Game Algorithm

▶ 第２回　衝突検出 ◀

衝突検出

ゲームの中で、衝突検出(Collision Detection:衝突判定、当たり判定、ヒットチェックなどとも呼ばれます)は、コンピュータゲームが出始めたころからすでに必要とされていました。たとえば、前回のスカッシュゲームです。ボールが壁やパッドに衝突したかどうかを検出しなければ、ゲームは成り立ちません。また、２次元のＲＰＧでは、キャラクタが街を歩き回るときに壁などの障害物に衝突すると、そこで足踏みしていました。これも、そこに障害物があることを検出できたからこその動きなわけです。

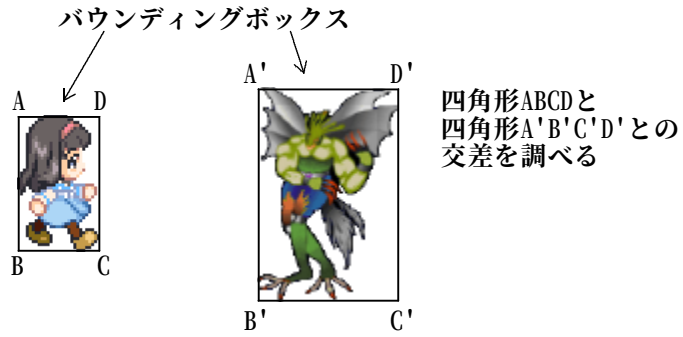
ゲームでは、キャラクタ同士の衝突を検出するという処理は、ひんぱんに必要になります。一般的なシューティングゲームを見ても、自機と敵、弾(自機)と敵、自機と弾(敵)など、さまざまな種類の衝突検出が行われます。さらに、それらが１秒間に６０回の移動が行われるとすると、そのぶんだけ衝突検出が必要となります。

このような衝突検出は、２次元でも３次元でも基本は同じです。３次元の場合は、２次元の衝突検出を拡張した処理が行われています。

バウンディングボックス

ゲームのようなリアルタイム性を追求するアプリケーションの場合、キャラクタの複雑な形状を完全に考慮した衝突検出をするのはたいへんです。実際には、バウンディングボックス(Bounding Box:衝突範囲)と呼ばれる架空の領域を用いて、その領域の交差判定を行います。

２次元の場合には、矩形や円を用いることが多く、この領域ともう一方のキャラクタの交差判定を行うことで、擬似的に衝突を検出します。たいていの場合、このバウンディングボックスは、そのキャラクタをすっぽりと包むようにします。そうでない場合には、オブジェクトのめり込みが起こる可能性が出てきます。



矩形のバウンディングボックスの場合、多くはその長方形の軸を画面座標系のｘｙ軸に平行にとります。この形式のものをAABB(Axis-Aligned Bounding Box)と呼びます。この場合には、バウンディングボックスの領域は、左上の頂点座標と右下の頂点座標(または右上と左下の頂点座標)で与えることができます。ゲームでは、キャラクタのだいたいの位置関係は把握できていることが多いので、AABBを使うと条件判定回数が非常に少なくて済むという利点があります。しかし、キャラクタが回転などをする場合には、それに合わせてボックスの頂点座標を計算し直すことになるので、その手間がかかることがあります。

矩形の衝突検出

上の図のバウンディングボックスの頂点Aの座標を(Ax, Ay)、頂点Cを(Cx, Cy)、頂点A'を(A'x, A'y)、頂点C'を(C'x, C'y)と表します。このとき、２つのバウンディングボックスが交差しているかどうかを調べるプログラムは、次のようになります。

```
if(Ax <= C'x && Cx >= A'x && Ay <= C'y && Cy >= A'y)
    交差している
else
    交差していない
```

画面座標系のｙ軸は、数学と違って下方向に向いていることに注意しましょう。また、APIには２つの矩形が交差しているかを調べるIntersectRect関数や、矩形内に座標があるかどうかを調べるPtInRect関数があります。これらの関数のほかに、リージョンという機能を使っても、衝突を検出することができます。リージョンでは、多角形や円などを始め、複雑な形状の交差を調べることができます。

円の衝突検出

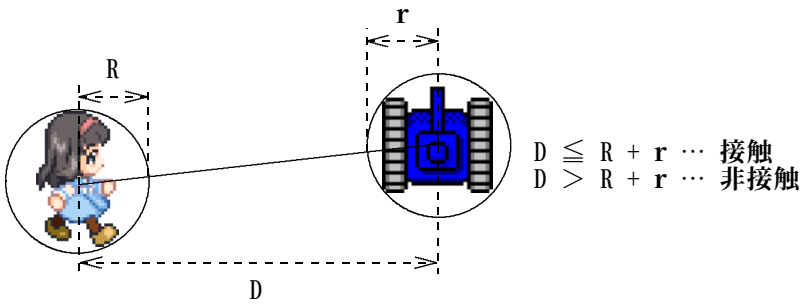
円のバウンディングボックスは、その中心と半径を定義するだけで扱うことができます。キャラクタの衝突を検出する場合には、それぞれの衝突範囲を定義している円の中心同士の距離を測定し、これが各半径の和以下であれば、このふたつのキャラクタは衝突しているとみなします。

距離は、

$$D = \mathrm{sqrt}((x0 - x1)^2 + (y0 - y1)^2)$$

と計算しますが、ここでは衝突しているかどうかを調べるだけなので、よぶんな平方根計算は省いて、D^2と$(R + r)^2$の大小比較で判定を行うことができます。

$D^2 \leqq (R + r)^2$ … 接触
$D^2 > (R + r)^2$ … 非接触



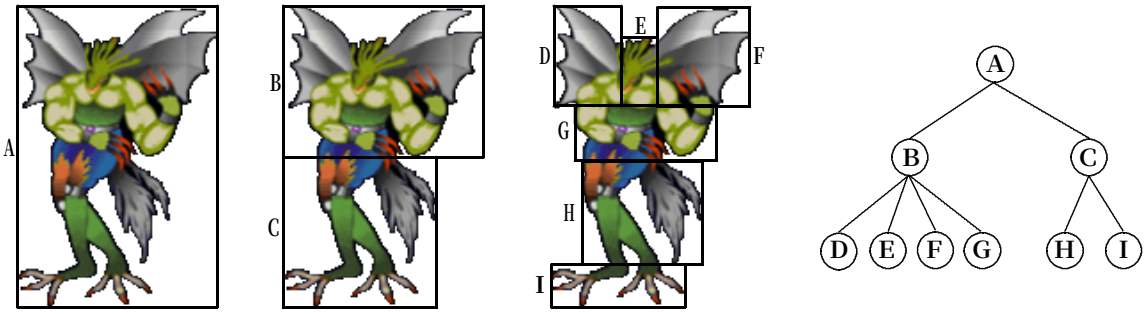
▲円の衝突検出

バウンディングボックスの細分化

AABBはよく使われますが、キャラクタの形状によって誤差が大きく変動します。たとえば、下の右側のようなキャラクタでは、実際には衝突していない部分を多く含んでしまう可能性があります。左側のように、直立しているキャラクタであれば、そのバウンディングボックスは非常に適合していますが、真ん中のように４５度傾いた状態では、これだけのニセ衝突領域が生じてしまいます。

これを極力減らすには、AABBによる細分化を考えます。これは、キャラクタをいくつかのパーツと考え、それにバウンディングボックスを被せる方法です。こうしてキャラクタの衝突を、複数の衝突判定で検出するのです。計算量はパーツの分割数にそのまま比例しますが、衝突検出の誤差はかなり改善されます。

また、逆に複雑な形状のキャラクタ同士の衝突判定に、このいくつかの異なったレベルのバウンディングボックスを使うことで、早い段階での判定が可能になります。



▲細分化されたAABBツリー

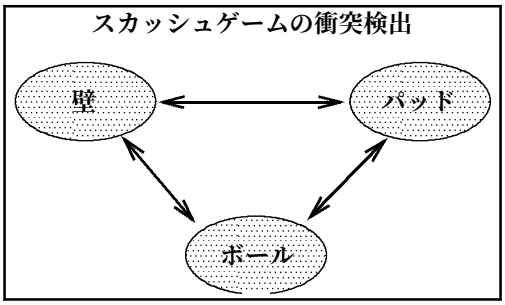
衝突するものの整理

前回の「スカッシュゲーム」は、単純にボールをパッドで打ち返すだけのゲームです。しかし、ここでもすでに簡単な衝突検出処理を行っています。

スカッシュゲームの場合、キャラクタとして存在するものは、

- ボール
- パッド
- 壁

の３種類で、これらの要素で衝突検出を行います。



パッドと壁の衝突

パッドと壁の衝突検出は、パッドが左右にしか動かないのと、壁が簡単な形状をしており、かつ移動しないということから、とても簡単に行うことができます。

パッドは、自身が画面上のどの領域に存在するのかという情報を持っています。パッドの形状は、長方形という非常に単純なものなので、この情報をそのままバンディングボックスとして利用することができます。

壁に関しては、グローバルな変数として、パッドとボールが移動できる領域が宣言されています(変数MOVENABLE)。この領域外が移動できない領域、すなわち壁となります。パッドが「移動できる」領域の外に出ているかどうかを調べれば、壁との衝突を検出できるというわけです。

実際には、パッドの左側または右側が領域外に出ているかどうかで衝突を検出しています。

ボールと壁の衝突

ボールの形状は円であるため、バウンディングボックスも円で扱います。壁との衝突は、この"円"と、壁の表面に相当する"直線"が、交わっているかどうかを調べればいいことになります。

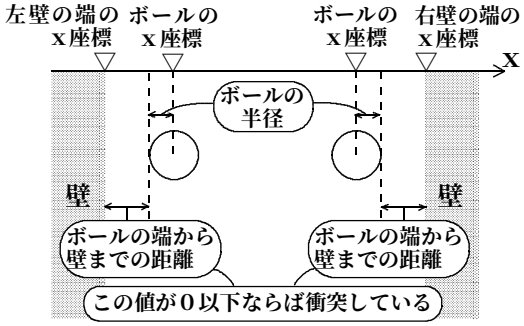
スカッシュゲームの壁は、好都合なことに画面座標系に対して"平行"または"垂直"な関係になっています。よって、ボールと壁の衝突検出は、「ボールの中心の座標にボールの半径を増減させた値」と「壁の座標の値」を判断すればいいわけです。

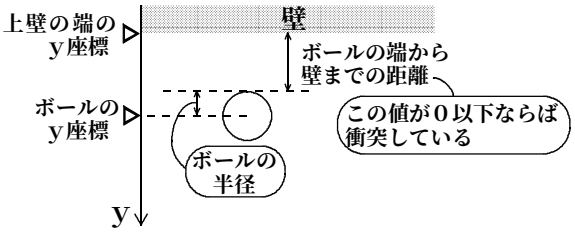
まず、ボールが左右の壁に衝突しているかどうかの判断について考えてみます。これは、ボールのｘ座標の値だけが必要となります。

ここで、右ページの図を参照してください。左の壁については、ボールの中心ｘ座標からボールの半径を引き、これが左の壁のｘ座標より等しいか小さければ、ボールが壁に衝突している状態になります。また、右の壁については左側とは逆に、ボールの中心ｘ座標にボール半径を加え、その座標が右の壁のｘ座標より等しいか大きければ衝突していることになります。

上の壁とボールの衝突検出は、座標軸がｘ座標からｙ座標に変わっただけで、基本的な考え方は同じです。つまり、ボールのｙ座標からボールの半径を引き、その値が上の壁のｙ座標より等しいか小さければ衝突していることになります。

このように、座標軸と平行または垂直な関係になっている壁との衝突検出は、かんたんな比較で実現することができます。





パッドとボールの衝突

パッドとボールの衝突アルゴリズムを考えてみましょう。パッドは、基本的に「短い壁が４つ組み合わされたもの」というように考えることができます。

しかし、壁の長さが短いことから、外側の壁のように片方の座標数値だけで衝突を検出することはできません。このことを踏まえた上でパッドの上下の辺とボールの衝突検出を考えてみましょう。このアルゴリズムは、以下のような手順で処理を行う必要があります。

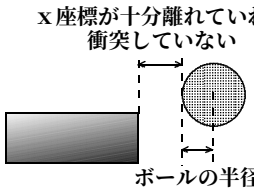
①両方のｘ座標による衝突可能性の判断
下の図のパターンＡのように、２つの物体のｘ座標が離れていれば、ｙ座標が接近していても衝突することはありません。

②上下の辺に衝突する可能性の判断
次に、下の図のパターンＢのように、ボールがパッドの角付近に位置している場合は、ボールが角に衝突している可能性があります。しかし、これはｘ座標やｙ座標単体での判断では不十分です。この場合は、両方の座標を用いた"平面的な判断"が必要となり、衝突判断がやや複雑になります。

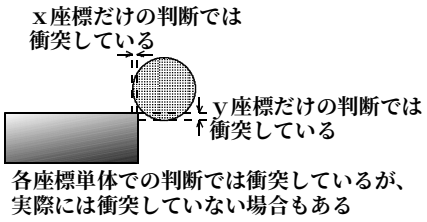
③パターンＣの位置における衝突判断
下の図のパターンＣの位置であれば、壁との衝突判断と同様の手順で判断ができます。

④パターンＢの位置における衝突判断
再びパターンＢの位置の場合、角とボールの外周が"接触するかどうか"がポイントになります。ここでは、ボールの中心から問題となる角までの距離を計算し、半径以下ならば衝突していると判断します。
ｘ座標とｙ座標の関係を入れ換えれば、パッドの左右の辺とボールの衝突判定も、同様に行えます。

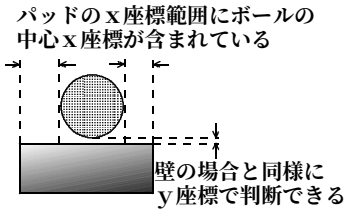
パターンＡ



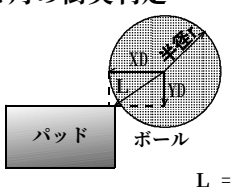
パターンＢ



パターンＣ



ブロックと角の衝突判定



ボールの中心からパッドの
角までの距離を計算
↓
ボールの半径以下なら衝突！

ｘ座標の差をXD、
ｙ座標の差をYDとすると、
距離Ｌは以下のようになる

L = sqrt(XD² + YD²)　　L≦ rならば衝突

§ 課題 §

前回のスカッシュゲームのプログラムは、パッドとボールの衝突検出が簡略化されていて、正しく動作しない場合があります。プログラムを変更し、正しく検出できるようにしましょう。また、衝突した辺から跳ね返るようにもしましょう。