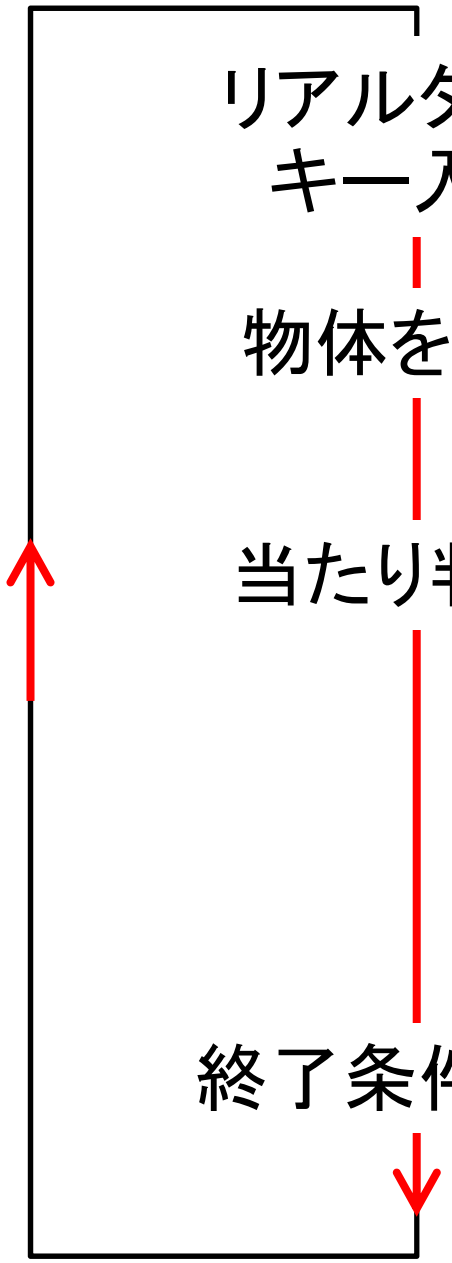
mlibを用いた物理シミュレーション制作（１）

ブロック崩し系の構成



リアルタイム
キー入力

左右キーを判定

物体を移動

パドルを移動 → キー入力
ボールを移動 → ボールの速度ベクトル、完全反射
ブロックの移動 → ブロックは移動しないが消える

当たり判定

パドルとボール　ボールの速度ベクトルy成分速度反転

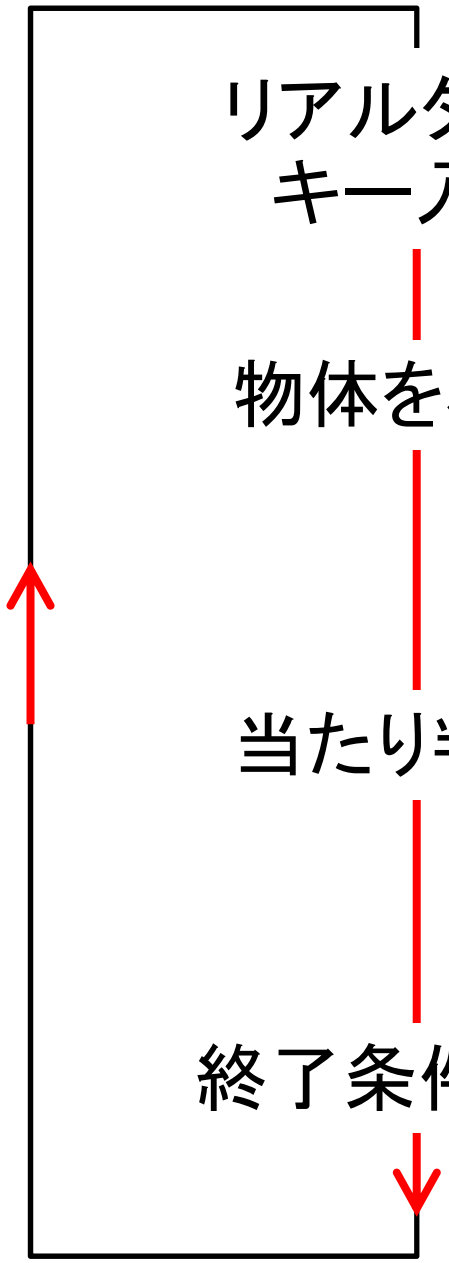
ボールと上壁　ボールのy成分速度反転
ボールと左右壁　ボールのx成分速度反転

ボールとブロック左右　ボールのx成分速度反転、ブロック消去
ボールとブロック上下　ボールのy成分速度反転、ブロック消去

終了条件判定

ボールがパドルより下に行ったとき。
ブロックをすべて消した。

インベーダ系の構成



リアルタイム
キー入力

左右キーを判定、
球発射ボタンでミサイルの初期位置と初速を与える

物体を移動

自機を移動	左右キーが押されている間移動
敵機を移動	一定の速度ベクトルの方向（横向き）
ミサイルを移動	ミサイルを上方向に移動

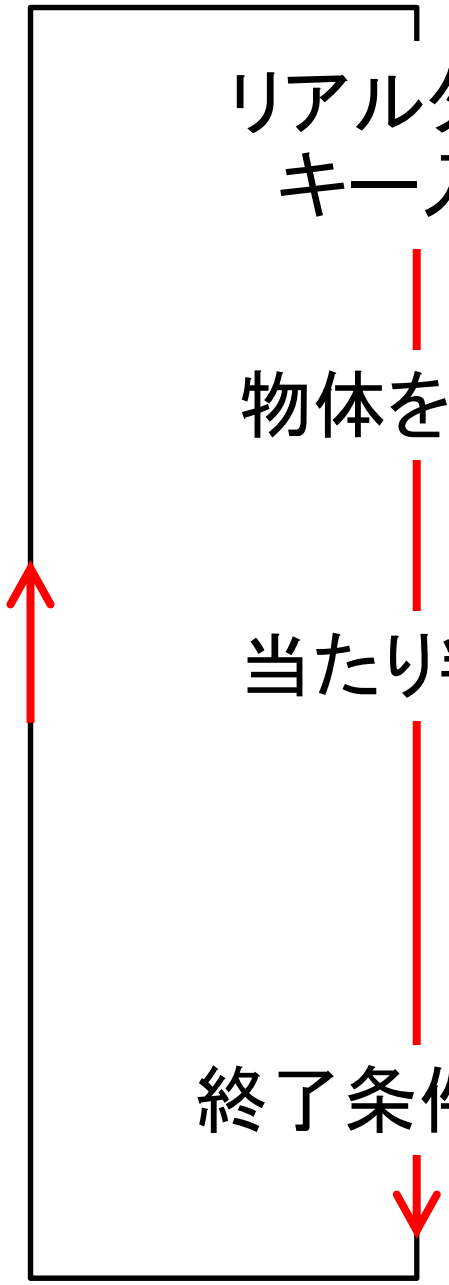
当たり判定

ミサイルと敵機	敵機を消して得点追加
敵機と壁	敵機の位置を下に下げて、移動向きを反転、敵機の速度アップ
ミサイルと上壁	超えたときミサイルを消去して

終了条件判定

敵機をすべて撃ち落としたとき、
敵機に侵略されたとき、

テトリス系の構成



リアルタイム
キー入力

左右キー、回転キーを判定

物体を移動

ブロックを移動	左右キーが押されている間
ブロックを回転	回転キーが押されたとき
ブロックの重力落下	加速度gで自由落下

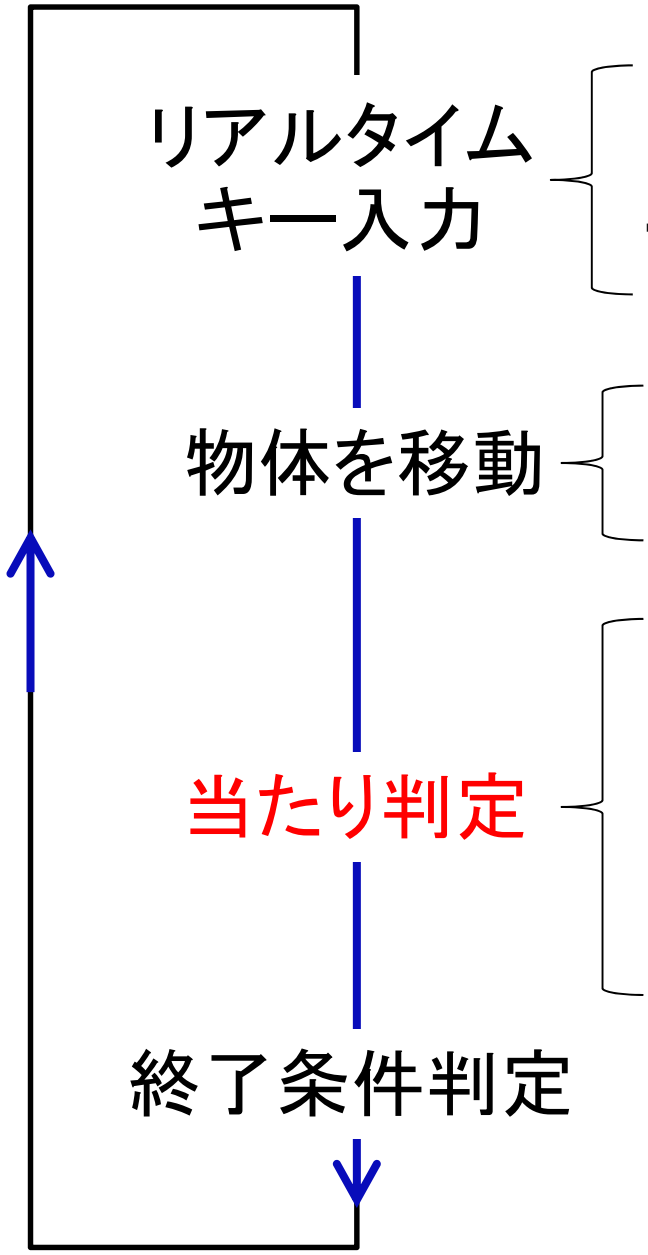
当たり判定

ブロックと固定ブロック	当たったとき移動できない
ブロックと左右壁	当たったとき移動できない
ブロックと下壁、固定ブロック	移動ブロックを固定化

終了条件判定

ブロックを固定したとき、横ラインがそろっていたらそのラインを消し、得点追加、その上のブロックを下げる。
ブロックが上までつみあがったら終了

物理シミュレーション



リアルタイム
キー入力

・方向キーが押されている間、その方向に速度 v 、加速度 a 、角速度 ω を与える

・あるキーが押されたとき、物体の移動したり、回転したり（状態変化開始のトリガー）

物体を移動

・物体の速度vと、微小時間Δtにより、移動量を求め、現在位置から次の位置を算出

当たり判定

・物体１の座標を、物体２の座標と比べる
　物体１，２が質点か？円か？四角形か？

・物体の座標があまり細かくないとき、座標を仮想空間（２次元グリッド）に変換し、仮想空間内座標が一致したかどうか判定する

終了条件判定

様々

質点とのCollision 判定

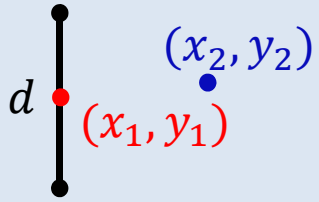
質点　大きさを持たない点　位置 (x, y)

点どうしの当たり判定は、座標が完全に一致　$x_1 = x_2 , y_1 = y_2$

(x_1, y_1) ●　●(x_2, y_2)

線分　太さを持たない有限長の線　中心位置 (x, y)　長さd

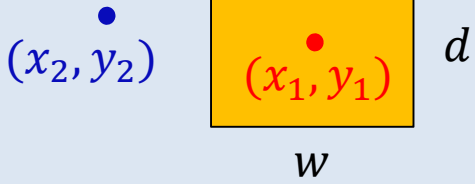
線分と平行な方向の座標が一致 $(y_1 = y_2)$　かつ
線分の中心と質点との距離が線分半分の長さ以下



$$x_1 - \frac{d}{2} < x_2 < x_1 + \frac{d}{2} \Rightarrow |x_1 - x_2| < \frac{d}{2}$$

四角　中心位置 (x, y)　奥行d　幅w

質点2が、四角の中に入る条件



$$x_1 - \frac{d}{2} < x_2 < x_1 + \frac{d}{2}$$

かつ $y_1 - \frac{w}{2} < y_2 < y_1 + \frac{w}{2}$

$$\Rightarrow |x_1 - x_2| < \frac{d}{2}$$

かつ $|y_1 - y_2| < \frac{w}{2}$

大きさを持つ物体どうしのCollision 判定

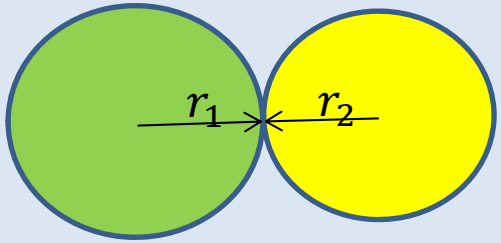
物体が円の場合

パラメータは中心間距離と半径

中心間距離は

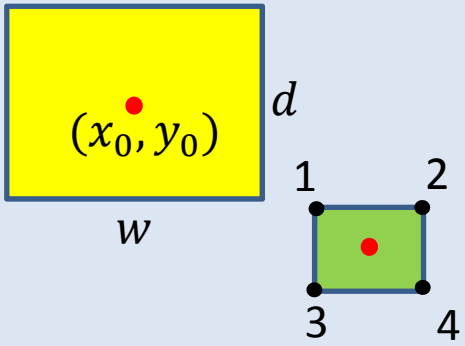
$$L = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

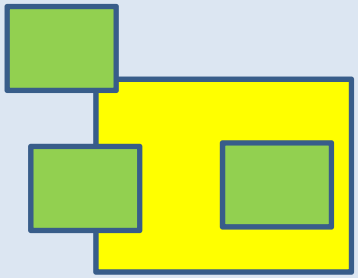
衝突の条件は　　$L < r_1 + r_2$



物体が長方形の場合

小さい方の長方形について、その角の4点を質点として、
そのうちいずれかが四角の中にあれば衝突とみなす。





$$(|x_0 - x_1| < \frac{d}{2} \text{ and } |y_0 - y_1| < \frac{w}{2}) \text{ or}$$

$$(|x_0 - x_2| < \frac{d}{2} \text{ and } |y_0 - y_2| < \frac{w}{2}) \text{ or}$$

$$(|x_0 - x_3| < \frac{d}{2} \text{ and } |y_0 - y_3| < \frac{w}{2}) \text{ or}$$

$$(|x_0 - x_4| < \frac{d}{2} \text{ and } |y_0 - y_4| < \frac{w}{2})$$

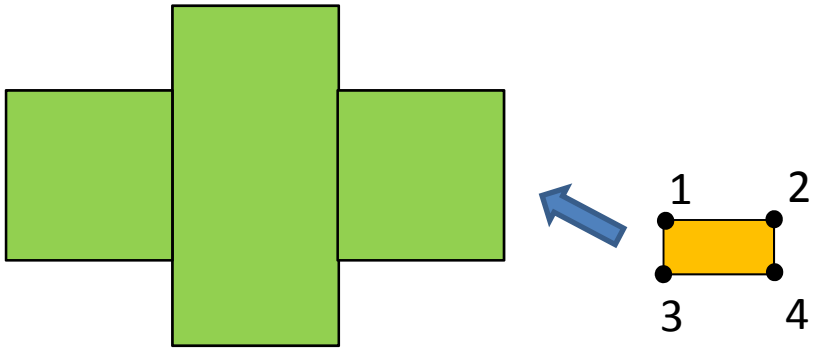
Collision 判定(1)　座標による判定

物体2 (x_2, y_2) / 物体1 (x_1, y_1)	点 ●	線分 d_2	半径r_2	d_2 / w_2
点 ●	((x1==x2) && (y1==y2))	((abs($y_1 - y_2$) < $d_2/2$) &&(x_1==x_2))	$\sqrt{(x_1-x_2)^2 + (y_1-y_2)^2} < r_2$	((abs($x_1 - x_2$) < $d_2/2$)&& (abs($y_1 - y_2$) < $w_2/2$))
線分 d_1	d_1が物体2のサイズより小さいとき→線分1を端点2点の集合体とみなす			
	(((abs($y_1 - d_1/2 - y_2$) < $d_2/2$) || (abs($y_1 + d_1/2 - y_2$) < $d_2/2$)) &&(x_1==x_2))	$\sqrt{(x_1-x_2)^2 + (y_1-d_1/2-y_2)^2} < r_2$ かつ $\sqrt{(x_1-x_2)^2 + (y_1+d_1/2-y_2)^2} < r_2$	(((abs($x_1 - x_2$) < $d_2/2$) && (abs($y_1 - d_1/2 - y_2$) < $w_2/2$)) && (abs($y_1 + d_1/2 - y_2$) < $w_2/2$))
半径r_1		$\sqrt{(x_1-x_2)^2 + (y_1-y_2)^2} < r_1 + r_2$		$d_2, w_2 < r_1$のとき 四角を角の4つの点の集合体とみなし、各点毎に円と点の判定を行い、どれか一つでも含まれて入ればよい
d_1 / w_1				$d_2, w_2 < d_1, w_1$のとき 物体2の四角を角の4つの点の集合体とみなし、各点毎に四角1と点の判定を行い、どれか一つでも含まれて入ればよい

・複雑な形どうしの判定は多くのif文が必要、一旦書いてしまえばあとは楽

Collision 判定の簡略化

ある大きさを持つ物体どうしのCollision判定は複雑になっていく

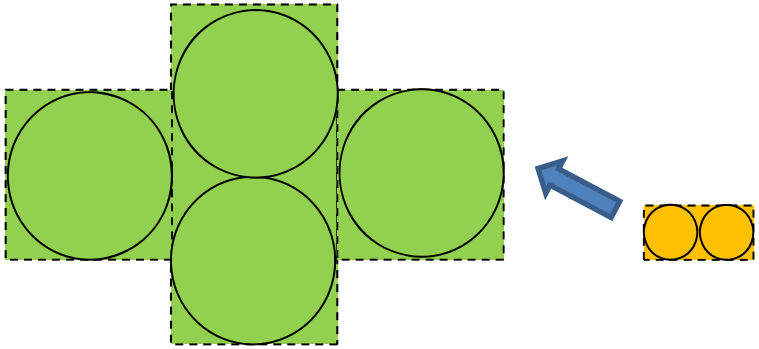


質点1が緑の範囲に入る条件　or
質点 2が緑の範囲に入る条件　or
質点 3が緑の範囲に入る条件　or
質点 4が緑の範囲に入る条件

点と長方形では2個の条件が必要なので、　２＊３＊４＝24個の条件

一般に、円どうしの判定はシンプル

物体1, 2 を円の集合体とみなして、円で判定



円どうしは1個の条件なので、
４＊２＝8個の条件

実際物体の角付近の当たり判定は厳しくなくてもよいので、円で当たり判定するのが基本

演習10

〇もしくは□の物体が、演習 B-1で仮定したように動くとし、演習7のようにキー入力により丸い形状の自機を移動させるとき、物体の当たり判定をプログラミングし、当たったとき、進行方向と逆方向に跳ね返すようにせよ。自機と物体の半径は10程度

第4回目、演習 7のプログラムを参考に実座標で〇を動かすプログラムに変更

```
void main (int Number){
  short pl,pr,pu,pd;
   int x=0,y=0,r=10;
  while(1) {
    pl=GetAsyncKeyState( VK_LEFT  );
    pr=GetAsyncKeyState( VK_RIGHT );
    pu=GetAsyncKeyState( VK_UP    );
    pd=GetAsyncKeyState( VK_DOWN  );

    Plot_pen(0,2,7);
    Circle(x-r,y-r,x+r,y+r,1);
    if (pl<0) x=x-1;
    if (pr<0) x=x+1;
    if (pu<0) y=y-1;
    if (pd<0) y=y+1;

    Plot_pen(0,2,3);
    Circle(x-r,y-r,x+r,y+r,1);
    UpdateWindow(hWnd);
  }
}
```

自機の座標はピクセル座標になっている。当たり判定をするには座標系を合わせる必要があるので、演習B-1の実座標に合わせるよう変更が必要

・x,y の変数名は演習B-1で使うのでxb, yb等に変更

・double型でxb, yb, r を宣言、r は実座標での半径

・円の描画には xb, yb, r をピクセル座標に変換した nxb, nyb, nr を使う。変換式は演習B-1と同じ

・自機の1回あたりの移動距離を実座標に合わせる (1では大きすぎる)

・自機の位置更新の際フィールドから外に出られないように -0.5<x,y<0.5 の制限をつける

演習10続き

前回、演習 B-1のプログラムを変更し、自機の移動を円の描画で表現するプログラムに変更し、
演習7を修正したプログラムと合体させる。

```
int i,m,nx,ny,np;
double v,th,pi=3.141;
double x=0,y=0,dt=0.01,vx,vy;

v=Get_double(0);
th=Get_double(1)/180*pi;
vx=v*cos(th);     //速度ベクトルx成分
vy=v*sin(th);     //速度ベクトルy成分

for(m=1;m<=1000;m++){

  x=x+vx*dt;    //x座標更新
  y=y+vy*dt;    //y座標更新
  if (abs(x)>0.5)  vx=-vx;
  else if (abs(y)>0.5)  vy=-vy;
  else {

     nx=(int)(x*N)+N/2;
     ny=(int)(y*N)+N/2;

 }


UpdateWindow(hWnd);
}
```

先ほどのプログラムの変数定義部を挿入

自機の移動（先ほどのプログラムのキー入力、
〇消去、〇の位置更新、〇描画部を挿入）、

過去の位置(nx,ny)の物体を消去

新しい位置(nx,ny)の物体を描画

当たり判定部　円どうしなら、
二つの円の中心間距離 L を計算。
L が二つの半径の和 より小さければ衝突
衝突したとき、速度ベクトルの符号を逆転