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１０回　メッセージループ　◁

ゲームプログラムとメッセージ処理

Windowsアプリケーションは、Windowsから送られてくるメッセージを処理する必要があります。メッセージが処理されないと、メッセージキューからメッセージがあふれ、「応答なし」状態と見なされます。このような状態が続くと、動作が不安定になるだけでなく、ほかのアプリケーションにも悪影響をおよぼす場合があります。

ワープロや表計算といったアプリケーションの場合、メッセージは非常に重要な役割を持っており、メッセージに応答しながら処理を進める形式がとられます。メッセージの取得にはGetMessage関数が使われ、メッセージが発生するまでプログラムを休止し、ほかのアプリケーションに実行権を渡します。メッセージがきたときだけ(なにかイベントが起きたときだけ)プログラムが動作するわけです。

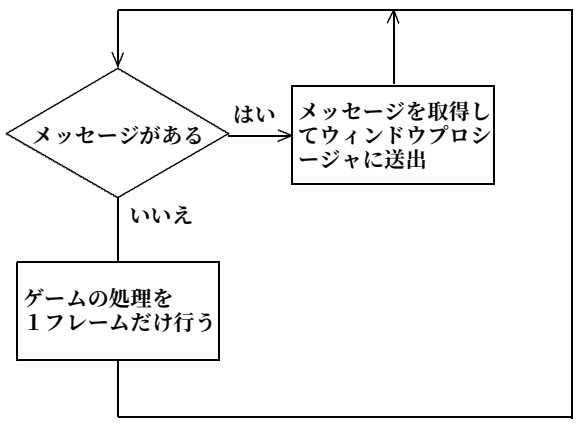
しかし、ゲームプログラムでは、ゲームの動作とメッセージはほとんど無関係です。たいていのゲームは、リアルタイムに動かし続けなければならないため、メッセージを待つことはありません。
ゲームプログラムでは、Windowsから送られてくるメッセージを処理しつつ、リアルタイムに処理を行わなければなりません。このようなプログラムでは、GetMessage関数は適さないので、代わりにPeekMessage関数でメッセージの取得を行います。この関数はメッセージを待つことがないので、メッセージがないときは直ちに次の処理を行えます。

ゲームプログラムでは、メッセージキューにあるメッセージをすべて処理し、なくなった隙にゲームの処理をします。こうすると、Windowsアプリケーションらしい振る舞いで、かつリアルタイムにゲームを動かすことができます。

```
while(true) {
    // メッセージキューに溜まったメッセージをすべて処理する
    while(PeekMessage(&msg, 0, 0, 0, PM_REMOVE) != 0) {
        if(msg.message == WM_QUIT)
            return msg.wParam;      // WM_QUITメッセージを受信した場合はプログラム終了
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }   // while(PeekMessage)

    // メッセージがなくなったら、ゲームを処理する
    ※ここで、ゲームを処理する関数の呼び出しを行います
}   // while(true)
```

－メッセージループ・フローチャート－



課　　題

ゲームアプリケーション用のメッセージループを実装しましょう。

(１)CGameAppクラスのMessageLoop関数を以下のように変更しましょう。

```
bool CGameApp::MessageLoop()
{
    MSG   msg;
    while(::???????????(&msg, 0, 0, 0, PM_REMOVE) != 0) {
        if(msg.message == WM_QUIT)
            return false;
        ::TranslateMessage(&msg);
        ::???????????????(&msg);
    }

    return true;
}
```