DXライブラリを用いてブロック崩しを作る

Part5 ボールとブロックの当たり判定

準備

- 今回はボールとブロックの当たり判定ということで、ボールの値とブロックの値両方使うことになる。
- つまり、全ての場所でボールの値とブロックの値を用いるので、ボールとブロックのグローバル化をする必要がある。
- 前回は main 関数内で宣言したBallとBlockを main関数外で宣言してやることになる。
- また前回では説明していない、BlockのsizeX,sizeYを取得する関数も今回使うので作っておくこと。

グローバル化

```
class Block_obj{
private:
    int x,y;
    int sizeX,sizeY;
    int color;
public:
    int GetX(),GetY();
    int GetSizeX(),GetSizeY();
    void Set(int setX,int setY,int setColor);
    void Graph();
};

class Ball_obj{
private:
    int x,y,r;
    int vx,vy;
    int color;
public:
    int GetX(),GetY(),GetR();
    int GetVX(),GetVY();
    void Set(int setX,int setY,int setR,
        int setVX,int setVY,int setColor);
    void Graph();
    void Calc();
};

Block_obj Block[BLOCK_X_NUM][BLOCK_Y_NUM];
Ball_obj Ball;

int Block_obj::GetX(){
    return x;
}
```

グローバル化とはmain関数より
上で宣言してしまうこと。
これにより、ほぼ全ての領域でその変数群を
使うことが出来るようになる。

クラスのグローバル化はどのようにするか
と言うと左の図のように

クラスの定義（中身の決定）

クラスの宣言（インスタンスの作成）

クラスの関数の定義（関数の定義）

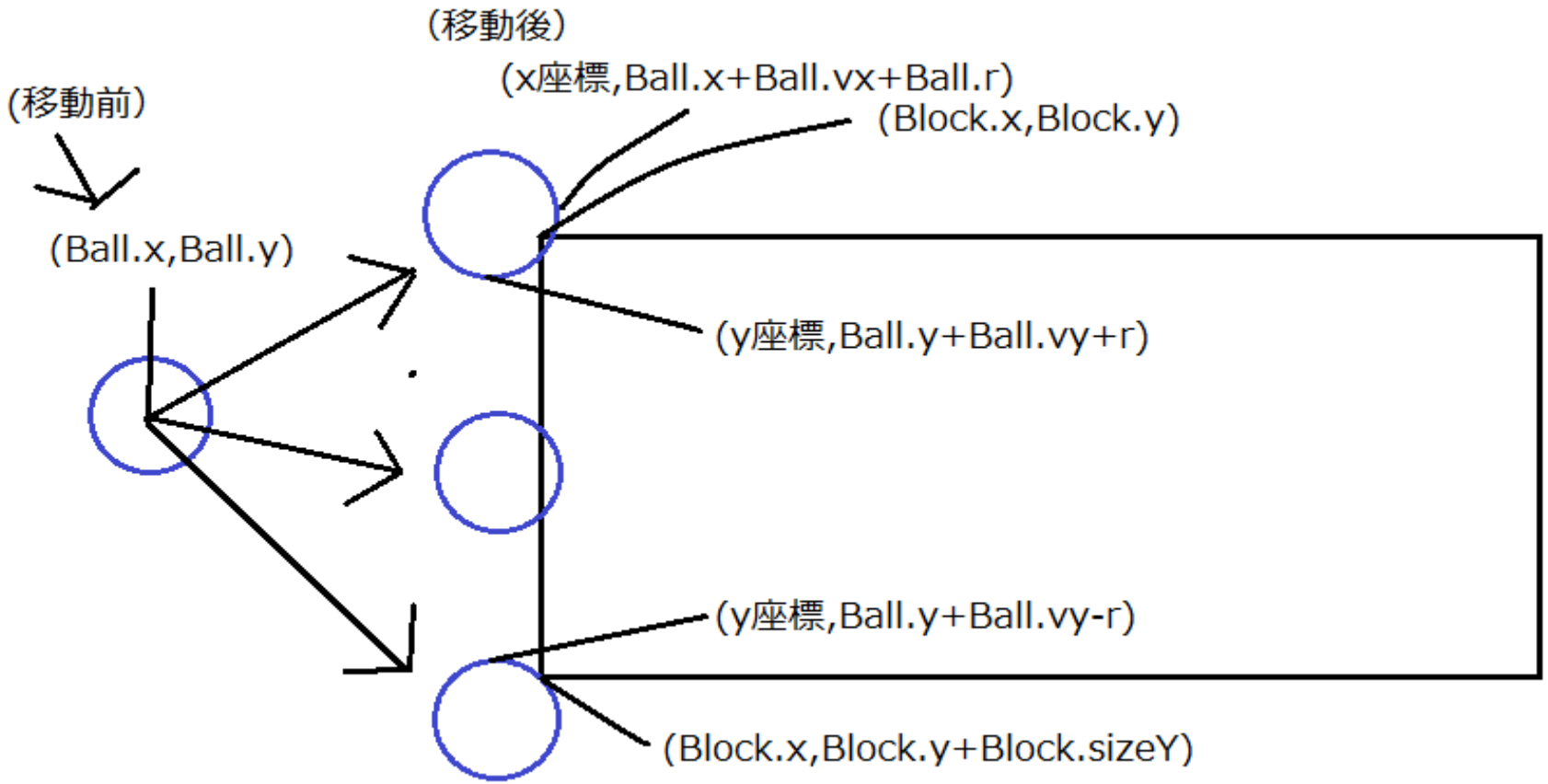
との順番で書いてやる。

よくわからなかったら
Part5開始時のソースをコピペしてもらって
構わない。

当たり判定の話

- まずはボールがブロックの左側に当たるときのことを考える。
- 意識したいのはボールの右端のx座標と
  ブロックの左端のx座標
- そして、ボールの一番上と一番下のy座標
  ブロックの一番上と一番下のy座標。

ブロック左側の当たり判定



ボールの右はじのx座標はBall.x+Ball.vx+Ball.r ブロックの左端のx座標はBlock.x
よって Ball.x+Ball.vx+Ball.r >= Block.xの時は接触してると見なせる
さらに条件としてはボールの一番下の座標がブロックの一番上の座標より大きいこと。
ボールの一番上の座標がブロックの一番下の座標より小さいことが条件。
気を付けなければいけないのは、→がx正の方向、↓がy正の方向。

関数を作り始める。

- 今回Blockの関数とするか、Ballの関数とするか迷ったが、ボールとブロックがあたった時、

  ブロックのx,y座標は変わらないが、ボールのvx,vy成分が変化するので、ボールのCalc関数内に書くこととする。
- Calc関数がボールの移動制御もしているので

  都合がいいと判断した。

ボールとブロックの当たり判定

```
void Ball_obj::Graph(){
    DrawCircle(x,y,r,color,TRUE);
}
void Ball_obj::Calc(){
    //ボールが場外に行ってしまった時。
    if( x + vx >= 640 || x + vx <= 0 ){
        vx *= -1;
    }
    if( y + vy >= 480 || y + vy <= 0 ){
        vy *= -1;
    }
    //ボールとブロックの当たり判定、
    for(int i=0;i<BLOCK_X_NUM;i++){
        for(int j=0;j<BLOCK_Y_NUM;j++){

            //ブロックの左に当たる時。
            if( x < Block[i][j].GetX() && x + vx + r >= Block[i][j].GetX() &&
                y + vy +r >Block[i][j].GetY() &&
                y + vy - r<= Block[i][j].GetY() + Block[i][j].GetSizeY()){
                    vx *= -1;
            }

        }
    }

    x+=vx;
    y+=vy;
}
```

Calc関数を変更

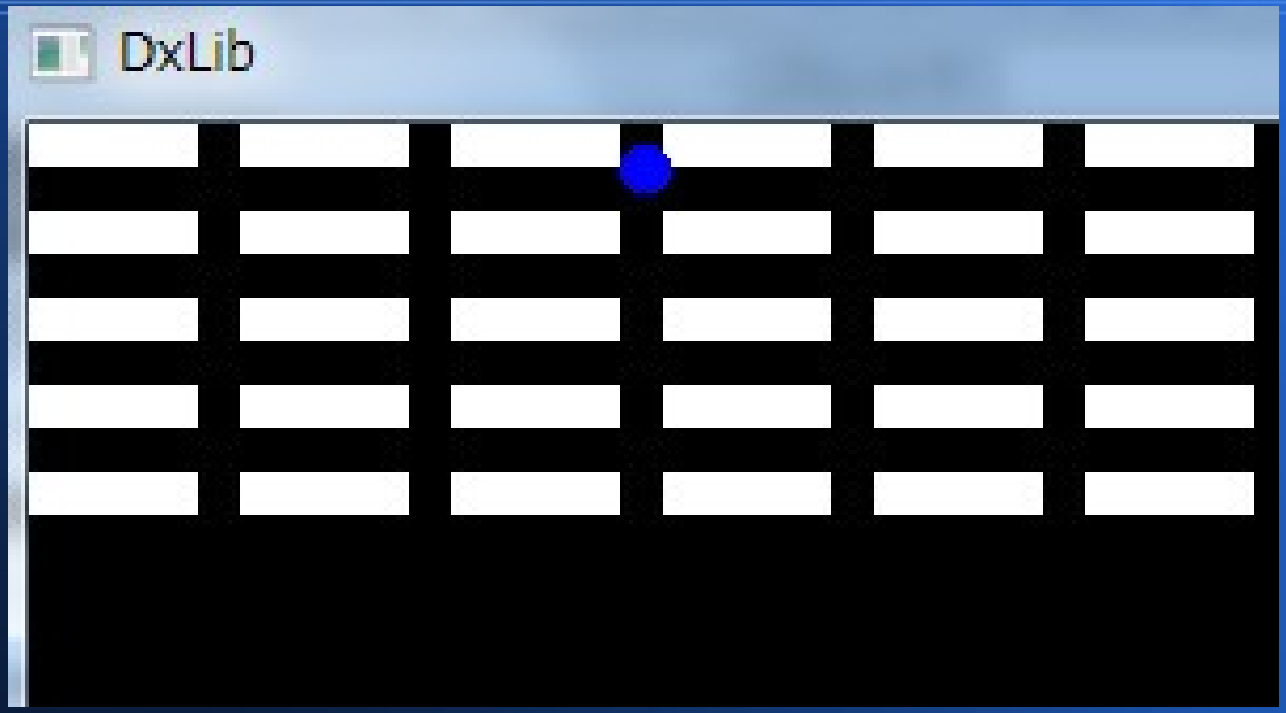
ブロック全てについて
あたってるかどうか判定
するので、forのループで
判定する。

If文の1行目は
ボールの右はじが
ブロックの左端を
貫いているか。

2行目、3行目は
ボールの上と下が
ブロックの範囲に
収まっているか。

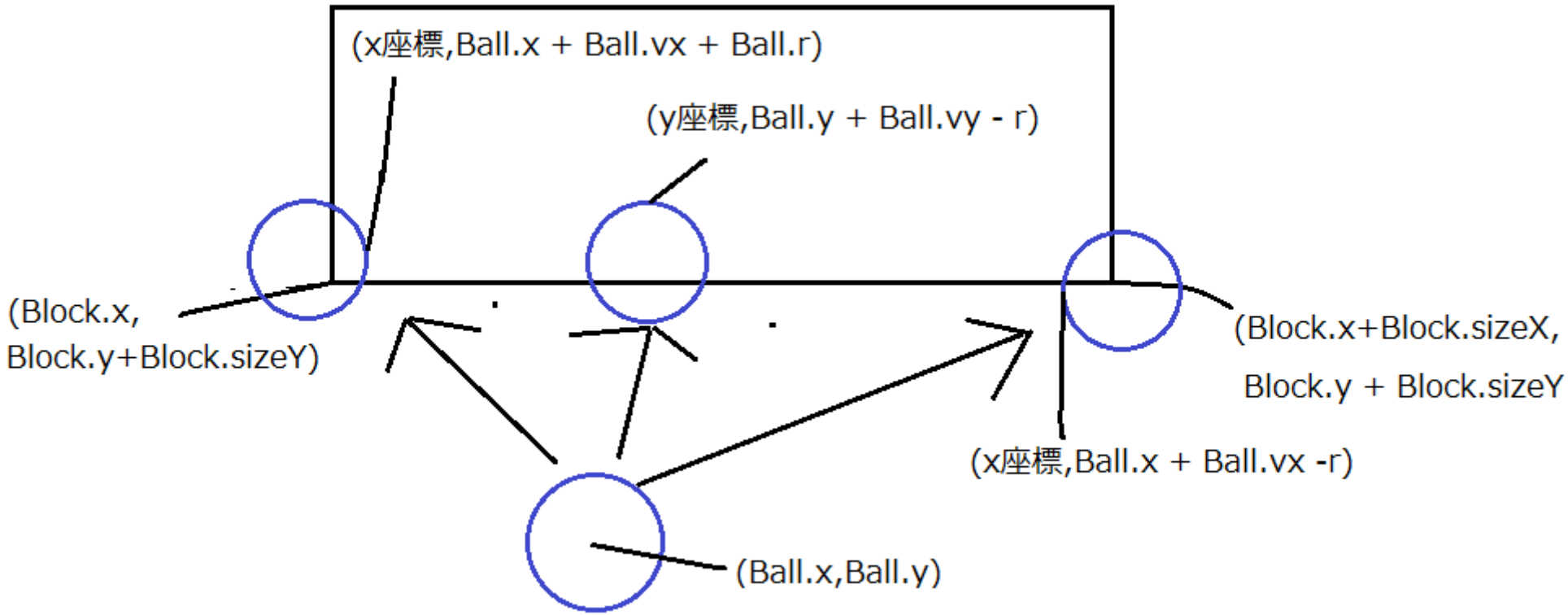
しっかりコメントアウトも
しておいたほうがいいだろう。

実行結果



もちろん、まだブロック左側の判定しか作っていないので、
めりこんでしまうことがある。
ただ、ブロックの左側でボールが跳ね返ったのがわかっただろうか。
これを上下右にも同じように作ってやる。
また、ボールがブロックに当たった時、ブロックを消すのも取り入れれば
ほぼ完成となる。

ブロックの下側の判定



下側の判定も同じように、ボールの右端、左端、一番上、一番下を気にする。
今回は移動後のボールの右端（Ball.x+Ball.vx+Ball.r)がBlockの左端より大きく
ボールの左端(Ball.x+Ball.vx-Ball.r)がBlockの右端より小さい。
そして,移動後のボールの一番上（Ball.y+Ball.vy – r)が
ブロックの一番下(Block.y + Block.sizeY)より小さくなればよい。

ブロックの下側の判定

```
}
//ボールとブロックの当たり判定、
for(int i=0;i<BLOCK_X_NUM;i++){
    for(int j=0;j<BLOCK_Y_NUM;j++){

        //ブロックの左に当たる時。
        if( x < Block[i][j].GetX() && x + vx + r >= Block[i][j].GetX() &&
            y + vy +r >Block[i][j].GetY() &&
            y + vy - r<= Block[i][j].GetY() + Block[i][j].GetSizeY()){

                vx *= -1;

        }
        //ブロックの下に当たる時。
        if ( y > Block[i][j].GetY() + Block[i][j].GetSizeY() &&
            y + vy - r <= Block[i][j].GetY() + Block[i][j].GetSizeY() &&
            x + vx + r >= Block[i][j].GetX() &&
            x + vx - r <= Block[i][j].GetX() + Block[i][j].GetSizeX()){

                vy *= -1;

        }

    }
}

x+=vx;
y+=vy;
```

Ball::Calc関数に追加。

このブロックの下に当たる時の
1行目は、移動前はあたってない
ということ。

2行目は移動後のボールの一番上
のy座標がブロックの一番下より
小さいということ。

3行目は移動後のボールの右端が
ブロックの左端より大きいということ

4行目は移動後のボールの左端が
ブロックの右端より小さいということ

実行結果

主にボールはブロックの一番下に当たると思うので、
ボールがブロックにめり込むことがなくなっただろう。
さて、後は同じようにブロックの上と右に判定をつくってやり、
その後、ブロックを消す作業を追加する。

ブロックの右と上の判定

```
//ブロックの左に当たる時。
if( x < Block[i][j].GetX() && x + vx + r >= Block[i][j].GetX() &&
    y + vy + r >Block[i][j].GetY() &&
    y + vy - r<= Block[i][j].GetY() + Block[i][j].GetSizeY()){

        vx *= -1;
        Block[i][j].Delete();
}
//ブロックの右に当たる時。
if( x > Block[i][j].GetX() + Block[i][j].GetSizeX() &&
    x + vx - r < Block[i][j].GetX() +Block[i][j].GetSizeX()&&
    y + vy + r >Block[i][j].GetY() &&
    y + vy - r<= Block[i][j].GetY() + Block[i][j].GetSizeY()){

        vx *= -1;
        Block[i][j].Delete();
}
```

```
//ブロックの下に当たる時。
    if ( y > Block[i][j].GetY() + Block[i][j].GetSizeY() &&
    y + vy - r <= Block[i][j].GetY() + Block[i][j].GetSizeY() &&
    x + vx + r >= Block[i][j].GetX() &&
    x + vx - r <= Block[i][j].GetX() + Block[i][j].GetSizeX()){

        vy *= -1;
        Block[i][j].Delete();
    }
//ブロックの上に当たる時。
if ( y < Block[i][j].GetY() &&
    y + vy + r >= Block[i][j].GetY() &&
    x + vx + r >= Block[i][j].GetX() &&
    x + vx - r <= Block[i][j].GetX() + Block[i][j].GetSizeX()){

        vy *= -1;
        Block[i][j].Delete();
}
```

図を書いて気づいたかもしれないが、
実は左と右の判定の関係は
Y方向については全く同じなので
ある程度はコピペで済ますことが可能。

また下と上の判定も同様で、
X方向についてはコピペで
済ますことが可能。

左の図のように制御文を
追加してやれば当たり判定が完成
ということになる。

ブロックを消す。

- ブロックを消すということを今回
  状態変数を一つ追加することで実装する。
- どういうことかと言うと、
  flagという変数を一つ追加して、
  ブロックがある状態をflag = 1
  ブロックがない状態をflag = 0とする。
- このことで、ボールがブロックに当たった時、
  flag = 0としてしまおう、ということ。

ブロックを消す。

```
class Block_obj{
private:
    int x,y;          //ブロックの左端の座標
    int sizeX,sizeY; //ブロックの横の長さ、縦の長さ
    int color;        //ブロックの色
    int flag;         //ブロックがあるかないか。 0:ない、1:ある
public:
    int GetX(),GetY();
    int GetSizeX(),GetSizeY();
    int GetFlag();
    void Set(int setX,int setY,int setColor); //ブロックの初期化を行う。
    void Graph();       //ブロックの描画
    void Delete();   //ブロックを消す関数
};
```

Block_objクラスに,privateな変数flagと
Publicな関数,GetFlag()とDelete関数を追加。
GetFlag()関数は、そのままprivateな変数、flagを返す関数。
Delete()関数は、ブロックのflagを0にする関数の予定。

ブロックを消す。

```
int Block_obj::GetSizeX(){
    return sizeX;
}
int Block_obj::GetSizeY(){
    return sizeY;
}
int Block_obj::GetFlag(){
    return flag;
}

void Block_obj::Set(int setX,int setY,int setColor){
    x = setX;
    y = setY;
    color = setColor;
    sizeX = 40;
    sizeY = 10;
    flag = 1;
}
void Block_obj::Graph(){
    if(flag == 1){
        DrawBox(x,y,x+sizeX,y+sizeY,color,TRUE);
    }
}
void Block_obj::Delete(){
    flag = 0;
}
```

変更点は左の図の赤線部

GetFlag()関数はflagを返す。

Set()関数内にflag = 1を追加。
これでブロックがある状態にする。

Graph()関数に
if(flag == 1)を追加
このことで、ブロックがある時のみ、
描画するようにしている。

そしてDelete()関数に
flag = 0と書いた。

これで、ブロックを消すことが
可能になった。

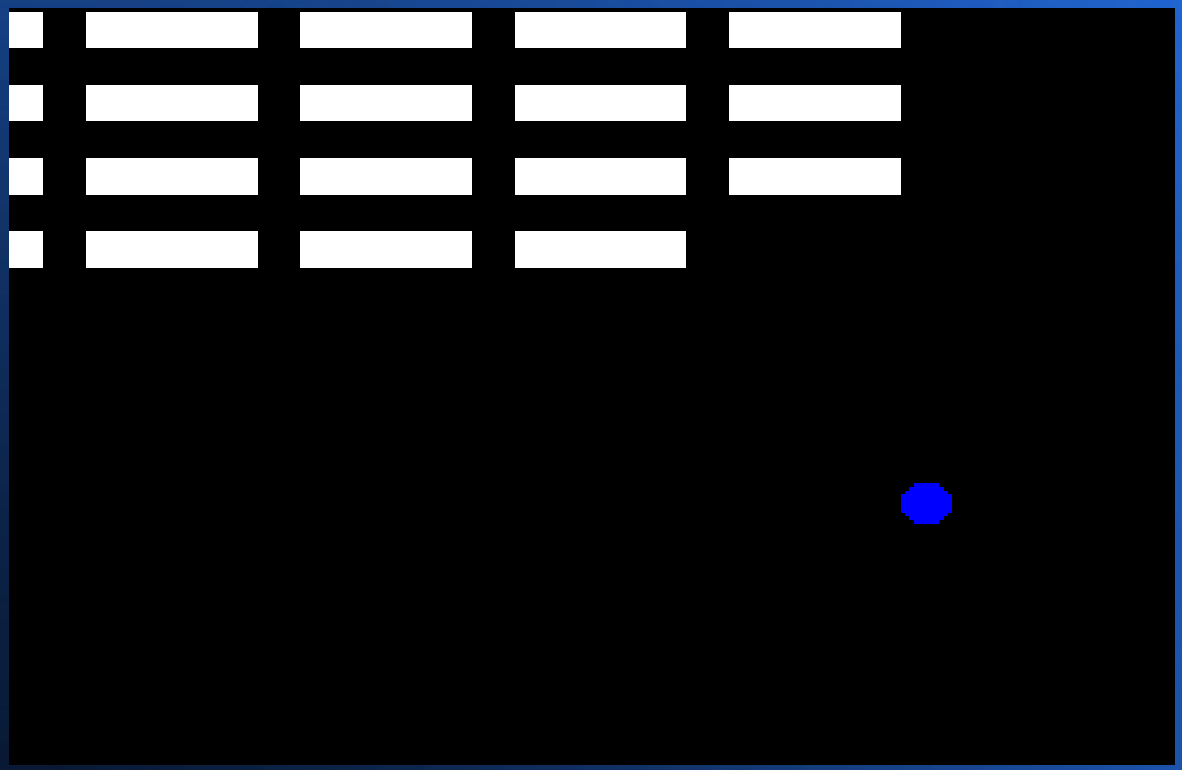
ブロックを消す。

```
//ブロックの左に当たる時。
if( x < Block[i][j].GetX() && x + vx + r >= Block[i][j].GetX() &
    y + vy + r >Block[i][j].GetY() &&
    y + vy - r<= Block[i][j].GetY() + Block[i][j].GetSizeY()){

        vx *= -1;
        Block[i][j].Delete();
}
//ブロックの右に当たる時。
if( x > Block[i][j].GetX() + Block[i][j].GetSizeX() &&
    x + vx - r < Block[i][j].GetX() +Block[i][j].GetSizeX()&&
    y + vy + r >Block[i][j].GetY() &&
    y + vy - r<= Block[i][j].GetY() + Block[i][j].GetSizeY()){

        vx *= -1;
        Block[i][j].Delete();
}|
//ブロックの下に当たる時。
if ( y > Block[i][j].GetY() + Block[i][j].GetSizeY() &&
    y + vy - r <= Block[i][j].GetY() + Block[i][j].GetSizeY() &&
    x + vx + r >= Block[i][j].GetX() &&
    x + vx - r <= Block[i][j].GetX() + Block[i][j].GetSizeX()){

        vy *= -1;
        Block[i][j].Delete();
}
//ブロックの上に当たる時。
if ( y < Block[i][j].GetY() &&
    y + vy + r <= Block[i][j].GetY() &&
    x + vx + r >= Block[i][j].GetX() &&
    x + vx - r <= Block[i][j].GetX() + Block[i][j].GetSizeX()){

        vy *= -1;
        Block[i][i].Delete();
}
```

そして、ボールがブロックに当たった時、Block[i][j]に対してDelete関数を使う。

これで、ブロックを消すことにしている。

実行結果



さて、消すことには成功したと思うが、今度はブロックが消えた場所でも
またボールの速度が変更されてしまっているだろう。
これは何故かというと、ブロックがある時ない時に関わらず、ブロックがあった場所で
ボールとブロックがある時は計算する、という命令になっているからである。

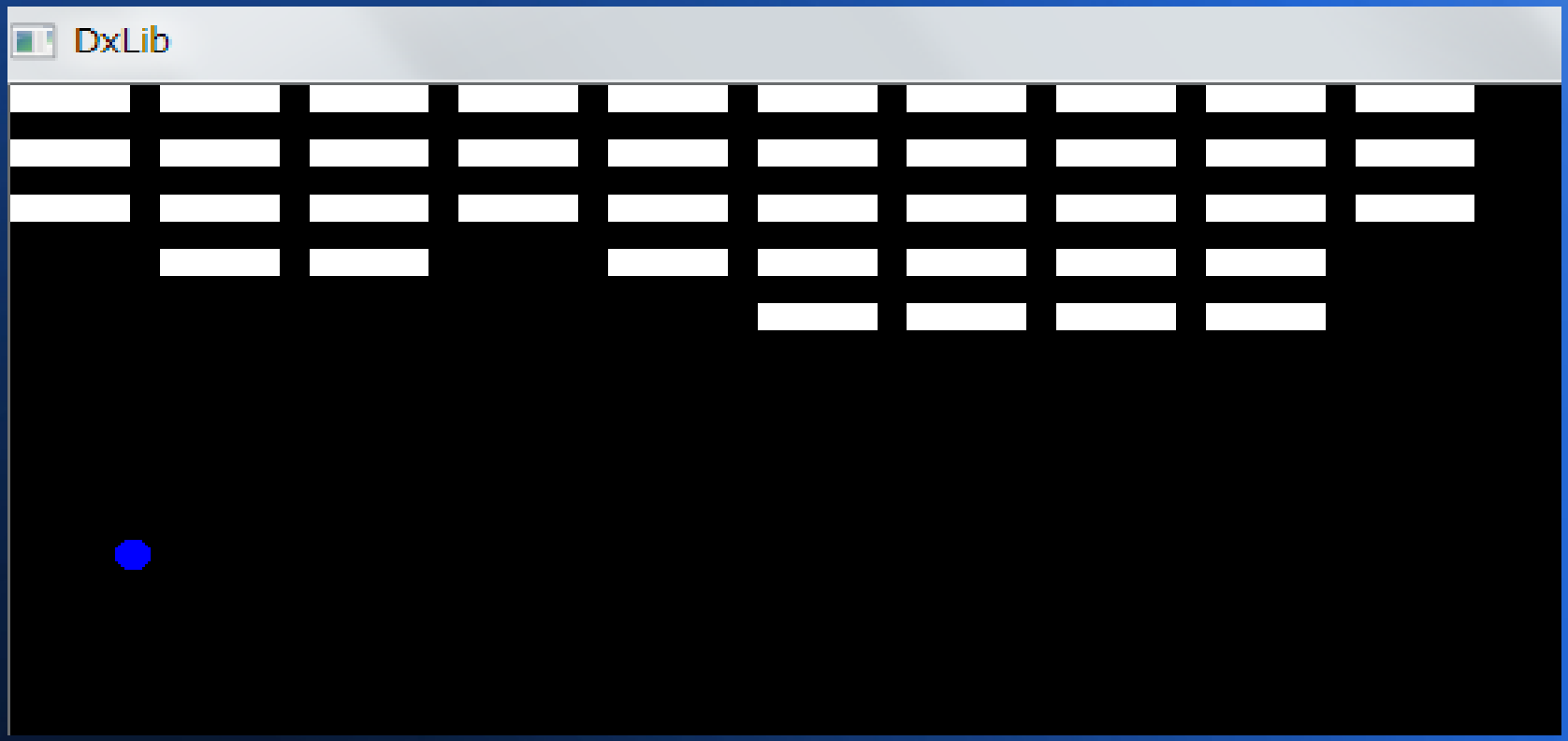
なので、ブロックがある時は計算をする、という命令も加えてやらなければならない。

改善方法

```
//ボールとブロックの当たり判定、
for(int i=0;i<BLOCK_X_NUM;i++){
    for(int j=0;j<BLOCK_Y_NUM;j++){
        //ブロックがある時は計算する。
        if(Block[i][j].GetFlag()==1){
            //ブロックの左に当たる時。
            if( x < Block[i][j].GetX() && x + vx + r >= Block[i][j].GetX() &&
                y + vy + r >Block[i][j].GetY() &&
                y + vy - r<= Block[i][j].GetY() + Block[i][j].GetSizeY()){

                    vx *= -1;
                    Block[i][j].Delete();
            }
            //ブロックの右に当たる時。
            if( x > Block[i][j].GetX() + Block[i][j].GetSizeX() &&
                x + vx - r < Block[i][j].GetX() +Block[i][j].GetSizeX()&&
                y + vy + r >Block[i][j].GetY() &&
                y + vy - r<= Block[i][j].GetY() + Block[i][j].GetSizeY()){

                    vx *= -1;
                    Block[i][j].Delete();
            }
            //ブロックの下に当たる時。
                if ( y > Block[i][j].GetY() + Block[i][j].GetSizeY() &&
                y + vy - r <= Block[i][j].GetY() + Block[i][j].GetSizeY() &&
                x + vx + r >= Block[i][j].GetX() &&
                x + vx - r <= Block[i][j].GetX() + Block[i][j].GetSizeX()){

                    vy *= -1;
                    Block[i][j].Delete();
                }
            //ブロックの上に当たる時。
            if ( y < Block[i][j].GetY() &&
                y + vy + r <= Block[i][j].GetY() &&
                x + vx + r >= Block[i][j].GetX() &&
                x + vx - r <= Block[i][j].GetX() + Block[i][j].GetSizeX()){

                    vy *= -1;
                    Block[i][j].Delete();
            }
        }
    }
}
```

画面全体を
使っているが、
変更箇所は
If文に
Flagが1の時
当たり判定があるか
を計算する。
という命令のみ
追加した。
大カッコが増えるので
忘れずに閉じること

実行結果



今回はブロックの無い所では当たり判定がないので、
このようにどんどんブロックが消せるようになっただろう。
ゲームとしてはもう80%は完成してると見てもいい。
後は自分が動かせるバーを作るだけだ。
それについて次回説明する。ただ、今までの知識＋αで作れるので自分で調べて作るのもいいだろう。

まとめ

- ある時や、ない時が存在する物体については

  flagという変数を追加してやればよい。
- 当たり判定を考える時は実際に図を書いて考えるのが非常に大事。