DXライブラリを用いて
ブロック崩しを作る Part 3

構造体を用いてブロック、ボールの描画

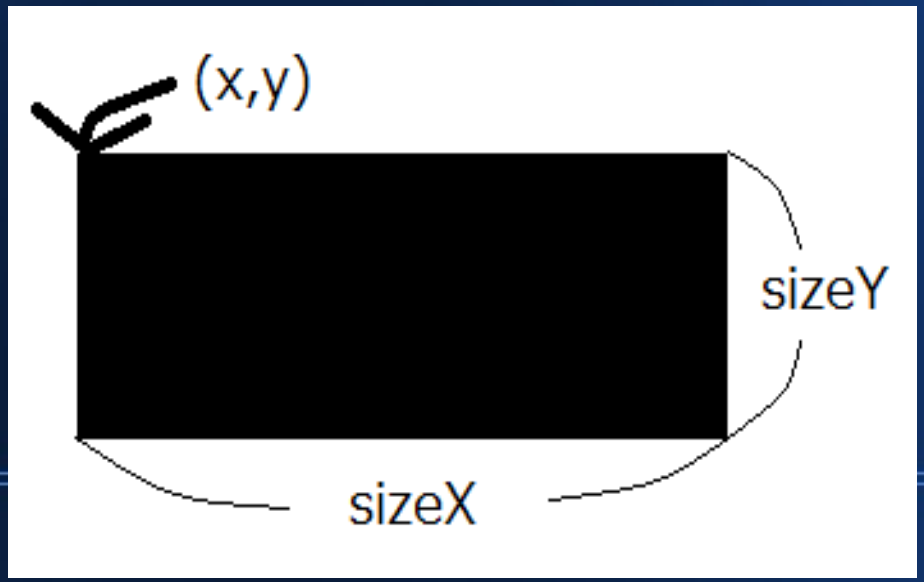
今回のスタートソース

```
#include "DxLib.h"

//x方向のブロックの数
#define BLOCK_X_NUM 10
//y方向のブロックの数
#define BLOCK_Y_NUM 5

// プログラムは WinMain から始まります
int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    LPSTR lpCmdLine, int nCmdShow )
{
    ChangeWindowMode(TRUE);//ウィンドウモードで起動

    DxLib_Init();   // ＤＸライブラリ初期化処理

    //裏画面に描画することを決定。
    SetDrawScreen(DX_SCREEN_BACK);

    while(ProcessMessage()==0){

        //描画されているものを消す。
        ClearDrawScreen();
        //メインループの動作はここに書く！！

        //裏画面の描画状態を表に反映
        ScreenFlip();
    }

    WaitKey() ;             // キー入力待ち

    DxLib_End() ;               // ＤＸライブラリ使用の終了処理

    return 0 ;              // ソフトの終了
}
```

Part2でやったものは
ほとんど消して
新しく作り直していく。

構造体って？？

- １つの変数のグループみたいなもの。
- ゲームなどを作る場合は一つ一つの物体で構造体を作り、変数などを１グループとしてまとめることが多い。



例えばブロックなら
左上の座標のx,y
縦、横の長さの
sizeX,sizeY,後は色

もしRPGのキャラクターを構造体で作るなら

```
typedef struct{

    int hp;      //体力
    int hpmax;   //体力の最大値
    int mp;      //魔法力
    int mpmax;   //魔法力の最大値
    int atk;     //攻撃力
    int def;     //防御力
    int mAtk;    //魔法攻撃力
    int mDef;    //魔法防御力
    int luk;     //運の良さ
    int speed;   //素早さ

}Chara_obj;
```

構造体の文法は

typedef sturct{
　(パラメータ)
}(構造体名);
という感じである。

構造体というのは
そのキャラの持っているパラメータを
まとめたもの。

と考えるとしっくりくるかもしれない。

左の例はRPGのキャラクターについて
構造体でまとめた例である。

これによって、
Chara_objという変数をまとめた
パッケージが完成した。

構造体の使い方

```
#include "DxLib.h"

//x方向のブロックの数
#define BLOCK_X_NUM 10
//y方向のブロックの数
#define BLOCK_Y_NUM 5

typedef struct{
    int x,y;
    int sizeX,sizeY;
    int color;
}Block_obj;


// プログラムは WinMain から始まります
int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    LPSTR lpCmdLine, int nCmdShow )
{
    ChangeWindowMode(TRUE);//ウィンドウモードで起動
```

赤線で囲った部分が構造体の宣言

これにより、Block_objという
変数のまとまったグループが作られた。

構造体の使い方

```
//裏画面に描画することを決定。
SetDrawScreen(DX_SCREEN_BACK);

//Block_objというパッケージを持ったBlockを宣言
Block_obj Block;



while(ProcessMessage()==0){
```

- 構造体のというのはグループを作っただけでは意味がなくて、しっかり宣言してやる必要がある。
- 宣言方法は普通のint型と同じように

  (構造体名) (変数名)のようにする。
- ここで作ったBlockのことをインスタンスと呼ぶ。

構造体の使い方

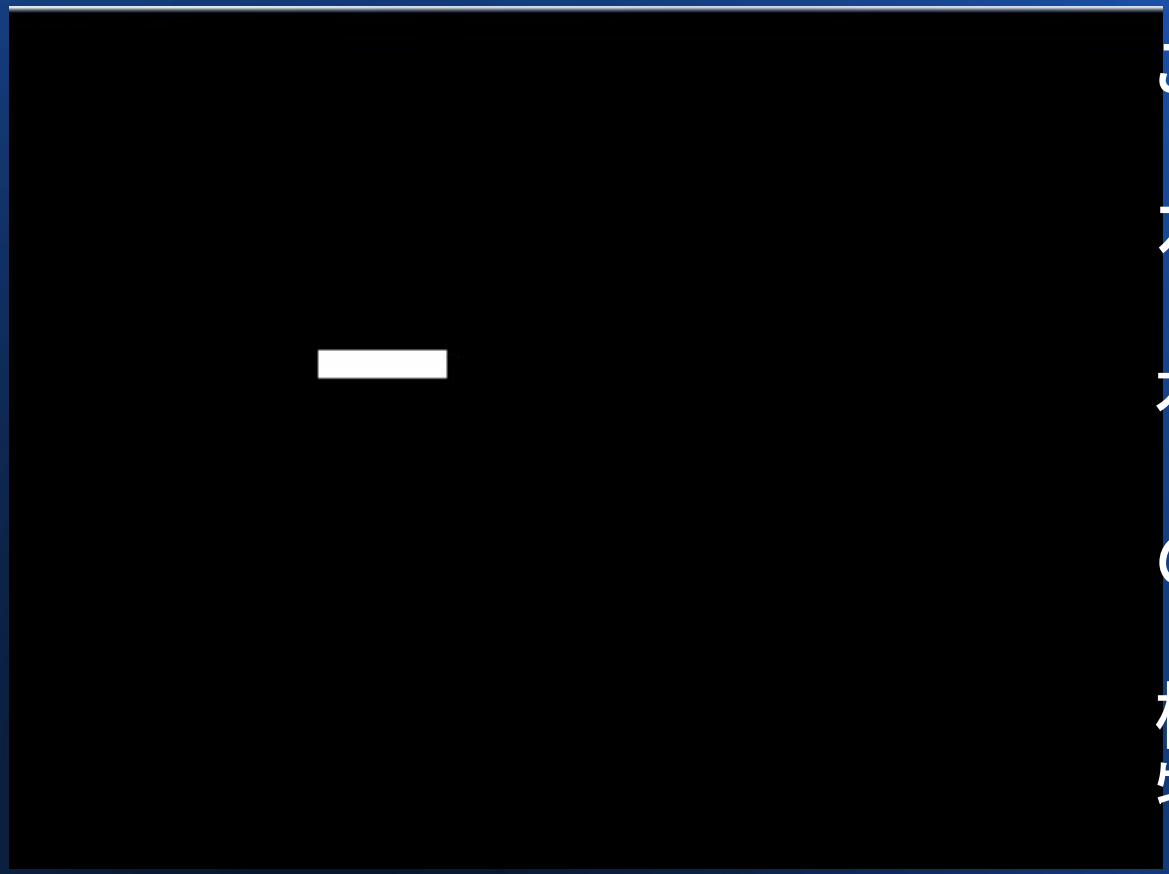
```
//Block_objというパッケージを持ったBlockを宣言
Block_obj Block;

Block.x=120;
Block.y=120;
Block.sizeX=50;
Block.sizeY=10;
Block.color=GetColor(255,255,255);

while(ProcessMessage()==0){

    //描画されているものを消す。
    ClearDrawScreen();
    //メインループの動作はここに書く！！
    DrawBox(Block.x,Block.y,Block.x+Block.sizeX,
                Block.y+Block.sizeY,Block.color,TRUE);
```

Block_obj Block;
という宣言により
BlockはBlock_objの
変数グループを扱うことが
できるようになる。

そしてその変数の
使い方は
←のようにBlock.(変数名)
のように行う。

そして初期化をして
メインループ内で実行する

実行結果



この結果は予測できただろう。

左上がBlock.x,Block.yの座標

右下がBlock.x+Block.sizeX,
Block.y+Block.sizeY
の座標

構造体のイメージとしては
物体がパラメータを持っている

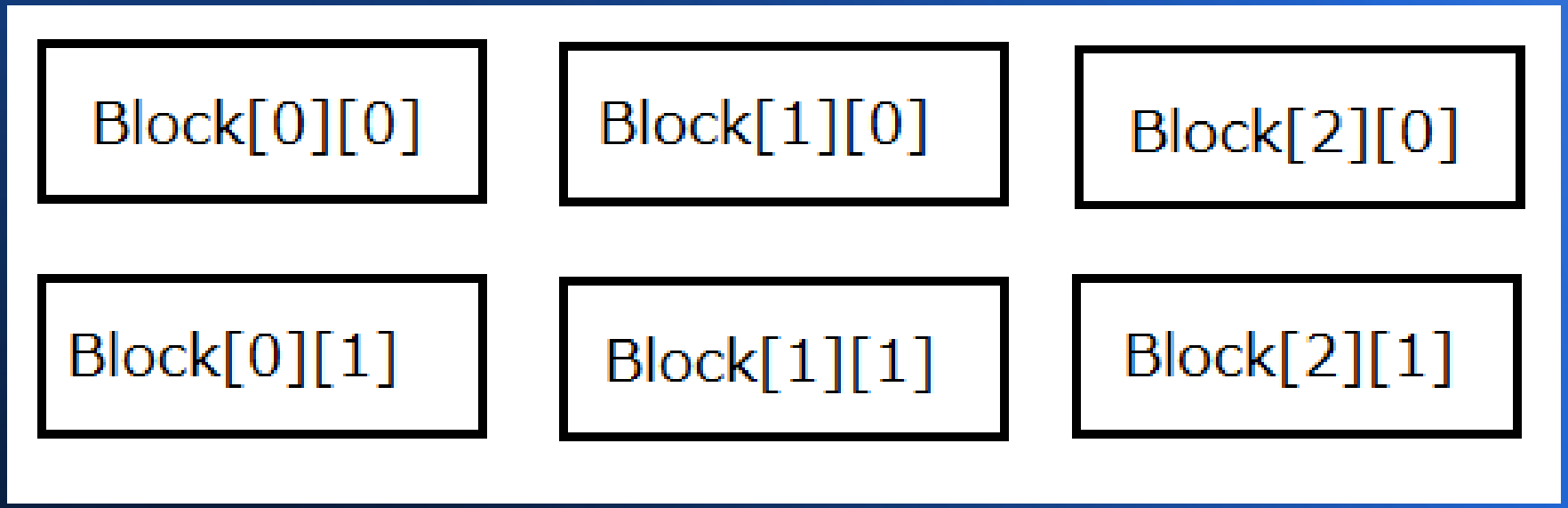
ということに慣れて欲しい。

ブロックを大量に作ってみる。

- Part 2では二重ループにて場所をずらしながらブロックを大量に描画した。
- しかし今回はひとつひとつのブロックがどこの座標にあるか、というのを意識し作る。
- そこで用いるのが構造体の二重配列。

ブロックを大量に作ってみる。

```
//Block_objというパッケージを持ったBlockを宣言
Block obj_Block[BLOCK_X_NUM][BLOCK_Y_NUM];
```



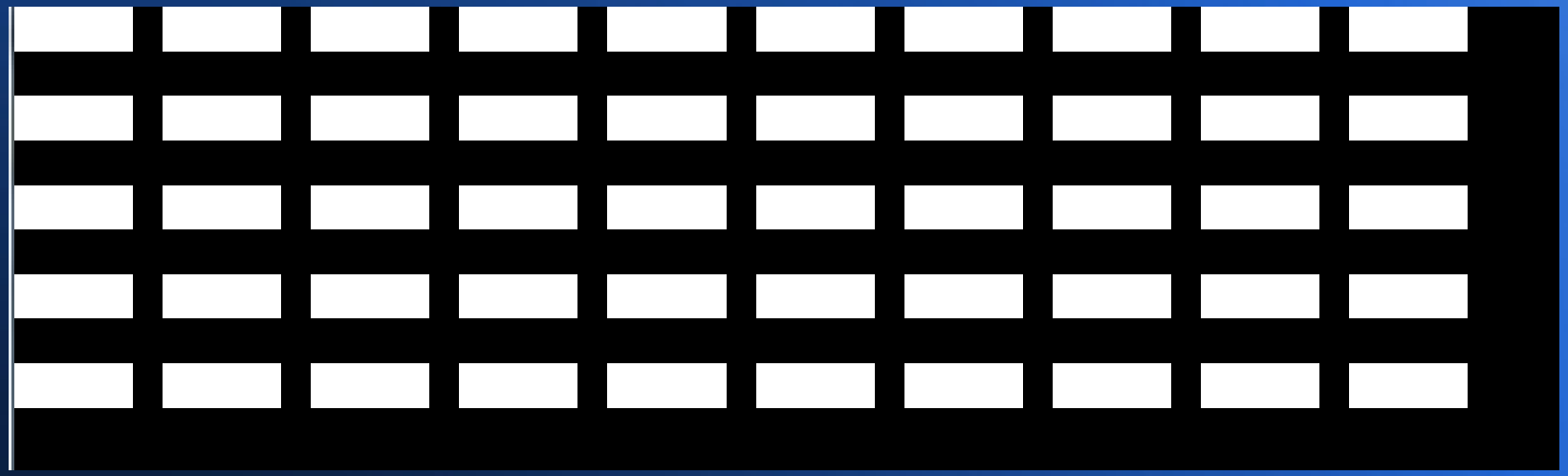
BLOCK_X_NUMとBLOCK_Y_NUMは宣言済み
ここが少し難しいかもしれないが二重配列によって
インスタンスを大量に作っていることが想像出来るだろうか。
さっきはBlockしか宣言していなかったので、
一つしかブロックを作れなかったが、今回は
Block[0][0],Block[0][1]・・・と大量にインスタンスを作ったのである。

ブロックを大量に作ってみる。

```
//ブロックの位置設定
for(int i=0;i<BLOCK_X_NUM;i++){
    for(int j=0;j<BLOCK_Y_NUM;j++){
        Block[i][j].sizeX = 40;
        Block[i][j].sizeY = 10;
        Block[i][j].x     = (Block[i][j].sizeX + 10)*i;
        Block[i][j].y     = (Block[i][j].sizeY + 10)*j;
        Block[i][j].color = GetColor(255,255,255);
    }
}

while(ProcessMessage()==0){

    //描画されているものを消す。
    ClearDrawScreen();
    //メインループの動作はここに書く！！

    //ブロックの描画
    for(int i=0;i<BLOCK_X_NUM;i++){
        for(int j=0;j<BLOCK_Y_NUM;j++){
            DrawBox(Block[i][j].x,Block[i][j].y,
                Block[i][j].x+Block[i][j].sizeX,Block[i][j].y+Block[i][j].sizeY,
                Block[i][j].color,TRUE);
        }
    }
```

変数名が長くなりがちなので、DrawBoxの引数がすごいことになってるが、
やってることはpart2と同じである。
ここでも意識したいのは一つ一つの物体がパラメータを持っているということ。

ブロックを大量に作ってみる。



実行結果は上のようになる。
これはpart2で大量にブロックを描画した時と同じ結果だろう。

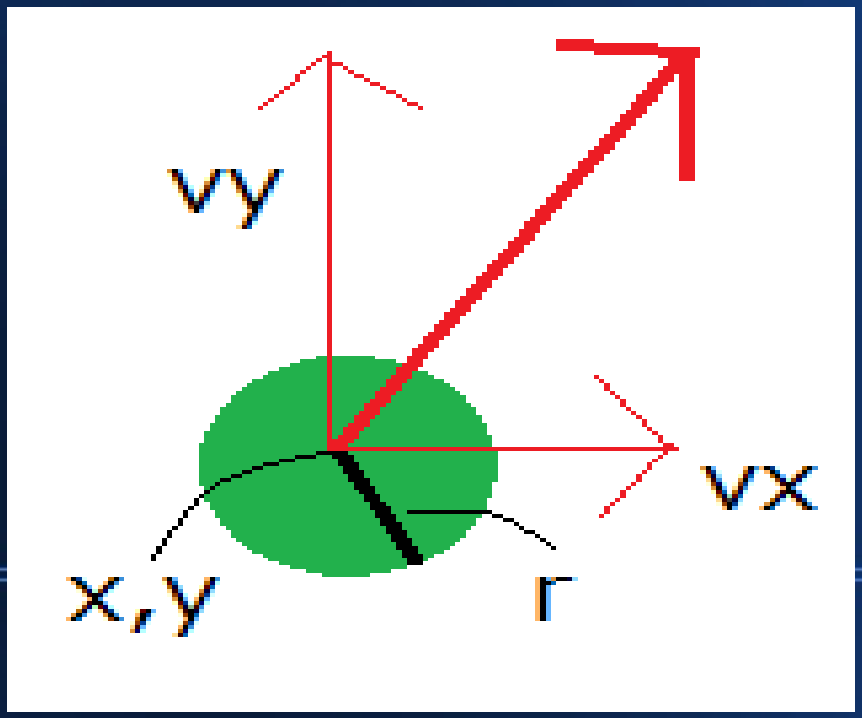
しかし、今回やったのは構造体で大量に
ブロックのインスタンスを作り、一つ一つに対して
パラメータを設定、そして描画したということが前回との違いである。

次はボールを作ってみる。

- ボールも構造体で作って見ることにする。
- ボールもひとつの物体としてみた時
  必要なパラメータを考えそれをパッケージ
  としてまとめあげる。
- とりあえず今回はボールのx,y座標
  そして、半径ｒ,x方向の速度vx,y方向の速度vy

ボールの構造体

```
typedef struct{
    int x,y,r;    //ボールの座標
    int vx,vy;    //ボールの速度
    int color;    //ボールの色
}Ball_obj;

// プログラムは WinMain から始まります|
int WINAPI WinMain( HINSTANCE hInstance,
```



ボールの構造体も同じように

typedef struct{

　　パラメータの宣言

}(構造体名);

にて作る。

パラメータのイメージとしては
左図を参照

ボールの構造体の初期値設定

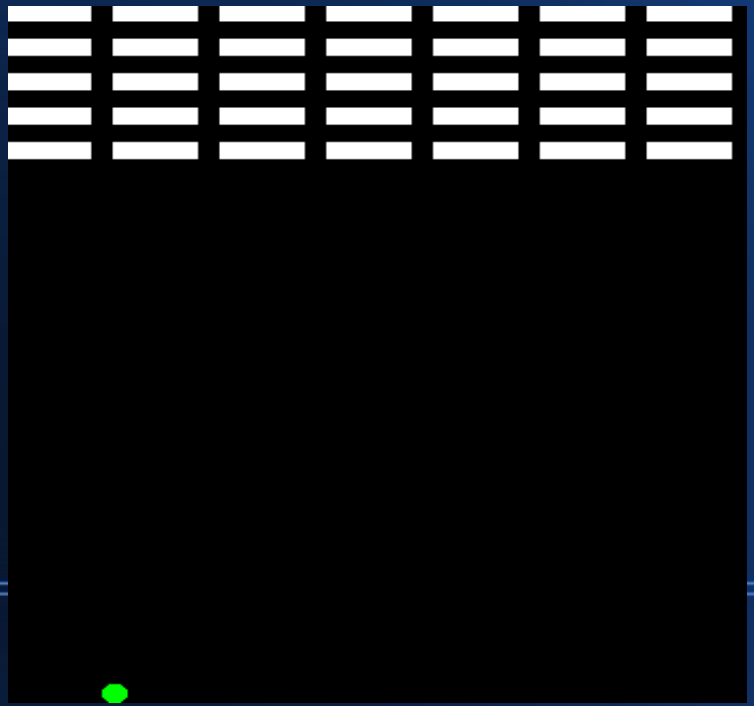
```
//Ball_objの変数を持ったインスタンスBallを宣言
Ball_obj Ball;

//ブロックの位置設定
for(int i=0;i<BLOCK_X_NUM;i++){
    for(int j=0;j<BLOCK_Y_NUM;j++){
        Block[i][j].sizeX = 40;
        Block[i][j].sizeY = 10;
        Block[i][j].x     = (Block[i][j].sizeX + 10)*i;
        Block[i][j].y     = (Block[i][j].sizeY + 10)*j;
        Block[i][j].color = GetColor(255,255,255);
    }
}

//ボールの初期値設定
Ball.x = 50;
Ball.y = 400;
Ball.r = 5;
Ball.color = GetColor(0,255,0);
Ball.vx = 5;
Ball.vy = -3;
```

Ball_obj Ballによって
Ballというインスタンスに
x,y,r,color,vx,vy
が一気にまとめられた。

これを左の画像の
下のように各自設定してやる。

ボールの描画

```
//ブロックの描画
for(int i=0;i<BLOCK_X_NUM;i++){
    for(int j=0;j<BLOCK_Y_NUM;j++){
        DrawBox(Block[i][j].x,Block[i][j].y,
            Block[i][j].x+Block[i][j].sizeX,Block[i][j].y+Block[i][j].sizeY,
            Block[i][j].color,TRUE);
    }
}
//ボールの描画
DrawCircle(Ball.x,Ball.y,Ball.r,Ball.color,TRUE);
```



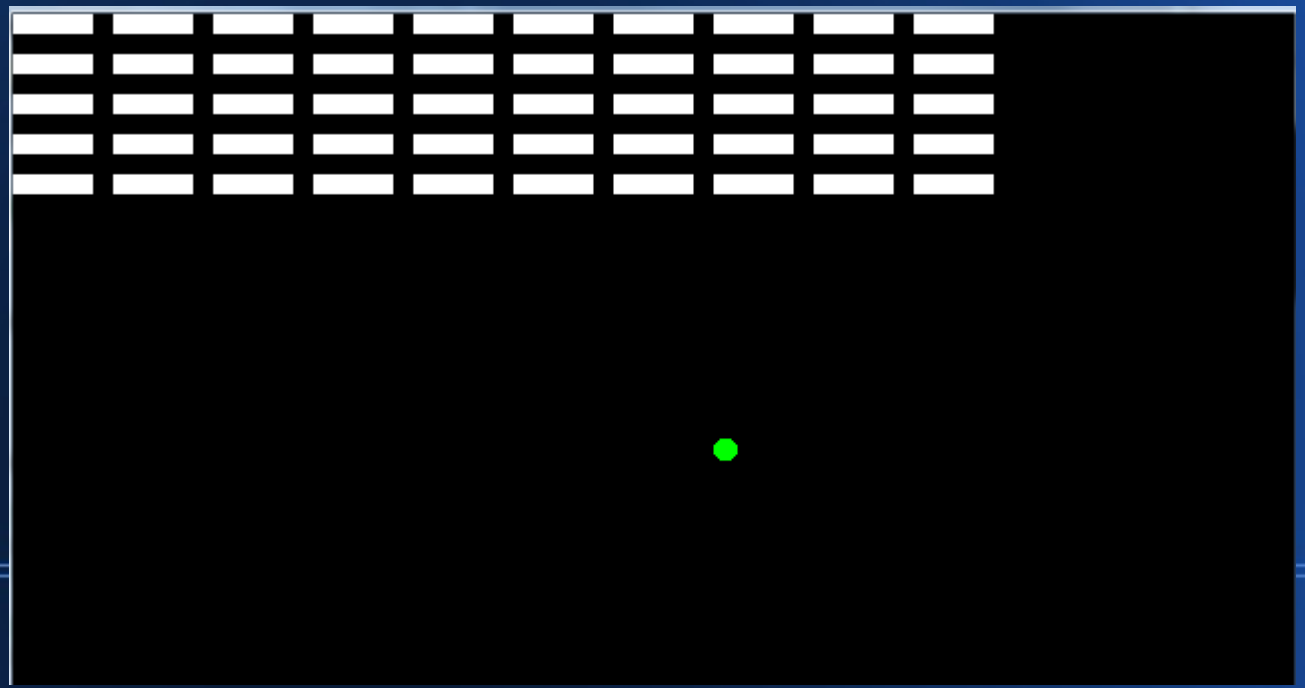
ブロックの描画の下にDrawCircleを
追加すると左の画像のように
球が描画されるだろう。

さて、つぎはこれを動かすのだが
どうしたよかっただろうか。

忘れてしまっている場合は
Part 1の資料を見てみよう。

ボールを動かす。

```
//ボールを動かす。
Ball.x += Ball.vx;
Ball.y += Ball.vy;

//ボールの描画
DrawCircle(Ball.x,Ball.y,Ball.r,Ball.color,TRUE);
```

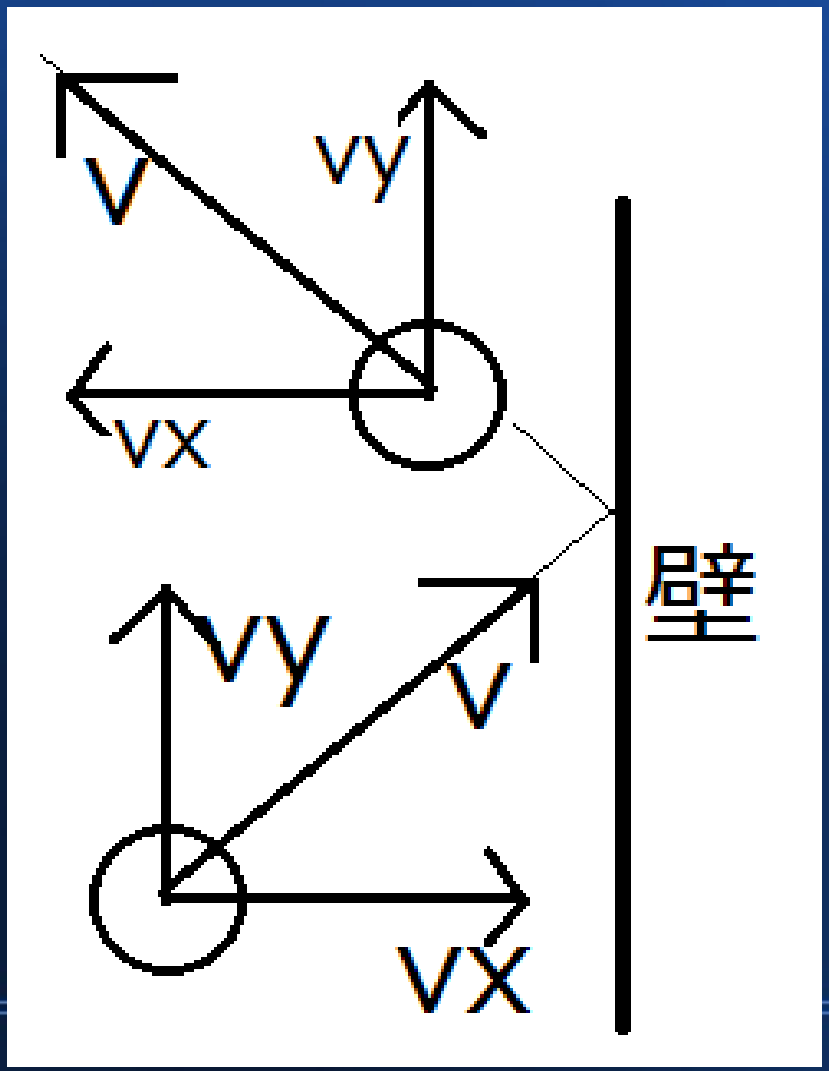


描画処理の前に
ボールのx,y成分に
それぞれvx,vy成分
加えてやれば良い

このようにしてから
プログラムを実行すると
球が動くのがわかるだろう。

さて、今回のパートでやる
最後のことは
壁で反射させることである。

ボールが壁にあたった時の反射



高校物理を習ったことがある人はわかると思われる。

横に進んでいて壁に球があたった場合は
反射係数等を一切無視すると、
Y方向の速度はそのまま。
X方向の速度は大きさがそのままで方向が逆になっている。
つまりx方向の速度は
−1倍されていることがわかる。

ボールが壁にあたった時の反射

画面の座標は上図のようになっている。
横の壁にぶつかった場合はx方向の速度を-1倍、同様に上下ならy方向の速度を-1倍。
このぶつかるというのは試しに弾を動かした時、壁を越えてしまうかどうか？で判定する。

ボールが壁にあたった時の反射

```
//ボールが壁に当たった時の処理
if( (Ball.x + Ball.vx)>=640 || (Ball.x + Ball.vx)<=0 ){
    Ball.vx *= -1;
}
if( (Ball.y + Ball.vy)>=480 || (Ball.y + Ball.vy)<=0 ){
    Ball.vy *= -1;
}
```

```
//ボールを動かす。
Ball.x += Ball.vx;
Ball.y += Ball.vy;
```

Ball.x + Ball.vx にて 移動後ボールのx座標はどこにあるかを計算
これが640を越えてしまった場合は壁を突き破ってるということになり
おかしいので速度を反転させる。
このことにより、壁で反射してるように見せかけているのである。

ボールが壁にあたった時の反射

どうしても静止画なので壁に反射してる様子が伝わりづらいが
前ページでの処理を加えたのなら、壁に反射するであろう。

まとめ

- ゲームプログラミングでは一つ一つの物体がパラメータを持っていて、それを処理してやるという仕組みで作ることが多い。
- 壁に反射するかどうか確かめる時は
  試しに動かした時、どこに行くかを計算し、
  そこで判定を行う。