

## 第6回 重力

今回は敵弾と重力について説明します。敵弾はシューティングゲームに必要不可欠な要素です。この原理は自機から発射するショットと対して変わりません。もう一つの今回の要素である重力は、シューティングゲームにおいて頻出の要素ではありませんが、様々なジャンルに応用することができ、また、表現の幅を広げることが出来ます。

### ○敵弾の発射

敵弾の発射のためには、敵弾の構造体、最大数の定義、描画、移動処理の作成をする必要があります。まずは **define.h** に画面上の敵弾の最大数の定義を記述しましょう。

```
#define BULLET_MAX ( 128 )
```

自機の弾は **SHOT**、敵弾は **BULLET** と今回のプログラムでは呼称することにします。間違えないよう、注意して下さい。記述は、**SHOT\_MAX** の下あたりなどが良いでしょう(構造体の最大数をまとめておく)。次に、構造体を **struct.h** に定義していきます。これも、敵やショットとそう変わりません。

```
// 敵弾の情報
struct SBullet{
    int exist ;    // 存在するか否か(0:存在 1:非存在)
    int pos[ 2 ] ; // 座標
    int type ;     // 種類
    float v ;      // 速度
    float th ;     // 角度
} ;
```

各メンバ変数の意味は右側のコメントで記述している通りです。角度については、右側を **0** とし、時計回りに角度が大きくなるものとします。記述が終わったら **global.h** で以下のようにグローバル変数を宣言して下さい。

```
struct SBullet bullet[ BULLET_MAX ] ; // 敵弾
```

これも今までどおりですね。当然 **extern.h** で **extern** するのを忘れないようにしましょう。記述位置は配列 **shot** の宣言の下、変数 **map** の宣言の上がオススメです。

さて、ここまで記述したら、**bullet.cpp** を追加しましょう。追加した **bullet.cpp** に **MoveBullet** 関数と **DrawBullet** 関数を記述していきます。

```
// C 言語標準ヘッダのインクルード
#define _USE_MATH_DEFINES
#include "math.h"

// DX ライブラリのヘッダをインクルード
#include "DxLib.h"

// 自作ヘッダのインクルード
#include "extern.h"
#include "function.h"

int SetBullet( int x , int y , int type ){
    int i ;
    for( i = 0 ; i < BULLET_MAX ; i++ ){
        if( bullet[ i ].exist == 0 ){
            bullet[ i ].exist = 1 ;
            bullet[ i ].pos[ 0 ] = x ;
            bullet[ i ].pos[ 1 ] = y ;
            bullet[ i ].th = -M_PI / 4 ;
            bullet[ i ].type = type ;
            bullet[ i ].v = 4 ;
        }
    }
}
```

```
}
}
```

今度の関数はいつもとは違います。引数に注目して下さい。**SetBullet** 関数が呼び出された際、同関数は引数として **int** 型の変数 **x**、**y**、**type** をうけとり、初期値として利用しています。つまり、この関数のプロトタイプ宣言さえしておけば、好きなタイミングで指定した座標に指定したタイプの敵弾を設置することができます(同じ方法で他のメンバ変数についても初期値を設定可能ですが、プログラムが猥雑になるので省略)。**function.h** に追加すべき **SetBullet** 関数のプロトタイプ宣言は以下の通り。

```
// bullet.cpp
int SetBullet( int x , int y , int type ) ;    // 敵弾設置処理
```

先程同様、**shot** 関連のまとめの下がオススの記述位置です。

さて、この **SetBullet** 関数はどこで呼び出すのでしょうか？ ずばり、**MoveEnemy** 関数内で呼び出します。**MoveEnemy** 関数内に **Setbullet** 関数の呼び出し処理を書いてやると、敵の移動後(または移動前)に敵弾を設置することになります。ちなみに、この場合敵が敵弾を設置したと同時に後の敵と自弾の当たり判定(未作成)の処理によって死亡する可能性があるので、注意が必要です。逆に言えば、敵が死亡したと同時に敵に弾を出させたい場合、敵の死亡処理の部分で **SetBullet** 関数を呼び出してやればよいのです。

さて、**MoveEnemy** 関数を見てください。ここでやっと、今まで無意味だった **SEnemy** 構造体のメンバ変数 **count** が役に立ってきます。敵は、設置された際に **count** を **0** で初期化しています。そして、出現後ゲームのループが **1** 周するたびに、**count** は **1** 加算されます。**count** を利用すれば、登場後の時間を条件に、弾を発射させるなどの様々な処理が可能になるのです。敵が画面外に出た場合の消去処理の直後、**break** 処理の直前に、以下の敵弾の発射処理を書いていきます。

```
// 敵弾の発射
if( enemy[ i ].count == 100 ){
    SetBullet( enemy[ i ].pos[ 0 ] , enemy[ i ].pos[ 1 ] , 0 ) ;
}
```

登場後、**100** ループ目に敵弾を設置しています。

さて、いざ敵弾を設置しても移動、描画処理をしなければ意味がありません。**SetBullet** 関数の下に以下のように **MoveBullet** 関数と **DrawBullet** 関数を記述してください。

```
void MoveBullet( void ){
    int i ;
    float v_x , v_y ;
    for( i = 0 ; i < BULLET_MAX ; i++ ){
        if( bullet[ i ].exist == 1 ){
            switch( bullet[ i ].type ){
                case 0:
                    bullet[ i ].pos[ 0 ] += cos( bullet[ i ].th ) * bullet[ i ].v ;
                    bullet[ i ].pos[ 1 ] += sin( bullet[ i ].th ) * bullet[ i ].v ;
                    break ;
                case 1:
                    bullet[ i ].pos[ 0 ] += cos( bullet[ i ].th ) * bullet[ i ].v ;
                    bullet[ i ].pos[ 1 ] += sin( bullet[ i ].th ) * bullet[ i ].v ;
                    v_x = cos( bullet[ i ].th ) * bullet[ i ].v ;
                    v_y = sin( bullet[ i ].th ) * bullet[ i ].v ;
                    v_y += 0.1 ;
                    if( v_y > 8 )
                        v_y = 8 ;
                    if( v_x != 0 && v_y != 0 )
                        bullet[ i ].th = atan2( v_y , v_x ) ;
                    bullet[ i ].v = pow( v_x * v_x + v_y * v_y , 0.5 ) ;
                    break ;
                default:
                    break ;
            }
        }
    }
}
```

```

    }
}

void DrawBullet( void ){
    int i ;
    for( i = 0 ; i < BULLET_MAX ; i++ ){
        if( bullet[ i ].exist == 1 ){
            DrawCircle( bullet[ i ].pos[ 0 ] , bullet[ i ].pos[ 1 ] , 8 , 0xFF0000 ,
                TRUE ) ;
        }
    }
}
}

```

今まで通りのプログラムですが、敵弾の種類が **1** だった場合のみ今までと異なることがわかります。

さて、その説明は後で行うとして、ここまでできたら、**function.h** 内で記述した関数のプロトタイプ宣言を行った後、**Game** 関数のループ内の適切な部分で **DrawBullet** 関数及び **MoveBullet** 関数を呼び出してやりましょう。赤い敵弾が右上方向に発射されれば成功です！

### ○重力

さて、敵弾の種類が **1** の時に行われている処理が重力の処理です。通常の変動の後に、関数先頭で宣言したローカル変数 **v\_x**、**v\_y** を使用し、速度と進行方向を変更しています。

プログラム内の薄く網掛けをした部分を見て下さい。ここでは、速度と角度から、速度の **X** 成分と **Y** 成分を取り出しています。その後の部分では、**v\_y** を **0.1** 加算、つまり画面下方向に加速しています。そして、下方向への速度が一定速度を超えてた場合、超えないように代入する処理を行っています。速度の **X** 成分と **Y** 成分の算出が終わったら、そのデータをもとに速度と角度を代入していきます。

ここで使用されている **atan2** 関数と **pow** 関数は、どちらも **math.h** 内でプロトタイプ宣言されている関数です。**atan2** 関数は **x** 成分と **y** 成分から角度を算出する関数です。また、**pow** 関数は、第一引数の第二引数乗を算出する関数です。ここでは、算出された **X** 方向の速度と **Y** 方向の速度から、**atan2** 関数を用いて角度を算出し、代入、また、その際の速度を三平方の定理から算出し、代入しています。

念のために補足すると、 $\sqrt{x} = x^{\frac{1}{2}} = x^{0.5}$  ですので、上記のような方法で速度が求められます。また、**atan2** 関数は **2** つの引数両方に **0** を渡すと、プログラムが強制終了してしまうため、**if** 文で避けています。

さて、実際にどのように重力がかかるか見てみるために、**MoveEnemy** 関数内での、**SetShot** 関数の呼び出しの際の、最後の引数の **0** を **1** に書き換えましょう。

### ○練習問題

**1.** 敵弾と自機の衝突判定を行う関数 **CollisionBulletToPlayer** を作成してください。なお、衝突時は自機のライフを **0** にしてください。

**2.** 自弾と敵の衝突判定を行う関数 **CollisionShotToEnemy** を作成してください。なお、衝突時は敵のライフを **0** にして下さい。