ソフトゼミＢ　第６回 当たり判定

■　はじめに

さて、弾が発射できるようになり、ファイルを分割して見やすくなりました。いよいよ敵を攻撃したりダメージを受けたりする機能をつけられる段階になったので、今回はそのための重要な考え方「当たり判定」を習得して、攻撃に関する機能を実装しましょう。

■　当たり判定って？

現実世界ではモノ同士が同じ場所に存在することはできません。同じ場所に来る直前にぶつかりますよね。しかし、ゲームでは《すり抜ける》ことがよくあります。これでは自分の発射した弾で相手にダメージを与えることができません。

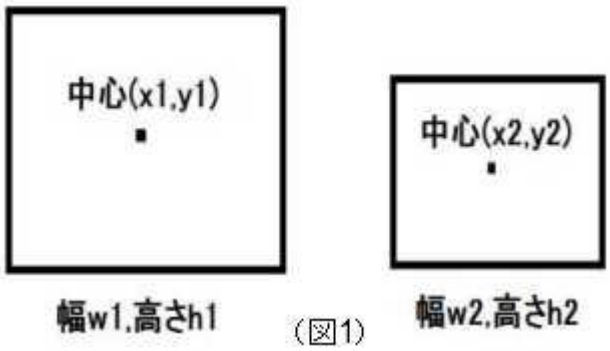
そこで、2 つの物体の衝突･接触の有無を判定する《当たり判定》を使うのです。

■　四角形同士の当たり判定

➢　やり方

右図 1 のような 2 つの四角形同士が衝突・接触するかを考えます。

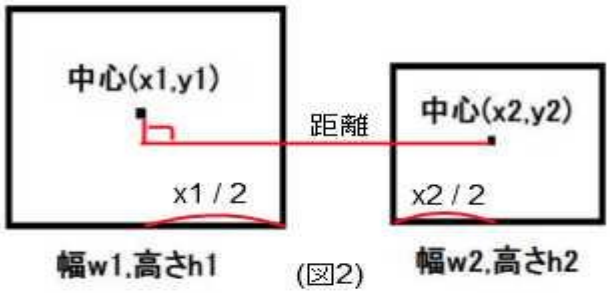
（ここでは、衝突＝重なる、
接触＝触れると考えてください。）



(図1)

図 1 ではまだ衝突も接触もしていませんが、そのことをコンピュータに認識させる為に､「幅･高さの半分」と「中心間の距離」を書いてみます。

図 2 に上記 2 つを書き込みました。もう分かりましたか？



(図2)

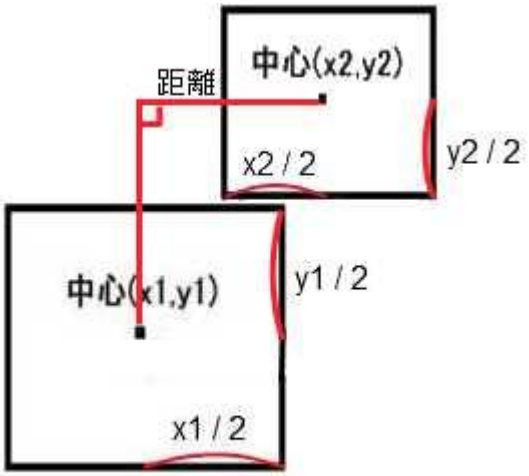
《中心間の距離 ＞ 幅の半分の和》

の結果が真であれば、衝突・接触していないと判断することができます。

この一連の判断が《当たり判定》です。

ですが、その逆であればぶつかっているのかというと、必ずしもそうではありません。

右図 3 では x 軸では、
《中心間の距離 ＜ 幅の半分の和》
と真になっていますが、y 軸では、
《中心間の距離 ＞ 幅の半分の和》
であり、偽になっています。



(図3)

この図では片方の判定が真であるにもかかわらず、接触も衝突もしていません。

四角形で当たり判定するには縦、横の両方で判定する必要があるのです。
これを真偽の式で表すと、

$$|x1 - x2| < (w1 + w2) \div 2 \quad \cap \quad |y1 - y2| < (h1 + h2) \div 2$$

となり、これが真であることが、2 つの四角形同士が接触・衝突している条件です。

➢ 当たり判定をプログラムにしてみる

さて、実際に当たり判定をプログラムにしてみましょう。前回のファイル分割を応用して、atari.cpp を新たに作成し、extern ヘッダをインクルードして書き込むのが楽です。

```
#include "Dxlib.h"
#include "extern.h"
#include "function.h"
```

(次ページへ続く)

```
int atariSquareToSquare( int x1, int y1, int w1, int h1, int x2, int y2, int w2, int
h2 ){
  if(
    abs( x1 - x2 ) < ( w1 + w2 ) / 2 &&
    abs( y1 - y2 ) < ( h1 + h2 ) / 2
  ) {
    return 1;
  }
  return 0;
}
```

引数としてもらってくる値は 2 つの四角形の中心座標 x、y と幅 w、高さ h です。この値を利用して if 文の中で当たっているかを判定します。当たっていれば値 1 が返され、そうでなければ 0 が返ってきます。

さて、見慣れない関数 abs()がいます。この関数は、受け取った値を絶対値(absolute value)にして返す機能を持っています。中心座標の差の計算結果をマイナスにすると必ず当たりと判定されてしまうので忘れずに書いてください。※abs は標準ライブラリ関数である stdlib.h 内で宣言されていますが、DX ライブラリをインクルードしている時点で stdlib.h 内の関数は全て使えるようになっています。(実際には、DxLib.h は Windows.h なるヘッダファイルをインクルードしていて、Windows.h をインクルードすることにより stdlib.h の関数は全て使えるようになっている。)

➢ 判定関数を応用してプログラムを完成させる

では、この判定関数を応用して自機、敵機、自弾、敵弾を実際に判定する関数を作って行きましょう。次のプログラムを、先ほど書いた当たり判定関数に続けて書いてください。

(※書く前に読むと悲しみを背負わないリードミー)

・長いプログラムなので、コピペをうまく使うなどして工夫して書いてみましょう。

・当たり判定関数 atariSquareToSquare()の引数は長いので改行で見やすくしています。

改行しただけなので、行の終端はカンマです。セミコロン打たないでね！

```
/*自機が敵機に衝突*/
void atariEnemyToPlayer( void ){
  int i;
  for( i = 0; i < ENEMY_MAX; i++ ){
    if( enemy[ i ].life == 0 ){ continue; }
    if( atariSquareToSquare(
      player.x, player.y, PLAYER_SIZE, PLAYER_SIZE,
      enemy[ i ].x, enemy[ i ].y, ENEMY_SIZE, ENEMY_SIZE
    ) ){
      player.life--;
    }
  }
}


/*自機が被弾  */
void atariPlayerToEnemyShot( void ){
  int i;
  for( i = 0; i < ENEMY_SHOT_MAX; i++ ){
    if( enemyShot[ i ].life == 0 ){ continue; }
    if( atariSquareToSquare(
      player.x, player.y, PLAYER_SIZE, PLAYER_SIZE,
      enemyShot[ i ].x, enemyShot[ i ].y, BULLET_SIZE, BULLET_SIZE
  ) ){
      player.life--;
    }
  }
}


/*敵機が被弾*/
void atariEnemyToPlayerShot( void ){
```

```
  int i, j;
  for( i = 0; i < PLAYER_SHOT_MAX; i++ ){
    if( playerShot[ i ].life == 0 ){ continue; }
    for( j = 0; j < ENEMY_MAX; j++ ){
      if( enemy[ j ].life == 0 ){ continue; }
      if( atariSquareToSquare(
        playerShot[ i ].x, playerShot[ i ].y, BULLET_SIZE, BULLET_SIZE,
        enemy[ j ].x, enemy[ j ].y, ENEMY_SIZE, ENEMY_SIZE
      ) ){
        enemy[ j ].life--;
        if( enemy[ j ].life == 0 ){
          enemyNum--;
          player.score += 100;
        }
      }
    }
  }
}
```

長文記述お疲れ様でした。しかし、前回のプログラム記述が終了していれば、断片的に意味は理解できるはずです。大まかな流れは次の通りです。

1. 扱う対象の各配列要素を使用しているかチェック。（関数によっては 2 回）
2. 扱う対象のパラメータを引数として与えて、当たり判定関数を使用する。
3. 判定が真（接触・衝突している）の場合に何らかの変化が起こるようにする。

今回の記述ではマクロ定数を多く使用しているので、小指が激しく疲れたかもしれませんが、逆に画像サイズに変更があった場合でも、マクロ定数を変化させるだけで問題無く対応できます。

➢ デバッグできるようにする

判定するプログラムは完成しましたが、関数として認識させ、メインプログラムに組み込まないと動作すらしません。ささっと組みこんでデバッグしてみましょう。

まずは main.cpp に「//当たり判定」のコメントの下 3 行のみを書きましょう。

```
（略）
    // 敵の移動, 弾発射
     （略）

    //当たり判定
    atariEnemyToPlayer();
    atariPlayerToEnemyShot();
    atariEnemyToPlayerShot();

    /**ここから描画 **/
    draw();
（略）
```

移動や発射の後で全ての物体がアクションを起こしてから判定するので、敵の移動や発射の後に書き、判定で消滅した敵などを描画しないようにするので、描画の前に書きます。

もう一つ、function.h の文頭に当たり判定関数を追加しましょう。

```
/* atari.cpp */
int atariSquareToSquare ( int, int, int, int, int, int, int, int );
void atariEnemyToPlayer ( void );
void atariPlayerToEnemyShot ( void );
void atariEnemyToPlayerShot ( void );

/* draw.cpp */
```

これで、デバッグできるようになったはずです。ですが、プレイしていて気付いたでしょうか？このゲームはここで初めてゲームオーバーになる可能性が出てきたのです。

最後の仕上げにゲームオーバー時には draw.cpp に表示させるようにしましょう。

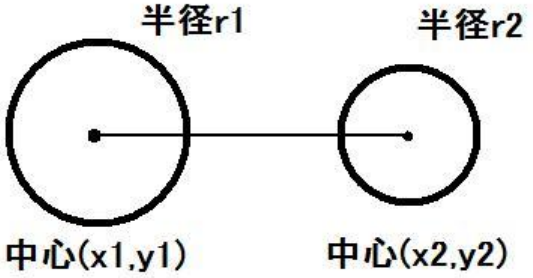
```
（略）
    //画面に描かれたものを消去する
    ClearDrawScreen();

    //この if 文を書こう
    if( player.life < 0 ){
      DrawString(WINDOW_WIDTH / 2 - 100, WINDOW_HEIGHT / 2, "Game Over", 0xffffff);
      DrawString(WINDOW_WIDTH / 2 - 100, WINDOW_HEIGHT / 2 + 20, "Press 'Esc' key to quit.", 0xffffff);
      ScreenFlip();
      return;
    }

    //自機を描画
（以下略）
```

当たり判定プログラムは、これに練習問題のプログラムを加えて完成です。あと一息！

■　円同士の当たり判定

今回教えたのは四角形同士の当たり判定ですが、当たり判定は判定する対象の形状によって異なる方法を使います。その一例として少しだけ円を取り扱います。

円は幅と高さが直径として統一されています。ですから縦横関係無く判定回数は１回です。



ここで、2 点間の距離を出すために使うある方法を使いたいと思います。
2 点間の距離といえば、そう「ピタゴラスの定理（三平方の定理)」です。
2 つの円同士が衝突している条件は、

$$(x1 - x2) \times (x1 - x2) + (y1 - y2) \times (y1 - y2) < (r1 + r2) \times (r1 + r2)$$

です。今回、平方根は必要ありません。２乗数でも差が出るので比較が可能だからです。
逆に不等号が＞のときは 2 つの円は離れており、＝のときは 2 つの円は接触しています。

他にも三角形や線分、複雑な形を対象とする当たり判定がありますが、ここでは紹介しません。自分のやりたいゲームに合った判定方法を使うことの方が重要です。

■ 練習問題

１．先ほど作った当たり判定プログラムには、唯一プレイヤーの弾と敵の弾の当たり判定を行なう関数が存在しません。
　無くてもゲームは成り立ちますが、このゲームでは敵弾は回避するしか手段が無いのでぜひ欲しいところです。他の関数と同じ要領で `AtariPlayerShotToEnemyShot` を作成し、プログラムに組み込んでください。

２．円同士の当たり判定を行なう関数 `AtariCircleToCircle` を作成してください。
今回のゼミでは円同士の当たり判定は使わないので、関数を作成し理解を深めておいてもらえれば大丈夫です。

３．暇になったら次の条件の当たり判定について考察してみてはどうでしょうか。
　　3-1. レーザービームの当たり判定（直線と点の当たり判定方法)
　　3-2. カニ型ボスの当たり判定（複雑な形の当たり判定方法)
　　（ピクセル毎に判定する方法は STG だと重すぎるので非推奨です。）