オブジェクト指向とゲームプログラミング

▶ DirectInput編 ― 第９回　フォースフィードバックと振動 ◀

フォースフィードバック

ゲームパッドには、フォースフィードバック機能が搭載されているものがあります。フォースフィードバックとは、ゲームパッドに内蔵されているモーターを制御し、軸に抵抗力や反発力などの力(フォース)をかける(フィードバックする)処理のことです。代表的なのがハンドル型コントローラです。レースゲームで車がコーナーを曲がったとき、曲がる方向と逆方向の力をハンドルにかけることにより、コーナーの外側に向かって働くＧを表現しています。
フォースフィードバック対応のゲームパッドなら、このような効果(エフェクト)をプログラムで作り出すことができます。

フォースフィードバック対応デバイスの初期化

フォースフィードバック対応のゲームパッドの初期化方法は、通常のゲームパッドとまったく同じですが、協調レベルを排他占有(DISCL_EXCLUSIVE)にする必要があります。また、デバイスの列挙に使用するEnumDevices関数の４つ目の引数に「DIEDFL_FORCEFEEDBACK」フラグを追加すると、フォースフィードバック対応デバイスのみ列挙されるようになります。

```
// フォースフィードバック対応デバイスの列挙
LPDIRECTINPUT8   pDInput;   // DirectInput8オブジェクト(初期化済みとする)
pDInput->EnumDevices(DI8DEVCLASS_GAMECTRL, (LPDIENUMDEVICESCALLBACK)DIEnumGamePadProc,
                     NULL, DIEDFL_FORCEFEEDBACK | DIEDFL_ATTACHEDONLY);
```

エフェクトの生成

エフェクトを使用するには、エフェクトを管理するDirectInputEffectオブジェクトを生成する必要があります。このオブジェクトは、IDirectInputDevice8::CreateEffectメソッドで生成します。引数に、エフェクトを定義したDIEFFECT構造体が必要になります。

```
// エフェクト周期設定
DIPERIODIC   diprd;
ZeroMemory(&diprd, sizeof(diprd));
diprd.dwMagnitude = 10000;                          // エフェクトの強さ(0～10,000)
diprd.lOffset     = 0;
diprd.dwPhase     = 0;
diprd.dwPeriod    = (DWORD)(0.5 * DI_SECONDS);      // エフェクトの間隔

// 振動エフェクト設定
DWORD      Axes[]      = {DIJOFS_X, DIJOFS_Y};      // エフェクト軸
LONG       Direction[] = {0, 0};                    // エフェクト方向

DIEFFECT   Effect;
ZeroMemory(&Effect, sizeof(Effect));
Effect.dwSize                  = sizeof(Effect);
Effect.dwFlags                 = DIEFF_POLAR | DIEFF_OBJECTOFFSETS;
Effect.dwDuration              = (DWORD)(0.5 * DI_SECONDS);     // エフェクト継続時間
Effect.dwSamplePeriod          = 0;
Effect.dwGain                  = DI_FFNOMINALMAX;
Effect.dwTriggerButton         = DIEB_NOTRIGGER;
Effect.dwTriggerRepeatInterval = 0;
Effect.cAxes                   = sizeof(Axes) / sizeof(Axes[0]);    // 配列のサイズ
Effect.rgdwAxes                = Axes;
Effect.rglDirection            = Direction;
Effect.lpEnvelope              = NULL;              // エンベロープ構造体
Effect.cbTypeSpecificParams    = sizeof(diprd);     // エフェクト周期構造体のサイズ
Effect.lpvTypeSpecificParams   = &diprd;            // エフェクト周期構造体
```

```
// エフェクト生成(pDIDGamePadはフォースフィードバック対応の初期化済みデバイスオブジェクト)
IDirectInputEffect*   pDIEffect;        // エフェクトオブジェクトへのインタフェース
pDIDGamePad->CreateEffect(GUID_Square, &Effect, &pDIEffect, NULL);
```

以上のようにエフェクトオブジェクトを生成すると、CreateEffectメソッドを呼び出したデバイスオブジェクトにエフェクトがダウンロードされ、いつでも再生できる状態になります。ひとつのデバイスに対し、複数のエフェクトオブジェクトを生成し、同時にエフェクトを再生することもできます。

エフェクトの解放

エフェクトオブジェクトの解放は、ほかのオブジェクトと同じようにReleaseメソッドで行いますが、デバイスオブジェクトにダウンロードされているので、Unloadメソッドでアンロードしてから解放します。

```
// エフェクトオブジェクトの解放
pDIEffect->Unload();        // アンロード
pDIEffect->Release();       // 解放
```

エフェクトの再生と停止

エフェクトの再生は、IDirectInputEffect::Startメソッドで行います。１つ目の引数は再生回数です。「INFINITE」を指定すると停止するまで再生します。２つ目の引数は再生フラグで通常は「0」または「DIES_SOLO」(ダウンロードされているほかのエフェクトをすべて停止してから再生する)を指定します。

```
// エフェクト再生
pDIEffect->Start(1, 0);
```

エフェクトの停止は、IDirectInputEffect::Stopメソッドで行います。

```
// エフェクト停止
pDIEffect->Stop();
```

IDirectInputEffect::GetEffectStatusメソッドを呼び出すと、エフェクトの状態を取得することができます。エフェクトが再生中かどうかを調べるときに使います。

```
DWORD   EffectState;
pDIEffect->GetEffectStatus(&EffectState);
if((EffectState & DIEGES_PLAYING) != 0)
    // エフェクトは再生中である
```

課　　題

入力デバイスクラスにフォースフィードバックを用いた振動機能を追加し、仮想世界で起きた振動をプレイヤーに直接伝えることができるようにしましょう。

(１)エフェクトをカプセル化したクラスを作成します。

DirectInputEffectオブジェクトは、IDirectInputEffectインタフェースをとおして操作を行います。このインタフェースは、エフェクトを必要としない場合や、生成に失敗した場合、NULLを保持します。インタフェースがNULLかどうかをいちいち調べると、コードが煩雑になってしまうので、エフェクトクラスもNull Objectパターンを適用します。

エフェクトクラスのヘッダファイル(InputEffect.hpp)を以下のように作成しましょう。

－InputEffect.hpp－

```
/*
==========================================================================================
                        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】
    InputEffect.hpp
                入力デバイスエフェクトクラスヘッダ

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

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

#pragma once

/*****************************************************************************************/
/*                                        バージョン宣言                                 */
/*****************************************************************************************/
#define DIRECTINPUT_VERSION 0x0800

/*****************************************************************************************/
/*                                      インクルードファイル                             */
/*****************************************************************************************/
#include <dinput.h>

/*****************************************************************************************/
/*                                入力デバイスエフェクトインタフェース定義                 */
/*****************************************************************************************/
class IInputEffect {
public:
    virtual ~IInputEffect() {}

    virtual bool IsNull() const = 0;

    virtual bool Star(const DWORD inIterations) = 0;
    virtual void Stop() = 0;
    virtual bool IsPlaying() = 0;
};

/*****************************************************************************************/
/*                                  入力デバイスエフェクトクラス定義                       */
/*****************************************************************************************/
class CInputEffect : public IInputEffect {
public:
    CInputEffect(IDirectInputEffect* pDIEffect);
    virtual ~CInputEffect();

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

    virtual bool Star(const DWORD inIterations);
    virtual void Stop() { m_pDIEffect->????(); }
    virtual bool IsPlaying();

private:
    IDirectInputEffect*   m_pDIEffect;
};

/*****************************************************************************************/
/*                                NULL入力デバイスエフェクトクラス定義                     */
/*****************************************************************************************/
class CNullInputEffect : public IInputEffect {
public:
    CNullInputEffect() {}
    virtual ~CNullInputEffect() {}

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

    virtual bool Star(const DWORD inIterations) { return true; }
    virtual void Stop() {}
    virtual bool IsPlaying() { return false; }
};
```

(３)エフェクトクラスのソースファイル(InputEffect.cpp)を以下のように作成しましょう。

－InputEffect.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】
    InputEffect.cpp
                入力デバイスエフェクトクラス

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

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

#pragma once

/*************************************************************************************************/
/*                                        インクルードファイル                                   */
/*************************************************************************************************/
#include "InputEffect.hpp"
#include <cassert>

/*************************************************************************************************/
/*                                        コンストラクタ                                         */
/*************************************************************************************************/
CInputEffect::CInputEffect(IDirectInputEffect* pDIEffect)
{
    assert(pDIEffect != NULL);
    m_pDIEffect = pDIEffect;
    m_pDIEffect->AddRef();          // 参照カウンタインクリメント
}

/*************************************************************************************************/
/*                                        デストラクタ                                           */
/*************************************************************************************************/
CInputEffect::~CInputEffect()
{
    m_pDIEffect->??????();
    m_pDIEffect->Release();
}

/*************************************************************************************************/
/*                                        エフェクト開始                                         */
/*************************************************************************************************/
bool CInputEffect::Star(const DWORD inIterations)
{
    if(m_pDIEffect->?????(inIterations, DIES_SOLO) != DI_OK) {
        ::OutputDebugString("*** Error - エフェクト開始失敗(CInputEffect_Star)\n");
        return false;
    }

    return true;
}

/*************************************************************************************************/
/*                                      エフェクト再生チェック                                   */
/*************************************************************************************************/
bool CInputEffect::IsPlaying()
{
    DWORD   State;
    m_pDIEffect->GetEffectStatus(&State);
    if((State & ?????????????) == 0)
        return false;

    return true;
}
```

IInputEffectクラスは、CInputEffectクラスとCNullInputEffectクラスのインタフェースとして使用したいので、それらのクラスの基底クラスにし、publicな関数は基本的に純粋仮想関数として宣言します。入力デバイスクラスがDirectInputEffectオブジェクトを生成しようとしたとき、その成否によってCInputEffectオブジェクトかCNullInputEffectオブジェクトを生成(するように設計)します。

CInputEffectクラスがDirectInputEffectオブジェクトをカプセル化したクラスです。CNullInputEffectクラスがそのNull Objectです。今後、CInputEffectクラスには、IInputEffectクラスのpublicな関数をすべて「何もしない処理」として実装します。

(４)CInputEffectクラスのメンバは、以下のとおりです。

CInputEffect　コンストラクタ

CInputEffectオブジェクトを構築します。

試CInputEffect(IDirectInputEffect* pDIEffect);

pDIEffect　DirectInputEffectオブジェクトのインタフェース

~CInputEffect　デストラクタ

CInputEffectオブジェクトを解放します。

IsNull　判定

インスタンスがNULLオブジェクト(CNullInputEffectオブジェクト)かどうかを調べます。

試bool IsNull() const;

Return　Nullオブジェクト：true　それ以外：false

Start　制御

エフェクトを開始します。

試bool Star(const DWORD inIterations);

Return　成功：true　それ以外：false

inInterations　連続してエフェクトを再生する回数

Stop　制御

エフェクトを停止します。

IsPlaying　判定

エフェクトが再生されているかどうかを調べます。

試bool IsPlaying();

Return　再生中：true　それ以外：false

(５)エフェクトクラスを入力デバイスクラスに集約します。

エフェクトクラスは、入力デバイスクラスの機能を追加するためのもので、両者の関係は「持っている」関係、すなわち集約になります。以下のメンバをCInputDevcieクラスの適切な場所に追加しましょう。

```
IInputEffect*   m_pEffect;      // エフェクト
```

(６)CInputDeviceクラスのコンストラクタで、m_pEffectメンバが初期化されるようにします。エフェクトが設定されるまで、「何もしない」エフェクトを設定しておきます。
以下の初期化子をCInputDeviceクラスのコンストラクタに追加しましょう。

```
m_pEffect(new C?????????????())
```

(７)CInputDeviceオブジェクトの解放時に、エフェクトオブジェクトが解放されるようにします。以下のプログラムを適切な場所に追加しましょう。

```
delete m_pEffect;      // エフェクト解放
```

(８)CNullInputDeviceクラスは、「何もしない」クラスなので、「何もしない」エフェクトを集約します。次のメンバを適切な場所に追加しましょう。

```
CNullInputEffect   m_Effect;    // 何もしないエフェクト
```

(９)入力デバイスクラスに、エフェクトへのポインタを取得する関数GetEffectを追加します。次の「追加１～３」を適切な場所に追加しましょう。

－追加１－
```
virtual IInputEffect* GetEffect() = 0;
```

－追加２－
```
virtual IInputEffect* GetEffect() { return m_pEffect; }
```

－追加３－
```
virtual IInputEffect* GetEffect() { return &m_Effect; }
```

(10)エフェクトを解放してNULLオブジェクトを設定する以下のReleaseEffect関数をCInputDeviceクラスの適切な場所に追加しましょう。
```
/*****************************************************************************************************/
/*                                          エフェクト解放                                           */
/*****************************************************************************************************/
void CInputDevice::ReleaseEffect()
{
    // pDeviceがNULLオブジェクトの場合は解放しない
    if(m_pEffect->IsNull())
        return;

    // エフェクトを解放し、NULLオブジェクトを設定
    delete m_pEffect;
    m_pEffect = new CNullInputEffect();
}
```

IInputDeviceクラスに純粋仮想関数と、CNullInputDeviceクラスに「何もしない」関数を追加することも忘れずに行いましょう。

(11)振動エフェクトを生成する以下のCreateMagnitudeEffect関数の足りない部分を補い、CInputDeviceクラスの適切な場所に追加しましょう。
```
/*****************************************************************************************************/
/*                                        振動エフェクト生成                                         */
/*****************************************************************************************************/
bool CInputDevice::CreateMagnitudeEffect(const DWORD inMagnitude, const double inPeriod)
{
    // エフェクト周期設定
    DIPERIODIC   diprd;
    ::ZeroMemory(&diprd, sizeof(diprd));
    diprd.dwMagnitude = inMagnitude;
    diprd.lOffset     = 0;
    diprd.dwPhase     = 0;
    diprd.dwPeriod    = (DWORD)(inPeriod * DI_SECONDS);
/*
    // エンベロープ設定
    DIENVELOPE   dienv;
    ::ZeroMemory(&dienv, sizeof(dienv));
    dienv.dwSize        = sizeof(dienv);
    dienv.dwAttackLevel = 0;
    dienv.dwAttackTime  = (DWORD)(2.5 * DISECONDS);
    dienv.dwFadeLevel   = 0;
    dienv.dwFadeTime    = (DWORD)(2.5 * DISECONDS);
*/
    // エフェクト設定
    DWORD       Axes[]     = {DIJOFS_X, DIJOFS_Y};       // エフェクト軸
    LONG        Direction[] = {0, 0};                    // エフェクト方向

    DIEFFECT   Effect;
    ::ZeroMemory(&Effect, sizeof(Effect));
    Effect.dwSize                  = sizeof(Effect);
    Effect.dwFlags                 = DIEFF_POLAR | DIEFF_OBJECTOFFSETS;
    Effect.dwDuration              = (DWORD)(inPeriod * DI_SECONDS);
    Effect.dwSamplePeriod          = 0;
    Effect.dwGain                  = DI_FFNOMINALMAX;
    Effect.dwTriggerButton         = DIEB_NOTRIGGER;
```

```
    Effect.dwTriggerRepeatInterval = 0;
    Effect.cAxes                   = sizeof(Axes) / sizeof(Axes[0]);
    Effect.rgdwAxes                = Axes;
    Effect.rglDirection            = Direction;
    Effect.lpEnvelope              = NULL;  // &dienv;
    Effect.cbTypeSpecificParams    = sizeof(diprd);
    Effect.lpvTypeSpecificParams   = &diprd;

    // エフェクト生成
    IDirectInputEffect*   pDIEffect;
    if(m_pDIDevice->CreateEffect(GUID_Square, &Effect, &pDIEffect, NULL) != DI_OK) {
        ::OutputDebugString("*** Error - エフェクト生成失敗(CInputDevice_CreateMagnitudeEffect)\n");
        ReleaseEffect();
        return false;
    }

    // エフェクトオブジェクト生成
    delete m_pEffect;
    m_pEffect = new CInputEffect(pDIEffect);
    pDIEffect->Release();

    return true;
}
```

CInputDevice::CreateMagnitudeEffect関数の仕様は、以下のとおりです。

CreateMagnitudeEffect　生成

振動エフェクトを生成し、それを管理するエフェクトオブジェクトを生成します。

試bool CreateMagnitudeEffect(const DWORD inMagnitude, const double inPeriod);

Return	成功：true　それ以外：false
inMagnitude	振動の強さ
inPeriod	エフェクトの間隔

IInputDeviceクラスに純粋仮想関数と、CNullInputDeviceクラスに「何もしない」関数を追加することも忘れずに行いましょう。

(12)エフェクトが正しく動作するか確認します。以下のようにプログラムを追加、変更しましょう。

・CTestScene::CTestScene関数に追加

```
// エフェクト生成
DIGamePad(0)->CreateMagnitudeEffect(10000, 0.5);
```

・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
    DIJOYSTATE   State;
    DIGamePad(0)->GetState(&State);
    if((State.rgbButtons[0] & 0x80) != 0 && DIGamePad(0)->GetEffect()->IsPlaying() == false)
        DIGamePad(0)->GetEffect()->Star(1);      // エフェクト開始
    if((State.rgbButtons[1] & 0x80) != 0)
        DIGamePad(0)->GetEffect()->Stop();       // エフェクト停止

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

    return 0;
}
```

応用問題　複数のエフェクトが掛けられるようにプログラムを改良しましょう。