27. 콜리전 클래스(2)

Object.h

```
public:
	template<typename T>
	T* AddCollider( const string& strTag )
	{
		T* pCollider = new T;

		pCollider->SetObj( this );

		if( !pCollider->Init() )
		{
			SAFE_RELEASE( pCollider );
			return nullptr;
		}

		pCollider->AddRef();
		m_ColliderList.push_back( pCollider );

		return pCollider;
	}

	bool CheckCollider()
	{
		return !m_ColliderList.empty();
	}
```

오브젝트에 콜라이더를 추가.

오브젝트가 콜라이더를 들고 있는지 체크. 콜라이더가 하나도 없으면 콜리전 테스트를 진행하지 않기 위함.

Object.cpp

```
int CObject::Update( float fDeltaTime )
{
	list<CCollider*>::iterator iter;
	list<CCollider*>::iterator iterEnd = m_ColliderList.end();

	for( iter = m_ColliderList.begin(); iter != iterEnd; ++iter )
	{
		if( !(*iter)->GetEnable() )
		{
			++iter;
			continue;
		}

		(*iter)->Update( fDeltaTime );

		if( !(*iter)->GetLife() )
		{
			SAFE_RELEASE( (*iter) );
			iter = m_ColliderList.erase( iter );
			iterEnd = m_ColliderList.end();
		}
		else
			++iter;
	}

	return 0;
}
```

CRef 클래스를 통해 상속받은 활성/비활성 생존여부 체크 해 콜라이더 업데이트 호출.

```
int CObject::LateUpdate( float fDeltaTime )
{
	list<CCollider*>::iterator iter;
	list<CCollider*>::iterator iterEnd = m_ColliderList.end();

	for( iter = m_ColliderList.begin(); iter != iterEnd; ++iter )
	{
		if( !(*iter)->GetEnable() )
		{
			++iter;
			continue;
		}

		(*iter)->LateUpdate( fDeltaTime );

		if( !(*iter)->GetLife() )
		{
			SAFE_RELEASE( (*iter) );
			iter = m_ColliderList.erase( iter );
			iterEnd = m_ColliderList.end();
		}
		else
			++iter;
	}

	return 0;
}
```

```
void CObject::Render( HDC hDC, float fDeltaTime )
{
	if( m_pTexture )
	{
		POSITION tPos = m_tPos - m_tSize * m_tPivot;
		POSITION camPos = GET_SINGLE( CCamera )->GetPos();

		tPos = tPos - camPos;

		if( m_pTexture->GetColorKeyEnable() )
		{
			TransparentBlt( hDC, tPos.x, tPos.y, m_tSize.x, m_tSize.y,
				m_pTexture->GetDC(), 0, 0, m_tSize.x, m_tSize.y, m_pTexture->GetColorKey(
		}
		else
		{
			BitBlt( hDC, tPos.x, tPos.y, m_tSize.x, m_tSize.y, m_pTexture->GetDC(), 0, 0, SRC
		}
	}

	list<CCollider*>::iterator iter;
	list<CCollider*>::iterator iterEnd = m_ColliderList.end();

	for( iter = m_ColliderList.begin(); iter != iterEnd; ++iter )
	{
		if( !(*iter)->GetEnable() )
		{
			++iter;
			continue;
		}

		(*iter)->Render( hDC, fDeltaTime );

		if( !(*iter)->GetLife() )
		{
			SAFE_RELEASE( (*iter) );
			iter = m_ColliderList.erase( iter );
			iterEnd = m_ColliderList.end();
		}
		else
			++iter;
	}
}
```

충돌체도 화면에 그려줌. 나중에 디버그 모드에서만 그려질 수 있도록 수정.

Types.h

```
typedef struct _tagRectangle
{
        float left;
        float top;
        float right;
        float bottom;

        _tagRectangle() :
                left( 0 ),
                top( 0 ),
                right( 0 ),
                bottom( 0 )
        {
        }

}RECTANGLE, *PRECTANGLE;
```

Rect Collision에 사용할 Rectangle 구조체를 만들어줌.

Collider

```
#pragma once

#include "../Ref.h"
class CCollider : public CRef
{
protected:
        friend class CObject;

protected:
        CCollider();
        CCollider( const CCollider& coll );
        virtual ~CCollider() = 0;

protected:
        COLLIDER_TYPE   m_eCollType;
        class CObject*  m_pObj;

public:
        COLLIDER_TYPE GetColliderType()const
        {
                return m_eCollType;
        }

        class CObject* GetObj() const
        {
                return m_pObj;
        }

public:
        void SetObj( class CObject* pObj )
        {
                m_pObj = pObj;
        }

public:
        virtual bool Init() = 0;
        virtual void Input( float fDeltaTime );
        virtual int Update( float fDeltaTime );
        virtual int LateUpdate( float fDeltaTime );
        virtual void Collision( float fDeltaTime );
        virtual void Render( HDC hDC, float fDeltaTime );
        virtual CCollider* Clone() = 0;
};
```

자신을 들고있는 오브젝트를 알게함.

```
#include "Collider.h"
#include "../Object/Object.h"

CCollider::CCollider()
{
}

CCollider::CCollider( const CCollider& coll )
{
        *this = coll;
}

CCollider::~CCollider()
{
}

void CCollider::Input( float fDeltaTime )
{
}

int CCollider::Update( float fDeltaTime )
{
        return 0;
}

int CCollider::LateUpdate( float fDeltaTime )
{
        return 0;
}

void CCollider::Collision( float fDeltaTime )
{
}

void CCollider::Render( HDC hDC, float fDeltaTime )
{
}
```

ColliderRect

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

class CColliderRect : public CCollider
{
protected:
        friend class CObject;

protected:
        CColliderRect();
        CColliderRect( const CColliderRect& coll );
        virtual ~CColliderRect();

private:
        RECTANGLE       m_tInfo;
        RECTANGLE   m_tWorldInfo;

public:
        void SetRect( float l, float t, float r, float b );
        RECTANGLE GetInfo() const
        {
                return m_tInfo;
        }
        RECTANGLE GetWorldInfo() const
        {
                return m_tWorldInfo;
        }

public:
        virtual bool Init();
        virtual void Input( float fDeltaTime );
        virtual int Update( float fDeltaTime );
        virtual int LateUpdate( float fDeltaTime );
        virtual void Collision( float fDeltaTime );
        virtual void Render( HDC hDC, float fDeltaTime );
        virtual CColliderRect* Clone();
};
```

```
#include "ColliderRect.h"
#include "../Object/Object.h"

CColliderRect::CColliderRect()
{
        m_eCollType = COLLIDER_TYPE::CT_RECT;
}

CColliderRect::CColliderRect( const CColliderRect& coll )
        :
        CCollider( coll )
{
        m_tInfo = coll.m_tInfo;
}

CColliderRect::~CColliderRect()
{
}

void CColliderRect::SetRect( float l, float t, float r, float b )
{
        m_tInfo.left = l;
        m_tInfo.top = t;
        m_tInfo.right = r;
        m_tInfo.bottom = b;
}

bool CColliderRect::Init()
{
        return true;
}

void CColliderRect::Input( float fDeltaTime )
{
        CCollider::Input( fDeltaTime );
}

int CColliderRect::Update( float fDeltaTime )
{
        CCollider::Update( fDeltaTime );
        return 0;
}
```

```
int CColliderRect::LateUpdate( float fDeltaTime )
{
        CCollider::LateUpdate( fDeltaTime );

        POSITION tPos = m_pObj->GetPos();
        m_tWorldInfo.left = tPos.x + m_tInfo.left;
        m_tWorldInfo.top = tPos.y + m_tInfo.top;
        m_tWorldInfo.right = tPos.x + m_tInfo.right;
        m_tWorldInfo.bottom = tPos.y + m_tInfo.bottom;

        return 0;
}

void CColliderRect::Collision( float fDeltaTime )
{
        CCollider::Collision( fDeltaTime );
}

void CColliderRect::Render( HDC hDC, float fDeltaTime )
{
        CCollider::Render( hDC, fDeltaTime );
}

CColliderRect* CColliderRect::Clone()
{
        return new CColliderRect(*this);
}
```

오브젝트가 움직이면 콜라이더도 맞춰 움직임.

Layer

```cpp
void CLayer::Collision( float fDeltaTime )
{
	list<CObject*>::iterator iter;
	list<CObject*>::iterator iterEnd = m_ObjList.end();

	for( iter = m_ObjList.begin(); iter != iterEnd; )
	{
		if( !(*iter)->GetEnable() )
		{
			++iter;
			continue;
		}

		if( !(*iter)->GetLife() )
		{
			CObject::EraseObj( *iter );
			SAFE_RELEASE( (*iter) );
			iter = m_ObjList.erase( iter );
			iterEnd = m_ObjList.end();
		}

		else
		{
			GET_SINGLE( CCollisionManager )->AddObject( *iter );
			++iter;
		}
	}
}
```

활성화 되어 있고 살아있는 오브젝트 들만
Collision Manager의 Add Object에 들어감.

Collision Manager

```cpp
#pragma once

#include "../Game.h"

class CCollisionManager
{
private:
	list<class CObject*> m_CollisionList;

public:
	void AddObject( class CObject* pObj );
	void Collision( float fDeltaTime );

	DECLARE_SINGLE( CCollisionManager )
};
```

```cpp
#include "CollisionManager.h"
#include "../Object/Object.h"
#include "Collider.h"

DEFINITION_SINGLE( CCollisionManager )

CCollisionManager::CCollisionManager()
{

}

CCollisionManager::~CCollisionManager()
{

}

void CCollisionManager::AddObject( CObject* pObj )
{
	if( pObj->CheckCollider() )
	{
		m_CollisionList.push_back( pObj );
	}
}

void CCollisionManager::Collision( float fDeltaTime )
{
	if( m_CollisionList.size() < 2 )
	{
		m_CollisionList.clear();
		return;
	}

	// process collision between objects
}
```

콜라이더가 있는 오브젝트 들을
리스트에 모아둠.

Core

```cpp
void CCore::Collision( float fDeltaTime )
{
    GET_SINGLE( CSceneManager )->Collision( fDeltaTime );

    GET_SINGLE( CCollisionManager )->Collision( fDeltaTime );
}
```

호출후에는
씬 매니저의 오브젝트 리스트에 콜라이더를 가진 오브젝트 들이 모인 상태
그럼이제 콜리젼 수행