONTAINS : INTERSECTS;

확인해 보니 삼항연산자로 CONTAINS->2 INTERSECTS->1 이렇게 반환해주었기 때문에 if문이 intersect에도 참으로 돌아간 것이였다…

이번 건으로 내가 처음 써보는 함수는 대충 이름으로 이런 기능이겠거니 추측하지말고 한번은 자세히 읽어보는게 좋은 습관이란 것을 깨달았다.

결과적으로 사진과 같이 게임객체가 있는 공간만 옥트리의 노드로 활성화되었으며 사실상 부모 노드도 있긴하지만, 일단은 게임 객체를 가지고 있는 노드만 그려지게 하였다.

이전 3D게임프로그래밍에서 만들었던 터레인의 코드를 조금 바꾸어 적용을 완료했으며 아직은 높이값을 따로 저장하지 않고 쉐이더로 넘겨주어 그리기만 하기때문에 물리적인 처리는 적용되지 않았다.