Android ゲームプログラミング

○　第４回　メインループ　○

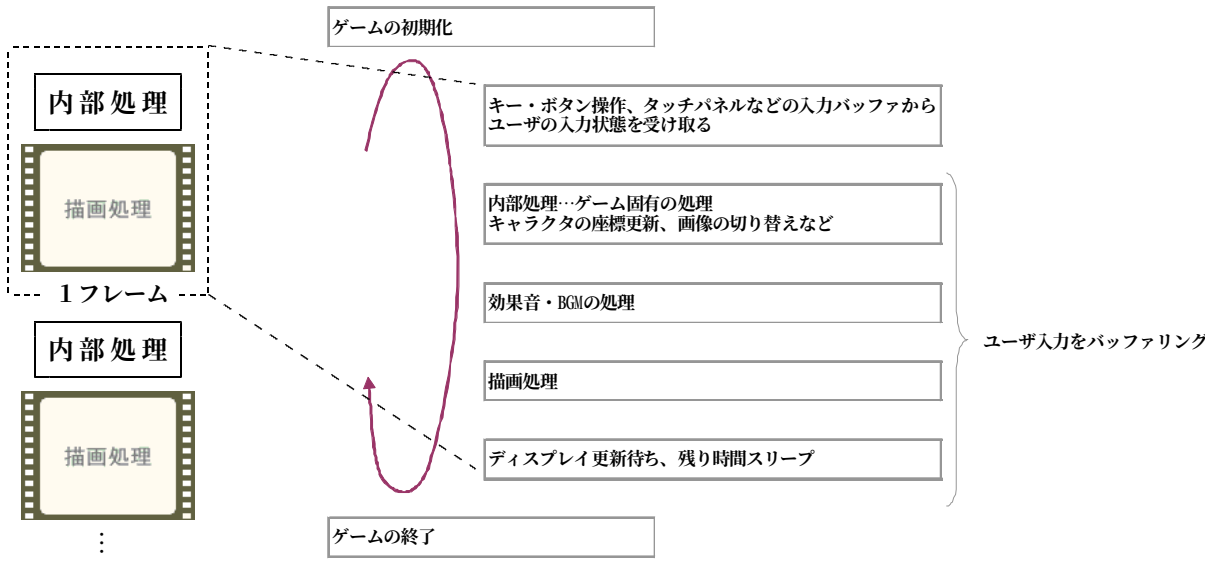
メインループ

・ゲームの仕組みはセルアニメーションと同じ
・１秒間に数十回程度のループを行う
・ループ内でゲーム固有の処理を行い、画面に描画する

概要

ゲームプログラムは、キャラクタや背景など、いろいろな「もの」の状態を時間にそって更新し、画面に描画するプログラムと考えることができます。これ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)
の２つにまとめることができます。この処理の繰り返しがメインループになります。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ます。

簡易図



フレームレート

・１秒間あたり何度画面が更新されるかを表す
・単位はfps(Frames Per Second)
・ディスプレイではリフレッシュレートと呼び、単位はHz

実例

・Androidは最大60fps ＝ 1ループは約16ms。
　各種処理に10ms→6msスリープ　　各種処理に20ms→スリープなし、処理落ち発生
・コンシューマ、アーケードは30または60fps
・携帯アプリは20～30fps
・映画は24fps、テレビアニメは8fps

メインループの実装

・メインループの実装方法は、画面の描画方法により異なる
・画面の描画は、View, SurfaceView, GLSurfaceViewを用いる３つの方法がある
・リアルタイムゲームでは、SurfaceViewまたはGLSurfaceViewを使う

概要

ゲームプログラムでは、ゲーム固有の処理と描画処理を定期的に繰り返し行うため、ループを構成します。
Androidでは、ループの構成方法は、画面に描画を行うクラスによって変わります。画面に描画ができるクラスには、Viewクラス、SurfaceViewクラス、GLSurfaceViewクラスの３つがあります。

View

Viewは、描画速度はそれほど速くありませんが、ユーザインタフェースの設計がしやすいため、ボタンやチェックボックスなどの部品を利用する各種ツール、速度を重視しないゲームに向いています。
Androidのもっとも基本的な描画を行う機能(クラス)で、SurfaceViewやGLSurfaceViewもこのクラスが基礎となっています。

SurfaceView

SurfaceViewは、2Dグラフィクスを高速に描画を行うための機能(クラス)です。高速描画が必要なリアルタイムゲームやマルチメディアなどに向いています。SurfaceViewはマルチスレッド対応のため、アプリケーションの処理(スレッド)とは別に独立した専用の描画を行う処理(スレッド)を作成することで、リアルタイムに処理を行うことができます。

GLSurfaceView

GLSurfaceViewは、組み込み機器向けの3DCGライブラリ"OpenGL ES"を利用し、3Dグラフィクスの描画を行うことができます。名前のとおり、SurfaceViewを継承して定義されており、SurfaceViewの3D版です。3Dだけではなく、工夫次第でSurfaceViewを超える速度で2Dグラフィクスの描画を行うこともできます。反面、プログラムが複雑になる傾向があります。

SurfaceViewの実装

・SurfaceViewクラスを継承したアプリケーション独自のクラスを定義する
・そのクラスでは、イベントを取得できるようにインタフェースSurfaceHolder.Callbackを実装する
・SurfaceHolder.Callbackの実装とは、３つのコールバックメソッドを定義すること
・専用のスレッドを作成し、スレッド内にゲーム固有の処理、描画処理などを記述する

概要

SurfaceViewを用いた描画を行うには、SurfaceViewクラスそのものを利用するよりも、継承した独自のクラスを定義する方法が一般的です。また、継承したクラスでは、SurfaceViewに起こったイベントを取得できるように、インタフェースSurfaceHolderのCallbackを実装します。実装といっても、SurfaceViewに起きた３つのイベントを処理するために、以下のメソッドを定義するだけです。

・surfaceCreatedメソッド　　SurfaceViewの生成時
・surfaceChangedメソッド　　SurfaceViewのサイズ等変更時
・surfaceDestroyedメソッド　　SurfaceViewの破棄時

これらのメソッドは、SurfaceViewに該当するイベントが起きた際、システムが自動的に呼び出すようになっています。
ゲームプログラムでは、SurfaceViewが生成されたということは「画面描画ができるようになった」ことになるので、このタイミングでゲームを初期化し、メインループを開始させます。また、アプリケーション終了の際などにSurfaceViewも破棄されますが、これは「画面描画ができない」ということになるので、このタイミングでメインループを終了させるようにします。

スレッドの利用

SurfaceViewは自由なタイミングで画面の描画を行うことができ、マルチスレッドにも対応しています(Viewはマルチスレッド非対応)。そのため、システムが自動的に生成するメインスレッド以外の(アプリケーションが生成した)スレッドからも、SurfaceViewに描画することができます。

SurfaceViewクラスは、基本的にイベントが発生したときしか動かないイベント駆動型となっているので、そのままではリアルタイムに動作させることができません。そこで、スレッドを利用します。

スレッドとは、１つのアプリケーション(プロセス)内でメモリを共有する小さな実行単位のことで、最低１つのスレッドが実行されています。複数のスレッドを実行させることをマルチスレッドといいます。スレッドを作成すれば、SurfaceViewのイベント処理にとらわれることなく、並列してゲーム固有の処理をリアルタイムに実行させ続けることができます。

Javaでは、標準の機能としてスレッドをサポートしています。Threadクラスを継承する方法とインタフェースRunnableを実装するという２つの方法でスレッドを定義することができます。

課　　題

SurfaceViewクラスを定義し、メインループを作成しましょう。

(１)SurfaceViewクラスを継承したアプリケーション独自のクラスを定義します。

１．メニューから「ファイル(F)→新規(N)→クラス」と選択します。
２．「新規 Java クラス」が表示されます。以下のように設定してください。

パッケージ(K):　　jp.android.*プロジェクト名*
名前(M):　　*プロジェクト名*SurfaceView
スーパークラス(S):　android.view.SurfaceView
※そのほかの項目はそのままでかまいません

３．設定できたら「完了(F)」ボタンをクリックします。

(２)コンストラクタを定義します。

SurfaceViewクラスを継承したクラスは、最低１つのコンストラクタを定義しないとコンパイルエラーになります。

コンストラクタは、クラスと同じ名前を持つ特殊なメソッドのようなもので、クラス型の変数(オブジェクト)が生成された際に１度だけ実行されます。変数の初期化や属性の設定などを行います。

１．メニューから「ソース(S)→スーパークラスからコンストラクタを生成(C)」を選択します。
２．ダイアログが表示されたら、以下のように"SurfaceView(Context)"だけにチェックを入れます。
☐SurfaceView(Context, AttributeSet, int)
☐SurfaceView(Context, AttributeSet)
☑SurfaceView(Context)
３．そのほかの項目はそのままで、設定できたら"OK"ボタンをクリックします。

(３)SurfaceViewのイベントを処理するメソッドを定義します。

イベントを処理することで、適切なタイミングでゲーム処理の開始や終了が行えるようになります。インタフェースSurfaceHolder.Callbackを実装することにより、イベントが起こると自動的に該当するメソッドが呼び出されるようになります。

１．SurfaceViewを継承したクラスの宣言を以下のように変更してください。

```
public class プロジェクト名SurfaceView extends SurfaceView
                                implements SurfaceHolder.Callback {
```

２．エラーが出るので訂正します。ソースエディタ左端の  をクリックしてください。
３．訂正候補が出ます。一番上の"'SurfaceHolder'をインポートします(android.view)"を選択してください。

４．さらにエラーが出るので訂正します。
・３．と同じように  をクリック→「実装されていないメソッドの追加」を選択
・メニューの「ソース(S)→メソッドのオーバーライド/実装(V)」を選択し"Callback"にチェックのどちらかを行うと３つのコールバック用メソッドのひな形が作成され、エラーが解除されます。

(４)メインループ用のスレッドを定義します。

(３)の状態では、SurfaceViewのイベントが起きたときのみプログラムが動作し、それ以外のときは休止しています。また、イベントしか処理できない状態です。そこで、スレッドを用います。

SurfaceViewはマルチスレッド対応のため、メインスレッド(SurfaceViewイベント処理スレッド)以外からも描画が行えます。スレッドを作成し、常に実行させ続けることにより、ゲームをリアルタイムに動かすことができます。

スレッドの定義方法は２つありますが、Threadクラスを継承する方法は、Javaでは１つのクラスしか継承できず、すでにSurfaceViewクラス継承済みのため、このままでは使えません。よって、インタフェースRunnableを実装する方法でスレッドを定義します。

１．SurfaceView継承クラスの宣言を以下のように変更してください。

```
public class プロジェクト名SurfaceView extends SurfaceView
                                implements SurfaceHolder.Callback, Runnable {
```

２．エラーが出るので訂正します。
・  をクリック→「実装されていないメソッドの追加」を選択
・メニューの「ソース(S)→メソッドのオーバーライド/実装(V)」を選択し"Runnable"にチェックのどちらかを行うとrunメソッドのひな形が作成され、エラーが解除されます。

(５)ゲームの処理を行うメソッドのひな形を作成します。

以下のメソッドを作成します。
・初期化……Initializeメソッド。ゲームを初期化する処理を記述します。
・内部処理…doUpdateメソッド。キャラクタの座標やゲームの状態を変更する処理を記述します。
・描画処理…doDrawメソッド。内部処理で変更された状態を画面に表示するための処理を記述します。
・休止……doSleepメソッド。フレームの更新を一定間隔で行うため、ループ１回(１フレーム)ごとにスレッドを休止させ、次の描画のタイミングを整えます。

以下のプログラムを適切な場所に追加しましょう。

```
// ゲーム初期化
private void initialize() {
}

// 内部処理
private void doUpdate() {
}

// 描画処理
private void doDraw() {
}

// 休止
private void doSleep() {
}
```

(６)スレッドの処理を作成します。

スレッドの実行が開始されると、(４)で作成したrunメソッドが自動的に呼ばれ、実行されます。このメソッドは、メイン処理から完全に独立していて、並列に実行されるのため無限ループを記述してもシステムに影響を与えることはありません。よって、ここにメインループを記述します。

runメソッドは、「ゲーム初期化処理」→「メインループ」→(必要ならば)「後処理」という流れになります。

１．ループのフラグ用変数を宣言します。以下のプログラムを適切な場所に追加しましょう。

```
private boolean    mLoop;
```

メインループは基本的に無限にループさせますが、Androidでは、スレッド内で無限にループさせるとアプリケーションを終了してもスレッドが残ったままになる場合があります。そこで、なんらかの形でループから抜けられるようにしておきます。

２．runメソッドを実装します。runメソッドを以下のように変更し、'?'を埋めて完成させましょう。

```
public void run() {
    ??????????();        // 初期化

    // メインループ
    while(?????) {
        ????????();      // 内部処理
        ??????();        // 描画処理
        ???????();       // 休止
    }
}
```

(７)内部処理を作成します。

１．スレッド間の同期用変数を宣言します。以下のプログラムを適切な場所に追加しましょう。

```
private final Object     mLock = new Object();
```

メインループと「スクリーンをタッチしたき」などのイベントは、並列に処理されるため、不具合が出ないようにするための変数です(詳細は第６回)。

２．doUpdateメソッドを以下のように変更しましょう。

```
// 内部処理
private void doUpdate() {

    synchronized(mLock) {
        // ここに入力処理を記述します
    }

}
```

(８)描画処理を作成します。

１．描画処理を行うには、SurfaceHolderが必要となります。以下のプログラムを適切な場所に追加しましょう。

```
private SurfaceHolder    mSurfaceHolder;
```

SurfaceHolderは、SurfaceViewの描画領域(サーフェス:Surface)を管理していて、画面への描画もSurfaceHolderをとおして行います。なお、SurfaceViewというのは「Surface機能を持っているView」のことで、SurfaceHolderはSurfaceそのものを管理している、ということです。

２．SurfaceHolderを取得します。SurfaceView継承クラスのコンストラクタを以下のように変更しましょう。

```
public プロジェクト名SurfaceView(Context context) {
    super(context);

    mSurfaceHolder = getHolder();
    mSurfaceHolder.addCallback(this);
}
```

(SurfaceViewクラスの)getHolderメソッドにより、SurfaceViewが持っているSurfaceを管理するSurfaceHolderを取得します。以後、SurfaceHolderを用いることにより、画面への描画が行えます。また、addCallbackメソッドは、渡したクラスにイベント処理用のコールバックメソッドがあることをシステムに通知します。コールバックメソッドは定義しただけでは使用されないので、addCallbackメソッドで登録する必要があります。

3．doDrawメソッドを以下のように変更しましょう。

```
// 描画処理
private void doDraw() {
    Canvas   canvas = null;
    try {
        canvas = mSurfaceHolder.lockCanvas();

        // ここに描画処理を記述します

    } finally {
        if(canvas != null)
            mSurfaceHolder.unlockCanvasAndPost(canvas);
    }
}
```

※エラーが出ますが、  →「'Canvas'をインポートします (android.graphics)」で解決します

SurfaceHolderのlock～unlockの間で画面に描画することができます。ロックするのは、画面はすべてのアプリケーションで共有しているので、ほかから画面に描画できないようにするためです。ロックを解除した時点で描画内容が反映されます。また、なんらかの理由でロックに失敗することもあるので、tryブロックで囲んでおきます。こうすると、エラー(例外)に強いプログラムとなります。
ロックをすると、Canvas型の変数が得られます。Androidの描画は、ほとんどがCanvasクラスのメソッドとなっています。ロック時に得られたCanvasから描画メソッドを呼び出すことによって、Canvas→ロックをしたSurfaceへと描画が行われます。

(９)休止処理を作成します。

画面の更新を一定の間隔で行うため、１フレームの処理にかかった時間を以下の式で求めます。

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

１フレームの処理時間に余裕が出た場合は、sleepメソッドで休止させます。休止時間は以下の式で求めます。

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)

1．フレームレートとフレームの間隔を定数で宣言します。以下のプログラムを適切な場所に追加しましょう。

```
public static final long   FPS      = 30;
public static final long   INTERVAL = 1000 / FPS;
```

2．フレームの終了時間を格納する変数を宣言します。以下のプログラムを適切な場所に追加しましょう。

```
private long   mLastTime;
```

3．doSleepメソッドを以下のように変更しましょう。

```
// 休止
private void doSleep() {
    long   sleepTime = INTERVAL - (android.os.SystemClock.uptimeMillis() - mLastTime);
    if(sleepTime >= 0) {
        try {
            Thread.sleep(sleepTime);
        } catch(Exception e) {
        }
        mLastTime = android.os.SystemClock.uptimeMillis();
    }
}
```

(10)スレッドを生成し、実行を開始する処理を作成します。
1．スレッドを管理する変数を宣言します。以下のプログラムを適切な場所に追加しましょう。

```
private Thread   mMainLoop;
```

２．スレッドを生成し、実行を開始させます。これは、サーフェスが生成されたタイミングで行います。以下のプログラムの'?'を埋めて完成させ、適切な場所に追加しましょう。

```
// メインループ開始
mLoop     = ????;
mMainLoop = new Thread(this);
mMainLoop.start();
```

(11)スレッドを終了させる処理を追加します。

スレッドは、サーフェスが破棄されたときに終了させます。基本的にサーフェス＝画面なので、サーフェスの破棄はアプリケーション終了を意味します。

以下のプログラムを適切な場所に追加しましょう。

```
// メインループを終了させる
mLoop = false;

// メインループが終了するまで待つ
boolean   retry = true;
while(retry) {
     try {
          mMainLoop.join();
          retry = false;
     } catch(InterruptedException e) {
     }
}
```

(12)SurfaceViewを生成し、画面として設定する処理を作成します。

(11)でSurfaceViewの骨組みは完成しましたが、「SurfaceViewの機能を持ったクラスを定義した」だけなので、このままでは使用されません。クラスの変数を生成し、画面として設定する必要があります。

*プロジェクト名*ActivityのonCreateメソッドを以下のように変更しましょう。

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    requestWindowFeature(Window.FEATURE_NO_TITLE);
    setContentView(new プロジェクト名SurfaceView(this));
}
```

※エラーが出ますが、インポートして訂正してください

setContentViewメソッドにより、カッコ内で生成したSurfaceViewが画面に設定されます。

(13)端末を回転しても反応しないようにします。

Androidはセンサ搭載のため、端末を回転させると向きにあわせて画面が回転します。このとき、いったんサーフェスが破棄された後、再生成されます。(12)までの状態だと再起動されることになります。また、縦画面用と横画面用の画像を準備するといった負担も発生します。
端末の向きにかかわらず、画面を固定したい場合は、以下のようにします。

１．"*プロジェクト名*Activity.java"のソースが表示されている状態で、メニューから「ソース(S)→メソッドのオーバーライド/実装(V)」を選択します。
２．"onConfigurationChanged(Configuration)"にチェックを入れ、"OK"ボタンをクリックします。
生成されたonConfigurationChangedメソッドは、そのままでかまいません。
３．"AndroidManifest.xml"の該当部分を以下のように変更してください。

```
<activity android:name=".プロジェクト名Activity"
          android:label="@string/app_name"
          android:screenOrientation="portrait"
          android:configChanges="orientation|keyboardHidden">
```

※横画面に固定したい場合は、"portrait"を"landscape"に変更してください。

以上で、端末を回転させても画面は固定されたままで再起動もしなくなります。エミュレータでは、"Ctrl+F11"で端末を回転させることができます。