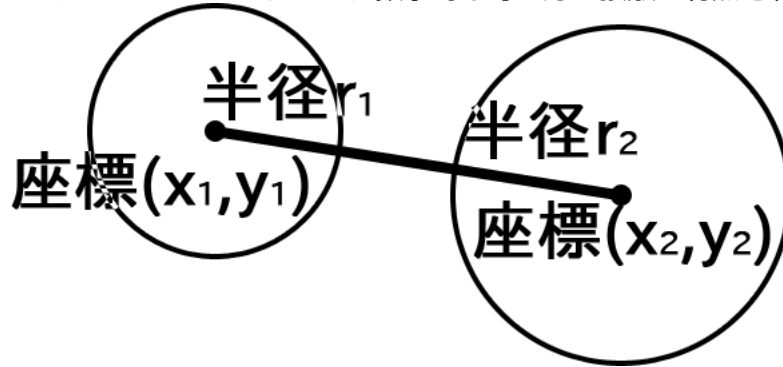


第4回 当たり判定1

当たり判定はゲームを作る上で必要不可欠な要素です。今日は当たり判定のプロセス・実際の計算方法について説明していきます。

○当たり判定の原理

当たり判定とは、2つの物体の衝突・接触の有無を判定することです。「そんなの見れば分かるじゃん」と思うかもしれませんが、コンピュータは計算しかできません。我々が判定方法を厳密に定義しなくてはなりません。そこでコンピュータには、数学的な考え方で接触の有無を判定してもらいます。



例えば上のように2つの円があったとします。この2つの円が衝突している(※)条件は、

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} < r_1 + r_2$$

となります。「ピタゴラスの定理(三平方の定理)」を利用して、2つの円の中心間の距離と半径の和を比較しています。上の画像では2つの円は衝突していないので、「中心間の距離が半径の和よりも大きい」状態です。また、接触時(※)は大小関係が等号になります。

※今回のゼミにおいて、衝突=重なっている状態、接触=触れている状態、とします。

それでは実際にプログラムを書いてみます。collision.cpp をソースファイルに追加して下さい。

・collision.cpp

```
/*
collision.cpp    当たり判定
*/
// インクルード
#define _USE_MATH_DEFINES
// C 言語標準関数
#include "math.h"
// 自作ヘッダ
#include "define.h"
#include "extern.h"

// 円同士の衝突判定
int CollisionCircleToCircle( int x_1, int y_1, int r_1, int x_2, int y_2, int r_2 ){
    // 円同士の当たり判定。x_1、y_1、x_2、y_2 は各々の円の中心の座標。
    // r_1、r_2 は各々の円の半径
    if( ( x_1 - x_2 ) * ( x_1 - x_2 ) +
        ( y_1 - y_2 ) * ( y_1 - y_2 ) <
        ( r_1 + r_2 ) * ( r_1 + r_2 ) ){
        // 三平方の定理より、衝突時は1を返り値として返す
        return ( 1 );
    }
}
```

```

    }
    // 非衝突時は0を返り値として返す
    return( 0 );
}

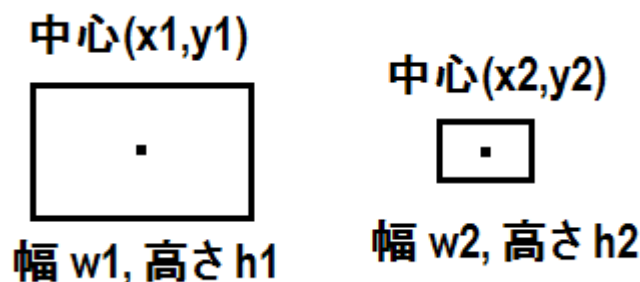
// 敵と自機の衝突判定
void CollisionEnemyToPlayer( void ){
    int i;
    // 衝突する対象の生存を確認の後、判定
    if( player.life > 0 ){
        // 全ての敵と判定を行う
        for( i = 0; i < ENEMY_MAX; i++ ){ // 敵の全ての構造体に対して
            if( enemy[ i ].life > 0 ){ // その敵が存在するとき
                // 敵の種類に応じて当たり判定を分岐可能
                switch( enemy[ i ].type ){
                    default: // 今回は全てデフォルト
                        if( CollisionCircleToCircle(
                            player.pos[ 0 ], player.pos[ 1 ], 8
                            enemy[ i ].pos[ 0 ], enemy[ i ].pos[ 1 ], 16 ) == 1 ){
                            // 敵とプレイヤーが衝突しているとき、
                            // プレイヤーのライフに0を代入
                            player.life = 0;
                        }
                        break;
                    }
                }
            }
        }
    }
}

```

今回作った関数は function.h 内でプロトタイプ宣言し、CollisionEnemyToPlayer 関数の呼び出しを Game 関数内のメインループ内の当たり判定部に追記しておきましょう。これで、敵との衝突判定が行われるはずです。この時点でのソースコードはフォルダ「a」を参照してください。

今回は円形同士の当たり判定しか使いませんが、矩形(長方形)同士の当たり判定もマスターしておきましょう。

○矩形(四角形)の当たり判定



矩形は円と違って縦と横の概念があるので、2回判定をする必要があります。上の2つの矩形が衝突している条件は、

$$|x_1 - x_2| < (w_1 + w_2) \div 2 \cap |y_1 - y_2| < (h_1 + h_2) \div 2$$

となります。まず、2つの矩形がX軸方向について考えた場合に接触しているかどうかを確認します。ここで接触していたら、Y軸方向について考えた場合に接触しているかどうかを確認、2つとも条件

が合えば、2つの矩形は接触していることになります。

今回は円同士と矩形同士の当たり判定について学びました。他にも様々な当たり判定の方法があります。

○練習問題

1. 矩形同士の当たり判定を行う関数 `CollisionSquareToSquare` を作成し、`CollisionCircleToCircle` 関数の後に追記してください。引数は任意とします。解答はフォルダ「b」を参照してください。

2. 敵と同様に、自機の弾の構造体 `SShot` を作成し、`struct.h` に追加、`define.h` 内で画面上のショットの最大数 `SHOT_MAX` を定数定義、`global.h` 内で `SShot` 型構造体変数の配列 `shot` をグローバル変数として宣言、`extern.h` 内で `extern`、また、自機の弾の画像を作成し、構造体 `SImg` 内に画像をロードするための変数 `shot` を追加、`ImgLoad` 関数内でロード、`shot.cpp` を作成し、今までの自機・敵同様に `SetShot` 関数と `MoveShot` 関数と `DrawShot` 関数を作成、それらを `function.h` 内でプロトタイプ宣言し、各々の構造体の初期化部分で宣言した自機の弾の初期化を行い、`Game` 関数の変数初期化部分で配列 `shot` の初期化を行った後、作成した `MoveShot` 関数と `DrawShot` 関数を適切な位置で呼び出してください。また、`CollisionShotToEnemy` 関数を作成し、これもプロトタイプ宣言後、適切な位置で呼び出して下さい。解答はフォルダ「c」を参照してください。