ここから、Androidのアプリを作る上で、気をつけなくてはいけないことの一つを学ぶことができます。つまり、Androidアプリは、開発者が意図しないタイミングで勝手に終了させられる可能性があるのです。そのため、pauseやresumeといった端末の状態を監視して、自動保存を実装する必要があります。

#04-03 加速度センサーで玉転がし

Androidアプリが面白いのは、様々なセンサーを利用したアプリを作ることができる点にあります。ここでは、Androidの加速度センサー /傾きセンサーを利用して、玉転がしを行うアプリを作ります。

表 04-011	ここで作るアプリについて
アプリ名	加速度センサーで玉転がし
説明	玉をコロコロ転がして遊ぶもの
プロジェクト名	AccelBall

POINT // 04-003

この項目のアプリ作成を通じて確認するポイント

・傾きセンサーの使い方

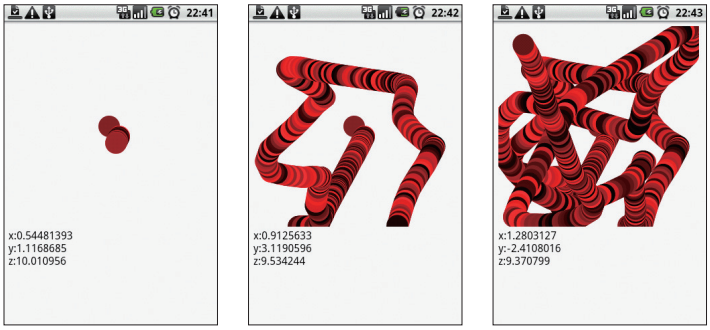
作成するアプリの紹介

この項目では、以下のようなアプリを作成します。

1.玉転がしして軌跡を楽しむ

加速度センサーを利用した「玉転がし」を作ってみます。ただ転がすだけではあまり面白味がないので、転がした軌跡を描くようにしてみました。アプリが開始されたら端末を上下左右に動かしてみてください。コロコロと玉が転がり、玉の動いた軌跡を楽しむことができます。

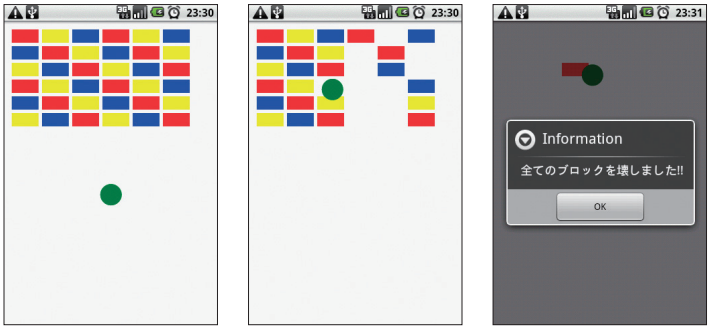
| 図 04-011 |



2.玉転がしでブロック崩し

玉転がしを改造して、ブロック崩しの要素を入れてみました。これは玉を転がしてブロックを壊すというゲームです。短時間のうちにブロックすべてを壊すことができるでしょうか。

| 図 04-012 |

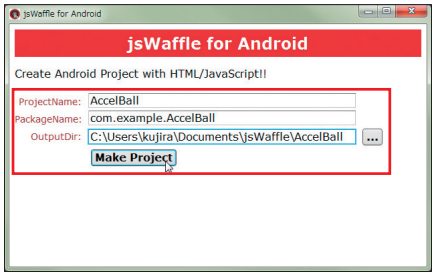


開発手順を確認しよう！

jsWaffleを利用してAndroidアプリのひな型を作り、それを元にして新規Androidプロジェクトを作成します。手順に沿って操作して、アプリを完成させましょう。

STEP_01

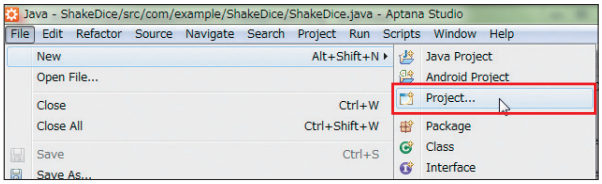
「jsWaffle」を起動したら、必要な項目を記入します。ここでは、以下の表のように項目を入力してください。そして、［Make Project］のボタンをクリックします。



ProjectName：AccelBall
PackageName：com.example.AccelBall
OutputDir：<マイドキュメント>AccelBall

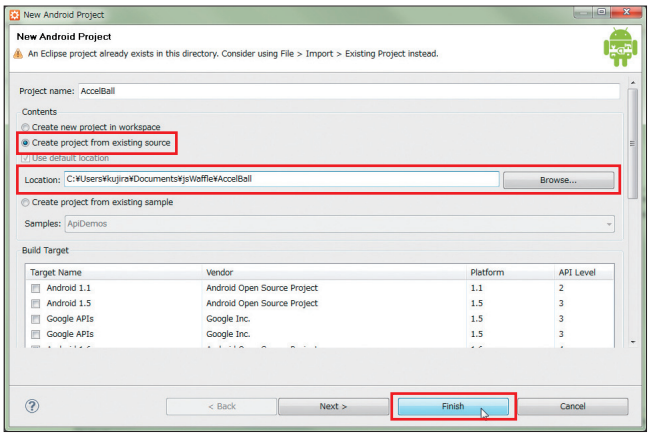
STEP_02

Aptanaを起動したらメインメニューより、[File > New > Project...]をクリックします。続いて、[Android > Android Project]を選択して[Next]をクリックします。



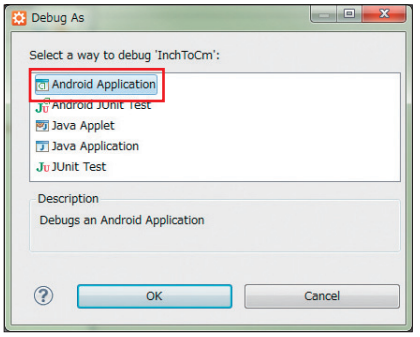
STEP_03

[New Android Project]のダイアログが表示されたら、[Create project from existing source]を選択して、STEP_01の「Output Dir」に指定したディレクトリを入力します。すると、Project nameなど自動的に入力されますので、[Finish]ボタンをクリックします。



STEP_04

プロジェクトが正しく作成されているかテストするためにデバッグ実行してみます。デバッグボタンをクリックし、その後表示されるダイアログで[Android Project]クリックします。うまくjsWaffleのデモプログラムが表示されれば問題ありません。



STEP_05

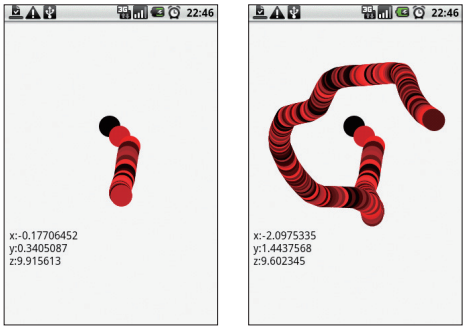
画面左上側にあるPackage Explorerでメインプログラム「assets/www/index.html」を表示したら、以下のようにプログラムを書き換えます。まずは、基本的なHTMLだけを記述します。画面に描画を行うために、HTML5の<canvas>タグを配置しています。

| スクリプト 04-026 | サンプルファイル:assets/www/index.html

```
<!DOCTYPE html>
<html><head><meta charset="utf-8">
<script type="text/javascript" src="jsWaffle.js"></script>
<script>
window.onload = function() {
    var a_canvas = document.getElementById("a_canvas");
    var context = a_canvas.getContext("2d");
    var canvasW = a_canvas.width;
    var canvasH = a_canvas.height;
    var ballX = 150, ballY = 150;  // ボールの初期配置位置
    var dirX = 0, dirY = 0;        // XY方向におけるボールの移動量
    setInterval(drawBall, 100);    // 定期的にボールを移動させ描画する
    function drawBall() {
        // ボールを描画
        context.beginPath();
        context.arc(ballX, ballY, 16, 0, 2*Math.PI, true);
        context.fillStyle="rgb("+Math.floor(Math.random()*256)+",0,0)";
        context.fill();
        // ボールを移動させる
        var tx = ballX + dirX;
        var ty = ballY + dirY;
        if (0 < tx && tx < canvasW) { ballX = tx; }
        if (0 < ty && ty < canvasH) { ballY = ty; }
    }
    // 傾きセンサーを監視
    droid.watchAccel(function(x,y,z){
        dirX = x * -1;
        dirY = y;
        $("x").innerHTML = "x:" + x;
        $("y").innerHTML = "y:" + y;
        $("z").innerHTML = "z:" + z;
    });
};
</script></head><body>
    <canvas id="a_canvas" width="300" height="300"></canvas>
    <div id="x"></div>
    <div id="y"></div>
    <div id="z"></div>
</body></html>
```

STEP_06

メニューのデバッグボタンの右側にある[▼]から[AccelBall]を選んでクリックすると、エミュレーター(あるいは実機)でプログラムが実行されます。加速度センサーを使ったアプリなので、できれば実機で試して楽しんでください。端末を傾けると、コロコロと玉が色を変えて転がり、その軌跡が描かれます。



端末を上下左右に動かすと玉が転がります

プログラムの解説～加速度センサーで玉転がし

今回作ったのは、Androidの加速度センサー(傾きセンサー)を使ったアプリです。jsWaffleで加速度センサーを使うには、以下の2つのメソッドを利用します。

表 04-012 傾きセンサー利用で使うメソッド

droid.watchAccel(callback)	加速度センサーの監視を開始
droid.clearAccel()	加速度センサーの監視を中止

使い方は簡単です。加速度センサーからの値を取るには「droid.watchAccel()」メソッドを一回呼ぶだけです。センサーから得られた値は、引数に指定した関数で得ることができます。

以下は、簡単な利用例です。これを実行すると、加速度センサーの値がリアルタイムに数値で表示されます。

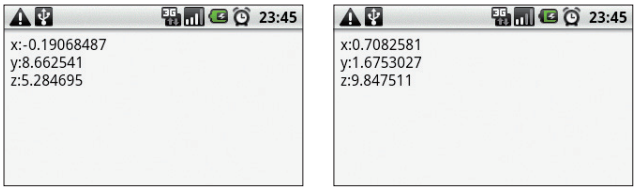
スクリプト 04-027 | サンプルファイル:assets/www/accel-test.html

```
<div id="info"></div>
<script type="text/javascript" src="jsWaffle.js"></script>
<script>
    // 加速度センサーの監視を開始
    droid.watchAccel(showInfo);
    // センサーから値が得られ時に呼ばれる関数
    function showInfo(x, y, z) {
        $("info").innerHTML =
            "x:" + x + "<br/>" +
```

```
            "y:" + y + "<br/>" +
            "z:" + z;
    }
</script>
```

Android端末で実行してみると、次々とセンサーから得られた値が表示されます。

| 図 04-013 |



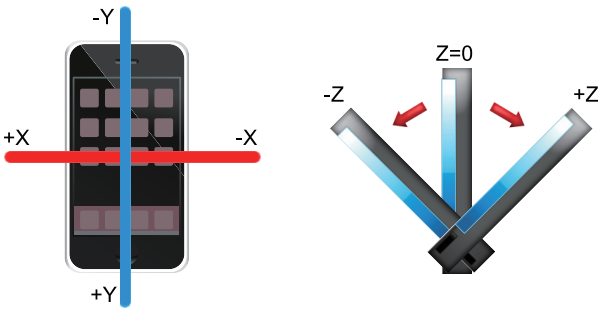
また、加速度センサーとして呼ばれる関数を無名関数として記述すると、より単純に加速度センサーの処理を記述できます。以下のソースコードは、上のものと全く同じ動きをするものですが、無名関数を使って書き直したものです。こちらの書き方の方が、よりJavaScriptらしい書き方と言えるかもしれません。

| スクリプト 04-028 | サンプルファイル:assets/www/accel-test2.html

```
<div id="info"></div>
<script type="text/javascript" src="jsWaffle.js"></script>
<script>
    droid.watchAccel(function (x, y, z) {
        $("info").innerHTML =
            "x:" + x + "<br/>" +
            "y:" + y + "<br/>" +
            "z:" + z;
    });
</script>
```

得られる値は、X、Y、Z方向の三種類で、次の図のようになっています。

| 図 04-014 | 傾きセンサーの値の図



つまり、端末の動きと値を表にすると次のようになります。

表 04-013 端末に各種アクションを加えた時の傾きセンサーの値

アクション	X	Y	Z
液晶を上向きに置く	0	0	10
液晶を下向きに置く	0	0	-10
頭を上に左向きに立てる	10	0	0
頭を上に右向きに立てる	-10	0	0
頭を起こして立てる	0	10	0
頭を下にして立てる	0	-10	0

玉転がしのプログラムを確認してみよう

加速度センサーの使い方が分かったところで、今回の玉転がしのアプリのプログラムをちょっとずつ確認していきましょう。仕組みとしては、Canvasにボールを描画し、加速度センサーの値に応じて、それを動かすというものです。

プログラムの前半では、変数などの初期設定を行っています。Canvasに描画するためにコンテキストを取得したり、ボールの配置位置や、移動量を表わす変数を宣言しています。

| スクリプト 04-029 |

```
01 window.onload = function() {
02     var a_canvas = document.getElementById("a_canvas");
03     var context = a_canvas.getContext("2d");
04     ...
05     var ballX = 150, ballY = 150;  // ボールの初期配置位置
06     var dirX = 0, dirY = 0;        // XY方向におけるボールの移動量
```

次に、タイマーで定期的にボールの描画を行うように設定しています。ここでは、setInterval()関数を使って、100ミリ秒に一回画面を書き換えるようにしています。

```
    setInterval(drawBall, 100);    // 定期的にボールを移動させ描画する
```

そして、ボールの描画を行う「drawBall()」関数では、次のようにボールの描画とボールの移動を行う処理を記述しています。

| スクリプト 04-030 |

```
01    function drawBall() {
02        // ボールを描画
03        context.beginPath();
04        context.arc(ballX, ballY, 16, 0, 2*Math.PI, true);
05        context.fillStyle="rgb("+Math.floor(Math.random()*256)+",0,0)";
06        context.fill();
07        // ボールを移動させる
08        var tx = ballX + dirX;
09        var ty = ballY + dirY;
10        if (0 < tx && tx < canvasW) { ballX = tx; }
11        if (0 < ty && ty < canvasH) { ballY = ty; }
12    }
```

ここで気になるのは、ボールの色を指定しているfillStyleプロパティでしょうか。ここでは「rgb(赤,緑,青)」の形式で各数値を指定しているのですが、赤の部分の割合を、0-255までの乱数で指定しています。これにより、ボールの軌跡が描かれたときに縞々の模様を楽しめるようになっています。

ところで、fillStyleプロパティには様々な形式で色を指定できます。次の表は色の指定方法を表にしたものです。塗りの色を指定する fillStyleプロパティに加えて、枠線（ストローク）の色を指定するstrokeStyleプロパティにも同様の形式で指定できます。

表 04-014 色の指定方法

色の指定方法	利用例
色の名前を直接指定	context.fillStyle = "red";
HTMLの色指定	context.fillStyle = "#FF0000";
RGBを数値で指定	context.fillStyle = "rgb(255,0,0)";
RGBAを数値で指定	context.fillStyle = "rgba(255,0,0,0.5)";

ここで疑問に思うのは、RGBAの形式かもしれません。これは、RGB(赤,緑,青) に加えて、A（Alpha）つまり透明度を0から1.0までの実数で指定するものです。rgba(255,0,0,0.5)というのは赤色を半透明にして背景に重ね合わせて描画します。

参考までに、玉転がしのボールの色を指定する部分を、次のように半透明な色でボールを描画するように改造してみましょう。

| スクリプト 04-031 |

```
01 var r = Math.floor(Math.random()*256);
02 context.fillStyle="rgba(" + r + ",0,0,0.1)";
```

すると、以下のようになり、これもまた面白い色合いを楽しむことができます。

| 図 04-015 |

TIPS // 04-002

グラデーションを色に指定したい場合は

ここでは紹介できませんが、コンテキストのfillStyleプロパティは他にもグラデーションの色を指定することもできるようになっています。興味がある方は、検索エンジンで「HTML5 Canvas fillStyle グラデーション」などのキーワードで検索してみましょう。

そして、玉転がしのプログラムの最後で加速度センサーの監視を行うように、「droid.watchAccel()」メソッドを呼び出しています。ここでは、ボールの移動する向きと移動量を表わす変数「dirX」と「dirY」にセンサーのXとYの値を設定しています。ちなみに、「dirX」の値を「x * -1」と書いているのは、センサーのX方向の値がAndroid端末を左に傾けると正数になり、右に傾けると負数になったためです。これは、Canvasの座標系で、左上の座標が基点(0,0)となるためです。端末の傾きと、X/Y/Zの値の動きを上の傾きセンサーの値の図で見ながら試してみてください。

| スクリプト 04-032 |

```
01      // 傾きセンサーを監視
02      droid.watchAccel(function(x,y,z){
03          dirX = x * -1;
04          dirY = y;
05          $("x").innerHTML = "x:" + x;
06          $("y").innerHTML = "y:" + y;
07          $("z").innerHTML = "z:" + z;
08      });
```

玉転がしをブロック崩しに改造してみよう

ところで、加速度センサーの利用用途として思いつくのはゲームが多いかもしれませんね。端末を振る、傾けるなど実際の動作を加えた体感的なゲームは直感的でとても楽しいものです。では、玉転がしを楽しんだところで、ブロック崩しの要素を加え、ちょっとしたゲームに仕上げてみましょう。

プロジェクトの「assets/www/index.html」を以下のプログラムに書き換えて実行してみてください。ゲームなので、ちょっとソースコードが長めですが、前項の玉転がしとそれほど違わないので頑張って眺めてみてください。

| スクリプト 04-033 | サンプルファイル:assets/www/index2.html

```
<!DOCTYPE html>
<meta charset="utf-8">
<script type="text/javascript" src="jsWaffle.js"></script>
<canvas id="a_canvas" width="300" height="300"></canvas>
<script>
window.onload = function() {
    var a_canvas = document.getElementById("a_canvas");
    var context = a_canvas.getContext("2d");
    var canvasW = a_canvas.width;
    var canvasH = a_canvas.height;
    var ballX = 150, ballY = 150; // ボールの初期位置
    var dirX = 15, dirY = 15;     // ボールの速度
    var blocks = [];              // ブロックを管理する配列変数
    var blockW = 40, blockH = 20; // ブロックの大きさを定義
    var blockPad = 5;             // ブロック同士の余白を定義
    var blockColors = ["red","blue","yellow"]; // ブロックの色
    // 各種初期化
    initBlocks();                  // ブロックを配置する
    setInterval(moveBall, 100);    // タイマーでボールの移動
    droid.watchAccel(function(x,y,z){ // 加速度センサーを監視
        dirX = x * -1; dirY = y;
    });
    // ブロックを配置する
    function initBlocks() {
        for (var y = 0; y < 6; y++) {
            for (var x = 0; x < 6; x++) {
                var color = blockColors[(x*5+y) % 3];
                blocks.push({
                    "x" : (x * (blockW + blockPad) + blockPad),
                    "y" : (y * (blockH + blockPad) + blockPad),
                    "color": color
                });
            }
        }
    }
```

```
    // ボールの移動と描画とブロックの衝突判定
    function moveBall() {
        context.clearRect(0,0,300,300); // 画面初期化
        // ブロックを描画
        for (var i = 0; i < blocks.length; i++) {
            var b = blocks[i];
            context.beginPath();
            context.rect(b.x, b.y, blockW, blockH);
            context.fillStyle = b.color;
            context.fill();
        }
        // ボールを描画
        context.beginPath();
        context.arc(ballX, ballY, 16, 0, 2*Math.PI, true);
        context.fillStyle="green";
        context.fill();
        var tx = ballX + dirX;
        var ty = ballY + dirY;
        if (0 < tx && tx < canvasW) { ballX = tx; }
        if (0 < ty && ty < canvasH) { ballY = ty; }
        // ブロックの衝突判定
        var j = 0;
        while (j < blocks.length) {
            var b = blocks[j];
            if (b.x <= tx && tx <= (b.x + blockW) &&
                b.y <= ty && ty <= (b.y + blockH)) {
                blocks.splice(j, 1); // ブロックを消す
                continue;
            }
            j++;
        }
        if (blocks.length == 0) {
            alert("全てのブロックを壊しました!!");
            initBlocks();
        }
    }
};
</script>
```

書き換えてデバッグ実行してみると、次のようにゲームを楽しむことができます。端末を傾けてボールを動かし、ブロックを壊していきましょう。

| 図 04-016 |

プログラムの解説～玉転がしでブロック崩し

ブロック崩しのプログラムですが、基本的には玉転がしと同じです。そこで、違う部分だけ抜粋して紹介します。まず、変数の初期化部分から見ていきましょう。

ブロック崩しでは、なんと言っても、ボールだけではなく、ブロックを管理する必要があります。前回の玉転がしでも、ボールの管理において、位置を表わす「ballX」と「ballY」、ボールの移動量を表わす「dirX」と「dirY」がありました。ブロックの管理では、ブロックは複数ありますから、複数のブロックを表わす配列変数の「blocks」と、ブロックの大きさを表わす「blockW」(幅)と「blockH」(高さ)や色の種類を表わす「blockColors」などを定義しています。

| スクリプト 04-034 |

```
01    var ballX = 150, ballY = 150; // ボールの初期位置
02    var dirX = 15, dirY = 15;     // ボールの速度
03    var blocks = [];              // ブロックを管理する配列変数
04    var blockW = 40, blockH = 20; // ブロックの大きさを定義
05    var blockPad = 5;             // ブロック同士の余白を定義
06    var blockColors = ["red","blue","yellow"]; // ブロックの色
```

このように、変数の初期化処理は、一カ所に（できるなら分かりやすく上の方に）まとめて書いておくと、後から数値を修正するのが楽になります。プログラムの至る所に初期化処理を散在させると、プログラムの見通しが悪く、メンテナンス性が悪くなり、いろいろなバグを作り込むことになってしまいます。ちょっとしたことなので気をつけましょう。

では、次に、ブロックの初期化処理を確認してみます。for文が複数出てきて、ちょっと読みにくいかもしれませんが、ブロックをY方向に6行、X方向に6列、合計36個のブロックを作成しています。そのために配列変数に値を追加するpush()メソッドを使って、ブロックのデータを追加しています。1つずつのブロックは、オブジェクトのリテラル式を用いて追加しています。オブジェクトのリテラルに関しては、03-04の「四択クイズ」で詳しく説明していますので参考にしてください。

| スクリプト 04-035 |

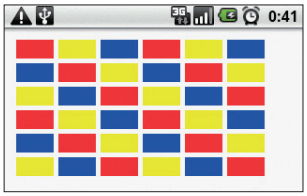
```
// ブロックを配置する
function initBlocks() {
    for (var y = 0; y < 6; y++) {
        for (var x = 0; x < 6; x++) {
            var color = blockColors[(x*5+y) % 3];
            blocks.push({
                "x" : (x * (blockW + blockPad) + blockPad),
                "y" : (y * (blockH + blockPad) + blockPad),
                "color": color
            });
        }
    }
}
```

ちなみに、このプログラムで作成されるブロックのデータは、以下のような形式となっています。

| スクリプト 04-036 |

```
[
  {"x":5,"y":5,"color":"red"},
  {"x":50,"y":5,"color":"yellow"},
  {"x":95,"y":5,"color":"blue"},
  {"x":140,"y":5,"color":"red"},
  {"x":185,"y":5,"color":"yellow"},
  {"x":230,"y":5,"color":"blue"},
  {"x":5,"y":30,"color":"blue"},
  {"x":50,"y":30,"color":"red"},
  ...
]
```

| 図 04-017 |



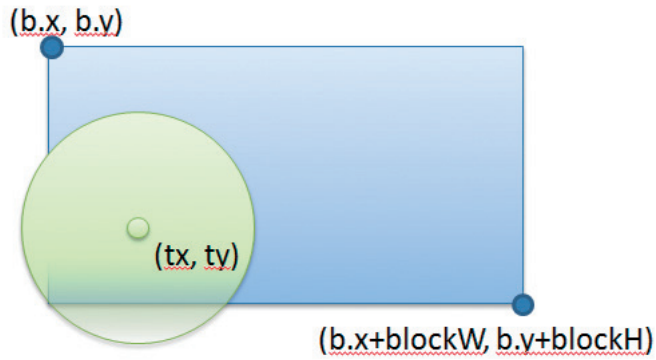
そして、ボールの移動と描画の部分は玉転がしとほぼ同じですが、ブロックとボールの衝突判定部分が追加されています。この衝突判定では、ブロックの座標を1つずつ調べていって、ブロックの座標範囲の中に、ボールの座標が含まれるかどうかを調べるという処理になっています。

| スクリプト 04-037 |

```
// ボールを描画 ...
var tx = ballX + dirX;
var ty = ballY + dirY;
...
```

```
// ブロックの衝突判定
var j = 0;
while (j < blocks.length) {
    var b = blocks[j];
    if (b.x <= tx && tx <= (b.x + blockW) &&
        b.y <= ty && ty <= (b.y + blockH)) {
        blocks.splice(j, 1); // ブロックを消す
        continue;
    }
    j++;
}
```

| 図 04-018 |



ボールがブロックの座標範囲にあるか調べる

そして、ブロックを壊す処理を見ると、配列変数（Array）のsplice() メソッドを使っています。これは「変数.splice(インデックス,個数,新要素)」のように書いて、配列変数のインデックス番目にある指定個数の要素を、新要素に入れ替えるという働きがあります。ブロックを壊す処理においては、新要素を省いて要素を1としていいますが、これはつまり、配列変数からインデックスにある要素を削るという意味になります。

```
blocks.splice(j, 1);        配列変数「blocks」のj番目の要素を1つ削除する
```

以上、加速度センサーを利用して、ボールを動かし、ブロックを壊すゲームを作ってみました。加速度センサーの使い方はアイデア次第です。ここで見てきたように、センサーの値を取得するのは簡単です。ぜひ、センサーを活かした面白いアイデアのアプリを考えてみてください。