

「Writing multiscreen AIR apps」の日本語訳

本ドキュメントは、Adobe サイトで公開されている記事「Writing multiscreen AIR apps」をヒム・カンパニー 永井勝則が自主的に日本語に訳したものです。

<http://www.himco.jp/>

[knagai@himco.jp](mailto:knagai@himco.jp)

(2010/10)

本ドキュメントの原文は

[http://www.adobe.com/devnet/air/flex/articles/writing\\_multiscreen\\_air\\_apps.html](http://www.adobe.com/devnet/air/flex/articles/writing_multiscreen_air_apps.html)

です。

## マルチスクリーン AIR アプリケーションの記述

Christian Cantrell

2010/6/10

### 必要な予備知識

本記事は、ActionScript 3.0 に習熟し、スプライトやビットマップといった一般的な Flash 概念を理解している読者を前提としています。また、Flash Platform によるマルチスクリーンのアプリケーション開発に習熟するには、本記事の前に「複数の画面サイズに適応できるモバイル Flash コンテンツのプログラミング」を読まれることを強くおすすめします。

### 必要な製品

Adobe AIR

Flash Player

Flash Builder

### ユーザーレベル

中級

### 追加として必要なもの

Adobe Flex SDK

### サンプルファイル

このプロジェクトのサンプルコードはすべてオープンソースで、下記リンクの[GitHub](#)に置いています。ソースコードは[git](#) (Linuxカーネルの開発に使われている分散バージョン管理システム)を使ってチェックアウトすることも、以下のプロジェクトページからZIPやTARファイルでダウンロードすることもできます。

- ・[Reversi](#)
- ・[ReversiAndroid](#)
- ・[ReversiBrowser](#)
- ・[ReversiDesktop](#)
- ・[ReversiIpad](#)
- ・[ReversiIphone](#)

訳者注:

この記事は具体的なマルチスクリーン AIR アプリケーションの記述方法または記述例ではなく、著者が作成した相当高度なマルチスクリーン AIR アプリケーションの要点の紹介です。

最近わたしは「[One Application, Five Screens](#)」という記事をわたしのブログに載せました。これは iReverse という名前の単体の AIR アプリケーションが OS X、Windows 7、Ubuntu、iPhone、iPad、Motorola Droid、そしてブラウザで動作することを示した記事です。その後、Android が AIR ツールと容易に結びつけられる例として、[Nexus One で動作する iReverse](#) を Adobe のブログで公開しました。Adobe AIR は、iPhone と iPad との以前のような関係は薄くなりましたが、ほかの大部分の次世代デバイスには進出しつづけていくので、複数のデバイスや画面サイズに対応するプログラミング方法を知っておく重要性は今後、どんどん増していきます。本記事では、あらゆる画面サイズと解像度に自動的に対応し、場所を選ばずに実行できる、単体のアプリケーションの記述にわたしが使ったテクニックを述べていきます。

訳者注:

その後、Apple の iPhone アプリケーションの開発規約緩和により、Adobe は Packager for iPhone の開発の再開を発表しました。

マルチスクリーンアプリケーションの開発には、いくつかの異なるテクニックがあります。わたしが検討したのは次の3つの事柄です。

1. アプリケーションは、デバイス特有のプレゼンテーションレイヤーをコアのアプリケーションロジックの最上位に配置できる方法で構成する。
2. 可能な限り多くの環境で再利用できるコンポーネントやライブラリを開発する。
3. あらゆる環境に自動的に対応できる、単一のコードベースを開発する。

初めの2つの選択肢については、ほとんどの開発者がその最善の方法を見出していくことでしょう。目的は、どこにでもデプロイできる単一のアプリケーションを記述することではなく、できるだけ多くのコードを再利用し、デバイス特有のコード量をできるだけ抑えることにあります。たとえば Twitter クライアントのデプロイでは、Twitter プロトコルのライブラリや永続レイヤー、おそらくほとんどのアプリケーションロジックを再利用しつつ、ターゲットプラットフォームやデバイスのプレゼンテーションレイヤーはカスタマイズしたいはずで

このモデルビューコントローラの方法は大部分のアプリケーションで有効だとは思っていますが、わたしは3つめの選択肢に挑戦することにしました。サポートされるすべてのプラットフォームとデバイスで、まったく変えることなく単一のコードベースによるアプリケーションが記述できるかどうかを確かめたかったのです。結果としていうと、少し練習すれば、思っていたよりも実用的だということが分かりました。

## このプロジェクトの本質

本プロジェクトについては以上ですが、少し時間を取ってこのプロジェクトの本質について述べておきます。「[One Application, Five Screens](#)」の公開後、わたしは、プロジェクトを誤解した開発者からのコメントを多く受け取りました。“Javaは 1995 年からこれができた”は論外として、2つめに大きな誤解は、わたしはifステートメントつまり#ifdefプリプロセッサディレクティブを使っているにすぎない、というものです。わたしに言わせるとこれを使うのは“ずる”です。このプロジェクトの 1500 行ほどのコードには、アプリケーションが動作するプラットフォームやデバイス、OSを調べるifステートメントはありません。iReverseは特定のデバイスに対応するのではなく、文脈や環境に対応するのです。言い換えると、iReverseはデバイスがMotorola DroidかWindows 7 コンピュータかを調べるのではなく、画面の解像度やPPIなどの事柄に注意を払うのです。実はわたしはもう1歩先まで踏み込んでいます。iReverseはデバイスの向きや画面の大小には関心がなく、多くの制約の中でもできるだけ最善のレイアウトを行おうとします。これはつまり、iReverseは現在サポートされる最小のデバイスでも最大のデバイス(すぐフラットパネルのテレビが登場するでしょう)でもちゃんと動作する、ということです。

## プロジェクトのアーキテクチャ

具体的な実装の詳細に進む前に、わたしが使用したアーキテクチャの概要から始めた方がよいでしょう。

iReverse は単一のコードベースですが、実行する各デバイス用のプロジェクトもあります。現行ではつぎのプロジェクトがあります。

- [Reversi](#)
- [ReversiAndroid](#)
- [ReversiBrowser](#)
- [ReversiDesktop](#)
- [ReversiIpad](#)
- [ReversiIphone](#)

Reversi は基本的に、ゲームに必要な全コードを含むプロジェクトで、個々のデバイスのプロジェクトはそのソースパスに Reversi プロジェクトを含む単純なラッパーに過ぎません(Flash Builder でプロジェクトのソースパス設定するには、[プロパティ]→[Flex ビルドパス]で行います)。Reversi プロジェクトにはおよそ 1500 行のコードがありますが、ReversiAndroid プロジェクトにはつぎのコードを書いているだけです。

```
package
{
    import flash.display.Sprite;
    import flash.display.StageAlign;
```

```
import flash.display.StageScaleMode;
import flash.events.Event;

[SWF(frameRate="24")]
public class ReversiAndroid extends Sprite
{
    private var reversi:Reversi;
    public function ReversiAndroid()
    {
        super();
        this.stage.scaleMode = StageScaleMode.NO_SCALE;
        this.stage.align = StageAlign.TOP_LEFT;
        this.reversi = new Reversi(265);
        this.addChild(this.reversi);
    }
}
}
```

ご覧のように、このラッパーの仕事は以下を行うことです。

- ・SWF メタデータを使って、フレームレートを特定させる
- ・Reversi クラスをインスタンス化する
- ・ステージを構成する
- ・Reversi インスタンスをステージに追加する

たったこれだけです。ではすべての iReverse ラッパーが基本的に同じことをするのなら、個別のプロジェクトをなぜ作成するのでしょうか？ それにはいくつかの理由があります。

・フレームレートがカスタマイズできる。わたしはまだ、デバイスごとの異なるフレームレート環境で実行していませんが、今後もサポートされるデバイスは増えていくので、フレームレートのカスタマイズは有効な選択肢となるでしょう。

・PPI (pixels per inch) がカスタマイズできる。Reversi クラスのコンストラクタに渡す引数-1 は PPI 値です。-1 を渡すと、Capabilities.screenDPI プロパティが PPI の決定に使用されます。環境によっては API はハードコーディングされており(デバイスの画素密度の取得が難しい環境があるので)、そのオーバーライドが役立つ場合があります。

訳者注:

原文では `this.reversi = new Reversi(-1);` となっていますが、実際にダウンロードした `ReversiAndroid.as` では `this.reversi = new Reversi(265);` になっています。

・アプリケーション記述ファイルを分ける。アプリケーション記述ファイルを使うと、初期ウィンドウのサイズ(デスクトップでのテスト用)や画面の向き、アイコン、デバイス特有のメタデータなどがカスタマイズできます。次のアプリケーション記述ファイルは `ReversiAndroid` のものです。

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<application xmlns="http://ns.adobe.com/air/application/2.0">
  <id>com.christiancantrell.reversiandroid</id>
  <supportedProfiles>desktop mobileDevice</supportedProfiles>
  <filename>iReverse</filename>
  <name>iReverse</name>
  <version>0.9</version>
  <description>A one or two player Reversi game.</description>
  <initialWindow>
    <autoOrients>true</autoOrients>
    <fullScreen>true</fullScreen>
    <aspectRatio>landscape</aspectRatio>
    <content>[This value will be overwritten by Flash Builder in the
output app.xml]</content>
    <title>Reversi</title>
    <systemChrome>none</systemChrome>
    <transparent>false</transparent>
    <visible>true</visible>
    <width>854</width>
    <height>480</height>
  </initialWindow>
  <icon>
    <image57x57>assets/Icon.png</image57x57>
  </icon>
</application>
```

Note: このコードでは、画面サイズとして Motorola Droid の解像度の 480 x 854 を設定しています。

Droidの解像度を指定することで開発時に1つのターゲットデバイスが“シミュレーション”できますが、デバイス自体のアプリケーションのサイズには影響しません。言い換えると、Nexus One(解像度は 400 x 800) 上でも、アプリケーションは適切なサイズで実行されます。

図1は、ここまで述べてきたアーキテクチャを示す、Flash Builder のプロジェクトパネルです。

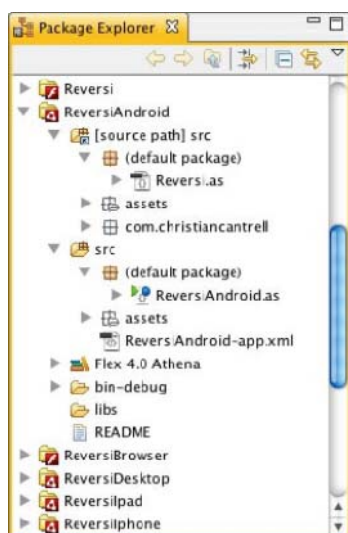


図1：ここまで述べてきたアーキテクチャ

### Reversi プロジェクトの設定

Reversi プロジェクトは以下のように設定します。これは必須ではありません。本記事を一般的な概念として参考にしたいだけの場合には、以下は飛ばしてください。

1. Flash Builder 4とAIR 2 SDKがインストールされていることを確認します。まだFlash Builder 4を持っていない場合は、Flex Builder 3でも動作するはずですが。

#### 訳者注：

Flash Builder 4のデフォルトのAIR SDKは1.5.3なので、最新のAIR SDK 2.0.2を含むFlex SDKの4.1.0をダウンロードし、Flash Builder 4で使用するSDKに設定する方法が一番簡単です。

2. 設定したいiReverseプロジェクトをダウンロードします。どのプロジェクトでもReversiプロジェクトは必要です。本記事ではReversiDesktopプロジェクトで進めます。

#### 訳者注：

Reversiのページ右上にあるダウンロードリンクをクリックすると、cantrell-Reversi-9b77be0.zipがダウン

ロードできます。これを展開すると cantrell-Reversi-9b77be0 フォルダが現れます。これをたとえば /Developer/ReversiTest フォルダにコピーします。同様に cantrell-Reversi-Desktop-636dfld フォルダと、cantrell-Reversi-Android-8733f3f フォルダもコピーします。

3. Flash Builder で、Reversi という名前の ActionScript プロジェクトを作成し、[プロジェクトコンテンツ]としてダウンロードした Reversi プロジェクトのディレクトリを指定します。これは src ディレクトリそのものではなく、src を含むディレクトリです。

訳者注:

すると[問題]パネルに AccelerometerEvent に関するエラーが表示されます。

4. AIR SDK にある airglobal.swc を Reversi プロジェクトに追加します。そのためには Reversi プロジェクトの[プロパティ]→[ActionScript ビルドパス]に進み、[ライブラリパス]タブをクリックします。[SWC の追加]ボタンをクリックし、AIR SDK 内の airglobal.swc まで移動し、選択します。

訳者注:

airglobal.swc は Flex SDK 4.1.0 の framework/libs/air にあります。airglobal.swc をライブラリパスに追加すると、AccelerometerEvent のエラーが消えます。

5. ReversiDesktop という名前で、新規の Flex デスクトッププロジェクトを作成し、ダウンロードした ReversiDesktop プロジェクトを含むディレクトリを指定します。
6. ReversiDesktop.as ファイルを右クリックし、[デフォルトのアプリケーションに設定]を設定します。自動生成された ReversiDesktop.mxml ファイルは削除できます。

訳者注:

この時点で、[問題]パネルに Reversi に関するエラーが表示されますが、次の手順7で消えます。

7. ReversiDesktop プロジェクトのソースパスに Reversi プロジェクトを追加します。そのためには ReversiDesktop プロジェクトで[プロパティ]→[Flexビルドパス]に進み、[ソースパス]タブをクリックします。[フォルダの追加]ボタンをクリックして、前にダウンロードした Reversi プロジェクト内の src ディレクトリまで移動して選択します。[OK]をクリックして変更を保存します。

ReversiDesktop がコンパイルされ適切に実行されます。ほかの Reversi プロジェクトについても手順は変わりません(開始は手順5からです)。ただし ReversiBrowser プロジェクトだけは異なるので、これを設定するには以下の手順にしたがいます。

1. ReversiBrowser プロジェクトをダウンロードします。
2. ReversiBrowser という名前の新規 ActionScript プロジェクトを作成し、ダウンロードした ReversiBrowser プロジェクトを含むディレクトリを指定します。
3. 前の手順7にしたがい、ReversiBrowser プロジェクトに Reversi プロジェクトのソースコードを追加します。

訳者注:

わたしの場合には、AccelerometerEvent のエラーが発生したので、ReversiBrowser プロジェクトにも airglobal.swc を追加しました。

4. Flash Player 10.1 かそれ以降がインストールされていることを確認します。
5. Flash Player 10.1 用の playerglobal.swc というファイルが必要です。これは Flex SDK4.1.0 の frameworks/libs/player/10.1 フォルダにあります。
6. [プロパティ]→[ActionScript コンパイラ]の[Adobe Flash Player オプション]で、[特定のバージョンを使用]を選択し、10.1.0 を設定します。[OK]をクリックします。

ReversiBrowser プロジェクトの[bin-debug]フォルダにある ReversiBrowser.html を開くと、デフォルトのブラウザで iReversi が起動します。

訳者注:

Flash CS5 の AIR for Android で ReversiAndroid を実行するには次のようにします。

- 1) Flash CS5 で AIR for Android の FLA ファイルを作成します。
- 2) FLA ファイルを ReversiAndroid プロジェクトの src フォルダ内に保存します。
- 3) FLA ファイルのドキュメントクラスに、ReversiAndroid を指定します。
- 4) FLA ファイルのソースパスに、Reversi プロジェクトの src フォルダを追加します。
- 5) コンピュータに Android 実機を接続し、パブリッシュします。

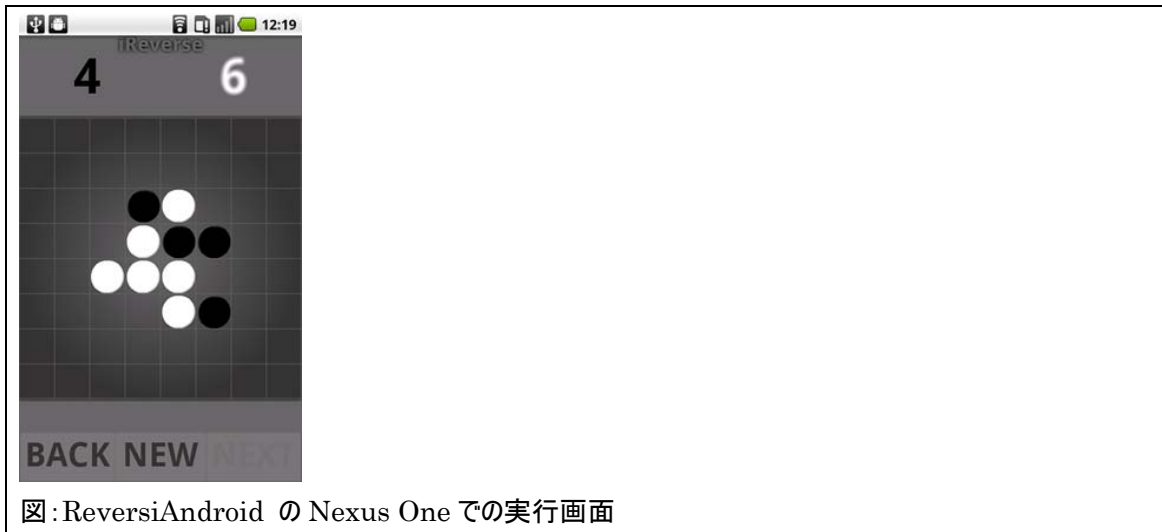


図: ReversiAndroid の Nexus One での実行画面

iReverse を設定し実行したところで、複数の画面サイズに対応するための具体的なテクニックを見ていきましょう。

#### 方向の変更の処理

iReverse は次の3つの方向をサポートしています。

- Portrait (縦長): アプリケーションの高さが幅よりも長い
- Landscape (横長): アプリケーションの幅が高さよりも長い
- Flat (平置き): 2 人で対戦するとき、デバイスを水平に置く

iReverse では向きについて十分に考慮されていますが、方向のイベントは使用しません。iReverse はデバイスの向きの変更に対し完璧に回答でき、AIR では方向の変更イベント (StageOrientationEvent.ORIENTATION\_CHANGE) の監視と回答が容易に行えますが、iReverse ではそれに関連する API は使用せず、ステージのリサイズイベント (Event.RESIZE) の監視というもっと汎用的でマルチスクリーンに適したテクニックを使っています。

方向の変更イベントではなく、ステージのリサイズイベントを使用する主なメリットには、アプリケーションが、デバイス上での回転であっても、ネイティブウィンドウのリサイズであっても、ブラウザ内のプラグインコンテナの任意のリサイズであっても、自分自身を適切にレイアウトする点にあります。

このテクニックを実現するには、Reversi クラスでまず、これが表示リストに追加されたタイミングを知る必要があります。表示リストに追加された後でないと、stage プロパティに安全にアクセスできません。次のコードは、stage プロパティが null でなくなるタイミングを iReverse が知る方法を示しています。

```
this.addEventListener(Event.ADDED, onAddedToDisplayList);
```

次のコードは、onAddedToDisplayList()関数の省略版です。

```
private function onAddedToDisplayList(e:Event):void  
{  
    this.removeEventListener(Event.ADDED, onAddedToDisplayList);  
    this.stage.addEventListener(Event.RESIZE, doLayout);  
}
```

ステージのサイズが変更されたときには、この doLayout()関数が呼び出されます。Event.RESIZE イベントが発射される環境には次の3つがあります。

1. アプリケーションが初期化されたとき
2. ウィンドウ(アプリケーションにウィンドウがある場合)がリサイズされたとき
3. デバイスの向きが変わったとき(適用可能な場合)

結果からいうと、われわれが必要とするのはこれです。アプリケーションは初期化時と、ウィンドウのサイズが変わったとき(ウィンドウを持つデスクトップなどの環境で)、そしてデバイスの向きが変わったとき(モバイルデバイスやタブレットでの実行中)に、それ自体をレイアウトし直す必要があります。

適切なイベントを取得しすべてのレイアウトの状況に対応する方法が分かったら、次はそのレイアウトのロジックです。無論これは個々のアプリケーションによって大きく変わりますが、以下の汎用的な事柄もあります。

#### すべての子の削除

最初に行いたいことの1つに、すべての子のステージからの削除があります。

```
while (this.numChildren > 0) this.removeChildAt(0);
```

#### 背景の描画

iReverse では次のコードを使って、背景の単色を描画しています。

```
var bg:Sprite = new Sprite();  
bg.graphics.beginFill(BACKGROUND_COLOR);  
bg.graphics.drawRect(0, 0, this.stage.stageWidth, this.stage.stageHeight);
```

```
bg.graphics.endFill();
this.addChild(bg);
```

iReverse では背景色として単色を使っているので、SWF のメタデータを使っても次のように設定できます。

```
[SWF(backgroundColor="#666666")]
```

しかしわたしは、次の理由から背景をコードから描画する方法を好みます。

- ・背景色がプログラミングから変更できる(これは、ユーザーが背景色を変更できる、ということも意味します)
- ・カラー値が、BACKGROUND\_COLOR などの定数としてアプリケーションに保持できる
- ・単色は容易に、グラデーションやイメージ、パターンと置き換えることができる

#### 方向の算出

iReverse は方向の変更イベントを使わないので、ゲームのレイアウト方法を知るには、デバイスの向きを知る必要があります。わたしはこれを簡単にするため、次の関数を書きました。

```
private function getOrientation():String
{
    return (this.stage.stageHeight > this.stage.stageWidth)
           ? PORTRAIT : LANDSCAPE;
}
```

デバイスの向きを調べる方法はほかにもありますが、これがもっともデバイスに依存しない方法です。言い方を変えると、これは方向の概念を持たないデバイス(デスクトップのような)でも動作します。ネイティブウィンドウのサイズが変わると、アプリケーションは適切に自分自身をレイアウトします。

#### アプリケーションの描画

iReverse のユーザーインターフェイスの描画はほとんどが次の条件内で行われます。

```
if (this.flat) // 対戦モード
{
    // 2人の対戦モードのときには、デバイスは平置き。
```

```
// スコアは各プレイヤーに正対して表示される
}
else if (this.getOrientation() == PORTRAIT) // 縦長
{
    // スコアは上部に、ゲームボタンは下部に配置
}
else // 横長
{
    // スコアは両端に、ボタンは隅に配置
}
```

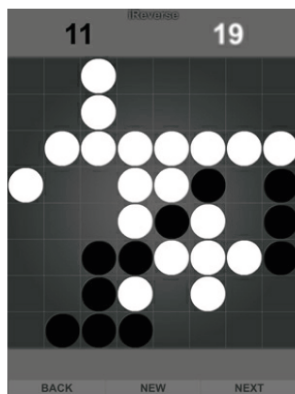


図2: 縦長 (portrait) モード

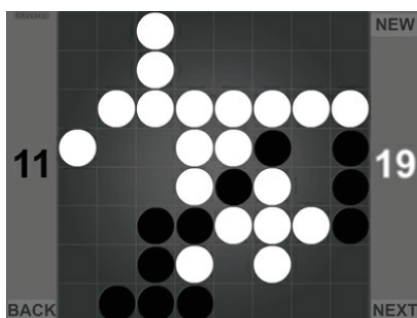


図3: 横長 (landscape) モード

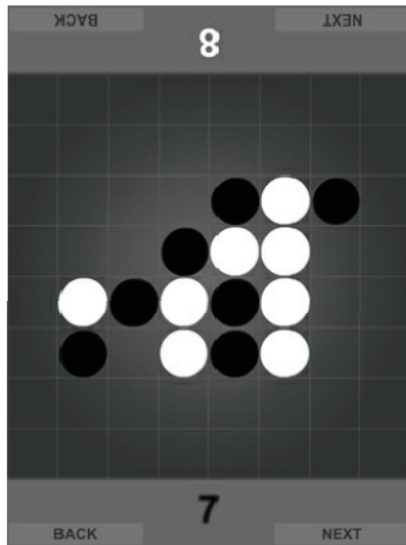


図4: 対戦 (flat)モード

#### サポートされない API の使用

iReverse は主にステージのリサイズイベント (Event.RESIZE) から再描画のタイミングを知りますが、また加速度センサーの更新イベント (AccelerometerEvent.UPDATE) を監視することで、“対戦”つまり“平置き”モードに移行するタイミングを知ります。もちろん iReverse を実行するすべてのデバイスが加速度センサーを備えているわけではなく、すべての AIR アプリケーションプロファイルが加速度センサーの API をサポートしているわけではありません (たとえばデスクトッププロファイルは、現時点で加速度センサーを備えたコンピュータは比較的少ないという理由から、Accelerometer クラスをサポートしません)。ではコードベースではどうやって、加速度センサーの有無を扱っているのでしょうか？

#### 訳者注:

プロファイルは AIR 2 で導入された、サポートされる API と機能のセットを定義する概念です。プロファイルにはデスクトップ、拡張デスクトップ、モバイルデバイス、拡張モバイルデバイスの4つがあります。

ラッキーなことに、Flash Platform ではこれを、サポートされない環境においても API を単に“もみ消す”ことで実現できます。言い換えると、加速度センサーのイベントがサポートされないデスクトッププロファイルでも、Accelerometer クラスは存在できるのです。これはわたしのコードもコンパイルされ検証され、問題なく実行できるということです。

しかしわたしは少し譲歩しています。次のコードは、Accelerometer インスタンスを設定し、加速度センサーの更新イベントの監視に使っているコードです。

```
if (Accelerometer.isSupported)
```

```
{
  this.accelerometer = new Accelerometer();
  this.accelerometer.setRequestedUpdateInterval(1500);
  this.accelerometer.addEventListener(
    AccelerometerEvent.UPDATE, onAccelerometerUpdated);
}
```

ご覧のようにここでは Accelerometer クラスの isSupported プロパティを使って、加速度センサーの API がサポートされているかを、API の使用前に調べています。実はこれは必須ではなく、if ステートメントをなくして、無条件で加速度センサーの更新イベントを登録することもできます。コードはコンパイルされ検証されて、問題なく実行されます。しかしここでは、どこでも動作するわけではない API を明確にしたいので、このようにしています。私見ではありますが、これによってコードが読みやすくなると思います。

#### 入力イベント

デスクトップや Web ブラウザアプリケーションのインタラクティブ操作は主にマウスとキーボードで行われるので、アプリケーションは通常、マウスイベントとキーボードイベントを捕捉します。モバイルアプリケーションのインタラクティブ操作は主に指で行われるので、モバイルアプリケーションは通常、タッチイベントを捕捉します。では、両方で動作するように設計するアプリケーションでは、入力イベントをどのように処理すればよいのでしょうか？

タッチイベントは、ホストシステムのハードウェアとソフトウェアがタッチをサポートする環境でのみ動作します。しかしこれはマウスイベントには当てはまりません。マウスイベントはデスクトップでもブラウザ内でも、そしてモバイルデバイス上でも動作します。したがってアプリケーションを複数の画面サイズで動作させたい場合には、通常はマウスイベントを捕捉し、タッチイベントは捕捉しないようにする方がうまくいきます。この例外は、タッチ特有のデータや、マルチタッチ、ジェスチャなどの機能、touchPointID などの TouchEvent クラスのプロパティが必要な場合です。しかしただ何かがタップされクリックされたタイミングを知りたいだけの場合には、マウスイベントがどこでも動作します。

iReverse ではマウスイベントを使用し、タッチイベントは使っていませんが、ユーザーが左右の矢印キーでゲームの履歴を巡回できるように、キーボードイベントを登録しています。次のコードはキーボードイベントに関して登録するコードです。

```
this.stage.addEventListener(KeyboardEvent.KEY_DOWN, onKeyDown);
```

次のコードは、左右矢印キーの押下げに応答します。

```
private function onKeyDown(e:KeyboardEvent):void
```

```

{
    switch (e.keyCode)
    {
        case Keyboard.RIGHT:
            this.onNext();
            break;
        case Keyboard.LEFT:
            this.onBack();
            break;
    }
}

```

ここではキーボードイベントのコードが、ホストデバイスにキーボードがついているかどうかすら関係していないことに注目してください。ここでも開発者は、Flash Platform によってプラットフォームごとの違いを気にせず済みます。キーボードのあるデバイスはキーボードイベントを投げます。キーボードのないデバイスはキーボードイベントを投げません。アプリケーションが、すべての場所で利用できるとは限らない機能に依存しない限り、これは、キーボードイベントをサポートする環境においてアプリケーションに追加機能を与えるすぐれた方法なのです。たとえば、Nexus One にはキーボードがないので、左右の矢印キーが動作しないことは明白です。しかしこれは D-pad を使った Droid (Droid に専用のキーボードをつないだ状態) では動作します。さらに言うと、タッチを感知する Windows 7 コンピュータでは、前後のジェスチャ(画面上の左または右へのスワイプ)は左右矢印キーのイベントが発生するので、ゲーム履歴の移動に使用できます。

#### ビットマップとベクター

Flash ではベクターもビットマップも使用できます。ビットマップを使ったアプリケーションのデザインとスタイリングは、デザイナーが使い慣れたツールやアセットが使用でき、ビットマップの方がトランジションやトゥイーン時のパフォーマンスがよくなるので、通常はすぐれたテクニックです。しかしビットマップには、いつもうまく縮小できるとは限らず、拡大にいたっては結果がまず不正確になるという問題があります。これはすなわち、アプリケーションを異なる画面サイズで使用できるようにしたいときには、ビットマップの埋め込みやロードは最良の方法でない場合もあるということです。

この点がベクターのグラフィックは多いに異なります。ベクターシェイプはアルゴリズムから計算されるので、さまざまな画面サイズに使用するために拡大縮小しても問題は発生しません。これはマルチスクリーンアプリケーションに理想的な特徴です。ベクターの問題は、適切に管理しないとリソースインテンシブになるということです。

iReverse ではこの中庸を行く方法を取っています。ゲームのすべてのグラフィック(石やゲーム盤、ボタンな

ど)は、どの画面サイズにも拡大縮小できるようにベクターから作成し、アプリケーションのパフォーマンスが向上するようにビットマップに変換しています。このベクターからビットマップへの変換には2つの異なるテクニックを使っています。

#### DisplayObject.cacheAsBitmap

すべての表示オブジェクト (iReverse のすべての視覚要素が継承する Sprite など) には cacheAsBitmap という名前のプロパティがあります。これを true にすると、ベクターはベクターからメモリ内のビットマップに変換されます。これは、トランジションやトゥイーンなどの視覚効果の状況ではパフォーマンスが向上することを意味します。しかしほとんどすべてのパフォーマンスに関するベストプラクティスとしては、cacheAsBitmap は有害無益の場合もあり得るということを覚えておく必要があります。たとえばわたしはゲームの石には cacheAsBitmap を使用していませんが、これは、cacheAsBitmap を true に設定した表示オブジェクトへのアルファトゥイーンの適用 (取った石を示すために使用しています) は実際には激しくパフォーマンスを落とすからです。表示オブジェクトは alpha プロパティが変わるごとに再キャッシュされます。とは言えオブジェクトを x や y、z プロパティを使って移動させる場合には、cacheAsBitmap を true に設定すると、パフォーマンスが著しく向上します。

#### BitmapData.draw()

iReverse の石は全部、ユーザーが目にするときにはビットマップになっていますが、実際にはベクターとしてスタートしています。ステージのサイズが変更されるときにはつねに、次の処理が発生します。

1. ステージのサイズのもとづいて、石の新しいサイズが計算されます。具体的に言うと、ゲーム盤のサイズを算出し (ステージの幅と高さの短い方の長さ)、それを 8 で割ります (オセロでは8行8列が必要です)。
2. 2つの“暫定的な”スプライトを作成します (黒石と白石用)。ActionScript の描画 API を使ってその中に円を描き、3次元風に見えるフィルタを適用します。
3. BitmapData.draw() を使って、2つのスプライトを BitmapData オブジェクトに描画します。
4. その BitmapData オブジェクトを使って、2つの新しい Bitmap オブジェクトを作成します。これはクラスレベルの変数として保持します。
5. Bitmap オブジェクトの新しいインスタンスを、ゲーム盤に置く必要があるときに石として作成します。

次のコードは、石のベクターをビットマップに変換します。

```
var cellSize:uint = (this.board.width / 8);
var stoneSize:uint = cellSize - 4;
var tmpStone:Sprite = new Sprite();
tmpStone.graphics.beginFill(WHITE_COLOR);
```

```
tmpStone.graphics.drawCircle(stoneSize/2, stoneSize/2, stoneSize/2);
tmpStone.graphics.endFill();
tmpStone.filters = [this.stoneBevel];
var tmpStoneBitmapData:BitmapData = new BitmapData(
    tmpStone.width, tmpStone.height, true, 0x000000);
tmpStoneBitmapData.draw(tmpStone);
this.whiteStoneBitmap = new Bitmap(tmpStoneBitmapData);
```

ここではベクターのメリット(たとえばベクターは画質をまったく落とさずに、どのサイズにも拡大縮小できます)と、ビットマップのパフォーマンスのメリット(速いフレームレートでもアルファトウインを維持できます)を最大限生かしています。

#### データの持続的な保持

ほとんどのアプリケーションでは、データを持続的に保持する必要があります。非常に簡易的なゲームでも、何らかの理由でアプリケーションを閉じる必要がある場合には、アプリケーションの現在の状態を保存すべきです。

Adobe AIR では、データを持続的に保持するための4つの仕組みが提供されています。

- ・ファイルシステム: AIR のファイル API を使用すると、アプリケーションの状態をファイルシステムに保存できます。
- ・SQL API: AIR ランタイムにはその内部に SQLite が埋め込まれています。これは、情報の保持や分類、取得にデータベースが非常に効率的に使用できるということです。
- ・EncryptedLocalStore: EncryptedLocalStore (暗号化されたローカルストア、AIR で使用できるクラス)は、扱いに慎重を要するデータの保持と取得を行うための非常に安全な方法です。
- ・SharedObject: 共有オブジェクト(今の場合には、“ローカル共有オブジェクト”と呼ばれます)はシリアル化されたデータを保持し取得するための簡単で速い方法を提供します。

どのテクニックを使用するかは、以下の事柄を考慮して判断します。

- ・データはどの程度セキュアにする必要があるのか? セキュリティで保護する必要がある場合には、EncryptedLocalStore や暗号化された SQLite データベースの使用を検討します。
- ・保持する必要があるデータはどれだけなのか? データ量が多いときには SQLite テーブルやファイルとして保存するようにします。
- ・データを整理分類する必要があるのか? 多量の複雑なリレーショナルデータはほぼ間違いなく SQLite テーブルに保存する必要があります。ゲームの状態のような複雑でないデータはフラットファイル(通常のプレ

ーンテキストのファイル)やローカル共有オブジェクトに保存できます。

・アプリケーションはどのような環境で実行するのか？ ブラウザのプラグイン(つまりブラウザ用の Flash Player)には SQLite API は含まれず、モバイルの AIR ランタイムでは EncryptedLocalStore はサポートされないので、アプリケーションを動作させたい場所によって、データを保持する方法が決まってきます。

iReverse は場所を選ばず実行し(デスクトップ、ブラウザ、電話、タブレット、そして最終的にはテレビや家庭用通信端末(セットボックス)などのデバイスで)、iReverse で保存する必要のあるデータは比較的単純なので、わたしはどこでも作業が行えるローカル共有オブジェクトを選びました。

iReverse は次の情報を保存します。

- ・ゲームの全体的な履歴(プレイヤーがオセロの差し手を戻れるように)
- ・ゲームのモード(1人プレイか2人プレイか)
- ・今の差し手はどっちか
- ・コンピュータが使用している石は白か黒か(ユーザーが1人プレイ(コンピュータとの対戦モード)でプレイしている場合)

これらの情報はすべて、何らかの理由でゲームがいつ終了されてもかまわないように、一手ごとに保存され、その状態は完全に復元されます。データの保持にはローカル共有オブジェクトを使っているため、すべてのデバイスでまったく同じように動作します。

以下はゲームの状態を保存するコードです。

```
private function saveHistory():void
{
    ++this.historyIndex;
    var historyEntry:HistoryEntry = new HistoryEntry();
    historyEntry.board = this.deepCopyStoneArray(this.stones);
    historyEntry.turn = this.turn;
    this.history[this.historyIndex] = historyEntry;
    for (var i:uint = this.historyIndex + 1; i < 64; ++i)
    {
        this.history[i] = null;
    }
    this.so.data[HISTORY_KEY] = this.history;
    this.so.data[PLAYER_MODE_KEY] = this.playerMode;
```

```
this.so.data[COMPUTER_COLOR_KEY] = this.computerColor;
this.so.flush();
}
```

わたしの選んだ持続性の例は主にモバイルに向いています。というのもモバイルアプリケーションはいつ閉じられるか分からず、アプリケーションの状態の変更後、できるだけ早くその状態を保存する必要があるからです。たとえばモバイルプラットフォームには、アプリケーションの切り替えでそのアプリケーションを閉じるものや、メモリ残量が少なくなってきたらアプリケーションを閉じるものがあります。また電話がかかってきたときやテキストメッセージに応答するときには、ゲームは一時停止(または終了)されるかもしれず、そもそも電話のバッテリー切れはつねに突然やってきます。わたしが選んだ持続性の方法はモバイルプラットフォームに適していますが、デスクトップやブラウザでもちゃんと動作することに、正直わたしは驚きました。デスクトップのアプリケーションが突然シャットダウンすることはそうそうありませんが、不意の終了やバックグラウンドでのほかのもののインストール後の再起動に備えてゲームを保存しておくにこしたことはありません。

相手がコンピュータのときのコード

相手がコンピュータのときに考慮しなければならなかったことの1つは CPU の使用量です。わたしは、コンピュータが数手先の手を全部計算する方法を考えていましたが、数学の天才でなくても、その結果の数が、特に最初の段階では膨大な数になることは容易に想像できます。コードはデスクトップでは問題なく実行できると思っていましたが、わたしは、もっと遅いモバイルのプロセッサではどのくらい落ちるのか分かりませんでした。

iReverse を記述する要点は先進的な AI を構築することではないので、わたしはどのプロセッサでもうまく動作することが分かっている非常に単純な方法を取ることにしました。コンピュータはまず最も戦略的な手を取り(自分のカラーの石の数が最も増える手のことか?)、必要に応じて順に戦略的でなくする方法です。このロジックは再帰的でなく計算にパワーが注ぎ込まれないので、わたしがテストしたどのデバイスでも、コンピュータは非常に短い時間で次の一手を選択できました。実はユーザーテストを始めたとき(友達や家族にコンピュータの相手をしてもらいました)、コンピュータは相手に失礼なほど早く次の手を打つことが分かったので、わたしはコンピュータの差し手を1秒遅らせました。数秒、場合によっては数分も時間をかけて次の一手を打った後、コンピュータが即座に石の最大数を計算するのはプレイヤーにとって面白いはずがありません。

シングルの 400MHz プロセッサからクアッドコアプロセッサまで、さまざまなデバイスで動作するアプリケーションを記述するときには、アプリケーションの視覚的な拡大縮小と同じように、計算にかかる負荷も拡大縮小して考える必要があります。以下はそのときに考慮すべき事柄です。

- ・計算に集中的に負荷がかかるロジックは、デバイスごとにカスタマイズできるように、ほかのコードと切り離す

ようにする。たとえば、ComputerPlayer インターフェイスを記述して、それを Reversi クラスのコンストラクタに渡すこともできます。それによってデスクトップではもっと上級の実装を記述し、パワーの劣るデバイスにはもっと単純で速いものが記述できます。

- ・プレイヤーに選択肢を与える。AI の記述にもう少し時間があれば、わたしはおそらく初級モードと上級モードを作り出して、プレイヤーにその選択権を与えたでしょう。上級の AI オプションを選択したユーザーには、コンピュータが次の一手にかかる時間をもっと増やしても平気でしょう。

- ・重たい処理はサーバーで行う。非常にプロセッサインテンシブな作業は、とても似つかわしくないデバイスではなく、サーバーに行わせることを検討します(言い換えると、今時のデバイスの多くは、CPU サイクルよりも帯域幅に余力を取っておく必要があります)。たとえば Google のモバイルデバイスでの音声認識技術は、iPhone の Dragon Dictation アプリケーションと同様、実際にはそのクライアント上ではなく、サーバーで処理を行っています。