DirectX 資料④

入力処理編

ゲームエンジニア科２年

1. キーボード

① Input クラスを作成

<Input.h>

```
#pragma once

#include "Global.h"


class Input
{
public:
    Input();
    ~Input();
};
```

② ヘッダー・ライブラリ

```
#pragma once

#include "Global.h"
#include <dinput.h>


#pragma comment(lib,"dxguid.lib")
#pragma comment(lib,"dinput8.lib")

class Input
{
```

③ DirectInput オブジェクトの準備

<Input.h>

```
class Input
{
    LPDIRECTINPUT8          pDinput;    //DirectInputオブジェクト

public:
    Input();
    ~Input();
    HRESULT Init(HWND hWnd);            //DirectInputの準備
};
```

<Input.cpp>

```
HRESULT Input::Init(HWND hWnd)
{
    //DirectInputオブジェクトの作成
    if (FAILED(DirectInput8Create(GetModuleHandle(NULL), DIRECTINPUT_VERSION,
                                                    IID_IDirectInput8, (VOID**)&pDinput, NULL)))
    {
        return E_FAIL;
    }
    return S_OK;
}
```

④　デバイスオブジェクト（キーボードの準備）

<Input.h>

```
class Input
{
    LPDIRECTINPUT8          pDinput;    //DirectInputオブジェクト
    LPDIRECTINPUTDEVICE8    pKeyDevice; //デバイスオブジェクト（キーボード）

public:
    Input();
```

<Input.cpp>

```
HRESULT Input::Init(HWND hWnd)
{
    //DirectInputオブジェクトの作成
    if (FAILED(DirectInput8Create(GetModuleHandle(NULL), DIRECTINPUT_VERSION,
                                                    IID_IDirectInput8, (VOID**)&pDinput, NULL)))
    {
        return E_FAIL;
    }

    // 「DirectInput デバイス」オブジェクトの作成
    if (FAILED(pDinput->CreateDevice(GUID_SysKeyboard, &pKeyDevice, NULL)))
    {
        return E_FAIL;
    }

    return S_OK;
}
```

<つづき>

```
    // デバイスをキーボードに設定
    if( FAILED(  pKeyDevice->SetDataFormat( &c_dfDIKeyboard ) ) )
    {
        return E_FAIL;
    }
```

<つづき>

```
    // 協調レベルの設定
    if( FAILED( pKeyDevice->SetCooperativeLevel( hWnd, DISCL_NONEXCLUSIVE | DISCL_BACKGROUND )) )
    {
        return E_FAIL;
    }
```

⑤　解放処理

```
Input::~Input()
{
    //キーボードのアクセス権を解放
    if (pKeyDevice)
    {
        pKeyDevice->Unacquire();
    }

    //DirectInput 解放
    [          ](pKeyDevice);
    [          ](pDinput);
}
```

⑥　キーの状態を取得

どのキーが押されているかを記憶する変数と、キーの状態を調べる関数を追加

<Input.h>

```
class Input
{
    LPDIRECTINPUT8          pDinput;            //DirectInput オブジェクト
    LPDIRECTINPUTDEVICE8    pKeyDevice;         //デバイスオブジェクト（キーボード）
    BYTE                    keyState [256];     //各キーの状態
public:
    Input();
    ~Input();
    HRESULT Init(HWND hWnd);            //DirectInput の準備
    HRESULT Update();                   //各入力デバイスの状態を取得
};
```

<Input.cpp>

```
Input::Input()
{
    ZeroMemory(keyState, [          ]);
}
        :
        :
HRESULT Input::Update()
{
    // デバイスのアクセス権を取得する
    HRESULT hr = pKeyDevice->Acquire();

    if ((hr == DI_OK) || (hr == S_FALSE))
    {
        //全てのキーの状態を取得
        pKeyDevice->GetDeviceState(sizeof(keyState), &diks);
        return S_OK;
    }
    return E_FAIL;
}
```

⑦　任意のキーが押されているかチェック

<Input.h>

```
class Input
{
        :
        :
    HRESULT Init(HWND hWnd);            //DirectInput の準備
    HRESULT Update();                   //各入力デバイスの状態を取得
    BOOL    IsKeyPush(DWORD keyCode);   //任意のキーが押されているかチェック
};
```

<Input.cpp>

```
BOOL Input::IsKeyPush(DWORD    keyCode)
{
    if(keyState [keyCode] & 0x80)
    {
        return TRUE;    //押している
    }
    return FALSE;       //押してない
}
```

2. 使ってみる

入力処理はどこでも使う可能性があるので、グローバルにしてしまう。

① オブジェクト作成

<Global.h>

```
//---------------インクルード-----------------------
#include <windows.h>
#include <d3dx9.h>
#include "Input.h"

    :
    :
    :

//Direct3D デバイスオブジェクト
extern LPDIRECT3DDEVICE9    g_pDevice;

//入力処理オブジェクト
extern Input* g_pInput;
```

こう書いてしまうと、Input.h で Global.h をインクルードしてるのでエラーになってしまう（**循環参照**）。

DirectX は Windows プログラムとも Direct3D とも関係ない独立したプログラムなので、

Global.h をインクルードしなくてもほぼ問題ない。

<Input.h>

```
#include "Global.h"
#include <dinput.h>
```

こうすると、解放処理でエラーが出てしまう。

SAFE_RELEASE マクロは Global.h で宣言していたためだ。

しかたないので、SAFE_RELEASE マクロを Input.h でも宣言するようにしよう。

<Input.h>

```
#define SAFE_RELEASE(p)        { if(p != NULL) { (p)->Release(); (p) = NULL; } }
```

最後に extern で宣言したものは、Game.cpp で再宣言を忘れないように。

<Game.cpp>

```
//-----------------グローバル変数-----------------
GAME_SCENE          g_gameScene;    //現在のゲームシーン
LPDIRECT3DDEVICE9   g_pDevice;      //Direct3D デバイスオブジェクト
Input*              g_pInput;       //入力処理オブジェクト
```

②　初期化と開放

Input クラスの Init 関数にはウィンドウハンドルが必要なので、Main.cpp で行うことにする。

<Main.cpp>

```
int WINAPI WinMain(HINSTANCE hCurInst, HINSTANCE hPrevInst, LPSTR lpsCmdLine, int nCmdShow)
{
        :
        :
        :

    //Direct3D の初期化
    game->InitD3d(hWnd);

    //入力処理の初期化
    g_pInput = new Input();
    g_pInput->Init(hWnd);

    //読み込み処理
    if (FAILED(game->Load()))
    {
        return FALSE;
    }
```

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT msg, WPARAM wp, LPARAM lp)
{
    switch (msg)
    {
    //■ウィンドウが消された
    case WM_DESTROY:

        //入力処理開放
        ［　　　　］(g_pInput);
```

③　毎フレームキーの状態を調べる

<Game.cpp>

```
//―――――――――――――――――――――――――――――――
//  更新処理
//―――――――――――――――――――――――――――――――
void Game::Update()
{
    g_pInput->Update();

        :
        :
}
```

入力情報の更新は一時的に失敗することがあるので、失敗したからといってプログラムを終了させる必要はない。

④　画像を動かしてみよう

現状、各オブジェクトの位置は POINT 構造体で管理するようになっている。

今後は小数も使えたり 3 次元も使えた方が良いので、D3DXVECTOR3 型に修正しておこう。

<UnitBase.h>

```
//―――――――――――――――――――――――――――――――
//  各ゲームユニットの親クラス
//―――――――――――――――――――――――――――――――
class UnitBase
{
protected:
    //ユニットの位置
    D3DXVECTOR3     position;

        :
        :

    //現在の位置を取得
    //戻値：現在の位置
    D3DXVECTOR3     GetPos()    { return position; }
};
```

x,y しか使わない分には今までと変わらず使える。

テストとしてタイトルクラスで position の位置に画像が表示されるようにする。

<Title.h>

```
void Title::Render()
{
    SpriteData data;
    data.pos = position;
    sprite.Draw(&data);
}
```

Update 関数を追加して、→キーを押したら右に移動するようにしよう。

<Title.h>

```
class Title : public UnitBase
{
    Sprite sprite;

public:
    Title();
    ~Title();
    HRESULT Load();
    HRESULT Update();
    HRESULT Render();
};
```

<Title.cpp>

```
void Title::Update()
{
    if (g_pInput->IsKeyPush(DIK_RIGHT))
    {
        position.x += 3;
    }
}
```

⑤　警告を消す

やたら警告が出るので、Input.h の先頭に次の 1 行を追加しておく。

<Input.h>

```
#pragma once

#define DIRECTINPUT_VERSION 0x800 //DirectInput のバージョン設定
```

つづく