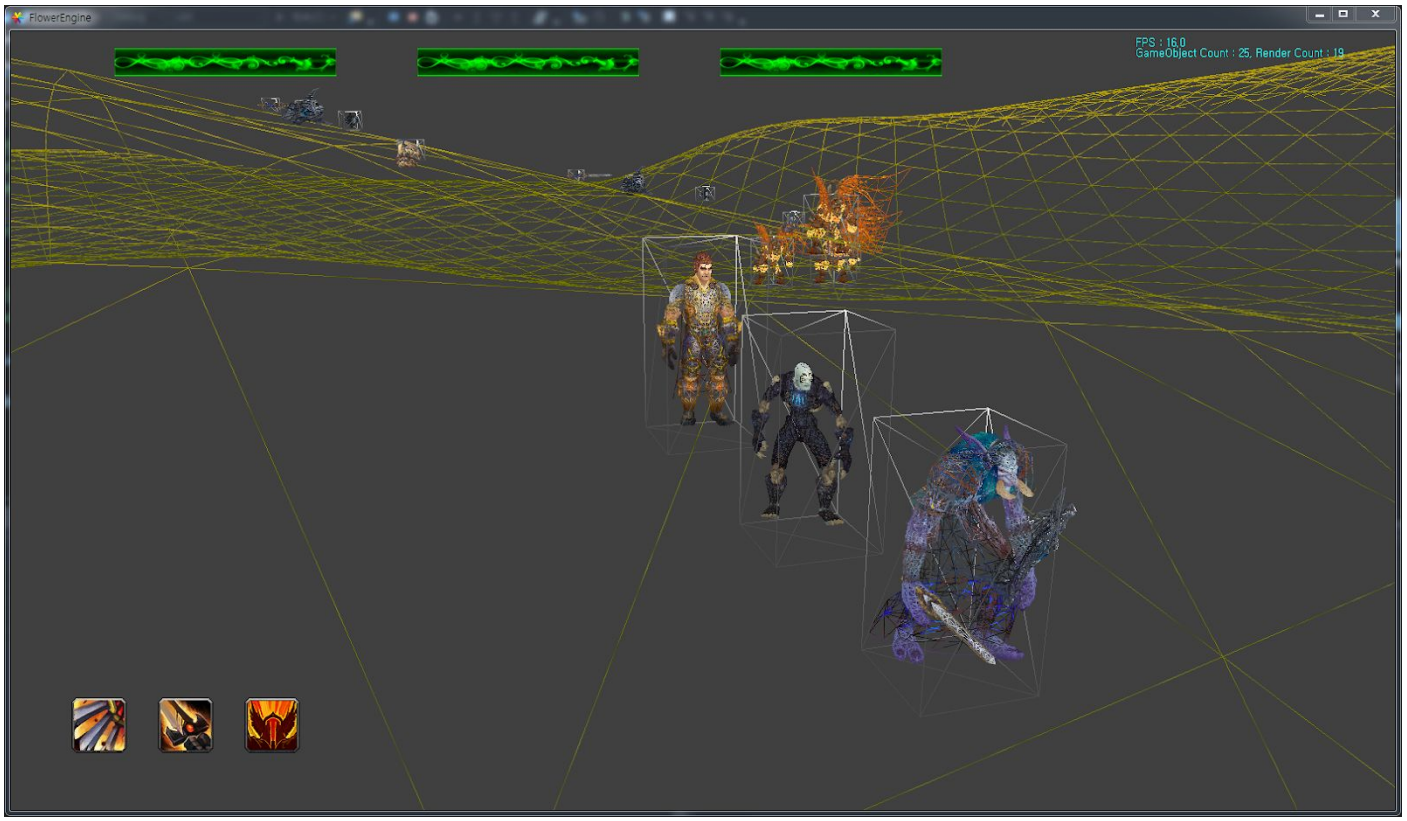
포트폴리오

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(학생용 무료 라이센스), 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.

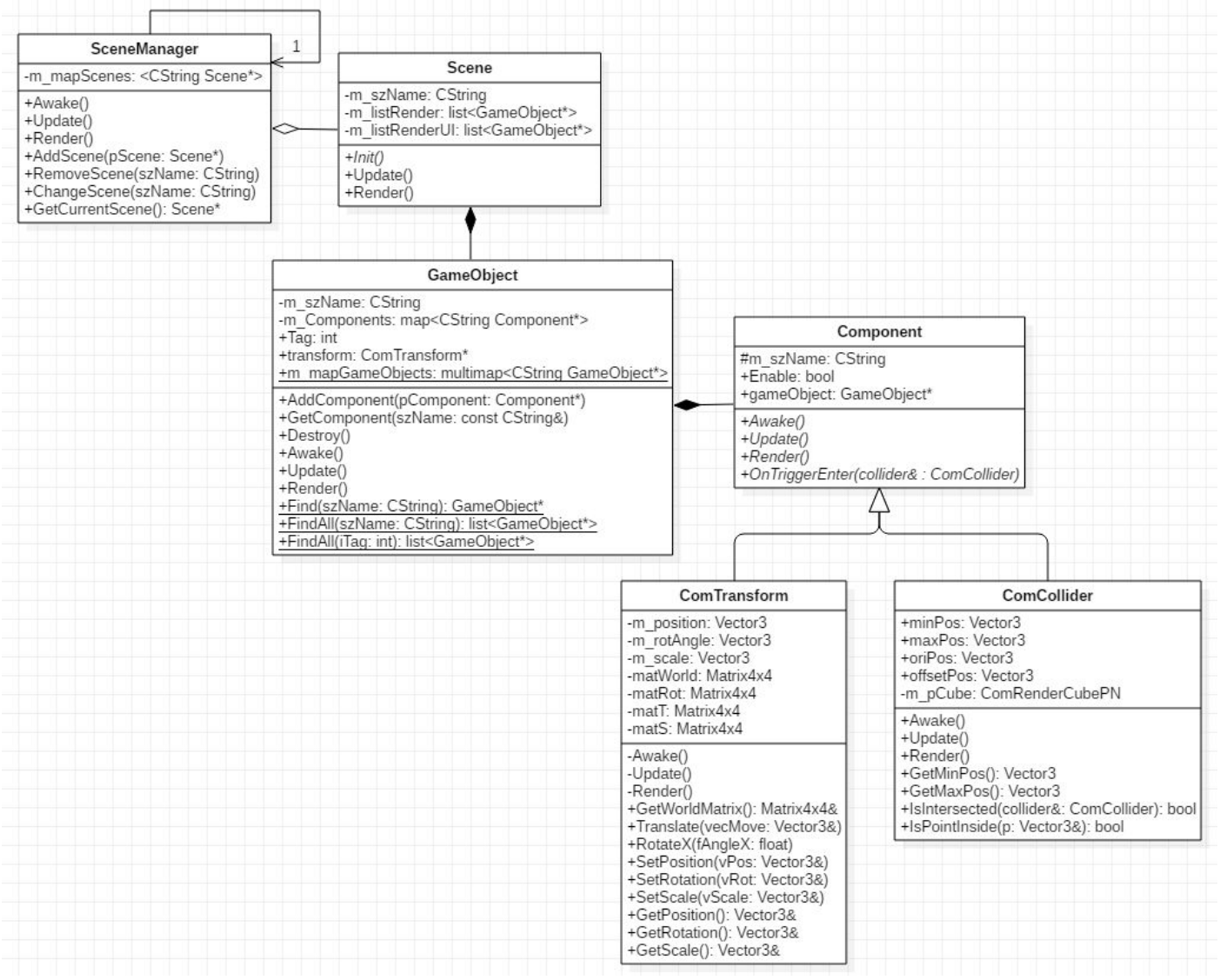
```
//--------------------------------------------------------------------------------------
// Entry point to the program. Initializes everything and goes into a message processing
// loop. Idle time is used to render the scene.
//--------------------------------------------------------------------------------------
int WINAPI wWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPWSTR lpCmdLine, int
nCmdShow)
{
	// Enable run-time memory check for debug builds.
#if defined(DEBUG) | defined(_DEBUG)
	_CrtSetDbgFlag(_CRTDBG_ALLOC_MEM_DF | _CRTDBG_LEAK_CHECK_DF);
#endif

	// 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 SceneLoading("SceneLoading"));
	/*SceneManager::GetInstance()->AddScene(new SceneAircraft("SceneAircraft"));
	SceneManager::GetInstance()->AddScene(new ScenePicking("ScenePicking"));*/
	SceneManager::GetInstance()->ChangeScene("SceneLoading");
	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

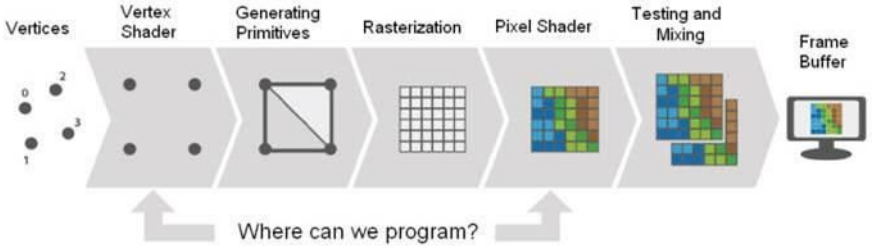
C++의 기본에 대해서 설명 드리겠습니다. C++은 C와는 달리 객체지향 언어입니다. Java와 C#도 객체지향 언어입니다. 객체지향 언어의 특징은 프로그램에서 하나의 개념 또는 사물을 하나의 클래스로 구현한 것입니다. 그 클래스는 매니저가 될 수도 있고 인물, 자동차, 과일과 같은 것이 될 수도 있습니다. 객체지향 개념은 C++, Java, C#이 동일합니다.

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);
```

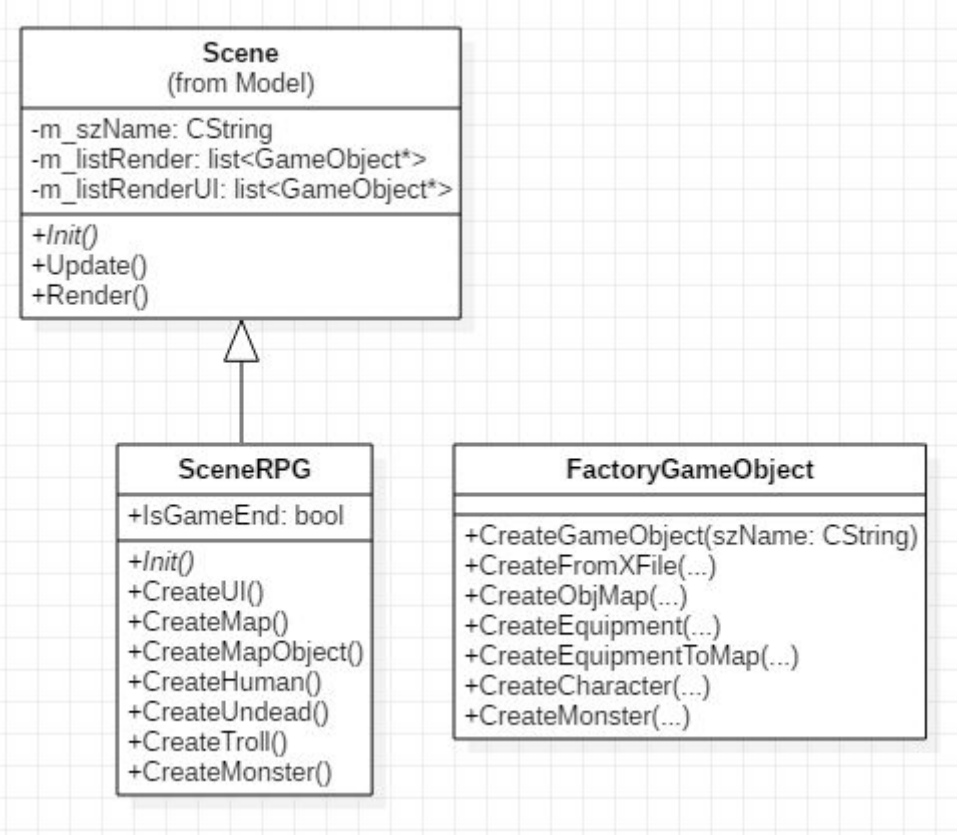
HLSL : 가장 기본적인 정점 셰이더 월드, 뷰, 투영 변환과 픽셀 셰이더 텍스쳐 UV 적용 셰이더 코드입니다.

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

texture DiffuseMap_Tex;
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



위 그림처럼 SceneRPG에서는 게임을 준비하고 객체들을 생성하는 역할을 합니다. 그리고 게임이 끝났는지 여부를 알 수 있고 상위 클래스에서는 그 씬에 추가된 게임 오브젝트들을 카메라와 거리로 정렬(Sort)하여 렌더링합니다.

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;
	void Update() 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;
	D3DLIGHT9 m_lightPoint; // 포인트 라이트

	ComDialog* m_pLoadingUI;
	UIProgressBar* m_pLoadingBar;
	int iLoadingPer;
};
```

SceneRPG.cpp
씬이 로딩 될 때 프로그래스바 처리 : Update()함수에서 로딩 처리를 해 주었습니다. 이유는 OnFrameUpdate()와 OnFrameRender()가 호출 되어야 Progress바의 Update()와 Render()가 호출되기 때문입니다.

```
SceneRPG::SceneRPG(CString szName) : Scene(szName),
	IsGameEnd(false),
	iLoadingPer(-1),
	m_pLoadingUI(NULL),
	m_pLoadingBar(NULL)
{
}

SceneRPG::~SceneRPG()
{
}

void SceneRPG::Init()
{
	GameObject* pLoadingUI = GameObject::Find("ScreenUI");
	m_pLoadingUI = (ComDialog*)pLoadingUI->GetComponent("ComDialog");

	m_pLoadingBar = m_pLoadingUI->GetProgressBar(eUI_LoadingBar);
	m_pLoadingBar->SetMaxValue(100);
	m_pLoadingBar->SetCurValue(0);
}
```

```
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;
}
```

```
void SceneRPG::CreateMap()
{
	// Obj Map 테스트
	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",
		"Human", 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",
		"Undead", Vector3(-260, 10.3810062, -261), new ComUndead("ComCharacter"));
}

void SceneRPG::CreateTroll()
{
	GameObject* pGOTroll = factory.CreateCharacter("troll_01",
"Resources/character/troll_01/", "troll_01.X",
		"Troll", 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("폭염몬", "Resources/monster/smallderon/",
"smallderon_orange.X", Vector3(-243, 7.7184200, -240),
		new ComSmallderonAI("ComChrControl"), GameObject::Find("undead_01"), monStatus);
```

```
	pGOMonX = factory.CreateMonster("폭염몬", "Resources/monster/smallderon/",
"smallderon_orange.X", Vector3(-243, 10.7184200, -243),
		new ComSmallderonAI("ComChrControl"), GameObject::Find("human_01"), monStatus);
	pGOMonX = factory.CreateMonster("폭염몬", "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("폭염몬", "Resources/monster/smallderon/",
"smallderon_orange.X", Vector3(-220, 10.7184200, -220),
		new ComSmallderonAI("ComChrControl"), GameObject::Find("human_01"), monStatus);
	pGOMonX = factory.CreateMonster("폭염몬", "Resources/monster/smallderon/",
"smallderon_orange.X", Vector3(-223, 10.7184200, -223),
		new ComSmallderonAI("ComChrControl"), GameObject::Find("human_01"), monStatus);
	pGOMonX = factory.CreateMonster("폭염몬", "Resources/monster/smallderon/",
"smallderon_orange.X", Vector3(-226, 10.7184200, -226),
		new ComSmallderonAI("ComChrControl"), GameObject::Find("human_01"), monStatus);
}

void SceneRPG::CreateUI()
{
	GameObject* pUIBar = factory.CreateUIDialog("testUI", Vector3(20.0f, 20.0f, 1.0f));
	ComDialog* uiDialog = (ComDialog*)pUIBar->GetComponent("ComDialog");
	uiDialog->SetIsVisible(true);

	GameObject* pUIInven = factory.CreateUIDialog("InvenUI", Vector3(120.0f, 120.0f,
2.0f));
	ComUIInventory* pComInven = new ComUIInventory("ComUIInventory");
	pComInven->SetInvenSize(16);
	pUIInven->AddComponent(pComInven);
	uiDialog = (ComDialog*)pUIInven->GetComponent("ComDialog");

	uiDialog->SetToggleKey('I');
	uiDialog->SetMoveable(true);
	uiDialog->AddImage(0, "Resources/ui/ui-backpackbackground.png");
	uiDialog->AddButton(2, "Resources/ui/ui-panel-minimizebutton-up.png",
"Resources/ui/ui-panel-minimizebutton-up.png",
		"Resources/ui/ui-panel-minimizebutton-down.png", pComInven, "InvenClose");

	uiDialog->GetButton(2)->SetPosition(Vector3(230, 8, 0));
}
```

로딩씬

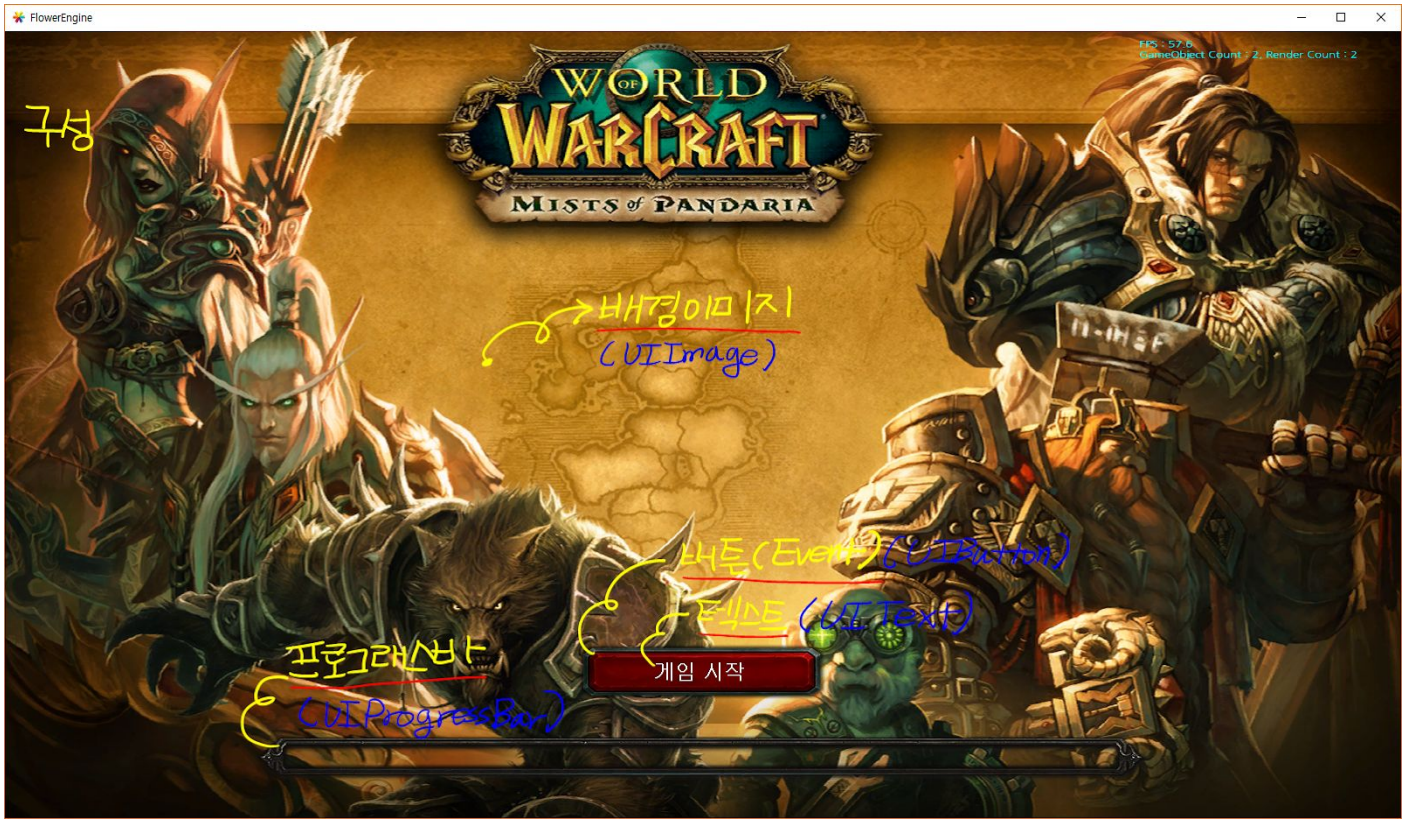
Init() 함수 : 유니티나 언리얼 엔진 에디터에서 해주는 역할을 코드로 작성하여 UI 레이아웃을 구성하였습니다.
OnClick() 함수 : 버튼이 클릭되었을 때, SceneManager를 통해 씬을 전환합니다.

SceneLoading.cpp

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

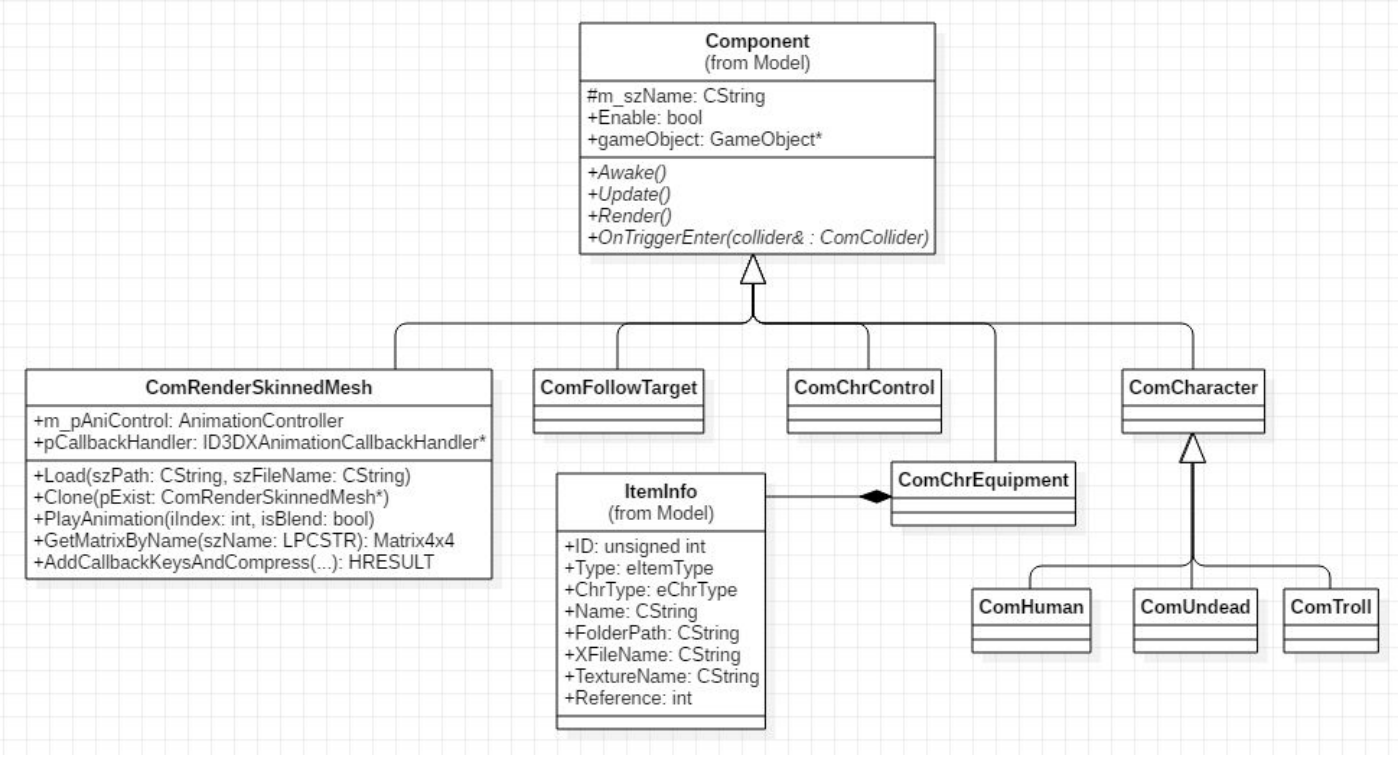
```
	float fScreenWidth = DXUTGetWindowWidth();
	float fScreenHeight = DXUTGetWindowHeight();

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

	// 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);
}
```

다음과 같이 구성하였습니다.


FlowerEngine
FPS : 57.6
GameObject Count : 2, Render Count : 2
구성
WORLD OF WARCRAFT
MISTS of PANDARIA
배경이미지
(UIImage)
버튼(Event)(UIButton)
텍스트 (UIText)
게임 시작
프로그래스바
(UIProgressBar)

캐릭터 개발

캐릭터의 전체적인 구성



설계

캐릭터 게임오브젝트의 구성요소들은 모두 Component에서 상속받아서 구현하여 캐릭터 GameObject에 AddComponent합니다.

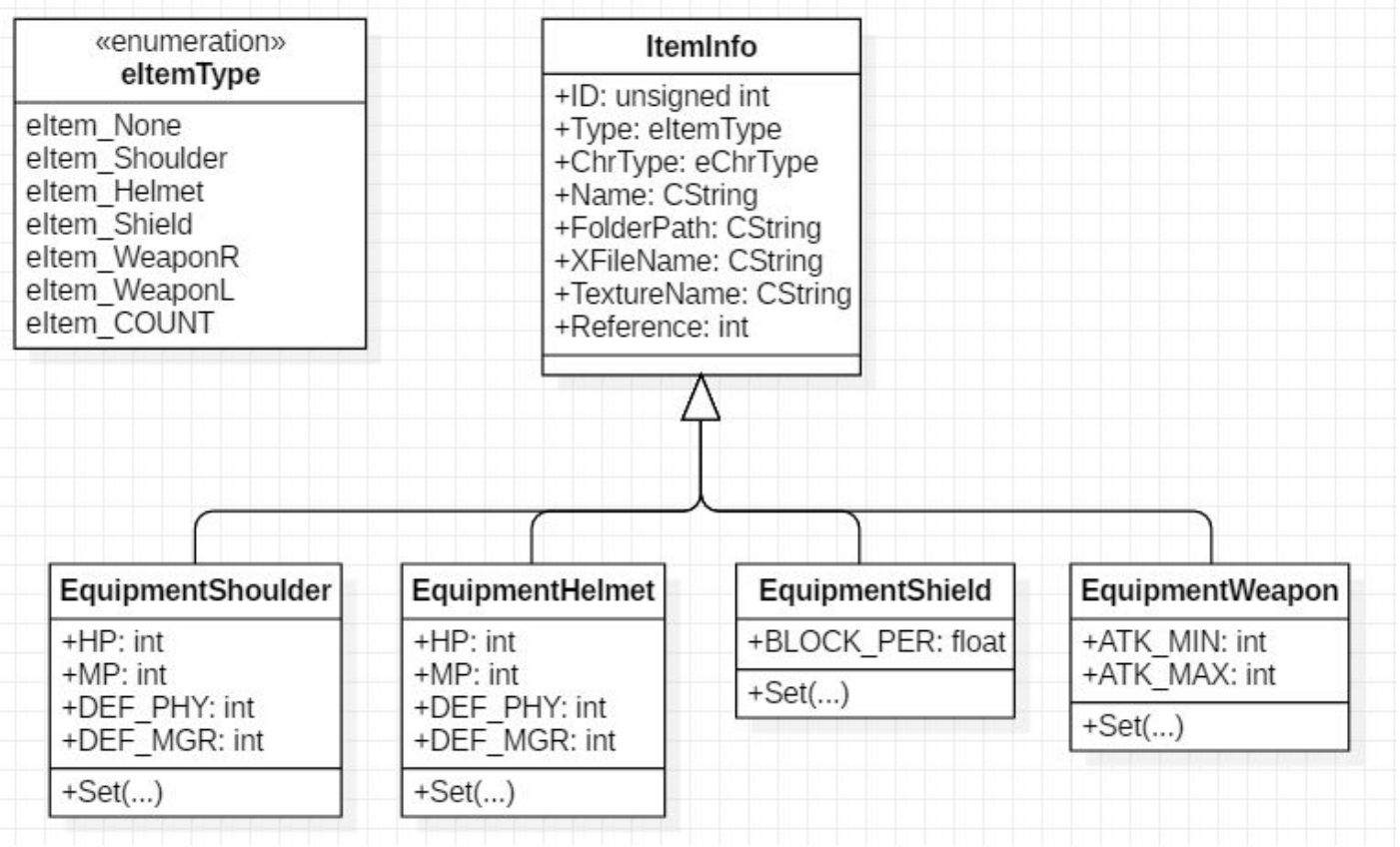
설명

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, CString szChrName, 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"));
	// 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);
	// 이 게임 오브젝트는 뭔가 발사 가능(Skill사용시 필요한 구성요소)
	pGOChr->AddComponent(new ComShooting("ComShooting"));
	// 이 게임 오브젝트는 이름추가 가능
	ComText3D* pChrName = new ComText3D("ComText3D");
	pChrName->SetChrName(szChrName);
	pChrName->SetChrNamePos(pos);
	pGOChr->AddComponent(pChrName);

	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.

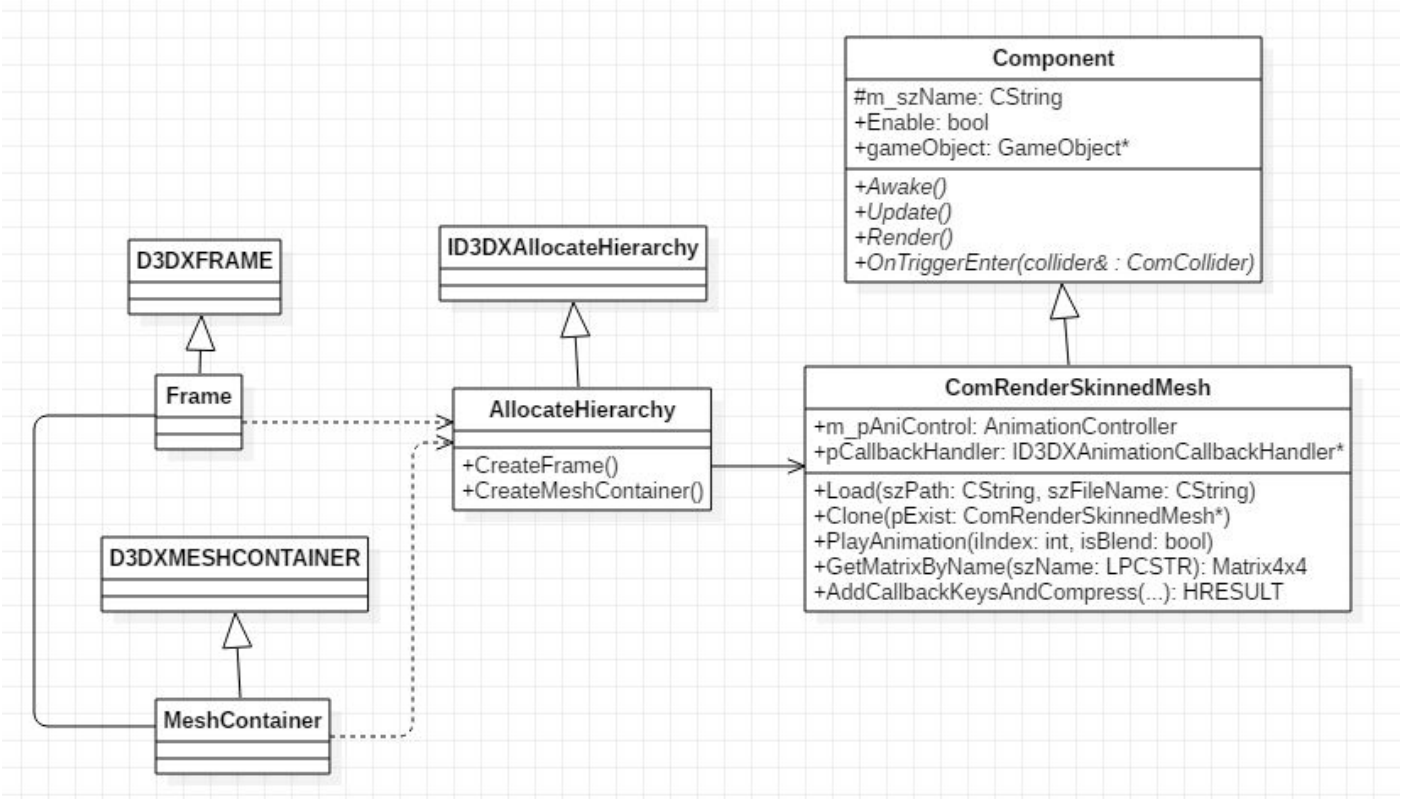
모든 함수의 매개변수에서 ItemInfo* 포인터를 사용하여 아이템 정보에 접근합니다. 그리고 (EquipmentShoulder*)pItemInfo 다형성을 이용하여 하위 클래스의 정보에 접근하여 사용합니다.

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.

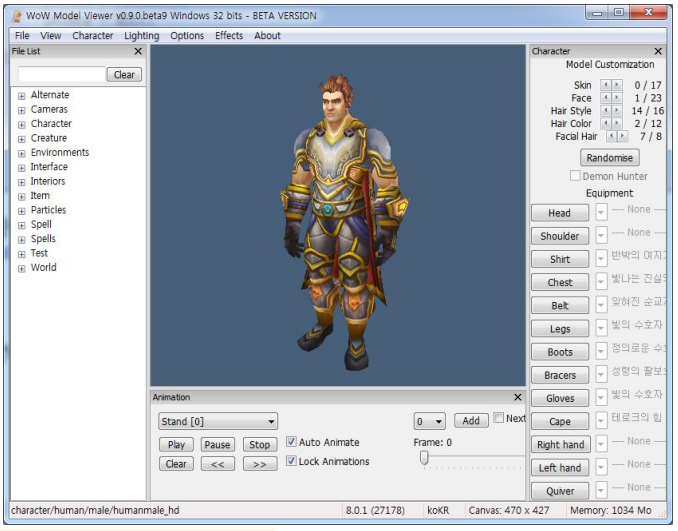
Load함수를 통해 .X파일을 읽어오며 AllocateHierachy를 통해 뼈대(Frame)와 메쉬(MeshContainer)들을 만듭니다. Clone은 기존 리소스를 가지고 복제해주는 함수입니다. PlayAnimation 함수는 인덱스를 통해서 애니메이션을 재생시켜 줍니다.

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,
```

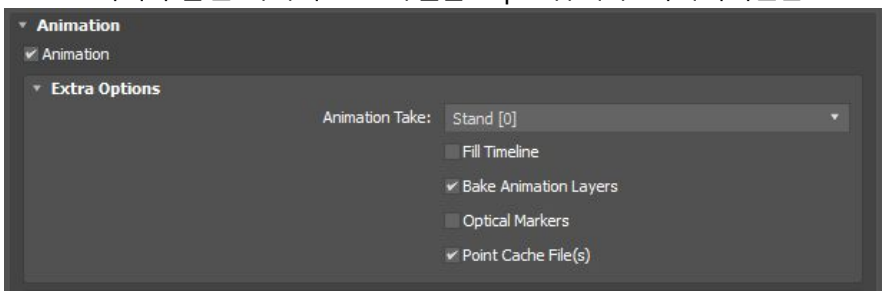
```
	D3DXKEY_CALLBACK aKeys[],
	DWORD dwCompressionFlags,
	FLOAT fCompression, D3DXPLAYBACK_TYPE playType)
{
	HRESULT hr;
	LPD3DXCOMPRESSEDANIMATIONSET pASNew = NULL;
	LPD3DXBUFFER pBufCompressed = NULL;

	// 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;
}
```

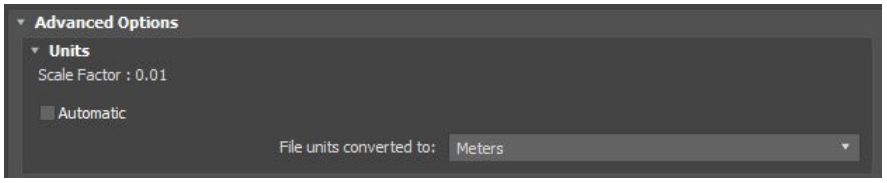
캐릭터 추출



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/

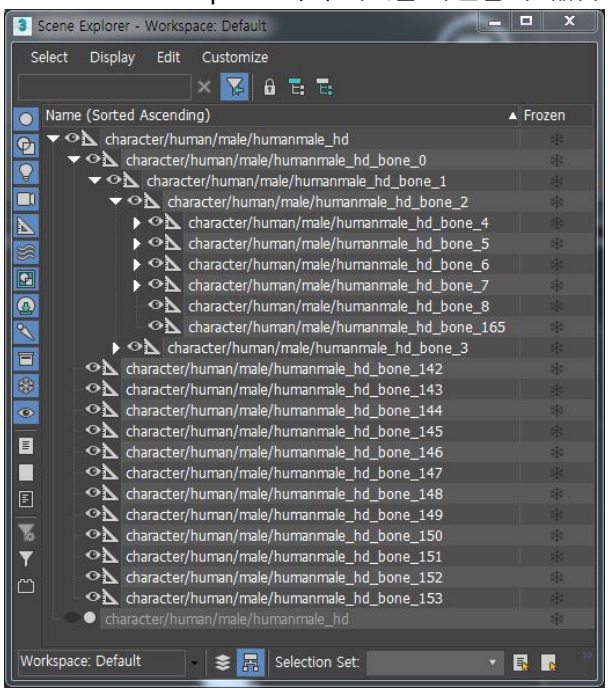
3D Max에서 추출된 캐릭터 FBX 파일을 Import합니다. 애니메이션은 Stand로 놓고 단위는 Meters를 사용합니다.

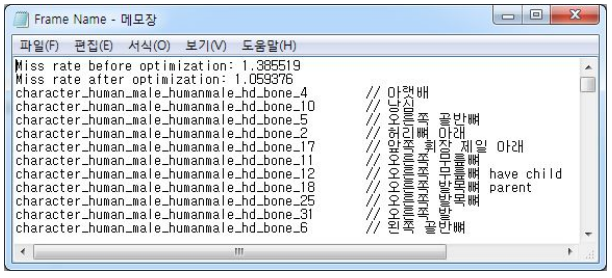




축은 MAX축 그데로 사용합니다. 위 그림은 FRONT에서 바라본 방향입니다.
매쉬 부분은 원래 나누어서 작업되어야 합니다. (실무에서 매쉬 부분은 디자이너와 협의)

Tools > Scene Explorer에서 뼈대를 확인할 수 있습니다.



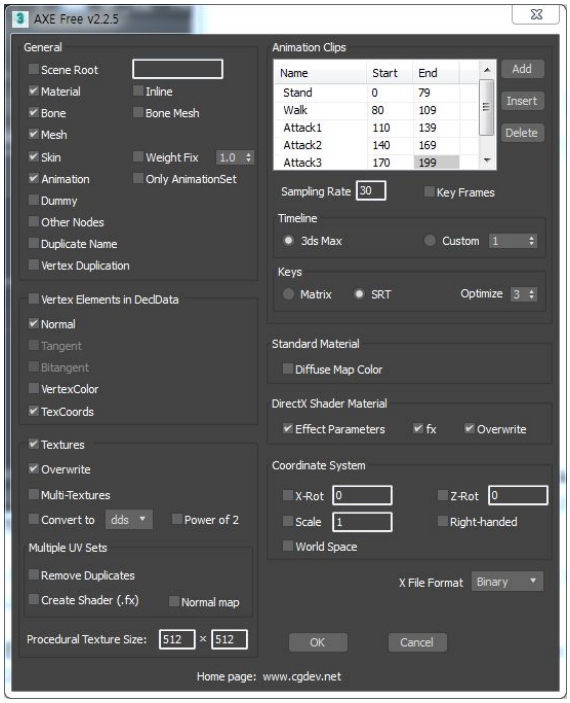


뼈대 분석 파일을 따로 만들었습니다.

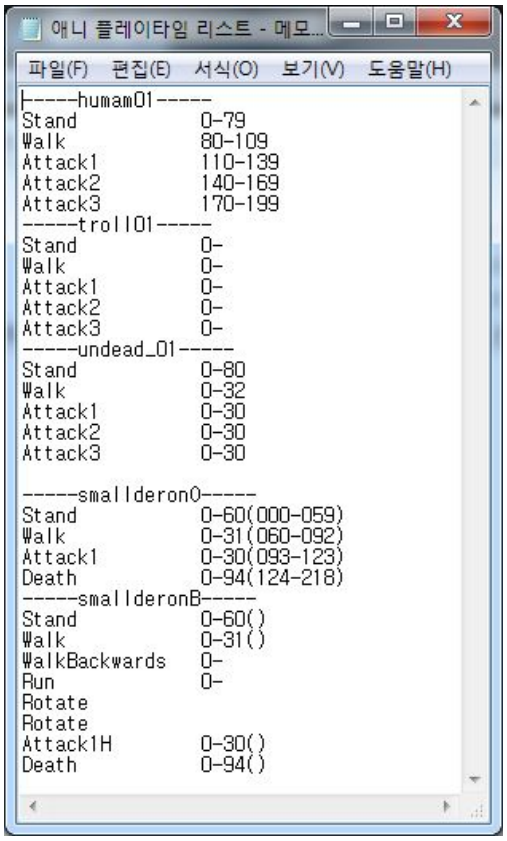
애니메이션을 편집하기 위해 Walk FBX를 Import 한 다음 Animation > Save Animation 합니다. 그 다음 통합본 캐릭터 파일에 Load Animation (프레임 번호 설정) 하여 애니메이션을 합칩니다. 애니메이션 작업시에는 위 메쉬 부분을 선택 해제 하고 작업해야 합니다.

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.

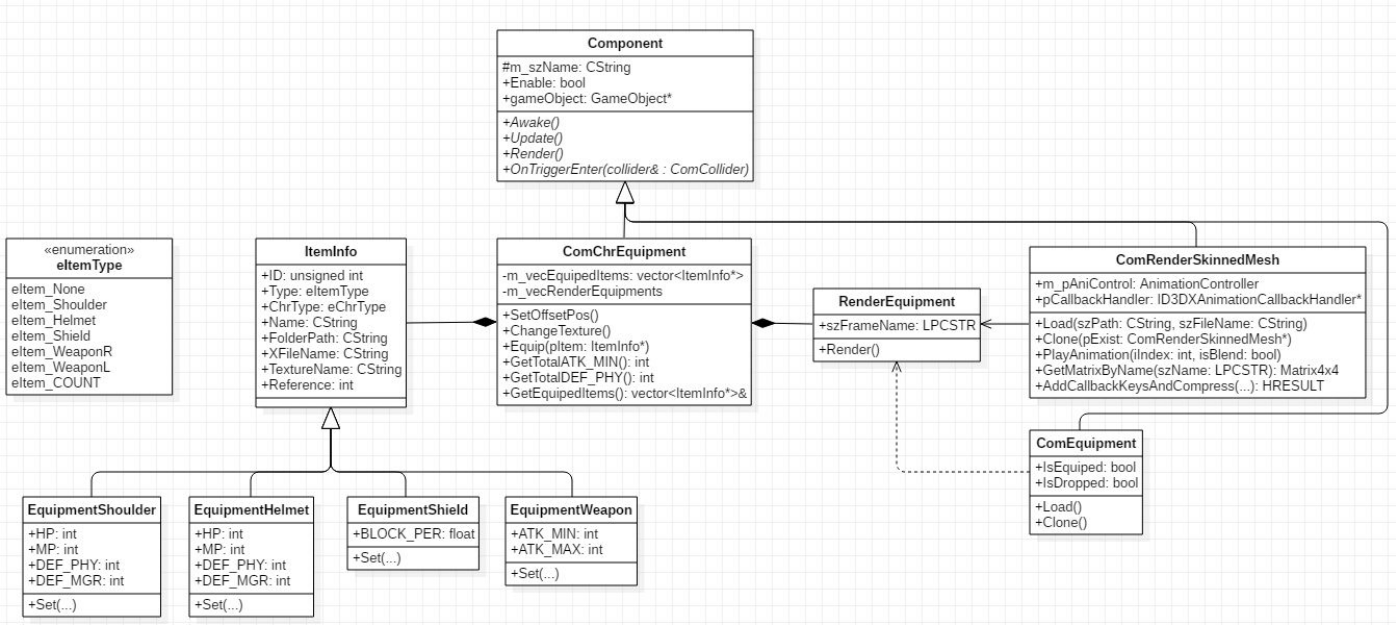


팀워크에 필요한 각 애니메이션 정보는 txt파일로 저장해두었습니다.



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;

	// eRenderEquipment랑 eItemType이랑 코드 정리 생각해 볼 것.
	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 벡터 자료구조에 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.
아이템 정보(ItemInfo)는 여러 객체에서 참조할 수 있기 때문에 참조카운트(ReferenceCount)를 증가시켜서 메모리 관리를 합니다.

```
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();
}
```

장비가 장착된 모습

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.

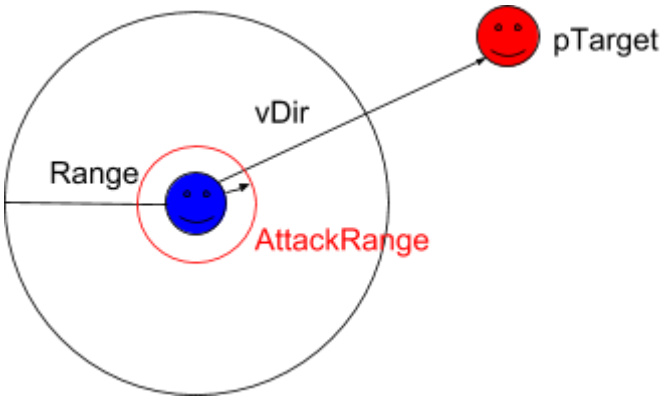
ComCharacter.cpp

```
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)

이 구성요소(Component)는 비행슈팅 게임을 개발할 때 만들어 두었던 것을 재사용한 것입니다. 이 코드를 작성할 때 다른경우에도 사용할 수 있도록 만들었고 조금 수정하였습니다.

설정한 특정거리 Range 안에 들어오면 Target을 향해 가게 됩니다. 이 방향벡터는 T - C 하면 얻어지고 정규화(Normalize) 한 후 Speed값을 곱하여 사용합니다.



ComFollowTarget.h

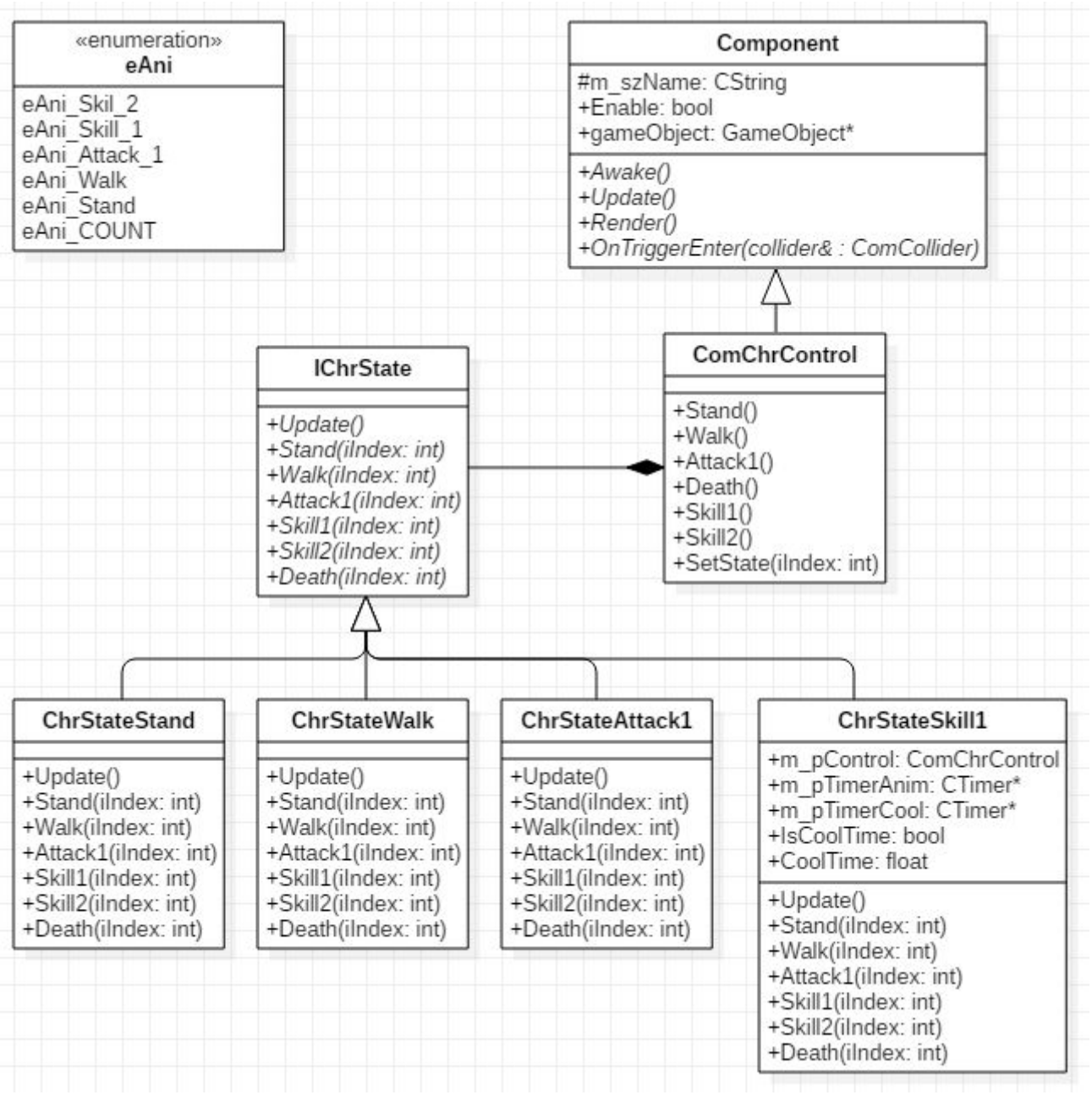
```
#pragma once
#include "stdafx.h"

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

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

	// 앞 방향 벡터
	Vector3 vDir;
	// 속도
	float fMoveSpeed;
	// 이 범위 안에 있으면 따라감
	float fRange;
	// 따라가야 할 타겟
	GameObject * pTarget;
	// 타겟을 따라가는 중 여부
	bool IsFollowing;
	// 공격 가능 여부
	bool AbleAttack;

	// 선형 보간 속도
	float accellation;

private:
	float m_fLerp;
};
```

ComFollowTarget.cpp

```cpp
#include "stdafx.h"
#include "ComFollowTarget.h"

ComFollowTarget::ComFollowTarget(CString szName) :
	Component(szName),
	fMoveSpeed(0.02f),
	accellation(0.0f),
	m_fLerp(0.003f),
	IsFollowing(false),
	AbleAttack(false),
	pTarget(NULL),
	fRange(20.0f)
{
}

ComFollowTarget::~ComFollowTarget()
{
}

void ComFollowTarget::Update()
{
	if (pTarget == NULL)
	{
		// 초기화
		AbleAttack = false;
		IsFollowing = false;
		return;
	}

	// Target 위치와 나의 위치가 특정 거리 이하이면
	float fDistance = ComTransform::Distance(gameObject, pTarget);

	// 초기화
	AbleAttack = false;
	IsFollowing = false;

	if (fDistance < 1.0f)
		AbleAttack = true;
	else if (fDistance < fRange)
	{
		IsFollowing = true;
		// 플레이어를 바라보는 방향 벡터
		vDir = pTarget->transform->GetPosition() - gameObject->transform->GetPosition();
		D3DXVec3Normalize(&vDir, &vDir);

		// 일단은 Y축으로만 회전하자
		float angleY = Vector::GetAngleY(&vDir);
		gameObject->transform->SetRotation(0.0f, angleY, 0.0f);

		Vector3 move;

		// 속도가 0.02f라면
		vDir *= fMoveSpeed;

		// 속도벡터 곱하는 방법
```

```
            gameObject->transform->Translate(vDir);

            // 내위치 ~ 상대위치, 선형보간 방법
            //D3DXVec3Lerp(&move, &gameObject->transform->GetPosition(),
&pTarget->transform->GetPosition(), m_fLerp);
            //gameObject->transform->SetPosition(move);

            /*m_fLerp += accellation;

            if (m_fLerp >= 1.0f)
                  m_fLerp = 0.0f;*/
      }
}
```

상태기계 (IChrState)



설계

IChrState 인터페이스에서 상속 받아서 각 상태를 구현해야 합니다. 현재는 애니메이션 변경 밖에 없습니다. 상속된 상태들을 일반적인 것과 특수한 것으로 나누어야 할 것 같습니다.
이 상태들은 ComChrControl에서 SetState를 통해 상태를 변경합니다.

ChrStateSkill1을 보면 상속된 상태에서 쿨타임을 적용하는 것을 볼 수 있습니다.

인터페이스에서 순수 가상함수를 통해 모든 상태를 구현하도록 하였는데 상태변환이 가능한 함수만 재정의 해서 구현하도록 하여 리팩토링 해야할 것 같습니다.

계속 코드를 정리해가면서 만들어야 하는 부분입니다.

설명

상태변환 하는 과정은 현재 상태가 ChrStateStand라면 m_pCurrentState->Walk(eAni_Walk);를 호출하면 ChrStateStand 클래스의 Walk함수가 호출되며 상태가 변환됩니다.

```
void ComChrControl::SetState(int iIndex)
```

```
{
	m_pCurrentState = m_vecState[iIndex];
}

void ComChrControl::Walk(float fDeltaZ)
{
	// 걸어갈 때는 마우스 이동 하지 않음 (마우스 이동시 키보드 이동하면 속도 2배되는 버그 방지)
	IsMoveToPoint = false;

	// 현재 상태에서 Walk로
	m_pCurrentState->Walk(eAni_Walk);
	GetHeight();

	Vector3 vecForward;
	gameObject->transform->GetForward(vecForward);
	Vector3 forward = fDeltaZ * vecForward * m_pCharacter->Status.MOVE_SPEED;
	gameObject->transform->Translate(forward);
}
```

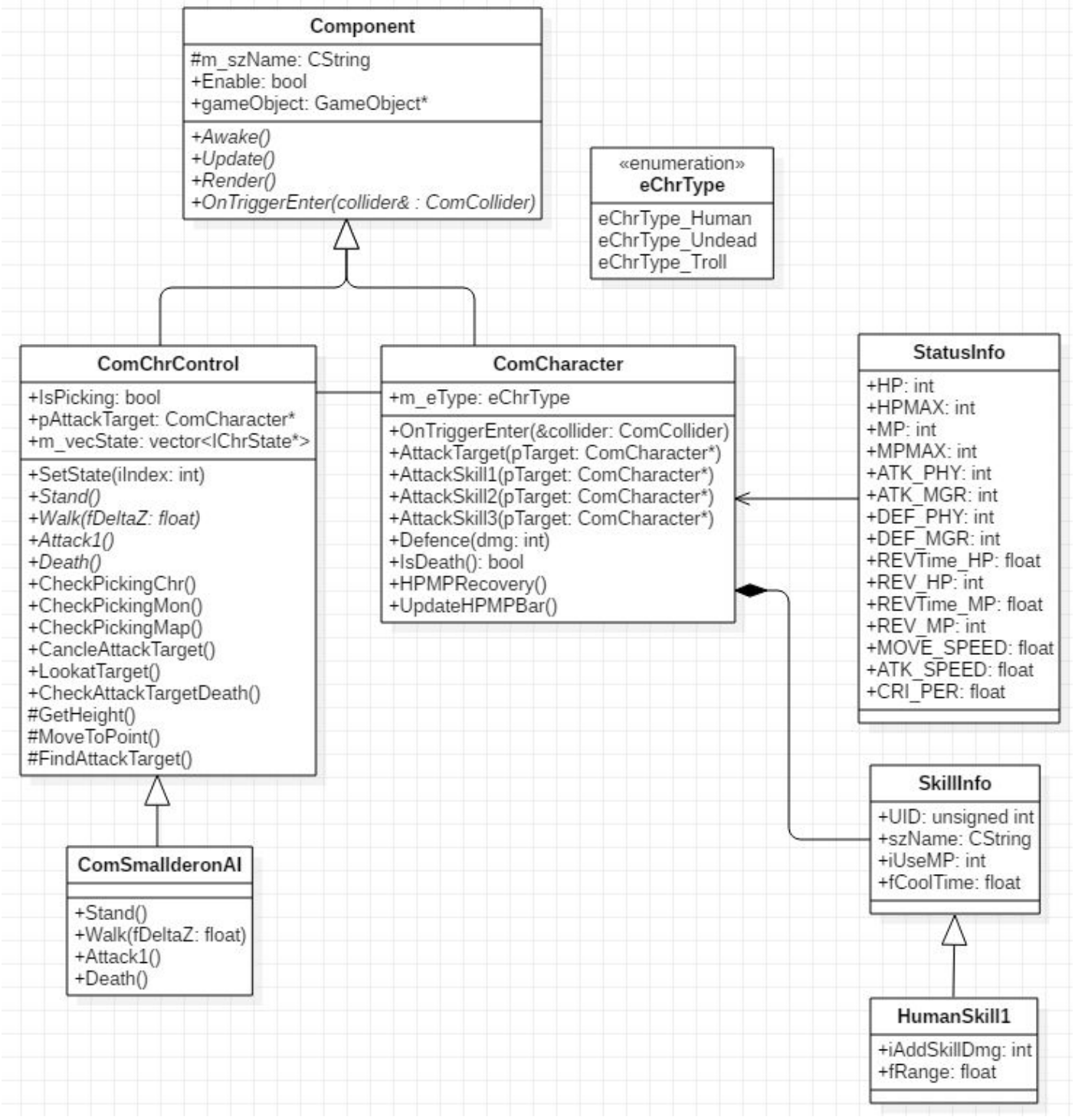
위에서 ComChrControl에 Walk라는 함수가 따로 있는데 여기에 있는 코드 내용들을 ChrWalk안으로 넣어야 하지 않을까 고민중에 있습니다.

모든 상태는 Awake에서 메모리 할당하여 미리 만들어 둡니다.

```
void ComChrControl::Awake()
{
	Init();

	m_vecState.resize(eAni_COUNT);
	m_vecState[eAni_Stand] = new ChrStateStand(this);
	m_vecState[eAni_Walk] = new ChrStateWalk(this);
	m_vecState[eAni_Attack_1] = new ChrStateAttack1(this);
	m_vecState[eAni_Skill_1] = new ChrStateSkill1(this);
	m_vecState[eAni_Skill_2] = new ChrStateSkill2(this);

	m_pCurrentState = m_vecState[eAni_Stand];
	Stand();

	if (m_pMap)
	{
		m_pMap->UpdateIndexBufferQuadTree();
		GetHeight();
	}
}
```

캐릭터 컨트롤과 몬스터AI (ComChrControl)

‘이 캐릭터는 컨트롤 할 수 있다’라는 구성요소 입니다. 보통 입력 관련된 처리를 여기에 코드 작성을 합니다. 필요한 기능들을 함수로 구현하였습니다.



설계

ComChrControl과 ComCharacter는 양방향 연관관계입니다. ComSmallderonAI는 스몰데론이라는 몬스터의 인공지능 구성요소입니다. 상태기계를 상속받아 사용합니다.
현재 중복되는 기능들이 있어보여서 설계 리팩토링을 해주어야 하는 시점입니다.

ComChrControl.h

```
class ComChrControl : public Component
{
public:
```

```
	ComChrControl(CString szName);
	virtual ~ComChrControl();

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

	// 초기화 관련
	void Init();

	//상태 기계
	void SetState(int iIndex);
	virtual void Stand();
	virtual void Walk(float fDeltaZ);
	virtual void Attack1();
	virtual void Death();

	// 이 객체가 픽킹되었는지 여부를 검사합니다. (캐릭터, 몬스터, 맵)
	void CheckPickingChr();
	void CheckPickingMon();
	void CheckPickingMap();

	// 공격대상을 취소합니다.
	void CancleAttackTarget();
	// 대상을 바라봅니다.
	void LookatTarget();
	// 공격대상이 죽었을 때 처리를 합니다.
	void CheckAttackTargetDeath();

protected:
	// 맵에서 높이를 얻어옵니다.
	void GetHeight();
	// 맵을 클릭하면 해당 위치로 이동합니다.
	void MoveToPoint();
	// 공격할 대상을 찾습니다.
	void FindAttackTarget();

public:
	// 이 객체가 픽킹되었는지 여부
	bool IsPicking;
	ComRenderSkinnedMesh * pAnimation;
	// 공격하고자 하는 타겟
	ComCharacter* pAttackTarget;
	ComCharacter* m_pCharacter;
	// 상태들
	vector<IChrState*> m_vecState;
	// 현재 상태
	IChrState * m_pCurrentState;

protected:
	ComObjMap * m_pMap;
	// 타겟으로 따라감
	ComFollowTarget* m_pFollow;
	// 이동하고하 하는 위치
	Vector3 vMoveToPoint;
```

```
	// 특정 좌표로 이동 여부
	bool IsMoveToPoint;
	// 초기 캐릭터 셋팅시 땅 위에 있는지 여부
	bool IsGroud;
};
```

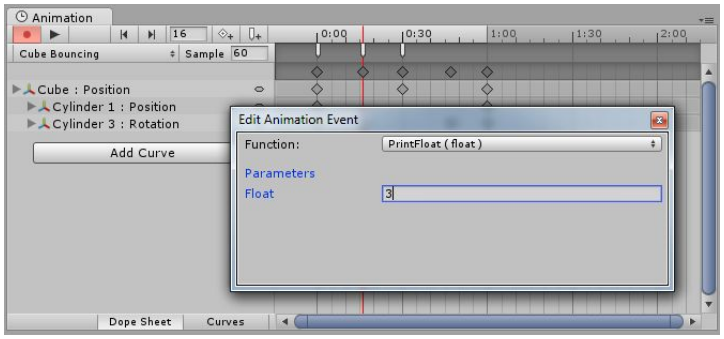
ComChrControl.cpp

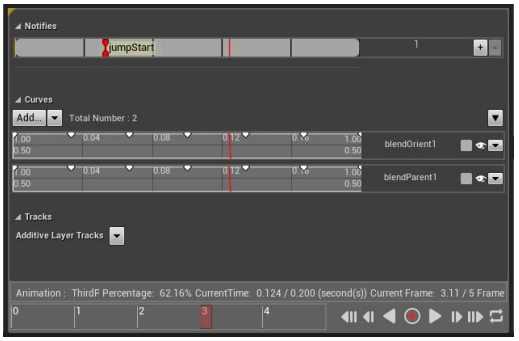
```
void ComChrControl::Update()
{
	if (m_pCurrentState != m_vecState[eAni_Skill_1])
	{
		// 캐릭터 회전
		if (Input::KeyPress('A') || Input::KeyPress(VK_LEFT))
			gameObject->transform->RotateY(-0.1f);
		if (Input::KeyPress('D') || Input::KeyPress(VK_RIGHT))
			gameObject->transform->RotateY(0.1f);

		// 캐릭터 이동
		if (Input::KeyPress('W') || Input::KeyPress(VK_UP))
		{
			CancleAttackTarget();
			Walk(1);
		}
		else if (Input::KeyUp('W') || Input::KeyUp(VK_UP))
		{
			CancleAttackTarget();
			Stand();
		}
		if (Input::KeyPress('S') || Input::KeyPress(VK_DOWN))
		{
			CancleAttackTarget();
			Walk(-1);
		}
		else if (Input::KeyUp('S') || Input::KeyUp(VK_DOWN))
		{
			CancleAttackTarget();
			Stand();
		}
	}

	if (Input::ButtonDown(VK_LBUTTON))
		CheckPickingChr();

	if (Input::ButtonDown(VK_RBUTTON))
	{
		CancleAttackTarget();
		CheckPickingMon();
		CheckPickingMap();
	}

	if (m_pFollow != NULL && m_pFollow->IsFollowing)
	{
		m_pCurrentState->Walk(eAni_Walk);
		GetHeight();
	}
```

```
	else if (m_pFollow != NULL && m_pFollow->AbleAttack)
		Attack1();

	MoveToPoint();
	CheckAttackTargetDeath();

	m_pCurrentState->Update();
}
```

애니메이션 이벤트

유니티와 언리얼의 애니메이션 이벤트와 비슷합니다.
해당 프레임이 되었을 때 이벤트가 발생하여 지정한 함수가 호출되는 방식입니다.





툴에서 UI를 통해 애니메이션 이벤트를 등록해 주는 것과 달리 코드로 작성해 주어야 합니다.

```
void ComHuman::SetAniEvent()
{
	if (m_pAnimation == NULL)
		return;

	// 애니메이션 키프레임셋
	vector<LPD3DXKEYFRAMEDANIMATIONSET> vecKeyFrameAnimSet;

	vecKeyFrameAnimSet.resize(eAni_COUNT);

	for (int i = eAni_Skill_2; i < eAni_COUNT; ++i)
		m_pAnimation->m_pAniControl->GetAnimationSet(i,
(LPD3DXANIMATIONSET*)&vecKeyFrameAnimSet[i]);

	// Register 하는 순서데로 Animation Index가 설정되기 때문에 미리 모두 Unregister 한다.
	for (int i = eAni_Skill_2; i < eAni_COUNT; ++i)
		m_pAnimation->m_pAniControl->UnregisterAnimationSet(vecKeyFrameAnimSet[i]);

	// 초당 발생하는 애니메이션 키 프레임 틱의 수를 가져옵니다.
	float fSrcTime = vecKeyFrameAnimSet[eAni_Attack_1]->GetSourceTicksPerSecond(); // 4800

	// eAni_Attack_1 총 프레임 수 : 29
	// eAni_Attack_1 때릴때 애니 프레임 Number : 12
	// 비례식 12 : 29 = x : 4800(SrcTime)
	float x = fSrcTime * 12 / 29;

	// 키 이벤트 콜백
	D3DXKEY_CALLBACK attackKey;
	attackKey.pCallbackData = this;
	attackKey.Time = x;
```

```
	// 키 이벤트 콜백
	D3DXKEY_CALLBACK skill1Key;
	skill1Key.pCallbackData = this;
	skill1Key.Time = x;

	// eAni 순서데로 추가한다.
	m_pAnimation->AddCallbackKeysAndCompress(vecKeyFrameAnimSet[eAni_Skill_2], 0, NULL,
D3DXCOMPRESS_DEFAULT, 1.0f, D3DXPLAY_ONCE);
	m_pAnimation->AddCallbackKeysAndCompress(vecKeyFrameAnimSet[eAni_Skill_1], 1,
&skill1Key, D3DXCOMPRESS_DEFAULT, 1.0f, D3DXPLAY_ONCE);
	m_pAnimation->AddCallbackKeysAndCompress(vecKeyFrameAnimSet[eAni_Attack_1], 1,
&attackKey, D3DXCOMPRESS_DEFAULT, 1.0f);
	m_pAnimation->AddCallbackKeysAndCompress(vecKeyFrameAnimSet[eAni_Walk], 0, NULL,
D3DXCOMPRESS_DEFAULT, 1.0f);
	m_pAnimation->AddCallbackKeysAndCompress(vecKeyFrameAnimSet[eAni_Stand], 0, NULL,
D3DXCOMPRESS_DEFAULT, 1.0f);

	vecKeyFrameAnimSet.clear();
}
```

이벤트 핸들러의 선언 : DirectX의 ID3DXAnimationCallbackHandler를 상속 받아 사용합니다.
HandleCallback 함수가 호출됩니다.

```
class AttackHandler : public ID3DXAnimationCallbackHandler
{
	HRESULT CALLBACK HandleCallback(THIS_ UINT Track, LPVOID pCallbackData);
};

class Skill1Handler : public ID3DXAnimationCallbackHandler
{
	HRESULT CALLBACK HandleCallback(THIS_ UINT Track, LPVOID pCallbackData);
};
```

이벤트 핸들러의 호출 : ID3DXAnimationController의 AdvanceTime 함수에 이벤트 핸들러를 설정해 줍니다.

```
void ComRenderSkinnedMesh::UpdateAnimation(AnimationController pAniControl)
{
	float fDeltaTime = GetElapsedTime();

	if (pCallbackHandler != NULL)
		pAniControl->AdvanceTime(fDeltaTime, pCallbackHandler);
	else
		pAniControl->AdvanceTime(fDeltaTime, NULL);
}
```

캐릭터 또는 몬스터 픽킹 방법

캐릭터에 큐브를 씌워 놓았는데 마우스 우클릭시 광선을 쏘아 그 큐브의 삼각형들과 충돌하였는지 여부를 가지고 캐릭터 또는 몬스터 픽킹 여부를 결정합니다.

캐릭터 (ComCharacter)

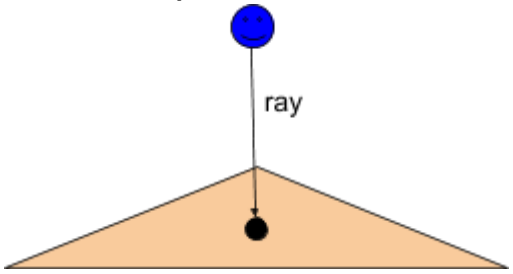
캐릭터의 행동 공격, 방어, 죽음, 스킬사용, 능력치 처리를 하기 위한 구성요소입니다.

ComCharacter.h

```
// 캐릭터의 공통요소입니다.
class ComCharacter : public Component
{
public:
	ComCharacter(CString szName);
	virtual ~ComCharacter();

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

	// 다른 콜라이더에 충돌했을 때 호출되는 함수
	virtual void OnTriggerEnter(ComCollider &collider) override;

	// 타겟을 공격합니다.
	void AttackTarget(ComCharacter* pTarget);

	// 스킬을 사용하여 타겟을 공격합니다.
	virtual void AttackSkill1(ComCharacter* pTarget) {}
	virtual void AttackSkill2(ComCharacter* pTarget) {}
	virtual void AttackSkill3(ComCharacter* pTarget) {}

	// 방어를 합니다.
	void Defence(int dmg);

	// 죽었는지 여부를 반환합니다.
	bool IsDeath();

	// 시간에 따라 HP가 회복됩니다.
	void HPMPRecovery();

	// UI 관련
	void UpdateHPMPBar();

protected:
	void Init();

protected:
	// 캐릭터 타입
	eChrType m_eType;

	// 공격할 타겟
	ComCharacter* m_pAttackTarget;

	// 장비 장착
	ComChrEquipment* m_pChrEquipment;

public:
```

```
	// 능력치
	StatusInfo Status;

	ComRenderSkinnedMesh* m_pAnimation;
	AttackHandler* m_pAttackHandler;
	Skill1Handler* m_pSkill1Handler;

protected:
	UIProgressBar* m_pHPBar;
	UIProgressBar* m_pMPBar;

	CTimer* m_pTimerHPRec;
	CTimer* m_pTimerMPRec;
};
```

지형타기 (광선과 폴리곤 Picking 검사) GetHeight함수



캐릭터는 삼각형들로 이루어져 있는 지형(Terrain) 위를 돌아다니는데 광선과 3개의 정점으로 Picking된 지점을 얻어올 수 있습니다. 이 값으로 캐릭터의 y위치를 설정해줍니다.

```
if (D3DXIntersectTri(&v1, &v2, &v3, &rayPos, &rayDir, NULL, NULL, &distance))
{
	height = rayPos.y - distance;
	return true;
}
```

전투 시스템

전투는 몬스터를 마우스로 우클릭시 그곳으로 가서 공격 상태를 취해 공격 애니메이션이 실행되고 설정되어 있는 이벤트가 발생하고 공격 함수가 호출되고 피격자는 방어 함수가 호출됩니다.

STEP 1. 마우스 우클릭시 기존의 공격 타겟을 해제하고 픽킹된 몬스터가 있는지 확인한다. 픽킹된 몬스터가 없다면 클릭된 지형으로 이동한다.

```
if (Input::ButtonDown(VK_RBUTTON))
{
	m_pCharacter->CancleAttackTarget();
	// 몬스터가 Picking되지 않으면 Map으로 이동 (중복 픽킹 방지)
	if (m_pCharacter->CheckPickingMon() == false)
		CheckPickingMap();
}
```

몬스터가 픽킹되었는지 여부를 반환하는 함수입니다.
마우스 위치를 가져와 광선으로 만들고 몬스터의 충돌박스(Cube)의 삼각형들을 가지고 픽킹 여부를 판단합니다.
몬스터가 픽킹 되었으면 따라가는 객체를 지정해주고 공격대상을 지정해줍니다.

```
bool ComCharacter::CheckPickingMon()
{
	if (m_pFollow == NULL)
		return false;

	Mouse* pMouse = Input::GetInstance()->m_pMouse;
	Vector3 mousePos = Input::GetInstance()->m_pMouse->GetPosition();

	list<GameObject*> listMonster = GameObject::FindAll(eTag_Monster);

	for (auto & o : listMonster)
	{
		ComRenderCubePN* pCube = (ComRenderCubePN*)o->GetComponent("ComRenderCubePN");

		// 죽었을 때는 몬스터 픽킹을 하지 않는다
		if (pCube->Enable == false)
			continue;

		Ray ray = Ray::RayAtWorldSpace(mousePos.x, mousePos.y);

		vector<Vector3>& vertices = pCube->GetVector();
		for (size_t i = 0; i < vertices.size(); i += 3)
		{
			float dist = 0;
			bool pickMon = ray.CalcIntersectTri(&vertices[i], &dist);

			if (pickMon == true)
			{
				// 몬스터를 따라간다.
				m_pFollow->pTarget = o;
				m_pFollow->fMoveSpeed = Status->MOVE_SPEED;
				pAttackTarget = (ComSmallderonAI*)o->GetComponent("ComCharacter");
				return true;
			}
		}
	}
```

```
	return false;
}
```

STEP 2. 캐릭터의 Update함수에서는 따라가는 객체가 있으면 Walk상태로 공격 범위 안에 들어오면 공격 함수를 호출하게 됩니다. 공격함수에는 현재 상태 변환만 있습니다. (함수를 정리할지 생각해봐야겠습니다.)

```
void ComCharacter::Update()
{
	if (m_pFollow != NULL && m_pFollow->IsFollowing)
	{
		m_pCurrentState->Walk(eAni_Walk);
		GetHeight();
	}
	else if (m_pFollow != NULL && m_pFollow->AbleAttack)
		Attack1();
}

void ComCharacter::Attack1()
{
	// 현재 상태에서 Attack1로
	m_pCurrentState->Attack1(eAni_Attack_1);
}
```

STEP 3. 공격 애니메이션이 실행되면 등록해둔 프레임에 이벤트가 발생합니다. 이벤트에서는 타겟을 공격하라라는 함수를 실행하고 매개변수로 공격타겟을 넘겨줍니다.

```
HRESULT AttackHandler::HandleCallback(UINT Track, LPVOID pCallbackData)
{
	// 특정 프레임에서 공격
	ComCharacter* pChr = (ComCharacter*)pCallbackData;

	// 죽어서 없으면
	if (pChr->pAttackTarget == NULL)
		return S_OK;
	pChr->AttackTarget(pChr->pAttackTarget);

	return S_OK;
}
```

STEP 4. 공격할 때는 대상을 바라봅니다. 그다음 공격력으로 데미지를 계산합니다. 현재 데미지 공식은 장착된 장비들에게서 총최소공격력을 가져오고 캐릭터의 기본 공격력에 더해주게 됩니다. 그리고 이 데미지와 크리티컬 여부를 계산해서 타겟의 방어함수를 호출합니다.
현재 물리 공격력만 계산이 되는데 마법 공격, 원거릭 공격등을 추가하게 되면 함수가 추가 될 수도 있고 구조가 조금 달라질 것입니다.

```
void ComCharacter::AttackTarget(ComCharacter * pTarget)
{
	pAttackTarget = pTarget;
	LookatTarget();

	// 총 공격력 계산 (내 공격력 + 장비 공격력)
	int equipmentDmg = 0;
	if (m_pChrEquipment)
		equipmentDmg = m_pChrEquipment->GetTotalATK_MIN();

	int dmg = Status->ATK_PHY + equipmentDmg;

	pTarget->Defence(dmg, Status->IsCritical());
```

```
	// 다시 기본 핸들러로
	m_pAnimation->pCallbackHandler = m_pAttackHandler;
}

bool StatusInfo::IsCritical()
{
	int iRandom = rand() % 100;
	if (iRandom < CRI_PER) // 크리티컬 확률
	{
		return true;
	}
	return false;
}
```

STEP 5. 피격자는 방어를 하게 됩니다. 총 방어력은 캐릭터 기본 방어력 + 장착하고 있는 장비의 방어력입니다.
데미지에서 방어수치의 반만큼 빼줍니다. 이 계산 공식들은 바뀔 수 있습니다.
최종적으로 UI를 표시해주게됩니다. (현재 ComCharacterUI관련 컴포넌트로 리팩토링 하기 전 코드)

```
void ComCharacter::Defence(int Damage, bool bCritical)
{
	// 총 방어력 계산 (내 방어력 + 장비 방어력)
	int equipmentDef = 0;
	if (m_pChrEquipment)
		equipmentDef = m_pChrEquipment->GetTotalDEF_PHY();

	int def = Status->DEF_PHY + equipmentDef;

	Damage -= (def / 2);
	if (bCritical)
		Damage *= 2.0f; // 크리티컬시 2배 데미지

	// HP 차감
	Status->HP -= Damage;

	CString szDmg;
	szDmg.Format(L"%d", Damage);

	if (bCritical)
	{
		// 크리티컬시 노란색 데미지 표시
		m_pComUICritical->SetText(szDmg, Color(1, 1, 0, 1), 0.6f, true);
		m_pComUICritical->Enable = true;
		m_pTimerCritical->Reset();
	}
	else
	{
		// UI 데미지 표시
		m_pComUIDamage->SetText(szDmg, Color(1, 0, 0, 1), 0.4f, true);
		m_pComUIDamage->Enable = true;
		m_pTimerDamage->Reset();
	}

	// UI 갱신
	UpdateUI();
}
```

스킬 시스템

캐릭터는 스킬을 사용하여 전투를 수월하게 진행할 수 있습니다. 파티클 시스템이 개발 된다면 화려한 이펙트를 추가할 수 있습니다.

스킬정보를 설정합니다. 보통 파일이나, 엑셀에서 정보를 읽어오고 쿨타임등 수치들은 서버DB에서 받아와서 설정합니다.

```
void ComHuman::SetSkillInfo()
{
	// 스킬 정보 생성 (파일 또는 엑셀에서 읽어야 하는 부분)
	m_vecSkillInfo.resize(3);

	HumanSkill1* pSkillInfo1 = new HumanSkill1();
	pSkillInfo1->szName = "스킬1";
	pSkillInfo1->UID = 1;
	pSkillInfo1->iAddSkillDmg = 2;
	pSkillInfo1->fRange = 3.0f;
	m_vecSkillInfo[0] = pSkillInfo1;

	SkillInfo* pSkillInfo2 = new SkillInfo();
	pSkillInfo2->szName = "스킬2";
	pSkillInfo2->UID = 2;
	m_vecSkillInfo[1] = pSkillInfo2;

	SkillInfo* pSkillInfo3 = new SkillInfo();
	pSkillInfo3->szName = "스킬3";
	pSkillInfo3->UID = 3;
	m_vecSkillInfo[2] = pSkillInfo3;
}
```

현재 스킬은 스킬 버튼을 클릭시 사용됩니다.

```
void ComHuman::OnClick(UIButton * pSender)
{
	if (pSender->GetButtonName() == "human_skill_1")
	{
		Skill1();
	}
	else if (pSender->GetButtonName() == "human_skill_2")
	{
	}
	else if (pSender->GetButtonName() == "human_skill_3")
	{

	}
}
```

휴먼 스킬1을 사용했을 때 1. 공격 대상이 지정되어 있는가? 2. 쿨타임 중인가? 3. MP가 충분한가? 검사를 하고 모든 조건이 충족되면 스킬 상태로 상태 기계가 변환이 되고 스킬1 애니메이션이 실행되며 설정되어 있는 프레임에서 애니메이션 이벤트가 발생합니다.

```
void ComHuman::Skill1()
{
	ComChrControl* pChrControl =
(ComChrControl*)gameObject->GetComponent("ComChrControl");
```

```
	// 공격 대상이 지정되어 있지 않으면
	if (pChrControl->pAttackTarget == NULL)
	{
		// UI Message: 공격 대상이 지정되어 있지 않습니다.
		return;
	}

	ChrStateSkill1* pStateSkill =
dynamic_cast<ChrStateSkill1*>(pChrControl->m_vecState[eAni_Skill_1]);
	m_pAnimation->pCallbackHandler = m_pSkill1Handler;

	if (pStateSkill->IsCoolTime == true)
	{
		// UI Message : 쿨타임 중입니다.
		return;
	}

	int useMP = m_vecSkillInfo[0]->iUseMP;

	if (Status.MP < useMP)
	{
		// UI Message : MP가 부족합니다.
		return;
	}

	pChrControl->SetState(eAni_Skill_1);
	pChrControl->m_pCurrentState->Skill1(eAni_Skill_1);

	// MP 사용
	Status.MP -= m_vecSkillInfo[0]->iUseMP;

	// UI 갱신
	UpdateHPMPBar();
}
```

특정 프레임에서 애니메이션 이벤트가 발생되면 그 캐릭터의 AttackSkill1 함수를 실행하며 공격 대상을 매개변수로 넘겨줍니다.

```
HRESULT Skill1Handler::HandleCallback(UINT Track, LPVOID pCallbackData)
{
	// 특정 프레임에서 공격
	ComCharacter* pChr = (ComCharacter*)pCallbackData;
	ComChrControl* pControl =
(ComChrControl*)pChr->gameObject->GetComponent("ComChrControl");

	CString szDebug;
	szDebug.Format(L"Skill1Handler Track : %d %s\r\n", Track, pChr->gameObject->Name());
	OutputDebugString(szDebug);

	// 죽어서 없으면
	if (pControl->pAttackTarget == NULL)
		return S_OK;

	pChr->AttackSkill1(pControl->pAttackTarget);
```

```
	return S_OK;
}
```

AttackSkill1은 가상함수로 되어 있기 때문에 ComCharacter를 상속받은 ComHuman의 AttackSkill1이 호출됩니다.

```
// 캐릭터의 공통요소입니다.
class ComCharacter : public Component
{
public:
	...
	// 스킬을 사용하여 타겟을 공격합니다.
	virtual void AttackSkill1(ComCharacter* pTarget) {}
	virtual void AttackSkill2(ComCharacter* pTarget) {}
	virtual void AttackSkill3(ComCharacter* pTarget) {}
```

휴먼의 스킬1 설명 : 일단 공격하고자 하는 대상과 거리가 스킬 유효범위보다 크면 이 스킬은 실패하게 됩니다. 즉, 적이 데미지를 입지 않습니다.

이 스킬의 데미지는 공격력의 1.5배에 스킬 추가 데미지가 붙습니다.
정확한 공식은 (캐릭터 기본 물리 공격력 + 장비 공격력) * 1.5배 + 스킬 추가 데미지 입니다.

```
void ComHuman::AttackSkill1(ComCharacter * pTarget)
{
	m_pAttackTarget = pTarget;
	ComChrControl* pControl = (ComChrControl*)(gameObject->GetComponent("ComChrControl"));
	pControl->LookatTarget();

	HumanSkill1* pSkill1 = ((HumanSkill1*)m_vecSkillInfo[0]);

	// 두 오브젝트 사이의 거리
	float fDist = ComTransform::Distance(gameObject, m_pAttackTarget->gameObject);

	// 공격 거리가 안돼면 데미지 입지 않음
	if (pSkill1->fRange < fDist)
		return;

	// 총 공격력 계산 (내 공격력 + 장비 공격력)
	int equipmentDmg = 0;
	if (m_pChrEquipment)
		equipmentDmg = m_pChrEquipment->GetTotalATK_MIN();

	// 스킬1 데미지 공식 = (캐릭터 기본 공격력 + 장비 공격력) * 1.5배 + 스킬 추가 데미지
	int dmg = (Status.ATK_PHY + equipmentDmg) * 1.5f + pSkill1->iAddSkillDmg;

	pTarget->Defence(dmg);
}
```

레벨 시스템

캐릭터는 모험중에 경험치를 쌓아 레벨을 올릴 수 있습니다. 모든 스탯에 대한 관련 처리는 Status 클래스에서 담당합니다. 현재 리팩토링이 안돼어 있는 상태입니다. 아마 MonStatInfo와 ChrStatInfo로 나누어야 할 것 같습니다.
레벨에 따라 사용할 수 있는 스킬, 장착할 수 있는 장비, 진행할 수 있는 퀘스트, 들어갈 수 있는 던젼등을 지정해 줄 수 있습니다.

```
class StatusInfo
{
public:
	StatusInfo();
	virtual ~StatusInfo();

	// 캐릭터는 기본적으로 체력(Hit Point)이 있습니다. 이 체력이 전부 소진되면 캐릭터는 사망하게 됩니다.
	int HP;
	int HPMAX;
	// 캐릭터는 기본적으로 마나(Mana Point)가 있습니다. 이 마나가 전부 소진되면 캐릭터는 스킬 또는 마법을 사용할 수 없게 됩니다.
	int MP;
	int MPMAX;
	// 캐릭터는 기본적으로 물리 공격력이 있습니다.
	int ATK_PHY;
	// 캐릭터는 기본적으로 마법 공격력이 있습니다.
	int ATK_MGR;
	// 캐릭터는 기본적으로 물리 방어력이 있습니다.
	int DEF_PHY;
	// 캐릭터는 기본적으로 마법 방어력이 있습니다.
	int DEF_MGR;
	// 치명타(Critical) 확률
	float CRI_PER;

	/// HP/MP 회복 관련
	// HP 회복 시간
	float REVTime_HP;
	// HP 회복양
	int REV_HP;
	// MP 회복 시간
	float REVTime_MP;
	// MP 회복양
	int REV_MP;
	// HP를 Value 만큼 회복합니다.
	bool RecoveryHP(int iValue);
	// MP를 Value 만큼 회복합니다.
	bool RecoveryMP(int iValue);

	/// 이동 속도/ 공격 속도 관련
	// 이동 속도
	float MOVE_SPEED;
	// 공격 속도
	float ATK_SPEED;

	/// LEVEL 관련
	int LEVEL;
	// 현재 경험치
```

```
	int EXP;
	// 필요 경험치
	int NextEXP() { return vecEXPNext[LEVEL - 1]; }
	vector<int> vecEXPNext;
	// 레벨업 여부를 확인합니다
	bool GetEXPAndCheckLevelUp();
};
```

```
#include "stdafx.h"
#include "StatusInfo.h"


StatusInfo::StatusInfo() :
HP(50),
HPMAX(50),
MP(10),
MPMAX(10),
ATK_PHY(3),
ATK_MGR(3),
DEF_PHY(1),
DEF_MGR(1),
MOVE_SPEED(0.02f),
ATK_SPEED(0),
CRI_PER(20),
REVTime_HP(3),
REV_HP(1),
REVTime_MP(3),
REV_MP(1),
LEVEL(1),
EXP(0)
{
	vecEXPNext.resize(99);

	for (int i = 0; i < 99; ++i)
		vecEXPNext[i] = i + 1;
}


StatusInfo::~StatusInfo()
{
}

bool StatusInfo::RecoveryHP(int iValue)
{
	// 캐릭터가 죽어있지 않을 때와 HP가 꽉차있지 않으면
	if (HP > 0 && HP < HPMAX)
	{
		HP += iValue; // 회복양
		return true;
	}

	return false;
}

bool StatusInfo::RecoveryMP(int iValue)
```

```
{
	if (MP < MPMAX)
	{
		MP += iValue; // 회복양
		return true;
	}

	return false;
}

bool StatusInfo::GetEXPAndCheckLevelUp()
{
	// 레벨 제한 99
	if (LEVEL <= 99)
	{
		// 경험치 증가
		++EXP;

		for (int i = LEVEL - 1; i < LEVEL; ++i)
		{
			// 레벨업
			if (EXP >= vecEXPNext[LEVEL - 1])
			{
				++LEVEL;
				return true;
			}
		}
	}

	return false;
}
```

몬스터 인공지능(AI)

몬스터의 전투 시스템도 캐릭터와 거의 비슷합니다. 기본 캐릭터와 상태기계가 다릅니다.

```
// Smallderon의 인공지능(AI)
class ComSmallderonAI : public ComCharacter
{
public:
	ComSmallderonAI(CString szName);
	~ComSmallderonAI();

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

	//상태 기계
	void Stand() override;
	void Walk(float fDeltaZ) override;
	void Attack1() override;
	void Death() override;

	// 공격할 대상을 찾습니다.
	void FindAttackTarget();
	// 캐릭터가 죽었는지 확인하고 처리합니다.
	void CheckPlayerDeath();

private:
	CTimer* m_pTimerAttack;
};
```

```
void ComSmallderonAI::Awake()
{
	Init();

	m_vecState.resize(eAniMon_COUNT);
	m_vecState[eAniMon_Death] = new ChrStateDeath(this);
	m_vecState[eAniMon_Attack_1] = new ChrStateAttack1(this);
	m_vecState[eAniMon_Walk] = new ChrStateWalk(this);
	m_vecState[eAniMon_Stand] = new ChrStateStand(this);

	m_pCurrentState = m_vecState[eAniMon_Stand];
}
```

몬스터는 항상 플레이어를 찾습니다. FindAttackTarget 함수에서는 공격 대상 플레이어 캐릭터를 찾는데 따라다니는 범위 안에 들어오면 플레이어를 따라갑니다. 타겟을 따라가는 구성요소(Component)를 사용합니다. 공격 범위 안에 들어오면 1초에 한 번씩 공격 함수를 실행합니다.

몬스터가 죽었을 때 처리와 캐릭터가 죽었을 때 처리를 여기서 해줍니다.

```
void ComSmallderonAI::Update()
{
	// 몬스터가 죽지 않았고 공격 대상이 없으면 공격 대상을 찾는다.
	if (IsDeath() == false && m_pFollow->pTarget == NULL)
		FindAttackTarget();
```

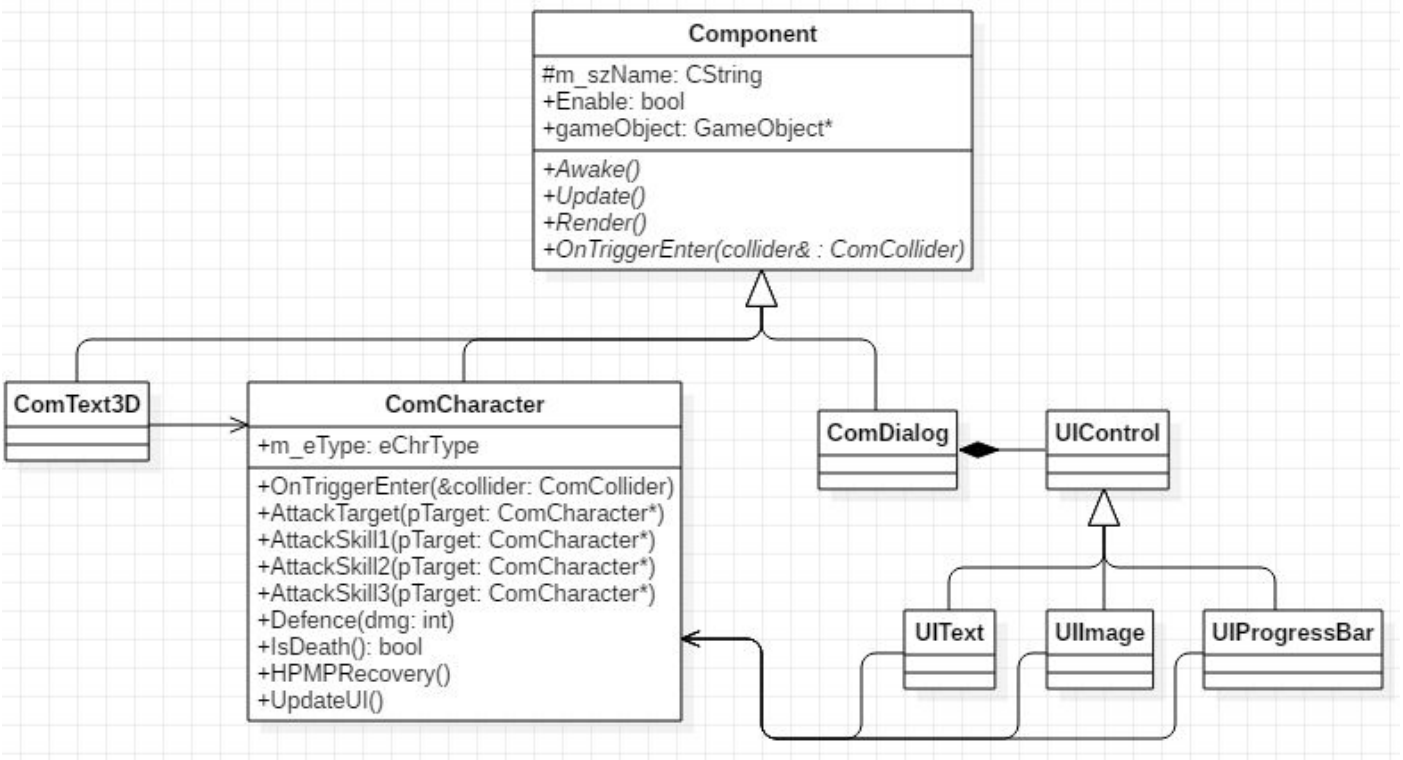
```
	if (m_pFollow->pTarget && m_pFollow->IsFollowing)
	{
		m_pFollow->fMoveSpeed = Status->MOVE_SPEED;
		GetHeight();
		Walk(1);
	}
	else if (m_pFollow->pTarget && m_pFollow->AbleAttack)
	{
		// 1초에 한번씩 Walk하는걸로 그렇지 않으면 Walk <-> Attack 왔다갔다 함
		if (m_pTimerAttack->GetTime() >= 1.0f)
		{
			m_pTimerAttack->Reset();
			// 공격 가능 거리
			Attack1();
		}
	}
	else
		Stand();

	CheckPlayerDeath();

	if (IsGroud == false)
		GetHeight();
}

void ComSmallderonAI::Render()
{
}

void ComSmallderonAI::Stand()
{
	// 현재 상태에서 Stand로
	m_pCurrentState->Stand(eAniMon_Stand);
}

void ComSmallderonAI::Walk(float fDeltaZ)
{
	// 현재 상태에서 Walk로
	m_pCurrentState->Walk(eAniMon_Walk);
}

void ComSmallderonAI::Attack1()
{
	// 현재 상태에서 Attack1로
	m_pCurrentState->Attack1(eAniMon_Attack_1);
}

void ComSmallderonAI::Death()
{
	m_pComUIDamage->Enable = false;
	m_pComUICritical->Enable = false;

	// 현재 상태에서 Death로
	m_pCurrentState->Death(eAniMon_Death);

	// 죽으면 따라가는 캐릭터를 NULL로
```

```
	m_pFollow->pTarget = NULL;

	// 충돌박스를 꺼서 공격 대상에서 제외한다.
	ComRenderCubePN* pCollider =
(ComRenderCubePN*)gameObject->GetComponent("ComRenderCubePN");
	pCollider->Enable = false;
}

void ComSmallderonAI::FindAttackTarget()
{
	SceneRPG* sceneRPG = (SceneRPG*)SceneManager::GetInstance()->GetCurrentScene();

	// 게임 종료라면
	if (sceneRPG->IsGameEnd)
		return;

	list<GameObject*> listGO = GameObject::FindAll(eTag_Chracter);
	size_t chrCnt = listGO.size();

	vector<GameObject*> vecChr;
	vecChr.resize(chrCnt);
	int i = 0;
	for (auto & chr : listGO)
		vecChr[i++] = chr;

	int rA = 0, rB = 0;
	GameObject* pTemp = NULL;
	for (int i = 0; i < 20; ++i) // 셔플로 10번 섞음
	{
		rA = rand() & 2;
		rB = rand() & 2;
		pTemp = vecChr[rA];
		vecChr[rA] = vecChr[rB];
		vecChr[rB] = pTemp;
	}

	int deathCnt = 0;
	for (auto & chr : vecChr)
	{
		ComCharacter* comChr = (ComCharacter*)chr->GetComponent("ComCharacter");

		if (comChr->IsDeath() == false)
		{
			pAttackTarget = comChr;
			m_pFollow->pTarget = comChr->gameObject;
			break;
		}
		else
			++deathCnt;
	}

	// 캐릭터 모두 사망시 게임 종료
	if (deathCnt >= chrCnt)
		sceneRPG->IsGameEnd = true;
}
```

```
void ComSmallderonAI::CheckPlayerDeath()
{
	// 공격 상대가 죽었으면
	if (pAttackTarget && pAttackTarget->IsDeath() == true)
	{
		// 캐릭터 죽음 처리
		if (pAttackTarget->gameObject->Tag == eTag_Chracter)
			pAttackTarget->gameObject->SetActive(false);

		// m_pFollow 변수가 나누어 졌으므로 여기에서 처리
		CancleAttackTarget();

		// 카메라 다시 셋팅
		list<GameObject*> listChr = GameObject::FindAll(eTag_Chracter);
		for (auto & chr : listChr)
		{
			ComCharacter* pComChr = (ComCharacter*)chr->GetComponent("ComCharacter");

			if (pComChr->IsDeath() == false)

Camera::GetInstance()->SetTarget(&pComChr->gameObject->transform->GetPosition());
		}

		Stand();
	}
}
```

UI

전투UI



설계

이호형 팀원이 설계하였고 저는 사용하였습니다. ComDialog가 UI객체들을 map자료구조에 포함해주고 있는 구성요소입니다. ComText3D는 김동오 팀원이 만든 구성요소입니다. 캐릭터의 머리 위에 텍스트를 표시하기 위해 DirectX Sample중에 3DText 코드를 참고하였습니다.

ComCharacter에서 공통되는 UIText, UIImage, UIProgressBar, ComText3D를 선언해주고 ComHuman에서 각 객체를 추가해주어 각 캐릭터 타입별로 UI를 설정해줄 수 있습니다.

```
class ComCharacter : public Component
{
// UI
protected:
	UIText* m_pUILevel;
	UIText* m_pUIEXP;
	UIProgressBar* m_pHPBar;
	UIProgressBar* m_pMPBar;
	UIImage* m_pFace;
	ComText3D* m_pComUIDamage;
	// 데미지 표시 시간
	CTimer* m_pTimerDamage;
	// HP/MP 회복 시간
	CTimer* m_pTimerHPRec;
	CTimer* m_pTimerMPRec;
```

설명

좌상단에는 캐릭터의 기본정보가 표시됩니다. 레벨, 경험치, 캐릭터 얼굴, HP, MP바가 표시됩니다.
캐릭터 위에 이름이 표시됩니다. (플레이어, 몬스터)
하단에는 캐릭터의 스킬 버튼이 표시되며 스킬 버튼 위에는 쿨타임 텍스트가 표시됩니다.

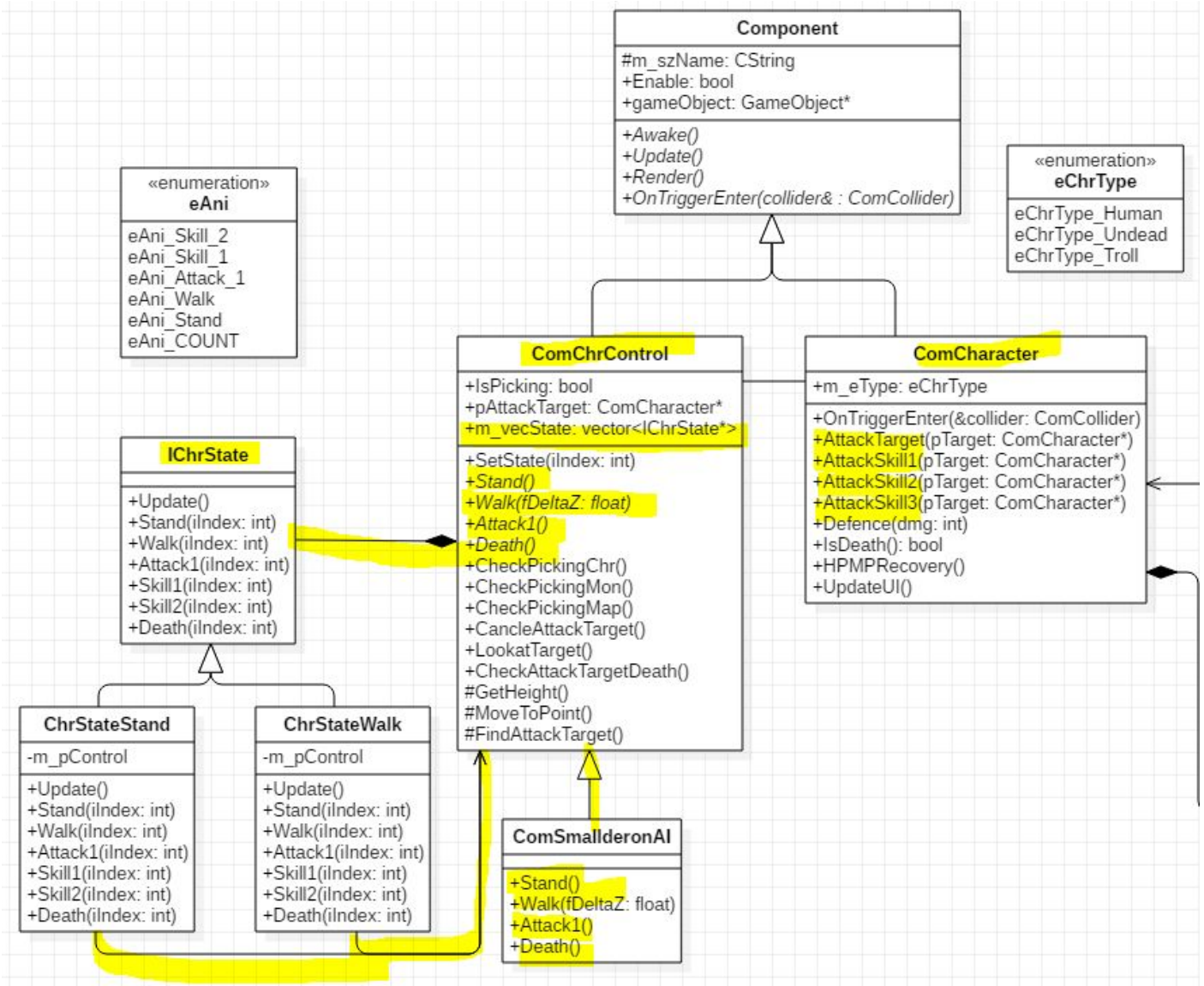
캐릭터의 능력치(Status)가 변경될 때 ComCharacter의 UpdateUI 함수에서 갱신됩니다.

```
void ComCharacter::UpdateUI()
{
	if (m_pHPBar)
	{
		if (Status.HP < 0)
			Status.HP = 0;

		m_pHPBar->SetCurValue(Status.HP);
	}

	if (m_pMPBar)
	{
		if (Status.MP < 0)
			Status.MP = 0;

		m_pMPBar->SetCurValue(Status.MP);
	}

	if (m_pUILevel && m_pUIEXP)
	{
		CString szLevel;
		szLevel.Format(L"LV:%d\r\n%d/%d", Status.LEVEL, Status.EXP, Status.NextEXP());
		m_pUILevel->SetText(szLevel);

		CString szEXP;
		szEXP.Format(L"EXP:%d/%d", Status.EXP, Status.NextEXP());
		m_pUIEXP->SetText(szEXP);
```

```
        }
}
```

리팩토링과 알고리즘 수정

개발을 하다보면 리팩토링과 알고리즘을 수정 해야하는 경우가 있습니다. 설계를 리팩토링 하여 혼잡도를 줄이고 알고리즘을 수정하여 버그 발생 가능성을 줄입니다. 즉 안정적인 프로그램이 되고 완성도가 향상됩니다.



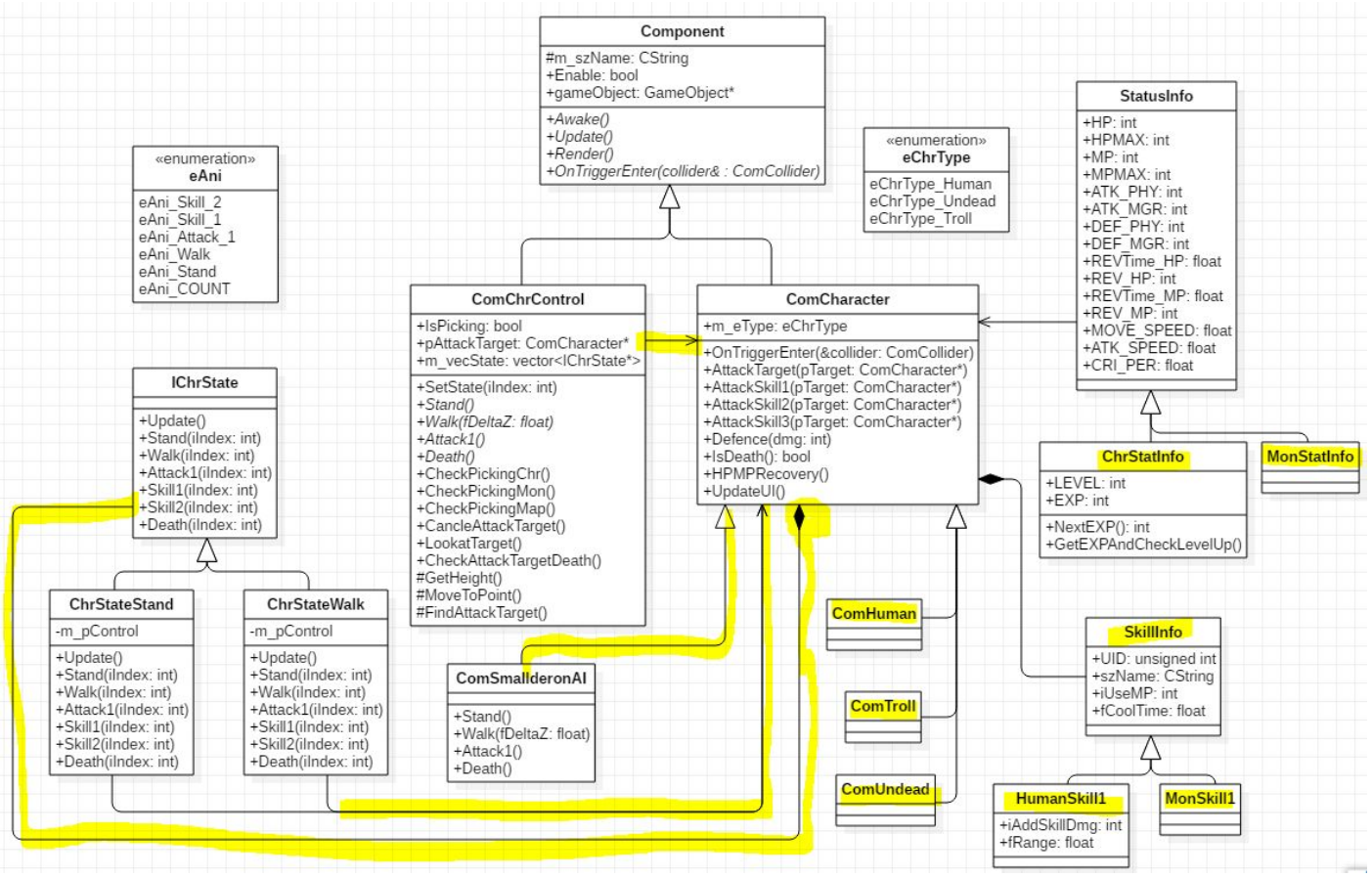
ComChrControl은 컨트롤 관련 구성요소입니다. 그래서 IChrState를 여기에 두었는데 캐릭터의 상태는 ComCharacter에 두어야 겠습니다. 그렇지 않으면 ComChrControl과 ComCharacter가 모호해지게 됩니다. ComChrControl은 오로지 컨트롤 관련된 기능만 넣을 생각입니다.

ComSmallderonAI 즉 몬스터AI는 컨트롤 할 수 있는 구성요소가 아닌데 ComChrControl에서 상속받은 것도 맞지 않습니다. 그러므로 이데로 계속 진행하다가는 설계가 꼬여서 복잡해질 우려가 있습니다.

캐릭터의 능력치(StatusInfo)도 캐릭터와 몬스터별로 나누어야 합니다. 캐릭터에만 들어가는 정보를 ChrStatInfo로 몬스터에만 들어가는 정보들을 MonStatInfo로 둘 예정입니다.

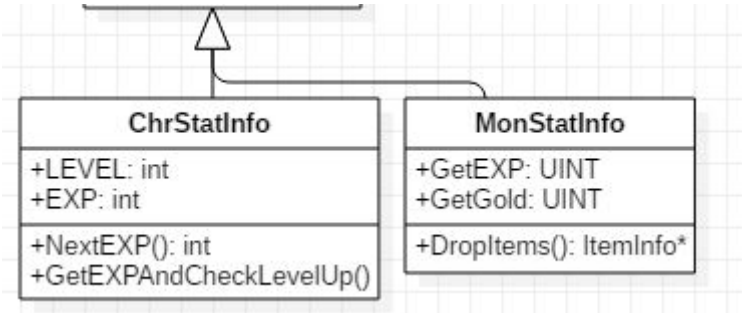
ComCharacter는 개발이 오래 될 수록 복잡해 질 것입니다. 그러므로 설계와 코드 관리에 더욱 신경써 주어야합니다.

설계를 다음과 같이 수정할 것입니다.



ComChrControl과 ComCharacter가 모호해서 기능들이 중복되는 것이 해결될 것이며 상태기계가 ComChrControl에서 ComCharacter로 옮겨짐에 따라 설계가 복잡해지지 않게 될 것입니다.

그래서 중요한 사실은 ComSmallderonAI, ComHuman, ComTroll, ComUndead가 ComCharacter에서 상속받게 됩니다. ComSmallderonAI는 MonStatInfo를 참조할 것이고 ComHuman은 ChrStatInfo를 참조할 것입니다.



몬스터는 레벨업하지 않습니다. 몬스터가 캐릭터를 모두 죽이면 게임이 끝나게 됩니다. 그러므로 몬스터 정보에는 LEVEL과 EXP등이 필요하지 않습니다. 반면에 캐릭터가 몬스터를 사냥했을 경우 경험치와 골드를 얻을 수 있고 이는 몬스터마다 그 값이 다릅니다. 그리고 몬스터를 사냥했을 때 얻을 수 있는 아이템들 또한 다릅니다. 이런 정보들을 분리해서 관리해주기 위해 설계를 수정해야 합니다.

설계를 수정하기 전에 백업을 잘 해두고 하나씩 수정해 나아갑니다.

STEP 1. GetHeight() 함수를 ComChrControl -> ComCharacter로 수정합니다. ComChrControl에서 GetHeight함수가 없어지기 때문에 상속받은 ComSmallderonAI에서 상위 함수를 사용할 수 없으므로 임시로 GetHeight함수를 만들고 ComCharacter에서 상속 받을 시 삭제할 것이라고 주석을 달아놓습니다.
BTC실행) 게임을 실행해서 버그가 발생하는지 확인합니다. 버그가 없으면 코드를 저장하기 위해 Commit합니다.

STEP 2. ComChrControl의 FindAttackTarget 함수는 자동으로 공격할 캐릭터를 찾는 AI 함수이기 때문에 ComChrControl에서 ComSmallderonAI로 함수를 이동해 줍니다. BTC실행) Bug Test And Commit

STEP 3. ComChrControl의 공격하기 전 타겟을 바라보는 함수 LookAtTarget 함수를 ComCharacter로 이동합니다. 코드 한 라인을 없애서 복잡도를 줄입니다. 또한 ComChrControl이 아닌 ComCharacter를 상속받은 클래스가 이 함수를 사용할 수 있도록 합니다. BTC)

```
void ComCharacter::AttackTarget(ComCharacter * pTarget)
{
	m_pAttackTarget = pTarget;

ComChrControl* pControl = (ComChrControl*)(gameObject->GetComponent("ComChrControl"));
	pControl->LookatTarget();
//...
}

void ComHuman::AttackSkill1(ComCharacter * pTarget)
{
	m_pAttackTarget = pTarget;
	LookatTarget();
//...
}
```

STEP 4. ComChrControl에 있는 특정 오브젝트를 따라다니는 m_pFollow 변수를 private으로 만들고 ComSmallderonAI의 변수를 하나 추가해서 나누어 줍니다. 이유는 Death() 상태기계 변수를 ComChracter로 옮기기 전 필요한 작업입니다.
또한, ComSmallderonAI의 Awake()에서 상위 클래스 ComChrControl의 Init()을 호출하는데 이를 주석처리 합니다. BTC)

```
void ComSmallderonAI::Death()
{
	// 현재 상태에서 Death로
	m_pCurrentState->Death(eAniMon_Death);

	// 죽으면 따라가는 캐릭터를 NULL로
	m_pFollow->pTarget = NULL;
}
```

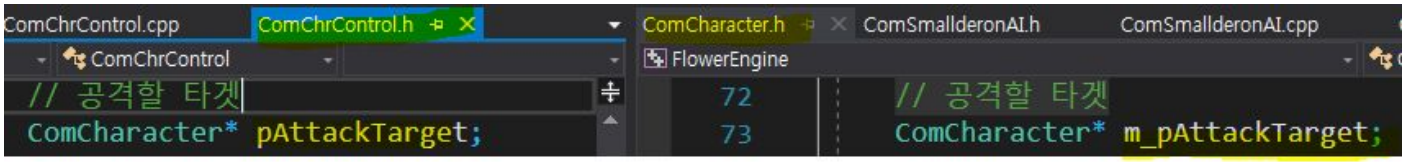
STEP 5. ComChrControl의 CheckAttackTargetDeath() 함수의 기능을 나누어 줍니다. 함수를 보면 캐릭터 죽음 처리와 몬스터 죽음 처리가 if else문으로 되어 있는것을 볼 수 있습니다. 캐릭터 죽음 처리는 ComSmallderonAI에서 처리하고 몬스터 죽음 처리는 ComChrControl에 둡니다. BTC)

```
void ComChrControl::CheckAttackTargetDeath()
{
	// 공격 상대가 죽었으면
	if (pAttackTarget && pAttackTarget->IsDeath() == true)
	{
		// 캐릭터 죽음 처리 -> 이부분 수정 필요
		if (pAttackTarget->gameObject->Tag == eTag_Chracter)
			pAttackTarget->gameObject->SetActive(false);
		else // 몬스터 죽음 처리 -> 이부분 수정 필요
		{
			ComChrControl* pControl =
(ComChrControl*)(pAttackTarget->gameObject->GetComponent("ComChrControl"));
			pControl->Death();

			// 캐릭터 레벨업 처리
			if (m_pCharacter->Status.GetEXPAndCheckLevelUp())
				m_pCharacter->LevelUp();
		}
		CancleAttackTarget();
		Stand();
	}
}
```

함수명도 명확하게 CheckMonDeath()와 CheckPlayerDeath()로 수정해 줍니다.

```
void ComChrControl::CheckMonDeath()
{
	// 공격 상대가 죽었으면
	if (pAttackTarget && pAttackTarget->IsDeath() == true)
	{
		// 몬스터 죽음 처리
		ComChrControl* pControl =
(ComChrControl*)(pAttackTarget->gameObject->GetComponent("ComChrControl"));
		pControl->Death();

		// 캐릭터 레벨업 처리
		if (m_pCharacter->Status.GetEXPAndCheckLevelUp())
			m_pCharacter->LevelUp();

		CancleAttackTarget();
		Stand();
	}
}
void ComSmallderonAI::CheckPlayerDeath()
{
	// 공격 상대가 죽었으면
	if (pAttackTarget && pAttackTarget->IsDeath() == true)
	{
		// 캐릭터 죽음 처리
		if (pAttackTarget->gameObject->Tag == eTag_Chracter)
			pAttackTarget->gameObject->SetActive(false);

		pAttackTarget = NULL;
		// m_pFollow 변수가 나누어 졌으므로 여기에서 처리
		m_pFollow->IsFollowing = false;
		m_pFollow->AbleAttack = false;
		m_pFollow->pTarget = NULL;

		Stand();
	}
}
```

STEP 6. ComCharacter* pAttackTarget 공격할 타겟이 중복되어 있는데 이를 하나로 합쳐서 코드를 최적화합니다.



어택 이벤트 함수에서 ComChrControl을 불러와서 매개변수로 지정해 주는 문제점이 해결됩니다. 조금씩 코드가 정리되며 간격해집니다. (Code Completed)

```
HRESULT AttackHandler::HandleCallback(UINT Track, LPVOID pCallbackData)
{
	// 특정 프레임에서 공격
	ComCharacter* pChr = (ComCharacter*)pCallbackData;
	~~ComChrControl* pControl =~~
~~(ComChrControl*)pChr->gameObject->GetComponent("ComChrControl");~~

	// 죽어서 없으면
	if (~~pControl~~->pAttackTarget == NULL)
```

```
                return S_OK;

        pChr->AttackTarget(~~pControl->~~pAttackTarget);

        return S_OK;
}
```

원하는 코드는 위와 같지만 현재 ComSmallderonAI가 ComChrControl을 상속받고 있으므로 일단은 Tag로 분리하여 기존 기능이 동작되도록 합니다. 추후 ComSmallderonAI가 ComCharacter에서 상속받을 때 수정해주도록 합니다.
현재 캐릭터 스킬은 휴먼 하나만 구현되어 있는데 해당 부분의 pAttackTarget 변수 관련 알고리즘도 수정합니다. BTC)

```
HRESULT AttackHandler::HandleCallback(UINT Track, LPVOID pCallbackData)
{
        // 특정 프레임에서 공격
        ComCharacter* pChr = (ComCharacter*)pCallbackData;

        switch (pChr->gameObject->Tag)
        {
        case eTag_Chracter:
                // 죽어서 없으면
                if (pChr->pAttackTarget == NULL)
                        return S_OK;
                pChr->AttackTarget(pChr->pAttackTarget);
                break;

                // ComCharacter로 상속받을 때 이부분 수정할 것
        case eTag_Monster:
        {
                ComSmallderonAI* pMon =
(ComSmallderonAI*)pChr->gameObject->GetComponent("ComChrControl");
                if (pMon->pAttackTarget == NULL)
                        return S_OK;
                pChr->AttackTarget(pMon->pAttackTarget);
                break;

        }
        }

        return S_OK;
}
```

STEP 7. ComChrControl에 있는 상태기계 관련 코드를 ComCharacter로 옮겨줍니다.

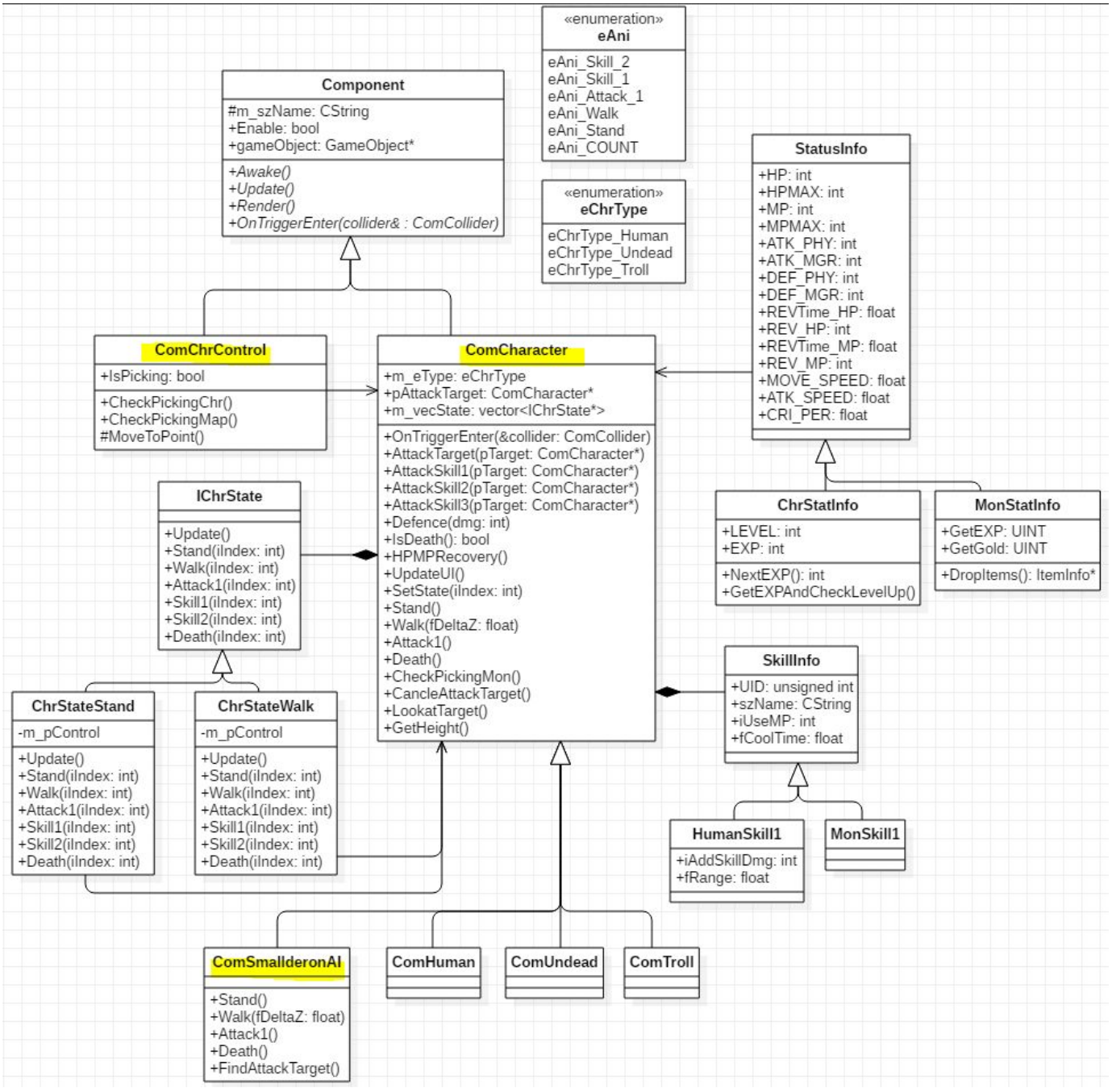
```
class ComChrControl : public Component
{
// ...

public:
        /// 상태기계 관련
        // 상태들
        vector<IChrState*> m_vecState;
        // 현재 상태
        IChrState * m_pCurrentState;
        void SetState(int iIndex);
        virtual void Stand();
        virtual void Walk(float fDeltaZ);
        virtual void Attack1();
```

```
        virtual void Death();
}
```

ComSmallderon의 상위 클래스를 ComChrControl에서 ComCharacter로 수정합니다. 이 부분 수정과정이 조금 복잡해서 문서에 다 적지는 않겠습니다. 결과적으로 설계와 알고리즘이 깔끔해지고 있습니다.

```
class ComSmallderonAI : public ComCharacter
```

설계도는 다음과 같이 수정되었습니다.



STEP 8. ComCharacter에 있는 UI 관련 요소들을 ComCharacterUI라는 구성요소로 옮겨 줍니다. 이유는 상속받은 몬스터 클래스에는 이 UI정보가 필요하지 않기 때문입니다.

```
class ComCharacter : public Component
{
// ...
protected:
        /// 캐릭터 정보
        // 캐릭터 레벨 텍스트 표시
        UIText* m_pUILevel;
```

```
	// 캐릭터 경험치 텍스트 표시
	UIText* m_pUIEXP;
	// 캐릭터 얼굴 이미지
	UIImage* m_pFace;
	/// 데미지 표시
	// 머리위에 데미지 표시
	ComText3D* m_pComUIDamage;
	// 데미지 표시 시간
	CTimer* m_pTimerDamage;
	/// HP/MP 관련
	// HP/MP바
	UIProgressBar* m_pHPBar;
	UIProgressBar* m_pMPBar;
	// HP/MP 회복시간 체크하기 위한
	CTimer* m_pTimerHPRec;
	CTimer* m_pTimerMPRec;
};
```

ComCharacter에 있는 능력치 정보도 다형성을 사용하기 위해 StatusInfo* 포인터형으로 바꾸어 줍니다. Status 변수를 여러곳에서 사용하고 있으므로 이 작업도 만만치 않습니다.

```
class ComCharacter : public Component
{
public:
	// 능력치
	StatusInfo Status;
};
```

나중에 이 부분을 MonStatInfo로 바꾸어 줄 계획입니다.

```
void SceneRPG::CreateMonster()
{
	// 몬스터 생성 (smallderon_orange)
	StatusInfo* monStatus = new StatusInfo(); monStatus->HP = 50; monStatus->HPMAX = 50;
monStatus->ATK_PHY = 5;
	GameObject* pGOMonX = factory.CreateMonster("폭염몬", "Resources/monster/smallderon/",
"smallderon_orange.X", Vector3(-243, 7.7184200, -240),
		new ComSmallderonAI("ComCharacter"), monStatus);
```

지형(Terrain)

지형은 삼각형들의 모음으로 되어있습니다.
와우 익스포터 툴에서 OBJ 파일로 지형 정보를 추출할 수 있는데 그 파일을 3D맥스에서 불러와서 조금 수정하고 다시 OBJ 파일로 추출하여 엔진에서 읽어들이는 방법을 사용하고 있습니다.

3D 게임 프로그래밍책에 있는 절두체 컬링, 쿼드트리, LOD 기법이 적용되어 있습니다.