25. 입력 관리자

Input.h

```h
#pragma once

#include "../Game.h"

typedef struct _tagKeyInfo
{
	string			strName;
	vector<DWORD>	vecKey;
	bool			bDown;
	bool			bPress; // hold key
	bool			bUp;

	_tagKeyInfo()
		:
		bDown( false ),
		bPress( false ),
		bUp( false )
	{
	}
}KEYINFO, *PKEYINFO;

class CInput
{
private:
	HWND		m_hWnd;
	unordered_map<string, PKEYINFO> m_mapKey;
	PKEYINFO	m_pCreateKey;

public:
	bool Init( HWND hWnd );
	void Update( float fDeltaTime );
	bool KeyDown( const string& strKey )const;
	bool KeyPress( const string& strKey )const;
	bool KeyUp( const string& strKey )const;

public:
	template <typename T>
	bool AddKey( const T& data )
	{
		if( !m_pCreateKey )
		{
			m_pCreateKey = new KEYINFO;
		}

		const char* pTType = typeid(T).name();

		if( strcmp( pTType, "char" ) == 0 || strcmp( pTType, "int" ) == 0 )
		{
			m_pCreateKey->vecKey.push_back( (DWORD)data );
		}
		else
		{
			m_pCreateKey->strName = data;
			m_mapKey.insert( make_pair( m_pCreateKey->strName, m_pCreateKey ) );
		}

		return true;
	}
```

종료 함수

인자의 갯수가 1개 남으면 호출

가변인자 템플릿

```h
	template <typename T, typename... Types>
	bool AddKey( const T& data, const Types&... arg )
	{
		if( !m_pCreateKey )
		{
			m_pCreateKey = new KEYINFO;
		}

		const char* pTType = typeid(T).name();

		if( strcmp( pTType, "char" ) == 0 || strcmp( pTType, "int" ) == 0 )
		{
			m_pCreateKey->vecKey.push_back( (DWORD)data );
		}
		else
		{
			m_pCreateKey->strName = data;
			m_mapKey.insert( make_pair( m_pCreateKey->strName, m_pCreateKey ) );
		}

		AddKey( arg... );

		if( m_pCreateKey )
			m_pCreateKey = nullptr;

		return true;
	}

private:
	PKEYINFO FindKey( const string& strKey )const;

	DECLARE_SINGLE( CInput )
};
```

재귀 탐색하고

↓ m_pCreateKey 비우고 종료

Input.cpp

```cpp
#include "Input.h"

DEFINITION_SINGLE( CInput )

CInput::CInput()
	:
	m_pCreateKey( nullptr )
{
}

CInput::~CInput()
{
	unordered_map<string, PKEYINFO>::iterator iter;
	unordered_map<string, PKEYINFO>::iterator iterEnd = m_mapKey.end();

	for( iter = m_mapKey.begin(); iter != iterEnd; ++iter )
	{
		if( iter->second )
		{
			delete iter->second;
			iter->second = nullptr;
		}
	}
}

bool CInput::Init( HWND hWnd )
{
	m_hWnd = hWnd;

	AddKey( 'W', "MoveUp" );
	AddKey( 'S', "MoveDown" );
	AddKey( "MoveLeft", 'A' );
	AddKey( "MoveRight", 'D' );
	AddKey( "Fire", VK_SPACE );
	AddKey( VK_CONTROL, "Skill1", '1' );

	return true;
}
```

```cpp
void CInput::Update( float fDeltaTime )
{
	unordered_map<string, PKEYINFO>::iterator iter;
	unordered_map<string, PKEYINFO>::iterator iterEnd = m_mapKey.end();

	for( iter = m_mapKey.begin(); iter != iterEnd; ++iter )
	{
		int iPushCount = 0;
		for( size_t i = 0; i < iter->second->vecKey.size(); ++i )
		{
			if( GetAsyncKeyState( iter->second->vecKey[i] ) & 0x8000 )
				++iPushCount;

			if( iPushCount == iter->second->vecKey.size() )
			{
				if( iter->second->bDown == false && iter->second->bPress == false
					iter->second->bDown = true;

				else if( iter->second->bDown == true && iter->second->bPress == f
				{
					iter->second->bDown = false;
					iter->second->bPress = true;
				}
			}
			else
			{
				if( iter->second->bDown || iter->second->bPress )
				{
					iter->second->bUp = true;
					iter->second->bPress = false;
					iter->second->bDown = false;
				}
				else if(iter->second->bUp )
					iter->second->bUp = false;
			}
		}
	}
}
```

키 상태 체크

```cpp
bool CInput::KeyDown( const string& strKey ) const
{
	PKEYINFO pInfo = FindKey( strKey );

	if( !pInfo ) return false;

	return pInfo->bDown;
}

bool CInput::KeyPress( const string& strKey ) const
{
	PKEYINFO pInfo = FindKey( strKey );

	if( !pInfo ) return false;

	return pInfo->bPress;
}

bool CInput::KeyUp( const string& strKey ) const
{
	PKEYINFO pInfo = FindKey( strKey );

	if( !pInfo ) return false;

	return pInfo->bUp;
}

PKEYINFO CInput::FindKey( const string& strKey ) const
{
	unordered_map<string, PKEYINFO>::const_iterator iter = m_mapKey.find( strKey );

	if( iter == m_mapKey.end() )
	{
		return nullptr;
	}

	return iter->second;
}
```

키의 경우 문자일수도 정수일수도(가상키의 경우) 있고, 조합키의 경우 키의 갯수가 여러개 이기 때문에 가변인자 템플릿을 사용한다.
가변인자 템플릿 함수의 경우 재귀구조로 진행되므로 마지막 종료시의 함수를 만들어줌.

함수에서 키의 이름과 해당하는 키들을 벡터에 넣어줌.

Macro.h

```h
#pragma once

#define SAFE_DELETE(p) if(p) { delete p; p = nullptr; }
#define SAFE_DELETE_ARRAY(p) if(p) { delete[] p; p = nullptr; }
#define SAFE_RELEASE(p) if(p) { p->Release(); p = nullptr; }

#define DECLARE_SINGLE(Type) \
	private:\
		static Type* m_pInst;\
	public:\
		static Type* GetInst()\
		{\
			if(!m_pInst)\
				m_pInst = new Type;\
			return m_pInst;\
		}\
		static void DestroyInst()\
		{\
			SAFE_DELETE(m_pInst);\
		}\
	private:\
		Type();\
		~Type();

#define DEFINITION_SINGLE(Type) Type* Type::m_pInst = nullptr;
#define GET_SINGLE(Type) Type::GetInst()
#define DESTROY_SINGLE(Type) Type::DestroyInst()

#define GETRESOLUTION CCore::GetInst()->GetResolution()

#define KEYDOWN(key) CInput::GetInst()->KeyDown(key)
#define KEYPRESS(key) CInput::GetInst()->KeyPress(key)
#define KEYUP(key) CInput::GetInst()->KeyUp(key)
```

키 상태 매크로

Player.cpp

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

    if( KEYPRESS( "MoveUp" ) )
    {
        MoveYAsSpeed( fDeltaTime, MD_BACK );
    }
    if( KEYPRESS( "MoveDown" ) )
    {
        MoveYAsSpeed( fDeltaTime, MD_FRONT );
    }
    if( KEYPRESS("MoveLeft") )
    {
        MoveXBySpeed( fDeltaTime, MD_BACK );
    }
    if( KEYPRESS( "MoveRight" ) )
    {
        MoveXBySpeed( fDeltaTime, MD_FRONT );
    }
    if( KEYDOWN( "Fire" ) )
    {
        Fire();
    }
    if( KEYDOWN( "Skill1" ) )
    {
        MessageBox( NULL, L"Skill1", L"Skill1", MB_OK );
    }
}
```

활용