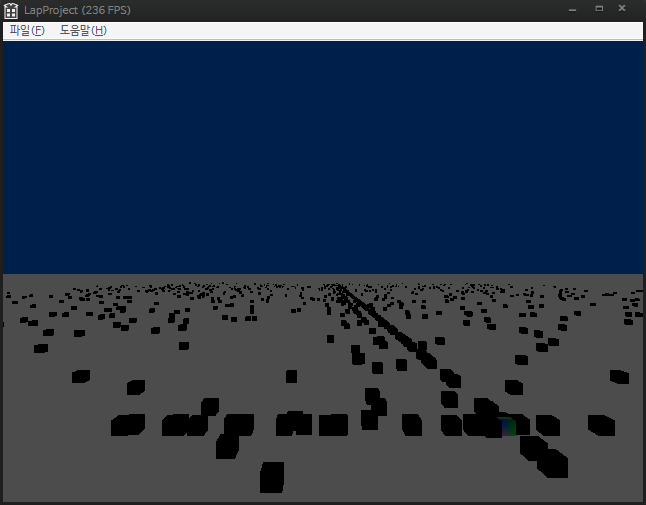
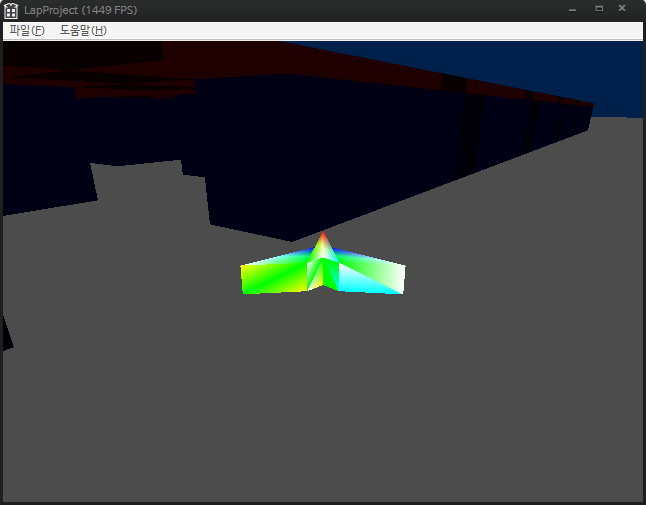
게임공학과 2012180004 권창현





1) 과제 완성 사진 및 과제를 완료하지 못한 부분

씬1 에서는 사각형들이 원형을 그리면서 돌아가면서 잡으러 온다는 부분을 직선으로 변경을 하였으며, 씬2는 구현을 하지 못하여, 씬1에서 모든 객체들이 사라질 경우, 씬 3로 바로 이동을 하게 만들었습니다. 씬 3의 경우 벽면과의 충돌 체크를 하지 못하였으며, 미로를 탈출 할 경우 보스를 구현하는 부분을 완료 하지 못하였습니다.

2) 프로그래밍 과정

처음 프로젝트를 시작하기 위하여 따라하기를 이용하려 하였지만, 중간에 코드 하나가 없는 것을 알게 되어 따라하기를 이용하지 않고, 과제2를 기반으로 작업을 시작하게 되었다.

우선 메인 도형을 하나 그려야 하기에, 과제2에서 도형을 한 개만 남기고 모두 삭제를 하였다. 그 뒤 해당 도형을 키보드로 움직일 수 있게 GameFramework.cpp 에서 WM\_KEYDOWN을 찾아서 Char 입력을 받아서 w, a, s, d를 이용하여 움직이게 하려고 하였다. 적용을 한 뒤 움직이려 하였으나, 키가 입력을 받지 않아서 Numpad를 이용한 4, 8, 5, 6 을 이용하여 움직이게 설정을 하였다.

메인 도형 객체가 정상적으로 움직이기 위하여, Stdafx.h 에서 extern으로 X, Y, Z에 대한 변수를 전역 변수로 만들게 되었다. 만든 변수를 Object.cpp의 Animate에서 각각의 변수를 위치에 넣어 실제 Numpad로 움직이면 객체 도형도 움직이게 설정을 하였다.

메인 도형 캐릭터가 정상적으로 움직이는 것을 확인했으니 다음은 캐릭터를 따라올 외부 객체를 만들어야 하기에, 우선 그림을 그리는 버퍼를 만들기 위하여 Shader.h 에서 ID3D11Buffer 버퍼를 생성하였다. 생성된 버퍼를 사용하여 1000개의 따라오는 적 캐릭터인 사각형을 그린 후 이제 따라오게 만들기 위하여 여러가지 방법을 생각하다가, 각각의 객체의 좌표를 가지고 있으면 움직이기 쉬겠다 생각을 하여 extern 을 이용한 p\_x, p\_y, p\_z를 만들어 가지고 있게 하고, 해당 객체가 몇 개인지를 확인할 수 있게 하기 위하여, c\_Value와 Draw 상태인지를 확인 하기 위한 c\_View를 만들었다.

이제 적 객체들이 본 객체를 따라오게 하기 위하여, Animation 쪽에서 작업을 하려 하였으나, m\_d3dxmtxWorld 좌표계가 이미 본인 캐릭터 객체로 작업이 되어있어서 다른 방법을 찾다가 해결한 방법은 Render 쪽에서 적 객체들을 움직이는 방법을 생각하게 되었다.

Render 부분에서 각 도형의 객체를 움직인 뒤 다시 SetPosition을 이용하여 위치를 잡아주니 그리기 전이라 정상적으로 적 객체가 모두 움직이는 것을 확인을 하였다. 그 뒤 본인 캐릭터를 따라 오게 하기 위하여 예전 WinAPI에서 배운 마우스 고양이 따라잡기 라는 코드를 사용하여 모든 적 객체들이 본인 캐릭터를 따라오게 작업을 하였다. 원래 구현은 적들이 본인 캐릭터를 발견하면 서서히 돌아서 잡는 방식인데 그 부분을 해결하지 못하여 x, z 좌표를 이용하여 캐릭터에 다가 오게 설정을 하였다.

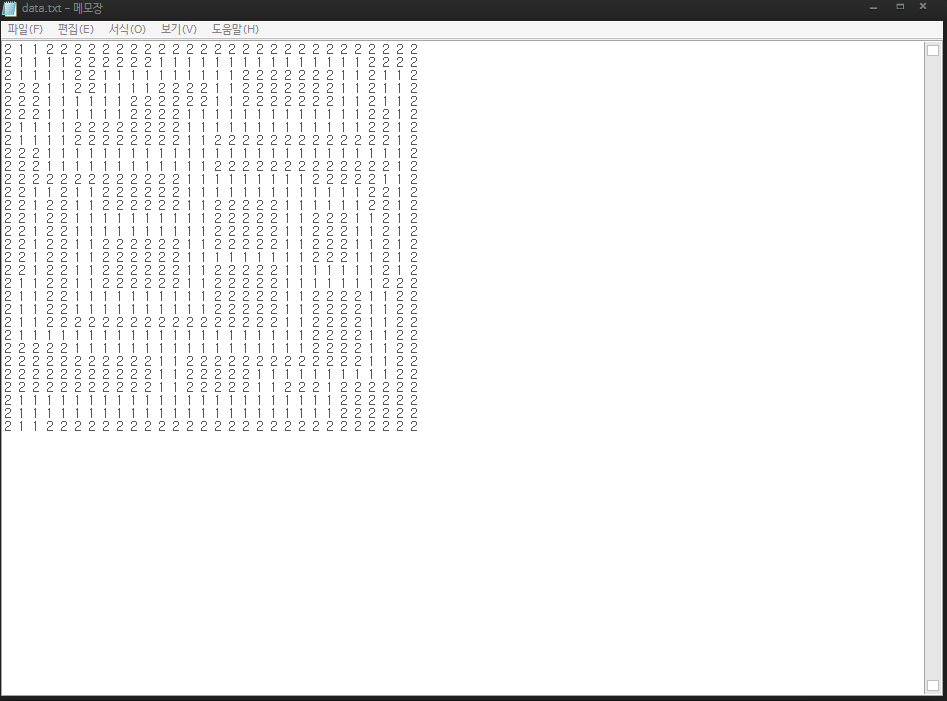
캐릭터와 적 캐릭터가 가까워지면 사라지게 하는 부분은 컴퓨터의 x, z 좌표와 보인 캐릭터의 x, z 좌표를 비교하여 오차 1.0 차이로 가까이 있으면 해당 객체를 위에서 만든 c\_View에서 false로 변경을 하여 그리지 못하게 하였다. 물론 그리지 못하게 하기 위하여 if ( m\_cppObjects[j]->IsVisible( pCamera ) && c\_View[j] == true ) 와 같이 if 문을 주어 그림을 그리지 못하게 하였다.

반경 10m 이내의 모든 적들을 지운다 부분의 경우 GameFramework.cpp 에서 Numpad 0번을 누를 경우 컴퓨터의 객체와, 본인 객체의 거리를 계산 하기 위하여 c\_X(컴퓨터) – p\_X(본인)의 제곱과 c\_Z(컴퓨터)-p\_Z(본인)의 제곱을 더하여 sqr를 하여 그 거리가 100 일 경우 해당 객체를 c\_View에서 false를 만들어 객체가 지워진것처럼 작업을 하였다.

이제 씬을 나누기 위하여 여러가지 방법을 생각하다가 가장 편하게 extern을 사용하여 int변수를 만들어 씬을 나누어 버렸다. 그렇게 하여 Render를 할 때 씬1일경우 이것을 그리게 하고, 씬 2일때는 다른 것을 그리게 설정을 할 수 있게 만들었다.

그 뒤 씬 1에서 모든 객체가 사라졌을 경우 씬 2로 넘어가기 위하여 Object.cpp 에서 씬 1일 경우 for문을 돌려 c\_View가 모두 false 일 경우 씬을 2로 넘기게 하였다.

씬 2를 작업하려 하였다가, 씬 3가 좀 더 쉬운 거 같은 생각이 들어서 바로 씬3로 작업을 하였다.



씬 3의 경우 미로를 만들어야 하는데 미로를 좀 더 편하게 생성을 하기 위하여, 메모장을 이용한 작업을 시작 하였다. 30x30 미로를 텍스트 파일로 만든 뒤 Shader.cpp 에서 BuildObjects 에서 ifstream 을 이용하여 아래와 같이 데이터를 불러 왔다.

ifstream inFile( "data.txt" ); //읽을 파일을 연다.

int num, i = 0, j = 0;

while ( inFile >> num ) {

mi\_map[i][j] = num;

++j;

if ( j >= maze\_Data ) {

i++, j = 0;

}}

Data.txt를 미리 만들어둔 mi\_map이라는 2차원 배열에 텍스트 값을 순서대로 넣어서 텍스트 값이 모두 mi\_map 2차원 배열에 정상적으로 들어가게 하였다.

2차원 배열을 이제 그려주기 위하여 새롭게 버퍼를 만든 뒤 BuildObjects에 새롭게 코드를 추가하였다. 벽 마 다의 크기를 가로 50, 세로 100, 넓이 50으로 하여금 큰 벽을 만든 뒤 30\*30의 배열이니 총 900개의 객체를 생성하게 하였다. 생성 부분은 아래와 같이 작성을 하였다.

int x = -40, y = -7, z = 50;

int k = 0, t = 0;

for ( int i = 0; i < m\_MazenObjects; ++i ) {

pMazeObject = new CRotatingObject();

pMazeObject->SetMesh( pMazeCubeMesh );

if ( mi\_map[k][t] == 2 ) {

pMazeObject->SetPosition( x, y, z );

}

else {

pMazeObject->SetPosition( x, -300, z );

}

x += 50;

++t;

if ( t >= maze\_Data ) {

++k;

t = 0;

x = -40;

z += 50;

}

바로 시작 위치에 그릴 x, y, z 위치를 지정해준 뒤 for문에서 객체를 생성하면, mi\_map에서 해당 텍스트가 2일경우 에는 setPosition에서 해당 x, y, z를 정상적으로 그려주게 하고, 만약 2가 아닌 다른 숫자 일 경우 벽이 아닌 공백이 들어가야 하므로 현재 보이는 시점에서 안보이게만 하기 위하여 y축을 -300을 줘서 해당 객체가 현재 화면에서 보이지 않게만 위치를 지정해 주었다.

900개의 객체는 for문으로 한번에 진행이 되지만, 내가 작업한 미로는 2차원 배열이기에 변수 k, t를 만들어 t 가 미리 정의된 maze\_Data = 30 이 넘어 갈 경우 그 뒤로 그려주게 하게 만들었다.

그 뒤 Render부분은 다른 객체 생성하는 부분과 별 차이 없이 버퍼만 변경을 하여 Render부분을 추가하였더니 제일 첫 장에서의 두번째 이미지와 똑 같은 미로가 정상적으로 보이게 되었다.

이제 미로를 지나갈 캐릭터를 구현해야 하는데, 따라하기 16에서 이미 만들어진 Airplane을 이용하기로 하였다.

본래 F3키를 누르면 비행기가 나타나며 3인칭 시점으로 변경이 되지만 이 부분을 GameFramework.cpp 에서 ProcessInput()부분에서 씬3일경우 자동으로 3인칭 시점으로 변경이 되면서 비행기 객체가 나오게 하였다. 이와 같이 마우스로 드래그 시 화면이 움직이는 부분도 씬 3일경우만 드래그가 가능하게 수정을 하여 씬 3가 될 경우 좌, 우 로 드래그를 하여 미로를 빠져나갈 수 있게 구현을 하였다.

미로의 끝에 보스를 구현하기 위하여 생각을 하던 중 과제2 에서 이용한 피킹 을 이용하여 보스를 구현하기로 하였다. 씬1에서 사용된 본인 객체가 미로가 끝나는 위치에서는 보스로 나타나게 하였다. 보스는 총 10번의 마우스로 클릭을 하게 되면 사라지게 되고, 이렇게 씬 3가 마무리가 된다.

보스를 구현하기 위하여 우선 미로의 출구 위치를 계산하여 해당 위치 정 가운데에 보스를 미리 옮겨 놓았다. 이 부분은 Render에서 SetPosition을 사용하여 구현을 하였다.

3) 프로그래밍 해결 못한 부분

씬1에서 적 캐릭터가 캐릭터를 향할 때 돌면서 들어오는 부분을 구현하지 못하였다. 씬2를 구현하지 못하였다. 씬3에서 미로와 캐릭터 간의 충돌체크를 구현하지 못하였다.

4) 조작 방법

씬1 = Numpad 4, 5, 6, 8 을 이용하여, 본인캐릭터를 이동할 수 있고, Numpad 0 을 누를 경우 주변 10m 이내의 적들이 모두 사라진다.

씬3 = 마우스를 이용하여 좌, 우가 이동이 되며 키보드 →, ←, ↑, ↓를 이용하여 캐릭터를 움직일 수 있으며, 보스는 마우스 클릭 시 죽일 수 있습니다.

5) 과제를 마치면서

과제를 진행하면서 분명 Render는 그리기만 할 뿐 이동이나, 다른 명령들을 하면 안되지만, 과제를 하다 보니 어쩔 수 없이 Render부분에 다른 명령들을 넣게 되었다. 시간이 된다면 이 부분들을 변경하여 Render부분이 아닌 Animate부분에서 실질적으로 움직이게 하게 만들 것 이다.

그 외 씬2 같은 경우 내가 생각하는 방법에서 구현을 하는 방법이 전혀 생각이 나지 않았는데 학기가 끝난 후 라도 한번 따라하기를 다시 한번 진행을 하여 꼭 구현을 해봐야겠다.