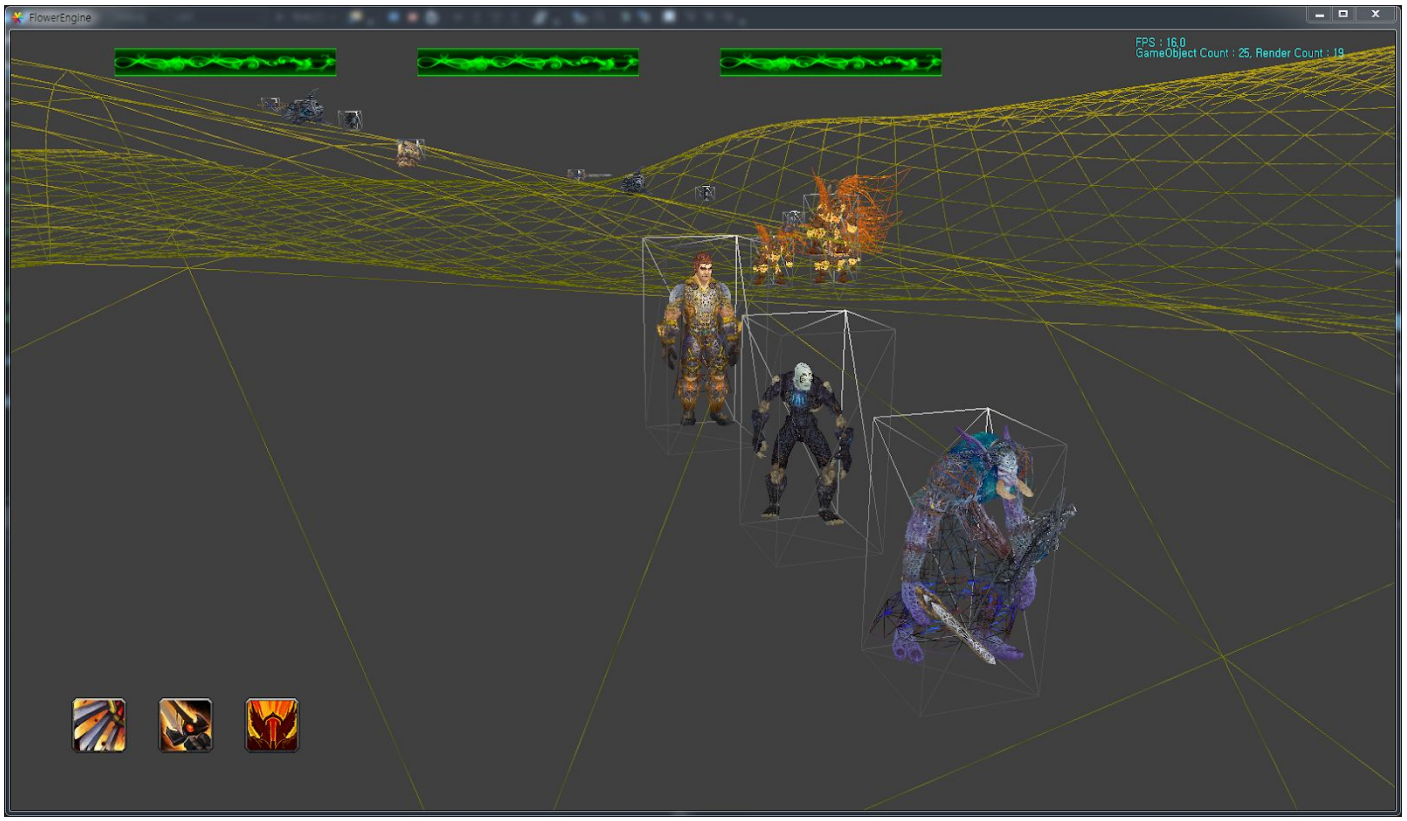
포트폴리오

클라이언트 프로그래머



작성자 : 송현국
hkn10004@naver.com
HP : 010-3592-5921

팀원 : 김동오, 이호형
선생님 : 서울게임아카데미(SGA) 김주안

목 차

게임 개발의 시작

작성자 소개

저는 경력 5년이상 된 프로그래머입니다. **안정적인 개발자가 되기 위해서** 국비지원 수업을 찾던 중 서울게임아카데미의 수업과정을 보고 신청하게 되었고 DirectX API를 이용하여 개발하게 되었습니다. 엔진은 언리얼, 유니티, Cocos2D-X 어떠한 엔진을 써도 상관은 없으나 함께 공부하는데에 DirectX를 사용하므로 자체엔진을 개발하게 되었습니다.

이 프로젝트의 개요

리소스는 다소 추출하기 편한 월드 오브 워크레프트의 모델을 사용하기로 정하였습니다. 게임 장르는 RPG로 정하고 게임 방법은 개발하면서 만들어 나가기로 했습니다. 역할 분담은 따로 나누지는 않고 팀원 모두가 만들고 싶은 것들을 만들어 나가기로 했습니다. 팀 목표로 정한 것은 **최대한 안정적인 개발, 기본기에 충실하자** 입니다. 제 개인적인 목표는 장비 착용과 전투 구현입니다.

사용 환경 : Windows 7, GPU Intel HD Graphics 630

프로세서:	Intel(R) Pentium(R) CPU G4600 @ 3.60GHz 3.60 GHz
설치된 메모리(RAM):	4.00GB(3.88GB 사용 가능)
시스템 종류:	64비트 운영 체제

사용 언어 : C++
사용 툴 : Visual Studio 2017 Community, 3ds Max 2017, Wow Client 8.0.1.27178 File Version, Wow Model Viewer 0.9.0.beta9, DirectX Exporter 2017 Ver
코드 공유 : GitHub 사용

역할 분담

송현국 : 기반 엔진, 전투, 캐릭터 휴먼 담당
이호형 : 3D모델 추출 방법, 지형(Terrain), UI (엔진 구현 포함), 인벤토리, 캐릭터 트롤 담당
김동오 : 상태기계, Text3D, 몬스터, 캐릭터 언데드 담당

기반 윈도우 프레임워크

일단, DXUT를 사용하면 DirectX Sample에서 제공하는 UI등 많은 것들을 사용할 수 있으므로 DirectX Samples에서 제공하는 SimpleSample을 사용하였습니다. 물론, 이 예제들에서 제공하는 것들을 가공해야 합니다.

엔진 개발

프로그램의 시작 WinMain

콜백함수란 함수 포인터로 프레임워크에 필요한 함수를 재정의 해서 그 함수를 연결하는 것입니다. DXUT에서 필요한 부분에 연결한 함수가 호출됩니다.
콜백함수를 간단히 설명하자면 윈도우 메시지 처리, 키보드 마우스 입력 처리, 프레임 업데이트, 렌더링, 디바이스 세팅 관련 콜백함수들이 있습니다.

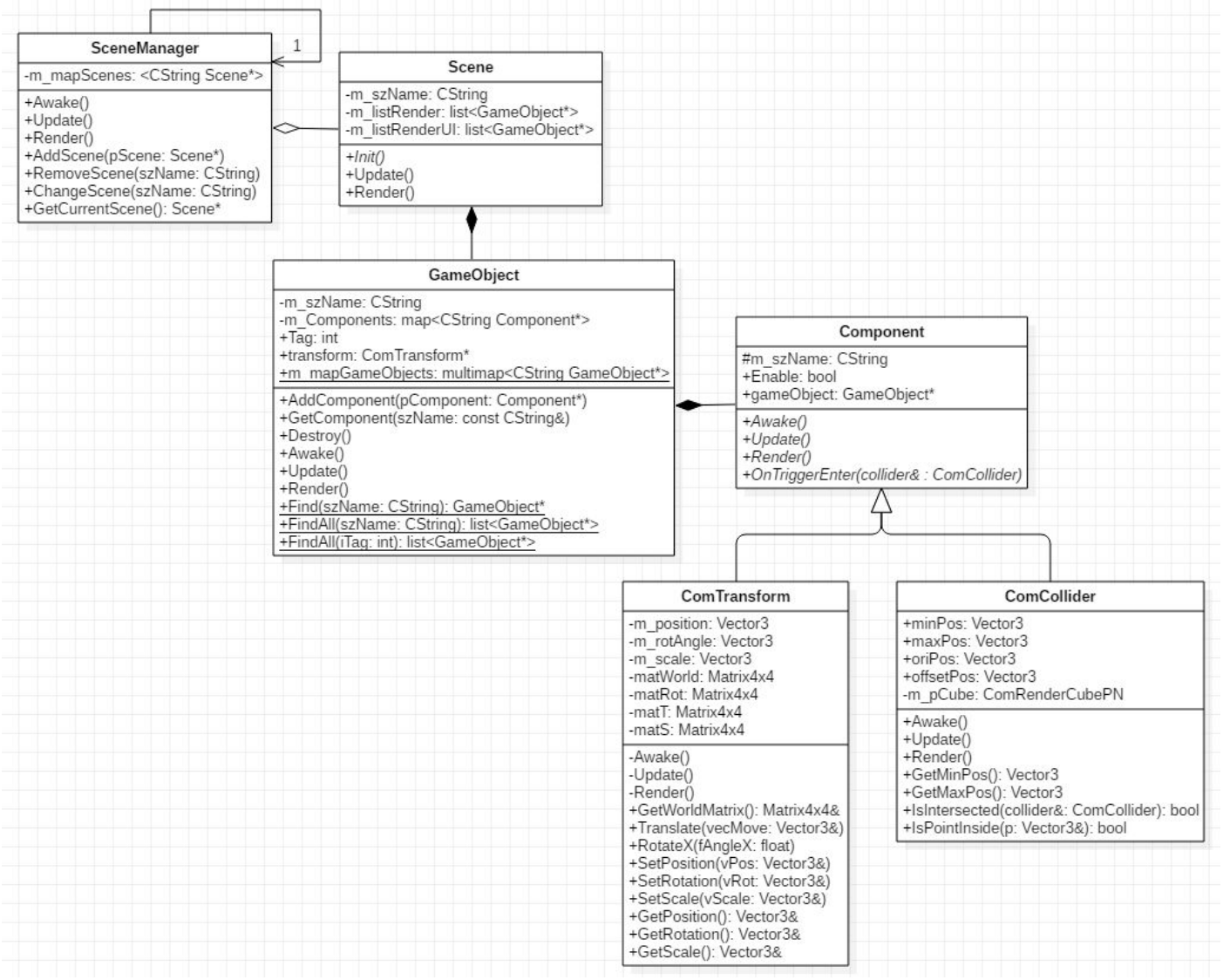
```
//--------------------------------------------------------------------------------------
// Entry point to the program. Initializes everything and goes into a message processing
// loop. Idle time is used to render the scene.
//--------------------------------------------------------------------------------------
int WINAPI wWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPWSTR lpCmdLine, int nCmdShow)
{
	// DXUT will create and use the best device (either D3D9 or D3D10)
	// that is available on the system depending on which D3D callbacks are set below

	// Set DXUT callbacks
	DXUTSetCallbackMsgProc(MsgProc);
	DXUTSetCallbackKeyboard(OnKeyboard);
	DXUTSetCallbackFrameMove(OnFrameMove);
	DXUTSetCallbackDeviceChanging(ModifyDeviceSettings);

	DXUTSetCallbackD3D9DeviceAcceptable(IsD3D9DeviceAcceptable);
	DXUTSetCallbackD3D9DeviceCreated(OnD3D9CreateDevice);
	DXUTSetCallbackD3D9DeviceReset(OnD3D9ResetDevice);
	DXUTSetCallbackD3D9DeviceLost(OnD3D9LostDevice);
	DXUTSetCallbackD3D9DeviceDestroyed(OnD3D9DestroyDevice);
	DXUTSetCallbackD3D9FrameRender(OnD3D9FrameRender);

	InitApp();
	DXUTInit(true, true, NULL); // Parse the command line, show msgboxes on error, no extra command
line params
	DXUTSetCursorSettings(true, true);
	DXUTCreateWindow(L"FlowerEngine");
	DXUTCreateDevice(true, 1600, 900);

	Camera::GetInstance()->Init();
	SceneManager::GetInstance()->AddScene(new SceneRPG("SceneRPG"));
	SceneManager::GetInstance()->ChangeScene("SceneRPG");
	SceneManager::GetInstance()->Awake();

	DXUTMainLoop(); // Enter into the DXUT render loop

	return DXUTGetExitCode();
}
```

GameObject Component System 설계

많은 분께서 아시다시피 GameProgramming Gems 6권에도 소개되어 있고 유니티, 언리얼등에서 사용하는 컴포넌트 기반 설계방식 입니다. 기본 내용을 공부하여 코드 작성 하였습니다.

간략히 설명 드리겠습니다. 위 코드에 보면 SceneManager를 사용하여 시작하는 것을 볼 수 있습니다. SceneManager는 Scene 관리를 담당하는 싱글턴 클래스입니다. 씬을 추가하거나 뺄 수 있고 변경할 수 있습니다. 그리고 현재 씬 객체를 가져오는 함수가 있습니다. 씬에는 추가한 GameObject 객체들이 있고 GameObject에는 구성요소(Component)들이 포함되어 있습니다.

Component의 하위 클래스인 ComTransform은 게임오브젝트의 위치, 회전, 크기 변환을 담당합니다. ComCollider는 큐브의 좌하단 우상단 정점을 이용한 AABB 충돌검사 방식입니다.



프로그래밍 언어

포인터와 레퍼런스

C언어 기본에 대해서 설명 드리겠습니다. 포인터(*)를 사용하는 이유는 다양한 이유가 있지만 인자값으로 객체를 넘기면 그 메모리 크기만큼 복사가 일어납니다. 그러므로 4byte 주소값인 포인터를 사용합니다. 또한, 배열을 사용할 때도 포인터를 사용하여 크기만큼 메모리를 할당하고 사용하고 해제합니다. 메모리의 어떠한 값에 접근하고자 할 때에도 포인터를 사용하여 그 주소값을 얻어오고 그 주소값을 통하여 메모리에 할당된 값에 접근하여 그 값을 사용합니다. 이중 포인터(**)는 포인터의 배열을 만들 때 사용합니다.

레퍼런스(&)는 그 객체의 다른 이름입니다. 선언과 동시에 초기화 해주어야 합니다. 레퍼런스를 사용하면 그 객체만큼 메모리가 복사되는 것이 아닌 레퍼런스의 크기만큼만 사용됩니다.

객체지향 프로그래밍

C++의 기본에 대해서 설명 드리겠습니다. C++은 C와는 달리 객체지향 언어입니다. Java와 C#도 객체지향

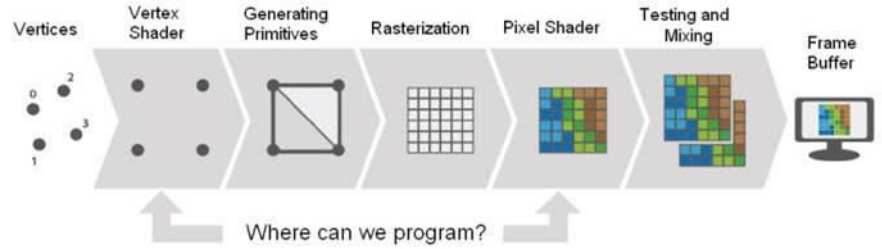
언어입니다. 객체지향 언어의 특징은 프로그램에서 하나의 개념 또는 사물을 하나의 클래스로 구현한 것입니다. 그 클래스는 매니저가 될 수도 있고 인물, 자동차, 과일과 같은 것이 될 수도 있습니다. 객체지향 개념은 C++, Java, C#이 동일합니다.

여기에서 정보의 은닉성 개념이 있습니다. private, protected, public인데 클래스에서 변수를 사용할 때, 자기 자신만 사용하는가(private), 자식 클래스도 사용하는가(protected), 외부 모든 곳에서 사용하는가(public)입니다. 위 그림에서 비공개(private)은 –로 표기, 자식 공개(protected)는 #으로 표기, 모두 공개(public)은 +로 표기합니다.

상속 개념이 있습니다. 즉 기반 클래스(부모)와 상속된 클래스(자식)으로 나눌 수 있는데 위에서는 Component의 자식 클래스가 ComTransform이고 자식 클래스는 부모 클래스의 protected로 선언된 변수들을 사용할 수 있습니다. 함수도 마찬가지 입니다. 가상 함수(virtual)로 선언된 함수를 재정의(override)해서 사용하면 컴파일러는 재정의된 함수를 호출합니다. 순수 가상 함수는 필수적으로 재정의 해서 사용해야 할 함수가 있을 때 사용합니다. 상속관계는 위 그림에서 삼각형으로 표기합니다.

static 변수는 선언하면 메모리에 그 변수 하나만 존재합니다. 싱글턴 클래스는 이 static의 성질을 이용한 것입니다. static 함수는 외부에서 그 클래스의 함수로 바로 접근하여 사용할 수 있게합니다. static 변수는 프로그램이 종료되는 시점에 메모리해제 됩니다. 위 그림(UML)에서 밑줄 그어진 함수가 static 함수이며 이탤릭체로 표기된 함수가 가상함수입니다.

렌더링 파이프라인



RGB 픽셀로 표시할 수 있는 전자기기에 그 픽셀을 그리기(Rendering) 위해서 계산하는 과정을 말합니다. 가장 기본적으로 정점(Vertex) 3개로 렌더링을 해 줄 수 있습니다. 이 정점들을 월드 변환, 뷰 변환, 투영 변환을 해주어야 합니다. 그래픽 관련 계산을 빠르게 하기 위해서 그래픽 처리 장치(Graphics Processing Unit)를 사용하는데 이를 정점 셰이더 픽셀 셰이더라 합니다.

월드 변환

각 정점은 벡터로 정의합니다. V1(0, 1, 0) V2(1, 0, 0), V3(-1, 0, 0) 이렇게 3개의 정점을 선언하면 이것은 로컬 공간의 위치가됩니다. 이 정점들을 월드 공간으로 보내주기 위해서 행렬(Matrix4x4)을 사용합니다.
행렬에는 크기 변환 행렬, 회전 행렬, 이동 행렬이 있습니다. 이 행렬을 모두 곱한값이 월드 행렬(World Matrix)이 됩니다.

```
pDevice9->SetTransform(D3DTS_WORLD, &gameObject->transform->GetWorldMatrix());
```

코드 설명 : 디바이스를 통해 변환을 설정(월드 변환, &월드 행렬); 주로 렌더링 하기 직전에 설정해 줍니다.

뷰 변환

월드 공간에 배치된 오브젝트들을 Z축을 바라보는 카메라 기준으로 변경하는 변환을 말합니다. 뷰 행렬이 있습니다.

```
D3DXMatrixLookAtLH(&m_matView, &m_eye, &m_lookat, &m_up);
pDevice9->SetTransform(D3DTS_VIEW, &m_matView);
```

코드 설명 : 카메라의 위치(m_eye), 바라 보는 곳(m_lookat), 위 방향(m_up) 세개의 벡터를 사용하여 뷰 행렬을 만들고 디바이스를 통해 변환을 설정(뷰 변환, &뷰 행렬); 카메라 Update() 부분에서 설정해 줍니다.

투영 변환

X, Y축은 -1~1사이 Z축은 0~1사이로 변환됩니다. 이 과정에서 카메라에 멀리 있는 정점들은 작게 보이게 됩니다. 카메라가 설정될 때 설정해 줍니다.

```
RECT clientRect = DXUTGetWindowClientRect();
// 투영 행렬 왼손 좌표계, 시야각 45도, 화면 사이즈 종횡비, 보이는 거리 1~1000
D3DXMatrixPerspectiveFovLH(&m_matProj, D3DX_PI / 4.0f, clientRect.right / (float)clientRect.bottom, 1,
fDistZ);
pDevice9->GetTransform(D3DTS_PROJECTION, &matProj);
```

셰이더

셰이더를 사용하는 방법은 DirectX와 유니티, 언리얼 엔진등 각각 다릅니다만 근본적인 내용은 같습니다.
DirectX에서 사용방법 : m_pEffect를 통해 HLSL의 변수에 접근합니다.

```
typedef LPD3DXEFFECT Shader;
Shader m_pEffect;
D3DXCreateEffectFromFile(pDevice9, szFilepath, NULL, NULL, D3DXSHADER_DEBUG, NULL, &pEffect, &pError);
```

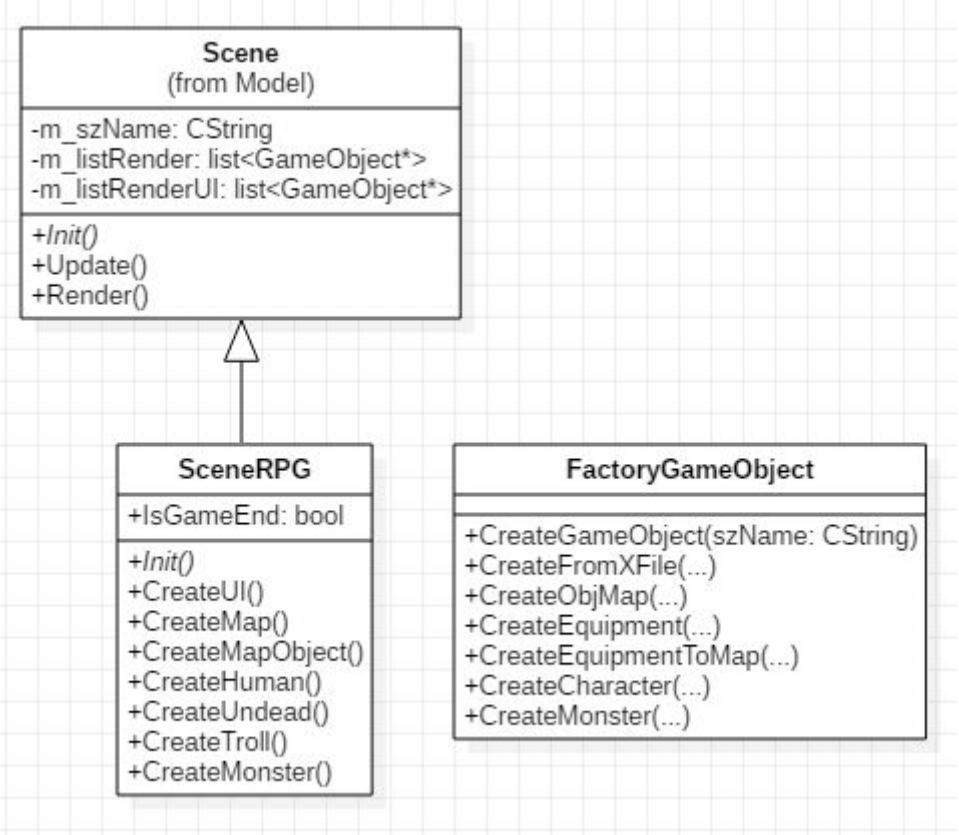
HLSL : 가장 기본적인 정점 셰이더 월드, 뷰, 투영 변환과 픽셀 셰이더 텍스쳐 UV 적용 셰이더 코드입니다.

```
float4x4 gWorldMatrix : World;
float4x4 gViewMatrix : View;
float4x4 gProjMatrix : Projection;

texture DiffuseMap_Tex
<
   string ResourceName = "..\\..\\..\\..\\..\\..\\..\\Program Files (x86)\\AMD\\RenderMonkey
1.82\\Examples\\Media\\Textures\\Earth.jpg";
>;
sampler2D DiffuseSampler = sampler_state
{
    Texture = (DiffuseMap_Tex);
    MipFilter = LINEAR;
    MinFilter = LINEAR;
    MagFilter = LINEAR;
};

struct VS_INPUT
{
   float4 Position : POSITION;
   float2 TexCoord : TEXCOORD0;
};

struct VS_OUTPUT
{
   float4 Position : POSITION;
   float2 TexCoord : TEXCOORD0;
};

VS_OUTPUT TextureMapping_Pass_0_Vertex_Shader_vs_main(VS_INPUT Input)
{
   VS_OUTPUT Output;
   Output.Position = mul(Input.Position, gWorldMatrix);
   Output.Position = mul(Output.Position, gViewMatrix);
   Output.Position = mul(Output.Position, gProjMatrix);
   Output.TexCoord = Input.TexCoord;

   return Output;
}

struct PS_INPUT
{
   float2 TexCoord : TEXCOORD0;
};

float4 TextureMapping_Pass_0_Pixel_Shader_ps_main(PS_INPUT Input) : COLOR
{
   // texture color value
   float3 albedo = tex2D(DiffuseSampler, Input.TexCoord).rgb; // use only 3
   return float4(albedo, 1); // final value is 1
}
//-----------------------------------------------------------------//
// Technique Section for TextureMapping
```

```
//-----------------------------------------------------------//
technique TextureMapping
{
   pass Pass_0
   {
      VertexShader = compile vs_2_0 TextureMapping_Pass_0_Vertex_Shader_vs_main();
      PixelShader = compile ps_2_0 TextureMapping_Pass_0_Pixel_Shader_ps_main();
   }
}
```

텍스쳐 UV란 U(1, 0) V(0, 1)입니다. 32x32 텍스쳐를 사용할 때 U(0.5)값은 16번째 텍셀값입니다.
비례식을 사용해서 텍셀을 구하려면 (32x32 텍스쳐, U(0.5)일 때)
x : 32 = 0.5 : 1
x = 32 * 0.5 = **16**
비례식을 사용해서 U를 구하려면
16 : 32 = x : 1
32 * x = 16
x = 16 / 32 = **0.5**

Scene



객체들 생성 순서
UI -> Map -> MapObject -> Character -> Monster 순서로 생성합니다.

객체 생성 팩토리
객체들은 팩토리 패턴 FactoryGameObject를 사용해서 생성합니다. FactoryGameObject는 하나의 게임오브젝트를 생성할 때 중복되는 코드를 최대한 없애고 객체 생성부분을 수정시 이 함수만 수정하면 되기 때문에 만들었습니다.

처음부터 이렇게 단순화된 함수는 아니였습니다. 코드 정리를 해가면서 함수를 함축시켰습니다.
FactoryGameObject의 Create관련 함수들에 게임 오브젝트의 이름, 파일명, 위치값, 필요한 객체의 포인터등을 매개변수로 전달해서 만드는 방식입니다.

유니티나 언리얼 같은 엔진툴을 사용하여 개발자가 추가하는 게임오브젝트나 구성요소들을 코드로 전부 작성해야 하기 때문에 중복적인 코드가 있을 수 있습니다.

SceneRPG.h

```
#pragma once
#include "stdafx.h"

// WOW 모델을 사용한 롤 플레잉 게임씬
class SceneRPG : public Scene
{
public:
	SceneRPG(CString szName);
	~SceneRPG();

	// Scene을(를) 통해 상속됨
	virtual void Init() override;

	// 게임이 끝났는지 여부
	bool IsGameEnd;

private:
```

```
	// 1. UI 생성
	void CreateUI();
	// 2. 맵을 생성합니다.
	void CreateMap();
	// 3. 맵 오브젝트들을 생성합니다.
	void CreateMapObject();
	// 4. 캐릭터를 생성합니다.
	void CreateHuman();
	void CreateUndead();
	void CreateTroll();
	// 5. 몬스터를 생성합니다.
	void CreateMonster();

	// 6. 테스트 객체들을 생성합니다.
	void CreateTest();

private:
	FactoryGameObject factory;
};
```

SceneRPG.cpp

```
SceneRPG::SceneRPG(CString szName) : Scene(szName),
	IsGameEnd(false)
{
}

SceneRPG::~SceneRPG()
{
}

void SceneRPG::Init()
{
	CreateUI();
	CreateMap();
	CreateMapObject();
	CreateHuman();
	CreateUndead();
	CreateTroll();
	CreateMonster();

	//CreateTest();
}

void SceneRPG::CreateMap()
{
	GameObject* pObjMap = factory.CreateObjMap("ObjMap", "./Resources/obj/Map/TestMap/",
"tempMap2.obj");
}

void SceneRPG::CreateMapObject()
{
	// 휴먼 장비들
	// 아이템 정보를 통하여 맵에 게임오브젝트(어깨 방어구) 생성
	EquipmentShoulder* pShoulder = new EquipmentShoulder("Equipment_shoulder_ItemName01",
"shoulder_01.X", "icon_shoulder_1.png");
	pShoulder->Set(10, 10, 1, 1, eChrType_Human);
	factory.CreateEquipmentToMap(pShoulder, Vector3(3, 10, -8), Vector3(-260, 15, -255));

	EquipmentHelmet* pHelmet = new EquipmentHelmet("Equipment_Helmet", "Helmet_01.X",
"icon_helmet_1.png");
	pHelmet->Set(10, 10, 1, 1, eChrType_Human);
	factory.CreateEquipmentToMap(pHelmet, Vector3(3, 10, -8), Vector3(-260, 15, -253));
```

```
	EquipmentShield* pShield = new EquipmentShield("Equipment_Shield", "Shield_01.X",
"icon_shield_1.png");
	pShield->Set(10, eChrType_Human);
	factory.CreateEquipmentToMap(pShield, Vector3(3, 10, -8), Vector3(-260, 15, -251));

	EquipmentWeapon* pWeaponR = new EquipmentWeapon("Equipment_weapon", "Sword_01.X",
"icon_sword_1.png");
	pWeaponR->Set(10, 20, eChrType_Human);
	factory.CreateEquipmentToMap(pWeaponR, Vector3(3, 10, -8), Vector3(-260, 15, -249));

	// 언데드 장비들
	pShoulder = new EquipmentShoulder("Equipment_shoulder_ItemName01", "shoulder_01.X",
"icon_shoulder_1.png");
	pShoulder->Set(10, 10, 1, 1, eChrType_Undead);
	pShoulder->TextureName = "Resources/character/Equipment/shoulder_robe_b_03blue.png";
	factory.CreateEquipmentToMap(pShoulder, Vector3(3, 10, -8), Vector3(-255, 15, -255));

	pHelmet = new EquipmentHelmet("Equipment_Helmet", "Helmet_01.X", "icon_helmet_1.png");
	pHelmet->Set(10, 10, 1, 1, eChrType_Undead);
	factory.CreateEquipmentToMap(pHelmet, Vector3(3, 10, -8), Vector3(-255, 15, -253));

	pShield = new EquipmentShield("Equipment_Shield", "Shield_01.X", "icon_shield_1.png");
	pShield->Set(10, eChrType_Undead);
	factory.CreateEquipmentToMap(pShield, Vector3(3, 10, -8), Vector3(-255, 15, -251));

	pWeaponR = new EquipmentWeapon("Equipment_weapon", "Sword_01.X", "icon_sword_1.png");
	pWeaponR->Set(10, 20, eChrType_Undead);
	factory.CreateEquipmentToMap(pWeaponR, Vector3(3, 10, -8), Vector3(-255, 15, -249));
}

void SceneRPG::CreateHuman()
{
	GameObject* pGOHuman = factory.CreateCharacter("human_01", "Resources/character/human_01/",
"human_01.X", Vector3(-260, 10.7184200, -260), new ComHuman("ComCharacter"));

	// 카메라
	Camera::GetInstance()->SetTarget(&pGOHuman->transform->GetPosition());
}

void SceneRPG::CreateUndead()
{
	GameObject* pGOUndead = factory.CreateCharacter("undead_01", "Resources/character/undead_01/",
"undead_01.X", Vector3(-260, 10.3810062, -261), new ComUndead("ComCharacter"));
}

void SceneRPG::CreateTroll()
{
	GameObject* pGOTroll = factory.CreateCharacter("troll_01", "Resources/character/troll_01/",
"troll_01.X", Vector3(-260, 10.0443125, -262), new ComTroll("ComCharacter"));

	// 이 게임 오브젝트는 장비 장착 가능
	ComChrEquipment* pEquipment = (ComChrEquipment*)pGOTroll->GetComponent("ComChrEquipment");

	// 장비 장착 테스트(추후 게임 도중 장착으로 수정할 예정)
	EquipmentShoulder*pShoulder = new EquipmentShoulder("Equipment_shoulder_ItemName01",
"shoulder_01.X", "icon_shoulder_1.png");
	pShoulder->Set(10, 10, 1, 1, eChrType_Troll);
	pShoulder->TextureName = "Resources/character/Equipment/shoulder_robe_b_03blue.png";
	pEquipment->Equip(pShoulder);

	EquipmentHelmet* pHelmet = new EquipmentHelmet("Equipment_Helmet", "Helmet_01.X",
"icon_helmet_1.png");
```

```
	pHelmet->Set(10, 10, 1, 1, eChrType_Troll);
	pEquipment->Equip(pHelmet);

	EquipmentShield* pShield = new EquipmentShield("Equipment_Shield", "Shield_01.X",
"icon_shield_1.png");
	pShield->Set(10, eChrType_Troll);
	pEquipment->Equip(pShield);

	EquipmentWeapon* pWeaponR = new EquipmentWeapon("Equipment_weapon", "Sword_01.X",
"icon_sword_1.png");
	pWeaponR->Set(10, 20, eChrType_Troll);
	pEquipment->Equip(pWeaponR);
}

void SceneRPG::CreateMonster()
{
	StatusInfo monStatus;
	monStatus.HP = 50;
	monStatus.HPMAX = 50;
	monStatus.ATK_PHY = 5;

	// 몬스터 생성 (smallderon_orange)
	GameObject* pGOMonX = factory.CreateMonster("Monster", "Resources/monster/smallderon/",
"smallderon_orange.X", Vector3(-243, 7.7184200, -240),
		new ComSmallderonAI("ComChrControl"), GameObject::Find("undead_01"), monStatus);
	pGOMonX = factory.CreateMonster("Monster", "Resources/monster/smallderon/",
"smallderon_orange.X", Vector3(-243, 10.7184200, -243),
		new ComSmallderonAI("ComChrControl"), GameObject::Find("human_01"), monStatus);
	pGOMonX = factory.CreateMonster("Monster", "Resources/monster/smallderon/",
"smallderon_orange.X", Vector3(-246, 9.7184200, -246),
		new ComSmallderonAI("ComChrControl"), GameObject::Find("troll_01"), monStatus);

	monStatus.HP = 60;
	monStatus.HPMAX = 60;
	monStatus.ATK_PHY = 6;

	pGOMonX = factory.CreateMonster("Monster", "Resources/monster/smallderon/",
"smallderon_orange.X", Vector3(-220, 10.7184200, -220),
		new ComSmallderonAI("ComChrControl"), GameObject::Find("human_01"), monStatus);
	pGOMonX = factory.CreateMonster("Monster", "Resources/monster/smallderon/",
"smallderon_orange.X", Vector3(-223, 10.7184200, -223),
		new ComSmallderonAI("ComChrControl"), GameObject::Find("human_01"), monStatus);
	pGOMonX = factory.CreateMonster("Monster", "Resources/monster/smallderon/",
"smallderon_orange.X", Vector3(-226, 10.7184200, -226),
		new ComSmallderonAI("ComChrControl"), GameObject::Find("human_01"), monStatus);
}
```

로딩씬

Init() 함수 : 유니티나 언리얼 엔진 에디터에서 해주는 역할을 코드로 작성하여 UI 레이아웃을 구성하였습니다.
OnClick() 함수 : 버튼이 클릭되었을 때, SceneManager를 통해 씬을 전환합니다.

SceneLoading.cpp

```
void SceneLoading::Init()
{
	FactoryGameObject factory;
	GameObject* m_pUILoading = factory.CreateUIDialog("ScreenUI", Vector3(0, 0, 0));
	ComDialog* comDialog = (ComDialog*)m_pUILoading->GetComponent("ComDialog");

	float fScreenWidth = DXUTGetWindowWidth();
	float fScreenHeight = DXUTGetWindowHeight();
```

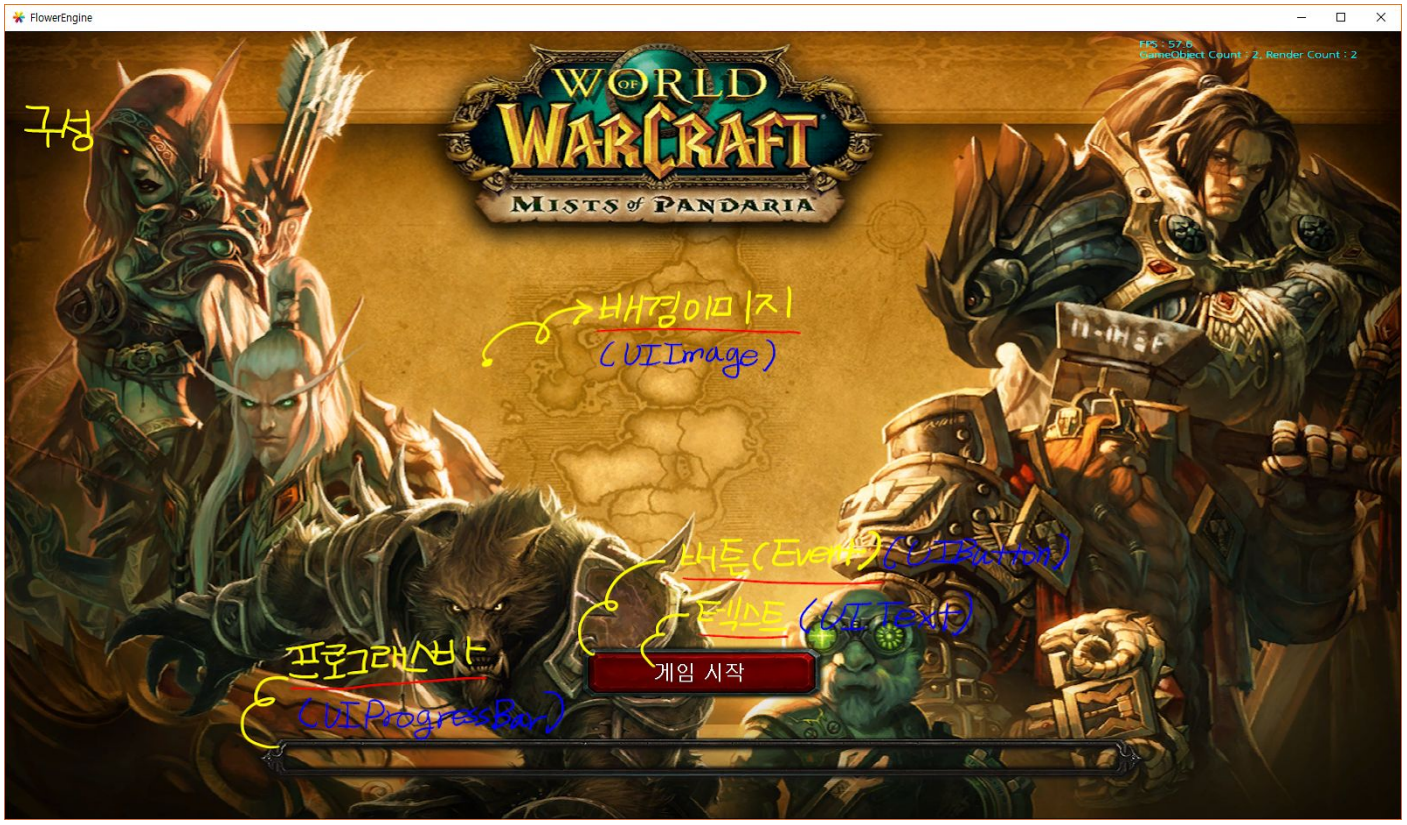
```
	// 배경 이미지
	comDialog->AddImage(eUI_Loading, "./Resources/ui/loadingWide.png");

	// 로딩바 배경
	comDialog->AddImage(eUI_LoadingBarFrame, "./Resources/ui/loading-barborder-frame-v2.png");
	UIImage* pLoadingBarFrame = comDialog->GetImage(eUI_LoadingBarFrame);
	pLoadingBarFrame->SetPosition(Vector3(fScreenWidth / 2 - 512, fScreenHeight - 100.0f, 0));

	// 로딩바
	comDialog->AddProgressBar(eUI_LoadingBar, "./Resources/ui/loading-barfill.png");
	UIProgressBar* pLoadingBar = comDialog->GetProgressBar(eUI_LoadingBar);
	pLoadingBar->SetCurValue(0);
	pLoadingBar->SetPosition(Vector3(fScreenWidth / 2 - 482, fScreenHeight - 85.0f, 0));
	pLoadingBar->SetSize(Vector2(964.0f, 64.0f));
	pLoadingBar->SetScale(Vector3(1.0f, 0.5f, 0.0f));
	pLoadingBar->SetMaxColor(D3DXCOLOR(1.0f, 1.0f, 1.0f, 1.0f));
	pLoadingBar->SetMinColor(D3DXCOLOR(1.0f, 1.0f, 1.0f, 1.0f));

	// 게임 시작 버튼
	comDialog->AddButton(eUI_StartBtn,
		"Resources/ui/Button-up.png",
		"Resources/ui/Button-up.png",
		"Resources/ui/Button-down.png", this, "start");
	UIButton* pBtnStart = comDialog->GetButton(eUI_StartBtn);
	pBtnStart->SetPosition(Vector3(fScreenWidth / 2 - 145, fScreenHeight - 200.0f, 0.0f));
	pBtnStart->SetScale(Vector3(0.7f, 0.5f, 0.0f));

	// 게임 시작 텍스트
	comDialog->AddText(eUI_StartText, Assets::GetFont(Assets::FontType_NORMAL), "게임 시작");
	UIText* uiTextStart = comDialog->GetText(eUI_StartText);
	uiTextStart->SetPosition(Vector3(fScreenWidth / 2 - 64, fScreenHeight - 180.0f, 0.0f));
	uiTextStart->SetDrawFormat(DT_CENTER);

	if (m_pUILoading)
		comDialog->SetIsVisible(true);
}

void SceneLoading::OnClick(UIButton * pSender)
{
	if (pSender->GetButtonName() == "start")
	{
		SceneManager::GetInstance()->AddScene(new SceneRPG("SceneRPG"), false);
		SceneManager::GetInstance()->ChangeScene("SceneRPG");
	}
}
```

씬이 넘어가서 Update()함수에서 로딩 처리를 해 주었습니다. 이유는 OnFrameUpdate()와 OnFrameRender()가 호출되어야 Progress바의 Update()와 Render()가 호출되기 때문입니다.

```
void SceneRPG::Update()
{
	Scene::Update();

	if (iLoadingPer >= 7)	// 로딩 완료
		m_pLoadingUI->SetIsVisible(false);	return;

	if (iLoadingPer == 0)
		CreateUI();			m_pLoadingBar->SetCurValue(10);

	if (iLoadingPer == 1)
		CreateMap();		m_pLoadingBar->SetCurValue(30);

	if (iLoadingPer == 2)
```

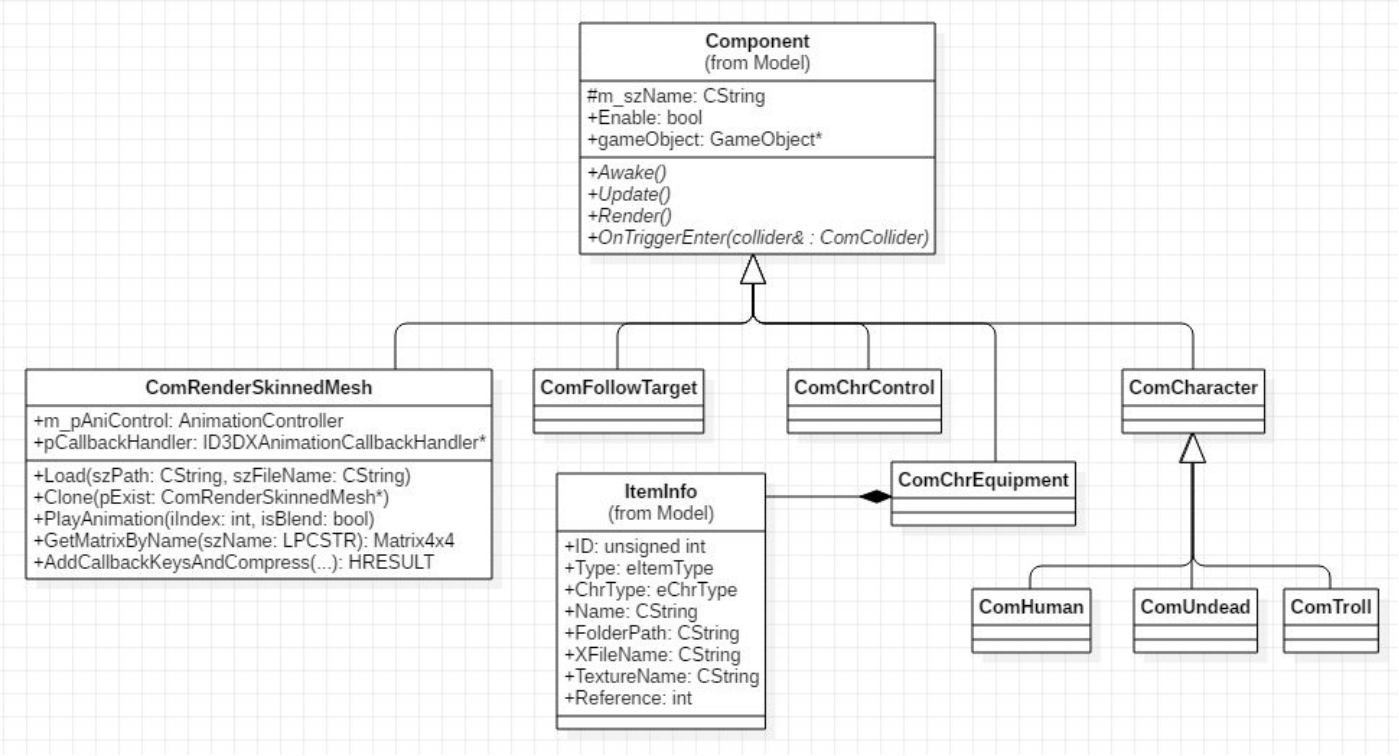
```
		CreateMapObject();	m_pLoadingBar->SetCurValue(40);

	if (iLoadingPer == 3)
		CreateHuman();		m_pLoadingBar->SetCurValue(60);

	if (iLoadingPer == 4)
		CreateUndead();		m_pLoadingBar->SetCurValue(80);

	if (iLoadingPer == 5)
		CreateTroll();		m_pLoadingBar->SetCurValue(95);

	if (iLoadingPer == 6)
		CreateMonster();	m_pLoadingBar->SetCurValue(100);

	++iLoadingPer;
}
```

다음과 같이 구성하였습니다.



캐릭터 개발

캐릭터의 전체적인 구성



설계

캐릭터 게임오브젝트의 구성요소들은 모두 Component에서 상속받아서 구현하여 캐릭터 GameObject에 AddComponent합니다.

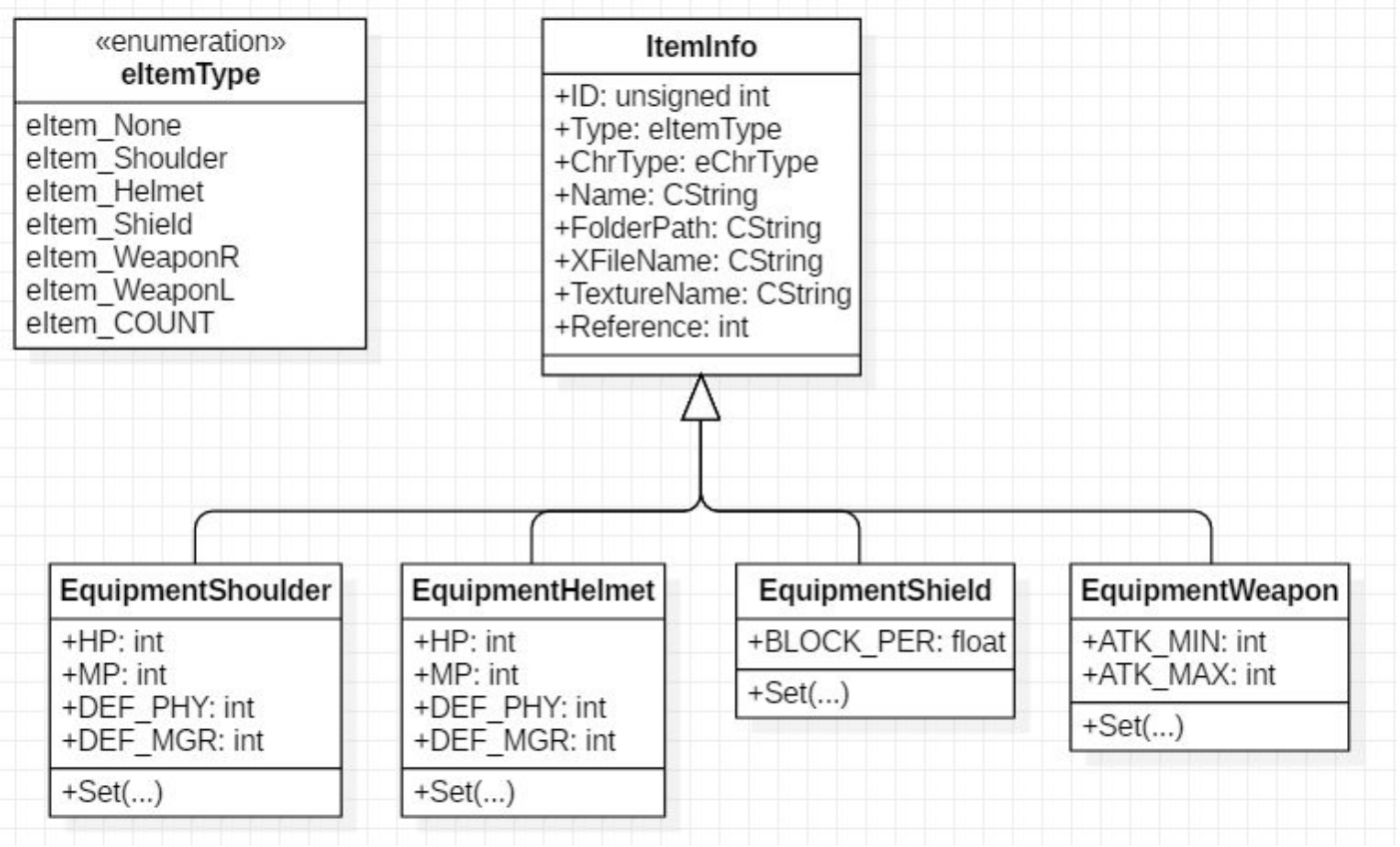
설명

그 중 장비 장착 관련된 구성요소 ComChrEquipment는 아이템 정보(ItemInfo) 최상위 클래스를 포함합니다.
캐릭터는 휴먼, 언데드, 트롤 세 종류가 있는데 팀에서 각자 구현하고 싶은 캐릭터를 정하였습니다. 스킬등 다른 요소가 있을 수 있으므로 ComCharacter에서 상속받아 구현합니다.
ComCharacter의 구현 방식은 일단 ComHuman에 먼저 구현하고 다른 캐릭터들도 공통으로 들어갈 기능들은 상위 클래스인 ComCharacter로 옮겨서 리팩토링합니다.

```
GameObject * FactoryGameObject::CreateFromXFile(CString szName, CString szFolderPath, CString
szFileName, Vector3 & pos)
{
	// PROTOTYPE PATTERN 이미 있는 오브젝트 검사
	GameObject* pGOExist = GameObject::Find(szName);

	GameObject* pGO = new GameObject(szName);
	pGO->transform->SetPosition(pos);
	ComRenderSkinnedMesh* pComSkinnedMesh = new ComRenderSkinnedMesh("ComRenderSkinnedMesh");
	pGO->AddComponent(pComSkinnedMesh);

	// 이미 존재하는 게임 오브젝트라면 복제(Clone) 하여 메쉬를 공유하여 사용합니다.
	/*if (pGOExist != NULL)
	{
		ComRenderSkinnedMesh* pExist =
(ComRenderSkinnedMesh*)pGOExist->GetComponent("ComRenderSkinnedMesh");
		pComSkinnedMesh->Clone(pExist);
```

```
	}
	else*/
		pComSkinnedMesh->Load(szFolderPath, szFileName);

	return pGO;
}

GameObject * FactoryGameObject::CreateCharacter(CString szName, CString szFolderPath, CString
szFileName, Vector3 & pos, ComCharacter* pComChr)
{
	GameObject* pGOChr = CreateFromXFile(szName, szFolderPath, szFileName, pos);
	pGOChr->Tag = eTag_Chracter;
	// 이 게임 오브젝트는 장비 장착 가능
	pGOChr->AddComponent(new ComChrEquipment("ComChrEquipment"));
	// 이 게임 오브젝트의 직업
	pGOChr->AddComponent(pComChr);
	// 이 게임 오브젝트는 대상을 따라다님
	pGOChr->AddComponent(new ComFollowTarget("ComFollowTarget"));
	// 이 게임 오브젝트는 컨트롤 가능
	pGOChr->AddComponent(new ComChrControl("ComChrControl"));
	// 애니 콜백 함수 설정
	ComRenderSkinnedMesh* pRenderSkinnedMesh =
(ComRenderSkinnedMesh*)pGOChr->GetComponent("ComRenderSkinnedMesh");
	pRenderSkinnedMesh->pCallbackHandler = new AttackHandler();
	// 이 게임 오브젝트는 충돌체크 가능
	ComCollider* pCollider = new ComCollider("ComCollider");
	pGOChr->AddComponent(pCollider);
	pCollider->Set(Vector3(0, 0.5f, 0), Vector3(0.3, 0.6, 0.3), false);

	return pGOChr;
}
```

아이템 : 장비아이템



설계

ItemInfo라는 기본 클래스를 만들고 각자 아이템 클래스는 상속 받아서 사용하는 계층 구조 기반 설계입니다.

설명

ItemInfo의 ID, Type, Name등은 아이템 툴을 사용한 파일이나 엑셀 데이터에서 읽어서 사용합니다.

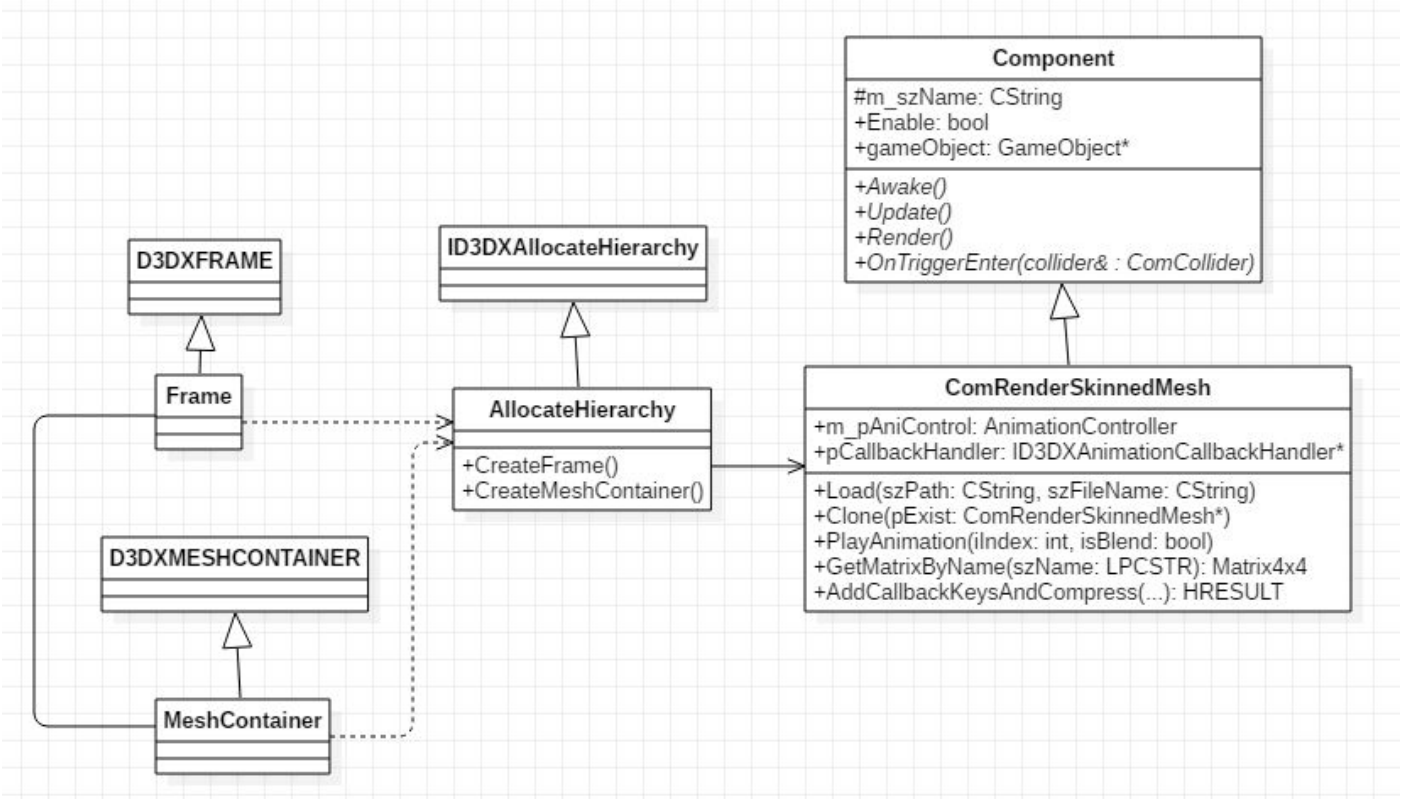
모든 함수의 매개변수에서 ItemInfo* 포인터를 사용하여 아이템 정보에 접근합니다. 그리고 (EquipmentShoulder*)pItemInfo 다형성을 이용하여 하위 클래스의 정보에 접근하여 사용합니다.

Set함수를 사용하여 HP, ATK_MIN등의 매개변수에 값을 할당하는데 이 Set함수는 보통 서버에서 데이터를 받아서 사용하는 것으로 합니다.

현재 아이템타입으로 어깨방어구, 헬멧, 방패, 오른손 무기, 왼손 무기가 있으며 장비는 추가될 수 있습니다.

모델 추출과 캐릭터 애니메이션 (ComRenderSkinnedMesh)

일단 캐릭터를 애니메이션 시키기 위해서는 SkinnedMesh에 대한 이해가 필요합니다. 팔 같은 부분이 접힐 때 주변의 정점들과 Weight정보를 이용하여 부드럽게 접히게 계산하는 것을 스킨드(Skinned)라 하며 코드는 DirectX Samples에 있는 MultiAnimation예제를 참고하여 가공하여도 좋으나 팀프로젝트 개발 기간과 게임 개발 속도를 감안하여 김주안 선생님이 제공해주는 코드를 참고하고 수정하여 사용하였습니다. 셰이더 코드는 DirectX Sample 코드와 같습니다.



설명

ID3DXAnimationController는 DirectX에서 제공해주는 애니메이션 제어 인터페이스 입니다.

Load함수를 통해 .X파일을 읽어오며 AllocateHierachy를 통해 뼈대(Frame)와 메쉬(MeshContainer)들을 만듭니다. Clone은 기존 리소스를 가지고 복제해주는 함수입니다. PlayAnimation 함수는 인덱스를 통해서 애니메이션을 재생시켜 줍니다.

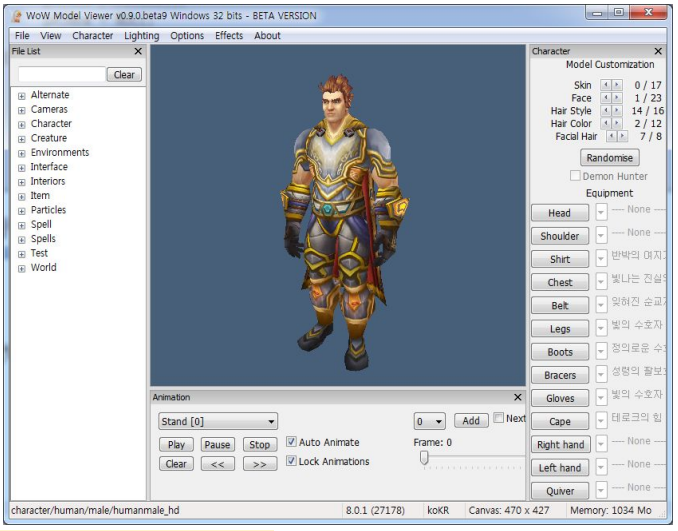
제가 추가한 함수로는 GetMatrixByName 이 함수는 프레임의 이름으로 행렬을 반환하는 함수입니다. AddCallbackKeysAndCompress 함수는 기존에 등록되어있는 AnimationSet 정보를 AniController에서 해제 해준 다음 원하는 프레임에 이벤트 등록과 PLAY_LOOP, PLAY_ONLY 정보를 재설정해 주는 함수입니다.

```
void ComRenderSkinnedMesh::PlayAnimation(int iIndex, bool isBlend)
{
	// 애니메이션 중복 재생 방지
	if (m_iCurrentAniIndex == iIndex)
		return;

	m_iCurrentAniIndex = iIndex;

	PlayAnimation(m_pAniControl, iIndex, isBlend);
}
```

```
HRESULT ComRenderSkinnedMesh::AddCallbackKeysAndCompress(
	LPD3DXKEYFRAMEDANIMATIONSET pAS,
	DWORD dwNumCallbackKeys,
```

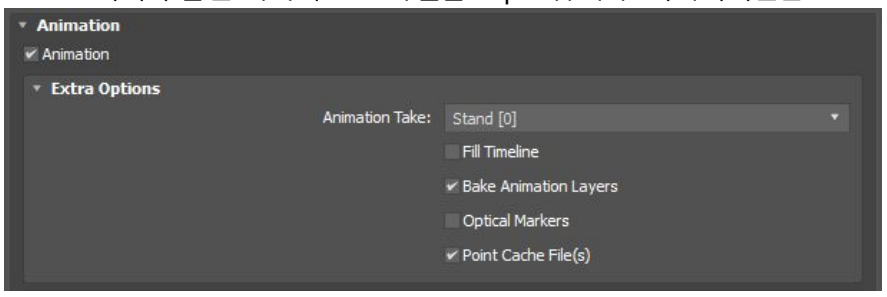
```
	D3DXKEY_CALLBACK aKeys[],
	DWORD dwCompressionFlags,
	FLOAT fCompression, D3DXPLAYBACK_TYPE playType)
{
	HRESULT hr;
	LPD3DXCOMPRESSEDANIMATIONSET pASNew = NULL;
	LPD3DXBUFFER pBufCompressed = NULL;

	// Compression 의미 : 압축
	hr = pAS->Compress(dwCompressionFlags, fCompression, NULL, &pBufCompressed);
	if (FAILED(hr))
		goto e_Exit;

	// 압축된 애니메이션을 pASNew에 만든다.
	hr = D3DXCreateCompressedAnimationSet(
		pAS->GetName(),
		pAS->GetSourceTicksPerSecond(),
		playType,
		pBufCompressed,
		dwNumCallbackKeys,
		aKeys,
		&pASNew);
	pBufCompressed->Release();

	if (FAILED(hr))
		goto e_Exit;

	// 기존 Animation Set을 등록해지 하고
	//pAC->UnregisterAnimationSet(pAS);
	pAS->Release();

	// 새로 만들어진 Animation Set을 등록한다.
	hr = m_pAniControl->RegisterAnimationSet(pASNew);

	if (FAILED(hr))
		goto e_Exit;

	pASNew->Release();
	pASNew = NULL;

e_Exit:

	if (pASNew)
		pASNew->Release();

	return hr;
}
```

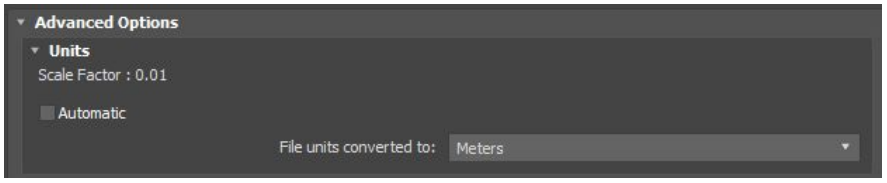
캐릭터 추출



원화가, 3D모델러, 3D애니메이터 분들이 하시는 작업을 WoW Model Viewer 툴을 사용하여 원하는 캐릭터를 만들고 FBX 파일로 추출(Export)합니다. 추출할 때 필요한 애니메이션을 체크합니다.
https://wowmodelviewer.net/wordpress/

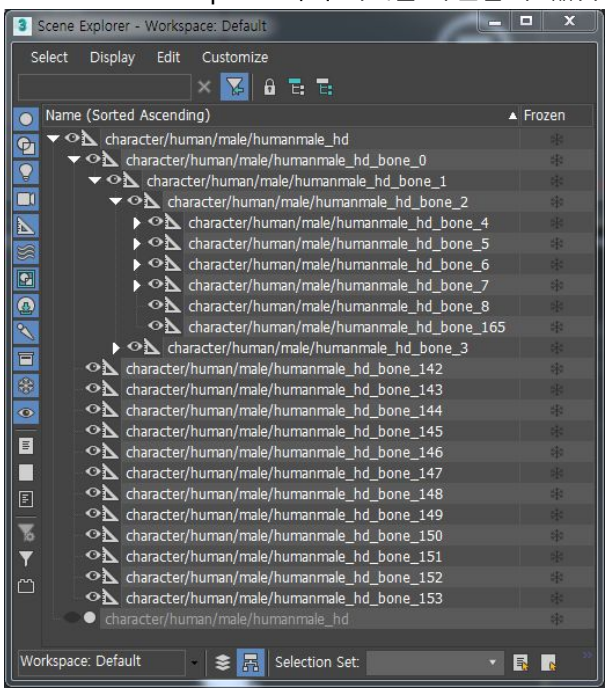
3D Max에서 추출된 캐릭터 FBX 파일을 Import합니다. 애니메이션은 Stand로 놓고 단위는 Meters를 사용합니다.

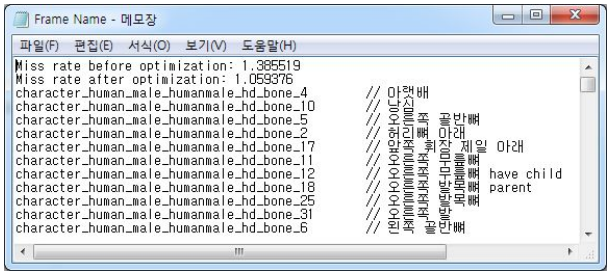




축은 MAX축 그데로 사용합니다. 위 그림은 FRONT에서 바라본 방향입니다.
매쉬 부분은 원래 나누어서 작업되어야 합니다. (실무에서 매쉬 부분은 디자이너와 협의)

Tools > Scene Explorer에서 뼈대를 확인할 수 있습니다.



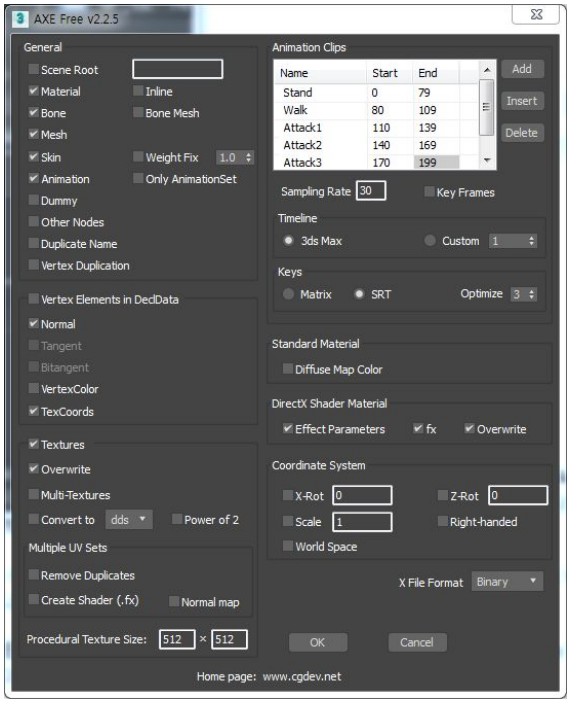


뼈대 분석 파일을 따로 만들었습니다.

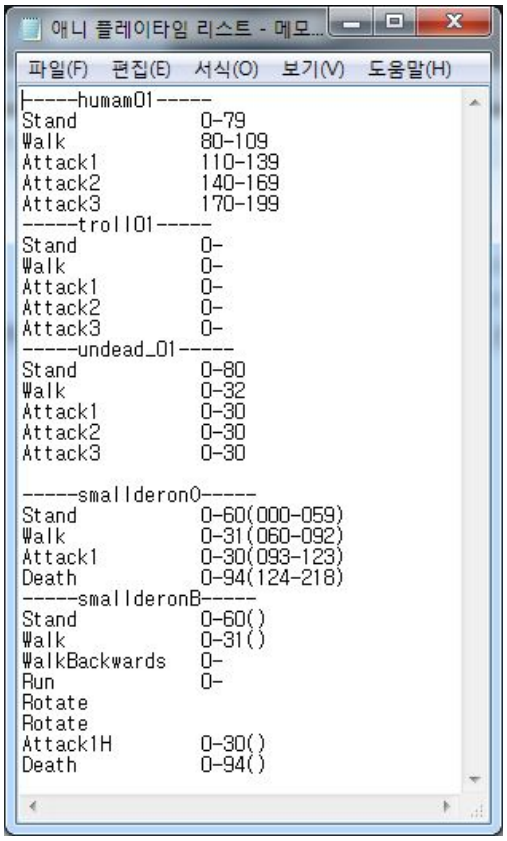
애니메이션을 편집하기 위해 Walk FBX를 Import 한 다음 Animation > Save Animation 합니다. 그 다음 통합본 캐릭터 파일에 Load Animation (프레임 번호 설정) 하여 애니메이션을 합칩니다. 애니메이션 작업시에는 위 메쉬 부분을 선택 해제 하고 작업해야 합니다.

위에서 뼈대의 이름을 바꾸어 작업해 보았으나 나중에 애니메이션 추가할 때 뼈대 이름이 위와 같이 필요하므로 수정하지 않습니다.

DirectX X File Exporter(http://www.cgdev.net/axe/download.php)를 3D Max Plugins 폴더에 복사하여 .X파일로 추출 가능하게 합니다. 추출시 옵션은 다음과 같습니다.

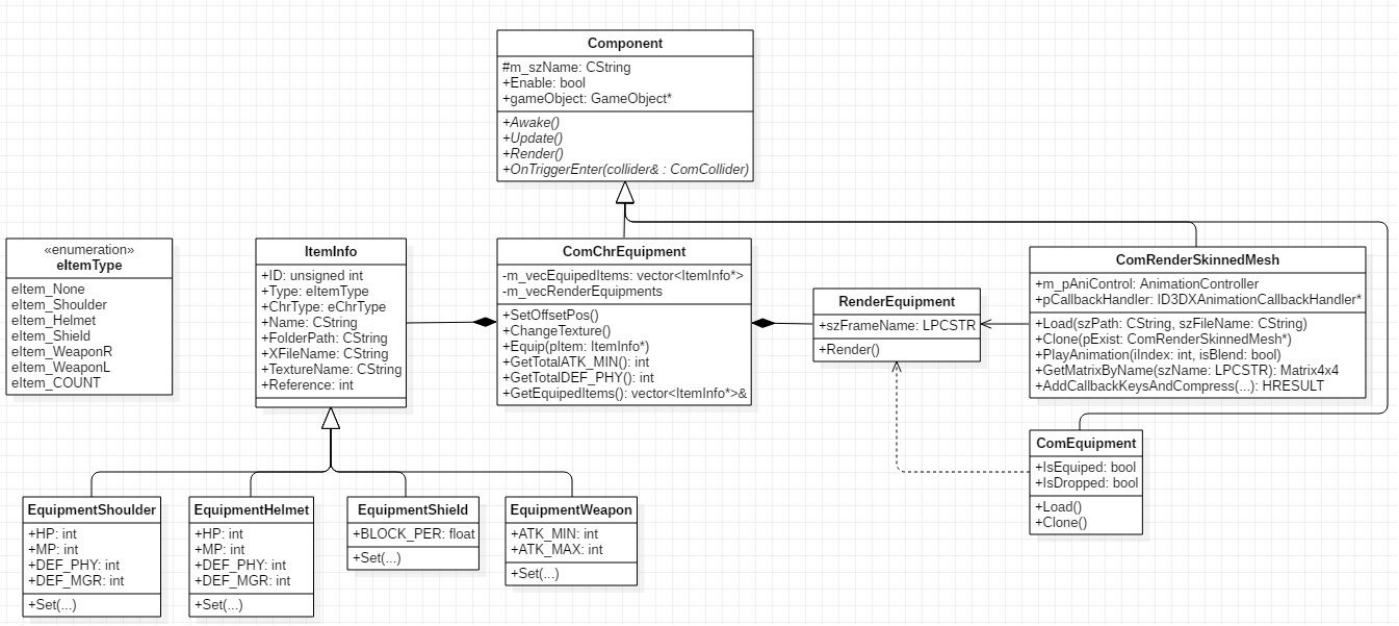


팀워크에 필요한 각 애니메이션 정보는 txt파일로 저장해두었습니다.



이렇게 캐릭터 애니메이션 정보가 담긴 X파일을 추출하여 ComSkinnedMesh를 사용하여 렌더링해주고 애니메이션 해주게 됩니다.

장비 시스템 (ComChrEquipment)



설계

"캐릭터는 장비를 장착할 수 있다"라는 것을 구성요소(Component)로 구현하였습니다. 이 컴포넌트는 ItemInfo 클래스를 포함하고 있습니다.

RenderEquipment는 렌더링에 필요한 정보들 클래스입니다. 이 클래스는 ComRenderSkinnedMesh에서 뼈대 행렬 정보를 가져오고 ComEquipment의 Render함수를 호출하여 장비 게임 오브젝트를 렌더링해줍니다.

ComEquipment는 장비 하나의 게임오브젝트에 추가되는 구성요소이며 지형 위에 렌더링 될 수도 있고 캐릭터의 뼈대 행렬을 이용하여 렌더링 될 수도 있습니다.

설명

각 장비들은 장착될 위치의 캐릭터 뼈대 행렬 정보를 알고 있어야 합니다. 그 변수가 szFrameName입니다.
ComChrEquipment에서 가장 중요한 함수는 Equip()함수입니다.

그리고 하나의 메쉬 정보를 가지고 ChangeTexture를 해주면 장비의 텍스쳐를 변경하여 사용할 수 있으므로 더욱 다양한 장비를 표현할 수 있습니다.

ComChrEquipment.h

```
#pragma once
#include "stdafx.h"

class EquipmentShoulder;
class ComEquipment;
class ItemInfo;

enum eRenderEquipment
{
	eRenderEquipment_ShoulderR,
	eRenderEquipment_ShoulderL,
	eRenderEquipment_Helmet,
	eRenderEquipment_Shield,
	eRenderEquipment_WeaponR,
	eRenderEquipment_WeaponL,
	eRenderEquipment_Count
};

class RenderEquipment
{
public:
	RenderEquipment();
```

```
        ~RenderEquipment();

        void Set(LPCSTR szName, GameObject* pGOParent, GameObject* pGOEquipment);

        void Redner();

        // 뼈이름
        LPCSTR szFrameName;
        // 위치 보정값
        Vector3 m_vOffsetPos;
        // 이 장비의 부모 게임 오브젝트
        GameObject* m_pGOParent;
        // 장비 오브젝트
        GameObject* m_pGOEquipment;
        // 렌더링 구성요소
        ComEquipment* m_pRender;
        // 애니메이션 포인터
        ComRenderSkinnedMesh* m_pAnimation;
};

class ComChrEquipment : public Component
{
public:
        ComChrEquipment(CString szName);
        ~ComChrEquipment();

        // Component을(를) 통해 상속됨
        virtual void Awake() override;
        virtual void Update() override;
        virtual void Render() override;

        // 어깨방어구 장착뼈가 Export되지 않으므로 보정값을 설정하여 위치를 보정해 줍니다.
        // .X File Export시 Frame이 Max축으로 되어있음 [z, x, y축]
        void SetOffsetPos(eRenderEquipment type, Vector3 vOffsetPos = Vector3(3, 10, -8));

        // 아이템 이름으로 텍스쳐를 변경합니다.
        void ChangeTexture(eRenderEquipment type, CString szItemName);

        // 장비를 장착합니다. (월드에서 아이템 획득시, 서버에서 데이터 받은경우, 초기 아이템 장착시 등)
        void Equip(ItemInfo* pItem);

        // 총 장비 공격력을 반환합니다.
        int GetTotalATK_MIN();

        // 총 장비 방어력을 반환합니다.
        int GetTotalDEF_PHY();

        // 장착된 아이템 정보를 반환합니다.
        vector<ItemInfo*>& GetEquipedItems() { return m_vecEquipedItems; }

private:
        LPCSTR GetFrameName(ItemInfo* itemInfo, eRenderEquipment renderType);

private:
        FactoryGameObject factory;

        // 장착된 장비 아이템들
        vector<ItemInfo*> m_vecEquipedItems;

        // 렌더링 할 장비 아이템들
        vector<RenderEquipment*> m_vecRenderEquipments;
};
```

아이템 정보(ItemInfo)를 가지고 장착해주면 해당 장비가 렌더링됩니다.

아이템 정보를 가지고 FactoryGameObject를 통해 아이템을 매쉬를 생성하게 되는데 어깨 방어구 같이 양쪽 모양이 같은 경우는 복제(Clone)하여 사용하게 됩니다.
뼈대가 전부 추출되지 않기 때문에 보정위치(OffsetPos)를 사용하여 수동으로 위치를 조정하였습니다.
ComChrEquipment.cpp

```
void ComChrEquipment::Equip(ItemInfo * pItem)
{
	m_vecEquipedItems[pItem->Type] = pItem;

	LPCSTR szShoulder_Right = GetFrameName(pItem, eRenderEquipment_ShoulderR);
	LPCSTR szShoulder_Left = GetFrameName(pItem, eRenderEquipment_ShoulderL);
	LPCSTR szHelmet = GetFrameName(pItem, eRenderEquipment_Helmet);
	LPCSTR szShield = GetFrameName(pItem, eRenderEquipment_Shield);
	LPCSTR szWeapon_Right = GetFrameName(pItem, eRenderEquipment_WeaponR);
	LPCSTR szWeapon_Left = GetFrameName(pItem, eRenderEquipment_WeaponL);

	GameObject* pGOEquipment = factory.CreateEquipment(pItem, Vector3(0, 0, 0));
	pGOEquipment->transform->SetScale(100, 100, 100);
	((ComEquipment*)pGOEquipment->GetComponent("ComEquipment"))->IsEquiped = true;

	RenderEquipment * pRenderEquipment = new RenderEquipment();

	// 렌더링 객체를 추가합니다.
	switch (pItem->Type)
	{
	case eItem_Shoulder:
	{
		// 방어구 어깨 오른쪽
		pGOEquipment->transform->SetPosition(3, 10, -8); // // [z, x, y축] 보정 위치

		pRenderEquipment->Set(szShoulder_Right, gameObject, pGOEquipment);
		m_vecRenderEquipments[eRenderEquipment_ShoulderR] = pRenderEquipment;

		// 방어구 어깨 왼쪽
		GameObject* pGOShoulderL = factory.CreateEquipment(pItem, Vector3(3, -10, -8), true);
		((ComEquipment*)pGOShoulderL->GetComponent("ComEquipment"))->IsEquiped = true;
		pGOShoulderL->transform->SetScale(100, -100, 100);

		RenderEquipment * pRenderEquipmentL = new RenderEquipment();
		pRenderEquipmentL->Set(szShoulder_Left, gameObject, pGOShoulderL);
		m_vecRenderEquipments[eRenderEquipment_ShoulderL] = pRenderEquipmentL;

		switch (pItem->ChrType)
		{
		case eChrType_Troll:
			SetOffsetPos(eRenderEquipment_ShoulderR, Vector3(3, 12, -6)); // [z, x, y축]
			break;
		}
	}
	break;

	case eItem_Helmet:
	{
		// 방어구 투구
		pRenderEquipment->Set(szHelmet, gameObject, pGOEquipment);
		m_vecRenderEquipments[eRenderEquipment_Helmet] = pRenderEquipment;
	}
	break;

	case eItem_WeaponR:
	{
		// 무기 오른손
```

```
                pGOEquipment->transform->SetPosition(0, 0, -6); // 보정위치 y축 아래로 조금 내림
                pRenderEquipment->Set(szWeapon_Right, gameObject, pGOEquipment);
                m_vecRenderEquipments[eRenderEquipment_WeaponR] = pRenderEquipment;
        }
        break;

        case eItem_WeaponL:
        {
                // 무기 왼손
                pGOEquipment->transform->SetPosition(0, 0, -6); // 보정위치 y축 아래로 조금 내림
                pRenderEquipment->Set(szWeapon_Left, gameObject, pGOEquipment);
                m_vecRenderEquipments[eRenderEquipment_WeaponL] = pRenderEquipment;
        }
        break;

        case eItem_Shield:
        {
                // 방어구 방패 왼손
                pGOEquipment->transform->SetPosition(0, -5, 0); // 보정위치 y축 아래로 조금 내림

                pGOEquipment->transform->SetRotation(Vector3(D3DXToRadian(90), D3DXToRadian(-90), 0));
                pRenderEquipment->Set(szShield, gameObject, pGOEquipment); // 보정위치 팔 밖쪽으로 조금
                m_vecRenderEquipments[eRenderEquipment_Shield] = pRenderEquipment;
        }
        break;
        }
}
```

초기에는 3DMax에서 필요한 뼈대 이름 정보를 Weapon_Left 이런식으로 바꾸어주었는데 애니메이션을 추가하면 원본 뼈대 이름정보가 필요하므로 다음과 같이 함수를 통해 해당하는 행렬의 이름 정보를 설정해주었습니다.

```
LPCSTR ComChrEquipment::GetFrameName(ItemInfo * itemInfo, eRenderEquipment renderType)
{
        switch (itemInfo->ChrType)
        {
        case eChrType_Human:
                switch (renderType)
                {
                case eRenderEquipment_ShoulderR:
                        return "character_human_male_humanmale_hd_bone_28";
                case eRenderEquipment_ShoulderL:
                        return "character_human_male_humanmale_hd_bone_27";
                case eRenderEquipment_Helmet:
                        return "character_human_male_humanmale_hd_bone_39";
                case eRenderEquipment_Shield:
                        return "character_human_male_humanmale_hd_bone_35";
                case eRenderEquipment_WeaponR:
                        return "character_human_male_humanmale_hd_bone_47";
                case eRenderEquipment_WeaponL:
                        return "character_human_male_humanmale_hd_bone_41";
                }

        case eChrType_Troll:
                switch (renderType)
                {
                case eRenderEquipment_ShoulderR:
                        return "character_troll_male_trollmale_hd_bone_34";
                case eRenderEquipment_ShoulderL:
                        return "character_troll_male_trollmale_hd_bone_33";
                case eRenderEquipment_Helmet:
                        return "character_troll_male_trollmale_hd_bone_46";
                case eRenderEquipment_Shield:
                        return "character_troll_male_trollmale_hd_bone_49";
```

```
		case eRenderEquipment_WeaponR:
			return "character_troll_male_trollmale_hd_bone_53";
		case eRenderEquipment_WeaponL:
			return "character_troll_male_trollmale_hd_bone_48";
		}

	case eChrType_Undead:
		switch (renderType)
		{
		case eRenderEquipment_ShoulderR:
			return "character_scourge_male_scourgemale_hd_bone_29";
		case eRenderEquipment_ShoulderL:
			return "character_scourge_male_scourgemale_hd_bone_30";
		case eRenderEquipment_Helmet:
			return "character_scourge_male_scourgemale_hd_bone_46";
		case eRenderEquipment_Shield:
			return "character_scourge_male_scourgemale_hd_bone_41";
		case eRenderEquipment_WeaponR:
			return "character_scourge_male_scourgemale_hd_bone_49";
		case eRenderEquipment_WeaponL:
			return "character_scourge_male_scourgemale_hd_bone_52";
		}
	}

	return "";
}
```

Equip함수에서 벡터에 설정해준 렌더링 객체를 렌더해줍니다.

```
void ComChrEquipment::Render()
{
	for (auto & equipment : m_vecRenderEquipments)
		if (equipment != NULL)
			equipment->Redner();
}
```

RenderEquipment에서 ComRenderSkinnedMesh에서 가져온 뼈대 정보를 가지고 렌더링 해주게 됩니다.

```
void RenderEquipment::Redner()
{
	Matrix4x4* matFrame = m_pAnimation->GetMatrixByName(szFrameName);

	if (matFrame != NULL)
		m_pRender->Render(matFrame, &m_pGOParent->transform->GetWorldMatrix());
}
```

FactoryGameObject에서 장비를 생성할 때 이미 있는 게임 오브젝트이면 Clone해주며 매개변수로
반전여부(IsMirrored)를 받는데 어깨 방어구처럼 반전된 오브젝트라면 스케일 값을 Y축으로 -1 곱해주어 반전해줍니다.

```
GameObject * FactoryGameObject::CreateEquipment(ItemInfo * pItemInfo, Vector3 & pos, bool IsMirrored)
{
	// PROTOTYPE PATTERN 이미 있는 오브젝트 검사
	GameObject* pGOExist = GameObject::Find(pItemInfo->Name);

	GameObject* pGOEquipment = new GameObject(pItemInfo->Name);

	ComEquipment* pEquipment = new ComEquipment("ComEquipment");
	pEquipment->IsMirrored = IsMirrored;
	pEquipment->pItemInfo = pItemInfo;
	++pItemInfo->Reference;

	// 이미 존재하는 게임 오브젝트라면 복제(Clone) 하여 메쉬를 공유하여 사용합니다.
	if (pGOExist == NULL)
```

```
	{
		pEquipment->Load(pItemInfo->FolderPath, pItemInfo->XFileName);
	}
	else
		pEquipment->Clone((ComEquipment*)pGOExist->GetComponent("ComEquipment"));

	// 변경된 텍스쳐 있을 경우 적용
	if (pItemInfo->TextureName.IsEmpty() == false)
		pEquipment->ChangeTexture(0, pItemInfo->TextureName);

	// 크기를 100으로 맞춰주는 이유는 .X File Export시 본 크기가 0.01인듯함.
	if (IsMirrored == true)
		pGOEquipment->transform->SetScale(1, -1, 1); // .X File Export시 Frame이 Max축으로 되어있음
[z, x, y축]

	pGOEquipment->AddComponent(pEquipment);
	pGOEquipment->transform->SetPosition(pos);

	return pGOEquipment;
}
```

맵상에 장비 게임오브젝트를 생성해줄 때에는 그 게임오브젝트에 충돌박스 구성요소를 넣습니다.

```
GameObject * FactoryGameObject::CreateEquipmentToMap(ItemInfo * pItemInfo, Vector3 & pos, Vector3 &
mapPos, bool IsMirrored)
{
	GameObject* pGOEquipment = CreateEquipment(pItemInfo, pos);
	pGOEquipment->Tag = eTag_Item;
	pGOEquipment->transform->SetPosition(mapPos);
	ComCollider* pCollider = new ComCollider("ComCollider");
	pGOEquipment->AddComponent(pCollider);
	pCollider->Set(Vector3(0, 0, 0), Vector3(0.1, 0.1, 0.1), false);
	return pGOEquipment;
}
```

셰이더를 통해 렌더링해주는데 반전된 장비 게임오브젝트라면 렌더링 상태 컬모드를 D3DCULL_CW로 바꾸어줍니다.

```
void ComEquipment::RenderShader()
{
	m_pEffect->SetMatrix("gWorldMatrix", &m_matFinal);
	m_pEffect->SetMatrix("gViewMatrix", &Camera::GetInstance()->GetViewMatrix());
	m_pEffect->SetMatrix("gProjMatrix", &Camera::GetInstance()->GetProjMatrix());

	UINT pass;
	m_pEffect->Begin(&pass, NULL);
	m_pEffect->BeginPass(0);

	for (DWORD i = 0; i < m_iNumMaterials; ++i)
	{
		m_pEffect->SetTexture("DiffuseMap_Tex", m_vecMtrl[i].pTexture);
		m_pEffect->CommitChanges();
		if (IsMirrored)
			pDevice9->SetRenderState(D3DRS_CULLMODE, D3DCULL_CW);
		else
			pDevice9->SetRenderState(D3DRS_CULLMODE, D3DCULL_NONE);

		m_pMesh->DrawSubset(i);
	}

	pDevice9->SetRenderState(D3DRS_CULLMODE, D3DCULL_NONE);
```

```
        m_pEffect->EndPass();
        m_pEffect->End();
}
```

파란색으로 표시된 장비들은 Human의 것이며 빨간색으로 표시된 장비들은 Undead의 것입니다. Troll은 서버에서 장비 정보를 받아와서 이미 장착되어 있는 경우처럼 이미 장착된 캐릭터를 표시합니다.

장비를 줍는 방법은 장비 이름 또는 장비 매쉬를 클릭 또는 몇 초후 자동으로 습득 구매한 팻이 습득하게 하는 방법등 다양할 수 있는데 현재는 장비에게 다가가 캐릭터와 충돌시 습득하여 바로 착용하게 되어 있습니다.

이 부분은 습득 방법을 수정하고 습득시 인벤토리로 들어가며 인벤토리에서 UI를 통하여 장착하게 하는 것으로 장비 장착 알고리즘을 수정해야 할 부분입니다.

ComCharacter.cpp

```
void ComCharacter::OnTriggerEnter(ComCollider & collider)
{
	if (collider.gameObject->Tag == eTag_Item)
	{
		if (m_pChrEquipment != NULL)
		{
			ComEquipment* pEquip =
(ComEquipment*)collider.gameObject->GetComponent("ComEquipment");

			// 장착 타입이 같으면 장착
			if (pEquip->pItemInfo && m_eType == pEquip->pItemInfo->ChrType)
			{
				m_pChrEquipment->Equip(pEquip->pItemInfo);
				collider.gameObject->SetActive(false);
			}
		}
	}
}
```

특정 객체를 따라다님 (ComFollowTarget)

ㅇㅇㅇ

상태기계 (IChrState)

○○○

캐릭터 컨트롤과 몬스터AI (ComChrControl)

ㅇㅇㅇ

애니메이션 이벤트

ㅇㅇㅇ

캐릭터 (ComCharacter)

ㅇㅇㅇ

스킬 시스템

○○○