オブジェクト指向と
ゲームプログラミング

▷　フレームワーク編 － 第１４回　フレームレートの制御　◁

メインループ

ゲームプログラムは、「ものの状態の集まり」と考えることができます。キャラクタの座標やパラメータ、背景、画面エフェクトなどすべて、それぞれ「もの」の状態です。ゲームプログラムとは、このいろいろな「もの」の状態を時間に沿って更新し、それを画面に描画するプログラムと考えることができます。
つまり、ゲームプログラムの処理を非常に単純化すると、

- 「もの」の状態を更新(内部処理)
- 「もの」を画面に描画(描画処理)

という２つにまとめることができます。
この２つの処理の繰り返しがメインループになります。このループ１回ぶんを「１フレーム」と呼びます。１フレームごとに、キャラクタの位置や状態、背景やそのほかの表示状態などを更新し、その状態を反映して描画を行う、という処理を繰り返していきます。

フレームレート

ゲームプログラムでは、メインループ(正確には画面の更新)を一定の間隔で行う必要があります。そうしないと、描画速度が環境により異なってしまい、速い環境ほど高速に動作してしまいます(ゲームによっては、環境の速度に合わせて背景やキャラクタの移動量を決定しているものもあります)。
１秒間に何回画面を更新するのかをフレームレート(FPS:Frame Per Second)と呼びます。フレームレートを設定し、この間隔にあわせてゲームを動かします。たいていのゲームでは60FPSで、１フレームを約16.67ミリ秒の間隔で処理しています。

フレームレートの制御

フレームレートを制御するには、処理にかかった時間を計測し、次のフレーム処理を開始する(または画面を更新する)タイミングを算出する必要があります。時間を取得するAPIはいくつかあり、それぞれ精度が異なります。
また、指定した時間だけプログラムを一時休止する処理も必要になります。この動作をするAPIのひとつに、Sleep関数があります。この関数は、指定した時間だけプログラムを休止し、実行権をほかのアプリケーションに渡す動作をします。
これらを組み合わせてフレームレートを制御します。

○GetTickCount関数

システムを起動してからの経過時間を、ミリ秒単位で取得します。精度は、システムタイマの分解能による制限を受けます。システムタイマの分解能を取得するには、GetSystemTimeAdjustment関数を使います。経過時間はDWORD型で保存されており、システムを4,294,967,295秒(約49.71日)連続して動作させると、経過時間は０に戻ります。

○timeGetTime関数

システムを起動してからの経過時間を、ミリ秒単位で取得します。精度は、OSによって異なります。精度を取得するには、timeGetDevCaps関数を使います。経過時間はDWORD型で保存されており、システムを4,294,967,295秒(約49.71日)連続して動作させると、経過時間は０に戻ります。
WindowsNT/2000/XPでは、timeGetTime関数の精度は、環境によっては５ミリ秒以上になる場合があります。timeGetTime関数の精度は、timeBeginPeriod関数とtimeEndPeriod関数を使って設定することができます。
Windows95/98/Meでは、timeGetTime関数の精度は１ミリ秒です。timeBeginPeriod関数とtimeEndPeriod関数でどのような精度に設定しても、１ミリ秒の精度を変更することはできません。
これらの関数は、ライブラリ"winmm.lib"が必要になります。

○QueryPerformanceCounter関数

高分解能パフォーマンスカウンタが存在する場合、そのカウンタの現在の値を取得します。パフォーマンスカウンタは0.838マイクロ秒の分解能を持ち、カウンタが１周するまで約700,000年かかります。

高分解能パフォーマンスカウンタの周波数(１秒あたりのカウント数)を取得するには、QueryPerformanceFrequency関数を使います。
最も高精度、高品質な性能を持ちますが、パフォーマンスカウンタが存在しない環境や、QueryPerformanceCounter関数がつねに０を返す環境も存在します。

課　　題

アプリケーションのメインループを一定の間隔で回し、ゲームが一定の速度で実行されるようにしましょう。

アプリケーションのメインループは、CGameApp::Main関数です。ここを一定の間隔で実行するようにします。一定の間隔で実行させるため、タイマで時間を計測してタイミングをとるようにします。

(１)フレームレート制御を行うクラスCFixTimerを作成します。CFixTimerクラスのヘッダファイル(FixTimer.hpp)を以下のように作成しましょう。

－FixTimer.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】
     FixTimer.hpp
                FPS制御タイマクラスヘッダ

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

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

#pragma once

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

/*********************************************************************************************/
/*                                      FPS制御タイマクラス定義                                */
/*********************************************************************************************/
class CFixTimer {
public:
    virtual ~CFixTimer();

    void SetFPS(const UINT inFPS) { ※(５) }
    UINT GetFPS() const { return m_FPS; }

    void DrawFPS(const HDC hDC, const int inX, const int inY);

    // シングルトンインスタンスの取得
    static CFixTimer& GetInstance()
    {
        static CFixTimer   theFixTimer;
        return theFixTimer;
    }

private:
    CFixTimer();
    bool Initialize();
```

```
    UINT        m_FPS;              // １秒あたりのフレーム数
    UINT        m_PeriodMin;        // マルチメディアタイマ最小分解能
    __int64     m_Frequency;        // パフォーマンスカウンタ周波数
    __int64     m_MaxWait;          // 最大休止時間

    // フレームレート計測用変数
    int         m_DrawCount;        // 描画数
    int         m_DrawFPS;          // フレームレート描画用
    DWORD       m_LastDraw;         // 前描画時間
};

/*****************************************************************************************************/
/*                                          マクロ定義                                               */
/*****************************************************************************************************/
inline CFixTimer& FixTimer() { return CFixTimer::GetInstance(); }
```

(２)CFixTimerクラスのメンバは、以下のとおりです。

CFixTimer — プライベートなメンバ：コンストラクタ

CFPSTimerオブジェクトを構築します。

~CFixTimer — デストラクタ

CFPSTimerオブジェクトを解放します。

GetInstance — クラス関数：アクセス関数

アプリケーションで唯一のCFixTimerオブジェクトを取得します。

書式 `static CFixTimer& GetInstance();`

Return　CFixTimerオブジェクトの参照

SetFPS — 設定

１秒あたりのフレーム数を設定します。

書式 `void SetFPS(const UINT inFPS);`

inFPS　FPS

GetFPS — アクセス関数

CFPSTimerオブジェクトに設定されているFPSを取得します。

書式 `UINT GetFPS() const;`

Return　FPS

DrawFPS — 描画

１秒あたりの描画数、スキップ数を指定されたデバイスコンテキストに描画します。

書式 `void DrawFPS(const HDC hDC, const int inX, const int inY);`

hDC　デバイスコンテキストのハンドル
inX　描画先のｘ座標
inY　描画先のｙ座標

Initialize — プライベートなメンバ：初期化

CFixTimerオブジェクトが使用するタイマの初期設定を行います。

書式 `bool Initialize();`

Return　成功：true　それ以外：false

m_FPS — メンバ変数

１秒あたりのフレーム数です。

書式 `UINT m_FPS;`

m_PeriodMin — メンバ変数

マルチメディアタイマの最小分解能を保持します。

書式 `UINT m_PeriodMin;`

m_Frequency — メンバ変数

高分解能パフォーマンスカウンタの周波数です。

書式 `__int64 m_Frequency;`

m_MaxWait　メンバ変数

　１フレームあたりの最大休止時間です。

```
__int64 m_MaxWait;
```

m_DrawCount　メンバ変数

　フレームレートの描画に使用する、描画数を保持する変数です。

```
int m_DrawCount;
```

m_DrawFPS　メンバ変数

　フレームレートの描画に使用する、実際に描画される描画数を保持する変数です。

```
int m_DrawFPS;
```

m_LastDraw　メンバ変数

　フレームレートの描画に使用する、前回の描画時点の時間を保持する変数です。

```
DWORD m_LastDraw;
```

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

－FixTimer.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】
    FixTimer.cpp
                FPS制御タイマクラス

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

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

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

/*********************************************************************************************/
/*                                        静的リンクライブラリ                                 */
/*********************************************************************************************/
#pragma comment(lib, "winmm.lib")

/*********************************************************************************************/
/*                                      デフォルトコンストラクタ                               */
/*********************************************************************************************/
CFixTimer::CFixTimer() : m_FPS(0), m_PeriodMin(0), m_Frequency(0), m_MaxWait(0),
                         m_DrawCount(0), m_DrawFPS(0), m_LastDraw(0)
{
    Initialize();
    SetFPS(60);
    ※(９)
}

/*********************************************************************************************/
/*                                          デストラクタ                                     */
/*********************************************************************************************/
CFixTimer::~CFixTimer()
{
    if(m_PeriodMin != 0)
        ::timeEndPeriod(m_PeriodMin);
}
```

```
/*****************************************************************************************************/
/*                                          タイマ初期化                                             */
/*****************************************************************************************************/
bool CFixTimer::Initialize()
{
    // マルチメディアタイマ能力取得
    TIMECAPS    tc;
    ::timeGetDevCaps(&tc, sizeof(tc));
    m_PeriodMin = tc.wPeriodMin;

    // マルチメディアタイマ分解能設定
    ::timeBeginPeriod(m_PeriodMin);

    // パフォーマンスカウンタ周波数取得
    if(::QueryPerformanceFrequency((LARGE_INTEGER*)&m_Frequency) == 0)
        return false;

    return true;
}

/*****************************************************************************************************/
/*                                        フレームレート描画                                         */
/*****************************************************************************************************/
void CFixTimer::DrawFPS(const HDC hDC, const int inX, const int inY)
{
    const DWORD   CURRENT_TIME = ※(4);                        // 現時間
    const DWORD   ELAPSED_TIME = CURRENT_TIME - m_LastDraw;    // 経過時間
    if(ELAPSED_TIME >= 1000) {
        m_DrawFPS   = m_DrawCount * 1000 / ELAPSED_TIME;
        m_DrawCount = 0;
        m_LastDraw  = CURRENT_TIME;
    }

    // FPS描画
    TCHAR   info[16];
    ::wsprintf(info, "FPS:%3d", m_DrawFPS);
    ::TextOut(hDC, inX, inY, info, ::lstrlen(info));
}
```

(4)「※(4)」に「::GetTickCount()」または「::timeGetTime()」のうち適切なものを入れましょう。

(6)以下のアルゴリズム1からアルゴリズム7は、フレームレートを安定させるための処理です。適切なものを選び、ゲームが一定の速度で実行されるようにしましょう。

●アルゴリズム1

1秒をFPSで割り、休止時間を求めます。メインループの最後で、Sleep関数を使って求めた時間休止し、タイミングを調整します。

－追加メンバ－

```
/*****************************************************************************************************/
/*                                            待　機                                                 */
/*****************************************************************************************************/
void CFixTimer::Wait()
{
    m_DrawCount++;
    ::Sleep(m_MaxWait);
}
```

－メインループ－

```
int CGameApp::Main()
{
    // メインループ
    while(true) {
        // メッセージ処理
        if(MessageLoop() == false)
            break;
        // ゲーム処理
        if(m_GameProc.???(m_Active) == false)
```

```
            Release();      // アプリケーション終了
        // ウェイト
        FixTimer().Wait();
    }

    return 0;               // 正常終了
}
```

●アルゴリズム２

１フレームを処理するのにかかった時間から休止時間を求め、Sleep関数で休止する方法です。
１フレームを処理するのにかかった時間は、

１フレームを処理するのにかかった時間 ＝ フレーム終了時間 － フレーム開始時間

で求めます。休止時間は以下の式で求め、Sleep関数で休止します。

休止時間 ＝ １フレームにかけられる時間 － １フレームを処理するのにかかった時間
(※休止時間が負の場合は、処理落ちになります)

時間の取得には、GetTickCount関数を使います。

－追加メンバ１－

```
DWORD   m_BeginTime;      // フレーム開始時間
```

－追加メンバ２－

```
/*****************************************************************************************************/
/*                                          フレーム開始                                             */
/*****************************************************************************************************/
void CFixTimer::BeginFrame()
{
    m_BeginTime = ::GetTickCount();
}

/*****************************************************************************************************/
/*                                          フレーム終了                                             */
/*****************************************************************************************************/
void CFixTimer::EndFrame()
{
    m_DrawCount++;

    // フレームレート制御処理
    const long   WAIT_TIME = m_MaxWait - (::GetTickCount() - m_BeginTime);
    if(WAIT_TIME > 0)
        ::Sleep((DWORD)WAIT_TIME);  // あまった時間は休止
}
```

－追加コンストラクタ初期化子－

```
m_BeginTime(0)
```

－メインループ－

```
int CGameApp::Main()
{
    // メインループ
    while(true) {
        // フレーム開始
        FixTimer().BigenFrame();

        // メッセージ処理
        if(MessageLoop() == false)
            break;
        // ゲーム処理
        if(m_GameProc.???(m_Active) == false)
            Release();      // アプリケーション終了

        // フレーム終了
        FixTimer().EndFrame();
    }

    return 0;               // 正常終了
}
```

●アルゴリズム３
アルゴリズム２のGetTickCount関数をtimeGetTime関数に変更したものです。

```
/*****************************************************************************************************/
/*                                          フレーム開始                                             */
/*****************************************************************************************************/
void CFixTimer::BeginFrame()
{
    m_BeginTime = ::timeGetTime();
}

/*****************************************************************************************************/
/*                                          フレーム終了                                             */
/*****************************************************************************************************/
void CFixTimer::EndFrame()
{
    m_DrawCount++;

    // フレームレート制御処理
    const long   WAIT_TIME = m_MaxWait - (::timeGetTime() - m_BeginTime);
    if(WAIT_TIME > 0)
        ::Sleep((DWORD)WAIT_TIME);  // あまった時間は休止
}
```

●アルゴリズム４
１フレームを処理するのにかかった時間を

１フレームを処理するのにかかった時間 ＝ フレーム終了時間 － 前回のフレーム終了時間

という式で求めます。休止はアルゴリズム２と同じ方法で行い、時間の取得にはGetTickCount関数を使います。

－追加メンバ１－

```
  DWORD   m_LastTime;       // フレーム開始時間
```

－追加メンバ２－

```
/*****************************************************************************************************/
/*                                            待　機                                                 */
/*****************************************************************************************************/
void CFixTimer::Wait()
{
    m_DrawCount++;

    // フレームレート制御処理
    const long   WAIT_TIME = m_MaxWait - (::GetTickCount() - m_LastTime);
    if(WAIT_TIME > 0)
        ::Sleep((DWORD)WAIT_TIME);  // あまった時間は休止
    m_LastTime = ::GetTickCount();
}
```

－追加コンストラクタ初期化子－

```
  m_LastTime(0)
```

－メインループ－

```
int CGameApp::Main()
{
    // メインループ
    while(true) {
        // メッセージ処理
        if(MessageLoop() == false)
            break;
        // ゲーム処理
        if(m_GameProc.???(m_Active) == false)
            Release();      // アプリケーション終了
        // ウェイト
        FixTimer().Wait();
    }

    return 0;               // 正常終了
}
```

●アルゴリズム５
アルゴリズム４のGetTickCount関数をtimeGetTime関数に変更したものです。

```
/*****************************************************************************************************/
/*                                          待　機                                                   */
/*****************************************************************************************************/
void CFixTimer::Wait()
{
    m_DrawCount++;

    // フレームレート制御処理
    const long   WAIT_TIME = m_MaxWait - (::timeGetTime() - m_LastTime);
    if(WAIT_TIME > 0)
        ::Sleep((DWORD)WAIT_TIME);  // あまった時間は休止
    m_LastTime = ::timeGetTime();
}
```

●アルゴリズム６
QueryPerformanceCounter関数で時間を取得し、１ミリ秒未満の休止時間も考慮する方法です。１ミリ秒未満の時間は、ビジーループで消費させます。
ビジーループとは、以下のような「何もしない」ループのことです。

```
while(true)
    ;   // なにもしない
```

ビジーループを長時間行うと、CPUを猛烈に消費し、Windowsが不安定になってしまいます。しかし、１ミリ秒に満たない時間の休止を行いたい場合は、このループでCPUを浪費させるしかありません。

－追加メンバ１－

```
__int64   m_LastTime;     // フレーム開始時間
```

－追加メンバ２－

```
/*****************************************************************************************************/
/*                                          待　機                                                   */
/*****************************************************************************************************/
void CFixTimer::Wait()
{
    m_DrawCount++;

    // 現時間取得
    __int64   current_time;
    ::QueryPerformanceCounter((LARGE_INTEGER*)&current_time);

    // 休止時間設定
    __int64   wait_time = m_MaxWait - (current_time - m_LastTime);

    // 休止処理
    if(wait_time > 0) {
        // Sleepによる休止(WaitTimeのパフォーマンス秒をミリ秒に変換して休止)
        ::Sleep((DWORD)(wait_time * 1000 / m_Frequency));

        // ミリ秒未満の休止
        __int64   current_time2;
        ::QueryPerformanceCounter((LARGE_INTEGER*)&current_time2);
        wait_time -= current_time2 - current_time;
        do {
            ::QueryPerformanceCounter((LARGE_INTEGER*)&m_LastTime);
        } while(wait_time > m_LastTime - current_time2);
    } else {
        // 現時間取得
        ::QueryPerformanceCounter((LARGE_INTEGER*)&m_LastTime);
    }
}
```

－追加コンストラクタ初期化子－

```
m_LastTime(0)
```

－メインループ－
```
int CGameApp::Main()
{
    // メインループ
    while(true) {
        // メッセージ処理
        if(MessageLoop() == false)
            break;
        // ゲーム処理
        if(m_GameProc.???(m_Active) == false)
            Release();      // アプリケーション終了
        // ウェイト
        FixTimer().Wait();
    }

    return 0;               // 正常終了
}
```

●アルゴリズム７

アルゴリズム６のWait関数を変更し、すべての休止時間を「Sleep(0)」のループで消費する方法です。Sleep関数に０を指定すると、実行の準備ができている同じ優先順位のほかのアプリケーション(スレッド)に実行権を譲ります。そのようなスレッドがない場合は、関数はすぐに戻ります。

```
/*****************************************************************************************************/
/*                                          待　機                                                   */
/*****************************************************************************************************/
void CFixTimer::Wait()
{
    // 現時間取得
    __int64   current_time;
    ::QueryPerformanceCounter((LARGE_INTEGER*)&current_time);

    // 休止時間設定
    __int64   wait_time = m_MaxWait - (current_time - m_LastTime);

    // 休止処理
    do {
        ::Sleep(0);
        ::QueryPerformanceCounter((LARGE_INTEGER*)&m_LastTime);
    } while(wait_time > m_LastTime - current_time);
}
```

(７)SetFPS関数を実装します。以下のうち適切なものを選びましょう。

```
void SetFPS(const UINT inFPS) { m_FPS = inFPS; m_MaxWait = 1000 / inFPS; }
void SetFPS(const UINT inFPS) { m_FPS = inFPS; m_MaxWait = m_Frequency / inFPS; }
```

(８)アルゴリズム１，２以外を採用する場合、タイマをリセットするReset関数を実装します。以下のうち適切なものを選び、追加しましょう。

```
void Reset() { m_LastTime = ::GetTickCount(); }
void Reset() { m_LastTime = ::timeGetTime();  }
void Reset() { ::QueryPerformanceCounter((LARGE_INTEGER*)&m_LastTime); }
```

(９)アルゴリズム１，２以外を採用する場合、CFixTimerクラスのコンストラクタの最後に、以下のプログラムを追加しましょう。

```
Reset();
```

(10)アルゴリズム１，２以外を採用する場合、CTestSceneクラスのコンストラクタの最後に以下のプログラムを追加しましょう。

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

(11)アルゴリズム１，２以外を採用する場合、アプリケーションがアクティブ状態に復帰した場合に、CFixTimerオブジェクトをリセットする必要があります。リセットしないと、復帰後最初のフレームの待機時間がゼロになる場合があります。
　CGameApp::OnActivateApp関数の適切な場所に、CFixTimerオブジェクトをリセットする処理を追加しましょう。

(12)CGameApp::Initialize関数に以下のプログラムを追加しましょう。

```
FixTimer().SetFPS(60);      // FPS設定
```

(13)メインループの最後に、以下のプログラムを追加すると、１秒あたりの実効フレーム数を表示できます。

```
// FPSの表示
HDC   hdc = ::GetDC(GetHWnd());
FixTimer().DrawFPS(hdc, 0, 0);
::ReleaseDC(GetHWnd(), hdc);
```