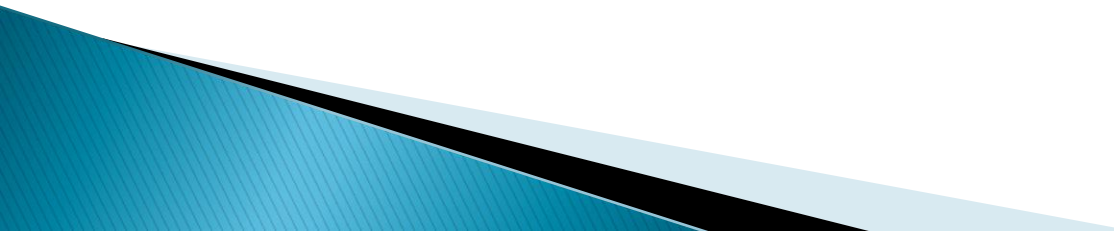
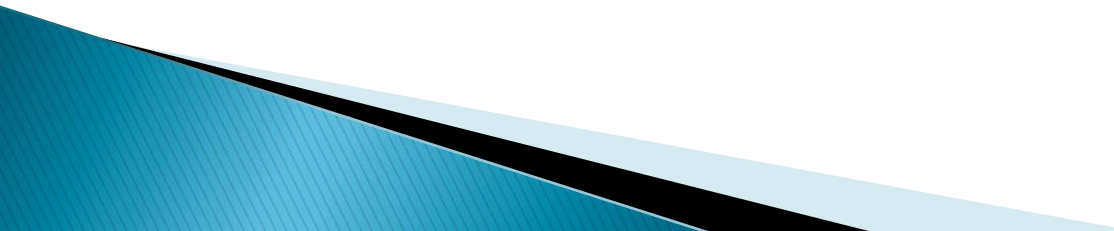


# WinAPI 게임 엔진 만들기

# 목차

1. 게임 엔진이란? . . . 4p
  2. 프로젝트 생성 . . . 5p
  3. Game Engine Mecro . . . 6p
  4. Game Engine Main . . . 8p
  5. Time Manager . . . 15p
  6. Scene Manager . . . 17p
- 

# 목차

- 7. Input Manager · · · 23p
  - 8. Game Engine Bitmap · · · 31p
  - 9. Resource Manager · · · 35p
  - 10. Game Engine UI Tool · · · 40p
  - 11. UI Manager · · · 53p
  - 12. 마무리 · · · 59p
- 

# WinAPI 2D Engine

## ▶ 게임 엔진이란?

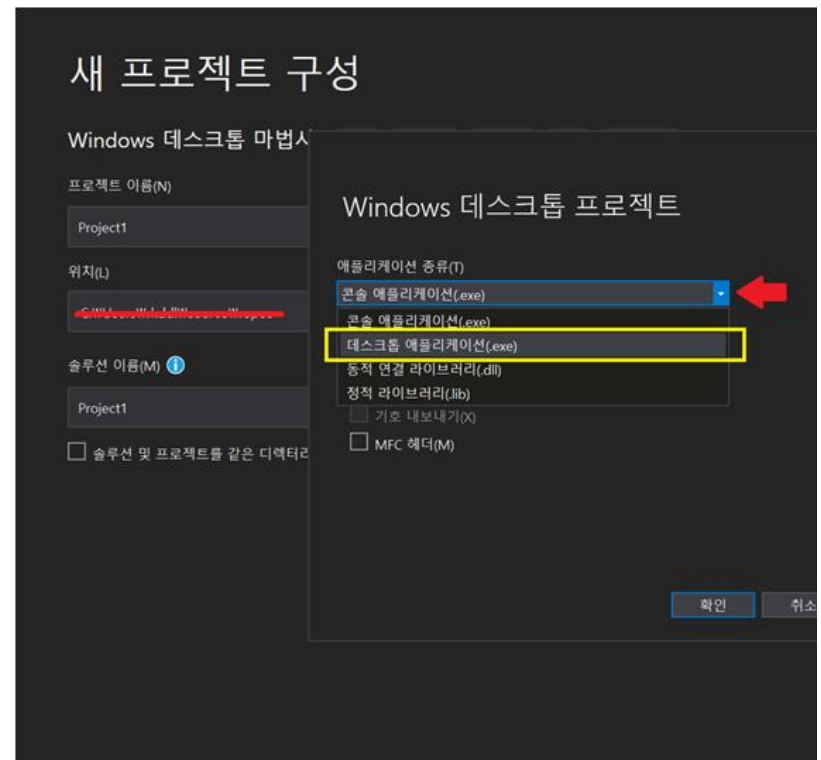
- 게임 개발에 필요한 요소를 미리 만들어 둔 **개발 도구**
- 상용화여 사용되는 게임 엔진은 **재사용에 염두 하여 개발**되기에 하나의 게임에 종속되지 않고 여러 종류의 게임을 만들 수 있게 도와준다
- 회사에 따라서, 특정한 장르의 게임을 만들기 위한 자체 제작한 게임 엔진을 만들어 사용한다



# WinAPI 2D Engine

- ▶ 게임 엔진을 위한 프로젝트 생성
  - 데스크톱 어플리케이션 형식의 **빈 프로젝트** 만들기
  - 빈 내용의 메인 함수 작성
  - **Math, Component 파일을 예제 소스에서 받아서 프로젝트에 추가**
  - Math 코드의 **미구현 부분** 작성하기

```
int APIENTRY wWinMain(_In_ HINSTANCE hInstance,  
                      _In_opt_ HINSTANCE hPrevInstance,  
                      _In_ LPWSTR lpCmdLine,  
                      _In_ int nCmdShow)  
{  
    return 0;  
}
```



# WinAPI 2D Engine

## ▶ Engine에서 사용할 Macro 만들기(EngineMacro.h)

- 엔진에서 전반적으로 사용될 매크로 함수 및 인터페이스 정의
- FPS 및 Window 크기를 지정해 둔다

```
#pragma comment(lib, "msimg32.lib")
#include<crtdbg.h>
#include<windows.h>
#include<time.h>
#include<string>
#include<vector>
#include<map>
#include<functional>
#include <algorithm>
#include "Component.h"

#ifdef ENGINE_MECRO_H
#define ENGINE_MECRO_H

#ifdef DEL
#define DEL(a) if(a) { delete a; a = nullptr; } // delete 매크로 함수.
#endif // !DEL
#ifdef REL_DEL
#define REL_DEL(a) if(a) { a->Release(); delete a; a = nullptr; } // Release() 함수 호출 후 delete
#endif // !REL_DEL
```

# WinAPI 2D Engine

```
typedef std::function<VOID()> EventListener;
namespace ENGINE
{
    enum // 무명(no name) enum, 게임 엔진에 사용되는 여러 기본 데이터 설정 값.
    {
        ClientSize_Width = 800,
        ClientSize_Height = 600,
        Client_Per_X = 50,
        Client_Per_Y = 50/*0 ~ 100%*/,
        FPS = 120/*초당 프레임.*/
    };

    __interface Scene
    {
        VOID Initialize();
        VOID Release();
        VOID Update(CONST FLOAT& deltaTime);
        VOID Draw();
    };
}

#endif // !ENGINE_MECRO_H
```

# WinAPI 2D Engine

## ▶ Engine 메인 만들기(WinApiEngine.h, WinApiEngine.cpp)

- 기본적인 설정을 하여 Window를 만들고 게임 루프를 작동한다
- 디폴트 생성자를 사용하지 않기 위하여 private에 선언

```
// WinApiEngine.h
#include "EngineMacro.h"
#ifdef ENGINE_MAIN_H
#define ENGINE_MAIN_H
namespace ENGINE
{
    class WinApiEngine
    {
    private:
        BOOL isInit;
        INT32 x, y;
        UINT32 width, height;
        std::wstring title;

        WinApiEngine();
    public:
        WinApiEngine(HINSTANCE hInstance, std::wstring title, INT32 per_x, INT32 per_y, UINT32 width, UINT32 height);
        ~WinApiEngine();

        INT Run();
        VOID Release();
    };
}
#endif // !ENGINE_MAIN_H
```



# WinAPI 2D Engine

```
// WinApiEngine.cpp
#include "WinApiEngine.h"

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

namespace ENGINE
{
    WinApiEngine::WinApiEngine(HINSTANCE hInstance, std::wstring title, INT32 per_x, INT32 per_y, UINT32 width, UINT32 height) : isInit(FALSE), title(title), x(0), y(0), width(width), height(height)
    {
        WNDCLASSEXW wcex =
        {
            sizeof(WNDCLASSEX),           // 이 구조체의 크기.
            CS_HREDRAW | CS_VREDRAW,       // 윈도우의 출력 형태, (CS_HREDRAW | CS_VREDRAW : 윈도우의 크기가 변경되면 다시 그린다)
            WndProc,                       // 윈도우 메시지 처리에 사용할 프로시저.
            0,                             // cbClsExtra : 클래스를 위한 여분 메모리 할당.
            0,                             // cbWndExtra : 윈도우를 위한 여분 메모리 할당. 일반적으로 사용 X
            hInstance,                    // 해당 어플리케이션의 인스턴스 핸들.
            0, 0, 0,                      // 아이콘, 커서, 윈도우 배경색 설정.
            NULL,                         // 메뉴 이름. 메뉴의 사용 여부를 결정.
            L"2DEngineWindowClass",       // 윈도우 클래스 이름. 윈도우 생성시 이용.
            NULL                          // 작은 아이콘 설정.
        };
        RegisterClassExW(&wcex);
    }
}
```

# WinAPI 2D Engine

```
HWND hWnd = CreateWindow(  
    TEXT("2DEngineWindowClass"), title.c_str(),  
    WS_SYSMENU | WS_MINIMIZEBOX,  
    CW_USEDEFAULT, 0, CW_USEDEFAULT, 0,  
    nullptr, nullptr, hInstance, nullptr);  
  
if (!hWnd) return;  
  
RECT rcWindow, rcClient;  
GetWindowRect(hWnd, &rcWindow);  
GetClientRect(hWnd, &rcClient);  
// CreateWindow의 인자로 들어가는 너비, 높이 값은 데스크 바 등등의 크기를 포함한 값이기 때문에,  
// 그 차이 값을 구하여 윈도우 사이즈를 원하는 클라이언트 영역으로 다시 만든다.  
UINT32 cx = (rcWindow.right - rcWindow.left) - (rcClient.right - rcClient.left);  
UINT32 cy = (rcWindow.bottom - rcWindow.top) - (rcClient.bottom - rcClient.top);  
  
int screenWidth = GetSystemMetrics(SM_CXFULLSCREEN); // SM_CXFULLSCREEN:전체화면의 너비.  
int screenHeight = GetSystemMetrics(SM_CYFULLSCREEN); // SM_CYFULLSCREEN:전체화면의 높이.  
per_x = clamp(per_x, 0, 100); // per_x, per_y 값 범위를 0 ~ 100 사이로 제한.  
per_y = clamp(per_y, 0, 100);  
x = per_x * 0.01f * (screenWidth - width); // 윈도우를 모니터(Screen) 범위의 특정 위치로 변경.  
y = per_y * 0.01f * (screenHeight - height);  
  
MoveWindow(hWnd, x, y, width + cx, height + cy, false); // Window의 위치 및 크기를 재설정.  
  
ShowWindow(hWnd, SW_SHOWDEFAULT);  
UpdateWindow(hWnd);
```

# WinAPI 2D Engine

```
srand((unsigned)time(NULL));  
/* Scene Initialized */  
  
isInit = TRUE;  
}  
  
WinApiEngine::~WinApiEngine() { Release(); }  
  
INT WinApiEngine::Run()  
{  
    if (!isInit) return -1;  
  
    MSG msg; ZeroMemory(&msg, sizeof(msg));  
    while (WM_QUIT != msg.message) // Game Loop  
    {  
        if (PeekMessage(&msg, nullptr, 0U, 0U, PM_REMOVE))  
        {  
            TranslateMessage(&msg); // 가상 키 코드를 WM_CHAR에서 사용가능한 문자로 변환.  
            DispatchMessage(&msg); // 발생한 메시지를 WndProc()에 발송, WndProc()가 호출.  
        }  
        else { /* Scene Render */ }  
    }  
    Release();  
  
    return (int)msg.wParam;  
}
```

# WinAPI 2D Engine

```
VOID WinApiEngine::Release()  
{  
    /* Scene Destroy */  
}  
} //namespace ENGINE  
  
LRESULT CALLBACK WndProc(HWND hWnd, UINT iMessage, WPARAM wParam, LPARAM lParam)  
{  
    switch ( iMessage)  
    {  
        case WM_DESTROY: PostQuitMessage(0); break;  
        default: return(DefWindowProc(hWnd, iMessage, wParam, lParam));  
    }  
  
    return 0;  
}
```



# WinAPI 2D Engine

## ▶ Main.cpp에 엔진 적용

```
#include "WinApiEngine.h"

int APIENTRY wWinMain(_In_ HINSTANCE hInstance,
                      _In_opt_ HINSTANCE hPrevInstance,
                      _In_ LPWSTR lpCmdLine,
                      _In_ int nCmdShow)
{
    ENGINE::WinApiEngine engine(hInstance, TEXT("WinAPI 2D Engine"),
                                ENGINE::Client_Per_X, ENGINE::Client_Per_Y,
                                ENGINE::ClientSize_Width, ENGINE::ClientSize_Height);

    // TODO: 여기에 새로운 씬 등록 및 로드.

    return engine.Run();
}
```

# WinAPI 2D Engine

## ▶ Template Singleton 만들기(Singleton.h)

- 매니저 클래스들을 쉽게 싱글톤 클래스로 만들기 위한 템플릿 싱글톤 클래스 구축

```
#include "EngineMacro.h"
#ifndef SINGLETON_H
#define SINGLETON_H
template<typename T>
class Singleton
{
private:
    static T* instance;

protected:
    Singleton() {}

public:
    static T* GetInstance()
    {
        if (nullptr == instance) instance = New T;
        return instance;
    }
    static void Destroy() { DEL(instance); }
};
template<typename T> T* Singleton<T>::instance = nullptr;
#endif // !SINGLETON_H
```

# WinAPI 2D Engine

## ▶ Time Manager 만들기(TimeManager.h, TimeManager.cpp)

- TimeManager 클래스의 생성자가 private이기 때문에 Singleton class를 friend로 만들어 준다

```
// TimeManager.h
#include "Singleton.h"
#ifndef TIME_MANAGER_H
#define TIME_MANAGER_H
namespace ENGINE
{
    class TimeManager : public Singleton<TimeManager>
    {
    private:
        UINT32 FPS;
        FLOAT elapsedTime;
        ULONGLONG currTime, lastTime, elapsed;

        TimeManager() : FPS(0), elapsedTime(0.0f), currTime(0), lastTime(0){ }
    public:
        VOID Initialize(UINT32 FPS);
        BOOL Update();
        FLOAT DeltaTime() CONST { return elapsedTime; }

        friend Singleton;
    };
#define TimeMgr TimeManager::GetInstance()
}
#endif // !TIME_MANAGER_H
```

# WinAPI 2D Engine

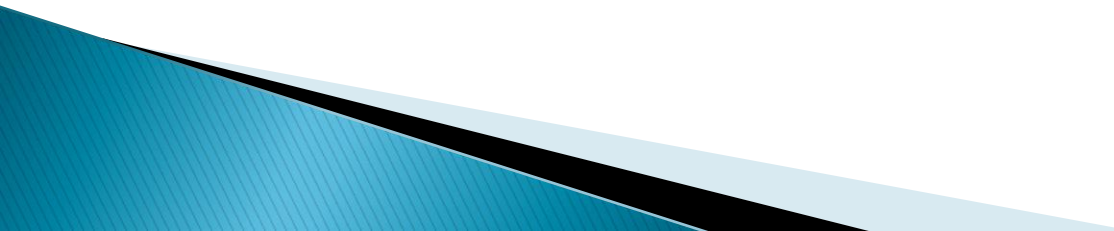
```
// TimeManager.cpp
#include "TimeManager.h"
namespace ENGINE
{
    VOID TimeManager::Initialize(UINT32 FPS)
    {
        this->FPS = 1000 / FPS;
        currTime = lastTime = GetTickCount64();
    }

    BOOL TimeManager::Update()
    {
        currTime = GetTickCount64();

        elapsed = (currTime - lastTime);
        if (elapsed < FPS) return FALSE;

        elapseTime = elapsed * 0.001f;
        lastTime = currTime;

        return TRUE;
    }
}
```





# WinAPI 2D Engine

## ▶ Scene Manager 만들기(SceneManager.h, SceneManager.cpp)

- SceneManager 클래스의 생성자가 **private**이기 때문에 **Singleton class**를 **friend**로 만들어 준다
- TimeManager 적용

```
// SceneManager.h
#include "Singleton.h"
#ifndef SCENE_MANAGER_H
#define SCENE_MANAGER_H
namespace ENGINE
{
    class SceneManager : public Singleton<SceneManager>
    {
    private:
        HWND hWnd;
        HDC hDC, hBackDC;
        UINT32 width, height;

        Scene* currScene;
        std::string nextScene;
        std::map<std::string, Scene*> scenes;

        SceneManager() : hWnd(NULL), hDC(NULL), hBackDC(NULL), width(0), height(0), currScene(NULL) { }
    public:
        ~SceneManager();
```

# WinAPI 2D Engine

```
VOID Initialize(HWND hWnd, UINT32 width, UINT32 height);  
VOID Release();  
VOID Render();
```

```
BOOL RegisterScene(LPCSTR sceneName, Scene* scene); // Scene을 등록.  
BOOL LoadScene(LPCSTR sceneName); // 불러 올 Scene을 설정.
```

```
UINT32 GetWidth() CONST { return width; } // 윈도우의 클라이언트 너비.  
UINT32 GetHeight() CONST { return height; } // 윈도우의 클라이언트 높이.  
HWND GetHWND() CONST { return hWnd; }  
HDC GetBackDC() CONST { return hBackDC; }
```

private:

```
VOID SetScene(); // 등록된 Scene을 적용.  
VOID Update();  
VOID Draw();  
HBITMAP CreateDIBSectionRe(); // Back Bitmap을 만들기위한 비트맵 Creator
```

friend Singleton;

}; // class SceneManager

```
#define SceneMgr SceneManager::GetInstance()  
} //namespace ENGINE  
#endif // !SCENE_MANAGER_H
```

# WinAPI 2D Engine

```
// SceneManager.cpp
#include "SceneManager.h"
#include "TimeManager.h"
namespace ENGINE
{
    SceneManager::~SceneManager() { Release(); }

    VOID SceneManager::Initialize(HWND hWnd, UINT32 width, UINT32 height)
    {
        if (!hWnd) return;

        this->hWnd = hWnd;
        this->width = width;
        this->height = height;
        hDC = GetDC(hWnd);
        hBackDC = CreateCompatibleDC(hDC);
        TimeMgr->Initialize(FPS); // FPS : EngineMacro::ENGINE::enum::FPS
    }

    VOID SceneManager::Release()
    {
        currScene = NULL;
        TimeMgr->Destroy();
        for (std::pair<std::string, Scene*> scene : scenes) REL_DEL(scene.second);
        scenes.clear();
        DeleteObject(hBackDC);
        ReleaseDC(hWnd, hDC);
    }
}
```

# WinAPI 2D Engine

```
VOID SceneManager::Render()
{
    if (!TimeMgr->Update()) return;

    if (currScene)
    {
        Update();
        Draw();
    }
    SetScene();
}

VOID SceneManager::SetScene()
{
    if (nextScene.empty()) return;
    if (currScene) { currScene->Release(); }
    currScene = scenes[nextScene];
    currScene->Initialize();
    nextScene = "";
}

VOID SceneManager::Update()
{
    currScene->Update(TimeMgr->DeltaTime());
}

VOID SceneManager::Draw()
{
    HBITMAP backBitmap = CreateDIBSectionRe();
    SelectObject(hBackDC, backBitmap);

    currScene->Draw();

    BitBlt(hDC, 0, 0, width, height, hBackDC, 0, 0, SRCCOPY);
    DeleteObject(backBitmap);
}
```

# WinAPI 2D Engine

```
BOOL SceneManager::RegisterScene(LPCSTR sceneName, Scene* scene)
{
    if ("" == sceneName || !scene || scenes.find(sceneName) != scenes.end()) return FALSE;
    scenes.insert(std::make_pair(sceneName, scene));
    return TRUE;
}

BOOL SceneManager::LoadScene(LPCSTR sceneName)
{
    if ("" == sceneName || scenes.find(sceneName) == scenes.end()) return FALSE;
    if (currScene) { currScene->Release(); }
    nextScene = sceneName;
    return TRUE;
}

HBITMAP SceneManager::CreateDIBSectionRe()
{
    BITMAPINFO bmpInfo;
    ZeroMemory(&bmpInfo.bmiHeader, sizeof(BITMAPINFOHEADER));
    bmpInfo.bmiHeader.biSize = sizeof(BITMAPINFOHEADER);
    bmpInfo.bmiHeader.biBitCount = 32;
    bmpInfo.bmiHeader.biWidth = width;
    bmpInfo.bmiHeader.biHeight = height;
    bmpInfo.bmiHeader.biPlanes = 1;
    LPVOID pBits;
    return CreateDIBSection(hDC, &bmpInfo, DIB_RGB_COLORS, (void**)&pBits, NULL, 0);
}
} //namespace ENGINE
```

# WinAPI 2D Engine

## ▶ Engine 메인에 SceneManager 적용

```
// WinApiEngine.h
#include "SceneManager.h"
// WinApiEngine.cpp
WinApiEngine::WinApiEngine(...)...
{
    ...
    /* Scene Initialized */
    SceneManager->Initialize(hWnd, width, height);
    ...
}
INT WinApiEngine::Run()
{
    ...
    // Game Loop
    {
        ...
        else { /* Scene Render */ SceneManager->Render(); }
    }
    ...
}
VOID WinApiEngine::Release()
{
    /* Scene Destroy */
    SceneManager->Destroy();
}
```

# WinAPI 2D Engine

## ▶ Input Manager 만들기(InputManager.h, InputManager.cpp)

- InputManager 클래스의 생성자가 private이기 때문에 Singleton class를 friend로 만들어 준다
- Input 전용 프로시저를 만들어 인풋 관련 처리를 한다

```
// InputManager.h
#include "Singleton.h"
#include <unordered_map>
#ifdef INPUT_MANAGER_H
#define INPUT_MANAGER_H
namespace ENGINE
{
    class InputManager : public Singleton<InputManager>
    {
    private:
        enum class InputState { NONE, DOWN, PRESSED, UP };

        // map보다 더 빠른 탐색을 위한 자료구조, 중복 데이터를 허용하지 않는다.
        // map보다 더 많은 데이터를 관리할 때 높은 성능을 발휘한다.
        std::unordered_map<UINT, InputState> keys; // key 입력 상태 정보.
        std::map<UINT, InputState> mouse; // 마우스 버튼 상태 정보.
        POINT mousePosition; // 현재 마우스 커서의 위치.
        std::string str; // 입력된 키의 문자 정보.

        InputManager() : mousePosition({ 0, 0 }) {}
    public:
        ~InputManager();
```

# WinAPI 2D Engine

```
VOID Initialize(){ }
VOID Release();
VOID Update();
VOID InputProc(UINT message, WPARAM wParam, LPARAM lParam); // Input Procedure
LPCSTR GetChar() { return str.c_str(); } // 입력한 키의 문자 정보를 알아온다.
BOOL GetKeyDown(UINT keyID) { return KeyCheck(keyID, InputState::DOWN); } // 확인 하려는 Key가 현재 프레임에서 눌러 졌는가?
BOOL GetKeyPressed(UINT keyID) { return (KeyCheck(keyID, InputState::PRESSED) || KeyCheck(keyID, InputState::DOWN)); } // 확인 하려는 Key가 눌려진 상태인가?
BOOL GetKeyUp(UINT keyID) { return KeyCheck(keyID, InputState::UP); } // 확인 하려는 Key가 현재 프레임에서 눌러 지지 않았는가?
BOOL PressedAnyKey();
BOOL GetMouseButtonDown(UINT keyID) { return ButtonCheck(keyID, InputState::DOWN); }
BOOL GetMouseButtonPressed(UINT keyID) { return (ButtonCheck(keyID, InputState::PRESSED) || ButtonCheck(keyID, InputState::DOWN)); }
BOOL GetMouseButtonUp(UINT keyID) { return ButtonCheck(keyID, InputState::UP); }
POINT GetMousePosition() CONST { return mousePosition; }
private:
VOID StartCapture(); // 마우스 커서 이동 범위 제한.
VOID EndCapture(); // 마우스 커서 이동 제한 종료.
VOID SetKeyDown(UINT keyID) { keys[keyID] = InputState::DOWN; } // 현재 입력된 키를 누른 상태로 변경.
VOID SetKeyUp(UINT keyID) { keys[keyID] = InputState::UP; } // 현재 입력된 키를 누르지 않은 상태로 변경.
VOID SetMouseButtonDown(UINT keyID);
VOID SetMouseButtonUp(UINT keyID);
VOID SetMousePosition(LPARAM lParam) { mousePosition = { LOWORD(lParam), HIWORD(lParam) }; }
BOOL KeyCheck(UINT keyID, CONST InputState& state);
BOOL ButtonCheck(UINT keyID, CONST InputState& state);

friend Singleton;
}; // class InputManager
#define InputMgr InputManager::GetInstance()
} // namespace ENGINE
#endif // !INPUT_MANAGER_H
```



# WinAPI 2D Engine

```
// InputManager.cpp
#include "InputManager.h"
#include "SceneManager.h"
namespace ENGINE
{
    InputManager::~InputManager() { Release(); }

    VOID InputManager::Release()
    {
        str = "";
        keys.clear();
        mouse.clear();
        EndCapture();
    }

    VOID InputManager::Update()
    {
        str = "";
        for (std::pair<CONST UINT, InputState>& key : keys)
        {
            switch (key.second)
            {
            {
                case InputState::DOWN: key.second = InputState::PRESSED; break;
                case InputState::UP: key.second = InputState::NONE; break;
            }
        }
        for (std::pair<CONST UINT, InputState>& button : mouse)
        {
            switch (button.second)
            {
            {
                case InputState::DOWN: button.second = InputState::PRESSED; break;
                case InputState::UP: button.second = InputState::NONE; break;
            }
        }
    }
}
```

# WinAPI 2D Engine

```
VOID InputManager::InputProc(UINT message, WPARAM wParam, LPARAM lParam)
{
    switch (message)
    {
    {
        case WM_CHAR: str = wParam; break;
        case WM_KEYDOWN: case WM_SYSKEYDOWN: SetKeyDown(wParam); break;
        case WM_KEYUP: case WM_SYSKEYUP: SetKeyUp(wParam); break;
        case WM_LBUTTONDOWN: SetMouseButtonDown(VK_LBUTTON); break;
        case WM_RBUTTONDOWN: SetMouseButtonDown(VK_RBUTTON); break;
        case WM_MBUTTONDOWN: SetMouseButtonDown(VK_MBUTTON); break;
        case WM_XBUTTONDOWN:
        {
            switch (wParam)
            {
            {
                case XBUTTONDOWN1: SetMouseButtonDown(VK_XBUTTONDOWN1); break; // 마우스 왼쪽 뒤 버튼.
                case XBUTTONDOWN2: SetMouseButtonDown(VK_XBUTTONDOWN2); break; // 마우스 왼쪽 앞 버튼.
            }
        }
        Break;
        case WM_LBUTTONUP: SetMouseButtonUp(VK_LBUTTON); break;
        case WM_RBUTTONUP: SetMouseButtonUp(VK_RBUTTON); break;
        case WM_MBUTTONUP: SetMouseButtonUp(VK_MBUTTON); break;
        case WM_XBUTTONUP:
        {
            switch (wParam)
            {
            {
                case XBUTTONUP1: SetMouseButtonUp(VK_XBUTTONUP1); break; // 마우스 왼쪽 뒤 버튼.
                case XBUTTONUP2: SetMouseButtonUp(VK_XBUTTONUP2); break; // 마우스 왼쪽 앞 버튼.
            }
        }
        Break;
        case WM_MOUSEMOVE: SetMousePosition(lParam); break;
    }
}
```

# WinAPI 2D Engine

```
BOOL InputManager::PressedAnyKey()
{
    for (std::pair<CONST UINT, InputState>& key : keys) { switch (key.second) { case InputState::DOWN: case InputState::PRESSED: return TRUE; } }
    return FALSE;
}

VOID InputManager::StartCapture()
{
    HWND hWnd = SceneMgr->GetHWND();
    SetCapture(hWnd);
    POINT leftTop = { 0, 0 };
    POINT rightBottom = { SceneMgr->GetWidth() + 1, SceneMgr->GetHeight() + 1 };
    ClientToScreen(hWnd, &leftTop);
    ClientToScreen(hWnd, &rightBottom);
    RECT clip = { leftTop.x, leftTop.y, rightBottom.x, rightBottom.y };
    ClipCursor(&clip);
}

VOID InputManager::EndCapture()
{
    ClipCursor(NULL);
    ReleaseCapture();
}

VOID InputManager::SetMouseButtonDown(UINT keyID)
{
    StartCapture();
    mouse[keyID] = InputState::DOWN;
}

VOID InputManager::SetMouseButtonUp(UINT keyID)
{
    mouse[keyID] = InputState::UP;
    EndCapture();
}
```

# WinAPI 2D Engine

```
BOOL InputManager::KeyCheck(UINT keyID, CONST InputManager::InputState& state)
{
    std::unordered_map<UINT, InputState>::iterator iter = keys.find(keyID);
    if (keys.end() != iter) return (state == iter->second);

    keys.insert(std::make_pair(keyID, InputState::NONE));

    return FALSE;
}

BOOL InputManager::ButtonCheck(UINT keyID, const InputState& state)
{
    std::map<UINT, InputState>::iterator iter = mouse.find(keyID);
    if (mouse.end() != iter) return (state == iter->second);

    mouse.insert(std::make_pair(keyID, InputState::NONE));

    return FALSE;
}
} // namespace ENGINE
```

# WinAPI 2D Engine

## ▶ Engine 메인에 InputManager 적용

- 윈도우 프로시저에 인풋 프로시저 추가

```
// WinApiEngine.cpp
#include "InputManager.h"
LRESULT CALLBACK WndProc(HWND hWnd, UINT iMessage, WPARAM wParam, LPARAM lParam)
{
    ENGINE::InputMgr->InputProc(iMessage, wParam, lParam);
    ...
}
```

## ▶ SceneManager에 InputManager 적용

```
// SceneManager.cpp
#include "InputManager.h"
VOID SceneManager::Initialize(HWND hWnd, UINT32 width, UINT32 height)
{
    ...
    InputMgr->Initialize();
}
```

# WinAPI 2D Engine

```
VOID SceneManager::Release()
```

```
{
```

```
    currScene = NULL;
```

```
    InputMgr->Destroy();
```

```
    ...
```

```
}
```

```
VOID SceneManager::Render()
```

```
{
```

```
    if (!TimeMgr->Update()) return;
```

```
    if (currScene)
```

```
    {
```

```
        ...
```

```
        InputMgr->Update();
```

```
    }
```

```
}
```



# WinAPI 2D Engine

## ▶ Bitmap 만들기(Bitmap.h, Bitmap.cpp)

```
// Bitmap.h
#include "EngineMacro.h"
#ifndef BITMAP_H
#define BITMAP_H
namespace ENGINE
{
    enum Pivot { Center= 1 << 0, Left= 1 << 1, Right= 1 << 2, Top= 1 << 3, Bottom= 1 << 4, };

    class Bitmap
    {
    private:
        HDC memDC;
        HBITMAP hBitmap;
        SIZE origin, dest;
        RECT src;

    public:
        ~Bitmap();
    };
}
```

# WinAPI 2D Engine

```
VOID Load(std::string name); // 비트맵 이미지를 로드하고 메모리 DC를 만든다.
VOID SetDrawSize(UINT width = 0U, UINT height = 0U); // 실제 화면에 출력될 크기를 변경.
VOID ResetSize() { dest = origin; } // 화면에 출력되는 크기를 원본 크기로 변경.

// Sprites 이미지의 임의의 위치부터 임의의 크기만큼 출력.
VOID SliceSource(UINT cx, UINT cy, UINT width, UINT height)
{
    src = { (LONG)cx, (LONG)cy, (LONG)width, (LONG)height };
}
VOID SetPivot(INT pivot);
VOID ResetPivot() { pivot = { 0, 0 }; }

BOOL BitBlt(INT32 x, INT32 y); // 원본 그대로 화면에 출력.
BOOL StretchBlt(INT32 x, INT32 y); // 설정한 출력 사이즈로 변경하여 출력.
// 설정한 출력 사이즈로 변경하고 지정 색을 투명 처리하여 출력.
BOOL TransparentBlt(INT32 x, INT32 y, UINT transparent = RGB(255, 0, 255));

SIZE GetBitmapSize() CONST { return origin; } // 원본 비트맵의 크기.
SIZE GetSize() CONST { return dest; } // 화면에 출력되는 이미지의 크기.
}; // class Bitmap
}
#endif // !BITMAP_H
```



# WinAPI 2D Engine

```
// Bitmap.cpp
#include "Bitmap.h"
#include "SceneManager.h"
namespace ENGINE
{
    Bitmap::~Bitmap()
    {
        DeleteDC(memDC);
        DeleteObject(hBitmap);
    }

    VOID Bitmap::Load(std::string name)
    {
        memDC = CreateCompatibleDC(SceneMgr->GetBackDC()); // 메모리 DC 생성.
        hBitmap = (HBITMAP)LoadImageA(NULL, name.c_str(), IMAGE_BITMAP, 0, 0, LR_CREATEDIBSECTION | LR_DEFAULTSIZE | LR_LOADFROMFILE); //
        비트맵 이미지 로드.
        SelectObject(memDC, hBitmap); // 메모리 DC에 비트맵 이미지 적용.

        BITMAP BitMap;
        GetObject(hBitmap, sizeof(BITMAP), &BitMap); // 비트맵의 정보 얻기.
        origin.cx = BitMap.bmWidth; // 비트맵의 원본 크기 저장.
        origin.cy = BitMap.bmHeight;
        dest = origin; src = { 0, 0, origin.cx, origin.cy }; // 화면에 그려줄 크기와 rec를 원본 크기로 지정.

        pivot = { 0, 0 };
    }

    VOID Bitmap::SetDrawSize(UINT width, UINT height) { dest = { (LONG)width, (LONG)height }; }
```

# WinAPI 2D Engine

```
VOID Bitmap::SetPivot(INT pivot)
{
    int halfWidth = dest.cx * 0.5f;
    int halfHeight = dest.cy * 0.5f;
    this->pivot = { -halfWidth, -halfHeight };

    if (pivot & Pivot::Left) this->pivot.x += halfWidth;
    if (pivot & Pivot::Right) this->pivot.x += -halfWidth;
    if (pivot & Pivot::Top) this->pivot.y += halfHeight;
    if (pivot & Pivot::Bottom) this->pivot.y += -halfHeight;
}

BOOL Bitmap::BitBlt(INT32 x, INT32 y)
{
    return ::BitBlt(SceneMgr->GetBackDC(), pivot.x + x, pivot.y + y, src.right, src.bottom, memDC, src.left, src.top, SRCCOPY);
}

BOOL Bitmap::StretchBlt(INT32 x, INT32 y)
{
    return ::StretchBlt(SceneMgr->GetBackDC(), pivot.x + x, pivot.y + y, dest.cx, dest.cy, memDC, src.left, src.top, src.right, src.bottom, SRCCOPY);
}

BOOL Bitmap::TransparentBlt(INT32 x, INT32 y, UINT transparent)
{
    return ::TransparentBlt(SceneMgr->GetBackDC(), pivot.x + x, pivot.y + y, dest.cx, dest.cy, memDC, src.left, src.top, src.right, src.bottom, transparent);
}
} // namespace ENGINE
```

# WinAPI 2D Engine

## ▶ Resource Manager 만들기(ResourceManager.h, ResourceManager.cpp)

- Resources 폴더에서 비트맵 이미지를 로드, 보관, 관리한다

```
// ResourceManager.h
#include "Singleton.h"
#include "Bitmap.h"
#ifndef RESOURCE_MANAGER_H
#define RESOURCE_MANAGER_H
namespace ENGINE
{
    static const char* defaultPath = "Resources\\";

    class ResourceManager : public Singleton<ResourceManager>
    {
    private:
        std::map<std::string, Bitmap*> resources;

        ResourceManager() {}
    public:
        ~ResourceManager();
    }
```

# WinAPI 2D Engine

```
VOID Initialize();  
Bitmap* GetBitmap(const std::string& name);  
// 비트맵 이미지를 로드한다.  
BOOL Load(const std::string& fileName);  
// 폴더의 모든 비트맵 이미지를 로드.  
// 하위 폴더는 검색하지 않는다.  
VOID LoadAll(const std::string& directoryName = "");  
// 로드되어 있는 모든 리소스를 지운다.  
VOID Clear();  
  
friend Singleton;  
}; // class ResourceManager  
#define ResourceMgr ResourceManager::GetInstance()  
} // namespace ENGINE  
#endif // !RESOURCE_MANAGER_H
```

# WinAPI 2D Engine

```
// ResourceManager.cpp
#include "ResourceManager.h"
#include <fstream>
#include <iostream> // access()

#define _SILENCE_EXPERIMENTAL_FILESYSTEM_DEPRECATION_WARNING // c++17 이상의 경우 #include<filesystem>를 사용.
#include <experimental/filesystem> // directory_iterator()
using namespace std::experimental::filesystem;
namespace ENGINE
{
    ResourceManager::~ResourceManager() { Clear(); }

    VOID ResourceManager::Initialize() { Clear(); }

    BOOL ResourceManager::Load(const std::string& fileName)
    {
        if (fileName.empty()) return FALSE;

        std::string path = defaultPath + fileName;
        // 0 : 파일 및 디렉토리의 존재 유무.
        // 2 : 쓰기 접근 가능 여부.
        // 4 : 읽기 접근 가능 여부.
        // 6 : 읽기, 쓰기 접근 가능 여부.
        // return : 0(true), -1(false)
        if(0 != _access(path.c_str(), 0)) return FALSE;

        Bitmap* bitmap = new Bitmap;
        bitmap->Load(defaultPath + fileName);
        resources.insert(make_pair(fileName, bitmap));

        return TRUE;
    }
}
```

# WinAPI 2D Engine

```
VOID ResourceManager::LoadAll(const std::string& directoryName)
{
    v1::path path;
    std::string name;
    // directory_iterator : 디렉토리(폴더) 내의 파일 및 폴더를 찾기는다, 하위 디렉토리는 탐색하지 않는다.
    for (const auto& file : directory_iterator(defaultPath + directoryName))
    {
        path = file.path();
        if (".bmp" != path.extension().u8string()) continue; // 비트맵 리소스만을 로드.
        name = path.u8string();
        if (nullptr != GetBitmap(name)) continue; // 이미 가지고 있는 파일은 로드에서 제외.

        name.replace(name.find(defaultPath), strlen(defaultPath), ""); // Load()에서 defaultPath를 추가하여 쓰기 때문에 문자열에서 제거.
        Load(name);
    }
}

Bitmap* ResourceManager::GetBitmap(const std::string& name)
{
    std::map<std::string, Bitmap*>::iterator iter = resources.find(name);
    if (resources.end() != iter) return iter->second;
    return nullptr;
}

VOID ResourceManager::Clear()
{
    for (std::pair<std::string, Bitmap*> resource : resources) DEL(resource.second);
    resources.clear();
}
} // namespace ENGINE
```

# WinAPI 2D Engine

## ▶ SceneManager에 ResourceManager 적용

```
// SceneManager.cpp
```

```
#include "ResourceManager.h"
```

```
VOID SceneManager::Initialize(HWND hWnd, UINT32 width, UINT32 height)
```

```
{  
    ...  
    ResourceManager->Initialize();  
}
```

```
VOID SceneManager::Release()
```

```
{  
    currScene = NULL;  
    ResourceManager->Destroy();  
    ...  
}
```

```
HRESULT SceneManager::SetScene()
```

```
{  
    ...  
    if (currScene)  
    {  
        currScene->Release();  
        ResourceManager->Clear();  
    }  
    ...  
}
```

# WinAPI 2D Engine

## ▶ UI Tool 만들기(UIPanel.h, UIPanel.cpp)

- 특별한 기능이 없는 UI Tool Base, 계층구조 등을 만들 때 사용 가능

```
// UIPanel.h
#include "EngineMacro.h"
#ifndef UI_PANEL_H
#define UI_PANEL_H
namespace ENGINE
{
    enum class UIType { PANEL, IMAGE, BUTTON, LABEL };
    class UIPanel
    {
    protected:
        BOOL isEnabled, pivotCenter;
        POINT position, localPosition;
        RECT rect;
        SIZE size;
        UIPanel* parent;
        std::vector<UIPanel*> child;
        UIType uiType;

    public:
        UIPanel();
        virtual ~UIPanel();
    };
}
```



# WinAPI 2D Engine

```
private:
    VOID SetParent(UIPanel* parent) { this->parent = parent; }
    VOID SetLocalPos(INT32 x, INT32 y);
protected:
    virtual VOID SetRect(BOOL pivotCenter) { rect = { localPosition.x, localPosition.y, localPosition.x, localPosition.y }; }
public:
    virtual VOID SetPosition(INT32 x, INT32 y, BOOL pivotCenter = FALSE); // 월드(스크린) 좌표 상의 위치.
    virtual VOID SetLocalPosition(INT32 x, INT32 y, BOOL pivotCenter = FALSE); // 부모 UI를 중심으로 한 위치.
    virtual VOID Update();
    virtual VOID Draw();
    // 자식으로 추가된 ui는 알아서 메모리를 해제한다.
    UIPanel* AddChildUI(UIPanel* ui); // 계층구조, 자식 UI 추가.
    VOID RemoveChildUI(UIPanel* ui); // 자식에서 제외하고 메모리 해제를 하지 않는다.
    VOID RefreshPos(); // 변경된 자신의 위치를 중심으로 자식 UI의 위치를 재배치.
    VOID SetEnable(BOOL enable) { isEnabled = enable; }
    BOOL Enable() CONST { return isEnabled; }
    UINT32 GetChildCount() CONST { return child.size(); }
    UIPanel* GetChild(INT32 index);
    UIPanel* GetParent() CONST { return parent; }
    POINT GetPosition() CONST { return position; }
    POINT GetLocalPosition() CONST { return localPosition; }
    UIType GetUIType() CONST { return uiType; }
    SIZE GetSize() CONST { return size; }
}; // class UIPanel
} // namespace ENGINE
#endif // !UI_PANEL_H
```

# WinAPI 2D Engine

```
// UIPanel.cpp
#include "UIPanel.h"
namespace ENGINE
{
    UIPanel::UIPanel() :
        isEnabled(TRUE), pivotCenter(FALSE),
        position({ 0, 0 }), localPosition({ 0, 0 }),
        rect({ 0, 0, 0, 0 }), size({ 0, 0 }),
        parent(nullptr) { uiType = UType::PANEL; }

    UIPanel::~UIPanel() { child.clear(); }

    VOID UIPanel::SetLocalPos( INT32 x, INT32 y)
    {
        position = localPosition = { x, y };
        if (parent) position = { x + parent->rect.left, y + parent->rect.top };
        SetRect(pivotCenter);
        for (UIPanel* ui : child) ui->RefreshPos();
    }

    VOID UIPanel::SetPosition( INT32 x, INT32 y, BOOL pivotCenter)
    {
        this->pivotCenter = pivotCenter;
        if (parent) SetLocalPos(x - parent->rect.left, y - parent->rect.top);
        else SetLocalPos(x, y);
    }

    VOID UIPanel::SetLocalPosition( INT32 x, INT32 y, BOOL pivotCenter)
    {
        this->pivotCenter = pivotCenter;
        SetLocalPos(x, y);
    }
}
```

# WinAPI 2D Engine

```
VOID UIPanel::Update()
{
    if (!isEnabled) return;
    for (UIPanel* ui : child) ui->Update();
}

VOID UIPanel::Draw()
{
    if (!isEnabled) return;
    for (UIPanel* ui : child) ui->Draw();
}

UIPanel* UIPanel::AddChildUI(UIPanel* ui)
{
    if (!ui) return nullptr;

    child.push_back(ui);
    ui->SetParent(this);

    // 중복 제거.
    std::sort(child.begin(), child.end());
    child.erase(std::unique(child.begin(), child.end()), child.end());

    return ui;
}
```

# WinAPI 2D Engine

```
VOID UIPanel::RemoveChildUI(UIPanel* ui)
{
    if (!ui) return;

    std::vector<UIPanel*>::iterator iter = std::find(child.begin(), child.end(), ui);
    if (child.end() != iter)
    {
        child.erase(iter);
        (*iter)->SetParent(nullptr);
    }
}

VOID UIPanel::RefreshPos()
{
    SetLocalPos(localPosition.x, localPosition.y);
}

UIPanel* UIPanel::GetChild(INT32 index)
{
    if (0 > index || child.size() <= index) return nullptr;
    return child[index];
}
} // namespace ENGINE
```

# WinAPI 2D Engine

## ▶ UI Tool 만들기(UILabel.h, UILabel.cpp)

- Text 출력을 위한 UI Tool, 폰트 및 색상 등의 변경이 가능

```
// UILabel.h
#include "UIPanel.h"
#ifdef UI_LABEL_H
#define UI_LABEL_H
namespace ENGINE
{
    class UILabel : public UIPanel
    {
    protected:
        std::string text;
        COLORREF color;
        HFONT font, oldFont;
    public:
        UILabel() : color(0), font(NULL), oldFont(NULL) { uiType = UType::LABEL; }
        VOID Initialize(CONST std::string& text, COLORREF color = RGB(0, 0, 0), HFONT font = NULL);
        VOID SetColor(COLORREF color) { this->color = color; }
        VOID SetText(CONST std::string& text);
        VOID SetFont(HFONT font);
        LPCSTR GetText() { return text.c_str(); }
        virtual VOID Draw() override;
    protected:
        virtual VOID SetRect(BOOL pivotCenter) override;
    };
}
```

```
#endif // !UI_LABEL_H
```

# WinAPI 2D Engine

```
// UILabel.cpp
#include "UILabel.h"
#include "SceneManager.h"
namespace ENGINE
{
    VOID UILabel::SetRect(BOOL pivotCenter)
    {
        if (font) oldFont = (HFONT)SelectObject(SceneMgr->GetBackDC(), font);
        GetTextExtentPoint32A(SceneMgr->GetBackDC(), text.c_str(), text.length(), &size);
        if (font) SelectObject(SceneMgr->GetBackDC(), oldFont);
        POINT pos = position;
        if (pivotCenter)
        {
            pos.x -= size.cx * 0.5f;
            pos.y -= size.cy * 0.5f;
        }
        rect = { pos.x, pos.y, pos.x + size.cx, pos.y + size.cy };
    }

    VOID UILabel::Initialize(const std::string& text, COLORREF color, HFONT font)
    {
        SetBkMode(SceneMgr->GetBackDC(), TRANSPARENT);
        SetColor(color);
        SetFont(font);
        SetText(text);
    }
}
```

# WinAPI 2D Engine

```
VOID UILabel::SetText(const std::string& text)
{
    this->text = text;
    SetRect(pivotCenter);
}

VOID UILabel::SetFont(HFONT font)
{
    if (font)
    {
        this->font = font;
        if (!text.empty()) SetRect(pivotCenter);
    }
}

VOID UILabel::Draw()
{
    if (!isEnabled) return;
    if (font) oldFont = (HFONT)SelectObject(SceneMgr->GetBackDC(), font);
    SetTextColor(SceneMgr->GetBackDC(), color);
    DrawTextA(SceneMgr->GetBackDC(), text.c_str(), text.length(), &rect, NULL);
    if (font) SelectObject(SceneMgr->GetBackDC(), oldFont);
    UIPanel::Draw();
}
} // namespace ENGINE
```

# WinAPI 2D Engine

## ▶ UI Tool 만들기(UIImage.h, UIImage.cpp)

- Image 형식의 UI를 만들기 위한 UI Tool, DrawType에 맞게 원본으로 그리거나 투명화하여 그린다
- 투명색은 분홍색(RGB(255, 0, 255))으로 고정되어 있다

```
// UIImage.h
#include "UIPanel.h"
#include "SceneManager.h"
#include "Bitmap.h"
#ifdef UI_IMAGE_H
#define UI_IMAGE_H
namespace ENGINE
{
    enum class DrawType { Normal, Transparent };
    class UIImage : public UIPanel
    {
    protected:
        Bitmap* image;
        DrawType type;

    public:
        UIImage() : image(nullptr), type(DrawType::Normal) { uiType = UType::IMAGE; }
        VOID Initialize(CONST std::string& ImageName, DrawType type = DrawType::Normal);
        virtual VOID Draw() override;
    protected:
        virtual VOID SetRect(BOOL pivotCenter) override;
    };
}
#endif // !UI_IMAGE_H
```



# WinAPI 2D Engine

```
// UIImage.cpp
#include "UIImage.h"
#include "ResourceManager.h"
namespace ENGINE
{
    VOID UIImage::SetRect(BOOL pivotCenter)
    {
        POINT pos = position;
        size = image->GetBitmapSize();
        if (pivotCenter)
        {
            pos.x -= size.cx * 0.5f;
            pos.y -= size.cy * 0.5f;
        }
        rect = { pos.x, pos.y, pos.x + size.cx, pos.y + size.cy };
    }
    VOID UIImage::Initialize(CONST std::string& ImageName, DrawType type)
    {
        image = ResourceManager->GetBitmap(ImageName);
        this->type = type;
        SetRect(pivotCenter);
    }
    VOID UIImage::Draw()
    {
        if (!isEnabled) return;
        if (image)
        {
            switch (type)
            {
            {
                case DrawType::Transparent: image->TransparentBlt(rect.left, rect.top); break;
                default: image->Blt(rect.left, rect.top); break;
            }
            }
        }
        UIPanel::Draw();
    }
} // namespace ENGINE
```

# WinAPI 2D Engine

## ▶ UI Tool 만들기(UIButton.h, UIButton.cpp)

- Mouse Button Pressed, click Event가 발생하면 Callback으로 Handler를 호출하여 해당 이벤트를 실행한다

```
// UIButton.h
#include "UIImage.h"
#ifdef UI_BUTTON_H
#define UI_BUTTON_H
namespace ENGINE
{
    class UIButton : public UIImage
    {
    public:
        enum class ButtonState { NONE, HOVER, PRESSED };
        protected:
            EventListener clickListener, pressedListener;
            ButtonState state;
            BOOL isUp, isInteractable;
            Bitmap *normal, *pressed, *hover, *disable;

        public:
            UIButton();

            VOID Initialize(CONST std::string& normal, CONST std::string& pressed = "", CONST std::string& hover = "", CONST std::string& disable = "", DrawType
            type = DrawType::Normal);
            VOID SetListener(EventListener click, EventListener pressed = nullptr);
            VOID SetInteractable(BOOL interactable) { isInteractable = interactable; }
            BOOL IsInteractable() CONST { return isInteractable; }
            virtual VOID Update() override;
    };
}
#endif // !UI_BUTTON_H
```

# WinAPI 2D Engine

```
// UIButton.cpp
#include "UIButton.h"
#include "ResourceManager.h"
#include "InputManager.h"
namespace ENGINE
{
    UIButton::UIButton() :
        clickListener(nullptr), pressedListener(nullptr),
        state(ButtonState::NONE),
        isUp(FALSE), isInteractable(TRUE),
        normal(nullptr), pressed(nullptr), hover(nullptr), disable(nullptr)
    {
        uiType = UIType::BUTTON;
    }

    VOID UIButton::Initialize(CONST std::string& normal, CONST std::string& pressed, CONST std::string& hover, CONST std::string& disable, DrawType type)
    {
        UIImage::Initialize(normal, type);

        this->disable = this->hover = this->pressed = this->normal = image;
        if(!pressed.empty()) this->pressed = ResourceManager->GetBitmap(pressed);
        if(!hover.empty()) this->hover = ResourceManager->GetBitmap(hover);
        if(!disable.empty()) this->disable = ResourceManager->GetBitmap(disable);
    }

    VOID UIButton::SetListener(EventListener click, EventListener pressed)
    {
        clickListener = click;
        pressedListener = pressed;
    }
}
```

# WinAPI 2D Engine

```
VOID UIButton::Update()
{
    if (!isEnabled) return;

    UIImage::Update();

    if (!isInteractable)
    {
        state = ButtonState::NONE;
        if(disable) image = disable;
        return;
    }
    isUp = InputMgr->GetMouseButtonUp(VK_LBUTTON);
    if (PtInRect(&rect, InputMgr->GetMousePosition()))
    {
        switch (state)
        {
            {
                case ButtonState::NONE: state = ButtonState::HOVER;
                case ButtonState::HOVER: if (InputMgr->GetMouseButtonDown(VK_LBUTTON)) state = ButtonState::PRESSED; break;
                case ButtonState::PRESSED: if (isUp && clickListener) clickListener(); break;
            }
        }
    }
    else if(ButtonState::PRESSED != state) state = ButtonState::NONE;
    switch (state)
    {
        {
            case ButtonState::NONE: image = normal; break;
            case ButtonState::HOVER: image = hover; break;
            case ButtonState::PRESSED: image = pressed; if(isUp) state = ButtonState::NONE; if (pressedListener) pressedListener(); break;
        }
    }
}
} // namespace ENGINE
```

# WinAPI 2D Engine

## ▶ UI Manager 만들기(UIManager.h, UIManager.cpp)

- 해당 씬에서 사용할 UI를 가지고, 사용되는 모든 UI를 갱신 및 그리기를 처리하여 준다

```
// UIManager.h
#include "Singleton.h"
#include "UIButton.h" //#include "UIImage.h"
#include "UILabel.h" //#include "UIPanel.h"
#ifndef UI_MANAGER_H
#define UI_MANAGER_H
namespace ENGINE
{
    class UIManager : public Singleton<UIManager>
    {
    private:
        std::map<std::string, UIPanel*> map_UI;
        std::vector<UIPanel*> child_UI;

        UIManager() {}
    public:
        ~UIManager();

        VOID Initialize();
        VOID Clear();
        VOID Update();
        VOID Draw();
    };
}
```

# WinAPI 2D Engine

```
template<typename T> T* AddUI(std::string name, UIPanel* parent = nullptr);
UIPanel* GetUI(std::string name);
BOOL Remove(std::string name);

friend Singleton;
}; // class UIManager

template<typename T>
inline T* UIManager::AddUI(std::string name, UIPanel* parent)
{
    if (name.empty()) return nullptr; // 빈 문자열을 이름으로 지정할 수 없게 한다.

    auto iter = map_UI.find(name); // 이미 같은 이름의 UI가 있을 경우 추가 실패.
    if (map_UI.end() != iter) return nullptr; // 이미 있는 UI의 타입이 다를 수 있기에 null을 반환.

    T* ui = New T;
    if (parent) parent->AddChildUI(ui);

    map_UI.insert(std::make_pair(name, ui));

    return ui;
}

#define UIMgr UIManager::GetInstance()
}
#endif // !UI_MANAGER_H
```

# WinAPI 2D Engine

```
// UIManager.cpp
#include "UIManager.h"
namespace ENGINE
{
    UIManager::~UIManager() { Clear(); }

    VOID UIManager::Initialize() { Clear(); }

    VOID UIManager::Clear()
    {
        for (std::pair<std::string, UIPanel*> pair : map_UI) DEL(pair.second);
        map_UI.clear();
    }

    VOID UIManager::Update()
    {
        // 부모가 있을 경우 부모 쪽에서 Update() 함수를 호출 하고 있다.
        for (std::pair<std::string, UIPanel*> pair : map_UI)
            if (!pair.second->GetParent()) pair.second->Update();
    }

    VOID UIManager::Draw()
    {
        // 부모가 있을 경우 부모 쪽에서 Draw() 함수를 호출 하고 있다.
        for (std::pair<std::string, UIPanel*> pair : map_UI)
            if (!pair.second->GetParent()) pair.second->Draw();
    }
}
```

# WinAPI 2D Engine

```
UIPanel* UIManager::GetUI(std::string name)
{
    auto iter = map_UI.find(name);
    if (map_UI.end() != iter) return iter->second;

    return nullptr;
}

BOOL UIManager::Remove(std::string name)
{
    auto ui = GetUI(name);
    if (nullptr != ui)
    {
        map_UI.erase(name);
        DEL(ui);

        return TRUE;
    }

    return FALSE;
}

} // namespace ENGINE
```



# WinAPI 2D Engine

## ▶ SceneManager에 UIManager 적용

```
// SceneManager.cpp
#include "UIManager.h"

VOID SceneManager::Initialize(HWND hWnd, UINT32 width, UINT32 height)
{
    ...
    UIManager->Initialize();
}

VOID SceneManager::Release()
{
    currScene = NULL;
    UIManager->Destroy();
    ...
}

HRESULT SceneManager::SetScene()
{
    ...
    if (currScene)
    {
        currScene->Release();
        UIManager->Clear();
        ResourceMgr->Clear();
    }
    ...
}
```

# WinAPI 2D Engine

```
VOID SceneManager::Update()
```

```
{
```

```
    UIMgr->Update();
```

```
    currScene->Update(TimeMgr->DeltaTime());
```

```
}
```

```
VOID SceneManager::Draw()
```

```
{
```

```
    HBITMAP backBitmap = CreateDIBSectionRe();
```

```
    SelectObject(hBackDC, backBitmap);
```

```
    currScene->Draw();
```

```
    UIMgr->Draw();
```

```
    BitBlt(hDC, 0, 0, width, height, hBackDC, 0, 0, SRCCOPY);
```

```
    DeleteObject(backBitmap);
```

```
}
```



# WinAPI 2D Engine

예제소스에서  
**Components, Resources** 폴더와 파일,  
**DemoScene** 파일을 **추가**하여  
실행하여 **구조**를 **파악**해 봅시다