オブジェクト指向と
ゲームプログラミング

▶ コンポーネント編 － 第８回　入力デバイス ◀

キーボード

キーボードのキーに関するイベントは、WM_KEYDOWNやWM_KEYUPといったウィンドウメッセージを利用することで取得することができますが、以下のように、APIにもキーボードに関するものいくつか用意されています。これらの関数を用いることで、メッセージを利用しなくてもキーの状態を取得することができます。

GetAsyncKeyState	キーの状態を取得します
GetKeyboardState	すべての仮想キーの状態を取得します
GetKeyState	仮想キーの状態を取得します
keybd_event	キーストロークを合成します
MapVirtualKey	仮想キーコードまたは走査コードを変換します
SetFocus	キーボードフォーカスをウィンドウに設定します
ToAscii	仮想キーコードをWindows文字に変換します
VkKeyScan	文字を仮想キーコードに変換します
VkKeyScanEx	文字を仮想キーコードとシフト状態に変換します

これらの関数のうち、よく使われるのがGetAsyncKeyState関数です。この関数は、指定されたキーボードキーおよびマウスのボタンの状態を返します。戻り値の最上位ビットが１なら、そのキーが「押されている」、０なら「押されていない」という状態を表します。

```
// ←キーが押されているか調べる
if((GetAsyncKeyState(VK_LEFT) & 0x8000) != 0)
    ←は押されている

// Zキーが押されているか調べる
if((GetAsyncKeyState('Z') & 0x8000) != 0)
    Zキーは押されている
```

マウス

マウスに関するイベントは、WM_MOUSEMOVEやWM_LBUTTONDOWN, WM_LBUTTONUPといったウィンドウメッセージを利用することで取得することができますが、以下のように、APIにもマウスに関するものいくつか用意されています。これらの関数を用いることで、メッセージを利用しなくてもマウスおよびマウスカーソルの状態を取得することができます。

DragDetect	マウスのドラッグを追跡します
mouse_event	マウスの移動やボタン操作に相当するイベントを合成します
ReleaseCapture	マウスのキャプチャを解放します
SetCapture	マウスをキャプチャします
SetDoubleClickTime	マウスのダブルクリック時間を設定します
SwapMouseButton	マウスボタンの左右の機能を反転するか、元に戻します
TrackMouseEvent	マウスがウィンドウを離れるとき、またはウィンドウ上にあるときにメッセージを送ります
ClipCursor	マウスカーソルの移動範囲を制限します
GetCursorPos	マウスカーソルのスクリーン座標を取得します
LoadCursor	マウスカーソルをリーソースから読み込みます
LoadCursorFromFile	マウスカーソルを.CURまたは.ANIファイルから読み込みます
ScreenToClient	画面上の指定されたスクリーン座標の点をクライアント座標に変換します
SetCursor	マウスカーソルの形状を設定します
SetCursorPos	マウスカーソルの位置を指定されたスクリーン座標に設定します
ShowCursor	マウスカーソルを表示または非表示にします

マウスは、マウスそのものとマウスカーソルとで関数が分かれています。マウスのボタンの状態は、GetAsyncKeyState関数のようなキーボードのキー状態を取得する関数で調べることができます。

```
// マウスの左ボタンが押されているか調べる
if((GetAsyncKeyState(VK_LBUTTON) & 0x8000) != 0)
    // マウスの左ボタンは押されている
```

マウスカーソルの座標は、GetCursorPos関数で取得することができます。ただし、この関数が返す座標はスクリーンの左上を原点とするスクリーン座標なので、ウィンドウのクライアント領域の座標系(クライアント座標)に変換する必要があります。

```
// マウスカーソルの座標を取得
POINT   Mouse;
GetCursorPos(&Mouse);           // マウスカーソルの座標を取得
ScreenToClient(hWnd, &Mouse);   // クライアント座標に変換
```

ジョイスティック

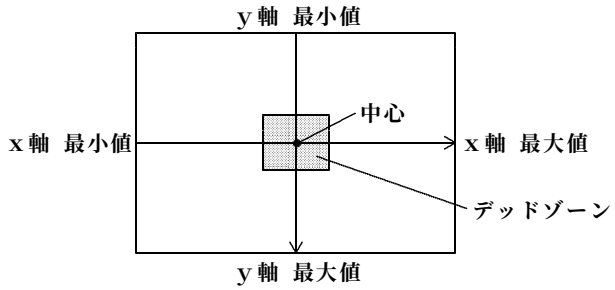
ジョイスティックを扱うAPIには、以下のようなものがあります。

joyGetDevCaps	ジョイスティックを照会して、その性能を取得します
joyGetNumDevs	サポートされるジョイスティック数を取得します
joyGetPos	ジョイスティックを照会して、軸の位置とボタンの状態を取得します
joyGetPosEx	ジョイスティックを照会して、軸の位置とボタンの状態を取得します

ジョイスティックは、装置により能力が異なるので、joyGetDevCaps関数で性能を取得しておきます。この関数により、以下のような詳細な情報がJOYCAPS構造体に取得できます。

```
struct JOYCAPS {
    WORD wMid;                                  // ジョイスティックの製造元ID
    WORD wPid;                                  // ジョイスティックのプロダクトID
    CHAR szPname[MAXPNAMELEN];                  // ジョイスティックの製品名
    UINT wXmin;                                 // X軸の最小値
    UINT wXmax;                                 // X軸の最大値
    UINT wYmin;                                 // Y軸の最小値
    UINT wYmax;                                 // Y軸の最大値
    UINT wZmin;                                 // Z軸の最小値
    UINT wZmax;                                 // Z軸の最大値
    UINT wNumButtons;                           // ボタンの数
    UINT wPeriodMin;                            // 最小ポーリング間隔
    UINT wPeriodMax;                            // 最大ポーリング間隔
    UINT wRmin;                                 // R軸の最小値
    UINT wRmax;                                 // R軸の最大値
    UINT wUmin;                                 // U軸の最小値
    UINT wUmax;                                 // U軸の最大値
    UINT wVmin;                                 // V軸の最小値
    UINT wVmax;                                 // V軸の最大値
    UINT wCaps;                                 // ジョイスティックの能力を示すフラグ
    UINT wMaxAxes;                              // サポートする軸の最大数
    UINT wNumAxes;                              // 軸の数
    UINT wMaxButtons;                           // サポートするボタンの最大数
    CHAR szRegKey[MAXPNAMELEN];                 // ジョイスティックのレジストリキー文字列
    CHAR szOEMVxD[MAX_JOYSTICKOEMVXDNAME];      // OEMドライバ名
};
```

軸を上下左右に傾けたときの値と、軸の最大値、最小値との関係は、以下のようになります。



ジョイスティックの軸とボタンの状態は、joyGetPosEx関数で取得することができます。この関数は、JOYINFOEX構造体にジョイスティックの状態を設定します。JOYINFOEX構造体のメンバは、以下のとおりです。

```
struct JOYINFOEX {
    DWORD dwSize;           // この構造体のサイズ
    DWORD dwFlags;          // 取得する情報を示したフラグ
    DWORD dwXpos;           // X軸の位置
    DWORD dwYpos;           // Y軸の位置
    DWORD dwZpos;           // Z軸の位置
    DWORD dwRpos;           // R軸の位置
    DWORD dwUpos;           // U軸の位置
    DWORD dwVpos;           // V軸の位置
    DWORD dwButtons;        // ボタンの状態
    DWORD dwButtonNumber;   // 押されているボタンの数
    DWORD dwPOV;            // 視点または方向コントローラの位置
    DWORD dwReserved1;      // 予約されており0に設定されます
    DWORD dwReserved2;      // 予約されており0に設定されます
};
```

軸は、位置そのものがJOYINFOEX構造体のdwXposメンバやdwYposメンバに設定されます。ボタンの状態は、まとめてdwButtonsメンバに設定されます。特定のボタンの状態を取得するには、ビット演算を使用します。

```
// ジョイスティック能力取得
JOYCAPS   JoyCaps;
joyGetDevCaps(JOYSTICKID1, &JoyCaps, sizeof(JoyCaps);

// ジョイスティック状態取得
JOYINFOEX   JoyInfo;
JoyInfo.dwSize  = sizeof(JoyInfo);
JoyInfo.dwFlags = JOY_RETURNALL;
joyGetPosEx(JOYSTICKID1, &JoyInfo);

// ジョイスティックのボタン0が押されているか調べる
if((JoyInfo.dwButtons & (1 << 0)) != 0)
    // 押されている

// ジョイスティックのボタン4が押されているか調べる
if((JoyInfo.dwButtons & (1 << 4)) != 0)
    // 押されている
```

ボタンの押し下げと押し上げ

APIで取得できるボタンの状態は、「押されている」「押されていない」の2つだけです。押し下げられた瞬間や押し上げられた瞬間という情報を取得することはできません。これらの情報が欲しい場合は、前フレームのボタンの状態と現フレームのボタンの状態が必要になります。前フレームのボタンの状態を準備するには、ゲーム処理(シーンの内部処理)の最後でボタンの状態を保存しておきます。次回のフレームでは、それが前フレームのボタンの状態になるわけです。これらを比較して状態が変わった瞬間を検出します。

ボタンの状態が、前フレームで「押されていない」、現フレームで「押されている」場合、押し下げられた瞬間を表しています。逆に、前フレームでは「押されている」、現フレームでは「押されていない」場合、押し上げられた(離された)瞬間を表しています。

課　　題

入力デバイスによりキャラクターを移動させてみましょう。

(１)ジョイスティックを管理するクラスCJoystickを作成します。CJoystickクラスのヘッダファイル(Joystick.hpp)を以下のように作成しましょう。

―Joystick.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) 2004 Hibikino software. All rights reserved.
================================================================================================

  【対象ＯＳ】
     Microsoft Windows98/2000以降

  【コンパイラ】
     Microsoft VisualC++ 6.0J ServicePack6

  【プログラム】
     Joystick.hpp
                ジョイスティッククラスヘッダ

  【履歴】
     * Version    1.00       2004/04/dd  hh:mm:ss

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

#pragma once

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

/*********************************************************************************************/
/*                                      ジョイスティックID定義                                 */
/*********************************************************************************************/
enum JOYSTICKID {
    JOYSTICKID3 = JOYSTICKID2 + 1,
    JOYSTICKID4,
    JOYSTICKID5,
    JOYSTICKID6,
    JOYSTICKID7,
    JOYSTICKID8,
    JOYSTICKID9,
    JOYSTICKID10,
    JOYSTICKID11,
    JOYSTICKID12,
    JOYSTICKID13,
    JOYSTICKID14,
    JOYSTICKID15,
    JOYSTICKID16,
    JOYSTICKID_MAX
};

/*********************************************************************************************/
/*                                    ジョイスティッククラス定義                               */
/*********************************************************************************************/
class CJoystick {
public:
    CJoystick(const UINT inJoyID);
    virtual ~CJoystick() {}

    bool GetState(JOYINFOEX& outJoyInfo);
    JOYCAPS GetCaps() const { return ※ここは各自考えましょう; }

private:
    UINT      m_JoyID;      // ジョイスティックID
    JOYCAPS   m_JoyCaps;    // ジョイスティック能力
```

```
    CJoystick(const CJoystick&);
    CJoystick& operator=(const CJoystick&);
};
```

(２)CJoystickクラスのメンバは、以下のとおりです。

CJoystick　コンストラクタ

CJoystickオブジェクトを構築し、指定されたIDのジョイスティックを使用できる状態にします。

書式 CJoystick(const UINT inJoyID);

inJoyID	ジョイスティックのID

~CJoystick　デストラクタ

CJoystickオブジェクトを解放します。

GetState　取得

ジョイスティックの状態を取得します。

書式 bool GetState(JOYINFOEX& outJoyInfo);

Return	成功：true　それ以外：false
outJoyInfo	ジョイスティックの状態を受け取るJOYINFOEX構造体

GetCaps　アクセス関数

ジョイスティックの能力を取得します。

書式 JOYCAPS GetCaps() const;

Return	ジョイスティックの能力を格納したJOYCAPS構造体

m_JoyID　メンバ変数

CJoystickオブジェクトが管理するジョイスティックのIDです。

書式 UINT m_JoyID;

m_JoyCaps　メンバ変数

CJoystickオブジェクトが管理するジョイスティックの能力を格納したJOYCAPS構造体です。

書式 JOYCAPS m_JoyCaps

(３)CJoystickクラスのソースファイル(Joystick.cpp)を以下のように作成しましょう。

－Joystick.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) 2004 Hibikino software. All rights reserved.
================================================================================================

  【対象ＯＳ】
    Microsoft Windows98/2000以降

  【コンパイラ】
    Microsoft VisualC++ 6.0J ServicePack6

  【プログラム】
    Joystick.cpp
                ジョイスティッククラス

  【履歴】
    * Version    1.00       2004/04/dd  hh:mm:ss

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

/*************************************************************************************************/
/*                                        インクルードファイル                                   */
/*************************************************************************************************/
#include ※ここは各自考えましょう

/*************************************************************************************************/
/*                                        コンストラクタ                                         */
/*************************************************************************************************/
```

```
CJoystick::CJoystick(const UINT inJoyID) : m_JoyID(-1)
{
    ::ZeroMemory(&m_JoyCaps, sizeof(m_JoyCaps));

    if(inJoyID > JOYSTICKID16) {
        ::OutputDebugString("*** Error - ジョイスティックIDエラー(CJoystick_CJoystick)\n");
        return;
    }

    // ジョイスティック能力取得
    if(::?????????????(inJoyID, &m_JoyCaps, sizeof(m_JoyCaps)) != JOYERR_NOERROR) {
        ::OutputDebugString("*** Error - ジョイスティック能力取得失敗(CJoystick_CJoystick)\n");
        return;
    }

    m_JoyID = inJoyID;
}

/*****************************************************************************************************/
/*                                          状態取得                                                 */
/*****************************************************************************************************/
bool CJoystick::GetState(JOYINFOEX& outJoyInfo)
{
    outJoyInfo.dwSize  = sizeof(outJoyInfo);
    outJoyInfo.dwFlags = JOY_RETURNALL;
    if(::???????????(m_JoyID, &outJoyInfo) != JOYERR_NOERROR) {
        ::OutputDebugString("*** Error - ジョイスティック状態取得失敗(CJoystick_GetState)\n");
        ::ZeroMemory(&outJoyInfo, sizeof(outJoyInfo));
        return false;
    }

    return true;
}
```

(４)CJoystickクラス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クラスのメンバに追加

```
CJoystick   m_Joystick;
```

・CTestSceneクラスのコンストラクタ初期化子に追加

```
m_Joystick(JOYSTICKID1)
```

・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
    // ジョイスティック状態取得
    JOYINFOEX   JoyState;
    m_Joystick.GetState(JoyState);

    // 描画処理
    if(FPSTimer().IsSkip() == false) {
        // バックバッファ消去
        ::PatBlt(m_pBackBuf->GetDC(), 0, 0, m_pBackBuf->GetWidth(), m_pBackBuf->GetHeight(),
                 BLACKNESS);

        // ジョイスティック情報描画
        TCHAR   JoyStr[64];
        ::wsprintf(JoyStr, "X:%5d Y:%5d Z:%5d R:%5d B:%04x",
                   JoyState.dwXpos, JoyState.dwYpos, JoyState.dwZpos, JoyState.dwRpos,
                   JoyState.dwButtons);
        ::TextOut(m_pBackBuf->GetDC(), 0, 16, JoyStr, ::lstrlen(JoyStr));

        // FPS描画
        FPSTimer().DrawFPS(m_pBackBuf->GetDC());

        // バックバッファからディスプレイへ転送
        HDC   hDestDC = ::GetDC(GameApp().GetHWnd());
        ::BitBlt(hDestDC, 0, 0, m_pBackBuf->GetWidth(), m_pBackBuf->GetHeight(),
                 m_pBackBuf->GetDC(), 0, 0, SRCCOPY);
        ::ReleaseDC(GameApp().GetHWnd(), hDestDC);
    }
```

```
    return 0;
}
```

(５)以下のような「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

(６)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

```
int    m_CharaX;        // キャラクタｘ座標
int    m_CharaY;        // キャラクタｙ座標
int    m_CharaAnime;    // アニメーションカウンタ
```

・CTestSceneクラスのコンストラクタ初期化子に追加

```
m_CharaX(0), m_CharaY(0), m_CharaAnime(0)
```

・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
    // キャラクタ移動
    if((::GetAsyncKeyState(VK_LEFT)  & 0x8000) != 0)
        m_CharaX--;
    if((::GetAsyncKeyState(VK_RIGHT) & 0x8000) != 0)
        m_CharaX++;

    // アニメーション
    m_CharaAnime = (m_CharaAnime + 1) % 5;

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

    // 背景転送
    ::BitBlt(m_pBackBuf->GetDC(), 0, 0, m_pBackBuf->GetWidth(), m_pBackBuf->GetHeight(),
             m_pBG->GetDC(), 0, 0, SRCCOPY);

    // キャラクタ転送
    ::TransparentBlt(m_pBackBuf->GetDC(), m_CharaX, m_CharaY, 120, 120,
                     m_pChara->GetDC(), 0, 0, 120, 120, RGB(0, 0, 0));

    // FPS描画
    FPSTimer().DrawFPS(m_pBackBuf->GetDC());


    // バックバッファからディスプレイへ転送
    HDC   hDestDC = ::GetDC(GameApp().GetHWnd());
    ::BitBlt(hDestDC, 0, 0, m_pBackBuf->GetWidth(), m_pBackBuf->GetHeight(),
             m_pBackBuf->GetDC(), 0, 0, SRCCOPY);
    ::ReleaseDC(GameApp().GetHWnd(), hDestDC);

    return 0;
}
```

(７)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。

(８)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。

(９)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。

(10)キャラクタがジョイスティックで動くようにプログラムを変更しましょう。

(11)ジョイスティックの軸がアナログの場合は、軸の傾き具合によってキャラクタの移動速度が変わるようにプログラムを変更しましょう。

(12)キャラクタをアニメーションさせましょう。

応用問題１　CJoysticクラスを継承し、ボタンの押し下げと押し上げの瞬間の検出など、必要な機能を強化したクラスCJoystickExを作成しましょう。
応用問題２　キーボードを管理するクラスCKeyboardを作成しましょう。
応用問題３　マウスを管理するクラスCMouseを作成しましょう。