オブジェクト指向とゲームプログラミング

▶ DirectInput編 － 第４回　デバイスの状態取得 ◀

デバイスの状態取得

入力デバイスのボタンの状態や軸の傾きといった情報の取得は、IDirectInputDevice8::GetDeviceStateメソッドで行います。このメソッドが返す情報は、SetDataFormatメソッドで設定したものです。定義済みのデータ形式を設定した場合は、キーボードは256バイトの配列、マウスはDIMOUSESTATE構造体またはDIMOUSESTATE2構造体の形式で情報が返されます。

このメソッドで取得できる情報は、ボタンは「押されている(押しっぱなし)」「押されていない(離されている)」、軸は前回からの座標の差または傾きの角度です。ボタンや軸が「押された」「離された」という情報は、このメソッドでは取得できませんが、DirectInputが提供するデバイスバッファ(第５回参照)という機能を使えば取得できます。

IDirectInputDevice8::GetDeviceStateメソッド

－説明－

GetDeviceStateメソッドは、デバイスから直接データを取得します。

－書式－

```
HRESULT GetDeviceState(DWORD cbData, LPVOID lpvData);
```

－パラメータ－

１つ目の引数(cbData)は、２つ目の引数(情報を受け取る領域)のバイト数です。

２つ目の引数(lpvData)は、デバイスの状態を受け取る領域のアドレスです。

－戻り値－

成功した場合はDI_OK、それ以外はエラーコードを返します。

SetDataFormatメソッドで定義済みの変数を指定した場合、このメソッドが返す形式は以下のようになります。

定義済み変数	対応デバイス	メソッドが返す情報
c_dfDIKeyboard	キーボード	256バイトの配列
c_dfDIMouse	マウス	DIMOUSESTATE構造体
c_dfDIMouse2	マウス	DIMOUSESTATE2構造体
c_dfDIJoystick	ゲームパッド	DIJOYSTATE構造体
c_dfDIJoystick2	ゲームパッド (フォースフィードバック対応)	DIJOYSTATE2構造体

キーボードの状態取得

GetDeviceStateメソッドがキーボードから取得する情報は、キーが「押されている」と「押されていない」というものです。１回のメソッド呼び出しで256種類のキー状態が取得されます。この情報は256バイトの配列(BYTE配列)に格納されます。

キーボードの状態を取得する場合、GetDeviceStateメソッドの１つ目の引数は256、２つ目の引数はキー状態を格納する配列の先頭アドレスを指定します。

```
// キーボード状態の取得(pDIDKeyboardは初期化済みのキーボードのデバイスオブジェクト)
BYTE   KeyState[256];        // キー状態を格納する配列
pDIDKeyboard->GetDeviceState(256, KeyState);
```

上記のようにGetDeviceStateメソッドを呼び出すと、256バイトの配列(KeyState)にキーの状態が格納されます。配列の要素１つに対して１つのキーが割り当てられています。特定のキーの状態を取り出すには、あらかじめ定義されている「デバイス定数」を添字として使用します。たとえば、Enter(Return)キーは「DIK_RETURN」、'A'キーは「DIK_A」を添字とします。このような「DIK_」で始まる定数をデバイス定数と呼び、すべてのキーに対して割り当てられています(詳しくは、DirectX9のヘルプの目次→DirectInput→DirectInput C/C++ リファレンス→デバイス定数を参照)。

配列に格納された値から「押されている」「押されていない」という状態を調べることができます。配列の１要素は８ビットあり、このうち上位１ビットで判定します。ここが１ならキーが押されていることを表し、０ならキーが押されていないことを表します。プログラムではビット論理積演算子&を使ったマスク処理で上位１ビットを取り出します。たとえば、Enterキーが押されているかどうかを調べるには、以下のようにします。

```
// Enterキーが押されているかを調べる
if((KeyState[DIK_RETURN] & 0x80) != 0)
    // Enterキーは押されている
```

マウスの状態取得

GetDeviceStateメソッドがマウスから取得する情報は、マウスの変化量とボタンの状態です。変化量は、ｘ座標、ｙ座標、ｚ座標(ホイール)が前回の取得位置からどのくらい動いたのかという情報です。これらの情報はDIMOUSESTATE構造体またはDIMOUSESTATE2構造体に格納されます。２つの構造体の違いは取得できるボタン数で、前者は４ボタン、後者は８ボタンまで取得できます。

マウスの状態を取得する場合、GetDeviceStateメソッドの１つ目の引数はsizeof演算子で構造体のサイズを指定し、２つ目の引数は状態を格納する構造体変数のアドレスを指定します。

```
// マウスの状態取得(pDIDMouseは初期化済みのマウスのデバイスオブジェクト)
DIMOUSESTATE   MouseState;      // マウスの状態を格納する構造体変数
pDIDMouse->GetDeviceState(sizeof(MouseState), &MouseState);
```

上記のようにGetDeviceStateメソッドを呼び出すと、DIMOUSESTATE構造体にマウスの状態が格納されます。この構造体のメンバと格納される情報は以下のようになっています。

```
struct DIMOUSESTATE {
    LONG lX;                // ｘ座標の変化量
    LONG lY;                // ｙ座標の変化量
    LONG lZ;                // ｚ座標(ホイール)の変化量
    BYTE rgbButtons[4];     // ボタンの状態(DIMOUSESTATE2構造体では要素数8)
};
```

ボタンはキーボードと同じようにBYTE配列になっており、特定のボタンを指定するにはボタンの番号を添字にします。「押されている」「押されていない」という情報も上位１ビットで判定します。

```
// マウスのボタン０(通常は左ボタン)が押されているかを調べる
if((MouseState.rgbButtons[0] & 0x80) != 0)
    // ボタン０は押されている
```

マウスカーソルの座標はDirectInputでは取得できません。移動量から計算するか、APIのGetCursorPos関数で取得します。ただし、排他占有モードの場合は、カーソルが存在しないので、移動量から計算するしかありません。

```
// マウスカーソルの座標取得(hWndはウィンドウのハンドル)
POINT   MousePos;
GetCursorPos(&MousePos);
ScreenToClient(hWnd, &MousePos);     // スクリーン座標をクライアント座標に変換
```

課　　題

入力デバイスクラスに、デバイス状態を取得する機能を追加しましょう。

(１)デバイス状態の取得は、GetState関数として実装します。この関数は、引数にデバイス状態を格納する領域のアドレスを渡し、戻り値は取得成功はtrue、それ以外はfalseを返すとします。これをふまえると、関数のプロトタイプは以下のようになります。IInputDeviceクラスの適切な場所に追加しましょう。

```
virtual bool GetState(LPVOID pState) = 0;
```

(２)CKeyboardクラスにGetState関数を実装します。以下のプログラムを適切な場所に追加しましょう。

```
virtual bool GetState(LPVOID pState);
```

(３)CKeyboard::GetState関数の足りない部分を補い、適切な場所に追加しましょう。

```
/*****************************************************************************************************/
/*                                          状態取得                                                 */
/*****************************************************************************************************/
bool CKeyboard::GetState(LPVOID pState)
{
    if(m_pDIDKeyboard->??????????????(???, pState) != DI_OK) {
        ::OutputDebugString("*** Error - 状態取得失敗(CKeyboard_GetState)\n");
        ::ZeroMemory(pState, 256);
        return false;
    }

    return true;
}
```

(４)CMouseクラスにGetState関数を実装します。以下のプログラムを適切な場所に追加しましょう。

```
virtual bool GetState(LPVOID pState);
```

(５)CMouse::GetState関数の足りない部分を補い、適切な場所に追加しましょう。

```
/*****************************************************************************************************/
/*                                          状態取得                                                 */
/*****************************************************************************************************/
bool CMouse::GetState(LPVOID pState)
{
    if(m_pDIDMouse->??????????????(sizeof(DIMOUSESTATE), ??????) != DI_OK) {
        ::OutputDebugString("*** Error - 状態取得失敗(CMouse_GetState)\n");
        ::ZeroMemory(pState, sizeof(DIMOUSESTATE));
        return false;
    }

    return true;
}
```

CKeyboard::GetState関数とCMouse::GetState関数には、m_pDIDKeyboardやm_pDIDMouseがNULLかどうかを調べるコードがありません。これが、Null Objectパターンの効果です。

(６)CNullInputDeviceクラスに、「何もしない」GetState関数を実装します。以下のプログラムを適切な場所に追加しましょう。

```
virtual bool GetState(LPVOID pState) { return true; }
```

(７)GetState関数が正しく動作するか確認します。次のような「背景ビットマップ」と「キャラクタビットマップ」を用意しましょう。

背景.BMP

キャラクタ.BMP

(８)カーソルキーの左右が押されたら、それに合わせてキャラクタが左右に動くようにしてみましょう。
CTestSceneクラスを以下のように追加、変更しましょう。

・CTestSceneクラスのメンバに追加

```
ISprite*    m_pBG;          // 背景
ISprite*    m_pChara;       // キャラクター

int         m_CharaX;       // キャラクタｘ座標
int         m_CharaY;       // キャラクタｙ座標
```

・CTestScene::CTestScene関数を以下のように変更

```
CTestScene::CTestScene()
{
    // ここに、機能テストの初期化処理を記述します
    m_pBG    = DXGraphics().CreateSpriteFromFile("Graphics\\BG.bmp",    D3DFMT_UNKNOWN, 0);
    m_pChara = DXGraphics().CreateSpriteFromFile("Graphics\\Chara.bmp", D3DFMT_UNKNOWN,
                                                 D3DCOLOR_XRGB(0, 0, 0));

    m_CharaX = 0;
    m_CharaY = 0;

    FPSTimer().Reset();         // タイマリセット
}
```

・CTestScene::~CTestScene関数を以下のように変更

```
CTestScene::~CTestScene()
{
    // ここに、機能テストの解放処理を記述します
    DXGraphics().ReleaseAllSprites();
}
```

・CTestScene::ActivateProc関数を以下のように変更

```
int CTestScene::ActiveProc()
{
    // ここに、機能テストのメイン処理を記述します
    // 内部処理
    // キーボード状態取得
    BYTE   KeyState[256];
    DIKeyboard()->GetState(KeyState);

    // キャラクタ移動
    if((KeyState[DIK_LEFT ] & 0x80) != 0)
        m_CharaX--;
    if((KeyState[DIK_RIGHT] & 0x80) != 0)
        m_CharaX++;

    // 描画処理
    if(FPSTimer().IsSkip() == true)
        return 0;

    DXGraphics().Clear();

    DXGraphics().BeginScene();
    DXGraphics().BeginSprite(DXGSPR_ALPHA | DXGSPR_SORT_FRBK);

    RECT   src;

    // 背景描画
    ::SetRect(&src, 0, 0, 640, 480);
    m_pBG->Draw(&src, DXVEC3(0.0f, 0.0f, 1.0f), 1.0f);

    // キャラクター描画
```

```
        src.left   = 0;
        src.top    = 0;
        src.right  = 120;
        src.bottom = 120;
        m_pChara->Draw(&src, DXVEC3(m_CharaX, m_CharaY, 0.5f), 1.0f);

        DXGraphics().EndSprite();
        DXGraphics().EndScene();

        return 0;

        // FPS描画
        HDC   hdc = DXGBackBuffer()->GetDC();
        FPSTimer().DrawFPS(hdc, 0, 0);
        DXGBackBuffer()->ReleaseDC();

        // フレーム更新
        DXGraphics().UpdateFrame();

        return 0;
    }
```

(９)キャラクタが上下にも動けるようにプログラムを変更しましょう。

(10)マウスの左ボタンが押されているときはキャラクタを左に、右ボタンが押されているときはキャラクタを右に動かすようにプログラムを変更しましょう。

(11)マウスカーソルの座標にキャラクタを表示するようにしましょう。