オブジェクト指向と ゲームプログラミング

▶ DirectInput編 － 第５回　デバイスバッファ ◀

デバイスバッファ

DirectInputが管理する入力デバイスは、デバイスバッファという領域を持っています。DirectInputは、軸やボタンが「押された」「離された」瞬間(状態が変わった瞬間)に、これらのイベントに関する詳細な情報をデバイスバッファという領域に格納します。
デバイスバッファの情報はDIDEVICEOBJECTDATA構造体に取得することができます。この構造体には、以下のような情報が格納されます。

```
struct DIDEVICEOBJECTDATA {
        DWORD    dwOfs;          // イベント発生源のデバイス定数
        DWORD    dwData;         // デバイスから取得したデータ(イベントの種類)
        DWORD    dwTimeStamp;    // イベントが発生したミリ秒単位のシステム時間
        DWORD    dwSequence;     // イベントの発生番号
        UINT_PTR uAppData;       // アプリケーション定義のアクション値
};
```

たとえば、キーボードで「Aキーが押された」→「SPACEキーが押された」→「SPACEキーが離された」→「Aキーが離された」という４つのイベントが発生した場合、キーボードのデバイスバッファには、以下のように情報が格納されます。

dwOfs :DIK_A	dwOfs :DIK_SPACE	dwOfs :DIK_SPACE	dwOfs :DIK_A
dwData:押した(0x80)	dwData:押した(0x80)	dwData:離した(0x00)	dwData:離した(0x00)

デバイスバッファ

デバイスバッファは、GetDeviceStateメソッドでは調べることができない「押された」「離された」というイベントを判定したり､格闘ゲームのような複雑なコマンド解析などに利用することができます。

デバイスバッファの設定

デバイスバッファは、デフォルトでサイズが０に設定されており、イベントを格納することができません。IDirectInputDevice8::SetPropertyメソッドを呼び出し、適切なサイズに変更します。

IDirectInputDevice8::SetPropertyメソッド

－説明－
SetPropertyメソッドは、デバイスの動作を定義するプロパティを設定します。プロパティには、デバイスバッファのサイズ、軸モード、軸の範囲などがあります。

－書式－
HRESULT SetProperty(REFGUID *rguidProp*, LPCDIPROPHEADER *pdiph*);

－パラメータ－
１つ目の引数(rguidProp)は、設定するプロパティを識別するGUIDです。おもに以下のものを使用します。

DIPROP_BUFFERSIZE	デバイスバッファのサイズを設定
DIPROP_RANGE	軸が報告可能な値の範囲を設定
DIPROP_DEADZONE	軸のデッドゾーンを設定
DIPROP_AXISMODE	軸モードを設定

２つ目の引数(pdiph)は、プロパティの設定情報を格納する構造体に必ず含まれるDIPROPHEADER構造体のアドレスです。

－戻り値－
成功した場合はDI_OKまたはDI_PROPNOEFFECT、それ以外はエラーコードを返します。

デバイスバッファの設定を行う場合は、DIPROPDWORD構造体にバッファの設定情報を格納します。この構造体は、DIPROPHEADER構造体型のdiphメンバと、DWORD型のdwDataメンバで構成されています。dwDataはデバイスバッファの数を設定し、diphメンバは以下のように初期化します。

メンバ	設定する値
dwSize	構造体全体のサイズ。sizeof(DIPROPDWORD)
dwHeaderSize	DIPROPHEADER構造体のサイズ。sizeof(DIPROPHEADER)
dwObj	dwHowメンバがDIPH_DEVICEのときは0
dwHow	デバイスバッファの設定のときはDIPH_DEVICE

```
// バッファサイズの設定
LPDIRECTINPUTDEVICE8   pDIDevice;   // デバイスオブジェクト(初期化済みとする)
DIPROPDWORD   dipd;
ZeroMemory(&dipd, sizeof(dipd));
dipd.diph.dwSize       = sizeof(dipd);
dipd.diph.dwHeaderSize = sizeof(dipd.diph);
dipd.diph.dwObj        = 0;
dipd.diph.dwHow        = DIPH_DEVICE;
dipd.dwData            = 16;    // バッファの数
pDIDevice->SetProperty(DIPROP_BUFFERSIZE, &dipd.diph);
```

デバイスバッファの取得

デバイスバッファに格納された情報の取得は、IDirectInputDevice8::GetDeviceDataメソッドで行います。

IDirectInputDevice8::GetDeviceDataメソッド

－説明－

GetDeviceDataメソッドは、デバイスバッファからデータを取得します。

－書式－

```
HRESULT GetDeviceData(DWORD cbObjectData, LPDIDEVICEOBJECTDATA rgdod, LPDWORD pdwInOut,
                      DWORD dwFlags);
```

－パラメータ－

1つ目の引数(cbObjectData)は、情報を格納するDIDEVICEOBJECTDATA構造体のサイズ(バイト数)です。sizeof演算子で求めます。

2つ目の引数(rgdod)は、取得した情報を格納する領域です。DIDEVICEOBJECTDATA構造体配列の先頭アドレスを指定します。NULLを指定すると情報を格納しません。

3つ目の引数(pdwInOut)は、DWORD型の変数のアドレスです。この変数には、2つ目の引数で指定した配列の要素数を代入しておきます。メソッド呼び出し後、実際に取得した要素数に書き換えられます。

4つ目の引数(dwFlags)は、データを取得する方法を示すフラグです。0を指定するとデバイスバッファの情報が削除されます。「DIGDD_PEEK」を指定するとバッファから情報を削除しません。これ以降のGetDeviceDataメソッドの呼び出しで、同じデータを読み取ることができます。

－戻り値－

成功した場合はDI_OKまたはDI_BUFFEROVERFLOW、それ以外はエラーコードを返します。

```
// バッファ情報の取得
DIDEVICEOBJECTDATA     DIData[16];      // バッファ情報を格納する領域
DWORD                  Items = 16;      // 配列の個数
pDIDevice->GetDeviceData(sizeof(DIDEVICEOBJECTDATA), DIData, &Items, 0);
```

上記のようにすると、DIDEVICEOBJECTDATA構造体のdwOfsメンバにイベントが発生したデバイス定数が格納され、dwDataメンバにイベントの種類が格納されます。dwDataメンバはDWORD型で32ビットありますが、「押された」「離された」というイベントの場合は、下位から数えて8ビット目に格納されます。このビットが1なら「押された」、0なら「離された」ことを表します。たとえば、Enterキーで「押された」イベントが発生したかを判定するには、以下のようになります。

```
DIDEVICEOBJECTDATA     BufData[16];     // バッファ情報を格納する領域
DWORD                  Items = 16;      // 配列の個数
```

```
// バッファ情報取得
pDIDevice->GetDeviceData(sizeof(DIDEVICEOBJECTDATA), BufData, &Items, 0);

// バッファ情報の解析
for(DWORD i = 0; i < Items; i++) {
    if(BufData[i].dwOfs == DIK_RETURN && (BufData[i].dwData & 0x80) != 0)
        // 「Enterキーが押された」イベントが発生
}
```

「離された」イベントを判定する場合は、以下のようになります。

```
if(BufData[i].dwOfs == DIK_RETURN && (BufData[i].dwData & 0x80) == 0)
    // Enterキーが離されたイベントが発生
```

デバイスバッファに格納された情報は、GetDeviceDataメソッドが呼び出されるまで消去されることはありません。デバイスバッファのサイズを超える情報を格納することはできないので、バッファのサイズは十分な大きさを設定し、一定のタイミングでバッファを読み取るか、消去する必要があります。

デバイスバッファが溢れてしまうことをオーバーフローと呼びます。オーバーフローが起こると、一部のデータが失われてしまい、正確な情報が取得できなくなります。

バッファ情報の消去

バッファ内に格納されている情報をすべて消去するには、GetDeviceDataメソッドの１つ目の引数は取得のときと同じにし、２つ目の引数はNULL、３つ目の引数は「INFINITE」を代入したDWORD型の変数のアドレス、４つ目の引数は０を指定します。

```
// バッファ情報の消去
DWORD   Items = INFINITE;
pDIDevice->GetDeviceData(sizeof(DIDEVICEOBJECTDATA), NULL, &Items, 0);
```

課　題

入力デバイスクラスに、デバイスバッファから情報を取得する機能を追加しましょう。

(１)バッファのサイズを定数で定義します。InputDevice.hppの適切な場所に、以下のプログラムを追加しましょう。

```
/*****************************************************************************************************/
/*                                          定　数                                                   */
/*****************************************************************************************************/
enum {
    DIDEV_BUFSIZE = 16      // バッファサイズ
};
```

(２)キーボードのデバイスバッファのサイズを設定します。CDInput8::CreateKeyboard関数の適切な場所に追加に、以下のプログラムの足りない部分を補って追加しましょう。

```
// バッファサイズ設定
DIPROPDWORD   dipd;
::ZeroMemory(&dipd, sizeof(dipd));
dipd.diph.dwSize       = sizeof(dipd);
dipd.diph.dwHeaderSize = sizeof(dipd.diph);
dipd.diph.dwObj        = ?;
dipd.diph.dwHow        = ??????????;
dipd.dwData            = DIDEV_BUFSIZE;
if(pDIDevice->???????????(????????????????????, &dipd.diph) != DI_OK) {
    ::OutputDebugString("*** Error - バッファサイズ設定失敗(CDInput8_CreateKeyboard)\n");
    pDIDevice->Release();
    return false;
}
```

(３)マウスのデバイスバッファのサイズを設定します。CDInput8::CreateMouse関数の適切な場所に、(２)のプログラムを参考にバッファのサイズを設定するプログラムを追加しましょう。

(４)デバイスバッファの取得は、GetBuffer関数として実装します。この関数は、引数にデバイスバッファを格納するDIDEVICEOBJECTDATA構造体配列と配列の要素数を渡し、戻り値はバッファの取得数を返すとします。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 DWORD GetBuffer(DIDEVICEOBJECTDATA outBufData[],
                        const DWORD inElements = DIDEV_BUFSIZE) = 0;
```

「inElements = DIDEV_BUFSIZE」により、引数inElementsの指定が省略可能になります。省略した場合、DIDEV_BUFSIZEを指定したものとみなされます。このように、値の指定を省略したときにデフォルト値が設定される引数をデフォルト引数と呼びます。

(５)CKeyboardクラスにGetBuffer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 DWORD GetBuffer(DIDEVICEOBJECTDATA outBufData[], const DWORD inElements);
```

(６)CKeyboard::GetBuffer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。

```
/*****************************************************************************************************/
/*                                          バッファ取得                                             */
/*****************************************************************************************************/
DWORD CKeyboard::GetBuffer(DIDEVICEOBJECTDATA outBufData[], const DWORD inElements)
{
    DWORD   Items = inElements;
    if(FAILED(m_pDIDKeyboard->※ここは各自考えましょう)) {
        ::OutputDebugString("*** Error - バッファ取得失敗(CKeyboard_GetBuffer)\n");
        ::ZeroMemory(outBufData, sizeof(DIDEVICEOBJECTDATA) * inElements);
        return 0;
    }

    return Items;
}
```

(７)CMouseクラスにGetBuffer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 DWORD GetBuffer(DIDEVICEOBJECTDATA outBufData[], const DWORD inElements);
```

(８)CMouse::GetBuffer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。

```
/*****************************************************************************************************/
/*                                          バッファ取得                                             */
/*****************************************************************************************************/
DWORD CMouse::GetBuffer(DIDEVICEOBJECTDATA outBufData[], const DWORD inElements)
{
    DWORD   Items = inElements;
    if(FAILED(m_pDIDMouse->※ここは各自考えましょう)) {
        ::OutputDebugString("*** Error - バッファ取得失敗(CMouse_GetBuffer)\n");
        ::ZeroMemory(outBufData, sizeof(DIDEVICEOBJECTDATA) * inElements);
        return 0;
    }

    return Items;
}
```

(９)CNullInputDeviceクラスに、「何もしない」GetBuffer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 DWORD GetBuffer(DIDEVICEOBJECTDATA outBufData[], const DWORD inElements) { return 0; }
```

(10)デバイスバッファの情報を消去するFlushBuffer関数を作成します。IInputDeviceクラスの適切な場所に、以下のプログラムを追加しましょう。

```
virtual void FlushBuffer() = 0;
```

(11)CKeyboardクラスにFlushBuffer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 void FlushBuffer();
```

(12)CKeyboard::FlushBuffer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。

```
/*****************************************************************************************************/
/*                                          バッファ消去                                             */
/*****************************************************************************************************/
void CKeyboard::FlushBuffer()
{
    DWORD   Items = INFINITE;
    m_pDIDKeyboard->※ここは各自考えましょう;
}
```

(13)CMouseクラスにFlushBuffer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 void FlushBuffer();
```

(14)CMouse::FlushBuffer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。

```
/*****************************************************************************************************/
/*                                          バッファ消去                                             */
/*****************************************************************************************************/
void CMouse::FlushBuffer()
{
    DWORD   Items = INFINITE;
    m_pDIDMouse->※ここは各自考えましょう;
}
```

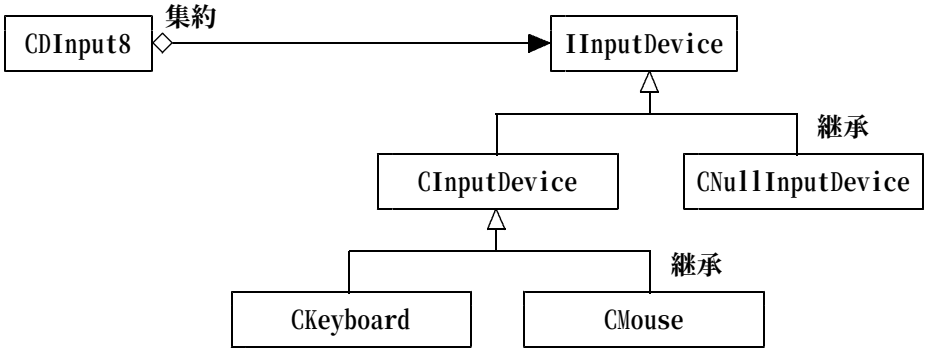
(15)CNullInputDeviceクラスに、「何もしない」FlushBuffer関数を実装します。以下のプログラムを適切な場所に追加しましょう。

```
virtual void FlushBuffer() {}
```

(16)CKeyboardクラスとCMouseクラスの共通する部分をまとめます。

CKeyboardクラスとCMouseクラスには、共通する部分が多くあります。たとえば、コンストラクタ、デストラクタ、IsNull関数、GetBuffer関数、FlushBuffer関数といった関数では、まったく同じ処理をしています。そしてメンバ変数の型もまったく同じです。唯一の違いはGetState関数内の処理ですが、処理の流れはまったく同じで、IDirectInputDevice8::GetDeviceStateメソッドとZeroMemory関数に与えている引数が異なっているだけです。この関数も、異なる部分を引数として渡してもらうようにすれば、共通する部分としてまとめることができます。

以上をふまえ、２つのクラスに共通する部分をまとめたCInputDeviceクラスを作成し、これを継承してCKeyboardクラスとCMouseクラスを作成するように変更します。クラス相関図は以下のようになります。



この設計を用いると、CInputDeviceクラス、CKeyboardクラス、CMouseクラスは以下のようになります。それぞれのクラス宣言を変更しましょう。

```
/*****************************************************************************************************/
/*                                        インプットデバイスクラス定義                                 */
/*****************************************************************************************************/
class CInputDevice : public IInputDevice {
public:
    CInputDevice(IDirectInputDevice8* pDIDevice);
    virtual ~CInputDevice();

    virtual bool IsNull() const { return ※ここは各自考えましょう; }

    virtual DWORD GetBuffer(DIDEVICEOBJECTDATA outDIData[], const DWORD inElements);
    virtual void FlushBuffer();

protected:
    virtual bool GetState(DWORD inSize, LPVOID pState);

    IDirectInputDevice8*   m_pDIDevice;      // デバイスオブジェクトへのインタフェース
};

/*****************************************************************************************************/
/*                                        キーボードクラス定義                                         */
/*****************************************************************************************************/
class CKeyboard : public CInputDevice {
public:
    CKeyboard(IDirectInputDevice8* pDIDKeyboard) : CInputDevice(pDIDKeyboard) {}
    virtual ~CKeyboard() {}

    virtual bool GetState(LPVOID pState) { return CInputDevice::GetState(256, pState); }
};

/*****************************************************************************************************/
/*                                        マウスクラス定義                                             */
/*****************************************************************************************************/
class CMouse : public CInputDevice {
public:
    CMouse(IDirectInputDevice8* pDIDMouse) : CInputDevice(pDIDMouse) {}
    virtual ~CMouse() {}

    virtual bool GetState(LPVOID pState) { return CInputDevice::GetState(sizeof(DIMOUSESTATE), pState); }
};
```

InputDevice.cppは以下のようになります。

－InputDevice.cpp－

```
/*
=====================================================================================================
                                オブジェクト指向ゲームプログラミング
     Programmed by Hibikino software. Copyright (c) 2005 Hibikino software. All rights reserved.
=====================================================================================================

  【対象ＯＳ】
     Microsoft Windows2000/XP

  【コンパイラ】
     Microsoft Visual C++ 2005

  【プログラム】
     InputDevice.cpp
                入力デバイスクラス

  【履歴】
     * Version    1.00       2005/03/dd  hh:mm:ss

=====================================================================================================
*/

/*****************************************************************************************************/
/*                                        インクルードファイル                                         */
/*****************************************************************************************************/
#include "InputDevice.hpp"
```

```
#include <cassert>

/*****************************************************************************************************/
/*                                          コンストラクタ                                           */
/*****************************************************************************************************/
CInputDevice::CInputDevice(IDirectInputDevice8* pDIDevice)
{
    assert(pDIDevice != NULL);
    m_pDIDevice = pDIDevice;
    m_pDIDevice->AddRef();      // 参照カウンタインクリメントｓ
}

/*****************************************************************************************************/
/*                                          デストラクタ                                             */
/*****************************************************************************************************/
CInputDevice::~CInputDevice()
{
    m_pDIDevice->※ここは各自考えましょう;
    m_pDIDevice->Release();
}

/*****************************************************************************************************/
/*                                          状態取得                                                 */
/*****************************************************************************************************/
bool CInputDevice::GetState(const DWORD inSize, LPVOID pState)
{
    if(m_pDIDevice->※ここは各自考えましょう(inSize, pState) != DI_OK) {
        ::OutputDebugString("*** Error - 状態取得失敗(CInputDevice_GetState)\n");
        ::ZeroMemory(pState, inSize);
        return false;
    }

    return true;
}

/*****************************************************************************************************/
/*                                          バッファ取得                                             */
/*****************************************************************************************************/
DWORD CInputDevice::GetBuffer(DIDEVICEOBJECTDATA outBufData[], const DWORD inElements)
{
    DWORD   Items = inElements;
    if(FAILED(m_pDIDevice->※ここは各自考えましょう)) {
        ::OutputDebugString("*** Error - バッファ取得失敗(CInputDevice_GetBuffer)\n");
        ::ZeroMemory(outBufData, sizeof(DIDEVICEOBJECTDATA) * inElements);
        return 0;
    }

    return Items;
}

/*****************************************************************************************************/
/*                                          バッファ消去                                             */
/*****************************************************************************************************/
void CInputDevice::FlushBuffer()
{
    DWORD   Items = INFINITE;
    m_pDIDevice->※ここは各自考えましょう;
}
```

このように、共通点を基底クラスにまとめ、派生クラスでは異なる部分だけを作成するような設計をTemplate Methodパターンと呼びます。

(17)デバイスバッファが正しく動作するか確認します。CTestSceneクラスを以下のように追加、変更しましょう。

・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
    // キーボードバッファ取得し、キャラクタを移動する
    DIDEVICEOBJECTDATA   BufData[DIDEV_BUFSIZE];
```

```
        for(UINT i = 0; i < DIKeyboard()->GetBuffer(BufData); i++) {
            if((BufData[i].dwData & 0x80) != 0) {
                // 「押された」とき
                if(BufData[i].dwOfs == DIK_LEFT)
                    m_CharaX--;         // 「←」キーが押された場合、左に移動
            } else {
                // 「離された」とき
                if(BufData[i].dwOfs == DIK_RIGHT)
                    m_CharaX++;         // 「→」キーが離された場合、右に移動
            }
        }

        // 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;
    }
```

(18)上記のプログラムを変更し、マウスの左ボタンが押されたらキャラクタが左に移動、マウスの右ボタンが離されたら右に移動するようにしましょう。