

研究テーマ | ブロック崩しの制作

1. 研究の動機・目的

物理演算を用いたゲームを製作したいと思ったため。

2. 研究内容、結果

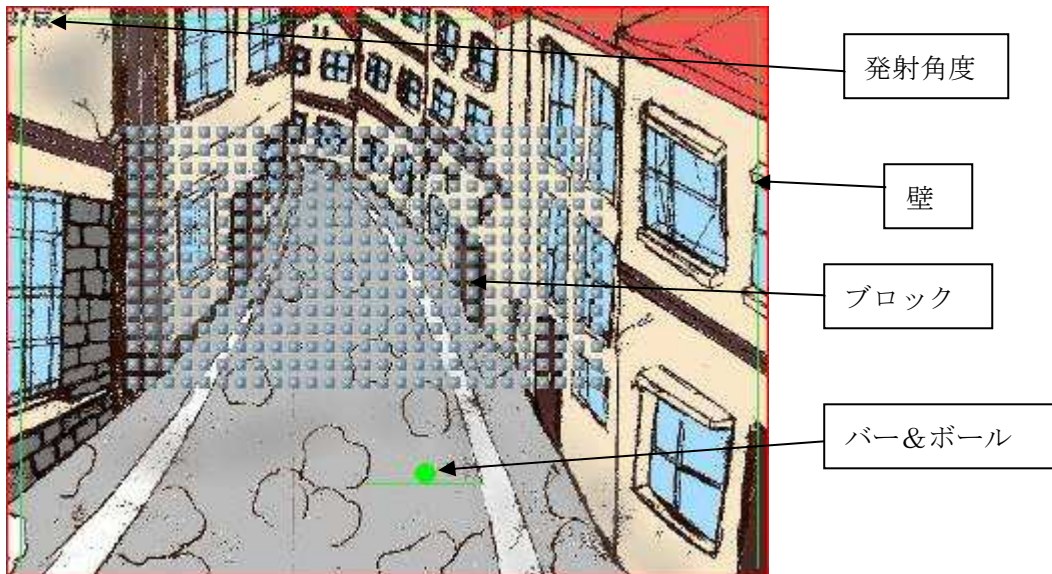
(1) 使用ソフト

プログラム Visual Studio C++

グラフィック DXライブラリ

(2) ゲーム概要

下のバーを使いボールを反射し画面にあるブロックをすべて消すことができればステージクリア、ボールを5回下に落としてしまうとゲームオーバーです。



※図はゲーム開始直後の画像（画像は開発中のものです）

操作キーについて

Zキー ボールを発射する

Xキー リセット

Qキー ボールを下に動かす

Wキー ボールを右に動かす

←キー バーを左に動かす

→キー バーを右に動かす

※隠しコマンドもあるよ！

解説、攻略

このブロック崩しはブロック数が多く（405個）普通にプレイするとクリアまでかなりの時間がかかりますが、Qキー、Wキーをうまく使うことで早くクリアすることができます、さらにボールの角度が90度180度付近のときはブロックを貫通するようになるので発射角度をよく見て利用しましょう。

(3) プログラム

```
int Handle=LoadGraph("画像/ブロック.jpg");  
int Handle2=LoadGraph("画像/写真3.jpg");  
int Handle3=LoadGraph("画像/玉.png");  
int Handle4=LoadGraph("画像/棒.png");  
DrawGraph(0,0,Handle2,TRUE);  
DrawGraph(posx-8,posy-8,Handle3,TRUE);  
DrawGraph(line[x].sx,line[x].sy,Handle4,TRUE);
```

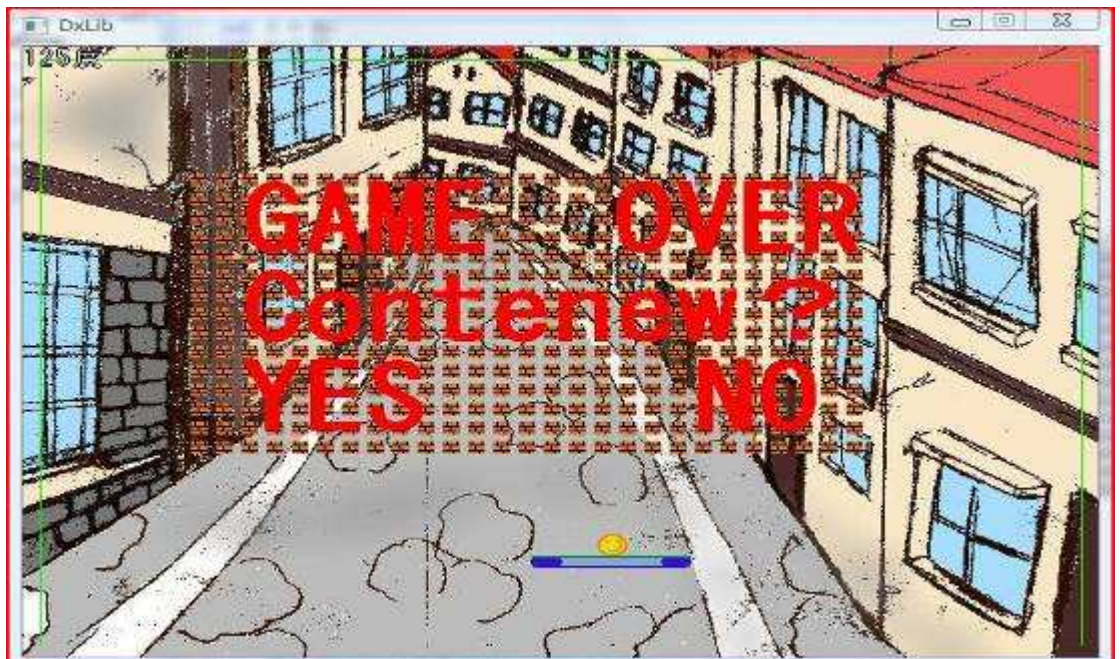
ここでは、玉や棒、ブロックをプログラミング上に肉付けのように重ねていきます。上のプレイ中の画像は棒と玉に肉付けをしていない状態で、ブロックは肉付けされている状態です。ちなみに写真3と出ているのは背景のことです。

次にバー（棒）が動く仕組みとボールが飛んでいく仕組みです。最初の二行はバーを、キーを使って動かすプログラムで、押してる間右左に 60 ずつ動く仕組みになっています。この+-を大きくすることでバーが速く動くようになります。そして発射される時の角度の方向に行く命令が 4&5 行目となります。ここでは sin と cos を使用して向きを決めます。angle という変数の中には角度がランダムで入るようになっています。(0 度~180 度)

```
if(zanki > 4){
    DrawString( 130 , 170 , "Continue?" , col );
    DrawString( 130 , 100 , "GAME OVER" , col );
    DrawString( 130 , 240 , "YES" , col );
    DrawString( 400 , 240 , "NO" , col );
    flag=2;

if( Mouse & MOUSE_INPUT_LEFT && g > 130 && g < 240 && h >240 && h < 310 )
    flag=0;
    zanki=0;
}
if( Mouse & MOUSE_INPUT_LEFT && g > 400 && g < 472 && h >240 && h < 310 )
    flag=0;
    zanki=0;
    for(int i=0; i<BOXNUM; i=i+1){
        box[i].sx=mo[i].a;
        box[i].sy=mo[i].b;
        box[i].ex=mo[i].c;
        box[i].ey=mo[i].d;
    }
    InitSoundMem();
    f = 0;
    function_status=0;
}
```

そして次は、残機が無くなった時に起こる行動で簡単にいえば GAME OVER が表示されコンテニューするかどうかを決めます。八行目と十二行目はマウスのポインタがある一定の座標まで来て、左クリックすると反応するような命令となっています。実際にやってみるとわかるのですが文字の YES にあたるところが、八行目で NO にあたるのが十二行目となっています。(ついでに下の画像は玉と棒も肉付けしました。)



```

if(key & PAD_INPUT_M && g>60 && g<570 && h>70 && h<350){
fspeed2 = erpm/60*2*PIE;
angle2 = angle2 + fspeed2 * frametime;
posx = g + cos(angle2)*hankei;
posy = h - sin(angle2)*hankei;
}
posx = posx + speedx * frametime;
posy = posy + speedy * frametime;
for(int j=0; j<LINENUM; j=j+1){
float ax,ay,bx,by;//2つのベクトルの成分を求める
ax=posx-line[j].sx; ay=posy-line[j].sy;
bx=line[j].ex-line[j].sx; by=line[j].ey-line[j].sy;
float inpro = ax*bx+ay*by;
float bl = bx*bx+by*by; //直線(ベクトルB)の長さの二乗
if(inpro>-ZER01 && inpro<(bl+ZER01)){ //lineとの衝突判定
//距離を求める
if( (ax*ax+ay*ay)-pow(inpro/sqrt(bl),2) < 8*8 ){
float nx = by/sqrt(bl); //正規化された法線ベクトル
float ny = -bx/sqrt(bl);
float inpro2 = nx * -speedx + ny * -speedy; //内積
nx = nx * inpro2; //法線ベクトルの延長
ny = ny * inpro2;
speedx = (speedx + nx * 2) * BOUND_E;
speedy = (speedy + ny * 2) * BOUND_E;
posx = posx+speedx*frametime*3;
posy = posy+speedy*frametime*3;
}
}
}

```

このプログラムは最も大切なところで、ブロック崩しの中心となっているプログラムです。何をしているかという、簡単にいえば、当たり判定の計算をしています。

難しいことを書くと、とても大量の説明となってしまいますので簡単に書きます。

とりあえず数 B の授業で習うベクトルの計算を行っています。これは逐次ボールとライン[線]との距離などを計算してその数値が一定になったら、入射してきた角度と同じ角度でボールを反射します。

ここで、一番苦労したのは、この命令だけだと線に対してだけなのでブロックに対してはうまくいきませんでした。なぜかという、ブロックには辺が4辺あるのですが、プログラム上一本としてしか反応されなかったからです。

なので、僕たちは4辺すべてに対して当たり判定を作りました、なので上の式があと 3 つあることになります。



これはスタート画面になります。Enter を押すことでゲームが始まります。

3.まとめ

今回は、僕の我がままで、どうしても今までに習った数学の知識とプログラムの知識を生かせる、ゲームの製作をしたいと思い、それらの要素を入れたゲームとしてブロック崩しの製作あたりでした。そして、当り判定など数学の授業で習ったことが生かして制作できとても満足のいくつくりになりました。

ゲームを作っていくうちに何度も分からなかったり、エラーが起きたりして、それを乗り越え成功した時はとても感激でした。

そして、乗り越えるたびにプログラムに対するの関心や知識が付いてきたのは今回の課題研究での大きな成果だと思っています。

右も左もわからない状態からサイトを見て勉強を始めました 基本的には画像を組みこんだりスタート画面の作成やプログラムの補助などを担当しました。少しずつゲームらしくなっていくのが嬉しかったです。

最初は物理演算を使おうと言われていただけでどのようなゲームにするかは全くもって決まっていなく初めのほうはとにかく案を出していきました、しばらくしてブロック崩しを製作することになり果てしないプログラムをとにかく打ち続けたりしましたがその分形になった時はとても嬉しかったです。