16. 플레이어와 몬스터 이동

<플레이어 & 몬스터 생성>

InGameScence

```cpp
#include "InGameScene.h"
#include "../Object/Player.h"
#include "../Object/Minion.h"
#include "Layer.h"

CInGameScene::CInGameScene()
{
}

CInGameScene::~CInGameScene()
{
}

bool CInGameScene::Init()
{
    // 부모 Scene 클래스의 초기화 함수를 호출해줌.
    if( !CScene::Init() )
    {
        return false;
    }

    CLayer* pLayer = FindLayer( "Default" );

    CPlayer* pPlayer = CObject::CreateObj<CPlayer>( "Player", pLayer );

    SAFE_RELEASE( pPlayer );

    CMinion* pMinion = CObject::CreateObj<CMinion>( "Minion", pLayer );

    SAFE_RELEASE( pMinion );

    return true;
}
```

```cpp
void CLayer::AddObject( CObject* pObject )
{
        pObject->SetScene( m_pScene );
        pObject->SetLayer( this );
        pObject->AddRef();

        m_ObjList.push_back( pObject );
}
```

오브젝트 추가시 오브젝트가 생성 되면서 Ref 카운트가 증가하고, 레이어에 오브젝트가 추가 되면서 Ref 카운트가 올라가서 최종 카운트가 2가 된다.

그런데, 실제로 오브젝트를 들고 있는 건 레이어 뿐이고, pPlayer, pMinion은 함수내 지역변수 이기 때문에 함수를 빠져나가면 사라진다.

따라서 Ref 카운트를 1로 만들어주기 위해 생성 후 바로 RELEASE를 통해 카운트를 감소시켜 주는 것이다.

Minion

```cpp
bool CMinion::Init()
{
        SetPos( 800, 100.f );
        SetSize( 100.f, 100.f );
        SetSpeed( 300.f );

        m_eDir = MD_FRONT;

        return true;
}

int CMinion::Update( float fDeltaTime )
{
        CMoveObject::Update( fDeltaTime );

        MoveYAsSpeed( fDeltaTime, m_eDir );

        RESOLUTION res = GETRESOLUTION;

        if( m_tPos.y + m_tSize.y >= res.iH )
        {
                m_tPos.y = res.iH - m_tSize.y;
                m_eDir = MD_BACK;
        }

        else if( m_tPos.y <= 0.f )
        {
                m_tPos.y = 0.f;
                m_eDir = MD_FRONT;
        }

        return 0;
}

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

void CMinion::Collision( float fDeltaTime )
{
        CMoveObject::Collision( fDeltaTime );
}

void CMinion::Render( HDC hDC, float fDeltaTime )
{
        CMoveObject::Render( hDC, fDeltaTime );
        Rectangle( hDC, m_tPos.x, m_tPos.y, m_tPos.x + m_tSize.x, m_tPos.y + m_tSize.y );
}
```

스크린 위아래 충돌 시 방향 전환

Player

```cpp
bool CPlayer::Init()
{
    SetPos( 100.f, 100.f );
    SetSize( 100.f, 100.f );
    SetSpeed( 400.f );

    return true;
}

void CPlayer::Input( float fDeltaTime )
{
    CMoveObject::Input( fDeltaTime );

    if( GetAsyncKeyState( 'W' ) & 0x8000 )
    {
        MoveYAsSpeed( fDeltaTime, MD_BACK );
    }
    if( GetAsyncKeyState( 'S' ) & 0x8000 )
    {
        MoveYAsSpeed( fDeltaTime, MD_FRONT );
    }
    if( GetAsyncKeyState( 'A' ) & 0x8000 )
    {
        MoveXBySpeed( fDeltaTime, MD_BACK );
    }
    if( GetAsyncKeyState( 'D' ) & 0x8000 )
    {
        MoveXBySpeed( fDeltaTime, MD_FRONT );
    }
}

int CPlayer::Update( float fDeltaTime )
{
    CMoveObject::Update( fDeltaTime );

    return 0;
}

int CPlayer::LateUpdate( float fDeltaTime )
{
    CMoveObject::LateUpdate( fDeltaTime );

    return 0;
}

void CPlayer::Collision( float fDeltaTime )
{
    CMoveObject::Collision( fDeltaTime );
}

void CPlayer::Render( HDC hDC, float fDeltaTime )
{
    CMoveObject::Render( hDC, fDeltaTime );
    Rectangle( hDC, m_tPos.x, m_tPos.y, m_tPos.x + m_tSize.x, m_tPos.y + m_tSize.y );
}
```

WASD로 이동

```cpp
// Direction
enum MOVE_DIR
{
        MD_BACK = -1,
        MD_NONE,
        MD_FRONT
};
```

✳ 메모리 누수 체크 함수

```
CCore::CCore()
{
    _CrtSetDbgFlag( _CRTDBG_ALLOC_MEM_DF | _CRTDBG_LEAK_CHECK_DF );
    //_CrtSetBreakAlloc(234);
}
```

crtdbg.h 헤더 파일에 포함되어 있으며 메모리 누수 체크가 가능함.

_CRTDBG_ALLOC_MEM_DF : 디폴트 값. 디버그 버전에서 모든 할당이 일어날 때마다 추적 가능하도록 하는 기능을 킴.

_CRTDBG_LEAK_CHECK_DF : 프로그램이 완전히 종료되기 직전에 아직 해제되지 않은 메모리가 있는지 체크.

이를 작성하고 디버깅에서 윈도우 창을 끄게 되면 메모리 누수를 체크 해 해당 블록 번호를 주는데, 이를 _CrtSetBreakAlloc() 함수에 넣으면 발생 부분으로 이동함. 이동 후 호출 스택을 통해 이전 호출 스택들을 조사해 누수 발생 원인을 찾을 수 있음.