**2018180020 박재우(클라이언트) 8주차 기록**

1. 모델이 포함한 정보를 저장할 객체를 만들어야 한다. (텍스쳐, 머터리얼, 애니메이션 등등..)
2. 모델이 포함한 정보를 읽을 함수를 만들어야 한다.
3. 모델을 렌더링할 때 사용할 쉐이더를 만들어야 한다.
4. 모델이 가지고 있는 리소스를 쉐이더로 넘겨줘야 한다.

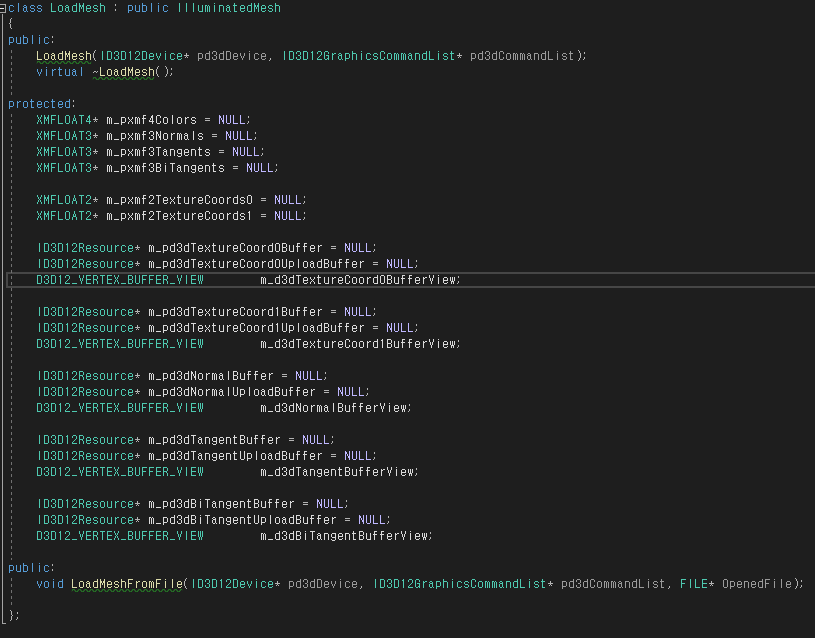
**모델 띄우기 2번**

저번 주차에 Material을 읽는 함수를 만들었고 이어서 Mesh를 읽어오는 함수를 만든다.

우선 읽은 데이터를 저장할 Mesh클래스를 만들어야 한다. 기본적으로 만들어 놓은 Mesh 클래스가 있지만 이 Mesh클래스를 상속받는 CubeMesh, SphereMesh 등 테스트 용도로 만들어 놓은 클래스도 있기 때문에 데이터를 읽는 함수를 부모인 Mesh에 넣으면 테스트 클래스에도 영향이 갈 수 있기 때문에 파생 클래스를 새로 하나 만들기로 했다. 우리의 프레임 워크는 컴포넌트 구조로 원래는 상속관계를 사용하는 것을 지양해야 하지만 우선 모델을 띄우는 것이 목표이기 때문에 성공하고 나면 이 Mesh를 컴포넌트로 변환하는 작업을 할 예정이다.

<Mesh>: 의 구조는 먼저 Vertex의 개수와 Mesh의 이름이 나온다. 그 후

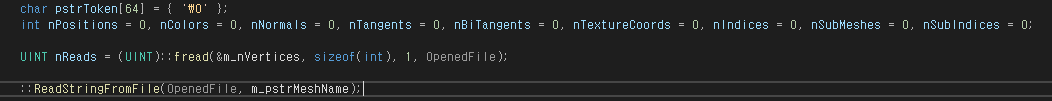
<Bound>, <Position>, <Colors>, <TextureCoords0>, <TextureCoords1>, <Normals>, <Tangents>, <BiTangents>, <SubMeshes> 총 9가지의 데이터가 입력된다.



LoadMesh라는 클래스를 만들었고 위에 설명한 9가지의 데이터를 담기 위해 멤버 변수를 위의 코드처럼 추가했다. Bound는 충돌 박스를 의미하는데 해당 데이터는 부모인 Mesh클래스에 포함되어있고 SubMesh도 마찬가지로 부모가 가지고 있다. 부모에 해당하는 IlluminatedMesh는 Mesh클래스를 상속받고있고 팀원인 서연이가 빛 계산을 위해 만들어 놓은 Mesh클래스이다.

이제 만든 클래스에 데이터를 받기 위해 LoadMeshFromFile 함수를 작성한다.

이후 나오는 대부분의 문구 다음에는 데이터의 수를 알려주는 정수 값이 나오므로 데이터를 받을 int형 변수들을 선언해 모두 0으로 초기화를 해준다. bin파일에서 데이터를 읽어야 하므로 Read—함수를 사용해야 한다. 하지만 해당 함수들은 모두 Object.cpp에 만들어 놨고 작업하는 파일은 Mesh.cpp파일이라서 그냥은 사용할 수 없다. Object.h 헤더파일을 불러오고 stdafx.h에 extern으로 함수를 선언해 Mesh.cpp에서 사용할 수 있게 했다.



<Mesh>: 가 나온 후 바로 불릴 함수이기 때문에 다음에 나오는 정수 값(Vetex개수)를 읽어 m\_nVertices 라는 변수에 저장을 해주고 다음에 나오는 Mesh의 이름을 읽어 저장한다.

이 다음부터 반복문을 이용해 문구를 읽고 해당하는 데이터를 읽어 저장한다.

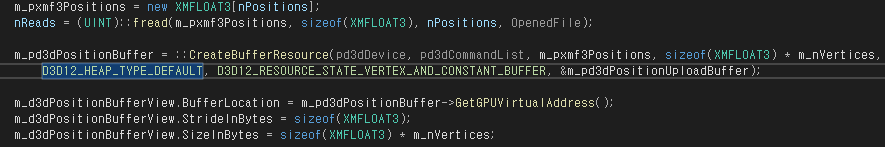
첫번째로 <Bounds>: 는 충돌 박스이고 이 다음에 6개의 실수가 입력되는데 앞의 3개의 입력은 충돌박스의 중심 좌표이고 뒤의 3개의 입력은 중심으로부터 각 좌표의 길이이다. 만약 0 0 0 1 1 1 이렇게 입력이 되었다면 0,0,0 위치에 있으며 길이는 x축으로 1 y축으로 1 z축으로 1의 길이만큼의 크기를 갖는다. 충돌 박스의 모양은 육면체가 될 수도 있지만 만들기에 따라서 구형으로도 만들 수 있고 구형으로 생각하면 뒤의 3개의 입력은 반지름으로 생각해도 될 듯하다. 육면체 구 둘 다 중심과 길이가 필요하다.



위와 같이 BoundingBox의 Center와 Extents에 값을 읽어 넣어 줬다.

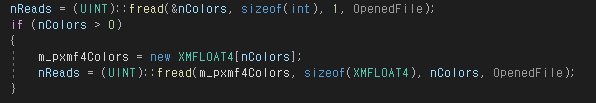
<Positions>: 는 위치에 대한 데이터로 각 Vertex의 위치 데이터이다. Vertex의 개수가 나오고 실제 데이터들이 입력되므로 먼저 개수들을 읽어 저장해준다. Vertex들의 위치 데이터 이므로 0이 나올 일은 없을 것이라 생각되지만 만약 0인경우 데이터를 읽을 필요도 없고 버퍼도 만들 필요가 없기 때문에 예외처리 해준다.

Vertex의 클래스가 존재하는 것 같지만 일단 임시로 각 position과 nomal등등 따로 받아 모델을 띄우는 일을 성공하고 Vertex에 넣어서 하는 방법으로 할지 따로 받아서 할지 결정하겠다.



Position은 XMFLOAT3 형태이고 개수는 파일에서 읽었으므로 개수만큼 생성해 주고 만든 만큼 파일에서 읽어서 저장한다. 저장한 데이터를 이용해 버퍼를 만들어 준다.

다음은 <Colors>: 인데 어차피 텍스쳐를 입혀 의미 없는 데이터 이기도 하다. 대부분 0이 들어올 것이다.



간단하게 개수만큼 배열을 생성하고 만든 만큼 읽도록 했지만 아마 nColors가 0일거라 의미 없을지도 모른다. 아무튼 다음 데이터가 나올 때까지 읽기는 해야 하므로 읽는 부분을 만든다.

다음은 <TextureCoords0>: 과 <TextureCoords1>: 이다. 둘 다 텍스쳐의 좌표 데이터이고 구조도 같다. 읽어오는 변수만 다르게 하여 같은 구조로 읽으면 된다. Position을 읽을 때처럼 개수만큼 배열을 만들고 읽은 후 버퍼를 만들 건데 Position은 실수 3개이지만 TextureCoords는 텍스쳐의 좌표이므로 실수 2개이다.

이 후 나오는 <Normals>: <Tangents>: <BiTangents>: 모두 구조가 <Position>: 과 완전히 똑같기 때문에 저장하는 변수만 다르게 바꿔 입력한다.

마지막으로 <SubMeshes>: 인데 이 아래에는 또 다른 계층이 존재한다. 하지만 다른 것들과는 다르게 <SubMesh>: 하나만 존재하므로 따로 함수를 만들지는 않고 바로 만든다. <SubMeshes>: 다음에 나오는 정수는 <SubMesh>: 의 개수를 의미한다. <SubMesh>: 다음에 입력되는 값은 자신의 인덱스 번호 -> 입력될 데이터의 개수 -> 실제 데이터들 순이다. 따라서 입력될 데이터 개수를 저장할 배열과 실제 데이터들이 입력될 2차원 배열을 만든다.



m\_pnSubSetIndices 에는 데이터 개수가 저장되고 m\_ppnSubSetIndices는 실제 데이터들이 저장된다. 이중 포인터 이므로 2차원 배열의 형태가 될 것이다. 이 다음 SubMesh의 개수 만큼 반복문을 돌며 데이터를 읽기만 하면 된다.



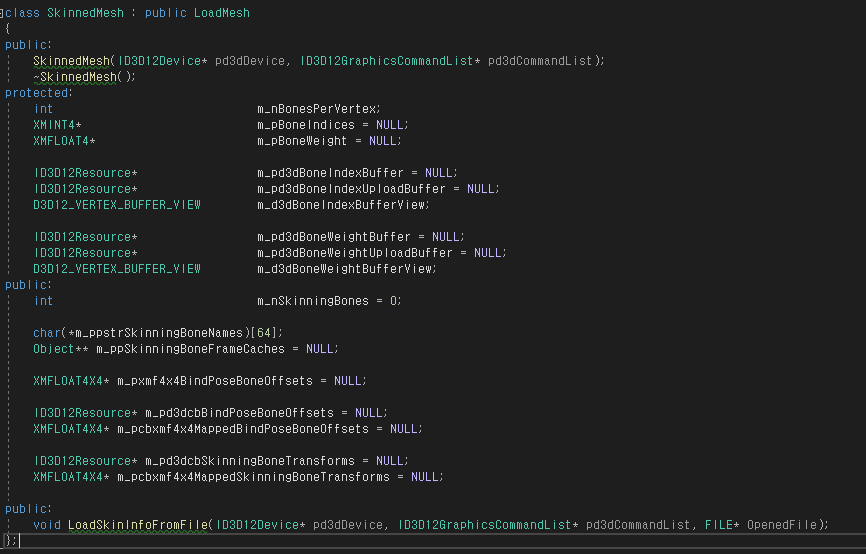
인덱스 번호를 읽고 다음 데이터의 개수를 읽어 저장한다. 데이터 개수가 0개이면 읽을 필요가 없으므로 if문으로 걸러주고 0보다 큰 수면 데이터가 있다는 뜻이므로 m\_ppnSubSetIndices의 해당 인덱스 번호에 배열을 생성해 준다. 그리고 입력 받은 데이터의 버퍼를 생성해준다.

이렇게 Mesh를 불러오는 함수는 완성이 되었고 이 함수를 Object.cpp에서 만들고 있던 함수의 <Mesh>: 부분에서 불러주면 된다.

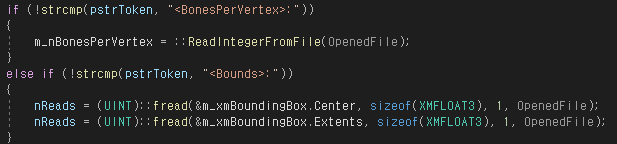


Mesh를 하나 새로 생성하고 생성한 Mesh에서 위에서 만든 LoadMeshFromFile함수를 불러 Mesh의 데이터를 읽고 SetMesh 함수로 오브젝트에 붙인다.

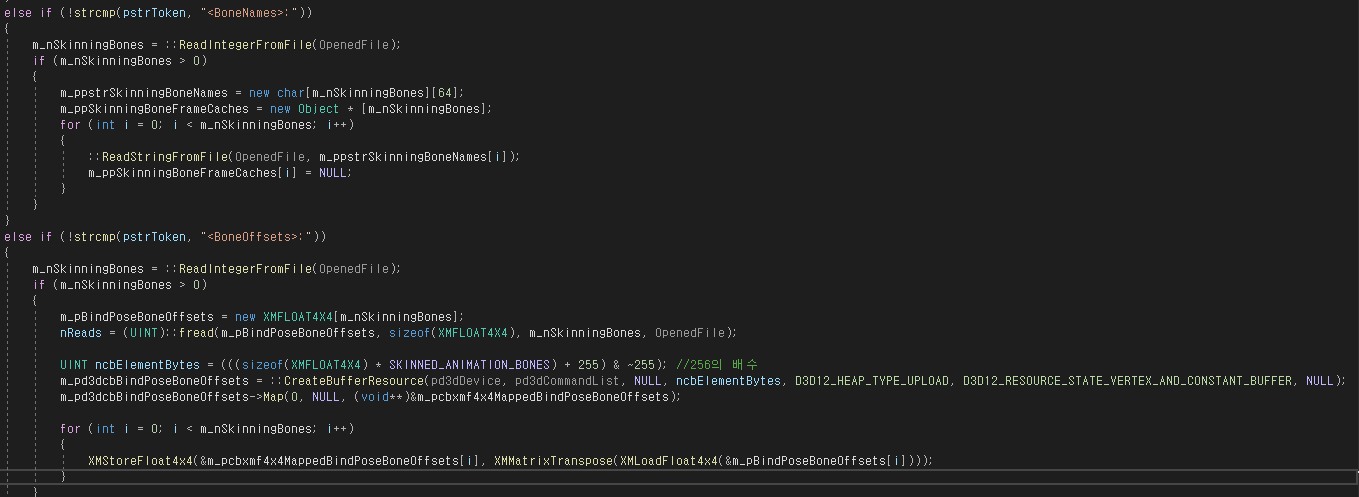
다음으로 만들 함수는 Skinning 데이터를 읽는 함수이다. Skinning 데이터는 애니메이션을 할 때 필요하다. 만들기 전에 새로운 클래스를 하나 만들어야 한다. 이전의 LoadMesh는 모델의 데이터만 읽어 저장하는 클래스였으나 이번에 만들 Mesh는 Skinning 데이터가 포함되어 있기 때문에 LoadMesh를 상속받는 SkinnedMesh 클래스를 새로 만든다.



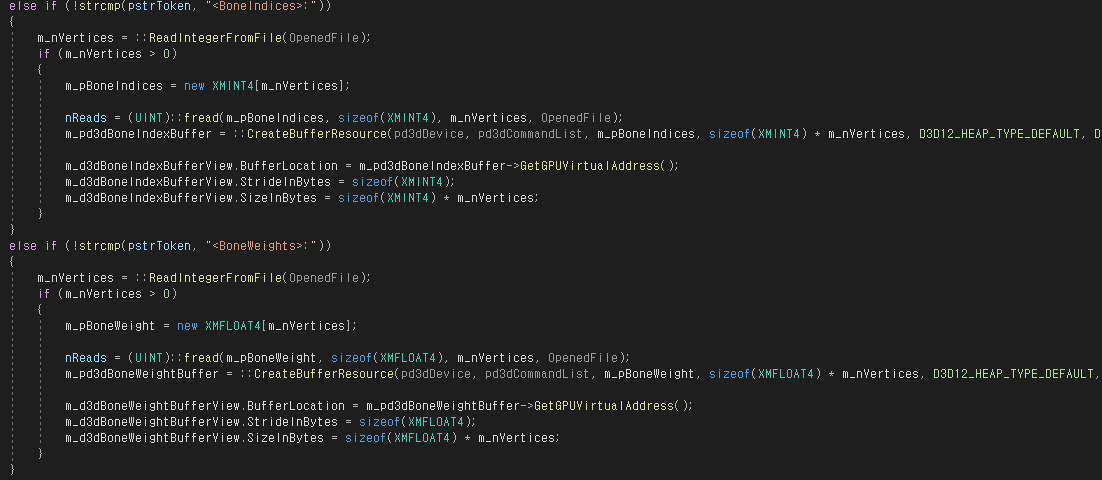
<SkinningInfo> 의 하위 계층들은 <BonesPerVertex> <Bounds> <BoneNames> <BoneOffsets> <BoneIndices> <BoneWeights> 로 총 6개이다. BonePerVertex는 int 한 개가 입력되고 <Bounds>는 LoadMesh의 <Bounds>와 동일하므로 다음과 같이 만들었다.



<BoneNames>: 는 데이터의 개수를 의미하는 정수 값 하나와 정수 값과 동일한 개수의 문자열들이 입력되고 이 문자열들은 각 뼈의 이름들이다. <BoneOffsets>는 Offset행렬이 입력되고 마찬가지로 데이터 개수를 나타내는 정수와 행렬 데이터들이 입력된다. Offset행렬은 버퍼를 생성해주고 매핑해서 루트파라미터에 넘겨줄 준비도 해준다.



<BoneIndices>와 <BoneWeights>는 각각 인덱스 버퍼와 각 뼈의 가중치에 대한 데이터이고 마찬가지로 이후 나오는 데이터의 개수와 실제 데이터 순으로 나온다. 이덱스 버퍼는 정수, 가중치는 실수이다.



함수는 이것으로 모두 만들었고 <Hierarchy>부분을 읽는 기능은 모두 만들었다. 이 함수를 Object.cpp에서 불어주도록한다.



먼저 SkinningInfo가 불렸다는 것은 SkinMesh가 생성된다는 뜻이고 이에 따라 pnSkinnedMeshes의 값을 1 증가시켜준다. 이렇게 해야 SkinMesh가 여러 개 나와도 문제없이 로드 할 수 있다.

Skinninginfo가 나오고 다음은 무조건 Mesh에 대한 내용이 나오므로 바로 다음에 Mesh를 부르도록 만들었다. 그냥 if문을 빠져나가도 Mesh는 불리겠지만 현재 만들 Mesh는 SkinMesh이다. 다른 if문에서 만드는 Mesh는 LoadMesh이므로 Skinning데이터가 들어갈 수 없다. 따라서 SkinnedMesh를 만들어 SkinningInfo를 읽은 후 바로 LoadMeshFromFile 함수를 불러줬다.

마지막으로 Hierarchy를 부르는 부분에서 LoadHierarchy함수를 부르게 한다.



마지막엔 </Hierarchy>가 입력되므로 이것까지 읽어주어 다음에 나올 <Animation>의 데이터를 읽을 준비를 해준다.

**캐릭터 능력치 구상**

캐릭터의 능력치에 관해서 저번 회의에서 정해진 내용이 있다. 우선 체력을 100 언저리로 하려 했으나 1000에 가깝게 만들기로 했다. 전사가 3단콤보의 기술을 갖고 있으므로 최소 3대는 때려야 죽도록 만들게 하기로 정했다. 내용이 정해지고 나서 수치를 정하려고 보니 수식을 조금 수정할 필요가 있다고 판단했다. 수식을 참고한 게임은 리그오브레전드이며 이 게임의 방어력 계수는 100이다. 하지만 리그오브레전드는 상대를 단 3대만에 죽일 수 없는 게임이다. 여러 공격이 오가며 아무 방해 없이 때리기만 해도 꽤 많이 때려야 한다. 공격력을 정말 높게 잡으면 3대정도로 죽일 피해량이 나오지만 그렇게 되면 공격력과 방어력의 수치 차이가 괴리감이 느껴질 정도로 너무 크게 벌어진다. 그렇다고 방어력 계수를 매우 크게 하거나 매우 작게 해도 1000가까이 되는 체력을 3대만에 줄일만한 수치가 나오지 않는다. 위의 수식은 처음부터 리그오브레전드에 맞게 설계되어 있는 수식이라는 것을 알게 되었다. 또한 리그오브레전드는 5대5 게임으로 서로 공평하게 데미지를 계산해야 한다. 하지만 우리 게임은 1대 다수의 전투가 많을 것이고 상대가 FSM으로 행동하는 몬스터인 점을 고려해야 하므로, 플레이어가 적을 공격할 때는 위의 수식에 3을 곱해 피해량을 3배한 수치만큼 데미지를 주고 플레이어는 위의 수식 그대로 적용 받도록 한다. 플레이어도 3대 맞고 죽어버리면 재미가 많이 떨어지고 말 것이다…

추가로 데미지에 랜덤성을 부여하기로 했다. 모든 적들이 3대만에 죽는다면 그것도 재미가 없을 것 같기에 운이 좋지 않다면 4대에 죽이는 경우가 발생하도록 하기로 했고 치명타도 추가되었다. 데미지 수치는 계산된 피해량에 5%정도 차이가 나도록 한다.

예) 피해량이 100이라면 실제 들어갈 수 있는 데미지는 95 ~ 105

방어력 계수 100 공격력 200에 방어력 100으로 딱 100의 피해량이 계산된다. 몬스터가 받는 데이미는 300이 될 것이다. 체력이 900이라면 딱 3대에 죽게 된다.

공격력 200, 방어력 100, 체력 900을 베이스로 조정을 해가며 캐릭터의 능력치를 정해본다.

우선 전사캐릭터의 능력치를 베이스로 삼은 능력치로 정한다.

마법사 캐릭터

마법사 캐릭터는 전사보다는 강한 공격력을 가지고 있지만 몸은 약하게 만들려고 한다. 이미 원거리 공격이라는 점에서 상당히 유리하게 싸움을 할 수 있으므로 방어력을 60정도로 확 낮추고자 한다. 대신 공격력은 220으로 전사보다 높게 설정한다.

플레이어블 캐릭터는 모두 체력을 1000으로 한다.

원거리 몬스터

원거리 몬스터는 전사의 공격 3대에 딱 죽도록 하고 운이 좋다면 2대만에 죽일 수 있도록 만들고 싶다. 방어력이 80이라고 했을 때 피해량은 약 333.3… 이다. 333의 5%는 16.65이다. 333의 데미지로 딱 3대에 죽으려면 체력은 최소 666이상이어야 하고 운이 좋게 2대에 죽는 경우는 333보다 더 많은 데미지가 들어갔을 때이고 5%가 16.65인 점을 보아 675가 적당해 보인다. 공격력은 180으로 전사 캐릭터보다 낮게 설정한다. 공격력은 낮지만 원거리라는 점에서 상당히 성가실 것으로 생각하기 때문에 이런 능력치로 하기로 한다.

근접 몬스터

근접 몬스터는 플레이어에게 붙어서 공격하므로 방어력과 공격력을 밸런스 있게 만들고 싶다. 전사의 공격 기준으로 3대에 죽고 운이 좋지 못하다면 4대까지 때려야 하도록 만들고 싶고 오는 적마다 3대에 다 죽으면 재미가 없을 것 같기에 4대를 때리는 비중이 높도록 하고 싶다.

우선 공격력은 200으로 전사캐릭터와 같게 만든다. 방어력은 90으로 원거리 몬스터보다 10 높다. 이 수치로 피해량을 계산해 보면 약 315의 데미지가 들어간다.3대에 죽기 위해 최소 630보다 높은 체력이 필요하다. 315의 5%는 15.75로 3번의 데미지를 받으면 최대 47.25의 데미지가 손실이거나 추가로 들어가게 된다. 4대를 때려야 죽는 비중을 늘리기로 했으므로 체력은 최소 945이어야 하고 이에 체력을 더 늘려 965로 3대로 죽지 않을 확률을 높인다.

돌진 몬스터

돌진 몬스터는 돌진 공격을 해온다. 돌진 공격이 위협이 될 만한 능력치로 하고 싶고 돌진이라는 공격이 자주 일어날 것 같진 않아 공격속도가 매우 느릴 것으로 생각하고 있다. 돌진 한번만에 아무것도 못하고 죽는 재미없는 몬스터가 되길 원치 않으므로 방어력이 높고 공격력도 높게 체력도 높게 설정하고 싶다. 공격속도 자체는 느려서 침착하게 잘만 피하면 재미있는 몬스터가 될 것 같다.

공격 하나에 위협적이게 느껴지게 하고 싶으므로 전사가 250 정도의 데미지를 받도록 만들고 싶다. 공격력이 460이면 약 242의 데미지가 들어간다. 치명적일 것이다. 방어력은 전사보다 높은 110으로 하고 체력은 플레이어와 마찬가지로 1130으로 한다. 대부분 4대에 죽을 것이고 운이 좋지 않으면 5대를 때려야 할 것이다. 어렵게 느껴질 수치이지만 공격속도가 느리고 침착하게 피하도록 플레이를 한다면 긴장감 있는 플레이가 가능할 것이다.

최종 능력치

전사

공격력 200

방어력 100

체력 1000

마법사

공격력 220

방어력 60

체력 1000

근접 몬스터

공격력 200

방어력 90

체력은 965

원거리 몬스터

공격력 180

방어력 80

체력 675

돌진 몬스터

공격력 460

방어력 110

체력 1130