ゲームプログラミング

▶　基礎編－第１０回　非アクティブ時の処理　◀

Windowsアプリケーションには、アクティブ状態と非アクティブ状態の２つの状態があります。DirectXを使用したアプリケーションが非アクティブ状態になると、画面への描画ができなくなったり、入力デバイスからデータが取得できなくなるなど、一部の処理が行えなくなります。このため、非アクティブ状態では専用の処理を行う必要があります。

アクティブ状態と非アクティブ状態

Windowsアプリケーションには、アクティブ状態と非アクティブ状態の２つの状態があります。アクティブ状態とは、おもに前面に表示されているアプリケーションで、キーボードやマウスの入力権が与えられ、CPU実行権も優先的に与えられる状態です。通常はタイトルバーが青色になっています。これに対し、非アクティブ状態とは、キーボードやマウスの入力権が与えられず、いっさいの入力ができない状態です。通常はタイトルバーが灰色になっています。
ゲームプログラムでは、２つの状態を考慮したプログラムを作成する必要があります。

非アクティブ状態での制限

DirectXを使ったアプリケーションは、非アクティブ状態になるとさまざまな制限が課せられます。たとえば、DirectDrawで画面を占有している場合、非アクティブ状態になると強制的に最小化され、画面はほかのアプリケーションに渡されます。この状態では、画面を更新することができなくなります。また、いくつかのサーフェイス(画像を保持するためのメモリ領域)がロストし、サーフェイスへの操作が行えなくなります。DirectInputでは、入力デバイスを完全に占有することができますが、非アクティブ状態になったときには、キーボードやマウスの占有を解いて、ほかのアプリケーションが使用できるようにする必要があります。この状態ではキーボードやマウスから入力データを取得することはできないので、キャラクターなどを動かすことはできません。
このように、非アクティブ状態では制限が多くなります。画面も入力デバイスも使用できないので、コンピュータの思考など一部の処理だけを行うようにするか、ゲームを休止するようにします。

非アクティブ状態での処理

非アクティブ状態の処理を考慮するには、まずアプリケーションの状態を知る必要があります。いくつか方法がありますが、アプリケーションの状態が変更になったときに発生するメッセージを使う方法がもっとも簡単です。
アプリケーションの状態が変更になるたび、WM_ACTIVATEAPPメッセージが発生します。このとき、ウィンドウプロシージャの引数WPARAMには、アプリケーションがどの状態に変更されたかが格納されています。WPARAMの値が０なら非アクティブ状態、非０ならアクティブ状態に変更されたことを表します。このWPARAMの値をグローバル変数に保存しておき、ゲームの処理を行うときに役立てます。

```
BOOL   g_bActive;      // アプリケーションの状態を格納するグローバル変数

// ウィンドウプロシージャ
LRESULT CALLBACK WndProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
    switch(uMsg) {
      case WM_ACTIVATEAPP:
        g_bActive = (BOOL)wParam;  // アプリケーションの状態を保存
        return 0;
          :(以下省略)
```

メッセージループ後のゲーム処理は、以下のようにアプリケーション状態によって分岐させます。

```
// メッセージループ
while(true) {
    // メッセージをすべて処理する
    while(0 != PeekMessage(&msg, 0, 0, 0, PM_REMOVE)) {
        if(WM_QUIT == msg.message)
            return msg.wParam;
        TranslateMessage(&msg);
        DispatchMessage(&msg);
```

```
        }    // while(PeekMessage)

        // ゲーム処理
        if(0 != g_bActive)
            アクティブ状態のゲーム処理       // アクティブ状態なら通常のゲーム処理
        else
            非アクティブ状態のゲーム処理     // 非アクティブ状態なら専用の処理。たとえば、
                                             // コンピュータの思考など一部の処理だけ行う
    }    // while(true)
```

これらの処理のほかに、ゲームを休止する処理と休止から復帰する処理も必要になります。ゲームを休止する処理では、DirectXで占有しているデバイスを一時解放したり、音楽の再生を止めたりといった処理を行います。休止から復帰する処理では、デバイスの再占有や再構築、音楽の再生の復帰といった処理を行います。これらの処理はWM_ACTIVATEAPPメッセージが発生したときに行いますが、詳しくは後の回で説明します。

```
    // ウィンドウプロシージャ
    LRESULT CALLBACK WndProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
    {
        switch(uMsg) {
          case WM_ACTIVATEAPP:
            g_bActive = (BOOL)wParam;
            if(0 != wParam)
                // ゲームの復帰処理を行う関数の呼び出し
            else
                // ゲームの休止処理を行う関数の呼び出し
            return 0;
                ⋮(以下省略)
```

○ 課題 ○

アプリケーションの状態にあわせた処理を行うようにプログラムを変更しましょう。

(１)GameFunc.cppに、ゲームを休止する関数と休止から復帰する関数を作成します。

```
/*****************************************************************************************************/
/*                                          ゲーム休止                                               */
/*****************************************************************************************************/
void SuspendGame()
{
}

/*****************************************************************************************************/
/*                                          ゲーム復帰                                               */
/*****************************************************************************************************/
void ResumeGame()
{
}
```

(２)WinMain.cppで、アプリケーションの状態を格納する変数を宣言します。

```
    static BOOL   g_bActive = 0;         // アプリケーション状態
```

(３)WM_ACITVATEAPPメッセージを処理する関数を作成します。

```
/*****************************************************************************************************/
/*                                    WM_ACTIVATEAPPメッセージ処理                                   */
/*****************************************************************************************************/
int OnActivateApp(const HWND hWnd, const WPARAM wParam, const LPARAM lParam)
{
    ※この関数の内容は、各自考えて実装してください。

    return 0;
}
```

(４)ウィンドウプロシージャを変更し、WM_ACTIVATEAPPメッセージを処理するようにします。

```
// アプリケーション状態変更
case WM_ACTIVATEAPP:
   return ※ここは各自考えましょう;
```

(５)メッセージループ後のゲーム処理は、アプリケーションの状態にあった処理を行います。アクティブ状態の場合は通常のゲーム処理を行います。非アクティブ状態の場合はゲーム処理は行ないませんが、メッセージは処理しなければなりません。そこで、WaitMessage関数でメッセージ待ち状態にしておきます。この関数は、メッセージがくるまで休止し、CPU実行権をほかのアプリケーションに渡す動作をします。こうしておくと、メッセージがくるまでプログラムは休止し、メッセージがくるとメッセージループに制御が戻るようになります。

```
// ゲーム処理
if(0 != g_bActive)
     GameFunc();          // アクティブ状態ならゲーム処理
else
     WaitMessage();       // 非アクティブ状態ならメッセージ待ち
```

メッセージループとゲーム処理は以下のような流れになります。