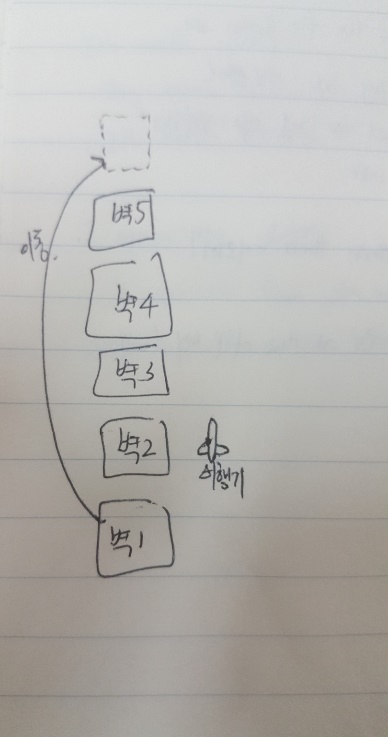
보고서에서는 GameFramework에 기본적으로 있는 내용을 제외하고 제가 구현한 부분을 위주로 서술하였습니다.(직접 구현하기 위해선 GameFramework의 이해를 했다고 가정할 수 있기 때문)

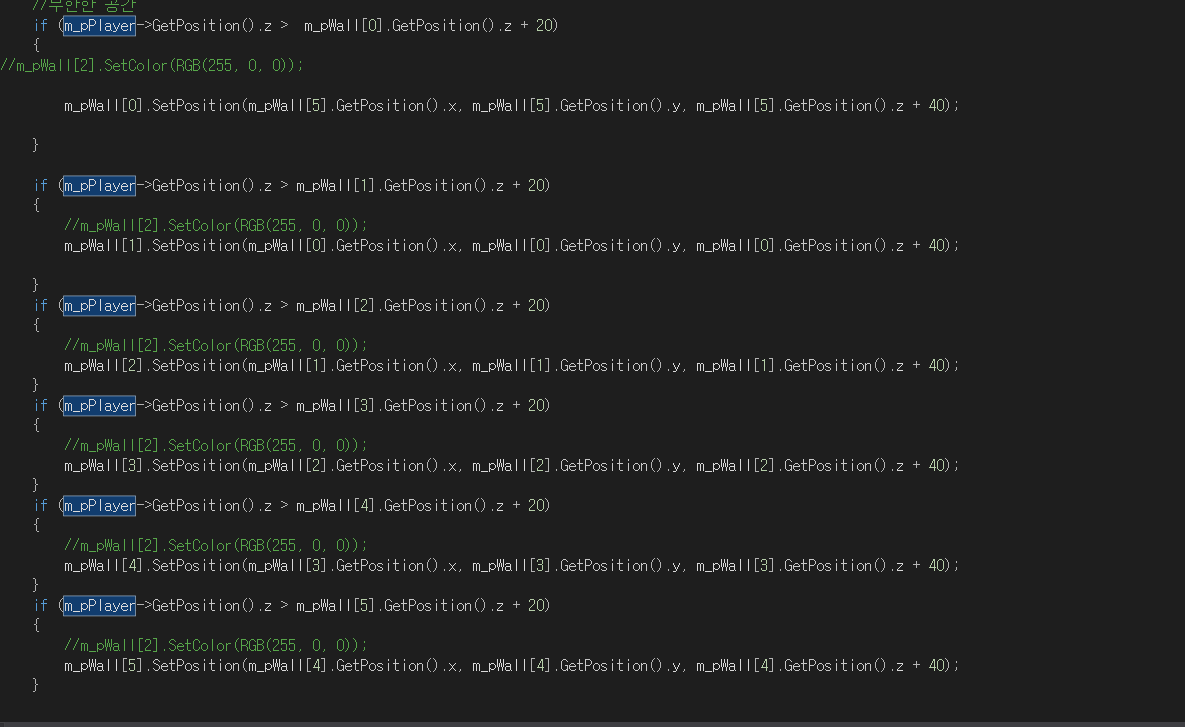
1. **무한한 벽 만들기**

**가정**

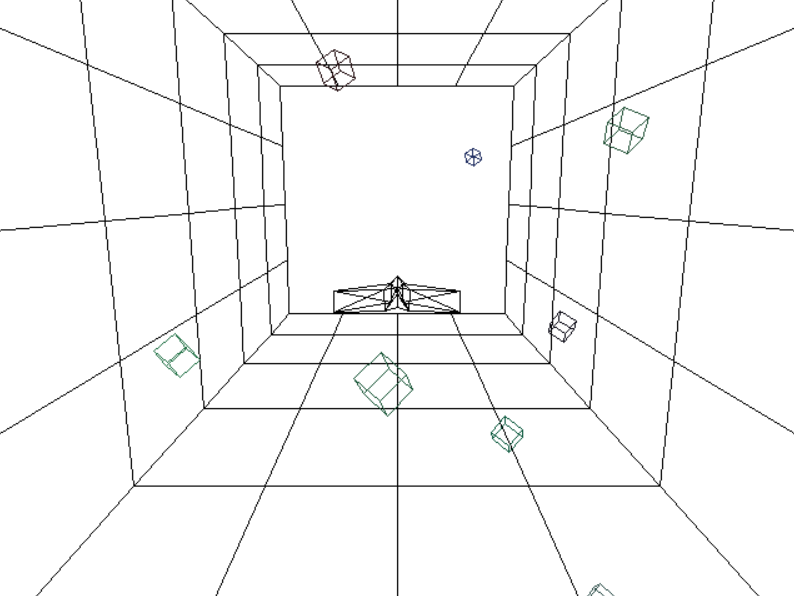


비행기의 Position의 Z값 > 벽들의 Position의 Z값을 비교하여 작은 벽들은 뒤로 밀어주면서 무한한 벽의 효과를 구현하게 가정 했습니다.

**실행**



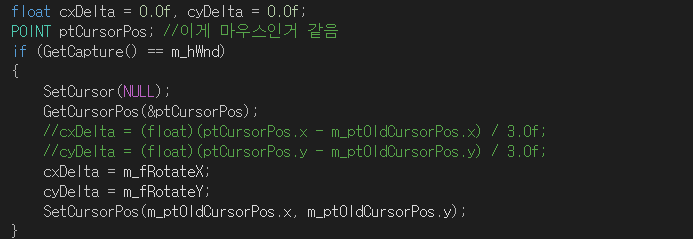
벽의 Z좌표가 비행기의 Z좌표보다 작다면 ( 비행기보다 벽이 더 뒤에 있다면) 맨 끝의 벽 뒤에 붙여주도록 하였습니다. 위와 동일한 방식으로 Object(적) 객체들도 비행기의 Z좌표보다 작다면 앞으로 밀어주면서 무한하게 생성되도록 구현하였습니다.



<그림 : 무한한 벽>

1. **카메라 각도 회전**

**가정**

****

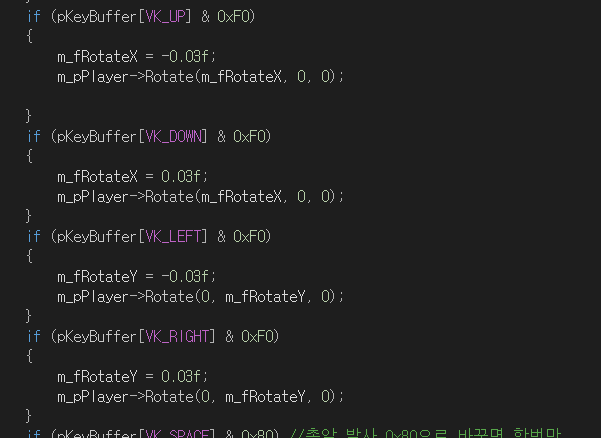
기존에 있던 코드에서 카메라의 회전이

cxDelta = (float)(ptCursorPos.x - m\_ptOldCursorPos.x) / 3.0f;

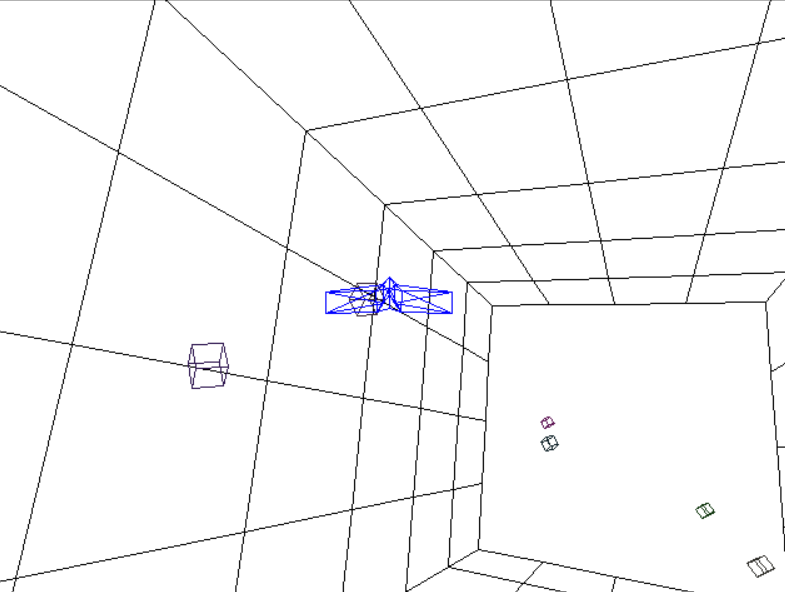
cyDelta = (float)(ptCursorPos.y - m\_ptOldCursorPos.y) / 3.0f;

마우스로 입력받은 x,y 좌표로 이루어졌는데 과제하면서 많이 불편해서 방향키로 회전을 하면 좋겠다고 생각했습니다. (회전 각을 Rotate함수로 넘겨주면 Rotate함수 내부에서 카메라도 회전시켜주고 Look과 Up도 새로 셋팅 해주어서 더 이상 손댈 곳이 없었음)

**실행**

****

키 입력에 따라 회전각도를 변화시키도록 구현했습니다.

****

<그림 : 방향키를 눌렀을 때 카메라의 회전>

1. **Object와 벽의 충돌처리 & Object 끼리 충돌처리**

**가정**

Object가 움직일때마다 OOBB가 새로 셋팅되어야 된다고 생각했습니다. Animate()함수에서

m\_xmOOBBTransformed.Transform(m\_xmOOBB, XMLoadFloat4x4(&m\_xmf4x4World));

XMStoreFloat4(&m\_xmOOBBTransformed.Orientation,XMQuaternionNormalize(XMLoadFloat4(&m\_xmOOBBTransformed.Orientation)));

부분에서 OOBB가 옮겨진다고 생각했습니다.

**실행**

GameFramework 클래스를 파헤쳐 보았는데 벽과 Object의 충돌은 코드 전반적인 내용을 살펴보았을 때 제가 따로 손댈 부분이 없었습니다.

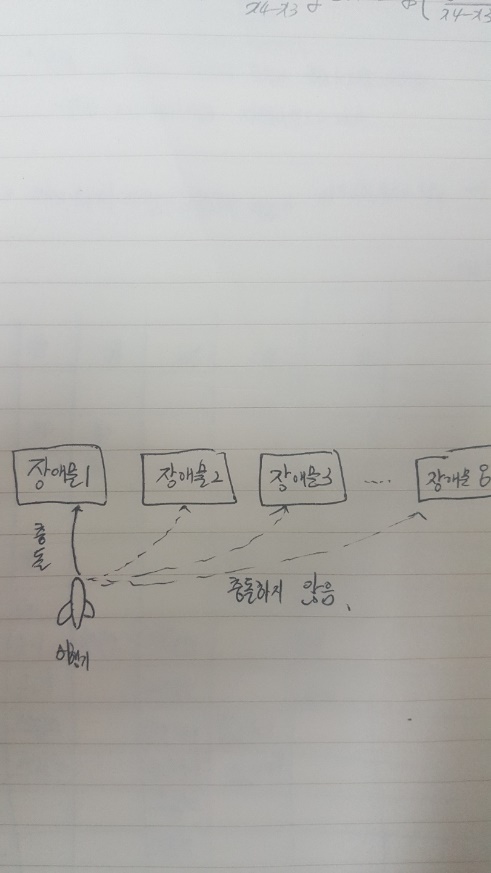
(하지만 위의 충돌부분을 이해하는데 노력했고 이를 바탕으로 비행기와 벽 충돌, 비행기와 장애물 충돌, 총알과 장애물 충돌, 총알과 벽면 충돌 등 충돌부분에서 많은 아이디어를 얻을 수 있었음)

1. **적을 적당히 피해가고 피하지 못하면 죽음**

**가정**

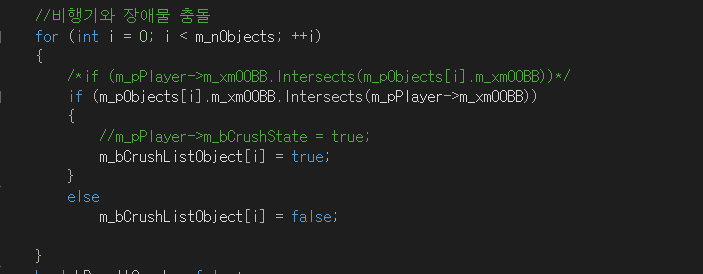
장애물과 부딪히면 장애물들과 비행기를 원점으로 돌리게 하도록 하면 좋겠다 생각했습니다.

그리고 이 부분에서 가장 많은 시간을 할애했던 부분은 장애물 충돌상태를 bool 변수로 선언을 하여 충돌 시 True, 충돌 안하면 False로 저장했는데 이를 **하나만** 선언 했다는 것 입니다

.

각 장애물 마다 for문을 돌며 충돌 시 bool변수 상태를 바꿨는데 이는 장애물1과 비행기가 충돌할 경우 장애물 2,3,4,5,6,7,8과는 충돌하지 않았기 때문에 장애물과 충돌을 하더라도 false값이 나오게 되었습니다. (장애물 충돌 하면서 좌표계의 문제인 건가 하는 생각도 했지만 말도 안되는 가정 이였음)

**실행**

****

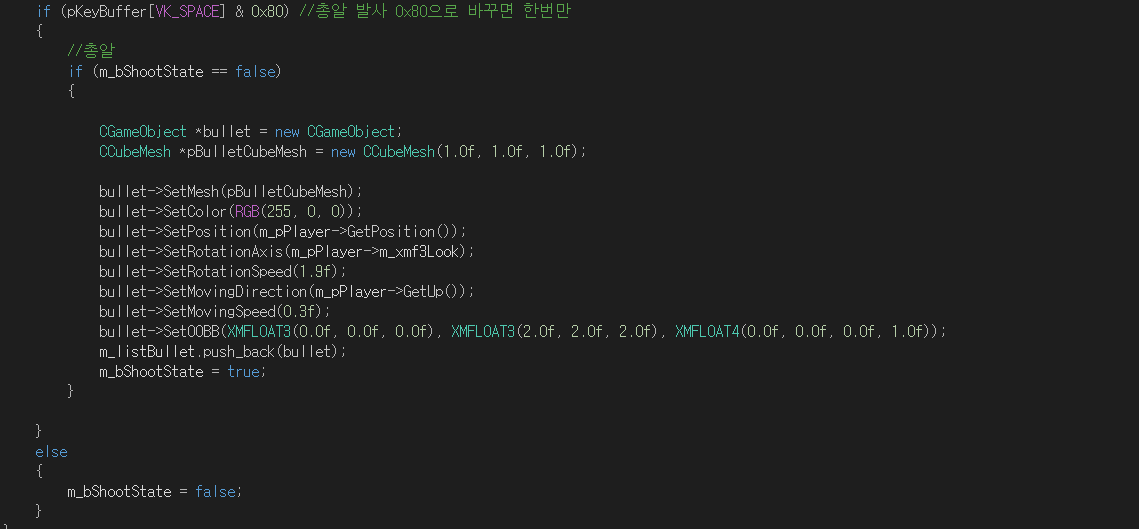
그래서 각 장애물마다 충돌을 체크하도록 bool변수를 배열로 선언하여 충돌검사를 하였습니다.

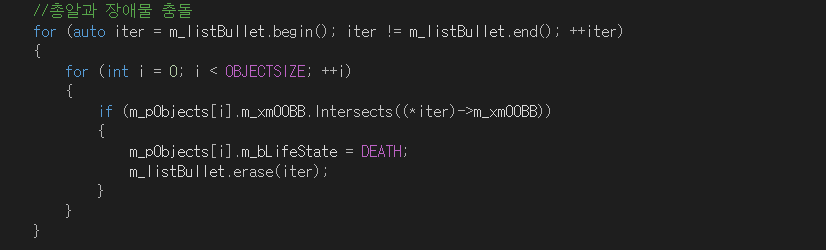
1. **총을 쏠 수 있어야한다. (회전, 파티클)**

**가정**

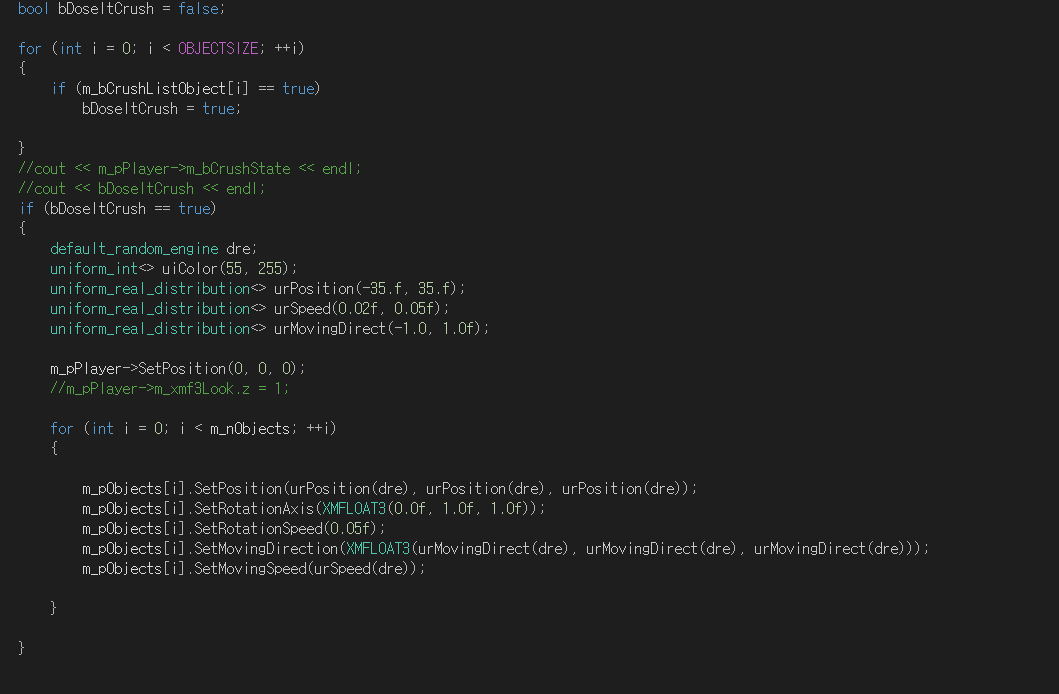
총알의 위치는 비행기의 좌표여야 하고, 총알의 회전축은 비행기가 바라보고 있는 Look벡터로 가정하였고 그대로 실행하였습니다.

**실행**

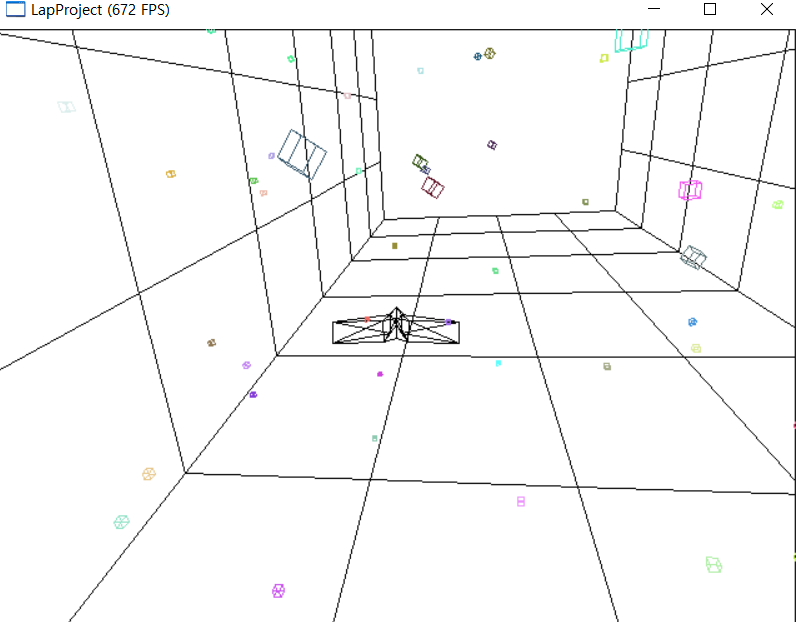
****

****

다만 문제는 Space키를 누르면 총알이 발사되도록 하였는데 총알이 1발씩 나가야되는데 여러발씩 나가게 되었습니다. 그래서 따로 m\_bShootState 변수로 총알을 쐈을 때 한발만 나가도록 구현했습니다.



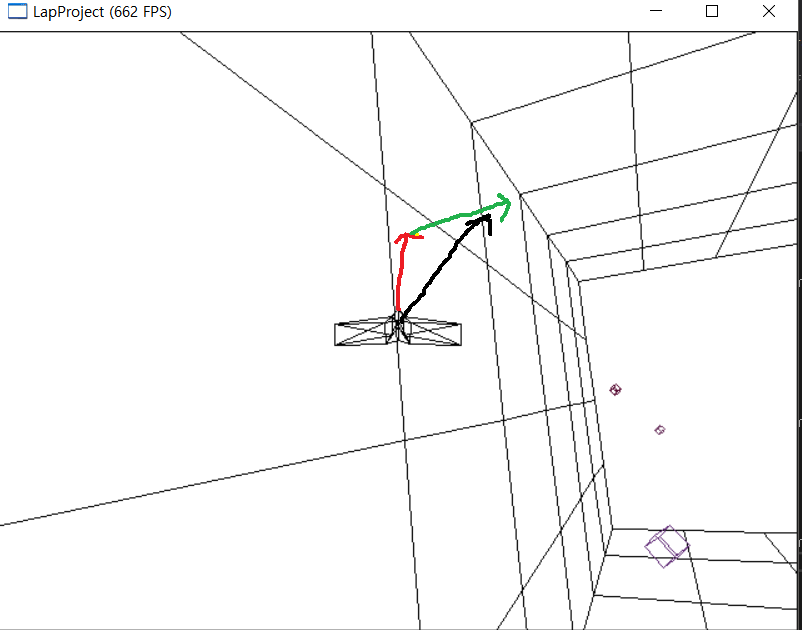
총알이 장애물과 부딪히면 파티클 효과를 넣기위해 파티클의 중점을 총알과 장애물과 부딪힌 위치(총알의 중점)에 맞추고 랜덤엔진을 이용하여 랜덤한 방향으로 방향벡터를 설정하여 파티클이 펼쳐지도록 하였습니다. 충돌한 객체는 랜덤한 위치로 다시 설정하여 장애물 또한 무한하게 보이게 하였습니다.



<그림 : 총알과 장애물 충돌시 터지는 파티클>

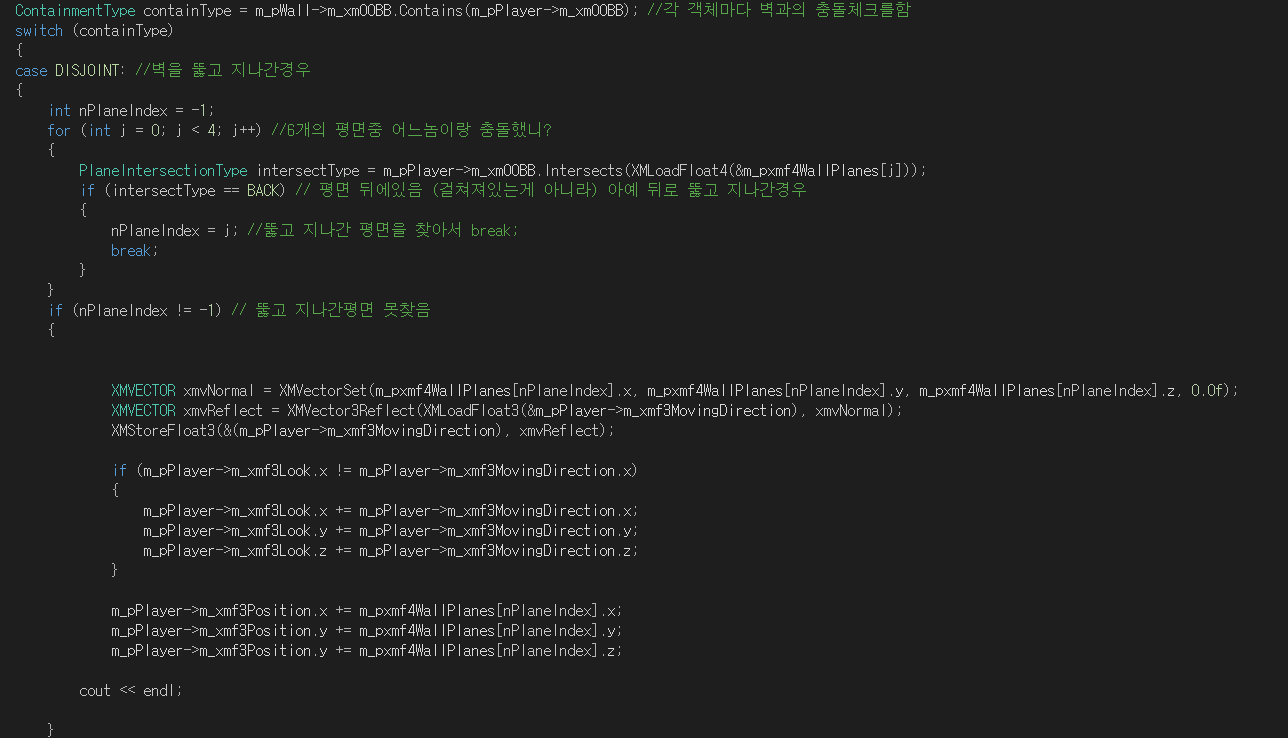
**비행기는 벽을 넘어갈 수 없어야 한다.**

**가정**

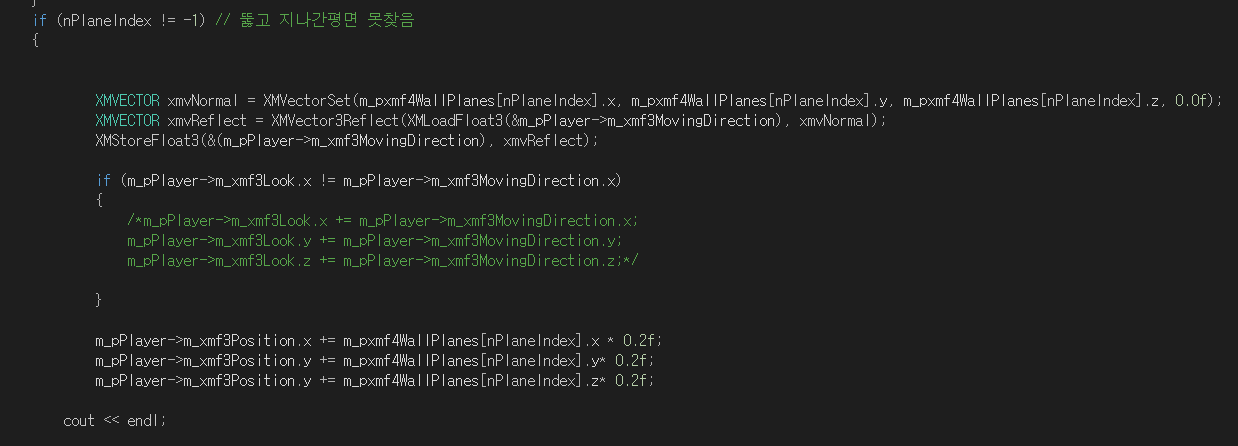


처음 구현하고 싶었던 내용은 비행기의 방향벡터와(빨간색) 벽과 충돌했을 때의 반사벡터(초록색 벡터) 의 합을 이용해 얻어지는 검정색 벡터 방향을 구해 비행기의 Look벡터를 바꾸려하였지만 이렇게하면 Look이 변하면 비행기도 그만큼 회전을 하여야 해서 상당히 자연스럽지 못했습니다. (수업시간의 내용을 이용해 빨간색 벡터와 초록색 벡터를 외적하여 회전축을 찾고, 내적을 이용해 회전 각도를 찾아 비행기를 회전시킬까 고민도 했지만 실패로 돌아감)

잘못된 가정 1



**고친 결과**

****

따라서 반사벡터를 구하는 것 보다, m\_pxmf4WallPlanes의 x,y,z만큼 밀어내도록 하였습니다. (0.2를 곱한 이유는 좀더 자연스럽게 보이게 하기 위해)

**결과와 조작법**

결과만 놓고 봤을 때 프레임워크를 이해하는 부분, 그리고 오류가 났을 때 왜 오류가 났을까 에 대한 부분에 대한 시간이 많이 걸렸지 실제로 개발을 하기위해 코드를 쓰는데는 시간이 얼마 걸리지 않았습니다. FrameAdvance()함수에선 렌더링 작업만 해주고 Animate()함수에서 입력에 대한 처리 작업을 분리해서 수행하고 싶었지만 그렇지 못했다는 점이 크게 많이 아쉬웠습니다.

**사용한 자료구조**

총알은 끝에 삭제가 빈번하여 리스트를 사용하였고

장애물, 파티클, 벽은 배열이 편하여 배열을 사용하였습니다.

**키 입력**

← : 카메라를 fyaw --

→ : 카메라를 fyaw ++

↑ 카메라를 fPitch --

↓ 카메라를 fPitch ++

W : 비행기 앞으로 이동

A : 비행기 좌로 이동

S : 비행기 뒤로 이동

D :비행기 우로 이동

Space : 총알 발사