# BREW & EZアプリ
# ゲームプログラミング

## ○　第４回　メインループ　○

### 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。キャラクター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)を一定の間隔で実行します。実行速度は環境によって異なるので、速い環境ほど高速に動作できます。

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。60FPSの場合、１秒間に60回画面を更新しているという意味になり、１フレームを約16.67ミリ秒の間隔で表示しています。一般的な家庭用ゲームでは60FPSです。BREW対応携帯電話では、30FPS以上で処理する能力を持ちます。


### タイマ

BREWでは、携帯Javaプログラミングと違い、メインループをループで構成することができません。これは、イベントが起きたときに必要な処理だけを行い、その後直ちに実行権をシステムに戻さなくてはならないという、イベントドリブン型のスタイルを採用しているためです(BREWでは、ひとつのイベント処理に多大な時間を費やすと、アプリケーションが強制終了させられます)。

メインループは、一定の間隔で実行されれば良いので、ループでなくても実現できます。たとえば、タイマという機能を用いる方法です。タイマは、一定の間隔で特定の動作を行うという機能です。

BREWのタイマは、「指定した時間」の経過後に、「指定した関数」を１度だけ実行させることができます。タイマの「指定した時間」をフレームの間隔に、「指定した関数」を内部処理および描画処理を行う関数に設定します。ゲームが終わるまでタイマを起動し続ければ、ループで構成するのとなんら変わりのない動作が得られます。

## 課　題

メインループを作成しましょう。

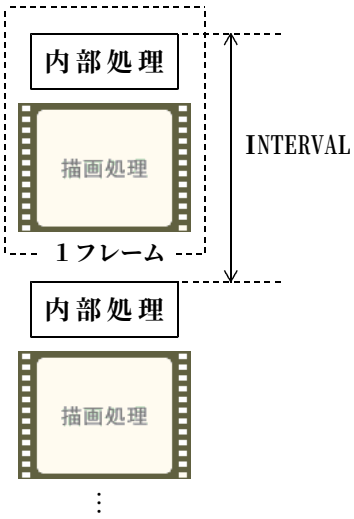
(１)*プロジェクト名*.cに、フレームレートを制御するための定数を定義します。定数FPSおよびINTERVALを「#include」群の下に追加しましょう。

```
/*=========================================================================
INCLUDES AND VARIABLE DEFINITIONS
=========================================================================== */
#include "AEEModGen.h"          // Module interface definitions
#include "AEEAppGen.h"          // Applet interface definitions
#include "AEEShell.h"           // Shell  interface definitions

#include "プロジェクト名.bid"

/*=========================================================================
CONSTANT NUMBERS
=========================================================================== */
#define  FPS        30
#define  INTERVAL   1000 / FPS
```

定数FPSが１秒あたりのフレーム数、定数INTERVALが現フレームの先頭から次フレームの先頭までの間隔です。




(２)メインループを記述する関数を作成します。以下のプログラムを適切な場所に追加しましょう。

```
/*=========================================================================
FUNCTION プロジェクト名_MainLoop

DESCRIPTION
    This is the MainLoop for this Game.

PROTOTYPE:
    static void プロジェクト名_MainLoop(プロジェクト名* pMe)

PARAMETERS:
    pMe: Pointer to the AEEApplet structure. This structure contains information specific
    to this applet. It was initialized during the AEEClsCreateInstance() function.

DEPENDENCIES
  none
```

```
RETURN VALUE
  none

SIDE EFFECTS
  none
===========================================================================*/
static void プロジェクト名_MainLoop(プロジェクト名* pMe)
{
}
```

※「プロジェクト名」は作成するプロジェクト名(半角英数字)に置き換えてください

(３)内部処理を記述する関数を作成します。以下のプログラムを適切な場所に追加しましょう。

```
/*===========================================================================
FUNCTION プロジェクト名_Proc

DESCRIPTION
    This is the InternalProcessing for this Game. All game processing to this app are handled in this
    function.

PROTOTYPE:
    void プロジェクト名_Proc(プロジェクト名* pMe)

PARAMETERS:
    pMe: Pointer to the AEEApplet structure. This structure contains information specific
    to this applet. It was initialized during the AEEClsCreateInstance() function.

DEPENDENCIES
  none

RETURN VALUE
  none

SIDE EFFECTS
  none
===========================================================================*/
void プロジェクト名_Proc(プロジェクト名* pMe)
{
}
```

※「プロジェクト名」は作成するプロジェクト名(半角英数字)に置き換えてください

(４)描画処理を記述する関数を作成します。以下のプログラムを適切な場所に追加しましょう。

```
/*===========================================================================
FUNCTION プロジェクト名_Draw

DESCRIPTION
    This is the DrawingProcessing for this Game. All game processing to this app are handled in this
    function.

PROTOTYPE:
    void プロジェクト名_Draw(プロジェクト名* pMe)

PARAMETERS:
    pMe: Pointer to the AEEApplet structure. This structure contains information specific
    to this applet. It was initialized during the AEEClsCreateInstance() function.

DEPENDENCIES
  none

RETURN VALUE
  none

SIDE EFFECTS
  none
===========================================================================*/
void プロジェクト名_Draw(プロジェクト名* pMe)
{
}
```

※「プロジェクト名」は作成するプロジェクト名(半角英数字)に置き換えてください

(５)タイマ起動を記述する関数を作成します。以下のプログラムを適切な場所に追加しましょう。

```
/*=====================================================================
FUNCTION プロジェクト名_SetTimer

DESCRIPTION
    This is the SetTimer for this Game. The timer to move this game at regular intervals is started.

PROTOTYPE:
    void プロジェクト名_SetTimer(プロジェクト名* pMe)

PARAMETERS:
    pMe: Pointer to the AEEApplet structure. This structure contains information specific
    to this applet. It was initialized during the AEEClsCreateInstance() function.

DEPENDENCIES
  none

RETURN VALUE
  none

SIDE EFFECTS
  none
=====================================================================*/
void プロジェクト名_SetTimer(プロジェクト名* pMe)
{
}
```

※「プロジェクト名」は作成するプロジェクト名(半角英数字)に置き換えてください

(６)(２)～(５)で作成した関数のプロトタイプを宣言します。以下のプログラムをコメント「Function Prototypes」の下に追加しましょう。

```
/*-------------------------------------------------------------
Function Prototypes
-------------------------------------------------------------*/
static  boolean プロジェクト名_HandleEvent(プロジェクト名* pMe, AEEEvent eCode,
                                           uint16 wParam, uint32 dwParam);
boolean プロジェクト名_InitAppData(プロジェクト名* pMe);
void    プロジェクト名_FreeAppData(プロジェクト名* pMe);

static void プロジェクト名_MainLoop(プロジェクト名* pMe);

void プロジェクト名_Proc(プロジェクト名* pMe);
void プロジェクト名_Draw(プロジェクト名* pMe);
void プロジェクト名_SetTimer(プロジェクト名* pMe);
```

※「プロジェクト名」は作成するプロジェクト名(半角英数字)に置き換えてください

(７)タイマを起動する処理を記述します。

タイマは、IShellインタフェースのISHELL_SetTimer関数で設定します。この関数を呼び出すと、タイマが設定されます。現在の時間から指定した時間(ミリ秒)経過後にタイマが作動し、指定した関数が実行されます。BREWのタイマは、１回しか作動しないので、タイマを繰り返すには、再度関数を呼び出し、再設定する必要があります。

以下のプログラムを適切な場所に追加しましょう。

```
ISHELL_SetTimer(pMe->pIShell, INTERVAL, プロジェクト名_MainLoop, pMe);
```

ISHELL_SetTimer関数の引数は、先頭から「①IShellインタフェース」(※インタフェースは次回説明します)、「②タイマ作動までの時間」、「③時間経過後に実行する関数」、「④引数③の関数に渡す変数のポインタ」です。上記の場合、定数INTERVAL経過後に、プロジェクト名_MainLoop関数が１回だけ実行されます。

(８)メインループを作成します。

メインループでは、以下の３つの処理を行います。
１．タイマの再設定
２．内部処理
３．描画処理

上記１～３が*プロジェクト名*_MainLoop関数で実行されるように、適切なプログラムを追加しましょう。

ヒント１：プログラムを追加するのは、*プロジェクト名*_MainLoop関数
ヒント２：(５)、(３)、(４)
ヒント３：３つの関数を呼び出しますが、それらの関数の引数は１つで、すべて「pMe」

(９)メインループが正しく実行されるか確認します。以下のプログラムを追加しましょう。

・ヘッダファイルの追加
```
#include <windows.h>
```

・*プロジェクト名*_HandleEvent関数－「case EVT_APP_START:」の下に追加
```
プロジェクト名_MainLoop(pMe);
```

・*プロジェクト名*_MainLoop関数の１行目に追加
```
OutputDebugString("*** Info - メインループ実行\n");
```

これらのプログラムの追加により、F5キーでデバッグ起動すると、*プロジェクト名*_MainLoop関数が実行されるたびに、デバッグウインドウに文字列が表示されるようになります。

※正しく動作することを確認したら、「#include <windows.h>」と「OutputDebugString("*** Info - メインループ実行\n");」は削除してください